TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl für Bildverarbeitung und Künstliche Intelligenz / IX

Real-time Tracking of Objects in Image Sequences
Beyond Axis-Aligned Boxes

Tobias Böttger-Brill

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr. Martin Schulz
Prüfer der Dissertation:		
	1.	Hon.-Prof. Dr. Carsten Steger
	2.	Prof. Dr.-Ing. Reinhard Koch
	3.	Prof. Dr. Nassir Navab

Die Dissertation wurde am 11.12.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 24.04.2019 angenommen.

Abstract

Single-object tracking is one of the fundamental problems in Computer Vision and has been actively researched for many years. Nonetheless, the object tracking capabilities of humans are still far superior to the current state-of-the-art tracking algorithms. Humans are not only able to track almost any object, even previously unseen ones, in difficult sequences more robustly than current algorithms, they are also able to do this effortlessly within a few milliseconds. Since tracking is usually only a single component in a Computer Vision application, the real-time capabilities of object trackers are an essential property in order for the complete system to work in real-time.

The overall goal of this thesis is to bridge the gap between human and machine tracking by developing tracking algorithms that are accurate, robust, and real-time-capable. This leads to the following four main contributions:

First, we extend the standard single-object tracking evaluation protocol to support pixel-precise ground truths and trackers. The new evaluation protocol is able to better measure the accuracy of trackers and additionally estimates the tracker's capability of detecting scale change. Furthermore, we present upper bounds for the accuracy of trackers that are restricted to bounding boxes given pixel-precise ground truth data.

Second, we present a new tracking dataset with pixel-precise ground truth. The precise ground truth labels are created automatically from a photo-realistic synthetic dataset. The dataset is focused on sequences in the domain of self driving cars. The sequences are very challenging and include long sequences with full occlusions and different lighting conditions. Together with the previously mentioned evaluation protocol, the pixel-precise labels allow to evaluate trackers with a new level of precision.

Third, we develop a new single-object tracker that is able to track homogeneous regions that undergo arbitrary deformations. It is based on component-trees and is computationally efficient. The tracker is successfully used for 2D temporal tracking and for 3D object segmentation.

Fourth, we develop an edge-based object tracker. The tracker has a failure mode detection, is robust to nonlinear illumination changes, and can cope with occlusions. It is very accurate and virtually drift-free even for long sequences. Furthermore, it is inherently capable of object re-detection when tracking fails. The tracker is thoroughly evaluated through a number of qualitative and quantitative experiments. It is able to perform on par with the current state-of-the-art deep-learning trackers but is at least 45 times faster.

Zusammenfassung

Die Verfolgung von Objekten in Bildsequenzen ist eines der grundlegenden Probleme der Bildverarbeitung und wird seit vielen Jahren aktiv erforscht. Obwohl die Verfahren immer robuster und schneller werden, ist der Mensch den aktuellen Algorithmen zur Objektverfolgung noch weit überlegen. Er ist in der Lage, auch unbekannte Objekte in schwieriger Umgebung robust zu verfolgen, und benötigt dafür nur wenige Millisekunden. Da die Objektverfolgung in der Regel nur eine einzelne Komponente in einer größeren Anwendung darstellt, ist die Echtzeitfähigkeit von Objektverfolgungsverfahren Voraussetzung dafür, dass das Gesamtsystem in Echtzeit arbeiten kann.

Ziel dieser Arbeit ist es, die Lücke zwischen Mensch und Maschine zu verkleinern und Verfahren zur Objektverfolgung zu entwickeln, die sowohl genau und robust als auch echtzeitfähig sind. Daraus ergeben sich die folgenden vier Hauptbeiträge:

Erstens erweitern wir das Standardevaluierungsprotokoll zur Objektverfolgung, so dass Verfahren pixelgenau evaluiert werden können. Das neue Auswertungsprotokoll ist in der Lage, die Genauigkeit von Objektverfolgungsverfahren besser zu messen und liefert zusätzlich ein Maß dafür, wie gut die Methoden Skalierungsänderungen erkennen können. Darüber hinaus berechnen wir für pixelgenaue Ground Truth Obergrenzen für die Genauigkeit von Objektverfolgungsverfahren, die Objekte durch umschließende Rechtecke repräsentieren.

Zweitens präsentieren wir einen neuen Datensatz zur Objektverfolgung mit pixelgenauer Ground Truth. Die Ground Truth wird vollautomatisch aus einem fotorealistischen synthetischen Datensatz erstellt und konzentriert sich auf anspruchsvolle Sequenzen mit selbstfahrenden Fahrzeugen. Sie beinhaltet lange Sequenzen mit Verdeckungen und variierenden Lichtverhältnissen. Zusammen mit dem zuvor erwähnten Auswertungsprotokoll ermöglicht die pixelgenaue Ground Truth die Auswertung von Objektverfolgungsalgorithmen mit einer erhöhten Genauigkeit.

Drittens entwickeln wir ein neues Objektverfolgungsverfahren, welches in der Lage ist, homogene Regionen zu verfolgen, die sich beliebig verformen können. Es basiert auf Komponentenbäumen und ist recheneffizient. Das Objektverfolgungsverfahren wird erfolgreich für die 2D-Objektverfolgung und für die Segmentierung von 3D-Objekten eingesetzt.

Viertens entwickeln wir ein kantenbasiertes Objektverfolgungsverfahren. Die Methode verfügt über einen Fehlererkennungsmodus, ist robust gegenüber nichtlinearen Beleuchtungsänderungen und kann mit Verdeckungen umgehen. Das Verfahren ist sehr genau und weicht auch bei langen Sequenzen kaum von der Ground Truth ab. Außerdem können Objekte wiedererkannt werden, wenn die Verfolgung fehlschlägt. Das Objektverfolgungsverfahren wird in einer Reihe von qualitativen und quantitativen Experimenten gründlich ausgewertet. Es erreicht eine vergleichbare Genauigkeit wie aktuelle Deep-Learning Methoden, ist aber mindestens 45-mal schneller als diese.

Acknowledgments

You don't attempt a doctoral dissertation for fame or fortune. I'm convinced you do it for the people you meet in the process and the lessons you learn from them. The process of this work was a challenging wee little endeavour and I required a network of support. I am indebted to many people that helped me along the road.

First and foremost, I want to thank my parents. For teaching me the value of life, love, and happiness. And of course, a huge thanks to my sisters, well, obviously, for being awesome!

This thesis was conducted in collaboration with MVTec Software GmbH, and I would like to express my thanks to the whole company, for the workspace, the support, and the very pleasant atmosphere. I would especially like to thank my academic advisor, Prof. Carsten Steger for his valuable guidance over the years and many insights into what really counts when designing algorithms for Machine Vision. I thank Dr.-Ing. Markus Ulrich for his guidance, feedback, and encouragement during my work at the research department. Also thanks go to Dr. Bertram Drost for his support, patience, always being the first in line when I had questions, and teaching me so much over the years. And of course, I am also very thankful to many of my colleagues and friends for the amazing time and countless fruitful and controversial discussions: Roman Moie, Matthias Messner, Michael Klostermann, Horst Osberger, Micha Fauser, Rebecca König, David Sattlegger, Phillipp Härtiger, Paul Bergmann,....

I would like to express my gratitude to my thesis reviewers Prof. Reinhard Koch and Prof. Nassir Navab. Thank you for your input and effort of reviewing my work.

Some very special thanks go to my co-workers and very good friends Christina Eisenhofer and Patrick Follmann. Thank you Christina, for your expertise, for always being there for me, and helping me with countless academic and personal questions. Thank you Patrick, for the numerous nights working on papers, drinking beer, your input and ideas, and being awesome. In a nutshell, this work would not exist if it was not for the both of you.

For the amazing last years, I am also greatly indebted to the network of friends that have accompanied me over the years and make my life as awesome as it is. Thanks for that!

Most importantly, thank you Saskia! For enduring this process together with me. Thank you for being there for me when I was going crazy. Thank you, for all the encouragement, unconditional support, and love. I look forward to shaping our future together.

▲

Contents

Abstract iii

Zusammenfassung v

Acknowledgments vii

Outline xiii

I Introduction and Basics 1

1 Introduction 3
- 1.1 Background and Motivation . . . 3
- 1.2 Objectives and Contributions . . . 5
- 1.3 Publications . . . 6

2 Notations and Fundamentals 9
- 2.1 Notations . . . 9

II Evaluation Metrics and Datasets 13

3 Related Work: Methodologies 15
- 3.1 Object Tracking Datasets and Benchmarks . . . 15
- 3.2 Common Accuracy Measures . . . 19
- 3.3 Discussion . . . 22

4 A Novel Approach for Evaluating the Accuracy of Object Trackers 23
- 4.1 Pixel-precise Object Tracking Annotations . . . 23
- 4.2 Relative Intersection over Union . . . 26
  - 4.2.1 Optimization: Computing Optimal Boxes . . . 27
  - 4.2.2 Validation of Optimal Boxes . . . 29

4.3 Theoretical Trackers 30
4.4 Measuring Scale Changes 32
4.5 Experiments 34
4.6 Discussion 39

5 Datasets and Evaluation Protocol **41**
5.1 Related Work: Segmentation Datasets 41
5.2 A Novel Tracking Dataset: Playing for Tracking Data (PFTD) 48
5.2.1 Semi-supervised Annotation 48
5.2.2 Dataset Statistics 50
5.3 Discussion 53

III Fast and Robust Object Tracking **55**

6 Related Work **57**
6.1 Multiple Object Tracking 57
6.2 Single-Object Tracking 60
6.2.1 Generative Trackers 60
6.2.2 Discriminative Trackers 64
6.2.3 Deep Learning Meets Tracking 67
6.2.4 Video Object Segmentation 68
6.3 Most Related Works 69

7 Efficiently Tracking Homogeneous Regions **71**
7.1 Related Work: Component-Trees 71
7.2 Derivative-based Component-Tree 74
7.2.1 Local Flooding Tree Construction 74
7.2.2 Characteristics of the Derivative-based Component-Tree 77
7.2.3 Implementation Details 78
7.3 Tracking Maximally Stable Homogeneous Regions 80
7.3.1 Maximally Stable Homogeneous Regions 81
7.3.2 MSHR Versus MSER 82
7.3.3 MSHR Tracking 84
7.4 Applications 86
7.5 Discussion 89

8 Shape Model Tracking **91**
8.1 Fundamentals: Shape Model Matching 91
8.2 Fast and Accurate Shape Model Tracking 96
8.2.1 Strengths and Weaknesses 100
8.3 Improved Shape Model Tracking 101
8.3.1 Model Update: Refinement of All Levels 101
8.3.2 Dynamic Pyramid Generation 102
8.3.3 Robust Tracking 110

8.3.4 Motion Estimation 112
8.3.5 Search Space Optimization 120
8.4 Discussion 121

9 Results and Comparison to the State of the Art 123
9.1 Evaluation Metrics and Protocol 123
9.2 Experiments: Playing For Tracking Data 125
9.2.1 Shape Model Tracking Improvements 125
9.2.2 Comparison to the State of the Art 137
9.3 Experiments: Real-time Tracking on a Raspberry Pi 145
9.4 Experiments: Miscellaneous Applications 146
9.5 Discussion 151

10 Conclusions 155
10.1 Summary 155
10.2 Future Work 156

Appendices 159

A Bayesian Tracking 161

B Correlation Filter Tracking 165

Bibliography 167

Outline of the Thesis

Part I: Introduction and Basics

motivates the work and gives an overview of the challenges, the proposed solutions, and the basic notation and methods this work builds apon.

Chapter 1: Introduction

motivates this work and gives an overview of the challenges and solutions.

Chapter 2: Notations and Fundamentals

gives an overview of the used notation and introduces some of the basic mathematical concepts used in this work.

Part II: Evaluation Metrics and Datasets

extends the commonly used tracking evaluation techniques for bounding boxes to pixel-precise segmentations of the ground truth. To get a perspective of how well bounding box approaches can perform for a given segmentation, an approach to compute upper bounds is presented.

Chapter 3: Related Work: Methodologies

presents the methods generally used for evaluating the acuracy and the robustness of object trackers.

Chapter 4: A Novel Approach for Evaluating the Accuracy of Object Trackers

introduces a novel method to generate upper bounds for all box-based trackers that are evaluated on pixel-precise segmentations. The bounds enable us to compare approaches that are restricted to bounding boxes and those that are not.

Chapter 5 Datasets and Evaluation Protocol

presents the datasets and the evaluation protocol used for the evaluation. The core of the evaluation is based on synthetic ground truth data.

Part III: Fast and Robust Object Tracking

presents two methods for robust and real-time tracking of objects in video sequences. Both approaches are not restricted to bounding boxes and are extremely efficient.

Chapter 6: Related Work
: presents an overview of the works in single- and multi-object tracking. The relationship to the two novel tracking approaches is presented.

Chapter 7: Efficiently Tracking Homogeneous Regions
: presents a method for tracking deformable objects in image sequences. It is based on component-trees and able to track homogeneous regions that undergo arbitrary deformations. The approach is extremely efficient, works in multi-channel images and can be extended to perform 3d segmentation in slice data.

Chapter 8: Shape Model Tracking
: presents a method for robust and real-time tracking of roughly rigid objects in video sequences. The tracker is robust to nonlinear illumination changes, occlusions, and drift. It has a failure mode detection and is inherently capable of re-detecting the object if the tracker fails.

Chapter 9: Results and Comparison to the State of the Art
: evaluates the shape model tracking and its different extensions against the state of the art on different datasets.

Chapter 10: Conclusions
: discusses open problems and possible extensions of the presented evaluation metrics and tracking approaches.

Part I

Introduction and Basics

1 Introduction

1.1 Background and Motivation

The human visual system is an incredible phenomenon. It is the center of our perception and allows us to navigate through complex environments. In fractions of a second we are able to identify the objects that are present in front of us. Even if we have never seen a specific object before, we can set the object into context very quickly and use our background knowledge to infer the most likely purpose of the object and in most cases also its category. For example, we recognize any flat surface near a table with some kind of feet to be a chair. Even if we have never seen the specific chair type before. However, as most of our perception, it is mostly a relative measuring systems. While it is easy for humans to identify if two objects are the same size or color, it is almost impossible to gain a notion of the metric size or color of any object without a reference.

For decades, it has been a fascinating research topic to understand and copy the human ability to interpret visual scenes. Computer Vision is the science that aims to give similar, or better, visual capabilities to a machine or computer. It covers the complete pipeline, from image acquisition, processing, analyzing to understanding. Thanks to technological breakthroughs and ever faster computers, the range of applications of Computer Vision has risen tremendously in the past few years. Today, most people carry a high-resolution camera with amazing computational capabilities in their pocket that is able to improve pictures, recognize faces, measure, or support impressive augmented reality applications. In contrast, Machine Vision refers to the use of Computer Vision in an industrial application or process. It attempts to integrate and extend existing technologies and apply them to solve real world industrial problems. It is typically concerned with quality inspection and control in production lines and robot guidance, i.e., allowing a robot to interact with its environment. The systems demand great robustness, reliability, accuracy, and stability. Furthermore, there is typically a strong restriction on the hardware cost and size. In general, Machine Vision systems are a driving factor in automation and mass production. They essentially allow to improve the production quality and reduce labor costs [208]. [1]

[1] While the automation of an industry may lead to unemployment in the short-term, the long-term

One of the fundamental problems in Computer Vision and Machine Vision is object tracking. It is one of the problems that the human visual system can solve without much effort. Humans are even capable of reasoning about objects that reappear in an image sequence and link identities between objects that have been acquired at different times, perspectives, and under different illumination. However, it is still a very active research area in Computer Vision and the performance of the state of the art algorithms is far from what humans are capable of [117]. There is significant room for improvement, both in terms of accuracy and in terms of runtime.

The main focus of this thesis is to develop algorithms for tracking objects in Computer Vision and Machine Vision applications that are fast, robust, and accurate. The focus is on developing computationally light weight and memory efficient algorithms that can run on small computation devices in real-time.

Object Tracking Object tracking is concerned with locating an object throughout an image sequence. It can be divided into two categories: multi-object and single-object tracking. The first category is concerned with tracking objects of a known object class. For example, pedestrians in an urban street scene. As a consequence, most methods make use of powerful object detection systems. Although the object's class is known, they interact with each other, occlude each other, or move in an unpredictable fashion. The task of multi-object tracking is to link the detections between frames, remove false detections, and correctly detect occlusions and missing detections (if possible, at frame rate). The main focus of the multi-object tracking research is on surveillance tasks such as pedestrian tracking or traffic surveillance. In contrast, single object detection is generally concerned with tracking a single unknown object through an image sequence. Since the object class is not known, it is usually prohibitive to use a general object detection algorithm.

In single object tracking, it is usually assumed that the object position is known in the first frame. In most cases, the image sequences originate from temporally connected video data. As such, object tracking has very strong computational restrictions. The object location generally needs to be known at a frame rate of 20-50 frames per second. Hence, the processing time should be no more than 20–50 ms. Furthermore, object tracking is usually only a subproblem of a Computer Vision or Machine Vision application. Therefore, the computational complexity is strictly restricted in terms of the processing time and the memory requirements. Last but not least, many applications that involve tracking are performed on small computational devices such as drones or smart cameras. As a consequence, the computational complexity and memory requirements are further restricted by the capacities of these devices.

The challenges in object tracking include abrupt object motion, nonrigid objects, changing appearance of both the object and the surrounding, object occlusion, motion blur, object disappearing and reappearing, and camera motion. As a consequence, it

consequences are more complex and may include a higher living standard, longer life expectancy, and less dangerous jobs. Although every researcher should be aware of the possible consequences of their work, the social and economical impact and consequences of automation in general are beyond the scope of this thesis.

is unreasonable to expect a single object tracking scheme to solve all of the mentioned challenges on its own. The domains and applications that are tackled by single object tracking schemes are very diverse and include amongst others:

1. Vehicle navigation and traffic control
2. Gesture recognition and Human machine interaction (e.g., eye gaze tracking for virtual reality)
3. Livestock monitoring in agriculture
4. Consumer electronics (e.g., Follow Me drones)
5. Surveillance (e.g., detection of suspicious and abnormal activities or unlikely events)

Object Representation There are numerous ways to represent an object in an image; by a single point (e.g., centroid), multiple points, geometric shapes (e.g., a box), contour lines, skeleton shapes, or a pixel-precise segmentation. Depending on the application, each representation has its justification. In object tracking, the objects are typically represented by axis-aligned bounding boxes. This has two reasons: (1) the ground truth data is easy and cheap to obtain and (2) restricting trackers to axis-aligned bounding boxes greatly reduces the complexity of their implementation and runtime. In general, the current datasets, benchmarks, and evaluation protocols are restricted to boxes. However, especially in Machine Vision applications that require a high level of accuracy and robustness, the bounding box of an object is not sufficient.

1.2 Objectives and Contributions

The objective of this thesis is to develop methods for tracking single objects in real-time, with low computational complexity, and without restricting the object representation to an axis-aligned bounding box. This enables the methods to solve real-world problems. In a nutshell, the objectives of this thesis are threefold:

1. extend the evaluation methodologies to a pixel-precise representation of objects,
2. create new tracking dataset with a pixel-precise representation of the ground truth,
3. develop new methods that are able to solve different real-world applications in real-time, with high accuracy, and with low computational complexity.

In more detail, this work tackles the three of the above mentioned objectives with the following contributions:

Evaluation As mentioned above, the current state of the art tracking methods and the common tracking benchmarks represent objects by axis-aligned bounding boxes. As also the evaluation is based on axis-aligned bounding boxes, there is no incentive for methods to extend beyond them. On the contrary, more precise methods do not necessarily obtain higher accuracy scores. However, especially in industrial applications, the bounding box of an object is of limited use. To overcome this, we extend the evaluation protocols to support pixel-precise representations of the objects and develop a measure to link the performance of trackers that are restricted to bounding boxes to methods that support a pixel-precise representation of objects. Furthermore, we introduce a scale score that can be computed automatically for every tracker. It measures how well the tracker is able to capture scale changes of the ground truth.

Dataset The new evaluation protocol requires a pixel-precise ground truth. To encourage the development of trackers with a more precise representation of an object, a new tracking dataset with pixel-precise ground truth is presented. In contrast to most of the common benchmarks that are designed to be as diverse as possible, we focus on a single application: car tracking in a self-driving car scenario. The dataset has extremely accurate pixel-precise annotations. Since the label effort of pixel-precise segmentations is immense, the data set builds on a photo-realistic synthetic dataset that was generated from extracting frames and ground truth automatically from a computer game. The new dataset includes many challenges such as long sequences, heavy occlusion, perspective change, and objects that disappear and reappear. The evaluation of prior-art methods shows that there is significant room for improvement over the current state of the art.

Tracking The goal is to extend the state of the art in object tracking beyond bounding boxes. For this, two new real-time trackers are presented. The first component-tree-based tracker can track an object undergoing arbitrary deformations. In each frame regions are segmented and matched to object. By using a component-tree, the method is extremely efficient and can track multiple regions in real-time. The second shape model-based method focuses on tracking roughly rigid objects very accurately. The object is represented by edge points and their direction. During tracking, those points are matched to the next input frame. Additionally, the points and the directions are updated consistently during tracking to allow shape changes of the object. The tracker is also capable of detecting tracking failure and can re-detect the object in real-time if tracking fails.

1.3 Publications

Parts of this thesis contain material previously published in the following publications.

- Tobias Böttger, Markus Ulrich, and Carsten Steger, *Subpixel-Precise Tracking of Rigid Objects in Real-Time*, in 20th Scandinavian Conference on Image Analysis (SCIA), 2017, pp. 54–65. doi: 10.1007/978-3-319-59126-1_5

- Tobias Böttger, Patrick Follmann, and Michael Fauser, *Measuring the Accuracy of Object Detectors and Trackers*, in German Conference on Pattern Recognition (GCPR), 2017, pp. 415–426. doi: 10.1007/978-3-319-66709-6_33

- Tobias. Böttger and Christina Eisenhofer, *Efficiently tracking extremal regions in multichannel images*, in 8th International Conference of Pattern Recognition Systems (ICPRS), Institution of Engineering and Technology, 2017, pp. 6–14. doi: 10.1049/cp.2017.0143

- Tobias Böttger and Patrick Follmann, *The Benefits of Evaluating Tracker Performance Using Pixel-Wise Segmentations*, in IEEE International Conference on Computer Vision Workshops (ICCVW), 2017, pp. 1983–1991. doi: 10.1109/ICCVW.2017.232

- Tobias Böttger, Gutermuth Dominik, and Christina Eisenhofer, *A Natural Extension of Component-Trees to Multi-Channel Images*, IEEE Transactions on Image Processing, - (-), p. UNDER REVIEW

- Tobias Böttger and Carsten Steger, *Accurate and Robust Tracking of Rigid Objects in Real-time*, IEEE Transactions on Image Processing, - (-), p. UNDER REVIEW

Parts of this thesis contains material that has been applied for patent (pending).

2 Notations and Fundamentals

This chapter introduces the basic concepts, nomenclature, and notations used in this thesis. A summary of the general notations used within this thesis is displayed in Table 2.1.

2.1 Notations

Throughout the thesis, we distinguish between discrete image coordinates that lie on the pixel grid and those that do not by a superscript bar $^{-}$. A point p that lies on the pixel grid is denoted by $\overline{p}$, whereas arbitrary image coordinates, not necessarily on the pixel grid, by p. Points may lie outside of the image boundaries and have negative values. Thus $p \in \mathbb{R}^2$ and $\overline{p} \in \mathbb{Z}^2$. For all coordinates, we assume the origin is at the upper left corner of the image. Furthermore, we denote direction vectors with a superscript arrow, e.g., $\vec{d}$. For every point coordinate $p = (x, y)$ or direction vector $\vec{d} = (u, v)$, the first tuple entry refers to the vertical axis (row) and the second entry to the horizontal axis (column). In the figures, points are colored blue, directions are colored orange, and normals green (provided this is unambiguous). A point, its direction, and the respective normal are depicted in Fig. 2.1.

Shape model points The shape model tracking used within this thesis represents the tracked object by a discrete set of n elements $m_i = (p_i, \vec{d}_i) \in \mathbb{R}^2 \times S^1$, which are each composed of a point p_i and a associated direction vector $\vec{d}_i$.

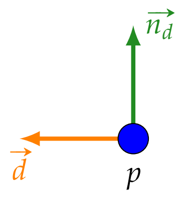


Figure 2.1: A point p, its direction $\vec{d}$ and a respective normal $\vec{n_d}$.

The collection of all model points is termed *shape model* and denoted as

$$\mathcal{M} = \{m_i \in \mathbb{R}^2 \times S^1; \text{ for } i = 1, \dots, n\}. \tag{2.1}$$

Transformations The transformation of a shape model requires the transformation of the points p_i and the directions $\vec{d}_i$. For a *rigid* transformation, i.e., one that preserves the distances between every pair of points, the transformation of a point p_i is given by

$$p_i' = \underbrace{\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}}_{R_\theta} p_i + \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \tag{2.2}$$

where R_θ represents a rotation matrix and $(t_x, t_y)^T$ the translation. In the following, the direction $\vec{d}_i$ of a point is determined by the normalized local image gradient direction at the point p_i. As a consequence, the directions $\vec{d}_i$ are normals to the tangent of p_i. Then, for a general 2×2 matrix A, the transformation of the direction vectors can be obtained by multiplying with the inverse transposed of the transformation matrix A (the directions are normals to the tangent of p). Since a general transformation may change the length of $\vec{d}$, the transformed direction $\vec{d}'$ needs to be normalized after the transformation to ensure $\vec{d}' \in S^1$:

$$\vec{d}_i' = \frac{\left(A^{-1}\right)^T \vec{d}_i}{\left\| \left(A^{-1}\right)^T \vec{d}_i \right\|}. \tag{2.3}$$

For a rotation matrix R_θ, the transformation of the directions simplifies to

$$\vec{d}_i' = R_\theta \vec{d}_i, \tag{2.4}$$

since $R^T = R^{-1}$ and $\det R = 1$. The model can further be scaled *isotropically* ($s_x = s_y$) and *anisotropically* ($s_x \neq s_y$) by

$$p_i' = \underbrace{\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}}_{R_\theta} \underbrace{\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}}_{S_{s_x,s_y}} p_i + \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \tag{2.5}$$

where S_{s_x,s_y} represents the scale matrix. The rotation R_θ and scale matrix S_{s_x,s_y} can be combined in a single transformation matrix, which we denote as T_{θ,s_x,s_y}. The direction vectors are transformed, by

$$\vec{d}_i' = \frac{\left(T^{-1}_{\theta,s_x,s_y}\right)^T \vec{d}_i}{\left\| \left(S^{-1}_{s_x,s_y}\right)^T \vec{d}_i \right\|}. \tag{2.6}$$

We express the more general (parallelism preserving) *affine transformation* of model points in homogeneous coordinates,

$$p_i' = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}_{p_i} . \tag{2.7}$$

The affine transformation matrix can be decomposed into rotation, anisotropic scaling, skew and translation matrices:

$$p_i' = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}_{p_i'} . \tag{2.8}$$

Similarly, *projective transformations* of model points can be expressed as

$$p_i' = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} p_i. \tag{2.9}$$

Shape model Transformations In the matching process, a shape model $\mathcal{M}$ is transformed and moved to various positions in a search image to compute its similarity for each pixel in a predefined search domain, denoted as $\mathcal{S} \subset \mathbb{Z}^2$. Note that $\mathcal{S}$ only contains the locations where the center of the shape model $\mathcal{M}$ is placed. In the search process, the transformations are usually restricted to *similarity transformations*, hence translations rotation and isotropic scaling. To further speed up the process, the possible transformations are discretized and typically restricted to expected and physically plausible ranges. We denote the set of possible rotations with $\mathcal{R}$ and the set of possible scales with Σ. In the following, we always assume isotropic scaling of the model, hence $\Sigma \subset \mathbb{R}_+$ and $s_x = s_y$. We denote the complete set of possible transformations as

$$\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma} := (\mathcal{S} \times \mathcal{R} \times \Sigma) \subset \left(\mathbb{Z}^2 \times [0°, 360°) \times \mathbb{R}_+\right), \tag{2.10}$$

and merely as $\mathcal{P}$ when the explicit ranges are not essential. Analogously, a transformation within $\mathcal{P}$ is expressed as $T_{\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}}$ and $T_{\mathcal{P}}$, respectively.

Table 2.1: General notation used throughout the thesis

	Spaces
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_+$	Set of real numbers > 0
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}$	Set of all integers
$\mathbb{R}^n$	Vector space of real numbers with dimension n
$S^1 \subset \mathbb{R}^2$	The unit circle, hence (x, y) that satisfy $x^2 + y^2 = 1$
$\mathbb{R}^n \times \mathbb{R}^m$	Cartesian product of the two vector spaces with dimension n and m, respectively
	Points and Vectors
$z \in \mathbb{R}^n$	Column vector of length n
z^T	Transposed vector z
$M \in \mathbb{R}^{n \times m}$	Matrix of size $n \times m$
$M_{i,j} \in \mathbb{R}$	The matrix entry in the i-th row and j-th column
$\overline{p} \in \mathbb{Z}^2$	An image coordinate (may be negative)
$p \in \mathbb{R}^2$	An image coordinate (not necessarily on the pixel grid)
$\vec{d} \in \mathbb{R}^2$	A direction, usually used in correspondence with an image point
$\vec{n}_d \in \mathbb{R}^2$	A normal of a direction $\vec{d}$ (hence $\vec{n}_d \perp \vec{d}$)
	Operators
$a \cdot b$	The Euclidean scalar product
$\hat{z}$	The Discrete Fourier Transform of z.
$\lVert \cdot \rVert$	The Euclidean norm of a point or vector, hence $\lVert a \rVert = \sqrt{a \cdot a}$.
$\lvert \cdot \rvert$	Either the area of a region or the cardinality of a set
	Subsets
$\mathcal{M}$	A *shape model*; a subset of $\mathbb{R}^2 \times S^1$, defined in (2.1)
$\mathcal{S}$	A *search region*; essentially an image domain and hence a subset of $\mathbb{Z}^2$

Part II

Evaluation Metrics and Datasets

3 Related Work: Methodologies

The evaluation of tracking and detection algorithms essentially depends on two components: (1) the dataset and (2) the performance evaluation measures, system, and protocol. In the following, we present and discuss the most prominent evaluation benchmarks and methodologies and point out open problems.

3.1 Object Tracking Datasets and Benchmarks

In many computer vision fields, it is common for the community to agree upon benchmark datasets and protocols to compare the performance of algorithms. Prominent examples are PASCAL VOC [75] for object detection, COCO for instance-aware object segmentation [134], or ImageNet [68] for image classification. For a long time, benchmark datasets of similar quality and diversity did not exist for object tracking. It used to be very difficult to compare two different trackers based on the results in the respective publications. The community suffered from the large variety of performance measures and the lack of consensus about which measures should be used in experiments [116, 119, 158, 198, 239, 40]. Since there are dozens of new tracking algorithms that are proposed each year at large conferences and in journals, it was hard to compare different trackers. The missing consensus on the evaluation metrics was complicated by the fact that the tested sequences were often hand-picked and sometimes had little overlap with other publications. In the mid 2010s, to unify the evaluation of trackers, different groups simultaneously proposed to standardize the evaluation datasets and protocols.

In 2014, Smeulders *et al*. [198] presented the Amsterdam Library of Ordinary Videos (ALOV300) for tracking. The dataset contains a diverse set of 315 videos that cover all kinds of objects and different challenges, such as varying illumination, transparency, specularity, similar objects within one scene, clutter, occlusion, low contrast, and severe shape changes. A few examples are shown in Fig. 3.1. Although the dataset is mostly mined from YouTube, it also includes the most common videos used for tracker evaluation within the literature to that date. With a few minor exceptions, the dataset focuses on short sequences. In total, the dataset has almost 90 000 frames. As is common, the objects are annotated by axis-aligned bounding boxes. To help understand a tracker's strengths

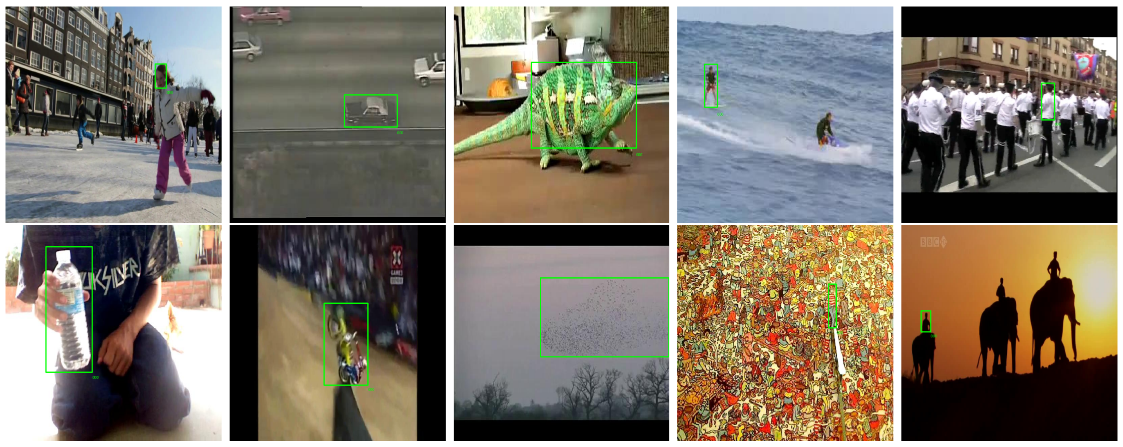


Figure 3.1: Example images from ALOV300 [198]. The objects are very diverse, including cars, people, static objects, and cartoons. The ground truth is labeled with axis-aligned bounding boxes.

and weaknesses, the videos are categorized into thirteen different categories, such as low contrast, camera motion, clutter, smooth motion, or specularity. Even though the benchmark paper brought clarity into the evaluation chaos that was present previously, the large amount of data and the dataset's overall complexity limited its use as a general benchmark across all tracking publications.

Around the same time, the *visual object tracking challenge* (VOT) [116, 117, 119] was started. The challenges are conducted on a publicly available dataset once a year. To meet the current challenges of the state of the art, the evaluation protocol and the dataset are updated each year. Nevertheless, a key goal is to keep the dataset as compact and diverse as possible. Starting from 16 short sequences (2013), the dataset has grown to 25 (2014) and, finally, to 60 (2015, 2016, 2017, 2018) sequences. It includes sports videos, dashcam videos of cars, webcam videos of humans and different objects, and animal videos.

In 2016, a thermal infrared imagery (TIR) challenge was added to address the fact that the current state of the art was performing very badly in this domain. In 2018, a new long-term tracking sub-challenge was added. The sequences within the VOT dataset are selected by an optimization process that makes sure the selected scenes cover different attributes and complexity. This ensures that many applications are covered and the dataset is still as compact as possible. The protocol of VOT uses two complementary performance measures: the tracker's accuracy (how well does it match the ground truth?) and the tracker's robustness (how often does the tracker fail?). The dataset is used very frequently in recent tracking publications.

In 2013, Wu *et al*. [238] proposed the Object Tracking Benchmark 50 (OTB-50). The benchmark includes 49 sequences and 50 labeled objects from the literature that had been used to evaluate trackers up to that date. It was extended to 98 videos with 100 labeled objects in 2015 (OTB-100) [239]. As with ALOV300, the sequences are labeled with attributes that represent challenging aspects such as illumination variation, occlusion, deformation, motion blur, fast motion, or low resolution. Although no explicit

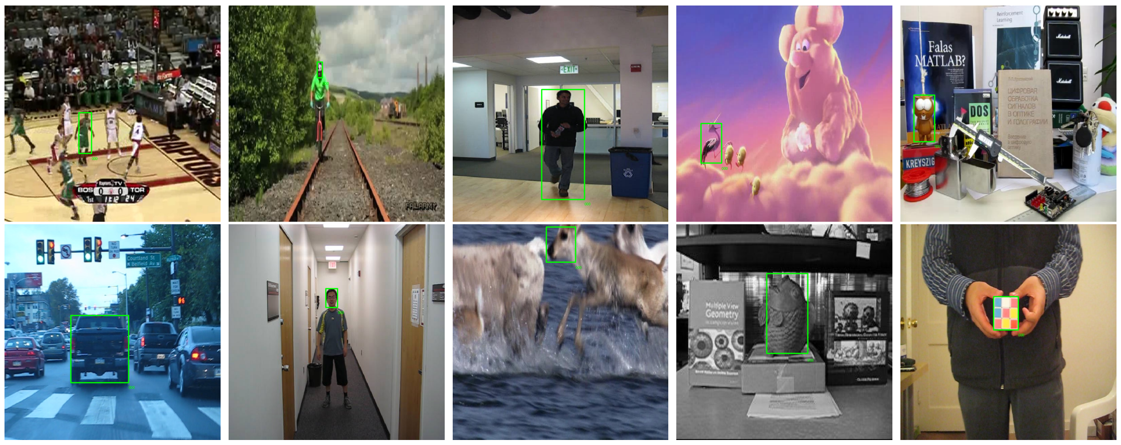


Figure 3.2: Example images from the OTB-100 challenge [239]. No specific application is addressed but rather many different challenges. The ground truth is labeled with axis-aligned bounding boxes.

optimization was performed to ensure the dataset is diverse, the videos within the dataset are very different and cover various applications. A few exemplary images are displayed in Fig. 3.2. The evaluation protocol measures the tracker's precision and success rate with two separate measures. Further tests are conducted to test how sensitive the trackers are to initialization by varying the time and the position of the initialization. Together with the VOT benchmark, the OTB-100 benchmark is the *de-facto* standard tracking benchmark.

The mentioned benchmarks all focus on short-term tracking. Since the performance has reached very impressive levels, the community is slowly pushing towards more complete trackers that can successfully track an object for a long period of time and recover on its own from losing the object. To address these challenges, in 2018 Valmadre *et al.* [222] presented a long-term tracking dataset (LTTD) that includes 366 sequences with a total of 14 hours of video. This made it the largest dataset for single-object tracking at the time of publication. The authors use YouTube Bounding Boxes [174] (YTBB) as data source from which they select and improve the annotations of the sequences. As above, the objects are labeled by axis-aligned bounding boxes. As evaluation metrics, the authors mainly use the True Positive Rate (TPR) (3.4) and True Negative Rate (TNR) (3.5). However, since most of the current trackers never predict the absence of an object, many currently have a TNR of 0.0.

Just a few weeks later, Müller *et al.* [155] proposed TrackingNet. With 30 000 videos and more than 14 million bounding box annotations, the dataset eclipses all other datasets. Among other criteria, it was constructed with the goal to further improve data hungry deep tracking approaches. Furthermore, the huge evaluation set poses a number of different challenges for the current state of the art. Again, both TrackingNet and LTTD do not focus on a specific application, but offer a large diversity to evaluate trackers "in the wild."

Since the benchmarks mentioned above focus on being as complex and diverse as possible, they do not tackle any industrial application specifically. As a consequence, they favor trackers that generalize across different domains. This requires an intense

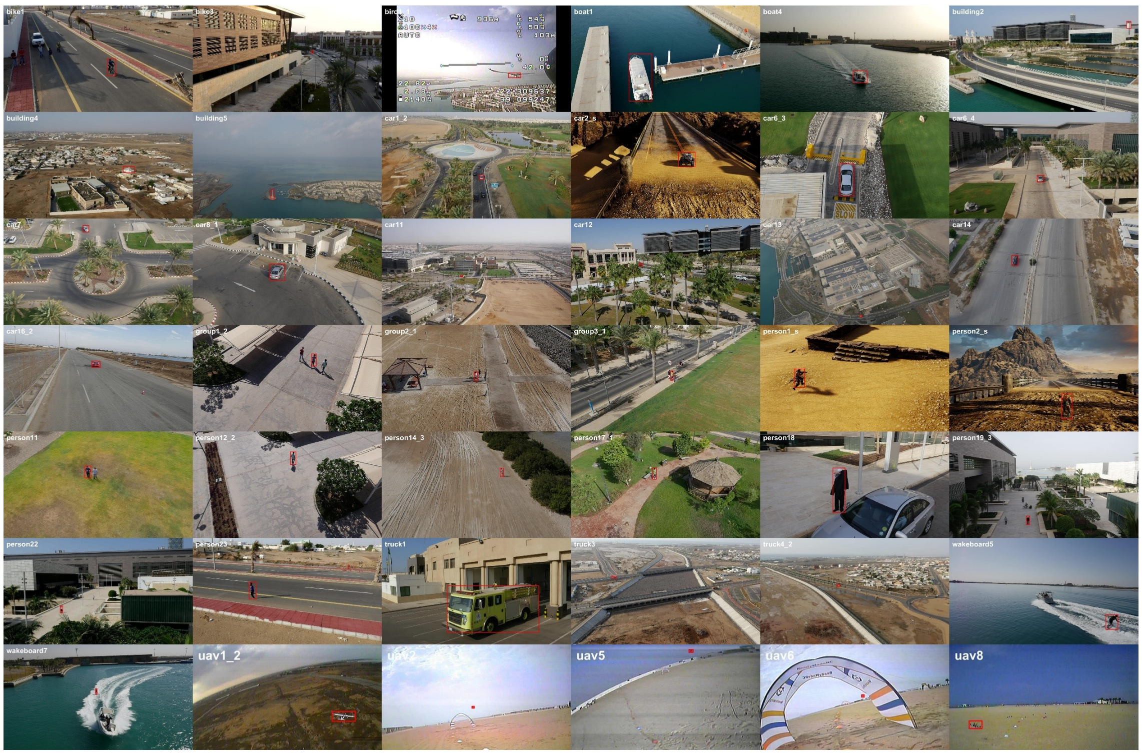


Figure 3.3: Example images from UAV123 [153]. The objects are mostly cars and people and are labeled with axis-aligned bounding boxes.

re-evaluation of the state-of-the-art trackers for specific tasks. For example, to address real-time tracking in the domain of drones, two new benchmarks were proposed very recently: the drone tracking benchmark (DTB70) [128] and UAV123 [153]. The dataset used for DTB70 contains 70 sequences obtained by a DJI Phantom 2 Vision+ drone and is labeled manually with axis-aligned bounding boxes. The UAV123 benchmark is specifically concerned with long-term tracking. It was proposed in 2016 by Müller *et al*. [153]. Independent of their performance in terms of accuracy and robustness, the benchmark excludes all trackers from the evaluation that are far from real-time (< 1 FPS). The UAV123 dataset contains 123 sequences that are obtained from three different sources. The first two sets consist of 115 sequences and contain real images acquired from UAVs. They are labeled manually with axis-aligned bounding boxes. The third set with 8 sequences, is obtained from a UAV simulator that renders different worlds and UAV trajectories with the Unreal4 Game Engine. A few example frames from different sequences are displayed in Fig. 3.3. UAV123 is the first tracking benchmark that contains synthetic sequences that are generated specifically for tracking from a graphics engine. The annotations are obtained automatically and include the full object masks/segmentations. The real-time restriction, the long sequences, and the complexity of the dataset are very challenging for the current state of the art. For UAV123, the performance of the best trackers is lower than for OTB-100 and the best trackers are not the same for both datasets.

A summary of the mentioned benchmarks is displayed in Table 3.1. As shown, all but one benchmark use axis-aligned boxes to represent the ground truth. To compare the performance of trackers with the ground truth, all of the benchmarks use an accuracy measure. Given the accuracy measure, different evaluation protocols are performed to evaluate different aspects, e.g., the robustness or the sensitivity to the initialization. Although most benchmarks use similar measures and protocols, some differences exist. In the following, we comment on the used accuracy measures and their advantages.

Table 3.1: Summary of the common object tracking benchmarks.

	# Sequences	# Frames	Axis-Aligned Box	Rotated Box
ALOV300 [198]	315	89 364	✓	
OTB-100 [239]	100	59 061	✓	
VOT 2017 [117]	60	21 356		✓
DTB70 [128]	70	15 781	✓	
UAV123 [153]	123	110 000	✓	
LTTD [222]	366	1.5 million	✓	
TrackingNet [155]	30 000	14 million	✓	

3.2 Common Accuracy Measures

Over the years, a number of different accuracy measures has been proposed in the tracking literature. However, all tracking measures assume that manual ground truth annotations of a sequence are given. In the following, we use a similar notation as Čehovin *et al.* [38, 40]. The works provide a highly detailed theoretical and experimental analysis of all common tracking performance measures. The authors show that many of the performance measures used throughout the tracking literature are highly correlated. We highlight the most prominent examples.

We assume the object state in frame t is described by an arbitrarily shaped region R_t and center $x_t \in \mathbb{R}^2$. As discussed above, in most benchmarks, R_t is an axis-aligned box and x_t its center point. Nevertheless, for more general representations of R_t, the object position x_t often cannot be derived directly from R_t.

For a sequence of length N, the objects states are collected in

$$\Lambda^G = \{(R_t, x_t);\ \text{for } t = 1, \dots, N\}. \tag{3.1}$$

The superscript G denotes that the set/point/region describes the ground truth object state and the superscript T denotes the tracker's prediction of the object state, respectively.

Many older tracking publications [183] use the Euclidean center distance

$$\delta_t = \|x_t^T - x_t^G\|_{l_2}, \tag{3.2}$$

as the accuracy measure. Also in the OTB-100 benchmark [239], the precision of a tracker is calculated as the percentage of frames for which the estimated location is within

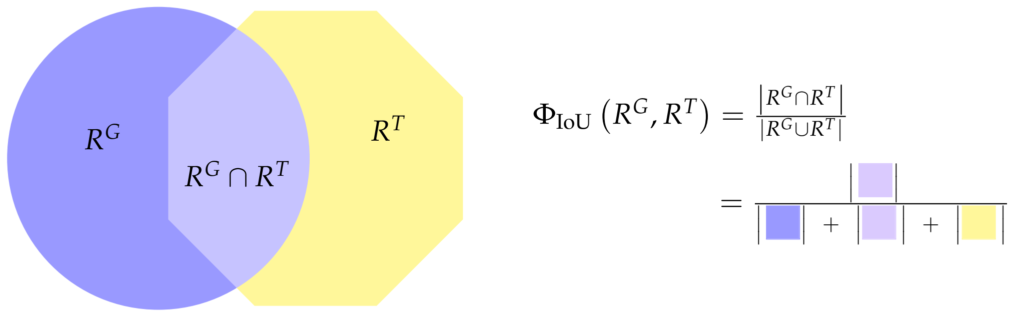

R^G
$R^G \cap R^T$
R^T
$\Phi_{\text{IoU}}\left(R^G, R^T\right) = \frac{|R^G \cap R^T|}{|R^G \cup R^T|}$
$= \frac{|🟪|}{|🟦| + |🟪| + |🟨|}$

Figure 3.4: The Intersection over union (IoU) of two regions is computed as the quotient of the area of their intersection and their union. It can be computed for arbitrarily shaped regions. Since the area of the union of two regions is always greater or equal than the area of their intersection, it lies in $[0, 1]$.

20 pixels of the center position of the ground truth ($\delta_t \leq 20$). An appealing property of the center distance is that the ground truth is easy to obtain and the core task of tracking, namely the localization of the object, is measured. The distances are often summarized over all N frames to obtain a single value per sequence, $\Delta(\Lambda^G, \Lambda^T) = \frac{1}{n}\sum_{t=1}^{n} \delta_t$. Nevertheless, the measure does not account for the object scale, is highly subjective for arbitrarily shaped regions, and not very robust (e.g., the center of gravity may change significantly for articulated objects). Furthermore, it is hard to define a scale-independent threshold to detect tracker failure from the l_2 distance alone.

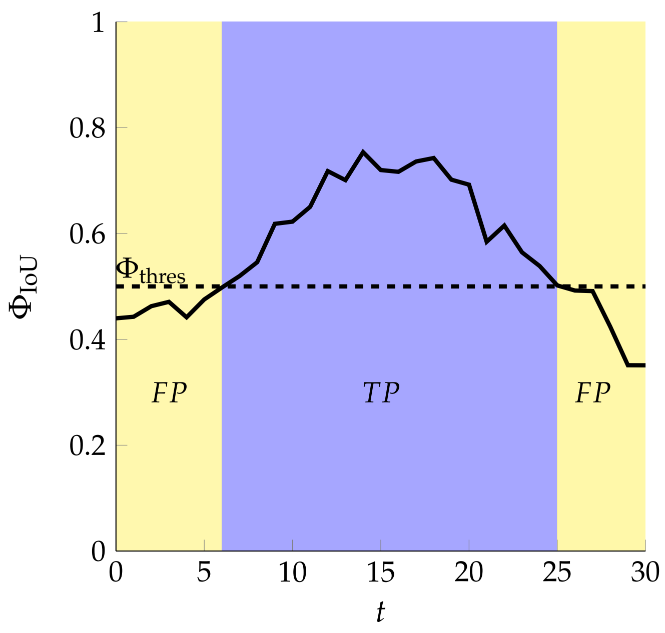
To remedy the mentioned problems, overlap-based measures are very often used [116, 153, 155, 198]. The most prominent one is the *Intersection over Union* (IoU),

$$\Phi_{\text{IoU}}\left(R^G, R^T\right) = \frac{\left|R^G \cap R^T\right|}{\left|R^G \cup R^T\right|}, \tag{3.3}$$

where R^G represents a ground truth region and R^T the corresponding tracker proposal. The computation of Φ_{IoU} is visualized in Fig. 3.4. It is often called Pascal overlap [75] or bounding box overlap when R^G and R^T are boxes [134].

The IoU can be computed for arbitrarily shaped regions and it accounts for both the localization accuracy as well as the shape prediction of the tracker. Furthermore, if the tracker fails completely, then $\Phi_{\text{IoU}} = 0$, since the intersection of the ground truth region and the tracker prediction is zero. This is an appealing property when averaging the IoU for each frame in a sequence. Furthermore, the IoU is correlated to region similarity measures that are based on the contour similarity [170]. However, these are generally more complex to compute and not as intuitive as the IoU.

As is common in object detection, the IoU is also used in tracking to measure the number of true positive (TP), false negative (FN), and false positive (FP) frames in a sequence by thresholding Φ_{IoU}. This is especially reasonable for long-term trackers, where objects may be occluded and disappear and reappear in a sequence. A tracker returns a TP whenever Φ_{IoU} is larger than a specific threshold Φ_{thres} and, conversely, a FP whenever Φ_{IoU} is smaller than the threshold. Please see Fig. 3.5 for an example. Throughout



$$TP = \sum_t \left(\Phi^t_{\text{IoU}} \geq \Phi_{\text{thres}}\right)$$

$$FP = \sum_t \left(\Phi^t_{\text{IoU}} < \Phi_{\text{thres}}\right)$$

$$\begin{aligned} precision &= \frac{|🟦|}{|🟦 + 🟨|} \\ &= \frac{TP}{TP+FP} \end{aligned}$$

Figure 3.5: The computation of the number of True Positives (TP), False Positives (FP) and of *precision* is displayed. A tracker returns a TP for frame t if $\Phi^t_{\text{IoU}} \geq \Phi_{\text{thres}}$ and a FP if not. Usually $\Phi_{\text{thres}} = 0.5$. As for object detection, the *precision* is the ratio of the number of true positives to the number of all detections (which is the number of frames for all trackers that cannot detect the *absence* of an object).

the object tracking and object detection literature, the most common threshold for Φ_{IoU} is 0.5. Nevertheless, specifically in object detection, it is sometimes reasonable to average precision over different thresholds to emphasize detectors with a more accurate localization. A FN may only be returned by trackers capable of predicting the *absence* of an object.

In [153], the *precision* of a tracker is used to compare the tracker's performance with each other. It is calculated from the TP and FP as $\frac{TP}{TP+FP}$. Further measures that can be calculated on the basis of Φ_{IoU} are the true positive rate (TPR) and the true negative rate (TNR). In LTTD [222], TPR is computed as

$$\text{TPR} = \frac{1}{\tilde{n}} \sum_{i=t}^{n} \text{TP}_t, \tag{3.4}$$

and TNR as

$$\text{TNR} = \frac{1}{\tilde{n} - n} \sum_{i=t}^{n} \text{TN}_t, \tag{3.5}$$

where for frame t, TP_t is 1 if $\Phi_{\text{IoU}} \geq 0.5$, TN_t is 1 if the tracker correctly predicts the absence of the object (hence $R^T = \emptyset$ when $R^G = \emptyset$), n is the number of frames, and $\tilde{n}$ is the number of frames were the object is present. Hence, TPR computes the fraction of present objects that are detected as present and TNR the fraction of absent objects that are detected as such.

In general, Φ_{IoU} is the heart of the evaluation of nearly every tracking benchmark. It is the predominant accuracy measure and is also often used as a measure of the trackers' robustness [38, 40]. For example, in the VOT [116, 117] evaluation framework, a tracker failure is identified whenever $\Phi_{\text{IoU}} = 0.0$ [119]. Nevertheless, since bounding boxes are the dominant representation of the tracking ground truth, a tracker that predicts a pixel-precise representation of an object has no advantage in the accuracy scores. In contrast, a pixel precise representation has a lower Φ_{IoU} score than its bounding box. As a consequence, to compare methods that represent objects by a pixel-precise segmentation it is common practice to calculate the bounding box of approaches and then compute their Φ_{IoU} [32]. This is unfortunate since it prevents the community from moving towards more precise trackers.

3.3 Discussion

In general, bounding boxes are fast to label and describe the object location and roughly its shape. This enables the creation of very large tracking benchmarks with very diverse sequences. There is a large collection of benchmarks that are labeled with boxes and focus on the generalizability of long- and short-term trackers. Nevertheless, bounding boxes are often very crude approximations of objects [134] and cannot accurately capture an object's shape, location, or characteristics. As a consequence, the accuracy measured in the current benchmarks has a strict upper bound. Very accurate trackers have no advantage. Nevertheless, the used accuracy measures are not generally restricted to bounding boxes but can cope with arbitrary representations of the ground truth. Hence, the obvious next step is to improve the representation of the tracking ground truth to improve the accuracy scores of very precise trackers. Furthermore, the focus on a strong diversity in the test sequences requires an intense re-evaluation of the state-of-the-art trackers for specific tasks [153]. In the following chapter, we extend the accuracy measure to pixel-precise representations of the ground truth and present upper bounds for all trackers that are restricted to bounding boxes.

4 A Novel Approach for Evaluating the Accuracy of Object Trackers

In this chapter, we extend the common tracking evaluation protocols from bounding boxes to arbitrarily shaped regions. Hence, the ground truth data is labeled with pixel-precision. The extension of the measures is straightforward but introduces a bias for all box-based methods. Since this is the majority of the current state of the art, we introduce upper bounds for all box-based trackers and introduce the *relative intersection over union* (rIoU). The measure makes box-based trackers comparable to pixel-precise methods. Furthermore, we present a scale measure that is based on the upper bounds and measures the capabilities of trackers to estimate scale change. To support the presented measure, we introduce a dataset with pixel-precise data in the subsequent chapter. This chapter covers some of the work from the following publications: Böttger *et al.* [24] and Böttger and Follmann [23].

4.1 Pixel-precise Object Tracking Annotations

Axis-aligned bounding boxes are often a very coarse approximation of an object. As displayed in Fig. 4.1, articulated, rotated, and non-compact objects cannot be represented reasonably by an axis-aligned bounding box. To improve the representation of objects, some datasets thus approximate the objects by rotated boxes. For example, for the VOT 2015 challenge [118], the objects where manually labeled by experts with rotated boxes. The choice of the rotated boxes was not straightforward and required predefined heuristics. To reduce the uncertainty of the rotated boxes, the VOT 2016 challenge introduced an automatic process to compute the rotated boxes from the manual axis-aligned labels [116]. For this, the segmentation mask of each object is computed from the bounding box annotations automatically. Then, a rotated bounding box is computed that minimizes a cost function. The cost function punishes background pixels inside the box and object pixels outside of the box. Interestingly, the optimization process thus optimizes a different measure than the one that is used later to compute the accuracy of the trackers (Φ_{IoU}). However, rotated boxes add an unwanted bias to the evaluation.

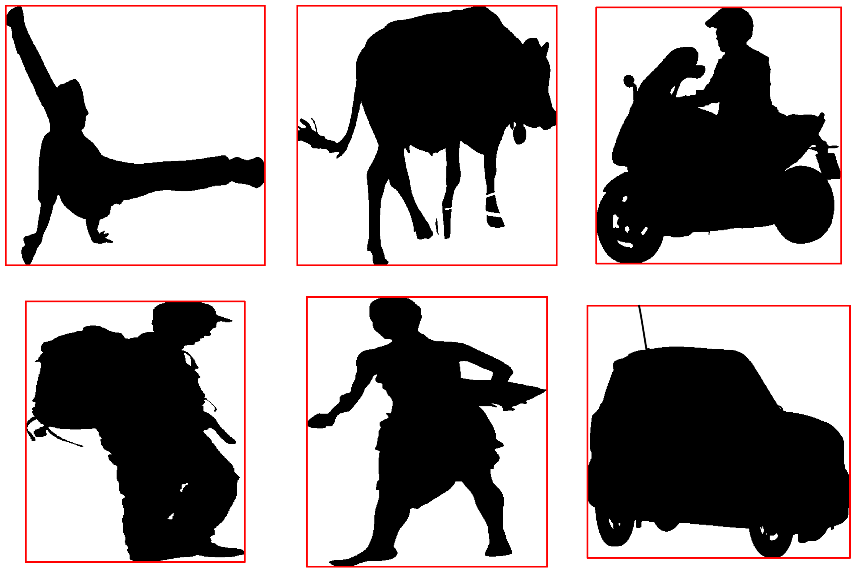


Figure 4.1: Example bounding boxes from selected frames from DAVIS [170]. In general, axis-aligned bounding boxes are very crude approximations.

Especially for symmetrical objects, the choice of the box is not straightforward. In Fig. 4.2, a circle is approximated by a collection of different boxes. All of the boxes have the same Φ_{IoU} with the segmentation. Nevertheless, the overlap of the bounding boxes with each other may be as low as $\frac{\sqrt{2}}{2} \approx 0.7071$. Hence, choosing any of the boxes as ground truth may introduce a disadvantage to a tracker that performs perfectly. The above example is not artificial. As shown in Fig. 4.3 (a), there are sequences in the VOT challenges with this exact problem.

In summary, using boxes as ground truth has two inevitable disadvantages:

1. The approximation of an object by a box is very crude. Especially articulated objects such as humans or animals cannot be approximated well by boxes. Although the approximation may be improved by using rotated boxes, the choice of the best suitable box can be highly ambiguous. For example, the bag in Fig. 4.3 (a) has multiple valid box approximations with the same overlap. Nevertheless, the IoU between two of the valid choices is only 0.71.

2. It is difficult to evaluate approaches that are not restricted to rotated or axis-aligned boxes on ground truth boxes without introducing an unwanted bias in the evaluation results. For example, the ground truth segmentation in Fig. 4.3 (c) only has an IoU of 0.35 with the red ground truth bounding box, but is a perfect approximation of the object itself.

Especially the latter point is of increasing concern. The recent advances of Fully Convolutional Networks (FCNs) for semantic segmentation [192, 212] have inspired approaches that are capable of tracking dense segmentations through image sequences in real-time. In One-Shot Video Object Segmentation (OSVOS), Caelles *et al.* [32] approximate the segmentations by bounding boxes to enable a comparison with the state-of-the-art bounding

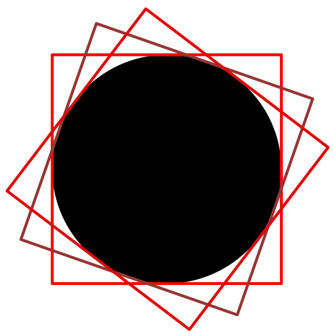


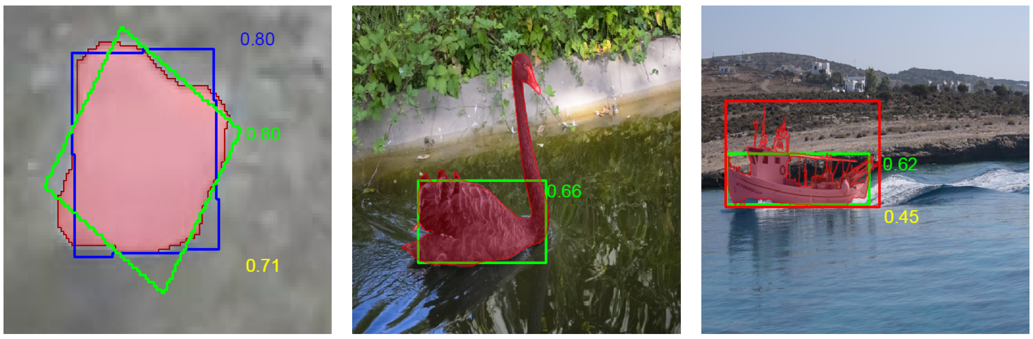
Figure 4.2: Using rotated boxes as ground truth may add ambiguities. In the above example, all of the red boxes have the same IoU with the circular region. Nevertheless, if only a single bounding box is stored for each object, trackers proposing any of the other boxes will obtain a suboptimal accuracy score, although they perfectly match the ground truth.

box tracker MDNET [157]. Hence, the accuracy gain of their pixel-precise method cannot be shown. This also accounts for approaches that approximate the object by general affine transformations and not only through rotated boxes [22, 27].

The obvious step to prevent the above mentioned problems is to directly use pixel-precise segmentations of objects as the ground truth. The segmentations capture every pixel belonging to the object and are generally unambiguous. Nevertheless, a pixel-precise representation of the ground truth significantly increases the required label effort. For example, the manual labels of the fine pixel-precise annotations of Cityscapes required more than 1.5h on average per image [56]. It should be noted that the Cityscapes labels are extremely precise and each object in the images is labeled comprehensively. Hence, for annotating a single object in a tracking sequence, this number is probably significantly lower. Nevertheless, it is not comparable to box annotations, which only require a few seconds per frame.

In spite of the large annotation efforts, in the last years, many different datasets with pixel-precise data have emerged. On the one hand, these datasets made data-hungry deep-learning-based techniques feasible in the first place. On the other hand, the impressive results from the deep-learning-based techniques and the fast saturation of small datasets have increased the pressure for large-scale high-quality datasets with pixel-precise annotations to exist. Hence, while densely annotated images were a rarity 4–5 years ago, numerous large-scale datasets exist today. Common examples include *ADE20K* [249], *D2S* [78], *Cityscapes* [56], *COCO* [134], *Places* [248], or *The Plant Phenotyping Datasets* [152]. The COCO 2014 dataset [134] alone includes more than 886 000 densely annotated instances of 80 categories of objects. The datasets have facilitated the state of the art in semantic segmentation, instance segmentation, object tracking, and object detection. The datasets related to object tracking are discussed in more details in Section 5.1.

Overall, the annotation effort needs to be set against the accuracy gain that may be obtained by the methods as well as in the evaluation. Although the most recent and the largest tracking datasets are still restricted to bounding boxes, datasets with dense labels have slowly started to emerge. For example, the DAVIS dataset [170] was released in



(a) `bag` from VOT2016 [116] **(b)** `blackswan` from DAVIS [170] **(c)** `boat` from DAVIS [170]

Figure 4.3: In image (a), both rotated boxes have an identical IoU with the ground truth segmentation. Nevertheless, their common IoU is only 0.71. Restricting the ground truth to boxes may introduce an undesired bias in the evaluation. In image (b), the best possible IoU of an axis-aligned box is only 0.66. Hence, for segmented data, it is difficult to use the absolute value of the IoU as an accuracy measure since it generally does not range from 0 to 1. Furthermore, although the object detection (green) in image (c) has an overlap of 0.62 with the ground truth segmentation, its IoU with the ground truth axis-aligned bounding box is only 0.45 and would be considered a false detection in the standard procedure.

2016. It has very precise manually labeled pixel-precise segmentations and consists of 50 short sequences. It was designed for video object segmentation, but can also be used for the evaluation of short-term object trackers. Furthermore, the segmentations used to generate the VOT2016 ground truths have very recently been released [226]. Hence, also in tracking, pixel-precise labels are starting to be available. Nevertheless, no evaluation protocol exists that enables a fair comparison of tracking approaches that are restricted to boxes and those that are not. For example, the VOT2016 Benchmark [116] generates plausible rotated boxes from densely segmented objects and the COCO 2014 Detection challenge [134] uses axis-aligned bounding boxes of the segmentations to simplify the evaluation protocol. As a consequence, approaches may have a relatively low IoU with the ground truth although their IoU with the actual object segmentation is the same (or even better) than that of the ground truth box (see Fig. 4.3(c)).

To eliminate the above problems, we propose to directly use the IoU of the object segmentation and the tracker proposal as an accuracy measure. This removes the ambiguities that are introduced by simplified representations such as boxes. It also rewards methods that are more accurate than the current state of the art.

4.2 Relative Intersection over Union

To improve the precision of the accuracy measure, we use a pixel-precise segmentation map of the object as ground truth. The accuracy is then computed as the IoU of the pixel-precise ground truth and the tracker proposal. This ensures that approaches that are not restricted to boxes or other simplified representations obtain higher accuracy scores.

Nevertheless, since the majority of the current trackers are restricted to boxes [117, 174, 222, 239], this introduces a problem when evaluating these trackers. As shown above and in Fig. 4.3, box trackers are generally unable to obtain an IoU of 1.0 for segmentations. Inevitability, the following question arises: What is the best IoU a box-based tracker can obtain for a scene where the ground truth is labeled with pixel precision? If this is known, the accuracy scores of the tracker can be normalized to range between 0 to 1 and the performance can be evaluated.

We introduce the *relative Intersection over Union* (rIoU) to enable a more precise measurement of the accuracy. The rIoU of a box $\mathcal{B}$ and a dense segmentation $\mathcal{S}$ is computed as

$$\Phi_{\text{rIoU}}\left(\mathcal{S}, \mathcal{B}\right) = \frac{\Phi_{\text{IoU}}(\mathcal{S}, \mathcal{B})}{\Phi_{\text{opt}}(\mathcal{S})}, \tag{4.1}$$

where Φ_{IoU} is the Intersection over Union (IoU) and Φ_{opt} is the best possible Φ_{IoU} a box can achieve for the segmentation $\mathcal{S}$. In comparison to the usual IoU (Φ_{IoU}), the rIoU measure (Φ_{rIoU}) truly ranges from 0 to 1 for all possible segmentations.

However, the optimal IoU Φ_{opt} does not only depend on the segmentation, but also on the traits of the tested tracker. A tracker that estimates rotated boxes can theoretically obtain a higher Φ_{opt} than one that does not. Similarly, a tracker that does not estimate the scale will have an even lower value for Φ_{opt}. In the most general case, the box $\mathcal{B}$ can be parameterized with 5 parameters

$$b = \left(r_c, c_c, w, h, \phi\right), \tag{4.2}$$

where r_c and c_c denote the row and column of the center, w and h denote the width and height, and ϕ the orientation of the box with respect to the horizontal-axis. An axis-aligned box can equally be parameterized with the above parameters by fixing the orientation to $0°$. A box with a fixed scale has a fixed value of w and h and only varies in the box location r_c and c_c. In our setting, the values of w and h are initialized in the first frame and denoted as w_0 and h_0.

The optimal box Φ_{opt} may be obtained in a fast and efficient optimization process. In the following, we present an efficient procedure and validate the quality of the boxes.

4.2.1 Optimization: Computing Optimal Boxes

For an arbitrary segmentation $\mathcal{S}$, the box with the best possible IoU can be computed as

$$\Phi_{\text{opt}}(\mathcal{S}) = \max_b \; \Phi_{\text{IoU}}\left(\mathcal{S}, \mathcal{B}(b)\right) \qquad s.t. \; b \in \mathbb{R}_+^4 \times [0°, 90°). \tag{4.3}$$

In general, there is no closed-form solution for Φ_{opt}. $\mathcal{S}$ does not need to be connected, nor fulfill any compactness constraints. Consequently, there are many examples where the optimal b is not even unique. However, we are merely interested in the value of Φ_{opt} and thus the uniqueness of b is of no concern.

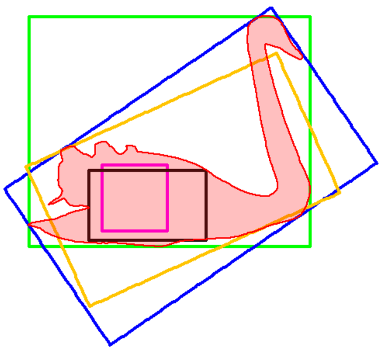


Figure 4.4: `blackswan` from DAVIS [170]. The initial values of the optimization process of (4.3) are displayed. We use the axis-aligned bounding box (green), the rotated bounding box (blue), the inner square of the largest inner circle (magenta), the largest inner axis-aligned box (black) and the rotated box with the same second order moments as the segmentation (orange).

For a convex segmentation, the above problem can be efficiently optimized with the method of steepest descent. To handle arbitrary, possibly unconnected, segmentations, we optimize (4.3) with a multi-start gradient descent with a backtracking line search. The gradient is approximated numerically by the central difference. The central difference is given by

$$\delta_h[\Phi_{\text{IoU}}(b)] = \frac{\Phi_{\text{IoU}}(b+h) - \Phi_{\text{IoU}}(b-h)}{2h}, \tag{4.4}$$

where $h \in \mathbb{R}^5_+$. Here $\Phi_{\text{IoU}}(b) = \Phi_{\text{IoU}}(\mathcal{S}, \mathcal{B}(b))$, since the equation is valid for all segmentations $\mathcal{S}$. It is reasonable to adapt the values of h_i to the scale of the parameter that is being optimized. Hence, we use a larger value for the box width w and height h than for the center position x and y and the angle ϕ.

Since the problem is non-convex, the gradient descent is prone to getting caught in local maximum. Hence, to solve (4.3) with a multi-start gradient decent, good initial values for b are required. The initial boxes are selected to be close to theoretical upper and lower bounds for $\Phi_{\text{IoU}}(\mathcal{S}, \mathcal{B}(b))$. We use a diverse set to improve the quality of the obtained values of Φ_{opt}. The used initial boxes are displayed in Fig. 4.4. The largest axis-aligned inner box (black) and the inner box of the largest inner circle (magenta) are completely within the segmentation. Hence, in the optimization process, they will gradually grow and include background if it improves $\Phi_{\text{IoU}}(\mathcal{S}, \mathcal{B}(b))$. On the other hand, the bounding boxes (green and blue) include the complete segmentation and will gradually shrink in the optimization to include less of the segmentation. The rotated box with the same second order moments as the segmentation (orange) serves as an intermediate starting point [182].

If the optimization process converges to the same optimal for the different initial values, we assume they converged to the optimal value of (4.3). However, if the initial values converge to different optima, further steps are taken. In these cases, the different

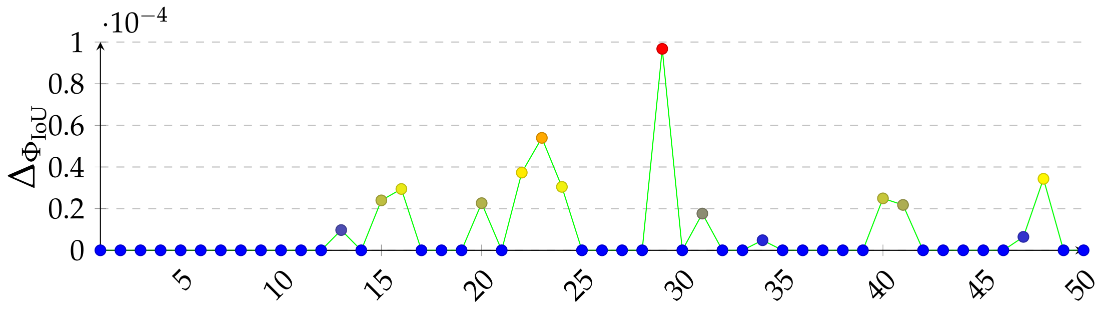


Figure 4.5: The absolute difference $\Delta_{\Phi_{\mathrm{IoU}}}$ of the exhaustively determined best axis-aligned box and the optimized axis-aligned box for a selected frame in each of the 50 DAVIS [170] sequences. Most boxes are identical, only a handful of boxes are marginally different (< 0.0001).

obtained optima Φ^i_{opt} define a multi-dimensional interval where the optimal box is likely to lie. The interval is constructed by considering the component-wise minimum and maximum of the different optimal boxes $\mathcal{B}^i$. To be more robust, the boundaries of the interval are extended by a fixed factor. Then, initial values to start the optimization process are sampled randomly. In our experiments we used 50 random samples. The optimization process with the highest resulting Φ_{IoU} is then selected as the optimum. Although this leads to many optimizations, the approach is still very efficient. A single evaluation of $\Phi_{\mathrm{IoU}}(\mathcal{S}, \mathcal{B})$ only requires around 0.04 ms on average for the segmentations within the DAVIS [170] dataset in HALCON[1] on an IntelCore i7-4810 CPU @2.8GHz with 16GB of RAM with Windows 7 (x64). As a consequence, the optimization of Φ_{opt} requires an average of 1.2 s for the DAVIS [170] and 0.7 s for the VOT2016 [116] segmentations.

The optimization of the IoU for axis-aligned rectangles bears some similarity to the 2D maximum subarray problem [3]. This might make an alternative algorithmic approach to the optimization possible. However, a straightforward adaptation of methods is difficult, since these methods rely on the additive nature of the maximum subarray problem. In contrast, the IoU is inherently non-linear due to the quotient in its definition.

4.2.2 Validation of Optimal Boxes

To validate the optimization process, we compare the Φ_{opt} values obtained from the numerical optimization of 4.3 to ones obtained exhaustively. For this, we selected the most difficult frame from each of the sequences in the DAVIS dataset [170]. The difficulty was determined by measuring the difference of Φ_{opt} for the different initial values.

The exhaustive computation of Φ_{opt} is a computationally elaborate task. The computation requires around 1 hour per segmentation $\mathcal{S}$, although the single evaluations for $\Phi_{\mathrm{IoU}}(\mathcal{S}, \mathcal{B})$ are extremely fast. The results for the axis-aligned boxes are displayed in Fig. 4.5. They indicate that the optimization is generally very close or identical to the exhaustively determined boxes. Only for a few examples the Φ_{IoU} values are marginally different. Although no tracker achieved a better score than Φ_{opt} in our experiments, we

[1]MVTec Software GmbH, `https://www.halcon.com/`

nonetheless compute the rIoU as

$$\Phi_{\text{rIoU}}(\mathcal{S}, \mathcal{B}) = \min\left(\frac{\Phi_{\text{IoU}}(\mathcal{S}, \mathcal{B})}{\Phi_{\text{opt}}(\mathcal{S})}, 1\right). \tag{4.5}$$

For the exhaustive determination of the optimal rotated boxes, one of the restrictions we can make is that the area must at least be as large as the smallest inner box of the segmentation and may not be larger than the bounding rotated box. Nevertheless, even with further heuristics, the number of candidates to test is in the number of billions for a single segmentation from the DAVIS dataset. Given a pixel-precise discretization for r_c, c_c, w, h and a 0.5° discretization of ϕ, it was impossible to find boxes with a better IoU than the optimized rotated boxes in the validation set. This is mostly due to the fact that the subpixel precision of the parameterization (especially in the angle ϕ) is of paramount importance for the IoU of rotated boxes.

4.3 Theoretical Trackers

The new accuracy measure can be used to generate three very expressive theoretical trackers. The concept of theoretical trackers was first introduced by Čehovin *et al.* [40] as an *"excellent interpretation guide in the graphical representation of results."* The theoretical trackers provide reference points for an evaluation sequence and put the results of the evaluated trackers into context. For example, one of the theoretical trackers always reports the region of the object to equal the image size of the sequence. Hence, the tracker is perfectly robust, since it never has an overlap of 0 with the ground truth ($\Phi_{\text{IoU}} \neq 0$). When weighing between accuracy and robustness, this tracker is thus the lowest reasonable bound of the accuracy. A further theoretical tracker always predicts the center position of the object correctly. However, the size of the object is fixed in the first scale. This tracker represents a practical performance limit for trackers that do not adapt the scale of the object.

In our case, we use the boxes with an optimal IoU to create upper bounds for the accuracy of trackers that are restricted to boxes. We introduce three theoretical trackers that are obtained by optimizing (4.3) for a complete sequence. Given the segmentation $\mathcal{S}$, the first tracker returns the best possible axis-aligned box (`box-axis-aligned`), the second tracker returns the optimal rotated box (`box-rot`), and the third tracker is similar to the tracker proposed by Čehovin *et al.* [40]: It returns the optimal axis-aligned box with a fixed scale (`box-no-scale`). The scale is initialized in the first frame with the scale of the box determined by `box-axis-aligned`.

The theoretical trackers normalize the IoU for a complete sequence. This enables a fair interpretation of a tracker's accuracy and removes the bias from the box-world assumption. The Φ_{IoU} scores for the `motorbike` sequence from DAVIS [170] are displayed in Fig. 4.6. Since the motorbike is driving towards the camera, there is an increasing gap between `box-no-scale` and the two scale-adaptive tracker `box-axis-aligned` and `box-rot`. In general, the difference between `box-axis-aligned` and `box-rot` is

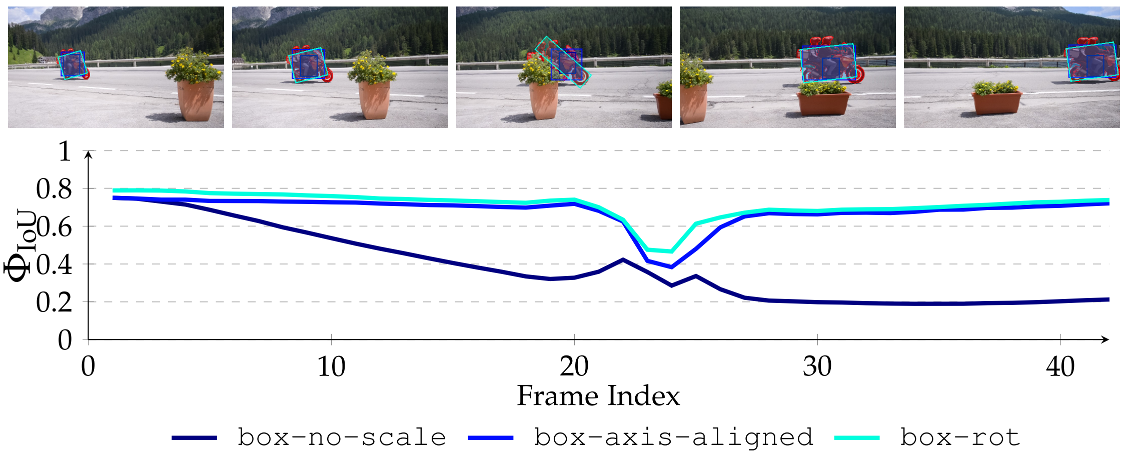


Figure 4.6: `motorbike` from DAVIS [170]. The increasing gap between the `box-no-scale` and the other two theoretical trackers indicates a scale change of the motorbike. The drop in all three theoretical trackers around frame 25 indicates that the object is being occluded. The best possible IoU is never above 0.80 for the complete sequence.

	Φ_{IoU}
VOT2015	0.577
VOT2016	0.651
`box-no-scale`	0.512
`box-axis-aligned`	0.722
`box-rot`	0.760

Table 4.1: Φ_{IoU} of the VOT2016 and VOT2015 ground truths [116] and of the theoretical tracker `box-no-scale`, `box-axis-aligned`, and `box-rot`, for the VOT2016 segmentations [226]. The VOT 2015 ground truths where obtained manually, while the VOT 2016 ground truths where generated directly from the VOT2016 segmentations [226].

usually not too extreme.

To get a perspective on how well box trackers can compete on pixel-precise ground truth data, we compare the Φ_{IoU} scores of the theoretical trackers to the VOT 2016 segmentations, VOTSEG [226] in Table 4.1. As shown, the average Φ_{IoU} scores are quite low. Hence, no box-based tracker is able to obtain an average Φ_{IoU} over 0.76 on the VOT 2016 segmentations. A tracker that does not estimate the scale is even bound by an average Φ_{IoU} of 0.512, which is only marginally above the standard Φ_{thres} of 0.5.

The three different theoretical trackers make it possible to interpret a tracking sequence without the need of by-frame labels. As is displayed in Fig. 4.6, the difference between the `box-no-scale`, `box-axis-aligned`, and `box-rot` trackers indicates that the object is undergoing a scale change. Furthermore, the decreasing IoUs of all theoretical trackers indicate that the object is either being occluded or deforming to a shape that can be approximated less well by a box. For compact objects, the difference of the `box-rot` tracker and the `box-axis-aligned` tracker indicates a rotation or change

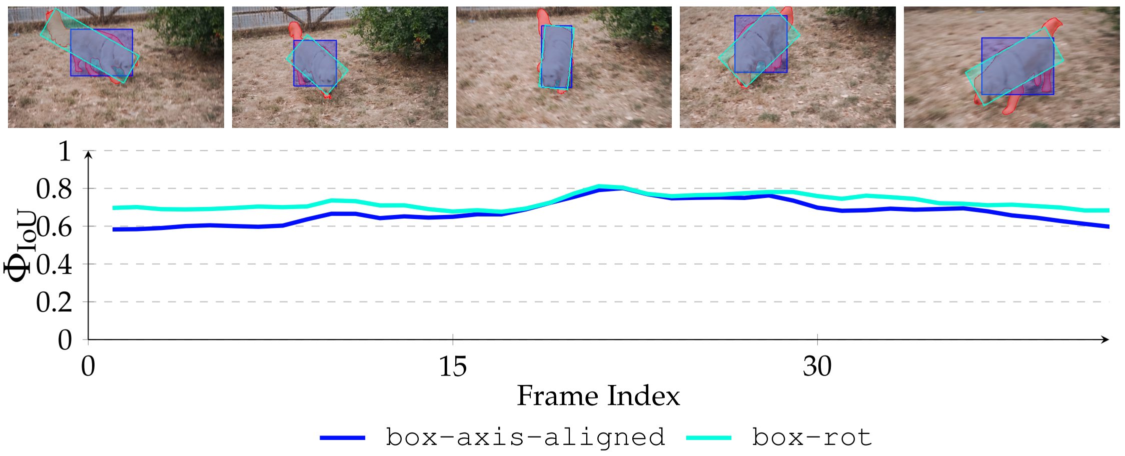


Figure 4.7: `dog` from DAVIS [170]. The gaps between the `box-axis-aligned` and `box-rot` tracker indicate a rotation of the otherwise relatively compact segmentation of the dog. The best possible IoU is never above 0.80 for the complete sequence. For clarity the `box-no-scale` tracker is omitted.

of perspective, as displayed in Fig. 4.7. In [23], the theoretical trackers are used to train a deep neural network and automatically label attributes of a sequence such as occlusion and rotation.

4.4 Measuring Scale Changes

The presented theoretical trackers enable us to compute a new scale measure. The measure captures how well a tracker can cope with scale changes. It builds on the fact that scale changes within a sequence result in a significant drop in Φ_{IoU} of `box-no-scale`, while `box-axis-aligned` remains unaffected, as can be seen nicely in Fig. 4.8. The derivative of the difference of `racing` curve (γ) is visualized in Fig. 4.9 and is a reliable indicator of the scale-change. It is the foundation of the presented scale measure. Whenever it exceeds a threshold, it is assumed that the scale is changing. A further threshold (e.g., 0.005) may be used to suppress minor scale changes and to compensate for noise in the segmentations. Moreover, a prior smoothing of the derivatives with a Gaussian function with $\sigma = 3$ further increases the robustness to noise.

For the frames that are identified as changing scale, we calculate the scale score s. For each of these frames, we compare the change of the size of the tracker predictions to that of the `box-axis-aligned` tracker. If both have the same direction, we assume the tracker is successfully registering a scale change. To make the approach as independent from the accuracy measure as possible, we do not regard the magnitude of the size changes, but merely their sign. Please note that the change of the size of the ground truth boxes or segmentations could equally be used to estimate the "*ground truth*" scale change. Nevertheless, we chose to use the `box-axis-aligned` tracker to obtain an estimate of the scale score for two reasons. First of all, the segmentations themselves are very noisy and secondly, by using the `box-axis-aligned` scale change, it is possible to bring the

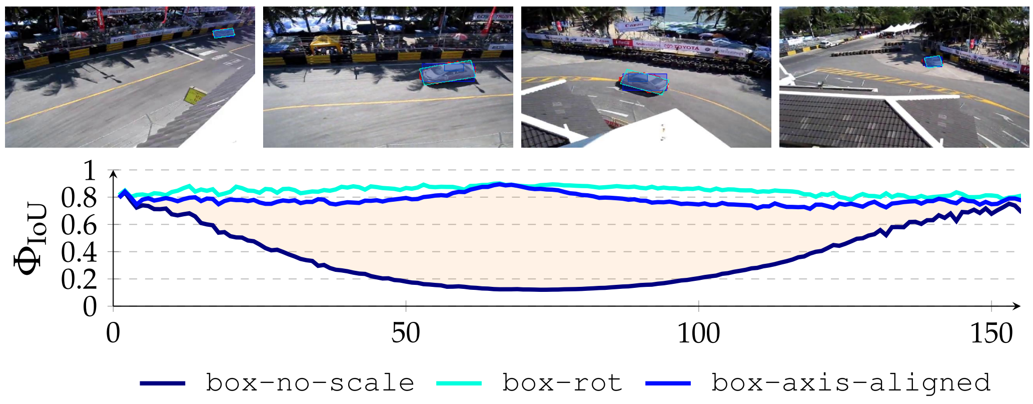


Figure 4.8: `racing` from VOT2016 [116]. The increasing gap between the `box-no-scale` and the other two theoretical trackers indicates a scale change. The best possible IoU is never above 0.80.

tracker scale scores into relation to the scale score of the VOT2016 ground truth boxes.

The scale score s for a sequence is computed as

$$s = \frac{1}{\tilde{n}} \sum_{i}^{n} \delta_{\widehat{\mathrm{sgn}}(\mathrm{size}(\mathcal{T}_i)'),\widehat{\mathrm{sgn}}(\mathrm{size}(\mathcal{G}_i)')} \tag{4.6}$$

where $\mathcal{T}_i$ is the tracker prediction and $\mathcal{G}_i$ is the box of the scale-adaptive theoretical tracker (`box-axis-aligned`). $\delta_{i,j}$ is the Kronecker delta, which is 1 if the variables are equal, and 0 otherwise:

$$\delta_{i,j} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases} \tag{4.7}$$

The derivative of the size of the region $\mathcal{R}$ is denoted as $\mathrm{size}(\mathcal{R})'$. The derivative is approximated by central differences. $\widehat{\mathrm{sgn}}$ is an approximation of the signum function sgn. It is not strictly 1 and -1 when the input is $\neq 0$, but only once a set threshold has been exceeded (we use 0.005 in the experiments) and 0 otherwise. Furthermore, $\tilde{n}$ is the number of frames where $\widehat{\mathrm{sgn}}(\mathrm{size}(\mathcal{G}_i)')$ is $\neq 0$.

Trackers that do not estimate the scale have a scale score of 0 ($\mathrm{size}(\mathcal{T}_i)' = 0 \;\; \forall \in n$) and a perfect scale-adaptive tracker has a score of 1. Please note that no per-frame labels are required and the scale score is, by construction, uncorrelated to the accuracy or robustness overlap. In the evaluation, we calculate the scale score without reinitialization on tracker failure and ignore the frames where the tracker has failed completely (hence $\Phi_{\mathrm{IoU}} = 0$).

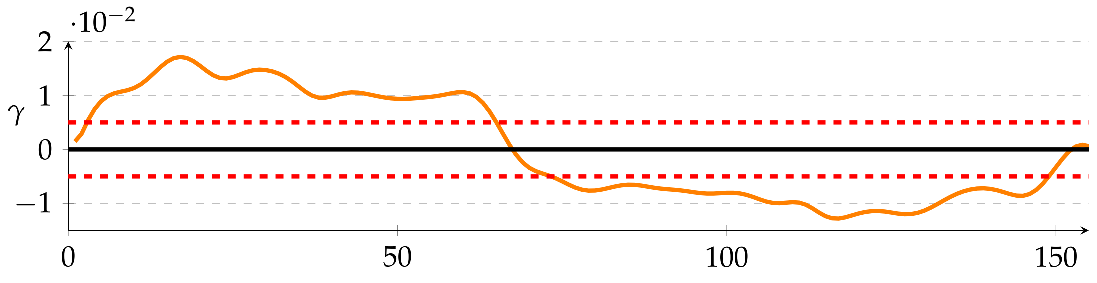


Figure 4.9: The derivative of the difference between the `box-no-scale` and `box-axis-aligned` tracker from Fig. 4.8. The magnitude of the derivative is a reliable indicator for scale change. To suppress minor scale changes and to compensate for noise in the segmentations, we require the magnitude to exceed the fixed threshold of 0.5×10^{-3} (visualized as the red-dotted lines).

4.5 Experiments

We evaluate state-of-the-art trackers with the new accuracy measures on the DAVIS2016 [170] and the VOT 2016 segmentations, denoted by VOTSEG [226]. Both datasets have pixel-precise labels. While the DAVIS [170] dataset has very precise labels in 60 short sequences, the segmentations of VOTSEG are less accurate but the scenes are longer. To make it easy to reproduce the results, the complete code of the evaluation system and the evaluation scripts have been make available to the community[2]. The evaluation framework is constructed such that it is easy to add new trackers from MATLAB[3], Python[4], or HALCON[5].

The evaluation was restricted to open-source state-of-the-art trackers. A reimplementation of many tracker algorithms is difficult, since the implementation details are not apparent from the publications themselves. The trackers are selected to be efficient, high performing and different in their nature. Hence, we selected the basic Kernelized Correlation Filter (KCF) [97] tracker since it was a top ranked tracker in the VOT2014 challenge. It does not estimate the scale of the object. The Discriminative Scale Space Tracker (DSST) [62] is essentially an extension of KCF that can handle scale changes and generally outperforms KCF [97]. As further axis-aligned trackers, we include ANT [39], L1APG [11], STAPLE [16], and the best-performing tracker from the VOT2016 challenge and top performer of VOT 2017, the Continuous Convolution Filter (CCOT) from Danelljan *et al.* [66]. CCOT was extended very recently to the Efficient Convolution Operators for Tracking (ECO) [61]. Furthermore, we include the LGT [37] tracking since it is one of the few open-source trackers that estimates the object position as a rotated box.

In the experiments, we do not reinitialize the tracker's when they move off target. We are primarily interested in the accuracy and not in the trackers robustness. Please note that the accuracy of the robustness measure is also improved when using segmentations.

[2] https://www.mvtec.com/company/research/
[3] The MathWorks, Inc., https://www.mathworks.com/
[4] Python Software Foundation, https://www.python.org/
[5] MVTec Software GmbH, https://www.halcon.com/

Table 4.2: The average (Φ_{IoU}) and relative IoU (Φ_{rIoU}) values for DAVIS [170] and VOTSEG [226]. To get a perspective of the Φ_{IoU} values, the results for the top performing segmentation technique OSVOS [32] are added for DAVIS

	DAVIS		VOTSEG	
	$\varnothing\Phi_{\text{IoU}}$	$\varnothing\Phi_{\text{rIoU}}$	$\varnothing\Phi_{\text{IoU}}$	$\varnothing\Phi_{\text{rIoU}}$
Axis-aligned boxes (fixed scale)				
KCF [97]	0.40	**0.78**	0.23	0.45
Axis-aligned boxes				
ANT [39]	0.41	0.64	0.26	0.38
CCOT [66]	0.47	0.73	0.42	**0.58**
DFST [180]	0.41	0.64	0.27	0.38
DPCF [1]	0.38	0.59	0.29	0.41
DSST [62]	0.43	0.67	0.24	0.33
ECO [61]	**0.48**	0.76	**0.42**	**0.58**
L1APG [11]	0.40	0.63	0.18	0.25
STAPLE [16]	0.45	0.71	0.33	0.46
Rotated boxes				
LGT [37]	0.40	0.60	0.25	0.34
Segmentations				
OSVOS [32]	0.80	-	-	-

The failure cases ($\Phi_{\text{IoU}} = 0$) are identified earlier since Φ_{IoU} is zero when the tracker has no overlap with the segmentation and not with a bounding box abstraction of the object (which may contain a large amount of background; see, e.g., Fig. 4.3).

The average Φ_{IoU} scores of all trackers and for both datasets are displayed in Table 4.2. To gain a perspective on the error made by the box trackers, we also add the results on DAVIS from One-Shot Video Object Segmentation (OSVOS) [32] to the evaluation. Of course, the computation of Φ_{rIoU} makes no sense for a tracker that is estimating the segmentation directly.

Relative Intersection over Union Φ_{IoU} In Table 4.2, the average Φ_{IoU} and Φ_{rIoU} values for the DAVIS and the VOT2016 segmentations are displayed. Each tracker is normalized with the Φ_{opt} value of the theoretical tracker that has the same abilities. Hence, the KCF tracker is normalized with the `box-no-scale` tracker, the LGT tracker with `box-rot`, and the others with `box-axis-aligned`. By these means, it is possible to observe how well each tracker is doing with respect to its abilities. For the DAVIS dataset, the KCF, ANT, L1APG, and LGT trackers all have the same absolute IoU, but when normalized by Φ_{opt}, differences are visible. Hence, it is evident that the KCF is performing very well, given the fact that it does not estimate the scale. On the other hand, the LGT tracker, which has three more degrees of freedom, is relatively weak. A more detailed example analysis of the `bmx-trees` sequence from DAVIS [170] is displayed in Fig. 4.10. Please

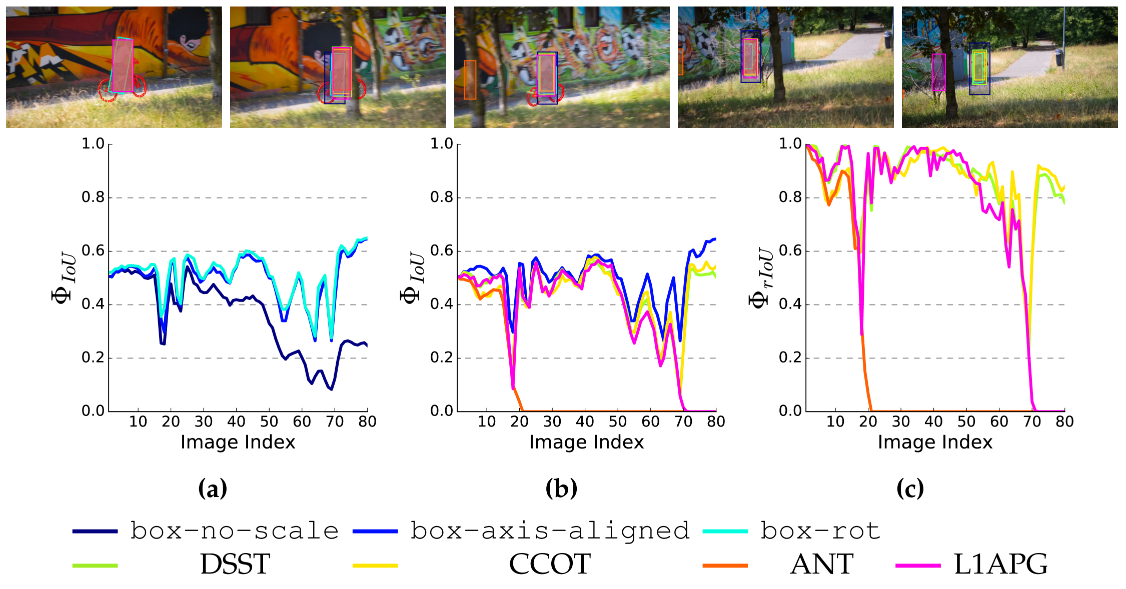


Figure 4.10: `bmx-trees` from DAVIS [170]. On the left, differences between `box-no-scale` and `box-axis-aligned` indicate that the object is changing scale and is occluded at frame 18 and around frames 60-70. In the middle plot, we compare the Φ_{IoU} of the axis-aligned box trackers and `box-axis-aligned`. The corresponding Φ_{rIoU} plot is shown on the right. It becomes evident that the ANT tracker fails when the object is occluded for the first time and the L1APG tracker at the second occlusion. The Φ_{rIoU} shows that DSST and CCOT perform well, while Φ_{IoU} would imply they are weak.

note that the significantly higher difference between Φ_{IoU} and Φ_{rIoU} for KCF compared to the other trackers is due to the different normalization factors used. The optimal Φ_{IoU} value for a box with fixed size is usually considerably lower than for a general axis-aligned box.

For the VOT2016 dataset, the overall accuracies are significantly worse than for DAVIS. On the one hand, this is due to the longer, more difficult sequences, and, on the other hand, due to the less accurate and noisier segmentations (see Fig. 4.11). Nevertheless, Φ_{rIoU} allows a more reliable comparison of different trackers. For example, ANT, LGT, and DSST have almost equal average Φ_{IoU} values, while ANT clearly outperforms LGT and DSST with respect to Φ_{IroU}. Again, we can see that the KCF tracker is quite strong when considering the fact that it cannot estimate the scale.

The scale score s We evaluated the new scale measure s for the DAVIS and VOT2016 segmentations. Because there are ground truth boxes available for the VOT sequences, we also evaluate the manually annotated boxes from the 2015 ground truths and the automatically obtained boxes from the 2016 ground truths. For DAVIS there are no box labels available. We use the size change of the `box-axis-aligned` tracker to compute $\text{sgn}(\text{size}(\mathcal{G}_i)')$ in the VOT2016 sequences. We refrained from using the segmentations directly since they are very noisy. For DAVIS, we use the segmentations directly.

The results are displayed in Table 4.3. For all trackers, the scale score is significantly higher for the DAVIS sequences than for the VOT sequences. This is not surprising since

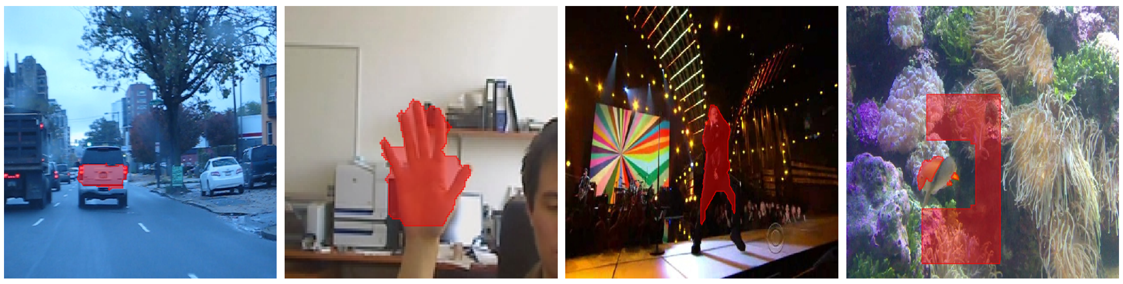

(a) car1 (b) hand (c) singer2 (d) fish3

Figure 4.11: Examples from VOTSEG [116] where the segmentations are degenerated, sometimes due to motion blur (e.g., (a) and (b), a weak contrast of the object and its background (c), or where the semi-automatic segmentation failed completely (d).

Table 4.3: The new scale measure (4.6) is displayed. On the left for the theoretical tracker `box-axis-aligned`, `box-no-scale`, and the VOT 2015 and VOT 2016 ground truths. On the right the scale score s for a collection of trackers is displayed. Even the top performing trackers (CCOT and ECO) have a relatively low scale score s (e.g., in comparison to the VOT2016 ground truths).

scale score s

VOTSEG				DAVIS
ground truths		tracker		tracker
`box-axis-aligned`	1.00	ANT [39]	0.36	0.57
`box-no-scale`	0.00	CCOT [66]	**0.54**	**0.69**
VOT2015	0.69	DFST [180]	0.32	0.51
VOT2016	0.81	DPCF [1]	0.31	0.54
		DSST [62]	0.29	0.62
		ECO [61]	0.52	0.66
		KCF [97]	0.00	0.00
		L1APG [11]	0.30	0.55
		STAPLE [16]	0.37	0.59

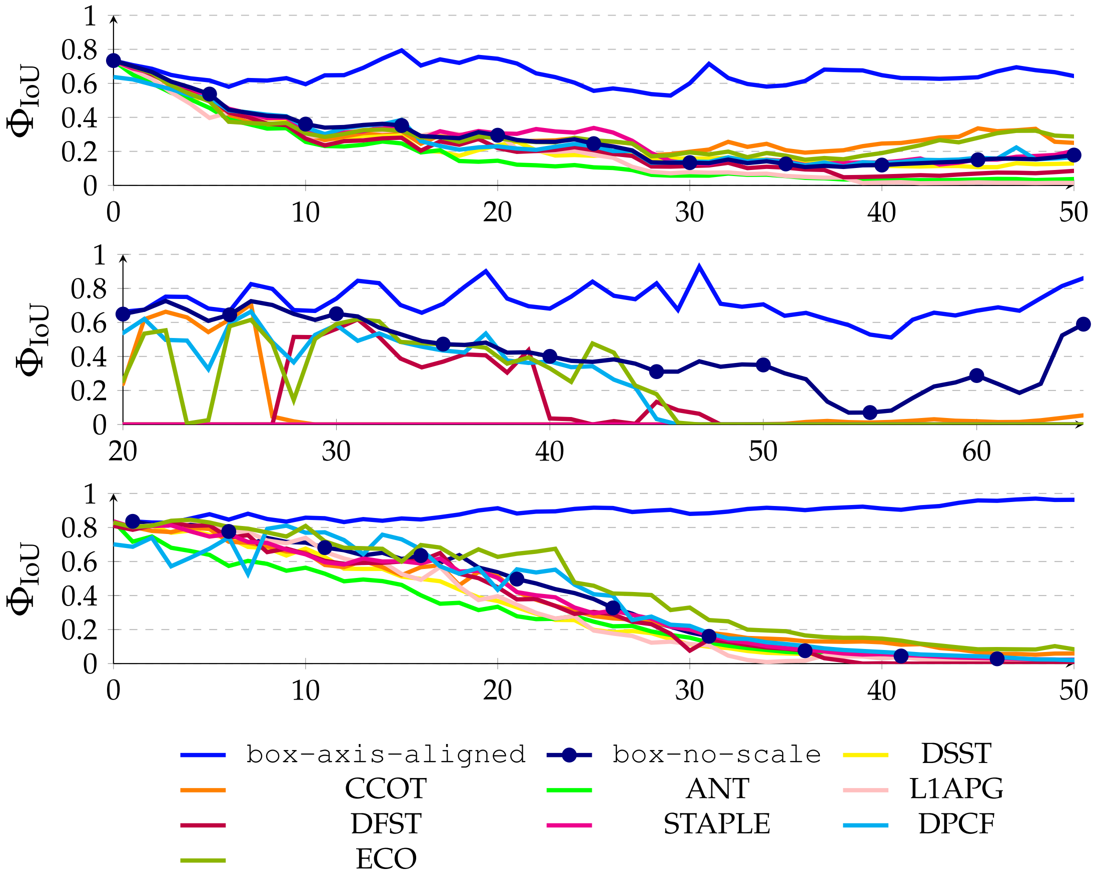


Figure 4.12: `bmx` (top), `book` (middle) and `singer3` (bottom) from VOT2016 [116]. None of the tested trackers can cope with the scale change. This becomes apparent since none of them can seriously outperform `box-no-scale` and all fail early on in the sequences.

the scenes are much shorter and there is significantly less scale change within them. In general, the CCOT tracker has the best scale adaption capabilities. Although it is outperformed in terms of Φ_{IoU} by ECO, it is stronger in sequences where there is only a modest amount of scale change. Nevertheless, there is significant room for improvement in sequences with strong scale change. As shown in Fig. 4.12, there are many examples where none of the tested trackers are able to correctly estimate the object size when there is significant scale change. In the shown examples, even the `box-no-scale` is able to outperform all of the tested trackers. The scale score s of the respective sequences is well below 0.25 for all trackers.

In general, the scale adaptation appears to be a problem of current state-of-the-art approaches. The scale scores are low and have significant room for improvement. The observation is not very surprising. When an object undergoes a strong scale change, there is often also a strong appearance change since new details become visible or disappear.

4.6 Discussion

The advance of tracking to more elaborate and robust techniques has two coherent development directions: (1) the technologies behind the tracking approaches and (2) the methodologies used to evaluate the tracking systems. We extend the tracking evaluation to support pixel-precise ground truths. This enables us to correctly measure the accuracy of newer, pixel-precise methods without placing a disadvantage on them by using a box-based evaluation. The prerequisite of the proposed evaluation is the availability of pixel-precise ground truth data. Although the DAVIS dataset is highly precise, it is restricted to very short sequences. On the other hand, the VOT 2016 sequences are more complex and significantly longer on average. Nevertheless, the segmentations are very coarse. Hence, although the community has presented many large-scale datasets lately, the quality of the ground truth needs to be improved.

The applicability of the proposed optimal boxes for a segmentation is not restricted to tracking. In general, it could also help to also improve the evaluation of object detection approaches. The optimal representation of the objects with respect to the evaluation metrics might also help to improve the object detectors themselves. Hence, instead of training on axis-aligned bounding boxes of objects, the optimal boxes could be used.

5 Datasets and Evaluation Protocol

The above accuracy measures require pixel-precise annotations. For tracking, the two most relevant datasets are DAVIS [170] and the segmentations of the VOT 2016 benchmark, VOTSEG [226]. However, although the DAVIS sequences have very accurate pixel-precise labels, they are very short and have limited complexity. The VOT 2016 sequences are significantly longer, more difficult, and complex. Nonetheless, they have very coarse segmentations that were obtained semi-automatically. Hence, to evaluate long-term trackers accurately, new data is necessary.

In general, pixel-precise ground truth annotations are very tedious to obtain. Nevertheless, a collection of datasets exists. However, most of them are not designed to evaluate trackers, but rather focus on semantic segmentation and instance-aware segmentation [56, 78, 79, 134, 152, 248, 249]. To reduce the workload of obtaining highly accurate pixel-precise annotations, a new research direction is to obtain the ground truth data from simulators or graphics debuggers. Although this creates artificial data, the annotations can be obtained for free. Some of the datasets have temporally connected images and could be extended for evaluating trackers. In Section 5.1, we highlight the relevant datasets. In Section 5.2, we present a new dataset with pixel-precise annotations that have the same quality as those in DAVIS. In contrast, the sequences are considerably longer and more complex. This chapter is concluded with a discussion in Section 5.3.

5.1 Related Work: Segmentation Datasets

Since the research progress depends greatly on the existence of large and diverse datasets, a significant effort has gone into the development of new datasets with pixel-precise semantic labels. The datasets can roughly be split into two groups: real-world and synthetic datasets. While the real-world datasets require immense label effort, the pixel-precise labels are usually generated automatically in synthetic datasets. However, the images in the synthetic datasets are synthetic and merely an approximation of the real-world.

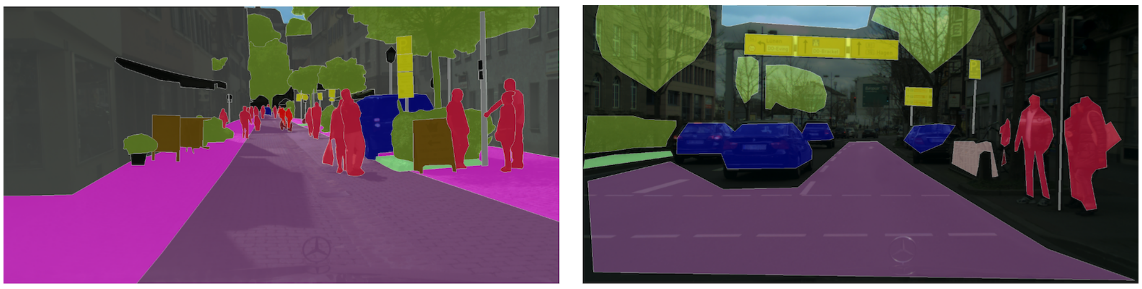


Figure 5.1: A part of the Cityscapes dataset [56] is annotated with very fine labels (left), and the other part with coarse labels (right).

Real-world Dataset A particularly popular and challenging application for semantic segmentation methods involves self driving cars. The first dataset in the domain of self-driving cars with semantic segmentations was the Cambridge-driving Labeled Video Database (CamVid) [30]. The camera was set up on the dashboard of a car, with a similar field of view as that of the driver. The dataset consists of a 10 minute video sequence captured with 30 Hz. 701 frames of the video have pixel-precise semantic labels. They are spaced evenly across the sequence (≈ 1 Hz). Hence, although there is a temporal connection between the frames, it is very coarse and difficult to use for tracking. Moreover, the dataset does not contain any instance-level annotations.

In 2016, the Cityscapes dataset [56] was released. It includes 25 000 annotated images with instance-wise semantic labels. The dataset is very diverse and obtained by driving through 50 different cities, at different times of day, with different weather conditions, and at different seasons. A fifth of the images (5 000) are annotated very precisely while the rest (20 000) is labeled with coarse labels. An example from both classes is displayed in Fig. 5.1. Although the images are labeled with instance-level segmentations, the time between the individual high-quality frames is very large. Hence, as for CamVid, this makes it difficult to use the dataset to evaluate trackers accurately.

The KITTI Vision Benchmark Suite [84, 85] also includes some instance-level semantic segmentation annotations. In general, the KITTI dataset contains over 40 000 stereo image pairs taken from a car driving through various European cities. The dataset is used to benchmark various different computer vision tasks, such as stereo reconstruction, optical flow, visual odometry, 3D object detection, and 3D object tracking. The instance-level segmentations are only available for 430 images [2]. Nevertheless, various extensions of the dataset have been proposed. For example, Siam *et al.* [196] present the KITTI MoSeg dataset. It contains ground truth annotations for moving 3D object detection. Their approach enables to generate pseudo ground truths for the segmentations as well. However, these are very coarse and cannot be used to accurately measure the Φ_{IoU} with reasonable precision.

Apart from autonomous driving, little semantic segmentation data with a temporal connection exists. A common approach is to build 3D scenes and label the 3D objects within. The scenes can be further augmented by 3D models of different objects. Through interaction with the scene, 2D snapshots of the scene and the respective segmentation

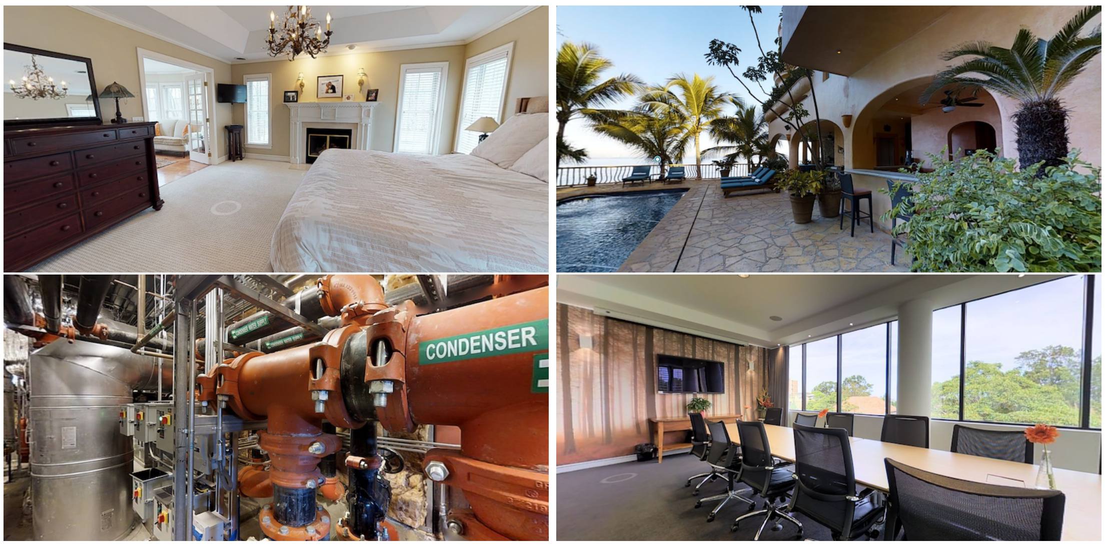


Figure 5.2: Example reconstructions from the Matterport3D Research Dataset [42].

maps can be generated. For example, very recently, Matterport[1] released the Matterport3D Research Dataset [42]. It is a large-scale RGB-D dataset containing over 10 000 panoramic views from almost 200 000 RGB-D images of 90 different building-scale scenes. The annotations were generated by using a crowd-sourced painting interface. The annotations include surface reconstructions, camera poses, and 2D and 3D semantic and instance-level segmentations. The 3D models in the dataset can be used to generate reconstructions interactively. A few example reconstructions are shown in Fig. 5.2. There are no moving objects in the scene, but tracking sequences could be generated from the ego-motion of the observer. Nevertheless, the possible complexity that can be generated within the tracking sequences is limited.

Similarly, ShapeNet [43] is an ongoing effort to establish a richly annotated, large-scale dataset of 3D shapes. Realistic 3D models are critical for creating real-life and diverse virtual worlds. Various works in this direction exist. A prominent example is the dataset of Choi *et al*. [50], which includes more than 10 000 3D scans of real objects, which have been generated from RGB-D data acquired with PrimeSense Carmine cameras.

In general, 3D models are appealing since they can significantly reduce the label effort. The models in the Matterport3D Research Dataset are generated from real images that were acquired with a Matterport Pro 3D Camera. Nevertheless, it is reasonable to attempt to generate ground truth data from synthetic 3D models altogether. A very recent trend in computer vision is thus to generate images and highly precise annotations from a graphics engine. In the following section, we highlight the most prominent examples and comment on the possibilities they create and their general limitations.

Synthetic Data In the last few years, the use of synthetic data has increased significantly. This is especially due to the rise of data-hungry deep learning techniques for many computer vision tasks such as object detection, semantic segmentation, pose estimation,

[1]https://matterport.com

and scene understanding [4, 46, 92, 95, 112, 144, 167, 209]. We group synthetic data into two overlapping groups and present a few examples that can be related to the synthetic dataset we propose later. The first group includes fixed datasets that are generated by a graphics engine. The second group is more general. It includes graphic simulators or applications that allow to interact with a 3D environment and generate new synthetic data. These systems are typically not used for benchmarking, but rather for generating completely new data, e.g., for training learning approaches.

In 2016, Song *et al*. proposed the SunCG dataset [201]. It is a richly-annotated large-scale dataset of 3D indoor scenes. The dataset contains over 45 000 scenes of manually generated room and furniture layouts. The different scenes are semantically annotated with instance labels. Furthermore, depth images can be generated. Like Matterport, the dataset only contains static objects and is not really suited for generating tracking data.

A further domain where temporal data is of interest is human pose and action estimation. The Synthetic hUmans foR REAL tasks (SURREAL) dataset [224] is a large dataset with synthetically generated images of human motion data. The dataset includes more than 6 million frames with RGB data, depth maps, segmentation masks, and ground truth human poses. However, the dataset was not constructed for object tracking specifically. Hence, the sequences are very short and the humans are placed in front of a static background. Nevertheless, the visual quality of the rendered humans is convincing.

Similarly, the Procedural Human Action Videos (PHAV) dataset [67] is a synthetic dataset of procedurally generated human action recognition videos. The labels include RGB frames, depth maps, optical flow, and instance-level segmentations. The dataset is constructed with the game engine UnityPro. The realism of the generated images is significantly better than in the above works. It contains almost 40 000 videos. Although most of the sequences are quite short, some longer sequences exist. The videos are mostly human centered but have a large variation in the actions performed. The dataset could probably be used to generate a single-human tracking benchmark system.

A further synthetic dataset that could be used to generate an object tracking dataset is the SYNTHetic collection of Imagery and Annotations (SYNTHIA)[181] dataset. It was generated with the Unity Development Platform[2] and consists of rendered images from a virtual city with different weather and times of day. The images are annotated for 13 classes with pixel-precise segmentations. The dataset is split into three different subsets; SYNTHIA-Rand, SYNTHIA-Seqs and SYNTHIA-SF. The first subset contains 13 400 frames of the city taken at random positions and illumination. The later two subsets consists of ten video sequences with thousands of frames each. The virtual vehicle moves through the city and interacts with various dynamic objects such as pedestrians, vehicles, and cyclists. The dataset has instance-level annotations that are consistent throughout the sequences. Furthermore, there are images and annotations for various cameras on the virtual vehicle. There are also depth images available. Nevertheless, the images in the dataset appear visually synthetic and the single textures of the different surfaces have little variation. As a result, the image statistics are very different from natural images. A few example frames from SYNTHIA-SF are display in Fig. 5.3.

[2]Unity Technologies, `https://unity3d.com`

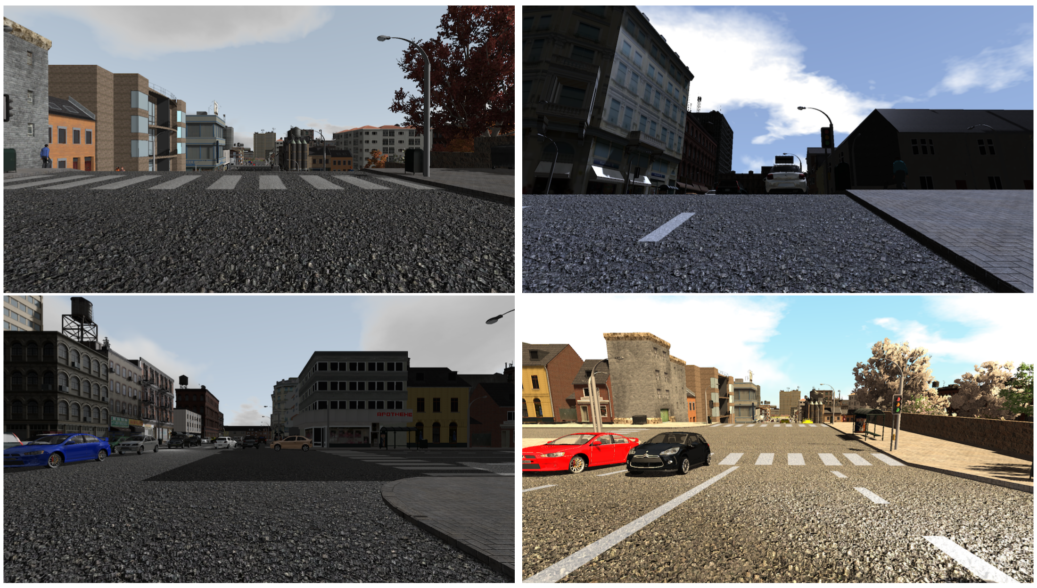


Figure 5.3: Example images from the SYNTHIA-SF [181] .

A very similar dataset is Virtual KITTI [83]. It was also rendered with the Unity game engine. It contains 50 high-resolution videos (21 260 frames) generated by driving through five different virtual worlds with a virtual car. Each sequence is acquired multiple times with different weather and lighting conditions. Again, there are multiple virtual cameras mounted on the vehicle to increase the viewpoint variation for each scene and to enable the testing stereo algorithms. The images have a similar quality to SYNTHIA and appear visually artificial.

The above methods attempt to create photo-realistic virtual worlds with an open-source graphics engine and manually generate sequences within them. This requires considerable effort and the quality of the images is typically restricted by the features of open-source graphics engines. A different approach is to exploit the advances in the quality of the graphics in computer games. Immense efforts have been undertaken by the gaming industry to produce computer games with realistic graphics and intelligent agents. Recent works exploit the quality of the virtual world used in Grand Theft Auto V[3] to generate photo-realistic images and annotations [104, 178, 179]. In their work, Richter *et al*. [179] use the graphics debugger RenderDoc [111] to gain access to the DirectX calls and generate ground truth labels for each generated image. Similarly, Johnson-Roberson *et al*. [104] use two open-source plug-ins to extract annotations at 1 Hz. To reduce the overhead required by the debugger, Richter *et al*. extend their framework to access the DirectX calls directly [178]. This allows to create ground truth annotations in real time. The resulting VIsual PERception benchmark (VIPER) dataset is very diverse and the pixel-precise annotations are available for 124 sequences of varying length and complexity. In total there are 254 064 frames with a resolution of 1920 $\times$ 1080 pixels. The sequences are acquired at different lighting and weather conditions and are very diverse with many moving cars, pedestrians, and cyclists in the scenes. Although there are

[3]Rockstar Games, https://www.rockstargames.com/V/

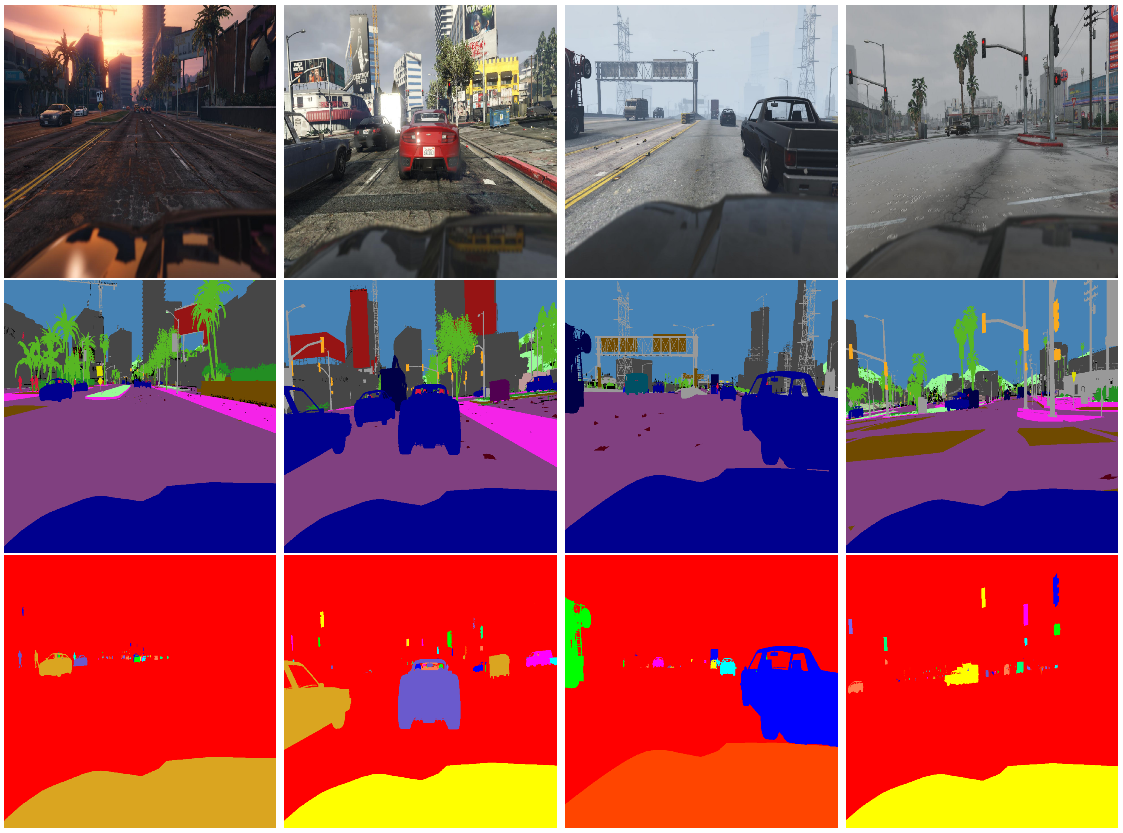


Figure 5.4: Example images and annotations for the VIPER dataset [178]. In the middle, the annotations of the classes are displayed with a color coding. At the bottom, the instance-level annotations are converted to an RGB image to display the individual instances.

instance-level segmentations available, the ID of the objects are not consistent throughout a sequences. Hence, the dataset cannot be used for object tracking out of the box. Nevertheless, the quality of the images and the diversity within the sequences makes it a very appealing starting point for generating a long-term object tracking benchmark with pixel-precise annotations. A few example images and annotations are shown in Fig. 5.4. Since the annotations are generated automatically from the DirectX calls, they are extremely accurate.

Simulators To generate new data from existing 3D models and environments, the Multimodal Indoor Simulator (MINOS) [187] has been published. It enables to navigate through complex indoor environments and create multi-sensory data. It currently supports the 3D environments from Matterport3D [42] and SunCG [201]. However, although the quality fo the rendered scenes is very high, there are no dynamic agents. Hence, the simulator is of restricted use for generating data for object tracking directly.

Similarly, UnrealCV[173] is an ongoing project to help computer vision researchers build virtual worlds and generate annotations and images in real-time. It uses the Unreal

Engine 4 (UE4)[4] game engine. The system requires little knowledge of the fundamentals behind the UE4 and new worlds can be added in a straightforward manner. However, again, there is no direct support of dynamic agents within the virtual scenes. Hence, apart from tracking static objects with respect to ego-motion, the evaluation of tracking is restricted.

In recent years, a number of open-source simulators to generate computer vision images and annotations with dynamic objects were published simultaneously. Like UnrealCV, they mostly build on the UE4 game engine. The UE4 game engine and the integrated tools have been made open source. This has invited many researchers to build their computer vision simulators with using the UE4. Two prominent examples are CARLA [74] and AirSim [191]. CARLA is an open-source simulator for autonomous driving research. It provides various urban layouts and models for buildings and vehicles. The code and protocols are open-source and the simulation platform supports various different sensors and environmental conditions. The annotation and sensor data can be extracted in real-time. Among others, it has been used successfully for the end-to-end training of an autonomous "virtual vehicle" [74]. Similarly, AirSim is a simulator for drones and cars. It was released by Microsoft[5] in 2017. Any Unreal environment is supported, which makes it very easy to generate very diverse datasets.

From Synthetic to Real Data The above methods allow to create an infinite amount of images and annotations. Nevertheless, the generated data is still synthetic. It has been observed that learning from synthetic images may not achieve the desired performance due to the gap between the distributions of synthetic and real images. As a consequence, algorithms may not generalize from synthetic images and perform poorly for real images, although they are top-performers on synthetic data. To close this gap, a number of research directions exist [28, 59, 195, 197, 228]. Most of the top performing approaches make use of Generative Adversarial Networks (GANs) [86]. The GAN framework consists of two networks, a generator and a discriminator, with competing tasks. The goal of the generator is to create realistic images and attempt to fool the discriminator, which distinguishes between real and synthetic images. Hence, both tasks have a competing loss. Initially, GANs where proposed to generate visually realistic images from random inputs [86]. Since then, they have been extended to generate realistic images from synthetic input images without manipulating the labels of the input too much. For example, Shrivastava *et al.* [195] use GANs to create a top-performing gaze estimation system from synthetic images and a large dataset of unlabeled real images. The real images are required to force the generator to create images that are as realistic as possible. Hence, the gaze direction of the real images does not need to be known. In general, the frameworks that refine synthetic data need to ensure that the labels of the objects (e.g., the gaze direction) are not manipulated by the generator to much. This is still a key problem of these approaches and a very active research field [59, 197].

[4]Epic Games Inc. `https://www.unrealengine.com/en-US/blog`
[5]Microsoft `https://github.com/Microsoft/AirSim`

5.2 A Novel Tracking Dataset: Playing for Tracking Data (PFTD)

The goal is to create a diverse dataset with pixel-precise annotations. The sequences should include long term scenes with occlusion, perspective change, and a large variety of different vehicles. Nevertheless, to keep the evaluation efficient, the dataset should also be as compact as possible. In general, this is the crux of every benchmark dataset; it needs to balance between a reasonable diversity and length and an efficient evaluation. For example, the protocol of the OTB 2015 benchmark [239] reinitializes every tracker multiple times for a single sequence (at different positions within the first frame and at different time steps within the sequence itself). Hence, to evaluate a single tracker on the 100 sequences, more than 1 000 single tracker runs are generated. For multiple trackers, this is very time-consuming process. Furthermore, a very large dataset makes a detailed by-scene analysis difficult. As a consequence, the proposed dataset is compact, has very precise labels, and diverse sequences.

In general, simulators are an excellent starting point for generating large-scale datasets and benchmarks. They allow fitting the environment to many domains and applications. However, to generate a diverse dataset, an intense interaction with the virtual worlds is still required. A user needs to manually control a virtual car or drone through the environment and create a diverse set of data. Furthermore, to generate a set of sequences to benchmark visual trackers, the ground truths need to be connected on an instance-level. Although the work load is significantly lower to create the pixel-level annotations, it doesn't come for free. To remove the necessity to generate completely new sequences manually, we turn to an existing synthetic dataset instead: we use the generated sequences from VIPER[178] to create a new object tracking benchmark with pixel-precise labels.

In the VIPER dataset, the single classes and instances in each image are identified in two further images. The `cls` image identifies the semantic class of each pixel. It is shown in the second row of Fig. 5.4. The `instcs` image identifies the single instances within the image. It is converted to RGB and displayed in the last row of Fig. 5.4. Together, both images can be used to segment and identify the objects of each class. Unfortunately, the instance IDs are not consistent between the single frames. Hence, a single car may change ID multiple times within a sequence. This makes it difficult to use the annotations for tracking directly. The authors recently released a further label image `inst_tracked` that encodes tracking IDs that are meant to solve this problem. Unfortunately, the IDs are also not unique throughout a scene and sometimes inconsistent with the other label images. Hence, to use the dataset to generate a tracking benchmark, we present a semi-supervised labeling approach that generates IDs that are consistent over time.

5.2.1 Semi-supervised Annotation

For each frame, it is possible to extract the "perfect" detections from the annotation images. Nevertheless, to create tracking sequences, the detections need to be matched to each other between the frames. The problem is visualized in Fig. 5.5, where the four blue detections at time step t need to be matched to the three detections at time step $t + 1$.

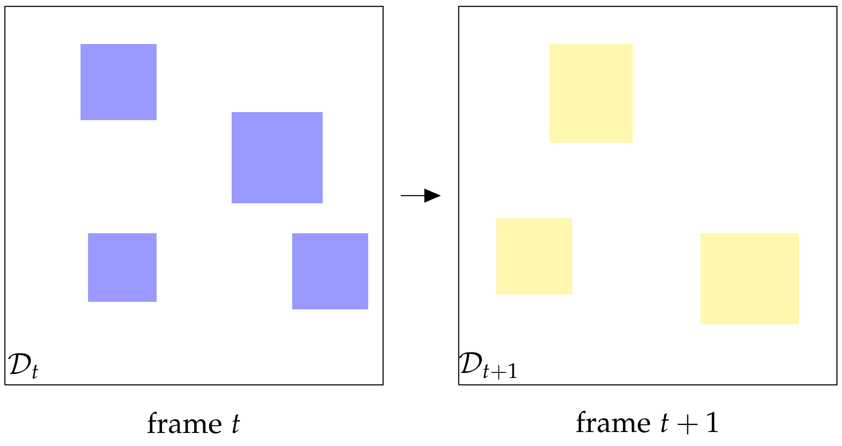


Figure 5.5: Although the detections for frame t and frame $t+1$ are perfect, to generate connected tracking sequences, the detections need to be matched with each other. Since the number of detections may change, this is not trivial.

Since detections may appear and disappear at any time, the mapping between the single frames is not always trivial. We denote the detection in frame t with $\mathcal{D}_t$ and the mapping of the detections from frame t to $t+1$ as $m(\mathcal{D}_t, \mathcal{D}_{t+1})$. The i-th detection in $\mathcal{D}_t$ is denoted as $\mathcal{D}_t^{(i)}$ and we assume there are n detections $\mathcal{D}_t$ and m detections $\mathcal{D}_{t+1}$.

The process of mapping the detections can be automated to a large extent. In an initial step, a cost matrix $M \in \mathbb{R}^{n \times m}$ to map every detection $\mathcal{D}_t$ to every detection in $\mathcal{D}_{t+1}$ in computed. Here, the cost to match $\mathcal{D}_t^{(i)}$ to $\mathcal{D}_{t+1}^{(j)}$ is encoded in $M_{i,j}$. As the matching costs consist of distances and similarity measures, the cost matrix is symmetrical ($M_{i,j} = M_{j,i}$). The matrix is initialized with c_{max} for each entry.

The number of cost computations can be reduced significantly by only calculating the cost for two detections that are reasonably close to each other. As a consequence, only very few values of M need to be computed explicitly. In a first step, a maximum Euclidean distance α is set (e.g., 100 pixels). Then, a regular grid is fit to the image. The square cells of the grid are chosen to be slightly larger than twice the maximum distance, hence $g = 2\alpha + \epsilon$. The grid for the detections in 5.5 is displayed in Fig. 5.6. This enables to reduce the calculation of the distances between two detections. For any detection, the distance only needs to be calculated between the detections that lie in the same, or in one of the three closest adjacent cells. For all other detections the distance is sure to be over α. The grid cell of each detection can be computed efficiently by dividing the row and column image coordinate by g and rounded down to the nearest integer.

For each detection $\mathcal{D}_{t+1}^{(j)}$ that is close enough to $\mathcal{D}_t^{(i)}$, the Euclidean distance is calculated and encoded in c_{dist}. All detections that are too far from each other are assigned a large cost value c_{max}. To simplify the matching even further, additional costs can be calculated based on the region similarity c_{sim}. For example, the first and second order moments are fast to compute and a reliable prior for the similarity of regions. The distance cost c_{dist} and the similarity costs are then normalized to $[0, 1]$ and aggregated to a single cost value $c = c_{\text{dist}} + c_{\text{sim}}$, which is encoded in $M_{i,j}$. The matrix M is then used to determine the mapping between $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ in a greedy manner. Starting from the

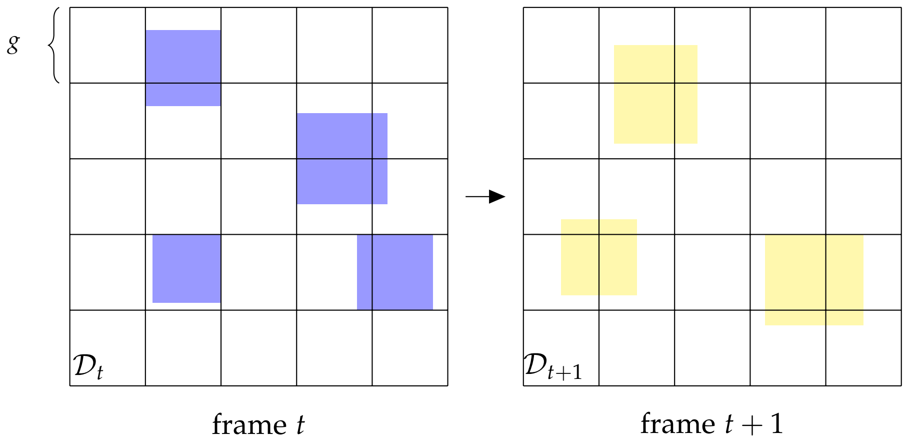


Figure 5.6: To efficiently calculate the mapping between the detections, the matching cost is only computed for detections that are close to each other. For this, a grid with cells that are twice the size of the maximal distance is fit to the images. The cost then only needs to be computed for the detections in the same and adjacent (8-neighborhood) grid cells. Here, $g = 2\alpha + \epsilon$ and α is the maximal distance detections may have.

smallest cost, the detections are mapped and the respective rows and columns removed from the matrix. As soon as the minimal cost is above a fixed threshold $c_{\text{match}} < c_{\max}$, the cost is considered too high to match the detections automatically and the user needs to intervene. The frame rate for the sequences is 30 Hz and, as a consequence, the objects cannot jump through the image in two subsequent frames. Adding a similarity measure to the cost matrix helps to correctly match objects that are very close too each other.

With the above described framework, it is possible to automatically label the majority of the frames. The remaining frames only require a manual matching of the left over detections and removing lost detections. We restricted the objects to cars, trucks, and vans. In total, we computed over 2 000 candidate tracks from the dataset. We excluded all objects that are only visible for a short period or that are never present without too much occlusion.

5.2.2 Dataset Statistics

The goal is to create a compact and diverse dataset. As a consequence, we limit the number of sequences and filter the candidate tracks that were initially obtained. We take care to generate a diverse, yet compact and descriptive benchmark dataset. For this, the candidate tracks were sorted into different categories such as occlusion, lighting change, vehicle turns, scale change, fast motion, very long, difficult weather, or reappearance. Then, difficult or unique sequences from each category where selected. In general, care was taken to ensure the sequences are diverse, different types of vehicles are visible and that many sequences include occlusion and re-detection scenarios. Furthermore, all different weather and daytime conditions are covered. In total, this leads to 100 new tracking sequences with pixel-precise labels. A collection of tracks are displayed in Fig. 5.7.

As shown in Fig. 5.8, the average scene length (308) is roughly the same as VOTSEG

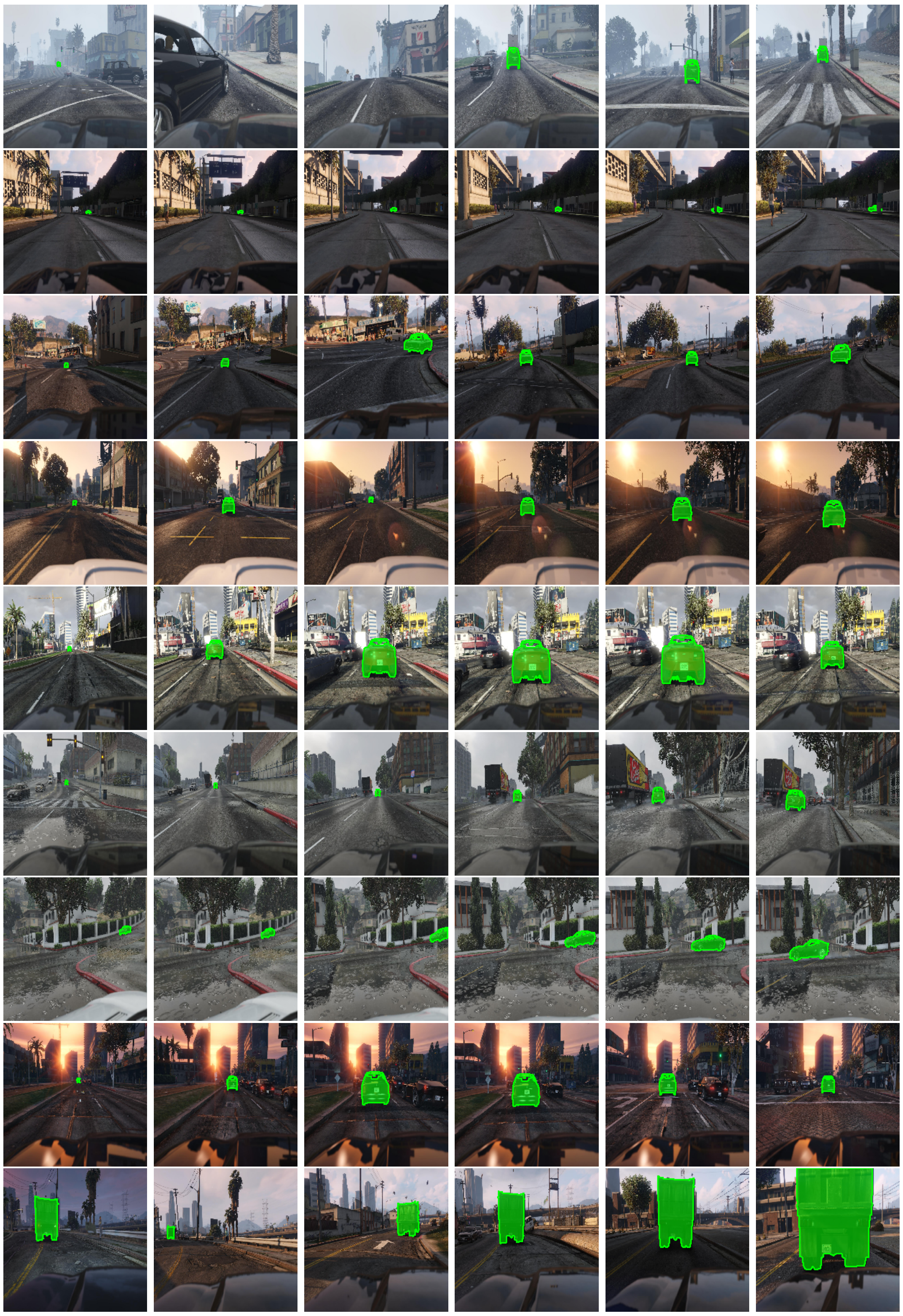


Figure 5.7: A collection of different sequences from Playing for Tracking Data (PFTD). The sequences are very diverse, with different weather, cars, and lighting.

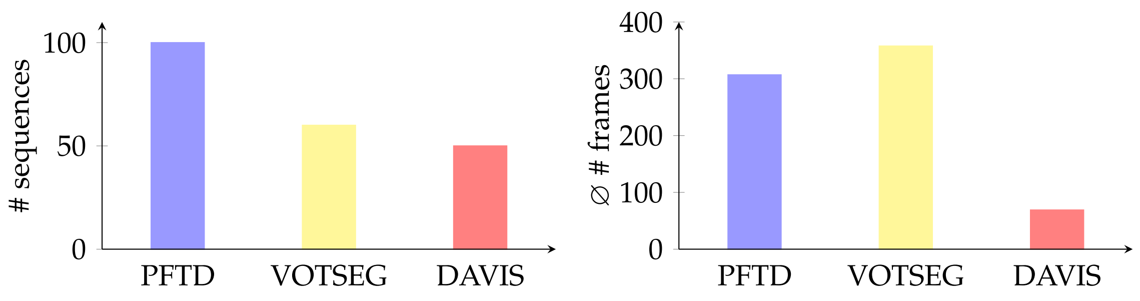


Figure 5.8: There are 100 sequences in the Playing for Trackng Data (PFTD) dataset. The average sequence length is similar to VOT. However, PFTD includes various long term sequences with more than 600 frames and object disappearance and reappearance.

(357) and significantly longer than for DAVIS (69). The label quality is comparable to DAVIS and more accurate than the VOTSEG segmentations, which are obtained from the axis-aligned bounding box labels in a semi-supervised fashion. The dataset has slightly more sequences than VOTSEG and DAVIS but is still reasonably compact. A distribution of the number of frames within the dataset is displayed in Fig. 5.9. The short sequences are added if they are sufficiently complex and add new viewpoints and attributes to the dataset. The dataset includes a number of long-term sequences with many frames. The sequences include full occlusion and reappearance of the objects.

The pixel-precise labels allow to calculate the accuracy measures presented in Section 4. Because the dataset is significantly longer and more complex than DAVIS, this allows to thoroughly evaluate the accuracy of the current trackers with new precision and detail. Since the objects are restricted to vehicles, there are no deformable transformations within the sequences. However, as very few of the current methods focus on deformable transformations, this is no drawback. On the contrary, the restriction to rigid objects allows to focus on the evaluation of the current accuracy of trackers without imposing too many new challenges within the dataset.

The realism of the VIPER images is validated in the original publication [178, 179] by a perceptual experiment that was adopted from prior work in photographic image synthesis [47]. In the experiment, random images from different datasets are displayed in pairs to Amazon Mechanical Turk workers. The worker needs to assess which of the images is more realistic. Among others, the datasets included SYNTHIA [181], Virutal KITTI [67] and Cityscapes [56]. From all of the synthetic datasets, VIPER was rated as the most realistic dataset. Nevertheless, the image statistics are still different from real-world images. The noise statistics do not agree with those of real cameras and homogeneous regions tend to have large patches of equal RGB values. Hence, although the dataset enables to measure tracker accuracy with high precision, the results may not equal those of a real-world dataset.

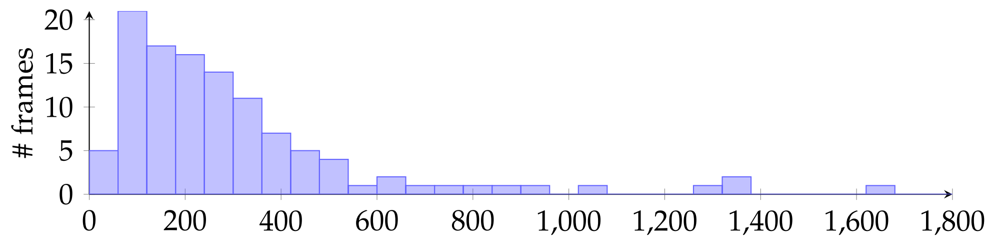


Figure 5.9: The distribution of the number of frames within PFTD. The dataset includes long-term sequences. A short-term sequence is only added if it is sufficiently complex and adds to the diversity within the dataset. Hence, all different weather and daytime conditions are covered.

5.3 Discussion

The new synthetic dataset Playing for Tracking Data (PFTD) with very accurate pixel-precise labels for evaluating tracker performance was presented. The dataset is compact, yet diverse and includes very difficult sequences with vehicles being occluded, disappearing completely, and changing perspective. In contrast to the tracking benchmarks with a similar quality of the labels, the sequences in the dataset are much longer and more complex. The high precision of the annotations enable measuring the accuracy of trackers with increased precision. This should enable to insights on the strengths and weaknesses of the current state of the art.

Although the images have a very realistic appearance, they are synthetic and have different image statistics than real-world images. Hence, the evaluation results should be considered with care and validated on real world images. A further direction could be to adapt the image statistics with GANs to make them become more realistic. Nevertheless, since the key value of the new dataset is the very precise ground truth annotations, any enhancement should not manipulate the high precision of the annotations. This is still an open and very active research direction.

Part III

Fast and Robust Object Tracking

6 Related Work

Visual object tracking is one of the fundamental problems in Computer Vision and has been actively researched for many years. There are literally hundreds of publications related to tracking presented at conferences and journals every year. As a consequence, the body of the related literature is enormous. It ranges from early studies on filter-based tracking (e.g., based on Kalman filters [108]) to multi-object tracking in crowded environments [216]. In general, it can be divided into two main categories: single-object tracking and multi-object tracking. The first category is concerned with tracking a single-object that is manually initialized in the first frame. The methods generally assume no prior knowledge about the type of target object or the expected movement and are often called model-free. The second category is concerned with trackers that track an unknown number of objects from a specific known object class. Since the object class is known, these methods typically make use of powerful object detectors and data association techniques to detect and track the single-object instances.

In the following, we present the most prominent works on object tracking, with a strong focus on single-object trackers that are capable of tracking objects in real-time. Where possible, we highlight their relations to our work. Although this thesis is focused on single-object tracking, we present a short overview of the recent trends in multi-object tracking for completeness.

6.1 Multiple Object Tracking

In multiple object tracking, the object category is known, but the number of instances of the objects is time-varying and unknown. In general, the approaches use some form of object detector, either specific to the known class, or unspecific and based on motion in the image (e.g., background subtraction or optical flow). The by-frame detections are then associated to each other in an independent data association step. There may be missing or duplicate detections and the detections are often noisy. In general, the results of multiple object tracking systems are very sensitive to the used object detector algorithm [232]. Furthermore, most methods calculate the data association globally. Consequently, they assume all of the detections for all frames are present at the time of the data association.

Some of the oldest techniques for global data association are Joint Probabilistic Data Association (JPDA) [81] and Multi Hypothesis Tracking (MHT) [175], which were first presented around 1980. The approaches build a joint hypothesis for all possible target assignments and then compute the marginal probabilities for the assignment of each measurement to each target. While JDPA was originally applied to a sonar tracking problem with multiple sensors and targets, MHT was originally applied to aircraft tracking [175]. For both approaches, a growing number of observations and targets lead to an exponential number of possible assignment hypotheses. As a consequence, they are computationally extremely complex. Recently, an approximation to JPDA was proposed [176]. The full joint probability is approximated by only considering the strongest hypotheses. The authors show that as little as 100 hypothesis are sufficient to obtain the same results as the full JPDA. The approximation is extremely fast to compute, efficient, and was competitive with the state of the art in 2015.

Filter-based tracking approaches are used for single- as well as multiple-object tracking. The tracking is modeled as a sequential state estimation from noisy observations. The state space is typically approximated by filter-based strategies such as Kalman filtering [125, 143] and particle filtering [101]. The methods are able to handle short-term occlusion and reason on the data association in complex scenes. In the first work on particle filters for object tracking, Isard and MacCormick [101] jointly estimate the number of objects and their state space. Extensions that efficiently sample the state space for many objects and add a Markov Random Field motion prior exist [13, 113, 150]. However, filtering-based approaches are currently not able to compete with the state of the art on multiple object tracking benchmarks [151, 232].

Although benchmarks on various different applications exist [139, 169, 232], the core of the recent multiple object tracking literature is concerned with pedestrian tracking [151, 213]. It is difficult to group the approaches into disjoint groups, as many of the multiple object tracking approaches overlap at some point. However, a large body of the methods are formulated as network-flow-based optimization or hierarchical data association problems. A recent further direction is graph partitioning, which jointly clusters detection hypotheses in space and time, thereby eliminating the need for a heuristic non-maximum suppression.

Note that the data association technique presented in Section 5.2.1, which links the detections in the PFTD dataset, assumes that the detections themselves are perfect. This assumption is valid since the detections are synthetically created and no false positives or false negatives exist. However, for detections that are obtained from an object detector, the assumption is not valid and linking the detections in a greedy manner does not work.

Network-flow-based optimization The basic idea is to model multiple object tracking as a graph, where each node represents an object detection and each edge represents a transition between two detections. To model the spawning and termination of trajectories, source and sink nodes are added. In the early work of Jiang *et al*. [103], a linear programming relaxation scheme is proposed to perform the data association for all tracks simultaneously. The methods explicitly models track interaction such as object interactions and mutual occlusions. Alternatively, the data association can be formulated as a min-flow problem [246]. Here, the global solution is obtained by finding the minimal cost flow through the graph. By adding binary and linear constraints, the model can cope with trajectory splits, multiple assignments, and false alarms [14, 194, 229]. The network-flow-based formulation can be solved in polynomial time by K-Shortest Paths Optimization [14].

Hierarchical data association The solution space of the network-flow-based schemes can be very large. To reduce its size, hierarchical data association techniques make use of the fact that short-term tracklets of the single-objects can be obtained quite easily and with high confidence. In a first step, short tracklets are generated by either applying a short-term tracker [237] or matching detections between frames that have high confidence and are unambiguous [69]. For example, Wu *et al*. [237] generate short-term tracklets with single-object mean-shift trackers. In a second step, hierarchical data association techniques refine the short local tracklets and connect them to longer tracks of the single-objects. Wen *et al*. [234] exploit the motion of the targets to connect the detections of a powerful object detector to tracklets. The process can be extended to run in real-time by using a RANSAC-style approach and matching the tracklets one at a time in a greedy fashion [233]. In general, the hierarchical data association methods are outperformed by graph partitioning methods in the current multiple object tracking benchmarks [151].

Graph partitioning The above methods solely link the detector response over time. Hence, to ensure the optimization process can reasonably link the single detections, they are refined to be more or less spatially unambiguous. For this, most methods use heuristic non-maximum suppression [194] in each single frame. However, especially in crowded scenes, the selection of the optimal detector response is not obvious. Acknowledging the fact that target detectors may produce multiple equally plausible detection responses, Tang *et al*. [214] propose to link and cluster plausible detections jointly across space and time. Their formulation leads to a Minimum Cost Subgraph Multicut Problem. The solutions of their formulation are such that possibly multiple hypotheses per track and time frame are selected and clustered jointly across space and time. Extensions of their approach are currently among the top-performing multi-object tracking approaches for pedestrians [213, 215]. However, the matching of the detections requires around 1 second per frame. Hence, the methods are currently not real-time capable.

Multiple object tracking has come a long way in recent years owing to the rise of powerful object detectors based on deep learning. They are able to handle a varying number of objects in crowded and complex scenes. However, most of the top-performing methods are far from real-time [139] and require all of the detections simultaneously to generate reasonable tracks. Furthermore, the current state of the art mostly regards the detection and tracking steps as independent problems. Errors introduced in the detection phase directly influence the performance of tracking and pose estimation. Hence, a further direction is to create end-to-end solutions for multiple object tracking [213].

6.2 Single-Object Tracking

The body of the literature on single-object tracking is extremely diverse. There are literally hundreds of different approaches that tackle object tracking in vastly different manners. Already the object representation differs greatly. Some methods use points [9, 13, 193, 218], others use boxes [36], or more general regions [29, 54], contours [18, 101], or articulated models [124, 227]. However, the great majority of approaches represents the object by an axis-aligned bounding box. On the one hand, this is due to the fact that a rectangular representation simplifies the implementation of many trackers. On the other hand, this is due to the fact that the ground truth is generally represented by an axis-aligned bounding box [117]. To gain an overview of the performance of the multitude of different trackers, numerous surveys and benchmarks have been proposed [116, 117, 118, 153, 238, 239, 242]. In the last few years, most of the top-performing trackers either rely on correlation filters [98], features generated from deep neural networks [157], or a combination of both [61, 66].

It is generally difficult to split tracking methods into more fine-grained categories, as many algorithms overlap at various points. However, very often tracking approaches are divided into two groups: generative and discriminative trackers. Generative trackers only model the appearance of the object and search for the best matching location in each frame. The model templates are either static [54] or adaptive [147]. Since generative trackers focus on modeling the object itself, they may fail in cluttered background or when similar objects appear. To cope with such complex scenes, discriminative trackers also model the background and consider tracking as a binary classification problem that determines the boundary between the object and the background [107]. However, trackers exist that are more or less a combination of both [243]. In the following, we comment on the most prominent examples from both classes.

6.2.1 Generative Trackers

A large and diverse set of generative tracking approaches has been proposed over the years. They range from formulating tracking as a sequence of image alignment problems [240] to using strong appearance descriptors based on distribution fields [190]. In the following, we focus on computationally light-weight and real-time capable approaches that have received much attention in the literature.

A very popular early tracking approach is the mean-shift tracker. The mean-shift algorithm was originally presented as a non-parametric technique to obtain density gradient estimates in 1975 by Fukunaga and Hostetler [82]. Cheng [48] generalized the mean-shift algorithm for cluster analysis to non-flat kernels and enabled the weighting of data points. In the following years, the mean-shift algorithm was proposed for object tracking by Bradski [29] and Comaniciu et al. [53, 54]. The object is represented by the distribution of its color and texture, which is encoded in a histogram. The similarity between the template and target location is then measured by the Hellinger distance. The

Hellinger distance H is related to the Bhattacharyya coefficient BC by

$$H(p,q) = \sqrt{1 - BC(p,q)}, \tag{6.1}$$

where p and q are discrete probability distributions of equal length. The Bhattacharyya coefficient BC is defined as

$$BC(p,q) = \sum_x \sqrt{p(x)q(x)}. \tag{6.2}$$

The optimal position of the template can be determined iteratively in very few steps. See [76] for a good overview of the theory. The implementation of the mean-shift tracker is simple and efficient. As a consequence, it has enjoyed extreme popularity even in more recent publications [8, 91, 123, 127, 250]. It has also been used successfully for generating initial tracklets for multi-object tracking [237]. Even though mean-shift is initially a generative tracker, it has been extended to include information from the background to become more robust. For this, Comaniciu et al. [54] proposed a background-weighted histogram to enhance the performance of the mean-shift tracker. The reasons for this are twofold: First, if features are present within the foreground and the neighboring background, their relevance for the localization of the target is reduced. Second, it is often difficult to find a hard border between background and foreground and hence the model might, per se, contain background features. In general, mean-shift tracking is a valid and very efficient approach for very short-term tracking without any complexities such as occlusion, clutter, fast motion, or appearance change.

Many tracking approaches formulate the tracking problem within a Bayesian framework. The state of the modeled object at time t is denoted x_t and its history is collected in the set $\mathcal{X}_t = \{x_1, \ldots, x_t\}$. The precise position of the object state is essentially determined by a set of observations $\mathcal{Z}_t = \{z_1, \ldots, z_t\}$, where z_t is the observation at time step t. A valid assumption in tracking is to restrict the object dynamics to a temporal Markov Chain. Hence, the new object state only depends on the immediately preceding state:

$$p(x_t|\mathcal{X}_{t-1}) = p(x_t|x_{t-1}). \tag{6.3}$$

By further assuming that the observations z_t are independent, both mutually and with respect to the dynamical process, (6.3) can be expressed as

$$p(\mathcal{Z}_{t-1}, x_t|\mathcal{X}_{t-1}) = p(x_t|\mathcal{X}_{t-1})p(\mathcal{Z}_{t-1}|\mathcal{X}_{t-1}) = p(x_t|\mathcal{X}_{t-1})\prod_{i=1}^{t-1} p(z_i|x_i). \tag{6.4}$$

Integration over x_t shows the mutual conditional independence of observations:

$$p(\mathcal{Z}_{t-1}|\mathcal{X}_{t-1}) = \prod_{i=1}^{t-1} p(z_i|x_i). \tag{6.5}$$

The rule for propagation of the state density can be manipulated to yield:

$$p(x_t|\mathcal{Z}_t) = k_t p(z_t|x_t) p(x_t|\mathcal{Z}_{t-1}), \tag{6.6}$$

where

$$p(x_t|\mathcal{Z}_{t-1}) = \int p(x_t|x_{t-1}) p(x_{t-1}|\mathcal{Z}_{t-1}) \, \mathrm{d}x_{t-1} \tag{6.7}$$

and k_t is a normalization constant that does not depend on x_t. The derivation of (6.6) and (6.7) is described in Appendix A for clarity. The above formulation can be solved efficiently with filter techniques such as Kalman filtering [125, 143] and particle filtering [101]. These techniques are able to overcome short-term occlusions and predict the most probable object location in each frame.

Kalman filtering assumes that the posterior density $p(x_{t-1}|\mathcal{Z}_{t-1})$ is Gaussian and can be parameterized by a mean and a covariance. Furthermore, it assumes both the system and process noise are Gaussian and that the state transition is linear. While these assumptions are justified for many applications, they are typically too strong for visual object tracking [101]. As a consequence, particle filters are often used to infer the posterior. Particle filtering is a sequential Monte Carlo method and is also known as bootstrap filtering [87], the condensation algorithm [101], interacting particle approximation [60], and survival of the fittest [109]. It essentially approximates (6.6) by a set of random samples with associated weights and computes estimates based on these samples. It was first used for visual tracking by Isard and Black [101]. The condensation tracker was able to track the contours of objects in real-time in simple gray-scale images. The idea was later extended to tracking a rectangular template using color features [162] and color histograms [171]. However, color (or in this case the intensity) is not very descriptive in gray-scale images and leads to difficulties when the object has a similar color to the background. For this reason, Lu *et al.* [136] use histograms of gradients as features to improve the tracking robustness. Variants that use colors and edges as features also exist [114]. The filtering step has been combined with a classifier to create an advanced motion model and to improve the sampling step [163]. Furthermore, extensions to more general object segmentations that have a reasonable runtime have also been proposed [12]. To cope with more complex scenarios, combinations of Kalman filtering, particle filtering, and mean-shift tracking exist [51, 102, 142]. For example, the mean-shift tracker is used to update the location of the individual samples and is able to significantly reduce the number of particles and improve the overall robustness [142]. In general, the particle filter can be implemented extremely efficiently and can cope with a reasonable amount of complexity and occlusion. It is still in the focus of current research and used for many applications. For example, the Locally Orderless Tracker (LOT) [165] uses a particle filter to efficiently predict the object location on a super-pixel representation of the image. The super-pixel representation enables a significant reduction of the number of particles and the tracking of rigid as well as deformable objects. However, although mostly real-time capable, even the modern particle filtering approaches are generally not able to compete with the current state of the art in object tracking [77, 102, 165]. This is partly due to the fact that the model update generally introduces significant drift and that the model update is dependent on various heuristics and application-specific parameters [114].

A further, very popular, generative tracker is the Kanade-Lucas-Tomasi (KLT) tracker. It is mostly used as a feature point tracker and is based on the early work of Lucas *et al*. [137]. The Lucas-Kanade algorithm was initially proposed for image registration and has since been used for diverse applications such as optical flow calculation, mosaic construction, or face coding. The algorithm was expanded to tracking by Tomasi and Kanade in 1991 [218]. For this reason, the tracker carries the name of all three authors, Kanade, Lucas, and Tomasi. The tracking method is explained elaborately in [193]. A fast implementation of the tracker was published 20 years after the original publication in [9] and used within a very precise tracking-by-detection scheme in [13]. For single-object tracking, it has been used in various publications. For example, the Flock of Trackers (FoT) [225] estimates the pose of an object by robustly combining displacement estimates from a subset of KLT point trackers that cover the object. Furthermore, it is possible to estimate the stability of the single KLT point trackers by tracking the points forward and backward in time and comparing the displacement vectors. By this, it is possible to identify tracker failure, as proposed in the median flow tracker by Kalal *et al*. [106]. Like the mean-shift tracker, the KLT trackers are extremely efficient but cannot cope with complexities such as occlusion, clutter, fast motion, or appearance change. As a consequence, they are merely useful for generating short-term tracklets of an object.

A further common generative tracking approach is the use of a general object recognition technique for tracking. For example, template matching algorithms can easily be adapted to perform object tracking. In the first frame, the object template is initialized and then, in the successive frames, the object is localized by applying template matching in a compact neighborhood of the prior object position. These approaches are easy to implement and capable of reasonable performance in simple settings. For example, up to 2013, many of the proposed trackers were outperformed by a simple normalized cross correlation (NCC) tracker [118]. Various variants and extensions that enable efficient tracking of objects with short-term occlusions or in more complex sequences exist [188, 251].

One of the few real-time generative methods that does not represent the object by an axis-aligned bounding box, but rather by a general segmentation map, is the MSER tracking method by Donoser and Bishop [72]. The approach tracks so-called Maximally Stable Extremal Regions (MSER) regions [145] through image sequences. The approach can cope with complex model deformations and is extremely fast. However, it is restricted to gray-value images and to tracking regions that are extremal. Hence, regions where all boundary pixels either have a gray value strictly greater or strictly smaller than all inner pixels. This restriction is very strong and prevents tracking many objects in the object tracking benchmarks.

The key advantage of generative models is their simplicity. Most methods can be implemented extremely efficiently. They work very well for objects that are not moving too fast, are not significantly occluded, and do not change their appearance too strongly.

However, because they model the object without taking the background of the object into context, their performance is not on par with that of discriminative methods. Especially in the last few years, most of the top-performing methods are discriminative in nature.

6.2.2 Discriminative Trackers

The discriminative tracking approaches can roughly be split into two groups. The first group performs feature selection based on cues from the object and its surrounding background [52, 54, 132, 160] and the second group formulates tracking as a binary classification problem that determines the boundary between the object and the background [6, 94]. The second group is often referred to as tracking-by-detection. The classifier is either tested on many candidate patches [6] or predicted directly [94]. Most of the early work on discriminative tracking belongs to the first group. For example, Collins *et al*. [52] use feature histograms from the object and the surrounding background to determine the most discriminative color space for tracking. Similarly, Nguyen and Smeulders [160] use Gabor filter responses from the object and the background to select the most discriminative feature set. In general, the online selection of the most discriminative features can greatly improve the tracking performance and comes at a reasonable cost [54]. However, these approaches are still restricted to the strengths and weaknesses of the underlying generative trackers.

The second group is much larger and diverse and uses a multitude of different classifiers to determine the object position. For example, Avidan [6] use an ensemble of classifiers to distinguish between the object and the background. A large set of weak classifiers are combined into a strong classifier with AdaBoost. Each pixel of the object is then classified as belonging to the object or the background. The optimal object location can be determined by applying mode-seeking algorithms, such as mean-shift, to the resulting confidence map. A collection of very similar approaches that essentially use a different classifier in a similar pipeline exists, for example, those based on Support Vector Machines (SVM) [5], Random Forest Classifiers [107], or other boosting variants [7]. The tracking-by-detection approaches have been extended to yield a failure detection mode. In [107], Kalal *et al*. combine an efficient generative tracker [106] to detect a tracking failure and an object detector based on Random Forests to perform object re-detection. The simultaneous use of a fast generative and a fast discriminative tracker allowed to track objects in long sequences in real-time. To date, it remains one of the few trackers that can predict the absence of the object [222].

To infer the object location directly, Hare *et al*. [94] propose to use SVMs and Gaussian kernels. Instead of learning a classifier that distinguishes between object and background, they propose to learn a prediction function that directly estimates the object transformation between frames. The method uses low-level Haar-like features and performed very well in the early benchmarks [198]. However, the restriction to simple low-level features restricts its capabilities on the modern tracking benchmark sequences [117].

In 2014, the top-performing object tracking algorithms were based on correlation filter tracking. They were not only able to outperform many of the state-of-the art approaches mentioned above, but also were significantly faster than many of them [62, 98]. Surprisingly, the third-best tracker in the 2014 VOT tracking challenge was based on correlation filter tracking and, in contrast to the competition, not even able to estimate the object scale [98]. The concept was first proposed in 2010 by Bolme *et al.* [19], who introduced the Minimum Output Sum of Squared Error (MOSSE) filter. It makes use of the convolution theorem to reduce the filter correlation to an element-wise multiplication in the Fourier domain. In 2012, Henriques *et al.* [98] presented the closely related Kernelized Correlation Filter (KCF), where the filter correlation is derived from a linear regression point of view. Their approach naturally introduces a regularization parameter and links the results to those presented by Bolme *et al.* [19]. Furthermore, by exploiting the fact that training data is circulant [89], an extension to nonlinear kernels is presented that greatly improves the performance and is nonetheless extremely fast. A further closely linked approach is the Dense Spatio-Temporal Context Tracker (DST), which was presented by Zhang *et al.* [245]. It is essentially a special case of the MOSSE tracker that assumes a single training image and neglects the regularization parameter.

The idea behind the mentioned trackers is to learn a correlation filter H from samples of the target. The object location in new frames is then determined as the maximal value of the filter convolution. The filter convolution G can be determined by convolving the input frame F with the correlation filter H, hence

$$G = F \otimes H. \tag{6.8}$$

The convolution theorem states that if two input signals x, y belong to $\mathcal{L}^1(\mathbb{R}^n)$ (Lebesgue-integrable), the Fourier Transform of their convolution can be expressed as the point-wise product of their Fourier transforms. Hence, in the above setting the convolution can be efficiently determined by the point-wise product of the filter and the current frame in the Fourier space

$$\hat{G} = \hat{F} \odot \hat{H}, \tag{6.9}$$

where $\hat{G}, \hat{F}, \hat{H} \in \mathbb{C}^{n \times m}$. In general, to ensure the input image is periodic at the border, it is filtered by a cosine window before the Discrete Fourier Transformation (DFT). MOSSE learns the filter $\hat{H}$ that minimizes the sum of squared differences of the desired filter output $\hat{G}_i$ to the actual filter output on a set of input images in the Fourier domain:

$$\hat{H} = \min_{\hat{\mathcal{H}}} \sum_i \left\| \hat{F}_i \odot \hat{\mathcal{H}} - \hat{G}_i \right\|^2 . \tag{6.10}$$

The initial filter output G is a Gaussian that is centered around the initial object location. Hence, the optimization explicitly attempts to generate a filter that can best discriminate the object from its direct surroundings. The optimization makes use of the fact that each element $\hat{H}$ can be solved for individually. Hence, the closed-form solution yields

$$\hat{H} = \frac{\sum_i \hat{G}_i \odot \hat{F}_i^{\,*}}{\sum_i \hat{F}_i \odot \hat{F}_i^{\,*} + \lambda}, \tag{6.11}$$

where λ is added to the denominator as a regularization factor. It helps to overcome problems when the energy of a specific frequency is 0 or very small.

While the MOSSE approach estimates the filter H directly, Henriques *et al.* [98] motivate their correlation approach from the perspective of Ridge Regression. In its basic form, Ridge Regression attempts to find an optimal linear function $f(z) = w^T z$ that minimizes the squared error over samples x_i and their regression target y_i,

$$w = \min_{\tilde{w}} \sum_i \left(\tilde{w}^T x_i - y_i \right)^2 + \lambda \|\tilde{w}\|^2. \tag{6.12}$$

Here λ is a regularization parameter used to control overfitting. The above equation can be rewritten in matrix notation:

$$w = \min_{\tilde{w}} \left(\tilde{w}^T X - y \right)^T \left(\tilde{w}^T X - y \right) + \lambda \tilde{w}^T \tilde{w}, \tag{6.13}$$

where the rows of X are the samples x_i and each element of y is the regression target y_i. The key assumption that is made in the KCF tracker is that the rows of the matrix X encode all possible shifts of the object within the template window. Then, the matrix X is cyclic and the closed form solution of the above equations is

$$\hat{\tilde{w}}^* = \frac{\hat{x} \odot \hat{y}}{\hat{x}^* \odot \hat{x} + \lambda}. \tag{6.14}$$

The derivation and more details about the structure of X is shown in Appendix B. Equation (6.14) has astonishing resemblance to (6.11), with the exception that $\hat{x}$ is the 1D DFT of the linearized input image and $\hat{F}$ is the 2D DFT of the input image. It is possible to generalize (6.14) to higher dimensions by using tensor notation and the theory of tensor diagonalization with circulant structure [177]. Hence, the MOSSE tracking approach can also be motivated by the theory of ridge regression. An advantage of the second derivation is the possibility to adapt the framework to non-linear regression with the help of the kernel trick. Furthermore, both representations can be extended to multiple channels and can also be applied to feature images derived from HoG-like features [97]. These trackers are extremely efficient, light weight, easy to implement, and run well above frame-rate even in large images. In the following years, many extensions and variants of correlation tracking where proposed and are still widely used today [16, 61, 62, 64, 65, 66, 100, 131, 154, 202, 210, 252]. For example, they have been extended to also estimate the object scale [62, 65, 100, 131], to a more robust filter update [154, 210], and to long term tracking [141, 252]. They form the basis of many of the current state-of-the-art trackers that we use as comparison in the experiments (see Chapter 9).

In the object tracking benchmarks, the current state of the art of discriminative trackers clearly outperforms the generative trackers described above. Through the expression of the convolution of correlation filters in the Fourier space, real-time capable discriminative methods exist that are not only powerful, but also extremely efficient. However, many of the current methods do not encode any failure detection or have an explicit re-detection mechanism. Furthermore, almost all methods are restricted to axis-aligned bounding boxes.

6.2.3 Deep Learning Meets Tracking

Inevitably, the rise of deep neural networks and features derived from them have found their way into object tracking [49, 61, 64, 66, 99, 126, 157]. In the last few years, the top-performing trackers in all benchmarks that do not restrict the time per frame are based on Convolution Neural Networks (CNNs) [116, 118, 153, 222]. Basically, all of the current methods are discriminative trackers and restricted to axis-aligned bounding boxes. The approaches can roughly be divided into two groups. The first group merely uses CNN features and a general tracking approach, such as a correlation filter, for tracking the object [63]. Hence, these methods typically do not require any fine-tuning of the CNN features themselves. However, it has been shown that it is beneficial to finetune the network features on object tracking or object localization sequences in a prior step [63, 90]. The second group includes methods that use CNNs in a tracking-by-detection scheme and classify all pixels as object or background and generate a confidence map [157]. The respective methods generally require a computationally demanding training on the first frame. Some approaches use Siamese networks that obtain the prior object location, the prior image, and the current image as input and directly infer the most probably new object location [17, 96]. Hence, the networks implicitly encode a similarity measure between two input images and the object depicted in them. They are typically fast to train and obtain good performance in the benchmarks [117].

Already in the 2015 VOT challenge [118], the top-two performing approaches, MDNet and Deep-SRDCF, were both based on CNNs [64, 157]. Here, the Multi-Domain Convolutional Neural Network (MDNet) [157] is a typical tracking-by-detection approach that treats tracking as a binary classification task. The network (VGG-M) is fed a set of patches and classifies them as either object or background. The patches are randomly sampled from a Gaussian distribution around the prior object location. For each new sequence, the binary classifier at the end of the neural network is trained on the initial frame. Even with a high performance GPU, the training cannot be conducted in real-time. In contrast, Danelljan *et al.* [63, 64] propose Deep-SRDCF, which does not explicitly retrain the network. Instead, it makes use of pretrained CNN features and correlation filters. In general, combining correlation tracking and deep features is a very popular direction of the current tracking research [49, 63, 140, 202, 223]. In Deep-SRDCF, the authors use a VGG network that is pre-trained on ImageNet [68] as the feature extractor. They compare the performance of correlation filter trackers that are applied to different feature levels. Although the results for the first-level features are the best, the features of the different levels appear to complement each other. As a consequence, later works attempt to fuse the information of feature maps from different levels [61, 66, 211]. For example, one of the top-performing trackers in the VOT 2017 challenge was the Learning Spatial Regressions for Visual Tracking (LSART) [211] tracker. LSART combines correlation filters and a tracker based on CNN features from three different levels. The correlation filter focuses on the holistic target and the latter focuses on small local regions. From all of the trackers submitted to the VOT 2017 challenge, LSART is the most robust. However, its performance is far beyond real-time. A much more efficient tracker was proposed by Danelljan *et al*. [61]. They introduce a factorized convolution operator to drastically reduce the number of parameters in the neural network. It is one of the fastest trackers based on deep neural networks and one of the best-performing trackers in the VOT 2017 challenge [117]. It was merely outperformed marginally by the CFCF [90] and CCOT [66] trackers. CCOT learns a discriminative continuous convolution operator for tracking. Like the above methods, the approach efficiently combines the output of feature maps from different levels of a CNN. The CFCF [90] tracker is closely related and builds on the framework of the CCOT tracker [66]. To improve the performance, the authors propose to train the CNN on the ImageNet Large Scale Visual Recognition (ILSVRC) [185] video dataset. Then the first, fifth, and sixth feature layer are combined with HoG and color features for tracking. Essentially, CFCF and CCOT are equally precise and robust. However, both approaches require a high performance GPU to provide near real-time performance.

In general, deep neural networks have pushed the performance of the current state of the art in tracking. They are the basis of the most accurate and robust trackers. However, they all require a high performance GPU to achieve real-time performance. Although the performance of computation devices is growing steadily, tracking is essentially always only a small part in a computer vision application. Hence, the necessity of a high-performance GPU is usually prohibitive. New tracking benchmarks are starting to address this issue by adding real-time restrictions.

6.2.4 Video Object Segmentation

A relatively new challenge in the computer vision community is video object segmentation. It is concerned with separating foreground objects from the background regions in image sequences. Like in object tracking, the methods are generally initialized in the first frame. However, in contrast to most object tracking benchmarks, the ground truth is represented by pixel-accurate labels [170]. The sequences in the existing benchmarks are typically much shorter and less complex. Therefore, few methods work well in both domains. An exception is One-Shot Video Object Segmentation (OSVOS) [32]. The approach requires a short fine-tunning in the first frame and is then able to segment objects in the successive frames without assuming any temporal connection. Surprisingly, the approach is able to compete with the current state of the art on some tracking sequences. However, the method generally fails when similar objects present in the background have a similar texture to the object [32]. Nonetheless, with the increasing capabilities of trackers and object segmentation techniques, both communities are overlapping more and more.

6.3 Most Related Works

In the subsequent sections, two real-time trackers are presented. Both methods are complementary to each other, light-weight, and able to run in real-time, even on embedded and low performance devices. In contrast to the current trend in object tracking, both approaches are generative trackers. They focus on specific real-world applications and, as such, need to be extremely efficient. As mentioned before, tracking is merely a building block in many applications and generally has very strong computational bounds. In the following, we comment on the more specific related works.

The first approach is presented in Chapter 7 and tracks Maximally Stable Homogeneous Regions (MSHR) in images with an arbitrary number of channels. MSHR are conceptually very similar to Maximally Stable Extremal Regions (MSER) [145] and Maximally Stable Color Regions (MSCR) [80], but can also be applied to hyperspectral and color images while remaining extremely efficient. The tracking approach is closely related to the generative MSER tracker of Donoser and Bishop [72]. Both approaches represent the objects by a pixel precise segmentation and can cope with complex object transformations and deformations. However, in contrast to MSER tracking, our approach is applicable to images with an arbitrary number of channels. Furthermore, the restriction to extremal regions is lifted. In contrast to extremal regions, homogeneous regions are more general. This enables our approach to track a greater variety of objects. Nonetheless, both extremal and homogeneous regions are very restrictive on the types of objects that can be tracked. Hence, although both methods can efficiently solve numerous applications, they do not focus on being able to track the diversity of objects in the common object tracking benchmarks.

The second approach, the shape-based tracker, is presented in Chapter 8 and is much less specific on the type of objects that may be tracked. The tracker is able to track arbitrary rigid objects in image sequences. The object is represented by a sparse set of model points that represent the significant object edges. Hence, the method does not necessarily require textured objects. A sparse set of model points is mostly used for object recognition as opposed to object tracking, e.g., approaches that are based on Chamfer-Matching [20], on the Hausdorff Distance [184], or on geometric hashing [122]. Some methods use model points and directions for matching a model to an image. For example, methods based on the generalized Hough transform [10, 221] or approaches that are based modifications of the Hausdorff Distance [164]. However, the model representations that are closely related to the shape model tracking are active shape models of Cootes *et al*. [55] and, even more similar, the model within the shape-based object recognition technique of Steger [205]. Here, the model points are represented by their point coordinate and their normalized directions. In contrast to the proposed shape-based tracker, both approaches are object recognition schemes that do not adapt the model templates after the object localization.

For an efficient object localization, an image pyramid is created for each input image. In image recognition techniques, image pyramids or feature pyramids are common practice [70, 133, 217]. However, some tracking approaches also use image pyramids to be more robust to fast movement or to increase the efficiency [127]. The different semantics of features from different pyramid levels is also implicitly encoded in modern CNN architectures for segmentation and recognition [133, 247]. A novelty of the proposed shape model tracking approach is that the depth of the image pyramid can be dynamically adapted to the complexity of the object model and the size of the search region.

In terms of runtime, the shape-based tracker is comparable to correlation filter approaches. However, in contrast to the above mentioned correlation filter methods, our approach estimates the location, scale, and rotation of the object with very high accuracy. In this sense, it is related to the key-point recognition system of Lepetit and Fua [124]. The authors present a recognition system that also estimates the pose of rigid objects. It is robust to occlusion and clutter, but requires an extensive offline training phase to generate the tracking model. In contrast to our approach, their tracking is restricted to textured objects to generate sufficiently stable keypoints for reliable tracking. Furthermore, our approach does not assume any complex offline training to generate the tracking template. Furthermore, as opposed to the majority of the above mentioned tracking approaches, the shape-based tracking efficiently tracks the pixel precise region of the object and is not restricted to bounding boxes. Hence, it is generally capable of obtaining a much higher accuracy than the existing approaches.

7 Efficiently Tracking Homogeneous Regions

This chapter presents an efficient tracking scheme that can track pixel-precise representations of an object and their deformations through image sequences. The tracker builds on the derivative-based component-tree, which is an extension of the ordinary component-tree to multi-channel images. The tree can be constructed efficiently in real time and scales linearly in the number of pixels and, in practice, sub-linearly in the number of channels. The trees can be used to efficiently extract Maximally Stable Homogeneous Regions (MSHRs), which are conceptually similar to Maximally Stable Extremal Regions (MSERs). In contrast to prior work that uses component-trees for tracking (e.g., [72]), the presented tracker works on multi-channel images and has a model update step to enable a more robust tracking. Since MSHRs are more general than MSERs, the tracker has the further advantage that a greater variety of objects can be tracked. We display how the approach can be used efficiently for the real time tracking of arbitrarily shaped regions in image sequences and for 3D reconstruction in CT slices.

The rest of this chapter is organized as follows: In Section 7.1, related work on component-trees and extensions to multi-channel images is discussed. In Section 7.2, the derivative-based component-tree and an introduction to MSHRs is presented. Next, in Section 7.3, the efficient MSHR-based tracking scheme is described in detail. Possible applications are presented in Section 7.4. The chapter concludes with a discussion of the presented tracking scheme in Section 7.5. The results presented in this chapter are covered in more detail in Böttger and Eisenhofer [22], Böttger and Gutermuth [25], and Böttger *et al*. [21].

7.1 Related Work: Component-Trees

The component-tree (also known as dendrone [45], confinement tree [146] or max-tree [34]) is a hierarchical data structure that models gray-scale images by considering the connected components of their binary level sets obtained from successive thresholdings [120]. It has a wide range of applications: image filtering [105, 186], motion extraction

[186], feature and region extraction with Maximally Stable Extremal Regions (MSERs) [145], pattern recognition in astronomical imaging [15], 3D visualization [235], and object tracking [22, 72]. For gray-scale images, efficient algorithms exist that enable the construction of the component-tree in linear time [34].

In general, component-trees have been used for a diverse set of applications and substantial efforts have been undertaken to enable their efficient computation [34]. There are essentially three different kinds of component-tree computation algorithms: immersion algorithms, flooding algorithms, and merge-based algorithms. Carlinet and Géraud [34] present an extensive comparison of the main approaches and show that the flooding-based approaches of Wilkinson [236], Salembier *et al.* [186], and Nistér and Stewénius [161] are superior in terms of speed for 8-bit and 16-bit images. Although many applications for gray-scale component-trees have been presented, most are devoted to image segmentation and filtering [105, 121, 186]. For example, MSERs can be extracted efficiently using component-trees [161].

MSERs themselves have a wide range of applications, ranging from stereo feature point extraction [145] over optical character recognition (OCR) [159] to image tracking [72]. Motivated by their success on gray-scale image processing applications, there also have been attempts to extend MSERs specifically to multi-channel images. Chavez and Gustafson [44] transform the RGB image to the HSV color space and extract gray-scale MSERs on three different combinations of the single HSV channels. Hence, the MSER algorithm is applied three times and the different MSERs are collected. Forssén [80] overcomes the problem that multi-channel images cannot be totally ordered by using pixel differences of neighboring RGB values as opposed to the RGB values directly. This allows the extraction of so-called Maximally Stable Color Regions (MSCRs). Although no component-tree is constructed in the process, the idea of using differences is appealing since it does not require a user-defined partial ordering and can be trivially extended to images with an arbitrary number of channels. Unfortunately, the approach is computationally demanding and, although theoretically very closely related to MSERs, MSCRs have completely different parameters. This makes it difficult to compare the performance of both approaches. Similarly, Donoser *et al.* [73] construct the component-tree of an RGB image from the gradient magnitudes of the input image. Stable regions are then extracted by comparing the shape of regions at different levels using a shape matching method. While computing MSERs from the gradient magnitude images allows to use an ordinary MSER implementation, it has several disadvantages for applications. The central differences at the image coordinates used to compute the gradient magnitude image make the edges at least two pixels wide. As a consequence, the extracted regions are smaller than the actual regions and very narrow regions cannot be extracted. As shown in Fig. 7.1, especially for applications like OCR, where characters are generally only a few pixels wide, this is a crucial disadvantage.

Inspired by the success of component-trees for gray-scale image filtering, general extensions of gray-scale component-trees to multi-channel component-trees have been proposed. Passat and Naegel [168] introduce the concept of component graphs. However, the multi-channel component graph is algorithmically very complex and requires a

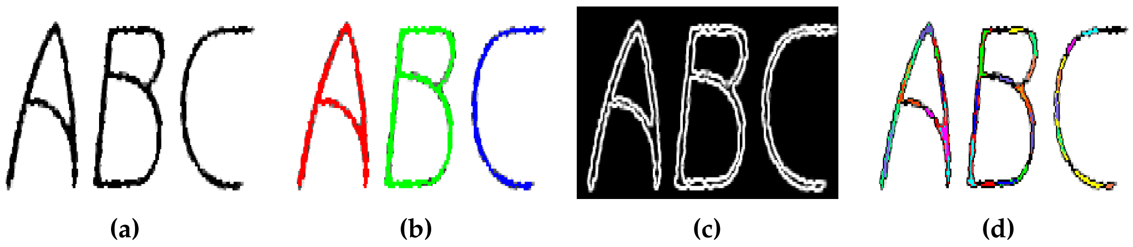


Figure 7.1: Extracting stable regions from the input image (a) or the image derivatives directly (our proposed method) leads to the results shown in (b). When using the gradient magnitude image (c), it is very difficult to extract stable regions (e.g., as in [73]) that are very narrow (d).

user-defined piecewise ordering of the image values that is specific to the target domain. A further extension of the component-tree to multi-channel images that is conceptually similar to ours is the Multivariate Tree of Shapes (MToS) [35, 241]. The MToS is a five-step process that first computes a tree of shapes (ToS) for each channel individually. Hence, the runtime has a large linear factor in the number of image channels. In contrast to the above approaches, the proposed derivative-based component-tree is constructed using pixel differences and is applicable to images of different domains and numbers of channels and does not require any pre-defined partial ordering. As a consequence, fewer parameters are required and the tree construction is significantly faster. We compared our approach to the binaries of MToS and are nine times faster for a three-channel image. The performance advantage will be even more prominent for hyper-spectral images, which contain significantly more channels.

The works most related to ours are those based on a Max-Tree computed on an edge-weighted graph [58]. These approaches create morphological hierarchies of connected pixels [199]. Here, two pixels are connected if there exists a path linking these pixels such that the maximal edge weight does not exceed a given threshold value. For a varying threshold, the resulting regions form a hierarchy. The concept originates from the single linkage clustering method [88] used for data analysis. It was introduced to image processing by Nagao *et al.* [156] in 1979. Prominent examples include α-components and α-trees [166, 199], and (in mathematical morphology) quasi-flat zones and the quasi-flat zone hierarchy [148, 149].

In contrast to the aforementioned approaches, the derivative-based component-tree can be constructed by a flooding-based immersion. Although the resulting tree is the same as a Max-Tree constructed on an edge-weighted graph [58], the presented immersion is significantly faster than a union-find-based immersion. It allows an efficient computation that is linear in the number of pixels and scales favorably in the number of channels. Since the flooding-based immersion flows through image derivatives, we term the resulting tree derivative-based component-tree.

7.2 Derivative-based Component-Tree

For gray-scale images, the component-tree is constructed by considering the binary level sets of the input image. The level sets are obtained from successive thresholds, e.g., for byte images, the thresholds are selected to include all pixels within $[0, \alpha]$ or $[\alpha, 255]$. The threshold α is either increased incrementally from 0 to 255 or decreased incrementally from 255 to 0. The evolution of the connected components of the respective level sets is then encoded in the component-tree. Each component in the tree represents an *extremal* region. These are identified by the fact that all pixels in the region have a gray value strictly larger or strictly smaller than the pixels in the outer boundary of the region. In the context of multi-channel images, the concept of *larger* or *smaller* is not well-defined and the component-tree cannot be trivially constructed in the same fashion.

To overcome this limitation, we consider the derivatives *between* neighboring pixels in x (row) and y (column) direction. The derivatives at the image coordinates between two pixels are approximated by central differences. Hence, for each image pixel at the coordinate (x, y), we compute four differences, two vertical ones and two horizontal ones. They are calculated as

$$\delta_{\text{upper}}\mathcal{I}(x,y) = \mathcal{I}(x-1,y) - \mathcal{I}(x,y) \tag{7.1}$$

$$\delta_{\text{lower}}\mathcal{I}(x,y) = \mathcal{I}(x,y) - \mathcal{I}(x+1,y) \tag{7.2}$$

$$\delta_{\text{left}}\mathcal{I}(x,y) = \mathcal{I}(x,y-1) - \mathcal{I}(x,y) \tag{7.3}$$

$$\delta_{\text{right}}\mathcal{I}(x,y) = \mathcal{I}(x,y) - \mathcal{I}(x,y+1), \tag{7.4}$$

where $\mathcal{I}(x,y)$ is the pixel value of the image $\mathcal{I}$ at coordinate (x,y). Independent of the number of channels of $\mathcal{I}$, the magnitude (absolute value for single-channel and the Euclidean norm for multi-channel images) of the derivatives is totally ordered and can be used for the tree construction. Conceptually, instead of computing the level sets from successive thresholds of the pixel values, the level sets are obtained from thresholds of the derivative magnitudes. As a consequence, the components of the derivative-based component-tree are characterized by the fact that each pixel within them has a derivative with a magnitude that is smaller than the derivatives at the boundary of the region. We denote such regions as *homogeneous* regions.

The concept of the derivative-based component-tree is illustrated in the toy example in Fig. 7.2. Homogeneous regions (e.g., regions with the same color) are identified by the fact that the pixels are connected by derivatives with a very small magnitude. Hence, the child nodes consist of disjoint and differently colored regions. With a growing derivative magnitude threshold, similarly colored regions merge into single components (e.g., red and pink, light-green and green) and, eventually, the whole image is connected.

7.2.1 Local Flooding Tree Construction

The tree can be efficiently constructed by a flooding-based immersion. The concept is very similar to the flooding-based immersion of the ordinary component-tree [161], with

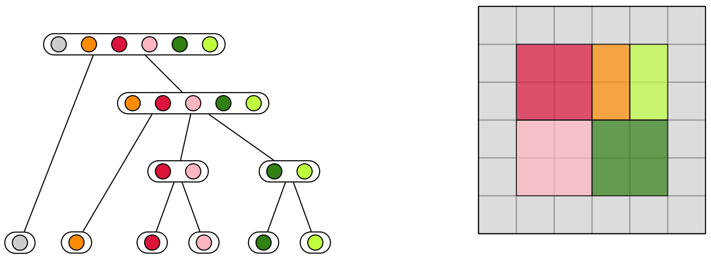


Figure 7.2: Toy example of the derivative-based component-tree for a three-channel image. Conceptually, the tree is constructed by iteratively thresholding the derivative magnitudes and connecting the resulting connected components. Hence, in an early stage, each of the uniquely colored regions is connected in a component (child node). In a next step, the most similar colors (pink/red and light green/dark green) are connected in parent components. Since the orange region has a similar distance to the red and green regions, it is connected to these components in a later step. Finally, the gray region, having the largest distance to all the colors, is connected to the other components in the root node, which represents the complete image.

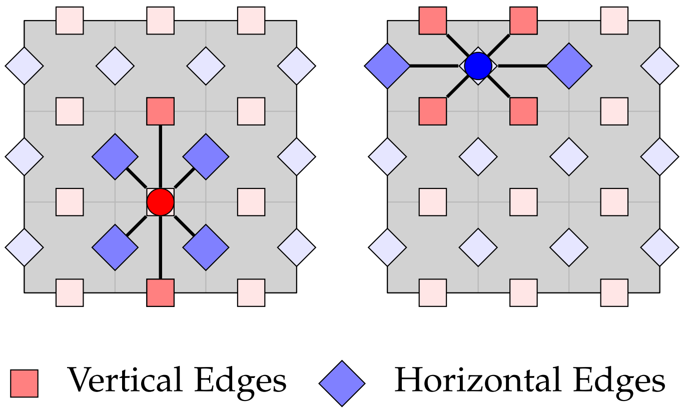
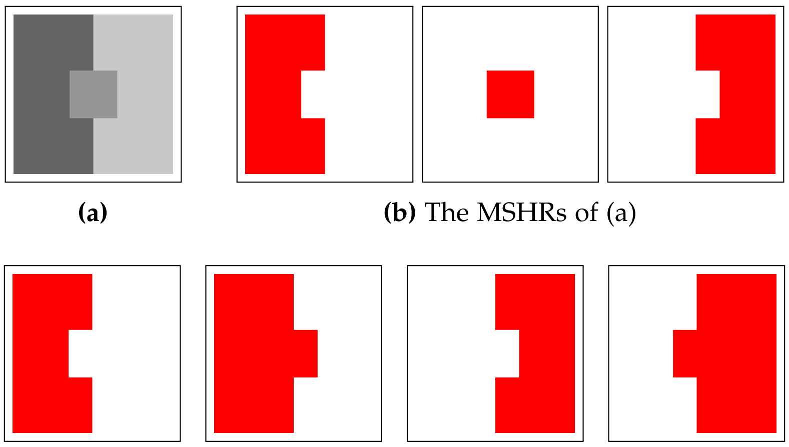


Figure 7.3: The derivatives are between two pixels (gray boxes) and each have six neighbors. On the left, the six neighbors of a vertical derivative (red) are displayed and on the right, those of a horizontal derivative (blue).



(a) **(b)** The MSHRs of (a)

(c) The MSERs of (a)

Figure 7.4: The center region of (a) is no extremal region since it is lighter *and* darker than its surroundings. Hence, regardless of the parameter settings, it will never be extracted as an MSER (c). On the other hand, the inner edges of the center region are smaller than its outer edges and hence it is a homogeneous region (b).

the exception, that the derivatives are flooded instead of the image pixels and need to be mapped to the image pixels in the construction process. In an initial step, starting from an arbitrary derivative, the flooding-based immersion searches for a derivative with a local minimal magnitude. The local minimum does not need to be strict. It is sufficient to find a lowland (a derivative where the neighboring derivatives have the same or larger magnitude). In general, this step requires the notion of the neighborhood of a derivative. This is determined by the two pixels the derivative connects. Each of these pixels has four derivatives: two vertical ones and two horizontal ones. However, since they share the derivative that connects them, both pixels only have three unique derivatives. Furthermore, the neighbors of vertical and horizontal derivatives are different. They are both displayed in Fig. 7.3 for clarification. The alternating 6-connected structure is the edge graph of the Khalimsky grid [57]. To ensure that each true inner distance has six neighbors and that no explicit border treatment is required, the derivatives at the border of the image are artificially added with an infinite magnitude. Therefore, there are $w + 1$ horizontal derivatives within each row of the image and w vertical derivatives, where w is the width of original image.

Starting from an arbitrary derivative, its magnitude is compared to the magnitude of its six neighboring derivatives. As soon as a derivative with a lower magnitude is encountered, the process stops checking the other neighbors and *floods* into the respective derivative. This process is continued until a derivative that has a locally minimal magnitude (not strictly minimal) is encountered. Then, the two pixels belonging to the respective derivative are merged into a new component of the component-tree. During this process, all visited derivatives are stored in a heap. As a consequence, each derivative needs to be visited exactly once during the tree construction. Hence, the flooding-based immersion is linear in the number of pixels.

In a next step, the derivative with the lowest magnitude in the heap is removed and compared to its neighbors. Either the process floods towards a new local minimum or, if all neighbors have been visited, merges the respective pixels with existing components. More precisely, every emerging derivative has four possibilities:

1. It connects two pixels that have not yet been visited $\Rightarrow$ a new child node is generated.
2. It connects a pixel that has not been visited yet to an existing component $\Rightarrow$ if the derivative magnitude is larger than those generating the respective component, a new parent node is generated. Otherwise, it is merely added to the component.
3. It connects two existing components $\Rightarrow$ a new parent node connecting both components is generated.
4. It connects two pixels already within the same component $\Rightarrow$ nothing needs to be done for this derivative. Continue with the next element in the heap

As soon as all derivatives have been visited (i.e., the heap is empty) the process terminates. A toy example of the tree construction that displays the horizontal and vertical derivatives and the respective local flooding-based approach is displayed in Fig. 7.5. It is important to note that similar to the the flooding-based immersion of the regular component-tree, the derivative-based component-tree does not depend on the choice of the starting point nor on the order in which the neighboring derivatives are visited for the local flooding-based immersion [236]. Furthermore, the resulting tree is the same as for a union-find-based immersion [34].

Although the flooding immersion walks through all of the image derivatives, the pixels belonging to the derivatives are added to the component-tree. The derivatives can be efficiently mapped to the pixel values by their linearized image index δ^l. The mapping is different for vertical and horizontal derivatives and can be computed as:

$$\begin{aligned}
\mathcal{P}_{\text{horiz}}(\delta^l) &= \{\delta^l - (w+1)(\lfloor \delta^l/(2w+1) \rfloor + 1), \\
&\qquad \delta^l - (w+1)(\lfloor \delta^l/(2w+1) \rfloor + 1) + 1\} \\
\mathcal{P}_{\text{vert}}(\delta^l) &= \{\delta^l - (w+1)(\lfloor \delta^l/(2w+1) \rfloor) - w, \\
&\qquad \delta^l - (w+1)(\lfloor \delta^l/(2w+1) \rfloor)\},
\end{aligned} \tag{7.5}$$

where w is the image width, $\mathcal{P}_{\text{horz}}$ are the index of the two image pixels for a horizontal derivative, and $\mathcal{P}_{\text{vert}}$ the index of the two image pixels for a vertical derivative, respectively.

7.2.2 Characteristics of the Derivative-based Component-Tree

The tree can be constructed efficiently by discretizing the derivative magnitudes. This is achieved by quantizing the derivative magnitudes into a certain number of *bins*. However, since the distribution of the derivative magnitude is far from uniform in natural images,

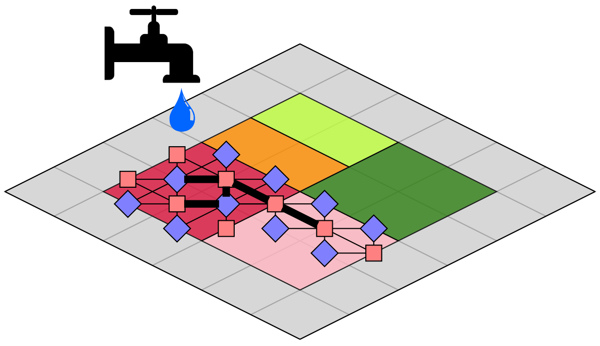


Figure 7.5: In the flooding-based immersion, starting from an arbitrary derivative, the immersion floods along the smallest neighboring derivative. When it finds a local minimum, the pixels belonging to the derivative are merged. In this example, the path first follows the zero derivatives within the red region, creating a red component within the component-tree. The next smallest derivative is at the border to the pink region, hence the path floods into the pink area.

it is reasonable to not bin the derivatives equidistantly. This was also observed by Forssén [80] when extracting so-called Maximally Stable Color Regions (MSCRs). The discretization enables to simplify the heap structure and has the further advantage that granularity of the component-tree can be configured: a very coarse binning leads to very compact trees, while a finer binning leads to more complex and descriptive trees. As shown in Fig. 7.6, although less descriptive, the coarse trees have the advantage that they can be computed significantly faster and that they reduce the computational complexity of the image processing techniques applied to them.

To ensure that the regions in the single components of the component-tree are pixel-precise and include each pixel within a homogeneous region, it is essential to consider the derivatives at their true position *between* two pixels. Other approaches flood the gradient magnitude image directly for simplicity [73]. Although this enables the algorithm to assume a 4-connected neighborhood of the derivatives and to use an ordinary component-tree construction algorithm, the resulting components do not contain the true pixels of each homogeneous region. The differences between two pixels influence the gradient of both pixels. As a consequence, every edge is it least 2 pixels wide in the resulting images and only regions that are large enough can be extracted. Furthermore, the resulting homogeneous regions are smaller than the actual homogeneous regions in the input image. For example, when flooding the gradient magnitude image of the toy example in Fig. 7.2 directly (as in [73]), it is impossible to extract the small regions. The gradient magnitude image has a high value at each pixel and has no valleys to flood. This is highlighted in Fig. 7.7.

7.2.3 Implementation Details

The resulting derivative-based component-tree has the same structure as its gray-value counterpart and, therefore, the same algorithms may be applied. Nevertheless, the following modifications help to improve the algorithm's robustness:

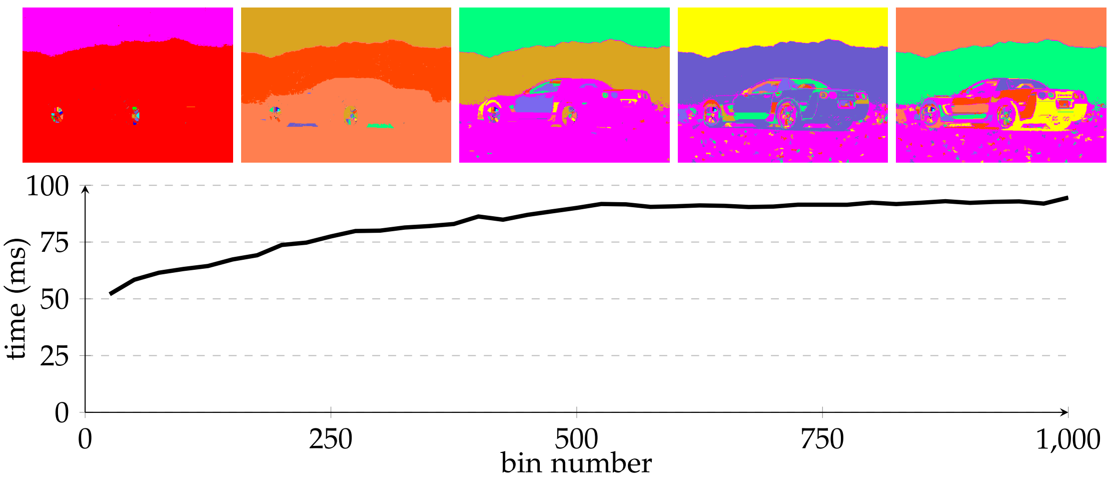


Figure 7.6: The granularity of the component-tree can be configured by quantizing the derivative magnitudes into bins. The number of bins influences the runtime and the granularity of the possible segmentations. The measurements were obtained for 50 random images from PASCAL VOC 2007 [75]. Since the variation of the runtime was very small ($\approx 0.3ms$), errorbars have not been added.

1. Since the derivative-based component-tree works on pixel differences, it is susceptible to image noise. This was also observed by Forssén [80], who proposed to perform Gaussian smoothing as a preprocessing step. Unfortunately, this may add artificial components at strict vertical or horizontal image derivatives. We found that edge-preserving smoothing, such as bilateral or guided image filtering [230], helps to remove these artifacts. An example is shown in Fig. 7.8.

2. Furthermore, since very small image regions are rarely of interest, we found it very useful to restrict the minimal area a component must have to create a node within the tree. This leads to more compact trees and can significantly reduce the runtime, while it has virtually no impact on later queries of the component-tree. In contrast to [231], we do not delete the regions from the tree explicitly and then apply a region growing algorithm to extend the remaining components. Instead, the flooding-based immersion allows to merge these regions into their parent component during the tree construction implicitly. For this, each component is only added to the tree once it is big enough. Until then, the connecting pixels are added to the component without creating parent components. This allows to filter the regions without adding any computational overhead. Each derivative still only needs to be visited once by the algorithm.

In our experiments, we also use both concepts for the gray-scale component-tree since they are equally applicable there.

In general, our approach has a larger computational overhead than that of the gray-scale component-tree. First of all, there are around twice as many image derivatives as there are pixels. Furthermore, each derivative has to consider six derivative neighbors compared to four pixel neighbors. Hence, the construction process is expected to be approximately three times slower than that of the gray-scale component-tree for a single

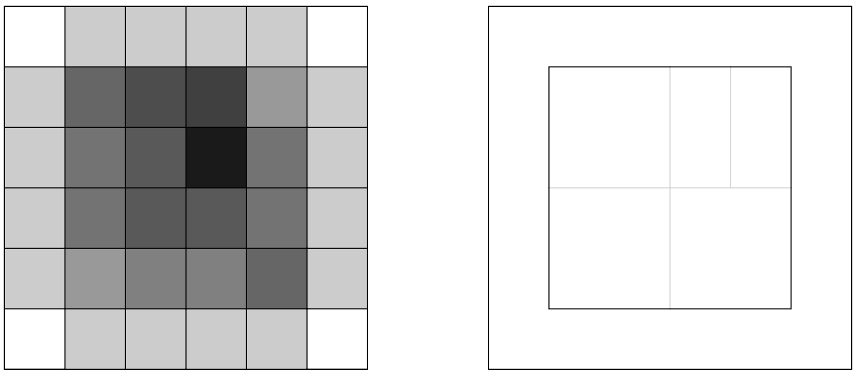


Figure 7.7: The gradient magnitude image (left) and the derivatives *between* the image pixels (right) of the toy image in Fig. 7.2 are indicated. The gradient magnitude image has the problem that it spreads derivatives into all adjacent pixel values. It is not suited for extracting small homogeneous regions and generates eroded versions of the actual homogeneous regions in an image.

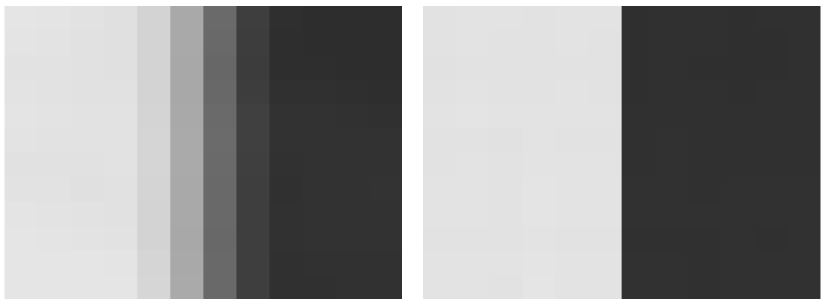


Figure 7.8: Gaussian smoothing (left) and edge-preserving bilateral filtering (right) is applied to a vertical image edge. Gaussian smoothing adds artefacts that create slim artificial homogeneous regions.

polarity (plus the overhead of calculating the image derivatives). However, since the derivative-based component-tree implicitly captures all regions that are lighter *or* darker than their background (for gray-scale images), the runtime is essentially only 1.5 times slower when extracting regions of both polarities. This factor is confirmed empirically in Fig. 7.9. Note that the complexity of the tree traversal is the same for both component-trees. All of the routines presented in this chapter are implemented in C and the code is optimized and parallelized where possible. In general, the tree construction is very efficient and requires less than 250ms for an 800×1000 image on an Intel Core i7-4810 CPU @2.8GHz with 16GB of RAM with Windows 7 (x64).

7.3 Tracking Maximally Stable Homogeneous Regions

The constructed derivative-based component-tree can essentially be used for the same image processing tasks as its gray-scale counterpart. For example, the tree can be used to efficiently extract stable regions similar to MSERs. The only difference is that the tree nodes do not consist of extremal regions but of homogeneous regions.

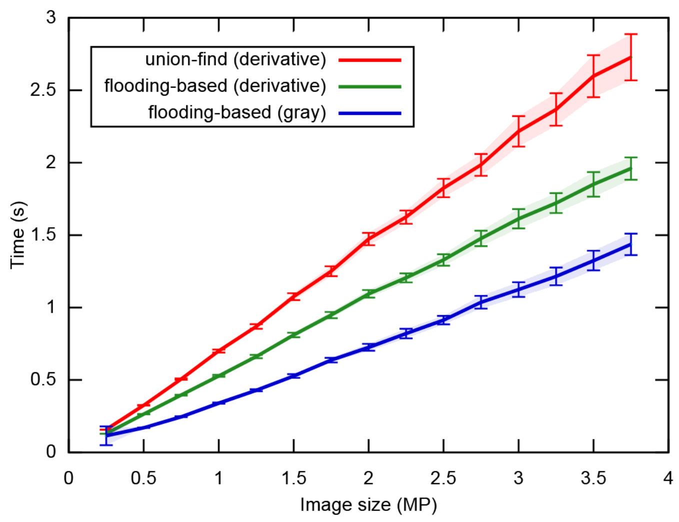


Figure 7.9: The runtime and standard deviation of constructing the derivative-based component-tree with the proposed flooding-based immersion and the standard union-find-based algorithm (Kruskal). The average runtimes are computed using 30 randomly selected pictures from the Pascal VOC 2007 dataset [75]. The computation time of the ordinary gray-scale component-tree is added for reference.

7.3.1 Maximally Stable Homogeneous Regions

As mentioned above, homogeneous regions are characterized by the fact that each pixel within the region has a vertical or horizontal derivative with a smaller magnitude than all outer derivatives of the region. Otherwise, the concept of stable regions is the same. Hence, both approaches share the same parameters and scale equally with growing image sizes. Let $R_1, \ldots, R_{i-1}, R_i, R_{i+1} \ldots$ be a set of nested homogeneous or extremal regions, respectively (i.e., $R_i \subset R_{i+1}$). In the context of component-trees, the index i encodes the gray-value threshold or the derivative magnitude threshold that generated the region. A Maximally Stable Region R_{i*} in the context of an MSER and an MSHR is a region that has a local minimum of

$$s(i) = \frac{|R_{i+\Delta} \setminus R_{i-\Delta}|}{|R_i|}, \tag{7.6}$$

at $i*$. Here $|\cdot|$ denotes the cardinality and Δ is a parameter of the method. The parameter Δ encodes how stable a region is over $\pm\Delta$ thresholds. The larger the value, the more stable the regions must be.

In a component-tree, the sequence of ancestor and descendant nodes for a node is a set of nested regions. To simplify the computation, each node of the derivative-based component-tree stores its area and the smallest derivative magnitude that connects its inner points (these can be adapted on the fly during the tree construction). Hence, $s(i)$ can be computed for each node by checking the area of the ancestor and descendant nodes at a distance of Δ, respectively. The resulting Maximally Stable Homogeneous Regions (MSHRs) are possibly overlapping regions that do not change their area significantly over a given derivative magnitude range.

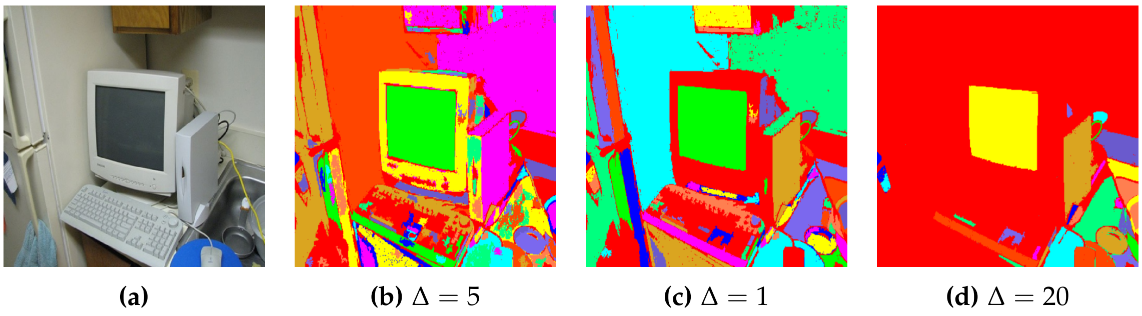


Figure 7.10: Derivative-based component-trees can be used to extract stable regions from color images. The images (b)-(d) display the extracted MSHRs from image (a) for different settings of Δ. Larger values of Δ lead to a coarser segmentation.

Table 7.1: The True Positive Rate (TPR) of MSERs [146], MSCRs [80] and MSERs augmented with MSHRs on the ICDAR 2015 "Focused Scene Text challenge" [110] dataset are displayed. MSHRs are able to outperform MSCRs and a combination of MSHRs and MSERs returns the best segmentation results.

Method	$\Delta = 1$	$\Delta = 5$	$\Delta = 10$
MSER [146]	89.69	85.44	79.88
MSCR [80]	80.75	71.41	57.28
MSHR	88.73	83.69	76.21
MSER + MSCR	90.84	87.68	80.76
MSER + MSHR	**93.64**	**89.12**	**84.46**

An example of MSHRs for an image from PASCAL VOC 2007 [75] is shown in Fig. 7.10. The parameter Δ in (7.6) determines the granularity of the segmentation. An advantage of using the derivative-magnitude-based component-tree for the MSHR extraction process is that various different parameter settings of Δ can be used to extract a large collection of different regions without significantly increasing the runtime. The complexity of the tree traversal is negligible compared to the time required for the tree construction.

7.3.2 MSHR Versus MSER

The building blocks of MSERs are extremal regions that either have gray-values strictly larger or strictly smaller than their outer border. This means that some regions of interest can never be segmented with the help of MSERs. On the other hand, the building blocks of MSHRs are homogeneous regions, which require that each inner pixel has a smaller derivative than all outer derivatives of the region. Hence, as shown in Fig. 7.4, our approach is able to extract regions that simultaneously have a lighter *and* darker background. In the applications section, we show how this attribute can be very helpful in applications such as OCR, where MSER-based approaches fail.

Although initially proposed as stereo features [145], MSERs are used extensively as a preprocessing step for Optical Character Recognition (OCR) systems [115, 159]. Note

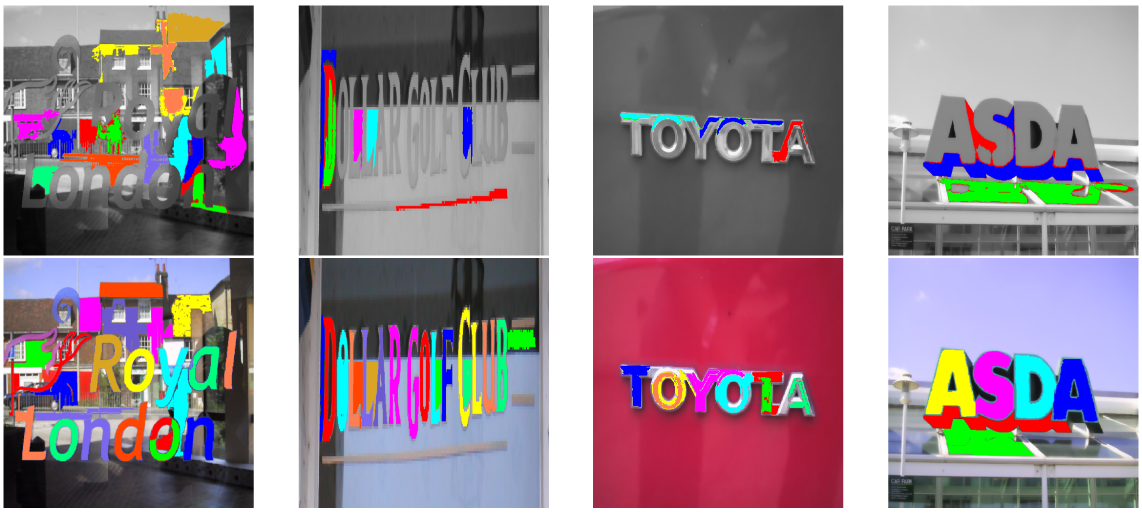


Figure 7.11: The MSER segmentation in the first row has difficulties with characters that are simultaneously lighter *and* darker than their background. The MSHR segmentation is able to extract all relevant character regions and displayed in the second row.

that although the performance of MSER-based OCR systems can be outperformed by techniques building on convolutional and recurrent neural networks [31, 172], they are still used in many running OCR systems. This is due to the fact that they have a much lower computational complexity and can be computed in real time on embedded devices and machines without a GPU.

Since MSERs assume the regions to be extremal, they cannot extract characters that have a lighter *and* darker background (see Fig. 7.4). This can be a problem in OCR systems, since most approaches fail if the characters cannot be segmented in an early stage. We evaluate the text segmentation capabilities of MSER, MSCR, [80] and MSHR on the ICDAR 2015 ”Focused Scene Text challenge“ [110] dataset. As in common in the ICDAR challenges [110], we consider a character to be found if it overlaps the ground truth bounding box according to the PASCAL overlap criterion [75] by more than 50%. As shown in the TPR displayed in Table 7.1, MSHR clearly outperform MSCR. However, when applied alone, both methods are weaker than MSERs. The different approaches of either flooding the derivatives or the image pixels essentially creates regions with complementary attributes. The complementary attributes of MSERs and MSHRs can be used to combine both methods. Hence, by extracting both MSERs and MSHRs, the recall rates can be improved considerably. As shown in Table 7.1, the combination of MSHRs and MSERs is able to significantly improve the segmentation obtained by only MSERs. Note that the initial recall of the segmentation is an important indicator of how well an OCR system can perform. Later steps are usually concerned with grouping and filtering out undesired regions. Hence, what is not found in an initial step will not be found. A handful of example images where MSHRs are superior to MSERs are presented in Fig. 7.11.

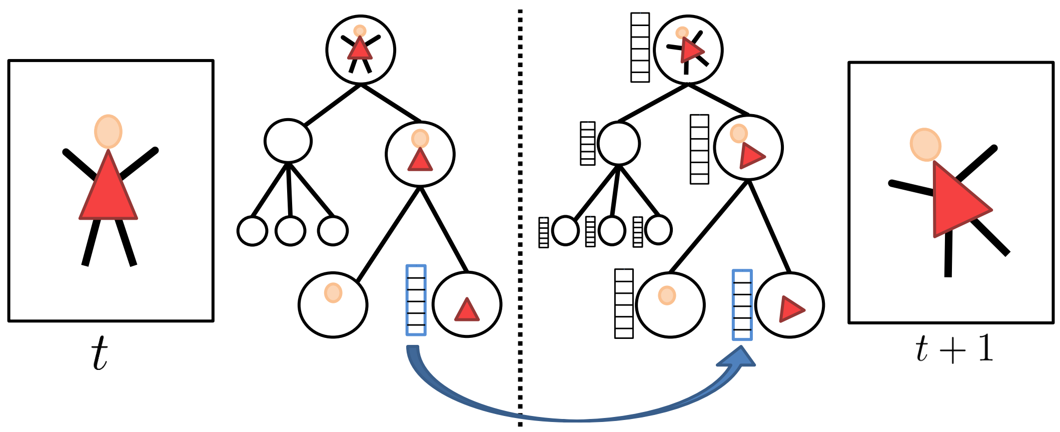


Figure 7.12: Overview of the proposed MSHR tracking approach. At time step t, the features are calculated for the currently tracked MSHR. In the next time step, merely the derivative-based component-tree is constructed and the features are calculated for every node. These features are then matched to those of the MSHR from time step t.

7.3.3 MSHR Tracking

In MSHR tracking, the object is represented by a pixel-precise segmentation. This enables a precise localization of the object and allows the object to undergo nonlinear deformations. However, the tracker assumes that the object can be represented by a homogeneous region. Hence, complex objects such as cars or pedestrians cannot be tracked as a whole by the proposed tracking scheme. Instead, the tracker is restricted to elements of objects that are stable, such as the license plate, windscreen, or the eyes of a human.

Object tracking is typically divided into three stages: the initialization, the tracking stage, and the update. In the first step, the object location is usually given and the tracker is initialized with the input image and the region of the object. In the second step, the tracker identifies the most probable object location in a new frame. In the final step, the tracker updates its object representation from the new object location and appearance. The steps of the proposed MSHR tracker can be divided into the above three stages without loss of generality.

Tracker Initialization

As is common, the tracker initialization assumes that the initial region of the object is known. The region may be given by a bounding box or by a pixel-precise segmentation. The tracker then constructs the derivative-based component-tree for the given input region. For each of the components in the tree, the stability from (7.6) is calculated and the MSHRs are extracted. In general, MSHRs divide the image into multiple, possibly overlapping, connected components. Then, the locally most stable regions are extracted from the target and selected for tracking. Hence, for a given input region, multiple MSHRs may be tracked.

Tracking Step

Given the object location in the prior frame, a suitable search domain is determined in the current frame. In all of our experiments, a rectangular domain with twice the object's bounding box extents as search region is used. As above, the derivative-based component-tree is computed for the search domain. However, in contrast to the initialization, the stability is not calculated for each region. Instead, region and gray-value features are computed for every component in the tree. Only features than can be efficiently calculated by region and gray-value moments [208] are used. This enables the algorithm to calculate these features on the fly during tree construction with little computational overhead.

The moment of order (p, q) of a region $\mathcal{R}$ is defined as

$$m_{p,q} = \sum_{(r,c)\in\mathcal{R}} r^p c^q, \tag{7.7}$$

where $p \geq 0$ and $q \geq 0$. Since the flooding-based immersion considers each image pixel in the tree construction anyway, our choice of features can be calculated while constructing the component-tree. We use the area of the region ($m_{0,0}$), the center of gravity ($m_{1,0}/m_{0,0}, m_{0,1}/m_{0,0}$) and the ellipse parameters r_1, r_2 and θ as tracking features. The ellipse parameters can be calculated with the normalized moments: see [93] for details. Analogously, we use gray-value moments to calculate the average gray-value and the gray-value deviation of the single channels as further features. Note that our selection of features makes the approach invariant to rotations of the MSHRs.

To further improve the robustness, the single features in the matching step can be weighted for specific applications. For example, if the object undergoes heavy deformations, but has a relatively constant color, the weight of the region moments is reduced and the gray-scale features' weights are increased. The weights are estimated automatically from the variation of the color and the variation of the region moments within the first five frames.

In the tracking step, the features from the MSHRs from the initialization are matched to the components in the derivative-based component-tree of the search region. Matching the MSHRs to all of the nodes in the tree improves the robustness and ensures that the search is not restricted to only the *maximally* stable homogeneous regions. The general idea of MSHR tracking is visualized in Fig. 7.12. The matching cost is determined by a weighted l_1 distance of the single features. Here, depending on the applications, the weights can be adapted before or during the tracking . In the examples, we weighted the mean gray-value of the regions the highest.

Tracker Update

To enable robust tracking, in contrast to [72], we update the region features incrementally in each frame. This enables the algorithm to handle short occlusions and detection failures in single frames. Hence, after successfully locating the node that best fits the MSHR to be tracked, we update the feature vector as

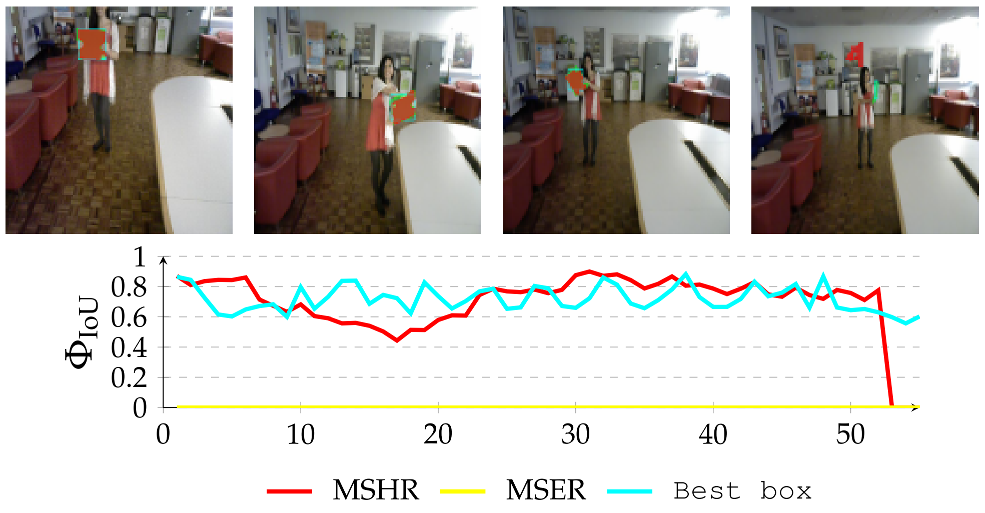


Figure 7.13: `book` from VOT2016 [116]. Since the gray-scale region is not an MSER, it cannot be tracked with MSER tracking (see Fig. 7.14 for details). The overlap scores of the MSHR tracking are comparable and sometimes even better than the overlap the best possible axis-aligned tracker could theoretically achieve.

$$\text{feat}_{t+1} = (1 - \lambda)\text{feat}_t + \lambda\text{feat}_{t+1}. \tag{7.8}$$

In all our experiments we used $\lambda = 0.5$. If no region with a large enough similarity is found, the features are not updated and the object is searched for with the old feature values in the next frame.

7.4 Applications

2D Object Tracking The proposed MSHR tracking approach is not restricted to bounding boxes. Hence, to evaluate the quality of the tracking results, we manually annotated dense pixel-precise segmentations of a handful of scenes from the OTB [239] and VOT2016 [116] datasets. Otherwise, the given bounding box ground truth would introduce an undesired bias when measuring the overlap scores of by-pixel segmentations. As accuracy measure, we use the Intersection over Union (IoU) criterion. Unfortunately, the cars within the Playing for Tracking Data dataset proposed in Section 5.2 are not MSERs or MSHRs themselves. This makes it difficult to track the cars on the whole and an evaluation of the results is difficult. Hence, we restrict the evaluation to the few sequences in the OTB [239] and VOT2016 [116] datasets that can be approximately quite well by MSERs and MSHRs. The PFTD is used extensively in Section 9.2.

To bring the results into perspective, we compute the best possible overlap an axis-aligned tracker could obtain for the segmentation of a given scene (see Section 4.2). By these means, the performance gain of using segmentations can be highlighted without introducing a bias by choosing a specific set of state-of-the-art axis-aligned trackers to

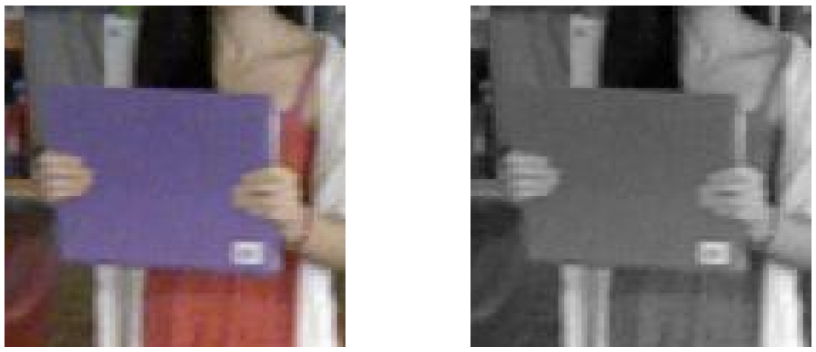


Figure 7.14: A close-up of `book` from VOT2016 [116]. The book is not an MSER in the gray-scale image since its background is lighter *and* darker than the book itself. Hence, it cannot be tracked with the existing MSER tracking. The book is a homogeneous region however, and can efficiently be tracked with the proposed MSHR tracking. See Fig. 7.13 for the overlap scores.

compete against. We refer to this tracker as the `Best box`.

To understand the difference between MSER and MSHR tracking, we further compare our approach to a version of the MSER tracker [72]. To focus the evaluation on the different regions both approaches use, and not on their features, we use the exact same parameters and moment-based features for both approaches.

For color images, their is a significant difference of MSER and MSHR tracking. Fr example, in the `book` sequence from VOT2016 [116], the MSER tracker fails completely, as shown in Fig. 7.13. The book is, by definition, not an extremal region in the gray-scale image, as can be seen in more detail in Fig. 7.14. Hence, the initialization is unsuccessful and the MSER tracker fails. Nevertheless, the book is a homogeneous region in both the gray-scale and the color image and, accordingly, the MSHR tracker is successful. In most frames, the MSHR tracker is even able to outperform the `Best box` and obtains an average IoU of 0.7.

For the `book` sequence, the MSHR tracking requires at maximum of 24ms per frame and for the `dress` sequence a maximum of 18ms per frame. The algorithm is implemented in HALCON and run on an Intel Core i7-4810 CPU @2.8GHz with 16GB of RAM with Windows 7 (x64).

For the gray-scale sequence `dress` from OTB [239], the MSER tracker outperforms the `Best box` and the MSHR tracker by a small margin, as is displayed in Fig. 7.15. In the respective sequence, the MSER tracker is able to track the head and the dress of the dancer, while the MSHR tracker only tracks the dress. Hence, the overlap scores of MSERs are superior. Nevertheless, it is important to note that both approaches are compared against the *best possible* axis-aligned tracker and, accordingly, the overlap scores are impressive.

3D Object Segmentation For MSER tracking, Donoser and Bischof [72] presented three different applications; license plate tracking, face tracking, and the segmentation of a fiber network. In the third application, a fiber network is reconstructed in 3D by tracking

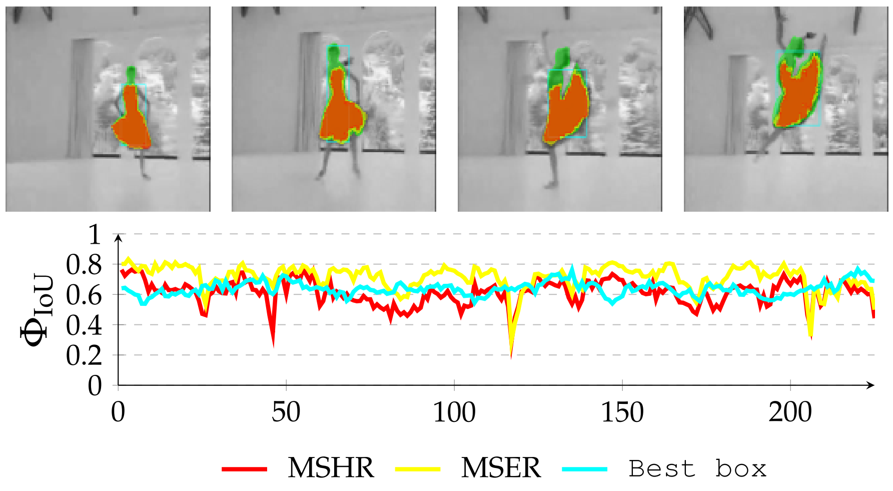


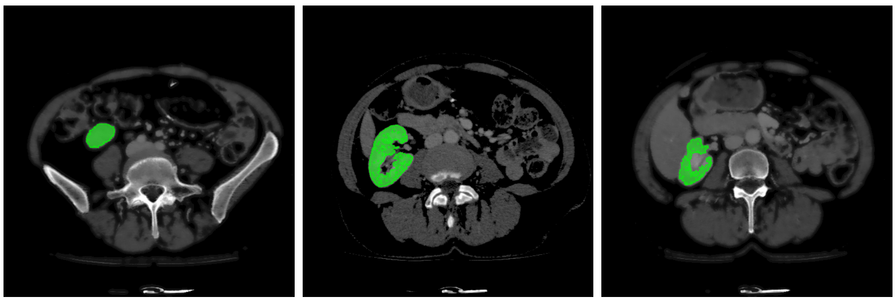
Figure 7.15: `dress` from OTB [239]. For this gray scale scene, the MSER tracker is able to outperform the MSHR tracker. The tracker also clearly outperforms the best possible overlap an axis-aligned tracker (`Best box`) can achieve for the segmentations within the scene.

a slice of the data along the axis orthogonal to the image data. Analogously, we track organs in slices of a Computed Tomography (CT) scan to generate a 3D segmentation. We use the CT data provided in the 3DIRCADb dataset[1] [200].

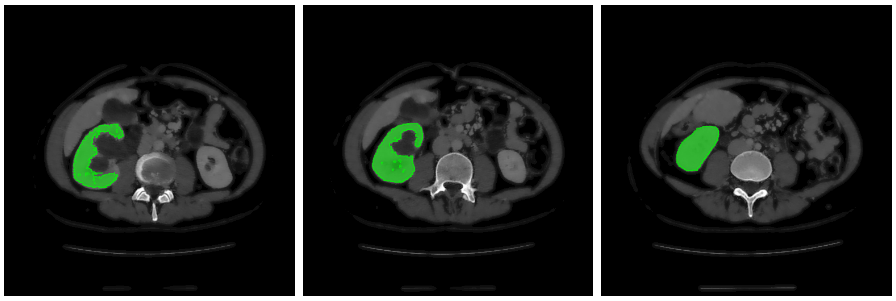
To initialize the tracking process, the organ is segmented in an arbitrary slice of the CT data by a bounding box. The most stable MSHR (i.e., with the lowest value of $s(i)$ from (7.6)) is then selected in the initialization process for tracking. The respective MSHR is tracked through the slice data along the axis orthogonal to the image data. An example of the tracked regions is visualized for two examples in Fig. 7.16. Given the segmentations of the single slices, the organ (in this case the right kidney) can be reconstructed in 3D. We compare the reconstruction for MSER and MSHR tracking in Fig. 7.17. To enhance the visualization, the datapoints are triangulated and the surface normals calculated. Since the contrast of the organs can be very low in CT images, the MSER tracking has difficulties catching the organ boundaries. Furthermore, the organ is sometimes partly lighter *and* darker than the background, which may lead to MSER tracking failure. The proposed MSHR tracking copes well with these difficulties, and the reconstructions are significantly better.

The tracking of the regions in the CT slices is extremely efficient and only requires an average of 5ms per slice. Hence, for the 45 slices in Fig. 7.17 the complete 3D reconstruction process, which includes the triangulation ($\approx 1s$), the calculation of the surface normals ($\approx 130ms$), and the segmentation ($\approx 220ms$), requires only around 1.5s.

[1]The dataset is available on `http://ircad.fr/research/3d-ircadb-01`



(a)



(b)

Figure 7.16: Two examples sequences from the 3DIRCADb dataset [200]. Given an initial selection of a single slice (the middle image in (a) and (b)) of the right kidney, the proposed MSHR tracking tracks the region forward and backward in space. The segmented slices can be used to reconstruct the organ, see Fig. 7.17 for an example reconstruction.

7.5 Discussion

The proposed tracker is extremely efficient and able to track arbitrarily shaped regions in image sequences. The tracker is able to cope with arbitrary deformations of the input region. Furthermore, the model update step allows to track regions robustly. The presented tracker can be used for 2D temporal tracking or for 3D object segmentation. Since the homogeneous regions are more general than extremal regions, the presented approach can help to solve problems where MSER tracking would typically fail. However, the restrictions placed on the tracker are still very strong. The objects either need to be homogeneous regions themselves or contain sufficient parts that are homogeneous regions. As a consequence, the tracker is difficult to use for more general tracking scenarios.

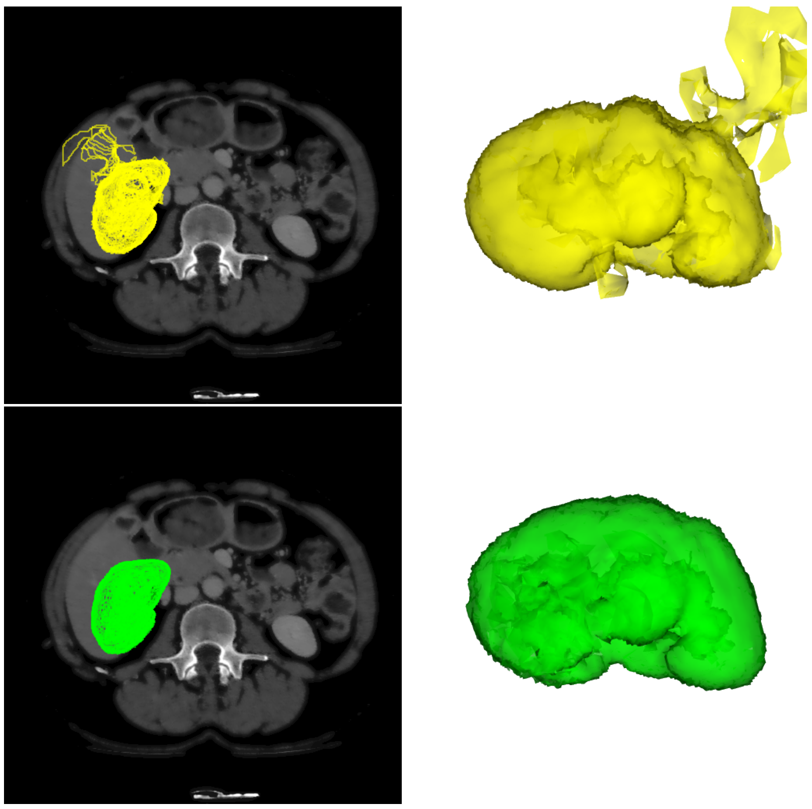


Figure 7.17: In the first row, the reconstruction of the right kidney is displayed for MSER tracking. The low contrast and the fact that the background is partly darker and lighter than the objects makes the reconstruction noisy. The proposed MSHR tracking can cope with these situations and the reconstruction is significantly better.

8 Shape Model Tracking

In the previous chapter, we proposed an MSHR-based tracking approach that works efficiently for selected applications. Unfortunately, although the approach can track deformable objects, it is restricted to tracking objects that can be represented by homogeneous regions. There are many industrial applications where this restriction is too strong. In this chapter, we present a more general tracking approach that is merely restricted to roughly rigid objects. The approach uses subpixel-precise image points and directions to track objects with high accuracy and does not require any significant object texture. It can determine the object position, scale, and rotation with subpixel-precision. The tracker returns a reliable score for each frame and is capable of self diagnosing a tracking failure, propagating the object motion, and reinitializing the tracking by itself. Furthermore, the choice of the similarity measure makes the approach inherently robust against occlusion, clutter, and nonlinear illumination changes.

In Section 8.1, we present the shape model localization technique of Steger [205, 206] that forms the basis of the proposed tracker. In Section 8.2, we present the baseline shape model tracker that was originally proposed in Böttger *et al*. [27] and identify the strengths and weaknesses of the baseline. Next, in Section 8.3, we present extensions of the baseline to tackle its weak spots and improve its performance. We add queues to allow long-term tracking without significant drift and present a dynamic model generation that allows to adapt the number of model pyramid levels on the fly. Furthermore, we represent a robust hierarchy of models that allows to track objects that severely change their scale within a sequence. The chapter is concluded with a discussion in Section 8.4. Some of the results presented in this chapter are covered in Böttger and Steger [26] and Böttger *et al*. [27].

8.1 Fundamentals: Shape Model Matching

The shape model tracker builds on the efficient shape-based object recognition technique of Steger [205, 206]. The object recognition technique is essentially a template matching scheme that searches for the object densely for each transformation and image position. The template model is represented by a sparse set of model points. To reduce the computational complexity, an image pyramid of different scales is created and the

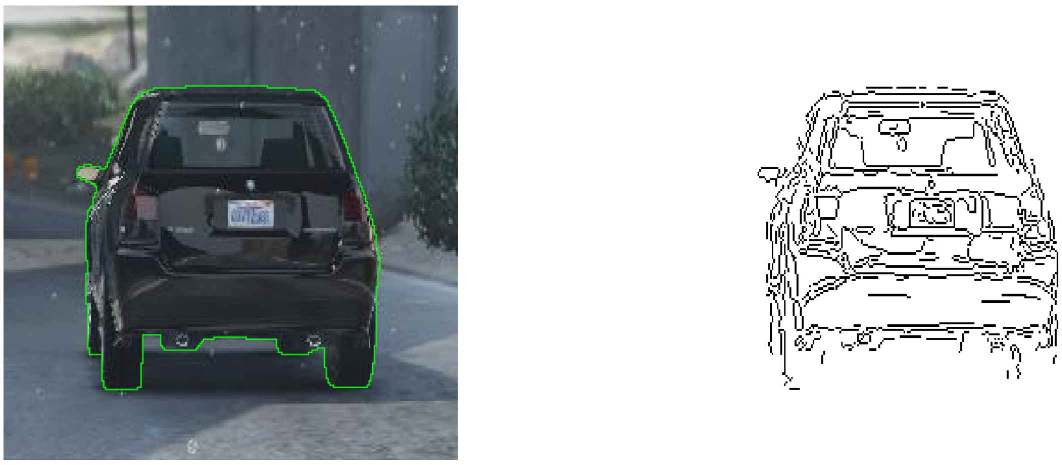


Figure 8.1: On the left, the typical input for the shape-based matching is displayed. The model initialization assumes the region (displayed as the green border) of the object is known. On the right, the points of the corresponding shape model are displayed.

initial dense matching process is merely conducted on the coarsest level. Promising transformations are then tracked through the image pyramid and either discarded or refined on each level. In the following, we describe the main steps of the shape-based object recognition [205, 206].

Model Generation The shape model is generated from a template image and an arbitrarily shaped region of interest (ROI). Model point candidates are extracted by applying a threshold on the Sobel filter edge amplitude of the input ROI. To thin out the number of points, non-maximum suppression is applied with automatically estimated thresholds (see [220] for details). An example of the pixel-precise model points for an input image displayed in Fig. 8.1. The pixel-precise model points can then be refined to subpixel precision, which is described in more detail in Chapter 3.3 of [204]. The coordinates of the model points are all expressed relative to an arbitrary reference point. We use the center of gravity of the ROI for simplicity. An independent shape model is generated for each pyramid level to account for scale-space effects. In the image pyramid, neighboring edges may merge in coarser levels or disappear completely. Hence, independent models ensure the optimal model is used on each pyramid level.

Model Localization The model localization essentially amounts to finding the best matching candidate within the target image in a template matching framework. The process is divided into two steps: (1) the initial detection of promising object locations in the coarsest pyramid level and (2) the tracking of these possible locations through the image pyramid and either refining or discarding them.

1. In a first step, the search space $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$ is determined. The search space is the Cartesian product of all possible values for s, θ, and every point in the search region $\mathcal{S}$. For each element in $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$ the model is then compared to the target image. Since this is a computationally demanding process, it is only conducted on the

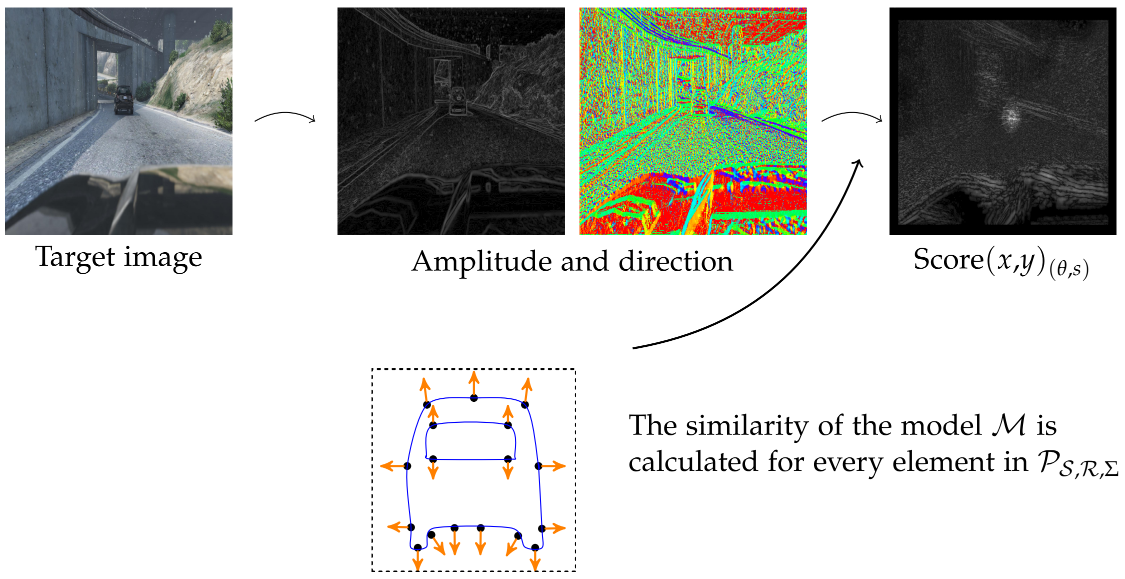

Target image
Amplitude and direction
Score$(x,y)_{(\theta,s)}$
The similarity of the model $\mathcal{M}$ is calculated for every element in $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$

Figure 8.2: The steps performed during the matching of a model: First, the gradient amplitudes and direction of the target image are calculated. The amplitudes are required for the subpixel-precise refinement of the object position. Then, the maximum similarity of the model $\mathcal{M}$ from Fig. 8.1 is calculated for the position, scale, and angle within the discretized 4d search space.

coarsest pyramid level, where specifically $\mathcal{S}$ is very small. Furthermore, due to the low resolution on the coarsest pyramid level, the resolution of $\mathcal{R}$ and Σ can also be decreased considerably. To compute the similarity, a direction vector is computed for a dilated search region. The size of the dilation is determined by the maximum extent of the transformed model (hence from $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$). The direction vectors in the dilated search region are identified as $\vec{e}_{x,y} = (v_{x,y}, w_{x,y})$. The $\vec{e}_{x,y}$ can essentially be obtained by any method that returns a direction vector for each image point, e.g., edge detectors such as Sobel or Canny [33] or line detectors [204, 207]. However, it is essential that the method for obtaining the directions of the model and the directions in the search image are compatible. After obtaining $\vec{e}_{x,y}$, it is possible to evaluate the similarity of the tracking model $\mathcal{M}$ for every image point and transformation in $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$. The similarity of a transformed model point is given by the sum of dot products of the normalized direction vectors of the transformed model and the target image:

$$score(\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma})_{\mathcal{M}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\vec{d}'_i \cdot \vec{e}_{p'_i}}{\|\vec{d}'_i\| \|\vec{e}_{p'_i}\|}, \tag{8.1}$$

with $score : \mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma} \rightarrow [-1, 1]$, p'_i computed from (2.5), and $\vec{d}'_i$ computed from (2.6). The normalization of the direction vectors to length 1 makes the similarity measure robust to non-linear illumination changes. Furthermore, the measure is robust to occlusion, clutter, and a moderate amount of defocussing [205, 206]. It is possible to adapt the similarity measure to be invariant to the contrast direction of the edges.

This may be achieved by considering the modulus

$$score(\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma})_{\mathcal{M}} = \frac{1}{n}\left|\sum_{i=1}^{n} \frac{\vec{d}\,'_i \cdot \vec{e}_{p'_i}}{\|\vec{d}\,'_i\|\|\vec{e}_{p'_i}\|}\right|, \tag{8.2}$$

where $score : \mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma} \rightarrow [0,1]$. Furthermore, for the measure to ignore local contrast changes, the similarity measure can be modified to

$$score(\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma})_{\mathcal{M}} = \frac{1}{n}\sum_{i=1}^{n} \frac{|\vec{d}\,'_i \cdot \vec{e}_{p'_i}|}{\|\vec{d}\,'_i\|\|\vec{e}_{p'_i}|}, \tag{8.3}$$

where $score : \mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma} \rightarrow [0,1]$. However, the element-wise modulus indirectly adds a significant computational overhead and is rarely used in practice.

By setting a minimal required similarity $score_{\min}$, it is possible to discard possible image positions and transformations without needing to compute the similarity for every model point in (8.1). This helps to avoid a large number of computations. See [208] for details.

2. In a second step, the image locations and transformations that are larger than a specific threshold $score_{\min}$ in the coarsest pyramid level are tracked through the image pyramid. The model on the lower (finer), pyramid level is tested for the respective locations and transformations and their similarity is computed. Locations and transformations that continue to yield a score larger than $score_{\min}$ are propagated deeper into the image pyramid, while those that do not are disregarded. The restriction to promising locations and transformations in the deeper pyramid levels provides significant reduction of the computations and makes the process very efficient.

The localization process is visualized in Fig. 8.2. At this point the optimal position, angle, and scales are determined with pixel accuracy. The $score_{\min}$ is also an indicator of how much of the object is visible. For example, if $score_{\min} = 0.8$, at least 80% of the model points need to be visible to enable a score of 0.8 to be reached. An example of different settings of $score_{\min}$ are displayed in Fig. 8.3.

In the following, we describe how the optima may be further refined to subpixel accuracy.

Subpixel-precise Refinement The accuracy of the localization step depends on the chosen discretization of $\mathcal{R}$ and Σ as well as the pixel resolution of $\mathcal{S}$. To refine the match, a second order polynomial is fit to the neighborhood of the best matches. The coefficients of the polynomial may be obtained by convolution with facet model masks [205, 206]. For example, for similarity transformations, the polynomial is fit to the $3 \times 3 \times 3 \times 3$ neighborhood of the 4d (x, y, θ, s) parameter space of the best matches. The maximum of the respective polynomials are the subpixel-precise optima of the object locations and transformations. An example of a second order polynomial fit to the surrounding

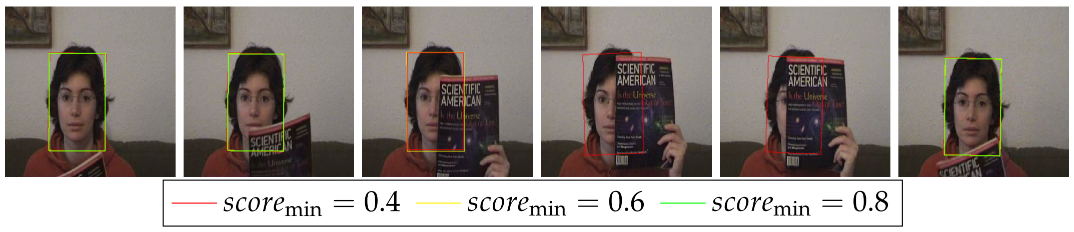


Figure 8.3: The results for 3 different parameters of $score_{\min}$ are displayed for the `FaceOcc1` sequence from OTB-2015. The value of $score_{\min}$ is an indicator of how much the object is allowed to be occluded. Lower values improve the robustness to occlusion but also require more time, since more score values need to be computed. For $score_{\min} = 0.8$, the object is not detected for the third to fifth image, while for $score_{\min} = 0.6$ it is lost for the fourth and fifth image. However, all approaches recover when the occlusion ends.

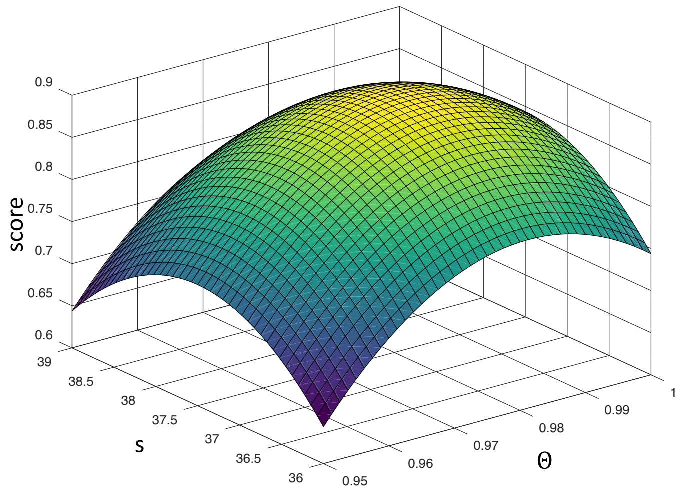


Figure 8.4: To compute object position and transformation parameters with subpixel precision, a second order polynomial is fit to the pixel-precise optima computed by the matching process. To simplify the visualization, in this image the polynomial is merely fit to the scale parameter s and the rotation angle θ. The maximum of the polynomial yields subpixel-precise optimum of s and θ, which are not restricted to the discretization of the transformation space.

of the pixel-precise optimal values for $\mathcal{R}$ and Σ is displayed in 8.4. The maximum of the polynomials are not restricted to the pixel grid nor to the discretization of the transformation space.

In a last step, a least-squares refinement is applied to the transformation parameters. This helps to further improve the localization accuracy and the robustness. The least-squares refinement assumes a good initial approximation of the current transformation and improves the global similarity transformation for all points. For each model point p_i, the best point match in the direction of $\pm\vec{d}\,'_i$ is determined. The concept is displayed in Fig. 8.5 and explained in more detail in [206].

Robustness to occlusion and illumination The choice of the similarity measure makes the shape model matching approach inherently robust to occlusions and illumination

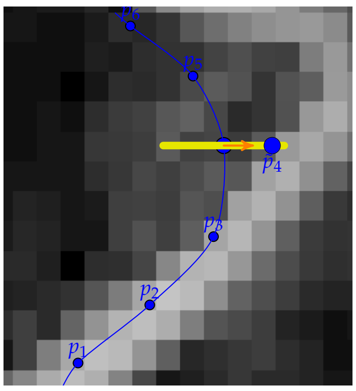


Figure 8.5: After the optimal model position, scale, and rotation have been determined, each point searches for its best match along a 1d search line perpendicular to its image tangent. The length of the search line is variable, but has a significant impact on the runtime.

changes. The robustness to non-linear illumination changes comes from the fact that all direction vectors are scaled to unit-length. The robustness to occlusion comes from the fact that missing points in the target image will, on average, contribute nothing to the sum of (8.1). Similarly, clutter lines or points in the target image not only need to coincide with the sparse set of model points, but also need to have similar direction vectors to contribute to the similarity.

The parameter $score_{\min}$ gives a good estimation of the allowed object occlusion. If half of the model points are occluded in the target image, the maximum score that can be obtained is 0.5. Please note that a low value of $score_{\min}$ increases the number of points for which the score needs to be calculated and increases the number promising locations that need to be tracked through the pyramid. Hence it may have a negative impact on the runtime.

Further speed-ups can be obtained by only using a subset of points for detecting the object. During the model generation step, a random subset of points may be selected from the model $\mathcal{M}$ and used for detection. Although some accuracy is lost, the execution time can be reduced.

8.2 Fast and Accurate Shape Model Tracking

In the following, the baseline shape model tracker proposed in Böttger *et al.* [27] is presented in detail. The approach can be divided into three stages: Model initialization, model localization, and model update.

Model Initialization In the first frame, a shape-based model $\mathcal{M}$ is generated from a given, arbitrarily shaped, region of interest (ROI). The generation of the shape model is analogous to the generation in the object recognition approach and is described in detail in Section 8.1. The shape model $\mathcal{M}$ consists of a set of n points $m_i = (p_i, \vec{d}_i) \in \mathbb{R}^2 \times S^1$

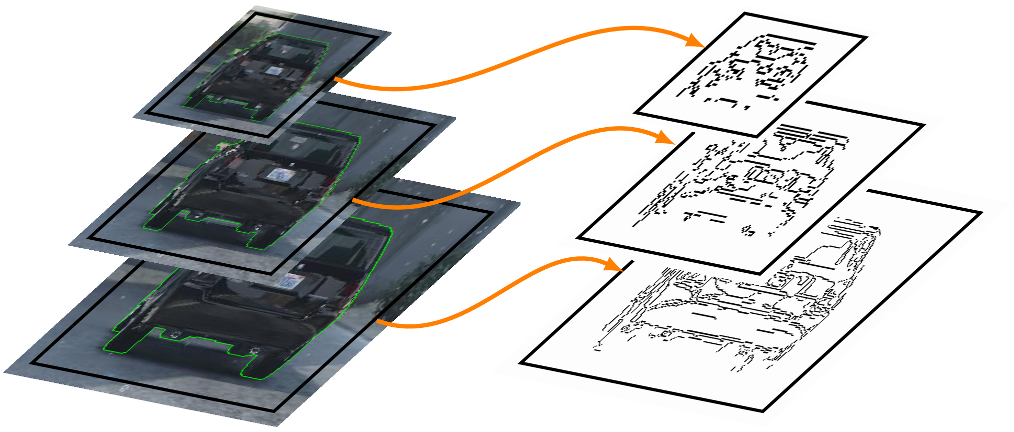


Figure 8.6: The default shape model pyramid generation is displayed. An image pyramid is generated from the input image. For each pyramid level an independent shape model is generated. In general, the number of model points descreases significantly for the coarser pyramid levels.

that each consists of a point p_i and the corresponding direction vector $\vec{d}_i$. The resulting points of an exemplary shape model are displayed in Fig. 8.1.

In general, the number of model points grows with size of the input ROI. Since many of the later computations depend on the number of model points, it is reasonable to reduce the number of model points as much as possible. As in the object recognition technique of Steger [205, 206], a shape model pyramid is generated from the input object. In a first step, an image pyramid is constructed from the input image. The number of pyramid levels is computed automatically. It ensures that enough model points are still present on the coarsest pyramid level. A shape model is constructed for each pyramid level independently of the other levels. As shown in Fig. 8.6, the number of model points decreases significantly for the coarser pyramid levels. In the baseline approach, the number of pyramid levels remains constant throughout the tracking process. This may be problematic when the object becomes very small or very large. If the object becomes very small, the detail in the finest model does not match the image anymore and if the object turns very large, the lowest pyramid level still has too much detail to enable an efficient tracking.

Tracking Step In a first step, a search region $\mathcal{S}$ is constructed. The search region is placed around the last known object position and scaled with the confidence of the tracker. In the baseline approach, the confidence is a function of the prior localization score and the number of successful localizations in the last n time steps [27]. More specifically, the search region is circular and the radius r is manipulated by the confidence $\alpha \in [0, 1]$ such that

$$r = (1 + r_{\max}(1 - \alpha)) \cdot \sigma_c, \tag{8.4}$$

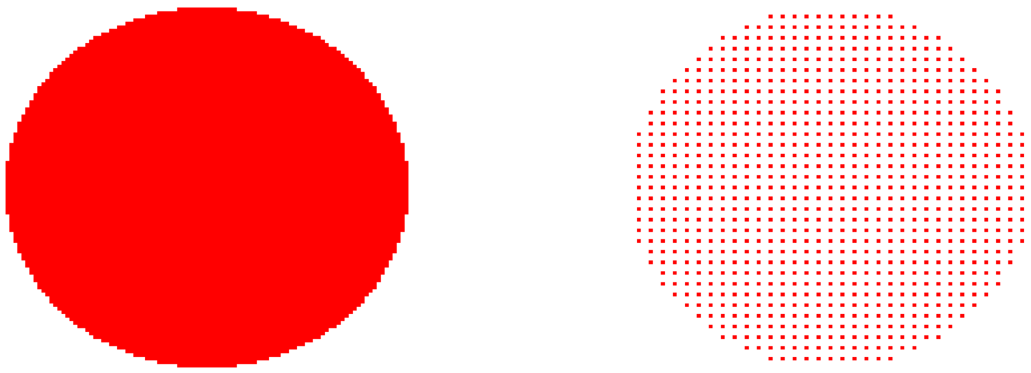


Figure 8.7: If the search region becomes too large, it is reduced to a search grid to ensure the runtime does not surpass the frame rate.

where $r_{\max}$ is the maximal radius factor and σ_c is the current object scale. Although $\sigma_c = s_x = s_y$ in the initial frames, as shown later, the search scales and the current model scale may change throughout the tracking process. The confidence is initialized at 1.0 and is incrementally decreased when tracking fails by dividing it by 1.05. When the object is re-detected, the confidence is reset to 1.0. If the confidence drops below a specified threshold $\alpha_{\min}$, the search region is thinned out even further. For this, the score is only computed for every n-th point in the search region. This creates a search grid rather than a dense search region as is shown in Fig. 8.7. Although this greatly reduces the localization accuracy, it ensures the runtime remains below frame rate. Once a rough location of the object has been obtained, the confidence is increased and the localization accuracy can be improved in subsequent frames.

Given the search region, the shape model is used to perform object recognition, as described in [205, 206] and Section 8.1. To reduce the computational complexity, the number of model points is further reduced. For this, a random subset of the model points is selected from the shape model $\mathcal{M}$. The points that were successfully tracked in prior frames are given a higher probability of being chosen. This ensures that strong model points are always present within the model. It has the further advantage that background points or model points resulting from noise are gradually removed from the model used for tracking.

Furthermore, given the last object pose, it is not necessary to search for all possible object poses exhaustively in each frame. To reduce the workload, we restrict the set of possible transformation parameters $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$ (see (2.10)) such that

$$\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma} = \left(\mathcal{S} \times \left[\theta_c - \frac{0.1}{\alpha}, \theta_c + \frac{0.1}{\alpha}\right] \times \left[\sigma_c - \frac{0.2}{\alpha}, \sigma_c + \frac{0.2}{\alpha}\right]\right), \tag{8.5}$$

where θ_c and σ_c refer to the current object rotation and scale, respectively. As for the search region, the range of possible rotations and scales grows with an increasing

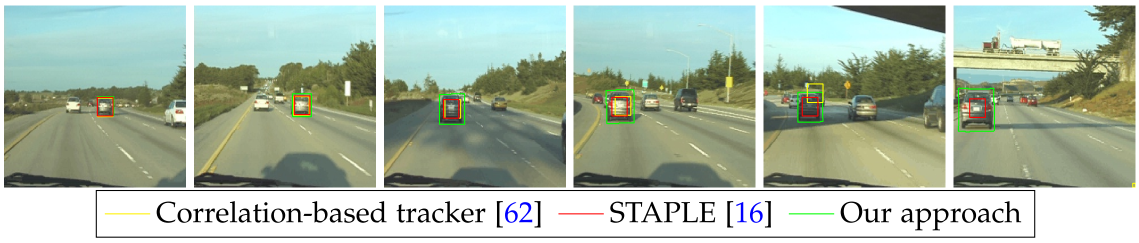


Figure 8.8: `Car24` from the OTB-2015 [239] benchmark. The baseline shape model tracker is compared to two equally fast trackers: STAPLE [16] and a scale adaptive correlation tracker [62]. The shape model tracker is virtually drift-free in the sequence (which has over 3000 frames). All three trackers run in real-time at 100 fps for this sequence.

uncertainty. For very low confidence values α, the ranges of θ_c, σ_c are clipped to realistic values, i.e., θ_c is restricted to a full rotation and σ_c to be within $[0.1, 10]$.

To achieve further speed-ups, no score is calculated for a model transformation as soon as it cannot reach a predefined minimal score $score_{\min}$ anymore. It is possible to be even stricter; please refer to [219] for further details.

Model Update The localization of the shape model $\mathcal{M}$ is computed with subpixel precision, as described in 8.1. The refinement involves finding a matching edge point in the search image for each model point p_i along its direction $\pm\vec{d}\,'_i$. In the update stage these correspondences may be used to refine the model points. For this, a final search for the corresponding target image points is conducted after the least-squares refinement. This time, the global similarity transformation of the model is not updated, but rather the relative positions of the model points p_i. This improves how well the model will fit to the target image at future time steps.

In the example shown in Fig. 8.5, the points p_4 is shifted towards the best match $\tilde{p}_4$ that is found along the yellow line. The model update is regularized with a parameter λ to be more robust to noisy object deformations. At frame t, each point is updated with its best match $\tilde{p}_i^t$ such that:

$$p_4^{t+1} = p_4^t + \lambda \tilde{p}_4^t. \tag{8.6}$$

If no matching point is found, the point is not updated. The update step does not only allow the approach to capture small model deformations, but also weakens the restriction of the approach to similarity transformations of the model. Consequently, projective transformations that increment over time may be captured by locally deforming the model points.

The update step of a tracking approach always needs to find the balance between keeping the localization accuracy high and generalizing well to new representations of the model. The same is true for the proposed update approach; too large parameter values of λ may add drift and can lead to a degeneration of the tracking model if no extra care is taken. However, since the model transformation is determined with subpixel-precision, the drift is minimal. Even long sequences with over 3000 frames, like the one in the

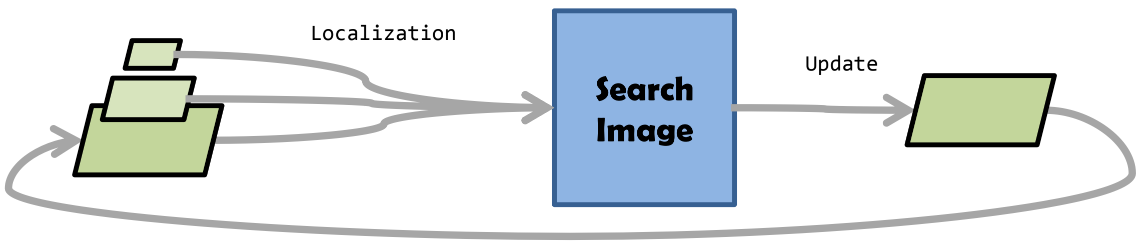


Figure 8.9: A simplified pipeline of the baseline shape model (`baseline_sm`) localization and update steps is visualized. The complete shape model pyramid is used for the object localization and only the finest level is updated after the localization.

example displayed in Fig. 8.8., do not drift significantly.

During the tracking process, it is monitored whether a matching edge point was found for every model point or not. This step enables the algorithm to identify points that are not contributing to the model localization. These points may be removed from the shape model completely. Generally, these points have either emerged from poorly initialized points in the first frame or by parts of the object becoming occluded or changing appearance. To prevent deleting all points and degenerating the model, the baseline method samples new points in sparse areas of the model. This allows to capture newly emerging object edges. A very simple pipeline of the baseline shape model localization and update steps is visualized in Fig. 8.9. Although each of the pyramid levels is used for the object localization, the update of the model is only performed on the finest pyramid level.

8.2.1 Strengths and Weaknesses

The baseline shape model tracking can be used to track roughly rigid objects in real-time. In the original paper, extensive experiments on the rigid objects within the OTB-2015 [239] and VOT2016 [116] datasets are conducted. It is shown that the tracker is able to compete with the state-of-the-art real-time trackers. In contrast to the compared trackers, it has a failure mode detection and a very high localization accuracy. Especially the high accuracy protects the tracker from drifting from the target. However, there are certain settings where the tracker has difficulties:

1. The number of pyramid levels is determined in the initialization step and kept constant throughout the object tracking. This has two drawbacks. On the one hand, if the object becomes significantly smaller in a sequence, the details on the finest pyramid level do not match the object anymore and the tracker may fail. On the other hand, if the object becomes larger, it may be necessary to increase the number of pyramid levels to remain efficient.

2. In long sequences, the tracked object may change considerably. The scale, rotation, and perspective may change. However, the shape model is only updated on the lowest pyramid level. Therefore, the model does not capture the changes of the

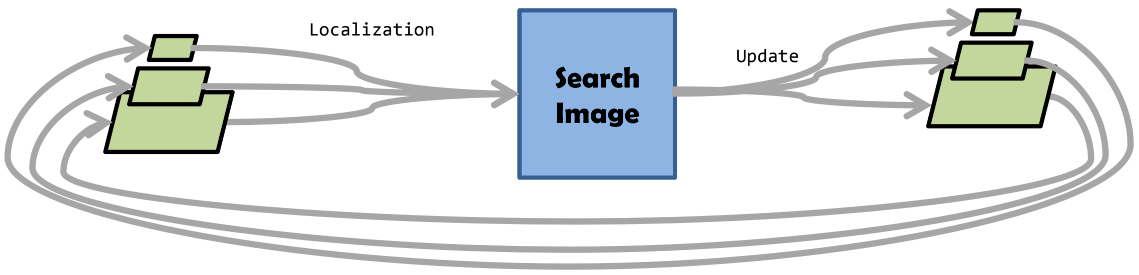


Figure 8.10: A simplified pipeline of the improved shape model (`all_level_sm`) localization and update steps is visualized. The shape model pyramid is updated for each level after the subpixel-precise object localization. This requires finding the best fit for each model point on each pyramid level.

object in the lower pyramid levels. As a consequence, the tracker may fail in longer sequences or sequences where the object changes scale or appearance considerably.

3. Although the model update can capture arbitrary deformations, they need to increment slowly over time. The tracker cannot cope with strong perspective view-point changes that appear quickly.

In the following, we address the above three problems and improve the shape model tracker.

8.3 Improved Shape Model Tracking

In this section, we address how the above described short comings of the baseline shape model tracking can be circumvented. We introduce improvements that turn the shape model tracking into an even more reliable long-term tracker that is fast, robust, and very accurate. To distinguish the different improvements in the experiments, we denote each improvement with a unique identifier,. For example, the above described baseline method is denote as `baseline_sm`.

8.3.1 Model Update: Refinement of All Levels

The baseline method only updates the model points on the finest level. This has the very practical reason that the least-squares refinement is only required on the finest pyramid level for a subpixel-precise localization of the object. Hence, the model update has a small computational overhead. Furthermore, for small model deformations, this is a reasonable assumption: the reduced resolution in the coarser pyramid levels makes them inherently robust to small deformations of the model. Unfortunately, for larger deformations that accumulate over time, this restriction can prevent the model from capturing more complex model deformations. In the following, we present an approach that also updates the models in the coarser pyramid levels. The method is denoted as `all_level_sm`.

In the baseline shape model tracking, the transformation parameters are refined to subpixel precision in the final step. The refined transformation is the starting point of the model update on the finest pyramid level. In the proposed approach, the respective refined transformation is applied to the model in every pyramid level. Therefore, the translation parameters need to be converted to the size of the respective pyramid level. The transformation parameters of the scale and rotation are the same for each pyramid level. After the models have been transformed, the update stage of the subpixel refinement is applied in each pyramid level individually. Hence, for each model point p_i, a matching edge is searched for along the point's directions $\pm\vec{d}'_i$. As for the finest pyramid level, the points are moved towards the found edge matches according to (8.6). The concept is visualized in Fig. 8.10.

The described approach allows to update the models on every pyramid levels. This is essential for long time tracking. However, this straight forward adaptation of the update has a few shortcomings:

- The least-squares refinement is computationally demanding. Therefore, the proposed update of all pyramid levels independently adds a significant computational complexity to the model update.
- The number of pyramid levels is fixed and determined by the model initialized in the first frame. Although it is possible to use fewer levels for the localization, it is not possible to increase the number of pyramid levels.
- The object localization is very dependent on the quality of the model on the coarsest pyramid level. In the first step, the similarity of this model is computed for each possible position and transformation in $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$. Hence, if this model does not match the current appearance of the object, no possible locations and transformations are found that can be tracked through the image pyramid. Since the computational complexity of the full parameter search is considerable, it is prohibitive to extend the full parameter search to lower levels in the image pyramid if nothing is found on the coarsest level. Therefore, the quality of the model on the coarsest pyramid level is extremely important. To ensure the tracking can succeed, every update of a model point needs to be done with extreme care. Furthermore, there are usually only a few model points in the coarsest pyramid level. This makes the model very fragile to any update of the model points.

8.3.2 Dynamic Pyramid Generation

As above, the idea is to update the model in lower levels as well as in the finest level. However, to reduce the computational overhead, no point matches are computed for the coarser pyramid levels. Instead, the coarser pyramid models are generated directly from the finest pyramid level. While this removes the necessity to compute point matches for every point in each pyramid level, it introduces a number of new challenges in the scale space. The conceptual idea of the model update displayed in Fig. 8.11. The dynamic generation of the image pyramid is able to significantly reduce the runtime.

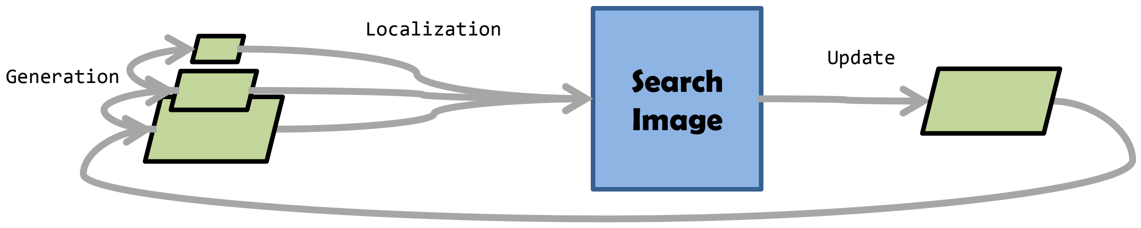


Figure 8.11: A simplified pipeline of the improved shape model (`gen_level_sm`) localization and update steps is visualized. Only the finest level is used for the object localization and only the finest level is updated after the localization. The coarser pyramid levels are generated from the lowest pyramid level in each frame.

If the search region turns very large, it may be prohibitive to test the complete search space of $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$ in the coarsest pyramid level. However, if the number of levels can be adapted dynamically, a further pyramid level can be generated and the cost of finding promising object locations can be reduced. Furthermore, if the size of the model becomes very small, the coarsest pyramid level may be too inaccurate. In these settings, fewer pyramid levels can be generated. We compare and evaluate three different ideas to dynamically generate the pyramid of shape models. Each of the three methods ensure the automatically generated models fit to the subpixel-precise edges in the image pyramid slightly differently. They are denoted as `gen_level_sm`, `gen_level_sm_regression`, and `gen_level_sm_precise`, respectively.

The first stages of all three approaches are the same. In the initialization stage, the shape model is only generated for the finest pyramid level. The shape models on the coarser pyramid levels are then generated dynamically in the localization stage. For this, the point coordinates from the first pyramid level $p_i^{(1)}$ are scaled such that

$$p_i^{(n)} = \frac{p_i^{(1)}}{2^{n-1}}, \tag{8.7}$$

and the directions $\vec{d}_i^{(n)} = \vec{d}_i^{(1)}$ are inherited from the first scale. This essentially allows to generate an arbitrary number of scales from the finest image pyramid level.

As observed before, in the ordinary shape model, the number of model points generally decreases rapidly in the coarser levels. This represents a further challenge when automatically creating the pyramid levels. To account for this, in a second step, all three approaches merge the transformed model points in a greedy manner;

1. For each model point p_i , the distance to its surrounding points q_i is computed. Points that are too far away are given a fixed distance $\delta_{\max}$. This generates a sparse and symmetrical distance matrix $D_{p,q}$ that encodes the pair-wise distances.

2. For each column d_i in $D_{p,q}$ the minimal distance $d_i^{\min}$ is computed.

3. Starting from the column with smallest minimal distance ($d^{\min} = \min_i d_i^{\min}$), all points with a distance smaller than a given threshold $\delta_{\min}$ are merged. The respec-

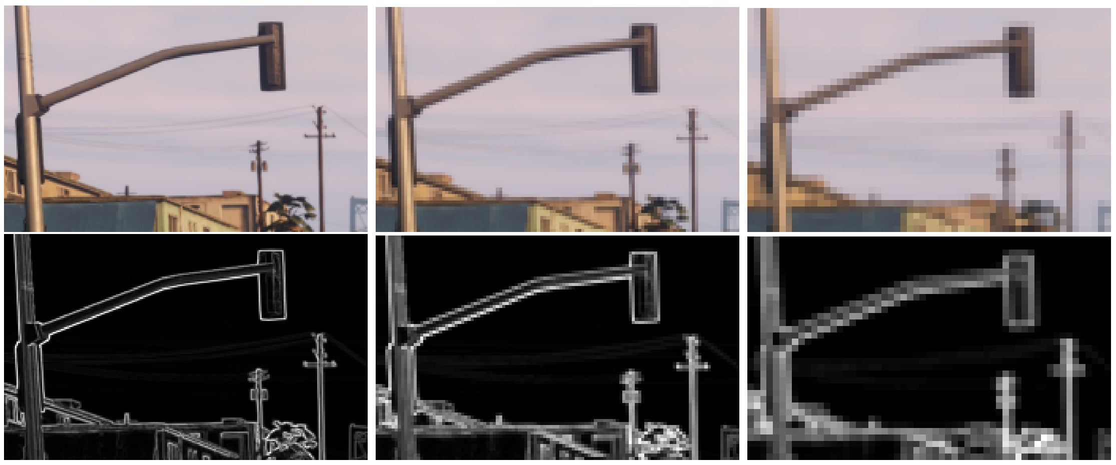


Figure 8.12: The images and the corresponding edge amplitudes of three levels in an image pyramid are displayed. From left to right, fine-scale structures are suppressed and edges may merge or disappear in the coarser pyramid levels.

Table 8.1: The coarser pyramid levels have a lower minimal score than the higher pyramid levels. This is to account for the increased uncertainty in the automatically generated pyramid levels

	$level = 2$	$level = 3$	$level = 4$	$level = 5$
$score_{\min}$ factor:	0.7	0.5	0.34	0.24

tive rows and columns are removed from the matrix.

4. The process is repeated for the all columns in $D_{p,q}$ in the order of their minimal distance $d_i^{\min}$.

The above steps ensure that the closest points are merged first. Overall, the thinning of the model points prevents having too many redundant points within the models of each level. This may significantly speed up the object localization. However, the described approach does not account for edges that would disappear completely in scale space.

The dynamically generated shape models do not coincide with the shape models that would be generated from the image pyramid directly. To ensure that the localization process is still able to find matches, the minimal score $score_{\min}$ is decreased for each level of the pyramid. More specifically, the further away a level of the image pyramid is from the initial first level, the more $score_{\min}$ is decreased. The respective factor for each level is displayed in Table 8.1.

We denote the method that performs the above described three steps of scaling the model points, thinning out the points, and a reducing the $score_{\min}$ as `gen_level_sm`. The general idea is visualized in Fig. 8.13. The other two methods that dynamically generate the image pyramid build on top of `gen_level_sm`. Merely scaling the model points to the coarser pyramid level does not ensure that the generated points lie on subpixel edges of the respective pyramid level. Hence, the other two methods further refine the position of the model points within the pyramid to make the model more robust.

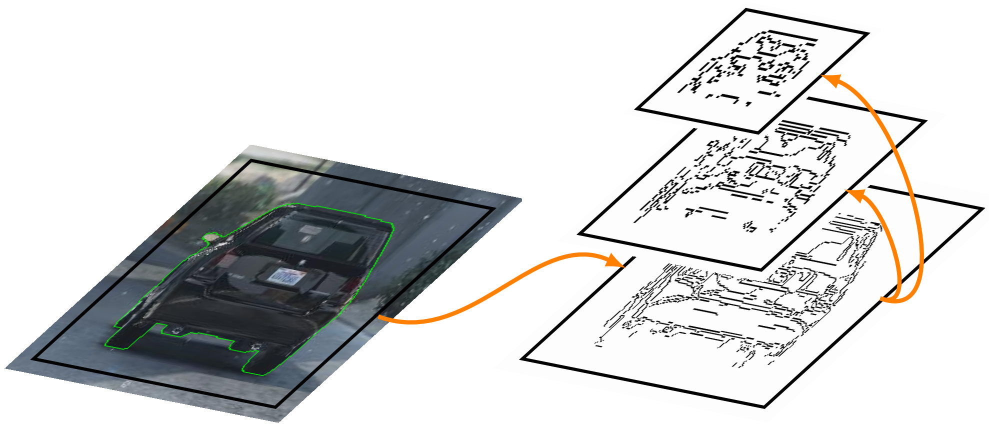


Figure 8.13: The dynamic shape model pyramid generation is displayed. To be able to dynamically adapt the number of levels, only the first model level is generated from the image template. The further levels are generated by propagating the points to a smaller scale. Closely neighboring points are merged and the points are refined based on the last gradient image of the tracked object.

In the image pyramid, edges may merge, disappear, or change their position significantly from the the finest to the coarsest level [135, 203, 244]. This is visualized in Fig. 8.12, where the edges of the traffic light and the power pole represent the object boundaries in the finest pyramid level and the center of the poles in the coarser levels. Hence, the boundaries merge to form the center line. Furthermore, the details of the tree blur and many edges disappear completely. When generating the model points dynamically, this is a problem. In the baseline, this problem is circumvented by generating the shape models on each pyramid level independently. This has the advantage that the scale-space effects are inherently accounted for. Nevertheless, as stated above, the independence of the models makes it difficult to update the models in dependency of each other. The automatic generation of the coarser pyramid levels enables us to integrate the deformations from the finest pyramid level to the coarser levels implicitly. The dynamic model generation of `gen_level_sm` does not ensure that the transformed model points lie on subpixel-precise image edges in the image pyramid. Hence, in a next step, we evaluate two methods that fit the model points to the edges of the respective pyramid level: contour point snapping and a learned point regression.

Contour Point Snapping The general idea is to move each model point to the closest subpixel-precise image edge. Hence, in a first step, the subpixel-precise image edges are determined for each level of the image pyramid. This requires applying an edge filter such as a Sobel filter, a hysteresis thresholding of the filter response, and a non-maximum suppression or skeletonization of the obtained edge region. Furthermore, the obtained pixel precise edges need to be refined to subpixel precision. In total, the process is computationally significant. After the subpixel-precise edges have been determined, the closest subpixel-precise edge is determined for each shape model point. If the distance is below two pixels, the shape model point is moved towards the image edge. However, if the maximal distance is too large, the model point is discarded. It is assumed that in

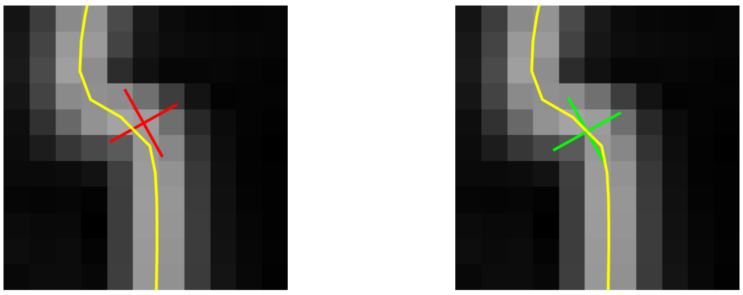


Figure 8.14: Scaling the model points to the coarser levels in the dynamic shape model pyramid generation may lead to model points (left image) that do not lie directly on the subpixel-precise image edges (yellow contour). In the right image, the respective model point that would be generated by the model initialization is displayed.

these circumstances the subpixel-precise edge is not present in the respective pyramid level anymore. The described method is denoted by `gen_level_sm_precise` and displayed in Fig. 8.14. Here, the scaled shape model point in the left image is moved to the closest subpixel-precise image edge.

Model Point Regression The above described contour point snapping of the model points is computationally demanding. However, it is reasonable to assume that the explicit position of the subpixel-precise image edges may not be necessary to optimize the positions of the shape model points. Instead, we evaluate whether the gradient amplitude in the neighborhood of every scaled model point has sufficient information to improve the position of the shape model points. Consider the scaled model point in the left image of Fig. 8.14. The gradient amplitude in the neighborhood of the scaled model point (red) indicates that the model point is slightly offset from the subpixel-precise edge. To verify the above assumption, a classifier that obtains the scaled shape model point coordinate p_i, the point's direction $\vec{d}_i$ and the surrounding gradient amplitude values as input is trained. The classifier returns an update of the shape model point coordinate and direction. The general concept is displayed in Fig. 8.15.

To train the classifier, we need a suitable representation of the shape model points. In the shape model, the model point coordinates p_i are determined by their relative coordinate to the reference point of the shape model. However, the update of the model points should be independent of their position in the shape model. Hence, as input, the classifier receives the row and column position of p_i in the coordinate system that is centered at the closest image pixel $\overline{p_i}$ to the model point, i.e., for $p_i = (245.6, 106.24)^T$ the local coordinates is centered at $\overline{p_i} = (246, 106)^T$ and thus $\hat{p}_i = (-0.4, 0.24)^T$. Furthermore, the classifier receives the 5×5 values of the gradient amplitudes surrounding the point $\overline{p}$ and the respective direction $\vec{d}_i$. As output, the network outputs a row and column offset o_i that shifts the point p_i towards the closest subpixel-precise edge and an update of the direction $\vec{o}_i$.

We use a Multilayer Perceptron (MLP) with a single hidden layer. Since we are considering a regression problem, a linear output function is used (in contrast to the commonly used softmax for classification problems). We tested a different number

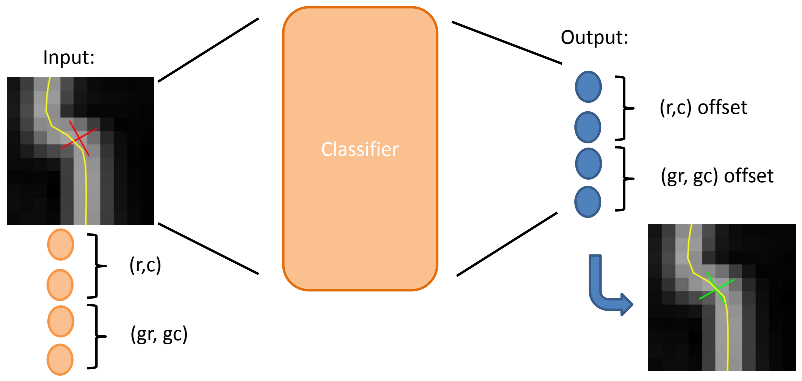


Figure 8.15: The pipeline of the learning-based model point regression is displayed. The classifier receives the gradients in the 5×5 neighborhood of each point, the point coordinates, and the point direction. The output is 4-dimensional and includes an update to the point position and direction.

of neurons in the hidden layer and found that 15 showed a good balance between performance and runtime. To train the MLP, it is possible to obtain a large amount of training data from arbitrary input images. Given an image, a shape model with two pyramid levels is generated. Then, the shape model points in the coarser pyramid level are scaled and placed in the finer pyramid level. For each scaled shape model point, the closest model point that was generated by the shape model initialization can be determined. This coincides with finding the closest model point in terms of the l_2 distance. The concept is visualized in the first row of Fig. 8.17. The respective offset between both points is used as the expected output o_i of the classifier. Furthermore, the offset in the directions of both points can be calculated and set as the offset of the point's direction $\vec{o}_i$. A variety of different images is used to train the classifier. A collection of the training and testing images is displayed in Fig. 8.16. They cover a number of different domains and applications.

Although the shape model points that are created by the initialization process lie on subpixel-precise image edges, they are merely a sparse approximation of the edge itself. Therefore, their precise location on the image edge is somewhat random. We cannot expect the classifier to learn the sampling of the subpixel-precise image edge from a small window around a model point. The problem is visualized in the first row of Fig. 8.17. Although the l_2 distance of p' and p is being minimized, the position of p could be anywhere on the blue line. All that the classifier can be expected to learn is the offset of each shape model point to the closest subpixel-precise edge. Hence, in a next step, we would like the classifier to minimize the distance of each model point to the closest contour point directly. The contour distance is denoted by l_{cont} and visualized in the middle row of Fig. 8.17. The approach is trying to achieve the same as the contour point snapping in `gen_level_sm_precise` but without explicitly computing the subpixel-precise edges and determining the closest image edge for each point.

However, the problem can be simplified even further. Instead of calculating the closest

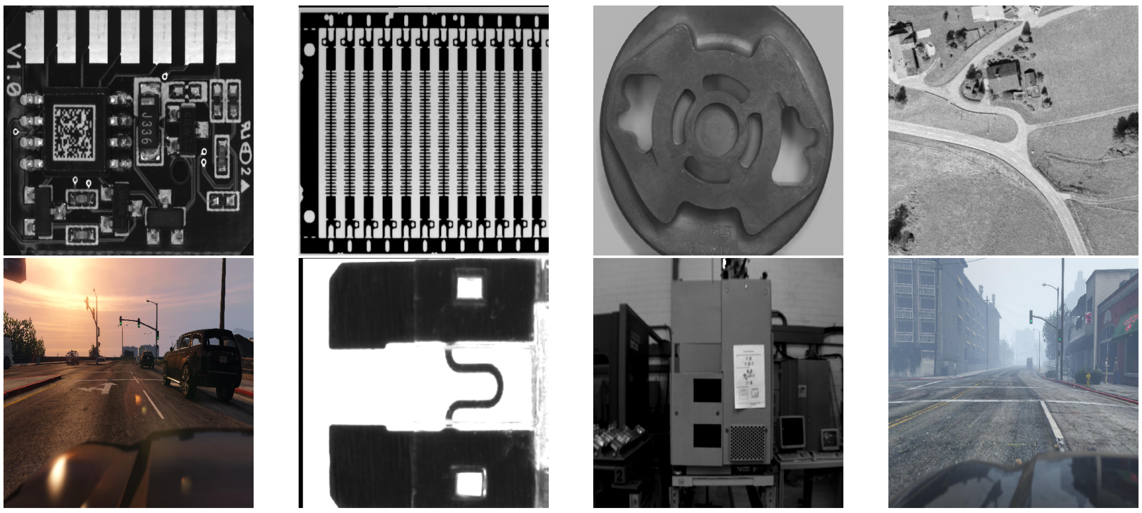


Figure 8.16: The images used for training and testing the MLP classifier are displayed. The images were selected to cover a range of different domains and applications.

Table 8.2: The results of the three point regression classifiers and of contour point snapping are displayed. The displayed error is the remaining average l_{cont} distance of model points after and before the regression. Therefore, the contour point snapping has $l_{\text{cont}} = 0$. The runtime of the classifiers is measured explicitly for each point, whereas the runtime of the contour point snapping is averaged over many sequences and points.

	train error l_{cont}	test error l_{cont}	runtime per point ms
optimize l_2	$0.443 \rightarrow 0.409$	$0.426 \rightarrow 0.391$	0.003 ms
optimize l_{cont}	$0.443 \rightarrow 0.359$	$0.426 \rightarrow 0.341$	0.003 ms
optimize $l_{\vec{d}}$	$0.443 \rightarrow 0.441$	$0.426 \rightarrow 0.458$	0.002 ms
contour snapping	$0.443 \rightarrow 0.0$	$0.426 \rightarrow 0.0$	0.43 ms

subpixel-precise edge l_{cont}, we can search for the closest subpixel-precise edge along the direction of each scaled shape model point. This should allow to simplify the problem even further. The distance is denoted as $l_{\vec{d}}$ and visualized in the third row of Fig. 8.17.

To verify which of the above measures performs the best, we trained three different classifiers that differ in the input and output they obtain. The first two classifiers get the image gradients in a 5×5 neighborhood of the scaled model point, the relative point coordinate $\hat{p}$, and the direction. However, the first classifier is expected to output the offsets based on the l_2 distance and the second classifier the offsets based on the l_{cont} distance. The third classifier merely outputs a single value. Namely, the direction and distance to walk along the point's direction to find the closest subpixel-precise edge. The direction is encoded in the sign of the output. The results are displayed in Table 8.2 and discussed in the following paragraph.

Comparison of Contour Point Snapping and Point Regression We compare the results of the three proposed point regression classifiers and the contour point snapping. As shown in Table 8.2, all but the $l_{\vec{d}}$ classifier are able to improve the position of the

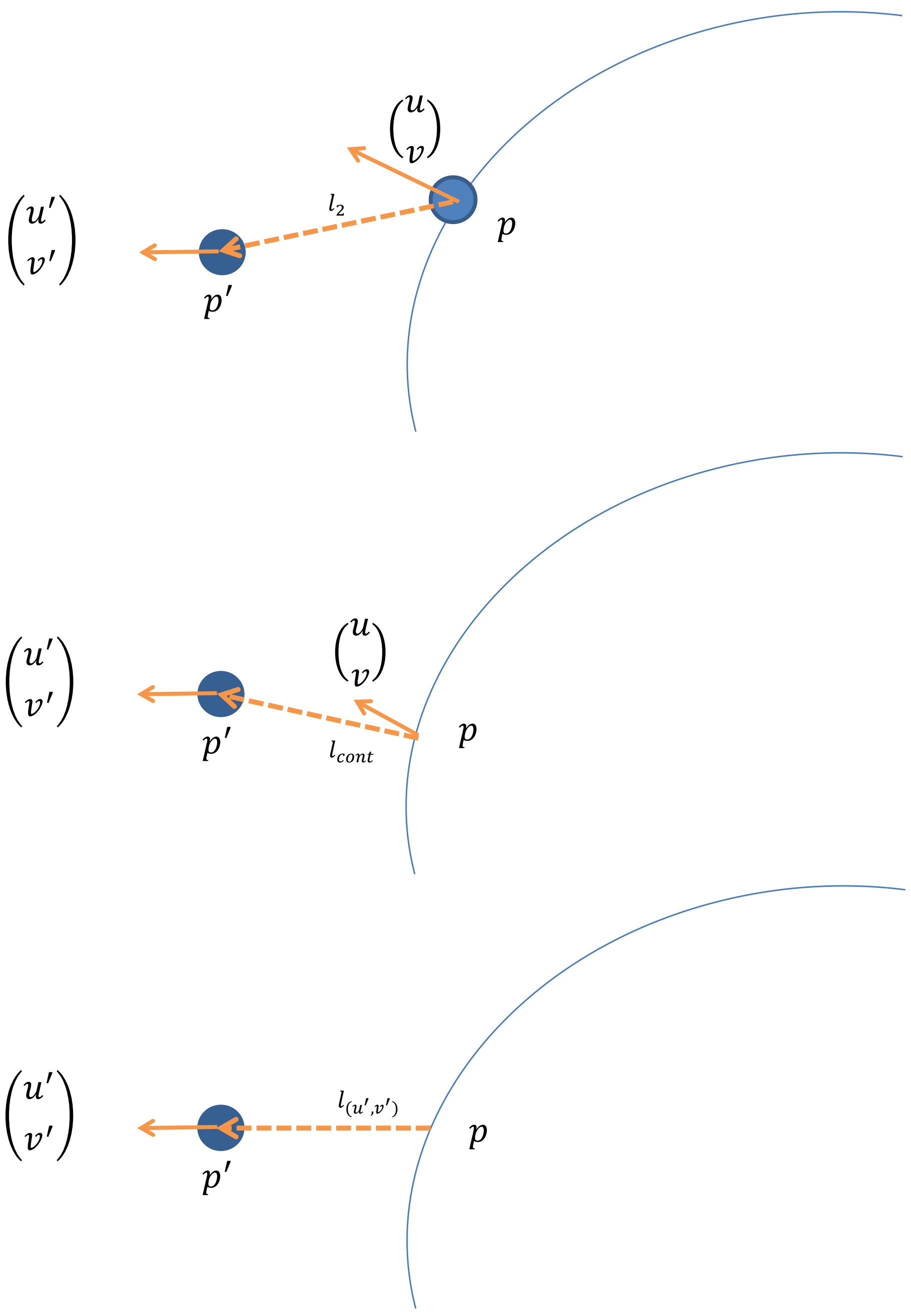

$\begin{pmatrix} u \\ v \end{pmatrix}$
$\begin{pmatrix} u' \\ v' \end{pmatrix}$
l_2
p
p'
$\begin{pmatrix} u \\ v \end{pmatrix}$
$\begin{pmatrix} u' \\ v' \end{pmatrix}$
p
p'
l_{cont}
$\begin{pmatrix} u' \\ v' \end{pmatrix}$
$l_{(u',v')}$
p
p'

Figure 8.17: The distance measure l_2, l_{cont} and $l_{\vec{d}}$ are displayed. The l_2 distance is the distance between the scaled shape model point p' and the closest shape model point p. The l_{cont} distance is the distance between the scaled shape model point p' and the closest subpixel-precise edge (blue). The $l_{\vec{d}}$ distance is also the closest subpixel-precise edge, but measured along the direction of the scaled shape model point

shape mode points of the test images. While the contour point snapping moves every point directly on top of a subpixel-precise edge, the classifier-based approaches are only able to improve the point positions. However, while the classifiers l_{cont} and l_2 come at a negligible computational cost, the contour point snapping costs almost half a millisecond per model point. Since the shape models typically have hundreds of model points, this is computationally demanding. As expected, the classifier trained directly on l_{cont} is able to outperform the one trained on l_2. The sampling of the subpixel-precise image edge makes the learning problem unnecessarily difficult. Both approaches have the same computational overhead. The classifier trained with the $l_{\vec{d}}$ distance is not able to improve the shape model point positions in the test images. A small movement of a model points direction $\vec{d}$ can lead to a significant change of the distance $l_{\vec{d}}$. As a consequence, the point updates are ill conditioned and not very stable.

Unfortunately, none of the proposed classifiers was able to improve the direction vectors of the shape model points. Since the gradient amplitude was used as input, it could be assumed that using the partial derivatives directly should improve the results. However, although this almost doubles the input to the classifiers, the results did not change considerably. As a consequence, in the later experiments, we merely input the point and the neighboring gradient amplitude and expect the classifier to output an offset to the shape model point only.

Qualitative results for the contour point snapping and the classifier trained with l_{cont} are displayed and discussed in Chapter 9.

8.3.3 Robust Tracking

The baseline shape model struggles in long sequences were the tracked object changes scale and appearance considerably. This has two main reasons: (1) when the scale changes too strongly, the model on the finest image pyramid does not fit the actual object anymore and (2) in long sequences, the incremental update of the model may change the shape model too much and prevent a detection of the original object. In the following, these two problems are tackled by introducing a hierarchy of shape models. The hierarchy stores shape models generated at different scales and from different perspectives. The shape model localization can then use multiple models to identify the most probable object locations. This enables the algorithm to make the tracking significantly more robust in long sequences. In a first step, we introduce how models of different scales are acquired and then proceed to introduce the collection of models from different perspectives.

Shape Model Hierarchy of Scales As shown in Figs. 8.6 and 8.13, the amount of detail in the shape model decreases considerably from the finest to the coarsest model of the pyramid. Hence, if the object that is being tracked increases its size, the amount of detail in the finest pyramid level does not match the actual object anymore. This may lead to a significant drop of the score returned by the tracker and eventually the tracker may lose the object or identify a different object as the target. To prevent this from happening, we introduce a more robust shape model that initializes a new shape model whenever the

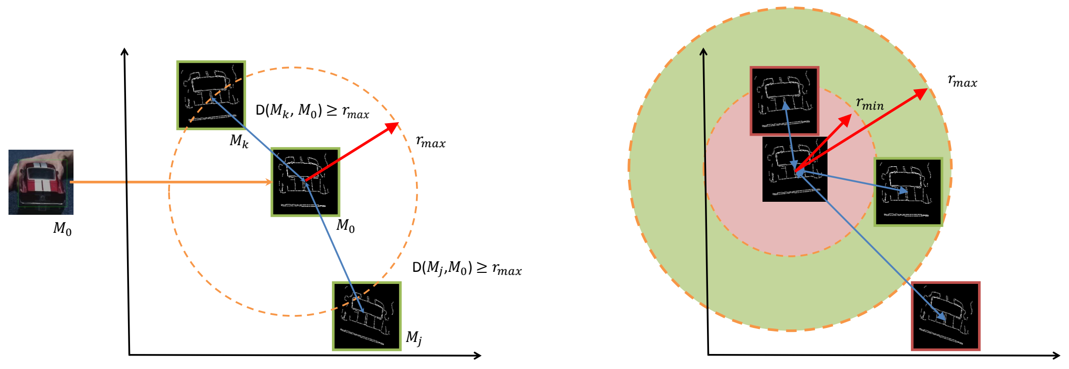


Figure 8.18: To further reduce drift, multiple models may be tracked simultaneously. On the left, the generation of the shape model hierarchy is displayed. Whenever the current model exceeds the distance of $r_{\max}$ from the existing shape models, a new tracking model is generated. On the right, the shape models that are used for tracking are displayed. Only the shape models in the hierarchy that have a distance larger than $r_{\min}$ and smaller than $r_{\max}$ are used for locating the object.

object changes its scale considerably. Since the levels of the shape model pyramid can be generated dynamically, the shape model is only generated for the finest pyramid level. Furthermore, the image directions are calculated in the object localization step anyway. As a consequence, the computational complexity of the shape model initialization can be reduced considerably.

When a new shape model is generated, the old shape model is not discarded. Instead, both shape models are kept in a hierarchy of shape models and the initialization scale of the models is stored. This enables the algorithm to search for an object with multiple shape models. In a long sequence, it is unnecessary to search for all models within the hierarchy. Instead, only the models that where initialized at a scale that is similar to the current scale of the model are used. In our experiments, a new model is generated whenever the object scale is greater than 1.6 or smaller than $1.0/1.6 = 0.625$. The initial shape model is initialized at scale 1.0. Then, all models that are within the range of half the current object scale and twice the current object scale are used for tracking. In theory, searching for multiple models introduces a linear factor in the runtime of the object localization. However, a significant part of the computational complexity comes from preparing the data structures and calculating the image directions. Hence, in practice, the runtime of the object localization is not affected too badly. Extensive experiments are shown in Section 9.2.

Shape Model Hierarchy of Perspective In long sequences, the scale change of the object is not the only concern. Also the perspective of the object may change throughout the image sequence. The shape model update accounts for this by incrementally updating the location of the model points, as described in (8.6). Although the update of the model points allows to capture arbitrary deformations of the object, the update is not perfect. Hence, if the object returns to its initial appearance, the shape model used for tracking may have diverged from its original shape. To tackle this, the original version of the shape model is stored within the shape model hierarchy. Whenever the model diverges too strongly from the original template, a new shape model is stored in the hierarchy. All models that are close to the current model state are used for locating the most probably object location. This requires a notion of how *close* two shape models are. In the current context (perspective variation of the model), all models have the same number of points. Hence, the summed l_2 distance of all model points is used to measure the similarity of two shape models. The distance between two models $\mathcal{M}_j$ and $\mathcal{M}_k$ is defined as

$$D(\mathcal{M}_j, \mathcal{M}_k) = \sum_i^n \left\| \begin{pmatrix} x_i^j \\ y_i^j \end{pmatrix} - \begin{pmatrix} x_i^k \\ y_i^k \end{pmatrix} \right\|_2 , \tag{8.8}$$

where $(x_i^j, y_i^j)^T$ refers to the coordinates of the model points m_j of $\mathcal{M}_j$, and $(x_i^k, y_i^k)^T$ to those of $\mathcal{M}_k$, respectively. Whenever the current shape model has a distance $D(\cdot, \cdot)$ greater than a predefined threshold $r_{\max}$ to all of the models within the shape model hierarchy, the current shape model is stored to the hierarchy. The idea has some resemblance to keyframe-based tracking methods [71]. The basic concept is visualized in the left image of Fig. 8.18.

However, to allow the shape model to develop and diverge from the original model, not all of the shape models in the hierarchy are used for tracking. Instead, only the shape models that have a distance from all models that is smaller than $r_{\max}$ and larger than $r_{\min}$ to the current shape model are used. This enables the algorithm to generate new shape models and create a meaningful shape model hierarchy. The concept is displayed in the right image of Fig. 8.18. Here, only the current model and the one within the green area are used for tracking.

8.3.4 Motion Estimation

The objective of the motion model is to estimate the state trajectory of the tracking target and to predict the object location in the subsequent frames. Although reliable methods date back as far as the 1960s [108], few modern object trackers use a motion model [117]. This is primarily related to the fact that the focus of current benchmarks is on short-term tracking, where the performance gain of a motion model is restricted [116, 117, 239]. Nevertheless, in long-term tracking, the use of a motion model to improve the localization and the estimation of the object position is indispensable. We present a universal approach to improve a short-term tracker performance with a motion model.

In the presented shape model tracking, the motion model is not only responsible for improving the initial estimate of the object position. Furthermore, it is responsible for estimating the most probable object location when localization fails. Moreover, the difference of the location prediction of the motion model and the tracker response can be used as an indicator for the system's confidence and used to generate optimal search regions.

Even though the tracker represents the object by an arbitrary region, the motion model assumes the target location is a single point in space. To simplify computations, the location is chosen as the center of gravity of the tracker region. The evolution of the respective location is described within the motion model and can be propagated based on prior observations. The propagation of the target location is essential to reduce the size of the search space and prevent mixing up the tracking target with other objects.

We assume that the target motion and its observations can be represented by the following *continuous* state-space model [129],

$$\dot{x}(t) = f\left(x(t), u(t), t\right) + w(t), \qquad x(t_0) = x_0 \tag{8.9}$$

$$z(t) = h\left(x(t), t\right) + v(t), \tag{8.10}$$

where $x(t), z(t)$ and $u(t)$ are the target state, observation, and control input functions, and $w(t)$ and $v(t)$ are the process and measurement noise, respectively. f and h are arbitrary time varying functions that model the target location and measurement evolution. Although the above equations may describe very complex state transitions, no matter what form f and h take, the state-space model is essentially memoryless, i.e., the future states do not depend on the past state history, but only on the current state. Such processes are often referred to as *Markov processes* (for continuous time) or *Markov chains* (for discrete time). In visual object tracking, the observations are only available for discrete time instances. Hence, it is reasonable to discretize the above equations for the measurements. For simplicity, we further assume $x_k = x(t_k)$, $v_k = v(t_k)$, $h_k(x_k) = h(x(t_k), t_k)$ and that the input is piecewise constant with $u_k = u(t), t_k \leq t < t_{k+1}$, and obtain the following *mixed-time* model:

$$x_{k+1} = f\left(x_k, u_k, t\right) + w_k \tag{8.11}$$

$$z_k = h_k(x_k) + v_k. \tag{8.12}$$

If we further assume that $f(x_k, u_k, t) = f_k(x_k, u_k)$, we can also discretize with respect to f and obtain the respective *discrete-time* model:

$$x_{k+1} = f_k\left(x_k, u_k\right) + w_k \tag{8.13}$$

$$z_k = h_k(x_k) + v_k. \tag{8.14}$$

Strictly speaking, the target motion should not depend on when samples are taken and is a value in continuous time [129]. Nevertheless, for slow motion speeds and the real-time capabilities of the shape model tracker, the discrete-time model is reasonable.

In the following, the target state x encodes the target location p, the target velocity v and acceleration a:

$$x_k = \begin{pmatrix} p_k \\ v_k \\ a_k \end{pmatrix}, \tag{8.15}$$

where p_k, v_k and $a_k \in \mathbb{R}^2$. A key feature in visual object tracking is that the measurement z_k will not fully determine the state x_k. Only the position p_k can be inferred, while the velocity and the acceleration cannot be measured directly. Hence, it is reasonable to assume h_k is linear in the above equations to obtain,

$$x_{k+1} = f(x_k, u_k, t) + w_k \tag{8.16}$$

$$z_k = H_k x_k + v_k, \tag{8.17}$$

where H_k is the measurement matrix. In visual object tracking, the measurement can only obtain reliable values for the state position. These measurements of p are mutually independent, but also independent of the other state variables. Moreover, they are also independent of the time step and thus $H_k = H \; \forall k$ and takes the form:

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{8.18}$$

In many tracking and maneuvering scenarios, the control input u_k are unknown, but essentially deterministic in nature. Hence, a straightforward way to model it is as an unknown, deterministic process and estimate the respective process from the measurements made during tracking. Due to the lack of any prior knowledge, this unknown process is often assumed to be piecewise constant and estimated over a fixed time window [41, 130]. The determination of the time window size is crucial and makes it difficult to use deterministic models. However, without any further information, visual object tracking has a hard time to infer which part of the observed motion comes from u_k, x_k or the process noise w_k. Since for the scenes we will be considering in the experiments the majority of all movement comes from the state transition encoded in f and the state vector of the tracked object, we assume the influence of u_k is negligible and omit it:

$$x_{k+1} = f_k(x_k) + w_k \tag{8.19}$$

$$z_k = H_k x_k + v_k. \tag{8.20}$$

The major challenge comes from the target motion uncertainty. Specifically, even though the general form of the above models is adequate, a tracker not only lacks knowledge about the actual control input u_k of the target but also of the actual form of f_k, its parameters, or statistical properties of the noise w_k for the particular target being tracked.

However, if we assume the system and the measurement errors to be uncorrelated and roughly zero-mean Gaussian distributed, a light-weight and recursive way to solve the respective equations is by the *Extended Kalman Filter*. In the following, we assume $w_k \sim \mathcal{N}(0, Q_k)$ and $v_k \sim \mathcal{N}(0, R_k)$.

Kalman Filtering To efficiently solve the state-space model, the Extended Kalman Filter linearizes (8.19) and (8.20) for each time step at the current estimates of the state mean and covariance. The problem is thus reduced to a locally linear state transition and the ordinary *Kalman Filter* can be applied. Hence, to highlight the general concept, it is helpful to first consider a linear propagation of the target state;

$$x_{k+1} = F_k x_k + w_k \tag{8.21}$$

$$z_k = H x_k + v_k, \tag{8.22}$$

where F_k is the state transition model which is applied to the previous state. In visual object tracking, it is reasonable to assume the state transition matrix is constant, hence $F_k = F \ \forall k$, and can be estimated by basic kinematics

$$F = \begin{pmatrix} 1 & 0 & T & 0 & \frac{T^2}{2} & 0 \\ 0 & 1 & 0 & T & 0 & \frac{T^2}{2} \\ 0 & 0 & 1 & 0 & T & 0 \\ 0 & 0 & 0 & 1 & 0 & T \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{8.23}$$

The Kalman filtering algorithm then works in a two-step process. In a first step, called the *prediction step*, the filter produces estimates of the current state vector variables, along with their uncertainties. The prediction is used to obtain an estimate of the tracking target location in the current frame. This can be used to create a compact but promising region to search for the object.

Prediction step (linear state transition)

Predicted state estimate : $$\hat{x}_{k|k-1} = F\hat{x}_{k-1|k-1} + w_k \tag{8.24}$$

Predicted covariance estimate : $$P_{k|k-1} = FP_{k-1|k-1}F^{\mathrm{T}} + Q_k, \tag{8.25}$$

where Q_k is the covariance of the process noise and $\hat{x}_{k|k-1}$ represents the estimate of the state at time k given the measurement at time $k-1$. Similarly, $P_{k|k-1}$ represents the state covariance at time k given the measurement at time $k-1$. The diagonal entries of P_k are the principal uncertainties of the single-target state variables, while the off-diagonal values represent the correlation between elements of the target state. The current state estimate $\hat{x}_{k-1|k-1}$ is propagated via the process matrix and the uncertainly, which is encoded in the covariance, is increased by the expected process noise variance Q_k. Hence, each prediction step essentially increases the overall uncertainty in the process matrix. This is a reasonable assumption, since without any measurement, the successive prediction of target states will become increasingly unreliable. To reduce the uncertainty, a measurement of the system needs to be obtained.

After the respective measurement (detection) of the state vector (object location) z_k has been made, the *update step* corrects the estimates and the uncertainties using a weighted average, with more weight being given to estimates with higher certainty. The Kalman filter also assumes that the measurements are corrupted with some amount of error, which is encoded in R_k.

Update step (after the measurement z_k was made)

$$\text{Measurement pre-fit residual :} \quad \tilde{y}_k = z_k - H\hat{x}_{k|k-1} \tag{8.26}$$
$$\text{Residual of pre-fit covariance :} \quad S_k = R_k + HP_{k|k-1}H^{\mathrm{T}} \tag{8.27}$$
$$\text{Optimal Kalman Gain :} \quad K_k = P_{k|k-1}H^{\mathrm{T}}S_k^{-1} \tag{8.28}$$
$$\text{Updated state estimate :} \quad \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\tilde{y}_k \tag{8.29}$$
$$\text{Updated estimate covariance :} \quad P_{k|k} = (I - K_kH)P_{k|k-1} \tag{8.30}$$
$$\tag{8.31}$$

Here K_k denotes the Kalman gain, I the identity matrix, and R_k is the covariance of the observation noise. As in the prediction step, the uncertainty of the measurement is added directly to the covariance $HP_{k|k-1}H^{\mathrm{T}}$ in (8.27). If we assume the noise of the measurement and the process to be uncorrelated and roughly zero-mean Gaussian distributed, the update of the state estimate can be interpreted as the intersection of the two Gaussians that come from of the state prediction and the state measurement, respectively. The Kalman Gain K_k is the corresponding normalization term that ensures the intersection remains Gaussian distributed.

The algorithm is recursive and computationally light-weight. It only uses the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required. Although the Kalman Filter does not explicitly assume that the measurements and the system errors are Gaussian, the filter estimates the exact conditional probability estimate of the state vector in case they are.

Furthermore, the update step does not need to be computed for every prediction step. If there is a measurement, the state and the covariance can be refined. If there is no measurement, the prediction can be performed on its own. Although this increases the system's uncertainty, this asynchronicity is very valuable for target tracking. Especially if the tracker has a failure mode detection (like the shape model tracker does), the prediction is a reliable indicator of where the tracker should continue looking. Furthermore, the uncertainly is a valuable indicator of how large the search region should be selected.

A key feature of the Kalman filter is that it does not assume that all parts of the target state are actually observable. Through the process transition matrix F and the covariance P_k, relationships between variables can be inferred and estimated without measuring them. This is an essential attribute, since visual object trackers can only measure the state position, but a state prediction without a velocity is not very valuable in target tracking.

The Influence of the State Transition Matrix The state transition matrix encodes the propagation of the target state. Even though in visual object tracking the object should

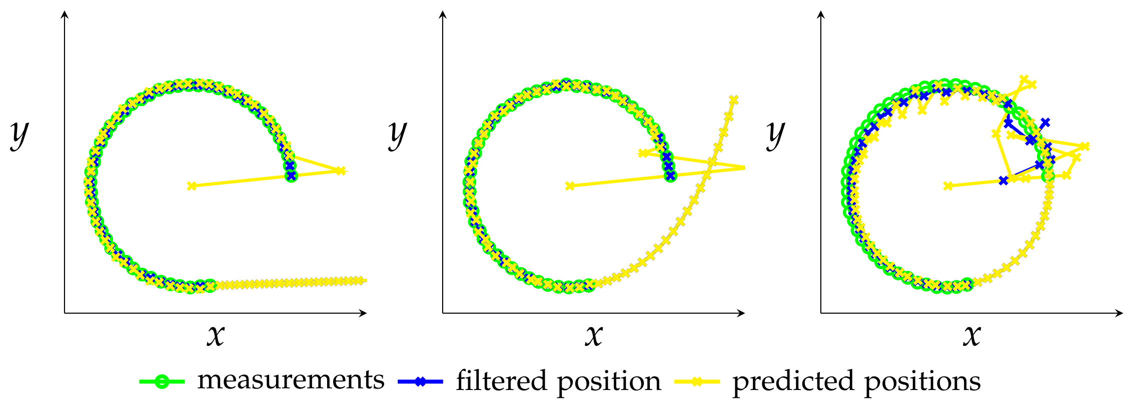


Figure 8.19: In the two left graphs, the ordinary Kalman filter predictions and the measurements for circular motion for two state transition models are displayed; one that has Cartesian acceleration components (right) and one that does not (left). In the right graph, the Extended Kalman Filter predictions and the measurements for a transition model with a constant turn rate and constant velocity are displayed. For all models, the state is initialized at $(0,0)$ and at around 270° the measurements are stopped and only the predictions of the Kalman filter are computed and displayed. The two left models only require a handful of iterations to follow the circular motion but are not able to actually *learn* the circular motion. The model o the right is able to learn the circular motion and correctly predict the location when the measurements are stopped, but requires significantly more measurements to deliver reasonable predictions and filter results.

be moving according to basic kinematics, too weak model assumptions on F may lead to errors in the state prediction. For example, consider Fig. 8.19. On the left, we restrict the target state to merely encode the state position and target velocity in Cartesian coordinates:

$$x_k = \begin{pmatrix} p_k \\ v_k \end{pmatrix}. \tag{8.32}$$

The state transisition matrix then simplifies to

$$F^{vel} = \begin{pmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{8.33}$$

Without encoding the state accelerations, it is impossible to predict any movement that is more complex than a linear transition. In the middle plot of Fig. 8.19, we display the model prediction that is derived from the state described in (8.15) and the transition matrix F from (8.23). Although the predictions are much better, they are incapable of inferring the circular motion correctly. Since both models assume Cartesian coordinates and a linear transition, they will never be able to learn the circular motion from the measurements. To overcome this, a more complex, non-linear, state transition needs to be assumed

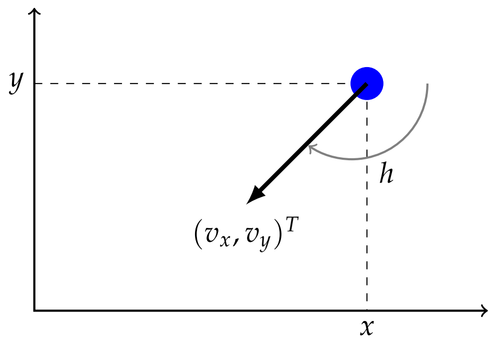


Figure 8.20: An alternative way to represent the target state is by Cartesian position and velocity and, furthermore, the turn rate as the derivate of the heading angle $\dot{h} = \omega$, where $h = \text{atan2}(v_y, v_x)$.

Nonlinearity of the State Transition Function To efficiently learn the circular motion, we assume a slightly adapted target state:

$$x_k = \begin{pmatrix} p \\ v \\ \omega \end{pmatrix}, \tag{8.34}$$

where $\omega = \dot{h}$ is the derivate of the heading angle. The heading angle is encoded in the Cartesian velocities of $v = (v_x, v_y)^T$ by $h = \text{atan2}(v_y, v_x)$. The parametrization is visualized in 8.20. For the state transition we assume a constant turn rate and a constant velocity

$$f = \begin{pmatrix} x + \frac{v_x}{\omega}\sin(\omega T) + \frac{v_y}{\omega}(1 - \cos(\omega T)) \\ y + \frac{v_x}{\omega}(1 - \cos(\omega T)) + \frac{v_y}{\omega}\sin(\omega T) \\ v_x \cos(\omega T) - v_y \sin(\omega T) \\ v_x \sin(\omega T) - v_y \cos(\omega T) \\ \omega \end{pmatrix}, \tag{8.35}$$

For nonlinear state transition function's only the prediction step of the Kalman Filter needs to be adapted. The restriction to a linear measurement matrix and thus the update equations remain unchanged. To predict the covariance estimate, the state transition function is linearized around the current state estimate $\hat{x}_{k-1|k-1}$. Hence, the prediction step becomes

Prediction step (nonlinear state transition)

Predicted state estimate : $$\hat{x}_{k|k-1} = f\left(\hat{x}_{k-1|k-1}\right) + w_k \tag{8.36}$$

Predicted covariance estimate : $$P_{k|k-1} = F P_{k-1|k-1} F^{\mathrm{T}} + Q_k, \tag{8.37}$$

where $$F = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}}. \tag{8.38}$$

For each prediction step, the first order Taylor expansion of (8.35) around the current state estimate $\hat{x}_{k-1|k-1}$ yields F. For the constant turn rate and constant velocity, the derivate can be computed as;

$$F = \begin{pmatrix} 1 & 0 & \frac{\sin(\omega T)}{\omega} & \frac{1-\cos(\omega T)}{\omega} & v_x\left(\frac{T\cos(\omega T)}{\omega} - \frac{\sin(\omega T)}{\omega^2}\right) + v_y\left(\frac{-T\sin(\omega T)}{\omega} + \frac{(1-\cos(\omega T)}{\omega^2}\right) \\ 0 & 1 & \frac{\cos(\omega T)-1}{\omega} & \frac{\sin(\omega T)}{\omega} & v_x\left(\frac{T\sin(\omega T)}{\omega} + \frac{\cos(\omega T)-1}{\omega^2}\right) + v_y\left(\frac{T\cos(\omega T)}{w} - \frac{\sin(\omega T)}{\omega^2}\right) \\ 0 & 0 & \cos(\omega T) & -\sin(\omega T) & -v_x\sin(\omega T) - v_y\cos(\omega T)) \\ 0 & 0 & \sin(\omega T) & \cos(\omega T) & v_x\cos(\omega T) - v_y\sin(T\omega T)) \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{8.39}$$

As shown in Fig. 8.19, in contrast to the linear models, the model is able to learn the circular motion and correctly predicts the motion when the measurements are stopped. This not only comes at the cost of an increased computational complexity, but the system also requires significantly longer to infer reasonable predictions of the state than the linear models in Fig. 8.19. The first few predictions are extremely crude. More complex models are able to also capture varying speeds and changing turn rates. Please see [189] for further details.

Kalman Motion Prediction for Shape Model Tracking The adaptation of the proposed motion model to shape model tracking is straightforward. In the first frame, the motion motion is initialized at the known location of the object. Since the initial position of the tracker is known and the measurements of the shape model are generally very precise, the measurement matrix R_k from (8.27) is generally initialized with a small measurement noise. However, if the object is not detected (and hence measured) in a number of subsequent frames, the measurement noise is increased. This is done to account for false detections that may appear when the search region is very large. Since the proposed motion models do not account for ego motion of the observer, the motion model is initialized with a rather large uncertainty in the state transition matrix.

If the tracking was successful, the motion model merely needs to be updated with the tracker position (measurement) according to (8.26) – (8.30). Then, the most probable location of the search region can be derived from the motion model by computing the state prediction from (8.24). In our experiments, we found that the nonlinear state transition from (8.39) did not improve the motion estimation considerably. Especially in sequences in which the object is occluded very early, the long initialization phase of the non-linear model is prohibitive. Hence, we restrict the motion model to the linear state transition from (8.23). More detailed experiments on the benefits of the motion mode are presented in Section 9.2. The generality of the motion model allows to add it to all of the proposed improvements of the shape model. The improved shape model that uses a dynamic generation of the pyramid levels, the robust improvements from Section 8.3.3 and does not use a motion model is denoted as `gen_level_sm_no_motion_model` within the later experiments.

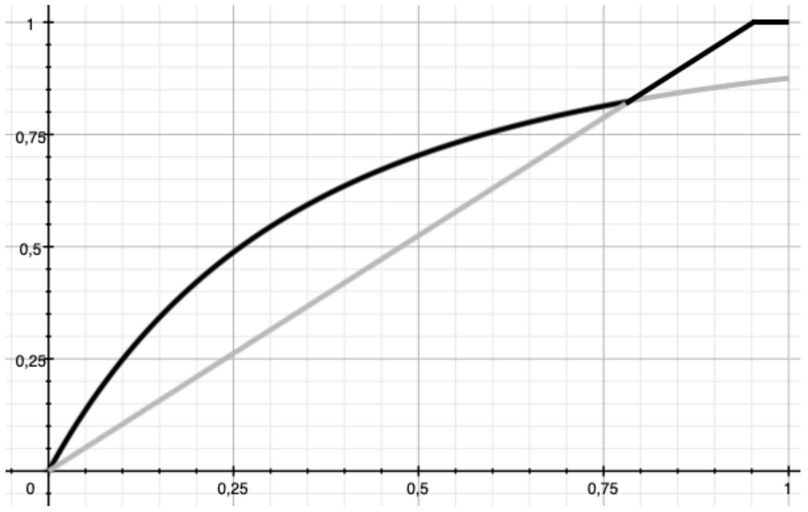


Figure 8.21: If the tracking is successful, the displayed bold black function is applied to the confidence α. The bold black line is the maximum described in (8.40). The gray lines visualize the single terms within the maximum. If the tracker is very uncertain, the confidence is incremented more than if it is just slightly uncertain. This allows the tracker to recover from detection failure and does not influence it too strongly if there have been positive detections in the last few frames.

8.3.5 Search Space Optimization

The factor that influences the speed of the shape model tracking the most is the size of the search region. In the baseline shape model, the search region radius is determined according to (8.4). Hence, it is a function of the current object scale σ_c and the confidence α. As stated in Section 8.2, the confidence is incrementally decreased by dividing it by 1.05 if the tracking fails and reset to 1.0 as soon as the tracking succeeds again. This has the disadvantage that a false detection is trusted instantaneously. Hence, tracking fails very quickly when a single false positive is detected. Therefore, in the improved shape model, a slightly more profound update of the search space is computed

First of all, the confidence is not initialized with 1.0, but with 0.5. This is to account for the uncertainty of the shape model in the first few frames. Especially the motion model has no initial idea in which direction the object might be moving. Then, the confidence is incrementally increased by

$$\alpha_{t+1} = \max\left(1.0 - \left(\frac{1}{\alpha_t + 1}\right)^n, \alpha_t \cdot 1.05, 1.0\right), \tag{8.40}$$

when tracking is successful. Here, n controls how much to trust a detection of the tracker. We use $n = 3$ in all of our experiments. The confidence update function is visualized in Fig. 8.21. As in the baseline shape model, the confidence is decreased by dividing it by 1.05 if tracking fails. The above initialization and update of the confidence allows the tracker to *warm up* to the tracked object and does not trust a re-detection of the object instantaneously.

The motion model allows to reduce the size of the search region in comparison to the baseline shape model that has no motion prediction. This generally speeds up the tracking process and also reduces the chance of false detections. Although the motion model predictions are generally accurate, unexpected motion does occur in tracking sequences. As a consequence, the object might not be within the search region and be lost by the tracker. In these cases, the confidence is reduced and the size of the search region is increased in the subsequent frames. This generally allows the tracker to locate the object within the next few frames. However, to account for very small objects, a minimal size of the search region is set for all sequences. We use a minimal radius of 30 pixels throughout the experiments.

The dynamic generation of the pyramid levels allows to use the model confidence to determine the optimal number of scales. If the tracker is very confident, the search region is typically quite compact and there is no reason to use too many object scales. However, when the tracker uncertainty grows, the increased search region may require more object scales to remain efficient. Hence, the confidence also indirectly encodes the number of pyramid levels that are used for tracking the object. Since the baseline shape model has a fixed number of scales, this efficient adaption is impossible. Especially for objects that are initialized very small and grow throughout the sequence, the baseline shape model may require very long to locate the object when the confidence drops and the search regions grows very large.

8.4 Discussion

This chapter provided a detailed discussion of the shape model tracking. The main weak points of the baseline shape model tracker were presented and discussed. They include a fixed number of pyramid levels during object localization and a weakness when the object scale and appearance changes considerably within a sequence. A number of extensions to the baseline shape model tracking approach are presented to tackle the above mentioned weaknesses. The extensions allow to improve the robustness and speed of the baseline, as shown in Chapter 9. The improvements include a dynamic pyramid generation that enables to generate a variable number of pyramid levels during tracking. This allows to reduce the runtime and improve the robustness of the shape model considerably. Similarly, the presented motion model and the search space optimization enable to reduce the search space as much as possible and are also expected to improve the overall performance. To tackle long sequences, where the object may change scale and appearance considerably, a robust extension of the shape model tracker that uses multiple shape models to locate the object location is presented. In the following chapter, quantitative and qualitative results and comparisons of the baseline and the different extensions are presented and discussed.

9 Results and Comparison to the State of the Art

This chapter presents detailed qualitative and quantitative experiments to validate the performance and efficiency of the shape model tracker proposed in the previous chapter. In a first step, the selected evaluation methodology is presented in Section 9.1. Then, different experiments to highlight the advantages of the improved shape model in comparison to the baseline shape model on the Playing for Tracking Data (PFTD) dataset (Section 5.2) are presented in Section 9.2. The section also includes extensive experiments that compare the shape model tracker to the current state of the art in real-time object tracking. In Section 9.3, the real-time capabilities of the shape model tracker are evaluated on a Raspberry Pi. Then, in Section 9.4, the versatility of the shape model tracker is displayed in a collection of different applications. Finally, the chapter is concluded with a discussion in Section 9.5.

9.1 Evaluation Metrics and Protocol

In the following, we present the selected performance measures and the evaluation methodology. As is common in single-object tracking, we include measures that estimate the accuracy, robustness, and computational complexity. Furthermore, since we are also interested in validating the failure mode detection of the shape model tracker, we include a measure to identify the tracker's capability of detecting the absence of the object. In general, our basic evaluation routine is very similar to that of the recently proposed long-term tracking dataset (LTTD) [222]. However, in contrast to the LTTD evaluation, we further include the Φ_{rIoU} and scale measure from Chapter 4. This enables us to gain further insights into the accuracy of the trackers and their capability of detecting scale changes. The individual measures are summarized in the following.

To measure the accuracy of the trackers, we use the standard Φ_{IoU} as presented in (3.3). The average IoU of all sequences is denoted as $\varnothing\Phi_{\text{IoU}}$. Since the ground truth is not represented by bounding boxes but by a pixel-precise representation of the objects, we also use the Φ_{rIoU} from (4.1) for all trackers that are restricted to boxes. This helps

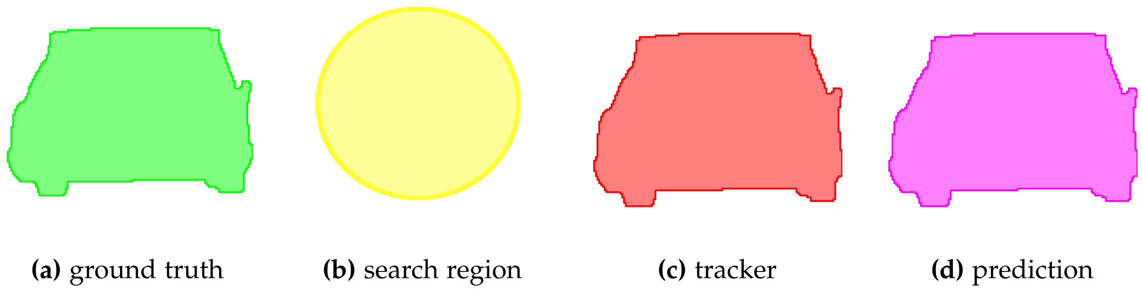


Figure 9.1: The coloring used for the regions throughout the evaluation is displayed. All regions are generally displayed with slight transparency to identify overlapping regions and to obtain a notion of the context underneath the regions.

to gain a perspective on how well the trackers are coping with the pixel-precise ground truth.

To measure the robustness of the trackers, we use the TPR from (3.4). We use the common Φ_{IoU} threshold of 0.5 to validate whether a the object is localized correctly or not. In contrast to the VOT evaluation protocol [116], we do not reinitialize the tracker when $\Phi_{\text{IoU}} = 0$. A complete long-term tracker must be able to cope with tracking failure and re-detect the object by itself. Hence, the measure should also capture how well a tracker copes with this. Similarly, if the tracker is able to predict the absence of an object, we also calculate the TNR from (3.5). As described in (3.4) and (3.5), the theoretical optimum for the TPR and TNR is 1.0.

The runtime of the trackers is measured in milliseconds per frame. Note that all tests were conducted on the same machine. Although the implementations may be different, we took care to re-implement the most prominent real-time trackers in C to enable a fair comparison. Therefore, the baseline correlation-based trackers are all based on an own implementation and not on the open-source Matlab code that is available online. The shape model tracker is also implemented in C and optimized where possible. However, the code is still experimental. When there was a choice between easy debugging and access to datatypes versus runtime efficiency, the first option was selected. Hence, keep in mind that the runtimes could be improved further. The experiments were conducted on an Intel Core i7-4810 CPU @2.8GHz with 16GB of RAM with Windows 7 (x64) unless specified otherwise. For a fair comparison, the deep-learning methods are configured to not make any use of the internal graphics card and solely run on the CPU.

In the evaluation, a consistent coloring is used in the figures to display the ground truth, the shape model search region, the tracking result, and the predicted region from the motion model when tracking is not successful. The coloring is visualized in Fig. 9.1. As described in Section 2 and Section 8.1, the search region $\mathcal{S}$ only contains the locations where the reference point of the shape model is placed. As a consequence, the search region may be smaller than the region that is being tracked.

Table 9.1: Overview of the average Φ_{IoU}, TPR, and average runtime of the baseline shape model and the shape model improvements.

	$\varnothing\Phi_{\text{IoU}}$	TPR	time per frame
`baseline_sm`	0.095	0.098	634.0 ms
`all_level_sm`	0.105	0.111	775.0 ms
`gen_level_sm`	**0.417**	**0.527**	29.1 ms
`gen_level_sm_no_motion_model`	0.404	0.511	32.1 ms
`gen_level_sm_regression`	0.394	0.486	35.7 ms
`gen_level_sm_precise`	0.379	0.475	64.1 ms

9.2 Experiments: Playing For Tracking Data

In a first step, we evaluate the improvements of the shape model tracker and compare their performance to the baseline shape model tracking approach. In a second step, the best performing shape model tracker configuration is compared to the current state of the art in object tracking. We restrict the methods are restricted to the top performing deep-learning methods and to real-time capable approaches.

9.2.1 Shape Model Tracking Improvements

In a first experiment, we evaluated the $\varnothing\Phi_{\text{IoU}}$, the TPR, and the average runtime per frame for the different improvements described in Chapter 8. This includes the `baseline_sm` that is described in detail in Section 8.2 and Böttger *et al.* [27] and the extension `all_level_sm`, which adapts the model update to all levels and is described in Section 8.3.1. Both methods have a fixed number of pyramid levels that is determined in the initialization of the tracker. In contrast, the other four improvements all dynamically adapt the number of pyramid levels during tracking and reinitialize the tracker when the scale changes significantly. The different scale models are stored in a hierarchy of shape models, as described in Section 8.3.3. Furthermore, all but `gen_level_sm_no_motion_model` make use of a motion model to improve the initial estimate of the object location and predict the most probable object location when tracking fails (see Section 8.3.4). The methods `gen_level_sm_regression` and `gen_level_sm_precise` further attempt to improve the quality of the dynamically generated pyramid levels from `gen_level_sm`. The concept is described in detail in Section 8.3.2.

The results of the six different methods are displayed in Table 9.1. In general, the baseline shape model tracker appears to have extreme difficulties with the PFTD dataset. The $\varnothing\Phi_{\text{IoU}}$ and TPR are very low. The extension of the update to all pyramid levels does not seem to help significantly. Both `baseline_sm` and `all_level_sm` have a runtime that is far from real-time. However, the extension of the shape model tracker to dynamically adapt the number of pyramid levels (`gen_level_sm`) brings a very large performance boost of the accuracy, robustness, as well as in terms of the average runtime. The motion model appears to only have a minor positive impact. Interestingly,

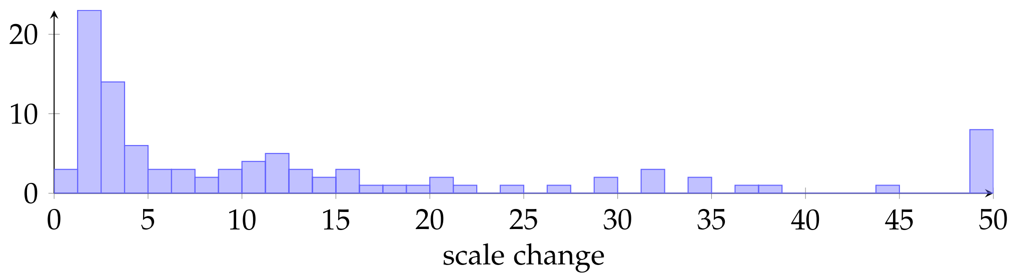


Figure 9.2: A histogram of the average scale change in the PFTD dataset. The scale change is measured as the factor by which the size of the initial tracking region changes throughout a sequence. If the object decreases its size, the inverse factor is used as the scale change.

the refinement of the dynamically generated pyramid levels does not improve the performance. In the following, the results of the individual extensions are evaluated and put into perspective in more detail.

Dynamic Pyramid Generation The most significant improvement comes from the dynamic generation of the pyramid levels. The gain in accuracy and robustness is primarily caused by the hierarchy of shape models. In the PFTD dataset, many objects are initialized at a very small or a very large scale and change their size and appearance considerably throughout the sequence. To quantify this, the average scale change is visualized in Fig. 9.2. Here, the scale change is measured by computing the size of the ground truth throughout a sequence. The factor by which the size changes with respect to the first frame is the scale change. If the size becomes smaller throughout a sequence, the inverse factor is used as the scale change. As shown in Fig. 9.2, the majority of the objects undergo a very significant scale change. For example, in almost half of the sequences, the object size changes by a factor of more than 10. Without reinitializing the shape model, the level of detail in the finest pyramid level does not match the object anymore. As a consequence, the baseline shape model tracker loses the object or mistakes it for a different object. In contrast, in the improved shape models, the object is reinitialized when the scale of the object changes considerably. This helps to improve the performance on these sequences by a large margin.

To further validate this assumption, the development of $\varnothing\Phi_{\text{IoU}}$ is visualized for all sequences in Fig. 9.3. Here, the length of each sequence is normalized to equal length and the average Φ_{IoU} value is displayed. As expected, the accuracy of the baseline shape model generally drops relatively early in the sequences and is not able to recover. Since the scale change generally accumulates over time, the fixed level of detail in the baseline shape model is a significant disadvantage. In contrast, the hierarchy of shape models used in `gen_level_sm` is substantially more robust.

While the increased accuracy and robustness is mostly due to the hierarchy of shape models, the performance gain is primarily caused by to the dynamic pyramid level generation. The PFTD dataset is acquired from inside a car that is driving through the streets. Hence, in many sequences, cars are initialized when they are relatively small

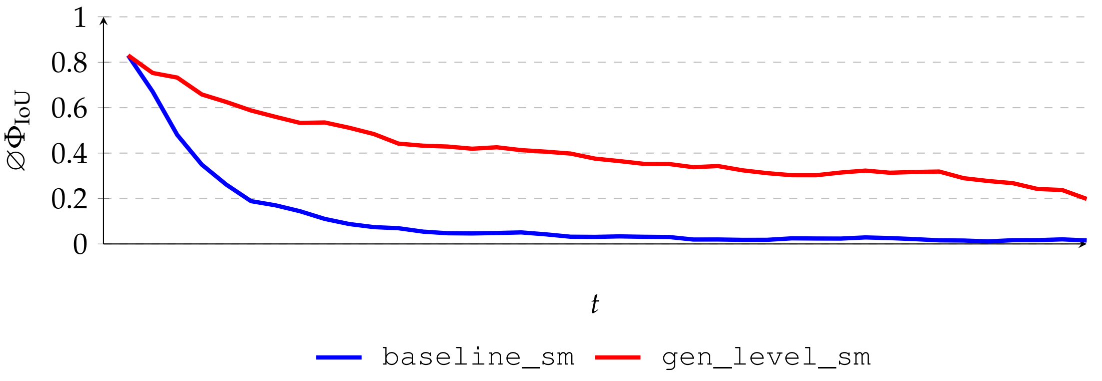


Figure 9.3: The development of $\varnothing\Phi_{\mathrm{IoU}}$ over time is averaged for each sequence and displayed for `baseline_sm` and `gen_level_sm`. Evidently, the improvements of the the baseline shape model enable the tracker to cope better with long sequences.

and distant from the car. Throughout the sequence they gradually come closer to the car and their size grows. In these settings, the fixed number of pyramid levels of the baseline shape model is a great disadvantage. On the one hand, the small model template does not match the object anymore and, consequently, the object cannot be localized. On the other hand, the larger search region increases the computational complexity considerably. Since the entire search space $\mathcal{P}_{\mathcal{S},\mathcal{R},\Sigma}$ is checked on the coarsest pyramid level, it is computationally expensive to have a large search region $\mathcal{S}$ on the coarsest pyramid level. Since the baseline method has a fixed number of pyramid levels, no new pyramid level can be generated to reduce the workload on the coarsest level. In contrast, `gen_level_sm` generates a further pyramid level in these settings. As a consequence, the computational complexity can be reduced. This enables efficient tracking and an average runtime of only 29.1 ms per frame, as shown in Table 9.1. Note that the number of pyramid levels does not only depend on the size of $\mathcal{S}$, but also on the size of the shape model template. A very small template sometimes cannot be sub-sampled any further. However, since the improved shape model tracker reinitializes the shape model when the scale changes to much, the template is often much larger than it was during initialization. As a result, the robust update and the dynamic pyramid level generation go hand in hand and depend on each other.

A further advantage of the dynamic pyramid level generation is the reduction of the computational complexity of the tracker initialization. For `baseline_sm` and `all_level_sm`, the shape model is generated on each pyramid level independently. This requires computing image edges and subpixel-precise model points on each level. In contrast, the methods based on `gen_level_sm` only generate the shape model on the first pyramid level and do not require an image pyramid for the tracker initialization. As a consequence, the average runtime of the tracker initialization is significantly smaller for these methods. They require an average of 20.5 ms for all sequences. In contrast, the initialization of `baseline_sm` and `all_level_sm` requires 31.9 ms on average. The reduced initialization time is essential for the generation of the shape model hierarchy. It allows to construct the shape model hierarchy in real-time during tracking.

Table 9.2: Overview of average Φ_{IoU}, the TPR, and the average runtime of the improved shape model with and without using a motion model.

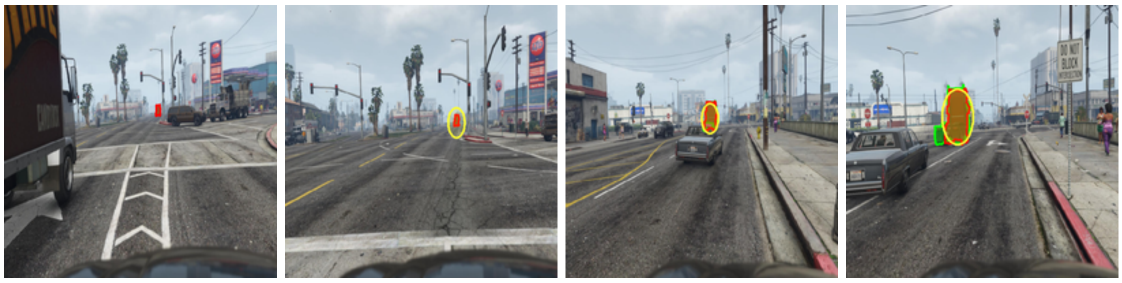
	$\varnothing\Phi_{\text{IoU}}$	TPR	time per frame
`gen_level_sm`	**0.417**	**0.527**	29.1 ms
`gen_level_sm_no_motion_model`	0.404	0.511	32.1 ms

A collection of examples from `gen_level_sm` are displayed in Fig. 9.4. The average Φ_{IoU} values , the TPR, and the number of frames are displayed for each sequence. The tracker is able to cope with significant scale changes, long sequences, and a collection of different settings.

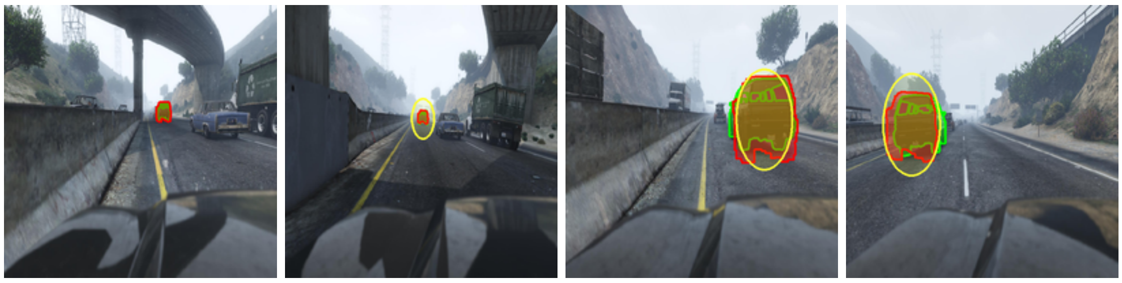
Motion Model It is expected that the motion model described in Chapter 8.3.4 improves the robustness of the shape model tracking. A reasonable prediction of the object position should improve the re-detection if tracking fails or the object is occluded. The results for `gen_level_sm` and `gen_level_sm_no_motion_model` are summarized in Table 9.2. Without a motion model, the accuracy as well as the robustness is reduced on average. Furthermore, the runtime is also decreased marginally. The reduction of the runtime is caused by the fact that the motion model allows to reduce the size of the search region. If the object is lost and no prediction of the object location is known, the search region is centered at the last known position. Hence, if the object is moving, the size of the search region must be increased until the object is within the search region again. However, with knowledge of the object motion, the search region is generally much closer to the moving object and the re-detection time can be reduced.

Two examples where the motion model is able to improve the tracking results are displayed in Fig. 9.5. In the first row, the object is occluded completely early in the sequence. The motion model creates reasonable predictions of the object location and enables the re-detection of the object when it reappears near the end of the sequence. Note that $\varnothing\Phi_{\text{IoU}}$ is quite low for the respective sequence. The reason is that the shape model tracker predicts the shape of object by applying transformations to the initial object region. Hence, if half of the object is occluded and the shape model makes a correct prediction, the Φ_{IoU} cannot be larger than 0.5. In the second row, the motion model enables the tracker to reduce the time required for re-detecting the object. The tracker fails when the car gets very close to the camera and the perspective is strong. However, the predictions of the motion model are reasonable and the object is re-detected when the perspective is reduced.

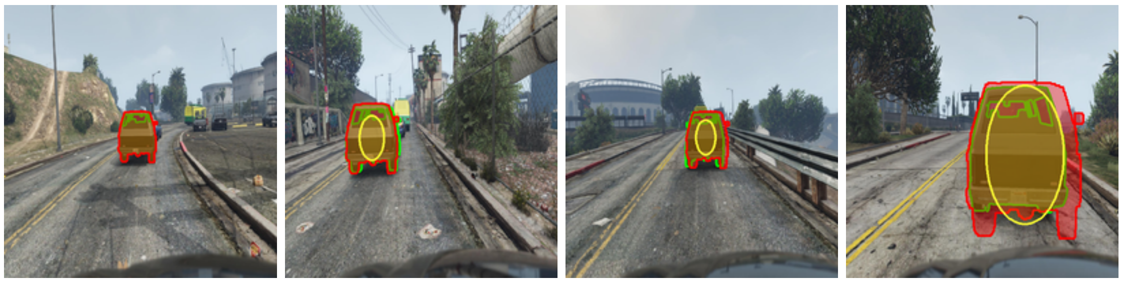
Detection of Object Absence In the above setting, the prediction of the motion model is assumed to be the tracker output if tracking fails. Therefore, none of the above methods predict the absence of the object directly. To measure how well the shape model can estimate the absence of the object, we conducted further experiments. Here, the predictions of the motion model are still used to update the search region. However, the prediction of the tracking region is not assumed to be the tracker output. Instead, the tracker is



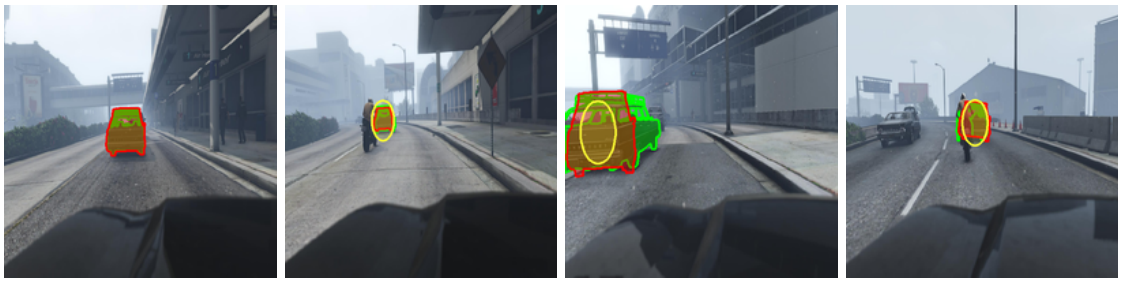
(a) $\varnothing\Phi_{\text{IoU}} = 0.8$, TPR = 1.0, #frames = 317



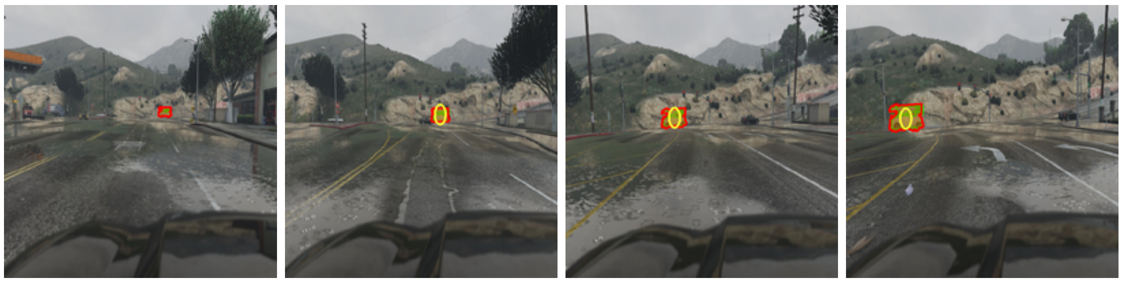
(b) $\varnothing\Phi_{\text{IoU}} = 0.68$, TPR = 0.89, #frames = 234



(c) $\varnothing\Phi_{\text{IoU}} = 0.73$, TPR = 1.0, #frames = 1337

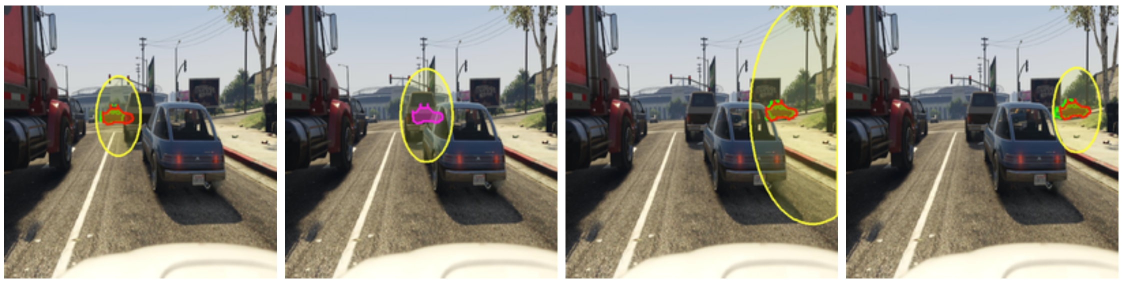


(d) $\varnothing\Phi_{\text{IoU}} = 0.441$, TPR = 0.5, #frames = 1264

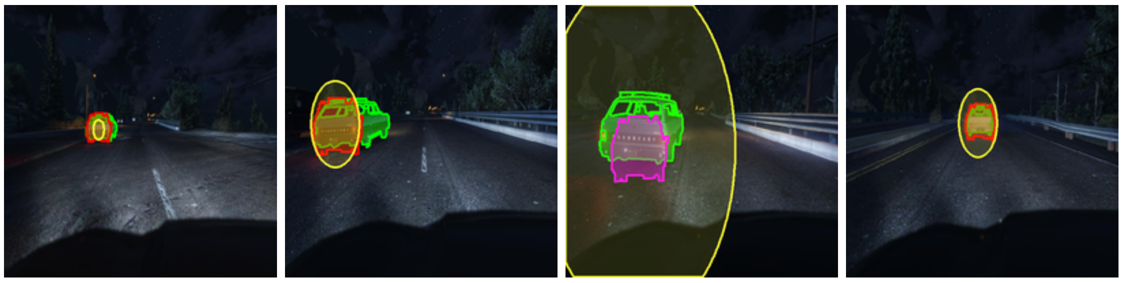


(e) $\varnothing\Phi_{\text{IoU}} = 0.71$, TPR = 0.82, #frames = 119

Figure 9.4: A collection of results from `gen_level_sm`. The tracker is able to track cars in various different settings and in long-term sequences with over 1000 frames (e.g., in (c) and (d)).



(a) $\varnothing\Phi_{\text{IoU}} = 0.52$, TPR = 0.41, #frames = 54



(b) $\varnothing\Phi_{\text{IoU}} = 0.68$, TPR = 0.88, #frames = 322

Figure 9.5: Two examples were the motion model allows an efficient re-detection of the object. In the first row, the object is completely occluded early on in the sequences. The prediction of the object motion creates reasonable predictions of the object position and allows to re-detect the object after the occlusion ends. In the second row, the perspective of the car is too strong and the car is lost. However, the predicted object position is reasonable and the car is re-detected very quickly.

assumed to output an empty region if it was not successful. The respective method is denoted as `gen_level_sm_no_pred`. To tackle small occlusions and short-term missing detections from the tracker, we also introduce the methods `gen_level_sm_pred_5` and `gen_level_sm_pred_10`. If tracking fails, the prediction is assumed to be the tracker output only for the first five and ten frames, respectively. If the object is not re-detected by then, the tracker region is assumed to be empty. If the tracker returns an empty region and the ground truth is also empty, a true negative is detected. The TPR and TNR of the approaches are displayed in Table 9.3. The best TNR is generated by `gen_level_sm_no_pred`. This is not surprising since it is the strictest of the four methods. As expected, the TPR drops continuously from `gen_level_sm` to `gen_level_sm_no_pred`. In general, the number of frames to trust the prediction is a parameter that allows a trade-off between the capability of detecting the absence of the object and robustness to short-term occlusions and missed detections. Note that the runtime is the same for all approaches since the tracker does exactly the same. Only the interpretation of the tracker output and the motion model output changes.

In general, detecting the absence of the object also reduces the average accuracy. For example, in Fig. 9.5 (a), although the object's absence is predicted correctly, the ground truth does not have an empty region *while* it is being occluded. In these frames, the prediction of the object has a $\Phi_{\text{IoU}} > 0$ while `gen_level_sm_no_pred` has $\Phi_{\text{IoU}} = 0$. Similarly, in Fig. 9.5 (b), the prediction improves the average accuracy since $\Phi_{\text{IoU}} > 0$ for

Table 9.3: Overview of average Φ_{IoU}, the TPR, and the average runtime of the shape model tracker improvements with and without using the predicted region from the motion model as the output of the tracker when tracking fails.

	$\varnothing\Phi_{\text{IoU}}$	TPR	TNR	time per frame
`gen_level_sm`	**0.417**	**0.527**	0.0	29.1 ms
`gen_level_sm_pred_10`	0.409	0.524	0.06	29.1 ms
`gen_level_sm_pred_5`	0.390	0.501	0.14	29.1 ms
`gen_level_sm_no_pred`	0.369	0.478	**0.18**	29.1 ms

Table 9.4: Overview of average Φ_{IoU}, the TPR, and the average runtime of the improved shape model tracker and the point regression approaches.

	$\varnothing\Phi_{\text{IoU}}$	TPR	time per frame
`gen_level_sm`	**0.417**	**0.527**	29.1 ms
`gen_level_sm_regression`	0.394	0.486	35.7 ms
`gen_level_sm_precise`	0.379	0.475	64.1 ms

the prediction and the ground truth label (without the prediction, it would be 0).

Point Regression Unfortunately, the point regression methods were neither able to improve the robustness nor the accuracy of the improved shape model tracker. A summary of the average Φ_{IoU} and TPR are displayed in Table 9.4. Especially the precise contour point snapping approach decreases the performance. While the results may seem counterintuitive, the snapping of the zoomed model points from (8.7) to the subpixel-precise edges in the image pyramid introduces an unwanted model update step. In the first frame, the model is initialized from the input image and the shape model is perfectly aligned with the subpixel-precise image edges. During tracking, the shape model predicts the best transformation of the shape model to the input image and the shape model is adapted to the matches found for each model point. The model points are not enforced to lie directly on subpixel-precise edges. Instead, the update is weakened by a λ factor as described in (8.6) and displayed Fig. 8.5. A too strong update has a very negative effect on the tracking performance: noise edges may be added to the model and mismatches are trusted instantaneously. As a consequence, the performance is very low. However, snapping the model points in the higher pyramid levels to the contour points in the dynamic pyramid generation stage does exactly the same. It expects the shape model to fit perfectly to the image and fixes the single model points in the higher pyramid levels accordingly. Hence, the shape model may drift and the performance is decreased.

Robustness to Illumination Changes In Section 8.1 and Böttger *et al.* [27], the shape model tracker is claimed to be robust to non-linear illumination changes. To validate this, there is a very interesting sequence within the PFDT dataset. In this sequence, the lighting changes abruptly from one frame to the next. In frame 97, the lighting changes

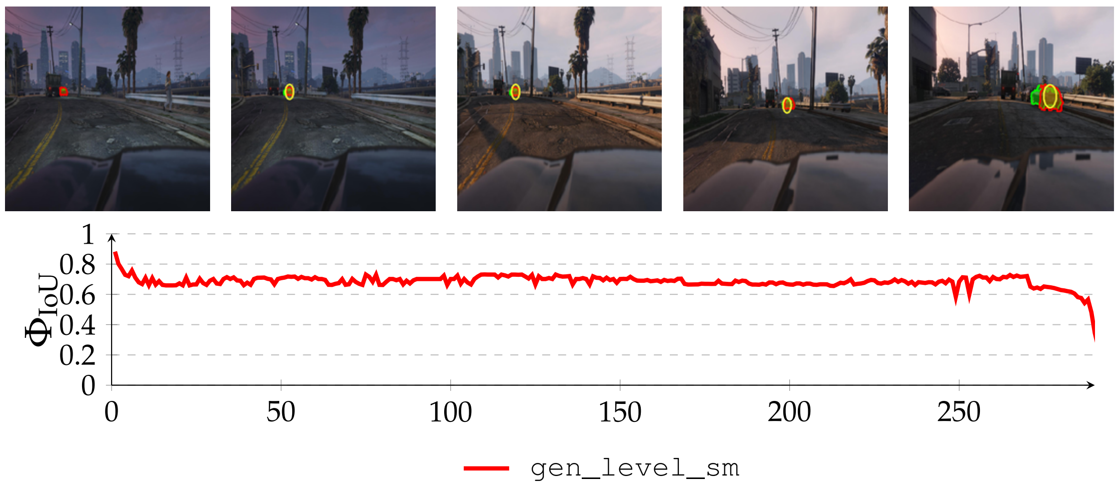
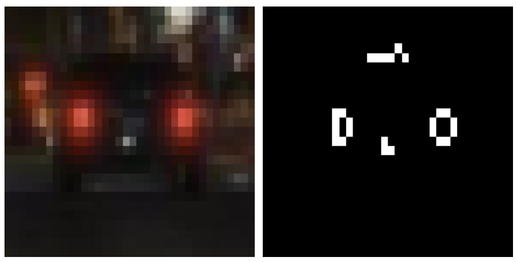


Figure 9.6: The Φ_{IoU} for the sequence indicated by five images in the first row is displayed. In the sequence, the lighting changes abruptly around frame 100. The score of `gen_level_sm` is not affected by the sudden change of the lighting.


(a)

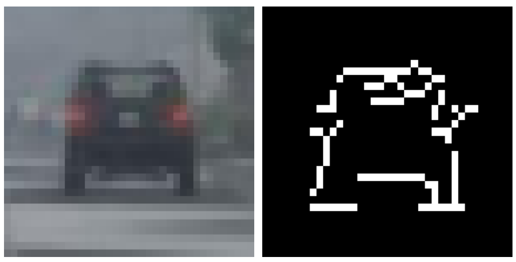

(b)

Figure 9.7: In (a), a model generated at night is displayed. The low contrast between the car and the road generates a model that is mainly focused on the lights of the car. These are not very descriptive and may easily be mistaken for any other car. In (b), a typical model that is generated in better lighting conditions is displayed..

from night time to dawn. As expected, the score of the tracker is not affected by this chanege. A few example images from the sequences and the respective Φ_{IoU} values are displayed in Fig. 9.6. In general, all moderate lighting changes within the PFTD dataset can be handled by the tracker without difficulties.

However, the tracker is not invariant to the general illumination within a sequence. If there is very little light, the contrast might be too low to extract reasonable subpixel-precise edges from the input template. As a consequence, the tracker has difficulties to locate the object and may fail. There are many sequences in the PFTD dataset that are initialized with a very small size. Especially in these sequences, the initialization process might have difficulties to find adequate subpixel-precise edges to create robust model points. An example is visualized in Fig. 9.7. The only significant edges in the left template are the lights of the car. However, since many cars have similar lights at night, these are not very descriptive in general. Therefore, the model is prone to mixing up the target with a different car. The similarly sized template that is initialized at daytime has significantly more pronounced edges. As a result, the template is much more descriptive.

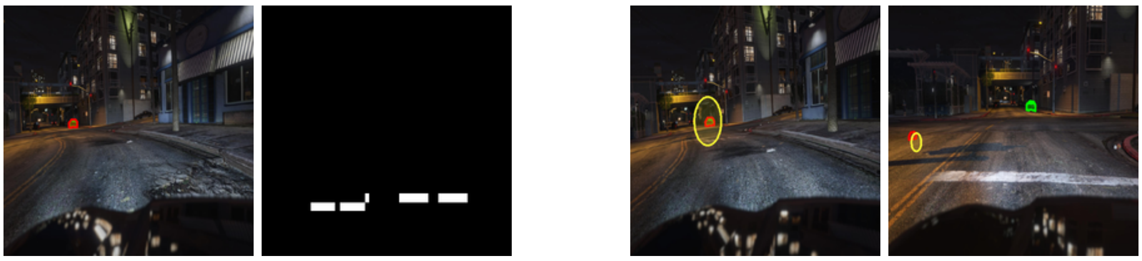

(a) (b)

Figure 9.8: In (a), the tracker is initialized with a very small template and a very low image contrast. The resulting shape model (right image in (a)) is merely a straight line. As a result, the tracking already fails in the fourth frame and returns false positive tracking results (right image in (b)).

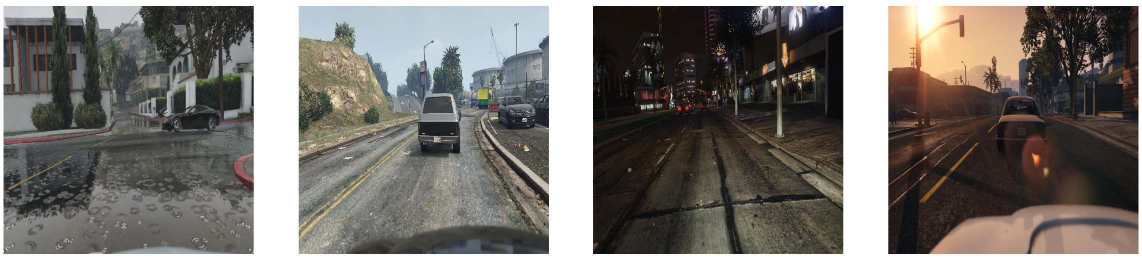

(a) rain **(b)** daytime **(c)** night **(d)** dawn

Figure 9.9: Example of the four lightings into which the PFTD sequences are split. In (a), an example of rain is displayed. Although not explicitly enforced, there are no sequences of rain at night. In (b), an example of daytime is displayed. The amount of sunshine may vary, but generally the weather is quite good. In (c), a night sequence and in (d), a dawn sequence are visualized.

In Fig. 9.8, an extreme example is visualized. Here, the object is very small and the only reasonable edge the initialization process finds is the bottom of the car. However, the resulting line-shaped model is not descriptive at all and the model is lost only a few frames after the initialization (Fig. 9.8 (b)).

To evaluate whether this is a general problem, a further experiment was conducted. In a first step, all of the sequences were divided into the four different kinds of lightings: rain, daytime, night, and dawn. Although it was not explicitly enforced during the dataset generation, there are no sequences at night where it rains. Hence, the lighting categories are unique. Examples of four different kinds of lightings are visualized in Fig. 9.9. The average Φ_{IoU} and the TPR are displayed for the four lighting categories in Fig. 9.10. As shown, the shape model has significantly lower scores in sequences with bad lighting, such as night and dawn. This is mostly due to the low contrast in these sequences. The shape models are not descriptive enough and the tracker may fail. Although it is possible to adapt the subpixel-precise edge extraction to support images with lower contrast, it also increases the vulnerability to image noise and false edges and does not help in these sequences.

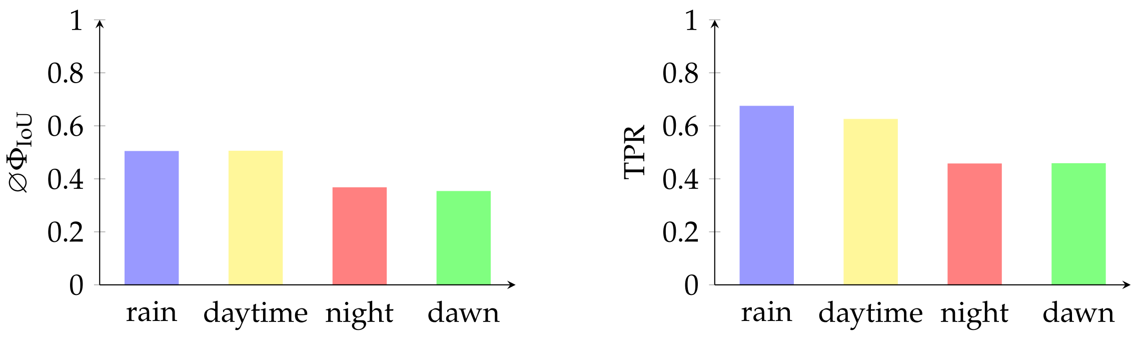


Figure 9.10: The sequences are split into four disjoint lightings (rain, daytime, night, and dawn) and the $\varnothing\Phi_{\mathrm{IoU}}$ (left) and TPR (right) values are displayed for the `gen_level_sm` tracker. The shape model tracker has difficulties in sequences where the lighting is not good.

Strengths and Weaknesses In general, the shape model is able to track the cars in the PFTD very successfully. In the following, a short summary of the observed strengths and weaknesses of the shape model tracker is presented.

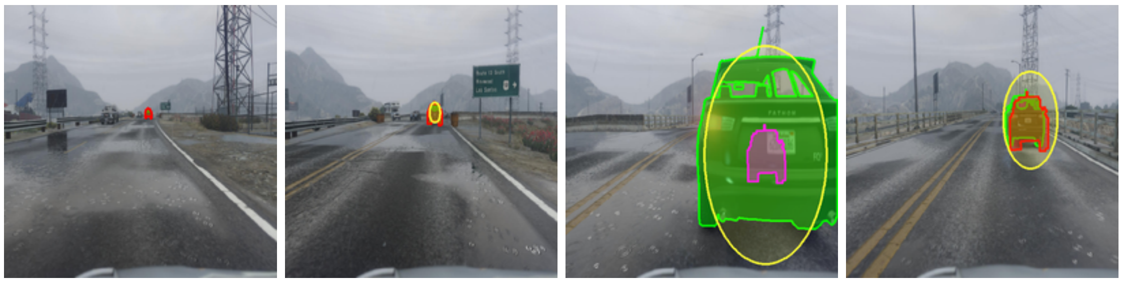
The main strengths of the shape model tracker include:

- The shape model tracker is able to successfully track the cars in many of the sequences with high robustness and accuracy. A few examples are shown in Figs. 9.4 and 9.11. The subpixel precision of the model points also allows long-term tracking in sequences with more than 1000 frames without drift, as shown in Fig. 9.4 (c), Fig. 9.4 (d), and Fig. 9.11 (b).

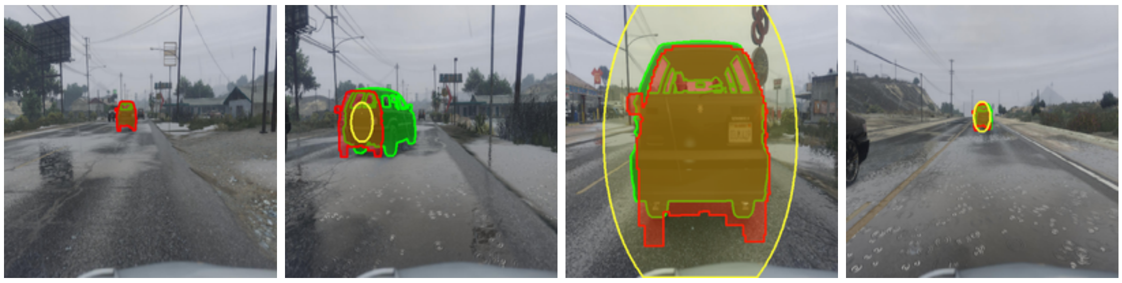
- The motion model and the failure mode detection of shape model tracker enable it to recover from full occlusion and if tracking fails for a short period of time, as shown in Fig. 9.5.

- The metric used to compute the shape model score is generally robust to nonlinear illumination changes. Consequently, the shape model tracker can cope very well with changing illumination, as displayed Fig. 9.6.

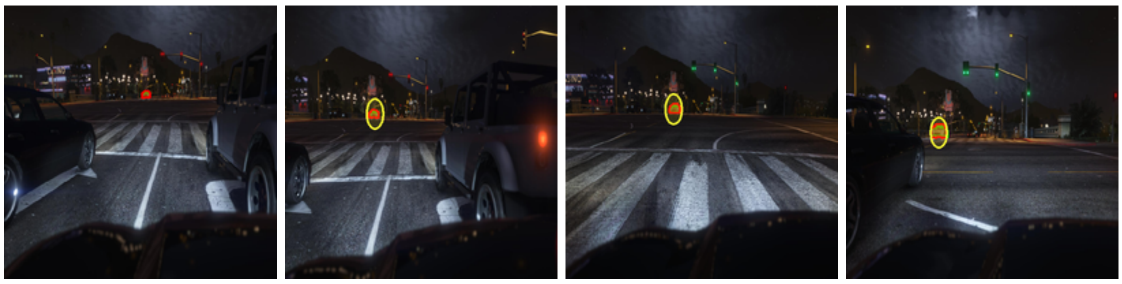
- The shape model tracker requires only an average of 29.1 ms to track the objects in the PFDT dataset, as indicated in Table 9.1. Hence, for the 30 fps in the PFDT dataset, the tracker is real-time capable. However, the runtime comparison between the different shape model configurations is difficult. Although the runtime of the shape model tracker is affected by different factors, the size of the search region has the most influence. Hence, if tracking is unsuccessful and the confidence of the tracer drops (and hence the size of the search region is increased), the runtime of the tracker is higher. Therefore, in difficult sequences, the tracker generally requires more than the average runtime and in easy sequences slightly less. Furthermore, the shape model tracker configurations that have a weaker robustness generally required longer on average. This is not only necessarily due to their inherent computational complexity, but also by the fact that they have a weaker robustness



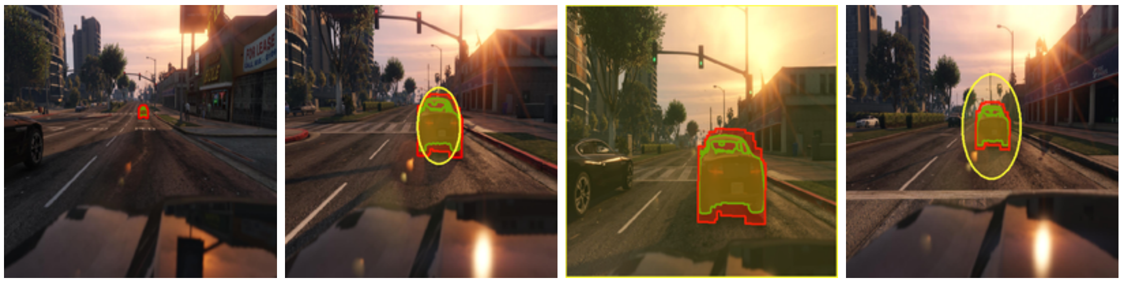
(a) $\varnothing\Phi_{\mathrm{IoU}} = 0.57$, TPR = 0.57, #frames = 186



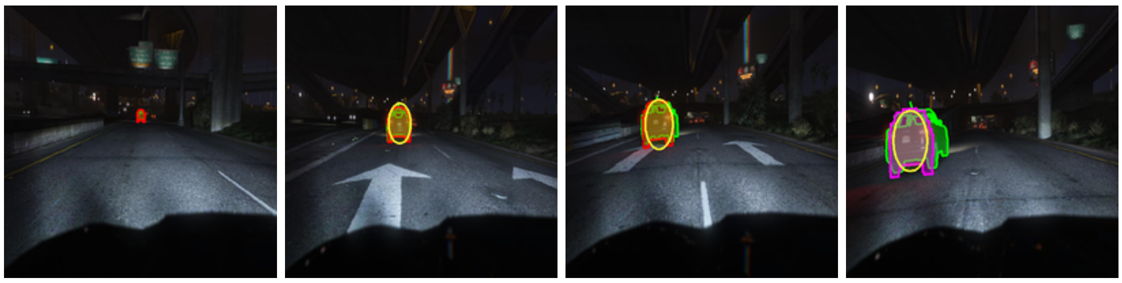
(b) $\varnothing\Phi_{\mathrm{IoU}} = 0.77$, TPR = 0.94, #frames = 1066



(c) $\varnothing\Phi_{\mathrm{IoU}} = 0.82$, TPR = 0.99, #frames = 231

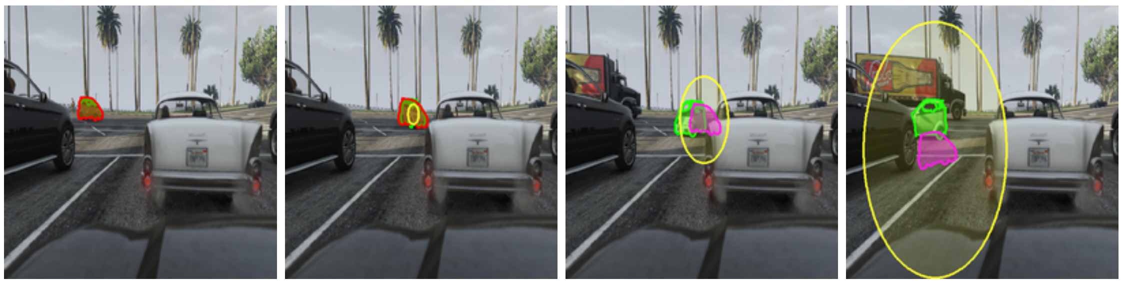


(d) $\varnothing\Phi_{\mathrm{IoU}} = 0.58$, TPR = 0.67, #frames = 397

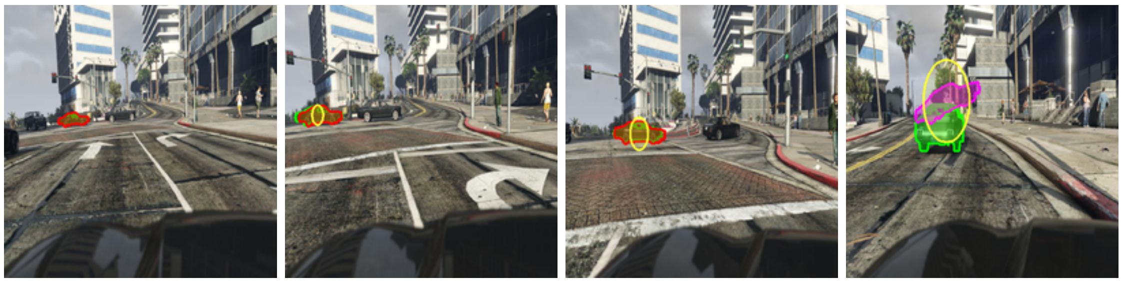


(e) $\varnothing\Phi_{\mathrm{IoU}} = 0.77$, TPR = 0.91, #frames = 219

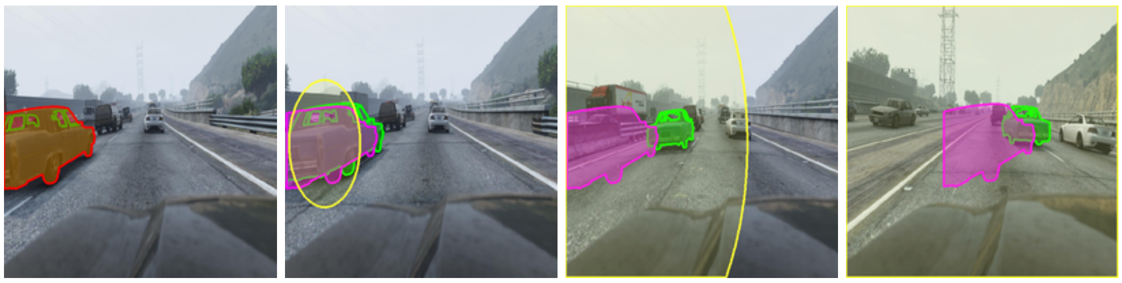
Figure 9.11: A collection of results from `gen_level_sm`. The tracker is able to track cars in various settings and in long-term sequences with more than 1000 frames (b).



(a) $\varnothing\Phi_{\text{IoU}} = 0.42$, TPR= 0.54, #frames = 69



(b) $\varnothing\Phi_{\text{IoU}} = 0.19$, TPR= 0.20, #frames = 288



(c) $\varnothing\Phi_{\text{IoU}} = 0.11$, TPR= 0.02, #frames = 412

Figure 9.12: Three example sequences in which the shape model tracker has difficulties are shown. In general, the shape model tracker struggles in sequences with strong viewpoint changes. This is illustrated in (a) and (b), where the cars turn by at least 90 degrees and are visible from a different viewpoint. Similarly, the shape model has difficulties with the strong perspective viewpoint changes shown in (c).

on average. As a consequence, they generally have a lower tracker confidence and thus a larger average search region.

The typical failure modes of the shape model tracking include:

- The shape model has difficulties handling fast perspective changes of the object. Although the model update can adapt the model to small perspective changes that accumulate over time, it has difficulties in sequences with fast or extreme perspective or view-point change. Three example sequences are shown in Fig. 9.12.
- Although the shape model tracker is robust to non-linear illumination changes in general, it struggles in sequences with a very low contrast and a very small template size. Two example sequences are displayed in Fig. 9.8.

Table 9.5: Overview of the average Φ_{IoU}, the TPR, the TNR, and the average runtime of the shape model and state-of-the-art trackers are displayed.

	$\varnothing\Phi_{\mathrm{IoU}}$	TPR	TNR	time per frame
`gen_level_sm`	0.417	0.527	0.0	29.1 ms
`gen_level_sm_pred_5`	0.390	0.501	0.14	29.1 ms
`box-axis-aligned`	0.779	0.966	1.0	-
ECO [61]	**0.444**	**0.555**	0.0	1389.7 ms
CSR [138]	0.268	0.272	0.0	254.0 ms
STAPLE [16]	0.226	0.267	0.0	67.4 ms
KCF [97]	0.142	0.138	0.0	21.3 ms
DSST [62]	0.211	0.254	0.0	31.8 ms
CCOT [66]	0.410	0.486	0.0	3482.5 ms
DFST [180]	0.122	0.019	0.0	141.7 ms
DPCF [1]	0.089	0.014	0.0	267.4 ms

9.2.2 Comparison to the State of the Art

The comparison against the state-of-the-art trackers was restricted to open-source trackers that showed good performance in recent benchmarks and are nearly real-time capable. We selected the basic Kernelized Correlation Filter (KCF) [97] tracker since it is extremely fast and was a top ranked tracker in the VOT2014 challenge. However, it does not estimate the scale of the object. The Discriminative Scale Space Tracker (DSST) [62] is an extension of KCF that can handle scale changes and generally outperforms KCF [97]. A robust extension of DSST that utilizes complementary feature cues for tracking is STAPLE [16]. The approach uses a HOG-based correlation tracker and a histogram-based template matching method to track objects efficiently in real-time. Furthermore, we include the Discriminative Correlation Filter with Channel and Spatial Reliability (CSR) tracker [138]. The tracker is also based on a correlation tracker, but specific care is taken in the update stage of the tracker. A spatial reliability map is computed to guide the update of the correlation filters to the part of the object that is suitable for tracking. Two further extensions of correlation trackers that only have a minor impact on the tracking speed are DFST [180] and DPCF [1]. DFST extends a correlation filter tracker with color information. In contrast, Akin *et al.* [1] improve the performance of a correlation tracker to deformable objects by adding coupled global and local correlation filter.

For completeness, two of the top-performing deep-learning-based trackers were also added. The first one is the Continuous Convolution Filter (CCOT) from Danelljan *et al.* [66]. It was the best-performing tracker in the VOT2016 challenge and a top performer of the challenge VOT 2017. We also added the extension of the CCOT tracker, the Efficient Convolution Operators for Tracking (ECO) [61] tracker, which was proposed in 2017. It was the top tracker in the VOT 2017 challenge. Both methods only require a single forward pass of a CNN and are relatively efficient for a deep-learning-based approach. If a high performance graphics card is available, they are near real-time capable and run at

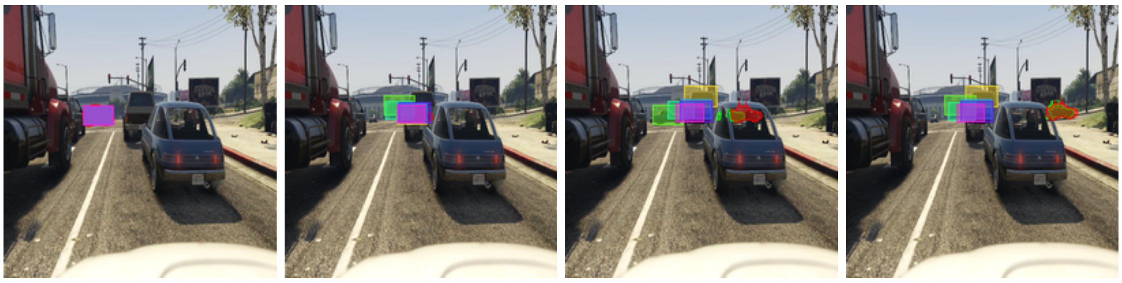
8 fps (ECO) and 3 fps (CCOT) on an NVIDIA GeForce GT 730M. However, to enable a fair comparison of the computational overhead, all methods are restricted to using only the CPU in the following experiments.

In a first experiment, we evaluated the $\varnothing\Phi_{\mathrm{IoU}}$, the TPR, the TNR, and the average runtime per frame for the above trackers and for `gen_level_sm`, the best performing shape model configuration from Section 9.2.1. Furthermore, the results of `gen_level_sm_pred_5` were added since it is one of the shape model configurations that is able to detect the absence of the object and has a reasonable balance between the TPR and the TNR. The theoretical upper bound for an axis-aligned tracker `box-axis-aligned`, which is presented in more detail in Section 4.2, is also added for reference. An overview of the results is shown in Table 9.5.

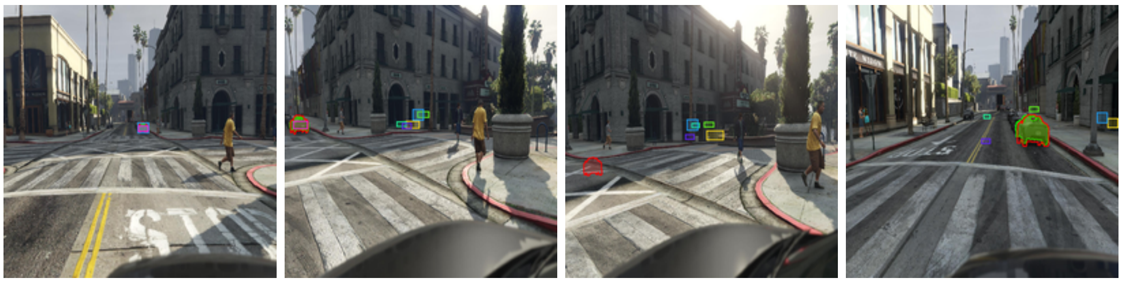
The shape model tracker `gen_level_sm` is only outperformed in terms of accuracy and robustness by the deep-learning-based method ECO. The CCOT tracker is both marginally less accurate and less robust than the shape model tracker `gen_level_sm`. However, both of the deep-learning methods require much more computing resources and are far from real-time when executed on the CPU. While ECO requires well over one second per frame, CCOT is significantly slower with an average of over 3 seconds per frame. As expected, the shallow methods are significantly faster. However, all of the tested methods are also much weaker in terms of accuracy and robustness than the shape model based trackers. The best performing non-deep-learning-based tracker is the CSR tracker. With an $\varnothing\Phi_{\mathrm{IoU}}$ of 0.268 it is much weaker than the shape model tracker ($\varnothing\Phi_{\mathrm{IoU}} = 0.417$). Furthermore, it is almost ten times slower than the shape model trackers.

The methods that are comparably fast to the shape model tracker are the correlation filter-based trackers STAPLE, DSST, and KCF. They perform worse than the CSR tracker both in terms of accuracy and robustness. They all are also clearly outperformed by both of the shape model trackers. In addition, `gen_level_sm_pred_5` is able to detect the absence of the object in 14% of the frames. Since many objects have a transition phase in which they are slowly occluded or leave the field of view, the optimal TNR of 1.0 is extremely difficult to obtain in practice.

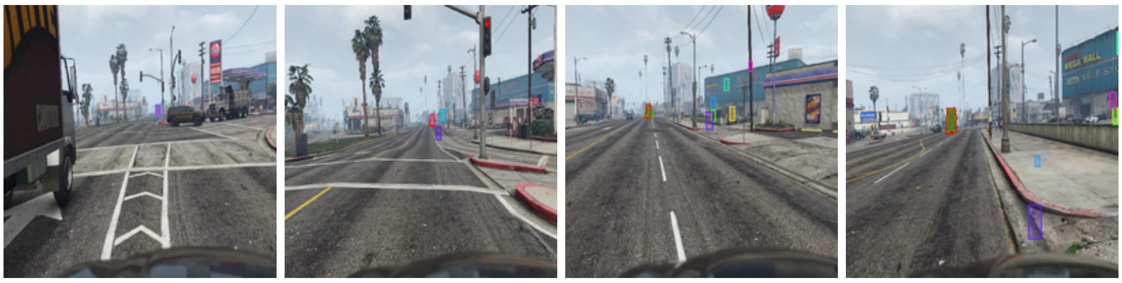
In general, the shape model performs better than all other methods in frames in which the object is either occluded for a short period of time or disappears from the frame briefly. A few examples are shown in Fig. 9.13. In Fig. 9.13 (a), all other methods fail when the car is occluded early in the sequence. The motion model and the prediction of the shape model allows the shape model tracker to re-detect the car and obtain a much higher average Φ_{IoU} than the other methods. Similarly, in Fig. 9.13 (b), the car disappears from the frame in the middle of the sequence. Since none of the other methods have a failure mode detection, they are unable to recover when the car reappears. In Fig. 9.13 (c), the other methods struggle with the very small initialization size of the truck. Since the lighting and contrast is reasonable, the shape model correctly detects the scale change and creates multiple new shape models as the object grows in the image. It is able to handle the extreme scale change and has a high $\varnothing\Phi_{\mathrm{IoU}}$. In the Fig. 9.13 (d), the camera shakes quite strongly in the middle of the sequence and many of the other methods fail.



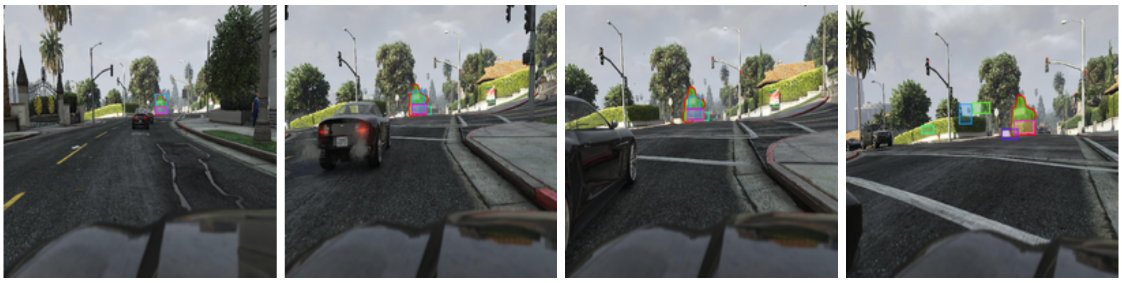
(a) `gen_level_sm` $\varnothing\Phi_{\text{IoU}} = 0.52$, DSST $\varnothing\Phi_{\text{IoU}} = 0.104$, #frames = 69



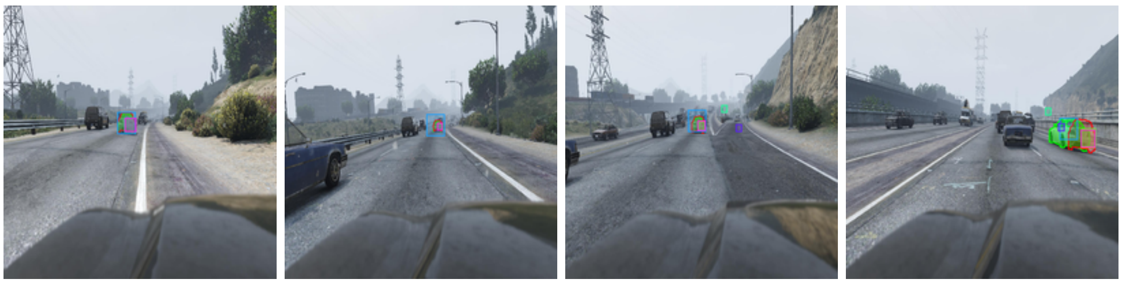
(b) `gen_level_sm` $\varnothing\Phi_{\text{IoU}} = 0.808$, CCOT $\varnothing\Phi_{\text{IoU}} = 0.176$, #frames = 96



(c) `gen_level_sm` $\varnothing\Phi_{\text{IoU}} = 0.802$, CCOT $\varnothing\Phi_{\text{IoU}} = 0.247$, #frames = 316



(d) `gen_level_sm` $\varnothing\Phi_{\text{IoU}} = 0.810$, ECO $\varnothing\Phi_{\text{IoU}} = 0.393$, #frames = 294



(e) `gen_level_sm` $\varnothing\Phi_{\text{IoU}} = 0.670$, KCF $\varnothing\Phi_{\text{IoU}} = 0.374$, #frames = 729

— ECO	— CSR	— STAPLE	— KCF
— DSST	— CCOT	— `gen_level_sm`	

Figure 9.13: $\varnothing\Phi_{\text{IoU}}$ is displayed `gen_level_sm` and second best performing method.

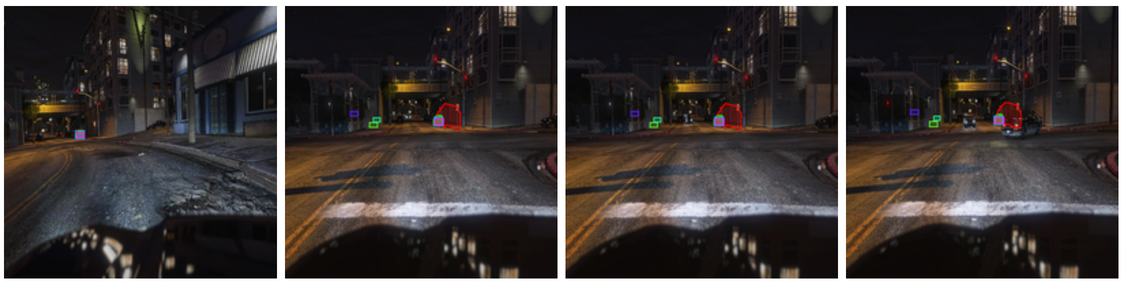
Although the shape model tracker also loses the model for a short period of time, it is able to detect the object absence and re-detect the object. As a consequence, it clearly outperforms the other methods in terms of accuracy. The sequence in Fig. 9.13 (e) is relatively long and the other trackers start to drift off target. The shape model tracker is much more stable and produces a much higher average Φ_{IoU}.

However, there are also cases in which the shape model trackers are clearly outperformed by the deep-learning methods. A few examples are shown in Fig. 9.14. In Fig. 9.14 (a), the example from Fig. 9.8 is shown in more detail. The low contrast causes the tracker initialization to create a degenerated shape model that only consists of a single line. The object is quickly lost and the scale is estimated completely wrongly. The deep-learning based ECO tracker is much stronger and successfully tracks the car. Similarly, in Fig. 9.14 (b), the object is initialized at a very small size in the rain. The resulting shape model tracker is not very descriptive and the shape model finds the object in various different background edges. The deep-learning-based trackers cope with this much more robustly. In Fig. 9.14 (c), an example is shown where the perspective of the initialization is an issue. In the first few frames, the car moves from the right to the left and the perspective on the rear end of the car changes. The shape model tracker does not adapt to the new viewpoint fast enough and fails. Again, the deep-learning trackers generalize better in this sequence and outperform the shape model tracker.

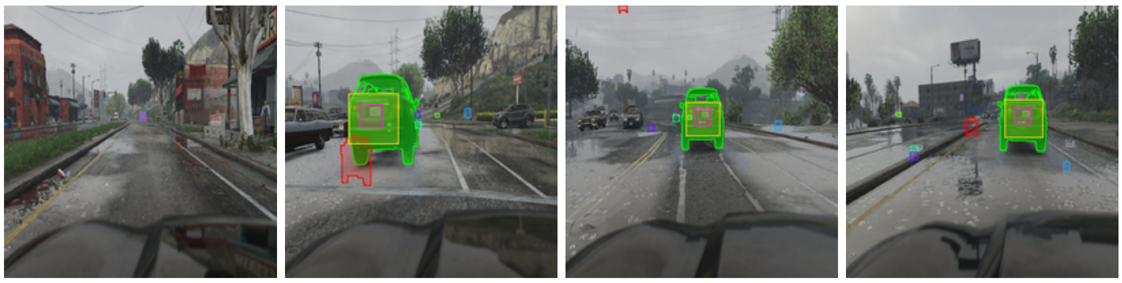
To validate the robustness of the deep-learning based approaches to the contrast in the initialization step, the same experiment as in Fig. 9.10 is conducted for the ECO tracker. The sequences were split into four lightings and the average Φ_{IoU} and TPR are computed. The results are shown in Fig. 9.15. As expected, the generalization capabilities of the deep-learning approaches are better than those of the shape model tracker. Although the average overlap scores are lower in good lighting conditions than for the shape model tracker (see Fig. 9.10), the scores are more stable across the different lightings.

Bounding Box Tracking All of the presented Φ_{IoU} scores are computed between the tracker proposal and the pixel-precise representation of the ground truth. Therefore, it is reasonable to ask: how well are the axis-aligned box-based trackers capable of performing for pixel-precise ground truths? To validate this, the scores of the best possible axis-aligned tracker `box-axis-aligned` are computed and shown in Table 9.5. Interestingly, even the `box-axis-aligned` tracker is not able to obtain a TPR of 1.0. There are pixel-precise ground truths that have a Φ_{IoU} that is smaller than 0.5 when they are approximated by a box. To create a TPR that is more descriptive for the trackers that arerestricted to boxes, we compute the TPR using the threshold of 0.5 for Φ_{rIoU} instead of Φ_{IoU} to identify a true positive. The measure is denoted as $\mathrm{TPR}_{\Phi_{\mathrm{rIoU}}}$ and correlates very strongly with the standard TPR. However, the absolute value is more descriptive for trackers that are restricted to bounding boxes since it ranges from 0.0 to 1.0.

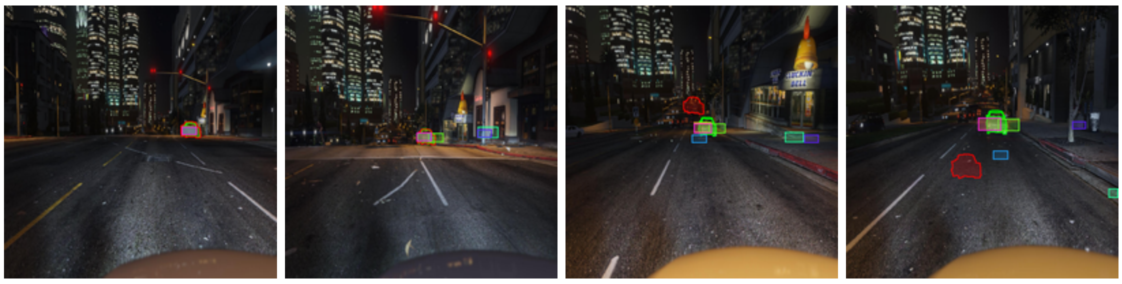
To enable a comparison to the shape model tracker, the axis-aligned bounding box of the tracker output is computed and the tracker is denoted as `box_gen_level_sm`. The respective Φ_{rIoU} values of all the box trackers are shown in Table 9.6. The results for the DFST and the DPCF tracker are omitted since they performed poorly in the above



(a) `gen_level_sm` $\varnothing\Phi_{\mathrm{IoU}} = 0.143$, ECO $\varnothing\Phi_{\mathrm{IoU}} = 0.618$, #frames = 434



(b) `gen_level_sm` $\varnothing\Phi_{\mathrm{IoU}} = 0.08$, ECO $\varnothing\Phi_{\mathrm{IoU}} = 0.490$, #frames = 520



(c) `gen_level_sm` $\varnothing\Phi_{\mathrm{IoU}} = 0.315$, CCOT $\varnothing\Phi_{\mathrm{IoU}} = 0.408$, #frames = 74

— ECO	— CSR	— STAPLE	— KCF
— DSST	— CCOT	— `gen_level_sm`	

Figure 9.14: Sequences in which `gen_level_sm` is outperformed by one of the deep-learning-based methods are displayed. The $\varnothing\Phi_{\mathrm{IoU}}$ is displayed for the `gen_level_sm` and the best performing method.

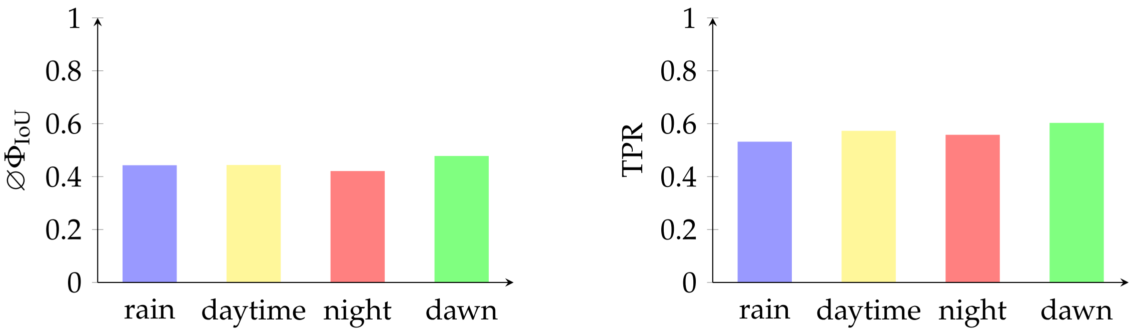


Figure 9.15: The sequences are split into four disjoint lightings (rain, daytime, night, and dawn) and the $\varnothing\Phi_{\mathrm{IoU}}$ values are displayed for the `eco` tracker (right). Although the deep-learning-based `eco` tracker performs weaker in rain and daytime, it is generally more robust to the different lightings of the sequences (see Fig. 9.10).

Table 9.6: An overview of the average Φ_{IoU}, the average Φ_{rIoU}, and the $\mathrm{TPR}_{\Phi_{\mathrm{rIoU}}}$.

	$\varnothing\Phi_{\mathrm{IoU}}$	$\varnothing\Phi_{\mathrm{rIoU}}$	$\mathrm{TPR}_{\Phi_{\mathrm{rIoU}}}$
`box_gen_level_sm`	0.351	0.459	1.0
ECO [61]	**0.444**	**0.580**	0.678
CSR [138]	0.268	0.363	0.380
STAPLE [16]	0.226	0.313	0.347
KCF [97]	0.142	0.209	0.212
DSST [62]	0.211	0.295	0.320
CCOT [66]	0.410	0.539	0.629

experiments. As expected, the difference between the deep-learning-based trackers and the shape model tracker are more distinct. The deep-learning approaches are able to clearly outperform the shape model tracker that is approximated by boxes. However, the `box_gen_level_sm` tracker performs significantly better than all shallow trackers both in terms of accuracy and precision.

In general, the Φ_{rIoU}, indicates that all box based trackers still have a significant room for improvement. They all have accuracy and robustness scores that are well below the optimal 1.0. This agrees to the observations from Fig. 9.13 and Fig. 9.14 in which no single tracker was able to solve all sequences.

Robustness to Scale Change The results from Fig. 9.13 and Fig. 9.14 indicate that all of the tested methods seem to struggle with extreme scale changes. Especially the methods that are not based on the shape model fail in sequences with significant scale change (e.g. Fig. 9.13 (c)). In a further experiment, the scale score from (4.6) is computed for all trackers. As mentioned in Section 4.4, the scale score is calculated without reinitialization of the tracker when it fails. Furthermore, frames where the tracker has failed completely (hence $\Phi_{\mathrm{IoU}} = 0$) are ignored for the computation. This makes the scale score less dependent on the accuracy of the tracker. The scale scores are shown in Table 9.7. As above, the results for the DFST and the DPCF tracker are omitted since they performed poorly in the first experiments.

As expected, the shape model methods have a higher scale score than other approaches. They clearly outperform the other methods by a large margin. Interestingly, the baseline shape model `baseline_sm` has the strongest scale score. The baseline shape model does not reinitialize the object for large scale changes. The reinitialization assumes the shape model was detected very precisely in the prior frame. Hence, it may introduce an ever so slight drift. As a consequence, the baseline shape model is very accurate when the object does not change its scale too strongly. However, as shown above, without the reinitialization, the shape model performs much poorer in terms of accuracy and robustness (see Table 9.1).

In general, the theoretical optimum of the scale score is 1.0. However, for most of the tested trackers, it is well below 0.5. The results correspond to the observations in Section 4.5 and Böttger *et al*. [23]. Here, it was observed that the current state-of-the-art

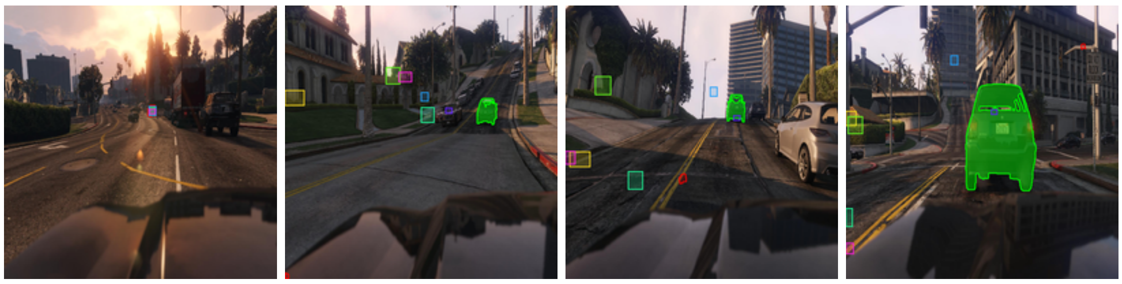
Table 9.7: Overview of the scale score s for the shape model and state-of-the-art methods. The KCF has a scale score of 0.0 size it does not estimate the scale.

	Scale score s
`baseline_sm`	0.48
`gen_level_sm`	**0.36**
ECO [61]	0.28
CSR [138]	0.26
STAPLE [16]	0.26
KCF [97]	0.0
DSST [62]	0.24
CCOT [66]	0.24

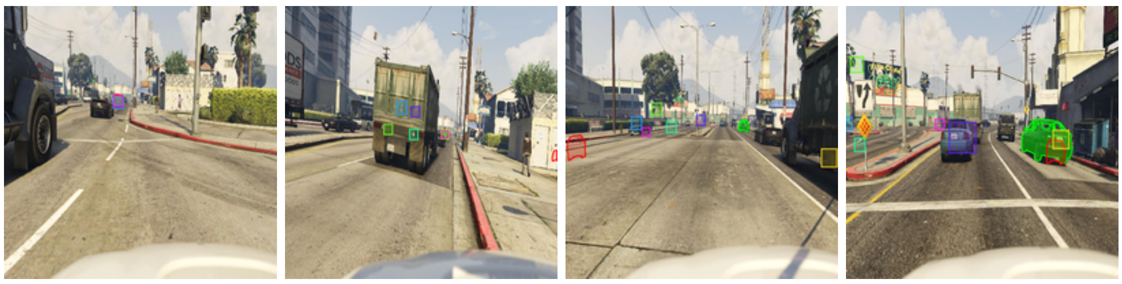
in tracking generally struggles in sequences with strong scale change in the VOT 2016 [226] and DAVIS [170] datasets. Although the shape model appears to be significantly more robust than the other methods, the large gap indicates that also the shape model has room for improvement.

Challenges In Fig. 9.16, examples are shown where all trackers fail early and have very low accuracy scores. In Fig. 9.16 (a), the car is initialized at a very small scale and makes a turn into the side street early in the sequence. None of the trackers is able to track the car while it turns and the `gen_level_sm` cannot re-detect the car since the scale and appearance changed considerably during the turn. In Fig. 9.16 (b), there is a full occlusion by the truck in the first third of the sequence. When the car reappears from behind the truck, the viewpoint has changed significantly. None of the trackers is able to cope with this. Similarly, in Fig. 9.16 (c), the viewpoint on car changes by 180° and none of the trackers is robust. The average Φ_{IoU} values of `gen_level_sm` are still relatively high since it is very accurate in the first half of the sequence. In Fig. 9.16 (d), the shape model tracker fails due to the low contrast and small initialization size. The other methods are all unable to detect the significant scale change and fail early in the sequence.

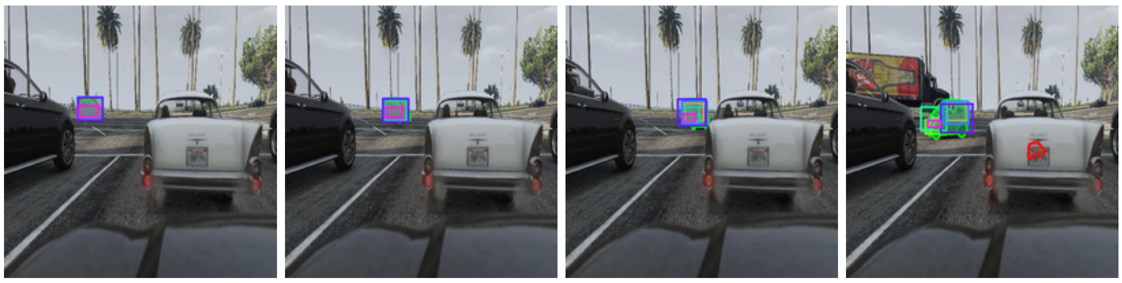
Summary The shape model tracker `gen_level_sm` is able to significantly outperform all of the tested real-time shallow trackers on the PFTD dataset. It performs on par with the top-performing deep-learning-based approaches ECO [61] and CCOT [66]. However, it only requires an average of 29.1 ms per frame and is over 45 times faster than ECO and over 110 times faster than the CCOT. Especially the failure mode detection and the re-detection capabilities allow the tracker to tackle sequences where all of the other methods fail. However, it has difficulties in sequences with very low contrast and in which the object is initialized at a very small size. In general, the average accuracy and robustness scores of all trackers indicate that the PFTD dataset is very challenging for the current state of the art. The results show significant room for improvement and no single tracker is able to tackle more than half of the sequences.



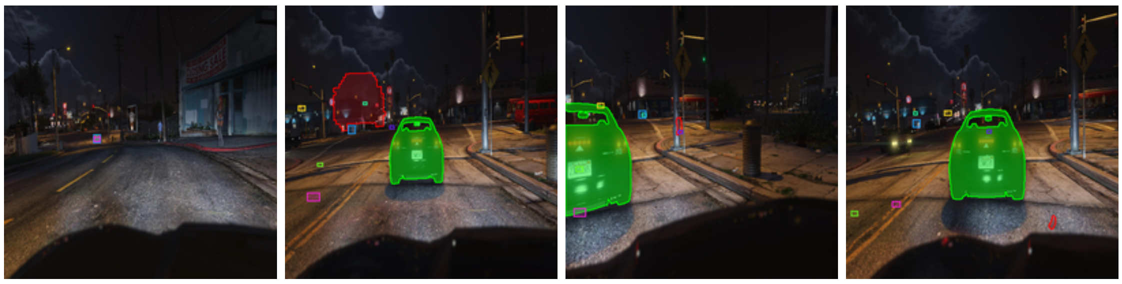
(a) `gen_level_sm` $\varnothing\Phi_{\text{IoU}}$ = 0.05 , ECO $\varnothing\Phi_{\text{IoU}}$ = 0.045 , #frames = 420



(b) `gen_level_sm` $\varnothing\Phi_{\text{IoU}}$ = 0.179 , ECO $\varnothing\Phi_{\text{IoU}}$ = 0.254 , #frames = 170



(c) `gen_level_sm` $\varnothing\Phi_{\text{IoU}}$ = 0.416 , STAPLE $\varnothing\Phi_{\text{IoU}}$ = 0.220 , #frames = 68



(d) `gen_level_sm` $\varnothing\Phi_{\text{IoU}}$ = 0.144 , CCOT $\varnothing\Phi_{\text{IoU}}$ = 0.023 , #frames = 504

— ECO	— CSR	— STAPLE	— KCF
— DSST	— CCOT	— `gen_level_sm`	

Figure 9.16: Challenging Sequences in which `gen_level_sm` and the other methods perform poorly are displayed. The $\varnothing\Phi_{\text{IoU}}$ is displayed for the `gen_level_sm` and the best performing other method.

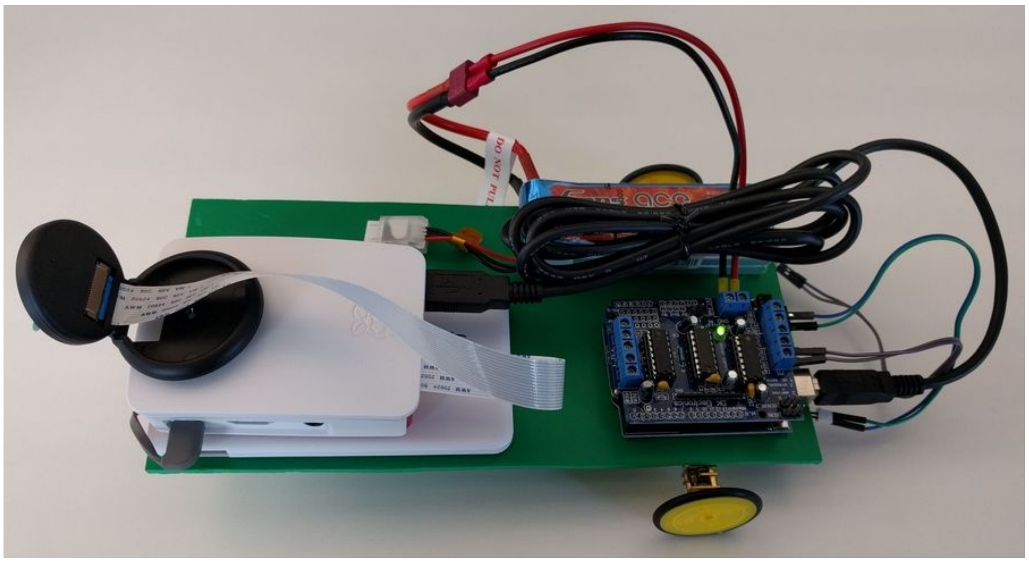


Figure 9.17: The setup for the prototypical auto-follow application is displayed. A Raspberry Pi 3 Model B is mounted on a rack with three wheels. The two rear wheels can accelerate independently from each other and enable the rack to turn corners. The control of the wheels is taken care of by an Arduino Uno SMD R3. The construction is denoted as `AutoFollow`.

9.3 Experiments: Real-time Tracking on a Raspberry Pi

To validate the real-time capabilities of the shape model tracker, further experiments are conduced on a smaller computing device. A Raspberry Pi 3 Model B is mounted onto a simple rack that has three wheels. The two rear wheels can turn independently from each other and enable the "vehicle" to turn corners. The front wheel can rotate freely. The control of the wheels is taken care of by an Arduino Uno SMD R3. The tracking is conducted by the Raspberry Pi with the help of a lightweight and energy.efficient camera that outputs 820×616 sized images. The power for the Raspberry Pi and the Arduino is supplied by a 1 000 mAh powerbank. The construction is shown in Fig. 9.17 and denoted as `AutoFollow` in the following.

The computational resources of the tracker are restricted to the capabilities of the Raspberry Pi 3 Model B: 1GB of RAM and an ARM-Cortex-A53 @1,2GHz. As a consequence, many of the trackers that were evaluated above are not real-time capable. The results of the shape model tracker `gen_level_sm` are merely compared to those of a C implementation of the DSST [62] tracker in the following section. Since the DSST tracker was able to clearly outperform the KCF tracker in the above experiments, a comparison to the KCF tracker is omitted. Both trackers have a runtime that is comparable to the shape model tracker. The other shallow trackers CSR [138], STAPLE [16], DPFT [180], and DPCF [1] are computationally too demanding for the Raspberry Pi 3. Similarly, the deep-learning-based approaches ECO [61] and CCOT [66] are significantly too slow and do not run at all on devices with so little memory.

To validate the general idea, the task is simplified as much as possible in a first experiment. For this, a visual marker is attached on to the back of a car to ensure

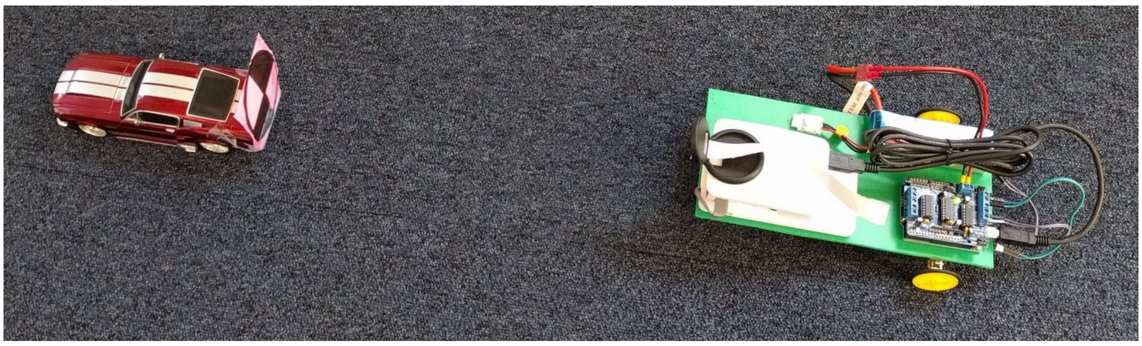


Figure 9.18: The car is setup to follow a remote-controlled car.

that tracking is as simple as possible. The `AutoFollow` is placed behind the remote-controlled car and the tracker determines the position of the object in each frame. From the movement of the object in the image, the motion of the `AutoFollow` vehicle can be computed. If the object becomes smaller, the vehicle must go faster. If the object is left or right of the image center, the car must turn left or right, respectively. The setup is displayed in Fig. 9.18.

The tracking results for 250 frames of the DSST and `gen_level_sm` tracker are displayed in Fig. 9.19. Although the camera is generally capable of around $15 - 20$ fps, writing the images to disk reduced the frame rate to around 8 fps in this setup. As shown, the shape model tracker is generally very robust and can cope with the moderate scale and view-point changes without a problem. Note that even after 250 frames, the shape model tracker has not drifted at all from the target. However, the DSST tracker has difficulties in detecting the scale change correctly and starts to drift off target in the middle of the sequence. The computation times of both approaches are shown in Table 9.8. Both methods are significantly slower on the Raspberry Pi than on the desktop PC. However, with around 100 ms, both methods are capable of locating the object in real-time. [1]

In general, one of the main advantages of using the shape model tracker in the above setting is not only its robustness and accuracy, but the fact that it can inherently be used to detect the object in the first frame. In the above setting, a model of the car was generated offline and stored on the `AutoFollow` vehicle. In the first frame, the search region is initialized as the whole image and the tracking can begin. The initialization of DSST tracker was performed in a similar fashion. In the first frame, the shape model was used to find the initial position of the `AutoFollow` vehicle and in the subsequent frames the localization was performed by the DSST tracker.

9.4 Experiments: Miscellaneous Applications

To emphasize the universal applicability of the shape model tracker, this section presents qualitative results of the tracker for a number of sequences in common tracking datasets.

[1] The ARM-Cortex-A53 has four cores. Hence, the image acquisition and the tracking can be computed in parallel.

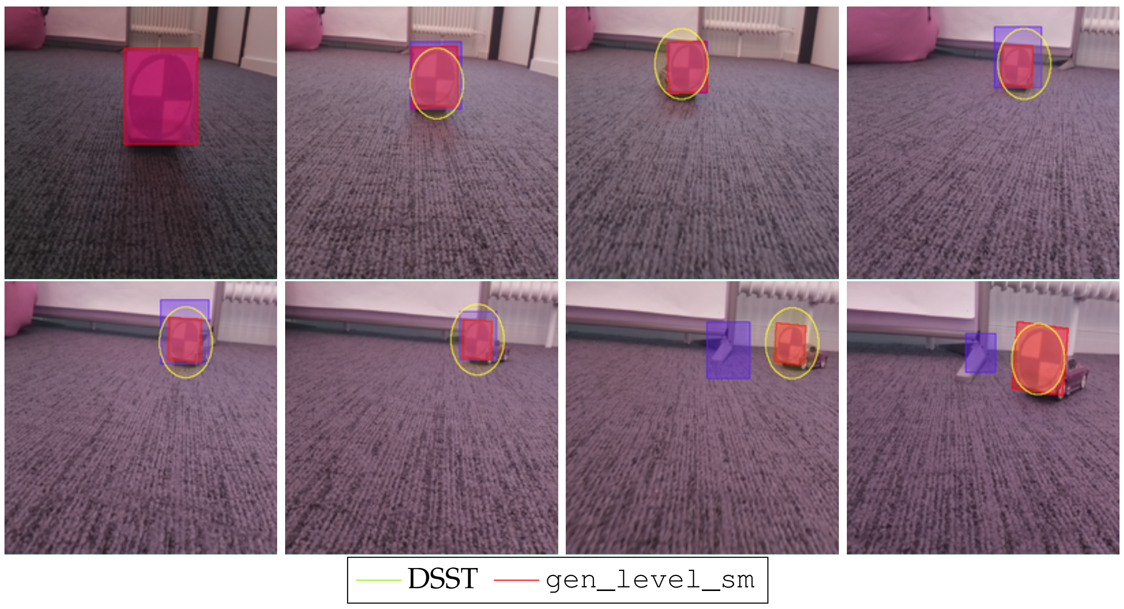


Figure 9.19: The `AutoFollow` vehicle is setup to follow the remote-controlled car from Fig. 9.18. The results for both the the shape model tracker and the correlation-filter tracker are displayed.

Table 9.8: Overview of the computation time on the Raspberry Pi and the desktop PC. Both the correlation tracker and the shape model are significantly slower.

	ARM-Cortex-A53 @1,2GHz	Intel i7-4810 CPU @2.8GHz
`gen_level_sm`	112.7 ms	13.6 ms
DSST [62]	106.3 ms	19.7 ms

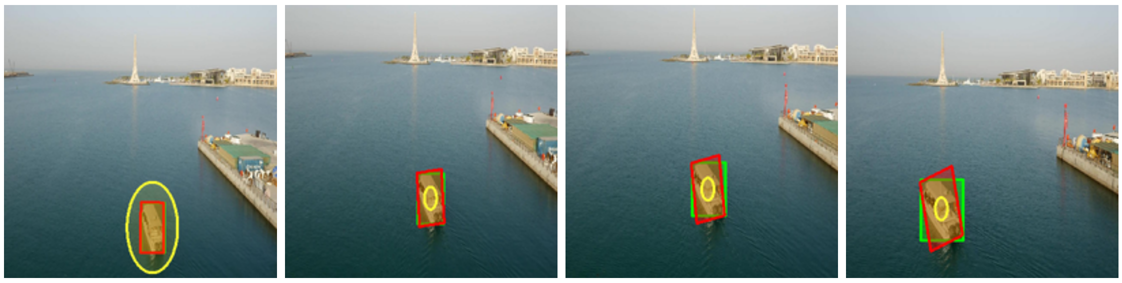
For each dataset, a number of example sequences and the respective accuracy and robustness scores are presented. To get a fair impression of the strengths and weaknesses, sequences in which the shape model tracker performs well and sequences where it struggles are presented. The respective datasets are presented in more detail in Chapter 5.

UAV123 In the UAV123 [153] dataset, approximately half of the objects are rigid objects such as cars, trucks, or boats. The other half of the objects are deformable and include pedestrians, wakeboarders, and birds (see Fig. 3.3 for some examples). The sequences are all acquired at 30 fps from a UAV that flies at different heights.

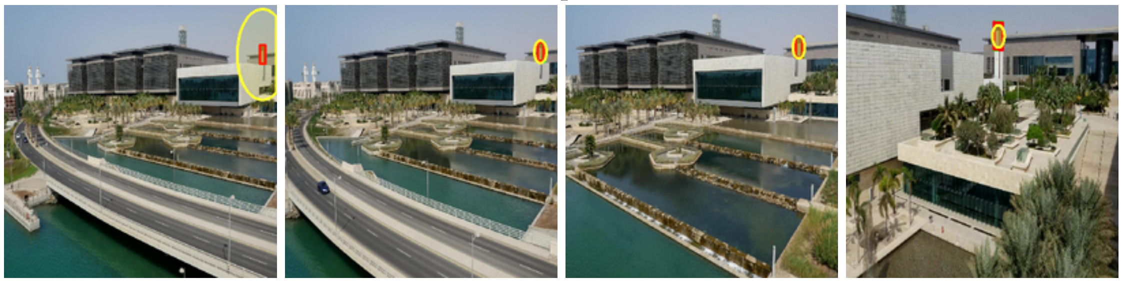
As discussed above, the shape model tracker is generally restricted to roughly rigid objects. Furthermore, it assumes that the viewpoint does not change too much or too fast. As a consequence, the shape model tracker cannot be used to solve all sequences in the UAV123 dataset. However, for the sequences that contain rigid objects and have a restricted amount of viewpoint change, the shape model tracker performs extremely well. Especially in the sequences that track buildings or vehicles from a distance, the shape model has very high accuracy and robustness scores. In Fig. 9.20, a subset of these sequences is displayed. The shape model tracker also achieves perfect robustness scores for the sequences were buildings are tracked (e.g., Figs. 9.20 (b) and (c)). Furthermore, the robustness of the tracker against drift allows it to track cars in sequences with well over 1000 frames (e.g., Figs 9.20 (d) and (e)).

However, the shape model tracker struggles in the sequences in which the objects are not rigid or the view point of the object changes considerably. A collection of challenging sequences are displayed in Fig. 9.21. In Fig. 9.21 (a), although the shape model tracker is able to tackle the initial change of perspective point, the tracker eventually loses the boat when the viewpoint changes too strongly. The search region is extended to the whole image, but the boat is not re-detected since it does not appear in a perspective that is known to the tracker. In Figs. 9.21 (b) and (c), pedestrians are tracked. Although the tracker is able to cope with slight deformations, it eventually fails and loses the pedestrians. In general, the shape model tracker is able to cope with a minor amount of deformations. In Fig. 9.20 (d), the shape model loses the pedestrian a number of times. However, since the UAV is relatively far away from the pedestrian, the deformable movement of the legs is not really visible significantly. As a consequence, the shape model tracker successfully re-detects the pedestrian numerous times and is relatively successful on average.

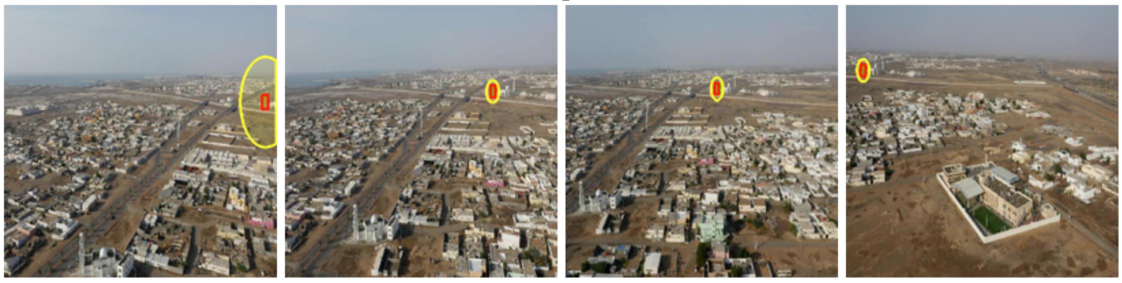
The sequences in the UAV123 dataset are all acquired at 30 fps. Hence, the average runtimes of the successful sequences displayed in Fig. 9.20 are very promising. However, in the challenging sequences, the runtime of the shape model tracker is significantly higher. Since the tracker is less successful, the search regions are larger on average in difficult sequences. As a consequence, the runtime increases and the tracker cannot process each incoming frame at runtime. The difference in the runtime is evident between the sequence in Fig. 9.21 (c) and in the sequence n Fig. 9.21 (d). Although both templates have a similar size and complexity, the tracker is much faster when it is more successful. Since the re-detection of the shape model tracker is relatively robust, it is generally not a



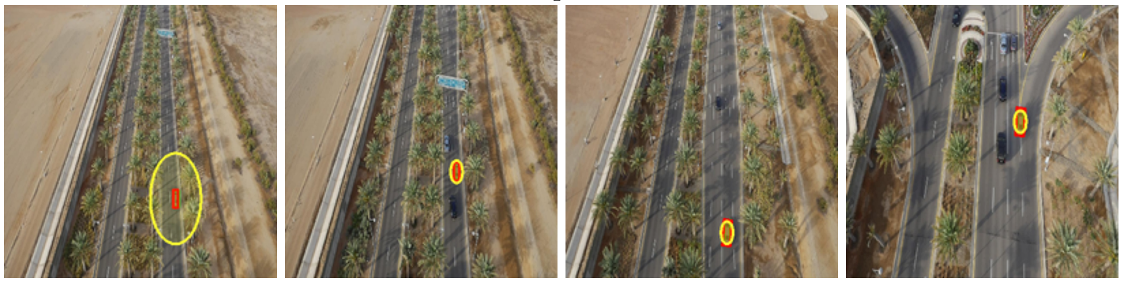
(a) $\varnothing\Phi_{\text{IoU}}$ = 0.788, TPR = 1.0, ms per frame = 33.2 , #frames = 799



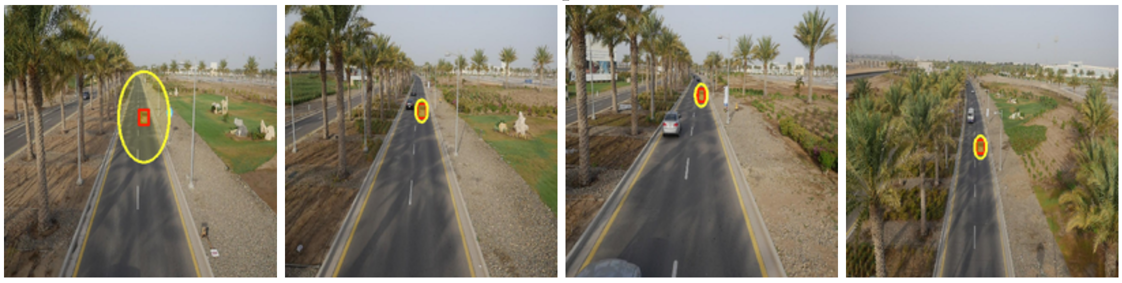
(b) $\varnothing\Phi_{\text{IoU}}$ = 0.875, TPR = 1.0, ms per frame = 27.0 , #frames = 577



(c) $\varnothing\Phi_{\text{IoU}}$ = 0.73, TPR = 1.0, ms per frame = 11.2 , #frames = 829

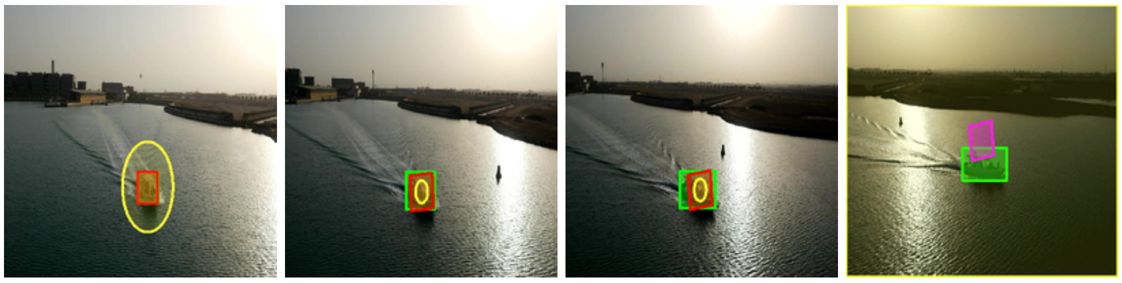


(d) $\varnothing\Phi_{\text{IoU}}$ = 0.76, TPR = 0.985, ms per frame = 35.1 , #frames = 1345

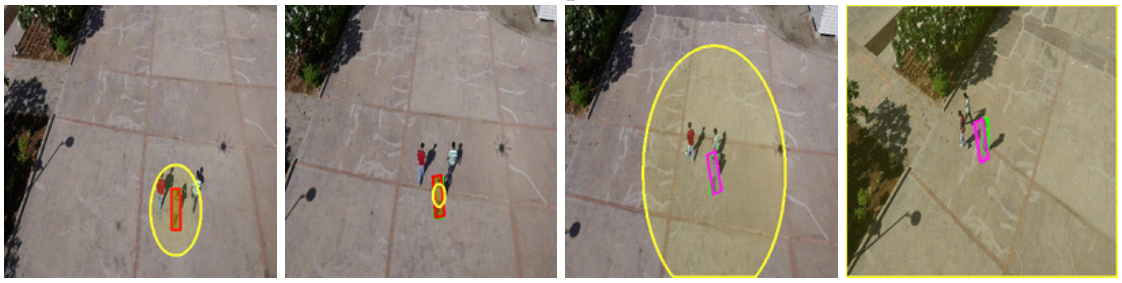


(e) $\varnothing\Phi_{\text{IoU}}$ = 0.85, TPR = 0.998, ms per frame = 20.6 , #frames = 1405

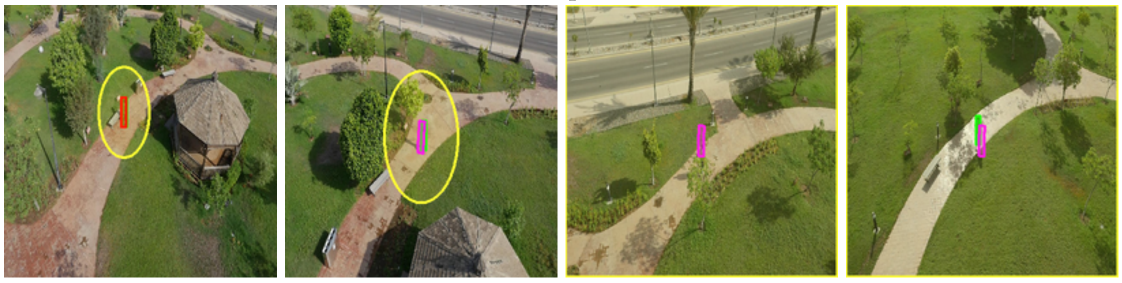
Figure 9.20: The $\varnothing\Phi_{\text{IoU}}$, TPR, and the average runtime are displayed for a collection of sequences in the UAV123 [153] dataset for which the `gen_level_sm` tracker performs very well.



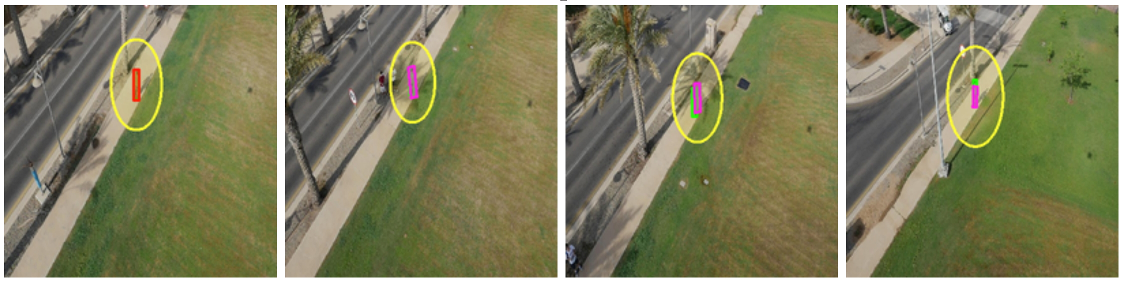
(a) $\varnothing\Phi_{\text{IoU}}$ = 0.404, TPR = 0.517, ms per frame = 160.0 , #frames = 553



(b) $\varnothing\Phi_{\text{IoU}}$ = 0.146, TPR = 0.155, ms per frame = 39.8 , #frames = 1579



(c) $\varnothing\Phi_{\text{IoU}}$ = 0.235, TPR = 0.234, ms per frame = 222.2 , #frames = 1501



(d) $\varnothing\Phi_{\text{IoU}}$ = 0.696, TPR = 0.91, ms per frame = 18.9 , #frames = 1339

Figure 9.21: The $\varnothing\Phi_{\text{IoU}}$, TPR, and the average runtime of `gen_level_sm` are shown for a collection of sequences in the UAV123 [153] dataset that are very challenging for the shape model tracker.

major problem to not run the re-detection of the object at frame rate.

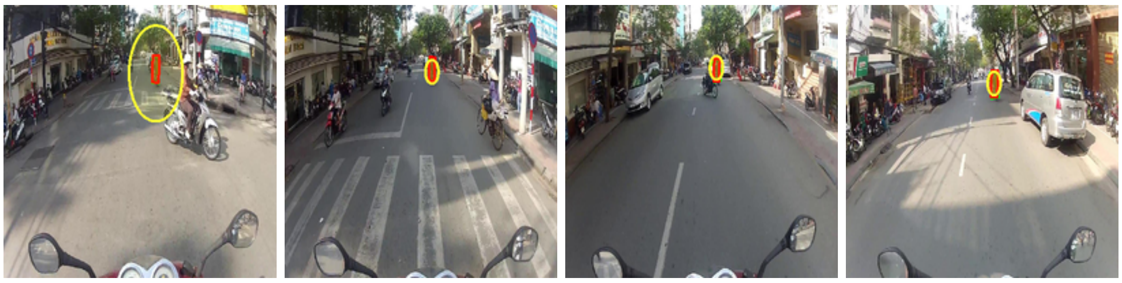
OTB 2015 and VOT 2017 The sequences in the OTB 2015 [239] and VOT 2017 [117] benchmarks do not address any specific single application. Instead, they include sequences from sports videos, dashcam videos of cars, webcam videos of humans, and animal videos. Many of the objects are deformable and change their appearance significantly throughout the sequence. As a consequence, the shape model tracker cannot be used to solve all of the challenging sequences. However, some of the sequences include rigid objects and can be tackled by the shape model tracker. A collection of sequences and the respective accuracy and robustness scores are shown in Fig. 9.22.

As shown in Figs. 9.22(a), (b), and (c), the shape model tracker performs very well on the sequences in the OTB 2015 [239] and VOT 2017 [117] benchmarks that are acquired by a dashcam. The TPR and the average Φ_{IoU} are very high. The images in both benchmarks are mostly 640×480 pixels large. For these small images, the shape model tracker is exceptionally fast and only requires an average of around 18 ms per frame. In Fig. 9.22 (d), the shape model tracker is able to track the box successfully in over 1000 frames. Although the box is occluded a number of times, the tracker re-detects the box when it reappears. In Fig. 9.22 (e), a face is tracked. In this sequence, there is little head movement and the main challenge comes from the camera motion. The shape model tracker is able to detect the head correctly in almost every frame.

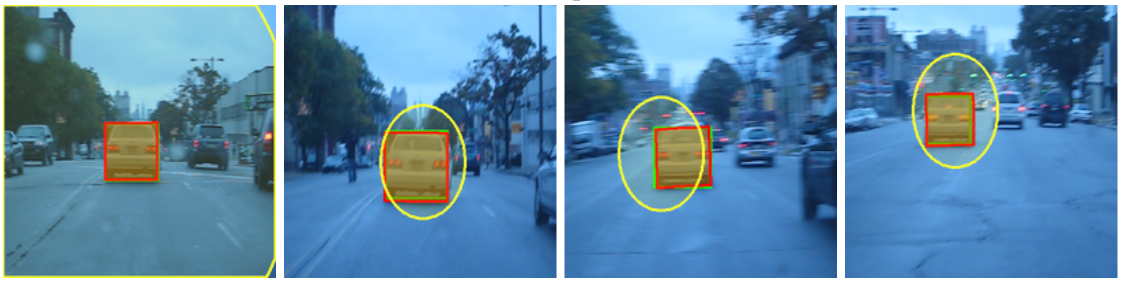
In Fig. 9.23, a number sequences that are very difficult for the shape model tracker are presented. In general, the tracker is not able to tackle many of the sequences in the OTB 2015 [239] and VOT 2017 [117] benchmarks that exhibit deformations (Fig. 9.23 (a)) or large changes in the viewpoint (Fig. 9.23 (b)). In Fig. 9.23 (c), the tracker has to cope with partial occlusions from the windscreen wiper and strong camera motion. Although the tracker is able to cope in the first half of the sequence, it eventually loses the target and fails. Furthermore, in the OTB 2015 [239] benchmark, there are sequences with strong motion blur of the camera. In these sequences, the shape model tracker has difficulties when the motion blur is too strong. However, the tracker is often able to re-detect the object when the motion blur decreases. Two examples are shown in Figs. 9.23 (d) and (e). Here, the average accuracy and robustness scores are not very good, but the tracker is able to track the object most of the time.

9.5 Discussion

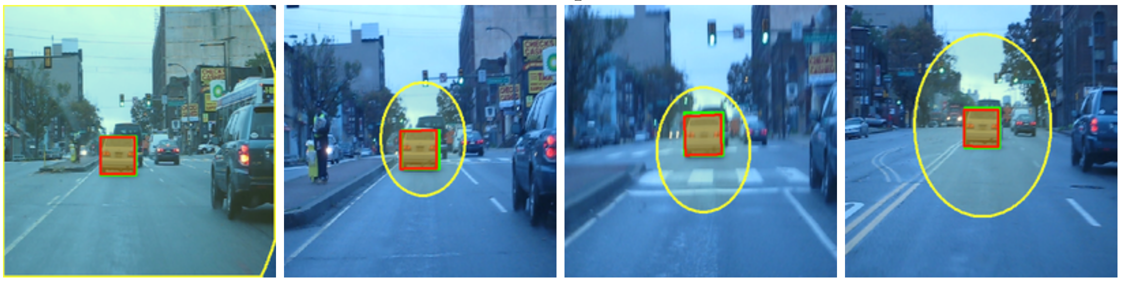
This chapter has presented an extensive evaluation of the shape model tracker. In a first step, the improvements from Chapter 8 were compared to the baseline shape model tracker. The modifications were able to improve the average runtime, robustness, and accuracy by a large margin on the PFTD dataset. Especially the hierarchy of shape models and the dynamic generation of the pyramid levels enhanced the performance. In a second step, the best configuration of the shape model tracker was tested against the current state of the art in real-time object tracking and the top-performing deep-learning trackers. The shape model tracker was able to perform on par with the best deep-learning



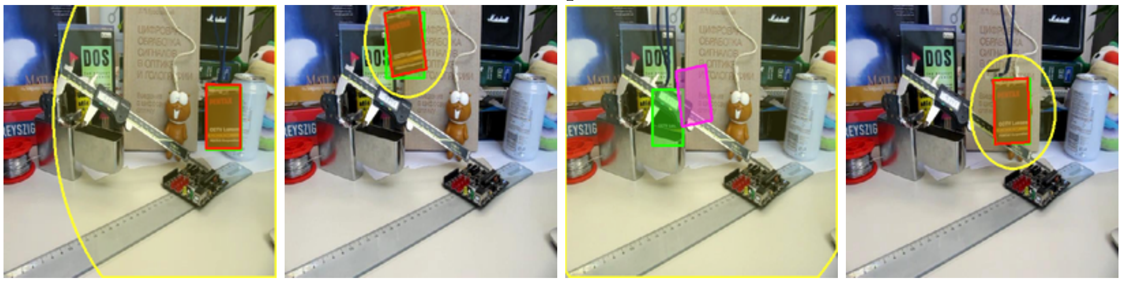
(a) $\varnothing\Phi_{\text{IoU}}$ = 0.811, TPR = 0.974, ms per frame = 19.1 , #frames = 191



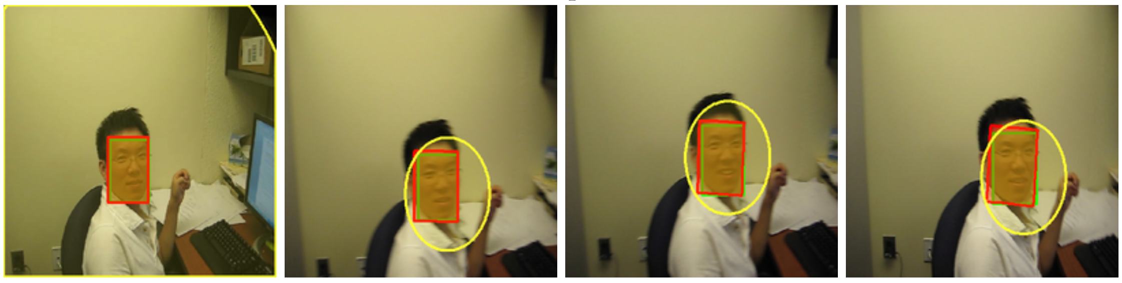
(b) $\varnothing\Phi_{\text{IoU}}$ = 0.875, TPR = 0.994, ms per frame = 17.1 , #frames = 585



(c) $\varnothing\Phi_{\text{IoU}}$ = 0.733, TPR = 0.83, ms per frame = 17.2 , #frames = 357

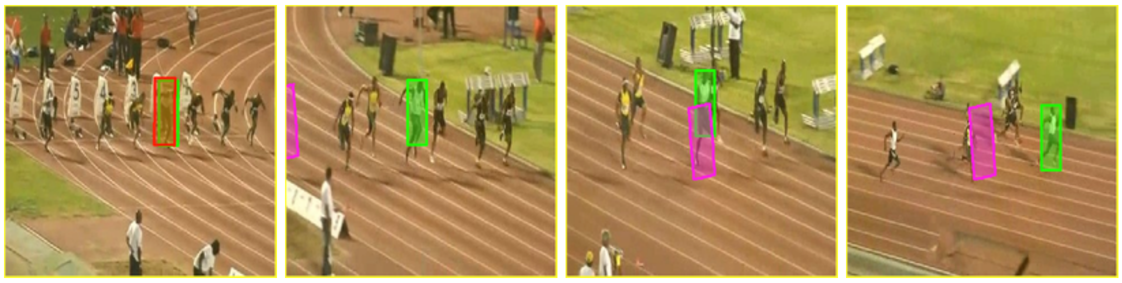


(d) $\varnothing\Phi_{\text{IoU}}$ = 0.705, TPR = 0.886, ms per frame = 12.9 , #frames = 1161

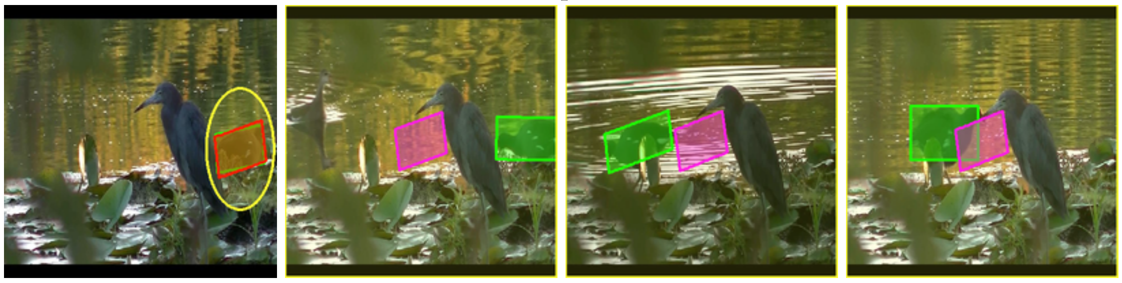


(e) $\varnothing\Phi_{\text{IoU}}$ = 0.820, TPR = 0.97, ms per frame = 19.4 , #frames = 493

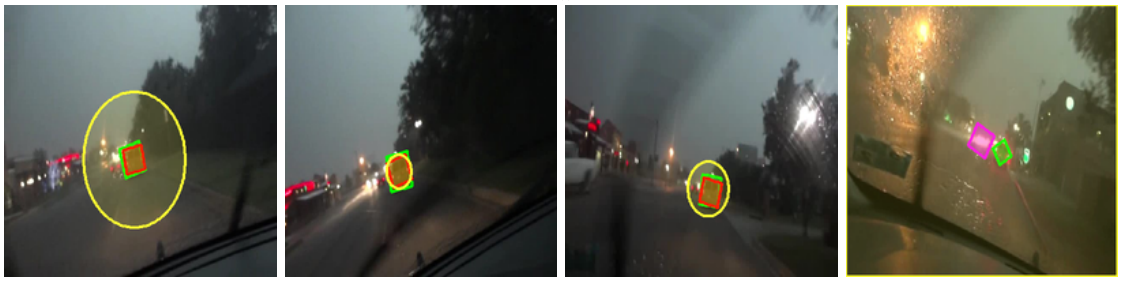
Figure 9.22: The $\varnothing\Phi_{\text{IoU}}$, TPR, and the average runtime are displayed for a collection of sequences from the OTB 2015 [239] and VOT 2017 [117] benchmarks where the `gen_level_sm` tracker performs very well.



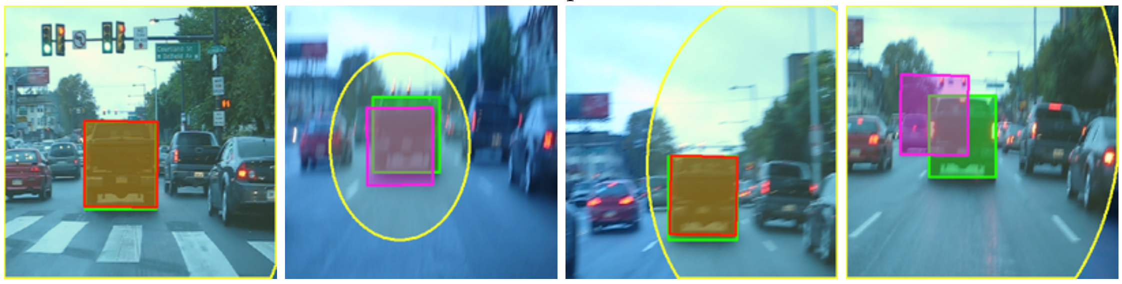
(a) $\varnothing\Phi_{\text{IoU}}$ = 0.03, TPR = 0.01, ms per frame = 65.3 , #frames = 293



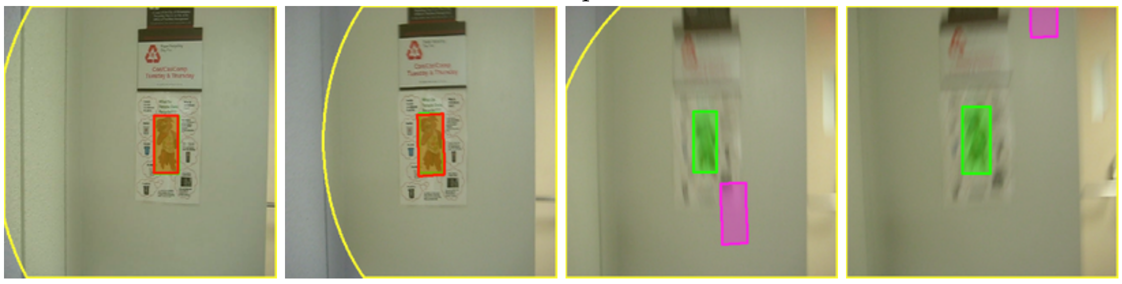
(b) $\varnothing\Phi_{\text{IoU}}$ = 0.12, TPR = 0.03, ms per frame = 15.6 , #frames = 999



(c) $\varnothing\Phi_{\text{IoU}}$ = 0.377, TPR = 0.49, ms per frame = 10.7 , #frames = 341



(d) $\varnothing\Phi_{\text{IoU}}$ = 0.533, TPR = 0.582, ms per frame = 7.3 , #frames = 380



(e) $\varnothing\Phi_{\text{IoU}}$ = 0.367, TPR = 0.38, ms per frame = 29.8 , #frames = 631

Figure 9.23: The $\varnothing\Phi_{\text{IoU}}$, TPR, and the average runtime of `gen_level_sm` are shown for a collection of sequences from the OTB 2015 [239] and VOT 2017 [117] benchmarks. The sequences are challenging for the shape model tracker.

approaches in terms of robustness and accuracy and significantly outperforms them in terms of speed. Furthermore, the shape model tracker was more robust and accurate than all of the tested shallow real-time trackers. In a third step, the efficiency of the tracker was validated on a Raspberry Pi and it was able to track a remote-controlled car successfully at 8 fps. In a final evaluation, the universal applicability of the tracker was shown for a number of different sequences from common object tracking benchmarks. In summary, the experiments show that the tracker is robust, fast, and accurate in a number of different applications and settings.

10 Conclusions

This chapter summarizes the contributions of this work and potential directions for future research in this area.

10.1 Summary

This thesis tackled the challenge of developing robust and accurate tracking algorithms for single-object tracking in real-time. The developed component-tree and shape-model-based tracking algorithms are computationally lightweight, accurate, and robust. In contrast to the majority of existing trackers, the developed methods are not restricted to bounding boxes and are exceptionally fast. Furthermore, as shown in the experiments, they are able to solve different challenges and applications.

The main contributions of this thesis are

- the extension of the standard single-object tracking evaluation protocol to support pixel-precise ground truths and trackers. The new evaluation protocol is able to measure the accuracy of trackers with improved precision and estimates the tracker's capability of detecting scale changes. Furthermore, it enables measuring the quality of pixel-precise trackers and comparing them to trackers that are restricted to bounding-boxes, something that was not possible before;

- a new tracking dataset with pixel-precise ground truth labels. The very precise ground truth labels are created automatically from a photo-realistic synthetic dataset. In general, the dataset is very challenging and includes long sequences with full occlusion and different lighting conditions. The pixel-precise labels allow to evaluate trackers with a new level of precision;

- a new tracker based on component-trees. The pixel-precise tracker utilizes homogeneous regions, which are derived and discussed in detail. In contrast to many existing trackers, the approach is able to track objects that undergo arbitrary deformations. Furthermore, the tracker is computationally lightweight and can be used for 3D object segmentation;

- the homogeneous regions developed for the component-tree tracker are generic and can be used for various applications unrelated to tracking. For example, as shown, they can be used for general image segmentation tasks and help to improve the results of MSER-based OCR systems;
- a fast, robust, and very accurate edge-based tracker. The tracker is robust to illumination changes and occlusion and can track roughly rigid objects in long sequences with virtually no drift. It is extremely efficient and applicable to different applications and sequences. It is able to outperform the current state of the art in real-time tracking and performs on par with the current state-of-the-art deep learning trackers, but is at least 40 times faster.

Together, the contributions cover the complete tracking pipeline, from the tracker evaluation protocol, creating a challenging tracking dataset and benchmark, and developing methods to tackle the new challenges.

10.2 Future Work

While the tracking algorithms presented in this thesis are able to improve the state of the art in real-time tracking, they are only a stepping stone to more elaborate tracking algorithms. Various open problems and possible extensions exist that are discussed in more detail in this section.

Shape Model Hierarchy The shape model hierarchy was able to increase the accuracy and robustness of the baseline shape model tracker significantly. The core idea is to collect all representations of the object during tracking that either have a significantly different viewpoint or a significantly different scale. However, there is no structure within the hierarchy of shape models itself. In each tracking step, the similarity between the current shape model and all models in the hierarchy is computed to estimate the models that should be used for tracking. However, since the shape models with a different scale usually have a different number of points, no similarity is computed (since the l_2 distance is used). As a consequence, more models are used for tracking than would actually be necessary. To remedy this, a more profound similarity measure than the l_2 distance could be used. For example, one that is either based on image similarity or on contour similarity. This could lead to a significant performance improvement

Furthermore, the appearance and the transformation parameters in the hierarchy could be used to generate estimates of the 3D movement of the object. In turn, this could help to create a semantically meaningful 3D hierarchy of shape models and create a rough 3D model of the object. This could be used to create descriptive shape models for the tracking step and improve the robustness to viewpoint changes of the object.

Shape Model Update If the shape model has many model points, only a random subset is used for tracking. This step allows to decrease the runtime and has little effect on the robustness and accuracy of the tracker. However, the update of the shape model tracker is currently restricted to the model points that are used for tracking. As a consequence, the model points that are not used in the tracking step are not updated. This incomplete update of the model decreases the score and may lead to the tracker losing the target. To counter this effect, it would be reasonable to approximate the shape model points by contours or splines. The updated model point could then act as support points of the contours and be used to update all models points. As a result, the shape model would be more robust and the robustness in long sequences should be further improved.

Moreover, the update step is currently restricted to updating the model points only. The region returned by the tracker itself is not updated. Instead, only the transformation parameters that are determined by the tracking process are applied to the initial object region. Especially in sequences with occlusion, the shape model inherently knows which part of the model is occluded since the respective model points have a very low score. To utilize this information, it would be possible to use the shape model points as support points for manipulating the region returned by the tracker. The scores of the shape model points could further be used to create an occlusion map.

Dynamic Pyramid Generation The regression of the shape model points in the dynamic shape model generation did not result in the expected performance gain. Instead of restricting the classifier to a small window around each model point, it would be reasonable to learn the scaled shape model directly. Hence, the classifier would obtain all model points and the current model image template. As output, the classifier would return the model points on the coarser levels of the pyramid. A near infinite amount of training data can be created automatically by applying the baseline tracker initialization to arbitrary input images. This should allow the classifier to learn scale space effects automatically and create more descriptive shape model pyramids.

Optimal Bounding Boxes The optimal boxes for a pixel-precise segmentation required for the computation of the Φ_{rIoU} are not restricted to tracking. In general, they could also be applied to improve the evaluation of object detection approaches. Furthermore, the optimal representation of the objects with respect to the evaluation metrics might also help to improve object detectors themselves. For example, instead of training on axis-aligned bounding boxes of objects, the optimal boxes could be used.

Towards More Holistic Trackers The presented shape model tracker is very accurate and fast. However, it is merely applicable to roughly rigid objects and a restricted amount of viewpoint change. These are both restrictions that the current state-of-the-art deep learning trackers do not have. However, although they are able to generalize well to deformations and different types of objects, they are much slower and prone to drifting from the object in long sequences. The strengths of both approaches could be fused to develop more elaborate methods in different manners. One possibility for this kind of fusion would be to apply some of the concepts developed for the shape model tracker directly to the current state-of-the-art deep learning approaches. For example, the logic behind the tracker confidence, the size of the search region, and the motion model are directly applicable to the deep learning approaches to generate reasonable predictions for the object location. Another possibility would be to use the high-level structure of the shape model as a reference for developing completely new learning-based algorithms. For example, the focus of the shape model on image edges and a similarity measure that is invariant to occlusion and illumination changes allows a very accurate and robust tracking. Furthermore, the least-squares refinement reduces the danger of tracker drift. All three concepts could be used to design a learning-based tracker that is much more stable.

Appendices

A Bayesian Tracking

To prove the validity of (6.6) and (6.7), three lemmas are required.

Lemma 1

$$p(A|B,C) = \frac{p(A,B|C)}{p(B|C)} \tag{A.1}$$

Proof.

$$p(A,B|C) = \frac{p((A \cap B) \cap C)}{p(C)} = \frac{p(A \cap (B \cap C)) \cdot p(B \cap C)}{p(C) \cdot p(B \cap C)} \tag{A.2}$$

$$= \frac{p(A \cap (B \cap C))}{p(B \cap C)} \cdot \frac{p(B \cap C)}{p(C)} = p(A|B,C)p(B|C). \tag{A.3}$$

The division by $p(B|C)$ completes the proof. The above proof assumes the intersection of B and C to be non-empty. If it is empty, the initial equation $p(A,B|C)$ is 0 anyway, as the intersection of B and C is empty. □

Lemma 2
Given the measurements $\mathcal{Z}_{t-1}$ and the the object state position history $\mathcal{X}_t$, the probability of the measurement z_t can be calculated as $p(z_t|x_t)$, hence

$$p(z_t|\mathcal{X}_t, \mathcal{Z}_{t-1}) = p(z_t|x_t). \tag{A.4}$$

Proof.

$$\prod_{i=1}^{t} p(z_i|x_i) \overset{(6.5)}{=} p(\mathcal{Z}_t|\mathcal{X}_t) \tag{A.5}$$

$$= p(z_t, \mathcal{Z}_{t-1}|\mathcal{X}_t) \tag{A.6}$$

$$\overset{(A.1)}{=} p(z_t|\mathcal{Z}_{t-1}, \mathcal{X}_t)p(\mathcal{Z}_{t-1}|\mathcal{X}_t) \tag{A.7}$$

$$\overset{(6.5)}{=} p(z_t|\mathcal{Z}_{t-1}, \mathcal{X}_t) \prod_{i=1}^{t-1} p(z_i|x_i). \tag{A.8}$$

Dividing both sides by $\prod_{i=1}^{t-1}$ yields the result. □

Lemma 3

The probability of the current object state x_t, given all the previous state dynamics and observations solely depends on the last state position $p(x_t|x_{t-1})$:

$$p(x_t|\mathcal{X}_{t-1}, \mathcal{Z}_{t-1}) = p(x_t|x_{t-1}). \tag{A.9}$$

Proof. The above term can be manipulated with the help of Lemma 1 to yield

$$p(x_t|\mathcal{X}_{t-1}, \mathcal{Z}_{t-1}) = \frac{p(x_t, \mathcal{Z}_{t-1}|\mathcal{X}_{t-1})}{p(\mathcal{Z}_{t-1}|\mathcal{X}_{t-1})}. \tag{A.10}$$

From (6.4), we further know that

$$p(\mathcal{Z}_{t-1}, x_t|\mathcal{X}_{t-1}) = p(x_t|\mathcal{X}_{t-1})p(\mathcal{Z}_{t-1}|\mathcal{X}_{t-1}). \tag{A.11}$$

Hence

$$p(x_t|\mathcal{X}_{t-1}, \mathcal{Z}_{t-1}) = p(x_t|\mathcal{X}_{t-1}) \overset{(6.3)}{=} p(x_t|x_{t-1}). \tag{A.12}$$

□

The propagation can now be derived with the help of the Bayes' theorem,

$$p(A|B) = \frac{p(B|A)p(B)}{p(A)}. \tag{A.13}$$

Furthermore, using Lemma 2, we can conclude that

$$\begin{aligned} p(\mathcal{X}_t|\mathcal{Z}_t) = p(\mathcal{X}_t|z_t, \mathcal{Z}_{t-1}) &\overset{(A.13)}{=} \frac{p(z_t|\mathcal{X}_t, \mathcal{Z}_{t-1})p(\mathcal{X}_t|\mathcal{Z}_{t-1})}{p(z_t|\mathcal{Z}_{t-1})} && \text{(A.14)}\\ &= k_t p(z_t|\mathcal{X}_t, \mathcal{Z}_{t-1})p(\mathcal{X}_t|\mathcal{Z}_{t-1}) && \text{(A.15)}\\ &= k_t p(z_t|x_t)p(\mathcal{X}_t|\mathcal{Z}_{t-1}), && \text{(A.16)} \end{aligned}$$

where k_t is a normalization constant that is independent of x_t. Furthermore, integrating with respect to $\mathcal{X}_{t-1}$ gives

$$p(x_t|\mathcal{Z}_{t-1}) = k_t p(z_t|x_t)p(x_t|\mathcal{Z}_{t-1}). \tag{A.17}$$

The last term can be expanded to yield

$$\begin{aligned} p(x_t|\mathcal{Z}_{t-1}) &= \int p(x_t|\mathcal{X}_{t-1}\mathcal{Z}_{t-1})p(\mathcal{X}_{t-1}|\mathcal{Z}_{t-1})\, \mathrm{d}\mathcal{X}_{t-1} && \text{(A.18)}\\ &\overset{\text{Lemma 3}}{=} \int\int p(x_t|x_{t-1})p(\mathcal{X}_{t-1}|\mathcal{Z}_{t-1})\, \mathrm{d}\mathcal{X}_{t-2}\, \mathrm{d}x_{t-1} && \text{(A.19)}\\ &= \int p(x_t|x_{t-1})p(x_{t-1}|\mathcal{Z}_{t-1})\, \mathrm{d}x_{t-1}, && \text{(A.20)} \end{aligned}$$

which is precisely what was proposed in (6.6). The posterior density in (6.7) describes all the knowledge about x_t that can be deduced from the observed data $\mathcal{Z}_t$.

B Correlation Filter Tracking

The closed-form solution of (6.13) can be obtained by setting the derivate with respect to $\tilde{w}$ to zero,

$$w = \left(X^T X + \lambda I\right)^{-1} X^T y. \tag{B.1}$$

The above formulation can be extended to complex settings in a straightforward manner

$$w^* = \left(X^H X + \lambda I\right)^{-1} X^H y. \tag{B.2}$$

A sample x_i is assumed to be the linearised image. Filtering in the Fourier domain assumes that the input image is cyclic. This can be enforced by filtering the input image with a cosine window and thus ignoring the pixel values at the image borders. Furthermore, if the samples used to optimize w are selected densely, the matrix X is of cyclic nature. Hence the rows are merely shifts of the input image:

$$X = C(x) = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_n & x_1 & \dots & x_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & \dots & x_1 \end{pmatrix}. \tag{B.3}$$

Cyclic matrices are square matrices and have the attribute that their DFT merely consists of the diagonal matrix generated by the vector x_i,

$$X = F^H \text{diag}(\hat{x}) F. \tag{B.4}$$

Hence, using dense shifts as the sample data and applying the DFT to (B.2), the optimal w^* can be determined as

$$\begin{aligned}
Fw^* &= F\left(X^H X + \lambda I\right)^{-1} X^H y \\
\hat{w}^* &= F\left(F^H \mathrm{diag}(\hat{x}^*)\,\mathrm{diag}(\hat{x}) F + \lambda F^H F\right)^{-1} F^H \mathrm{diag}(\hat{x}^*) F y \\
\hat{w}^* &= F F^H \left(\mathrm{diag}(\hat{x}^*)\,\mathrm{diag}(\hat{x}) + \lambda\right)^{-1} F F^H \mathrm{diag}(\hat{x}^*) \hat{y} \\
\hat{w}^* &= \left(\mathrm{diag}(\hat{x}^*)\,\mathrm{diag}(\hat{x}) + \lambda\right)^{-1} \mathrm{diag}(\hat{x}^*) \hat{y} \\
\hat{w}^* &= \mathrm{diag}\left(\frac{\hat{x}}{\hat{x}^* \odot \hat{x} + \lambda}\right) \hat{y} \\
\hat{w}^* &= \frac{\hat{x} \odot \hat{y}}{\hat{x}^* \odot \hat{x} + \lambda}
\end{aligned} \tag{B.5}$$

where the second step makes use of the fact that B.4 also yields

$$X^H X = F^H \mathrm{diag}(\hat{x}^*)\,\mathrm{diag}(\hat{x}) F, \tag{B.6}$$

and the fact that F is unitary, hence $FF^H = I$.

Bibliography

[1] Osman Akin, Erkut Erdem, Aykut Erdem, and Krystian Mikolajczyk, *Deformable part-based tracking by coupled global and local correlation filters*, Journal on Visual Communication and Image Representation, 38 (2016), pp. 763–774. doi: 10.1016/j.jvcir.2016.04.018.

[2] Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother, *Augmented Reality Meets Deep Learning for Car Instance Segmentation in Urban Scenes*, in British Machine Vision Conference (BMVC), 2017.

[3] Senjian An, Patrick Peursum, Wanquan Liu, and Svetha Venkatesh, *Efficient algorithms for subwindow search in object detection and localization*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 264–271. doi: 10.1109/CVPRW.2009.5206822.

[4] Mathieu Aubry, Daniel Maturana, Alexei A. Efros, Bryan C. Russell, and Josef Sivic, *Seeing 3D Chairs: Exemplar Part-Based 2D-3D Alignment Using a Large Dataset of CAD Models*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 3762–3769. doi: 10.1109/CVPR.2014.487.

[5] Shai Avidan, *Support Vector Tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 26 (2004), pp. 1064–1072. doi: 10.1109/TPAMI.2004.53.

[6] Shai Avidan, *Ensemble Tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 29 (2007), pp. 261–271. doi: 10.1109/TPAMI.2007.35.

[7] Boris Babenko, Ming-Hsuan Yang, and Serge J. Belongie, *Visual tracking with online Multiple Instance Learning*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 983–990. doi: 10.1109/CVPRW.2009.5206737.

[8] R Venkatesh Babu, Sundaram Suresh, and Anamitra Makur, *Online adaptive radial basis function networks for robust object tracking*, Computer Vision and Image Understanding, 114 (2010), pp. 297–310. doi: 0.1016/j.cviu.2009.10.004.

[9] Simon Baker and Iain A. Matthews, *Lucas-Kanade 20 Years On: A Unifying Framework*, International Journal of Computer Vision, 56 (2004), pp. 221–255. doi: 10.1023/B:VISI.0000011205.11775.fd.

[10] Dana H. Ballard, *Generalizing the Hough transform to detect arbitrary shapes*, Pattern Recognition, 13 (1981), pp. 111–122. doi: 10.1016/0031-3203(81)90009-1.

[11] Chenglong Bao, Yi Wu, Haibin Ling, and Hui Ji, *Real time robust L1 tracker using accelerated proximal gradient approach*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 1830–1837. doi: 10.1109/CVPR.2012.6247881.

[12] Vasileios Belagiannis, Falk Schubert, Nassir Navab, and Slobodan Ilic, *Segmentation Based Particle Filtering for Real-Time 2D Object Tracking*, in European Conference on Computer Vision (ECCV), 2012, pp. 842–855. doi: 10.1007/978-3-642-33765-9_60.

[13] Ben Benfold and Ian D. Reid, *Stable multi-target tracking in real-time surveillance video*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2011, pp. 3457–3464. doi: 10.1109/CVPR.2011.5995667.

[14] Jérôme Berclaz, François Fleuret, Engin Türetken, and Pascal Fua, *Multiple Object Tracking Using K-Shortest Paths Optimization*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 33 (2011), pp. 1806–1819. doi: 10.1109/TPAMI.2011.21.

[15] Christophe Berger, Thierry Géraud, Roland Levillain, Nicolas Widynski, Anthony Baillard, and Emmanuel Bertin, *Effective Component Tree Computation with Application to Pattern Recognition in Astronomical Imaging*, in Proceedings of the International Conference on Image Processing, (ICIP), 2007, pp. 41–44. doi: 10.1109/ICIP.2007.4379949.

[16] Luca Bertinetto, Jack Valmadre, Stuart Golodetz, Ondrej Miksik, and Philip H. S. Torr, *Staple: Complementary Learners for Real-Time Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1401–1409. doi: 10.1109/CVPR.2016.156.

[17] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr, *Fully-Convolutional Siamese Networks for Object Tracking*, in European Conference on Computer Vision Workshops on Benchmarking Mutliple Object Tracking (ECCVW), 2016, pp. 850–865. doi: 10.1007/978-3-319-48881-3_56.

[18] Charles Bibby and Ian D. Reid, *Robust Real-Time Visual Tracking Using Pixel-Wise Posteriors*, in European Conference on Computer Vision (ECCV), 2008, pp. 831–844. doi: 10.1007/978-3-540-88688-4_61.

[19] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui Man Lui, *Visual object tracking using adaptive correlation filters*, in IEEE Conference on Computer

Vision and Pattern Recognition (CVPR), 2010, pp. 2544–2550. doi: 10.1109/CVPR.2010.5539960.

[20] Gunilla Borgefors, *Hierarchical Chamfer Matching: A Parametric Edge Matching Algorithm*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 10 (1988), pp. 849–865. doi: 10.1109/34.9107.

[21] Tobias Böttger, Gutermuth Dominik, and Christina Eisenhofer, *A Natural Extension of Component-Trees to Multi-Channel Images*, IEEE Transactions on Image Processing, - (-), p. UNDER REVIEW.

[22] Tobias. Böttger and Christina Eisenhofer, *Efficiently tracking extremal regions in multichannel images*, in 8th International Conference of Pattern Recognition Systems (ICPRS), Institution of Engineering and Technology, 2017, pp. 6–14. doi: 10.1049/cp.2017.0143.

[23] Tobias Böttger and Patrick Follmann, *The Benefits of Evaluating Tracker Performance Using Pixel-Wise Segmentations*, in IEEE International Conference on Computer Vision Workshops (ICCVW), 2017, pp. 1983–1991. doi: 10.1109/ICCVW.2017.232.

[24] Tobias Böttger, Patrick Follmann, and Michael Fauser, *Measuring the Accuracy of Object Detectors and Trackers*, in German Conference on Pattern Recognition (GCPR), 2017, pp. 415–426. doi: 10.1007/978-3-319-66709-6_33.

[25] Tobias Böttger and Dominik Gutermuth, *Edge-based Component-Trees for Multi-Channel Image Segmentation*, Computing Research Repository (CoRR), abs/1705.01906 (2017).

[26] Tobias Böttger and Carsten Steger, *Accurate and Robust Tracking of Rigid Objects in Real-time*, IEEE Transactions on Image Processing, - (-), p. UNDER REVIEW.

[27] Tobias Böttger, Markus Ulrich, and Carsten Steger, *Subpixel-Precise Tracking of Rigid Objects in Real-Time*, in 20th Scandinavian Conference on Image Analysis (SCIA), 2017, pp. 54–65. doi: 10.1007/978-3-319-59126-1_5.

[28] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke, *Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping*, in 2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018, 2018, pp. 4243–4250. doi: 10.1109/ICRA.2018.8460875.

[29] Gary R. Bradski, *Computer Vision Face Tracking For Use in a Perceptual User Interface*, in Intel Technology Journal, 1998.

[30] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla, *Segmentation and Recognition Using Structure from Motion Point Clouds*, in European

Conference on Computer Vision (ECCV), David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman, eds., vol. 5302 of Lecture Notes in Computer Science, Springer, 2008, pp. 44–57. doi: 10.1007/978-3-540-88682-2_5.

[31] Michal Bušta, Lukáš Neumann, and Jiri Matas, *Deep textspotter: An end-to-end trainable scene text localization and recognition framework*, in IEEE International Conference on Computer Vision (ICCV), 2017, pp. 22–29. doi: 10.1109/ICCV.2017.242.

[32] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool, *One-Shot Video Object Segmentation*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5320–5329. doi: 10.1109/CVPR.2017.565.

[33] John F. Canny, *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8 (1986), pp. 679–698. doi: 10.1109/TPAMI.1986.4767851.

[34] Edwin Carlinet and Thierry Géraud, *A Comparison of Many Max-tree Computation Algorithms*, in International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing (ISMM), vol. 7883 of Lecture Notes in Computer Science, Springer, 2013, pp. 73–85. doi: 10.1007/978-3-642-38294-9_7.

[35] Edwin Carlinet and Thierry Géraud, *MToS: A tree of shapes for multivariate images*, IEEE Transactions on Image Processing, 24 (2015), pp. 5330–5342. doi: 10.1109/TIP.2015.2480599.

[36] Andrea Cavallaro, Olivier Steiger, and Touradj Ebrahimi, *Tracking video objects in cluttered background*, IEEE Transactions on Circuits and Systems for Video Technology, 15 (2005), pp. 575–584. doi: 10.1109/TCSVT.2005.844447.

[37] Luka Cehovin, Matej Kristan, and Ales Leonardis, *Robust Visual Tracking Using an Adaptive Coupled-Layer Visual Model*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 35 (2013), pp. 941–953. doi: 10.1109/TPAMI.2012.145.

[38] Luka Cehovin, Matej Kristan, and Ales Leonardis, *Is my new tracker really better than yours?*, in IEEE Winter Conference on Applications of Computer Vision (WACV), 2014, pp. 540–547. doi: 10.1109/WACV.2014.6836055.

[39] Luka Cehovin, Ales Leonardis, and Matej Kristan, *Robust visual tracking using template anchors*, in IEEE Winter Conference on Applications of Computer Vision (WACV), 2016, pp. 1–8. doi: 10.1109/WACV.2016.7477570.

[40] Luka Cehovin, Ales Leonardis, and Matej Kristan, *Visual Object Tracking Performance Measures Revisited*, IEEE Transactions on Image Processing, 25 (2016), pp. 1261–1274. doi: 10.1109/TIP.2016.2520370.

[41] YT Chan, AGC Hu, and JB Plant, *A Kalman filter based tracking scheme with input estimation*, IEEE Transactions on Aerospace and Electronic Systems, (1979), pp. 237–244. doi: 10.1109/TAES.1979.308710.

[42] Angel X. Chang, Angela Dai, Thomas A. Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang, *Matterport3D: Learning from RGB-D Data in Indoor Environments*, in IEEE International Conference on 3D Vision (3DV), 2017, pp. 667–676. doi: 10.1109/3DV.2017.00081.

[43] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu, *ShapeNet: An Information-Rich 3D Model Repository*, Computing Research Repository (CoRR), arXiv:1512.03012 (2015).

[44] Aaron Chavez and David Gustafson, *Color-Based Extensions to MSERs*, in Advances in Visual Computing - 7th International Symposium (ISVC), 2011, pp. 358–366. doi: 10.1007/978-3-642-24031-7_36.

[45] Luojian Chen, Michael W. Berry, and William W. Hargrove, *Using dendronal signatures for feature extraction and retrieval*, International Journal on Imaging Systems and Technology, 11 (2000), pp. 243–253. doi: 10.1002/ima.1009.

[46] Liang-Chieh Chen, Sanja Fidler, and Raquel Urtasun, *Beat the MTurkers: Automatic Image Labeling from Weak 3D Supervision*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 3198–3205. doi: 10.1109/CVPR.2014.409.

[47] Qifeng Chen and Vladlen Koltun, *Photographic Image Synthesis with Cascaded Refinement Networks*, in IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1520–1529. doi: 10.1109/ICCV.2017.168.

[48] Yizong Cheng, *Mean Shift, Mode Seeking, and Clustering*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 17 (1995), pp. 790–799. doi: 10.1109/34.400568.

[49] Jongwon Choi, Hyung Jin Chang, Sangdoo Yun, Tobias Fischer, Yiannis Demiris, and Jin Young Choi, *Attentional Correlation Filter Network for Adaptive Visual Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4828–4837. doi: 10.1109/CVPR.2017.513.

[50] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun, *A Large Dataset of Object Scans*, Computing Research Repository (CoRR), arXiv:1602.02481 (2016).

[51] Hongxia Chu, Zhongyu Xie, Xiangju Nie, Zhanying Li, and Xin Li, *Particle filter target tracking method optimized by improved mean shift*, in IEEE International

Conference on Information and Automation, IEEE, 2013, pp. 991–994. doi: 10.1109/ICInfA.2013.6720439.

[52] Robert T. Collins, Yanxi Liu, and Marius Leordeanu, *Online Selection of Discriminative Tracking Features*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27 (2005), pp. 1631–1643. doi: 10.1109/TPAMI.2005.205.

[53] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer, *Real-Time Tracking of Non-Rigid Objects Using Mean Shift*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2000, pp. 142–149. doi: 10.1109/CVPR.2000.854761.

[54] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer, *Kernel-Based Object Tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 25 (2003), pp. 564–575. doi: 10.1109/TPAMI.2003.1195991.

[55] Timothy F. Cootes, Christopher J. Taylor, David H. Cooper, and Jim Graham, *Active Shape Models-Their Training and Application*, Computer Vision and Image Understanding, 61 (1995), pp. 38–59. doi: 10.1006/cviu.1995.1004.

[56] 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 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 3213–3223. doi: 10.1109/CVPR.2016.350.

[57] Jean Cousty, Michel Couprie, Laurent Najman, and Gilles Bertrand, *Weighted fusion graphs: Merging properties and watersheds*, Discrete Applied Mathematics, 156 (2008), pp. 3011–3027. doi: 10.1016/j.dam.2008.01.005.

[58] Jean Cousty, Laurent Najman, and Benjamin Perret, *Constructive Links between Some Morphological Hierarchies on Edge-Weighted Graphs*, in Mathematical Morphology and Its Applications to Signal and Image Processing, 2013, pp. 86–97. doi: 10.1007/978-3-642-38294-9_8.

[59] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath, *Generative Adversarial Networks: An Overview*, IEEE Signal Processing Magazine, 35 (2018), pp. 53–65. doi: 10.1109/MSP.2017.2765202.

[60] Dan Crisan, Pierre Del Moral, and Terry Lyons, *Non-linear filtering using branching and interacting particle systems*, in Markov Processes Related Fields, vol. 35, 1999, pp. 293–319. doi: an:0967.93088.

[61] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg, *ECO: Efficient Convolution Operators for Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 6931–6939. doi: 10.1109/CVPR.2017.733.

[62] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg, *Accurate Scale Estimation for Robust Visual Tracking*, in British Machine Vision Conference (BMVC), BMVA Press, 2014.

[63] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg, *Convolutional Features for Correlation Filter Based Visual Tracking*, in IEEE International Conference on Computer Vision Workshops (ICCVW), 2015, pp. 621–629. doi: 10.1109/ICCVW.2015.84.

[64] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg, *Learning Spatially Regularized Correlation Filters for Visual Tracking*, in IEEE International Conference on Computer Vision (ICCV), 2015, pp. 4310–4318. doi: 10.1109/ICCV.2015.490.

[65] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg, *Discriminative Scale Space Tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 39 (2017), pp. 1561–1575. doi: 10.1109/TPAMI.2016.2609928.

[66] Martin Danelljan, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg, *Beyond Correlation Filters: Learning Continuous Convolution Operators for Visual Tracking*, in European Conference on Computer Vision (ECCV), vol. 9909 of Lecture Notes in Computer Science, Springer, 2016, pp. 472–488. doi: 10.1007/978-3-319-46454-1_29.

[67] César Roberto de Souza, Adrien Gaidon, Yohann Cabon, and Antonio Manuel López Peña, *Procedural Generation of Videos to Train Deep Action Recognition Networks*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2594–2604. doi: 10.1109/CVPR.2017.278.

[68] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li, *ImageNet: A large-scale hierarchical image database*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 248–255. doi: 10.1109/CVPRW.2009.5206848.

[69] Caglayan Dicle, Octavia I. Camps, and Mario Sznaier, *The Way They Move: Tracking Multiple Targets with Similar Appearance*, in IEEE International Conference on Computer Vision (ICCV), 2013, pp. 2304–2311. doi: 10.1109/ICCV.2013.286.

[70] Piotr Dollár, Ron Appel, Serge J. Belongie, and Pietro Perona, *Fast Feature Pyramids for Object Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 36 (2014), pp. 1532–1545. doi: 10.1109/TPAMI.2014.2300479.

[71] Zilong Dong, Guofeng Zhang, Jiaya Jia, and Hujun Bao, *Efficient keyframe-based real-time camera tracking*, Computer Vision and Image Understanding, 118 (2014), pp. 97–110. doi: 10.1016/j.cviu.2013.08.005.

[72] Michael Donoser and Horst Bischof, *Efficient Maximally Stable Extremal Region (MSER) Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2006, pp. 553–560. doi: 10.1109/CVPR.2006.107.

[73] Michael Donoser, Hayko Riemenschneider, and Horst Bischof, *Linked edges as stable region boundaries*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010, pp. 1665–1672. doi: 10.1109/CVPR.2010.5539833.

[74] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun, *CARLA: An Open Urban Driving Simulator*, in Annual Conference on Robot Learning (CoRL), vol. 78, 2017, pp. 1–16.

[75] Mark Everingham, S. M. Ali Eslami, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman, *The Pascal Visual Object Classes Challenge: A Retrospective*, International Journal of Computer Vision, 111 (2015), pp. 98–136. doi: 10.1007/s11263-014-0733-5.

[76] Mark Fashing and Carlo Tomasi, *Mean Shift Is a Bound Optimization*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27 (2005), pp. 471–474. doi: 10.1109/TPAMI.2005.59.

[77] Marjan Firouznia, Karim Faez, Hamidreza Amindavar, and Javad Alikhani Koupaei, *Chaotic particle filter for visual object tracking*, Journal of Visual Communication and Image Representation, 53 (2018), pp. 1–12. doi: 10.1016/j.jvcir.2018.02.014.

[78] Patrick Follmann, Tobias Böttger, Philipp Härtinger, Rebecca König, and Markus Ulrich, *MVTec D2S: Densely Segmented Supermarket Dataset*, in European Conference on Computer Vision (ECCV), 2018, pp. 569–585. doi: 10.1007/978-3-030-01249-6_35.

[79] Patrick Follmann, Bertram Drost, and Tobias Böttger, *Acquire, Augment, Segment & Enjoy: Weakly Supervised Instance Segmentation of Supermarket Products*, in German Conference on Pattern Recognition (GCPR), 2018.

[80] Per-Erik Forssén, *Maximally Stable Colour Regions for Recognition and Matching*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2007. doi: 10.1109/CVPR.2007.383120.

[81] Thomas E Fortmann, Yaakov Bar-Shalom, and Molly Scheffe, *Multi-target tracking using joint probabilistic data association*, in IEEE Conference on Decision and Control including the Symposium on Adaptive Processes, 1980, pp. 807–812. doi: 10.1109/ICC.2015.7249379.

[82] Keinosuke Fukunaga and Larry D. Hostetler, *The estimation of the gradient of a density function, with applications in pattern recognition*, IEEE Transactions on Information Theory, 21 (1975), pp. 32–40. doi: 10.1109/TIT.1975.1055330.

[83] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig, *Virtual Worlds as Proxy for Multi-Object Tracking Analysis*, Computing Research Repository (CoRR), arXiv:1605.06457 (2016).

[84] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun, *Vision meets robotics: The KITTI dataset*, The International Journal of Robotics Research, 32 (2013), pp. 1231–1237. doi: 10.1177/0278364913491297.

[85] Andreas Geiger, Philip Lenz, and Raquel Urtasun, *Are we ready for autonomous driving? The KITTI vision benchmark suite*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 3354–3361. doi: 10.1109/CVPR.2012.6248074.

[86] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio, *Generative Adversarial Nets*, in Conference on Advances in Neural Information Processing Systems (NIPS), 2014, pp. 2672–2680.

[87] Neil J Gordon, David J Salmond, and Adrian FM Smith, *Novel approach to nonlinear/non-Gaussian Bayesian state estimation*, IEE Proceedings F (Radar and Signal Processing), 140 (1993), pp. 107–113. doi: 10.1049/ip-f-2.1993.0015.

[88] John C Gower and Gavin JS Ross, *Minimum spanning trees and single linkage cluster analysis*, Applied statistics, (1969), pp. 54–64. doi: 10.2307/2346439.

[89] Robert M. Gray, *On the asymptotic eigenvalue distribution of Toeplitz matrices*, IEEE Transactions on Information Theory, 18 (1972), pp. 725–730. doi: 10.1109/TIT.1972.1054924.

[90] Erhan Gundogdu and A. Aydin Alatan, *Good Features to Correlate for Visual Tracking*, IEEE Transactions on Image Processing, 27 (2018), pp. 2526–2540. doi: 10.1109/TIP.2018.2806280.

[91] Bohyung Han, Dorin Comaniciu, Ying Zhu, and Larry S. Davis, *Incremental Density Approximation and Kernel-Based Bayesian Filtering for Object Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2004, pp. 638–644. doi: 10.1109/CVPR.2004.130.

[92] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla, *SceneNet: Understanding Real World Indoor Scenes With Synthetic Data*, Computing Research Repository (CoRR), arXiv:1511.07041 (2015).

[93] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision, Vol. 1*, Addison-Wesley Longman Publishing Co., Inc., 1991.

[94] Sam Hare, Amir Saffari, and Philip H. S. Torr, *Struck: Structured output tracking with kernels*, in IEEE International Conference on Computer Vision (ICCV), 2011, pp. 263–270. doi: 10.1109/ICCV.2011.6126251.

[95] Hironori Hattori, Vishnu Naresh Boddeti, Kris M. Kitani, and Takeo Kanade, *Learning scene-specific pedestrian detectors without real data*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3819–3827. doi: 10.1109/CVPR.2015.7299006.

[96] Anfeng He, Chong Luo, Xinmei Tian, and Wenjun Zeng, *A twofold Siamese network for real-time object tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4834–4843.

[97] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista, *High-Speed Tracking with Kernelized Correlation Filters*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 37 (2015), pp. 583–596. doi: 10.1109/TPAMI.2014.2345390.

[98] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge P. Batista, *Exploiting the Circulant Structure of Tracking-by-Detection with Kernels*, in European Conference on Computer Vision (ECCV), 2012, pp. 702–715. doi: 10.1007/978-3-642-33765-9_50.

[99] Seunghoon Hong, Tackgeun You, Suha Kwak, and Bohyung Han, *Online tracking by learning discriminative saliency map with convolutional neural network*, in International Conference on Machine Learning (ICML), 2015, pp. 597–606.

[100] Dafei Huang, Lei Luo, Mei Wen, Zhaoyun Chen, and Chunyuan Zhang, *Enable Scale and Aspect Ratio Adaptability in Visual Tracking with Detection Proposals*, in British Machine Vision Conference (BMVC), 2015, pp. 185.1–185.12. doi: 10.5244/C.29.185.

[101] Michael Isard and Andrew Blake, *CONDENSATION - Conditional Density Propagation for Visual Tracking*, International Journal of Computer Vision, 29 (1998), pp. 5–28. doi: 10.1023/A:1008078328650.

[102] Irene Anindaputri Iswanto and Bin Li, *Visual Object Tracking Based on Mean-shift and Particle-Kalman Filter*, Procedia Computer Science, 116 (2017), pp. 587–595. doi: 10.1016/j.procs.2017.10.010.

[103] Hao Jiang, Sidney S. Fels, and James J. Little, *A Linear Programming Approach for Multiple Object Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2007. doi: 10.1109/CVPR.2007.383180.

[104] M. Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan, *Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?*, in IEEE International Conference on Robotics and Automation, 2017, pp. 1–8. doi: 10.1109/ICRA.2017.7989092.

[105] Ronald Jones, *Connected Filtering and Segmentation Using Component Trees*, Computer Vision and Image Understanding, 75 (1999), pp. 215–228. doi: 10.1006/cviu.1999.0777.

[106] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas, *Forward-Backward Error: Automatic Detection of Tracking Failures*, in nternational Conference on Pattern Recognition (ICPR), 2010, pp. 2756–2759. doi: 10.1109/ICPR.2010.675.

[107] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas, *Tracking-Learning-Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 34 (2012), pp. 1409–1422. doi: 10.1109/T.2011.239.

[108] Rudolph Emil Kalman, *A new approach to linear filtering and prediction problems*, Journal of basic Engineering, 82 (1960), pp. 35–45. doi: 10.1115/1.3662552.

[109] Keiji Kanazawa, Daphne Koller, and Stuart J. Russell, *Stochastic simulation algorithms for dynamic probabilistic networks*, in Proceedings on Uncertainty in Artificial Intelligence (UAI), 1995, pp. 346–351.

[110] Dimosthenis Karatzas, Lluis Gomez-Bigorda, Anguelos Nicolaou, Suman K. Ghosh, Andrew D. Bagdanov, Masakazu Iwamura, Jiri Matas, Lukas Neumann, Vijay Ramaseshan Chandrasekhar, Shijian Lu, Faisal Shafait, Seiichi Uchida, and Ernest Valveny, *ICDAR 2015 competition on Robust Reading*, in 13th International Conference on Document Analysis and Recognition (ICDAR), 2015, pp. 1156–1160. doi: 10.1109/ICDAR.2015.7333942.

[111] Baldur Karlsson, *RenderDoc*, https://renderdoc.org, (2018).

[112] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski, *ViZDoom: A Doom-based AI research platform for visual reinforcement learning*, in IEEE Conference on Computational Intelligence and Games (CIG), 2016, pp. 1–8. doi: 10.1109/CIG.2016.7860433.

[113] Zia Khan, Tucker R. Balch, and Frank Dellaert, *MCMC-Based Particle Filtering for Tracking a Variable Number of Interacting Targets*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27 (2005), pp. 1805–1918. doi: 10.1109/TPAMI.2005.223.

[114] Tobias Koch, *Robustes Tracking mithilfe des Partikelfilters - Ein merkmalsbasierter Ansatz und dessen Anwendung im Bereich der Verkehrsüberwachung*, Master's thesis, Ingenieurfakultät Bau Geo Umwelt, Technische Universität München, Germany, 2014.

[115] Hyung Il Koo and Duck Hoon Kim, *Scene Text Detection via Connected Component Clustering and Nontext Filtering*, IEEE Transactions on Image Processing, 22 (2013), pp. 2296–2305. doi: 10.1109/TIP.2013.2249082.

[116] Matej Kristan and et al., *The Visual Object Tracking VOT2016 Challenge Results*, in European Conference on Computer Vision Workshops (ECCVW), vol. 9914 of Lecture Notes in Computer Science, 2016, pp. 777–823. doi: 10.1007/978-3-319-48881-3_54.

[117] Matej Kristan and et al., *The Visual Object Tracking VOT2017 Challenge Results*, in IEEE International Conference on Computer Vision Workshops (ICCVW), 2017, pp. 1949–1972. doi: 10.1109/ICCVW.2017.230.

[118] Matej Kristan, Jiri Matas, Ales Leonardis, Michael Felsberg, Luka Cehovin, Gustavo Fernández, Tomás Vojír, Gustav Häger, Georg Nebehay, and Roman P. Pflugfelder, *The Visual Object Tracking VOT2015 Challenge Results*, in

IEEE International Conference on Computer Vision Workshops (ICCVW), 2015, pp. 564–586. doi: 10.1109/ICCVW.2015.79.

[119] Matej Kristan, Jiri Matas, Ales Leonardis, Tomás Vojír, Roman P. Pflugfelder, Gustavo Fernández, Georg Nebehay, Fatih Porikli, and Luka Cehovin, *A Novel Performance Evaluation Methodology for Single-Target Trackers*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 38 (2016), pp. 2137–2155. doi: 10.1109/TPAMI.2016.2516982.

[120] Camille Kurtz, Benoît Naegel, and Nicolas Passat, *Connected Filtering Based on Multivalued Component-Trees*, IEEE Transactions on Image Processing, 23 (2014), pp. 5152–5164. doi: 10.1109/TIP.2014.2362053.

[121] Camille Kurtz, Benoît Naegel, and Nicolas Passat, *Multivalued Component-Tree Filtering*, in International Conference on Pattern Recognition (ICPR), 2014, pp. 1008–1013. doi: 10.1109/TIP.2014.2362053.

[122] Yehezkel Lamdan, Jacob T. Schwartz, and Haim J. Wolfson, *Affine invariant model-based object recognition*, IEEE Transactions on Robotics and Automation, 6 (1990), pp. 578–589. doi: 10.1109/70.62047.

[123] Ido Leichter, Michael Lindenbaum, and Ehud Rivlin, *Mean Shift tracking with multiple reference color histograms*, Computer Vision and Image Understanding, 114 (2010), pp. 400–408. doi: 10.1016/j.cviu.2009.12.006.

[124] Vincent Lepetit and Pascal Fua, *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey*, Foundations and Trends in Computer Graphics and Vision, 1 (2005). doi: 10.1561/0600000001.

[125] William F. Leven and Aaron D. Lanterman, *Unscented Kalman Filters for Multiple Target Tracking With Symmetric Measurement Equations*, IEEE Transactions on Automatic Control, 54 (2009), pp. 370–375. doi: 10.1109/TAC.2008.2008327.

[126] Hanxi Li, Yi Li, and Fatih Porikli, *DeepTrack: Learning Discriminative Feature Representations Online for Robust Visual Tracking*, IEEE Transactions on Image Processing, 25 (2016), pp. 1834–1848. doi: 10.1109/TIP.2015.2510583.

[127] Shu-Xiao Li, Hong-Xing Chang, and Cheng-Fei Zhu, *Adaptive pyramid mean shift for global real-time visual tracking*, Image and Vision Computing, 28 (2010), pp. 424–437. doi: 10.1016/j.imavis.2009.06.012.

[128] Siyi Li and Dit-Yan Yeung, *Visual Object Tracking for Unmanned Aerial Vehicles: A Benchmark and New Motion Models*, in Conference on Artificial Intelligence (AAAI), AAAI Press, 2017, pp. 4140–4146.

[129] X Rong Li and Vesselin P Jilkov, *Survey of maneuvering target tracking: dynamic models*, in Signal and Data Processing of Small Targets, vol. 4048, International Society for Optics and Photonics, 2000, pp. 212–236. doi: 10.1109/TAES.2003.1261132.

[130] X Rong Li and Vesselin P Jilkov, *Survey of maneuvering target tracking: decision-based methods*, in Signal and Data Processing of Small Targets, vol. 4728, International Society for Optics and Photonics, 2002, pp. 511–535. doi: 10.1117/12.478535.

[131] Yang Li and Jianke Zhu, *A Scale Adaptive Kernel Correlation Filter Tracker with Feature Integration*, in European Conference on Computer Vision (ECCV), 2014, pp. 254–265. doi: 10.1007/978-3-319-16181-5_18.

[132] Ruei-Sung Lin, David A. Ross, Jongwoo Lim, and Ming-Hsuan Yang, *Adaptive Discriminative Generative Model and Its Applications*, in Conference on Advances in Neural Information Processing Systems (NIPS), 2004, pp. 801–808.

[133] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie, *Feature Pyramid Networks for Object Detection*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 936–944. doi: 10.1109/CVPR.2017.106.

[134] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick, *Microsoft COCO: Common Objects in Context*, in European Conference on Computer Vision (ECCV), vol. 8693 of Lecture Notes in Computer Science, Springer, 2014, pp. 740–755. doi: 10.1007/978-3-319-10602-1_48.

[135] Tony Lindeberg, *Scale-space theory: A basic tool for analyzing structures at different scales*, Journal of applied statistics, 21 (1994), pp. 225–270. doi: 10.1080/757582976.

[136] Wei-Lwun Lu, Kenji Okuma, and James J. Little, *Tracking and recognizing actions of multiple hockey players using the boosted particle filter*, Image and Vision Computing, 27 (2009), pp. 189–205. doi: 10.1016/j.imavis.2008.02.008.

[137] Bruce D. Lucas and Takeo Kanade, *An iterative image registration technique with an application to stereo vision.*, in International Joint Conference on Artificial Intelligence IJCAI, vol. 81, 1981, pp. 674–679.

[138] Alan Lukezic, Tomás Vojír, Luka Cehovin Zajc, Jiri Matas, and Matej Kristan, *Discriminative Correlation Filter Tracker with Channel and Spatial Reliability*, International Journal of Computer Vision, 126 (2018), pp. 671–688. doi: 10.1007/s11263-017-1061-3.

[139] Siwei Lyu, Ming-Ching Chang, Dawei Du, Longyin Wen, Honggang Qi, Yuezun Li, Yi Wei, Lipeng Ke, Tao Hu, Marco Del Coco, et al., *UA-DETRAC 2017: Report of AVSS2017 & IWT4S Challenge on Advanced Traffic Monitoring*, in IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 2017, pp. 1–7. doi: 10.1109/AVSS.2017.8078560.

[140] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang, *Hierarchical Convolutional Features for Visual Tracking*, in IEEE International Conference on Computer Vision (ICCV), 2015, pp. 3074–3082. doi: 10.1109/ICCV.2015.352.

[141] Chao Ma, Xiaokang Yang, Chongyang Zhang, and Ming-Hsuan Yang, *Long-term correlation tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5388–5396. doi: 10.1109/CVPR.2015.7299177.

[142] Emilio Maggio and Andrea Cavallaro, *Hybrid particle filter and mean shift tracker with adaptive transition model*, in IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 2, 2005, pp. 221–224. doi: 10.1109/ICASSP.2005.1415381.

[143] Luca Marchesotti, Lucio Marcenaro, Giancarlo Ferrari, and Carlo S. Regazzoni, *Multiple object tracking under heavy occlusions by using Kalman filters based on shape matching*, in International Conference on Image Processing (ICIP), 2002, pp. 341–344. doi: 10.1109/ICIP.2002.1038975.

[144] Francisco Massa, Bryan C. Russell, and Mathieu Aubry, *Deep Exemplar 2D-3D Detection by Adapting from Real to Rendered Views*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 6024–6033. doi: 10.1109/CVPR.2016.648.

[145] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla, *Robust wide-baseline stereo from maximally stable extremal regions*, Image and Vision Computing, 22 (2004), pp. 761–767. doi: 10.1016/j.imavis.2004.02.006.

[146] Julian Mattes and Jacques Demongeot, *Efficient Algorithms to Implement the Confinement Tree*, in Discrete Geometry for Computer Imagery (DGCI), vol. 1953 of Lecture Notes in Computer Science, Springer, 2000, pp. 392–405. doi: 10.1007/3-540-44438-6_32.

[147] Iain A. Matthews, Takahiro Ishikawa, and Simon Baker, *The Template Update Problem*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 26 (2004), pp. 810–815. doi: 10.1109/TPAMI.2004.16.

[148] Fernand Meyer and Petros Maragos, *Morphological Scale-Space Representation with Levelings*, in International Conference of Scale-Space Theories in Computer Vision, 1999, pp. 187–198. doi: 10.1007/3-540-48236-9_17.

[149] Fernand Meyer and Petros Maragos, *Nonlinear Scale-Space Representation with Morphological Levelings*, Journal on Visual Communication and Image Representation, 11 (2000), pp. 245–265. doi: 10.1006/jvci.1999.0447.

[150] Dan Mikami, Kazuhiro Otsuka, and Junji Yamato, *Memory-based Particle Filter for face pose tracking robust under complex dynamics*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 999–1006. doi: 10.1109/CVPRW.2009.5206661.

[151] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, *MOT16: A Benchmark for Multi-Object Tracking*, arXiv:1603.00831 [cs], (2016). arXiv: 1603.00831.

[152] Massimo Minervini, Andreas Fischbach, Hanno Scharr, and Sotirios A. Tsaftaris, *Finely-grained annotated datasets for image-based plant phenotyping*, Pattern Recognition Letters, 81 (2016), pp. 80–89. doi: 10.1016/j.patrec.2015.10.013.

[153] Matthias Mueller, Neil Smith, and Bernard Ghanem, *A Benchmark and Simulator for UAV Tracking*, in European Conference on Computer Vision (ECCV), vol. 9905 of Lecture Notes in Computer Science, Springer, 2016, pp. 445–461. doi: 10.1007/978-3-319-46448-0_27.

[154] Matthias Mueller, Neil Smith, and Bernard Ghanem, *Context-Aware Correlation Filter Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1387–1395. doi: 10.1109/CVPR.2017.152.

[155] Matthias Müller, Adel Bibi, Silvio Giancola, Salman Al-Subaihi, and Bernard Ghanem, *TrackingNet: A Large-Scale Dataset and Benchmark for Object Tracking in the Wild*, in European Conference on Computer Vision (ECCV), 2018, pp. 310–327. doi: 10.1007/978-3-030-01246-5_19.

[156] Makoto Nagao, Takashi Matsuyama, and Yoshio Ikeda, *Region extraction and shape analysis in aerial photographs*, Computer Graphics and Image Processing, 10 (1979), pp. 195–223. doi: 10.1016/0146-664X(79)90001-7.

[157] Hyeonseob Nam and Bohyung Han, *Learning Multi-domain Convolutional Neural Networks for Visual Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4293–4302. doi: 10.1109/CVPR.2016.465.

[158] Tahir Nawaz and Andrea Cavallaro, *A Protocol for Evaluating Video Trackers Under Real-World Conditions*, IEEE Transactions on Image Processing, 22 (2013), pp. 1354–1361. doi: 10.1109/TIP.2012.2228497.

[159] Lukas Neumann and Jiri Matas, *Real-time scene text localization and recognition*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 3538–3545. doi: 10.1109/CVPR.2012.6248097.

[160] Hieu Tat Nguyen and Arnold W. M. Smeulders, *Tracking Aspects of the Foreground against the Background*, in European Conference on Computer Vision (ECCV), 2004, pp. 446–456. doi: 10.1007/978-3-540-24671-8_35.

[161] David Nistér and Henrik Stewénius, *Linear Time Maximally Stable Extremal Regions*, in European Conference on Computer Vision (ECCV), vol. 5303 of Lecture Notes in Computer Science, Springer, 2008, pp. 183–196. doi: 10.1007/978-3-540-88688-4_14.

[162] Katja Nummiaro, Esther Koller-Meier, and Luc J. Van Gool, *An adaptive color-based particle filter*, Image and Vision Computing, 21 (2003), pp. 99–110. doi: 10.1016/S0262-8856(02)00129-4.

[163] Kenji Okuma, Ali Taleghani, Nando de Freitas, James J. Little, and David G. Lowe, *A Boosted Particle Filter: Multitarget Detection and Tracking*, in European Conference on Computer Vision (ECCV), 2004, pp. 28–39. doi: 10.1007/978-3-540-24670-1_3.

[164] Clark F. Olson and Daniel P. Huttenlocher, *Automatic target recognition by matching oriented edge pixels*, IEEE Transactions on Image Processing, 6 (1997), pp. 103–113. doi: 10.1109/83.552100.

[165] Shaul Oron, Aharon Bar-Hillel, Dan Levi, and Shai Avidan, *Locally Orderless Tracking*, International Journal of Computer Vision, 111 (2015), pp. 213–228. doi: 10.1007/s11263-014-0740-6.

[166] Georgios K Ouzounis and Pierre Soille, *The alpha-tree algorithm*, JRC Scientific and Policy Report, (2012).

[167] Jeremie Papon and Markus Schoeler, *Semantic Pose Using Deep Networks Trained on Synthetic RGB-D*, in IEEE International Conference on Computer Vision (ICCV), 2015, pp. 774–782. doi: 10.1109/ICCV.2015.95.

[168] Nicolas Passat and Benoît Naegel, *Component-Trees and Multivalued Images: Structural Properties*, Journal of Mathematical Imaging and Vision, 49 (2014), pp. 37–50. doi: 10.1007/s10851-013-0438-3.

[169] Jose Luis Patino, Tom Cane, Alain Vallee, and James M. Ferryman, *PETS 2016: Dataset and Challenge*, in IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2016, pp. 1240–1247. doi: 10.1109/CVPRW.2016.157.

[170] Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc J. Van Gool, Markus H. Gross, and Alexander Sorkine-Hornung, *A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 724–732. doi: 10.1109/CVPR.2016.85.

[171] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet, *Color-Based Probabilistic Tracking*, in European Conference on Computer Vision (ECCV), 2002, pp. 661–675. doi: 10.1007/3-540-47969-4_44.

[172] Siyang Qin and Roberto Manduchi, *A fast and robust text spotter*, in IEEE Winter Conference on Applications of Computer Vision (WACV), 2016, pp. 1–8. doi: 10.1109/WACV.2016.7477663.

[173] Weichao Qiu, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, and Yizhou Wang, *UnrealCV: Virtual Worlds for Computer Vision*, in ACM Multimedia Conference, ACM, 2017, pp. 1221–1224. doi: 10.1145/3123266.3129396.

[174] Esteban Real, Jonathon Shlens, Stefano Mazzocchi, Xin Pan, and Vincent Vanhoucke, *YouTube-BoundingBoxes: A Large High-Precision Human-Annotated Data Set for Object Detection in Video*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 7464–7473. doi: 10.1109/CVPR.2017.789.

[175] Donald Reid et al., *An algorithm for tracking multiple targets*, IEEE Transactions on Automatic Control, 24 (1979), pp. 843–854.

[176] Seyed Hamid Rezatofighi, Anton Milan, Zhen Zhang, Qinfeng Shi, Anthony R. Dick, and Ian D. Reid, *Joint Probabilistic Data Association Revisited*, in IEEE International Conference on Computer Vision (ICCV), 2015, pp. 3047–3055. doi: 10.1109/ICCV.2015.349.

[177] Mansoor Rezghi and Lars Eldén, *Diagonalization of tensors with circulant structure*, Linear Algebra and its Applications, 435 (2011), pp. 422–447. doi: 10.1016/j.laa.2010.03.032.

[178] Stephan R. Richter, Zeeshan Hayder, and Vladlen Koltun, *Playing for Benchmarks*, in IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2232–2241. doi: 10.1109/ICCV.2017.243.

[179] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun, *Playing for Data: Ground Truth from Computer Games*, in European Conference on Computer Vision (ECCV), vol. 9906 of LNCS, Springer International Publishing, 2016, pp. 102–118. doi: 10.1007/978-3-319-46475-6_7.

[180] Giorgio Roffo and Simone Melzi, *Online Feature Selection for Visual Tracking*, in British Machine Vision Conference (BMVC), BMVA Press, 2016.

[181] Germán Ros, Laura Sellart, Joanna Materzynska, David Vázquez, and Antonio M. López, *The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 3234–3243. doi: 10.1109/CVPR.2016.352.

[182] Paul L. Rosin, *Measuring shape: ellipticity, rectangularity, and triangularity*, Machine Vision and Applications, 14 (2003), pp. 172–184. doi: 10.1007/s00138-002-0118-6.

[183] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang, *Incremental Learning for Robust Visual Tracking*, International Journal of Computer Vision, 77 (2008), pp. 125–141. doi: 10.1007/s11263-007-0075-7.

[184] William Rucklidge, *Efficiently Locating Objects Using the Hausdorff Distance*, International Journal of Computer Vision, 24 (1997), pp. 251–270. doi: 10.1023/A:1007975324482.

[185] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li, *ImageNet Large Scale Visual Recognition*

Challenge, International Journal of Computer Vision, 115 (2015), pp. 211–252. doi: 10.1007/s11263-015-0816-y.

[186] Philippe Salembier, Albert Oliveras-Vergés, and Luis Garrido, *Antiextensive connected operators for image and sequence processing*, IEEE Transactions on Image Processing, 7 (1998), pp. 555–570. doi: 10.1109/83.663500.

[187] Manolis Savva, Angel X. Chang, Alexey Dosovitskiy, Thomas A. Funkhouser, and Vladlen Koltun, *MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments*, Computing Research Repository (CoRR), arXiv:1712.03931 (2017).

[188] Glauco Garcia Scandaroli, Maxime Meilland, and Rogério Richa, *Improving NCC-Based Direct Visual Tracking*, in European Conference on Computer Vision (ECCV), 2012, pp. 442–455. doi: 10.1007/978-3-642-33783-3_32.

[189] Robin Schubert, Eric Richter, and Gerd Wanielik, *Comparison and evaluation of advanced motion models for vehicle tracking*, in IEEE International Conference on Information Fusion (FUSION), 2008, pp. 1–6.

[190] Laura Sevilla-Lara and Erik G. Learned-Miller, *Distribution fields for tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 1910–1917. doi: 10.1109/CVPR.2012.6247891.

[191] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor, *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*, in Field and Service Robotics, Results of the 11th International Conference, (FSR), 2017, pp. 621–635. doi: 10.1007/978-3-319-67361-5_40.

[192] Evan Shelhamer, Jonathan Long, and Trevor Darrell, *Fully Convolutional Networks for Semantic Segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 39 (2017), pp. 640–651. doi: 10.1109/TPAMI.2016.2572683.

[193] Jianbo Shi and Carlo Tomasi, *Good features to track*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1994, pp. 593–600. doi: 10.1109/CVPR.1994.323794.

[194] Horesh Ben Shitrit, Jérôme Berclaz, François Fleuret, and Pascal Fua, *Tracking multiple people under global appearance constraints*, in IEEE International Conference on Computer Vision (ICCV), 2011, pp. 137–144. doi: 10.1109/ICCV.2011.6126235.

[195] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb, *Learning from Simulated and Unsupervised Images through Adversarial Training*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2242–2251. doi: 10.1109/CVPR.2017.241.

[196] Mennatullah Siam, Heba Mahgoub, Mohamed Zahran, Senthil Yogamani, Martin Jägersand, and Ahmad El Sallab, *MODNet: Moving Object Detection*

Network with Motion and Appearance for Autonomous Driving, Computing Research Repository (CoRR), arXiv:1709.04821 (2017).

[197] Leon Sixt, Benjamin Wild, and Tim Landgraf, *RenderGAN: Generating Realistic Labeled Data*, Front. Robotics and AI, 2018 (2018). doi: 10.3389/frobt.2018.00066.

[198] Arnold W. M. Smeulders, Dung Manh Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah, *Visual Tracking: An Experimental Survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 36 (2014), pp. 1442–1468. doi: 10.1109/TPAMI.2013.230.

[199] Pierre Soille, *Constrained Connectivity for Hierarchical Image Decomposition and Simplification*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 30 (2008), pp. 1132–1145. doi: 10.1109/TPAMI.2007.70817.

[200] L. Soler, A. Hostettler, V. Agnus, A. Charnoz, J.B. Fasquel, J. Moreau, A. Osswald, M. Bouhadjar, and J. Marescaux, *3d image reconstruction for comparison of algorithm database: a patient-specific anatomical and medical image database*, 2010.

[201] Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas A. Funkhouser, *Semantic Scene Completion from a Single Depth Image*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 190–198. doi: 10.1109/CVPR.2017.28.

[202] Yibing Song, Chao Ma, Lijun Gong, Jiawei Zhang, Rynson W. H. Lau, and Ming-Hsuan Yang, *CREST: Convolutional Residual Learning for Visual Tracking*, in IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2574–2583. doi: 10.1109/ICCV.2017.279.

[203] Jon Sporring, Mads Nielsen, Luc Florack, and Peter Johansen, eds., *Gaussian Scale-Space Theory*, vol. 8 of Computational Imaging and Vision, Springer, 1997. doi: 10.1007/978-94-015-8802-7.

[204] Carsten Steger, *An Unbiased Detector of Curvilinear Structures*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (1998), pp. 113–125. doi: 10.1109/34.659930.

[205] Carsten Steger, *Similarity Measures for Occlusion, Clutter, and Illumination Invariant Object Recognition*, in DAGM-Symposium, vol. 2191 of Lecture Notes in Computer Science, Springer, 2001, pp. 148–154. doi: 10.1007/3-540-45404-7_20.

[206] Carsten Steger, *Occlusion, clutter, and illumination invariant object recognition*, International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences, 34 (2002), pp. 345–350.

[207] Carstan Steger, *System and method for object recognition*, June 13 2006. US Patent 7,062,093.

[208] Carsten Steger, Markus Ulrich, and Christian Wiedemann, *Machine Vision Algorithms and Applications*, Wiley-VCH, Weinheim, 2nd ed., 2018.

[209] Hao Su, Charles Ruizhongtai Qi, Yangyan Li, and Leonidas J. Guibas, *Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views*, in IEEE International Conference on Computer Vision (ICCV), 2015, pp. 2686–2694. doi: 10.1109/ICCV.2015.308.

[210] Yao Sui, Ziming Zhang, Guanghui Wang, Yafei Tang, and Li Zhang, *Real-Time Visual Tracking: Promoting the Robustness of Correlation Filter Learning*, in European Conference on Computer Vision (ECCV), 2016, pp. 662–678. doi: 10.1007/978-3-319-46484-8_40.

[211] Chong Sun, Dong Wang, Huchuan Lu, and Ming-Hsuan Yang, *Learning spatial-Aware regressions for visual tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 8962–8970.

[212] Weiwei Sun and Ruisheng Wang, *Fully Convolutional Networks for Semantic Segmentation of Very High Resolution Remotely Sensed Images Combined With DSM*, IEEE Geoscience and Remote Sensing Letters, 15 (2018), pp. 474–478. doi: 10.1109/LGRS.2018.2795531.

[213] Siyu Tang, *People detection and tracking in crowded scenes*, PhD thesis, Saarland University, Saarbrücken, Germany, 2017.

[214] Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele, *Subgraph decomposition for multi-target tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5033–5041. doi: 10.1109/CVPR.2015.7299138.

[215] Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele, *Multi-person tracking by multicut and deep matching*, in European Conference on Computer Vision Workshops on Benchmarking Mutliple Object Tracking (ECCVW), Springer, 2016, pp. 100–111. doi: 10.1007/978-3-319-48881-3_8.

[216] Siyu Tang, Mykhaylo Andriluka, Bjoern Andres, and Bernt Schiele, *Multi People Tracking with Lifted Multicut and Person Re-identification*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. doi: 10.1109/CVPR.2017.394.

[217] Steven L Tanimoto, *Template matching in pyramids*, Computer Graphics and Image Processing, 16 (1981), pp. 356–369. doi: 10.1016/0146-664X(81)90046-0.

[218] Carlo Tomasi and Takeo Kanade, *Detection and tracking of point features*, IEEE Transactions on Circuits and Systems for Video Technology, 15 (1991).

[219] Markus Ulrich and Carsten Steger, *Performance Evaluation of 2D Object Recognition Techniques*, Technical Report PF–2002–01, Lehrstuhl für Photogrammetrie und Fernerkundung, Technische Universität München, 2002.

[220] M. Ulrich and C. Steger, *System and methods for automatic parameter determination in machine vision*, May 31 2011. US Patent 7,953,290.

[221] Markus Ulrich, Carsten Steger, and Albert Baumgartner, *Real-time object recognition using a modified generalized Hough transform*, Pattern Recognition, 36 (2003), pp. 2557–2570. doi: 10.1016/S0031-3203(03)00169-9.

[222] Jack Valmadre, Luca Bertinetto, João F. Henriques, Ran Tao, Andrea Vedaldi, Arnold W. M. Smeulders, Philip H. S. Torr, and Efstratios Gavves, *Long-Term Tracking in the Wild: A Benchmark*, in European Conference on Computer Vision (ECCV), 2018, pp. 692–707. doi: 10.1007/978-3-030-01219-9_41.

[223] Jack Valmadre, Luca Bertinetto, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr, *End-to-End Representation Learning for Correlation Filter Based Tracking*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5000–5008. doi: 10.1109/CVPR.2017.531.

[224] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid, *Learning from Synthetic Humans*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4627–4635. doi: 10.1109/CVPR.2017.492.

[225] Tomás Vojír and Jiri Matas, *The Enhanced Flock of Trackers*, in Registration and Recognition in Images and Videos, 2014, pp. 113–136. doi: 10.1007/978-3-642-44907-9_6.

[226] Tomas Vojir and Jiri Matas, *Pixel-Wise Object Segmentations for the VOT 2016 Dataset*, Research Report CTU–CMP–2017–01, Center for Machine Perception, Czech Technical University, Prague, Czech Republic, January 2017.

[227] Liang Wang, Weiming Hu, and Tieniu Tan, *Recent developments in human motion analysis*, Pattern Recognition, 36 (2003), pp. 585–601. doi: 10.1016/S0031-3203(02)00100-0.

[228] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro, *High-Resolution Image Synthesis and Semantic Manipulation With Conditional GANs*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 8798–8807.

[229] Xinchao Wang, Engin Türetken, François Fleuret, and Pascal Fua, *Tracking Interacting Objects Using Intertwined Flows*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 38 (2016), pp. 2312–2326. doi: 10.1109/TPAMI.2015.2513406.

[230] Ziyu Wang, Huafeng Hu, Lefei Zhang, and Jing-Hao Xue, *Discriminatively guided filtering (DGF) for hyperspectral image classification*, Neurocomputing, 275 (2018), pp. 1981–1987. doi: 10.1016/j.neucom.2017.10.046.

[231] Jonathan Weber and Sébastien Lefèvre, *Fast quasi-flat zones filtering using area threshold and region merging*, Journal of Visual Communication and Image Representation, 24 (2013), pp. 397–409. doi: 10.1016/j.jvcir.2013.01.011.

[232] 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*, arXiv preprint arXiv:1511.04136, (2015).

[233] Longyin Wen, Zhen Lei, Siwei Lyu, Stan Z. Li, and Ming-Hsuan Yang, *Exploiting Hierarchical Dense Structures on Hypergraphs for Multi-Object Tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 38 (2016), pp. 1983–1996. doi: 10.1109/TPAMI.2015.2509979.

[234] Longyin Wen, Wenbo Li, Junjie Yan, Zhen Lei, Dong Yi, and Stan Z. Li, *Multiple Target Tracking Based on Undirected Hierarchical Relation Hypergraph*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1282–1289. doi: 10.1109/CVPR.2014.167.

[235] Michel A. Westenberg, Jos B. T. M. Roerdink, and Michael H. F. Wilkinson, *Volumetric Attribute Filtering and Interactive Visualization Using the Max-Tree Representation*, IEEE Transactions on Image Processing, 16 (2007), pp. 2943–2952. doi: 10.1109/TIP.2007.909317.

[236] Michael H. F. Wilkinson, *A fast component-tree algorithm for high dynamic-range images and second generation connectivity*, in IEEE International Conference on Image Processing (ICIP), 2011, pp. 1021–1024. doi: 10.1109/ICIP.2011.6115597.

[237] Bo Wu and Ramakant Nevatia, *Detection and Tracking of Multiple, Partially Occluded Humans by Bayesian Combination of Edgelet based Part Detectors*, International Journal of Computer Vision, 75 (2007), pp. 247–266. doi: 10.1007/s11263-006-0027-7.

[238] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang, *Online Object Tracking: A Benchmark*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013, pp. 2411–2418. doi: 10.1109/CVPR.2013.312.

[239] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang, *Object Tracking Benchmark*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 37 (2015), pp. 1834–1848. doi: 10.1109/TPAMI.2014.2388226.

[240] Yi Wu, Bin Shen, and Haibin Ling, *Online robust image alignment via iterative convex optimization*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 1808–1814. doi: 10.1109/CVPR.2012.6247878.

[241] Yongchao Xu, Edwin Carlinet, Thierry Géraud, and Laurent Najman, *Hierarchical segmentation using tree-based shape spaces*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 39 (2017), pp. 457–469. doi: 10.1109/TPAMI.2016.2554550.

[242] Alper Yilmaz, Omar Javed, and Mubarak Shah, *Object tracking: A survey*, ACM Computing Surveys (CSUR), 38 (2006). doi: 10.1145/1177352.1177355.

[243] Qian Yu, Thang Ba Dinh, and Gérard G. Medioni, *Online Tracking and Reacquisition Using Co-trained Generative and Discriminative Trackers*, in European Conference on Computer Vision (ECCV), 2008, pp. 678–691. doi: 10.1007/978-3-540-88688-4_50.

[244] Alan L. Yuille and Tomaso A. Poggio, *Scaling Theorems for Zero Crossings*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8 (1986), pp. 15–25. doi: 10.1109/TPAMI.1986.4767748.

[245] Kaihua Zhang, Lei Zhang, Qingshan Liu, David Zhang, and Ming-Hsuan Yang, *Fast Visual Tracking via Dense Spatio-temporal Context Learning*, in European Conference on Computer Vision (ECCV), 2014, pp. 127–141. doi: 10.1007/978-3-319-10602-1_9.

[246] Li Zhang, Yuan Li, and Ramakant Nevatia, *Global data association for multi-object tracking using network flows*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008. doi: 10.1109/CVPR.2008.4587584.

[247] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia, *Pyramid Scene Parsing Network*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 6230–6239. doi: 10.1109/CVPR.2017.660.

[248] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Antonio Torralba, and Aude Oliva, *Places: An Image Database for Deep Scene Understanding*, Computing Research Repository (CoRR), arXiv:1610.02055 (2016).

[249] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba, *Semantic Understanding of Scenes through the ADE20K Dataset*, Computing Research Repository (CoRR), arXiv:1608.05442 (2016).

[250] Huiyu Zhou, Yuan Yuan, and Chunmei Shi, *Object tracking using SIFT features and mean shift*, Computer Vision and Image Understanding, 113 (2009), pp. 345–352. doi: 10.1016/j.cviu.2008.08.006.

[251] Jun Zhou, Junping Du, and Suguo Zhu, *Continues target tracking based on NCC and Kalman algorithm*, in IEEE Conference on Cloud Computing and Intelligence Systems (CCIS), 2014, pp. 634–638. doi: 10.1109/CCIS.2014.7175812.

[252] Guibo Zhu, Jinqiao Wang, Yi Wu, and Hanqing Lu, *Collaborative Correlation Tracking*, in British Machine Vision Conference (BMVC), 2015, pp. 184.1–184.12. doi: 10.5244/C.29.184.