Graz University of Technology

Graz University of Technology

Ecole polytechnique federale de Lausanne

PhD Thesis

Tracking-by-Detection using Randomized Online Ensemble Methods

---

Martin Godec

Graz, Austria, 2013

Thesis supervisors

Prof. Dr. Horst Bischof

Dr. Vincent Lepetit

To my small family...

I am so smart, I am so smart, s-m-r-t ... I mean s-m-A-r-t.

Homer Simpson; The Simpsons 05-03
http://www.youtube.com/watch?v=anXKnXVm-Kc

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)

Abstract

Estimating the movement of an object in an image sequence is a very important task in computer vision and is utilized in many different applications, such as human-computer interaction, augmented reality, and video editing. Additionally, tracking of vehicles or pedestrians is an essential functionality in security and surveillance tasks. However, often the type of the object is not known a priori. Thus, knowledge about the object has to be collected during runtime.

In this thesis, we target *Tracking-by-Detection*, a concept that uses an iterative loop of online learning and object detection to establish an object model of the target during runtime. This allows for online accumulation of information about the target object that is collected frame by frame by utilizing machine learning techniques. Since the object appearance will change during runtime, the model has to be adapted continuously and outdated information must be discarded to prevent that the size of the object model grows to infinity.

To introduce the basics of tracking-by-detection, we explain the different building blocks that form the so-called tracking-loop, which defines the processing cycle of the concept. These blocks are object representation, statistical learning, detection, and sample generation, which contribute to the overall performance of the approach and have to be carefully harmonized. We review the individual implementations of these blocks that have been proposed in related work. Additionally, we give an overview about the progress of tracking in the last decade and touch various different concepts.

In an application chapter, we present three approaches that target improvements of the individual parts of the tracking loop and compare each of them to appropriate related work. First, we present Online Random Naïve Bayess (ORNBs), a simple but powerful learning algorithm. We demonstrate the capabilities of the algorithm by performing machine learning experiments and examine its core characteristics. Subsequently, we apply it to tracking-by-detection.

The second application presents a novel scheme of labeling training data during the

tracking process. We propose to dynamically split the background class into a various number of virtual classes, by using an online multi-class learning algorithm and a labeling scheme that is able to select a proper number of virtual classes during runtime. Therefore, we modify the learning algorithm to be able to cope with very unbalanced datasets.

Finally, we present a tracking approach that is able to follow non-rigid objects in complex scenes and avoids the bounding-box restriction of many other approaches. We combine a novel learning algorithm (i.e., Online Random Ferns (ORFes)) with Hough-based object detection, which combines a large number of parts in a center-voting fashion. Using this detection scheme, we create a strong cue for image segmentation resulting in exact determination of the object boundaries while the initial object annotation is still a bounding box! We experimentally justify the additional effort of the segmentation and demonstrate state-of-the-art performance of the proposed tracking approach.

These three approaches give insight into the individual building blocks and their influence in the overall tracking performance. Additionally, they show the wide variability of tracking-by-detection approaches.

Keywords. Computer Vision, Object Detection, Object Tracking, Machine Learning, Online Learning, Object Representation, Image Features, Visual Servoing, Surveillance

Kurzfassung

Die Verfolgung von Objekten in Videos ist ein wichtiger Bestandteil vieler Computer-Vision Anwendungen und spielt speziell in der Videoüberwachung eine entscheidende Rolle. In vielen dieser Anwendungen ist jedoch der Typ des zu verfolgenden Objekts nicht im Vorhinein bekannt, wodurch die Objektbeschreibung erst zur Laufzeit generiert werden kann.

In dieser Dissertation wird dieses Problem durch *Tracking-by-Detection* adressiert, einem Ansatz der Lernverfahren einsetzt, die auch zur Laufzeit weiter trainiert werden können. Bild für Bild wird so eine Beschreibung des Objekts aufgebaut, wobei eine zeitliche Gewichtung der Information erfolgt, um die Beschreibung immer an die aktuellen Anforderungen anzupassen.

Die Abläufe bei Tracking-by-Detection werden durch die sogenannte Tracking-Loop definiert. Sie beinhaltet alle nötigen Verarbeitungsschritte (Objektbeschreibung, Statistisches Lernverfahren, Objekterkennung und Trainingsmechanismus) und bringt diese in die richtige Reihenfolge. Diese Verarbeitungsschritte müssen sorgfältig aufeinander abgestimmt werden, um gute Leistung erzielen zu können. Zu jedem dieser Teile werden verschiedene Umsetzungsvarianten aus der verwandten Literatur vorgestellt.

Als praktische Anwendung von Tracking-by-Detection werden drei Ansätze präsentiert, in denen spezielle Verbesserungen der Verarbeitungsschritte in der Tracking-Loop beschrieben werden. Der erste Ansatz verwendet Online Random Naïve Bayess (ORNBs), einen einfachen aber effizienten Lernalgorithmus. Dessen Eigenschaften werden auf Lerndatensätze und für den Einsatz in der Tracking-Loop analysiert und evaluiert.

Die zweite Anwendung verwendet ein neuartiges Konzept zur Generierung von Trainingsdaten während der Laufzeit. Dieses Konzept spaltet komplexere Klassen automatisch in mehrere virtuelle Klassen auf, sodass die Beschreibung des eigentlichen Objekts in mehrere, einfachere Teile aufgeteilt wird. Dazu wird ein modifizierter Lernalgorithmus

verwendet, dessen Statistik zur Laufzeit erweitert werden kann und der unbalanzierten Datensätzen umgehen kann.

Der dritte vorgestellte Ansatz ermöglicht die Verfolgung von nicht-rigiden Objekten ohne die übliche Einschränkung auf eine rechteckige Objektbeschreibung. Dazu werden Online Random Ferns (ORFes) mit einer flexiblen Objektbeschreibung kombiniert, welche das Objekt als eine Kombination vieler kleiner Teile beschreibt. Durch zusätzliche Segmentierung des Objekts wird ein Pixel-genaues Ergebnis generiert, welches auch zur Generierung der Trainingsdaten verwendet wird.

Durch diese drei Beispiele werden die große Variabilität der Ansätze für Tracking-by-Detection und der Einfluss der einzelnen Vorgehensweisen innerhalb der Tracking-Loop gezeigt. Durch experimentelle Vergleiche zu relevanter Literatur wird gezeigt, dass die präsentierten Ansätze sehr gute Ergebnisse am aktuellen Stand der Technik liefern.

Keywords. Maschinelles Sehen, Objekterkennung, Objektverfolgung, Maschinelles Lernen, Online Lernen, Objektrepresentation, Bildeigenschaften, Visuelle Steuerung, Überwachung

Acknowledgments

First of all, i want to thank Horst Bischof for giving me the opportunity to write this thesis and to provide overall guidance and the financial support that made it possible. Additionally, i want to thank Vincent to act as an examiner and reviewer of my thesis. Of course, this thesis is not the product of one guy sitting in his office and creating new ideas and has been influences by a large number of people that i want to mention.

I am especially thankful to Peter M. Roth for his never ending support, corrections, reading, discussions, and guidance that have paved my way through this thesis, Christian Leistner and Amir Saffari for giving me the opportunity to work on many fascinating publications and to introduce me to their way of doing research and writing publications, Sabine Sternig for long-lasting discussions, code sharing and reviewing, publications and to walk on the same paths of research for some years, my colleague Thomas Mauthner, Martin Hirzer, Peter Kontschieder, Hayko Riemenschneider, Markus Heber, Mike Donoser, Martin Köstinger, Paul Wohlhart, Christian Reinbacher, Samuel Schulter, and Andi Wendel for many, many discussions during our outside of our reading group, Bernhard Rinner and his team for their good work and climate within our research projects, Helmut Grabner for motivating me to do my master's thesis in computer vision, and all the colleagues from ICG (including the above mentioned) for having a good time, the one or other beer, always someone that listens to your problems and for having a lot of fun together!

Beside the support within work, there are also several people outside work that are worth to be mentioned here, especially my family that gave me the opportunity to go to the university and my friends from in- and outside the university. But most important, i have to mention my small family, Katja, Elena and Nora, that filled my non-working hours with life, accepted my immanent confusion and have always brought me back down to earth with my thoughts.

Contents

Chapter 1

Introduction

Contents

Already in the ancient history, humans dreamed of machines that interact with us or assist in complex tasks. In the Hellenistic civilization, the *Automaton* was a machine that performs a specific task autonomously, but the concept of the *humanoid robot* did not evolve since the modern age. Such robots have been a frequent theme in science fiction stories. Thus, nearly every kid knows examples of this type of robot, such as *C-3PO* from *Star Wars* or *Lieutenant Commander Data* from *Star Trek*. Additionally, there exist different variations that look more like robots, such as *Johnny 5* from *Short Circuit* or *WALL-E* from the film of the same title.

All of these robots share the characteristic that they try to copy the human senses as they mainly use visual information to understand their environment[1]. Thus, they try to copy the ability of the visual cortex. However, in contrast to computer vision algorithms that are developed by scientists, the human visual sense evolved in thousands of years

[1] In the computer vision community, this capability is denoted as *scene understanding*.

to it's present state. Additionally, the knowledge about the processes within the human visual cortex and brain is rather limited and computers that are capable of complex image processing in real-time exist only since a decade. Thus, we cannot simply build a system that copies our own visual capabilities.

Yet, how far is the development of computer vision algorithms that target such tasks?

While autonomous robots in the sense of the examples mentioned above are far from real, there exist many applications where computer vision successfully assist humans. Probably the most popular of these applications is face detection, nowadays integrated into nearly every consumer camera helping us to take better images. Others are Optical Character Recognition (OCR), which is integrated into every scanning and image processing software, driving assistance, such as lane keeping or traffic sign detection, and surveillance systems which help, for instance, to detect persons and vehicles in public places. Especially for the latter one, automatic or semi-automatic computer vision techniques help to process that vast amount of visual information that is captured every day, which could only hardly be done by humans.

Of course, such tasks are complex and require sophisticated computer vision techniques. Thus, as with any complex computing system, computer vision applications consist of many different building blocks, where especially detection and tracking of objects are very important for video analysis (e.g., in surveillance applications). Additionally, such real-world applications require robust and fast methods because more and more information is generated in form of videos.

In particular, this thesis focuses on tracking of arbitrary objects in a natural environment. This means that we want to determine the location of an object within an image sequence, where the term location is not limited to the position of the object in the image, but may also include the size, shape, orientation, and many other parameters. Thereby, we take real-time capabilities speed, robustness, scene adaptation and online learning into account.

In general, object tracking is a task that may be used in various applications:

Focus Assistance Consumer cameras provide several features to improve the quality of the taken image (e.g., smile shutter, face detection). Keeping a moving object in focus to enable sharp images under difficult conditions can help hobby photographers to further improve their images.

Image Stabilization In sports broadcasts, the target object may move quite fast. Track-

ing of the sportsperson, ball, and puck can be used to stabilize the trajectory of the camera movement to retrieve smooth videos. Additionally, when tracking all sportspersons of a team, insight about the strategy can be obtained.

Surveillance In public space, analysis of the stream of pedestrians and vehicles is very interesting to optimize security or traffic flow. Manual recording of such streams is very tedious and time-consuming. This makes automatic tracking of the objects an interesting alternative.

Driver Assistance The vehicle tracks all pedestrians that are moving around the vehicle and alerts the driver of critical situations.

This short and incomplete list illustrates the large spectrum of applications where tracking can be used.

All tracking approaches have in common that we want to estimate the motion of some object. From an application point of view, there are three types of motion estimation: (a) motion estimation for every individual *pixel* in the image, also known as *optical flow* (e.g., [46]), (b) motion estimation of specific points in the image (i.e., *key-points*) as used in structure-and-motion (e.g., [15]), and (c) *object* specific motion considering all kind of appearance changes.

In this thesis, we focus on object tracking, especially avoiding any assumptions about the object appearance, category, size or shape. In short, we define object tracking as follows:

> Object tracking is to estimate the location (e.g., position, size, shape, and orientation) of an object in every frame of an image sequence, considering motion, appearance changes, occlusion, clutter, changing illumination, and other challenges, given a single object annotation in the first frame of the sequence.

Beside the simple and sketchy definition of the problem, there is a large number of boundary conditions that divide the problem space into numerous fragments:

- Tracking of *natural objects* or image patches
- Knowledge about the *category of objects* we want to track
- *Changing appearance* of the object over time
- *Rigid* or *deformable* object structure

- *Degrees of freedom* of the object motion that should be estimated
- *Changing background* due to camera motion
- ...

Hence, it is obvious that the term *object tracking* delineates a problem with many different facets, where a prominent concept is **tracking by detection**.

1.1 From Object Detection to Object Tracking

As already mentioned, one of the most known computer vision applications is face detection, which is integrated in nearly every consumer camera or smartphone today to optimize the camera settings (e.g., focus, white-balance, shutter speed). This is even more impressive since there has been just one decade since the first real-time enabled approach has been presented [153]. In the meantime, camera manufacturers developed the next functionality that simplifies the users life called *focus tracking* or *AI servo*. This means that an arbitrarily selected region in the image will be in focus whether the camera is moved or not, where a visual tracking algorithm tries to determine the current position of that region. However, this functionality does not perform as good as face detection up to now and is limited to rectangular image patches that have a stable appearance.

Wouldn't it be great to establish such a functionality to automatically track and focus on arbitrary objects while their appearance changes?

Of course, there are many different approaches to follow the movement of an object and implementations, such as focus tracking, combine several techniques to achieve the required robustness. Tracking-by-detection tackles the task in the following way: Sample images of the object are acquired and a model is estimated that can detect the object using the collected knowledge. This is very similar to established detection algorithms that gain knowledge about the target object category (e.g., faces or cars) using a large database of sample images. However, detection and tracking differ in time-scale of sample acquisition and model generation. E.g., the human face does not significantly change during the runtime of an usual tracking application, but considering the environment, such as lighting, shadows, and 3D movement, the object may look very different even after a few frames. Therefore, we perform tracking as repeated re-detection of an object based on acquired knowledge about the object and iterative accumulation of that knowledge from already processed frames. Thereby, the recent appearance of the object is most important.

1.1.1 Tracking Challenges

Having in mind the different time-scales of tracking and detection, we will face various problems during runtime. We have separated those into two groups, intrinsic and extrinsic challenges:

Intrinsic Challenges are caused by the nature of 3D objects that are projected to a 2D image, such as their inherent **non-rigidity**. Many approaches do not consider this problem as they use rectangular, highly rigid object representations. Additionally, many benchmark datasets include many simple sequences where only translational movement of the object is present. However, this scenario is very unlikely under real world conditions.

Beside the deformation that non-rigid objects may undergo, also non-convexity may cause **self-occlusions**, which are hard to model inherently in currently used object models. Moreover, the **appearance** of the object may change due to **out-of-plane rotations** resulting in a change of the view on the object.

Also **in-plane rotation** and **scaling** of an object result in drastic changes of the object's appearance when using an approach that does not handle scaling and rotation. The same also applies for changes of the **aspect ratio** of the object. From a camera perspective, fast motion may generate **motion blur** and changing frame-rates will result in **non-smooth motion**, which may decrease the tracking performance.

Extrinsic Challenges are caused by the interaction between objects and their environment. We already mentioned **appearance** changes caused by out-of-plane rotations of the object. However, they can also be caused by **illumination** changes which can change color, contrast or texture of the object significantly. Also shadows caused by the environment or by the object itself can confuse tracking approaches due to partial appearance changes.

When objects interact with their environment, partial and full **occlusions** may occur. While partial occlusions can be handled well by part-based approaches, they are still critical concerning the information that is collected to enhance the model of the object. On the other hand, full occlusions and objects leaving the field-of-view require robust re-detection mechanisms. There is a smooth transition between those cases depending on the granularity of the model of the object, which makes it an even problem.

Another challenge that may cause immediate tracking failures are **cluttered background** or the presence of **similar objects**. This is especially important if multiple instances of the object category occur within the scene or the object category occurs very frequently, such as pedestrians or cars. This can be handled using strong prior knowledge, such as a pedestrian detector which changes the problem to discrimination of one pedestrian from others that are present in the scene. However, this requires information about the expected object category that should be tracked.

To summarize, tracking approaches have to distinguish between valid object transformations, such as

- deformation,
- rotation in- and out-of the image plane,
- changing appearance, and
- changing illumination,

and invalid transformations, such as

- occlusions,
- target confusion, and
- drift (i.e., slight inaccuracies of the object detection).curacies of the object detection).

Many different concepts to handle this problem have been proposed in recent years, mostly using **online adaption** of an object model or classifier. Despite of all advances that have been made, the overall task is still far from being solved.

1.2 The Tracking Loop

Defining tracking as an repeated re-detection of an object directly leads to a number of building blocks that are required within a tracking system. Thinking merely of object detection, these blocks are (a) the **representation** that extracts meaningful knowledge from the raw image data (i.e., image features), (b) the **statistical model** that describes the object properties regarding the representation and (c) the **detection mechanism** that locates the object within an image using the description within the statistical model.

However, in tracking the knowledge about the target object is usually limited, which means that we may observe changes of the object appearance that are novel, i.e., that have not been seen before. Thus, we need to adapt the model during runtime to cover such changes which leads to one additional building blocks (d) **sampling and labeling** of training data during runtime and the statistical model must be extended to support **online model update**. This results in the overall **tracking loop** illustrated in Figure 1.1, which describes the required steps for each iteration (i.e., for each frame).

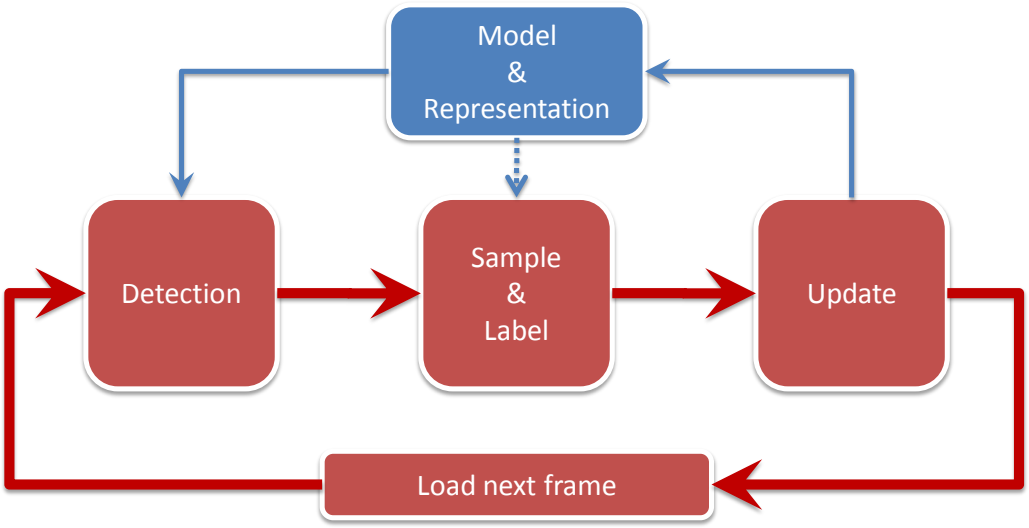


Figure 1.1: The main building blocks of a tracking-by-detection system that define the **Tracking Loop**. Starting from the *detection* of the object, we extract new *samples* and *update* the current *object model*. This procedure is repeated for every frame of the sequence.

During the last decade, a large number of approaches following this loop have been developed, adding different modifications and extensions. Overall, the goal of this concept is to learn a statistical model of the object with respect to the used object representation. The representation is given by the used image features and their spatial arrangement. The statistical model and the representation are not completely independent, as the model has to be able to capture the complexity of the representation.

Most publications focus on statistical modeling and propose novel learning algorithms to solve this task. However, the statistical model can only be as good as the data is that is used to establish it. If the training strategy selects poor training samples, this

will influence the tracking performance tremendously, even if the sample labeling itself is correct. Therefore, in this thesis we will deal with all parts of the tracking loop, including sampling and labeling of training samples, which is often totally ignored. The following sections describe the individual building blocks of the tracking loop and list concepts that have been used in popular approaches.

1.2.1 Object Representation

The object representation defines the combination of *visual* and *geometric* information of the object. This combination allows to interpret the raw image information and to extracting meaningful information that can be processed using statistical modeling or learning techniques (see Section 1.2.2) afterwards. However, since both visual and geometric information can change over time, strict separation of representation and model often difficult.

The most important attribute of image features is that they are distinctive for the tracked object. This is easy if the target object or object category are known in detail. Then, feature selection algorithms can perform a pre-selection resulting in a tailored and high-quality feature pool. However, this is not possible for tracking of arbitrary objects and feature selection has often to be done during runtime. Anyhow, a single feature type is often not able to describe an object in a distinctive way, especially when the environment (e.g., background and illumination) is changing. Therefore, many approaches generate pools of heterogeneous features and select the ones that perform best regarding the current settings. Also processing speed is of high importance for visual tracking which makes the computational complexity of features very relevant. Therefore, complex filter operations (e.g., Gabor-filters [35]) or normalizations (as used in, e.g., HOG [34]) cannot be used for real-time applications.

Recently, there is a raising interest also to incorporate geometry information that is captured within the representation. While traditional approaches use global descriptions, such as color histograms or templates, there is a clear trend towards including of fine-grained and flexible geometric information. While global, pixel-based representations allow for efficient and simple modeling of the object's appearance, they do not include any geometric information. Patch-based representations using features which capture a large portion of the object (e.g., Haar-like [68, 119, 153]) respect the geometric composition of an object but limit the flexibility of appearance changes. On the other hand, fully part-based representations need computationally expensive inference to establish both overall result

and individual placement or parts. Thus, there has to be a trade-off between flexibility of the model, geometric information and computational complexity.

1.2.2 Statistical Model and Learning

Having a good object representation is only the half way to success, the extracted information has also to be modeled in a sophisticated way. While raw feature values could be directly incorporated into statistical models, such as histograms, current approaches use sophisticated learning techniques to extract the essential information about the object. Basically, the main intention is to establish a statistical model of the object that allows to map input information (i.e., observed properties) to a value that indicates of the input represents the object (i.e., object class). To accomplish this, there are two ideologies that can be described as the following:

Generative approaches establish a joint probability distribution of all object properties and a prediction can be made by picking the object class that most likely created the observed properties.

Discriminative approaches directly model the dependent probability of the object class and the object properties, which leads to a direct prediction of the object class based on the observed properties.

While these descriptions sound complex, the main difference is easy to explain: Generative models describe the properties for a specific object class, independent of any other information, while discriminative models establish a direct mapping from the observed information to the desired object class information. This also explains the difference in getting to a prediction using the different models. A mathematically more sophisticated description of generative and discriminative classification can be found in [108]. Both concepts have advantages and disadvantages and there have been several attempts to fuse these two ideologies (e.g., [40, 85, 150, 158]).

Even though *generative* and *discriminative* methods are conceptually quite different, both finally calculate a score (e.g., a similarity score, probability, likelihood, or confidence value) for candidate regions in the image to belong to a certain object class. By maximizing (or sometimes minimizing) this measure the current object position can be estimated.

Within this thesis we primarily focus on discriminative, learning-based methods, as commonly used in recent tracking approaches. In Chapter 3, we describe randomized

online ensemble methods, i.e., Online Random Forests (ORFs), Online Random Ferns (ORFes), and Online Random Naïve Bayes (ORNB) and their application to tracking.

1.2.3 Detection and Training

The final goal of object tracking is to determine the object's position in the next frame. Depending on the object model there are different solutions. The simplest way to estimate the position of the object is to select candidate positions and to calculate a similarity score to the object model. The use of such a straight-forward technique to estimate the object position is motivated by the high complexity of current object models, where no closed-form solution exists. However, the complexity of this search-problem grows with the degrees of freedom that are used to describe the position of the object (i.e., coordinates, width, height, scale, rotation,...).

Therefore, several simplifications and assumptions have been integrated into the sampling process to enable real-time performance. To reduce the size of the search space, most approaches assume continuous motion, which means that the object cannot jump from one position to a completely different one. Besides the computational reasons, this assumption is based on physical constraints and also reduces the complexity of discriminative models because only a limited amount of background has to be separated from the object. Additionally, natural motion is smooth because of inertia of masses of real-world objects[2]. To further reduce the computational effort, special sampling strategies, such as particle filtering [4], are applied in many approaches.

Another important point in object tracking is the continuous improvement of the object model. Therefore, the statistical model has to be adapted to perform optimally under the current environment. In discriminative, learning-based approaches this means that positive and negative training samples have to be generated. Hence, it is obvious that good sample generation is an important within the tracking loop. However, in most approaches this issue is neglected or only discussed insufficiently. In Chapter 4, we give a comprehensive review of related work regarding sampling and sample generation in tracking approaches.

[2]This assumption does not hold for, e.g., comics, which explains that tracking approaches usually fail on such sequences.

1.3 About this Thesis

In this thesis, we focus on the tracking concept *tracking-by-detection*. The concept uses learning-based object detection approaches and extends them to the online domain. This means that the detector is not trained before runtime using a large amount of labeled training samples, but during runtime. To adapt to the current object appearance it learns on a frame by frame basis and optimizes the representation and the statistical model based on it's own prediction from previous frames.

This concept is the starting point of the thesis. Based on an comparison of rather similar approaches [60–62] we wanted to investigate the key performance parameters of the tracking loop more detailed. Therefore, we present, analyze and discuss a large amount of concepts that have been used in related work. This should give a detailed overview about the current state-of-the-art. Thereby, we found many limitations and drawbacks of the related literature that ignore several properties of tracking-by-detection, such as a noise in sample generation process, unstable foreground appearance of the object or the presence of background information in the rectangular object patch. This motivated us to think further on the individual building blocks and to come up with solutions that do not ignore these properties.

1.3.1 Structure of this Thesis

In this chapter, we introduced tracking-by-detection and described the *tracking loop*. In the following chapters, we will examine the individual parts of the tracking loop and discuss the implementation of this building block in different tracking approaches.

In Chapter 2 we discuss common representations that are used for tracking-by-detection. We interpret the representation as a combination of image features and a geometric model that defines the relation of the used features. Of course we have a look at the important properties runtime and discriminative power and the selection of useful features. Chapter 3 introduces the basic mathematical foundations of randomized ensemble learning algorithms. Therein, we describe the theory of Random Forests (RFs) and Random Ferns (RFes) that are important techniques in randomized learning. Furthermore, we develop novel online formulations of Random Ferns (RFes) and Random Naïve Bayess (RNBs) that are used in the presented applications. Sample generation and detection of the target object are the topic of Chapter 4, a very important point that is neglected in most scientific publications. However, we think that the sample generation crucially influences the tracking performance and directly links to the robustness of the

learning method.

Chapter 5 addresses three specific implementations of the tracking-by-detection concept. These applications focus on different blocks of the tracking loop and show that these blocks are not independent and influence each other. Finally, we conclude this thesis in Chapter 6 and give an outlook on future trends in tracking-by-detection.

We awarely did not structure this thesis by concatenating several publications of the author because we wanted to give a wide overview on the topic tracking-by-detection. Therefore, we integrated the contribution of the author in the more general discussions. However, the main contributions of the author are summarized at an application level in Section 5.3.1. At last, this thesis should give a wide overview on tracking approaches based on the tracking-by-detection concept which is a very flexible and powerful concept to successfully tackle many sub-problems of the infinitely large area of object tracking.

Chapter 2

Object Representation

Contents

In contrast to humans, computers have a tough time interpreting the visual information that they receive via digital cameras. Therefore, the extraction of useful information from raw pixel intensities is important for every computer vision application. Additionally, the geometric combination of the extracted image information is essential. In this thesis, we focus on appearance-based methods.

In object detection, the most popular example for a highly sophisticated model is the *deformable parts model* [41] that combines gradient-based information in a geometrically flexible manner. While they showed impressive performance, such complex models require a very large amount of information (i.e., training samples) about the object.

Tracking of unknown objects requires a different thinking to establish a good representation. Especially the lack of knowledge about the object requires flexibility of both parts to keep all options open. Therefore, tracking-by-detection approaches often use a mixture of diverse features. Additionally, when tracking non-rigid objects, the geometric characteristics of the target object may change significantly and flexible re-configuration might be necessary. Therefore, we will focus on established geometric models for tracking in this section. These models define the geometric configuration of the extracted image features and the flexibility of this configuration.

A very basic approach in tracking is the use of templates [17, 69, 100] to describe the object's appearance. Such methods estimate the transformation parameters (i.e., translation, scaling, affine, homography) by comparing the stored template to the current image and to establish the object's current position. Template based approaches have been frequently used in computer vision and thus, their behavior is well-known. However, one can expect that template-based approaches show limited performance when applied to highly non-rigid objects.

Another popular concept is followed by kernel-based methods that establish a global model of the object according to the shape of a kernel [9, 29, 123] rather than to individual pixels. Such methods are very efficient and have been very popular because of their simplicity and speed. However, they do no integrate any structural information of the object. This is an advantage for decreasing the variance of the object appearance regarding scaling and rotation, but also decreases the discriminant power of the object model.

More recently, patch-based appearance models using set of features became popular in the tracking community [12, 55, 61, 133]. Using learning-based feature selection, these approaches are similar to the work of Viola and Jones [153]. Since these features have a fixed position within a rectangular patch, their geometric relation is integrated into the model. While the feature positions are fixed, basically, the learning approaches have been used to select the most discriminative features out of the given pool, which introduces flexibility to deformations of the object. Therefore, this description can also be interpreted as a *collection of parts*, where only stable parts are selected to describe the object.

Recently, the flexibility of the object description has been increased further by integrating dedicated geometric relations of object parts [82, 103, 139], where parts can be shifted during runtime. These approaches usually use template-based or kernel-based descriptions of the individual parts, but this concept has been combined with segmentation algorithms to determine the object shape more accurately [26, 40, 56].

2.1 Geometric Models

As denoted in the Introduction, we define the geometric model as the geometric structure of the used image features, where different properties, such as the initial construction or dynamic restructuring are important. Especially in the case of real-world object that may deform during runtime, geometric models have a high influence on the capabilities of a tracking approach. While template-based approaches have a long tradition in computer vision, current developments clearly go towards flexible, part-based models. Especially in recent years, the complexity of such models has significantly increased. In this section, we discuss recent approaches and explain their advantages and drawbacks.

2.1.1 Template-based Models

Traditional tracking methods often rely on image templates, where the fundamental idea can be dated back to the method of Lucas and Kanade [100]. The tracking task is typically solved by detecting and matching or by correlating salient features from consecutive frames, which allows the estimation of a transformation matrix.

Benhimane and Malis [16] introduce a homography-based approach to image-based visual tracking and servoing. They propose to use an efficient second order minimization method to estimate a homography between consecutive and relies on an iterative minimization technique. Another prominent approach is Incremental Visual Tracking (IVT) by Ross et al. [128], which uses an incremental algorithm to calculate a subspace representation of the object. Their incremental Principal Component Analysis (PCA) efficiently updates the Eigenbases (see Figure 2.1) and allows for continuous registration of the model to the current image in position, scale and rotation.

Bolme et al. [20] proposed an approach based on adaptive correlation filters. They use a Minimum Output Sum of Squared Error (MOSSE) filter for correlation, in order to obtain stable and compact peaks for the object detection. Recently, Sevilla-Lara and Learned-Miller [138] introduced *Distribution Fields* for tracking. They split the template into several layers based on the gray-scale values of an image and perform spatial smoothing within the layers individually. This preserves fine structures while enabling slight deformations of the object. However, most real-world objects are inherently non-rigid or perform complex deformations, e.g., for viewpoint changes which are hard to cope with using template-based approaches.

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

Figure 2.1: Sample results for **Incremental Visual Tracking (IVT)**: Tracking result (first row) for four sample frames, sample mean, tracked image region, reconstructed image with the mean and Eigenbases, and the reconstruction error (second row), 10 largest Eigenbases (third and fourth row) (images from [128]).

2.1.2 Kernel-based Models

In contrast to template-based models, kernel-based models basically ignore the geometric structure of the object completely. However, the introduce the shape of a kernel function into the representation. While template based models describe every pixel and its concrete position, kernel-based models describe the whole object at once, based on the shape of a kernel. This means, that the object is described using an overall image statistic, such as a color histogram, calculated on a parametrized kernel function. In some approaches, the kernel's parameters, such as size and rotation, are additionally optimized based on the current image and the object's statistics. These methods are very popular because of their simplicity and computational efficiency. One of the most popular approaches is *Mean-shift tracking* by Comaniciu et al. [29]. They model the object using a color histogram and use an ellipse as a kernel.

In their $\mathcal{WSL}$-tracker, Jepson et al. [75] represent the object using an elliptical kernel. Additionally, they separate the description into the three components (a) stable features (i.e., $\mathcal{S}$table), (b) transient features (i.e., $\mathcal{W}$andering), and (c) noise (i.e., $\mathcal{L}$ost). As features they use filter responses of a steerable pyramid and the object motion is calculated using a weighted combination of stable and transient features, but this representation has been used also with other kinds of features that can be calculated fast given a specific kernel. One of the most recent features that has been used for kernel-based tracking is

the covariance descriptor [151] and its Euclidean approximations [72, 79], respectively.

Several approaches separated the global object description into several parts that are described individually. For example, Adam et al. [1] use 5×2 regions that are individually modeled with color histograms. This separation preserves the geometric relation of the extracted statistics with respect to the object's structure, which is especially useful when dealing with partial occlusions. As an alternative, Kluckner et al. [79] separate the object into left, right, upper and lower part, as also proposed in [96] (see Figure 2.2). This separation allows to handle local variations and partial occlusions more robustly and can also be interpreted as part-based model (see Section 2.1.4).

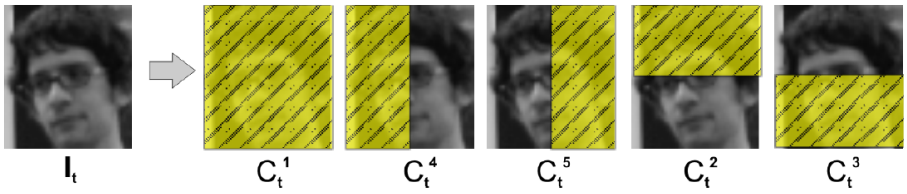


Figure 2.2: Representing the object as multiple overlapping parts increases the robustness regarding partial occlusions (images from [79]). C^1 represents the global description of the object, while $C^2 - C^4$ enables robustness to partial occlusions.

Additionally, many segmentation-based tracking approaches can be categorized as kernel-based methods, but using a dynamic kernel shape rather than a fixed one (i.e., elliptical or rectangular). Bibby and Reid [18] describe the tracking problem within a probabilistic framework. Using pixel-wise posteriors they model the fore- and background appearance and the object contour jointly. However, the high complexity of their theoretic framework makes it computationally infeasible. Thus, they separate the tracking process into registration, level-set segmentation, and online appearance learning for continuous refinement of both object and background models. As a limitation, they use global color histograms to describe the object.

Fan et al. [40] use several different techniques to describe the object. They combine salient points, bag-of-patches and a kernel-based model based on discriminative colors to generate scribbles for object and background that are used to perform image matting (e.g., [93]). There also exist many other segmentation-based approaches, but they either need prior knowledge about the object or object category (e.g., [31]), use only very simple object appearance models limiting the discriminative power of the model (e.g., color histograms [127]), require offline processing of the sequence (e.g., [67, 148]), or are computationally too complex to allow for real-time applications (e.g., [106, 165]).

2.1.3 Patch-based Models

Patch-based representations have been the common representation concept in object tracking for the last decade and are an extension to the kernel-based description. Basically, they use a rectangular kernel to capture the object dimensions, but instead of using a global description within the kernel, they use local features[1] the represent also the geometric properties of the object (see Figure 2.3).

Figure 2.3: **Patch-based appearance model**: A large number of efficient image features are placed at fixed positions within a rectangular bounding box (images from [153]). A learning-based feature selection is applied to reduce the over-complete feature set to a reasonable pool that is optimized for the current object appearance.

Based on the detection framework by Viola and Jones [153], many tracking approaches used this scheme consisting of a number of features that are placed within a rectangular patch. Grabner et al. [61] used online Adaboost to learn the object's appearance during runtime. At each stage of the boosting algorithm, they select the best rectangular feature within a *selector*. The features itself are randomly placed within a rectangular patch and have a fixed position, size and type regarding the rectangular patch. In their approach they use a selected feature set of 50 features, where each feature is selected from an individual pool of about 50 features (i.e., they select 50 out of 2500). Their concept has been extended to various learning concepts [12, 62, 91, 133]. Kalal et al. [77] uses the patch-based geometry but with slightly different features. Instead of intensity differences, they model oriented gradients using combined horizontal and vertical Haar-like features (see Figure 2.4). These features result in two binary decisions that are used within decision trees.

[1] There are other interpretations of patch-based appearance model, but we refer to the feature-based concept introduced by [153] and [119].

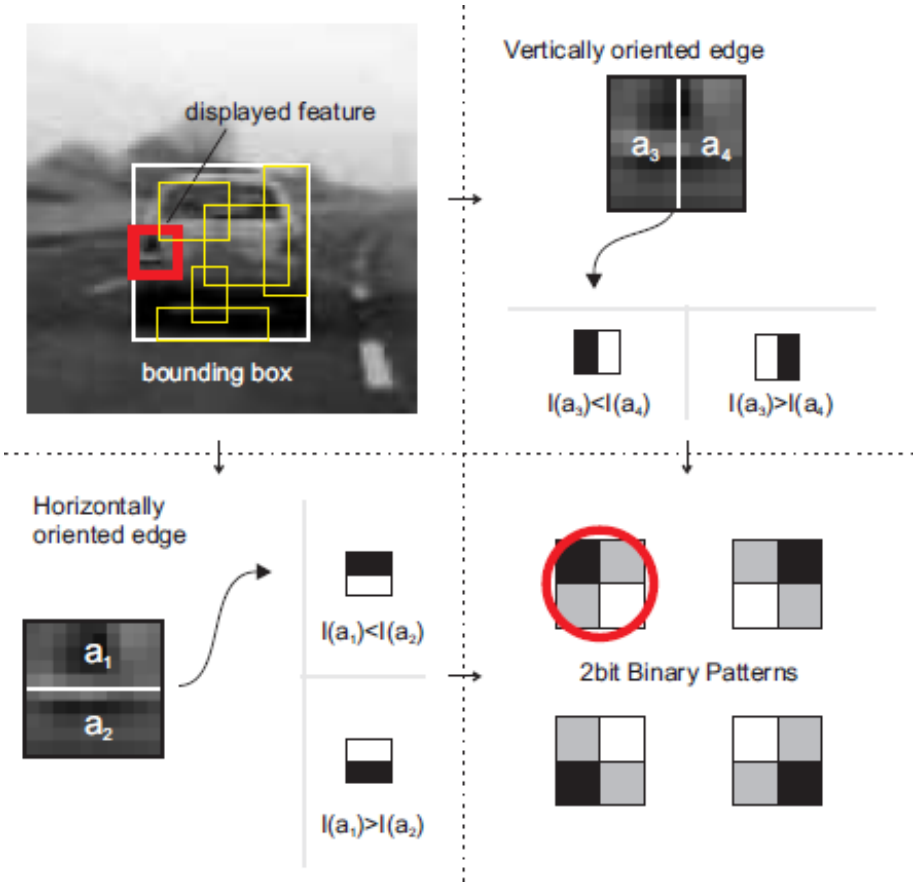


Figure 2.4: **Patch-based appearance model**: Kalal et al. model gradient images at selected rectangular positions (images from [76]) by using combined horizontal and vertical Haar-like features.

2.1.4 Part-based Models

More recently, the granularity and the flexibility of the geometric model became more and more important. Many of the approaches listed above can be seen in the context of part-based representations. For example, Adam et al. [1] and Kluckner et al. [79] separate the rectangular patch into several smaller parts and use individual statistical descriptions for these parts. Kluckner et al. [79] use overlapping blocks (upper and lower half, left and right half and the overall patch) to allow for stable tracking during changes of the object's appearance, while Adam et al. separate the object into $n \times m$ non-overlapping parts to tackle partial occlusions.

Also many patch-based approaches (e.g., [12, 61, 77, 91, 133]) can be interpreted as part-based models, if each part is described by a single feature. However, these approaches

are usually not able to shift, remove and add new parts during runtime, which would respect the non-rigid deformations of real-world objects. However, these approaches tackle such transformations by increasing the learning rate that adapts the statistics to the current object appearance.

Traditional part-based methods in object detection [3, 23, 30, 41–43, 45, 87, 156] have shown impressive results for object detection but are computationally too expensive to be used in real-time applications, such as tracking. Therefore, simplifications have been applied to speed up the process. Sigal et al. [144] use a *loose-limbed* model to decrease the effort for combining human body parts in a tree-like structure. Their approach combines votes from individual body parts regarding the position of the torso, but is only applicable for humans in a multi-camera setup.

Schindler and Dellaert [136] track a fixed number of parts in a particle-filter setup. They integrate the shape of the object to decrease the dimensionality of their constellation model and to use a particle filter. However, their model parameters are trained from a fixed training set. Thus, their approach is limited to a specific class of objects and no parameter update is performed during tracking. Mauthner et al. [103] use a hierarchical setup to model both the individual particle appearance and the spatial arrangement of the parts, respectively. The individual parts are tracked by particle filters using a kernel-based appearance model.

Nejhum et al. [139] use blocks of appearance histograms and shape descriptions to model articulated objects. Additionally, they use a rough segmentation to find the object outline and re-arrange the blocks to maximize the overlap and similarity to the current object appearance and shape. However, they assume stationary foreground appearance and cannot cope with significant appearance changes of the object during tracking. Kwon and Lee [82] use a number of object parts that are automatically renewed during tracking and track the geometric relations of these parts over time (see Figure 2.5). Additionally, to reduce the computational complexity they apply Basin Hopping Monte Carlo (BHMC) sampling. They use a template-based method to model the individual parts and adjust the number of parts during runtime of the tracker dynamically.

In the presented application, we use the generalized Hough-transform to retrieve the object center. Thereby, the voting-based aggregation of small patches can be interpreted as a multi-part object model. Additionally, a segmentation is used to establish a pixel-accurate separation of parts that have a stable geometric relation.

Figure 2.5: Sample results for **BHMC**: Kwon and Lee use tracking of multiple template-patches to establish a flexible bounding-box over the object (images from [82]). They update the appearance, position and number of blocks that are used during a video sequence.

2.1.5 More Geometric Models

Another branch of part-based models is related to the bag-of-words concept. This means that there is no real geometric relation between the parts expect their co-occurance within a given region. Beside those, there are many concepts that heavily rely on the geometric information of the object, e.g., using a CAD model, or that construct a full 3D model of the object. However, such approaches either require a large amount of information about the object that is not given in our task or they require a static scene. Therefore, we do not discuss such approaches in this thesis.

2.2 Image Features

While geometric models connect parts of the object to establish a powerful representation, image features transform the raw pixel intensities into more meaningful information. Depending on the presented application, researchers have proposed both, well-known and novel image features for the use within tracking. Thus, a comprehensive list of features would go beyond the scope of this thesis. However, we give a short introduction on the features that are used in the approaches presented in Chapter 5. Additionally, we take a glance at several other image features that have been used in recent and popular approaches.

The most desirable characteristics of image features are (a) **calculation speed** and **high discriminative power**. As with geometric models, the lack of knowledge about the object may require the **combination** of different image features within one representation. Often, this is handled by using feature selection and a heterogeneous feature pool. However, in the subsequent discussion of approaches we completely ignore that

many implementations couple features, learning algorithms and geometric models tightly.

To enable fast computation, most approaches use intermediate representations, such as integral images [33] of intensities or gradients. These representations are only calculated once per image to speed up the computation.

2.2.1 Integral Images and Histograms

Calculating sum, average or histograms over rectangular regions would be inefficient if every pixel has to be addressed for every feature again and again. Thus, many descriptors can be sped up using integral images (i.e., summed area tables [33]). An integral image, denoted as II, basically is a lookup table

$$II(x,y) = \sum_{0 \le x' \le x} \sum_{0 \le y' \le y} I(x',y'), \tag{2.1}$$

where I is the actual image and x' and y' iterates over the rectangular area from image coordinates $(0,0)$ to (x,y). This lookup table can be calculated in a single pass over the actual image I, which makes the computation very fast. Using II, the sum over rectangle with corners (x,y) and $(x+w, y+h)$ can be calculated as

$$\sum_{x \le x' \le x+w, y \le y' \le y+h} I(x',y') = II(x,y) + II(x+w,y+h) - II(x+w,y) - II(x,y+h) \tag{2.2}$$

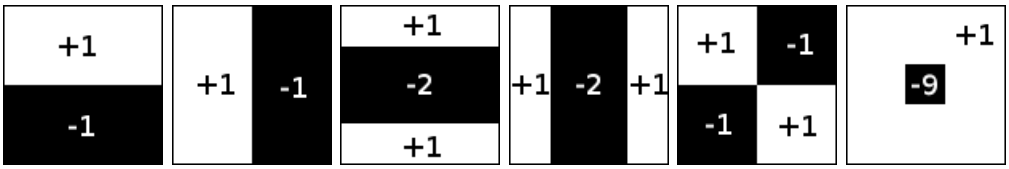
accessing only 4 image coordinates (the corners of the rectangle) of the integral image II instead of $w \cdot h$ coordinates of the image I. w and h are the width and height of the rectangular area, respectively. Since the number of calculated features is usually very large, the cost of creating the integral image can be neglected[2]. While the original version of Integral Images [33] is applicable to rectangular areas, there also exist versions to compute the sum over rotated rectangles ([104] for 45° or even for arbitrary rotations [121]). Naturally, integral images can be extended to more complex structures which allows to efficiently calculate, e.g., histograms over regions [122].

2.2.2 Haar-like Features

Using single image intensities to describe an object is prone to noise and slight deformations of the image. Papageorgiou et al. [119] discussed features based on Haar-wavelets [68] and

[2]The break even point of integral images is reached if the area of the calculated features exceeds the area of the image, which is easily given if several thousands of features are calculated.

Viola and Jones [153] finally used them for object detection in combination with integral images for faster computation. Haar-like features are a very simple kind of wavelet filter that calculates the difference of the sum of two or more rectangular patches in an image. Due to their extend, they are not too sensitive to slight shifts, and due to the difference, they are also quite robust to illumination changes. Figure 2.6 lists some possible types of Haar-like features. There also exist Haar-like features that are rotated by 45° or even consist of polynomial shape that can be calculated efficiently [95, 121]. Kalal et al. [77] use Haar-like features in a slightly different way by simply separating a square into upper, lower, left and right halves and binary comparison of the sums. The generated binary pattern is learned using a variant of Decision Trees (DTs). Most common, Haar-like features are applied to image intensities, however, they can also be used to describe color channels or gradients.

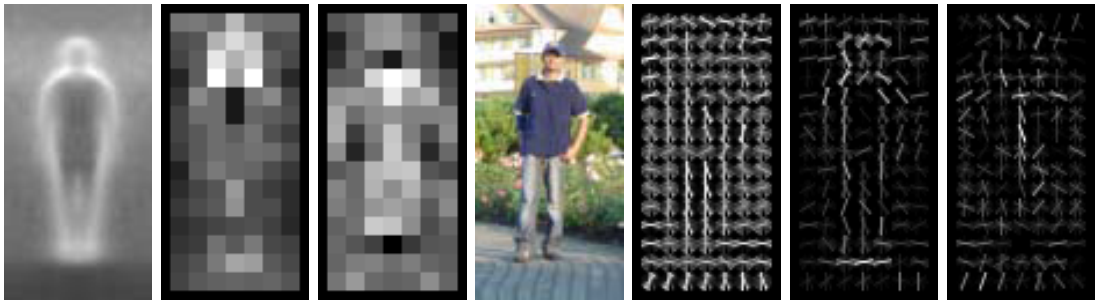


(a) horiz. edge (b) vert. edge (c) horiz. line (d) vert. line (e) diagonal line (f) center surrounding

Figure 2.6: Examples for **Haar-like Features**.

2.2.3 Histogram of Oriented Gradients (HOG)

Dalal and Triggs [34] proposed the Histogram of Oriented Gradients (HOG) descriptor for pedestrian detection, but the approach has been extended to other categories and applications and is also used within the Deformable Parts Model (DPM) [41]. Calculation of the HOG descriptor consists of the four steps (a) Gradient calculation, (b) orientation binning, (c) block description, and (d) block normalization. Figure 2.7 shows block weights and gradient orientations for a pedestrian detector using the HOG descriptor. The highly related Histogram of Gradients (HoG) descriptor (also known as Edge Orientation Histogram (EOH)) is a simpler variant without steps (c) and (d). While this prevents normalization of the gradient values, HoG descriptors can be efficiently calculated over rectangular regions using integral histograms (see Section 2.2.1).



(a) (b) (c) (d) (e) (f) (g)

Figure 2.7: **HOG detectors** cue mainly on silhouette contours (especially the head, shoulders and feet). The most active blocks are centered on the image background just outside the contour. (a) The average gradient image over the training examples. (b) Each "pixel" (i.e., cell) shows the maximum positive SVM weight in the block centered on the pixel. (c) Likewise for the negative SVM weights. (d) A test image. (e) A visualization of the computed descriptors for the image cells (f,g). A visualization of the positive negative SVM weights of the descriptors (Images from [34]).

2.2.4 More Features

Of course, *Haar-like* features and *HOG* are not the only features that are used for object tracking. Another popular feature description would be the covariance descriptor [151] and its Euclidean-space approximations Sigma Sets [72] and Sigma Points [79]. They can be seen as a combination of various feature channels, RGB-color and first and second derivatives in X and Y direction are the common choice. Also standard image features, such as Local Binary Patterns (LBPs) [109] have been used. Beside that, also tailored features, such as *discriminant colors* [40], key-point descriptors, such as SIFT [99], or optical flow (see [13] for a recent evaluation of optical flow approaches) have been used for tracking and the set of suitable features will increase further with nearly every new tracking approach.

2.2.5 Feature Selection and Combination

As already mentioned in the beginning of this chapter, many approaches do not rely on single image features because different tracking applications arise the need for different descriptions (i.e., untextured objects vs. textured objects, color vs. gray-scale video). This is also the biggest difference to vector-shaped features, such as HOG, because vector-

shaped features consist of several values that are not independent, i.e., they do not cover diverse information channels. Thus, selecting the right features for a specific task is an important issue [147]. In tracking, often learning algorithms are applied to select the best-performing features according to the current tracking task, which may change during runtime.

Collins et al. [28] perform online feature selection based on the ability of the features to discriminate foreground and background. However, they only use simple modifications of the color channels, such as raw R, G, and B values, intensity ($R+G+B$), approximate chrominance features (e.g., $R-B$), and so-called excess color features (e.g., $2G-R-B$) and do not integrate other cues such as gradients.

Grabner et al. [61] use *Boosting for Feature Selection* to find the best features for the current situation. They use several small feature pools (denoted as selectors) and perform feature selection in each of these pools. Since all features are updated during runtime, the feature ranking may change and thus also the one that is picked for evaluation. They use Haar-like features, Histogram of Gradients (HoG) and Local Binary Patterns (LBPs) for tracking, which can be computed efficiently and enable real-time performance. Finally, the selectors are combined using online boosting. In two of our applications we use random sampling from the feature space during construction of the used classifiers and perform the selection of suitable members of the feature ensemble during runtime. These approaches are explained in detail in Chapter 5, Section 5.1 and 5.3, respectively. Beside the presented combination methods, many other concepts have been proposed.

Chapter 3

Online Machine Learning using Randomized Ensemble Methods

Contents

Machine learning has a strong association to terms like *Artificial Intelligence* and *autonomous Robots* in the general public caused by their use in science fiction movies. However, for most scientists it is more or less a statistical tool to process a large amount of data and to extract useful information out of it. Arthur Samuel defined Machine learning as the following (1959)

> Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed.

This means that a computer may extract rules from a number of training samples that try to estimate an unknown *label* from a data sample. This is very useful if training samples for a specific problem are present, but an explicit algorithm cannot be formulated (i.e., implemented) because of a lack in knowledge or the high complexity of the task. In computer vision, this applies to a lot of tasks because every human (that has a working visual sense) has the ability to understand the surrounding environment but we do not have deep understanding how this is performed within our visual cortex.

For instance, e.g., finding of a specific object in an image seems to be very easy because we have wide experience in detection and recognition of humans, but we cannot describe how we solve that problem explicitly. Thus, machine learning tries to rebuild the mechanism of learning from experience as every human does when he grows up. This shifts the complexity of solving a specific problem towards the ability to provide enough training samples so that a machine learning algorithm can extract the solution approach on it's own.

In the following, we introduce the mathematical definitions of *machine learning* that are used to describe the different concepts and algorithms that are described within this thesis. In Sections 3.2 and 3.3, we give an overview over common machine learning terms and related work in randomized learning. In Sections 3.4 and 3.5 we introduce randomized online learning algorithms that are used in the applications at the end of this thesis.

3.1 Mathematical Definitions

In the following, we give mathematical definitions, where we follow the notation of [32]. Formally, let $\boldsymbol{v} = (x_1, x_2, \cdots, x_d) \in \mathbb{R}^d$ be a feature vector (or observation) and $c \in \mathcal{C}$ be a class label. Then, a training sample is formed as the tuple $\langle \boldsymbol{v}, c \rangle$. The label space $\mathcal{C}$ can be defined arbitrarily, according to the task that should be solved. Common cases are $\mathcal{C} = \{+1, -1\}$ or $\mathcal{C} = \{1, 2, \cdots, k\}$ for binary or multi-class / k-class classification, $\mathcal{C} = \mathbb{R}$ for regression or $\mathcal{C} = \mathbb{R}^n$ for structured output prediction. If the algorithm returns a real-valued *confidence* along with it's prediction, this is called *confidence-rated prediction*. This additional output is often used for comparison or combination of the predictions of several learning algorithms or instances.

In general, the goal is to create a mapping H that is able to predict the correct class label c for an unlabeled test sample $\boldsymbol{v}$,

$$H : \mathbb{R}^d \to \mathcal{C}.$$

3.2 Terminology

Beside the basic mathematical formulation of machine learning, we distinguish many different learning tasks. To ease understanding of this chapter, we introduced common terms within this section. This is especially important since some of these have ambiguous meaning in the community.

Supervised Learning In supervised learning, the experience is presented in form of sample and label pairs $\langle \boldsymbol{v}, c \rangle$. This means that for every training sample the correct output value is given (i.e., labeled data). The task of "learning from experience" as described above is an example for supervised learning.

Unsupervised Learning In contrast, for unsupervised learning no labels c are available for the training samples $\boldsymbol{v}$ (i.e., unlabeled data). Unsupervised learning algorithms often try to uncover a structure within the training samples, e.g., performing grouping of the samples to retrieve homogeneous groups in the sample space (i.e., clustering).

Semi-supervised Learning This is a midway of the above extremal cases. In semi-supervised learning, the training data consists of labeled and unlabeled samples. In this case, the unlabeled data is often utilized to recover the whole structure of the data space whereas the labeled data assigns labels to the retrieved clusters. The aim of semi-supervised learning is to improve the quality of a learning algorithm based on a very large amount of unlabeled data, because unlabeled data is usually much cheaper to acquire than labeled data.

Multiple-Instance Learning Multiple-Instance learning tries to incorporate the uncertainty of labels. For instance, cropping pedestrians in images includes a large amount of ambiguities, which can be, e.g., slight changes in scale or shifts which add uncertainty to a specific sample. This problem is respected in multiple-instance learning, where the label is assigned to a group (i.e., bag) of samples rather to an individual one. This makes the generation of training data much easier, because the precision of the labeling process can be lower but increases the complexity of the learning process.

Multi-Label Learning This is a learning concept, where one sample can represent several different classes at once. For example, this is the case if a still image is labeled

to include several different categories of objects, without stating the individual positions of the objects. The learning algorithm has to recover the parts of the sample that are representative for each class by analyzing the whole dataset.

Beside the kind of samples that are used for learning, there is also a distinction regarding the arrival of the training data.

Offline Learning In offline learning, all samples are available for the learning algorithm at once. This is often also called *batch learning*. Usually, there are no limitations in memory consumption, complexity and training time or in which order the training data is processed. This means that a sample can be accessed several times.

Incremental Learning Incremental learning is similar to offline learning, but the samples are only available in sequential order. Thus, the learning algorithm cannot access future samples but should converge to an algorithm that is trained offline. This training mode is important for large-scale datasets that do not fit into memory or if samples are generated sequentially and have to be processed as soon as possible. An incremental learning algorithm may be able to perform predictions before the whole training samples have been processed.

Online Learning In online learning only a limited portion of the dataset is available at once. This portion may be as small as a single sample or may consist of a larger chunk of data (i.e., batch online learning). A sample has to be processed immediately and has to be discarded after training. Thus, the learning algorithm will automatically adapt to current samples (i.e., the algorithm is *updated*). As with incremental learning, the algorithm has to be able to generate predictions at any time. We distinguish between **incremental** and **online** learning as that online learning has a fixed size knowledge, i.e., memory footprint, whereas incremental learning can accumulate the information and increase it's knowledge over time. According to the actual task, we imply that online learning relates the importance of an information to the arrival time, which results in out-dating of old information. This implies a strong relation to the information that has been learned recently.

Basically, there are two types of applications that require incremental or online learning techniques. First, large-scale data analysis has increased the interest in incremental learning. Today's databases may consist of millions of samples that are hard to store at once without increasing the runtime significantly. This is also an issue when learning from

Internet image collections, that cannot be simply "downloaded" to a computer. The second example are applications where the environment may change. For example, a mobile traffic surveillance system may be deployed at several different locations and should adapt to the current needs. The same also applies for object tracking, because both the object and the background may changeover time.

Highly related to that are (inter-)active and reinforcement learning problems.

Interactive Learning Interactive learning uses the knowledge of a teacher (usually a human) that labels unlabeled samples and selects unlabeled samples to question the teacher under the aspect of maximal gain in knowledge.

Reinforcement Learning In reinforcement learning, the teacher tries to maximize the knowledge gain of the learner by choosing the best training samples at the right time, which is highly related to the concept of tracking-by-detection as described in this thesis.

Regarding the algorithms that are discussed within this thesis we have to define some terms that are especially related to randomized learning and ensemble learning.

Meta-Learning Meta-learning algorithms are concepts to combine a number of instances of other learning algorithms. Thus, they work on a higher level of abstraction and do not perform learning on their own (see [125] for an overview). Popular examples for meta-learning algorithms are Ensemble methods, Transfer Learning [118], or Co-training [19].

Ensemble Learning Ensemble learning algorithms perform meta-learning. They combine a larger number of similar learners (see [112] for an overview) to increase the overall performance. Within this thesis, we refer to the individual learners as **members** of the ensemble. The major goal of ensemble methods is that the overall ensemble should perform better than just using several individual members of the ensemble in parallel. This is mainly given by the decreased variance due to the averaging of the ensemble.

Bagging Bagging, also known as bootstrap aggregation [21], is a very popular ensemble learning method that is used in, e.g., Random Forests. To improve the overall performance of the ensemble in Bagging, the de-correlation of the individual members is enforced. This is done by generating individual training sets for all members by sampling with replacement from the overall training set.

Randomized Learning Another way to increase the de-correlation of the members of an ensemble is to include randomization at several parts of the learning algorithm. First, random sub-sampling of the training set avoids that all members of the ensemble receive the same dataset. Additionally, the individual members should include random feature selection, random threshold selection or similar to prevent training members that are too similar. This results in increased diversity of the ensemble members and decreased variance of the resulting classifier. In this thesis, we build on Random Forests (RFs) and Random Ferns (RFes) and their online equivalents, which are the most common randomized ensemble learning algorithms (see Sections 3.4 and 3.5.2).

Boosting Boosting [47] is another popular ensemble learning method in computer vision. While Bagging uses randomization to diversify the ensemble members, Boosting arranges the members within a chain and re-weights samples while they pass through this chain. The samples are weighted according to the correctness of the prediction of the subsequent stages in the chain. This enables that succeeding stages in the chain to focus on samples that haven't been solved well in the preceding stages.

Weak and Strong Learner These terms describe the different levels of meta-learning algorithms. Usually, the members of an ensemble are denoted as weak learners, because they do not provide maximum performance, while the ensemble is depicted as strong learner.

Bootstrapping While Bootstrapping has a well-defined meaning, the term is often used with different meanings in the literature. For us, Bootstrapping describes the process of refining a learning algorithm based on failures it made within the training set. Usually, these failure cases are presented to the learning algorithm again or additional members of an ensemble focus on them.

3.3 Online Ensemble Learning in Computer Vision

As already mentioned at the beginning of this chapter, many problems in computer vision are hard to solve explicitly because humans are not able to transfer their "approach" to an actual implementation. Therefore, machine learning has a long tradition in this field of research. Since we mainly focus on problems that require online learning capabilities, we summarize the development of such algorithms with special focus on computer vision

applications.

Many online learning algorithms can trace their roots back to the work of Littlestone, Warmuth, and Vovk [97, 98, 155]. They defined online learning as a sequence of trials. Every time the learner gets a new sample, it tries to make a prediction. Subsequently, feedback can be used to refine the model for prediction.

Later, the most popular ensemble methods, Bagging [21] and Boosting [47], have found their way into the computer vision community. Papageorgiou et al. [119] presented their concept for a general object detection framework. They proposed to learn a subset from an over-complete dictionary of wavelet basis functions to retrieve a compact representation for specific object category. This compact representation is then used to detect instances of the object category in unconstrained environments using a Support Vector Machine (SVM) classifier. Subsequently, Viola and Jones [153, 154] proposed one of the most prominent object detection approaches in computer vision based on [119]. They perform Boosting to select a set Haar-like features and construct a cascaded classifier. Their approach was the first to enable real-time object detection. Motivated by their work, offline boosting has been used for many different computer vision applications, such as rotation-invariant object detection [152], simultaneous detection and segmentation [161], multi-view detection [159], feature combination [164], or object recognition [111].

Combining the two branches of research, online learning and ensemble methods, Oza and Russell [114] proposed an online formulation for Boosting and Bagging. Their work was the basis and motivation for a large number of online ensemble algorithms (e.g., [12, 55, 56, 60, 62, 91, 131, 133, 137]), where tracking-by-detection was a standard use-case for evaluation.

While online model adaption has already been used for tracking before (e.g., [29, 75]), Avidan [5, 6] was the first who used ensemble learning methods for tracking. However, he did not apply online learning on the used members but replaced old members of the ensemble with newly trained ones.

Online Boosting Based on the work of [153], Grabner and Bischof [60] combined Boosting for feature selection with the concept of online Ada-Boost [114]. To increase the number of used features, they use feature pools called selectors. This concept has been extended to various learning paradigms, such as Multiple Instance Learning (MIL) [12], Semi-supervised Learning (SSL) [62], or Multi-view Learning (MVL) [131] and was applied to various tasks, such as key-point tracking [64], background modeling [63], or scene-adaptive object detection [129]. Leistner et al. [91] proposed an online extension of Boosting (i.e.,

Gradient Boost [49]), which allows to use more robust loss-functions. Also Babenko et al. [11] developed a very similar idea at the same time, but limited their evaluations to binary classification problems.

Online Bagging Beside Boosting, also Bagging gained a lot of interest in the computer vision community, mostly caused by the popularity of Random Forests (RFs) [22]. Lepetit et al. [92] used an ensemble of randomized trees to recognize key-points for 3D tracking and Özuysal et al. extended this idea for the use with Random Fern (RFe) [116]. Later, Saffari et al. proposed an online extension of Random Forest (RF), which has been used for tracking [133] and extended to Multiple Instance Learning [90] and Multi-view Learning [88]. Additionally, Online Random Forest (ORF) have been used to transform the Hough-based detection approach of Gall and Lempitsky [50] to the online domain by Schulter et al. [137]. Besides that, online Bagging has been used in combination with Naïve Bayes (NB) classification (see Section 5.1) and Hough-based Ferns for tracking (see Section 5.3). This list makes no claim to be comprehensive because Random Forests (RFs) [22] has been applied to a many different computer vision problem so far and is part of many current state-of-the-art approaches (see also [32]).

3.4 Online Random Forests (ORFs)

Randomized Trees, also known as Random Forest, or Random Forests (RFs) [22] are a very popular ensemble learning method. They basically combine a randomized set of Decision Trees (DTs) (see Section 3.4.2) using Bagging [21]. Additionally, each DT is trained using a sub-sampled set of training samples to enforce de-correlation. Because of their speed, capability of parallelization and robustness to noisy training data they are often applied for various tasks such as key-point recognition [92], semantic segmentation [143] or medical image analysis [105].

RFs utilize a set of T DTs and belong to the category of Ensemble Learning algorithms. The final prediction is computed by averaging over the whole set of classifiers

$$H(\boldsymbol{v}) = \frac{1}{T} \sum_{t=1}^{T} p_t(c|\boldsymbol{v}), \tag{3.1}$$

where $p_t(c|\boldsymbol{v})$ is the probability for class c given feature vector $\boldsymbol{v}$.

However, Ensemble Learning algorithms have to ensure that their individual learners are *decorrelated*. Otherwise, their prediction will be too similar and the grouping of the

classifiers does not give benefits because all members will make the same mistakes. Then, the classifier will not generalize, i.e., work well on samples not included in the training data. This axiom is supported by the generalization error GE of RFs that is upper bounded by

$$GE \leq \bar{\rho}\frac{1-s^2}{s^2}, \tag{3.2}$$

where s is the strength of the ensemble (i.e., the expected value of the margin over the entire distribution) and $\bar{\rho}$ is the mean correlation between pairs of trees in the forest [22]. The correlation is measured in terms of the similarities of the predictions.

3.4.1 Bagging

Therefore, randomized learning methods (e.g., Bagging) rely on the two principles (a) random input selection and (b) random feature selection. The main advantages of these methods are the increased stability and decreased variance of the resulting classifier. Using random input selection, the classifiers are trained on different subsets of the training space. Random feature selection further increases the problem complexity by limiting a specific classifier to a subspace of the whole feature space. Bagging has shown to improve the final classifier in terms of stability and classification accuracy and helps to avoid overfitting.

In RFs, the individual training subsets for each DT are generated using *sampling with replacement*, which results in $1 - \frac{1}{e}$ non-identical samples per set on average:

> Given N possible outcomes of a trial (the N cases in the learning set) and N trials, the probability that the n^{th} outcome is selected $0, 1, 2, \ldots$ times is approximately Poisson distributed with $\lambda = 1$ for large N. The probability that the n^{th} outcome will occur at least once is $1 - \frac{1}{e} \approx 0.632$.
>
> (Breiman [21])

Sub-sampling of the feature space used for a single DT is performed inherently by using randomly generated tests.

Training The construction rules for RFs also compensate for the complex construction of DTs. The applied heuristic generates randomized test parameters over a reduced set of candidate features. Thus, the feature selection (i.e., selection of parameters $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$) is done randomly and only τ is optimized over the whole spectrum. This gives an overall number of tests of $N - 1$ per parameter pair $\langle \boldsymbol{\phi}, \boldsymbol{\psi} \rangle$, where N is the number of training

samples. Naturally, the optimization is a demanding task if the number of randomly chosen parameter pairs $\langle \boldsymbol{\phi}, \boldsymbol{\psi} \rangle$ is large.

Extreme Randomization Geurts et al. [53] proposed another construction scheme called Extremely Randomized Decision Tree (ERT). They randomly sample all three parameters $\langle \boldsymbol{\phi}, \boldsymbol{\psi}, \tau \rangle$ during optimization, where the threshold τ will be selected K times and the best performing value is selected. Thus, the number of elements that have to be considered for optimization does not depend on the size of the training set, but on K. If K is set to be $N - 1$, the training algorithm is very similar to RFs, setting $K = 1$ yields complete randomization and the structure of the resulting trees does not depend on the training data any more. Additionally, due to the full randomization of the tests, sub-sampling of training data is not necessary any more.

Evaluation During evaluation, a sample $\boldsymbol{v}$ traverses down the tree t according to the subsequent node tests until it reaches a leaf node. For all leave nodes the probability $p_t(c|\boldsymbol{v})$ that a sample ending up in this node has the label c is modeled. Finally, the overall probability $p_t(c|\boldsymbol{v})$ is determined by averaging the individual leaf probabilities p_t over an ensemble of T trees. For classification, the class c of a specific data sample $\boldsymbol{v}$ can be predicted by late or early fusion. Early fusion performs prediction for each individual element of the ensemble, while late fusion considers the individual probabilities, such that

$$H(\boldsymbol{v}) = \arg\max_c \sum_{t=1}^{T} p_t(c|\boldsymbol{v}) \,. \tag{3.3}$$

Because of the tree-like classifier structure, training can be done recursively and evaluation is very fast (i.e., logarithmic complexity of the classifier according to the number of nodes). A very comprehensive tutorial about RFs and their applications to classification, regression and density estimation can be found in [32].

3.4.2 Decision Trees (DTs)

DTs are the basic building block of RFs. Beside the high performance of DTs, they are very simple to implement. The basic concept is to use very simple tests to split the set of training samples into smaller sets until the problem defined by the smaller set can be solved. This is done by performing binary splits in a recursive manner which results in a tree-like structure.

During construction of these trees, for all nodes j, except the leave nodes, binary splitting tests

$$h\left(\boldsymbol{v}, \boldsymbol{\theta}_j\right) \in \{0, 1\} \tag{3.4}$$

are defined, which decide if a sample $\boldsymbol{v}$ arriving at the node j is sent to its left (0) or right (1) child node. Given the parameter triple $\boldsymbol{\theta} = (\boldsymbol{\phi}, \boldsymbol{\psi}, \tau)$, binary tests are defined as

$$h\left(\boldsymbol{v}, \boldsymbol{\theta}_j\right) = \left[\boldsymbol{\phi}(\boldsymbol{v}) \times \boldsymbol{\psi} > \tau\right], \tag{3.5}$$

where $\boldsymbol{\phi}$ is a feature transformation, $\boldsymbol{\psi}$ calculates a linear combination of the features and τ denotes a threshold.

Using this framework, we can calculate the parameters to perform linear and non-linear data separation [32]. For example, using a vector $\boldsymbol{\phi}(\boldsymbol{v})$ with a single entry larger than zero results in feature thresholding with τ. If several entries are larger than zero, a linear combination of several feature values is thresholded. Such a test, using two feature entries, is very common and used in, e.g., [50, 56]. If $\boldsymbol{\phi}(\boldsymbol{v})$ is defined properly, non-linear feature transformation is performed.

During training, DTs optimize these binary tests according to a certain criterion based on the given training set S (i.e., the subset of S that is processed by the corresponding node). Some common optimization criteria are described below. After choosing the optimal test parameters, the training set is split recursively until the subsets are internally consistent (i.e., belonging to the same class c) or a maximum tree depth D has been reached.

When the tree is fully grown, the probability $p\left(c\right)$ can be calculated for all classes in all leaf nodes. The prediction of the DT is made according to the leaf node they end up, and the corresponding probability, and is denoted as $p\left(c|\boldsymbol{v}\right)$, where $\boldsymbol{v}$ indirectly defines the leaf node the sample ends up. If a DT is fully grown, it is prone to over-fitting because of a possibly small number of samples in the leaf nodes. Therefore, a large number of modifications have been proposed, such as restricting the maximum tree depth or different pruning methods. However, an ensemble of DTs does not show this behavior [22].

Optimization Criteria To optimize the splitting criterion in DT nodes, the *Information Gain IG* of a set of testing parameters $\boldsymbol{\theta}_j$ is measured. In machine learning, the information gain is equivalent to the Kullback-Leibler divergence [81].

The Information Gain measures the weighted improvement of the purity from an in-

dividual node to it's child nodes, such that

$$IG(n) = M(n) - \sum_k \frac{\eta^{n_k}}{\eta^n} M(n_k), \tag{3.6}$$

where n is the root node, $\eta^{(\cdot)}$ is the number of samples in leaf node $(\cdot)$ and k iterates over all child nodes n_k.

The measurement of purity of a distribution is of high relevance in statistics and thus, quite a range of methods has been proposed. However, we will focus on the three most common criteria (a) *Gini coefficient*, (b) *Entropy*, and the (c) *Classification Error*. While there has been a lot of discussion if *Gini coefficient* or *Entropy* performs better, the difference between those two has little influence within RF (see Figure 3.1).

The *Gini coefficient* [54] measures the inequality among values of a frequency distribution. In the original formulation, a minimal value expresses perfect equality of all values while a maximal value is reached for completely unbalanced values. The Gini coefficient is defined as

$$M_{gini} = 1 - \sum_{\mathcal{C}} p\left(c\right)^2, \tag{3.7}$$

where $p\left(c\right)$ measures the probability of class c of the set of training samples.

Another popular purity measure is the *Entropy* [140] that quantifies the expected value of the information contained in a message, which is related to the unpredictability of a data source. Considering our problem, the prediction for a specific node is definitely if all samples according to this node share the same label, i.e., the node is pure. Therefore, the information content of such a node is zero. The Entropy can be formulated as

$$M_{entropy} = -\sum_{\mathcal{C}} p\left(c\right) log\left(p\left(c\right)\right), \tag{3.8}$$

which also shows that the computational effort is a bit higher than for the Gini coefficient due to the computational complexity of the $log(\cdot)$ operation.

For completeness, we also mention the *classification error* measure, which directly considers the maximum class probability

$$M_{error} = 1 - max\left(p\left(c\right)\right). \tag{3.9}$$

Hence, for a pure node $M_{error} = 0$ and for an equal probability for each classes $M_{error} = \frac{1}{|\mathcal{C}|}$. Figure 3.1 compares the different optimization criteria visually. It shows that equal

distribution of all classes results in the highest measure and purity of the dataset results in the lowest measures. This means that pure nodes give the highest information gain. Gini coefficient and Entropy appear rather similar, beside of a scaling factor which does not influence the overall result significantly.

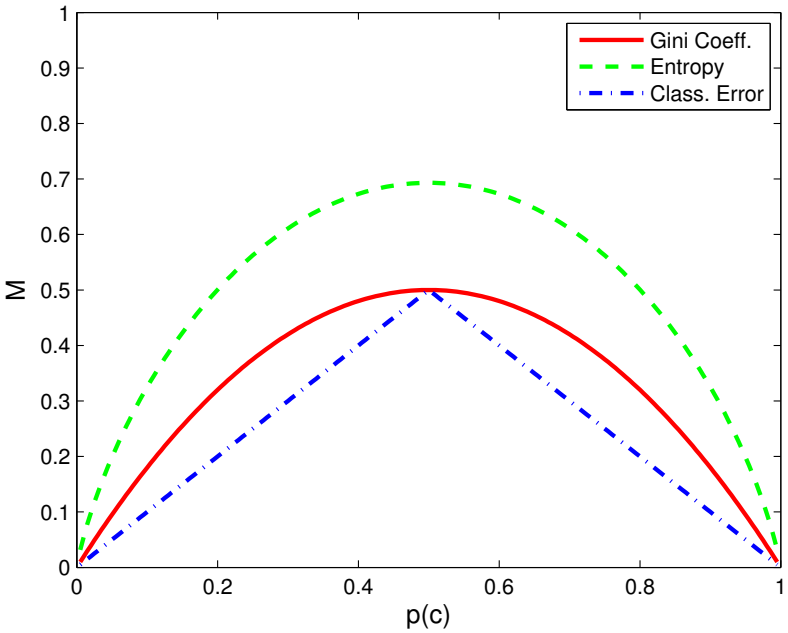


Figure 3.1: Comparison of different **optimization criteria** for the two-class case. The class probability is plotted on the x-axis, which means that both ends 0 and 1 are cases where the analyzed dataset is pure and 0.5 is the value of equal class probabilities.

The goal is to find the best test $\boldsymbol{\theta}$ for a given node with respect to the given training data set. The construction of an optimal DT is known to be NP-complete under several aspects of optimality and even for simple concepts [73, 107]. Therefore, the optimization

$$\boldsymbol{\theta} = \underset{(\phi,\psi,\tau)}{\arg\max}\, IG(n) \tag{3.10}$$

is usually based on heuristics or brute-force search on a reduced space of possible parameter settings. Some of these concepts are discussed in the next section.

3.4.3 Online Learning

RFs have been applied to various tasks in computer vision within the last years and demonstrated excellent performance. Nevertheless, RFs have several limitations in the

context of object tracking. First, a large training set is required to establish the classifier structure. Especially when tracking unknown objects, the amount of available training data is limited. Hence, to cope with streaming data, Online Random Forests (ORFs) [133] have been proposed recently.

Online learning of RFs consists of (a) online Bagging and (b) online tree learning. Online Bagging has been proposed by Oza and Russel [114]. They propose to model the sequential arrival of the data by a Poisson distribution. Each tree t is updated using each sample k times in a row, where k is a random number generated by

$$k \sim Poisson\,(\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{3.11}$$

where λ is usually set to a constant number, in our case $\lambda = 1$. This simulates the selection with replacement scheme from offline Bagging. In fact, in [114] it was proven that online Boosting converges to the offline version. However, this proof is only valid for naïve Bayes weak classifiers and an infinite amount of training samples.

While the work of Oza and Russel [114] models the arrival of training samples at the individual members on the ensemble, the selection of the optimal splitting test, as performed in offline RFs, is an open issue. Therefore, Extremely Randomized Decision Trees (ERTs) are grown by generating the test functions and thresholds randomly. During growing of a randomized tree, each decision node randomly creates a set of tests and picks the best according to a quality measurement, but the label probabilities $p(c)$ are established online. More specifically, when a node is created it generates a set of n random tests $S = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n\}$. During online training, the node maintains the class distributions that are generated by each test in S. Also the achieved information gain IG can be calculated online based on the class distributions for each test.

A specific node is only split into two child nodes if (a) the number of samples that are modeled by the individual distributions is significant (i.e., above a certain amount) and (b) the information gain quality of the candidate splits is high enough. When splitting a node, the already collected class distribution is passed to the corresponding child nodes, which enables immediate evaluation of these nodes without observing an additional sample. Beside Online Random Forests (ORFs), also other approaches use a tree-growing scheme to create classifiers (e.g., [36, 117]).

3.4.4 Properties and Discussion

Random Forests (RFs) have been applied to various tasks in machine learning and demonstrated to be an efficient tool. Due to the recursive training strategy, they can be implemented very easily. The tree-like structure allows for extremely fast evaluation. Another benefit of the structure is that the training data is split into small coherent groups. This allows for inherently modeling multi-class and multi-modal data because the data is implicitly clustered within the tree.

Many extensions and modifications of RFs have been developed to further improve the performance for specific tasks. Tu proposed Probabilistic Boosting Trees (PBTs) [149], where he uses Boosting to train more complex classifiers at each tree node. The used weak classifiers improve the overall performance of the process of decision-making. However, PBTs are prone to over-fitting if the amount of training data is small. Geremia et al. [52] introduced a spatial component in their RFs, considering the expert knowledge that the segmented volumes are often symmetric for their medical task. Later, Montillio et al. [105] introduced Entangled Decision Forests (EDFs), where test results can depend on results that are on a higher level of the tree structure. This allows to capture long-range semantic context and shows superior results for the task of anatomy segmentation in Computer Tomography (CT) images.

Nevertheless, RFs share some properties that may be improper for specific types of applications. First, they require a large amount of training data to work well. This is especially a problem if only a small number of samples is available or if a classifier is required after a small amount of samples (i.e., as this is the case for tracking-by-detection). Second, the classifier is not able to cope with changing target distributions, as all offline learning algorithms.

Especially for tracking of unknown objects, it is important that the used learning algorithm is very flexible because the targets often completely change their appearance. Another issue is the lack of training data because there is only one sample / annotation available from the first frame. Therefore, online learning algorithms have been a popular means for continuous adaption of the used model.

ORFs [133] achieved very good results for the task of tracking [133] and have been extended to other tasks [141] and learning paradigms (e.g., [88, 90]). However, the used splitting tests in ORFs are still optimized for a specific subset of the training data (i.e., the part of data that arrived first) and cannot be changed afterwards, which would be beneficial to adapt to changing appearance. One possible way to address this issue would

be to drop individual trees and start to train new ones from time to time.

Referring to our definition of online learning algorithms (fixed memory footprint), ORF could also be categorized as *incremental learning algorithms*, because it's memory footprint increases during training. However, the size of ORF is bounded by the maximum depth of the individual trees and the overall number of trees. Regarding the computational efficiency, both Offline and Online Random Forests (ORFs) use tests that sequentially / hierarchically depend on each other. Such conditional jumps are not ideal for sequential execution, especially specialized hardware such as General Purpose Graphics Processing Units (GPGPUs) cannot be fully exploited. However, the concept of RF can be transferred to other weak learners, such as MultiNomial Logit (MNL) [65] and NB [124]. As with growing trees for ORF [133], using different members in the ensemble can be used to establish special properties, such as linear execution without conditional jumps (i.e., RFe) and online learning (i.e., ORNB and Online Random Fern (ORFe)).

3.5 Online Random Naïve and Semi-Naïve Bayes

While Random Forests (RFs) are very popular for ensemble learning, also other weak learners can be used within an ensemble. In the following, we focus on naïve and semi-naïve Bayes formulations, that are quite similar to RFs.

Bayes' theorem is named after Thomas Bayes (1701–1761), an English mathematician and Presbyterian minister, and was first proposed as a solution to the problem of "inverse probability". At that time, this idea gained limited exposure until it was independently rediscovered and further developed by Laplace [83]. According to the mathematical definitions from Section 3.1, the theorem can be formulated as

$$p(c|\boldsymbol{v}) = \frac{p(c)p(\boldsymbol{v}|c)}{p(\boldsymbol{v})}, \tag{3.12}$$

where $p(c)$ is the class label prior, and $p(c|\boldsymbol{v})$ is the unknown probability distribution of the joint space of features $\boldsymbol{v}$ and labels $c \in \mathcal{C}$. Since these distributions are unknown, we need to estimate them using the given training data.

We want to model the class conditional probabilities of N binary tests and during

evaluation we want to select the best matching class c for a given feature vector $\boldsymbol{v}$ as

$$\begin{aligned} c &= \arg\max_c p\left(c|\boldsymbol{v}\right) \\ &= \arg\max_c p\left(c|x_1, x_2, \ldots, x_N\right) \\ &= \arg\max_c p\left(x_1, x_2, \ldots, x_N|c\right), \end{aligned} \tag{3.13}$$

assuming a uniform prior over all classes. However, this formulation is problematic, because the joint distribution over all features cannot be modeled in practice. Therefore, often a naïve or semi-naïve Bayesian formulation is used to approximate the full Bayesian formulation.

Naïve Bayes (NB) Naïve Bayes (NB) assumes that all used features are independent. Thus, the probability of the individual feature values x_n can be separately modeled such that the joint probability can be formulated as

$$P\left(x_1, x_2, ..., x_n|c\right) = \prod_{n=1}^{N} p\left(x_n|c\right). \tag{3.14}$$

Although this model is very simple, it has been shown to produce competitive results [124].

Semi-naïve Bayes A more complex formulation to integrate dependencies between features can be realized via the semi-NB formulation. In this case, features are grouped into larger sets (not only individual ones) and the joint distribution within these sets is modeled. The Bayesian formulation is approximated by

$$P\left(x_1, x_2, ..., x_n|c\right) = \prod_{m=1}^{M} p\left(\tilde{\boldsymbol{v}}_m|c\right), \tag{3.15}$$

where $\tilde{\boldsymbol{v}}_m$ denotes a specific set of tests. Adjusting the set size $|\tilde{\boldsymbol{v}}_m|$ and the number of used groups M enables flexibility in terms of complexity versus performance. This formulation relates to Random Ferns (RFes) [115], which we describe more detailed in Section 3.5.2.

Sample correction The probability estimate will be zero if a given class and feature value never occur together in the training set. In this case, at least one probability will be zero which eliminates the other in the multiplication. Therefore, a small sample correction is used to preclude that any probability can be exactly zero. To generate an

initial distribution, a *Dirichlet prior* [44] can be used. The Dirichlet distribution is defined as

$$f(x_1, \ldots, x_d; \alpha_1, \ldots, \alpha_d) = \frac{1}{B(\alpha)} \prod_{i=1}^{d} x_i^{\alpha_i - 1}, \tag{3.16}$$

where $B(\alpha)$ is a normalization factor and α_i controls the distribution. In the special case of $\alpha_i = 1 \ \forall i$ it boils down to an uniform distribution, resulting in a probability of $\frac{1}{|\mathcal{C}|}$ for all classes c and feature values that have not been included in the training set. The distribution is the multivariate generalization of the beta distribution [80] and is often used as prior distribution in Bayesian statistics. If, e.g., histograms are used to model the probability distribution, this means that all bins are filled with a constant initial value.

Additionally, Bayesian classifiers can be used within a classifier ensemble, such as Bagging. In this case, *model averaging* increases the robustness of the classifiers. Both methods will be used in the practical implementations in Sections 3.5.1 and 3.5.2. Since Oza and Russell [113] proposed online Bagging, we only have to adapt the naïve Bayes classifier to enable online learning.

3.5.1 Online Random Naïve Bayes (ORNB)

Despite this naïve independence assumption, NB has been used in the past and shown to deliver good results [37, 70, 124]. Especially because features are usually pieces of local information in computer vision, thus considering them independently is a feasible assumption. Assuming independence and uniform label distribution, a classifier $H(\boldsymbol{v})$ can be written as

$$H(\boldsymbol{v}) = \arg\max_{c} \prod_{n=1}^{N} p(x_n|c), \tag{3.17}$$

where x_n is an individual feature value and the classification can be estimated by building the product of all class probabilities for the current feature values.

Randomized Learning Like DTs, single NB classifiers only reach limited performance according to their simplicity. Therefore, Prinzie and Van den Poel generalized the idea of Random Forests (RFs) to MultiNomial Logit (MNL) and Naïve Bayes (NB) [124]. As with RFs, they apply Bagging and random feature selection to increase the stability and decrease the variance of the resulting classifier:

$$H(\boldsymbol{v}) = \arg\max_{c} \sum_{t=1}^{T} \prod_{f=1}^{F} p^t(x_f|c), \tag{3.18}$$

where T randomly trained NB classifiers are combined, each using $F \leq |N|$ features.

On-line Learning When creating the Random Naïve Bayes (RNB) classifiers ensemble, for each classifier we select F features randomly. The probability distribution $p(x_f|c)$ of each individual feature is then modeled for each class $c \in \mathcal{C}$. Using randomized threshold selection, Geurts [53] states that Bagging (i.e., random input selection) can be skipped without decreasing the performance of the classifier. Since we want to enable on-line learning for our RNB ensemble, we need to estimate the probability distribution for the given feature x_f on-line. Therefore, we use equally binned histograms to estimate the probability distributions since they are very fast and easy to implement. Moreover, histograms are applicable to incremental learning and can handle multi-modal distributions easily. This results in a more fine-grained description of the probability distribution, but can also be interpreted as a larger number of thresholds that are used to split the data which tends towards a semi-naïve formulation.

Temporal weighting of training data Some learning problems require temporal weighting, i.e., for unlearning of information. This is required, for instance, if we have to cope with temporary noise or outliers and *concept drift* which refers to a non stationary learning problem over time (see [167]). Since online RFs create their tree-structure based on training data, they have difficulties to "unlearn" data after some time.

For the fixed structure of RNB, we use an *iir-like* (i.e., infinite impulse response) filtering for each histogram bin, where the value of each bin can be calculated as

$$w_{t_0}^{norm} = \sum_{t=-\infty}^{t_0} w_t \cdot r^{t_0 - t}. \tag{3.19}$$

Here, w_t is the learned sample weight at time t and t_0 represents the time of the current update. The speed of forgetting can be defined with the parameter r.

Hyperplane Features To enhance the significance of the feature pool and to further weaken the independence assumption, we create random hyperplanes within the feature space. These hyperplanes x_h are computed as a weighted linear combination of the features $\boldsymbol{v}$ as $x_H = \mathbf{w}^T \boldsymbol{v}$, where the weights $\mathbf{w} \in [-1, 1]^d$ for each feature are randomly chosen and $\sum_{i=1}^{d} |w_i| = 1$. The weight vector $\mathbf{w}$ is chosen to be very sparse to create only small local subspaces. Breiman [22] used a heuristic of $\log d$ features with non-zero weights for feature sub-sampling.

Although the naïve formulation is intuitive and works well it is hard to model complex distributions. Even using hyperplane features, it cannot be guaranteed that dependent distributions are reflected in the generated model. Thus, Random Ferns (RFes) have been developed, combining groups of features in a semi-naïve Bayes formulation.

3.5.2 Random Ferns (RFes)

RFes [115] are ensemble classifiers that can be interpreted as semi-naïve Bayes classification (Equation 3.15). Thus, again assuming an uniform label distribution, a classifier $H(\boldsymbol{v})$ can be written as

$$H(\boldsymbol{v}) = \arg\max_{c} \prod_{m=1}^{M} p\left(\tilde{\boldsymbol{v}}_m | c\right), \tag{3.20}$$

where $\tilde{\boldsymbol{v}}_m$ denotes a set of individual tests. The classification result can be calculated by building the product of all class probabilities for the current feature values. As already mentioned in Section 3.5, the estimated probability distributions have to be initialized appropriately to prevent that any $p\left(\tilde{\boldsymbol{v}}_m | c\right)$ gets zero. Therefore, as with ORNB, we initialize the used statistics with a Dirichlet prior (i.e., uniform distribution). Additionally, we perform Bagging to increase the robustness of the classifier, which results in

$$H(\boldsymbol{v}) = \arg\max_{c} \sum_{t=1}^{T} \prod_{m=1}^{M} p\left(\tilde{\boldsymbol{v}}_m | c\right). \tag{3.21}$$

This formulation allows for more robust classification and is more similar to the idea of RFs, only the underlying structure of the learning algorithm is changed from tree-like to flat structures (see Figure 3.2). Again, adjusting the set size $|\tilde{\boldsymbol{v}}_m|$ and the number of used groups M enables flexibility in terms of complexity versus performance.

Comparison to Random Forests (RFs) RFs and RFes can be implemented in a very similar way, because a Fern can be interpreted as trees using the same test at a whole depth level as depicted in Figure 3.2. In contrast to RFs, Ferns are usually used with completely randomized tests [115, 152] and the structure is not optimized at all, which is very similar to ERTs with choosing only one parameter set.

The main advantage of RFes over RFs is the independence of the individual tests. In RFs, a node test is only performed if the subsequent tests have a pre-defined pattern (i.e., the sample is traverses down the tree to exactly this node). This evaluation scheme implies a large number of `if-then-else` statements (i.e., conditional branches), which prevents

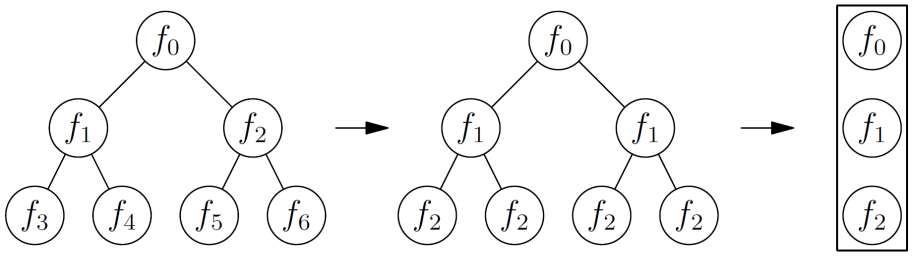


Figure 3.2: Comparison of the classifier structure of **Random Ferns** and **Random Forests** (figure taken from [115]). Random Ferns (right) can be interpreted as flat projection of Random Forests (left).

an unhampered execution on modern processing architectures (i.e., frequent flushing of the instruction pipeline).

Online Learning Online learning with RFes works the same way as with NB classifiers. Since the whole structure of RFes is fixed and is not optimized during training, it can be established randomly before runtime. During online training of a RFe, we have to model the class probabilities of the leaf nodes. This can be done incrementally by counting arriving at a specific leaf node during runtime for each class c. Since this would limit the adaptivity of the classifier due to saturation effects, we again apply temporal weighting of the samples as described in Section 3.5.1.

Unbalanced Datasets To overcome unbalanced number of training data per class, normalization of class probabilities in each leaf node is performed to adjust for unequal amount of samples for each class. A common normalization is term frequency – inverse document frequency (TF-IDF)m which is defines as

$$p(c|\boldsymbol{v}) = \frac{\eta_c^n}{\eta^n} \cdot log \frac{\eta^t}{\eta_c^t}, \tag{3.22}$$

where η_c^n is the amount of samples of class c in node n, η_c^t is the amount of samples of class c in Fern t.

In our applications (see Section 5.3), we use a slightly simpler normalization

$$p(c|\boldsymbol{v}) = \frac{\eta_c^n}{\eta^n} \cdot \frac{\eta^t}{\eta_c^t}, \tag{3.23}$$

which simply simulates an equal amount of samples within each class. Naturally, dataset balancing can be used in the statistics of all described learning algorithms.

Chapter 4

Detection and Training

Contents

In the previous chapters, we have discussed how target objects are modeled in tracking-by-detection approaches and how the model is learned and updated over time. However, there are two more building blocks that are needed to close the tracking loop (see Figure 1.1 that have significant influence on performance and runtime of an approach. These are (a) **detection**, i.e., estimation of the current object position, and (b) generation of **training samples** that are used to update the statistical model.

Object detection is a common task in computer vision, the generation of training samples is mostly ignored in most publications. Nevertheless, the quality of the training samples has crucial influence on the quality of the overall tracking process. This is a

special characteristic of tracking-by-detection approaches because the generated training samples usually depend on the detected object position and, thus, the approach performs self-training during runtime. This implies that errors in the detection process may cause to imprecise training sample generation. The imprecise training data causes the model to drift away from the actual object properties which again causes more imprecise detection results. Thus, the error is accumulated over time and may cause tracking failure.

In this section, we describe two common ways to estimate the object position, sliding window and particle filtering. For both concepts, we list sample approaches and explain the modifications they apply to the concepts. Subsequently, we explain several concepts of training sample generation and list approaches that make use of these concepts. Finally, we discuss evaluation criteria that are used to measure the quality of the estimated object position given ground-truth annotation.

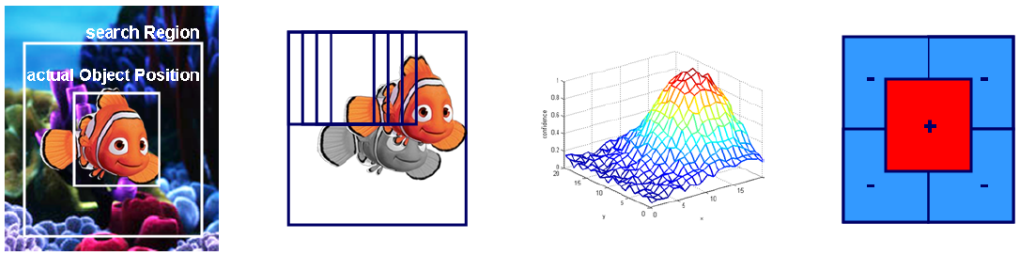
4.1 Detecting the Object Position

In single object tracking, finding the object in the current frame often boils down to finding the maximum in a similarity function between image regions and the object model. This introduces a search problem in a multi-dimensional likelihood function where the dimensionality is given by the number of parameters in the object model. Especially for more complex object representations, such as multi-part models, the number of candidate positions to evaluate heavily influences the runtime of the approach.

One way to reduce the computational complexity of the size of the search-space is to assume natural behavior of the tracked object, such as physical constraints (i.e., conservation of momentum). This results in a limited distance the object can move from one frame to another. However, this does not work for animated or comic films and explains the poor performance of many tracking approaches on such sequences.

4.1.1 Sliding Window

The easiest way to establish the similarity distribution is to perform a exhaustive search in the parameter space. In object detection, this is known as *sliding window*. For most learning-based tracking approaches, only translational movement of the object is considered. Then, the used learning algorithm is evaluated on every position of the image plane. As one can imagine, this can only be done in real-time of the used algorithm and features are very efficient.



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

Figure 4.1: The four main steps of tracking by a classifier: Given an initial position of the object (a) in time t, the classifier is evaluated at many possible positions in a surrounding search region in frame $t + 1$. The achieved confidence map (c) is analyzed in order to estimate the most probable position and finally the tracker (classifier) is updated (d) (images from [60]).

While using pixel-based object probabilities, Avidan [8] estimated the position of the object by sliding a rectangle over the probability map. Grabner and Bischof [60] used efficient features that enable real-time processing (see Figure 4.1). After them, many patch-based approaches (see Section 2.1.3) used the same concept because the authors focused on the learning concept rather than the engineering part of the tracking-by-detection loop. To decrease the runtime of the brute-force search, most approaches limit the radius of the search based on the assumption of smooth motion, which means that the object will appear near the current position in the next frame. Of course, if the dimensionality of the search-space increases, the runtime increases exponentially. Therefore, [157] utilize integral images and present an efficient way to incrementally calculate the objective function.

4.1.2 Particle Filtering

Since sliding window creates a dense estimate of the similarity function, it has one major problem: the computational complexity. Regarding the degrees of freedom, the search space grows exponentially which makes it expensive for more than 3–4 degrees (i.e., translation, rotation, scaling), especially if the motion within the degrees may be large (i.e., large displacement). Therefore, it would be beneficial to estimate the posterior distribution of the similarity given a fixed number of measurements to estimate the state of the object.

Particle filtering (see [4]) can be used to effectively estimate the state of a system using a sequence of noisy measurements $\mathbf{z}$ according to a set of N_P weighted particles $\left\{\mathbf{x}_{1:k}^i, \omega_{1:k}^i\right\}$

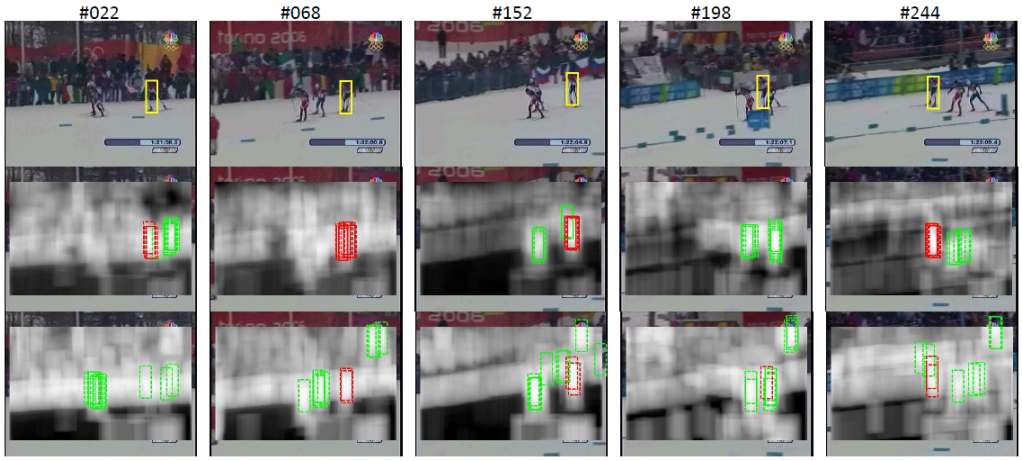


Figure 4.2: A tracking example of 250 frames. Top: ground truth tracking results of the right skier (yellow rectangles). Middle: likelihood maps using 16^3 bins RGB histogram. Bottom: likelihood maps using 16 bins intensity histogram. On each likelihood map, 10 best local optima are overlayed and labeled as correct (red dashed rectangles) or wrong (green dashed rectangles) based on their overlap with ground truth (images from [157].

with $\sum_{i=1}^{N_P} \omega_k^i = 1$ and time $k \in \{1, \ldots, t\}$ denoted as $1:k$ within the following equations. The posterior density $p\left(\mathbf{x}_k|\mathbf{z}_{1:k}\right)$ can be estimated using the observation model $p\left(\mathbf{z}_k|\mathbf{x}_k\right)$, the state transition model $p\left(\mathbf{x}_k|\mathbf{x}_{k-1}\right)$ and the proposal density function $q\left(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k\right)$ using Eq. (4.1) and (4.2):

$$\omega_k^i \propto \omega_{k-1}^i \frac{p\left(\mathbf{z}_k|\mathbf{x}_k^i\right) p\left(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i\right)}{q\left(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k\right)} \tag{4.1}$$

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N_P} \omega_k^i \delta\left(\mathbf{x}_k - \mathbf{x}_k^i\right). \tag{4.2}$$

Choosing the importance density to be the prior $q\left(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k\right) = p\left(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i\right)$ reduces Eq. (4.1) to $\omega_k^i \propto \omega_{k-1}^i p\left(\mathbf{z}_k|\mathbf{x}_k^i\right)$, where the particle weights are directly proportional to the observation model. These formulations allow for iterative estimation of the posterior distribution using only the actual measurements and the last object state density, which is given by the finite set of particles, where each particle simulates the object behavior using Monte-Carlo simulations and a motion model. To avoid the degeneracy of the particle set, re-sampling of the weights is done regularly.

In the last decade, particle filtering has been used extensively in many tracking applications to overcome the complexity of sliding window approach especially if the used object model is rather complex, as described in the following section.

4.1.3 Detection using part-based and combined Models

Recent approaches use part-based geometric models (see Section 2.1.4) or a combination of different representations to describe the object. Thus, they have a higher complexity and more degrees of freedom than simple models which implies that the detection of the current object position has a higher computational effort. This raises the need for simplified detection concepts to enable real-time execution.

Basically, three concepts are commonly used to establish flexible object models:

Bag-of-words In the Bag-of-words concept, the different parts of the object representation exist in parallel and do not have a geometric relation to each other. This means that each part results in a score for a region or position and the scores are accumulated to determine the combined result. Naturally, this means also an accumulation of the detection complexity of the overall approach (e.g., [40]).

Layered Splitting the representation into several layers, the object is often described from coarse to fine. Thus, the first layer is often used to find candidate regions that limit the computational effort for subsequent layers. Additionally, the layers may complement each other during the learning phase (in e.g., [24, 82]).

Part-based Part-based representations are very common for high quality object detection but did not find their way into tracking because of the computational complexity of the inference of the overall result. In our application (see Section 5.3), we present a way to utilize the generalized Hough transform to enable part-based tracking. Here, inference is solved by simple voting towards an expected object center of the individual parts.

While many other concepts for part-based detection exist, most of them fit into one of those three categories or share the same computational complexity.

4.2 Training Sample Generation

While most related work describes in detail how the learning algorithm works, sampling of the used training data is mostly a miracle. Thus, when re-implementing related approaches, it's hard to establish similar results due to the missing information. Matthews et al. [102] discuss the *template update problem*, which is to find a trade-off between two extremal cases of online adaption:

Complete update The appearance of the object is determined by the current frame, information from previous frames is discarded. Thus, the object model is always up-to-date, but errors are introduced immediately.

No update After the very first example, which is usually user-annotated, no additional information about the object is introduced into the object model. Thus, errors cannot be introduced and the model will be very stable. However, if the object changes, the model will not fit to the novel appearance.

This problem can also be interpreted as an instance of the *stability-plasticity dilemma* [66]. In tracking, it turns out that the update mechanism has to distinguish between valid (i.e., 3D rotations, scale changes, non-rigid deformations) and invalid transformations (i.e., occlusions, inclusion of background) of the object to correctly adjust the update strategy.

To establish a general interpretation for sampling and labeling of training data, we use the active learning framework. An active learner can be described by the quintuple (H, s, T, L, U) [94], where H is a classifier, s is a sampling function which identifies valuable samples, T is a teacher (supervisor), L is a set of labeled data and U is a set of unlabeled data. In general, an active learning process can be described as follows. First, a classifier H is trained by the labeled samples L. Then, the sampling function s selects valuable samples $\boldsymbol{v}_j$ from U. For those samples the teacher T assigns a label c_j , which is used to update the classifier H.

4.2.1 Geometry-based sampling

One approach to generate training data for the learning algorithm is to use a fixed sampling and labeling scheme based on the current object position. Using such a sampling scheme, one has to trade-off between two aspects: (a) Sampling negative samples in a very small surrounding of the object decreases the complexity of discriminative models because the part of the image where the object can be discriminated is small; (b) Sampling at a larger distance decreases the risk of labeling errors because the current object position might not be very precise and a "safety zone" that is not considered for learning can be beneficial. Considering the rising processing power available, a more complex model and distant sampling is the easiest way to improve the robustness of a tracking approach.

In their patch-based tracking framework, Grabner et al. [61, 62] use the current detection as a positive sample and four negative patches placed at the corners of the positive sample. Babenko et al. [12] create their positive training bag in a radius of a few pixels

around the current position and perform random sampling in a larger distance to generate negative samples. This sampling scheme has also been applied in various other approaches (e.g., [55, 133, 166]). For a more formal definition, U is defined by the set of samples that can be sampled from the current image. Then, s performs random sample selection within a specific region that forms a "doughnut" around the target object and additionally includes the current object position in the set of samples. The teacher T uses the assumption that the object has been correctly detected and labels all samples excluding the current object position as negative. Usually, all samples are weighted equally. Figure 4.3 displays several possibilities for geometric sampling. Nevertheless, pure geometric sampling highly depends on the correctness of the current object position.

(a) Current object position

(b) Local sampling (for supervised learning, e.g., [61])

(c) Distant sampling (for MIL), e.g., [12]

(d) Distant sampling (for supervised learning, e.g., [133])

Figure 4.3: Geometry-based training sample generation (Green: current object position; Blue: positive training samples; Red: negative training samples).

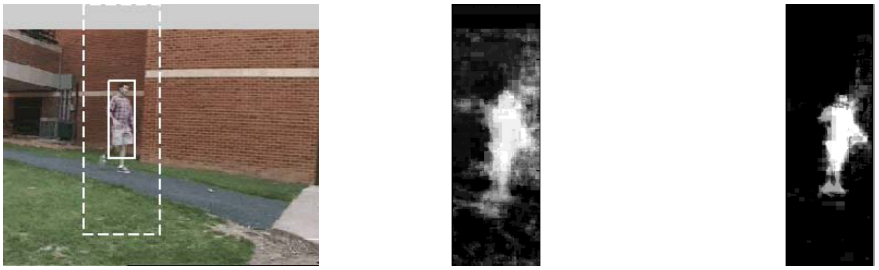
4.2.2 Confidence-based Sampling

To overcome completely ignoring the learner H in the purely geometry-based sampling methods, different ways to incorporate H have been developed. Avidan [7, 9] performs an outliers rejection scheme to lower the risk of false labels. In his tracking approach, he uses pixel-based classification confidences but rectangular object boundaries. While patch-based approaches have to live with the fact that the rectangular regions may include portions of background, individual pixels are rejected for training in [7], based on their confidence. Formally, the training label of pixels within the object region r_j is defined as

$$c_j = \begin{cases} +1 & \boldsymbol{v}_j \in r_j \wedge H(\boldsymbol{v}_j) < \Theta \\ -1 & \text{otherwise} \end{cases}, \tag{4.3}$$

where Θ is a threshold. Thus, labels of samples that are too "difficult" are changed to

negative. This is especially important because the used Boosting algorithm is known to be prone to outliers. Figure 4.4 shows the difference between a training map with and without outliers rejection. However, Avidan does not use a "safety zone", which may cause problems when the object is hard to distinguish from the near background.



(a) Image (b) Update without outliers rejection (c) Update with outliers rejection

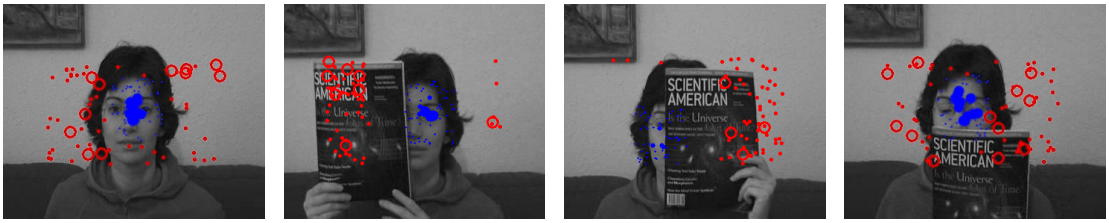
Figure 4.4: Rejection of outliers (pixel intensity corresponds to the weight of the training samples; images from [7]).

In [88], we used an ensemble setup to weight the training data during update. The training samples are randomly selected from the whole image, but the weight is assigned using multi-view training. All ensemble members are individually trained and the sample label and impact is determined using the prediction of a subset of ensemble members, denoted as H_t^*, such that

$$\tilde{c}_t = \arg\max_{c \in \mathcal{C}} H_t^*(\boldsymbol{v}) \tag{4.4}$$

$$\tilde{w}_t = \frac{1}{|H_t^*|} \sum_{t \in H_t^*} p_t(\tilde{c}_t|\boldsymbol{v}). \tag{4.5}$$

Thus, the ensemble members may retrieve different sample weights. Additionally, a geometric prior is included to prevent outliers. In Figure 4.5, the labels and weights of the training samples are depicted for selected frames of a test sequence. It clearly can be seen that on disagreement or uncertainty within the ensemble, the training samples only gain low weight, while the sample weight is increased if the ensemble members agree.



(a) Normal update (b) Partial occlusion (c) Nearly full occlusion (d) Re-appearing object

Figure 4.5: Confidence-based training sample generation (blue: positive training samples; red: negative training samples). The circle diameter represents the sample weight.

4.2.3 Labeling with Virtual Classes

This training strategy [59] has been developed to increase the complexity of the model on demand to be appropriate for the current complexity of the problem. The basic idea is to split the current scene into homogeneous parts that can be described individually by simple models. After determination of the current object position, the whole image is randomly sampled for false positive detections. If a false positive sample is found, either a new class is injected or an existing one is updated to prevent false positive detections in the next frames. To cope with the runtime injection of new classes, we use a multi-class learning algorithm and normalization of the used statistical models according to the number of samples per class c as defined in Eq. (3.23) and (3.22).

(a) Update class −4 (b) Add class −8 (c) No update (d) Update class −4

Figure 4.6: Sample generation and labeling with virtual classes (Blue: positive training samples; Red: negative training samples with according label).

4.2.4 Segmentation-based Sampling

In contrast to the rectangular description used in most tracking approaches, natural objects usually do not fit well into a bounding box. Most objects of interest are somehow articulated or non-rigid and may have an arbitrary shape when they move in 3D space. Therefore, a pixel-based granularity gives a more precise description and also a more appealing visualization. Segmentation-based tracking is not a novel idea, but the proposed approaches usually do not use a classifier to perform appearance learning.

Nejhum et al. [139] use several rectangular patches to describe the object. To improve the placement of these parts, they perform segmentation of the target object and maximize the overlap of the parts to the gained segmentation. Fan et al. [40] use discriminative colors, salient points for short-term description and bag-of-patches for long-term description of the object. Based on their description they generate scribbles (i.e., sparse foreground and background markings) that are used to perform image matting. Image matting (e.g., [93]) then generates a segmentation of the object, also considering non-binary values at the border of the object. This aims at generating smoother transitions between the object and the background if matted objects are transferred into another image. The gained foreground/background separation is used to update the model of the target object. This concept has also been used in [57], where a part-based model has been combined with a standard segmentation algorithm.

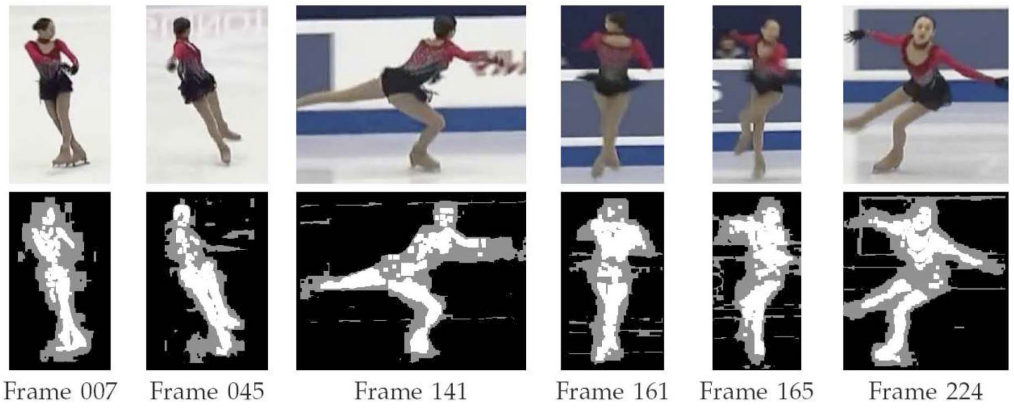


Figure 4.7: Scribble generation for selected frames: Cropped images (upper row) and the generated scribbles that are used for image matting (lower row) (images from [40]).

4.3 Detection Scores for Quantitative Evaluation

To evaluate the tracking accuracy of an approach and to compare different ones we have to perform a quantitative evaluation with respect to a common dataset. However, there does not exist a common measure for evaluation of tracking approaches which makes comparison complicated and tedious. Thus, we introduce three common evaluation criteria and note additional performance measures that give a more detailed view on the quality of a tracking approach.

Distance Measure A simple measure to quantify the quality of a detection is the distance between the tracked object center and the center of the ground-truth annotation. This measure is used in many publications but has two drawbacks: (a) It does not take into account the size of the object, which results in no penalty for differently-sized detections of the tracker; (b) It is limited by the image size because trackers and the target object usually do not leave the image. These two facts may cause misleading results and suppress the effects of a lost tracker.

Agarwal Overlap Score Another measure has been used by Agarwal et al. [2] (see Section 4.3). Given the ground-truth detection rectangle R_{GT} and the currently tracked rectangle R_T, the Agarwal score is defined as

$$score_{Agarwal} = \frac{R_T \cap R_{GT}}{R_T}. \tag{4.6}$$

Of course, this score completely ignores the scale of the detections. However, we use this score within the evaluation in Chapter 5 because standard benchmark datasets do not consider scaling in the ground-truth annotation.

VOC Overlap Score The VOC overlap criterion is defined as

$$score_{VOC} = \frac{R_T \cap R_{GT}}{R_T \cup R_{GT}} \tag{4.7}$$

and is used within the VOC-Challenge [38] to calculate the detection scores within the evaluation dataset. This score is widely used within the detection community but not for tracking approaches. Having a look on Figure 4.8 shows that using the VOC overlap criterion makes it much harder to achieve a high value in comparison to the Agarwal criterion. However, it also results in a value of zero if there is no overlap between tracking

result and ground-truth. Thus, it does not influence the ranking of the different approaches significantly but requires different levels of accuracy of the ground-truth annotation.

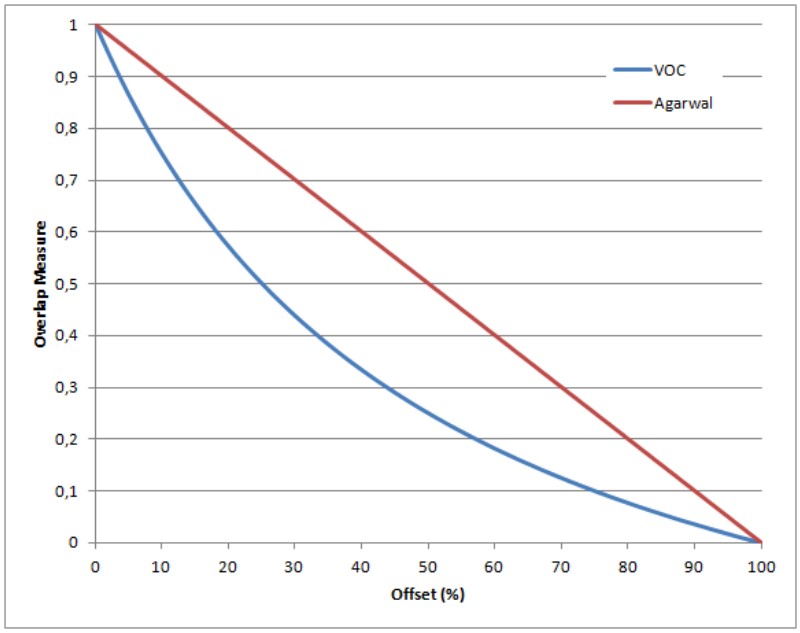


Figure 4.8: Comparison of VOC and Agarwal Overlap Scores: While the Agarwal score gives 0.5 for 50% overlap, the VOC score already decreases to 25%. This penalizes inprecise detection results, but requires highly accurate groundtruth annotation.

4.3.1 Robustness of Scores

Many tracking approaches make use of randomization which causes different tracking result in each run. Therefore, the score of a single run is not meaningful. This can be compensated easily by averaging over several runs or giving the median result of several runs. However, the averaging hides frames where the track is lost in a single run. This is especially a problem when using standard sequences to compare to other approaches because the possibility of re-detecting the object is given. In real-world scenarios, where the camera movement may be controlled by the tracker itself this may not the case. A common method to overcome this corruption of the actual performance of the tracking approach is to also report the number of lost frames or the number of frames until tracking failure. This can be easily calculated for the Agarwal or VOC score.

Chapter 5

Implemented Approaches

Contents

In this chapter, we focus on three particular tracking approaches that have been implemented and evaluated in this thesis. They focus on improvements of the statistical model (i.e., the learning algorithm), the geometric model, and the update mechanism of

the tracking loop. We evaluate all three approaches against related work to demonstrate their strengths and weaknesses.

In Section 5.1, we present *Online Random Naive Bayes for Tracking* [55]. Therein, an online learning algorithm based on Naïve Bayes (NB) is used to establish the statistical model and is evaluated for machine learning and visual tracking. The object representation and update methodology used for tracking has been inherited from related approaches, such as [61] and [133].

In Section 5.2 we present *Context-driven Clustering by Multi-class Classification in an Active Learning Framework* [59]. We investigate both, the learning algorithm and the labeling of training data during training. The key assumption is that the tracked object can often be modeled by simple statistics, while the background may change heavily or be arbitrarily cluttered. Thus, we model complex scenes by using online multi-class learning and an intelligent way to label background samples. The same tracking concept was also used in [131] but in combination online another, novel learning algorithm.

In Section 5.3 we present *Hough-based tracking of non-rigid objects* [56–58], where we investigate all parts of the tracking loop, representation, learning and labeling of samples. We learn a flexible representation based on Hough-voting using an Online Random Ferns (ORFes) classifier and utilize GraphCut segmentation [130] to improve the labeling process. The combination of these methods enables tracking of highly non-rigid objects and partial occlusions quite naturally.

Finally, in Section 5.4 we discuss the assets and drawbacks of the presented approaches and compare them to each other.

5.1 Online Random Naive Bayes for Tracking

In Section 3.5.1, we propose Online Random Naïve Bayes (ORNB), a randomized ensemble learning algorithm. Compared to Online Random Forest (ORF) [133], the algorithm utilizes several simplifications which makes it very easy and straight forward to implement. Furthermore, the parameter set of the approach is very small. Thus, the configuration is easy and the computational complexity and memory consumption is low. Finally, the classifier converges very fast to the final classification performance (see Section 5.1.2) as denoted for generative models in general in [108]. Hence, it can be used for tasks where a limited amount of training data is available.

While the work of Prinzie and van den Poel [124] compares on specific machine learning datasets only, we show that also the online version is able to compete with Random Forest (RF) and ORF on both, machine learning and object tracking tasks. Additionally, we perform several experiments that show the characteristics of the algorithm. First, we evaluate on several multi-class learning datasets and compare the performance of Online Random Naïve Bayes (ORNB) to Online Ada-Boost (OAB) and Online Random Forests (ORFs). Second, we examine different characteristics of the algorithm, i.e., speed of convergence and parameter influences. Finally, we apply the method to tracking-by-detection and compare to related approaches on a standard tracking benchmark.

5.1.1 Machine Learning

For evaluation of the learning performance, we use the *DNA*, *Letter*, and *USPS* datasets from the LibSVM [25] repository, because the are very different in terms of numbers of samples, number of classes, and number of features. Table 5.1 shows the statistics of these datasets.

Dataset	# Train	# Test	# Class	# Feat.
DNA	1400	1186	3	180
Letter	15000	5000	26	16
USPS	7291	2007	10	256

Table 5.1: Datasets used for different machine learning experiments.

For these experiments, we use a setting of 200 ORNB classifiers, each using a set of 20 features. To increase the descriptiveness of the features we combine 2 features and use histograms to model the probability distributions. Since the proposed algorithm is

randomized, we process all datasets 10 times and report the average classification error.

One of the biggest advantages of the proposed ORNB algorithm is that it is readily trained after one epoch and converges very fast to the final classification performance (see Figure 5.1). Training the classifier over several epochs does not increase the performance any more because the selected features do not change and the statistical information of the training samples from one epoch to another is the same.

For comparison to other on-line algorithms, we also train an OAB classifier with histograms as weak learners and an ORF, both trained for 5 epochs. Note that for OAB, we employ the one-vs-all strategy for the multi-class datasets. These two algorithms as way more complex than ORNB, because they use either a complex tree-growing scheme or have to train a large number of binary classifiers.

The results for these experiments in terms of classification error are shown in Table 5.2. As can be seen, on these datasets the ORNB classifier outperforms the on-line OAB classifier and reaches comparable results to the ORF. Using histograms for OAB showed better results than using stumps, which also supports the decision on using histograms for the ORNB classifier.

Dataset	ORNB	OAB	ORF
DNA	0.098	0.146	0.101
Letter	0.196	0.223	0.169
USPS	0.183	0.184	0.127

Table 5.2: The average classification error on the test sets for ORNB, OAB using histograms and ORF after 5 training epochs.

5.1.2 Algorithm Characteristics

Speed of Convergence Since theoretically the statistics of the Random Naïve Bayes (RNB) classifier in off-line and on-line version should be the same after one epoch, we show the convergence speed of the trained classifier. Therefore, we evaluated the trained classifier for the *DNA* dataset during training. Figure 5.1 shows the classification error over the amount of training data used for learning. It can be seen, that with a usage of more than 30% of the training set, the classification performance is more or less constant and reaches the final error rate. This effect has been observed on all datasets listed in Table 5.1.

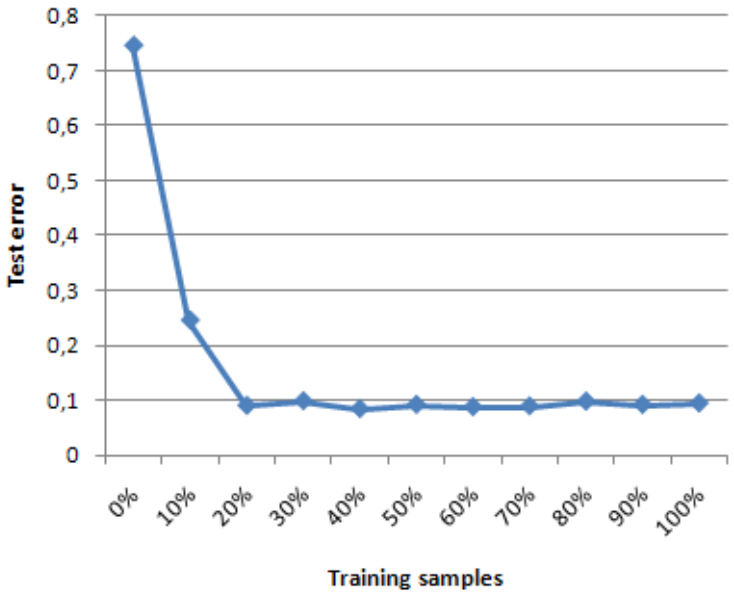


Figure 5.1: Classification performance of the ORNB classifier on the test set over percentage of trained samples of the training set for the DNA dataset.

Parameter Settings As already mentioned in previous sections, one main advantage of the ORNB classifier is the very small parameter set, consisting only of the bag size b (i.e., number of NB used), the number of hyperplane features within one NB f and the number of features used to build the hyperplane features h. To visualize the influence of these parameters, we evaluated classifiers with different settings on the *DNA* dataset (see Table 5.3). It can be seen that the dimension of the hyperplane features and the number of features dramatically increase the performance of the classifier. Although the bag size does not have that much influence, it reduces the variance of the classifier.

Runtime and Memory Consumption To examine all characteristics of the algorithm, we have a look at the runtime complexity and memory consumption of the compared classifiers. For training, all three have a linear relation between the number of samples to process and their runtime, apart from the time for calculating the boosting loss and splitting the tree nodes. The main speedup here for the ORNB comes from the fact that it is already converged after the first epoch, while the other classifiers need several training epochs (i.e., iterations on the training data).

During testing, OAB is much faster, since it only selects a small subset of features (i.e., weak learners) which are evaluated. In comparison, the complexity of ORF is linear with

Setting	Number of Features
$h = 1$	0.156
$h = 2$	0.098
$h = 3$	0.072
Setting	Number of Hyperplanes
$f = 5$	0.244
$f = 10$	0.143
$f = 20$	0.098
Setting	Number of NB
$b = 50$	0.104
$b = 100$	0.101
$b = 200$	0.098

Table 5.3: The influence of different parameters on the evaluation result of the ORNB classifier for the DNA dataset. We use the basic settings of $h = 2$, $f = 20$ and $b = 200$ and vary only one of the parameters. For $h = 3$, we can even improve the result from Table 5.2.

the depth of the tree. ORNB depends linearly in the number of used features, but has an advantage concerning parallelization in comparison to ORF. Considering the memory consumption, we can see that it is linear with the number of used weak learners for OAB and ORNB, but grows exponentially with the tree depth of ORF, which results in a higher memory-consumption.

5.1.3 Experimental Evaluation

This experiment evaluates the performance of our on-line Random Naive Bayes classifier on various publicly available tracking scenarios in comparison to trackers based on OAB [60] and ORF [133] because they represent two very prominent learning concepts in computer vision, namely Ada-Boost (AB) and Random Forests (RFs). To allow for a fair comparison to these two methods, we use the same type of features for all algorithms (i.e., simple Haar-like features). We did not implement rotation, scaling, or any complex post-processing to directly compare the learning algorithms, not any additional improvement. While this would definitely lead to more precise tracking results, the implementation is not as straight forward as one would assume[1].

[1]We do not want to go into details here, but using scaling and Haar-like features in a self-learning environment has shown to be very tricky and quickly leads to a self-affirmation of the statistical model that has nothing to do with the object under observation.

For all experiments, we use a setting of 50 NB classifiers, each consisting out of 10 hyperplanes with 3 features. To measure the probabilities within each feature we use histograms with 32 bins and the infinite impulse response (IIR) like forgetting method as described in Equation 3.19. The feature pool used for all methods is completely randomized and the initial training is done on the first frame and virtual samples generated out of this frame (i.e., applying affine transformations on the frame and train on the transformed images). The forgetting rate r was set to 0.95, which corresponds to a rather stable model and slow adaptation.

We use a subset of the publicly available sequences from [12] that give a representative overview on common tracking challenges. These sequences including variations in illumination, pose, scale, rotation and appearance, and partial occlusions. The *Sylvester* and *David* sequence are initially taken from [128], and *Face occlusion 2* from [12][2].

(a) Start Frame (b) Scaling (c) Illumination (d) Pose

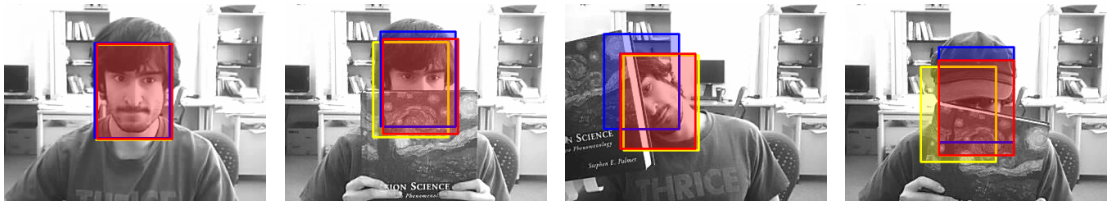
Figure 5.2: Results for *Sylvester* sequence (Red: ORNB; Blue: OAB; Yellow: ORF).

(a) Low contrast (b) Scaling (c) Out-of-plane rotation (d) Appearance change

Figure 5.3: Results for *David* sequence (Red: ORNB; Blue: OAB; Yellow: ORF).

For the evaluation of our tracker we use the Agarwal Detection-Criterion as described in Section 4.3. We measure the accuracy of a tracker by computing the average detection

[2]We do not compare to this work, since they use Multiple Instance Learning (MIL), which is an extension to the original OAB algorithm.



(a) Start Frame (b) Occlusion (c) Rotation (d) Appearance change

Figure 5.4: Results for *Face Occlusion 2* sequence (Red: ORNB; Blue: OAB; Yellow: ORF).

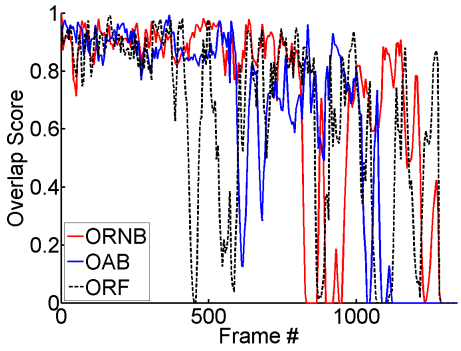
score for the entire video. To eliminate effects of the randomized feature pool, we run each tracker 5 times and report the average score of the median run.

Sequence	ORNB	OAB	ORF
Sylvester	*0.60*	0.60	**0.62**
David	**0.88**	0.39	*0.82*
Face Occlusion 2	**0.90**	*0.81*	0.72

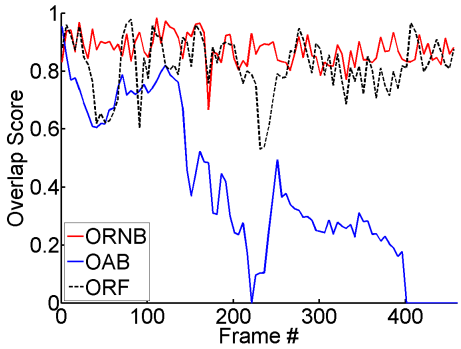
Table 5.4: Detection score: bold-face shows the best method, while italic-font indicates the second best.

Having a detailed look on the *David* and the *Face Occlusion 2* sequence (see Figure 5.3 and 5.4), it can be seen that tracking with ORNB is very stable under illumination and scale changes. Especially the alignment of the object, and thereby also the selected positive samples for self-training, are very well aligned using the ORNB. Considering large appearance variations, as present in the *Sylvester* sequence (see Figure 5.2), it seems that the learned object model is sometimes too inertial for the sequence with the given settings, even if tracking works well. Therefore, the *forgetting factor r* should be chosen appropriate to the type of application domain.

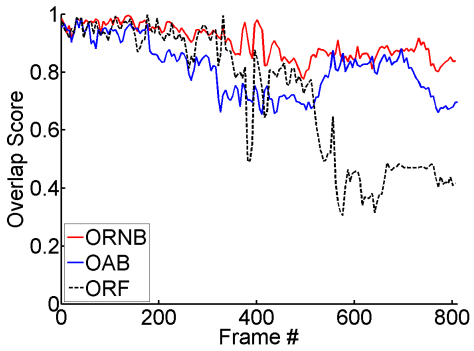
The experiments show that ORNB achieves competitive performance while being more robust to noise and outliers than the OAB approach. ORF deliver a slightly better performance than ORNB, but has a larger memory footprint. Since ORF uses (randomized) thresholding on feature values to decide which branch to use in the tree, a lot of information of the available features is not used at all, whereas ORNB directly links between feature responses and statistics.



(a) Sylvester



(b) David



(c) Face Occlusion 2

Figure 5.5: Tracking Score Comparison for different sequences.

5.1.4 Discussion

The difficult task in tracking using an on-line adapting classifier is to continuously self-train an appearance model while avoiding wrong updates that may cause drifting. Robust statistics can help to keep track of the object in case of small occlusions or partial changes of the object appearance. This is an advantage for objects that only change slightly or stay more or less constant during runtime. However, such a behavior of the object is very common in standard tracking benchmark datasets because of the use of plush toys or faces as target objects. The integrated forgetting scheme (see Eq. 3.19) allows for slow adaption as it slowly removes out-dated information. While ORNB delivers a confidence-rated classification, the underlying statistics are generative, which means that the more likely class defines the classification result, while the ratio of the likelihoods of the two modeled classes defines the confidence. However, the main advantage of the ORNB algorithm is it's simplicity.

Failure Cases Due to the slow adaption of the statistics, the algorithm will not be able to follow highly dynamic object changes during runtime. As denoted above, this is basically caused by the stability-plasticity-dilemma, that is present in every tracking application where no information about the object is present from the beginning.

5.2 Online Active Learning for Tracking

Usually, object detection and single target tracking are formulated as binary classification problems, where a discriminative classifier distinguishes between the object of interest and the background. While this is a natural interpretation of the problem, the assumption oversimplifies the real-world scenario. Thus, many vision systems work well on test sequences with low complexity but are unable to cope with real-world scenarios. Very often this problem comes from large intra-class variability that causes multi-modality in the data. This arises the need for a rather complex and large classifier complicates learning, reduces the evaluation speed, and may cause over-fitting. Additionally, in binary classification, the complexity of the two classes can vary considerably. For instance, for surveillance scenarios, the resolution is low and the object class usually can be described with a simple model, whereas the background might be cluttered, changing and arbitrarily complex.

In object detection or tracking, however, often either the object of interest or the background are changing over time. Hence, an adaptive representation would be beneficial. Therefore, the goal would be to introduce a classifier that automatically adapts its complexity to the complexity of the current task. We realize this by using a binary classifier and a multi-class representation. In particular, the background multi-modality is described by a number of *virtual classes*, which are generated autonomously using context information (i.e., the number of classes corresponds to the complexity of the background class). Furthermore, we robustly adapt the classifier to changing conditions (e.g., changing illumination conditions, changing backgrounds, etc.). A number of approaches has been proposed where the multi-modality in the data is described by multiple classes or multiple classifiers (e.g., [10, 74, 78, 146, 160]). Babenko et al. [10] developed a boosting algorithm that performs multiple pose learning, where the aim is to simultaneously split the data into groups and to train a separate classifier for each group. Another approach has been proposed by Kim and Cipolla [78], where image clustering and training of multiple boosted classifiers are performed in parallel using multiple classifiers. Torralba et al. [146] developed a multi-class and multi-view object detector, where features used for different views or different classes are shared. Wu and Nevatia [159] split the training samples into different classes by unsupervised clustering. They select the image features for clustering using a boosting algorithm. For most of these approaches the number of classes needs to be given in advance and all of them are trained in an off-line manner. Jacobs et al. [74] used the *mixture of experts*, where a separation scheme for the training set is learned and

each part is addressed by an individual expert, i.e., learner.

In the following, we describe the concept of virtual classes for unsupervised training an adaptive, scene specific classifier. Then, we demonstrate this approach for visual tracking on several publicly available datasets.

5.2.1 Virtual Classes for Scene-specific Classification

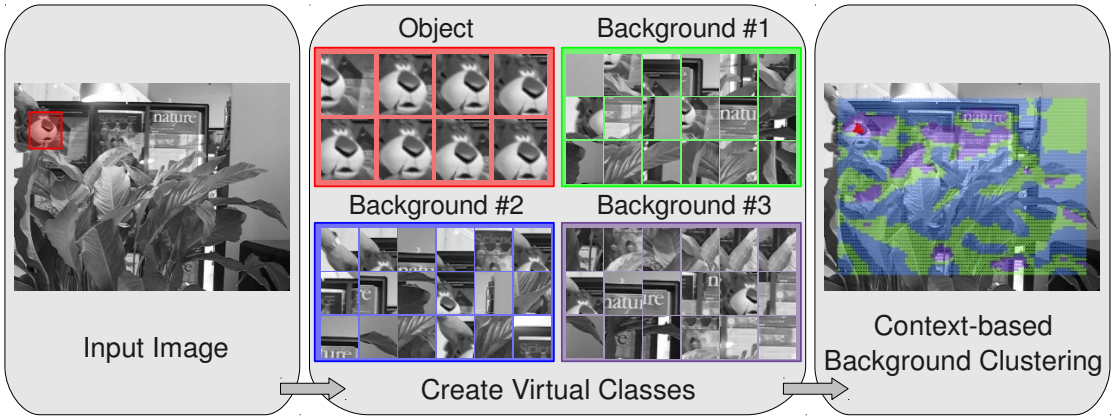


Figure 5.6: Generation of virtual classes: The input image is used for the bootstrapping (left image), the created virtual classes (right image), and an illustration of the virtual classes (center image).

We introduce a concept for context-driven adaption of the classifier complexity to the actual task and to changing situations. An online multi-class classifier is used to model the multi-modality within the data. In a first stage, bootstrapping is performed to train an initial classifier. Then, during evaluation, the complexity of the classifier is adapted to changing situations.

Context-driven On-line Clustering In order to adapt the complexity of a classifier to a scene, we propose to split the object as well as the background class into a number of *virtual classes*. The crucial point is how to find these clusters. Manual pre-clustering of the training data would require to manually label all samples, which is tedious and often not possible. To avoid this, we propose an iterative clustering approach to deal with this intra-class variability. In particular, we apply a classifier-based bootstrapping using an on-line multi-class classifier (e.g., [84, 133]) and *virtual classes*.

Such a virtual class can be considered as one normal class within a multi-class setup. However, the virtual classes are grouped into positive $\mathcal{C}_{pos}$ and negative $\mathcal{C}_{neg}$ to represent

complex data. During evaluation, a normal multi-class classification is performed. Using the group assignment of the virtual classes a binary classification result is established (e.g., a sample is classified as positive if it is classified as one virtual class of the positive group).

To start the clustering, we use an initial classifier H_0 which can be arbitrarily good. We apply this general classifier to the current scene and add virtual classes with the label c_v for samples $\boldsymbol{v}$ where the output of the classifier H_0 differs from our context knowledge about the scene or where the confidence of the classifier is very low (i.e., the samples are very close to the decision boundary). In this way, the complexity of the classifier is automatically adapted to the complexity of the current scene, i.e., for more complex scenes more virtual classes are generated. The creation of virtual classes is described in Algorithm 1 more formally and illustrated in Figure 5.6, where the input image and the clusters created within the bootstrapping stage are shown.

Algorithm 1 Generation of virtual classes during update

Require: Initialized Classifier $H = H_0$
Output: Final classifier: H

Extract background samples $\boldsymbol{V}_{neg}$
$\mathcal{C}_{pos} = \{+1\}$
$\mathcal{C}_{neg} = \{-1\}$
$v = -1$
for $\boldsymbol{v}_i \in \boldsymbol{V}_{neg}$ **do**
  $c = eval(H, \boldsymbol{v}_i)$
  // Evaluated class label for $\boldsymbol{v}_i$ should be of group $\mathcal{C}_{neg}$
  if $c \in \mathcal{C}_{pos}$ **then**
    // If not in group $\mathcal{C}_{neg}$, get new virtual class index and add virtual class
    $v = v - 1$
    $c_{new} = c_v$
    $update(H, \boldsymbol{v}_i, c_{new})$
    $\mathcal{C}_{neg} = \mathcal{C}_{neg} \cup c_{new}$
  else
    // Update classifier
    $update(H, \boldsymbol{v}_i, c)$
  end if
end for

5.2.2 Active Learning

After training the initial multi-class classifier in the bootstrapping phase, the classifier is able to discriminate between object and actual background using several virtual classes. However, this initial classifier is not able to cope with changing environmental conditions

(e.g., changing illumination conditions and background) that typically occur during operation. Hence, an on-line adaption is required which allows to adapt the classifier to changing scenes. To reduce the learning effort (i.e., the number of required samples) we use a context-driven active learning strategy.

Active learning (see Section 4.2 for a formal definition) is a widely used strategy when dealing with labeled and unlabeled data for sampling along the decision boundary in order to (a) select a reduced set of samples arranged around an optimal decision boundary and (b) to reduce the labeling effort (e.g., [27, 94, 120, 162]).

In the presented approach, the sampling function s as well as the teacher T are defined by using contextual information about the actual application. Since Park and Choi [120] showed that it is most effective to sample at the current estimate of the decision boundary, the most informative samples are those which are misclassified by the current classifier. Hence, we define our sampling function s such that it identifies samples close to the decision boundary. In particular, we run the classifier H yielding confidences on background samples extracted from the current scene and identify the samples which are very close to the decision border (i.e., result in a very low confidence of the classifier)[3]. Those samples are then labeled by using the teacher (i.e., denoted as scene context) as the following: If the decision is close to one of the actual background classes (i.e., the sample is similar to a virtual class that has already been trained), the corresponding virtual class is updated. Otherwise (i.e., the sample does not fit to an existing virtual class) a new virtual class is added to the multi-class classifier. This procedure is basically an online version of Algorithm 1.

5.2.3 Experimental Evaluation

In the following, we integrate our concept into a *tracking-by-detection* approach [28, 60]. In particular, we use online multi-class gradient-boosting derived from [91] and [132]. Boosting is the ideal playground for the active learning concept because it is very prone to errors in the labels (see [48]) of the training samples in contrast to, e.g., Online Random Forests (ORFs). However, the concept can be integrated into any other learning algorithm.

In our setup, we assume that the background changes continuously, but slowly, during runtime, which holds for most tracking scenarios. Further, we use the context knowledge that only a single instance of the tracked object is present in the scene at a time. This implies the assumption that background samples can be drawn from positions different

[3]This assumption only holds for margin-classifiers, such as boosting [135].

from the object's current position.

We use this context information to create and update the set of virtual classes within the multi-class classifier. We initialize our classifier H_0 by randomly selecting a single sample from the background and using it together with the object position to update the classifier. These two samples are the only "manual" update of the classifier and enable the context-based mechanism and the use of virtual classes. Subsequently, we perform bootstrapping to update or add virtual classes by using all background samples (see Algorithm 1). During tracking, we seek for false-positives within the current scene and perform the same update strategy on the extracted patches. We enable an adaptive number of classes in the statistics by using intelligent normalization in the weak classifier statistics and confidence rated prediction by using on-line histograms (see Sections 3.5.1 and 3.5.2).

In the current implementation, we use the estimated object position within the current frame to update the positive class, but virtual classes, semi-supervised or multiple instance learning could easily be integrated. However, since the foreground appearance usually is much more stable than the background appearance we performed our experiments using a single foreground class only. To further increase the stability of our classifier, we perform classifier averaging by combining one classifier that is not updated any more after bootstrapping and one classifier that performs online learning during runtime. Figure 5.7 compares the tracking performance with and without the use of virtual classes. In general, for the used tracking sequences, the amount of virtual classes was in a range of 3 to 5 during tracking. This number is directly related to the complexity of the scene and is automatically adapted by the algorithm.

For the evaluation of our tracker we use the overlap-criterion of Agarwal [2]. This criterion is directly related to the accuracy of the detection of the classifier, in comparison to the raw distance measure between the target and background. We compute the overlap score for the entire video sequence and run each tracker 5 times, reporting the overlap score of the median run.

Table 5.5 lists the average overlap score for several publicly available benchmark sequences [12, 128] in comparison to other state-of-the-art tracking methods. In fact, in 4 out of 8 sequences our tracker outperforms the compared methods. For the remaining 4, it delivers state-of-the-art results close to the best method.

We've carefully selected the approaches for our comparison to represent different kinds of tracking concept. Fragment-based tracking [1] uses a simple, color-based model of the

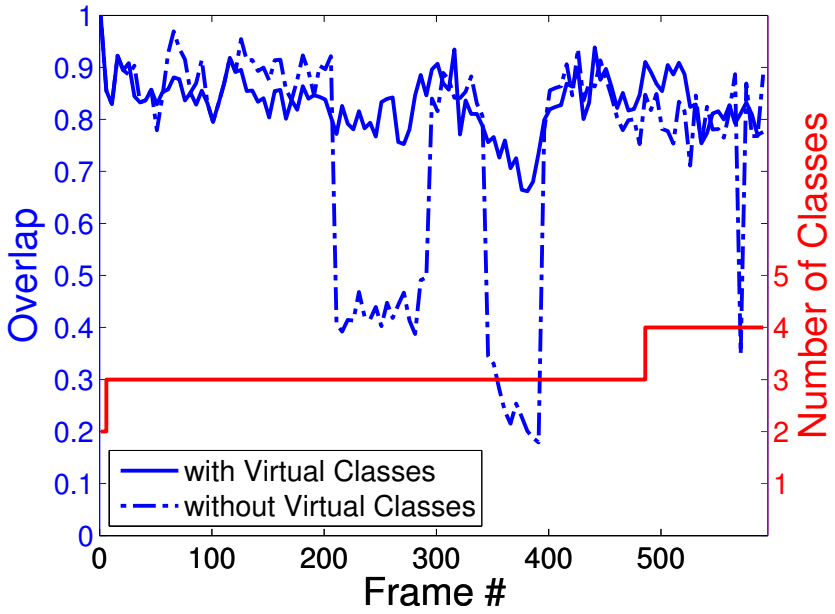


Figure 5.7: Comparison of tracking with and without virtual classes for the Sylvester sequence (blue solid: overlap for tracking with virtual classes; blue dashed: overlap without virtual classes - binary; red: number of virtual classes over time).

Sequence	CONTEXT	MIL [12]	Frag [1]	OAB [60]
Sylvester	**0.74**	0.73	**0.74**	0.60
Face 1	*0.93*	0.73	**0.94**	0.63
Face 2	**0.89**	*0.81*	0.51	*0.81*
Girl	**0.84**	0.68	*0.73*	0.57
Tiger 1	**0.65**	**0.65**	0.26	0.33
Tiger 2	*0.49*	**0.69**	0.22	0.41
David	*0.71*	**0.73**	0.52	0.39
Coke	*0.42*	**0.47**	0.10	0.25

Table 5.5: Average detection score: bold-face shows the best method, while italic-font indicates the second best.

object. To overcome multi-modality, the object is separated into rigid, rectangular parts. Multiple-Instance Tracking [12] uses a complex learning paradigm to make the self-learning process more robust. However, the used statistics are not designed to cope with multi-modal distributions. This is also the case for Online AdaBoost [60] that is the basis for our approach.

Table 5.5 shows that we reach state-of-the-art performance on this dataset and outperform both static [1] and adaptive [60] approaches. Figure 5.8 shows several illustrative

samples from different tracking sequences. It is clearly visible that our tracker is able to recover if the object was occluded or the tracking result was not well aligned. With our implementation we achieve a frame rate of about 15 frames per second on a standard desktop computer.

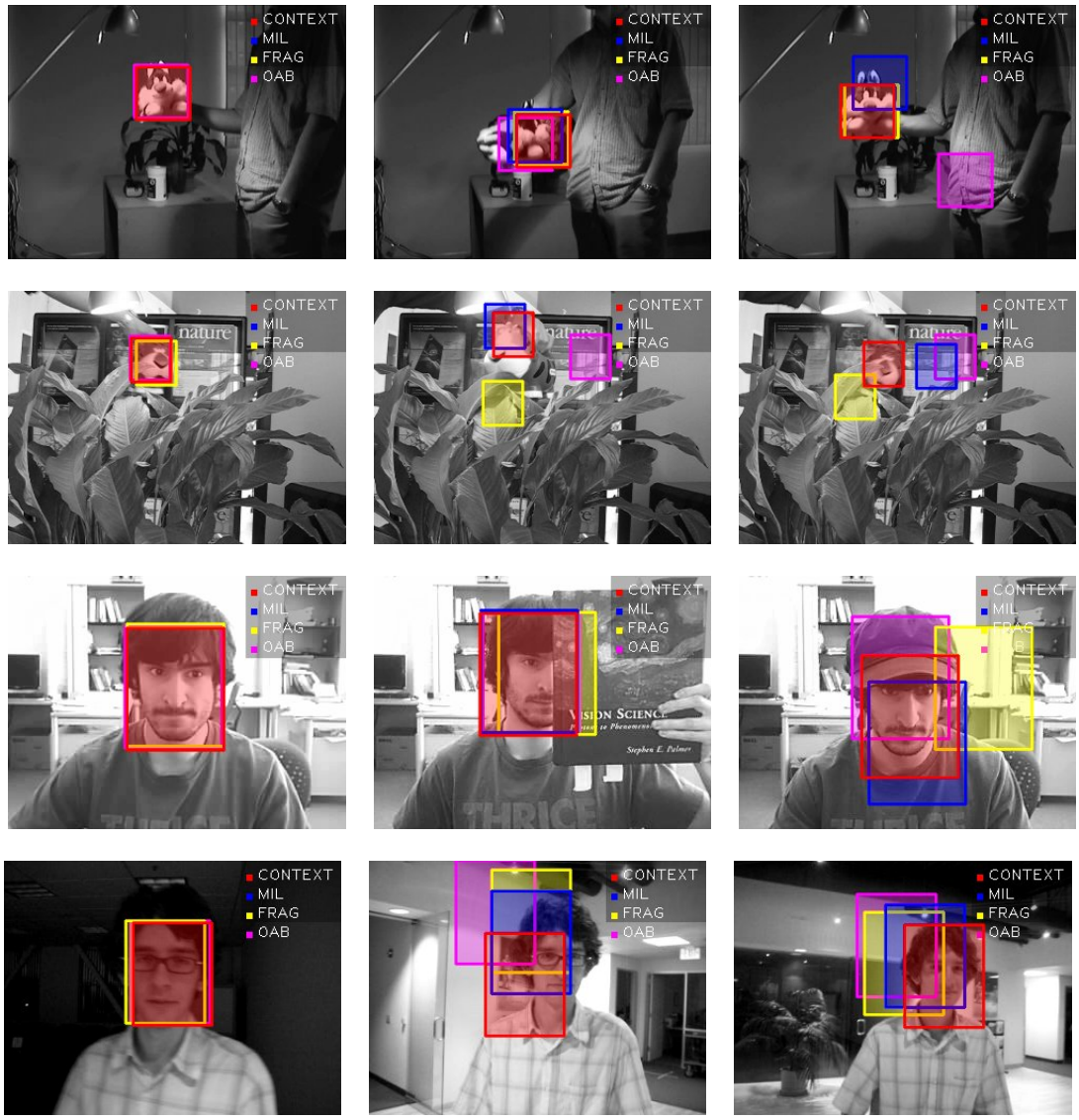


Figure 5.8: Illustrative tracking results on the Sylvester, Tiger1, Faceocc2, David sequences (red: our approach; blue: MIL [12]; yellow: Frag [1]; magenta: OAB [60]).

5.2.4 Discussion

The presented approach shows that it performs on a state-of-the-art level. However, the main message it delivers is the fact, that multi-modality in the data may be difficult do handle if the learning algorithm does not support complex statistics. Therefore, it provides a concept to automatically split the samples that are present in the scene into groups, where the group size corresponds to the level of complexity the classifier can handle. We've also applied this concept to bootstrapping for object detection [59], where we were able to cope with different scenes where the complexity was significantly different.Thus, the concept may make sense also for more complex classifiers to absorb the multi-modality of very complex scenarios. Beside that, it showed that flexible and dynamic labeling reduces the learning effort during tracking, because in the presented approach, samples that are already covered well by the current classifier are directly discarded. This reduces the amount of training samples to only a few per frame without loss in performance.

Failure Cases One issue regarding the presented approach comes to the fore if a second instance of the tracked object or a very similar one appears in the scene. Then, the labeling scheme will automatically assign a new negative class to the second object and those class will basically get very similar statistics to the positive one. This effect will finally lead to tracking failure, since learning of the appearance of the second object will be enforced in every frame. In pure geometric sample generation processes, this problem will only appear if the second object appears within the range of the update process.

Implementation Issues While implementation of a multi-class learning algorithm is quite straight forward if the underlying theory is known, the on-the-fly integration of new classes is not. Basically, there are two problems that have to be solved for that: (a) extension of the statistical model within the weak learners and (b) handling of a completely unbalanced number of samples per class. The first issue can be tackled easily by the use of dynamic maps that allow for extension during runtime. The second issue can be addressed by intelligent normalization of the statistics as described in Section 3.5.2.

5.3 Hough-based Tracking of Non-Rigid Objects

While object tracking has been a vital field of research, most of the presented approaches are limited to a bounding-box-based representation. Therefore, they have to cope with a rather inaccurate object description (e.g., parts of the bounding-box may consist of background). To avoid this problem, non-rigid or articulated objects can be represented by a part-based representation such as the Deformable Parts Model [41] and models obtained via the generalized Hough-transform [50, 101, 110]. However, these methods need a very large amount of labeled training data. This is not a problem for detection/tracking tasks where the object classes are known in advance (e.g., pedestrians [51]), but makes them infeasible for tracking of unknown objects. In contrast, level-sets have been used for tracking frequently (e.g., [18, 142]). They are able to copy with a changing shape of the object, but use holistic object descriptions. Thus, they are quite similar to kernel-based methods.

In this section, we address two major limitations of previous approaches in the tracking-by-detection domain. First, we get rid of the bounding-box description by using a part-based model. In particular, we transfer the Hough-based classification idea to the online domain by introducing totally randomized Hough Ferns using simple pixel comparisons on different feature channels as splitting tests. This allows us to robustly detect non-rigid objects. Second, we use back-projection to locate the support of our detection, which gives a fine-grained detection of object parts that have a valid geometric relation. This support guides a segmentation process (using GrabCut [130] in our case), which roughly separates the object from the background pixel-wise. While bounding-box annotations are common and sufficient for detection tasks, tracking-by-detection approaches directly use this annotation to update themselves. Since the amount of training data is also limited to the current frame, all pixels within the bounding-box that are not covered by the object are technically noise (i.e., false positive samples). Because the rough segmentation delivers a more precise description of the object than a simple bounding-box, the amount of background pixels included in the object description (i.e., noise) is much lower. Thereby, our approach, denoted as *HoughTrack*, allows tracking of objects with changing aspect ratio, scale, and orientation.

Figure 5.9 shows an illustrative example for our approach; the overall principle is depicted in Figure 5.13 and described in Section 5.3.1. Starting from a bounding-box initialization, the Hough-based detector is continuously trained with the current object appearance and guides the segmentation process. Our approach robustly tracks the object

Figure 5.9: **Tracking of non-rigid objects**: simple bounding-box initialization in the first frame (a) and continuous tracking and segmentation of the object (b) to (f) (green: initialization; red: tracking result).

during non-rigid transformations, appearance changes, and partial occlusions.

To avoid the limitations of a bounding-box, Nejhum et al. [139] propose a tracker for articulated objects. They use blocks of appearance histograms and shape descriptions but assume stationary foreground appearance. Additionally, they use a rough segmentation to find the object outline and re-arrange the blocks to maximize the overlap and similarity to the current object appearance and shape. Kwon and Lee [82] define a fixed number of object parts that are automatically renewed during tracking and track the geometric relations of these parts over time. Additionally, to reduce the computational complexity they apply Basin Hopping Monte Carlo (BHMC) sampling.

Bibby and Reid [18] describe the tracking problem within a probabilistic framework. Using pixel-wise posteriors they model the fore- and background appearance and the ob-

ject contour jointly. However, the high complexity of their theoretic framework makes it computationally infeasible. Thus, they separate the tracking of non-rigid objects into registration, level-set segmentation, and online appearance learning for continuous refinement of both object and background models.

Another branch of research is the development of segmentation-based trackers. Such methods, however, either need prior knowledge about the object or object category (e.g., [31]), use only very simple object appearance models limiting the discriminative power of the model (e.g., color histograms [18, 127]), require offline processing of the sequence (e.g., [67, 148]), or are computationally too complex to allow for real-time applications (e.g., [106, 165]). Recently, Fan et al. [39] proposed a tracking approach, where salient points within and outside the object are tracked and used to generates *scribbles* (i.e., foreground/background markings with high probability). Subsequently, these scribbles are used for image matting (e.g., similar to interactive image segmentation) which results in a high-quality object segmentation. Cehovin et al. [24] proposed a coupled-layer visual model for tracking of non-rigid objects. They combine a local layer for tracking of single patches and their geometric relations and a global layer describing holistic object properties.

In the domain of generic object detection, part-based representations have recently become very popular, since they provide excellent generalization power but still can handle intra-class variations very well [3, 23, 30, 42, 43, 45, 87, 156]. The most prominent approach is the *deformable parts model* [41], which allows to reliably detect objects even under heavy non-rigid transformations and partial occlusions. Using a latent SVM a discriminative part-based object detector is trained which is able to handle a small number of parts selected automatically during training phase. However, due to its complexity the approach is infeasible for real-time applications and no online variant of the learning algorithm exists so far.

A different, recently revisited approach is the *Generalized Hough Transform* [14, 86], which was successfully applied to object detection [50, 101, 110], action recognition [163], and tracking [51]. In addition to the detection of objects, the Hough-based classification framework also allows to determine the support of a detectors decision (i.e., which positions in the image voted for the assumed object center position). This issue has been addressed in detail by Razavi et al. [126] and is of major interest for our work. Though, in the case of tracking, only objects from a certain pre-defined class can be recognized using a pre-trained classifier. During tracking only specific instances are distinguished from each other by an additionally online estimated prior.

5.3.1 Online Hough Ferns

The theoretical basics of Random Forests (RFs) and Random Ferns (RFes) are described in Chapter 3. However, for tracking of unknown objects, offline training and node optimization makes only little sense for two reasons. First, since only the initial frame is labeled, there is not much training data available to optimize the tree or fern structure. This issue could be addressed by a tree-growing scheme as proposed by Saffari et al. [133]. However, this is computationally demanding considering the more complex statistics used for the Hough-transform. Second, considering the limited training data, optimization on the object appearance of the very first frame only may result in very tailored node tests. Though, these tests may not be able to cover the changing appearance of the object. Therefore, we use completely randomized node tests.

Hough Voting While the leaf nodes of Random Forests and Ferns only store probabilities $p_t(c|\boldsymbol{v})$ of a sample $\boldsymbol{v}$ ending up in this node being of class c, a Hough Forest additionally stores displacement vectors $\boldsymbol{d_t} \in \mathbb{R}^2$ that point toward the expected object center. Thus, a positive (i.e., $c = +1$) training sample for a Hough Forest consists of the triplet $\langle \boldsymbol{v}, c, \boldsymbol{d} \rangle$, where $\boldsymbol{d}$ is the displacement vector to the object's center position. Negative training samples (i.e., $c = -1$) do not contain a displacement vector since there is no relation towards the object center. The distribution of these votes within each leaf node can be modeled by a sum of Dirac measures according to the displacement vectors $\boldsymbol{d_t}$ from all samples $\langle \boldsymbol{v}, +1, \boldsymbol{d} \rangle$ that ended up in leaf node t. While training a tree node of a Hough Forest, either the information gain or the uncertainty of the displacement vectors of the given training set is optimized while selecting the best test [51].

During evaluation, a voting map $\mathbf{M}$ can be generated by accumulating the displacement vectors d_t, weighted by the foreground probability $p_t(+1|\boldsymbol{v})$ of the corresponding leaf node (see Figure 5.10). This is done for all possible locations in the image. The value of the voting map on a specific position corresponds to the probability of an object being centered there.

Incremental Leaf Node Statistics To establish a Hough-based classifier, we have to model (a) the foreground probability of the leaf node and (b) the displacement vectors during online training. We model the foreground probability incrementally by counting positive and negative samples arriving at a specific leaf node during runtime.

Since the tests in our classification tree are not trained to cluster similar voting di-

(a) (b) (c)

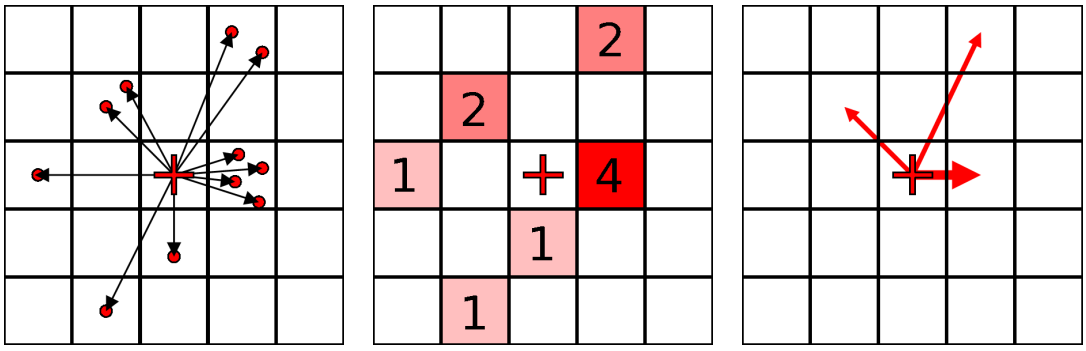
Figure 5.10: **Hough-based Detection**: Input Image (a); Patches (i.e., feature vectors $\boldsymbol{v}$) classified as foreground (green squares) (b) are allowed to cast their votes into the common voting map $\mathbf{M}$ (c). The detected object center is given by maximum of $\mathbf{M}$ (red circle).

rections, we have to handle a very diverse set of displacement vectors within a single leaf node. Therefore, we discretized the displacement space into small rectangular cells and measure the weight of each cell incrementally. When the classifier is applied to a certain image position, we retrieve the corresponding leaf node n and select a subset of strong displacement vectors from the collected displacement map (see Figure 5.11). This is done by picking v voting cells with the largest weights ω_{cell} and setting their vote strength to

$$\omega_{vote} = P_n^+ \cdot \omega_{cell}. \tag{5.1}$$

Since this learning procedure would limit the adaptivity of the classifier due to saturation effects, we apply *infinite impulse response*-like forgetting, as described in Chapter 3. The same function is also applied to the weight the negative data η_n^-. Additionally, we normalize the foreground probability P_n^+ in each leaf node n to simulate an equal amount of positive and negative training data for each tree. To adapt the displacement map to the current object configuration, we apply the same forgetting scheme to each cell in the displacement map.

The described adaptations allow for online training of Hough Forests using the current frame and to detect the object in the subsequent frame. Therefore, we apply the classifier to all positions in the image and accumulate the responding votes and their weights. After performing a non-maxima suppression, we assume the maximum to be the current object position.



(a) (b) (c)

Figure 5.11: **Displacement Map:** (a) training / input votes, (b) weighted voting cells, (c) weighted output votes for $v = 3$.

Support Beside the detection capabilities, the voting mechanism of Hough Forests can also be applied in the opposite direction to localize the *support* of a specific center position. Given a local maximum at position m, we define the support of this maximum as the sample set $S(m, \rho)$ containing all samples $\boldsymbol{v}$ that have voted to the center position m with maximum position deviation ρ. By using the corresponding displacement vectors d_t, we can back-project the original position of a sample $\boldsymbol{v}$ onto the image space. In this way, we obtain a sparse point-set of positions supposable belonging to the object that voted for the center position m (see Figure 5.12).

5.3.2 Closing the Tracking Loop

Up to now, we have defined all parts that are necessary to perform online learning in a Hough-voting based classification framework. However, there is a crucial part missing to close the tracking loop: *online training sample selection*. Selecting the right update strategy is important for online tracking-by-detection. The major problem is that the correctness of the tracking result is not guaranteed (due to misalignments, occlusions and cluttered background) but the learning algorithm has to generate training samples including as little noise as possible.

Therefore, we propose to use a rough segmentation of our object, initialized by the support set S of the detected object center. We then use this segmentation to accurately update our classifier, which allows for learning of highly non-rigid objects during tracking. Figure 5.13 illustrates the application flow and all parts of our tracking system.

(a) (b) (c)

Figure 5.12: **Support**: The object center is determined by the voting vectors of foreground patches (i.e., feature vectors $\boldsymbol{v}$). Contrary, given a specific image position (red circle), i.e., the object's center (a), we can retrieve patches that successfully voted for this position (red squares) (b). The final *support* is the sparse map of positions of those foreground patches (red dots) (c).

We use the support S of the detected object position (i.e., the parts having a stable geometric relation to the object center) to guide a rough segmentation process that extracts the object. Even if this segmentation is not very precise, it lowers the amount of label noise that is produced during self-learning. We apply the well-known *GrabCut* [130][4] algorithm to establish a reasonable binary segmentation B using the color channels, initialized by the support $S(m, \rho)$ of our object position as foreground and a maximum-object-sized rectangle as background. This rough segmentation separates our image into two regions: positive samples, located on the object and negative ones, located in the background. To further improve the visual result of our segmentation, we could easily incorporate image matting methods (e.g., [93]) or alternative interactive segmentation approaches (e.g., [134]), where our back-projection provides scribbles (i.e., the necessary user input). However, we do not rely on an exact segmentation (due to, e.g., missing parts, over-segmentations). Thus, we consider a narrow band in-between these two sets as uncertain and do not use this region for training.

To be adaptive to geometric reconfiguration of the object, we shift the object's center position to the current center-of-mass in the foreground segment, even if this point does not belong to the object. This mechanism does not distinguish if object parts are occluded or vanished. However, this simple but efficient strategy delivers accurate training data which

[4]Implementation from http://opencv.willowgarage.com.

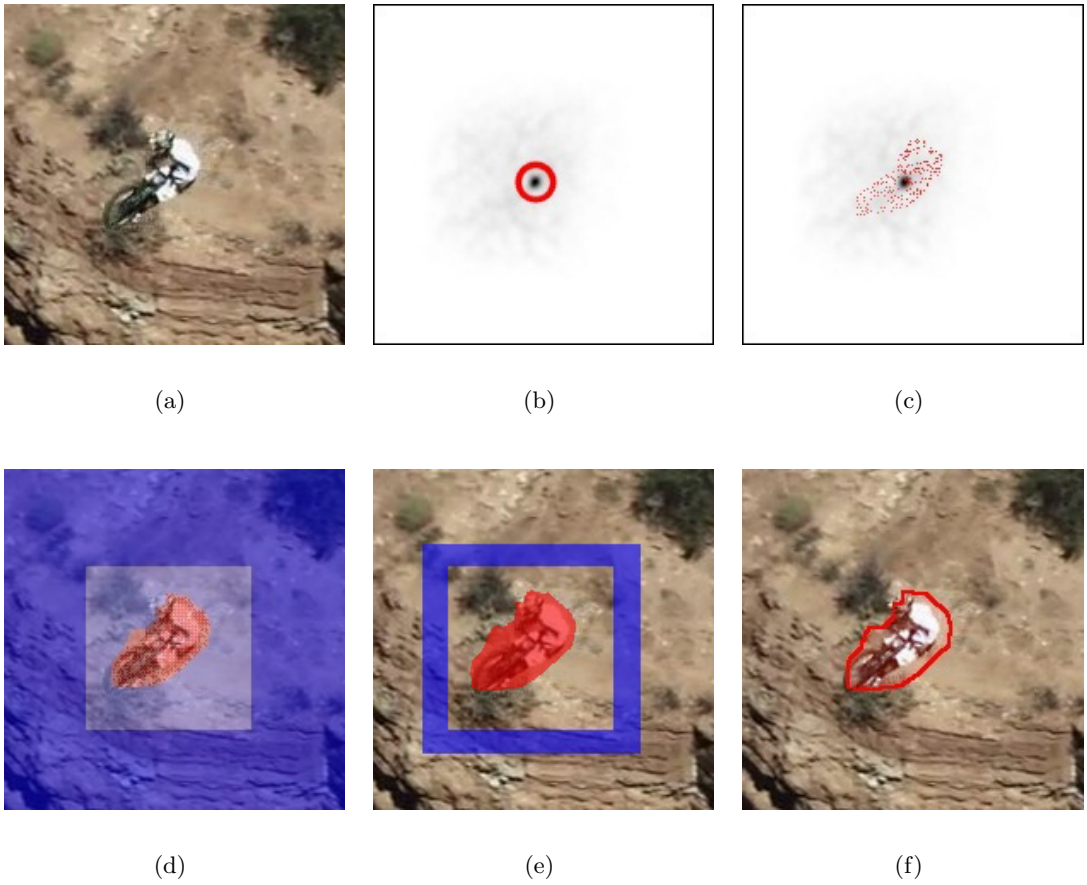


Figure 5.13: **Tracking Loop**: (a) current image, (b) Hough-based object detection, (c) back-projection and supporting image positions, (d) guided segmentation, (e) robust updating and (f) tracking result (red: foreground support, segmentation and updates; blue: background segmentation and updates).

is used to update our classifier during tracking. If the segmentation fails, our tracker acts like a bounding-box-based tracker, but Random Forests are known to be robust to noise and are able to handle a notable amount of incorrectly labeled samples. Algorithm 2 describes the complete tracking process in detail.

5.3.3 Experimental Evaluation

To demonstrate the performance of our tracking approach denoted as *HoughTrack* (HT), we evaluate and compare it to existing approaches using two different datasets. First, we compare to two standard tracking approaches, [12] and [133], using a bounding-box dataset from [12]. This demonstrates that our tracker produces competitive results on this

Algorithm 2 HoughTrack: The tracking algorithm in detail

```
Mark object in the first frame I_0
Generate object mask B from the user's initialization
Calculate center of mass of B → m
Train Fern using I_0, B and m
for all frames I_n do
  Evaluate Fern on I_n → M
  Find maximum of M → m
  Back-project votes from m to image space → S
  Segment object using S → B
  Calculate center of mass of B → m
  Train Fern I_n, B and m
  Output segmentation B
end for
```

well-known dataset. The second evaluation compares our approach to recent part-based tracking approaches [24, 82] using their set of sequences. Additionally, we collected a set of very diverse and challenging sequences including highly non-rigid object transformations. Finally, we also justify the additional effort of segmenting the object in comparison to a simple bounding-box and compare to tracking approaches that focus on accurate segmentation [26, 148].

Parameter Settings We use the same settings for all sequences: the classifier pool consists of 20 ferns and we pick the $T = 10$ ferns with the highest population for detection and the used ferns consist of $M = 1$ groups of size $S = 8$. Please note that the group size S corresponds to the tree depth D and that we use single-group ferns ($M = 1$) as we embed them into an ensemble of size T. We are using Lab-color space (3 channels), first and second derivatives in X and Y directions (4 channels) and a 9-bin histogram of gradients (9 channel) as feature vector $\boldsymbol{v}$ (as used in [50]). The used patch size of our samples is 12×12 and we return $v = 10$ strong votes from a leaf node if a sample ends up there. The forgetting constant τ is set to 0.9 (see Eq. 3.19) and the maximum support deviation ρ is 0.5 (see Section 5.3.1).

Bounding-Box Dataset For quantitative analysis, we use the publicly available tracking dataset of Babenko et al. [12] (see Figure 5.14 for illustrative samples). We compare to *MILTrack* [12] using the original configuration of 50 weak classifiers and *Online Random Forests* [133] using 50 trees and standard settings provided by the implementation. Since the compared trackers only report bounding-boxes, we also convert our result to bounding-

boxes of original size, centered around the center-of-mass of our segmentation. Table 5.6 clearly shows that our approach delivers competitive results, even not considering partial and full occlusions in the evaluation due to the lack of annotations.

Sequence	**HT**	MIL [12]	ORF [133]
David	**100/9**	84/23	95/16
Sylvester	**99/7**	93/11	71/19
Girl	86/38	85/32	**99/16**
Face Occlusion 1	**100**/22	91/27	**100/11**
Face Occlusion 2	**100**/20	94/20	70/29
Coke	24/21	**46**/21	17/45
Tiger 1	45/35	**78/15**	27/48
Tiger 2	71/**16**	**78**/17	21/44
Average	78/21	81/21	63/29

Table 5.6: **Babenko Sequences**: Percentage of correctly tracked frames ($score > 0.5$) / mean center distance in pixels for all sequences and average.

Based on the ground-truth annotation included in the dataset of Babenko et al. [12], which is represented by a simple bounding-box of the same size as the initialization, our tracker cannot be compared fairly with other bound-box-based trackers because object occlusions are ignored completely. We measure the tracking accuracy using the Agarwal-criterion [2], which is defined in Section 4.3. We report the amount of successfully tracked frames ($score > 0.5$), since this value is less sensitive to the effect described above. Additionally, we state the average center distance error, as also reported in [12]. Figure 5.14 shows selected frames from the dataset and demonstrates that the raw accuracy values from Table 5.6 fail to meet the true performance of our tracking approach.

Tracking of Non-Rigid Objects Since the intended purpose of our tracking approach is the tracking of objects that may deform during runtime, we want to demonstrate the performance on several challenging sequences. Therefore, we have collected several videos showing different ranges of complexity and non-rigid deformations, consisting of about 2500 frames. We compare to Basin Hopping Monte Carlo Tracking (BHMC)[5] [82], and LGT[6] [24] because these trackers also perform tracking of non-rigid objects. We also include the sequences provided by the authors in our comparison (see Table 5.7). However, both trackers do not report a segmentation of the object, but a bounding-box containing all tracked parts.

[5]Implementation from http://cv.snu.ac.kr/research/~bhmctracker/.
[6]Implementation from http://vicos.fri.uni-lj.si/lukacu/.

Figure 5.14: **Illustrative Tracking Results**: Selected frames from the *Babenko Sequences*.

We also list the results of Online Random Forests (ORF)[7] [133] and MILTrack (MIL) [12], two bounding-box-based trackers that are not designed to cope with the amount of transformation presented in these videos. Since these trackers cannot adapt the aspect ratio of the bounding-box, we accept the tracking result to be correct if the center position of the tracked bounding-box is roughly correct, although the result is much more inaccurate than using the three part-based approaches.

Table 5.7 depicts tracking results of the selected approaches evaluated on our sequences. We have denoted the percentage of frames for each sequence until the tracking approach fails by visual inspection. Figure 5.17 shows some selected frames of our sequences and our tracking results.

Sequence	**HT**	BHMC [82]	LGT [24]	ORF [133]	MIL [12]
Cliff-dive 1	**100**	**100**	**100**	**100**	**100**
Motocross 1	**100**	5	**100***	15	17
Skiing	**100**	–	5	5	10
Mountain-bike	**100**	50	**100***	**100**	41
Cliff-dive 2	**100**	30	26	50	30
Volleyball	**100**	60	**100***	45	**100**
Motocross 2	**100**	25	**100**	10	**100**
Transformer	**100**	**100**	**100**	**100***	**100***
Diving	75*	**100**	**100**	30	43
High Jump	**100**	**100**	60	5	10
Gymnastics (BHMC)	**100***	**100***	50*	65	55
Dinosaur	95	32*	**100**	18	25
Gymnastics (LGT)	60*	31*	**100**	15	17
Hand 1	**100**	–	**100**	8	20
Hand 2	50*	13	**100**	5	10
Torus	**100***	75	**100**	4	6
Average	**92**	59	84	36	43

Table 5.7: **Tracking of non-rigid objects**: Percentage of frames correctly tracked until failure (tracks that only include parts of the object are marked with *).

Bounding-Box vs. Segmentation-based Tracking The major remaining question is if the effort of an additional segmentation is justified. Therefore, we perform a simple experiment comparing our approach with and without the subsequent segmentation step. Thus, the only difference between the compared methods is the set of update patches that is used to update the classifier.

[7] Implementation from http://lrs.icg.tugraz.at/download/.

We use a subset of the sequences from Sections 5.3.3 and 5.3.3, which have different grades of deformation and occlusion. The first block of Table 5.8 shows the comparison for standard sequences taken from the Babenko [12] dataset including Sylvester, Girl, Face Occlusion 2, and Coke. The target objects in this sequences only get slightly deformed, which does not affect the bounding-box version of our approach too much (e.g., Sylvester, Girl). However, significant occlusions (e.g., Face Occlusion 2) decrease the tracking performance and the tracker starts to drift if the occluding object is not removed from the update region. Also out-of-plane rotations of the appearance of the tracked object (e.g., Coke) decrease the performance because the segmentation helps to stick to the original object if the appearance is different than that of the background.

Comparing both versions on more challenging sequences Motocross 1, Volleyball, Transformer, High Jump, Gymnastics (LGT), and Hand 1 from [24, 56, 82] including highly non-rigid deformations of the target object, the influence of the segmentation gets much higher. This can be clearly seen in the second block of Table 5.8, where rotations (e.g., Motocross 1 and Gymnastics (LGT)), non-rigid deformations (e.g., Volleyball, Transformer, and High Jump) and fast motion (e.g., Hand 1) are present. Overall, the results with segmentation are far better than without, which experimentally justifies the additional effort of back-projection and segmentation of the target object.

Sequence	**with Seg.**	without Seg.
Sylvester	**99**	90
Girl	**86**	84
Face Occlusion 2	**100**	91
Coke	**24**	10
Motocross 1	**100**	5
Volleyball	**100**	42
Transformer	**100**	30
High Jump	**100**	14
Gymnastics (LGT)	**60***	14
Hand 1	**100**	16
Average	**87**	40

Table 5.8: **Bounding-Box vs. Segmentation-based Tracking**: Percentage of correctly tracked frames for selected sequences and average.

Comparison of Segmentation Quality Although, our method does not rely on highly accurate segmentations, we compare our tracking result to two approaches that especially focus on that. We use the sequences presented in [148], and compare to Motion Coherent

Tracking (MCT) [148] and Adaptive fragments-based tracking (AFT) [26]. Table 5.9 shows that our approach gives reasonable segmentation results for 4 out of 6 sequences. However, our approach fails on tracking the Penguin sequence due to similar colors in the background. This is especially a problem of the used GrabCut implementations, separating foreground and background using a color-based Gaussian mixture model.

Sequence	**HT**	MCT [148]	AFT [26]
Parachute	*350*	**235**	502
Girl	3301	**1304**	*1755*
Monkeydog	*651*	**563**	683
Penguin	16097	**1705**	*6627*
Birdfall	*271*	**252**	454
Cheetah	**1037**	*1142*	1217

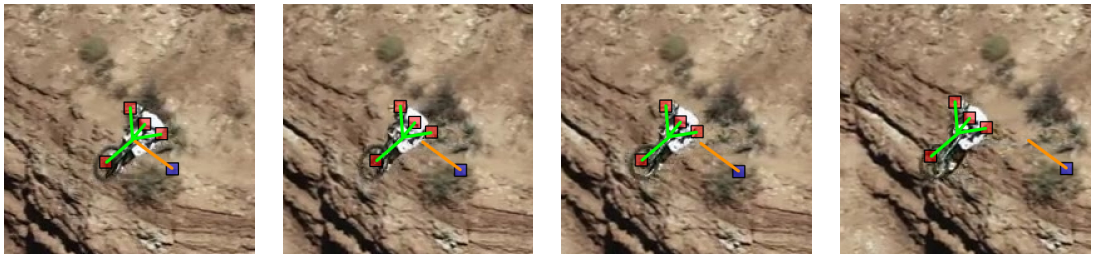
Table 5.9: **Segmentation Quality**: Average number of wrong segmented pixels per frame (**bold** numbers mark the approach that performs best, *italic* numbers second best).

5.3.4 Discussion

Tracking of unknown, non-rigid objects is a hard task, because of the lack of prior knowledge. While the object model has to be updated during runtime to cope with appearance and illumination changes, the tracker has also to distinguish between valid and invalid transformations of the object. Object parts that have not been visible may appear during runtime, while others may disappear. Therefore, the sample generation (i.e., the decision which part belongs to the object and which not) is extremely important and substantially influences the tracking result. While mis-detections and small failures may not disturb the visual result much, the inherent self-training of the classifier enforces the propagation of errors, which may finally lead to drifting and to failure of tracking. The challenge is now to update the detector regularly without introducing such failures. Therefore, we use the assumption that the overall object exhibits similar appearance. We utilize back-projections to initialize this appearance model (in our case a Gaussian mixture model of the color channels as provided by the OpenCV GrabCut implementation) and let the segmentation algorithm find the object boundaries. Even if the segmentation does not provide perfect results in every frame it improves the sample selection noticeable.

Another assumption of our approach is that several parts of the object will share a stable geometric relationship within a few frames. This requirement is directly enforced by the generalized Hough-transform. While this requirement will not hold for more than

a few frames considering a highly non-rigid object deformation, shifting the objects center position to the center-of-mass of the current segmentation enforces the object model to adapt to the current needs. Combining a part-based detection method to handle non-rigid objects with a segmentation mechanism that enforces aggregation of all parts is the quintessence of the proposed approach. Figure 5.15 illustrates the geometric relationship of foreground and background patches and their corresponding votes.



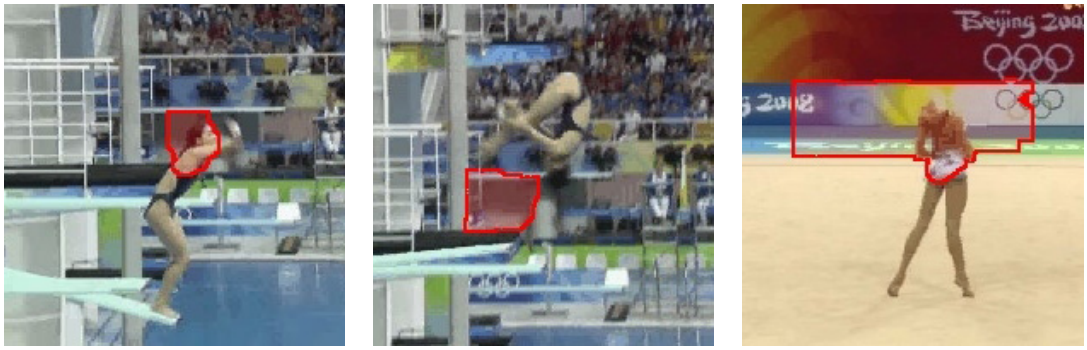
(a) frame t (b) frame $t+3$ (c) frame $t+6$ (d) frame $t+12$

Figure 5.15: **Geometric relation of parts**: Votes from foreground patches (red/green) share a stable geometric relationship and establish a stable voting maximum while background votes do not support the voting maximum if the object moves.

Failure Cases Since we use a rectangular initialization of our tracker in the first frame, the support of our detection in the subsequent frames may also include background positions because they have also been included in the initial training set. This may end up in confusion of the classifier.

However, the stable geometric relation of the background collapses as soon as the object moves and the according votes do not match the support criterion (i.e., distance smaller than ρ) any more. Only if the majority of the support originates from the near background of the object, the recognized object center will be supported by the background and the tracker is not able to follow the object any longer. This effect may occur when there is very cluttered background (Diving) or the segmentation algorithm fails due to similar colors in the background (Gymnastics) as visible in Figures 5.16 (a-c). Figure 5.16 also shows some frames where our approach fails due to fast motion (d and e) or the segmentation is not optimal due to the shape of the object (f).

Parameter Optimization The chosen settings are a trade-off between performance of the classifier and execution speed of the tracker. As stated above, $T = 10$ ferns of depth



(a) Diving (b) Diving (c) Gym (BHMC)

(d) Gym (LGT) (e) Hand2 (f) Torus

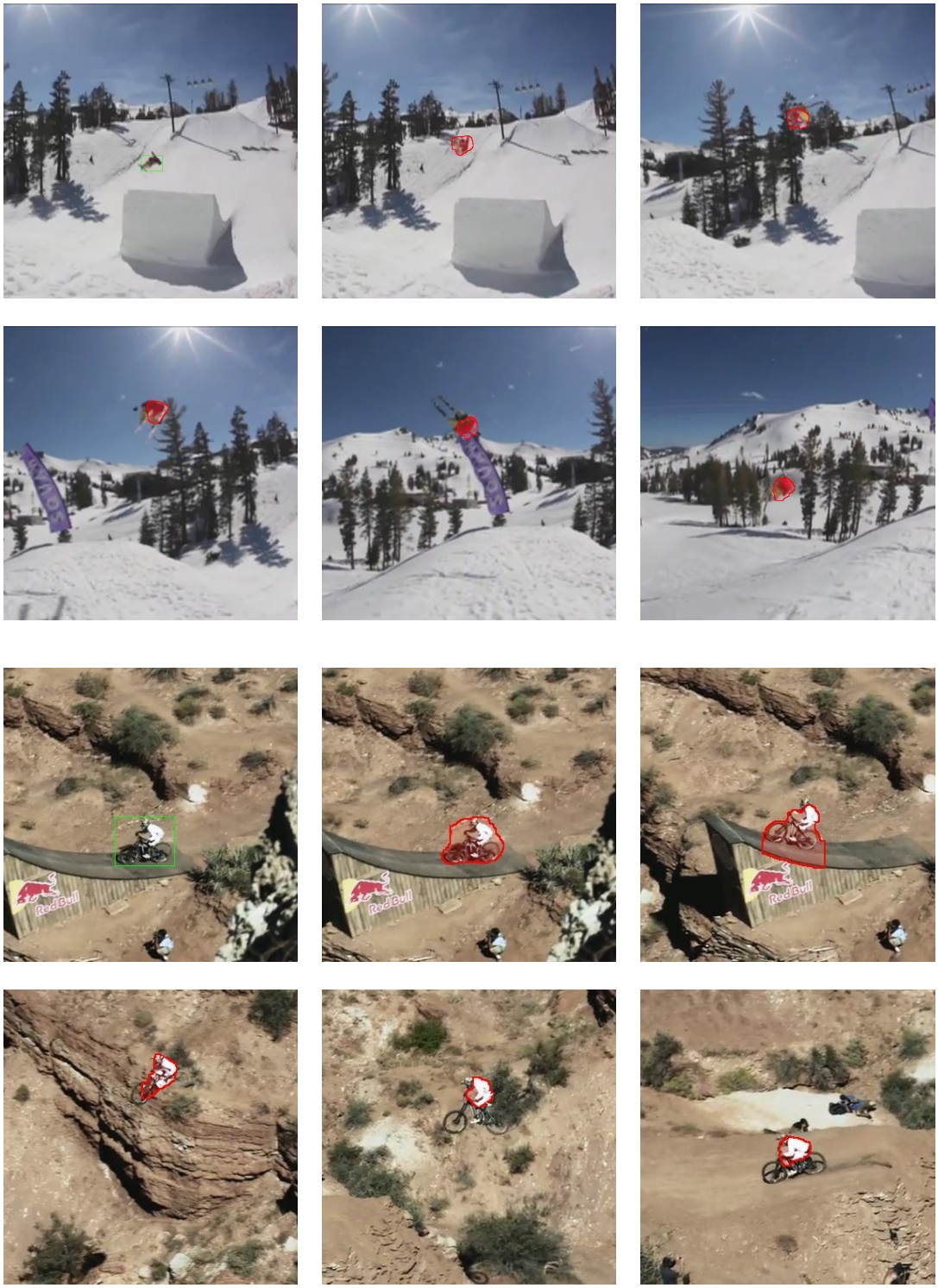
Figure 5.16: **Failure Cases**: Selected frames where the approach fails or does not work well (images are cropped for better visibility).

$S = 8$ are used during evaluation, which gives a reasonable simpler classifier than used in [51], which uses 15 trees of depth 12. Thus, the statistics within the leaf nodes of the used Hough Ferns are not as distinctive as that of Hough Forests. Additionally, no clustering based on the voting direction is performed, which results in a scattered displacement map. A viable way to automatically optimize the fern depth S would be to grow ferns on demand similar to Schulter et al. [137]. Using online Hough Forests would also enable to perform test optimization during runtime, but using forests is computationally more complex and the obtained tree structure is less flexible. The number of returned votes v can also be adapted such that the returned votes represent the majority of the weight of all votes. This would automatically adjust the number and the weight of the returned votes according to the spreading of the displacement map within each leaf node. The forgetting is a trade-off between speed of adaptation of the classifier and robustness of the model.

Complexity The memory complexity of the approach is $O(T \cdot M \cdot 2^S)$ (approximative 195 MB for the used settings) and the runtime mainly depends on the object size (i.e., number of pixels that cover the object). Without parallelization our approach runs with about 2–5 fps on a 3GHz (single core) desktop computer. In the current implementation, the runtime consumption is split to 20% for feature calculation, detection and training of the classifier, respectively, and 40% for the GrabCut segmentation. Thus, the whole process could be speeded-up by using a more efficient segmentation algorithm and parallelizing the fern implementation.

Implementation Issues We have defined a maximum object size for background initialization of our segmentation algorithm, preventing too severe over-segmentations. This can be recognized in sequence Cliff-dive 2 (see Figure on next page). It is clearly visible that the segmentation changes over time and that it gets more accurate during tracking. To overcome changing aspect ratio of the object, we reshape the maximum object size to a square box. However, if no change in the aspect ratio is expected this setting can be switched off. For comparison to future approaches, our reference implementation and the used sequences are available online[8].

[8] Download from http://lrs.icg.tugraz.at/research/houghtrack/.



Illustrative Results: Initialization (green box) and selected frames (red segmentation).

Figure 5.17: **Illustrative Results**: Initialization (green box) and selected frames (red segmentation). More results can be found at http://lrs.icg.tugraz.at/research/houghtrack/.

5.4 Discussion

This chapter presents three individual implementations of the tracking-by-detection concept. Especially Sections 5.1 and 5.3 have a strong relation to each other. Bayes' theorem has shown to be an effective way to model the statistical distribution of related feature channels. However, since the number of features in tracking is very high, the resulting dimensionality of the statistical model precludes real-time performance. Therefore, we made several simplifications that are shared by both approaches:

- Randomization is used to create a pool of features to extract some information from the images. While this is not the best choice, it may be the only one that eliminates any bias that may be introduced by wrong assumptions.
- Histograms are used in both cases to model the distribution in the leaf nodes of the classifiers. In ORNB, we directly use them to model the probability of a sample ending up in this node to belong to the foreground, while in ORFe, we use 2D-histograms to capture the voting vectors towards the expected object center. However, in both cases we use the exponential forgetting function (see Eq. 3.19) to get rid of out-dated information and to ensure adaptivity of the classifier.
- During evaluation, the probabilities of the individual members of the ensemble is summed up for the overall result. While multiplication of the individual probabilities would be the correct choice, addition is the more stable and robust one due to the averaging effect.

As a complement, Section 5.2 presents a completely different concept to handle the complexity of the data present in the video sequences. Here, the data is autonomously separated into groups, which can be interpreted as a simple clustering algorithm. Therefore, new classes are introduced until there is no conflict of any portion of the current background with the target object. This concept works as long as there is only one instance of the object visible in the image because we assume that everything except the current detection is background. Otherwise, the active learning scheme would repeatedly penalize the detection of the second instance by repeatedly adding new negative classes that are trained with the second object instance. As expected, this will cause immediate failure of the approach. Nevertheless, the presented concept is an interesting way to overcome the problem of very complex classes that can only be handled by using huge classifier statistics which makes the learning algorithm slow and unbalanced, because usually only one class if complex.

In contrast to Sections 5.1 and 5.2, Section 5.3 integrates a flexible object representation into the tracking loop. Hough-based object detection allows for combination of a very large number of small object parts into a unified detection by combining votes of the individual parts that point towards the expected object center. Of course, the single parts are way to simple and small to give in a useful detection result, but the very large number of parts enables highly accurate center estimation without complex inference.

However, the big change in the object representation causes big changes in the learning algorithm, because the use statistics are much more complex due to the integration of the voting mechanism. Therefore, we again utilize the concept of histograms that we already use as statistics in the other applications, but in this case they are two-dimensional. The use of histograms again allow to use the established forgetting scheme. While the detection of the object center basically works straight-forward by selecting the maximum in the distribution of the center votes, there is much more that we can gain from the part-based concept. Using the back-projections helped us to immediately get rid of both, the rectangular bounding-box result of the originally proposed Hough-based object detector and the large amount of noise that is introduced into the learning process due to non-rectangular objects that are surrounded by the bounding-box. This is achieved by using an out-of-the-box segmentation algorithm that is initialized using the object parts that successfully voted for the selected center position. This was basically the enabling factor that resulted in very robust behavior and high performance of the tracking approach.

Beside the contribution, the kind of presentation we chose for the approaches may raise several questions that we want to address here:

Why didn't we compare the three approaches against each other?

The presented approaches address the individual parts of the tracking loop. However, we did not explicitly compare these approaches to each other, because they represent very different examples and interpretations of tracking: While Section 5.1 primarily focuses on the learning algorithm and statistical model, Section 5.2 presented a novel update strategy. Section 5.3 then modifies a flexible object detection approach to be used for tracking. Since the focus of the methods is quite different, a direct comparison of results on a benchmark dataset may not really give a benefit, because all three focus on very different parts of the tracking-by-detection concept.

Why din't we use exactly the same scheme to perform the evaluation of the approaches?

To compare against related work, we calculated the average Agarwal detection score (see Section 4.3) for each sequence based on the given ground-truth. For Sections 5.1 and 5.2 we reported these numbers without any further processing (see Tables 5.4 and 5.5). However, since the approach presented in Section 5.3 does not give bounding-box results, the reported scores did visually not correspond to the perceived tracking performance. This was caused by the fact that we reported a rectangle surrounding the *center of mass* of our segmentation result, but the center of mass of the objects did not correspond to the center of the given ground-truth annotation. Therefore, we decided to widely eliminate the influence of the ground-truth annotation by reporting only the correctness of the tracking result.

Another issue regarding evaluation datasets is the very limited range of operation of the objects. Thus, a tracker that lost the object can simply wait for the object to pass by its current position. Since this will not happen in real-world scenarios, we decided to stop tracking on a complete failure ($score = 0$) and report the percentage of successfully tracked frames for each sequence.

Why did we present exactly these three applications?

Given the fact that these three methods are presented with respect to the chronological order of their development, they cover the evolution of tracking-by-detection from the basic concept of [61] towards a more flexible interpretation in [56]. Many more approaches could have been presented here, but we think that these three are the corner posts of this thesis and demonstrate the different aspects of tracking-by-detection quite well.

Also, the selected approaches show that the statistical model and the update mechanism can be exchanged quite easily for bounding-box approaches (Sections 5.1 and 5.2), but using a more complex object representation may cause comprehensive changes of all other parts (Section 5.3). Altogether, we wanted to show different facets of tracking-by-detection by showing very different ideas and implementations.

Finally, what are the components to create the optimal tracking loop?

While we have evaluated and implemented a large number of different components, i.e., learning algorithms, labeling schemes, and object models, this question cannot be answered easily. The main problem is that "'tracking of unknown objects"' is an enormously broad field of applications and the requirements from task to task may vary significantly. Thus, it is very important to have a closer look on the object (i.e., static or dynamic appearance,

rigid or non-rigid shape), the scene (i.e., static camera or PTZ camera, static or dynamic background, changing illumination) and the desired output (i.e., bounding-box, convex hull, or precise shape). Finally, we think that there can be no optimal solution, as long as no knowledge about the object is available.

Chapter 6

Summary and Conclusion

Contents

Within this thesis, we examine the basic principle of tracking-by-detection, which is a popular approach for tracking of unknown objects and has been used frequently in the last decade. To explain the basic concept of tracking-by-detection, we introduce the tracking loop (see Section 1.2) that is composed of (a) object representation (see Chapter 2), (b) statistical model and learning algorithms (see Chapter 3), and (c) detection of the object and generation of new training samples (see Chapter 4). Beside a detailed discussion of every building block, we give a detailed overview on the different possibilities of implementation that have been presented in the recent literature.

For object representation, we put a special focus on the geometric model, which mainly defines the complexity of the approach. While traditional approaches preferred kernel- (see Section 2.1.2) or template-based (see Section 2.1.1) models, more recent approaches go towards part-based (see Section 2.1.4) models. Since concepts that have been used in detection tasks are usually computationally too expensive for real-time operation, many tailored solutions have been presented and are discussed in Section 2.1.4.

The extracted information is then integrated into a statistical model, mainly using machine learning techniques. We introduce the mathematical foundations (see Section 3.1), explain the relevant terminology (see Section 3.2). Technically, we focus on randomized ensemble learning techniques and describe Random Naïve Bayes (RNB), Random

Forests (RFs), and Random Ferns (RFes) and their online learning variants (e.g., Online Random Naïve Bayes (ORNB), Online Random Forests (ORFs), and Online Random Ferns (ORFes)) and discuss their use in related tracking approaches in Sections 3.4 and 3.5.

At last, regarding the tracking loop, we analyze the detection mechanism and the training sample generation process. Especially the second is very important, because this is mostly neglected in scientific publications. However, it has a large influence on the performance and stability of every tracking approach (see Section 4.2.

In these three chapters, we discuss different solutions to the individual building blocks that have been presented in related work within the last decade. Since there is a very large amount of related publications that use similar concepts, we present representative approaches that cover all different types of implementations.

In Chapter 5, we introduce three approaches that target improvement of the individual parts and perform detailed evaluations against related work. In Section 5.1, we propose a novel online learning algorithm called Online Random Naïve Bayess (ORNBs). We evaluate the algorithm based on machine learning datasets and within a tracking-by-detection framework. Additionally, the core characteristics, such as the convergence speed and the parameter influence of the algorithm are explored. While this algorithm is pretty simple to implement and configure it delivers good performance. In Section 5.2, we concentrate on the update mechanism within the tracking loop. We discover the problem of complex background classes and propose to dynamically split the background class into several virtual classes. However, this approach requires an online multi-class learning algorithm that is able to add new classes on the fly. Therefore, we propose several modifications to the weak learners that allow for use of our update mechanism and can handle highly unbalanced datasets.

Finally, we overcome the bounding-box limitation of many recent approaches by proposing a tracking approach based on ORFes and Hough-based object detection in Section 5.3. This approach investigates all three parts of the tracking-loop by using a flexible, part-based object representation, an online randomized ensemble learning method to establish the object model and an update scheme based on segmentation of the object. The combination of these three techniques results in a robust tracking approach that delivers accurate object segmentation during runtime.

6.1 Contributions of this Thesis

We awarely did not mark the contribution of the thesis in the Sections 2, 3, and 4 where it is discussed beside related work. Therefore, we summarize the individual points in the following:

Online Random Naive Bayes Learning The work of Prinzie and van den Poel [124] gave the main motivation to establish an online version of Naïve Bayes (NB) learning (see Section 3.5). It turned out that the algorithm performed quite well on classical test data while being very simple and easy to implement. Additionally, we demonstrated the applicability of the algorithm to the task of tracking (see Section 5.1.1). The experimental outcome of the experiments motivated on the use of statistical models that are able cover multi-modality, e.g., histograms, for weak learners instead if using uni-modal Gaussian distributions.

Context-based Clustering for Multi-class Learning To further improve the granularity of the statistical model, we split complex classes into several simple ones (see Section 5.2). To fit the complexity of the model to the complexity of the task, i.e., the tracking sequence, we adapt the number of classes that model the background dynamically during runtime. The experimental evaluations show that complex scenes result in very complex distributions (see Figure 5.6), even using weak learns that are basically able to model multi-modal data (i.e., using histograms). However, this requires a learning algorithm that is capable of multi-class learning. In the presented application, we used online Gradient Boost [131].

Online Hough-Ferns Online Random Forests (ORFs) have shown great capabilities in modeling of complex data. However, their tree-growing scheme is very complex to implement. On the other hand, Online Random Naïve Bayes (ORNB) is very easy to implement, but has limited capabilities in modeling of complex objects due to the limited number of features that are used. Random Ferns (RFes), a learning algorithm that can be interprets as an implementation of the semi-naïve Bayes formulation, has been introduced by Özuysal [116] and shows impressive performance on different computer vision tasks (e.g., key-point recognition). Therefore, we develop an online version of RFes, Online Random Ferns (ORFes) (see Section 3.5). Therefore, we use the findings and concepts of ORNB to calculate our statistical information online and to perform forgetting to get rid of out-dated information. To get rid of the bounding-box scheme of previous approaches, we combine the ORFes implementation with an

adaptation of the Hough-based object representation from [50].

Sampling of Training Data using Segmentation While a Hough-based object representation allows to detect flexible objects, the update mechanism presented in [50] still uses rectangular annotations. However, this does not fully utilize the capabilities of the approach since every background pixel in the rectangular box still introduces label noise into the update mechanism. Therefore, we introduce an out-of-the-box segmentation algorithm to give a more precise annotation (see Section 4.2). This results in a much better tracking quality (see Table 5.8).

In our applications, we demonstrate state-of-the-art performance of the individual approaches by comparison to related work. Section 5.4 discusses the individual characteristics of the implementations and their assets and drawbacks.

6.2 Future Work

During the implementation and evaluation of the presented approaches, several ideas could not be addressed that may lead to further improvements and real-world applicability of the tracking-by-detection concept:

Combination There are two very prominent concepts that are used in computer vision to represent objects: (a) Key-points, and (b) patches. Both have shown good performance in different tasks and approaches, but a recent trend is to combine them. This idea follows the problem that, especially for tracking of unknown objects, the properties of the object are unknown and so is the optimal representation and feature type. The use of a larger pool of diverse features and representations should be investigated to enable runtime adaption of the representation to the current requirements. It would also be interesting to combine the strength of several tracking approaches into an unified framework.

Customization Online learning has reached a level of performance where improvements are very difficult. Naturally, tuning of parameters improves the result but it is much more important to check if the learning concept fits the target application. Thus, we will adapt also other standard learning algorithms to fit to the task of tracking-by-detection. Thereby, integration of the geometric model, runtime adaptation (i.e., forgetting of outdated information) and reduced computational complexity are the main issues that will be addressed.

Flexibility In recent object tracking approaches there is a strong trend towards flexible object representations that can cope with deformable, non-rigid objects. For the geometric model, flexibility will be one of the key features of upcoming approaches which will also bring up more complex evaluation datasets consisting of natural objects performing complex motion. Reporting of an object segmentation is an interesting option, as it allows for, e.g., high-quality extraction of a moving object from an arbitrary background during runtime. However, estimation of the current orientation of the object would also be beneficial and will be investigated in the future.

6.3 Closing

In the Introduction (see Chapter 1), we raised some questions about the tasks that computer vision applications should fulfill, mainly focused on assistance for humans in their daily life.

So, what is today's state of the development?

Well, computing devices proceed entering our daily life and getting more and more ubiquitous. Nowadays, everyone carries a mobile phone, tablet-pc, digital music player and other devices that are equipped with cameras. This means that the hardware, i.e., cameras, processing power, and network connections, are today available everywhere and of course many applications using vision exist. Face recognition (e.g., used in Picasa[1] and iPhoto [2]) for name tagging, smile shutter, and focus assistance will definitely find their way into everyday's life.

All these applications share that they target a more or less isolated task or use a vast amount of data and are thus overdetermined. However, these computer vision services and applications are far from the imagination that is given by science fiction movies as denoted in the motivation of this thesis. Thus, while there is a large amount of tasks where computer vision is integrated already today, there is still a long way to go to use computer vision everywhere, where it can give a benefit.

[1] Official Picasa website: http://picasa.google.com
[2] Official iPhoto website: http://www.apple.com/ilife/iphoto

Appendix A

Acronyms and Symbols

AB Ada-Boost

CAD Computer-aided Design

CT Computer Tomography

DPM Deformable Parts Model

DT Decision Tree

EDF Entangled Decision Forest

EOH Edge Orientation Histogram

ERT Extremely Randomized Decision Tree

GPGPU General Purpose Graphics Processing Unit

HoG Histogram of Gradients

HOG Histogram of Oriented Gradients

IIR infinite impulse response

IVT Incremental Visual Tracking

LBP Local Binary Pattern

MIL Multiple Instance Learning

MNL MultiNomial Logit

MOSSE Minimum Output Sum of Squared Error

MVL Multi-view Learning

NB Naïve Bayes

OAB Online Ada-Boost

OCR Optical Character Recognition

ORF Online Random Forest

ORFe Online Random Fern

ORNB Online Random Naïve Bayes

PCA Principal Component Analysis

PBT Probabilistic Boosting Tree

PTZ pan-tilt-zoom

RF Random Forest

RFe Random Fern

RNB Random Naïve Bayes

SSL Semi-supervised Learning

SVM Support Vector Machine

TF-IDF term frequency – inverse document frequency

Appendix B

List of Publications

1. **Hough-based Tracking of Non-rigid Objects** [57]
   Martin Godec, Peter M. Roth, and Horst Bischof
   Computer Vision and Image Understanding, to appear in 2013

2. **Segmentation-based Tracking by Support Fusion** [71]
   Markus Heber, Martin Godec, Matthias Ruether, Peter M. Roth, and Horst Bischof
   Computer Vision and Image Understanding, to appear in 2013

3. **Hough-based Tracking of deformable Objects** [57]
   Martin Godec, Peter M. Roth, and Horst Bischof
   Decision Forests for Computer Vision and Medical Image Analysis
   Antonio Criminisi and Jamie Shotton (Eds.)

4. **Hough-based Tracking of Non-rigid Objects** [56]
   Martin Godec, Peter M. Roth, and Horst Bischof
   In Proceedings International Conference on Computer Vision, 2011

5. **Improving Classifiers with Unlabeled Weakly-Related Videos** [89]
   Christian Leistner, Martin Godec, Samuel Schulter, Amir Saffari, Manuel Werlberger, and Horst Bischof
   In Proceedings IEEE Conference on Computer Vision and Pattern Recognition, 2011

6. **Proceedings 16th Computer Vision Winter Workshop**
   Andreas Wendel, Sabine Sternig, and Martin Godec (Eds.)
   Verlag der Technische Universität Graz, 2011

7. **Autonomous Audio-Supported Learning of Visual Classifiers for Traffic Monitoring**
   Horst Bischof, Martin Godec, Christian Leistner, Andreas Starzacher, and Bernhard Rinner
   IEEE Intelligent Systems, 2010

8. **On-line Random Naive Bayes for Tracking** [55]
   Martin Godec, Christian Leistner, Amir Saffari, and Horst Bischof
   In Proceedings International Conference on Pattern Recognition, 2010

9. **Context-driven Clustering by Multi-class Classification in an Active Learning Framework** [59]
   Martin Godec, Sabine Sternig, Peter M. Roth, and Horst Bischof
   In Proceedings Workshop on Use of Context in Video Processing (CVPR), 2010

10. **Online Multi-View Forests for Tracking** [88]
    Christian Leistner, Martin Godec, Amir Saffari, and Horst Bischof
    In Proceedings DAGM Symposium, 2010

11. **Online Multi-Class LPBoost** [131]
    Amir Saffari, Martin Godec, Thomas Pock, Christian Leistner, and Horst Bischof
    In Proceedings IEEE Conference on Computer Vision and Pattern Recognition, 2010

12. **Robust Multi-View Boosting with Priors** [132]
    Amir Saffari, Christian Leistner, Martin Godec, and Horst Bischof
    In Proceedings European Conference on Computer Vision, 2010

13. **TransientBoost: On-line Boosting with Transient Data** [145]
    Sabine Sternig, Martin Godec, Peter M. Roth, and Horst Bischof
    In Proceedings IEEE Online Learning for Computer Vision Workshop (CVPR), 2010

14. **Audio-Visual Co-Training for Vehicle Classification**
    Martin Godec, Christian Leistner, Horst Bischof, Andreas Starzacher, and Bernhard Rinner
    In Proceedings IEEE Conference on Advanced Video and Signal Based Surveillance, 2010

15. **Speeding Up Semi-Supervised On-line Boosting for Tracking**
    Martin Godec, Helmut Grabner, Christian Leistner, and Horst Bischof
    In Proceedings Workshop of the Austrian Association for Pattern Recognition, 2009

16. **On-line Random Forests** [133]
    Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof
    In Proceedings IEEE On-line Learning for Computer Vision Workshop, 2009

Bibliography

[1] Adam, A., Rivlin, E., and Shimshoni, I. (2006). Robust Fragments-based Tracking using the Integral Histogram. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 798–805.

[2] Agarwal, S., Awan, A., and Roth, D. (2004). Learning to Detect Objects in Images via a Sparse, Part-Based Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490.

[3] Amit, Y. and Trouve, A. (2007). POP: Patchwork of parts models for object recognition. *International Journal of Computer Vision*, pages 267–282.

[4] Arulampalam, S., Maskell, S., and Gordon, N. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Trans. on Signal Processing*, pages 174–188.

[5] Avidan, S. (2001). Support Vector Tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 184–191.

[6] Avidan, S. (2004). Support Vector Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1064–1072.

[7] Avidan, S. (2005). Ensemble Tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 494–501.

[8] Avidan, S. (2006). SpatialBoost: Adding Spatial Reasoning to AdaBoost. In *Proc. European Conference on Computer Vision*, pages 386–396.

[9] Avidan, S. (2007). Ensemble Tracking. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 2, pages 261–271.

[10] Babenko, B., Dollár, P., Tu, Z., and Belongie, S. (2008). Simultaneous Learning and Alignment: Multi-Instance and Multi-Pose Learning. In *Faces in Real-Life Images*.

[11] Babenko, B., Yang, M.-H., and Belongie, S. (2009a). A family of online boosting algorithms. In *Proc. On-line Learning for Computer Vision Workshop*, pages 1346–1353.

[12] Babenko, B., Yang, M.-H., and Belongie, S. (2009b). Visual Tracking with Online Multiple Instance Learning. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–990.

[13] Baker, S., Scharstein, D., Lewis, J. P., Roth, S., Black, M. J., and Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31.

[14] Ballard, D. (1981). Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, pages 714–725.

[15] Beardsley, P. A., Zisserman, A., and Murray, D. W. (1997). Sequential updating of projective and affine structure from motion. *International Journal of Computer Vision*, 23(3):235–259.

[16] Benhimane, S. and Malis, E. (2007). Homography-based 2D Visual Tracking and Servoing. *International Journal of Robotics Research*, pages 661–676.

[17] Bergen, J. R., Anandan, P., Hanna, J., and Hingorani, R. (1992). Hierarchial model-based motion estimation. In *Proc. European Conference on Computer Vision*, pages 237–252.

[18] Bibby, C. and Reid, I. (2008). Robust Real-Time Visual Tracking Using Pixel-Wise Posteriors. In *Proc. European Conference on Computer Vision*, pages 831–844.

[19] Blum, A. and Mitchell, T. (1998). Combining Labeled and Unlabeled Data with Co-training. In *Proc. Conference on Computational Learning Theory*, pages 92–100.

[20] Bolme, D. S., Beveridge, J. R., Draper, B. A., and Lui, Y. M. (2010). Visual object tracking using adaptive correlation filters. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2544–2550.

[21] Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24:123–140.

[22] Breiman, L. (2001). Random Forests. *Machine Learning*, 45:5–32.

[23] Burl, M., Weber, M., and Perona, P. (1998). A probabilistic approach to object recognition using local photometry and global geometry. In *Proc. European Conference on Computer Vision*, pages 628–641.

[24] Cehovin, L., Kristan, M., and Leonardis, A. (2011). An adaptive coupled-layer visual model for robust visual tracking. In *Proc. IEEE International Conference on Computer Vision*, pages 1363–1370.

[25] Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[26] Chockalingam, P., Pradeep, N., and Birchfield, S. T. (2009). Adaptive fragments-based tracking of nonrigid objects using level sets. In *Proc. IEEE International Conference on Computer Vision*, pages 1530–1537.

[27] Cohn, D. A., Ghahramani, Z., and Jordan, M. I. (1996). Active Learning with Statistical Models. *Journal of Artificial Intelligence Research*, pages 129–145.

[28] Collins, R., Liu, Y., and Leordeanu, M. (2005). Online Selection of Discriminative Tracking Features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1631–1643.

[29] Comaniciu, D., Ramesh, V., and Meer, P. (2000). Real-time tracking of non-rigid objects using mean shift. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 142–149.

[30] Crandall, D., Felzenszwalb, P., and Huttenlocher, D. (2005). Spatial Priors for Part-Based Recognition Using Statistical Models. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 10–17.

[31] Cremers, D. and Funka-lea, G. (2006). Dynamical statistical shape priors for level set based tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1262–1273.

[32] Criminisi, A., Shotton, J., and Konukoglu, E. (2011). Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. Technical Report MSR-TR-2011-114, Microsoft.

[33] Crow, F. C. (1984). Summed-Area Tables for Texture Mapping. In *Proc. ACM SIGGRAPH*, volume 18, pages 207–212.

[34] Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893.

[35] Daugman, J. (1980). Two-dimensional spectral analysis of cortical receptive field profiles. *Vision Research*, 20(10):847–856.

[36] Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proc. ACM Conference on Knowledge discovery and data mining*, pages 71–80.

[37] Domingos, P. and Pazzani, M. (1997). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, pages 103–130.

[38] Everingham, M., van Gool, L., Williams, C., Winn, J., and Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge.

[39] Fan, J., Shen, X., and Wu, Y. (2010). Closed-Loop Adaptation for Robust Tracking. In *Proc. European Conference on Computer Vision*, pages 411–424.

[40] Fan, J., Shen, X., and Wu, Y. (2012). Scribble Tracker: A Matting-based Approach for Robust Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1633–1644.

[41] Felzenszwalb, P., Girshick, R., McAllester, D., and Ramanan, D. (2010). Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1627–1645.

[42] Felzenszwalb, P. and Huttenlocher, D. P. (2005). Pictorial Structures for Object Recognition. *International Journal of Computer Vision*, pages 55–79.

[43] Fergus, R., Perona, P., and Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271.

[44] Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, pages 209–230.

[45] Fischler, M. and Elschlager, R. (1973). The representation and matching of pictorial structures. *IEEE Transactions on Computer*, pages 67–92.

[46] Fleet, D. and Weiss, Y. (2005). *Optical Flow Estimation*, pages 239–257. Springer.

[47] Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, 121:256–285.

[48] Freund, Y. and Shapire, R. E. (1996). Experiments with a New Boosting Algorithm. In *Proc. International Conference on Machine Learning*, pages 148–156.

[49] Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, pages 1189–1232.

[50] Gall, J. and Lempitsky, V. (2009). Class-specific Hough forests for object detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1022–1029.

[51] Gall, J., Razavi, N., and van Gool, L. (2010). On-line Adaption of Class-specific Codebooks for Instance Tracking. In *Proc. British Machine Vision Conference*, pages 55.1–55.12.

[52] Geremia, E., Menze, B., Clatz, O., Konukoglu, E., Criminisi, A., and Ayache, N. (2010). Spatial Decision Forests for MS Lesion Segmentation in Multi-Channel MR Images. In *Proc. Medical Image Computing and Computer Assisted Intervention*, pages 111–118.

[53] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely Randomized Trees. *Machine Learning*, pages 3–42.

[54] Gini, C. (1912). *Variability and Mutability (Italian: Variabilita e mutabilita)*. C. Cuppini, Bologna.

[55] Godec, M., Leistner, C., Saffari, A., and Bischof, H. (2010a). On-line Random Naive Bayes for Tracking. In *Proc. International Conference on Pattern Recognition*, pages 3545–3548.

[56] Godec, M., Roth, P. M., and Bischof, H. (2011). Hough-based Tracking of Non-Rigid Objects. In *Proc. IEEE International Conference on Computer Vision*, pages 81–88.

[57] Godec, M., Roth, P. M., and Bischof, H. (2013a). Hough-based tracking of deformable objects. In Criminisi, A. and Shotton, J., editors, *Decision Forests for Computer Vision and Medical Image Analysis*, chapter 11, pages 159–174. Springer.

[58] Godec, M., Roth, P. M., and Bischof, H. (2013b). Hough-based Tracking of Non-Rigid Objects. *Computer Vision and Image Understanding*.

[59] Godec, M., Sternig, S., Roth, P. M., and Bischof, H. (2010b). Context-driven Clustering by Multi-class Classification in an Active Learning Framework. In *Proc. Workshop on Use of Context in Video Processing*, pages 19–24.

[60] Grabner, H. and Bischof, H. (2006). On-line Boosting and Vision. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 260–267.

[61] Grabner, H., Grabner, M., and Bischof, H. (2006a). Real-Time Tracking via On-line Boosting. In *Proc. British Machine Vision Conference*, volume 1, pages 47–56.

[62] Grabner, H., Leistner, C., and Bischof, H. (2008). Semi-supervised On-Line Boosting for Robust Tracking. In *Proc. European Conference on Computer Vision*, pages 234–247.

[63] Grabner, H., Roth, P. M., Grabner, M., and Bischof, H. (2006b). Autonomous Learning of a Robust Background Model for Change Detection. In *Proc. IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 39–46.

[64] Grabner, M., Grabner, H., and Bischof, H. (2007). Learning Features for Tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[65] Greene, W. H. (2003). *Econometric Analysis*. Prentice Hall, Upper Saddle River, NJ, 5. edition.

[66] Grossberg, S. (1987). Competitive Learning: From Interactive Activation to Adaptive Resonance. *Cognitive Science*, pages 23–63.

[67] Grundmann, M., Kwatra, V., Han, M., and Essa, I. (2010). Efficient Hierarchical Graph Based Video Segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2141–2148.

[68] Haar, A. (1910). Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69(3):331–371.

[69] Hager, G. D. and Belhumeur, P. N. (1998). Efficient Region Tracking With Parametric Models of Geometry and Illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:1025–1039.

[70] Hand, D. J. and Yu, K. (2001). Idiot's Bayes: Not So Stupid after All? *International Statistical Review*, pages 385–398.

[71] Heber, M., Godec, M., Ruether, M., Roth, P. M., and Bischof, H. (2013). Segmentation-based tracking by support fusion. *Computer Vision and Image Understanding*, 117(6):573–586.

[72] Hong, X., Chang, H., Shan, S., Chen, X., and Gao, W. (2009). Sigma set: A small second order statistical region descriptor. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1802–1809.

[73] Hyafil, L. and Rivest, R. L. (1976). Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, pages 15–17.

[74] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, pages 79–87.

[75] Jepson, A. D., Fleet, D. J., and El-Maraghi, T. (2001). Robust Online Appearance Models for Visual Tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1296–1311.

[76] Kalal, Z., Matas, J., and Mikolajczyk, K. (2009). Online learning of robust object detectors during unstable tracking. In *Proc. On-line Learning for Computer Vision Workshop*, pages 1417–1424.

[77] Kalal, Z., Matas, J., and Mikolajczyk, K. (2010). P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 49–56.

[78] Kim, T.-K. and Cipolla, R. (2009). MCBoost: Multiple Classifier Boosting for Perceptual Co-clustering of Images and Visual Features. In *Advances in Neural Information Processing Systems*, pages 841–856.

[79] Kluckner, S., Mauthner, T., and Bischof, H. (2009). A Covariance Approximation on Euclidean Space for Visual Tracking. In *Proc. Workshop of the Austrian Association for Pattern Recognition*.

[80] Kotz, S., Johnson, N., and Balakrishnan, N. (2004). *Continuous Multivariate Distributions, Models and Applications*. Continuous Multivariate Distributions. Wiley.

[81] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, pages 49–86.

[82] Kwon, J. and Lee, K. (2009). Tracking of a non-rigid object via patch-based dynamic appearance modeling and adaptive Basin Hopping Monte Carlo sampling. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1208–1215.

[83] Laplace, P. (1812). *Théorie analytique des probabilités*. Courcier.

[84] Laskov, P., Gehl, C., Krüger, S., and Müller, K.-R. (2006). Incremental Support Vector Learning: Analysis, Implementation and Applications. *Journal of Machine Learning Research*, pages 1909–1936.

[85] Lasserre, J. A., Bishop, C. M., and Minka, T. P. (2006). Principled Hybrids of Generative and Discriminative Models. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 87–94.

[86] Leibe, B., Leonardis, A., and Schiele, B. (2004). Combined Object Categorization and Segmentation with an Implicit Shape Model. In *Proc. Workshop on Statistical Learning in Computer Vision*, pages 17–32.

[87] Leibe, B., Leonardis, A., and Schiele, B. (2008). Robust Object Detection with Interleaved Categorization and Segmentation. *International Journal of Computer Vision*, 77:259–289.

[88] Leistner, C., Godec, M., Saffari, A., and Bischof, H. (2010a). Online Multi-View Forests for Tracking. In *Proc. DAGM Symposium*, pages 493–502.

[89] Leistner, C., Godec, M., Schulter, S., Saffari, A., Werlberger, M., and Bischof, H. (2011). Improving Classifiers with Unlabeled Weakly-Related Videos. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2753–2760.

[90] Leistner, C., Saffari, A., and Bischof, H. (2010b). MIForests: Multiple-Instance Learning with Randomized Trees. In *Proc. European Conference on Computer Vision*, pages 29–42.

[91] Leistner, C., Saffari, A., Roth, P. M., and Bischof, H. (2009). On Robustness of Online Boosting – A Competitive Study. In *Proc. On-line Learning for Computer Vision Workshop*, pages 1362–1369.

[92] Lepetit, V. and Fua, P. (2006). Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1465–1479.

[93] Levin, A., Lischinski, D., and Weiss, Y. (2008). A Closed Form Solution to Natural Image Matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 228–242.

[94] Li, M. and Sethi, I. K. (2006). Confidence-Based Active Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1251–1261.

[95] Lienhart, R. and Maydt, J. (2002). An extended set of Haar-like features for rapid object detection. In *Proc. IEEE International Conference on Image Processing*, pages 900–903.

[96] Lim, J., Ross, D., Lin, R., and Yang, M. (2005). Incremental Learning for Visual Tracking. In *Advances in Neural Information Processing Systems*, pages 793–800.

[97] Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, pages 285–318.

[98] Littlestone, N. and Warmuth, M. K. (1994). The weighted majority algorithm. *Information and Computation*, pages 212–261.

[99] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–110.

[100] Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proc. International Joint Conference on Artificial Intelligence*, volume 2, pages 674–679.

[101] Maji, S. and Malik, J. (2009). Object detection using a max-margin Hough transform. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1038–1045.

[102] Matthews, I., Ishikawa, T., and Baker, S. (2004). The Template Update Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:810 – 815.

[103] Mauthner, T., Donoser, M., and Bischof, H. (2008). Robust Tracking of Spatial Related Components. In *Proc. International Conference on Pattern Recognition*, pages 1–4.

[104] Messom, C. H. and Barczak, A. L. C. (2006). Fast and Efficient Rotated Haar-like Features Using Rotated Integral Images. In *Proc. Australian Conference on Robotics and Automation*, pages 1–6.

[105] Montillo, A., Shotton, J., Winn, J. M., Iglesias, J. E., Metaxas, D. N., and Criminisi, A. (2011). Entangled Decision Forests and Their Application for Semantic Segmentation of CT Images. In *Proc. Information Processing in Medical Imaging*, pages 184–196.

[106] Moreno-Noguer, F., Sanfeliu, A., and Samaras, D. (2008). Dependent Multiple Cue Integration for Robust Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 670–685.

[107] Murthy, S. K. (1997). Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2:345–389.

[108] Ng, A. Y. and Jordan, M. I. (2001). On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems*, pages 841–848.

[109] Ojala, T., Pietikäinen, M., and Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:971–987.

[110] Okada, R. (2009). Discriminative generalized Hough transform for object dectection. In *Proc. IEEE International Conference on Computer Vision*, pages 2000–2005.

[111] Opelt, A., Pinz, A., Fussenegger, M., and Auer, P. (2006). Generic Object Recognition with Boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(3):416–431.

[112] Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, pages 169–198.

[113] Oza, N. C. (2001). *Online Ensemble Learning*. PhD thesis, University of California, Berkeley.

[114] Oza, N. C. and Russell, S. (2001). Online bagging and boosting. In *Proc. Artificial Intelligence and Statistics*, volume 3, pages 2340–2345.

[115] Özuysal, M., Calonder, M., Lepetit, V., and Fua, P. (2010). Fast Keypoint Recognition Using Random Ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 448–461.

[116] Özuysal, M., Fua, P., and Lepetit, V. (2007). Fast Keypoint Recognition in Ten Lines of Code. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[117] Pakkanen, J., Iivarinen, J., and Oja, E. (2006). The Evolving Tree-Analysis and Applications. *IEEE Transactions on Neural Networks*, 17:591–603.

[118] Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1345–1359.

[119] Papageorgiou, C., Oren, M., and Poggio, T. (1998). A General Framework for Object Detection. In *Proc. IEEE International Conference on Computer Vision*, pages 555–562.

[120] Park, J.-H. and Choi, Y.-K. (1996). On-line Learning for Active Pattern Recognition. *IEEE Signal Processing Letters*, pages 301–303.

[121] Pham, M.-T., Gao, Y., Hoang, V. D., and Cham, T.-J. (2010). Fast polygonal integration and its application in extending haar-like features to improve object detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 942–949.

[122] Porikli, F. (2005). Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 829–836.

[123] Porikli, F., Tuzel, O., and Meer, P. (2006). Covariance Tracking using Model Update Based on Lie Algebra. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 728–735.

[124] Prinzie, A. and Van den Poel, D. (2007). Random Multiclass Classification: Generalizing Random Forests to Random MNL and Random NB. In *Database and Expert Systems Applications*, pages 349–358.

[125] R., V. and Y., D. (2002). A perspective view and survey of meta-learning. *Journal of Artificial Intelligence Review*, pages 77–95.

[126] Razavi, N., Gall, J., and van Gool, L. (2010). Backprojection Revisited: Scalable Multi-view Object Detection and Similarity Metrics for Detections. In *Proc. European Conference on Computer Vision*, pages 620–633.

[127] Ren, X. and Malik, J. (2007). Tracking as Repeated Figure/Ground Segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[128] Ross, D., Lim, J., Lin, R.-S., and Yang, M.-H. (2007). Incremental Learning for Robust Visual Tracking. *International Journal of Computer Vision*, pages 125–141.

[129] Roth, P. M., Sternig, S., Grabner, H., and Bischof, H. (2009). Classifier Grids for Robust Adaptive Object Detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2727–2734.

[130] Rother, C., Kolmogorov, V., and Blake, A. (2004). GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics*, pages 309–314.

[131] Saffari, A., Godec, M., Pock, T., Leistner, C., and Bischof, H. (2010a). Online Multi-Class LPBoost. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 3570–3577.

[132] Saffari, A., Leistner, C., Godec, M., and Bischof, H. (2010b). Robust Multi-View Boosting with Priors. In *Proc. European Conference on Computer Vision*, pages 776–789.

[133] Saffari, A., Leistner, C., Santner, J., Godec, M., and Bischof, H. (2009). On-line Random Forests. In *Proc. On-line Learning for Computer Vision Workshop*, pages 1393–1400.

[134] Santner, J., Leistner, C., Saffari, A., Pock, T., and Bischof, H. (2010). PROST Parallel Robust Online Simple Tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 723–730.

[135] Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. (1998). Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. *The Annals of Statistics*, 26(5):1651–1686.

[136] Schindler, G. and Dellaert, F. (2005). A rao-blackwellized parts-constellation tracker. In *Proc. Workshop on Dynamic Vision*, pages 178–189.

[137] Schulter, S., Leistner, C., Roth, P. M., van Gool, L., and Bischof, H. (2011). On-line Hough Forests. In *Proc. British Machine Vision Conference*, pages 128.1–128.11.

[138] Sevilla-Lara, L. and Learned-Miller, E. (2012). Distribution Fields for Tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1910–1917.

[139] Shahed Nejhum, S., Ho, J., and Yang, M.-H. (2008). Visual tracking with histograms and articulating blocks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[140] Shannon, C. E. (1948). A mathematical theory of communication. *Bell Systems Technical Journal*, pages 379–423.

[141] Shi, X., Zhang, X., Liu, Y., Hu, W., and Ling, H. (2011). Multi-cue based multi-target tracking using online random forests. In *Proc. IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 1185–1188.

[142] Shi, Y. and Karl, W. (2005). Real-time tracking using level sets. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 34–41.

[143] Shotton, J., Johnson, M., and Cipolla, R. (2008). Semantic texton forests for image categorization and segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[144] Sigal, L., Bhatia, S., Roth, S., Black, M. J., and Isard, M. (2004). Tracking loose-limbed people. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 421–428.

[145] Sternig, S., Godec, M., Roth, P. M., and Bischof, H. (2010). TransientBoost: Online Boosting with Transient Data. In *Proc. On-line Learning for Computer Vision Workshop*, pages 22–27.

[146] Torralba, A., Murphy, K. P., and Freeman, W. T. (2007). Sharing Visual Features for Multiclass and Multiview Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:854–869.

[147] Toussaint, G. (1971). Note on optimal selection of independent binary-valued features for pattern recognition. *IEEE Transactions on Information Theory*, 17:618–618.

[148] Tsai, D., Flagg, M., and Rehg, J. M. (2010). Motion Coherent Tracking with Multi-label MRF optimization. In *Proc. British Machine Vision Conference*, pages 190–202.

[149] Tu, Z. (2005). Probabilistic boosting-tree: learning discriminative models for classification, recognition, and clustering. In *Proc. IEEE International Conference on Computer Vision*, volume 2, pages 1589–1596.

[150] Tu, Z. (2007). Learning Generative Models via Discriminative Approaches. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[151] Tuzel, O., Porikli, F., and Meer, P. (2006). Region Covariance: A Fast Descriptor for Detection and Classification. In *Proc. European Conference on Computer Vision*, pages 589–600.

[152] Villamizar, M., Moreno-Noguer, F., Andrade-Cetto, J., and Sanfeliu, A. (2010). Efficient rotation invariant object detection using boosted Random Ferns. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1038–1045.

[153] Viola, P. and Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 511–518.

[154] Viola, P. and Jones, M. (2002). Robust Real-time Object Detection. *International Journal of Computer Vision*, pages 137–154.

[155] Vovk, V. G. (1990). Aggregating strategies. In *In Proc. Workshop on Computational Learning Theory*, pages 371–386.

[156] Weber, M., Welling, M., and Perona, P. (2000). Towards automatic discovery of object categories. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 101–108.

[157] Wei, Y. and Tao, L. (2010). Efficient histogram-based sliding window. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 3003–3010.

[158] Woodley, T., Stenger, B., and Cipolla, R. (2007). Tracking Using Online Feature Selection and a Local Generative Model. In *Proc. British Machine Vision Conference*, pages 86.1–86.10.

[159] Wu, B. and Nevatia, R. (2007a). Cluster Boosted Tree Classifier for Multi-View, Multi-Pose Object Detection. In *Proc. IEEE International Conference on Computer Vision*, pages 1–8.

[160] Wu, B. and Nevatia, R. (2007b). Improving Part based Object Detection by Unsupervised, Online Boosting. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[161] Wu, B. and Nevatia, R. (2007c). Simultaneous Object Detection and Segmentation by Boosting Local Shape Feature based Classifier. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[162] Yan, R., Yang, J., and Hauptmann, A. (2003). Automatically Labeling Video Data Using Multi-class Active Learning. In *Proc. IEEE International Conference on Computer Vision*, volume 1, pages 516–523.

[163] Yao, A., Gall, J., and van Gool, L. (2010). A Hough transform-based voting framework for action recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2061–2068.

[164] Yin, X.-C., Liu, C.-P., and Han, Z. (2005). Feature Combination using Boosting. *Pattern Recognition*, 26:2195–2205.

[165] Yin, Z. and Collins, R. (2009). Shape Constrained Figure-Ground Segmentation and Tracking. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 731–738.

[166] Zeisl, B., Leistner, C., Saffari, A., and Bischof, H. (2010). On-line Semi-supervised Multiple-Instance Boosting. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1879–1886.

[167] Zliobaite, I. (2009). Learning under Concept Drift: an Overview. Technical report, Vilnius University, Faculty of Mathematics and Informatic.