Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

Efficient Hough Forests for Multi-Camera, Multi-Instance Object Detection and Tracking

Georg Poier

Graz, Austria, December 2013

Thesis supervisor

Prof. Dr. Horst Bischof

Advisors

DI Samuel Schulter

Dr. Peter Michael Roth

To Mama and Papa

Irrend lernt man

Johann Wolfgang von Goethe

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

Visual object tracking represents a crucial task for many computer vision applications. Moreover, continued research interest in recent years shows that there are still open issues for tracking algorithms. Especially, when tracking multiple instances of the same class, occlusions are likely and can often only be resolved by observing the scene from several viewpoints. The multiplied amount of data coming from the different viewpoints demands methods of low computational complexity. Hence, simple approaches, *e.g.*, based on background subtraction, are regularly applied to locate the objects of interest. Nevertheless, due to their simplicity, these methods often suffer from severe problems like mistaken instances or ghost detections. To overcome these problems, stronger, classifier-based models can be used. However, transferring complex models straightforwardly to multi-camera systems is not applicable due to the runtime requirements.

This obvious discrepancy between the need for stronger models and short response times is addressed in this work. Our goal is the utilization of a more complex model. This is permitted by a preliminary investigation of the efficiency of such a model, revealing that the computational cost can be drastically reduced without sacrificing detection accuracy. To this end, we focus on Random Forests as an object model classifier in a tracking-by-detection setup. Random Forests received strong interest recently, which is mainly based on the fact that they allow fast predictions – regardless of the dimensionality of the feature space. A powerful and versatile extension for object detection are the recently proposed Hough Forests. However, their application is hampered in cases where a huge amount of data has to be processed, since for their original formulation runtime is correlated to the number of training samples. In line with this, we thoroughly analyze the efficiency of Hough Forest based object detection in order to obtain a fast detector applicable to online learning during tracking. Runtime critical parts are identified and investigated, which reveals several insights yielding a drastic improvement of the overall runtime. Moreover, scalability is induced by removing the correlation between the runtime and the amount of training data. Most importantly, the achieved runtime reduction does not imply a significant loss in accuracy. In fact, the proposed method scores within a few percent of the baseline, while being one to two orders of magnitude faster.

The gathered insights can be straightforwardly applied to tracking in a tracking-by-detection manner. To this end, we perform tracking from single as well as multiple cameras and test our method on several standard evaluation datasets. The qualitative and quantitative results show that the system is able to perform comparable or even better than state-of-the-art methods in terms of accuracy. Despite that, the runtime of our method is reduced significantly compared to the related work.

Keywords. object detection, object tracking, Hough forests, efficient methods, multiple cameras.

Kurzfassung

Die visuelle Objektverfolgung bildet die Grundlage für eine Vielzahl von Anwendungen in der Bildverarbeitung. Darüberhinaus zeigt das andauernde Forschungsinteresse der letzten Jahre, dass es dabei noch immer ungelöste Aspekte gibt. Speziell bei der Verfolgung von mehreren Objekten der selben Klasse von nur einem Betrachtungspunkt kommt es unweigerlich zu Verdeckungen, welche oft nur durch die Betrachtung der Szene von mehreren Blickwinkeln aufgelöst werden können. Die Verarbeitung der vervielfachten Datenmenge, die durch die Betrachtung aus mehreren Blickwinkeln entsteht, verlangt jedoch nach Methoden mit geringem Rechenaufwand. Aus diesem Grund werden üblicherweise sehr einfache Methoden, z.B. basierend auf "background subtraction" angewandt, um die gesuchten Objekte zu lokalisieren. Bei diesen Methoden kann es jedoch – bedingt durch ihre Einfachheit – zu Verwechslungen der einzelnen Objektinstanzen oder auch zur Detektion nicht vorhandener Objekte kommen. Um derartige Probleme zu bewältigen, können stärkere Modelle basierend auf Klassifikatoren herangezogen werden. Jedoch ist eine direkte Übertragung solcher Modelle auf Systeme mit mehreren Kameras – aufgrund der Laufzeitanforderungen – nicht möglich.

Diese offensichtliche Diskrepanz zwischen der Notwendigkeit für stärkere Modelle und kurzer Reaktionszeiten wird in der vorliegenden Arbeit adressiert. Das gesetzte Ziel ist ein stärkeres Modell anwenden zu können. Ermöglicht wird das durch eine vorangehende Untersuchung der Effizienz eines solchen Modells, welche aufzeigt, dass der Rechenaufwand – ohne Genauigkeitsverlust – drastisch reduziert werden kann. Zu diesem Zweck wird ein Random Forest als Klassifikator im Rahmen einer sogenannten "Verfolgung durch Detektion" Vorgehensweise herangezogen. Random Forests waren in den letzten Jahren von großem Interesse im Bereich der Bildverarbeitung, da sie – unabhängig von der Dimensionalität des Feature-Raums – in der Lage sind, äußerst schnell Aussagen aus den Daten abzuleiten. Eine leistungsstarke und gleichzeitig vielseitige Erweiterung der Random Forests stellen die unlängst vorgestellten Hough Forests dar. In Situationen, in welchen große Datenmengen verarbeitet werden müssen, ist ihre Verwendung aufgrund der Korrelation zwischen Laufzeit und Anzahl der Daten jedoch nur bedingt möglich. In diesem Zusammenhang wird die Laufzeit der Objektdetektion mittels Hough Forests analysiert, um schließlich einen schnellen Objektdetektor zu erhalten, welcher sich eignet, um ihn "on-line" – während der Objektverfolgung – weiter zu trainieren. Dabei werden Laufzeit-kritische Teile des Prozesses identifiziert und untersucht, um eine drastische Verbesserung der Gesamtlaufzeit herbeizuführen. Unter anderem wird der Zusammenhang zwischen Laufzeit und Anzahl der Trainingsdaten beseitigt. Entscheidend dabei ist vor allem, dass die Laufzeitverbesserung keine signifikante Verschlechterung der Genauigkeit mit sich bringt. Die Ergebnisse zeigen, dass die Genauigkeit der vorgeschlagenen Methode im Bereich der Ausgangsmethode liegt, jedoch um ein bis zwei Größenordnungen schneller ist.

Die gewonnenen Erkenntnisse können anschließend direkt auf die Objektverfolgung angewandt werden. In diesem Zusammenhang führen wir Experimente auf Basis einer als auch mehrerer Kameras durch und evaluieren unsere Methode anhand einer Anzahl von Standard-Datensätzen. Die qualitativen sowie quantitativen Ergebnisse zeigen, dass die Genauigkeit des Systems mit dem Stand der Technik vergleichbar oder sogar besser ist. Die Laufzeit jedoch konnte deutlich reduziert werden.

Schlüsselwörter. Objektdetektion, Objektverfolgung, Hough forests, effiziente Methoden, mehrere Kameras.

Acknowledgments

This work would not have been possible without my supervisors. Thank you for patiently answering my questions and being open for an inspiring discussion at any time. For the same reasons, I want to thank Horst, Tom, Martin and especially Paul. Thank you for giving me an impression what it means to “actually do science”.

Furthermore, I want to thank my colleagues at Siemens for helping me develop in many different ways and – even though some people might not believe – for making up a “funny” place to work.

This work also manifests at least one sad event: The end of my studies. My studies were a good time mostly because of Viky, my friends, my colleagues and the support from my family, in particular from my parents. Thank you for giving me the chance to do my studies. As always, we better know to cherish things we had after we lost them.

I am sorry for all the people I forgot to thank now, and who will come into my mind tomorrow, only. Lastly, I have to thank you, my dear reader, for actually reading (parts of) this thesis.

You won’t regret.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Visual object tracking builds the basis for many computer vision systems. Examples range from surveillance [50, 60] over sports [55, 90] to industrial domains [74]. In line with this, various successful approaches have been developed [38, 42, 46, 84, 92]. Nevertheless, in situations where multiple interacting object instances are present in the scene, occlusions are likely and generally raise problems.

Several approaches have been proposed in order to resolve occlusions from single images. While surprisingly good results were achieved by conducting well elaborated reasoning relying on part based approaches (see *e.g.*, [6, 88]), situations arise where these methods fail. For example, if instances are fully occluding each other, it is virtually impossible to distinguish between them from a single image, only. To overcome this, recent approaches analyze the tracks of individual instances over time [12, 22]. This can help, *e.g.*, if object instances are passing by each other with approximately constant velocity. However, cases where the occlusion appears while people are stopping and interacting still introduce problems. These cases can neither be resolved in the sequel, since instance separation is solely based on the tracks, whereas visual appearance is neglected.

A natural solution to the aforementioned problems is to exploit the additional information provided by multiple cameras [24, 55, 72]. Such a solution became feasible due to the increasing amount of cameras mounted, *e.g.*, for security reasons or sport broadcasts. A drawback of this solution, however, is the multiplied amount of data which needs to be processed. Visual object tracking, being a primal task for many computer vision applications, requires a significant amount of data to be processed already in the monocular case. Moreover, this has to be conducted as fast as possible. Typically: The faster, the better, because the less time spent on this primal task, the more sophisticated the subsequent processing might be. Hence, efficiency is an essential requirement for tracking systems, but the increased amount of data coming from multiple cameras makes it imperative to use efficient algorithms in multi-camera tracking systems. Thus, the employed methods are typically quite simple. They often base on background subtraction or simple color models in order to locate foreground objects [24, 47, 67]. However, due to the simplicity of the models, problems like mistaken instances or ghost detections are common. To overcome these problems, more complex object models have to be applied. Learned classifiers are able to provide such accurate models [72, 81]. Nevertheless, they are usually too complex to fulfill the runtime requirements.

The described discrepancy between the need for accurate models and the runtime requirements indeed hampers applications. In the present work, this discrepancy is addressed by focusing on the computational complexity of object tracking-by-detection. Specifically, an object detection model based on Random Forests is investigated in order to apply it to single- as well as multi-camera tracking. To this end, Random Forests are employed not least because of their efficiency.

1.1 Efficiency

Object tracking is by far not the only task within the field of computer vision requiring short response times. For example, applications which demand real-time processing are ranging from gaming [78] over navigation for autonomous cars [85] to surveillance systems [60]. Moreover, since many computer vision applications also have to cope with a huge amount of data, delivering results in acceptable time requires large computational power as well as highly efficient methods [49, 80]. Hence, efficient algorithms have always been of major interest in computer vision research.

Considering classifiers, recently, Random Forests [10] received strong interest in the field. While there are several reasons for their popularity, one of their major advantages is the ability of predictions with low computational complexity. Furthermore, they are inherently multi-class, robust against label noise and have been demonstrated to give competitive results to other state-of-the-art approaches [10, 14, 53]. A powerful and versatile extension are Hough forests [32, 63], which are successfully applied to various tasks including object detection [32, 63], tracking [35, 75], and pose estimation [26, 37, 83]. Hough Forests are Random Forests integrating part based classification and regression into a single model. That is, given a small image patch, their goal is to classify if it originates from an object as well as to predict the possible object location and scale.

Originally, Random Forests are formulated in an off-line manner, requiring access to the entire training data at once. Thus, they cannot cope with situations where the data arrives sequentially during test time, as it is the case in tracking applications. To overcome this limitation, on-line variants have been proposed for both, the original Random Forests [73], as well as Hough Forests [75]. These approaches provide the ability of building a forest model on-line, without having access to any training sample in advance.

By building the model on-line, one is able to exploit the possibly huge amount of data arriving during runtime, *e.g.*, from a video stream. Random Forests are capable of efficiently handling such a huge number of data samples [34]. This is based on the fact that the test time is independent from the number of training samples. However, this major advantage is lost for the original formulation of Hough Forests [32]. Their test time is strictly correlated with the number of training samples, which is a severe flaw, since the amount of training data is a crucial parameter for detection accuracy [34]. Hence, applying Hough Forests straightforward to on-line learning – where the data is constantly increasing over time – would be impractical. Fortunately, methods to circumvent this dead end exist.

In this work, we build on the ideas of a recently proposed approach [37] used to avoid data dependency of the runtime. The method was proposed for off-line trained forests, however, on-line updates appear to be problematic. To this end, we introduce a new method which promises better flexibility during on-line learning. To investigate the differences between the approaches in terms of accuracy and efficiency, we first compare both for the task of object detection before employing the winning approach in a tracking-by-detection system. The results demonstrate that our novel method does not only provide constant runtime[*], but shows even an increase in accuracy compared to the aforementioned approach.

As there are other aspects of the Hough Forest model hampering efficiency, by-passing the data dependency of the runtime doesn't directly imply short response times. Moreover, the primal nature of the tracking task and, above all, the increased amount of data to be processed in a multi-camera system motivates an in-depth study of *all* runtime critical

[*]With *constant runtime* we refer to the independence of the amount of input data.

parts of Hough Forest based tracking-by-detection. To this end, we provide a theoretical as well as empirical analysis of the overall runtime. This reveals the critical parts of the process, starting from the classifier complexity, over the underlying data complexity, through to the prediction task. Subsequently, we analyze each part, which brings up several interesting insights. Based on those we are able to build a significantly faster detector for only a slight loss in accuracy. In fact, the obtained results are within a few percent of the baseline [35] while the runtime is reduced by one to two orders of magnitude.

1.2 Tracking

The gathered insights can be straightforwardly applied to tracking in a tracking-by-detection manner. First, we conduct several experiments for tracking single objects from a single camera. For this task we show that our method outperforms the state-of-the-art on several standard evaluation sequences. Moreover – in conjunction with the previously discussed performance improvements – our method is flexible in a way that it is easily adjustable to fulfill the task in real time, while the accuracy is kept within the range of the *state-of-the-art* methods [4, 38, 73].

Second, we apply our insights to multi-camera tracking. Recently, several approaches have been proposed in this field. Nevertheless, many use simple models hampering their applicability [24, 47]. Some are also limited to specific settings and requirements [24, 41, 67]. Others use individual classifiers for each view [72, 81], and thus, neglect to comprehensively exploit the information given by the different views of the same objects. Consequently, it is hard to learn a strong model for each instance, and related works restrain from doing so. In contrast, we propose to adapt a generic pre-trained object detector on-line to the specific instances, instead of the views. Hence, we employ a single comprehensive classifier, which is able to discriminate between the instances, without requiring costly post-processing steps.

Overall, the approach is able to robustly track multiple instances without confusing them. This is confirmed by experimental results, showing that the accuracy of our system is on par with related approaches, despite the reduced complexity of the detector, and thus, significantly faster processing.

1.3 Outline

The work is structured as follows: In Chapter 2 we motivate and review the basic learning algorithms used throughout this work, *i.e.*, Random Forests, as well as Hough Forests. Chapter 3 starts with a theoretical analysis of the Hough Forests detection process and subsequently discusses how to reduce the runtime cost of each critical part, without significantly affecting accuracy. In Chapter 4 the gathered insights are applied to tracking. Moreover, we will first prove the concepts for single camera tracking, and then move on to tracking multiple instances in multiple camera views. Finally, Chapter 5 concludes the work by summarizing the main insights, as well as pointing out open issues.

Chapter 2

Preliminaries

Contents

Several studies have been conducted to compare the performance of different learning methods on a wide range of problems. Prominent studies [48, 51] performed comprehensive evaluations but could not include newer methods, since they date back more than 15 years. In fact, most successful methods for object detection [8, 23, 35] tend to employ methods, like *Support Vector Machines* (SVMs) [17, 86], *AdaBoost* [30, 31] or *Random Forests* [1, 10], which have been proposed after completion of these studies. A newer study [14] included these methods and found that ensembles of trees usually outperform other prominent supervised learning methods, like SVMs, neural nets [58] or k-nearest neighbors (k-NN) [28]. Overall, the best performance was obtained using boosted decision trees, and Random Forests. However, it has also been shown that the performance of boosting is less robust across different problems and deteriorates in special cases. This confirms what has already been shown earlier by other authors, where the problem cases for AdaBoost were attributed to its sensitivity to outliers and noise, as well as to over-fitting [7, 69]. Furthermore, in a more recent study, Caruana *et al.* [13] found that the performance of boosting decreases with increasing dimensionality, whereas Random Forests robustly provide exceptional results across problems of varying dimensionality. Consequently, in this study [13], best overall results were obtained using Random Forests.

Object detection and tracking are commonly formulated as tasks of high dimensionality [23, 33, 38]. Furthermore, in the on-line case, *i.e.*, during tracking, a significant amount of noise must be handled and for the task of multi-instance tracking the instances represent individual classes (multi-class problem). Although, extensions to boosting handling these cases were proposed [31, 40], recently, tracking approaches based on Random Forests have shown to outperform the approaches based on boosting [39, 73]. This may be attributed to the fact that Random Forests scale efficiently to high dimensions [13], are less susceptible to noise [7, 53] and that they are inherently multi-class [10, 71]. Furthermore, Random Forests are easy to parallelize on modern hardware and able to efficiently handle the huge amount of data arriving sequentially from a video stream [13, 34, 77]. These are another two important advantages for the task at hand, *i.e.*, object detection and tracking. Thus, we decide to base this work on Random Forests.

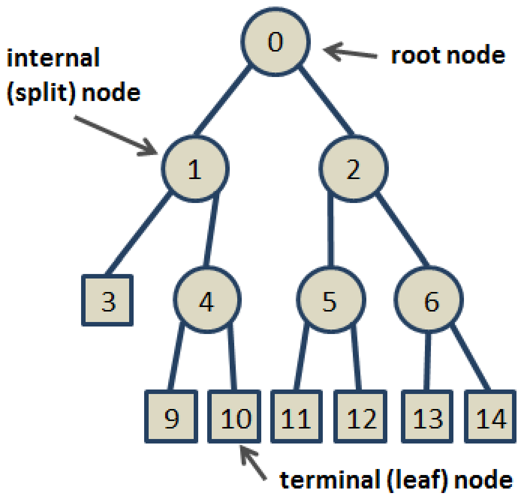
In this chapter we will now introduce some preliminaries for the chosen learning framework. To start with, in Sec. 2.1 we will shortly review the concept of Random Forests [1, 10, 18] as it is necessary for our task. This is followed by their Hough Forest [32] extension for object detection in Sec. 2.2.

2.1 Random Forests

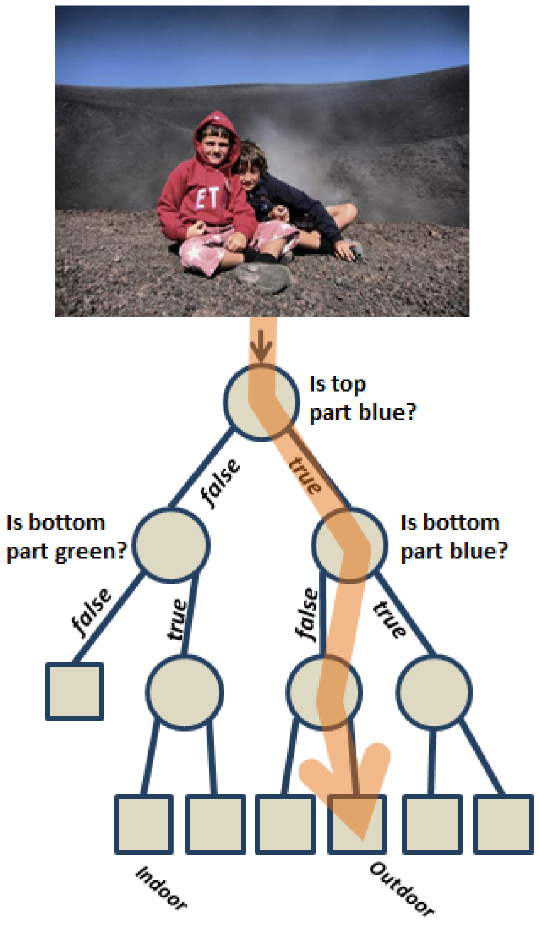
A Random Forest is an ensemble of T decision trees, which are hierarchically organized graphs. A graph itself is made up from a collection of nodes and edges, and does not contain any loops. More specifically, the nodes of a decision tree are divided into internal (or split) nodes and terminal (or leaf) nodes [11, 18] (see Fig. 2.1).

Provided with a data sample $\mathbf{x} \in \mathcal{X} = \mathbb{R}^D$ a decision tree consecutively applies a number of tests. More specifically, starting with the root node, at each internal node a decision is made to which of its child-nodes the sample is sent. The decision itself is based on any property of the input data sample. Thus, the decision of a node is based on its individual subset of properties and the property on which the next decision is based depends on the previous decision. This process is repeated until a leaf node, *i.e.*, a node without any child nodes, is reached. A leaf contains a predictor model, which associates an output $y \in \mathcal{Y}$ to the given data sample. This could be a discrete value representing a class-label, *i.e.*, $\mathcal{Y} = \{1, \ldots, C\}^*$, a distribution representing probabilities, *i.e.*, $p(y|\mathbf{x})$, or a continuous value, *i.e.*, $\mathcal{Y} = \mathbb{R}$. The output of the whole forest is subsequently combined from the outputs of the individual trees. As described here, a Random Forest solves either a classification or a regression problem [10, 11]. However, for the sake of simplicity, in this section we will review the concept of Random Forests only for classification.

$^*C$ would be the number of classes in this case.



(a)



(b)

Figure 2.1: Illustration of a decision tree: (a) A tree is a graph without loops, containing edges and nodes. Circles denote internal nodes, whereas squares represent leaf nodes. (b) A decision tree is a tree where each internal node applies a split to the incoming data. Each leaf stores a prediction. The example tree illustrates how such a decision tree may be used to decide whether an image shows an indoor or outdoor scene. The sketches were taken from [18].

2.1.1 Training

During training, one aims at separating the labeled training set $\mathcal{L} = \left\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{|\mathcal{L}|}, y_{|\mathcal{L}|})\right\} \subseteq \mathcal{X} \times \mathcal{Y}$ into sufficiently small partitions, representing clusters in the D-dimensional input space[†]. For example, in computer vision, this may correspond to the goal of finding a separation into subsets of images of

[†]Throughout this work $|\mathcal{S}|$ denotes the number of elements in any set $\mathcal{S}$.

the same class, exhibiting a similar appearance. Hence, the main task of the training phase is to find the parameters for the splitting tests at the internal nodes of each tree so that the resulting clustering is useful for inference.

A splitting test evaluates one or more features and decides to which of the child nodes a data sample is sent. In the sequel, we will consider two commonly used test types. The first uses a single (*single-value-test*), the second, two randomly selected features (*two-value-test*) in order to make its decision. Note that more complex splitting tests exist (see *e.g.*, [18, 76]). Nevertheless, for the reason of efficiency – as will be discussed in more detail in Sec. 3.6.2 – we use only these two.

Furthermore, in this work we will only consider binary decision trees, *i.e.*, a split node has always two child nodes. As a consequence our test functions return either zero or one. Note, while also n-ary trees were successfully applied to computer vision tasks (see *e.g.*, [45, 54]), a recent thorough evaluation did not report significant accuracy differences [18].

A single-value-test function $\phi^{(1)}$ compares a single feature value $\mathbf{x}(d)$, where $d \in \{1, \ldots, D\}$ indicates the feature dimension, against some threshold τ:

$$\phi^{(1)}(\mathbf{x}, \Theta^{(1)}) = \begin{cases} 0 & \text{if } \mathbf{x}(d) < \tau \\ 1 & \text{otherwise,} \end{cases} \tag{2.1}$$

where $\Theta^{(1)} = (d, \tau)$ specifies the parameters for a single-value-split. Hence, the sample is sent to the left child node if $\phi^{(1)}(\mathbf{x}, \Theta^{(1)})$ returns zero, and to the right child node otherwise. A two-value-test denotes a comparison between two features $d_1, d_2 \in \{1, \ldots, D\}$:

$$\phi^{(2)}(\mathbf{x}, \Theta^{(2)}) = \begin{cases} 0 & \text{if } \mathbf{x}(d_1) < \mathbf{x}(d_2) + \tau \\ 1 & \text{otherwise,} \end{cases} \tag{2.2}$$

where $\Theta^{(2)} = (d_1, d_2, \tau)$ again denotes the corresponding parameters.

During training, a splitting test, yielding a good separation of the data, shall be found. Thus, the quality of the separation needs to be measured based on an objective function. For classification tasks we aim at minimization of class uncertainty at each split node. Therefore, usually the entropy or the Gini index is employed as an objective [10]. In our case, we build on the entropy:

$$H(\mathcal{L}) = -\sum_{y=1}^{C} p(y|\mathcal{L}) \ln p(y|\mathcal{L}), \tag{2.3}$$

where C is the number of classes, and $p(y|\mathcal{L})$ specifies the empirical class probability for class y, estimated from the dataset $\mathcal{L}$.

At each node P a pool of K possible splitting tests, with randomly sampled parameters $\Theta_k, k = \{1, \dots, K\}$, is generated and the best test $\phi(\mathbf{x}, \Theta^*)$, minimizing the objective function, is selected

$$\underset{k}{\operatorname{argmin}}\ U_{\mathcal{Y}}(\mathcal{L}_P, \phi(\mathbf{x}, \Theta_k)), \tag{2.4}$$

where

$$U_{\mathcal{Y}}(\mathcal{L}, \phi) = \frac{|\mathcal{L}_L|}{|\mathcal{L}_L| + |\mathcal{L}_R|} H(\mathcal{L}_L) + \frac{|\mathcal{L}_R|}{|\mathcal{L}_L| + |\mathcal{L}_R|} H(\mathcal{L}_R). \tag{2.5}$$

Here, $\mathcal{L}_L$ and $\mathcal{L}_R$ are the sets of data samples which are sent to the left and the right child node, when applying a test function ϕ to each of the samples $\mathbf{x} \in \mathcal{L}$, *i.e.*, $\mathcal{L}_L = \big(\mathbf{x}|\phi = 0\big)$, and $\mathcal{L}_R = \big(\mathbf{x}|\phi = 1\big)$.

Construction of a tree starts at the root node by splitting all available training samples according to the test function. This process is repeated at each child node by using only the data samples sent to it. The process stops if either the maximum depth is reached, there are less than a minimum number of samples remaining, or if all remaining samples belong to the same class. Finally, the label statistics, which are needed for prediction, are collected at the leafs.

Each individual tree in the forest is constructed in this way. Since, the splitting tests are randomly selected, each tree represents a different partition of the same training data. This means that throughout the whole forest each training sample is assigned to a diverse set of clusters, *i.e.*, leaf nodes.

2.1.2 Evaluation

When classifying a previously unseen data sample, it is passed down through each tree in the forest by evaluating the learned splitting tests. Arriving at the leafs, the label statistics gathered after training are used for prediction. Thereby, the posterior distribution $p_t(y|\mathbf{x})$ of tree t, $t \in \{1, \dots, T\}$, is estimated by the proportion of training samples per class label reaching the leaf during training. Alternatively, we may obtain a single class label by employing a Maximum A-Posteriori (MAP) estimate [18]. However, usually, we want to keep the distribution, since it provides a measure of confidence on the level of the whole forest, for successive steps. Subsequently, the prediction of the whole forest is computed by simply averaging the posteriors of the individual trees:

$$p(y|\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} p_t(y|\mathbf{x}). \tag{2.6}$$

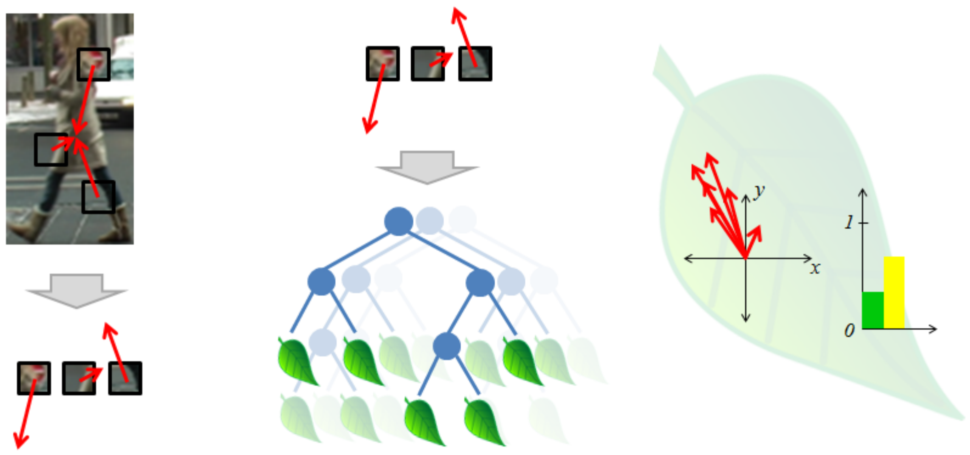
Again, a prediction for a class label y^* label may be retrieved by a MAP estimate:

$$y^* = \operatorname*{argmax}_y p(y|\mathbf{x}). \tag{2.7}$$

2.2 Hough Forests

Hough Forests combine the ideas of Implicit Shape Models (ISMs) and Random Forests. The ISM introduced by Leibe *et al.* [52] is a method for object detection based on the Generalized Hough Transform [5]. Thereby, votes for specific object configurations are cast based on the local appearance of an image. Each of those votes is cast for a hypothesis within the space of all possible object configurations, termed Hough space. Hence, the Hough space may be a multi-dimensional space, where the individual dimensions represent parameters of the object configuration like position, scale, or pose. During training of an ISMs the patches around detected interest points in the training images are clustered by using an agglomerative clustering method. Thus, an ISM may be seen as a generative codebook of local image appearances, where the individual clusters are often referred to as codebook entries. Together with each codebook entry the spatial distribution specifying where the respective image appearance may be found on the object (*i.e.*, the parameters of the object configuration), as well as corresponding foreground masks are stored. At the detection stage the features at interest points are matched to the codebook entries, casting votes into the Hough space. Subsequently, rough segmentations of the found hypotheses are obtained using the foreground masks associated to the features supporting the hypotheses. Based on these segmentations the hypotheses are finally verified in order to decrease the number of false positives, while still keeping the correct detections.

Gall and Lempitsky [32] combined the flexibility of the ISM with the ability of Random Forests to efficiently match samples to the corresponding leafs, *i.e.*, codebook entries. Hence, a Random Forest is utilized to build up the codebook. Additionally, Hough Forests allow optimizing the spatial distribution and learning a discriminative codebook. At test time, the efficiency of the forest framework permits dense sampling of local image patches, yielding an additional gain in accuracy. In the following, we will review the method introduced in [32, 35].



(a) (b) (c)

Figure 2.2: Illustration of the characteristics of a Hough Forest for object detection: (a) Image patches together with offset vectors specifying the relation to the object center are extracted from each training image. (b) The patches and offsets are used to construct the trees of the forest. (c) The statistics at the leafs of the trees are finally estimated from the proportion of samples per class and the corresponding offset vectors arrived at the leaf.

From the Random Forest point of view, the main extension is that the predictor model in the leafs includes a number of offset vectors $\mathbf{o} \in \mathcal{O}$ representing the relation of the observed image patch to a specified position on the object, *e.g.*, the object center. Hence, the prediction of a leaf node l is based on the class probability $p(y|\mathcal{L}_l)$, estimated from the ratio of samples per class y arrived at the leaf during training, and the set of offset vectors $\mathcal{O}_l^{(y)}$ corresponding to each class. These offsets are incorporated in the training process by a specifically designed objective function. In this way, the variance of the offsets at the leafs is enforced to be rather small. For detection, the offsets are used to cast votes in a Hough space, which – in our case – encodes the hypotheses for object positions in scale-space and class, respectively. Hence, Hough Forests are combining classification and regression. They do not only determine a class label, but also the position of the possible object center. See Fig. 2.2 for an illustration of the specifics of a Hough Forest.

In terms of object detection with Hough Forests, a data sample $\mathbf{x}$ represents the appearance of a small image patch. In [32] 16×16 pixel sized patches were utilized, where the objects were scaled to a unit size s_u, such that the longest spatial dimension is about 100 pixel. This was shown to provide a good balance between discriminability of the patch appearance, allowing inference of the relative location with low ambiguity, and repeatability, allowing good generalization during learning.

2.2.1 Training

For training, a data sample $\mathbf{x}$, representing the local image features, is packed with the corresponding class label $y \in \mathcal{Y}$ and an offset vector $\mathbf{o} \in \mathcal{O}$. Hence, the training set looks as follows: $\mathcal{L} = \left\{(\mathbf{x}_1, y_1, \mathbf{o}_1), \ldots, (\mathbf{x}_{|\mathcal{L}|}, y_{|\mathcal{L}|}, \mathbf{o}_{|\mathcal{L}|})\right\} \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$[‡]. Thereby, an offset vector $\mathbf{o}$ represents the displacement from the center of the image patch to the object center[§].

For Hough Forests, both, classification as well as regression objective functions are used for split selection. Gall and Lempitsky [32] proposed to randomly select the objective at each node. In contrast, in [63] the objective for split selection is adaptively combined from regression and classification objectives. Thereby, the weighting depends on the ratio of positive samples arrived in a node. Hence, they start with growing the trees as decision trees and gradually shift the strategy for node-splitting to building regression trees. For a more recent evaluation it has been shown that this strategy might improve the results for some datasets, using an optimal setting of the weighting parameter – but not in general [35]. Based on these findings, for our task, we will randomly select whether to use a classification or regression objective function.

For classification nodes the uncertainty measure from Eq. (2.5) is used. The regression objective on the other hand aims at minimization of the uncertainty of the offset vectors, and is thus based on their deviation from the mean [35][¶]:

$$V(\mathcal{L}) = \sum_{y \in \mathcal{Y} \setminus \{0\}} \sum_{\mathbf{o} \in \mathcal{O}_P^{(y)}} \|\mathbf{o} - \bar{\mathbf{o}}\|_2^2, \tag{2.8}$$

where $\bar{\mathbf{o}}$ is the mean of the offset vectors corresponding to the training samples of class y reaching node P:

$$\bar{\mathbf{o}} = \frac{1}{\left|\mathcal{O}_P^{(y)}\right|} \sum_{\mathbf{o} \in \mathcal{O}_P^{(y)}} \mathbf{o}. \tag{2.9}$$

[‡]For the sake of readability the training set is again denoted $\mathcal{L}$. Throughout the remainder of the document $\mathcal{L}$ is used as specified here.

[§]Note that negative samples, *i.e.*, samples from the background are not assigned with an offset vector.

[¶]Note that $y = 0$ for the background class.

The uncertainty within the samples sent to the left ($\mathcal{L}_L$) and the right child node ($\mathcal{L}_R$) are simply added up to obtain an uncertainty measure for the given split function ϕ:

$$U_{\mathcal{O}}(\mathcal{L}, \phi) = V(\mathcal{L}_L) + V(\mathcal{L}_R). \tag{2.10}$$

Subsequently, the best test is selected based on the chosen objective function, *i.e.*, either the regression (Eq. (2.10)) or the classification (Eq. (2.5)) objective:

$$\operatorname*{argmin}_k \; U_\star(\mathcal{L}_P, \phi(\mathbf{x}, \Theta_k)), \tag{2.11}$$

where $\star$ indicates the chosen uncertainty measure, *i.e.*, $U_{\mathcal{Y}}$, or $U_{\mathcal{O}}$.

2.2.2 Detection

When searching for an object in an image, image locations are densely sampled and associated to a leaf of each tree of the Hough Forest. Subsequently, the predictor models at the leafs cast votes for object positions in the Hough space. Specifically, from the local image appearance around each sampled position $\mathbf{v}$, a data sample $\mathbf{x}_\mathbf{v}$, representing the image features, is extracted. The sample is passed down each tree, and associated to a leaf l_t. The class probability $p_t(y|\mathbf{x}_\mathbf{v})$ and the offset vectors $\mathbf{o} \in \mathcal{O}_{l_t}^{(y)}$ of class y stored at the leaf are then used to cast votes for an object at position $\mathbf{u}$ and scale s. To this end, we are interested in $p\big(\mathbf{h}(y, \mathbf{u}, s)|\mathbf{x}\big)$, where $\mathbf{h}(y, \mathbf{u}, s)$ denotes the hypothesis for an object of class y centered at $\mathbf{u}$, with scale s. For a single data sample $\mathbf{x}_\mathbf{v}$ and a single tree t this can be decomposed as follows:

$$p_t\big(\mathbf{h}(y, \mathbf{u}, s)|\mathbf{x}_\mathbf{v}\big) = p_t\big(\mathbf{d}(\mathbf{u}, \mathbf{v}, s)|y, \mathbf{x}_\mathbf{v}\big)\; p_t(y|\mathbf{x}_\mathbf{v}), \tag{2.12}$$

where

$$\mathbf{d}(\mathbf{u}, \mathbf{v}, s) = \frac{s_u(\mathbf{u} - \mathbf{v})}{s} \tag{2.13}$$

specifies the offset from the patch center $\mathbf{v}$ to the object center $\mathbf{u}$ with respect to the unit size s_u. As mentioned before, the class probability $p_t(y|\mathbf{x}_\mathbf{v}) = p(y|\mathcal{L}_{l_t})$ is estimated from the ratio of samples from class y arrived at leaf l_t during training. The distribution $p_t\big(\mathbf{d}(\mathbf{u}, \mathbf{v}, s)|y, \mathbf{x}_\mathbf{v}\big)$ is modeled by the set of offset vectors $\mathcal{O}_{l_t}^{(y)}$ corresponding to class y.

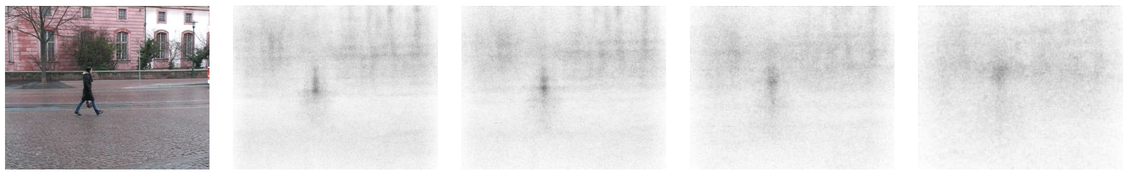


Figure 2.3: Example input image (left), and output of our Hough Forest, *i.e.*, resulting slices of the Hough space corresponding to different scales. Note, light areas represent low, dark areas high probability for an object center location.

As a result Eq. (2.12) becomes

$$p_t\big(\mathbf{h}(y,\mathbf{u},s)|\mathbf{x_v}\big) = \frac{1}{\left|\mathcal{O}_{l_t}^{(y)}\right|}\left(\sum_{\mathbf{o}\in\mathcal{O}_{l_t}^{(y)}} \mathbf{1_o}\left(\frac{s_u(\mathbf{u}-\mathbf{v})}{s}\right)\right) p_t(y|\mathbf{x_v}), \tag{2.14}$$

where $\mathbf{1}$ specifies the indicator function[‖].

To obtain a prediction for the whole forest, the predictions from the individual trees are averaged:

$$p\big(\mathbf{h}(y,\mathbf{u},s)|\mathbf{x_v}\big) = \frac{1}{T}\sum_{t=1}^{T} p_t\big(\mathbf{h}(y,\mathbf{u},s)|\mathbf{x_v}\big). \tag{2.15}$$

Furthermore, the votes from all sampled image patches of the input domain $\Omega \subseteq \mathbb{R}^D$, *i.e.*, from all data samples $\mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} = \Omega$, extracted from the test image, are accumulated:

$$p\big(\mathbf{h}(y,\mathbf{u},s)|\Omega\big) = \frac{1}{n}\sum_{i=1}^{n} p\big(\mathbf{h}(y,\mathbf{u},s)|\mathbf{x}_i\big). \tag{2.16}$$

It should be noted that the resulting values $p\big(\mathbf{h}(y,\mathbf{u},s)|\Omega\big)$ are not true probabilities[**]. Nevertheless, since we are only interested in an ordinal measurement, this can be ignored and the result still serves as a confidence measure. In fact, we only aim at finding the modes of $p\big(\mathbf{h}(y,\mathbf{u},s)|\Omega\big)$, which are extracted by applying *mean shift* [15]. See Fig. 2.3 for an example input image and slices of the resulting Hough space corresponding to different scales.

2.2.3 Implementation Details

For detection the patches are sampled densely, *i.e.*, at each pixel [32]. This was shown to outperform the strategy when sampling the patches at interest points, as it was proposed

[‖]In this case, simply returns 1 if the parameter equals offset $\mathbf{o}$, and 0 otherwise.

[**]See [6] for a discussion on this issue.

for ISMs [52]. Nevertheless, later it was shown that a *grid-like* sampling strategy can be used to reduce the run time, while the loss in accuracy can be kept in an acceptable range [34].

In the original formulation, 32 feature channels were employed [32]: The three color channels of the *Lab* color space, the absolute values of the first and second order derivatives in x-, and y-direction, and nine HOG [20] channels. A HOG channel was obtained as the soft bin count (*i.e.*, the weighted sum) of gradient orientations in a 5×5 neighborhood. The resulting 16 channels were each min-, and max-filtered, again using a 5×5 filter size, to finally obtain 32 channels. Thus, given 16×16 image patches, for the data samples $\mathbf{x}_i \in \mathbb{R}^D$ this results in a feature dimensionality $D = 8192$. Gall and Lempitsky [32] solely used two-value-tests (*c.f.*, Eq. (2.2)), *i.e.* two features were randomly sampled for each test. Additionally, these two features were restricted to be sampled from the same channel.

2.2.4 Further Extensions and Related Work

In several works efforts were made to learn discriminative weights for voting [57, 70, 89]. While [57] builds upon the original ISM formulation by learning weights for the codebook entries using a max-margin framework, [89] employs the Hough Forest framework and learns discriminative weights for each training patch, and thus, for each individual vote. Another way to measure and incorporate the discriminative power of local image patches throughout training and testing of a Hough Forest was shown in [70]. To this end, a measure of self-similarity of an image patch, based on the already trained forest, is utilized, incorporating it as a sparsity potential for object detection.

Many applications – including multiple instance tracking – require discrimination between multiple objects, or object classes, respectively. The inherent ability of Random Forests to handle multiple classes is also utilized with Hough Forests [35, 71]. For instance, in [71] the amount of features shared among different classes is explicitly exploited. To this, a taxonomy of the classes is generated based on the sharing distributions of features among them. Subsequently, the taxonomy is employed to reduce the number of votes to be cast.

Hough Forests have also been applied to tracking. In [33] a Hough Forest based general person detector is trained off-line and adapted on-line, by updating the instance specific statistics, only. Hence, the foreground probability, as well as the offsets are kept unchanged. For tracking, the detector is coupled with a particle filter. Nevertheless, the tracker may be confused with other instances, which are similar according to the pre-trained codebook, since no features for explicit instance discrimination are learned. Refusing to add instance offsets may keep the runtime stable but inhibits the trackers ability to adapt to the specific instances, or localize them accurately.

Godec *et al.* [38] propose a more adaptive model based on Hough Forests, where they also utilize back-projection to locate the support of a detection. This, in turn, guides a rough GrabCut segmentation used to reduce the amount of noisy updates. These ideas are re-used in [68], where the GrabCut algorithm is exchanged for an alpha matting based segmentation. Furthermore, their model is built upon three sub-forests: A non-adaptive, a moderately adaptive, and a highly adaptive sub-forest [68]. The non-adaptive sub-forest is solely offline trained at the first frame, the moderately adaptive (on-line) sub-forest is updated at each frame, and the highly adaptive sub-forest is completely re-trained at each frame.

Random Forests are also extensively applied to tasks of the medical image processing domain (see, *e.g.*, [18, 19, 61]). In [19] Criminisi *et al.* aim at regressing the position and size of objects similar to Hough Forests. In their work they detect organs like the lung, the kidney or the gall bladder in CT scans. However, in contrast to the approach described above, the positions of the 6 walls of the objects 3D bounding box are estimated instead of the object center and scale.

For a more comprehensive review on Hough Forests for object detection and tracking please be referred to [34]. However, it should be noted that the idea of Hough Forests have not only been employed to object detection and tracking. In fact, it was very successfully transferred to other tasks like pose estimation.

In [37] human poses are estimated based on a Hough voting scheme. Relying on the input of a depth sensor, a Hough Forest is trained for regression of the joint locations. Instead of casting votes in a discrete voting space, in this approach the votes are cast into a continuous 3-dimensional space. The follow-up work of [82] is motivated by the fact that locations of different joints are not independent. In contrast, the joint locations are sometimes even strongly dependent on global variables like human height or torso orientation. They show that accuracy can be significantly improved by incorporating such a global variable in the inference process. To this end, prior knowledge as well as assumptions about temporal consistency are exploited for initial estimates about the state of the global variable.

A similar idea is employed for detection of facial features [16, 21]. In [21] the forest is again partitioned into several sub-forests, each trained on its individual subset of the data in order to solve a specific part of the task. During test-time, the head pose is initially estimated and, subsequently, a subset of trees trained on the corresponding subset of the training data is selected to predict the facial feature points. Cootes *et al.* [16] provide an evaluation of Random Forest based regression of facial features and hand joint locations as the basis of a shape model fitting. In their work they show that the Random Forest based regression outperforms other methods like boosted regression or Random Forest based classification in terms of accuracy and speed.

Recently, the Hough Forest voting scheme was also applied to regression of hand poses [83]. Thereby, labeled and unlabeled real-, as well as synthetic training data is exploited jointly. The discrepancies between realistic and synthetic data are particularly considered by integrating a transductive term into the split objectives. Whereas, another semi-supervised term aims at proper incorporation of labeled and unlabeled data. In this way, associated realistic and synthetic data, as well as labeled and unlabeled data of similar appearance are enforced to be clustered together by the proposed *semi-supervised transductive regression forest*.

Chapter 3

Efficient Hough Forests

Contents

Computer vision applications typically require efficient methods. This comes from the fact that they either have to deal with a huge amount of data or require short response times. The latter is usually the case for object tracking, as it builds the basis for many computer vision applications. However, for numerous tracking tasks multiple cameras have to be used, which implies a multiplied amount of data to be processed, and thus, requires even faster methods. In this section we will outline several enhancements towards efficient object detection using Hough Forests. In this regard, the first step is to theoretically analyze which of the individual processing parts are critical in terms of runtime.

3.1 Theoretical Analysis

The detection process with Implicit Shape Model based approaches usually consists of three main parts: Feature computation, matching local image patches to the codebook, and casting votes in the Hough space. In the following, we will identify the parameters which determine the runtime of these parts.

To start with, we consider the computational cost of feature extraction. This is related to the number of data samples n, *i.e.*, the number of sampled patches used for detection. Given dense, or nearly dense sampling, the features for neighboring, *i.e.*, overlapping, patches are partly the same and don't need to be recomputed. Hence, as long as the sampling distance is sufficiently small, which has shown to be crucial for accurate detection [35], the time for feature computation is independent from n. Furthermore, assuming that the cost of extracting a single feature is constant, the runtime for feature computation for a single scale, only depends on the number of feature channels F, and image pixels in the search area I:

$$\mathcal{C}_{\mathcal{F}} = O(FI). \tag{3.1}$$

While the cost of matching a single data sample to the codebook for the original ISM [52] scales linearly with the size of the codebook, for codebooks based on a Random Forest, this cost reduces to logarithmic dependency on codebook size. In fact, it is the sum of the cost of matching a sample to a leaf of each tree within the forest:

$$\mathcal{C}_{\mathcal{M}} = \sum_{t=1}^{T} O\big(\kappa \log(|l_t|)\big), \tag{3.2}$$

where $|l_t|$ specifies the number of leafs of tree $t \in \{1, \dots, T\}$, and κ denotes the complexity of a single splitting test. Ignoring that $|l_t|$ may slightly differ from tree to tree, the cost of matching becomes:

$$\mathcal{C}_{\mathcal{M}} = O\big(T\kappa \log(|l|)\big). \tag{3.3}$$

Furthermore, let $|\mathcal{O}_{\mathcal{L}}|$ be the number of all offsets in the training data, assuming balanced trees, *i.e.*, the training samples were evenly distributed among the leafs of a tree, we can finally derive the cost of detection:

$$\begin{aligned} \mathcal{C} &= \mathcal{C}_{\mathcal{F}} + n\Big(\mathcal{C}_{\mathcal{M}} + \frac{|\mathcal{O}_{\mathcal{L}}|}{|l|}\Big) \\ &= O\Big(FI + nT\kappa \log(|l|) + n\frac{|\mathcal{O}_{\mathcal{L}}|}{|l|}\Big). \end{aligned} \tag{3.4}$$

3.2 Outline

In the following, we take a look at every single parameter of Eq. (3.4). To start with, in Section 3.3 we will discuss the influence of standard parameters of a Random Forest, *i.e.*, the number of trees and their depth, on the forests efficiency (parameters: T, and $|l|$). Subsequently, we will outline possible speed-ups through non-dense sampling (Sec. 3.4; parameter n), reduction of data complexity (Sec. 3.5; parameter: I), reducing the feature dimensionality (Sec. 3.6; parameters: F, κ), and compression of the offset distributions at the leafs (Sec. 3.7; parameter: $|\mathcal{O}_\mathcal{L}|/|l|$). Additionally, for speeding up the training time, we consider sub-sampling the training data at the split nodes in Sec. 3.8. Finally, in Sec. 3.9 we will summarize the results, conclude a final system, and show the overall performance gain.

The analysis in this section is based on a standard dataset: The *TUD-Pedestrian* dataset [2][*]. It contains 400 training images, and 250 test images with 311 annotated persons. For the training data a detailed segmentation is available. However, for our experiments we did not experience a significant change in accuracy, when utilizing the provided foreground masks. Thus, in the following, we make only use of the bounding box annotations. For performance evaluation we employ the PASCAL overlap criterion with a threshold of 0.5 [25]. For the results, we are solely interested in the change of accuracy and runtime, and also want to disregard the performance of the test machine. Therefore, we always compute the accuracy and runtime measurement relative to a reference, which is explicitly stated for each experiment. Furthermore, to account for the randomization, we conducted several runs for each experiment and report only the averaged result.

3.3 Classifier Complexity

Since each of the individual trees in a Random Forest has to be evaluated for every single data sample, an increase in the number of trees T inevitably yields an increase in processing time. In fact, the runtime scales linearly with the number of trees in the forest, while the accuracy only shows a slight increase for $T > 4$. This is illustrated in Figure 3.1 and was also indicated by other authors [64].

In contrast, the relation between tree depth and runtime cost is inverse. An increase in depth of a tree results in an increase of the number of leafs $|l|$, and thus, a decrease in the number of votes per leaf. In fact, from Eq. (3.4) it can be seen that increasing

[*]Training and test data is publicly available at `http://www.d2.mpi-inf.mpg.de/andriluka_cvpr08`.

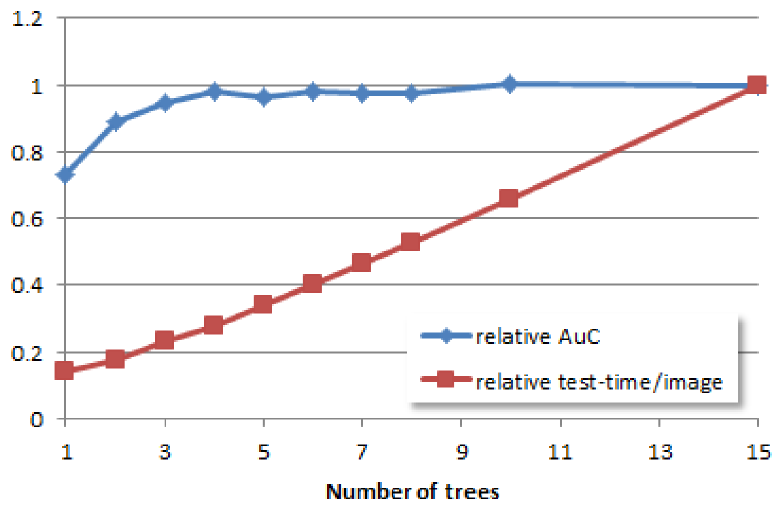


Figure 3.1: Relative accuracy (in terms of *Area under the Curve* (AuC)) and relative runtime measurements as a function of the number of trees in the forest; values are relative to using 15 trees.

$|l|$ will decrease the overall cost for typical values, where $|\mathcal{O}_{\mathcal{L}}|$ is huge, while T and κ are comparably small. Nevertheless, this is only true for perfectly balanced trees, since Eq. (3.4) is based on this assumption.

In practice, we did not experience a significant change in runtime when increasing the maximum tree depth from 15 up to 18. This indicates an imbalance at deeper levels of the trees. Furthermore, we have to note that increasing the depth of the trees also bears the risk of overfitting to the training data (see, *e.g.*, [18]). For our experiments, we found only a small change in accuracy for increasing the tree depth, but the deeper we trained our trees, the more frequently we were facing high scoring false positives. This could be recognized starting from a maximum depth of 18 in our case. Nevertheless, we have to note this strongly depends on the amount of training data and variance in it.

3.4 Sampling Distance

The number of data samples n, extracted from a test image is obviously a decisive factor for the runtime. It may be reduced by different sampling strategies. For example, in the original ISM formulation [52] the patches were sampled at interest points, only. With Hough Forests [32], on the other hand, dense sampling were applied, which was shown to improve results significantly. Nevertheless, it was also shown that the accuracy only gracefully degrades for nearly dense sampling strategies [32, 34].

Sampling distance	dense (1)	2	3
Relative AuC	1	0.999	0.973
Relative test time	1	0.590	0.513

Table 3.1: Mean accuracy and test time for different sampling distances during detection; values are relative to those for dense sampling

To this end, we evaluated sampling from a larger grid, *i.e.*, run the detection, *e.g.*, at every second or third pixel. Fixing the sampling distance, δ, in x- and y-direction reduces the number of samples by a factor of $1/\delta^2$. Hence, this provides the possibility to significantly reduce the computational burden for small sampling distances, already. We found that setting $\delta = 2$ didn't influence detection accuracy, while yielding a speed up of about 40%. Increasing δ further gave – despite the much smaller number of samples which need to be evaluated – a slight additional speed-up, only. This indicates that the computational bottleneck moved to other parts of the pipeline. See Tab. 3.1 for the detailed results.

3.5 Data Complexity

To detect objects of different sizes with Hough Forests, the common procedure of modern object detectors is used: Thereby, a model is trained on a single unit scale (s_u), and at detection time a test image is scaled so that possible objects fit to the unit scale. Subsequently, all features are solely computed from this scaled image. Hence, the smaller s_u, the less pixels have to be considered for feature computation. This arises the question if we are able to discriminate the object from background at a smaller resolution as originally suggested in [32], where s_u were set to 100 pixel.

We evaluated for three different scales $s_u \in \{100, 75, 50\}$, and adapted the patch size accordingly. See Fig. 3.2 for an object shown at each utilized resolution. Our experiments reveal that setting $s_u = 75$ does not yield any loss in accuracy, while achieving a runtime gain of more than 50%. By setting $s_u = 50$ one has to sacrifice some accuracy, but in turn the runtime even drops to one fifth. The detailed accuracy and runtime changes are given in Tab. 3.2. Note, for the presented evaluation we didn't use min-, and max-filtration of the feature channels, since noise suppression is implicitly done by down-scaling, and we found that an additional filter step decreases discriminability at smaller scales.

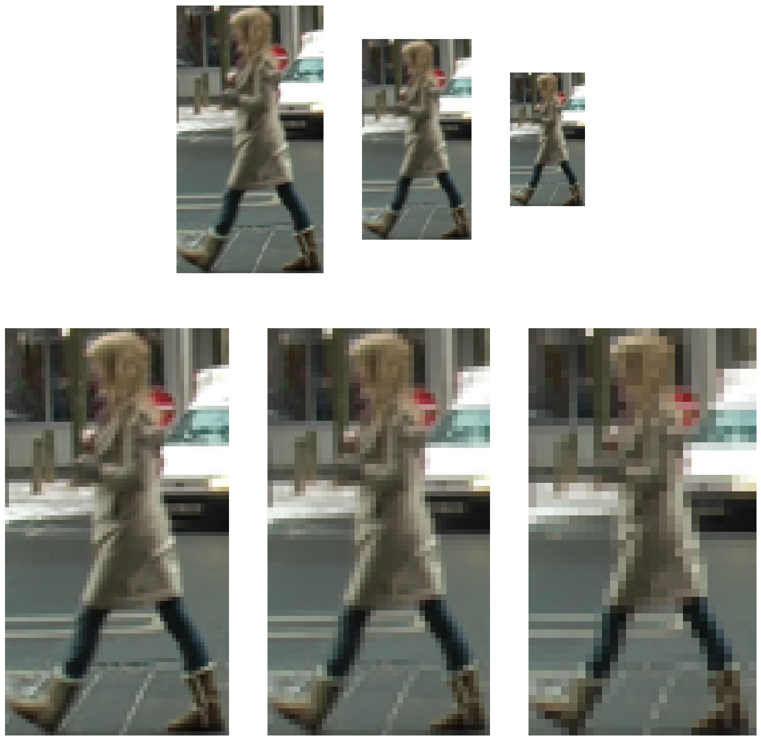


Figure 3.2: Illustration of a training sample at various resolutions. Top row: images at original size with a height of 100, 75, and 50 pixels; Bottom row: the same images re-scaled to equal size in order to illustrate the differences in captured information at different resolutions

3.6 Features

In this section we will first evaluate the impact of using different subsets of feature channels on accuracy and runtime. Hence, we will investigate whether and how the number of feature channels F can be reduced. This is followed by a discussion on the influence of the splitting test complexity κ.

Object height	100	75	50
Relative AuC	0.987	1.001	0.903
Relative test time	0.870	0.480	0.190

Table 3.2: Relative accuracy and test time for models trained on different object resolutions (*i.e.*, $s_u \in \{100, 75, 50\}$); no min-, and max-filtration were used for these runs, *i.e.*, 16 feature channels; values are relative to the result obtained using the standard setting, *i.e.*, $s_u = 100$, patch size: 16×16, and using all 32 channels (including min- and max-filtration).

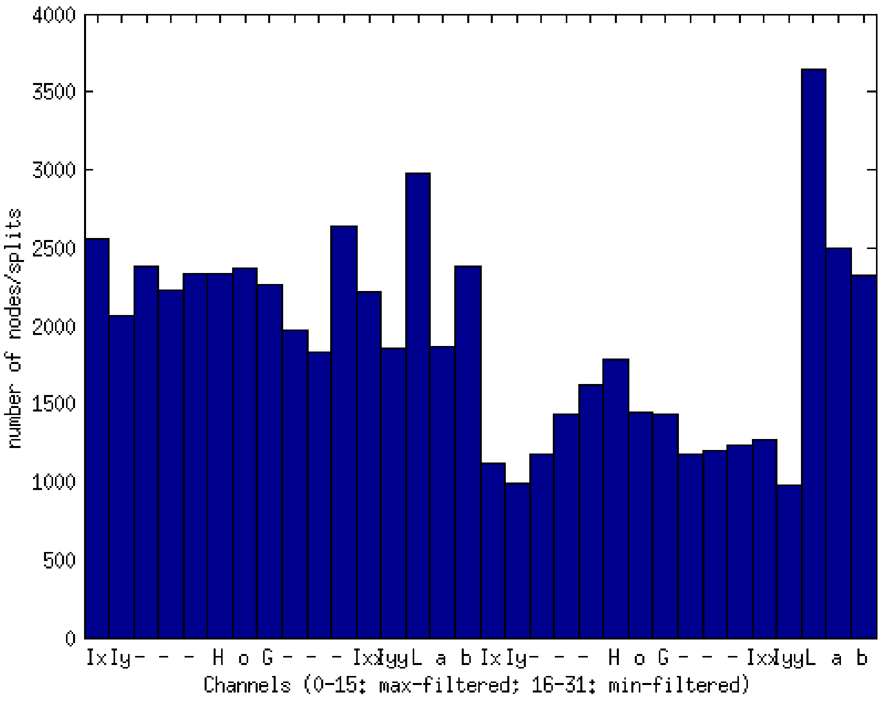


Figure 3.3: Histogram over the 32 feature channels, showing how often specific channels are selected for a split test.

3.6.1 Feature Channels

Differences in the importance of feature channels have been exploited recently. For example, in [61] the features are not selected from a uniform distribution, as usually. Instead, the utility of the individual features is learned from an initial forest, and re-used at the construction of the final classifier. Similarly, we want to measure the contribution of the 32 individual feature channels (First- and Second-Order gradients, HOG, and *Lab*). To get an idea about their importance, we start by counting the number of nodes at which a splitting test based on a specific channel is selected. Hence, by doing this for each channel, we can obtain a histogram over the channels. These histograms have been analyzed for several Random Forests, finding that they usually have a couple of properties in common. For instance, the color channels usually represent modes in the histogram, for the min-, as well as the max-filtered channels, and at least some of the HOG channels often make up another peak. An example for such a histogram is shown in Fig. 3.3.

Since we also found that each of the 32 feature channels is selected in a significant number of nodes (*c.f.*, Fig. 3.3), this analysis would indicate that only very few channels

Setting	# channels	rel. runtime	rel. AuC
Baseline [35]	32	1.00	1.00
Without min/max filtering	16	0.78	0.98
Without min/max and 1st deriv.	14	0.77	0.97
Without min/max and 2nd deriv.	14	0.76	0.97
Without HoG	14	0.71	0.95
Without Lab	26	0.96	0.90

Table 3.3: Relative test time and accuracy (in terms of *Area under the Curve* (AuC)) for different combinations of feature channels.

may be discarded without implying a significant drop in accuracy. Nevertheless, different features may be highly correlated, and carry similar information, respectively. Discarding one of these wouldn't affect accuracy detrimentally, despite its frequent selection. Thus, the raw information about how often a feature channel is selected is not enough when judging about which of them may be discarded. This fact was already shown by Breiman in some initial experiments [10]. Hence, aiming at a reduction of the feature channels with minimal loss in accuracy, we have to evaluate the impact directly, by training forests using different subsets of feature channels. Doing so, we found that more than half of the feature channels may be simply omitted, while the loss in accuracy can still be kept at an acceptable level. Especially, omitting the min- and max-filtration – reducing the number of feature channels by half – results in only a slight decrease in accuracy, while the runtime drops at about 20%. Note that, even though the number of feature channels is halved, still the same features have to be computed and only the additional filtration is omitted. Tab. 3.3 gives the detailed results on how accuracy and runtime are affected by using different combinations of feature channels.

It should be noted, that the results shown in Tab. 3.3 were generated using vote compression. Thus, the relative runtime gain noted in Tab. 3.2 for omitting min-, and max-filtration (produced without vote compression) differs from the results stated here. Theoretically, this difference can simply be derived from the fact that the number of offsets, $|\mathcal{O}_\mathcal{L}|/|\mathcal{L}|$, and the number of feature channels, F, are contained in different additive terms of Eq. (3.4). Please also note, that in order to keep the "randomness" at the same level, the number of randomly generated splitting tests has to be reduced when reducing the dimensionality of the feature space, as it is done here.

Type	Training error	rel. AuC
Single-value splits	0.01	0.817
Two-value splits	0.06	1.000
Selection w.r.t. objective	0.01	0.840

Table 3.4: Error rate on training data, and relative accuracy (in terms of *Area under the curve*) for different split types; Values are relative to the accuracy for using two-value splits, which is the standard approach [32]

3.6.2 Test Types

Another, not negligible issue is the complexity of the actual test type. To this end, in Sec. 2.1 we introduced two commonly used types, namely single-value- and two-value-tests. Single-value-tests compare a single pixel value against a threshold, while two-value-test compare the difference of two values against a threshold. Hence, by using two-value-tests, the retrieval of twice as much values, plus an additional arithmetic operation per split is necessary. Processing an image, each of the – probably densely – sampled patches has to be matched to a leaf of each tree. Thus, $n \times T \times \log(|l|)$ splits need to be computed, which is usually in the range of 10^7 to 10^8 for a 640×480 image.

We found the possible speed up, when moving from two-value-splits to single-value-splits, to be up to 35%. Hence, we will restrain from using even more complex features (see *e.g.*, [18, 34, 76]), and employ only these two.

In search for a setup, which keeps the loss in accuracy as small as possible, we compare three settings: Restricting the system to use (i) single-value splits, (ii) two-value splits, or (iii) let it randomly choose tests from both types and select the best using the objective function (Eq. (2.11)). To this end, we expect the latter to give the best results. Nevertheless, best performance is reached by restricting the system to two-value splits, only (see Tab. 3.4). This indicates that the system is more likely to over-fit to, *e.g.*, noise, or intensity variations if single-value splits are used. Which, furthermore, causes the system to tend to selecting single-value-tests rather than two-value-tests if it is left over with the decision (see Fig. 3.4). The aforementioned suspicion is also confirmed by the error rates on the training set – shown in Tab. 3.4.

From our experiments we can conclude that the entropy and the variance of offset vectors, respectively, lack in their ability to assess the quality of a split under noisy conditions. To improve towards this end, better information gain estimates have recently be shown to slightly improve the overall accuracy of Random Forests [62], whereas, more appropriate uncertainty measures are able to capture the multi-modality in the distributions of the

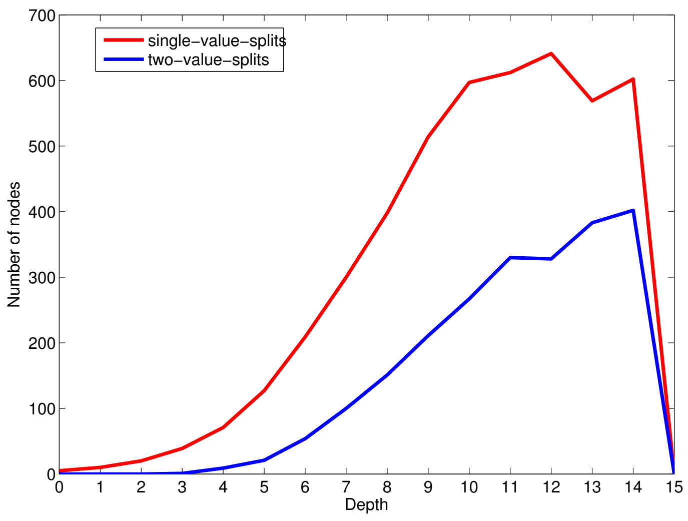


Figure 3.4: Amount of nodes in which a specific test type (*i.e.*, a single-, or two-value-test) is selected as a function of the depth level. For this experiment the system selected the test type at each node based on the objective function.

offset vectors [87]. Nevertheless, one can still not expect them to be robust against noise or other misleading input. Instead, the space of possible splitting tests has to be carefully set up during system design, already.

3.7 Vote Compression

Random Forests are extremely efficient. They allow for fast predictions, even in cases were a huge amount of data has to be processed. Since, they do not generalize well from a small number of training samples, it is vital to provide them with a sufficiently large training set [34]. With Hough Forests the runtime is dependent on the number votes to be cast, and thus, on the amount of training data. An increased amount of training data results in an increased number of votes to be cast, and hence, increased runtime. This inhibits one of the most important advantages of Random Forests – its efficiency.

Girshick *et al.* [37] address this drawback by summarizing voting information at the leaf nodes. To this end, only a fixed number of votes are cast per leaf, making the voting process independent of the number of training samples again. This is especially important for our application of on-line learning during tracking, where the amount of training data is constantly increasing over time. Additionally, – for the application of pose estimation – Girshick *et al.* showed that this process not only speeds up the voting process but even increases the overall accuracy.

The offset distribution at the leafs is often still multi-modal and contains outliers. Hence, uniformly sub-sampling the offsets, or fitting a single Gaussian to them is a bad choice. In [37] this problem is overcome by applying *mean shift* [15] for vote compression. Their approach is discussed and evaluated for object detection in Sec. 3.7.1. For the task of object tracking the offset distribution has to be updated on-line. This is not straightforwardly implemented for a distribution represented by mean shift modes. More specifically, adding, merging, or discarding modes is virtually impossible without collecting and storing all arriving offsets. And, even if we would ignore that collecting all offsets forever is infeasible for many applications, efficient updates would still be a problematic issue. A possibility to overcome these drawbacks is to employ a *grid-like* quantization of the offset distribution [39] for vote compression. This approach is not only scalable, but also allows simple application of a forgetting factor, and above all, for straightforward on-line updating, which is imperative for the task of tracking. We introduce and compare to this approach in Sec. 3.7.2.

3.7.1 Mean Shift

For vote compression using mean shift, the offset vectors at a leaf are clustered, and the found modes define the new vote vectors. Hence, the whole distribution is solely represented by these modes. In [37] the number of offsets that reached a mode is used as the vote weight. Nevertheless, since the offset vectors supporting a mode can still be strongly distributed in some cases, this approach for weight computation may poorly represent the actual distribution. For this reason, we compare three different methods for vote weighting, where we base the weight on: (i) All points ended up in the mode, (ii) the number of points inside the kernel of the final mode, and (iii) the density within the kernel of the final mode. Furthermore, the weighting approaches are evaluated using each of the following three mean shift kernels: Uniform, normal, and Epanechnikov kernel. Even though, vote weighting method (iii) performs best over all three kernel, the differences in the results are insignificant and thus indicate that neither the vote weighting method, nor the kernel type is a critical factor in terms of accuracy. See Tab. 3.5 for the detailed results.

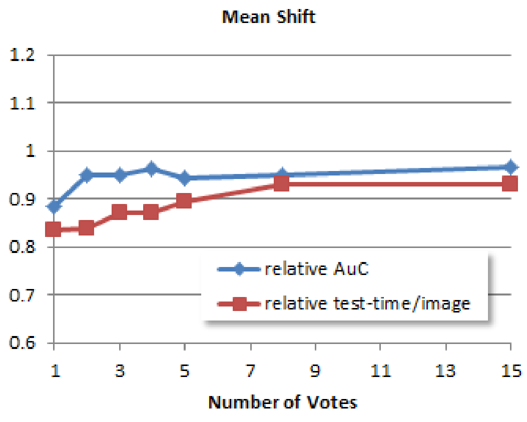
Furthermore, we evaluated the influence of varying the maximum number of mean shift modes, *i.e.*, the number of votes cast per leaf. While, increasing the number of votes from one to two resulted in an accuracy gain, increasing it further didn't show any significant differences. Regarding the test time, we experienced an approximate linear increase for

Vote weighting method	# points ended in mode	# points in kernel	kernel density at mode
Uniform kernel	0.944	**0.951**	**0.951**
Normal kernel	0.946	0.942	**0.951**
Epanechnikov kernel	0.933	0.934	**0.942**

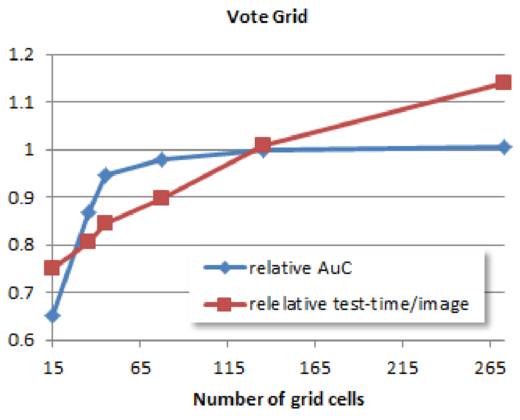
Table 3.5: Relative accuracy (in terms of *Area under the Curve*) for different combinations of kernel type and vote weighting method. Each row gives the result for a specific mean shift kernel; column header specifies the measurement the vote weighting is based upon; given accuracies are relative to the accuracy when no vote compression is used; result for the best vote weighting method printed in bold letters for each kernel.

one to about eight modes per leaf, whereas for a higher number of modes the test time didn't change, anymore. This indicates that – aiming at a speed-up by vote compression – a maximum of five to eight modes can be extracted from the the entire offset distribution at a leaf. Fig. 3.5(a) compares the relative decrease in runtime and accuracy.

Finally, we have to note that using mean shift for vote compression we were not able to improve the results for object detection. This is in contrast to the results obtained by Girshick *et al.* [37] for the task of pose estimation.



(a)



(b)

Figure 3.5: Relative accuracy (in terms of *Area under the Curve* (AuC)) and relative runtime measurements: (a) As a function of the number of votes for vote compression by mean shift, and (b) as a function of the number of cells for vote compression by vote grids. Values are relative to voting with all offsets.

3.7.2 Vote Grid

In order to handle a very diverse set of offset vectors, Godec *et al.* [39] discretize the displacement space into small rectangular cells. For their approach, the main reason for using such small cells was that they didn't enforce to cluster similar offset vectors together during training. More specifically, their approach is based on Random Ferns, where the tests at the split nodes are not optimized, but selected completely at random [36, 65]. In contrast, we incorporate an regression objective during tree construction (*c.f.*, Eq. (2.10)) and can thus expect the offset vectors at the leafs to be sufficiently well separated. In this way, we are able to adapt their idea and utilize it for vote compression by employing grids with a relatively small number of cells. Thereby, we assign each offset vector $\mathbf{o}$ to the cell exhibiting the closest center $\mathbf{c}$. Subsequently, at detection the centers of the cells form the relative votes, weighted by the number of assigned offset vectors. Hence, the distribution $p_t(\mathbf{h}(y, \mathbf{u}, s)|\mathbf{x_v})$ formulated in Eq. (2.14), defining a vote weight becomes

$$p_t(\mathbf{h}(y, \mathbf{u}, s)|\mathbf{x_v}) = \sum_{\mathbf{c}} \left(\mathbf{1_c}\left(\frac{s_u(\mathbf{u}-\mathbf{v})}{s}\right) w_{\mathbf{c}} \right) p_t(y|\mathbf{x_v}), \tag{3.5}$$

$$w_{\mathbf{c}} = \frac{1}{\left|\mathcal{O}_{l_t}^{(y)}\right|} \sum_{\mathbf{o}\in\mathcal{O}_{l_t}^{(y)}} \mathbf{1_c}(\,\chi(\mathbf{o})\,), \tag{3.6}$$

where $\chi(\mathbf{o})$ gives the cell center closest to the offset vector $\mathbf{o}$:

$$\chi(\mathbf{o}) = \underset{\mathbf{c}}{\operatorname{argmin}} \left\|\mathbf{o} - \mathbf{c}\right\|_2 . \tag{3.7}$$

See Figure 3.6 for some example grids with different resolutions. Using grids with high resolution, *i.e.*, a large number of cells, results in high accuracy, but low speed. By reducing the grid resolution one may easily reduce the runtime – for the cost of decreased detection performance. Nevertheless, our investigation reveals that using a grid voting scheme, the runtime scales nearly linearly with the grid resolution, whereas the accuracy suffers only for a very small number of cells. Thus, the accuracy level of voting with all offsets is reached with grid resolutions where the runtime is significantly lower.

In our experiments the runtime reaches the same level when using about 135 cells, while the accuracy reaches the same range with resolutions far below that. Hence, a significant speed-up can be achieved for a small decrease in accuracy (see Fig. 3.5(b)). For a fair comparison we used the same codebooks for voting with all offsets, and grid-voting. Only the representation of the offset distribution is changed accordingly. For example

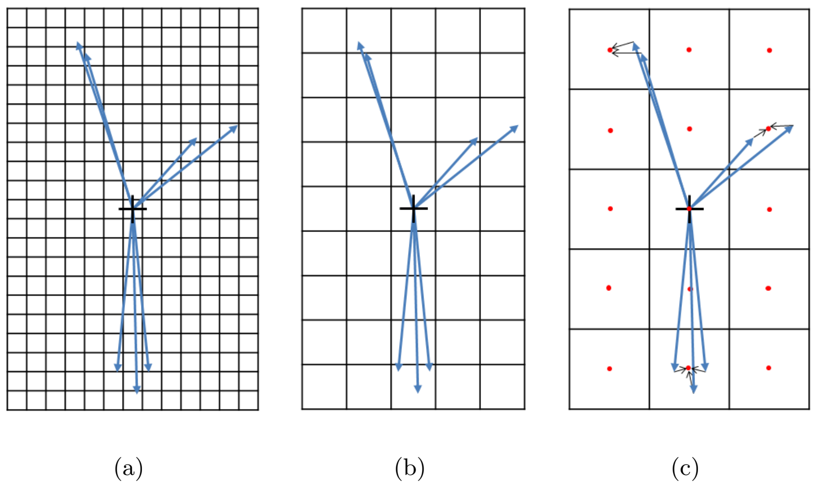


Figure 3.6: Illustration of example vote grids and offset vectors, with (a) 21×13, *i.e.*, 273 cells, (b) 9×5, *i.e.*, 45 cells, and (c) 5×3, *i.e.*, 15 cells. For the rightmost grid, the assignment to cell centers is visualized.

slices of the resulting Hough space see Fig. 3.7. Please note that the exact runtime for the grid voting scheme certainly depends upon implementation. For this evaluation we were disregarding a possible performance gain through the use of efficient data structures, exploiting, e.g., the sparsity of the grids. Hence, while the conclusion remains valid, the grid voting scheme is expected to bear potential for further improvements in efficiency.

3.8 Sub-sampling Revisited

While training time efficiency may be disregarded for fully off-line trained tree-models, it is a crucial issue for adaptive models, and on-line growing of trees, respectively. To this end, sub-sampling the training data for split optimization at the nodes was proposed by Schulter *et al.* [75]. This simple, but effective strategy keeps training time constant, *i.e.*, independent of the amount of training data. Additionally, it has been shown that the accuracy may even be increased significantly. While in [75] it was hypothesized that the reason for the gain in accuracy is the increased decorrelation between the individual trees of the forest, we experimentally identify the balance of the sub-samples to be another essential ingredient for the success of this strategy. Later, in Sec. 4 this insight is exploited for our task of on-line learning during tracking.

For the approach introduced in [75] the training data is sub-sampled at each split node, a split is found by optimizing on the sub-sample (*c.f.*, Eq. (2.11)), and finally, all available

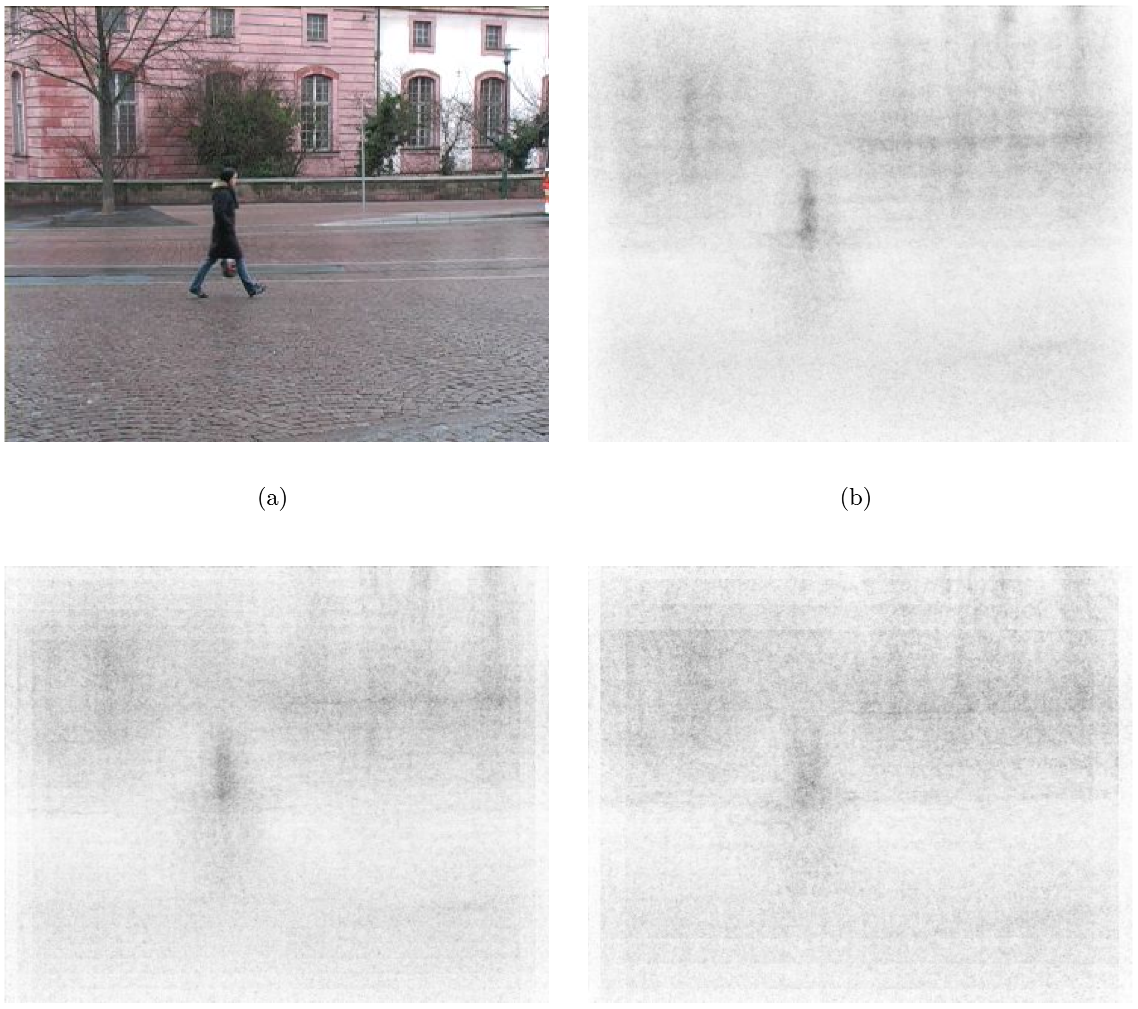


Figure 3.7: Comparison of the Hough maps obtained for the input image (a) when voting with all offsets ended up in the leaf (b), using a vote-grid with a resolution of 9×5, *i.e.*, 45 cells (c), and 5×3, *i.e.*, 15 cells (d). It can be observed that the Hough maps are becoming sparser, and the peaks less discriminative when decreasing grid resolution.

samples are sent to the newly created child nodes. When training a tree down to a depth of 17 using about 60,000 data samples, this simple procedure yields a training time speed up by a factor of six, while increasing accuracy (see Tab. 3.6).

In [75], the drawn sub-sample used for split optimization is balanced, *i.e.*, if possible, the same number of samples are drawn for each class, disregarding the true distribution at the node. To measure the effect of this approach, we compare it to sub-sampling with

Strategy	rel. AuC
No sub-sampling [35]	1.000
Sub-sampling [75]	1.040
Sub-sampling w.r.t. distr.	1.018
Local "un-bias"	1.035

Table 3.6: Relative accuracy (in terms of *Area under the Curve* (AuC)) for different sub-sampling, as well as the re-weighting strategy (*Local "un-bias"*), compared to the standard approach with optimization over all samples.

respect to the class distribution. In the latter case, at each node, we draw a sub-sample which represents the distribution of the available samples at the node. The results in Tab. 3.6 show that more than half of the gain in accuracy is lost by doing so. Hence, the balance of the sub-sample can be regarded to be crucial for the success of sub-sampling as proposed by Schulter *et al.*

The aforementioned approach of balanced sub-sampling already represents a step away from the commonly used procedure, where the split at a node N is optimized under the "current" posterior $p(y|\mathcal{L}_N)$ [1, 18]. Thus, it seems interesting if the balance of the available training data at a node as such, *i.e.*, without sub-sampling, does influence the overall accuracy.

For classification problems, where not equally many samples are available from the different classes, a common approach, is to re-weight the individual data samples so that the sample weights sum up to the same value for each class. Nevertheless, this is usually done globally, at the level of the whole data set, and subsequently the fixed global weights are used locally, *i.e.*, in our case, at each node, when searching for the best split. To further investigate this issue, we set up an experiment, where all available training data at a node is re-weighted so that the sums of the weights per class are balanced. The split at each node is then optimized on this virtually balanced data set. This simple procedure results in a gain in accuracy slightly below that of sub-sampling. For the detailed results see Tab. 3.6, where this strategy is referred to as *Local "un-bias"*.

This confirms our expectation, that local balancing of the training data is a critical factor for generalization performance. Although, this insight can not provide a gain in runtime, in the sequel we will show how to exploit it for robust tracking (see Sec. 4).

	Training time	Test time per image
Rel. runtime (Speed-up factor)	0.054 (∼20)	0.024 (∼40)

Table 3.7: Relative runtime gain; Runtimes of our approach relative to originally proposed approach [35].

3.9 Summary & Conclusion

Throughout this section we, individually, analyzed each critical part of the detection process. In a final experiment we will now integrate all the findings, and evaluate for the overall accuracy and runtime difference to the original approach [32]. This, at the same time, represents the starting point for our multi-camera tracking system.

From the analysis in this section, we know that – without significant loss of accuracy – we can increase the maximum depth of a tree from 15 to 17, and the sampling distance up to two pixels, in x-, and y-direction. Additionally, we set the unit scale of an object to 75 pixel. By concurrently discarding the min-, and max-filtration of the feature channels, accuracy still doesn't suffer. Furthermore, – in this case, accepting a slight loss in accuracy – we reduce the number of trees from 15 to 5, and summarize offset distributions by utilizing grids of 77 cells. To this end, compared to vote compression by mean shift, the reached speed up was slightly smaller when keeping the accuracy in an acceptable range. On the other hand, vote compression using grids have shown to provide comparable or even better results in terms of accuracy. In the end, the possibility for straightforward on-line updates, which would be problematic for mean shift modes, makes us preferring grids over mean shift with respect to the task of tracking. For the choice of a test type, we saw that using single-value-splits affects accuracy detrimentally. Thus, despite their efficiency, we prefer two-value-splits here. In this regard, we want to note that global illumination normalization has been shown to significantly improve results [8] and also bears potential to enable the use of single value splits for our case. Nevertheless, this requires costly pre-processing of training and test data, respectively.

Integrating all these findings brings up a detector which gives comparable results in terms of accuracy, while yielding a speed up of one to two orders of magnitude. Thanks to sub-sampling [75], as well as the reduction of the number of trees a comparable speed-up is also reached for the training procedure (see Tab. 3.7, and Fig. 3.8, resp.).

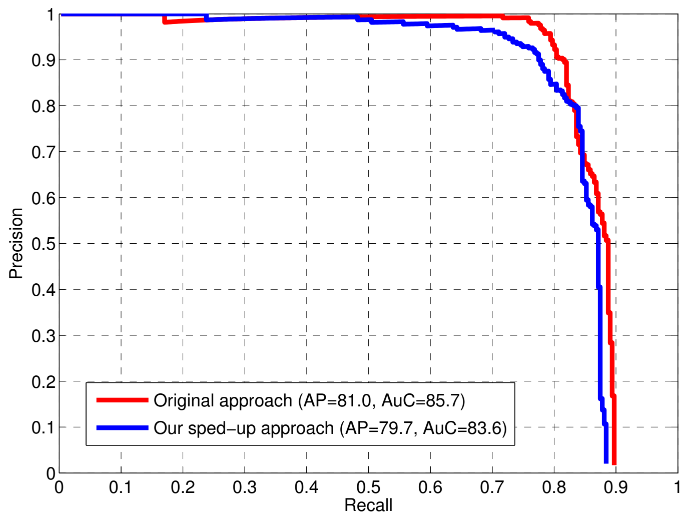


Figure 3.8: Precision-Recall curves for the original [35], and our sped-up approach; It can be seen that the loss in accuracy is small, while Tab. 3.7 shows that this is reached for a speed-up factor of 40.

Chapter 4

Tracking By Detection

Contents

In the previous section, we pointed out several enhancements towards the efficiency of Hough Forest based object detection. A task where efficient methods are essential is visual object tracking. The demand for efficiency is caused by runtime requirements as well as the significant amount of data to be processed. To this end, all adaptions introduced in the previous section are designed to be straightforwardly applied to object tracking in a tracking-by-detection manner. In this section, we empirically show that the previously introduced concepts can be used to build a multi-camera tracking system. Despite the multiplied amount of input data, this system is able to achieve acceptable runtime without sacrificing robustness.

To start with, the system is employed to track single instances from single cameras (see Sec. 4.1). In this way we can prove our concepts in a less complex setup, where less parameters influence and, thus, disturb the performance. Hence, we can draw more reliable conclusions focusing on the tracking-by-detection task, and subsequently find an efficient but still robust setup. Based on the gathered insights, in Sec. 4.2, the built system is finally extended for dealing with multiple instances and exploiting the information provided by multiple cameras.

4.1 Single Camera Tracking

A considerable amount of approaches for tracking using Random Forests has been developed (see *e.g.*, [33, 38, 45, 53, 73, 90]). Nearly a decade ago Random Forests were successfully applied to real time key point recognition [54], which is closely related to the subsequent tracking approaches. Thereby, a model is built up from a single, initial image of an object, by generating a number of randomly deformed and noised up variants of it. This process is well known, and has been termed *one-shot* learning. The mentioned approach [54], however, is not able for on-line updates, or handling significant changes in object appearance, since it is designed for key point recognition.

To overcome this, Saffari *et al.* [73] brought Random Forests to the on-line domain and applied them to tracking. In their tree growing scheme, incoming training samples are buffered in a node until a minimum number of samples arrived, and the information gain for a split exceeds a certain value. Decorrelation of the individual trees is, furthermore, enforced by bagging. Later, in [75], the authors show that it is unnecessary to keep track of the information gain for each newly arrived sample. Instead it is just enough to wait for a fixed number of samples to arrive. This can be regarded as a sub-sample of all data arriving in a node. They show that using sub-sampling, an additional amplification of the decorrelation between the trees by means of bagging becomes obsolete. In this regard, from the findings in Sec. 3.8 we know that the balance of a sub-sample used for optimization is crucial for the accuracy of the constructed trees. Nevertheless, during tracking, the distribution of the first samples arrived can neither be guaranteed to represent the true distribution, nor to be balanced. Hence, to overcome this issue, in our approach, we impose the sub-sample to be balanced, and optimize the split on this balanced sample. As experimental evaluations prove, this yields a robust on-line model, in spite of reduced complexity.

In the following we describe our approach to single camera tracking, and, subsequently, analyze its performance on standard tracking sequences. Thereby, we find a parameter setting using which stable tracking can be performed at the smallest possible computational cost.

4.1.1 Method

In our single camera tracking approach we follow the on-line formulation of Hough Forests, as introduced in [75]. Thus, we build an initial forest by one-shot learning, based on the first frame annotation. In the subsequent frames, the forest is evaluated within a search

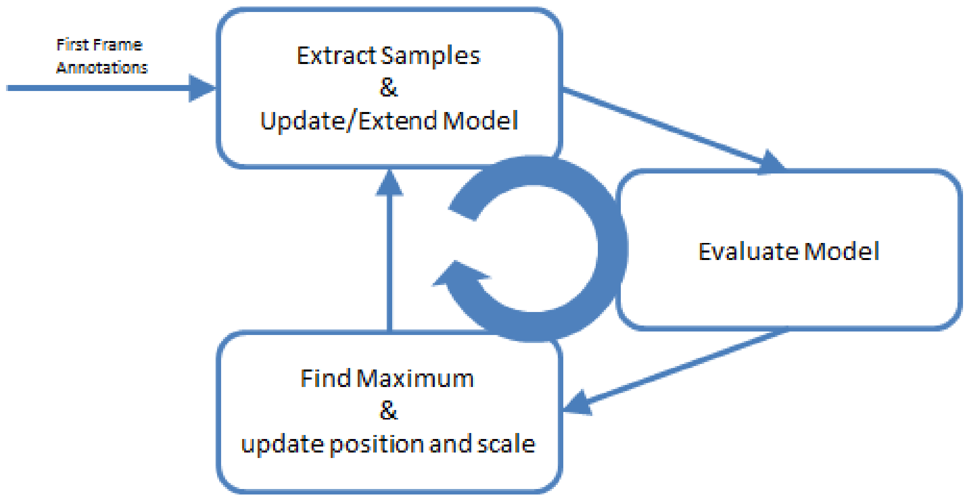

First Frame Annotations
Extract Samples & Update/Extend Model
Evaluate Model
Find Maximum & update position and scale

Figure 4.1: The single-camera tracking loop; illustrates the overall workflow of our approach to single-camera tracking.

region around the object location in the last frame. Thereby, we sample only two nearby scales, since the object scale change between two consecutive frames can be assumed to be small. The maximum of the resulting Hough space is found and the object location and scale is updated. See Fig. 4.1 for a general overview of the workflow.

The size of the peak in the Hough space gives an intuitive measurement for the confidence of the tracked position. As long as this confidence measure is above some predefined threshold – relative to the mean confidence of the first frames – the model is updated using randomly sampled patches. Positive patches are sampled from within the found object bounding box, whereas negatives are sampled from the neighboring region. Thereby, negatives are either randomly sampled, or bootstrapped by extracting them from the respective bounding box given by the highest peak in the neighboring region. We alternate from frame to frame between these two strategies for extraction of negatives.

In order to split a node Schulter *et al.* [75] propose to wait for a fixed number of samples n' to arrive, and optimize the split on those. To this end, we already learned that the balance of a sub-sample used for optimization is crucial for the accuracy of the constructed trees (see Sec. 3.8). Nevertheless, the first n' arrived samples can not be guaranteed to be balanced over the classes. Moreover, for small n' – which is the usual case – the optimization may even be strongly disturbed, since the class distribution over these samples can represent virtually any distribution. Hence, it may not only be a bad estimate for the true distribution, but even be completely degenerated. To overcome this issue, we adapt the sub-sampling strategy for on-line use by simply drawing a balanced

	david	faceocc2	girl	tiger1	mean
Proposed ($\delta = 1; F = 32$)	**9**	**14**	*20*	**6**	**12.3**
HT [38]	**9**	*20*	38	35	25.5
MIL [4]	23	*20*	32	*15*	*22.5*
ORF [73]	16	29	**16**	48	27.3

Table 4.1: Mean center distance on several standard tracking sequences for our method, compared to state-of-the-art tracking-by-detection approaches; best scoring method bold-faced, second best in italics.

sub-sample from the arrived samples. Thus, we specify a number of samples $m : mC < n'$, which defines the size of the sub-sample for each class. In other words, we wait for m samples to arrive for each class, unless an overall number of n' samples arrived from all classes. Subsequently, the splitting test is optimized on the drawn mC samples.

4.1.2 Experiments

For our experiments we start with a setting were we use five trees trained to a maximum depth of eight. Furthermore, the unit scale s_u for an object is set to 80 pixels. In order to have a baseline for comparison, we start with an experiment were we use all 32 feature channels, as well as dense sampling during detection as proposed in the original Hough Forest formulation. We evaluate using several commonly used video sequences. Each of them is publicly available*. As evaluation criterion the mean center distance is employed, since we only have the object center annotations available for these sequences. Nevertheless, it provides us with an accurate measurement instead of a binary decision as for evaluations based on the number of correctly tracked frames. We compare to Hough-Ferns (HT) [38], multiple instance learning (MIL) [4] and on-line Random Forests (ORF) [73]. The detailed results, shown in Tab. 4.1, prove that our method is on par with state-of-the-art tracking-by-detection approaches.

Measuring the runtime separately for each individual part of the workflow, we immediately see that there are two main parts demanding nearly the whole computation time: feature computation and forest evaluation (see the leftmost bar of Fig. 4.2). Hence, the opportunities for achieving a runtime gain are virtually broken down to two remaining parameters. Thus, in the following experiments we aim at reducing the computational effort for these two by reducing the number of feature channels F to be computed, and increasing the sampling distance δ. As one can readily see from the second bar in Fig. 4.2,

*`http://vision.ucsd.edu/~bbabenko/project_miltrack.shtml`

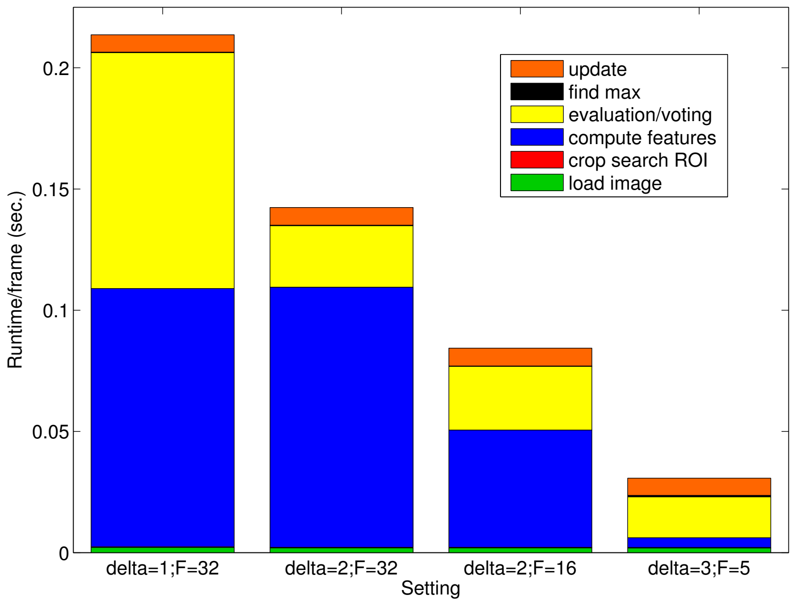


Figure 4.2: Partition of runtime for different settings of sampling distance δ (*delta*), and number of used feature channels F; δ is fixed to the same value in x-, and y-direction; $F = 16$: all channels, but no min-, and max-filtration used; $F = 5$: only *Lab*, and first order derivatives used.

solely changing δ gives a significant speed up when increasing it from one to two, but increasing it further can only provide a slight additional speed-up. This is because feature computation is now the – by far – dominating part of the overall runtime. When discarding the min- and max-filtration, and thus, reducing F to 16, by concurrently setting $\delta = 2$, tracking becomes slightly less robust but is still successful in most cases (see Tab. 4.2). Furthermore, by increasing δ to three, and restricting the model to color and first order gradient information, only, *i.e.*, to use five channels, the tracker is still able to handle plenty of the challenges contained in the standard sequences, like partly occlusions, illumination, and scale changes, or moderate appearance changes. Nevertheless, robustness under hard conditions, *e.g.*, nearly full occlusions, or self occlusions, is lost. See Tab. 4.2, and Fig. 4.2 for the results in terms of mean center distance, and the runtime partition, respectively. Fig. 4.3 shows some images from one of the sequences overlaid with the results, and Fig. 4.4 illustrates the relation between tracking error and confidence of the tracker.

The results from this section suggest that the runtime may be significantly reduced by focusing on two parameters: The size of the feature space and the number of sampled patches. In fact, by carefully selecting the feature channels as well as the number of sampled patches to be fed into the forest model, we are still able to perform robust

	david	faceocc2	girl	tiger1	msec./frame
$\delta = 1; F = 32$	9	14	20	6	208
$\delta = 2; F = 16$	11	31	21	43	85
$\delta = 3; F = 5$	13	17	28	67	32

Table 4.2: Mean center distance on several standard tracking sequences for our method, when reducing its complexity, *i.e.*, increasing the sampling distance δ, and reducing the number of feature channels F, respectively; rightmost column: runtime of our approach on a single dual-core CPU.

tracking. Depending on the difficulty of the task we are now able to select an appropriate setting, running at up to more than 30 frames per second on single dual-core CPU.

4.2 Multiple Camera Tracking

When dealing with multiple interacting instances, occlusions are likely and commonly raise problems for visual detection and tracking algorithms. Even though, recently, progress has been made towards detection of partially occluded objects from a single image (see *e.g.*, [6, 88]), situations arise where occlusions can not be resolved anymore. In some situations it is virtually impossible to detect and discriminate between two or more object instances from a single view and given point in time. For example, when one person hides others (see, *e.g.*, Fig. 4.5). One way to resolve those situations is to analyze the possible tracks for each instance over several frames [3, 22, 92] and find the best track, *e.g.*, by integrating constraints like track smoothness [12], or spatial separation of individual detections and tracks [59]. However, long-term occlusions, or occlusions appearing while tracked people are stopping and interacting still introduce ambiguities. A natural solution to circumvent those, is to use multiple cameras observing the same scene [24, 47, 67, 72, 91]. While this bears the potential to resolve the discussed problems, the multiplied amount of data requires extremely efficient algorithms. Thus, in the following we apply the concepts and gathered insights, discussed in the previous sections, to the task of tracking multiple instances from multiple cameras.

For multi-camera tracking many of the previously proposed approaches learn a background model and identify every deviation as foreground [29, 47, 67]. In this way, they are unable to distinguish objects of the searched class from other moving objects. Some also rely on calibrated cameras [9, 24, 67]. Hence, they require special calibration effort and are not applicable to data from arbitrary scenes. Learned classifiers provide a possibility to overcome the problem of distinguishing between different foreground objects.

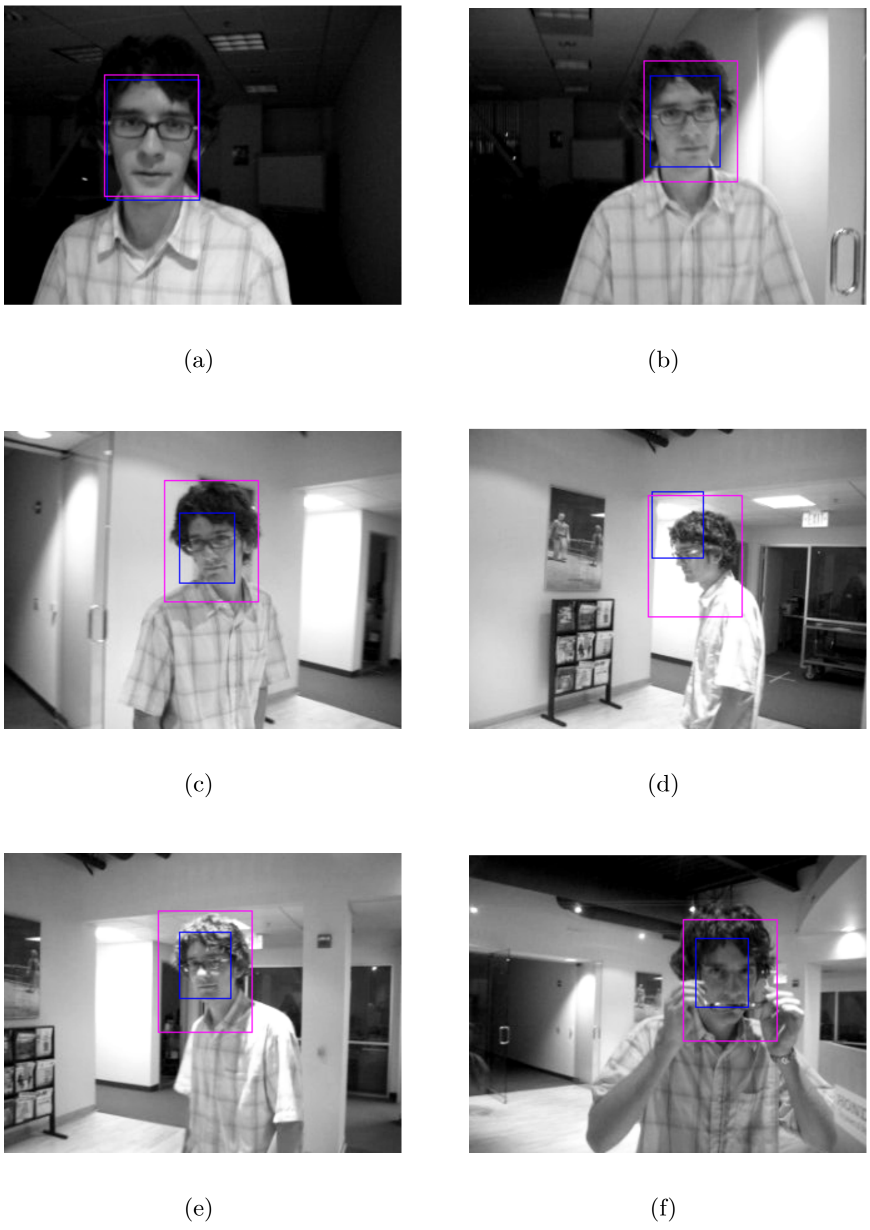


Figure 4.3: Tracking results (blue) and ground truth (pink) for frame 11 (a), 87 (b), 129 (c), 162 (d), 181 (e), and 388 (f) of the "david" sequence; ground truth annotations are solely given at a single scale.

In this regard, either fixed pre-trained [9], or on-line updated classifiers are used [72, 81]. Fixed classifiers, on the one hand, suffer from the inherent inability to discriminate the individual object instances in the scene. On-line approaches, on the other hand, usually train a separate classifier on each view, which neglects to comprehensively exploit the

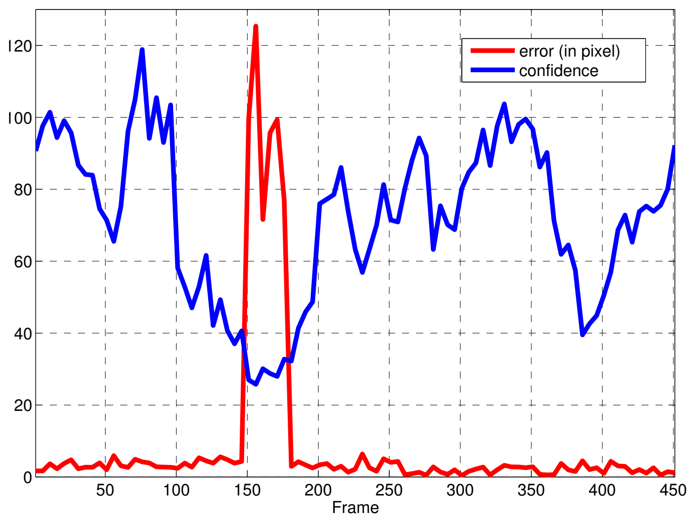


Figure 4.4: Center distance (*i.e.*, error) in pixel (red) and confidence (blue) as a function of time for the "david" sequence. Error and confidence are inversely correlated to some extend. Note, the normalization factor for the confidence is not known, and thus, it is just scaled to approx. fit the range of the error. The tracked face turns away from the camera between frame 150 and 180 (see also Fig. 4.3).

information given by the different views of the same scene, and objects, respectively. For example, after turning, a person will appear different in one view, but, its appearance in another view may be similar to its appearance in the first view before the turn, and vice versa. This real world constraint can be employed to learn stronger models, but is ignored by using a separate classifier for each view.

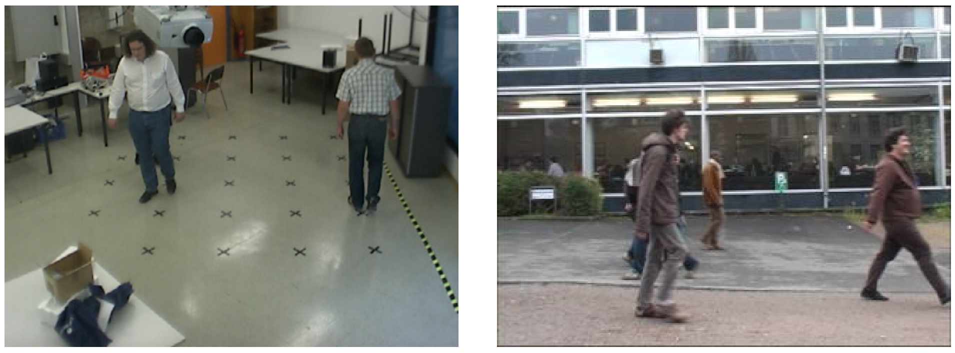


Figure 4.5: Example images for strongly overlapping object instances; Left image: the person in the white shirt completely hides another one – only small parts of the shoes are visible (taken from *Set 1* of the Medium dataset [72]); Right image: at least three persons are hid behind the person with the backpack on the left (taken from the *TUD-Campus* sequence [2]).

The approach which is most related to ours is the one of Sternig *et al.* [81]. They propose to use an off-line trained, Hough Forest based person detector for each view, where, similar to [33], the instance statistics at the leafs are updated on-line. Instead of voting for the object center, the votes are cast for the location of the foot, and projected to a common reference plane. Subsequently, a particle filter approach is used for tracking on the common plane. While, in their approach, the discrimination between instances is handed over to the particle filter, in contrast, we exploit the redundant information from all views to build strong models of the instances. To this end, we do not only update the leaf statistics, but effectively extend the pre-trained model, enforcing discrimination between the instances within the updated model. Hence, our system is able to distinguish them based on their appearance instead of solely relying on the predictions of a motion model.

4.2.1 Method

In general, a multi-camera setup consists of several cameras observing the same scene. For a fully calibrated setup each image point from each camera view can be related to a ray in world coordinates, and thus, to a line in each other view [43]. However, for tracking applications the objects are often exclusively moving on a common ground plane. This constraint can be exploited in order to get rid of the necessity for calibrated cameras.

Multi-Camera Hough Voting

Given the precondition, that all objects to be tracked are moving on a plane, a planar homography can be utilized to map an image point to the corresponding point on a common plane. Formally, given an image point $\mathbf{p}$, and the homography $\mathbf{H}$, the corresponding point $\mathbf{p}'$ on the common plane can be computed by

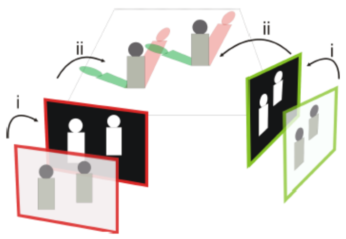
$$\mathbf{p}' = \mathbf{H}\mathbf{p}, \tag{4.1}$$

where $\mathbf{p}$ and $\mathbf{p}'$ are homogeneous three-dimensional vectors, and $\mathbf{H}$ is a homogeneous 3×3 matrix, specifying the projective transformation between the two planes. To obtain 2D coordinates, the resulting vector $\mathbf{p}'$ is normalized by its third component, *i.e.*, $\tilde{\mathbf{p}'} = \left(\mathbf{p}'_1/\mathbf{p}'_3,\, \mathbf{p}'_2/\mathbf{p}'_3,\, \mathbf{p}'_3/\mathbf{p}'_3\right)^\top$. The transformation is defined up to scale, consequently, it has only eight degrees of freedom [43].

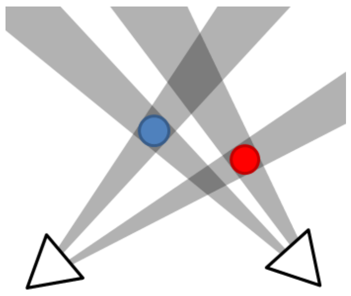
Even though, such a *ground plane assumption* may be easily invalidated by *out-of-plane* motions, *i.e.*, if an object doesn't move on the ground plane, its advantage is that it can be applied to arbitrary scenes, requiring no more input than the raw images. Likewise, it is also employed in order to avoid special calibration effort, which is necessary for a fully calibrated camera setup. In fact, planar homographies can easily be estimated by extracting interest points from each view (*e.g.*, SIFT points [56]), and estimating the homography from one view to each other, *e.g.*, by utilizing RANSAC [27].

A major limitation for approaches which are basing on the ground plane assumption is that the computed projective transformation is only valid for points lying on the ground plane. This causes unreliable projections for other points, and may results in ghost projections which need to be explicitly handled (see Fig. 4.6(a)). In order to overcome these limitations, 3D reconstructions based on calibrated cameras are employed [41, 67]. However, also these approaches suffer from ghost detections in special situations (see Fig. 4.6(b)). Even worse – such situations are more likely the higher the number of observed objects is, since it then becomes increasingly difficult to carve out unoccupied regions. Furthermore, this problem is even aggravated for robust schemes, which do not strictly require silhouettes to be observed in every view [41]. To overcome the problem of ghost detections without requiring calibrated cameras, we follow the approach of [81]. It is based on the previously introduced Hough Forests (see Sec. 2.2). They propose to cast votes for the foot-points of persons. Subsequently, based on the ground plane assumption, these votes can be reliably projected to the common plane and fused therein (see Fig. 4.6(c)).

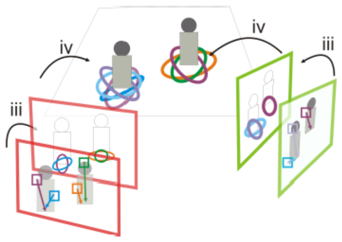
This approach has several advantages. First, only points actually lying on the ground plane are projected to the common plane and, thus, no unreliable projections need to be handled in the sequel. Furthermore, by mapping the foreground likelihood directly to the common ground plane, the information from all views can easily be fused. In this way, the uncertainty for the foot-point prediction as well as the uncertainty in the projection is expressed, while still preserving robustness to occlusions [81]. Nevertheless, for various applications it was shown that the performance of Hough voting schemes decreases with the length of offsets [16, 21, 79]. Hence, we have to aim at using as short offsets as possible. Voting for the foot-point, though, would mean to use offsets up to twice as long as for center-voting. To this end, we slightly adapt their ideas. We follow the original formulation [32] and train our forest for center voting. However, at detection the cast votes are offset, so that they actually vote for the foot-point. For that purpose, the offset from center to foot-point is computed based on the scale of the detected object. Hence,



(a) Background subtraction based approach: Foreground pixels are projected to the common plane, causing ghost detections by projecting points not actually lying on the ground plane



(b) 3D reconstruction based approach: Relying on calibrated cameras, the foreground regions are used to reconstruct the visual hull. Thereby, unoccupied regions also happen to lie inside the silhouettes, and thus, cause ghost detections



(c) Joint Hough voting based approach: Votes are cast for the foot-points of the persons, and only then, projected to the common plane. Hence, uncertainties are implicitly considered by projecting the actual Hough votes.

Figure 4.6: Comparison between common approaches to multi-camera tracking (a) based on background subtraction, (b) 3D reconstruction, and (c) the approach based on Hough voting onto a common ground plane, which we are following. Images (a), and (c) are taken from [81].

we can now vote for the foot-points of the persons, without increasing the uncertainty in the voting process due to artificially elongated votes.

Strong Adaptive Models

One-shot trackers usually face problems for strong appearance changes of the target object. For example, with a simple turn, a persons appearance changes, *e.g.*, from front to side view. For a one-shot tracker trained on a single view this may already causes severe problems (see *e.g.*, Fig. 4.7). Nevertheless, for a task where the object class is known in advance this problem can easily be overcome by employing a pre-trained classifier. In our case, the goal is to track persons moving on a common ground plane. Thus, we can pre-train a generic person detector off-line. Nevertheless, we want to integrate instance discrimination into the model, and therefore, the pre-trained detector needs to be adapted towards this end. This could be done in advance, however, requiring a massive labeling effort. Hence, for our case, the model is updated on-line.

An important issue towards separation of the instances are the splitting tests of the Hough Forest. Obviously, the splitting tests of a generic person detector do not appear to be very discriminative when aiming at separation of specific instances. For example,

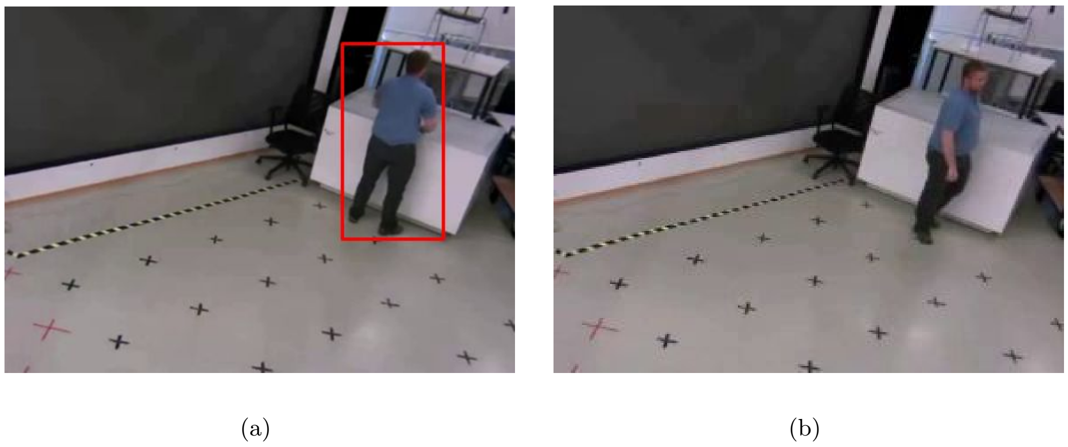


Figure 4.7: Illustration of a problem case for a one shot tracker: The person remains at the position of the first frame annotation (a) while working at the table for some time; (b) when the person finally walks away and thus, its appearance changes, the tracker tends to stick to the background, which exhibits several distinctive image features, to which the part based model has been adapted.

color information (*i.e.*, the color channels) is not intensively used by a generic forest classifier, since it is not well suited for distinguishing between foreground and background. However, for instance separation this is apparently one of the most important cues. Hence, a generic person detector is not an appropriate choice for separation of specific instances. Nevertheless, a Hough Forest – being a part based detector – trained for person detection is able to segregate the individual parts an object consists of. For example, for person detection, parts like the feet, the head, or the torso are clustered together. The color of each part is highly informative when it comes to discrimination between object instances. Thus, a straightforward way to differentiate between individual instances is to apply a few additional splitting tests based on the existent clustering of a pre-trained forest. To this end, we restrict the optimization (*c.f.*, Eq. (2.11)) solely to the instance samples, when selecting the best split at a node. Figure 4.8 illustrates the importance of color information for the splits, which are added to the pre-trained forest.

The data samples extracted from the detected instances are traversed down the trees reaching the respective leafs. Analogous to the process described for single cam tracking, we collect the samples together with their instance specific offsets at the leafs until a fixed number of samples m arrived from each instance. Subsequently, a split is optimized on $m\,|E|$ instance samples, where $|E|$ denotes the number of instances. To ensure the samples to be balanced over the instances, we draw a sub-sample from all instance samples arrived

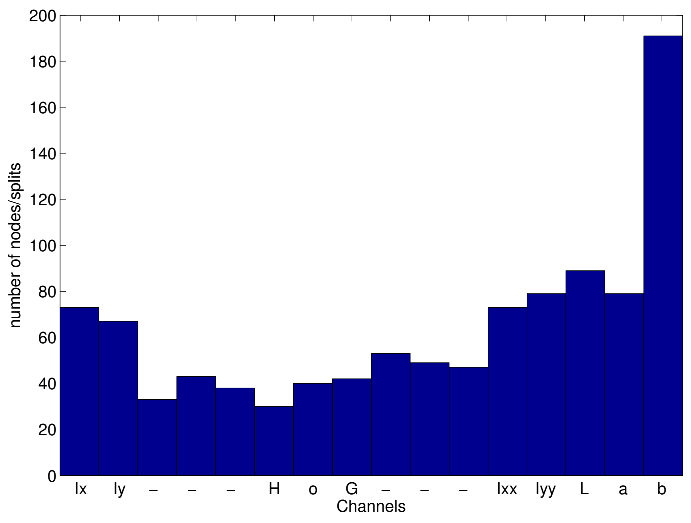


Figure 4.8: Histogram over the feature channels selected for model extension during tracking. The b-channel is by far most important to separate the instances. For computation of the histogram we used the *Set 1* sequence from the Medium dataset [72]. See Fig. 4.12–4.14 for some example images.

at the leaf. Considering the issues regarding over-fitting, mentioned in Sec. 3.3, this process is repeated only for a small number of additional depth levels. Furthermore, in order to keep the good generalization performance, we keep the samples from the pre-trained classifier, split them according to the selected splitting tests, and use them to build reliable statistics at the newly created leafs. Nevertheless, since only very few additional splits are made, the samples doesn't have to be kept forever, which would obviously require infinite memory and hamper runtime. Instead, after the defined number of splits are made, we keep the tree depth fixed and discard all collected samples. Subsequently, only the class-, instance-, and offset-statistics are updated.

During detection, we are interested in $p(\mathbf{h}(E, y, \mathbf{u}, s)|\mathbf{x})$, where $\mathbf{h}(E, y, \mathbf{u}, s)$ denotes the hypothesis for an object instance E, which is from class y, and centered at $\mathbf{u}$, with scale s. Similar to Eq. (3.6), for a single data sample $\mathbf{x_v}$ and tree t we get

$$p_t(\mathbf{h}(E, y, \mathbf{u}, s)|\mathbf{x_v}) = \sum_{\mathbf{c}} \left(\mathbf{1_c}\left(\frac{s_u(\mathbf{u}-\mathbf{v})}{s}\right) w_{\mathbf{c}} \right) p_t(E|y, \mathbf{x_v})\, p_t(y|\mathbf{x_v}), \tag{4.2}$$

where $\mathbf{1}$ again denotes the indicator function, and $p_t(E|y, \mathbf{x_v})$ is the instance probability, which is estimated from the ratio of samples from instance E to all samples from the corresponding class y arrived at the leaf l_t during on-line updating. Furthermore, the weight for a grid cell center $w_{\mathbf{c}}$ is now computed based on all pre-trained class offsets $\mathcal{O}_{l_t}^{(y)}$

and the instance specific offsets $\mathcal{O}_{l_t}^{(E)}$ supporting it:

$$w_{\mathbf{c}} = \frac{1}{\left|\mathcal{O}_{l_t}^{(y)}\right| + \left|\mathcal{O}_{l_t}^{(E)}\right|} \sum_{\mathbf{o} \in \mathcal{O}_{l_t}^{(y)} \cup \mathcal{O}_{l_t}^{(E)}} \mathbf{1}_{\mathbf{c}}\big(\chi(\mathbf{o})\big), \tag{4.3}$$

where $\chi(\mathbf{o})$ shall again give the cell center closest to the offset vector $\mathbf{o}$:

$$\chi(\mathbf{o}) = \underset{\mathbf{c}}{\operatorname{argmin}} \left\|\mathbf{o} - \mathbf{c}\right\|_2 . \tag{4.4}$$

As one can see, the confidence $p\big(\mathbf{h}(E, y, \mathbf{u}, s)|\mathbf{x}\big)$ is separately evaluated for each instance. Hence, at detection time, a separate Hough space needs to be set up for each instance.

When simultaneously observing an object from multiple viewpoints the appearance of the object is different in each view. For tracking, the appearance of a target object in a single view will change over time when it is moving. Similarly, this holds for all other views. Since it is still the same object, at some point, its appearance in one view may be very similar to the appearance previously observed in some other view. This real world constraint can be exploited by employing a single comprehensive classifier, simultaneously updated from all views, instead of training a separate classifier for each view.

Random Forests are able to handle a huge amount of data and a significant amount of noise [7, 34, 66], which could be introduced by the large variability in the data from different views. Hence, together with our efficiency-raising adaptions, Random Forests are a good choice for the task of multi-camera, multi-instance tracking.

Overall Workflow

We now summarize the whole process: A generic person detector is trained as described in Section 2.2. At the first frame the detector is updated with samples from each object instance in the scene. These are extracted in a one-shot manner, either from the first frame annotations, or alternatively, from the objects detected by the initial model. The model is subsequently evaluated in each frame, the votes are projected to the foot-point, and further to the common ground-plane. Therein, the votes from all views are accumulated and the maximum is found. This is back-projected to the individual views, where the object position is updated accordingly. The object scale on the other hand is solely estimated based on the information from the specific camera. From the updated object bounding boxes new samples are extracted for each instance from all views and feed into the model. To this end, not only the statistics at the leafs are updated, but the model is extended

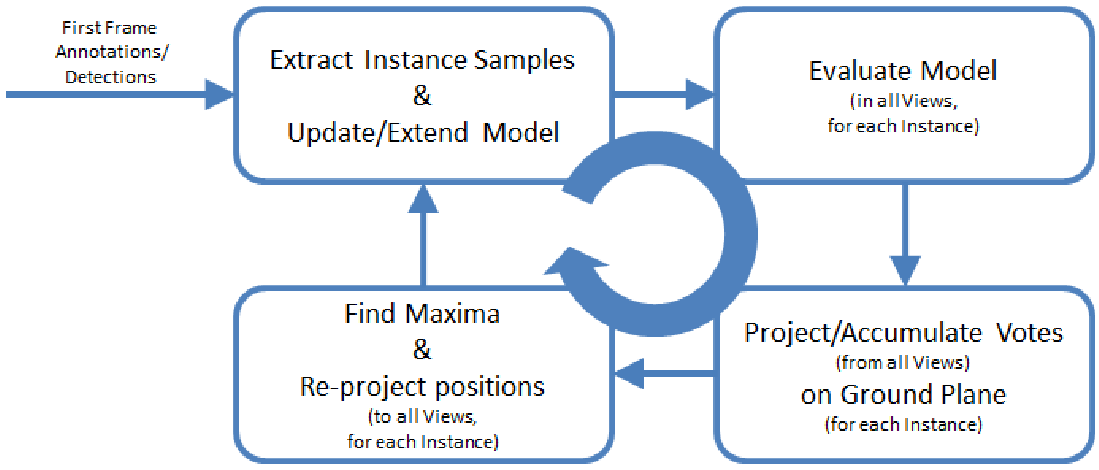

First Frame Annotations/ Detections
Extract Instance Samples & Update/Extend Model
Evaluate Model (in all Views, for each Instance)
Project/Accumulate Votes (from all Views) on Ground Plane (for each Instance)
Find Maxima & Re-project positions (to all Views, for each Instance)

Figure 4.9: The multi-camera tracking loop; illustrates the overall workflow of our approach to multi-camera tracking.

towards instance separation. See Fig. 4.9 for an overview of our approach to multi-camera tracking.

4.2.2 Experiments

In the following, we describe our experiments on a dataset for multi-camera object tracking. In order to be able to directly compare our method to the baseline of Sternig *et al.* [81], we utilize the same publicly available dataset [72], together with their ground truth annotations, kindly provided by the authors of [81]. The dataset, denoted *Set 1*, shows an indoor scene, with three persons walking around, whereby they are regularly occluding each other. The scene is captured by three different cameras and the whole sequence consists of more than 2500 frames. For quantitative evaluation every tenth frame has been annotated.

For the experiment we pre-trained a Hough Forest on the publicly available INRIA person dataset[†]. The forest contains 5 trees, and was trained down to a maximum depth of 17. We used sub-sampling already during off-line training and restricted the splitting tests to two-value-splits. For performance reasons, we sampled every second pixel (*i.e.*, $\delta = 2$), discarded the min-, and max-filtration (*i.e.*, $F = 16$), and rescaled the objects to an unit scale $s_u = 75$. Similar to our approach for single camera tracking, we updated the forest using positive and negative data samples, as long as the confidence exhibited some minimum value in the given view. During updating we use our on-line sub-sampling

[†]`http://pascal.inrialpes.fr/data/human/`

Method	Error (in pixel)
Sternig *et al.* [81]	23.9
MultiInstanceHF ($\delta = 2; F = 16$)	**18.8**
MultiInstanceHF ($\delta = 3; F = 5$)	152.2

Table 4.3: Mean error in pixel on ground plane for different settings of our approach (*MultiInstanceHF*) and the related work on *Set 1*.

strategy, introduced in Sec. 4.1, in order to extend the pre-trained trees for a maximum of three additional depth levels.

For our evaluation, we omit the first part were no person, or not all persons are visible. For the remaining sequence of about 1500 frames, we computed the pixel errors on the ground plane. In Tab. 4.3 the averaged error for our approach is compared to the one given in [81]. It shows that our single comprehensive classifier (*MultiInstanceHF* ($\delta = 2; F = 16$)), adapted for instance discrimination, is able to outperform their model, without relying on the results of an additional particle filter. The corresponding errors for each frame are shown in Fig. 4.10, whereas illustrative results are given in Fig. 4.12–4.14.

As for the evaluation in Sec. 4.1, we compare to a setting where we use only color and first order gradient information as well as a higher sampling distance. Specifically, only 5 instead of 16 feature channels, and a sampling distance of three is used (*MultiInstanceHF*

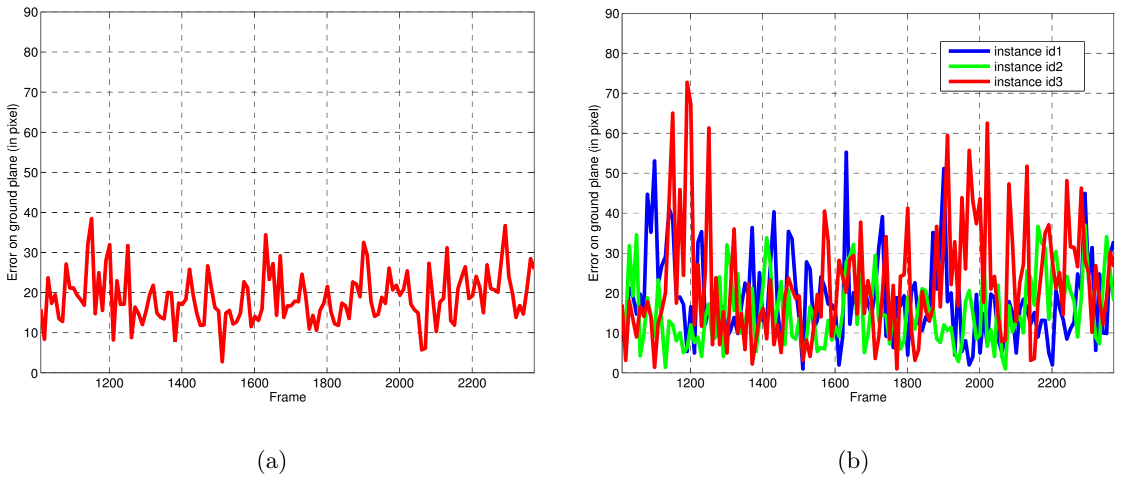


Figure 4.10: Error in pixel on the ground plane over time: (a) Mean error over the instances (to be compared to the plot given in [81]), and (b) broken down for each instance. See Fig. 4.15 for further details on the problem around frame 1200. The results were generated by setting $F = 16$ and $\delta = 2$.

($\delta = 3; F = 5$)). For single cam tracking this resulted in an additional runtime drop by more than 60 percent (see Sec. 4.1). In contrast, for tracking with multiple cameras, the additional runtime gain is only about 20 percent. The reason for this relatively small gain is that the projection of the Hough votes from each camera to the ground plane is the dominating part, taking more than 70 percent of the overall runtime. In fact, without the efficiency enhancements introduced in this work (see Sec. 3) the overall runtime is about 13 seconds per frame on a single dual core machine. Approximately 4 of these 13 seconds (*i.e.*, ∼30%) are spent on the projection of the votes from each camera to the common plane. For our sped-up approach (*MultiInstanceHF ($\delta = 2; F = 16$)*) the runtime is still 5.5 seconds, but, since the projection alone accounts for 4 seconds, the detection part has been reduced from nearly 70% to less than 30%. For the further reduced setting (*MultiInstanceHF ($\delta = 3; F = 5$)*) the runtime is brought down to 4.4 seconds, *i.e.*, the projection part takes even 90% of the time. This shows that for addressing the detection speed the maximum possible runtime gain is nearly reached. Hence, for an additional, significant runtime gain we would have to address the fusion of information from the individual cameras instead of the detection part.

Nevertheless, we can see that relying on color and first order gradient features may not be enough to discriminate the instances. The reduced setting (*MultiInstanceHF ($\delta = 3; F = 5$)*) is not able to correctly distinguish between similarly dressed instances in special cases. For our evaluation the detector fails to discriminate *instance id2* and *id3*, which both wear blue jeans, a mostly white shirt, and black shoes (see Tab. 4.3 and Fig. 4.11).

While the more complex tracker (*MultiInstanceHF ($\delta = 2; F = 16$)*) is usually able to follow the instances, it suffers in situations where the foot point of a person cannot be reliably estimated from all three cameras. An example is shown in Fig. 4.15. In this case, the foot points, which provide relevant information for the tracker are not visible in two of the three cameras. As a result, the tracker is stuck behind the person, and recovers only a few frames later. This happens despite the fact that the foot point is visible in one of the camera views. However, in the specific case, the tracker is most confident for a view where most of the person is visible except for the foot point. Thus, the tracker fails to correctly localize the foot point in this view. But, due to the high confidence, the position proposal from this view still wins against the proposal from the view where the foot point is indeed visible. While, the reason for the lower confidence in that view is not completely obvious, we hypothesize that the specific instance appearance isn't sufficiently modeled by our forest up to this point.

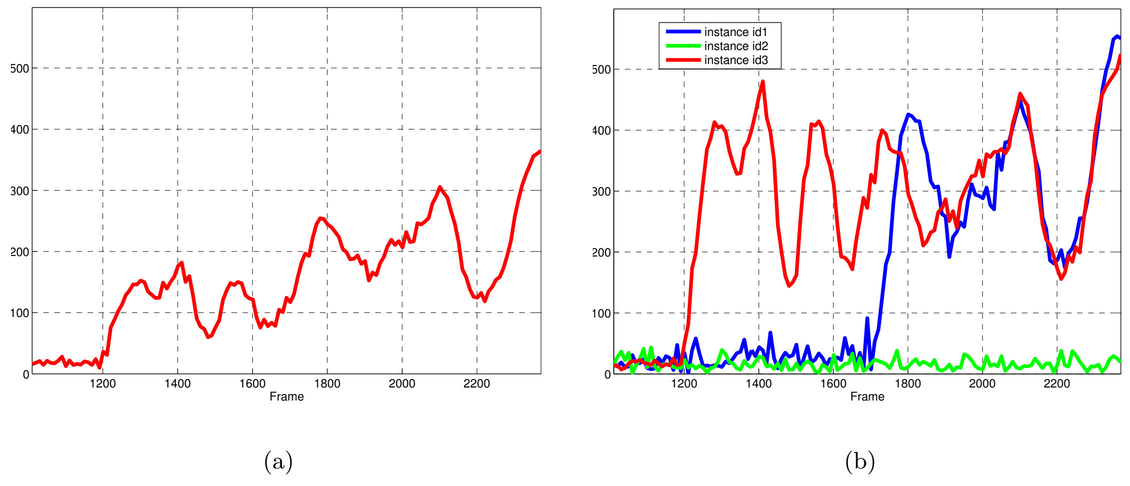
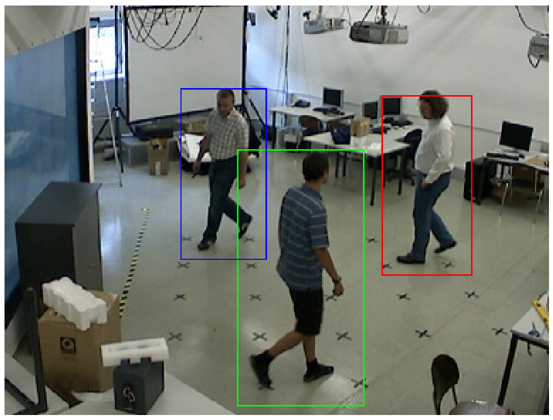


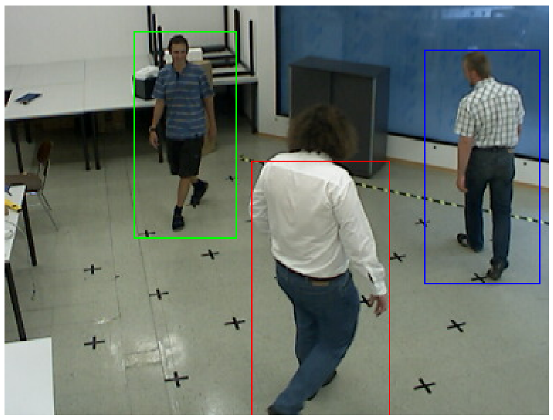
Figure 4.11: Error in pixel on the ground plane over time: (a) Mean error over the instances, and (b) broken down for each instance. The results were generated by setting $F = 5$ and $\delta = 3$. The "red tracker" falsely coalesces with the "blue instance" after frame 1200, whereas later the labels are even switched. Note, the different scale on the y-axis when comparing to Fig. 4.10.

Another problem is robust estimation of the scale[‡]. The individual trackers are only enforced to agree on the position estimation. While, this information – gathered from all views – is fused on the common plane, the scale has to be estimated solely based on a single view. The left and middle image in the top row of Fig. 4.15 shows examples, where the scale estimation is imperfect. The position is usually reliably determined despite occlusions in specific views, but not the object scale. In situations, where an instance is occluded in a view, a scale still needs to be estimated for that view in order to be able to resume tracking after the occlusion disappeared. However, the scale estimation in such cases may be nearly arbitrarily wrong. To this end, one may simply fix the scale to what it was before the confidence dropped below some predefined threshold. This is sane, because, in cases where not enough parts of an object are visible – which is indicated by low confidence – no scale should be estimated. Nevertheless, the scale of an object may change during occlusion. Without any additional information about the relative positions of the cameras, we are unable to detect that. However, in our experiments this problem doesn't significantly affect the tracking accuracy on the ground plane. This is because in the employed dataset an object is always visible in at least one of the cameras. Thus,

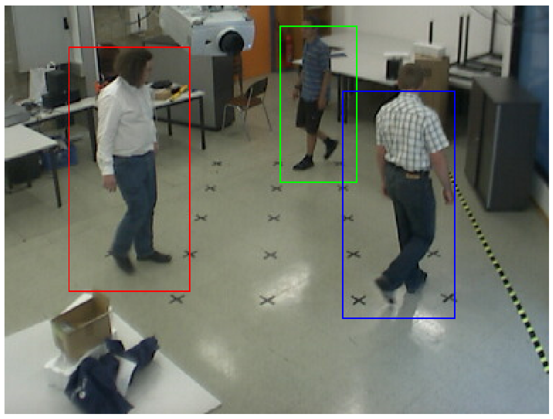
[‡]From our experiments it seems that the scale estimation suffers most from the reduced number of trees in the forest.



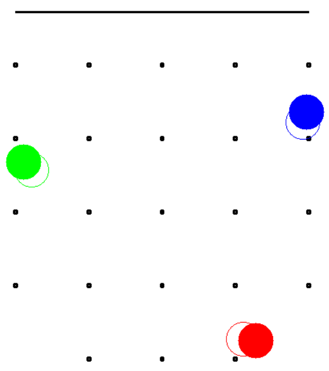
(a)



(b)



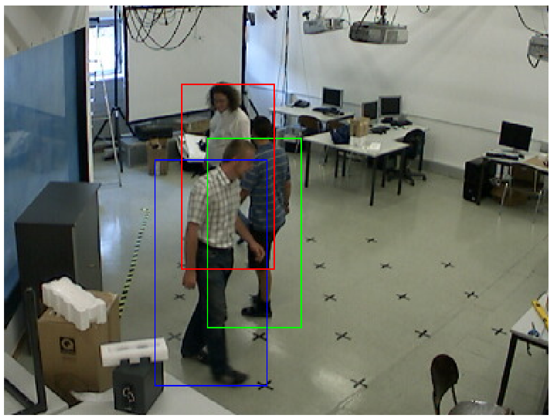
(c)

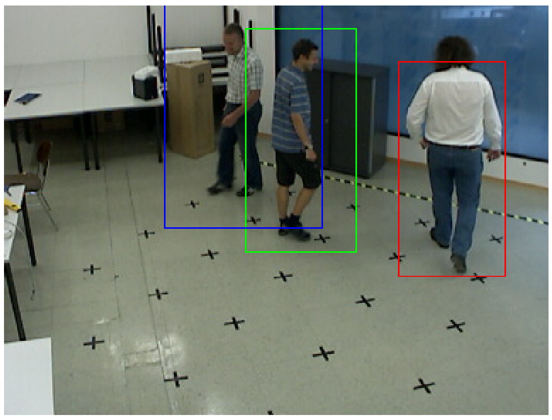


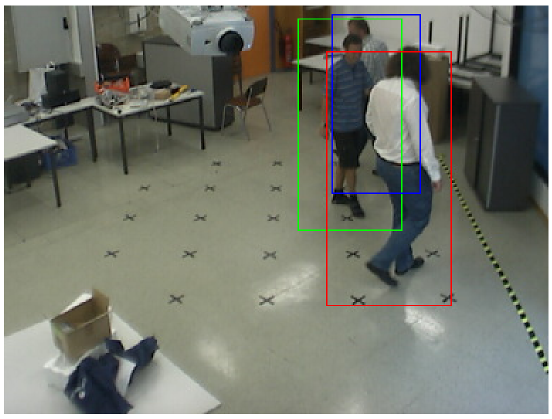
(d)

Figure 4.12: Illustrative Tracking results on *Set 1* overlaid on the camera views (a-c), and the ground plane (d), where the filled circles represent the tracking result and the unfilled the ground truth annotation; colors correspond to instances.

the ground plane position can be reliably estimated from this view, and the correct scale could usually be recovered after occlusion.


(a)


(b)


(c)

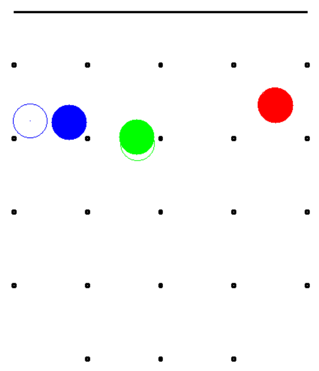

(d)

Figure 4.13: Illustrative Tracking results on *Set 1* overlaid on the camera views (a-c), and the ground plane (d), where the filled circles represent the tracking result and the unfilled the ground truth annotation; colors correspond to instances.

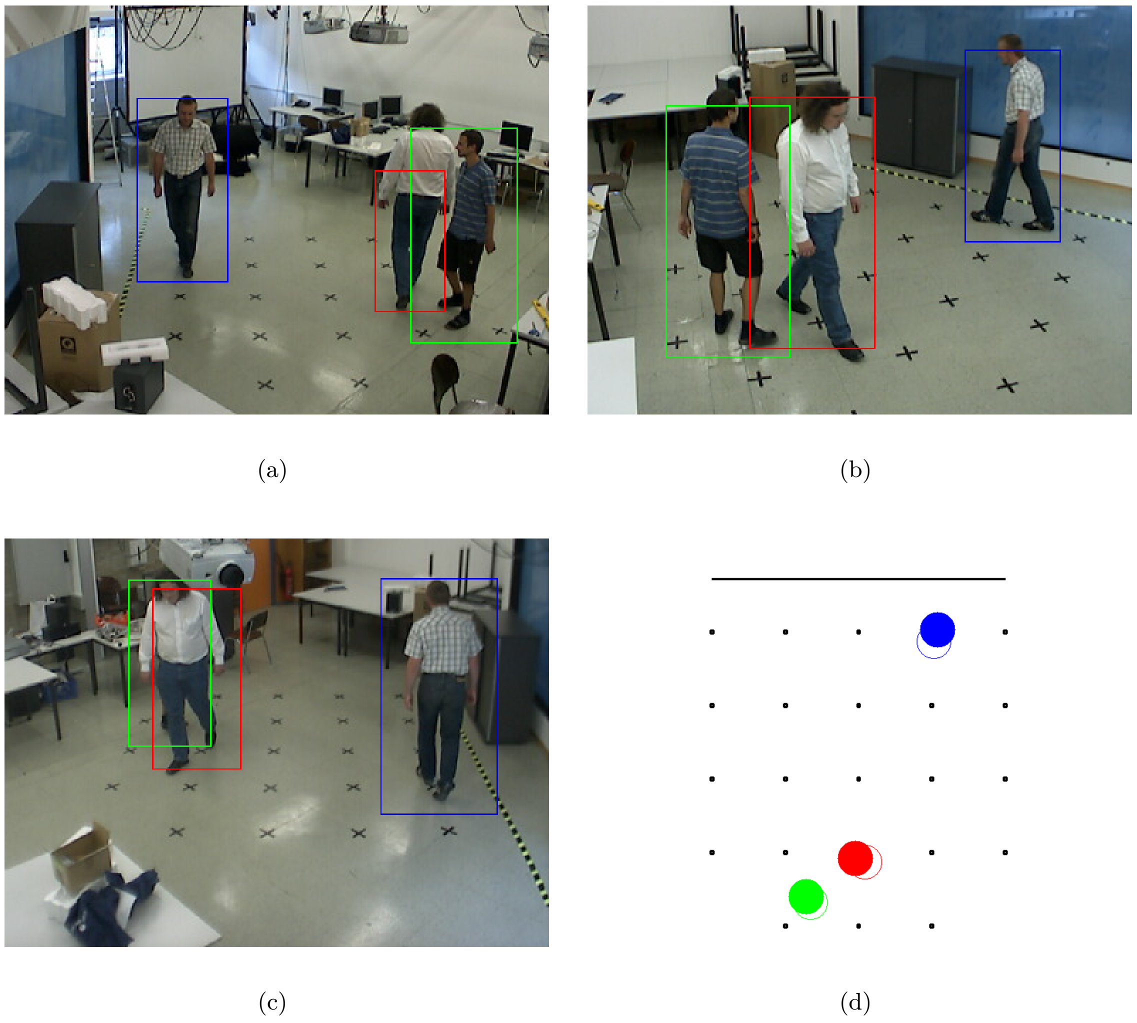


Figure 4.14: Illustrative Tracking results on *Set 1* overlaid on the camera views (a-c), and the ground plane (d), where the filled circles represent the tracking result and the unfilled the ground truth annotation; colors correspond to instances.

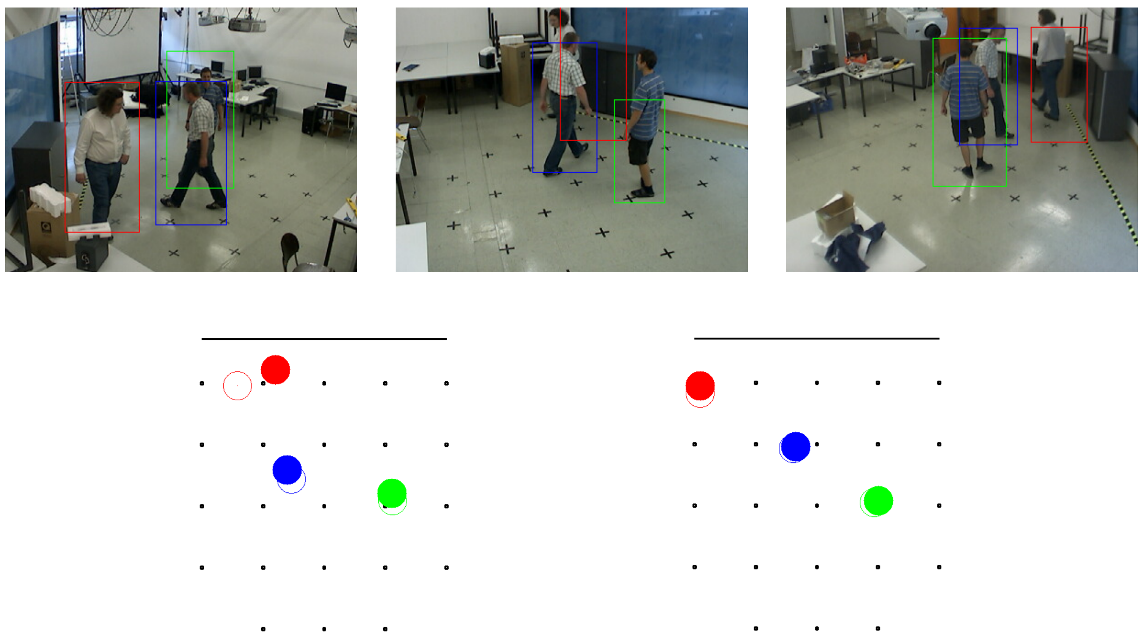


Figure 4.15: Problem case where the multi-cam tracker can't follow one of the instances; top row shows frame 1201 for each of the views, overlaid with the resulting bounding boxes; colors correspond to the instances; the "red" tracker is stuck behind its instance (see text for discussion); bottom row shows illustrations of the ground plane (for orientation reasons, black crosses and the yellow-black dashed line in the scene are here visualized by black dots and a solid line, respectively); the ground plane is overlaid with the tracking result (filled circles), and the respective ground truth locations (unfilled circles); ground plane results are given for frame 1201 (bottom left), and the next annotated frame 1211 (bottom right), where the tracker has already recovered from failure; another problem which can be seen in the left and middle camera view is the imperfect scale estimation for the "green" bounding boxes.

Chapter 5

Conclusion & Outlook

Contents

In this work we reviewed the Random Forest framework and employed their recently introduced adaption for visual object detection, namely Hough Forests. We pointed out several limitations regarding their efficiency, as well as their applicability to the on-line domain. For object detection we showed that the runtime of Hough Forests can be reduced by one to two orders of magnitude for only little loss in accuracy. The gathered insights were subsequently applied to the task of tracking in a tracking-by-detection manner. To this end, we utilized the system to track single instances from single cameras, as well as to track multiple instances from multiple cameras. For both cases we showed that our method is able to perform robust tracking at a significantly increased frame rate.

5.1 Conclusion

Our study of the computational cost is built on a theoretical analysis, where we identified each runtime critical part of a tracking-by-detection approach based on Hough Forests. Thus, we analyzed the whole process starting from the required data complexity through to prediction. For instance, we pointed out that the classifier complexity can be easily reduced, enabling us to cut the runtime to one third without any loss in accuracy. Furthermore, we found that the underlying data complexity, the number of required feature channels, as well as the sampled image patches fed into the forest can be reduced without significantly affecting accuracy. Another issue with Hough Forests is that the time for prediction is correlated with the number of training samples. Hence, the Random Forests ability of handling a large amount of training data, which is crucial for building strong models, is lost for Hough Forests. In this work, we investigated this issue and brought up a novel solution, which enabled us to extend Hough Forests on-line during tracking in an effective but still efficient way. More specifically, we combined incremental offset statistics and vote compression in order to build efficient and strong models which are still open for adaption. Overall, we were able to build an object detector, which scores within a few percent of the baseline, while reducing the detection time by a factor of 40.

The proposed method for object detection is designed to be easily adaptable for the task of tracking. It enables us to update a pre-trained classifier on-line, so that the classifier can discriminate between specific object instances. Consequently, we are able to track multiple instances without the inherent requirement for costly post-processing, on which other approaches are relying. Together with the aforementioned efficiency enhancements, we were able to build a tracking system which runs at a fraction of the runtime of the original approach. For instance, reducing the feature dimensionality and the sampling density may be expected to negatively affect the accuracy. However, we found that by carefully adapting the system towards this end, we can reduce the runtime without suffering a performance loss. Despite this reduction of the detectors complexity, we showed that our system is still able to match the state-of-the-art in terms of accuracy.

5.2 Outlook

Despite the discussed advantages, some open issues are still remaining: The proposed method for instance separation is based on visual appearance, which is one of its major strengths – distinguishing it from related approaches. Nevertheless, relying solely on appearance will certainly cause failures in some cases. For example, when many similarly dressed persons are present in the scene visual appearance lacks in its discriminability. Prior knowledge may be used to overcome this problem for special applications. For instance, in team sports the behavior of the target objects is strongly correlated to the current positions of other objects, which has shown to be beneficial for distinguishing many similarly dressed players [55]. For more general cases, the proposed method can be adapted, *e.g.*, by adding a particle filter [33, 44], or an additional analysis of individual trajectories [12, 22, 59]. Such a post-processing could also be used in order to further reduce the computational complexity of the employed detector.

Another possible cause of failures is weak estimation of the object scale. It has been shown that the estimation of the exact bounding box, defined by the scale of an object, is easily disturbed with our approach. The foot point positions are found by fusing the information from all views and are, thus, reliably localized. The scale on the other hand is solely based on the information from a single view and, hence, less robustly estimated. While scale estimation can hardly be fused from all views in our case, calibrated cameras may be used to do so.

Bibliography

[1] Amit, Y. and Geman, D. (1994). Randomized inquiries about shape; an application to handwritten digit recognition. Technical Report 401, Department of Statistics, University of Chicago, IL.

[2] Andriluka, M., Roth, S., and Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[3] Andriyenko, A. and Schindler, K. (2011). Multi-target tracking by continuous energy minimization. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[4] Babenko, B., Yang, M.-H., and Belongie, S. J. (2009). Visual tracking with online multiple instance learning. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[5] Ballard, D. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111 – 122.

[6] Barinova, O., Lempitsky, V. S., and Kohli, P. (2012). On detection of multiple object instances using hough transforms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 34(9):1773–1784.

[7] Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139.

[8] Benenson, R., Markus, M., Tuytelaars, T., and Van Gool, L. (2013). Seeking the strongest rigid detector. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[9] Berclaz, J., Fleuret, F., and Fua, P. (2008). Principled detection-by-classification from multiple views. In *Proc. Int'l Conf. on Computer Vision Theory and Applications*.

[10] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[11] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

[12] Butt, A. A. and Collins, R. T. (2013). Multi-target tracking by lagrangian relaxation to min-cost network flow. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[13] Caruana, R., Karampatziakis, N., and Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. In *Proc. Int'l Conf. on Machine Learning*.

[14] Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proc. Int'l Conf. on Machine Learning*.

[15] Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(5):603–619.

[16] Cootes, T. F., Ionita, M. C., Lindner, C., and Sauer, P. (2012). Robust and accurate shape model fitting using random forest regression voting. In *Proc. European Conf. on Computer Vision*.

[17] Cortes, C. and Vapnik, V. N. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[18] Criminisi, A., Shotton, J., and Konukoglu, E. (2012). Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2-3):81–227.

[19] Criminisi, A., Shotton, J., Robertson, D. P., and Konukoglu, E. (2010). Regression forests for efficient anatomy detection and localization in CT studies. In *Medical Computer Vision Workshop (MICCAI)*.

[20] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[21] Dantone, M., Gall, J., Fanelli, G., and Van Gool, L. (2012). Real-time facial feature detection using conditional regression forests. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[22] Dicle, C., Camps, O., and Sznaier, M. (2013). The way they move: Tracking multiple targets with similar appearance. In *Proc. IEEE Int'l Conf. on Computer Vision*. (to be published).

[23] Dollár, P., Wojek, C., Schiele, B., and Perona, P. (2012). Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 34(4):743–761.

[24] Eshel, R. and Moses, Y. (2010). Tracking in a dense crowd using multiple cameras. *Int'l Journal of Computer Vision*, 88(1):129–143.

[25] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *Int'l Journal of Computer Vision*, 88(2):303–338.

[26] Fanelli, G., Gall, J., and Van Gool, L. (2011). Real time head pose estimation with random regression forests. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[27] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

[28] Fix, E. and Hodges, J. (1951). Discriminatroy analysis - nonparametric discrimination: Consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, Texas.

[29] Fleuret, F., Berclaz, J., Lengagne, R., and Fua, P. (2008). Multicamera people tracking with a probabilistic occupancy map. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 30(2):267–282.

[30] Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proc. European Conf. on Computational Learning Theory*.

[31] Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proc. Int'l Conf. on Machine Learning*.

[32] Gall, J. and Lempitsky, V. S. (2009). Class-specific hough forests for object detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[33] Gall, J., Razavi, N., and Van Gool, L. (2010). On-line adaption of class-specific codebooks for instance tracking. In *Proc. British Machine Vision Conf.*

[34] Gall, J., Razavi, N., and Van Gool, L. (2011a). An introduction to random forests for multi-class object detection. In *Theoretical Foundations of Computer Vision*.

[35] Gall, J., Yao, A., Razavi, N., Van Gool, L., and Lempitsky, V. S. (2011b). Hough forests for object detection, tracking, and action recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(11):2188–2202.

[36] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

[37] Girshick, R. B., Shotton, J., Kohli, P., Criminisi, A., and Fitzgibbon, A. W. (2011). Efficient regression of general-activity human poses from depth images. In *Proc. IEEE Int'l Conf. on Computer Vision*.

[38] Godec, M., Roth, P. M., and Bischof, H. (2011). Hough-based tracking of non-rigid objects. In *Proc. IEEE Int'l Conf. on Computer Vision*.

[39] Godec, M., Roth, P. M., and Bischof, H. (2013). Hough-based tracking of non-rigid objects. *Computer Vision and Image Understanding*, 117(10):1245–1256.

[40] Grabner, H., Leistner, C., and Bischof, H. (2008). Semi-supervised on-line boosting for robust tracking. In *Proc. European Conf. on Computer Vision*.

[41] Guan, L., Franco, J.-S., and Pollefeys, M. (2008). Multi-object shape estimation and tracking from silhouette cues. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[42] Hare, S., Saffari, A., and Torr, P. H. S. (2011). Struck: Structured output tracking with kernels. In *Proc. IEEE Int'l Conf. on Computer Vision*.

[43] Hartley, A. and Zisserman, A. (2006). *Multiple view geometry in computer vision (2. ed.)*. Cambridge University Press.

[44] Hess, R. and Fern, A. (2009). Discriminatively trained particle filters for complex multi-object tracking. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[45] Kalal, Z., Matas, J., and Mikolajczyk, K. (2010). P-N learning: Bootstrapping binary classifiers by structural constraints. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[46] Kalal, Z., Mikolajczyk, K., and Matas, J. (2012). Tracking-learning-detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422.

[47] Khan, S. M. and Shah, M. (2009). Tracking multiple occluding people by localizing on multiple scene planes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 31(3):505–519.

[48] King, R. D., Feng, C., and Sutherland, A. (1995). StatLog: Comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):289–333.

[49] Kluckner, S., Mauthner, T., Roth, P. M., and Bischof, H. (2009). Semantic classification in aerial imagery by integrating appearance and height information. In *Proc. Asian Conf. on Computer Vision*.

[50] Küttel, D., Breitenstein, M. D., Van Gool, L., and Ferrari, V. (2010). What's going on? discovering spatio-temporal dependencies in dynamic scenes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[51] LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Müller, U., Säckinger, E., Simard, P., and Vapnik, V. N. (1995). Comparison of learning algorithms for handwritten digit recognition. In *Proc. Int'l. Conf. on Artificial Neural Networks*.

[52] Leibe, B., Leonardis, A., and Schiele, B. (2008). Robust object detection with interleaved categorization and segmentation. *Int'l Journal of Computer Vision*, 77(1-3):259–289.

[53] Leistner, C., Godec, M., Saffari, A., and Bischof, H. (2010). On-line multi-view forests for tracking. In *Proc. DAGM Symposium*.

[54] Lepetit, V., Lagger, P., and Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[55] Liu, J., Carr, P., Collins, R. T., and Liu, Y. (2013). Tracking sports players with context-conditioned motion models. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[56] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int'l Journal of Computer Vision*, 60(2):91–110.

[57] Maji, S. and Malik, J. (2009). Object detection using a max-margin hough transform. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[58] McCulloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):127–147.

[59] Milan, A., Schindler, K., and Roth, S. (2013). Detection- and trajectory-level exclusion in multiple object tracking. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[60] Monteiro, G., Ribeiro, M., Marcos, J., and Batista, J. (2007). Wrongway drivers detection based on optical flow. In *Proc. Int'l Conf. on Image Processing*.

[61] 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. Int'l Conf. on Information Processing in Medical Imaging*.

[62] Nowozin, S. (2012). Improved information gain estimates for decision tree induction. In *Proc. Int'l Conf. on Machine Learning*.

[63] Okada, R. (2009). Discriminative generalized hough transform for object dectection. In *Proc. IEEE Int'l Conf. on Computer Vision*.

[64] Özuysal, M., Calonder, M., Lepetit, V., and Fua, P. (2010). Fast keypoint recognition using random ferns. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(3):448–461.

[65] Özuysal, M., Fua, P., and Lepetit, V. (2007). Fast keypoint recognition in ten lines of code. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[66] Perlich, C., Provost, F. J., and Simonoff, J. S. (2003). Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 4:211–255.

[67] Possegger, H., Sternig, S., Mauthner, T., Roth, P. M., and Bischof, H. (2013). Robust real-time tracking of multiple objects by volumetric mass densities. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[68] Qin, T., Zhong, B., Chin, T.-J., and Wang, H. (2012). Matting-driven online learning of hough forests for object tracking. In *Proc. Int'l Conf. on Pattern Recognition*.

[69] Quinlan, J. R. (1996). Bagging, boosting, and c4.5. In *Proc. National Conf. on Artificial Intelligence*.

[70] Razavi, N., Alvar, N. S., Gall, J., and Van Gool, L. (2012). Sparsity potentials for detecting objects with the hough transform. In *Proc. British Machine Vision Conf.*

[71] Razavi, N., Gall, J., and Van Gool, L. (2011). Scalable multi-class object detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[72] Roth, P. M., Leistner, C., Berger, A., and Bischof, H. (2010). Multiple instance learning from multiple cameras. In *IEEE Workshop on Camera Networks (CVPR)*.

[73] Saffari, A., Leistner, C., Santner, J., Godec, M., and Bischof, H. (2009). On-line random forests. In *IEEE Workshop on On-Line learning for Computer Vision (ICCV)*.

[74] Schreiber, D., Cambrini, L., Biber, J., and Sardy, B. (2008). Online visual quality inspection for weld seams. *Int'l Journal of Advanced Manufacturing Technology*, 42(5-6):497–504.

[75] Schulter, S., Leistner, C., Roth, P. M., Van Gool, L., and Bischof, H. (2011). On-line hough forests. In *Proc. British Machine Vision Conf.*

[76] Schulter, S., Roth, P. M., and Bischof, H. (2013). Ordinal random forests for object detection. In *Proc. DAGM Symposium*.

[77] Sharp, T. (2008). Implementing decision trees and forests on a GPU. In *Proc. European Conf. on Computer Vision*.

[78] Shotton, J., Fitzgibbon, A. W., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[79] Shotton, J., Girshick, R. B., Fitzgibbon, A., Sharp, T., Cook, M., Finocchio, M., Moore, R., Kohli, P., Criminisi, A., Kipman, A., and Blake, A. (2012). Efficient human pose estimation from single depth images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 99(PrePrints):1.

[80] Snavely, N., Seitz, S. M., and Szeliski, R. (2008). Modeling the world from internet photo collections. *Int'l Journal of Computer Vision*, 80(2):189–210.

[81] Sternig, S., Mauthner, T., Irschara, A., Roth, P. M., and Bischof, H. (2011). Multi-camera multi-object tracking by robust hough-based homography projections. In *IEEE Workshop on Visual Surveillance (ICCV)*.

[82] Sun, M., Kohli, P., and Shotton, J. (2012). Conditional regression forests for human pose estimation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*.

[83] Tang, D., Yu, T., and Kim, T.-K. (2013a). Real-time articulated hand pose estimation using semi-supervised transductive regression forests. In *Proc. IEEE Int'l Conf. on Computer Vision*. (to be published).

[84] Tang, S., Andriluka, M., Milan, A., Schindler, K., Roth, S., and Schiele, B. (2013b). Learning people detectors for tracking in crowded scenes. In *Proc. IEEE Int'l Conf. on Computer Vision*. (to be published).

[85] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.-W., Singh, S., Snider, J., Stentz, A., Whittaker, W. â., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466.

[86] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer New York, Inc.

[87] Wohlhart, P. (2013). *Object detection based on local evidence*. PhD thesis, Graz University of Technology.

[88] Wohlhart, P., Donoser, M., Roth, P. M., and Bischof, H. (2012a). Detecting partially occluded objects with an implicit shape model random field. In *Proc. Asian Conf. on Computer Vision*.

[89] Wohlhart, P., Schulter, S., Köstinger, M., Roth, P. M., and Bischof, H. (2012b). Discriminative hough forests for object detection. In *Proc. British Machine Vision Conf.*

[90] Yao, A., Uebersax, D., Gall, J., and Van Gool, L. (2010). Tracking people in broadcast sports. In *Proc. DAGM Symposium*.

[91] Yilmaz, A., Javed, O., and Shah, M. (2006). Object tracking: A survey. *ACM Computing Surveys*, 38(4).

[92] Zamir, A. R., Dehghan, A., and Shah, M. (2012). GMCP-Tracker: Global multi-object tracking using generalized minimum clique graphs. In *Proc. European Conf. on Computer Vision*.