FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Robust Object Tracking Based on Tracking-Learning-Detection

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Georg Nebehay
Matrikelnummer 0426011

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: a.o.Univ.-Prof. Dipl.-Ing. Dr.techn. Robert Sablatnig
Mitwirkung: Dr. Roman Pflugfelder

Wien, 8. Mai 2012

(Unterschrift Verfasser) (Unterschrift Betreuung)

Abstract

Current state-of-the-art methods for object tracking perform adaptive tracking-by-detection, meaning that a detector predicts the position of an object and adapts its parameters to the object's appearance at the same time. While suitable for cases when the object does not disappear from the scene, these methods tend to fail on occlusions. In this work, we build on a novel approach called Tracking-Learning-Detection (TLD) that overcomes this problem. In methods based on TLD, a detector is trained with examples found on the trajectory of a tracker that itself does not depend on the object detector. By decoupling object tracking and object detection we achieve high robustness and outperform existing adaptive tracking-by-detection methods. We show that by using simple features for object detection and by employing a cascaded approach a considerable reduction of computing time is achieved. We evaluate our approach both on existing standard single-camera datasets as well as on newly recorded sequences in multi-camera scenarios.

Kurzfassung

Aktuelle Objektverfolgungsmethoden am Stand der Technik verwenden adaptives Tracking-By-Detection, was bedeutet, dass ein Detektor die Position eines Objekts ermittelt und gleichzeitig seine Parameter an die Erscheinung des Objekts anpasst. Während solche Methoden in Fällen funktionieren, in denen das Objekt nicht vom Schauplatz verschwindet, neigen sie dazu, bei Verdeckungen fehlzuschlagen. In dieser Arbeit bauen wir auf einem neuen Ansatz auf, der Tracking-Learning-Detection (TLD) genannt wird und der dieses Problem bewältigt. In TLD-Methoden wird der Detektor mit Beispielen trainiert, die auf der Trajektorie eines Trackers liegen, der unabhängig vom Detektor ist. Durch die Entkopplung von Objektverfolgung und Objektdetektion erreichen wir eine große Robustheit und übertreffen existierende adaptive Tracking-By-Detection-Methoden. Wir zeigen, dass durch den Einsatz von einfachen Features zur Objekterkennung und mit der Verwendung eines kaskadierten Ansatzes eine beträchtliche Reduktion der Rechenzeit erzielt wird. Wir evaluieren unseren Ansatz sowohl auf existierenden Standarddatensätzen in einer Kamera als auch auf neu aufgenommenen Sequenzen in mehreren Kameras.

Acknowledgements

I want to thank Roman Pflugfelder for being a wonderful mentor over the entire course of the development of this thesis. The constant support of Robert Sablatnig also was of great importance to me. I am deeply grateful to Gustavo Javier Fernandez, Branislav Micusik, Cristina Picus and Csaba Beleznai for endless hours of dicussion, for valuable input and for helping me with the evaluation. I also want to thank Bernd Lukatschek and Clemens Korner for assisting me with the implementation. I also appreciated learning from Andreas Zoufal's profound knowledge in software engineering.

This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT08-030.

Contents

CHAPTER 1

Introduction

The visual cortex of the human brain locates and identifies objects by analysing the information arriving as action potentials that are triggered in the retina [20]. While perceptual psychologists study how the human visual system interprets environmental stimuli, researchers in **computer vision** develop mathematical techniques in order to extract information about physical objects based on camera images [44]. Computer vision methods are applied to optical character recognition, quality inspection, robot guidance, scene reconstruction and object categorisation [47]. One domain of research in computer vision is **object tracking**, in which methods are studied that estimate the location of targets in consecutive video frames [34]. The proliferation of high-powered computers, the availability of high quality and inexpensive video cameras, and the need for automated video analysis have drawn interest to applying object tracking algorithms in automated surveillance, automatic annotation of video data, human-computer interaction, traffic monitoring and vehicle navigation [50].

1.1 Problem Definition

In this work we focus on semi-automated single-target tracking. The problem of **single-target** tracking is defined as follows [34]. Given a sequence of images $I_1 \dots I_n$, estimate the state x_k of the target for each frame I_k. Object tracking methods encode the state x_k as centroids, bounding boxes, bounding ellipses, chains of points or shape [34]. For example, in Fig. 1.1, a bounding box is shown around an object of interest. In this case, the parameters of x_k consist of the upper left corner of the rectangle (x, y) and its width and height. Maggio and Cavallaro [34] group approaches based on the amount of user interaction that is required to identify the objects of interest. *Manual tracking* requires the interaction with the user in every frame. *Automated tracking* methods use a priori information in order to initialise the tracking process automatically. In **semi-automated** tracking, user input is required in order to initialise the tracking process.

According to Maggio and Cavallaro [34], the main challenge in object tracking is **clutter**. Clutter is the phenomenon when features expected from the object of interest are difficult to

Figure 1.1: The state of an object encoded in a bounding box. The enlarged patch constrained by the bounding box is displayed on the right. The image is part of the PETS 2009[1] dataset.

discriminate against features extracted from other objects in the scene. In Fig. 1.2 an example for clutter is shown. In this image, several objects are present that are similar in shape to the object of interest. Another challenge is introduced by **appearance variations** of the target itself. Intrinsic appearance variability includes pose variation and shape deformation, whereas extrinsic appearance variability includes illumination change, camera motion and different camera viewpoints [41]. Approaches that maintain a template of the object of interest typically face the **template update problem** that relates to the question of how to update an existing template so that it remains a representative model [35]. If the original template is never changed, it will eventually no longer be an accurate representation of the model. When the template is adapted to every change in appearance, errors will accumulate and the template will steadily **drift** away from the object. This problem is closely related to the **stability-plasticity dilemma**, which relates to the trade-off between the stability required to retain information and the plasticity required for new learning [22]. This dilemma is faced by all learning systems [1]. Objects undergo **occlusions** when covered by other object or when they leave the field of view of the camera. In order to handle such cases, a mechanism is necessary that re-detects the object independently of its last position in the image [50]. Requirements on the **execution time** pose another difficulty. [50].

1.2 Related Work

Lepetit et al. [30] identify two paradigms in object tracking. **Recursive tracking** methods estimate the current state x_t of an object by applying a transformation on the previous state x_{t-1} based on measurements $z_1 \dots z_t$ taken in the respective images. The recursive estimation of a state depends on the state of the object in the previous frame and is susceptible to error accumulation [30]. For instance, Lucas and Kanade [33] propose a method for estimating sparse optic

[1] Performance Evaluation for Tracking and Surveillance: http://www.cvg.rdg.ac.uk/PETS2009/

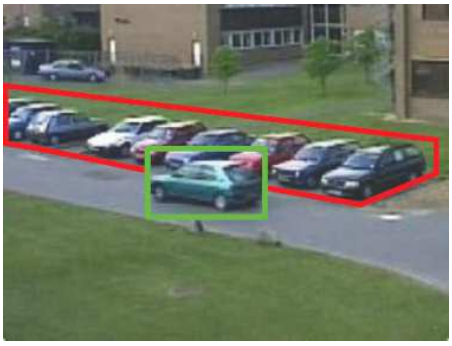


Figure 1.2: The main challenge in object tracking is to the distinguish the object of interest (green) from clutter in the background (red). Image is from [10].

flow within a window around a pixel. The optic flow is fit into a transformation model that is used to predict the new position of the object. In our work, we use the method of Lucas and Kanade for tracking the object of interest in consecutive frames. Comaniciu et al. [15] propose a tracker based on *mean shift*. The transformation of the object state is obtained by finding the maximum of a similarity function based on color histograms. In contrast, **Tracking-by-detection** methods estimate the object state solely by measurements taken in the current image. This principle remedies the effect of error accumulation. However, the object detectors have to be trained beforehand. Özuysal et al. [38] generate synthetic views of an object by applying affine warping techniques to a single template and train an object detector on the warped images. The object detector is based on pairwise pixel comparison and is implemented efficiently. Object detection is then performed in every frame in order to track the object. We use an online variant of this method as a part of an object detection cascade.

In-between these paradigms, **adaptive tracking-by-detection** methods have been developed that update an object detector online. Avidan [4] integrates a support vector machine classifier into an optic-flow-based tracker. Instead of minimizing an intensity difference function between successive frames, he maximises the classifier score. The support vector machine is trained beforehand and unable to adapt. Collins et al. [14] were the first to treat tracking as a binary classification problem, the two classes being the object of interest and background. They employ automatic feature selection in order to switch to the most discriminative color space from a set of different color spaces. They employ **self-learning** in order to acquire new training examples. In self-learning, a supervised method is retrained using its own predictions as additional labeled points. This setting is prone to drift [12]. Javed et al. [25] employ co-training in order to label incoming data and use it to improve a detector trained in an offline manner. It has been argued that in object tracking the underlying assumption of co-training that two conditionally independent views of the same data are available is violated, since in object tracking training examples are sampled from the same modality [27]. Ross et al. [41] incrementally learn a low-dimensional subspace representation and adapt this representation to changes in the appearance of the target. Adam et al. [2] propose an approach called *FragTrack* that uses a static part-based appearance model based on integral histograms. Avidan [5] uses self-learning for boosting in

order to update an ensemble classifier. Grabner et al. [21] employ a **semi-supervised** approach and enforces a prior on the first patch, while treating the incoming images as unlabeled data. However, if the prior is too strong, then the object is likely not to be found again. If it is too weak, then it does not discriminate against clutter. Babenko et al. [6] apply Multiple Instance Learning (*MIL*) to object tracking. In multiple instance learning, overlapping examples of the target are put into a labeled bag and passed on to the learner, which is therefore allowed more flexibility in finding a decision boundary. Stalder et al. [46] split the tasks of detection, recognition and tracking into three separate classifiers and achieve robustness to occlusions. Santner et al. [42] propose *PROST*, a cascade of a non-adaptive template model, an optical-flow-based tracker and an online random forest. The random forest is updated only if its ouput overlaps with the output of one of the two other trackers. Hare et al. [23] generalize from the binary classification problem to structured output prediction. In their method called *Struck* they directly estimate the expected state transformations instead of predicting class labels.

Kalal et al. [27] propose a method called **TLD** (**Tracking-Learning-Detection**) that uses patches found on the trajectory of an optic-flow-based tracker in order to train an object detector. Updates are performed only if the discovered patch is similar to the initial patch. What separates this method from the adaptive tracking-by-detection methods is that the output of the object detector itself is used only to reinitialize the optic-flow-based tracker in case of failure but is never used in order to update the classifier itself. Kalal et al. achieve superior results as well as higher frame rates compared to adaptive tracking-by-detection methods.

1.3 Scope of Work

We use the approach of Kalal et al. [28] for recursive tracking. This approach is based on estimating optical flow using the method of Lucas and Kanade [33]. For object detection, we follow [26] and maintain templates that are normalised in brightness and size. We keep separate templates for positive examples of the object and for negative examples found in the background. These templates form the basis of an object detector that is run independently of the tracker. New templates are acquired using P/N-learning as proposed in [27]. If the detector finds a location in an image exhibiting a high similarity to the templates, the tracker is re-initialised on this location. Since the comparison of templates is computationally expensive, we employ a cascaded approach to object detection. In [27] a random fern classifier [38] based on 2-bit-binary patterns and a fixed single template is used. Our object detection cascade consists of a foreground detector, a variance filter, a random fern classifier based on features proposed in [31] and the template matching method. In contrast to Kalal et al., we do not employ image warping for learning. Fig. 1.3 depicts the workflow of our approach. The initialisation leads to a learning step. Next, the recursive tracker and the detector are run in parallel and their results are fused into a single final result. If this result passes a validation stage, learning is performed. Then the process repeats.

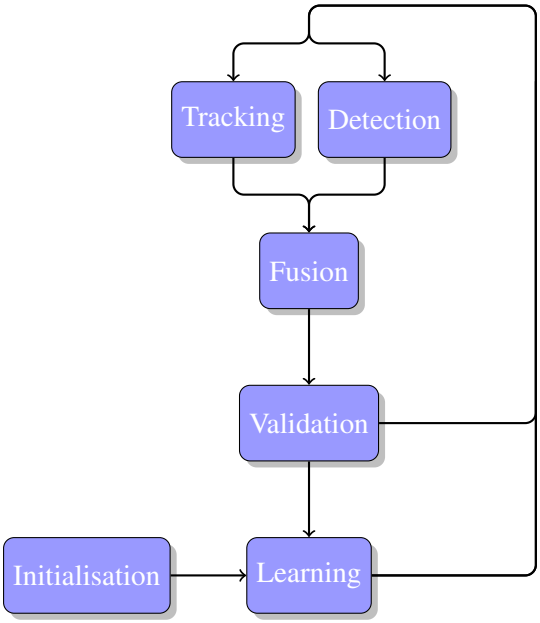


Figure 1.3: The tracking process is initialised by manually selecting the object of interest. No further user interaction is required.

1.4 Contribution

In this work, we follow the Tracking-Learning-Detection approach of Kalal et al. [27]. We extend their object detection cascade and use different features in order to reduce execution time. This document encompasses all the details that are necessary to fully implement our approach. We implement our approach[2] in C++ and evaluate it both on existing and newly recorded test data. We give a performance comparison to existing methods and analyse whether our approach is suitable for multi-camera scenarios.

1.5 Organisation

This work is organised as follows. In Chapter 2, the tracking method based on the estimation of optical flow is described. In Chapter 3, the cascaded approach to object detection is explained. Chapter 4 deals with the question of how to fuse the results of the tracker and describes what happens during the learning step. Chapter 5 shows experimental results on test data as well as a comparison to other tracking methods. Chapter 6 gives a conclusion and final remarks.

[2]The code is available at `http://gnebehay.github.com/OpenTLD`

1.6 Summary

In this chapter we introduced the field of object tracking and gave a problem definition. We then explained that tracking is made a non-trivial task by clutter, appearance variations of the target and the template update problem. We then gave a description of related work and explained that existing approaches use elements of recursive tracking and tracking-by-detection. We explained that we base our work on a novel approach that integrates a tracker and a detector.

CHAPTER 2

Tracking

In this chapter we describe a recursive method for object tracking. In this method, no a priori information is required about the object except for its location in the previous frame, which means that an external initialisation is required. In our approach, the initialisation is accomplished by manual intervention in the first frame and by the results of an object detection mechanism in consecutive frames.

We follow the approach of Kalal et al. [28] for recursive tracking. We explain this method according to Fig. 2.1. First, an equally spaced set of points is constructed in the bounding box in frame t, which is shown in the left image. Next, the optical flow is estimated for each of these points by employing the method of Lucas and Kanade [33]. This method works most reliably if the point is located on corners [45] and is unable to track points on homogenous regions. We use information from the Lucas-Kanade method as well as two different error measures based on normalised cross correlation and forward-backward error in order to filter out tracked points that are likely to be erroneous. In the right image the remaining points are shown. If the median of all forward-backward error measures is above a certain threshold, we stop recursive tracking entirely, since we interpret this event as an indication for drift. Finally, the remaining points are used in order to estimate the position of the new bounding box in the second frame by employing a transformation model based on changes in translation and scale. In the right image, the bounding box from the previous frame was transformed according to the displacement vectors from the remaining points.

This chapter is organised as follows. Sec. 2.1 describes the Lucas-Kanade for estimating optical flow. In Sec. 2.2 the error measures are introduced. In Sec. 2.3 the transformation model that we use is described and an algorithm is given. Sec. 2.4 concludes this chapter with a summary.

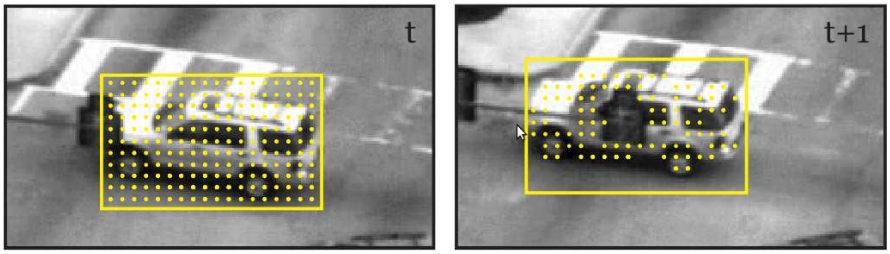


Figure 2.1: The principle of the recursive tracking method consists of tracking points using an estimation of the optical flow, retaining only correctly tracked points and estimating the transformation of the bounding box. Images are from [28].

2.1 Estimation of Optical Flow

Lucas and Kanade base their approach on three assumptions. The first assumption is referred to as **brightness constancy** [8] and is expressed as

$$I(X) = J(X+d). \tag{2.1}$$

Eq. 2.1 states that a pixel at the two-dimensional location (X) in an image I might change its location in the second image J but retains its brightness value. In the following, the vector d will be referred to as the displacement vector. The second assumption is referred to [8] as **temporal persistence**. It states that the displacement vector is small. Small in this case means that $J(X)$ can be approximated by

$$J(X) \approx I(X) + I'(X)d. \tag{2.2}$$

In Eq. 2.2 $I'(X)$ is the gradient of I at location X. An estimate for d is then

$$d \approx \frac{J(X) - I(X)}{I'(X)}. \tag{2.3}$$

For any given pixel, Eq. 2.3 is underdetermined and the solution space is a line instead of a point. The third assumption, known as **spatial coherence**, alleviates this problem. It states that all the pixels within a window around a pixel move coherently. By incorporating this assumption, d is found by minimizing the term

$$\sum_{(x,y)\in W} (J(X) - I(X) - I'(X)d)^2, \tag{2.4}$$

which is the least-squares minimisation of the stacked equations. The size of W defines the considered area around each pixel. In [48] it is shown that the closed-form solution for Eq. 2.4 is

$$Gd = e, \tag{2.5}$$

where

$$G = \sum_{X\in W} I'(X)I'(X)^\top = \sum_{X\in W} \begin{pmatrix} I_x^2(X) & I_{xy}(X) \\ I_{xy}(X) & I_y^2(X) \end{pmatrix} \tag{2.6}$$

and

$$e = \sum_{(x,y)\in W} (I(X) - J(X))I'(X). \tag{2.7}$$

Additional implementational details are in [8].

2.2 Error Measures

In order to increase the robustness of the recursive tracker, we use three criteria in order to filter points that were tracked unreliably. The first criterion is established directly from Eq. 2.5. It can be seen from this equation that d can be calculated only if G is invertible. G is reliably invertible if it has two large eigenvalues (λ_1, λ_2), which is the case when there are gradients in two directions [8]. We use the formula

$$min(\lambda_1, \lambda_2) > \lambda \tag{2.8}$$

of Shi and Tomasi [45] as a first criterion for reliable tracking of points.

Kalal et al. [28] propose the **forward-backward error** measure. This error measure is illustrated conceptually in Fig. 2.2. In the left image, the point *1* is tracked correctly to its corresponding position in the right image. The point *2*, however, ends up at a wrong location as an occlusion occurs. The proposed error measure is based on the idea that the tracking of points must be reversible. Point *1* is tracked back to its original location. In contrast, point *2* is tracked back to a different location. The proposed error measure is defined as the Euclidean distance

$$\varepsilon = |p - p''|. \tag{2.9}$$

In Eq. 2.9 p'' is

$$p'' = LK(LK(p)), \tag{2.10}$$

meaning that the Lucas-Kanade method is applied twice on p.

In [28] the forward-backward error measure is used in conjunction with another measure based on the similarity of the patch surrounding p and the patch surrounding the tracking result p'. The similarity of these two patches P_1 and P_2 is compared using the Normalised Correlation Coefficient (NCC) of two image patches P_1 and P_2 that is defined as

$$\mathrm{NCC}(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^{n} \frac{(P_1(x) - \mu_1)(P_2(x) - \mu_2)}{\sigma_1 \sigma_2}. \tag{2.11}$$

In Eq. 2.11 where μ_1, μ_2, σ_1 and σ_2 are the means and standard deviations of P_1 and P_2. The normalised correlation coefficient is invariant against uniform brightness variations [32].

2.3 Transformation Model

Following the approach of Kalal et al. [28], we calculate the median of all forward-backward errors med_{FB} and the median med_{NCC} of all similarity measures and keep only those points exhibiting a forward-backward error less than med_{FB} and a similarity measure larger than med_{NCC}.

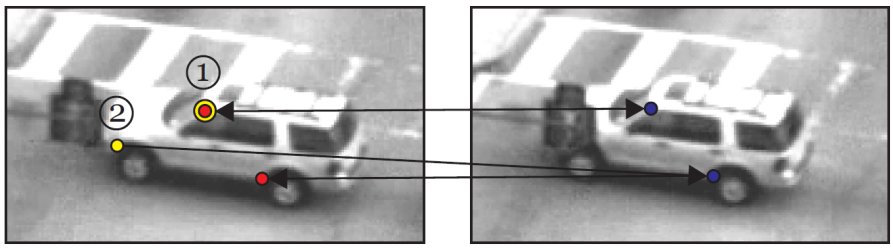


Figure 2.2: The idea of the forward-backward error measure lies in the observation that certain points cannot be re-tracked to their original location. Images are from [28].

Furthermore, if med_{FB} is larger than a predefined threshold θ_{FB}, we do not give any results as we interpret this case as an unreliable tracking result. The remaining points are used to calculate the transformation of the bounding box. For this, the pairwise distances between all points are calculated before and after tracking and the relative increase is interpreted as the change in scale. The translation in x-direction is computed using the median of the horizontal translations of all points. The translation in y-direction is calculated analogously. An algorithmic version of the proposed tracker is given in Alg. 1. We use a grid of size 10×10, a window size $W = 10$ and a threshold $\theta_{FB} = 10$ for all of our experiments.

Algorithm 1 Recursive Tracking

Input: B_I, I, J

$p_1 \dots p_n \leftarrow \text{generatePoints}(B_I)$
for all p_i **do**
  $p_i' \leftarrow LK(p_i)$
  $p_i'' \leftarrow LK(p_i')$
  $\varepsilon_i \leftarrow |p_i - p_i''|$
  $\eta_i \leftarrow NCC(W(p_i), W(p_i'))$
end for
$med_{NCC} \leftarrow \text{median}(\eta_1 \dots \eta_n)$
$med_{FB} \leftarrow \text{median}(\varepsilon_1 \dots \varepsilon_n)$
if $med_{FB} > \theta_{FB}$ **then**
  $B_J = \emptyset$
else
  $C \leftarrow \{(p_i, p_i') \mid p_i' \neq \emptyset, \varepsilon_i \leq med_{FB}, \eta_i \geq med_{ncc}\}$
  $B_J \leftarrow \text{transform}(B_I, C)$
end if

2.4 Summary

In this chapter we described the method that we employ for recursive estimation of an object of interest. No a priori information about the object is required except its position in the previous

Figure 2.3: Recursive tracking is possible as long as the selected object is visible in the image. In the third frame an occlusion occurs. Images are from the SPEVI[1] dataset.

frame. We explained that the transformation of the bounding box of the previous frame is estimated by calculating a sparse approximation to the optic-flow field and explained in detail the method of Lucas-Kanade that we use for this estimation. We introduced two error measures in order to improve results and to act as a stopping criterion. In Fig. 2.3 an example result of this tracking method is shown. In the left-most image, the initial bounding box is depicted in blue. In the second image, it is shown that the selected object was tracked correctly. By employing the stopping criterion, the method is able to identify when an occlusion takes place, as it is shown in the third image. However, in the fourth image the method is unable to re-initialise itself as it lacks a mechanism for object detection.

[1] Surveillance Performance EValuation Initiative: http://www.eecs.qmul.ac.uk/~andrea/spevi.html

CHAPTER 3

Detection

In this chapter we discuss the method that we employ for object detection. Object detection enables us to re-initialise the recursive tracker that itself does not maintain an object model and is therefore unable to recover from failure. While the recursive tracker depends on the location of the object in the previous frame, the object detection mechanism presented here employs an exhaustive search in order to find the object. Since several thousands of subwindows are evaluated for each input image, most of the time of our overall approach is spent for object detection.

Our object detector is based on a sliding-window approach [49, 16], which is illustrated in Fig. 3.1. The image at the top is presented to the object detector, which then evaluates a classification function at certain predefined subwindows within each input image. Depending on the size of the initial object, we typically employ 50,000 to 200,000 subwindows for an image of VGA (640×480) resolution. Each subwindow is tested independently whether it contains the object of interest. Only if a subwindow is accepted by one stage in the cascade, the next stage is evaluated. Cascaded object detectors aim at rejecting as many non-relevant subwindows with a minimal amount of computation [43]. The four stages that we use for image classification are shown below the input image. First, we use a background subtraction method in order to restrict the search space to foreground regions only. This stage requires a background model and is skipped if it is not available. In the second stage all subwindows are rejected that exhibit a variance lower than a certain threshold. The third stage comprises an ensemble classifier based on random ferns [38]. The fourth stage consists of a template matching method that is based on the normalised correlation coefficient as a similarity measure. We handle overlapping accepted subwindows by employing a non-maximal suppression strategy.

This chapter is organised as follow. In Sec. 3.1 the sliding-window approach is described in detail. Sec. 3.2 shows how a background model restricts the search space to foreground regions. In Sec. 3.3 the variance filter is described. Sec. 3.4 comprises a description of the ensemble classifier that is able rapidly identify positive subwindows. The template matching method is described in Sec. 3.5. In Sec. 3.6 it is shown how overlapping detections are combined into a single result. In Sec 3.7 a summary of this chapter is given.

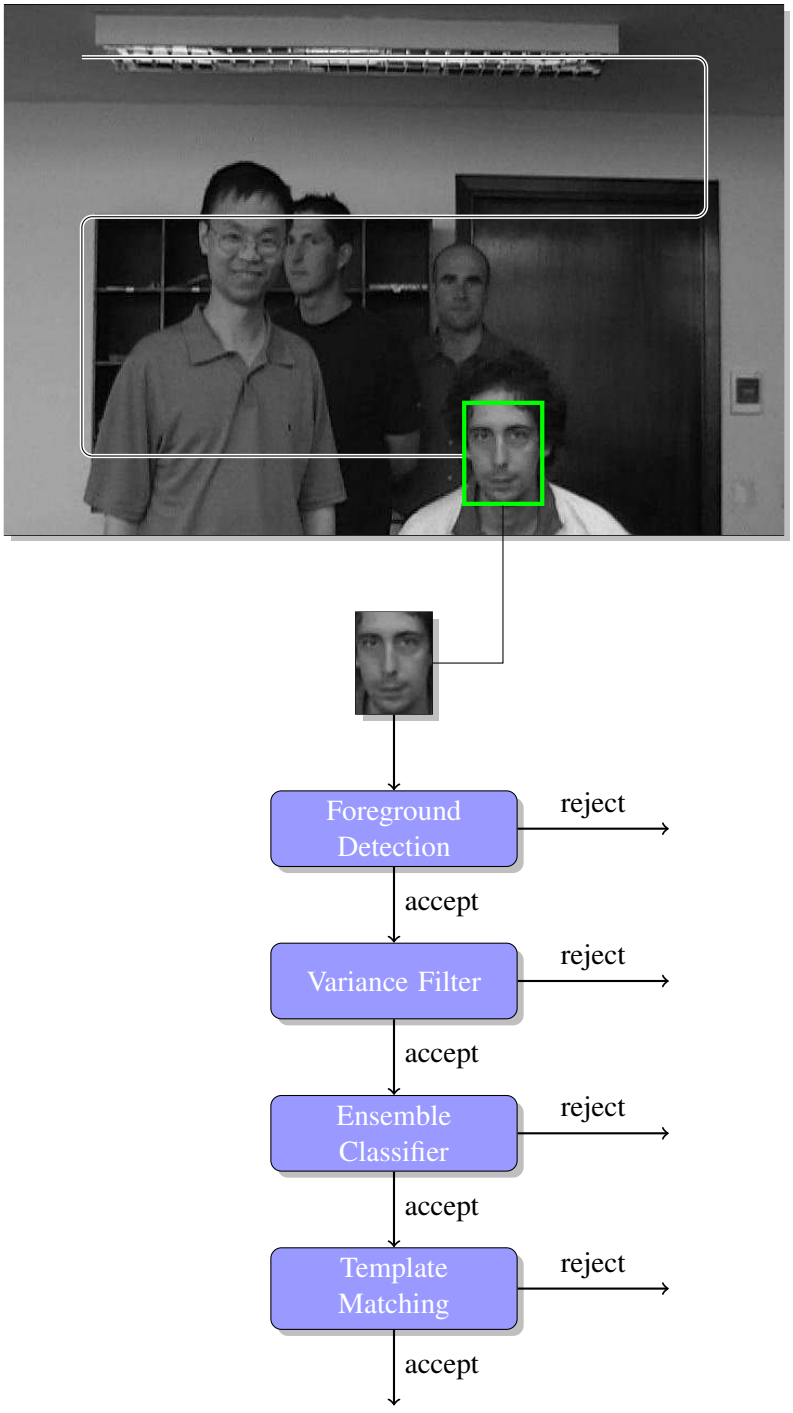


Figure 3.1: In sliding-window-based approaches for object detection, subwindows are tested independently. We employ a cascaded approach in order to reduce computing time. The input image is from the SPEVI dataset.

3.1 Sliding-Window Approach

In sliding-window-based approaches for object detection, subimages of an input image are tested whether they contain the object of interest [29]. Potentially, every possible subwindow in an input image might contain the object of interest. However, in a VGA image there are already 23,507,020,800 possible subwindows and the number of possible subwindows grows as n^4 for images of size $n \times n$ (see App. A.1 for a proof). We restrict the search space to a subspace $\mathcal{R}$ by employing the following constraints. First, we assume that the object of interest retains its aspect ratio. Furthermore, we introduce margins d_x and d_y between two adjacent subwindows and set d_x and d_y to be $\frac{1}{10}$ of the values of the original bounding box. In order to employ the search on multiple scales, we use a scaling factor $s = 1.2^a, a \in \{-10 \dots 10\}$ for the original bounding box of the object of interest. We also consider subwindows with a minimum area of 25 pixels only. The size of the set of all subwindows $\mathcal{R}$ constrained in this manner is then

$$|\mathcal{R}| = \sum_{s \in 1.2^{\{-10 \dots 10\}}} \left\lfloor \frac{n - s(w + d_x)}{s d_x} \right\rfloor \left\lfloor \frac{m - s(h + d_x)}{s d_y} \right\rfloor . \tag{3.1}$$

In Eq. 3.1 w and h denote the size of the initial bounding box and n and m the width and height of the image. A derivation for this formula is given in App. A.1. For an initial bounding box of size $w = 80$ and $h = 60$ the number of subwindows in a VGA image is 146,190. Since each subwindow is tested independently, we employ as many threads as cores are available on the system in order to test the subwindows.

3.2 Foreground Detection

One approach in order to identify moving objects in a video stream is background subtraction, where each video frame is compared against a background model [13]. In this section, we describe how a background model speeds up the detection process. The problem of establishing a background model itself is non-trivial and out of scope for this work, for a survey see [39]. We perform background subtraction in four steps, as it is depicted in Fig. 3.2. In this figure, the right upper image is the background image I_{bg} and the top left image is the image I in which object detection is to be performed. We start by calculating the **absolute difference** of I_{bg} and I

$$I_{absDiff} = |I_{bg} - I|. \tag{3.2}$$

The result of Eq. 3.2 is shown in the first image of the second row. We now apply a **thresholding** of 16 pixels to $I_{absDiff}$ in order to create a binary image I_{binary}, which is shown in the second image of the second row.

$$I_{binary}(x,y) = \begin{cases} 1 & \text{if } I_{absDiff}(x,y) > 16 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

In the following, we will refer to connected white pixels as components. In order to calculate the area and the smallest bounding box the blob fits into, we now apply the **labeling** algorithm proposed in [11]. This algorithm calculates labels in a single pass over the image. The idea of

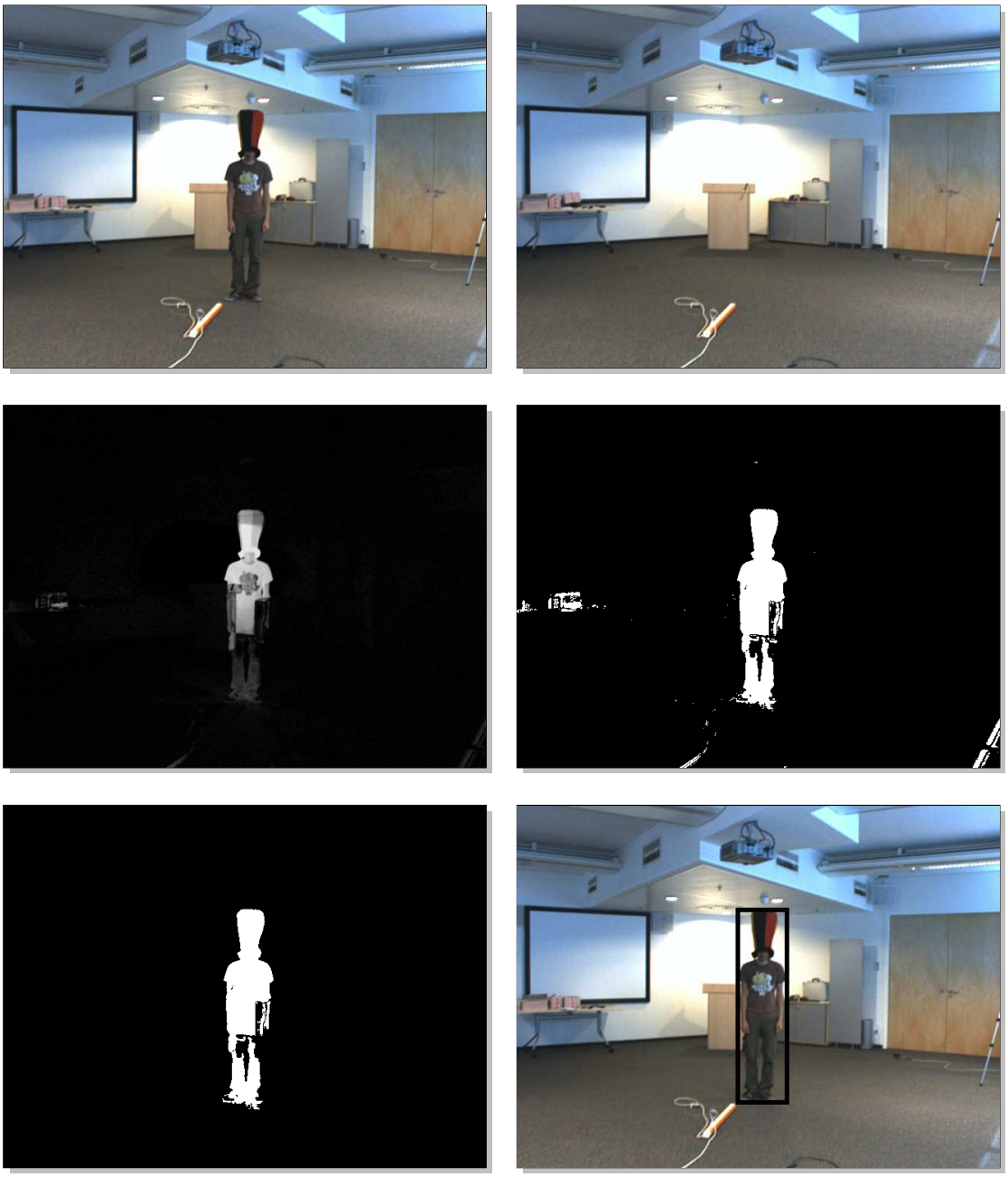


Figure 3.2: The process of background subtraction. From top left to bottom right: The input image, the background image, the result of the subtraction, the image after thresholding, after the removal of small components, the minimal bounding box around the foreground.

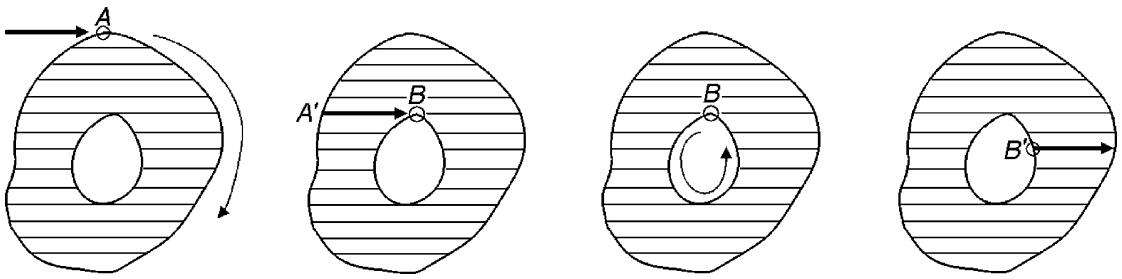


Figure 3.3: Labeling algorithm. Image is from [11].

this algorithm is shown in Fig. 3.3. Starting from the top row, each line is scanned from left to right. As soon as a white pixel A is encountered that is not yet labeled, a unique label is assigned to A and all the points lying on the contour of the component are assigned the same label as A. This contour is considered an external contour. This case is shown in the first image. If a pixel A' on an external contour is encountered that is already labeled, all white pixels to the right are assigned the same label until another contour is encountered. If this is an external contour it is already labeled and the labeling algorithm proceeds. In the second image, this corresponds to to all the lines above point B. If it is not yet labeled, as it is the case for the contour on which B lies, then it considered an internal contour and all of its pixels are assigned the same label as B. This case is shown in the third image. If a labeled internal contour point B' is encountered, all subsequent white pixels are assigned the same label as A. This case is shown in the fourth image. The smallest bounding box the component fits into is determined by the coordinates of the outermost pixels of the component. The area of the component is the sum of all white pixels in a component.

Going back to Fig. 3.2, we now remove all components from the binary image with an area less than the size of the originally selected bounding box. The result of this operation is shown in the first image in the third row. All subwindows are rejected that are not fully contained inside one of the smallest bounding boxes around the remaining components. We call this set of bounding boxes C. If no background image is available, then all subwindows are accepted.

3.3 Variance Filter

The variance of an image patch is a measure for uniformity. In Fig. 3.4 two sample subwindows are shown, marked in red, that are evaluated in uniform background regions. Both of these subwindows contain patches that exhibit a variance lower than the patch of the object selected for tracking, which is contained in the right green rectangle. In this section we describe a mechanism that calculates the variance of a patch in a subwindow using integral images and that rejects patches exhibiting a variance lower than a threshold σ^2_{min}. Such a variance filter is able to rapidly reject uniform background regions but unable to distinguish between different well-structured objects. For instance, the left green bounding box in Fig. 3.4 will be accepted as well.

We use an efficient mechanism in order to compute the variance that is shown in [49]. In order to simplify the following explanation, image patches defined by the bounding box B are

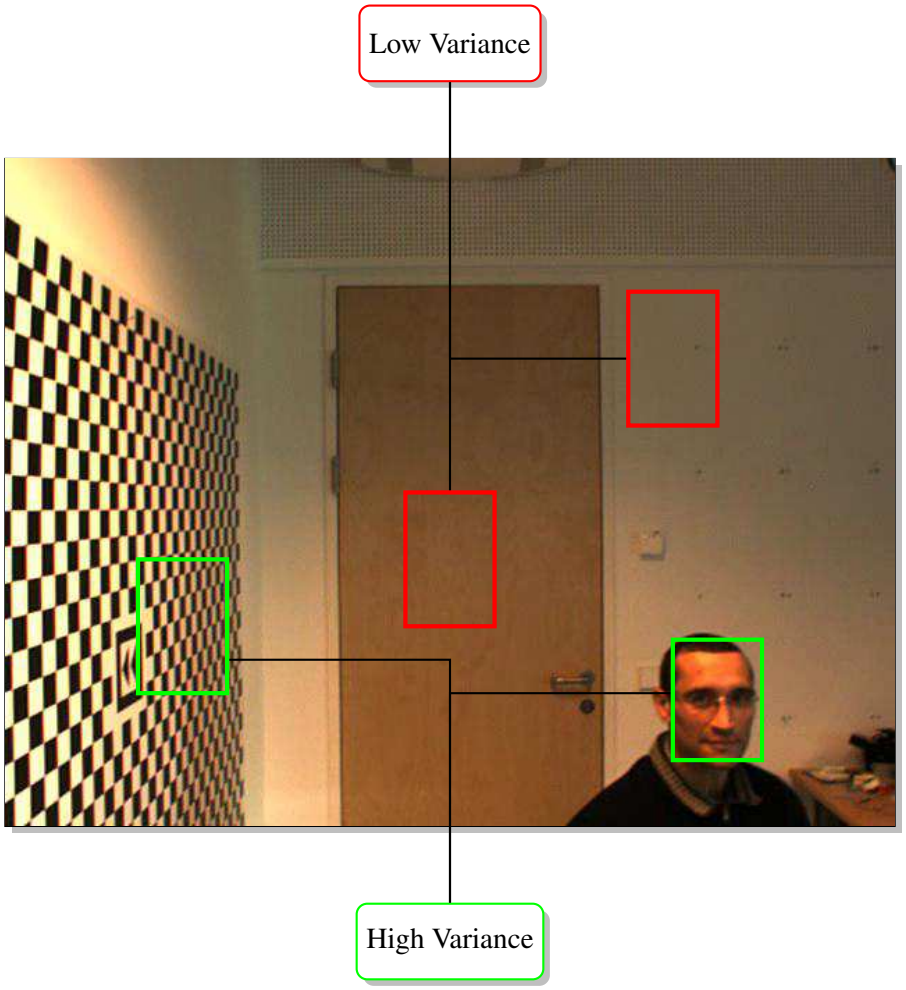


Figure 3.4: Uniform background regions are identified by setting a variance threshold.

considered as a one-dimensional vector of pixels and its elements are addressed using the notation x_i for the i^{th} pixel. For images, the variance σ^2 is defined as

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2, \tag{3.4}$$

where n is the number of pixels in the image and μ is

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{3.5}$$

An alternative representation of this formula is

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \mu^2. \tag{3.6}$$

the derivation for this formula is given in App. A.2.

In order to calculate σ^2 using Eq. 3.6 for an image patch of size n, n memory lookups are needed. By taking advantage of the fact that two overlapping image patches partially share the same pixel values, we will now show a way to calculate σ^2 for an image patch that uses only 8 memory lookups after transforming the input image I into two **integral images**. An integral image I' is of the same size as I and contains at location (x,y) the sum of all pixel values between the points $(1,1)$ and (x,y). This can be formulated as

$$I'(x,y) = \sum_{x' \leq x, y' \leq y} I(x',y'). \tag{3.7}$$

An integral image is computable in a single pass over the image by using the fact that $I'(x,y)$ can be decomposed into

$$I'(x,y) = I(x,y) + I'(x-1,y) + I'(x,y-1) - I'(x-1,y-1). \tag{3.8}$$

In Eq. 3.8 $I'(x,y) = 0$ for $x = 0$ or $y = 0$. By using the integral image representation, the computation of the sum of pixels up to a specific point no longer depends on the number of pixels in the patch. In Fig. 3.5, the sum of the pixel values within the rectangle $ABCD$ is obtained the following way. First, the sum of all pixels between $(0,0)$ and the point D is computed. Next, the pixels in the area between $(0,0)$ and B are subtracted as well as the pixels in the area between and $(0,0)$ and C. The area between $(0,0)$ and A must be added again, since it is subtracted twice. Using this observation, a formula for computing the sum of pixels within a bounding box B consisting of the parameters (x,y,w,h) is given by

$$\sum_{i=1}^{n} x_i = I'(x-1,y-1) - I'(x+w,y-1) - I'(x-1,y+h) + I'(x+h,y+w). \tag{3.9}$$

and use the notation

$$\sum_{i=1}^{n} x_i = I'(B). \tag{3.10}$$

as a shorthand for Eq. 3.9. We use Eq. 3.10 in order to calculate μ in Eq. 3.6. In order to calculate also the first term of the right-hand side of this equation using integral images, we modify Eq. 3.7 to use the squared value of $I(x,y)$. We get

$$I''(x,y) = \sum_{x' \leq x, y' \leq y} I(x',y')^2. \tag{3.11}$$

In analogy to Eq. 3.10 we write

$$\sum_{i=1}^{n} x_i^2 = I''(B). \tag{3.12}$$

By combining Eq. 3.5, Eq. 3.6, Eq. 3.10 and Eq. 3.12, we get

$$\sigma^2 = \frac{1}{n} I''(B) - \left[\frac{1}{n} I'(B)\right]^2. \tag{3.13}$$

This formula allows for a calculation of σ^2 by using eight memory lookups. In A.3 the maximum resolution for integral images and typical data types are given. For σ^2_{min}, we use half of the variance value found in the initial patch.

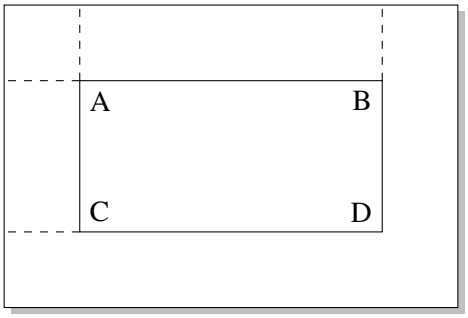


Figure 3.5: The sum of the pixel values within the rectangle *ABCD* is calculable by summing up the pixels up to *D*, subtracting the pixels up to both *B* and *C* and adding the pixels up to *A*. The computation is achieved using four look-ups when done on integral images.

3.4 Ensemble Classifier

In the third stage of the detection cascade we employ an ensemble classification method presented in [38] that is known as **random fern classification**. This classifier bases its decision on the comparison of the intensity values of several pixels. For each tested subwindow, a probability P_{pos} is calculated. If this probability is smaller than 0.5 the subwindow is rejected. This method of classification is slower than the variance filter, but still is very fast compared to classification methods using SIFT features, as it was experimentally evaluated in [38].

We use features that are proposed in [31]. Fig. 3.6 depicts the process of feature calculation. In this figure, a sample image to be classified is shown. In each of the four boxes below this image, a black and a white dot are shown. Each of these dots refers to a pixel in the original image. The positions of these dots are drawn out of a uniform distribution once at startup and remain constant. For each of these boxes it is now tested whether in the original image the pixel at the position of the white dot is brighter than the pixel at the position of the black dot. Mathematically, we express this as

$$f_i = \begin{cases} 0 & \text{if } I(d_{i,1}) < I(d_{i,2}) \\ 1 & \text{otherwise.} \end{cases} \tag{3.14}$$

In Eq. 3.14 $d_{i,1}$ and $d_{i,2}$ are the two **random** locations. Note that this comparison is invariant against constant brightness variations. The result of each of these comparisons is now interpreted as a binary digit and all of these values are concatenated into a binary number. In Fig. 3.6 the resulting binary number is 1101. When written in decimal form, this translates to 13, as it is shown in the box below the binary digit. The *i*th feature determines the value of the *i*th bit of a number. In Alg. 2, an algorithmic variant of this calculation is shown, in which I is the input image, F is the calculated feature value and S is the number of features to be used. The value of S influences the maximum feature value, which is $2^S - 1$. A feature group of size S, such as the one shown in Fig. 3.6 is referred to as a **fern** in [38]. The feature value obtained is used to retrieve the probability $P(y = 1 \mid F)$, where $y = 1$ refers to the event that the subwindow has a positive class label. These probabilities will be discussed in Chap. 4.

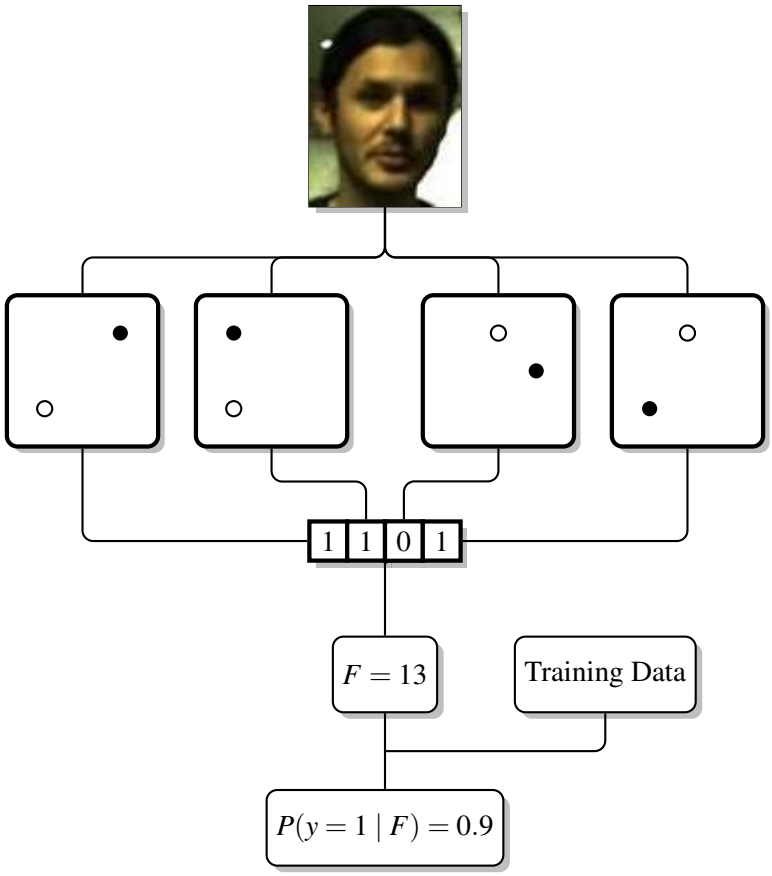


Figure 3.6: Feature Calculation for a single fern. The intensity values of the ith pixel pair determine the ith bit of the feature value. This feature value is then used to retrieve the posterior probability $P(y = 1 \mid F_k)$.

Algorithm 2 Efficient Fern Feature Calculation

Input: I
Output: F
  $F \leftarrow 0$
  for $i = 1 \ldots S$ **do**
    $F \leftarrow 2 \times F$
    if $I(d_{i,1}) < I(d_{i,2})$ **then**
      $F \leftarrow F + 1$
    end if
  end for

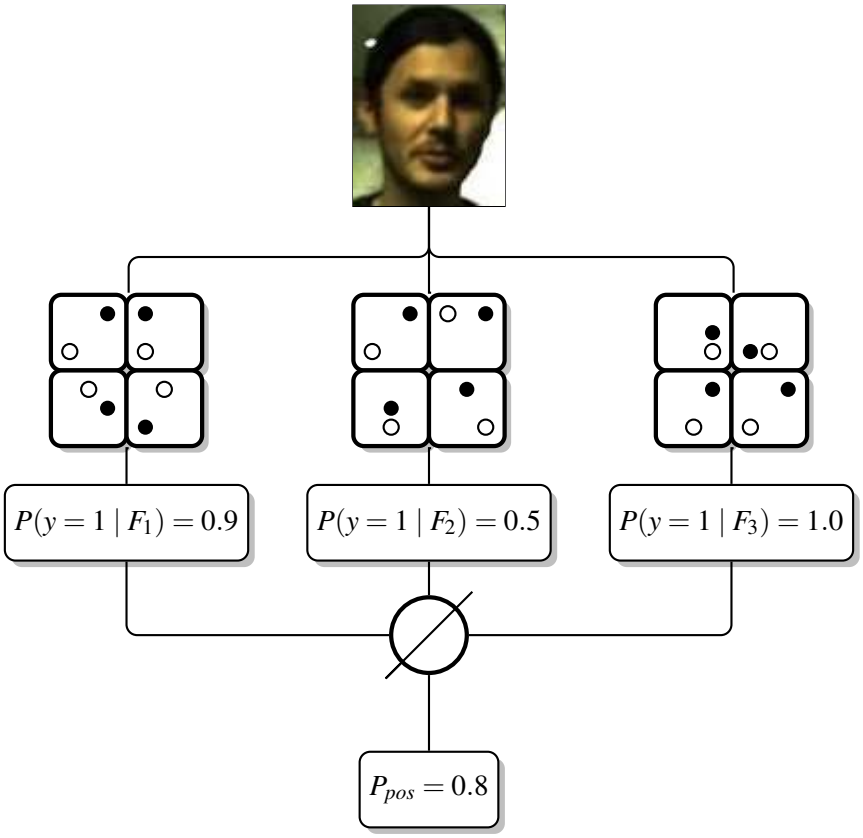


Figure 3.7: Ensemble classification using three random ferns. The posteriors $P(y = 1 \mid F_k)$ of each individual fern are averaged to produce a final confidence value P_{pos}.

With only one fern, it is necessary to use a large number of features to achieve satisfactory results [38]. However, the amount of training data needed to estimate the $P(y = 1 \mid F_k)$ increases with each additional feature. This problem is known as **curse of dimensionality** [36]. Amit and Geman [3] encounter the same problem when using randomised decision trees for character recognition and alleviate it by not using one large tree, but several smaller trees. They then average their output. This finding was adopted in [38] and leads to the classifier depicted in Fig. 3.7. Below the image to be classified, there are three ferns, each consisting of a different set of feature positions and each yielding a different value for $P(y = 1 \mid F_k)$. In the bottom rectangle, the average of these values P_{pos} is shown. P_{pos} is expressed as

$$P_{pos} = \frac{1}{M} \sum_{k=1}^{M} P(y = 1 \mid F_k). \tag{3.15}$$

In Eq. 3.15, M refers to the number of ferns used. M and S are evaluated empirically in Sec. 5.3.

3.5 Template Matching

In the fourth stage of the detector cascade we employ a template matching method. This stage is even more restrictive than the ensemble classification method described in the previous section, since the comparison is performed on a pixel-by-pixel level. We resize all patches to 15×15 pixels. For comparing two patches P_1 and P_2, we employ the Normalised Correlation Coefficient (NCC)

$$\mathrm{ncc}(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^{n} \frac{(P_1(x) - \mu_1)(P_2(x) - \mu_2)}{\sigma_1 \sigma_2}, \tag{3.16}$$

In Eq. 3.16 μ_1, μ_2, σ_1 and σ_2 are the means and standard deviations of P_1 and P_2. This distance measure is also known as the Pearson coefficient [40]. When interpreted geometrically, it denotes the cosine of the angle between the two normalised vectors [10]. NCC yields values between -1 and 1, with values closer to 1 when the two patches are similar. We use the following formula in order to define a distance between two patches that yields values between 0 and 1.

$$\mathrm{d}(P_1, P_2) = 1 - \frac{1}{2}(\mathrm{ncc}(P_1, P_2) + 1). \tag{3.17}$$

We maintain templates for both the positive and negative class. We refer to the positive class as $\mathcal{P}^+$ and to the negative class as $\mathcal{P}^-$. The templates are learned online, as it will be described in Sec. 4.2. In Fig. 3.8 positive and negative examples are shown that were learned on the sequence *Multi Face Turning* (see Sec. 5.4 for a description). Given an image patch P that is of unknown class label, we calculate both the distances to the positive class

$$d^+ = \min_{P_i \in \mathcal{P}^+} \mathrm{d}(P_0, P_i) \tag{3.18}$$

and the distance to the negative class

$$d^- = \min_{P_j \in \mathcal{P}^-} \mathrm{d}(P_0, P_j). \tag{3.19}$$

In Fig. 3.9, the green dots correspond to positive instances and red dots correspond to negative instances. The black dot labeled with a question mark corresponds to a patch with unknown class label. The distance to the nearest positive instance according to Eq. 3.18 is $d^+ = 0.1$ and the distance to the nearest negative instance according to Eq 3.19 is $d^- = 0.4$. We fuse these distances into a single value using the formula

$$p^+ = \frac{d^-}{d^- + d^+}. \tag{3.20}$$

Eq. 3.20 expresses the confidence whether the patch belongs to the positive class. A subwindow is accepted if p^+ is greater than a threshold θ^+. A confidence value above this threshold indicates that the patch belongs to the positive class. We use a value of $\theta^+ = 0.65$ for all of our experiments. In Fig. 3.10, p^+ is shown for all possible values. As it can be seen from this figure, p^+ is 1 if d^+ is 0. This corresponds to the event that an exact positive match has been found. If d^- is 0, then p^+ is 0. In the example depicted in Fig. 3.10, p^+ is 0.8.

(a) Positive Examples

(b) Negative Examples

Figure 3.8: Positive and negative patches acquired for the template matching method during a run on a sequence from the SPEVI dataset.

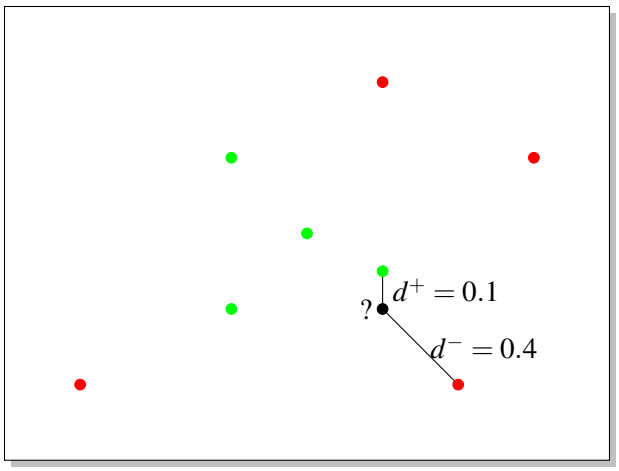


Figure 3.9: An unknown patch, labeled with a question mark, and the distance d^+ to the positive class and the distance d^- to the negative class. The distance is based on the normalised correlation coefficient. The actual space in which the distance measurement takes place has 15×15 dimensions.

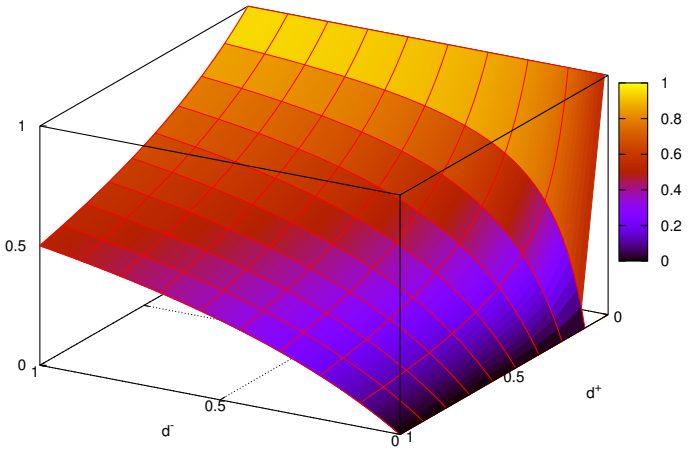


Figure 3.10: The confidence for a patch being positive depends on the distance to the closest positive d^+ and on the distance to the closest negative patch d^-.

3.6 Non-maximal Suppression

So far, the components of the detector cascade were introduced. Each subwindow is assigned a value p^+ that expresses the degree of belief whether it contains the object of interest. According to Blaschko [7], an object detection mechanism ideally identifies positive subwindows with a confidence value of 1 and negative subwindows with a confidence of 0, but in practice the output consists of hills and valleys characterizing intermediate belief in the fitness of a given location. In Fig. 3.11 this situation is illustrated. In this figure, several detections with high confidence occur around the true detection marked in green. Blaschko also states that considering only the subwindow yielding the highest confidence is problematic because this leads to other local maxima being ignored. Instead it is desirable to employ **non-maximal suppression** strategies that identify relevant local maxima.

For non-maximal suppression, we use the method described in [49] that clusters detections based on their spatial overlap. For each cluster, all bounding boxes are then averaged and compressed into a single detection. The confidence value p^+ of this single bounding box is then the maximum value confidence value in the corresponding cluster. In Fig. 3.12, B_1 is the area of the first bounding box, B_2 the area of the second bounding box and I is the area of the intersection of the two bounding boxes. For measuring the overlap between two bounding boxes, we use the formula from the PASCAL challenge [18]

$$overlap = \frac{B_1 \cap B_2}{B_1 \cup B_2} = \frac{I}{(B_1 + B_2 - I)}. \tag{3.21}$$

This measure is bounded between 0 and 1. We now use the **hierarchical clustering** algorithm described in [37] that works as follows. First we calculate the pairwise overlap between all

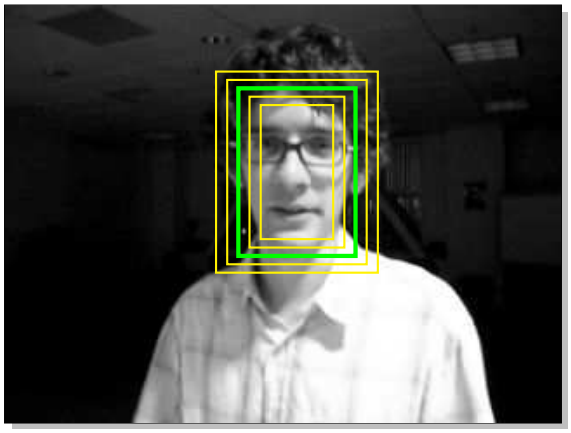


Figure 3.11: Overlapping subwindows with high confidence (yellow) are averaged and form a single result (green). The underlying image is from [41].

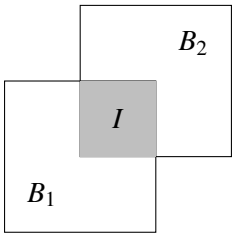


Figure 3.12: We define an overlap measure between two bounding boxes B_1 and B_2 to be the area of their conjunction I divided by the area of their disjunction $B_1 + B_2 - I$.

confident bounding boxes. We then start at one bounding box and look for the nearest bounding box. If this distance is lower than a certain cutoff threshold, we put them into the same cluster and furthermore, if one of the bounding already was in a cluster, we merge these clusters. If the overlap is larger than the cutoff threshold, we put them into different clusters. We then proceed to the next bounding box. We use a cutoff of 0.5 for all of our experiments.

3.7 Summary

In this chapter, we described how we run a cascade object detector on a subset of all possible subwindows in an input image. The components of the detection cascade were described and it was shown how overlapping positive bounding boxes are grouped into a single detection by employing non-maximal suppression. We explained that the foreground detector and the variance filter are able to rapidly reject subwindows, but require a preprocessing step. The ensemble classifier is computationally more expensive but provides a more granular decision mechanism. The final decision is obtained by comparing each subwindow to normalised templates. In Table 3.1 it is shown for each component how many memory lookups are necessary in order to test a subwindow.

Method	Memory Lookups
Foreground Detection	$\|C\|$
Variance Filter	8
Ensemble Classifier	$2 \cdot M \cdot S$
Template Matching	$255 \cdot \|\mathcal{P}^+\| \cdot \|\mathcal{P}^-\|$

Table 3.1: Comparison of necessary memory lookups for testing a subwindow. The variance filter and the ensemble Classifier operate at constant time (M and S are constant values). The foreground detection depends on the number of detected foreground regions $|C|$. The template matching method depends on the learned number of templates.

In Alg. 3 an algorithm for implementing the detection cascade is given. In Line 1 the set D_t that contains confident detection is initialised. In Lines 2-4 the required preprocessing for the foreground detection and the variance filter is shown. Lines 5-16 contain the actual cascade. In Line 11 the subwindow is added to the set of confident detections if it has passed all stages of the cascade. In Line 17 the non-maximal suppression method is applied.

Algorithm 3 Detection Cascade

1: $D_t \leftarrow \emptyset$
2: $F \leftarrow \text{foreground}(I)$;
3: $I' \leftarrow \text{integralImage}(I)$;
4: $I'' \leftarrow \text{integralImage}(I^2)$;
5: **for all** $B \in \mathcal{R}$ **do**
6: **if** $\text{isInside}(B, F)$; **then**
7: **if** $\text{calcVariance}(I'(B), I''(B)) > \sigma^2_{min}$ **then**
8: **if** $\text{classifyPatch}(I(B)) > 0.5$ **then**
9: $P \leftarrow \text{resize}(I(B), 15, 15)$
10: **if** $\text{matchTemplate}(I(B)) > \theta^+$ **then**
11: $D_t \leftarrow D_t \bigcup B$
12: **end if**
13: **end if**
14: **end if**
15: **end if**
16: **end for**
17: $D_t \leftarrow \text{cluster}(D_t)$

CHAPTER 4

Learning

When processing an image, both the recursive tracker and the object detector are run in parallel. In this chapter we deal with the question of how to combine the output of both methods into a single final result. We then show what happens during the learning step and when it is performed.

The background model and the threshold for the variance filter are not adapted during processing, while the ensemble classifier and the template matching method are trained online. We address the template update problem by defining certain criteria that have to be met in order to consider a final result suitable for performing a learning step. In learning, we enforce two P/N-learning constraints [27]. The first constraint requires that all patches in the vicinity of the final result must be classified positively by the object detector. The second constraint requires that all other patches must be classified negatively by the object detector.

The remainder of this chapter is organised as follows. In Sec. 4.1 it is shown how the results of the recursive tracker and the object detector are combined. Furthermore, the criteria for validity are given. In Sec. 4.2 it is explained how the constraints are implemented. In Sec. 4.3 the main loop of our approach is given. Sec. 4.4 concludes this chapter with a summary.

4.1 Fusion and Validity

In Alg. 4 our algorithm for fusing the result of the recursive tracker R_t and the confident detections D_t into a final result B_t is given. The decision is based on the number of detections, on their confidence values $p^+_{D_t}$ and on the confidence of the tracking result $p^+_{R_t}$. The latter is obtained by running the template matching method on the tracking result. If the detector yields exactly one result with a confidence higher than the result from the recursive tracker, then the response of the detector is assigned to the final result (Line 5 and 15). This corresponds to a **re-initialisation** of the recursive tracker. If the recursive tracker produced a result and is not re-initialised by the detector, either because there is more than one detection or there is exactly one detection that is less confident than the tracker, the result of the recursive tracker is assigned to the final result (Line 7). In all other cases the final result remains empty (Line 1), which suggests that the object is not visible in the current frame.

We use the predicate $valid(B_t)$ to express a high degree of confidence that the final result B_t is correct. Only if the final result is valid the learning step described in the next section is performed. As it is stated in Alg. 4 the final result is valid under the following two circumstances, both of which assume that the tracker was not re-initialised by the detector. The final result is valid if the recursive tracker produced a result with a confidence value being larger than θ^+ (Line 9). The final result is also valid if the previous result was valid and the recursive tracker produced a result with a confidence larger than θ^-. (Line 11). In all other cases, the final result is not valid. The first bounding box is always valid. As it was noted already in Sec. 3.5, the threshold θ^+ indicates that a result belongs to the positive class. The threshold θ^- indicates that a result belongs to the negative class and is fixed at $\theta^- = 0.5$ for all of our experiments.

Algorithm 4 Hypothesis Fusion and Validity.

Input: R_t, D_t
Output: B_t

1: $B_t \leftarrow \emptyset$
2: $valid(B_t) \leftarrow$ false
3: **if** $R_t \neq \emptyset$ **then**
4: **if** $|D_t| = 1 \wedge p^+_{D_t} > p^+_{R_t}$ **then**
5: $B_t \leftarrow D_t$
6: **else**
7: $B_t \leftarrow R_t$
8: **if** $p^+_{R_t} > \theta^+$ **then**
9: $valid(B_t) \leftarrow$ true
10: **else if** $valid(B_{t-1}) \wedge p^+_{R_t} > \theta^-$ **then**
11: $valid(B_t) \leftarrow$ true
12: **end if**
13: **end if**
14: **else if** $|D_t| = 1$ **then**
15: $B_t \leftarrow D_t$
16: **end if**

4.2 P/N-Learning

According to Chapelle [12], there are two fundamentally different types of tasks in machine learning. In **supervised learning** a training set is created and divided into classes manually, essentially being a set of pairs $\langle x_i, y_i \rangle$. The x_i correspond to training examples and the y_i to the corresponding class label. The training set is used to infer a function $f : X \rightarrow Y$ that is then applied to unseen data. Supervised learning methods in object detection have been successfully applied most notably to face-detection [49] and pedestrian detection [16]. However, the learning phase prevents applications where the object to be detected is unknown beforehand. In the same way, the learned classifier is also unable to adapt to changes in the distribution of the data. The second task in machine learning is **unsupervised learning**. In this setting, no class labels

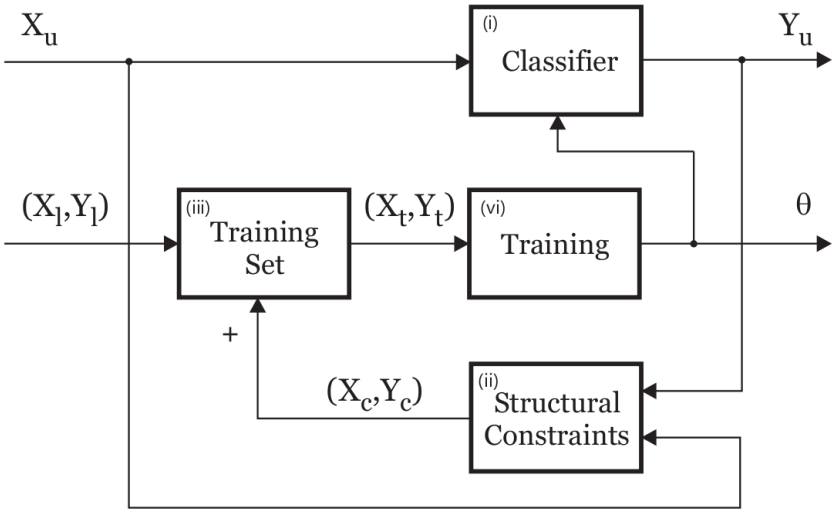


Figure 4.1: P/N Constraints. Image is from [27].

are available and the task is finding a partitioning of this data, which is achieved by density estimation, clustering, outlier detection and dimensionality reduction [12].

Between these two paradigms there is **semi-supervised learning**. In semi-supervised learning, there are labeled examples as well as unlabeled data. One type of semi-supervised learning methods uses the information present in the training data as supervisory information [12] in order to find a class distribution in the unlabeled data and to update the classifier using this class separation as a training set. In our tracking setting there is exactly one labeled example. In [27], a semi-supervised learning method called **P/N-learning** is introduced. This method shows how so-called structural constraints can extract training data from unlabeled data for binary classification. In P/N-learning, there are two types of constraints: A P-constraint identifies false negative outputs and adds them as positive training examples. An N-constraint does the opposite. As it is depicted in Fig. 4.1, X_u refers to the unlabeled data available. This data is first classified by an existing classifier that assigns labels Y_u to X_u. Then, the structural constraints, according to some criterion, identify misclassified examples X_c with new labels Y_c. These examples are then added to the training set and training is performed, which results in an update of the classification function.

We use the following constraints for object detection that are proposed in [27]. The P-Constraint requires that all patches that are highly overlapping with the final result must be classified as positive examples. The N-Constraint requires that all patches that are not overlapping with the valid final result must be classified as negative examples. We consider a bounding box B highly overlapping with B_t if it exhibits an overlap of at least 60%. B is considered not to be overlapping with B_t if the overlap is smaller than 20%. For measuring overlap, we employ the metric already described in Sec. 3.6.

A complete algorithmic description of the constraints is given in Alg. 5. We will now describe the measures that we take in order to adapt the ensemble classifier and the template matching method in order to classifiy these examples correctly. For the ensemble classifier, we have

not yet explained how the posterior values are calculated for each fern. Recall that $P(y = 1 \mid F_k)$ is the probability whether a patch is positive given the features F_k. We define the posterior to be

$$P(y = 1 \mid F_k) = \begin{cases} \frac{p_{F_k}}{p_{F_k} + n_{F_k}}, & \text{if } p_{F_k} + n_{F_k} > 0 \\ 0, & \text{if } p_{F_k} + n_{F_k} = 0. \end{cases} \tag{4.1}$$

In Eq. 4.1 p_{F_k} is the number of times the P-constraint was applied to this combination of features and n_{F_k} is the number of times the N-constraints was applied. In Line 2 we test whether a bounding box overlapping with the final result is misclassified by the ensemble classifier. We increment p_{F_k} in Line 5 for each fern if the overlap is smaller than 0.6 and the ensemble classifier yielded a confidence lower than 0.5. In Line 10 n_{F_k} is incremented for misclassified negative patches. When updating the ensemble classifier, the computational overhead does not increase. This is different for the template matching method, as every additional patch in the set of positive or negative templates increases the number of comparisons that must be made in order to classify a new patch. In order to change the label of a misclassified positive patch for the template matching method, we add it to the set of positive templates. This patch then has a distance of $d^+ = 0$, which means that its confidence is 1. However, as it is shown in Line 18, we do this only for the patch contained in the final result B_t. Note that the learning step is performed only if the final result is valid, which already implies that $p^+_{B_t}$ is larger than θ^-. As for the N-constraint for the template matching method, we add negative patches to the template matching method if they were misclassified by the ensemble classifier and also are misclassified by the template matching methods.

In Fig. 4.2 it is illustrated when positive templates are added. At point A, tracking starts with a high confidence and a valid result. Learning is performed, but no positive examples are added, because the confidence is above θ^+. The confidence then drops below θ^+ (Point B) but remains above θ^-. According to Alg. 4 this means that the final result is still valid. Exactly in this case positive patches are added, which leads to an increased confidence. At Point C, the confidence drops below θ^-, which leads to the final result not being valid anymore. In Point D, confidence is at the same level as in Point B, but no learning is performed since the final result is not valid.

4.3 Main Loop

We now have described all components that are used in our approach. In Alg. 6 the main loop of our implementation is given. When the initial patch is selected, a learning step is performed (Line 1). For each image in the sequence, the tracker and the detector are run (Line 3 and 4), their result is fused (Line 5) and if the final result is considered valid then the learning step is performed (Line 7). The final result is then printed (Line 9) and the next image is processed.

4.4 Summary

In this chapter, we described that we obtain a final result based on the results of the recursive tracker and the object detector by basing our decision on the confidence of the template matching method run on both results. We further defined criteria for validity based on the confidence value

Algorithm 5 Applying P/N-constraints

Input: I, B_t

for all $B \in \mathcal{R}$ **do**
  if overlap$(B, B_t) > 0.6$ **and** classifyPatch$(I(B)) < 0.5$ **then**
    for $k = 1 \ldots M$ **do**
      $F \leftarrow$ calcFernFeatures$(I(B), k)$
      $p_{F_k}[F] \leftarrow p_{F_k}[F] + 1$
    end for
  else if overlap$(B, B_t) < 0.2$ **and** classifyPatch$(I(B)) > 0.5$ **then**
    for $k = 1 \ldots M$ **do**
      $F \leftarrow$ calcFernFeatures$(I(B), k)$
      $n_{F_k}[F] \leftarrow n_{F_k}[F] + 1$
    end for
    if $p^+_{B_t} > \theta^-$ **then**
      $\mathcal{P}^- \leftarrow \mathcal{P}^- \cup I(B)$
    end if
  end if
end for
if $p^+_{B_t} < \theta^+$ **then**
  $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup I(B_t)$
end if

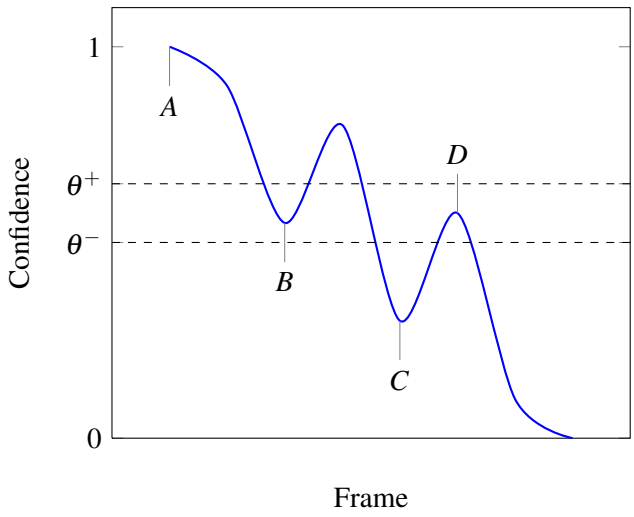


Figure 4.2: Positive examples are added only when the confidence value drops from a high value (Frame A) to a value between θ^+ and θ^- (Frame B). In Frame D no learning is performed, since the confidence rises from a low value in Frame C. This measure ensures that the number of templates is kept small.

Algorithm 6 Main loop

Input: $I_1 \dots I_n, B_1$

learn(I_1, B_1)
for $t = 2 \dots n$ **do**
  $R_t \leftarrow$ track(I_{t-1}, I_t, B_{t-1})
  $D_t \leftarrow$ detect(I_t)
  $B_t \leftarrow$ fuse(R_t, D_t)
  if valid(B_t) **then**
    learn(I_t, B_t)
  end if
  print(B_t)
end for

and of the validity of the previous bounding box. We perform learning only if the final result is valid. The learning step consists of identifying falsely labeled examples and updating the ensemble classifier and the template matching method in order to correctly classify them.

CHAPTER 5

Results

In this chapter, our approach is evaluated empirically both on sequences that have been used in the literature as well as on newly recorded sequences. We employ the standard metrics recall and precision for assessing performance. For this evaluation, a C++ implementation was created, where the calculation of the optical flow (Sec. 2.1), the calculation of the normalised correlation coefficient (Seq. 3.5), all the operations for the foreground detection (Sec. 3.2) as well as low-level image operations are implemented as function calls to the OpenCV[1] library. The multi-threaded optimisation described in Sec. 3.1 was implemented using an OpenMP[2] pragma. All experiments were conducted on an Intel Xeon dual-core processor running at 2.4 Ghz.

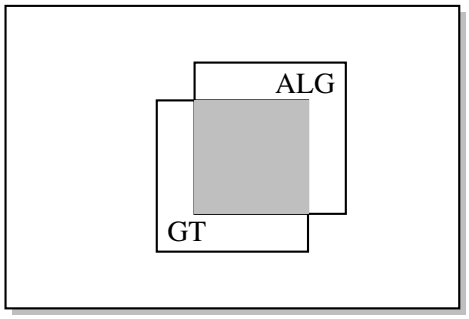
This chapter is organised as follows. Sec. 5.1 explains the evaluation protocol. In Sec. 5.2, the video sequences that are used in our experiments are described. In Sec. 5.3, the parameters for the ensemble classifier are evaluated empirically. In Sec. 5.4, qualitative results are shown on two video sequences. The requirement set on the overlap when comparing to ground truth is discussed in Sec. 5.5. In Sec. 5.6 quantitative results for performance and execution time are obtained for our approach and two state-of-the-art methods. In Sec. 5.7 our algorithm is evaluated in a multi-camera scenario. Each experiment is concluded with a discussion.

5.1 Evaluation Protocol

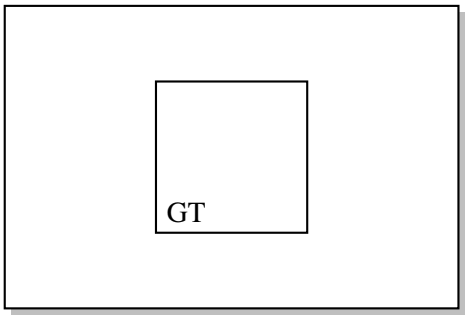
In order to compare the output of an algorithm to ground truth values we use the overlap measure from Eq. 3.21. In [24] it is shown that this measure equally penalizes translations in both directions and scale changes. Based on the overlap between algorithmic output and ground truth values, each frame of a sequence is categorised as one of the five possible cases shown in Fig. 5.1. A result is considered **true positive** if the overlap is larger than a threshold ω (Case a). A result is counted as **false negative** when the algorithm yields no result for a frame even though there is an entry in the ground truth database (Case b). The opposite case is a **false positive** (Case c).

[1] Open Computer Vision: http://opencv.willowgarage.com
[2] Open Multi-Processing: http://openmp.org

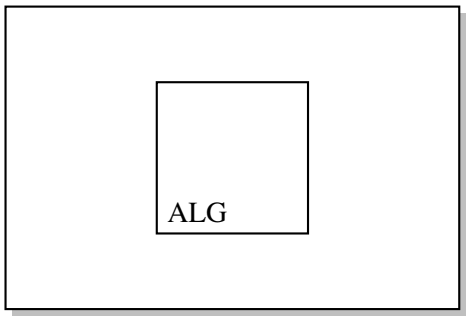
If the overlap is lower than the threshold ω, then this is counted both a false negative and a false positive (Case d). If for a frame neither an algorithmic output nor an entry in the ground truth database exists then this case is considered a **true negative** (Case e).



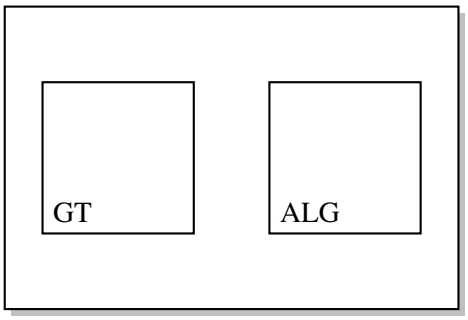
(a) True Positive



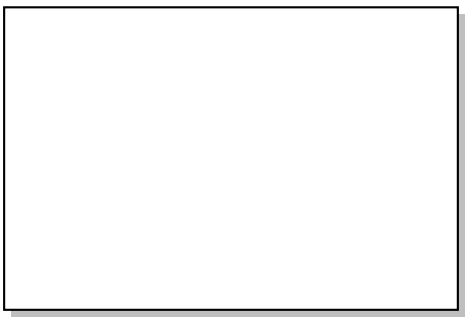
(b) False Negative



(c) False Positive



(d) False Negative and False Positive



(e) True Negative

Figure 5.1: Five possible cases when comparing algorithmic results to ground truth values. The case (e) is not considered in metrics that we use.

After processing a video sequence, all occurrences of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) are counted. Based on these values we calculate two performance metrics. Recall is defined as

$$recall = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}. \tag{5.1}$$

Eq. 5.1 measures the fraction of positive examples that are correctly labeled [17]. Precision is defined as

$$precision = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}. \tag{5.2}$$

Eq. 5.2 measures the fraction of examples classified as positive that are truly positive [17]. Depending on the application, high recall or high precision (or both) may be demanded. Since our approach uses random elements, we repeat every experiment five times and average the values for precision and recall.

Our algorithm provides a confidence measure for each positive output it produces. By applying thresholding, results exhibiting a confidence lower than a certain value θ are suppressed. This suppression affects the performance metrics as follows. True positives with a confidence less than θ become false negatives, meaning that both recall and precision get worse. Also, false positives with a confidence less than θ become true negatives, meaning that precision improves. Thresholding is most effective if false positive results are produced with low confidence values and true positive results with high confidence values. A precision-recall curve visualise how different values for θ impact precision and recall. A sample curve is given in Fig. 5.2, where the right bottom end of the curve refers to the precision and recall values when the threshold is 0, meaning that no true positives and no false positives were removed. The rest of the curve represents recall and precision values as θ is increased, ending in a point where $\theta = 1$. We chose not to use precision-recall curves because it turned out during experiments that no relevant improvement for precision was achievable. We attribute this to the mechanism for automatic failure detection described in Sec. 2.3 that prevents false positives.

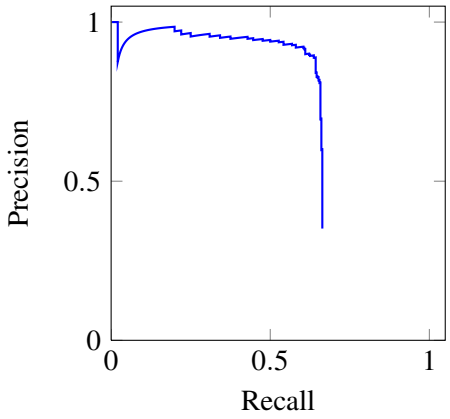


Figure 5.2: Sample precision-recall curve.

5.2 Sequences

In this section we describe the sequences that we use for evaluation, all of which are accompanied by manually annotated ground truth data. **Multi Face Turning** from the SPEVI[3] dataset

[3] Surveillance Performance EValuation Initiative: http://www.eecs.qmul.ac.uk/~andrea/spevi.html

consists of four people moving in front of a static camera, undergoing various occlusions and turning their faces right and left. The difficulty in this sequence lies in the fact that four instances of the same category are present and that all persons undergo various occlusions. The sequence consists of 1006 individual frames. The sequence **PETS view 001** is taken from the PETS 2009[4] dataset. It shows pedestrians walking across a T junction and consists of 794 frames. The pedestrians exhibit a similar appearance due to low spatial resolution. The following six sequences that are shown in Fig. 5.3 were used in [51, 27] for evaluating object tracking methods. The sequence **David Indoor**[5] consists of 761 frames and shows a person walking from an initially dark setting into a bright room. No occlusions occur in this sequence. **Jumping** consists of 313 frames and shows a person jumping rope, which causes motion blur. **Pedestrian 1** (140 frames), **Pedestrian 2** (338 frames) and **Pedestrian 3** (184 frames) show pedestrians being filmed by an unstable camera. The sequence **Car** consists of 945 frames showing a moving car. This sequence is challenging because the images exhibit low contrast and various occlusions occur.

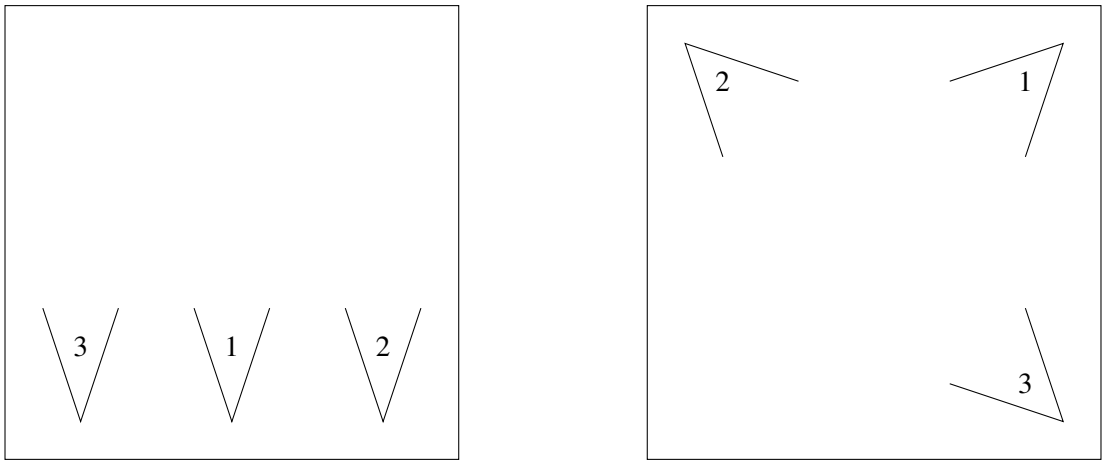
(a) David Indoor (b) Jumping (c) Pedestrian 1

(d) Pedestrian 2 (e) Pedestrian 3 (f) Car

Figure 5.3: The data set used for analysing the importance of the overlap measure and for comparing to existing approaches.

We recorded two datasets, **Multi Cam Narrow** and **Multi Cam Wide**, each consisting of three sequences, for the evaluation in multi-camera scenarios. The camera positions and orientations for both datasets are shown in Fig. 5.4. In *Multi Cam Narrow* the cameras were placed next to each other, each camera looking in the same direction. For *Multi Cam Wide* each camera was placed in a corner of the room, facing to its center. In Fig. 5.5 a frame from each camera

[4]Performance Evaluation of Tracking and Surveillance: http://www.cvg.rdg.ac.uk/PETS2009/a.html
[5]http://www.cs.toronto.edu/~dross/ivt/



(a) Multi Cam Narrow (b) Multi Cam Wide

Figure 5.4: Camera setup for *Multi Cam Narrow*.

Figure 5.5: Sample frames from *Multi Cam Narrow*.

in *Multi Cam Wide* is shown recorded at the same instant of time. In Fig. 5.6 a frame from each sequence in *Multi Cam Wide* is shown as well as enlarged views of the selected object.

5.3 Parameter Selection for Ensemble Classifier

In this experiment we analyse the effects of varying the parameters for the ensemble classifier on the sequence *Multi Face Turning*. The two parameters in question are the number of features in a group (S) and the total number of groups (M). In [38] it was concluded that setting $S = 10$ and $M = 10$ gives good recognition rates. Since we do not employ random ferns as a final classification step but as part of a detector cascade and furthermore use on online learning approach, we re-evaluate these parameters. Breiman [9] shows that randomized decision trees do not overfit as the number of trees is increased but produce a limiting value of the generalization error. This means that increasing M does not decrease recall.

Since S is the number of features that are assumed to be correlated, for large values of S the curse of dimensionality arises, meaning that the amount of training data increases with S. On the

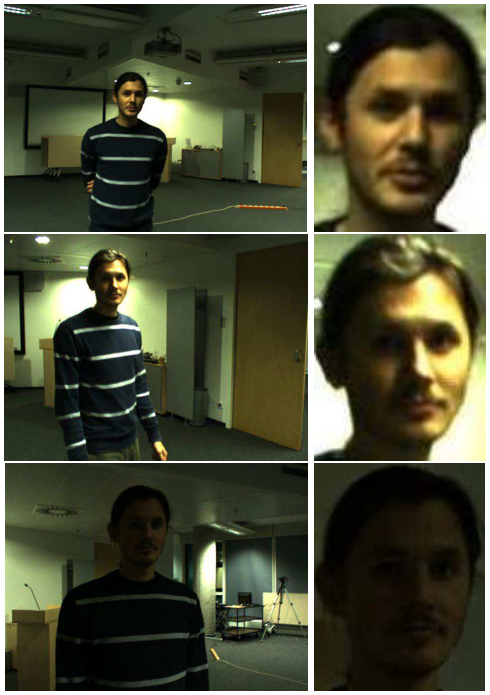


Figure 5.6: Sample frames of *Multi Cam Wide* and an enlarged view of the selected object.

other hand, low values of S ignore correlation in the data [38]. Another aspect for choosing S is the amount of memory required. For $S = 24$, at least 16,777,216 entries for the posterior values have to be stored for every fern. For this experiment, we set $M = 50$ and let S vary from 1 to 24. In Fig. 5.7, S is plotted against the achieved recall. The recall first increases linearly with S and reaches its maximum at $S = 13$. For higher values of S, less recall is achieved.

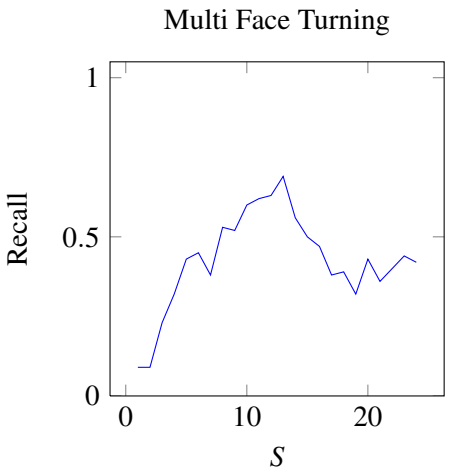


Figure 5.7: Varying the size S of the feature groups when $M = 50$.

Since the time spent on testing each sliding window depends linearly on M, this means that M should be small when low execution time is desired. Fig. 5.8 shows how the recall is affected by M. For this experiment, we set $S = 13$. The results show that recall increases up to $M = 30$ and does not change afterwards.

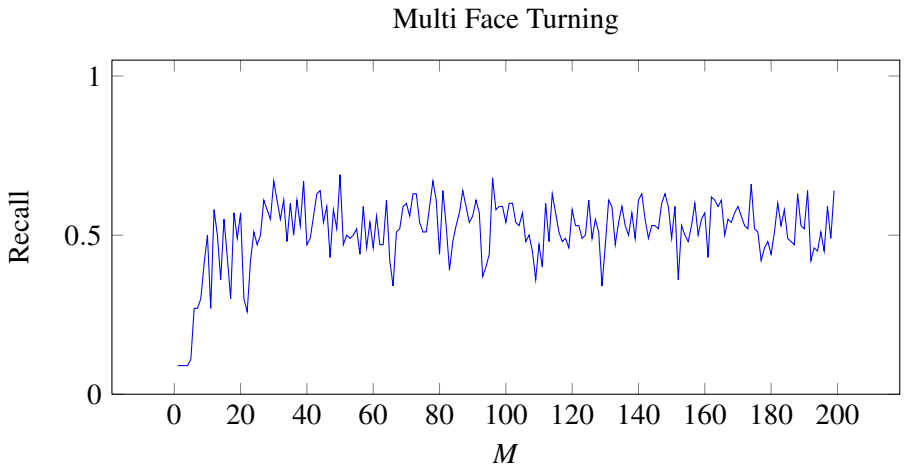


Figure 5.8: Varying the number of ferns M when $S = 13$.

Discussion

This experiment shows that the parameters of the ensemble classifier influence recall. We get best results for $S = 13$ and $M > 30$. The observations here are in line with the finding in [9] that randomised decision trees do not overfit as more trees are added. For the rest of the experiments in this chapter, we set $S = 13$ and $M = 10$, as a compromise between recall, speed and memory consumption.

5.4 Qualitative Results

In this section, qualitative results for the sequences *Multi Face Turning* and *PETS view 001* are given. In all of the presented images, a blue bounding box denotes a result with its confidence being larger than 0.5, all other bounding boxes are drawn in yellow. We show results for *Multi Face Turning* in Fig. 5.9. In the top left image, the face initially selected for tracking is shown. The person then moves to the right and gets occluded by another person. The recursive tracker correctly stops tracking and the face is detected as it appears to the right of the occluding person. This is shown in the top right image. The selected face then turns left and right several times and undergoes another occlusion, which is again handled correctly, as it is depicted in the first image in the second row. The selected person then leaves the camera view on the right and enters on the same side in the back of the room. At first, the head of the person is rotated but as soon as a position frontal to the camera is assumed a detection occurs, as it is shown in the second image in the second row. In the first image in the third row the recursive tracker does not stop tracking

even though an occlusion occurred, however no learning is performed since the distance to the model remains below 0.5, as indicated by the yellow bounding box. The detector then yields a detection with a higher confidence than the recursive tracker, as it is depicted in the second image in the third row. The person then leaves the field of view on the left-hand side and enters at a position close to the camera (see first image in the fourth row), appearing twice as large as in the first frame. The face is detected and tracked correctly until the last appearance of the selected face, depicted in the second image in the fourth row.

Results for *PETS view 001* are shown in Fig. 5.10. The person shown in the top-left image is selected for tracking. The person walks to the left and gets almost fully occluded several times by the pole in the middle of the image. Additional occlusions occur when walking behind two other people. The tracker stops and the detector correctly re-initialises the target as soon as it reappears, which is shown in the second image in the first row. The person then returns to the right-hand side of the image and leaves the field of view while being tracked correctly, as it is depicted in the first image of the second row. In the second image of the second row a false detection occurs. The original target then appears on the right-hand side of the field of view and causes another detection, as it is shown in the first image of the third row. The person then walks to the left and gets occluded by another person, as depicted in the second image in the third row. Both persons walk in opposite directions, which causes the recursive tracker to erroneously remain at the current position for the following 297 frames with a confidence lower than 0.5. The target person in the meantime walks around the scene without being re-detected. Finally, a correct detection occurs in the last image presented and the person is tracked correctly until the end of the sequence.

Discussion

The experiments discussed in this section demonstrate that our method is able to learn the appearance of objects online and to re-detect the objects of interest after occlusions, even at different scales. On *Multi Face Turning*, the face is re-detected after every but one occlusion. In both the first and the second experiment there are frames where the object of interest is not detected even though it is visible. In *Multi Face Turning* this is due to the face appearing in a pose that never appeared before, which is a situation that our method cannot handle. In the second experiment, the object of interest is not detected even it appears in poses similar to the ones encountered in learning steps. The problem here is that during learning, the other persons walking around the scenes are (correctly) recognised as negative examples. However, since the appearance of these persons is similar to the appearance of the object of interest, a close positive match is needed in order to achieve a detection. In *Multi Face Turning* the faces exhibit sufficiently dissimilarity.

5.5 Requirement on Overlap

As it was pointed out in Sec. 5.1, the categorisation of algorithmic output depends on the parameter ω that defines the required overlap between an algorithmic result and ground truth values. When ω is increased, both recall and precision decrease. Depending on the intended application,

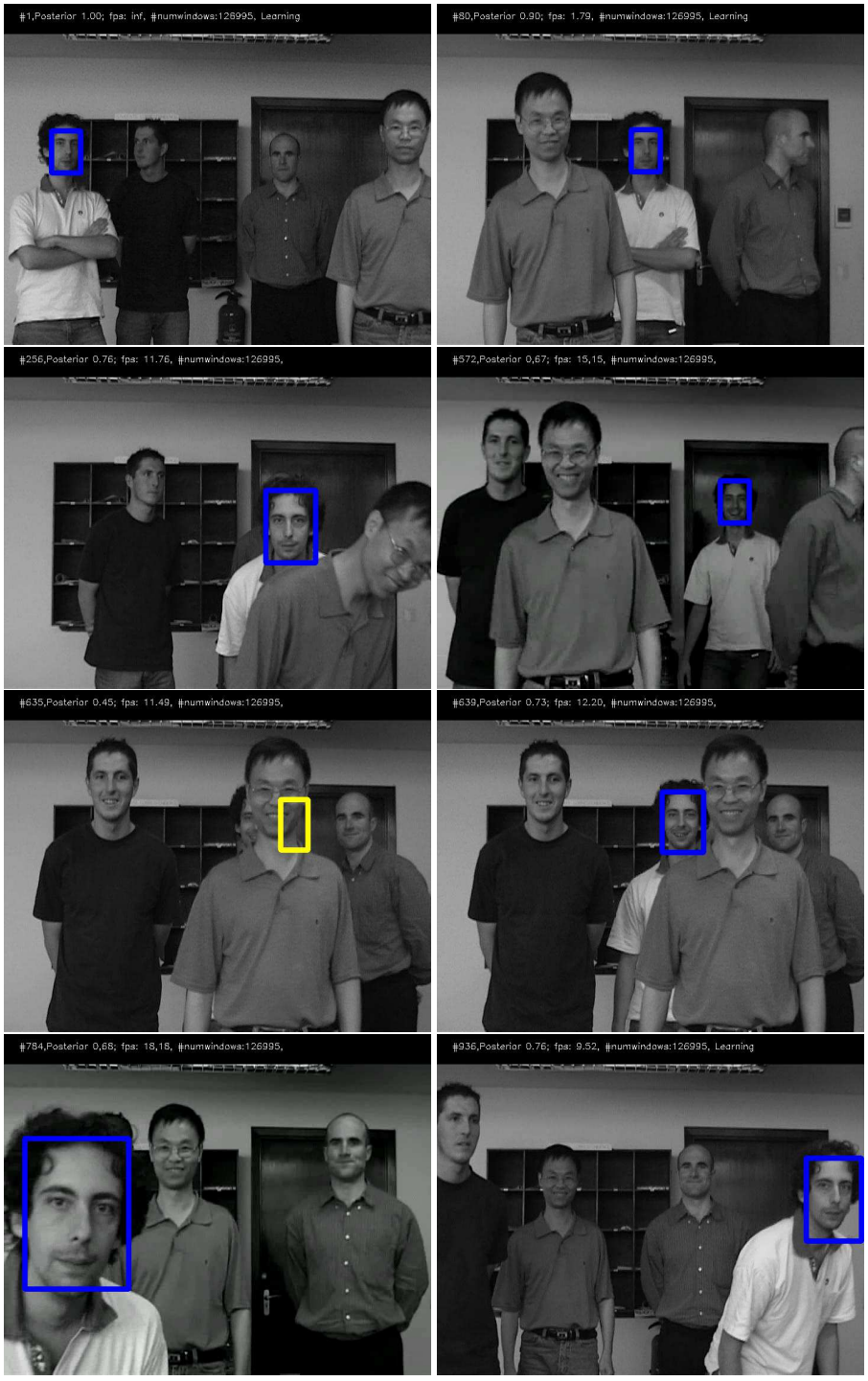


Figure 5.9: Qualitative results for the sequence *Multi Face Turning*.

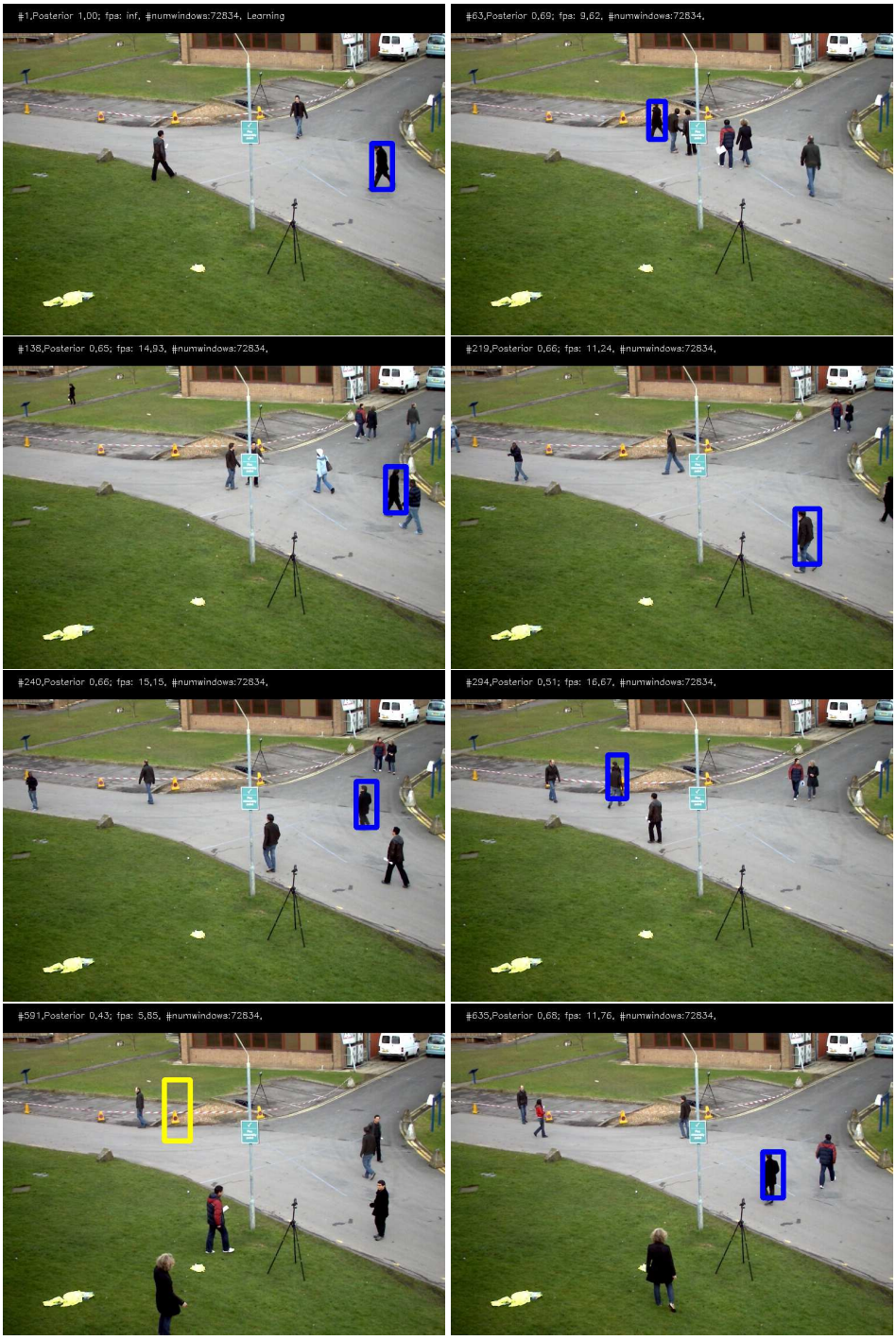


Figure 5.10: Qualitative results for the sequence *PETS view 001*.

different requirements on accurate results are desired. In this section we analyse empirically to what extent precision and recall change when varying ω. For this experiment, we use three different values for ω (0.25, 0.5 and 0.75) and evaluate precision and recall on the sequences *David Indoor*, *Jumping*, *Pedestrian 1-3* and *Car*. The resulting values for precision and recall that are obtained when running our implementation with the three different values for ω are shown in Table 5.1.

	$omega = 0.75$	$omega = 0.5$	$omega = 0.25$
David Indoor	0.08/0.08	0.65/0.65	0.66/0.66
Jumping	0.13/0.14	0.63/0.63	1.00 1.00
Pedestrian 1	0.00/0.00	0.04/0.04	1.00 1.00
Pedestrian 2	0.06/0.11	0.55/1.00	0.56/1.00
Pedestrian 3	0.22/0.35	0.32/0.51	0.32/0.51
Car	0.08/0.09	0.95/0.94	0.95/0.94

Table 5.1: Performance metrics improve when the requirement on the overlap ω is relaxed. The first value in a cell denotes recall, the second value is precision.

Discussion

In the conducted experiment, for all sequences there is a drastic increase in both recall and precision when the requirement of an exact match is relaxed. This is most visible for the sequence Pedestrian 1, where the recall increases from 0 ($\omega = 0.75$) over 0.04 ($\omega = 0.5$) to 1 ($\omega = 0.25$). We attribute this to two causes. First, the process of manually annotating video data is ambiguous [6]. The algorithmic output therefore might not completely overlap with the ground truth data. Another source for inaccuracies is the recursive tracker, since small errors accumulate and let the tracker drift away from the original target until it is re-initialised. The effect on performance is reduced by setting a low requirement on overlap. An overlap requirement of $\omega = 0.25$ is sufficient for our intended applications and we perform the rest of the experiments with this setting.

5.6 Comparison to State-of-the-Art Methods

In this section our approach is evaluated quantitatively with respect to precision, recall and execution time. We compare our approach to two other tracking approaches. **TLD** is proposed in [27] and **Struck** is proposed in [23]. Both approaches are briefly described in Sec. 1.2. An implementation for *Struck* was obtained from the respective project website[6]. The results for *TLD* were taken directly from [27]. The parameters of *Struck* were left at their default values. For this evaluation the dataset from the previous section is used, which also used in [51] for a comparison of tracking systems. In Table 5.2, the values for recall and precision are shown. The

[6] http://www.samhare.net/research/struck

first value in each table cell is the recall, the second value is precision. The maximum recall values for a sequence are emphasised.

	Ours	**TLD [27]**	**Struck [23]**
David Indoor	0.66/0.66	**0.94**/0.94	0.23/0.23
Jumping	**1.00**/1.00	0.86/0.77	**1.00**/1.00
Pedestrian 1	**1.00**/1.00	0.22/0.16	0.84/0.84
Pedestrian 2	0.56/1.00	**1.00**/0.95	0.26/0.21
Pedestrian 3	0.32/0.51	**1.00**/0.94	0.67/0.57
Car	**0.95**/0.94	0.93/0.83	0.88/0.80

Table 5.2: Comparative performance metrics. The first value in a cell denotes recall, the second is precision. Best recall values for a sequence are emphasised.

We also present a comparison of the time needed for execution for the selected algorithms. For this evaluation, each algorithm was run 5 times on the sequence *Jumping* and the average was calculated. The results are presented in Table 5.3. In the second column the average time in seconds needed to process the sequence as a whole is given. In the third column, the number of frames in the sequence divided by the processing time is given.

Method	**Time (s)**	**Frames / s**
Ours (multi-threaded)	14.12	22.04
Ours (single-threaded)	18.39	17.02
TLD	38.17	8.20
Struck	809.57	0.38

Table 5.3: Speed comparison on *Jumping*.

Discussion

Our algorithm achieves the highest recall on three out of six sequences and ties on one sequence with *Struck*. The highest recall for the three other sequences is achieved by *TLD*. We explain the different results for the original implementation of *TLD* and our implementation with the different features that are used for the ensemble classifier. *Struck* yields good results as well but performs worse on sequences with occlusions. Our multi-threaded implementation takes least computing time. It is almost three times as fast as the original *TLD*. This is caused by the use of random ferns over 2-bit-binary patterns, the addition of a variance filter in the detection cascade and the implementation in C++. Compared to *Struck*, our implementation is 57 times faster.

5.7 Evaluation on Multiple Cameras

For evaluating the applicability in multi-camera scenarios we use the sequences *Multi Cam Narrow* and *Multi Cam Wide*. First, we select an initial bounding box in the first frame the object appears in the sequence. For *Multi Cam Narrow*, we select the left-most face on all sequences. For *Multi Cam Wide* we select the face of the person initially to the right. We then let our algorithm run without intervention for the rest of the sequence. When processing is done, we export all the data that was learned, which are the variance of the initial patch, the posterior values of the ensemble classifier $P(y = 1 \mid F_k)$ and the positive and negative patches of the nearest neighbor classifier $\mathcal{P}^+$ and $\mathcal{P}^-$. We call this combination of data the **model** of the object. We now apply the extracted model without any manual initialisation to all sequences in the data set, including the one it was generated with. The obtained recall values for *Multi Cam Narrow* are in Table 5.4. The diagonal values show results where a model was applied in a camera where it was learned initially. The minimum recall value for all results is 0.45, which was achieved when the model from camera 3 was applied in camera 2. In all sequences, false detections occur, but are recovered from. The model recorded in camera 2 scores best in all sequences. The results for *Multi Cam Wide* are in Table 5.5. Ignoring the diagonal values, 2 out of 6 model-camera combinations achieved a recall larger than 0. The best recall value 0.54 was obtained when running the model from camera 3 in camera 1. In all cases where recall is 0, the object is not detected over the course of the image sequence.

		Model learned in		
		Cam 1	**Cam 2**	**Cam 3**
Model applied to	**Cam 1**	0.60	0.79	0.69
	Cam 2	0.78	0.80	0.45
	Cam 3	0.85	0.87	0.84

Table 5.4: Recall for *Multi Cam Narrow*.

		Model learned in		
		Cam 1	**Cam 2**	**Cam 3**
Model applied to	**Cam 1**	0.78	0.00	0.54
	Cam 2	0.16	0.66	0.00
	Cam 3	0.00	0.00	0.44

Table 5.5: Recall for *Multi Cam Wide*.

Discussion

The experiments described in this section give insight about the applicability in scenarios with multiple cameras. It can be seen from the results for dataset *Multi Cam Narrow* that in principle

it is possible to learn an object in one camera and apply it in a second camera. However, as the baseline between the cameras increases, results get worse. The second image column of Fig. 5.6 depicts the difficulties present in *Multi Cam Wide*. In this column, the selected face is shown as it appears in the three different cameras. Due to varying lighting conditions the appearance of an object in one camera is distorted significantly compared to the two other cameras. One could alleviate this problem by employing dedicated lighting equipment in order to create homogenous lighting settings in all three cameras, but there are application scenarios where this is not feasible, for instance when images are taken from a surveillance camera.

CHAPTER 6

Conclusion

In this work we presented an implementation of a novel approach to robust object tracking based on the Tracking-Learning-Detection paradigm. We made the following contributions: We reproduced the results of Kalal et al. and showed that the use of features based on pairwise pixel comparison and two additional stages in the detection cascade lead to a reduced computing time and do not degrade results. In our implementation, we reduce computing time by a factor of three. For a GPU implementation, we expect a further reduction of computing time by a factor of 4. We demonstrated empirically that applying our approach to multi-camera scenarios is feasible as long as the lighting conditions and the orientations of the cameras remain similar.

In sequences containing occlusions, approaches based on Tracking-Learning-Detection outperform adaptive tracking-by-detection methods. We attribute this to the following reasons. Adaptive tracking-by-detection methods typically perform a form of self-learning, meaning that the output of a classifier is used for labeling unlabeled data. In Tracking-Learning-Detection, unlabeled data is explored by a tracking mechanism that is not dependent on the detector but bases its decision on a different measure, which in our case is the optical flow. The performance of approaches based on Tracking-Learning-Detection is further improved by the automatic detection of tracking failures and by introducing criteria for validity that have to be met when learning is performed.

Clearly, our approach heavily depends on the quality of the results delivered by the recursive tracker. Principally, the quality of the results can be improved in two ways. First, the timespan during which the tracker is following the object of interest correctly could be increased. This would present the object detector with more true positive examples. Second, The automatic detection of tracking failures could be improved, which would further prevent the object detector from drifting.

One problem that was encountered during the experiments is that the object detector is unable to discriminate against objects that exhibit a similar appearance. This problem is partially caused by the fact that the comparison of templates is performed on images of reduced size. One solution to this problem might be to increase the resolution of the template images, but this will

introduce the curse of dimensionality. As a compromise, one could employ image pyramids for the templates and perform the comparison using a coarse-to-fine strategy.

The use of bounding boxes, while convenient for implementation, also has its shortcomings. Since bounding boxes always cover a rectangular region around the object, they partially may contain background. We assign class labels on a bounding-box level which causes the appearance of the background to be considered part of the object of interest. This leads to the problem that the object of interest is not recognised when it appears on a different background. In order to separate the object of interest from the background in a bounding box for learning, one could use segmentation techniques, such as the one presented in [19].

Currently, our approach gives information about the location of the object of interest only, but not about its orientation. Information about the orientation of objects could be retrieved by employing an affine transformation model for the Lucas-Kanade tracker.

A severe unsolved problem consists of the fact that the detector is unable to recognise appearance changes that occur while the tracker is not active. In [38], image warping is applied to training examples in order to achieve invariance to affine transformations. However, affine transformations do not cover changes in local illumination or perspective. These changes occur frequently in multi-camera scenarios and are caused by different lighting conditions and camera viewpoints. A solution for this problem yet has to be found.

APPENDIX A

Appendix

A.1 Number of Subwindows in an Image

Generally, in an image of size $n \times m$, the exhaustive set $\mathcal{R}_{exh}$ of all possible subwindows is

$$\mathcal{R}_{exh} = \{\langle x, y, w, h\rangle \mid 1 \leq x < n, 1 \leq y < m, 1 \leq w \leq n-x, 1 \leq h \leq m-y\}. \tag{A.1}$$

The size of this set defined by Eq. A.1 is

$$|\mathcal{R}_{exh}| = \sum_{x=1}^{n}(n-x)\sum_{y=1}^{m}(m-y) \tag{A.2}$$

$$= \sum_{x=1}^{n}(n-x)\left(m^2 - \sum_{y=1}^{m} y\right) \tag{A.3}$$

$$= \sum_{x=1}^{n}(n-x)\left(m^2 - \frac{m(m+1)}{2}\right) \tag{A.4}$$

$$= \sum_{x=1}^{n}(n-x)\frac{m(m-1)}{2} \tag{A.5}$$

$$= \frac{n(n-1)}{2}\frac{m(m-1)}{2}. \tag{A.6}$$

For an image of size $n \times n$, Eq. A.6 amounts to

$$|\mathcal{R}_{exh}| = \frac{n(n-1)}{2}\frac{n(n-1)}{2} \tag{A.7}$$

$$= \frac{n^4 - 2n^3 + n^2}{4}. \tag{A.8}$$

If the constraints from Sec. 3.1 are used, the number of possible subwindows decreases as follows. Let w, h be the width and height of the initial window and let d_x, d_y be the pixels that

each subwindow is to be shifted. The number of subwindows in one row then is

$$n_x = \left\lfloor \frac{n - w + d_x}{d_x} \right\rfloor \tag{A.9}$$

and the number of subwindows in one column is

$$n_y = \left\lfloor \frac{m - h + d_y}{d_y} \right\rfloor. \tag{A.10}$$

The total number of sliding windows then is

$$|\mathcal{R}| = n_x n_y. \tag{A.11}$$

A.2 Alternative Formulation of Variance

The alternative formulation of the variance is obtained by the following derivation[1].

$$\begin{aligned}
\sigma^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 && \text{(A.12)}\\
&= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2x_i\mu + \mu^2 && \text{(A.13)}\\
&= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\sum_{i=1}^{n} 2x_i\mu + \frac{1}{n}\sum_{i=1}^{n}\mu^2 && \text{(A.14)}\\
&= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2\mu\frac{1}{n}\sum_{i=1}^{n} x_i + \mu^2 && \text{(A.15)}\\
&= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2\mu^2 + \mu^2 && \text{(A.16)}\\
&= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \mu^2. && \text{(A.17)}
\end{aligned}$$

A.3 Maximum Resolution for Integral Images

When pixel values in an image I of size $n \times m$ are stored in unsigned integers of a bits, the maximal value at the integral image $I'(n,m)$ is $I'_{max} = 2^a mn$. This means that $\lceil a + \log_2(mn) \rceil$ is the number of bits needed to hold I'_{max}. For instance, if the resolution of I is 640×480 and pixel information is stored using an 8-bit integer, then for the values in I' $8 + \log_2(640 \cdot 480) = 27$ bits have to be used. The maximal number of pixels for 32-bit integers therefore is 2^{32-a}, which corresponds to a resolution of 4730×3547. For a value in squared integral images I'' $\lceil 2a + \log_2(nm) \rceil$ bits are necessary.

[1] Note that in [49] the alternative formulation of the variance is incorrectly stated as $\sigma^2 = \mu^2 - \frac{1}{n}\sum_{i=1}^{n} x_i^2$.

Bibliography

[1] W. C. Abraham and A. Robins. Memory retention–the synaptic stability versus plasticity dilemma. *Trends in neurosciences*, 28(2):73–78, Feb. 2005. 2

[2] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR'06)*, volume 1, pages 798–805. IEEE, July 2006. 3

[3] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, Oct. 1997. 21

[4] S. Avidan. Support vector tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):1064–1072, 2004. 3

[5] S. Avidan. Ensemble tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):261–271, Feb. 2007. 3

[6] B. Babenko, Ming-Hsuan Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 983–990. IEEE, June 2009. 4, 43

[7] M. B. Blaschko. *Branch and Bound Strategies for Non-maximal Suppression in Object Detection*, volume 6819 of *Lecture Notes in Computer Science*, chapter 28, pages 385–398. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. 24

[8] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1st edition, Oct. 2008. 8, 9

[9] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct. 2001. 37, 39

[10] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley Publishing, 2009. 3, 22

[11] F. Chang. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220, Feb. 2004. 14, 16

[12] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. The MIT Press, Sept. 2006. 3, 28, 29

[13] S.-C. S. Cheung and C. Kamath. Robust techniques for background subtraction in urban traffic video. In *Visual Communications and Image Processing 2004 (Proceedings Volume)*, volume 5308, pages 881–892. SPIE, 2004. 14

[14] R. T. Collins, Y. Liu, and M. Leordeanu. Online selection of discriminative tracking features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1631–1643, 2005. 3

[15] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 142–149 vol.2. IEEE, 2000. 3

[16] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893. IEEE, June 2005. 12, 28

[17] J. Davis and M. Goadrich. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM. 35

[18] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. 24

[19] M. Godec, P. M. Roth, and H. Bischof. Hough-based tracking of non-rigid objects. In *IEEE International Conference on Computer Vision*, pages 81–88. IEEE, Nov. 2011. 48

[20] E. B. Goldstein. *Sensation and Perception*. Wadsworth Publishing, 8 edition, Feb. 2009. 1

[21] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised On-Line boosting for robust tracking. In D. Forsyth, P. Torr, and A. Zisserman, editors, *Proceedings of the 10th European Conference on Computer Vision*, volume 5302, pages 234–247, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 4

[22] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–63, Jan. 1987. 2

[23] S. Hare, A. Saffari, and P. H. S. Torr. Struck: Structured output tracking with kernels. In *IEEE International Conference on Computer Vision*, pages 263–270. IEEE, Nov. 2011. 4, 43, 44

[24] B. Hemery, H. Laurent, and C. Rosenberger. Comparative study of metrics for evaluation of object localisation by bounding boxes. In *International Conference on Image and Graphics*, pages 459–464. IEEE, Aug. 2007. 33

[25] O. Javed, S. Ali, and Mubarak Shah. Online detection and classification of moving objects using progressively improving detectors. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 696–701. IEEE, 2005. 3

[26] Z. Kalal, J. Matas, and K. Mikolajczyk. Online learning of robust object detectors during unstable tracking. In *Proceedings of the IEEE On-line Learning for Computer Vision Workshop*, pages 1417–1424, 2009. 4

[27] Z. Kalal, J. Matas, and K. Mikolajczyk. P-N learning: Bootstrapping binary classifiers by structural constraints. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 49–56. IEEE, June 2010. 3, 4, 5, 27, 29, 36, 43, 44

[28] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-Backward Error: Automatic Detection of Tracking Failures. In *International Conference on Pattern Recognition*, pages 23–26, 2010. 4, 7, 8, 9, 10

[29] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. 14

[30] V. Lepetit and P. Fua. Monocular model-based 3D tracking of rigid objects. *Found. Trends. Comput. Graph. Vis.*, 1(1):1–89, 2005. 2

[31] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for Real-Time keypoint recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 775–781, Los Alamitos, CA, USA, 2005. IEEE Computer Society. 4, 19

[32] J. P. Lewis. Fast normalized cross-correlation. In *Vision Interface*, pages 120–123. Canadian Image Processing and Pattern Recognition Society, 1995. 9

[33] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 674–679, 1981. 2, 4, 7

[34] E. Maggio and A. Cavallaro. *Video Tracking: Theory and Practice*. Wiley, 2011. 1

[35] L. Matthews, T. Ishikawa, and S. Baker. The template update problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):810–815, 2004. 2

[36] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1 edition, Mar. 1997. 21

[37] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, Nov. 1983. 24

[38] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *IEEE Conference on Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA, June 2007. IEEE. 3, 4, 12, 19, 21, 37, 38, 48

[39] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: a systematic survey. *IEEE Transactions on Image Processing*, 14(3):294–307, Mar. 2005. 14

[40] J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988. 22

[41] D. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1):125–141, May 2008. 2, 3, 25

[42] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. PROST: Parallel robust online simple tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 723–730. IEEE, June 2010. 4

[43] H. Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2. IEEE, June 2004. 12

[44] L. G. Shapiro and G. C. Stockman. *Computer Vision*. Prentice Hall, Jan. 2001. 1

[45] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, June 1994. 7, 9

[46] S. Stalder, H. Grabner, and L. van Gool. Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. In *IEEE International Conference on Computer Vision Workshops*, pages 1409–1416. IEEE, 2009. 4

[47] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. 1

[48] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, Apr. 1991. 8

[49] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518, Los Alamitos, CA, USA, Apr. 2001. IEEE Comput. Soc. 12, 16, 24, 28, 50

[50] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4), Dec. 2006. 1, 2

[51] Q. Yu, T. B. Dinh, and G. Medioni. Online tracking and reacquisition using co-trained generative and discriminative trackers. In *European Conference on Computer Vision*, volume 5303 of *Lecture Notes in Computer Science*, pages 678–691, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 36, 43