TRACKING LEARNING DETECTION

Zdenek Kalal

Submitted for the degree of Doctor of Philosophy

Centre for Vision, Speech and Signal Processing
Faculty of Engineering and Physical Sciences
University of Surrey

September 2010

© Zdenek Kalal 2010
Summary

Long-term tracking is the process of locating a moving object in a video sequence, where the object moves in and out of the camera view. This thesis investigates long-term tracking of unknown objects in a video stream. The object is defined by its location and extent in a single frame. In every frame that follows, the task is to determine the object’s location and extent or indicate that the object is not present. We propose a novel tracking framework (TLD) that decomposes the long-term tracking task into three sub-tasks: tracking, learning and detection, which operate simultaneously. Each sub-task is addressed by a single component. The tracker follows the object from frame to frame. The detector localizes all appearances that have been observed during tracking and corrects the tracker if necessary. Exploiting the spatio-temporal structure in the video stream, the learning component estimates errors performed by the detector and updates it to avoid these errors in the future. The components of the framework are analyzed in detail. In particular, we focus on: (i) detection of tracking failures, (ii) online learning of an object detector from a video stream, and (iii) offline learning of an object detector from a large labeled data set. A real-time implementation of the TLD is described and comparatively evaluated on benchmark sequences. A significant improvement over state-of-the-art methods is achieved.

Key words: long-term tracking, failure detection, boosting, face detection, real-time tracking, unsupervised bootstrapping

Email: zdenek.kalal@gmail.com
Acknowledgements
# Contents

1 Introduction .......................................................... 1
   1.1 Objectives .................................................. 1
   1.2 Motivation .................................................. 3
   1.3 Challenges ................................................ 4
   1.4 Contributions ............................................ 7
   1.5 Thesis outline ............................................ 8
   1.6 Publications ............................................. 10

2 Related work ......................................................... 11
   2.1 Tracking .................................................. 11
      2.1.1 Prerequisites ....................................... 12
      2.1.2 Classification ....................................... 13
      2.1.3 Generative trackers ................................ 14
      2.1.4 Discriminative trackers ............................. 17
   2.2 Detection .................................................. 20
      2.2.1 Detection of image features ....................... 20
      2.2.2 Detection of object instances ...................... 21
      2.2.3 Detection of faces ................................. 23
   2.3 Machine learning ........................................ 26
      2.3.1 Bootstrapping ....................................... 26
      2.3.2 Boosting ............................................ 27
      2.3.3 Semi-supervised learning ......................... 30
   2.4 Observations .............................................. 32
## Tracking: failure detection

3.1 Detection of tracking failures
---
3.1.1 Forward-Backward error
3.1.2 Quantitative evaluation
3.1.3 Visualization

3.2 Median-Flow tracker
---
3.2.1 Quantitative evaluation

3.3 Conclusions

## Detection: supervised bootstrap

4.1 Introduction
4.2 Related work
4.3 Integration of bootstrapping and boosting
---
4.3.1 Known sampling strategies
4.3.2 Proposed sampling strategies
4.3.3 Properties of sampling strategies

4.4 Application to face detection
---
4.4.1 Frontal face detector
4.4.2 Profile face detector
4.4.3 Specific face detector

4.5 Conclusions

## Learning: unsupervised bootstrap

5.1 Introduction
5.2 P-N learning
---
5.2.1 Formalization
5.2.2 Stability
5.2.3 Experiments

5.3 Learning an object detector from a video sequence
5.3.1 Problem specification ........................................... 65
5.3.2 P-N experts ......................................................... 66
5.3.3 Experiments ......................................................... 69
5.4 Conclusions ......................................................... 70

6 Tracking-Learning-Detection (TLD) .................................. 71

6.1 Introduction ......................................................... 71
6.2 The TLD framework .................................................. 72
  6.2.1 Components ...................................................... 72
  6.2.2 Initialization ...................................................... 73
  6.2.3 Run-time ......................................................... 74
6.3 Implementation of TLD ............................................... 74
  6.3.1 Object representation ........................................... 74
  6.3.2 The object model ................................................ 75
  6.3.3 The object detector .............................................. 76
  6.3.4 The object tracker .............................................. 79
  6.3.5 The integrator .................................................... 80
  6.3.6 The learning component ....................................... 81
6.4 Quantitative evaluation ............................................... 83
  6.4.1 Comparison 1: CoGD .......................................... 83
  6.4.2 Comparison 2: PROST .......................................... 84
  6.4.3 Comparison 3: TLD data set .................................... 85
6.5 Long-term tracking of faces ......................................... 87
  6.5.1 Sitcom episode ................................................. 87
  6.5.2 Surveillance footage ............................................ 88
6.6 Qualitative analysis .................................................. 89
  6.6.1 Strengths ....................................................... 90
  6.6.2 Weaknesses .................................................... 92
7 Discussion

7.1 Contributions ......................................................... 95
7.2 Recent development ................................................... 97
7.3 Future work ............................................................. 98

A Compared algorithms ................................................. 101

B Sequences used for evaluation ....................................... 103

Bibliography ............................................................... 105
Chapter 1

Introduction

Computer vision aims at interpretation of image data. One of the basic tasks is the estimation of the state (e.g. location, extent) of an object in a video sequence. This task has been studied for several decades and it still remains challenging. To make the problem tractable, a common approach is to make assumptions about the object, its motion or motion of the camera. In contrast, this thesis studies the task without making any of these assumptions.

This chapter provides an overview of the entire thesis. Section 1.1 formalizes our objectives. Section 1.2 introduces possible applications of our research. Section 1.3 discusses the main challenges that have to be tackled. Section 1.4 introduces the contributions made in the thesis. Section 1.5 outlines the rest of the thesis and section 1.6 lists the publications.

1.1 Objectives

Consider a video stream depicting various objects moving in and out of the camera field of view. Given a bounding box defining the object of interest in a single frame, our goal is to automatically determine the object’s bounding box or indicate that the object is not visible in every frame that follows. The video stream is to be processed at full frame-rate and the process should run indefinitely long. We refer to this task as long-term tracking.

A number of algorithms related to long-term tracking have been proposed in the past. However, these typically make strong assumptions about the task. In particular, tracking-based algorithms assume that the object moves on a smooth trajectory and typically fail if the object moves out of the image. Detection-based algorithms assume that an object is known in advance and require a training stage. In contrast, our goal is to track an arbitrary object that moves in and out of the camera view immediately after initialization. The problem and the achieved results are shown in figure 1.1.
Figure 1.1: The long-term tracking task and the achieved results. The top row depicts the objects of interest selected for tracking. The remaining images show the results of our long-term tracker. The red dots indicate that the object is not visible.
1.2 Motivation

The research in this thesis is mainly motivated by real-time, interactive applications.

- **Surveillance.** Consider a surveillance camera and an operator. At a certain moment, the operator marks the object of interest. A long-term tracker should be able to monitor the motion of the object while being visible, indicate that the object has moved out of the field of view and re-initialize the monitoring once the objects reappears (possibly in a another camera). This task is essential for security purposes, analysis of customer behavior or even for navigation of robots. Existing systems do not scale well for unexpected objects. An algorithm able to monitor the motion of an arbitrary object for long periods of time, would find a number of practical applications in surveillance and robot navigation.

- **Augmented reality.** Existing applications in augmented reality are often restricted to planar and textured objects. At the core of these applications is robust tracking which often relies on an offline training stage. The ability to robustly track an arbitrary object without prior training is therefore of great interest. The range of applications in augmented reality is large: games, advertisement, medical, education, tourism or military.

- **Object-centric camera stabilization.** Consider a hand-held camera and a user that select an arbitrary object. Using object tracking, one can imagine an object-centric video stabilization or adjustment of the camera setting. Existing tracking algorithms would be able to stabilize the video as long as the object is in the field of view, in contrast, a long-term tracker would be able to restart the stabilization whenever the object reappears in the field of view. A typical usage would be when observing a distant object using digital zoom.

- **Object recognition.** Contemporary smart phones feature visual recognition of objects captured by the camera. An emerging approach is to perform client-side tracking of the object of interest to acquire a sufficient number of query images and send them to a server which performs object recognition. Development of long-term tracking methods is therefore of high interest as it can improve robustness of the whole recognition system.

- **Video analysis.** A number of applications in video analysis (e.g. action recognition, automatic video annotation) require tracking of object or their parts in long video sequences. These objects are often not visible throughout the entire sequence and therefore long-term tracking methods can be potentially applied in these problems.
1.3 Challenges

The range of possible applications of long-term tracking systems is vast. However, there are a number of issues which need to be addressed by a successful long-term tracker.

- **Occlusion and disappearance of the object.** The object gets occluded or disappears from the camera view for arbitrarily long time. The object may reappear at any time and at any location. Therefore, the long-term tracker should have a detection mechanism to resolve these cases. Figure 1.2 illustrates the scenario.

- **Appearance and viewpoint changes.** The object of interest may significantly change its appearance and viewpoint throughout the sequence. This complicates the tracking process as the only information given to the long-term tracker is a single appearance (defined by a bounding box), which may not be relevant throughout the entire sequence. The long-term tracker should have a mechanism (e.g. adaptation, learning) for dealing with the appearance changes. See figure 1.3 for illustration.

- **Background clutter and identification.** The object may appear in cluttered environments or may be surrounded by other objects of the same visual class. The long-term tracker should not get distrusted by background clutter and correctly identify the object once it reappears. Figure 1.4 illustrates the scenario where the task is to track a human face.

- **Scale changes.** The object may change scale. While estimation of the scale can be considered as an implementation detail, it brings additional degree of freedom to the tracking process and as such increases vulnerability of the tracker to fail. The long-term tracker should be able to estimate the scale of an object as illustrated in figure 1.5.

- **Illumination changes.** The object changes its appearance under different illumination. The long-term tracker should be able to deal with illumination changes as illustrated in figure 1.6.

- **Image noise.** Video sequences may be corrupted by motion blur, interlacing or compression. This influences the accuracy of features that are extracted from every frame, which may corrupt the output of the tracking algorithm. The long-term tracker should deal with image noise. Figure 1.7 illustrates the object corrupted by motion blur.

- **Real-time performance.** To be useful in interactive applications, the long-term tracker should work at full frame rate. Therefore, the algorithm must be extremely efficient and causal.
1.3. Challenges

Figure 1.2: Challenges in long-term tracking: occlusion and disappearance.

Figure 1.3: Challenges in long-term tracking: appearance and viewpoint changes.

Figure 1.4: Challenges in long-term tracking: background clutter and identification.
Chapter 1. Introduction

Figure 1.5: Challenges in long-term tracking: scale changes.

Figure 1.6: Challenges in long-term tracking: illumination changes.

Figure 1.7: Challenges in long-term tracking: image noise.
1.4 Contributions

This thesis contributes to the research in long-term tracking with the algorithms summarized below.

- **TLD framework.** We design a novel framework that decomposes the long-term tracking task into three sub-tasks: tracking, learning and detection, each of which is tackled by a dedicated component. The tracker follows the object from frame to frame. The detector localizes all appearances that have been observed during tracking and corrects the tracker if necessary. Exploiting the spatio-temporal structure in the data, the learning component estimates errors performed by the detector and updates it to avoid these errors in the future. The uniqueness of the framework is in the close integration of all these components which enables mutual compensation of their individual flaws. The block diagram of the TLD framework is shown in figure 1.9. The components of the framework are studied in detail.

- **Forward-Backward error.** Building on the forward-backward consistency assumption, we develop a novel measure that estimates the reliability of tracking of an arbitrary object in a video stream.

- **Median-Flow tracker.** We develop an adaptive tracker that is robust to partial occlusions and deals with appearance and illumination changes. An object of interest is represented by a bounding box, within which a sparse motion field is estimated. Reliability of each motion vector is accessed and the most reliable vectors determine the motion (shift, scale-change) of the object.
• **P-N learning.** We develop an unsupervised learning method for online learning of object detectors from video streams. The learning method is able to learn an object detector from a single example (defined by a bounding box) and a video stream where the object may appear. The learning method is formulated as unsupervised bootstrapping, where the detector errors are estimated by a pair of “experts”: (i) P-expert estimates missed detections, and (ii) N-expert estimates false alarms. Both of these experts are allowed to make errors themselves. The learning process is modeled as a discrete dynamical system and the conditions under which the learning guarantees improvement of the detector are found using stability criteria developed in control theory [Zhou 96, Ogata 09].

• **Boosting+Bootstrap.** Motivated by long-term tracking of human faces, we develop a novel learning method for training efficient object detectors. The learning method is based on a fusion of two popular learning approaches: boosting [Freund 97] and bootstrapping [Sung 98]. We formulate bootstrapping as weighted sampling where the weights are driven by boosting. The optimal sampling strategy is analytically deduced. Extensive experimental evaluation shows superior performance with respect to commonly used methods. The learning method is tested on the task of face detection and state-of-the-art performance is achieved.

• **Implementation.** We show how to build a long-term tracking system based on the TLD framework and the components developed in the thesis. The system tracks, learns and detects an unknown object in an unknown environment in real-time. An extensive quantitative evaluation shows a significant improvement over state-of-the-art systems. Furthermore, we show how to adapt the TLD framework for face tracking, which combines an offline trained face detector with online face identification.

1.5 **Thesis outline**

The structure of the thesis is outlined in figure 1.9. In chapter 2 we review the work related to long-term tracking from three points of view: object tracking, object detection and machine learning. In object tracking and object detection, special attention is given to methods that represent the object by a bounding box. In object learning, we focus on methods for training of sliding window-based detectors.

Chapters 3, 4 and 5 consider the tracking, learning and detection problems independently and propose the basic components that help us build the complete TLD system.

Chapter 3 considers tracking algorithms when applied to unconstrained videos. Accepting the fact that every tracker eventually fails is the starting point in the design
of the TLD system. First, we study how to detect tracking failures. We define the Forward-Backward error, study its properties and compare to relevant error measures. Second, we study how the detected tracking failures can improve tracking itself. We develop the Median-Flow tracker, compare it to state-of-the-art methods and demonstrate its superior performance on a number of sequences.

Chapter 4 considers offline training of an object detector from large data sets. We review the algorithms often used for this setting and identify two approaches (boosting [Freund 97] and bootstrapping [Sung 98]) that are often employed. The optimal combination of boosting and bootstrapping is developed, and evaluated on both synthetic and real data. Superior performance in face detection is demonstrated.

Chapter 5 focuses on online learning of an object detector from a single example and a video stream. The P-N learning theory is developed and formalized as a semi-supervised [Chapelle 06] learning method. We show how to design a pair of experts that estimate errors of the detector during learning. We show that tracking can estimate false negatives (P-expert) and non-maxima-suppression can estimate false positives (N-expert) made by the detector. The resulting algorithm is quantitatively evaluated on a number of sequences leading to surprising gains in the detector performance.

Chapter 6 develops the TLD framework and describes the implementation of our long-term tracking system. The system is evaluated on two types of experiments: (i) tracking of unknown objects, and (ii) tracking of human faces. A quantitative evaluation is performed on 21 video sequences and compared to 13 relevant tracking algorithms. A significant improvement over state-of-the-art methods is demonstrated. Furthermore, perform a qualitative valuation and comment on pros and cons of the system.

Chapter 7 summarizes the contribution of the thesis, recent development and discusses possible directions for future research.
1.6 Publications

The work from this thesis has been presented in the following publications:


Chapter 2

Related work

The long-term tracking is a complex problem that is closely related to tracking, detection and machine learning. These terms are understood as follows. Tracking estimates the object motion between consecutive frames relying on temporal coherence in the video. Detection considers the video frames as independent and localizes all objects that correspond to an object model. Machine learning is often employed in both of these approaches. Trackers use machine learning to adapt to changes of the object appearance. Detectors use machine learning to build better models that cover various appearances of the object.

To give an overview of the most relevant approaches, the chapter is split into four parts. Section 2.1 gives an overview of the tracking approaches ranging from simple template tracking up to trackers that learn online a discriminative classifier. Section 2.2 reviews detection approaches focusing on detection of object instances as well as detection of human faces. Section 2.3 reviews the learning strategies commonly used for training of object detectors. In particular we review bootstrapping, boosting as well as methods based on semi-supervised learning. The section 2.4 comments on observed trends and outlines challenges that motivated our research.

2.1 Tracking

Tracking is a task of estimating object motion between consecutive frames. It is one of the basic tasks in computer vision [Yilmaz 06]. The implicit assumption of all tracking algorithms is that the location of the object in previous frame is known. This is in contrast to long-term tracking where this location might not be defined. In the following, the term tracking will be sometimes substituted with more accurate frame-to-frame tracking to emphasize the meaning. This review is organized as follows. Sub-section 2.1.1 introduces the terms used in tracking. Sub-section 2.1.2 classifies
Chapter 2. Related work

2.1.1 Prerequisites

At every time instance, trackers characterize the object of interest by several parameters (e.g. location, scale, pose), which together represent the so-called state of the object. A temporal sequence of states defines the object trajectory. Difference of two consecutive states defines the object motion.

Model. Tracking algorithms represent the object by a model. We distinguish two classes of models based on the type of information they represent: (i) generative, and (ii) discriminative. Generative models represent the appearance of the object ignoring the environment where the object moves. Discriminative models focus on differences between the object and the environment. Both of these models are either static – remain fixed during tracking, or adaptive – accept new information during tracking.

Motion estimation. Given the object state in previous frame and the object model, tracking estimates the object motion by fitting the object model to the current frame using some estimation algorithm (e.g. mean-shift, gradient descent). Only the vicinity of the previous state is typically explored. Figure 2.1 illustrates the scenario.

Drift & failure. State-of-the-art trackers are often adaptive, i.e. update the object model during tracking, which allows them to handle changes in object appearance, illumination or environment. The drawback of the adaptation is drift: the errors of the update accumulate over time and the tracker slowly slips away from the object. Drift is different from tracking failure, which is a sudden incorrect estimation of the object state. Tracking failures typically happen when the object dramatically changes appearance, gets fully occluded or moves out from the camera view.

Figure 2.1: Illustration of a typical tracking system.
2.1. Tracking

2.1.2 Classification

One of the most distinctive properties of a tracking algorithm is the object state, which determines the parameters that are estimated during tracking. Here we use the object state to classify tracking algorithms into five categories shown in figure 2.2.

1. **Points** are often used to represent the smallest objects that do not change their scale dramatically. Algorithms that represent the object by a point will be called point trackers. Point trackers estimate only translation of the object. The estimation can be performed using frame-to-frame tracking [Lucas 81, Shi 94], key-point matching [Veenman 01], key-point classification [Lepetit 05], or linear prediction [Zimmermann 09]. Recent work is directed towards optimizing performance of these methods [Takacs 10].

2. **Geometric shapes** such as bounding boxes or ellipses, are often used to represent motion of objects which undergo significant change in scale. These methods typically estimate object location, scale and in-plane rotation, all other parameters are typically modeled as the change of the object appearance.

3. **3D models** are used to represent rigid objects, for which the 3D geometry is known. These models estimate location, scale and pose of the object. These methods have been applied to various objects including human faces [Vacchetti 04]. A significant effort in tracking is directed towards systems that build the 3D models online such as SLAM [Davison 03] or PTAM [Klein 07].

4. **Contours** are used to represent non-rigid objects. Parametric representation of contours has been used for tracking of human heads [Birchfield 98] or arbitrarily complex shapes [Isard 98]. Non-parametric representation has been applied for tracking of people in sport footage [Yilmaz 04], or various flexible objects including animals and human hands [Bibby 08, Bibby 10].

5. **Articulated models** are used to represent motion of non-rigid objects consisting of several rigid parts. These models typically consist of several geometric
shapes, which relative motion is restricted by a model of their geometric relations. Articulated models have been used for tracking of humans [Wang 03, Ramanan 07] or human arms [Buehler 08].

6. **Motion field** [Horn 81, Brox 04] is a non-parametric representation of the object motion which gives the displacement of every pixel of the object between two frames. An extensive comparison of these methods can be found in [Barron 94]. Recent development aims at producing long, continuous trajectories of image points [Sand 08, Goldman 07].

In this thesis we represent the object state by a bounding box. This representation balances the tradeoff between the expressive power of the representation and the difficulty to reliably estimate the object motion. The related methods will be now analyzed in detail.

### 2.1.3 Generative trackers

Generative trackers model the appearance of the object. In this section we focus on trackers that represent the object by a geometric shape (i.e. rectangle or ellipse) within which the appearance is modeled. Motion estimation will be typically formulated as a search for the best match between an image patch and the model. Even though these methods were not designed for long-term tracking, we review them as our system is relying on number of ideas from this category.

**Template tracking**

Template trackers represent the object appearance by a single exemplar – a *template* (e.g. image patch). They define a *similarity measure* between two templates and search for such a displacement (or warp) that maximizes the similarity match.

Exhaustive search for the best similarity match is straightforward, but not efficient. To make it faster, the search can be performed using integral images [Schweitzer 02], or in frequency domain [Reddy 02]. Another strategy is to restrict the search to the vicinity of the previous location. Exploration of the neighborhood not only increases the efficiency of the template tracker, but also reduces the number of false matches. On the other hand the tracker may lose the target if the frame-to-frame motion is larger than expected. The most popular strategies for searching within the surrounding of the previous location are: (i) gradient-based methods, and (ii) mean-shift.

Gradient-based methods optimize the similarity measure using gradient descent. One of the most popular methods is the Lucas-Kanade tracker [Lucas 81] and its pyramidal implementation [Bouguet 99], which estimates translation of an image patch.
2.1. Tracking

Affine warping was later proposed in the Kanade-Lucas-Tomasi tracker [Shi 94]. Both of these approaches have been unified in more efficient Inverse Compositional Algorithm [Baker 04]. These methods base the similarity function on SSD. Recently, the similarity function based on mutual information has been proposed [Dowson 08] and demonstrated larger convergence basins.

The mean-shift [Comaniciu 03, Bradski 98] is another approach to avoid the brute-force search for the best template match. The object is described by a distribution of colors. Given a new image, every pixel is assigned a probability score from the distribution resulting in the so-called back-projection image. Mean-shift then iteratively searches for a mode in the back-projection image. Similarly as the gradient-based methods, the mean-shift cannot handle large displacements as the search is local. On the other hand, the histogram-based representation handles large appearance changes as long as the color distribution remains similar.

Improvements of the template tracking

The template tracking has three main drawbacks. First, the template tracking faces the tradeoff between static and adaptive tracking. A single static template is often not sufficient to represent all appearances of the object and adaptation of the template (using the template from the previous frame) suffers from drift. Second, the template tracking is often sensitive to partial occlusions. Third, a single template does not allow encoding of multiple appearances.

To tackle the tradeoff between static and adaptive tracking, the basic idea is to adapt the template only if necessary and re-use the previously observed templates otherwise. For this purpose a function that evaluates the usefulness of the previously observed templates has to be designed. This idea has been used for tracking of image patches [Matthews 04, Dowson 05] as well as 3D pose estimation [Rahimi 08].

To tackle the problem of partial occlusions, the WSL tracker [Jepson 03] decomposes
Chapter 2. Related work

Figure 2.4: Incremental Visual Tracking [Ross 07] builds online a PCA-based model of the object. The method demonstrated a strong resistance to appearance and illumination variations.

the template into three layers: "Wandering", "Stable" and "Lost". It was shown that by focusing on stable parts of the template, the tracker deals better with partial occlusions and appearance changes. For illustration see figure 2.3. Fragment-based tracker [Adam 06] decomposes the template into a set of randomly chosen fragments. Motion of each fragment is estimated independently and the global motion is estimated using a robust statistics.

To encode multiple appearances of the object, the single image patch has been replaced by multiple projections given by Principal Component Analysis (PCA). EigenTracking [Black 98] considers a scenario when all the object appearances are known in advance and trains an object model offline. This approach has been applied to tracking of non-rigid objects on non-cluttered background. Incremental Visual Tracking [Ross 07] builds the PCA-based model during tracking. It has been demonstrated to handle illumination and appearance variations. See figure 2.4 for illustration.

The research in generative trackers demonstrated that drift can be reduced by reusing already seen examples of the object. And that the resistance to partial occlusions can be achieved by decomposing the template into independent parts. Both of these ideas are used in our system. However, the principled drawback of generative approaches is that only the object appearance is modeled. As a consequence, the generative trackers get easily confused in cluttered background, where the clutter may look similar as the object. To increase the tracker robustness, the background class has to be considered in the modeling.
2.1.4 Discriminative trackers

Discriminative trackers encode the differences between the object appearance and the environment where the object moves. A common approach is to build a binary classifier that distinguishes the object from its background. These methods represent the closest competitors to our system as they often demonstrate re-detection capabilities.

Static discriminative trackers

One of the earliest discriminative trackers was proposed by Avidan [Avidan 04], who integrated an offline trained SVM classifier into the tracking process. The motion was estimated by maximizing the classifier confidence in a gradient-ascent manner. The performance was demonstrated on the task of vehicle tracking. The main limitation of static discriminative trackers is in the training data since all appearances of the object and background have to be captured in advance. To alleviate this problem, the adaptive discriminative trackers have been proposed.

Adaptive discriminative trackers

The adaptive discriminative trackers learn the discriminative classifier during tracking. This allows them to track a wide range of objects immediately after initialization. These trackers typically operate as follows. In the first frame, the tracker builds a simple classifier separating the selected object from its background. The tracking then proceeds in a frame-by-frame fashion. In every frame, the classifier is evaluated on the surrounding of the previous object location and the new location is established, e.g. by taking the maximally confident location. The tracker then performs an update. The current location of the object is used to sample positive examples, and the surrounding is used to sample negative examples. These labeled examples update the classifier, which is then used in the next frame.

One of the earliest works from this category was done by Collins et al. [Collins 05] who built a generative model by discriminative training. The object model, based on color projections, was adapted during tracking which made it robust to significant illumination variations. Ensemble Tracking [Avidan 07] updates a boosting-based classifier to discriminate between pixels on the object and pixels on the background. Online Boosting [Grabner 06] applies the same principle to a grid of overlapping bounding boxes instead of individual pixels. Their classifier was based on a set of Haar-like features [Viola 01] and was trained in an online boosting [Oza 05] framework.

The research in adaptive discriminative tracking has enabled tracking of objects that significantly change appearance and move in cluttered background. The speed of adaptation of the classifier plays an important role in these systems. It controls the impact
Figure 2.5: Ensemble tracking [Avidan 07] represents the object of interest by a dis-
criminative model classifying every pixel as the object or background. The classifier
is adapted in every frame. The tracker demonstrated the ability to handle appearance
changes in presence of cluttered environment.

Figure 2.6: The SemiBoost tracker [Grabner 08] reduces drift of discriminative track-
ers by guiding the update by an offline trained classifier. Performance was demon-
strated (among others) on: (TOP) 24h tracking of a still object by static camera, (BOT-
TOM) re-detection of a stationary object after a brief occlusion.
of new appearances on the classifier, but also the speed by which the old information is forgotten. If the speed of adaptation is set correctly for a given problem, these trackers demonstrate robustness to short-term occlusion. On the other hand, if the object is not visible for longer time than expected, the tracker will eventually forget the relevant information and never recover.

**Constrained adaptation for discriminative trackers**

This section reviews the discriminative tracking approaches that perform *constrained* adaptation of the classifier, which is in contrast to every-frame-update reviewed in the previous sub-section. The constrained adaptation aims at drift reduction as well as long-term tracking resistant to long-lasting full occlusions. Three classes of approaches can be identified: (i) semi-supervised learning, (ii) multiple-instance learning, and (iii) co-training.

Semi-Supervised Learning (SSL) [Chapelle 06] is a learning approach that learns from both labeled and unlabeled training data. SSL has been applied to a number of problems in machine learning [Blum 98] and recently also to adaptive discriminative tracking. Semi-supervised Online Boosting [Grabner 08] is one of the earliest approaches. The initializing frame is considered as a collection of labeled examples, all the remaining frames from the sequence are considered as unlabeled. The method employs two classifiers: (i) a classifier used for tracking, and (ii) an auxiliary classifier used for update. In the first frame, both classifiers were trained using the labeled data. During the tracking, the auxiliary classifier remained fixed and provided soft-labels to the unlabeled patches, which were then used to update the tracking classifier. The method demonstrated reduction of drift and certain re-detection capabilities. See figure 2.6 for illustration. Beyond Semi-supervised Online Boosting [Stalder 09] extended the approach with one more auxiliary classifier to increase the adaptability of the system.

Multiple Instance Learning (MIL) [Dietterich 97] is a variant of supervised learning, where examples are delivered in groups. Within each group, the examples share the same label. MILTrack [Babenko 09] combines Online Boosting [Grabner 06] and MIL. In contrast to the naive update, the classifier is updated by spatially related groups of patches. The introduction of this spatial-information reduced drift and improved accuracy. Later on, a combination of MILTrack and SSL was proposed [Zeisl 10].

Co-training [Blum 98] is a specific instance of SSL methods, which trains in parallel two independent classifiers. Confident predictions of one classifier train the second classifier and vice versa. The idea behind the co-trained trackers is the following. If the object disappears from the view, neither of the classifiers is confident and the update does not take place. Co-trained Support Vector Machine tracker [Tang 07] represents the object in two feature spaces (color, gradient orientations [Dalal 05]), which were used to train two online-SVM [Cauwenberghs 01] classifiers. Co-trained Generative
2.2 Detection

Object detection is a task of localizing objects in an input image. In long-term tracking, the detection capability is essential as the object freely moves in and out of the camera field of view. Object detectors do not make any assumptions about the number of objects nor their location in the image. The objects are described by a model that is built in a training phase. In run-time, the model remains typically fixed. Figure 2.7 illustrates a typical detection system.

This section reviews the detection approaches starting from the simplest up to the most complex. Sub-section 2.2.1 reviews detectors of image features, which represent the basic building blocks of more complex approaches. Sub-section 2.2.2 reviews detectors of object instances (e.g. a specific book cover) and briefly comments on recent methods for online learning of these methods. Finally, as we are also interested in long-term tracking of human faces, sub-section 2.2.3 reviews the methods for detection of human faces and comments on commonly used features.

2.2.1 Detection of image features

Feature detectors [Tuytelaars 07] are algorithms that are used to localize salient points (or regions) in an input image. We distinguish two classes of approaches for feature detection.

**Designed detectors.** These detectors design a saliency measure and localize features that maximize it. The widely known approaches are Harris [Harris 88] or Shi-Tomasi
2.2. Detection

Detector [Shi 94], which localize image points. These detectors have been extended to scale and affine covariant region detectors with Hessian-Laplace [Mikolajczyk 05] detector. Other examples from this category are Difference of Gaussians (DoG) [Lowe 04] or Maximally Stable Extremal Regions (MSER) [Matas 04].

**Learned detectors.** Feature detection has reached a level of maturity and the research was directed toward proposing more efficient solutions that stem from machine learning community. A popular algorithm is FAST [Rosten 06] keypoint detector, which approximates the Harris corner detector [Harris 88]. Efficient approximations of Hessian-Laplace [Mikolajczyk 05] were also proposed [Sochman 09].

### 2.2.2 Detection of object instances

This section focuses on methods that detect object instances, such as a specific book cover. We distinguish approaches with model the object appearance (i) globally and (ii) locally. We also mention methods for learning of these detector.

**Global appearance models** typically represent the object appearance by a collection of examples and define the detection (and the pose estimation) as the search for the most similar example in the database. In one of the earliest works, Murase and Nayar [Murase 95] use a controlled capturing process to collect a large number of examples of various 3D objects. Similar idea underpins more recent methods for detection and rectification of patches [Hinterstoisser 09] or detection of texture-less objects using dominant orientation templates [Hinterstoisser 10]. Global appearance models are appealing due to their simplicity, however they suffer from partial occlusions and background clutter.

**Local appearance methods** represent the object by a collection of local patches that are related by geometric constraints. Their advantage, with respect to the global appearance representation, is the resistance to partial occlusions. The seminal work in this area was done by Lowe [Lowe 04], where the object is modeled by a collection of SIFT descriptors [Lowe 04] extracted around DoG features. The detection has two stages: (i) the detected DoG features are assigned a nearest neighbor descriptor stored in a database, and (ii) the assignments are validated using geometric constraints (similarity transformation is considered). Lowe demonstrated near real-time detection of multiple objects, resistant to significant occlusions as shown in figure 2.8. A number of approaches followed this research line [Obdrzalek 05, Lepetit 05, Taylor 09, Pilet 10].

**Learning approaches.** The methods discussed above typically separate the training and testing stage, which restricts their application to scenarios when the object appearance is known in advance. The training state is therefore essential and often requires a large number of human-annotated training examples. In order to simplify the training stage, Feature Harvesting [Ozuysal 06] learns the geometry and the appearance of
Figure 2.8: Lowe [Lowe 04] proposed a detection system that represented objects locally using keypoints and SIFT descriptors. The detector is based on an approximate nearest neighbor and global geometric constraints. The system demonstrated a significant illumination and pose invariance and robustness to partial occlusions.

Figure 2.9: Feature harvesting [Ozuysal 06], a method for automatic training of an object detector in a controlled environment (TOP). After the training phase the detector operates in cluttered environments (BOTTOM).

For certain classes of objects, the detection of object instances reached a level of maturity. In particular, planar and textured objects can be reliably detected in real-time and are often used in augmented reality applications. Objects that are non-rigid and
non-textured still remain challenging for detection. With respect to the online learning, several methods are designed to enable instantaneous online learning, but the decision when to learn new appearances is not directly addressed.

### 2.2.3 Detection of faces

Face detection is a task of localizing of human faces in an input image, regardless of their pose, illumination, expression or identity. This section briefly mentions the history of face detection, describes the most popular approaches and comment on the commonly used features.

**History.** Research in face detection started in early 70’ with approaches that modeled the face appearance by a set of rules provided by the researcher [Fischler 73, Sinha 94]. This view was mainly motivated by the ease to come up with new rules (e.g. a face has two eyes) and the lack of computational resources that would enable to learn these rules explicitly. With increasing computational capacity, learning-based methods started to dominate in 90’, which were learning the rules from training examples [Turk 91, Belhumeur 97, Osuna 97, Sung 98, Rowley 98, Papageorgiou 98]. While a number of these approaches achieved high detection rates [Schneiderman 04], their practical applicability was limited due to their speed. This has been changed by Viola and Jones [Viola 01] approach, who introduced the first real-time face detector. Since then, the research in face detection iterates over the Viola and Jones approach [Li 04, Fleuret 01, Jones 03, Sochman 05, Huang 07] and is often considered as solved.

**Viola and Jones face detector** combined a number of techniques developed in face detection earlier. Many of these techniques reach beyond face detection and are used in our long-term tracking system as well. Refer to the figure 2.10.

- **Training set.** The face is represented by a model, which is learned from a large collection of labeled training examples. The positive examples depict tightly cropped faces, negative examples depict non-faces.
Chapter 2. Related work

Figure 2.11: Features used to represent appearance in objects detection.

- **Local features.** The training examples are described by local features. Popular features are Haar wavelets [Papageorgiou 98] which encode intensity patterns. The features are efficiently measured using integral images [Viola 01].

- **Cascaded classifiers.** The face model has a form of a binary classifier which is split into a number of stages. Every stage enabled early rejection of background examples. This cascaded architecture, which leads to a significant increase of the classification speed, has been first used in [Yang 94].

- **Sliding window.** The cascaded classifier is evaluated on a grid of locations in multiple scales. At every location, the classifier decides about presence or absence of the face.

- **Non-maximum suppression.** Due to the sliding window approach, the classifier typically produces multiple overlapping responses around the face. A common approach is to take the locally maximally confident response and suppress all the remaining responses.

- **Boosting+Bootstrap.** The classifier is typically learned using a combination of boosting [Freund 97] and bootstrapping [Sung 98]. As one of the contribution of this thesis is the optimal combination of these methods. A detailed review of these methods is done in section 2.3.

Features used in object detection. Features play an important role in object detection as they encode our knowledge about the object. Here we give a brief overview of features that are commonly used for description of the object appearance. See figure 2.11 for illustration.

- **Raw pixels.** An image patch can be considered a simple feature. Its advantage is its simplicity and efficiency. The drawback is its high dimensionality as well as low robustness to appearance variations. The similarity is usually measured using Normalized Cross-Correlation (NCC) or by Sum of Square Differences (SSD).
• **Histograms.** A histogram represents the object appearance by a distribution of colors, gray-scale values, edge orientations, etc. Spatial relations between pixels are discarded in histograms.

• **Filter responses.** Filters are used to detect predefined intensity patterns in an image patch. One of the most popular types are Haar-like filters [Papageorgiou 98], which encode difference of average intensities between neighboring rectangular regions. These features can be measured in constant time using integral images [Viola 01]. A number of extensions have been proposed such as comparison of non-neighboring regions [Li 04], features rotated by 45 degrees [Lienhart 02], or pixel differences measured at different scale-space levels [Huang 07].

• **Gradients.** Gradients represent a significant cue in many object recognition systems. Scale-invariant Feature Transform [Lowe 04] is probably the most successful. A patch surrounding a keypoint is split into 4x4 cells, within each cell an eight-bin histogram of gradient orientation is measured which produces a feature vector of 128 elements. A number of modifications have been later proposed. Histograms of Oriented Gradients [Dalal 05] uses an arbitrary number of cells and various normalization schemes. Edge Orientation Histograms [Levi 04] compares just two orientations.

• **Codes.** Codes convert a real-valued intensity patterns to discrete codes. Local Binary Patterns [Ojala 02] are probably the most commonly used. These features were originally designed for texture classification but later were applied to face detection [Hadid 04]. Codes were also used for encoding combinations of wavelet coefficients [Schneiderman 04] or patterns of edge orientations [Mikolajczyk 04].

![Classification of learning methods](image)

Figure 2.12: Classification of learning methods. Colored dots correspond to labeled training examples; white dots correspond to unlabeled examples.
Chapter 2. Related work

2.3 Machine learning

This section reviews strategies for learning of sliding window-based object detectors. At the core of these detectors is a binary classifier, which classifies patches in an input image. During training, the image patches are interpreted as points in the feature space (training examples), and the goal is to find a decision boundary that separates the positive examples from the negative examples. Figure 2.12 illustrates two settings discussed in this section: (i) supervised learning, (ii) semi-supervised learning.

Detectors are traditionally trained using supervised learning. While this setting is not directly relevant for long-term tracking of unknown objects, it becomes valuable when the class of the object is known in advance. For instance, if it is known that the object of interest will be a face, it is possible to train a face detector in advance. Two popular learning methods, namely bootstrapping and boosting will be reviewed in section 2.3.1 and 2.3.2 respectively. Recently, the research in the detector training has focused on semi-supervised learning methods, which, in addition to labeled data, enable processing of unlabeled data. Section 2.3.3 reviews the corresponding methods.

2.3.1 Bootstrapping

Bootstrapping [Sung 98] is a general strategy that iteratively improves a classifier by training it on increasingly larger and more informative subset of the entire training set. This subset is called an active set and the entire training set is referred to as the pool. Bootstrapping first randomly samples the active set from the pool and then iterates over: (i) training a classifier on the active set, (ii) testing of the classifier on the pool, and (iii) enlarging the active set with misclassified examples from the pool. This procedure stops when no more misclassified examples exist or when the performance
2.3. Machine learning

Figure 2.14: Boosting: a) the block diagram, b) a weak classifier and a weighted training set, c) a strong classifier.

stabilizes. The result is an augmented active set which is not sampled randomly but with emphasis on the decision boundary. See figure 2.13 for illustration.

Relation to active learning. Bootstrapping is closely related to other methods such as Active Learning [Lewis 94]. The Active Learning addresses a problem when a relatively small number of examples are labeled and a large number of examples are unlabeled. Learning starts by training a classifier from labeled examples. The classifier then evaluates unlabeled set and those examples that are not confidently classified are labeled by a human annotator. In contrast, bootstrapping assumes that all training data are already annotated. One of the contributions of this thesis is extension of bootstrapping for unlabeled data, which we call unsupervised bootstrap. The unsupervised bootstrap is discussed in chapter 5.

Bootstrapping in statistics. The definition of bootstrapping as described above is in line with the one considered in object detection [Sung 98, Papageorgiou 98, Rowley 98, Schneiderman 04, Viola 01], where it is typically used to handle large negative training sets. Notice that in statistics, bootstrapping refers to a set of re-sampling techniques used to measure statistical properties of estimators [Efron 93].

2.3.2 Boosting

Boosting [Freund 97] is a supervised learning strategy which aim is to improve the performance of a weak classifier. The intuition behind boosting is: focus on the most difficult training examples. This is achieved by maintaining a distribution of example weights that represent the example’s difficulty. This section briefly comments on the history of boosting, its relations to ensemble methods and links to bootstrapping. See figure 2.14 for illustration.
Consider a labeled training set and a set of arbitrary weak classifiers. Boosting combines these weak classifiers into a strong classifier which performs better than any of the weak classifiers. Boosting is an iterative process which iterates over: (i) selection of the weak classifier that performs best on current distribution of weights, and (ii) update of the distribution to emphasize misclassified examples. The set of weak classifiers selected in each round form the strong classifier. Boosting methods differ in the way how the example weights are updated, which depend on a particular loss function. A common property of all of boosting methods is that a weak classifier selected in iteration $k$ has 50% error in iteration $k+1$ which prevents it to be selected again.

**History.** Boosting has been studied for two decades now. Schapire [Schapire 90] developed the first boosting procedure that could always improve the performance of a classifier by training two additional classifiers on modified versions of the training data. One year later, Freund [Freund 95] generalized the algorithm to combine arbitrary number of weak classifiers. Both of these versions assume weak classifiers which have identical error rates. This assumption was dropped by discrete AdaBoost [Freund 97] algorithm, which enabled to boost classifiers with arbitrary error rates. The strong classifier was defined as weighted majority vote. In a comparative study, Quinlan [Quinlan 96] observed that trees can be boosted into more accurate classifier if each node outputs a confidence value rather than a discrete class. Subsequently, real AdaBoost [Schapire 98b] was proposed, which allowed the classifier to output confidences and achieved a significant improvement in classification over discrete AdaBoost.

**Improvements of boosting.** Early versions of Boosting [Schapire 90, Freund 95] used a stopping criterion for the number of combined weak classifiers to avoid over-fitting. Even though, practitioners observed [Schneiderman 04] that testing error keeps decreasing and then stabilizes when the boosting is not terminated. This phenomenon was explained by the "margin theory" [Schapire 98a]. Moreover, the performance of boosting was explained from statistical point of view [Friedman 00] as an additive logistic regression. Boosting algorithms are still evolving. Robustness to noisy data was addressed by BrownBoost [Freund 01] or LogitBoost [Friedman 00]. An online variant of boosting have been also proposed [Oza 05]. A comprehensive review of the methods can be found in [Schapire 02].

**Relation to ensemble methods.** Boosting has close links to other algorithms which internally maintain a set of weak classifiers and integrate their responses by majority voting, i.e. ensemble classifiers. The underlying assumption of these methods is that the classifiers make independent errors and therefore the majority voting improves their performance [Polikar 06]. The methods differ in the way how the independence of the classifiers is achieved. Bagging [Breiman 96] uses sampling with repetition to form different training distributions. If the weak classifiers are trees, bagging becomes Randomized Forrest [Breiman 01], which has been applied in computer vision to a number of problems [Amit 97, Lepetit 05, Shotton 08]. Apart from sampling with
2.3. Machine learning

Figure 2.15: Standard AdaBoost versus cascaded AdaBoost: (TOP) standard AdaBoost has to evaluate all weak classifiers (here Haar-like filters), (BOTTOM) cascaded AdaBoost enables to reject negative examples after each stage of the classifier thus reducing the computation demands.

repetition, the classifier independence can be achieved by designing features which are likely to be independent such as in Randomized Ferns [Ozuysal 07]. Similar approach is used in our implementation.

**Boosting and sliding-window.** Boosting has been developed for general machine learning problems. However, such algorithm may struggle in some practical scenarios such as face detection and sliding window. In a typical image, the sliding-window classifier is evaluated on thousands ($\approx 10^5$) of patches and only a small fraction of them ($\approx 10$) may depict the object of interest. It follows that the positive and negative classes are extremely asymmetric, i.e. majority of windows are negative. The classifier has to be therefore tuned toward precision (probability of false positive should be around $1/10^5$). Another consequence is that the negative class is virtually unlimited and it is not straightforward to maintain a distribution of weights on it.

**Boosting and bootstrapping.** Boosting has been first applied to object detection by Viola and Jones [Viola 01] with two important modifications: (i) a cascaded classifier, and (ii) bootstrapping. The standard AdaBoost classifier measures all weak classifiers in its ensemble and outputs a decision after that. Such a strategy is too slow and not even necessary for a sliding-window classifier. As noted earlier, the majority of patches in a typical image depict background. Therefore the classifier is split into stages, each
of which rejects (classifies as negative) a subset of candidates before progressing to the next stage. This cascaded structure has a great impact on the classifier speed but also influences the quality of training. Standard training maintains only a distribution of weights and considers the training set as fixed. In contrast, training with cascaded classifier involves resampling of the training set to better represent the examples close to the current decision boundary, which has direct links to bootstrapping [Sung 98].

### 2.3.3 Semi-supervised learning

This section reviews the semi-supervised learning methods, which, in addition to labeled examples, exploit unlabeled training examples.

A natural question is whether unlabeled data can help in training of a classifier. Consider the mind experiment illustrated in figure 2.16. The goal is to train a classifier in two settings: (i) from two labeled examples, and (ii) from the identical labeled examples and a collection of unlabeled examples. Given the labeled examples only, a number of supervised methods can be used to train a classifier. Figure 2.16 (a) illustrates the decision boundary of a classifier that maximizes the classification margin. If the labeled examples are augmented by unlabeled (b) several semi-supervised learning strategies can be used. One of the simplest is to perform clustering of the unlabeled examples and label each cluster by the corresponding labeled example within it. This produces a decision boundary that is illustrated in (c). This classifier will make correct predictions as long as the clustering is aligned with the classification that is demanded (d). This example demonstrates that if the underlying distribution of unlabeled examples forms clusters that are aligned with the classification function to be learned, the unlabeled data can help when training a classifier. This property of unlabeled data, also referred to as “cluster assumption”, is often assumed by the semi-supervised learning algorithms [Chapelle 06]. A number of algorithms relying on similar assumptions have been proposed in the past including Expectation-Maximization, self-learning and co-training.

**Expectation-Maximization** (EM) is a generic method for finding estimates of model
EM is an iterative process, which in case of binary classification alternates over: (i) estimation of soft-labels of unlabeled data, and (ii) training a classifier exploiting the soft-labels. EM was successfully applied to document classification [Nigam 00] and learning of object categories [Fergus 03]. In the semi-supervised learning terminology, EM algorithm relies on the “low density separation” assumption [Chapelle 06], which means that the classes are well separated in the feature space. EM is sometimes interpreted as a “soft” version of Self-learning [Zhu 09].

**Self-learning** starts by training an initial classifier from a labeled training set, the classifier is then evaluated on the unlabeled data. The examples with the most confident classifier responses are added to the training set and the classifier is retrained. This is an iterative process. The self-learning has been applied to training of a human eye detector [Rosenberg 05]. However, it was observed that the detector improved more if the unlabeled data was selected by an independent measure rather than the classifier confidence. Rosenberg et al. suggested that the low density separation assumption is not satisfied for object detection and other approaches may work better.

**Co-training** [Blum 98] is a learning method built on the idea that independent classifiers can mutually train one another. To create such independent classifiers, co-training assumes that two independent feature-spaces are available. See figure 2.17 for illustration of the process. The training is initialized by training of two separate classifiers using the labeled examples. Both classifiers are then evaluated on unlabeled data. The confidently labeled samples from the first classifier are used to augment the training set of the second classifier and vice versa in an iterative process. Co-training works best for problems with independent modalities, e.g. text classification [Blum 98] (text and hyper-links) or biometric recognition systems [Poh 09] (appearance and voice). In visual object detection, co-training has been applied to car detection in surveillance [Levin 03] or moving object recognition [Javed 05]. We argue that co-training is not a good choice for object detections, since the examples (image patches) are sampled from a single modality. Features extracted from a single modality may be dependent and therefore violate the assumptions of co-training.
2.4 Observations

**Tracking.** We reviewed methods starting from the most basic approaches such as template tracking, up to complex trackers which define tracking as classification and update the classifier during tracking. Trackers are becoming increasingly complex to handle increasingly challenging environments and appearance changes. The assumption is that the object state in previous frame is known. In unconstrained video, this assumption is violated and therefore any tracker eventually fails. The tracking failure will be investigated in chapter 3.

**Detection.** Object instance detection reached a level of maturity for scenarios where a sufficient number of training examples can be either generated or annotated. A cascaded architecture, or fast keypoint detectors enable tracking-by-detection, which overcomes drift and initialization problems of contemporary trackers. The underlying assumption of object detectors is a strict separation of training and run-time. This limits their applicability to objects that can be modeled in advance. Methods that enable efficient online update of detectors have been proposed, but the problem how to update these detectors was not addressed.

**Learning.** We reviewed two classes of learning approaches for training of object detectors: supervised and semi-supervised. In the supervised setting the learning is often realized by boosting. However, the standard variants of boosting, do not perform well and have to be combined with bootstrapping to handle large training sets. Chapter 4 closely analyses this relation between boosting and bootstrapping and proposes a unifying algorithm. In the semi-supervised setting we discussed several approaches popular in text classification, however in object detection the improvements are marginal. One of the reasons is that in object detection it is hard to find independent features that would efficiently drive the learning process. Chapter 5 discusses other information sources (spatio-temporal structure in data) that could be used for training an object detector.
Chapter 3

Tracking: failure detection

This chapter investigates long-term tracking from the perspective of frame-to-frame tracking, the first component of our system. The starting point is the observation that any frame-to-frame tracker eventually fails, for instance, when the object moves out of the camera view. Therefore, the primary goal of this chapter is to estimate the reliability of a tracker and use it to detect the tracking failures. The secondary goal is to use the reliability measure to improve the frame-to-frame tracking itself.

The chapter is structured as follows. Section 3.1 focuses on point trackers and proposes a novel error measure that evaluates the reliability of arbitrarily long point trajectories. Section 3.2 develops a novel tracker called Median-Flow. The object is represented by a grid of points, which influence the estimated object motion based on their reliability. The Median-Flow tracker is comparatively evaluated with relevant approaches and superior performance is achieved. The chapter is concluded in section 3.3.

3.1 Detection of tracking failures

This section is concerned with the detection of tracking failures of point trackers, e.g. Lucas-Kanade tracker [Lucas 81]. These trackers build up a point trajectory from a sequence of frame-to-frame displacements. Such trackers are: (i) prone to drift due to accumulation of localization errors, and (ii) likely to fail if the point suddenly changes appearance, gets occluded or disappears from the camera view.

Our approach to failure detection is based on so called forward-backward consistency assumption that correct tracking should be independent of the direction of time-flow. Algorithmically, this assumption is exploited as follows. First, a tracker produces a trajectory by tracking a point forward in time. The trajectory has arbitrary length, and can be obtained by an arbitrary tracker. Second, the point location in the last frame of the forward trajectory initializes a validation trajectory. The validation trajectory is
Figure 3.1: Illustration of the Forward-Backward error measure: (TOP) consistent (1) and inconsistent (2) trajectories, (BOTTOM) terms used for the definition of the Forward-Backward error measure.

obtained by backward tracking. Third, the two trajectories are compared and if they differ significantly, the forward trajectory is considered as incorrect.

Figure 3.1 (TOP) illustrates the method when tracking a point between two images (trajectory of length one). Point no. 1 is visible in both images and the tracker is able to localize it correctly. Tracking this point forward or backward results in consistent trajectories. On the other hand, point no. 2 is not visible in the right image and the tracker localizes a different point. Backward-tracking of this point results in a different location than the original one. The corresponding forward and backward trajectories are significantly different in that case.

Related work. A commonly used approach to detect tracking failures is to compare consecutive point-centered patches using Sum of Square Differences (SSD) [Bouguet 99, Nickels 02]. High SSD means that the patches are dissimilar, which indicates tracking failure. This differential error indicates failures caused by occlusion or rapid movements, but does not detect slowly drifting trajectories. The detection of drift can be approached by defining an absolute error. A popular approach is to consider affine warps of the initial patch [Shi 94] and compare them to the current patch. Recently, a general method for assessing the tracking performance was proposed [Wu 07], which is based on the “forward-backward” idea. The approach was designed for particle filters. Adaptation of their method to template tracking was not suggested. Moreover, the idea of forward-backward consistency is often used wide-baseline matching [Strecha 03] or
3.1. Detection of tracking failures

in optical flow estimation [Alvarez 07]. These methods typically consider a pair of images. In contrast, our approach defines the error for arbitrarily long point trajectories.

3.1.1 Forward-Backward error

This section defines the Forward-Backward (FB) error, see figure 3.1 (BOTTOM) for illustration. Let \( S = (I_t, I_{t+1}, \ldots, I_{t+k}) \) be an image sequence and \( x_t \) be a point location in time \( t \). Using an arbitrary point tracker, the point \( x_t \) is tracked forward for \( k \) steps. The resulting trajectory is \( T^k_f = (x_t, x_{t+1}, \ldots, x_{t+k}) \), where \( f \) stands for forward and \( k \) indicates the length of the trajectory. Our goal is to estimate the reliability of trajectory \( T^k_f \) given the image sequence \( S \). For this purpose, the validation trajectory is first constructed. Point \( x_{t+k} \) is tracked backward up to the first frame and produces \( T^k_b = (\hat{x}_t, \hat{x}_{t+1}, \ldots, \hat{x}_{t+k}) \), where \( \hat{x}_{t+k} = x_{t+k} \).

The Forward-Backward error is defined as the distance between the forward and the backward trajectory:

\[
FB(T^k_f | S) = \text{distance}(T^k_f, T^k_b).
\]

In our implementation, we use the Euclidean distance between the initial point of the forward trajectory and the end point of the backward trajectory:

\[
\text{distance}(T^k_f, T^k_b) = || x_t - \hat{x}_t ||.
\]

The Forward-Backward error provides a real value. Higher values indicate the forward-backward inconsistency and possibility of tracking failures. The hard decision whether the tracker failed is done by a thresholding.

3.1.2 Quantitative evaluation

This experiment quantitatively evaluates the ability of FB and SSD to identify correctly tracked points (inliers) between two frames. One hundred images depicting scenes of nature were warped by random affine transformations and Gaussian noise was added. This process resulted in a set of one hundred image pairs. In the original images, a set of points was initialized on a regular grid. Two types of displacements of these points were then established: (i) ground truth displacements were obtained by projecting the points to the warped images, and (ii) estimated displacements were obtained by tracking the points using Lucas-Kanade tracker [Lucas 81] to the warped images. The estimated displacements that ended up closer than 2 pixels from the ground truth were labeled as inliers. The inliers represented approximately 65% of all displacements. The estimated displacements were then evaluated by a failure detection measure (FB, SSD), displacements with error below \( \theta \) were classified as inliers. A correct classification of an inlier is denoted as a true positive (TP), an incorrect classification is denoted
as false positive (FP). The quality of the error measures is accessed using precision and recall statistics, which are both function of threshold \( \theta \):

\[
\text{precision} = \frac{TP}{TP + FP}, \quad \text{recall} = \frac{TP}{\#\text{inliers}}.
\]

Figure 3.2 (TOP) shows the resulting performance of the FB error as a function of the threshold \( \theta \). The figure illustrates that the proposed FB error is able to reliably recognize majority of inliers at high precision level. Notice for instance the working point indicated by the red dot. For \( \theta = 1 \) pixel, the recall is of 95% and precision of 96%. Figure 3.2 (BOTTOM) shows the corresponding precision and recall curves for FB in comparison to SSD. FB significantly outperforms SSD for majority of working points. SSD was unable to detect inliers for small thresholds, its precision starts below 70% (notice that random guessing would start around 65%). We have carried out another experiment where the regular grid of features was replaced by FAST [Rosten 06] feature points and consistent observations were made.

### 3.1.3 Visualization

This sub-section visualizes the proposed Forward-Backward error for long point trajectories. Given a video sequence, every pixel in the first frame initializes a point
3.1. Detection of tracking failures

Figure 3.3: Error map: visualization of the Forward-Backward error. Dark colors in the error map indicate low Forward-Backward error that considers 50 frames. Yellow points in the original frames indicate 5% of the most reliable points.

The error map has been constructed for a sequence of 50 frames, with a moving camera following two pedestrians (sequence PEDESTRIAN 1). The figure shows the error map as well as the first and the last frame of the sequence. The error map encodes the pixel reliability by colors. Dark colors indicate low FB error: shadow cast by tree branches (A), upper bodies of two pedestrians (B). Any point selected from these “reliable” areas is tracked accurately in the whole sequence. Brighter colors indicate areas which were evaluated as not reliable. These areas may become occluded (C), disappear from the camera view (D) or lack enough texture (E). The first and the last frame also depict the 5% of the most reliable pixels. Notice that these points correspond to identical physical points, which is best visible in the zoomed-in versions.

**Application.** The error map can be used to detect key-points that can be tracked reliably throughout the entire video. This is in contrast to traditional key-point detectors [Shi 94, Rosten 06], which localize the key-points based on a single frame only and therefore cannot guarantee that the point will be reliably tracked. Moreover, it has been observed [Jepson 03] that tracker performance increases if the tracker focuses on reliably tracked parts of a template. The Forward-Backward error allows identification of such parts. This aspect will be studied in the following section.
3.2 Median-Flow tracker

The aim of this section is to use the proposed Forward-Backward error to build a better bounding box-based tracker. The basic approach is to estimate the bounding box motion using a large number of independent points, measure their reliability and integrate the predictions using a robust statistic. Using this approach, we aim at a tracker that is robust to partial occlusions, is fast and efficient.

![Block diagram of the Median-Flow tracker](image)

Figure 3.4: The block diagram of the Median-Flow tracker.

The block diagram of the proposed tracker is shown in figure 3.4. The tracker accepts a pair of images $I_t, I_{t+1}$ and a bounding box $\beta_t$ and outputs the bounding box $\beta_{t+1}$. A set of points is initialized on a rectangular grid within the bounding box $\beta_t$. These points are tracked by Lucas-Kanade tracker from $I_t$ to $I_{t+1}$. The quality of the point displacements is then estimated and each point is assigned an error (e.g. FB, SSD or NCC). 50% of the worst predictions is filtered out. The remaining predictions estimate the bounding box motion using median. We refer to this tracker as Median-Flow.

**Estimation of the motion by median.** The bounding box motion is parameterized by horizontal displacement, vertical displacement and scale change. All three parameters are estimated independently using median. Figure 3.5 illustrates the estimation of the displacements. The scale change is estimated as follows: for each pair of points, a ratio between current point distance and previous point distance is computed. The bounding box scale change is defined as the median over these ratios.
3.2. Median-Flow tracker

A number of variants of the Median-Flow tracker were tested. The baseline tracker $T_0$ estimates the bounding box displacement based on all points on the grid. Trackers $T_{FB}, T_{NCC}, T_{SSD}$ estimate the point reliability using FB, NCC and SSD, respectively. FB performs one backward prediction, NCC and SSD compare consecutive point-centered patches. 50% of the worst points is filtered out. Tracker $T_{FB+NCC}$ combines FB and NCC, each error measure independently filters out 50% of the worst points. These trackers were compared with the following approaches: Incremental Visual Tracking (IVT) [Ross 07], Online Discriminative Features (ODF) [Collins 05], Ensemble Tracking (ET) [Avidan 07] and Multiple Instance Learning (MIL) [Babenko 09]. The evaluation was performed on 6 video sequences from [Yu 08]. The sequences contain appearance changes, fast motion, and partial occlusion and in some cases the object disappears from the camera view and later reappears. The sequences are described in detail in Appendix B.

**Evaluation protocol.** The objects were manually initialized in the first frame and tracked up to the end of the sequence. The trajectory was considered correct if the bounding box overlap with ground truth was larger than 50%. The overlap was defined as a ratio between intersection and union of two bounding boxes. Performance was assessed as the maximal frame number up to which the tracker was correct.

Table [3.1] shows the quantitative results for all sequences. The last row shows the number of times the particular algorithm performed best. The best results obtained the Median-Flow based on a combination of FB and NCC error. This tracker was able to score best three times.
Table 3.1: Comparison of the Median-Flow tracker with state-of-the-art approaches in terms of the number of correctly tracked frames.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Frames</th>
<th>IVT</th>
<th>ODF</th>
<th>ET</th>
<th>MIL</th>
<th>$T_0$</th>
<th>$T_{FB}$</th>
<th>$T_{NCC}$</th>
<th>$T_{SSD}$</th>
<th>$T_{FB+NCC}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>761</td>
<td>17</td>
<td>n/a</td>
<td>94</td>
<td>135</td>
<td>93</td>
<td>761</td>
<td>144</td>
<td>28</td>
<td>761</td>
</tr>
<tr>
<td>Jumping</td>
<td>313</td>
<td>75</td>
<td>313</td>
<td>44</td>
<td>313</td>
<td>36</td>
<td>76</td>
<td>87</td>
<td>79</td>
<td>170</td>
</tr>
<tr>
<td>Pedestrian 1</td>
<td>140</td>
<td>11</td>
<td>6</td>
<td>22</td>
<td>101</td>
<td>15</td>
<td>37</td>
<td>40</td>
<td>12</td>
<td>140</td>
</tr>
<tr>
<td>Pedestrian 2</td>
<td>338</td>
<td>33</td>
<td>8</td>
<td>118</td>
<td>37</td>
<td>97</td>
<td>97</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>Pedestrian 3</td>
<td>184</td>
<td>50</td>
<td>5</td>
<td>53</td>
<td>49</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>Car</td>
<td>945</td>
<td>163</td>
<td>n/a</td>
<td>10</td>
<td>45</td>
<td>248</td>
<td>510</td>
<td>394</td>
<td>353</td>
<td>510</td>
</tr>
<tr>
<td>Best</td>
<td>n/a</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 3.6 shows the bounding box overlap with the ground truth as a function of time for several variants of the Median-Flow tracker. The baseline $T_0$ tracker as well as $T_{SSD}$ loose the object quickly after initialization (overlap goes to zero). $T_{FB}$ and $T_{NCC}$ perform better and are able to follow the object for twice as long than the baseline method. $T_{FB+NCC}$ dominates and is able to track the target throughout entire sequence. This result shows partial complementarity of FB and NCC.

Figure 3.7 shows several frames from the sequence CAR overlaid with the output of the Median-Flow tracker (yellow bounding box) and the 50% of the most reliable points estimated by the combined FB+NCC error measures (blue dots). The car is tracked across a partial occlusion caused by a traffic light. Notice, that the selected points are always covering visible, well textured parts of the car, while the occluded parts are filtered out.
3.3 Conclusions

This chapter was concerned with frame-to-frame tracking in unconstrained video sequences and focused on detection of tracking failures. We confirmed two ideas that appeared in tracking literature already: (i) forward-backward consistency assumption can be used to measure tracker’s reliability [Wu 07], (ii) frame-by-frame tracking can be improved by focusing on reliable parts [Jepson 03].

First, we proposed a novel measure, Forward-Backward (FB) error, which evaluates reliability of arbitrarily long trajectories. The benefits of the measure and complementarity to appearance-based SSD were shown. Second, we developed a novel frame-to-frame tracker, Median-Flow, which internally estimate reliability of its components. We showed that the Median-Flow works best when the reliability is estimated using combined FB and NCC errors. A superior performance in comparison to state-of-the-art trackers was demonstrated on several sequences.

As any other frame-to-frame tracker, also the Median-Flow eventually fails. This is apparent in table 3.1 (sequences JUMPING, PEDESTRIAN 2 and PEDESTRIAN 3) where all tested trackers failed due to full occlusion or disappearance of the tracked object. This shows that the frame-to-frame tracking has its limitation and a different model has to be developed in order to handle such sequences. This problem is addressed in chapter 6 which proposes the TLD framework.
Chapter 4

Detection: supervised bootstrap

This chapter investigates long-term tracking from the perspective of object detection, the second component of our system. For a large number of tracking scenarios, the visual class of the object of interest is known. It is therefore possible to train an object class detector in advance and integrate it into the tracking process. This approach has been successfully applied to a variety of scenarios including tracking of human faces [Li 07], pedestrians [Leibe 07] or ice-hockey players [Okuma 04]. If the information about the object class is available, a long-term tracker should be able to use it. Therefore, our goal is to develop a method for offline learning of object class detectors. In particular, we focus on supervised learning from large data sets using boosting [Freund 97] and bootstrapping [Sung 98].

The chapter is organized as follows. In section 4.1 we introduce the problem of large-scale learning of object detectors. Section 4.2 reviews the related work. Section 4.3 develops a novel learning method that integrates boosting and bootstrapping optimally. A number of experiments show the properties of the learning method. Section 4.4 applies the learning method to training of a face detector and state-of-the-art performance is achieved. The chapter is concluded in section 4.5.

4.1 Introduction

Consider a class of detectors that are based on a scanning window and binary classification of image patches [Viola 01]. The crucial problem of these detectors is to train the binary classifier that covers all appearances of the object and discriminates from arbitrary background clutter. Efficient processing of large data sets that cover all the variations is an important problem.

With the information available on the Internet, it is possible to acquire pools of training data of virtually unlimited sizes. Large pools sideline the problems related to
classifier generalization \cite{Vapnik98}, however, due to computational limitations and training complexity, it is rarely the case that the entire pool can be processed at once. As we have reviewed in sub-section \ref{sec:active-set}, one possible solution is to use bootstrapping and train the classifier iteratively using an active set that focuses on the decision boundary only. In object detection, bootstrapping is often combined with boosting \cite{Viola01,Fleuret08}. The combination is, however, often ad hoc \cite{Sochman05}, which presents scope for improvements.

**Contributions.** The main contribution this chapter is the optimal integration of bootstrapping with boosting, where the bootstrapping is formalized as a weighted sampling strategy. We build on the weighted sampling approach \cite{Fleuret08} and propose a generalized strategy based on Quasi-random Weighted Sampling + Trimming (QWS+) that optimally combines its components in order to minimize the variance of hypothesis error in each round of boosting. The QWS+ sampling method is applied to face detection and leads to a significant increase in the classification performance as well as the training efficiency on large pools of data.

### 4.2 Related work

In object detection, bootstrapping has been first used to train a Gaussian mixture model \cite{Sung98}. In every round of training, the active set was extended by examples misclassified in previous round and the entire classifier was retrained. The same approach was applied to refine the decision surface of an SVM classifier \cite{Papageorgiou98,Dalal05}, to train the neural network \cite{Rowley98}, or to estimate the histograms of Wavelet coefficients \cite{Schneiderman04}.

Viola and Jones \cite{Viola01} proposed a cascaded classifier composed of a sequence of stages. A bootstrapping strategy based on re-sampling of the undecided parts of the pool was used. Many authors followed this approach \cite{Levi04,Jones03,Laptev06}. A number of improvements have been proposed for the classical Viola-Jones cascade. For example, the standard cascade ignores the confidence which is output from each stage thus making the classifier unnecessarily long. This drawback was addressed in \cite{Sochman05,Wu04,Xiao03} where bootstrapping based on (random) re-sampling and re-weighting of the selected samples according to the confidence was used. Later on, Fleuret and Geman \cite{Fleuret08} proposed the weighted sampling. Instead of weighting examples in the active set, examples were sampled with replacement from the pool with respect to their weights. The approach let to improved classifier performance.

Apart from iterations on the Viola and Jones detector, Friedman et al. \cite{Friedman00} proposed trimming, a technique that selects only a fraction of examples with the highest weights resulting in improvement of training speed.
4.3 Integration of bootstrapping and boosting

This section formalizes the combination of boosting and bootstrapping. The analysis will be performed on AdaBoost formulated as a logistic regression [Friedman 00], but the proposed method is applicable to other boosting methods as well. The bootstrapping will be formulated as a sampling strategy. Next, we present the commonly used approaches within our framework and develop two novel strategies, one of which is optimal. All strategies will be evaluated on several different experiments.

Let \( X = \{(x_1, y_1), \ldots, (x_m, y_m)\} \) be a pool of training pairs where each example \( x_i \) belongs to an example space \( \mathcal{X} \), and each label \( y_i \) belongs to a finite label space \( \mathcal{Y} = \{-1, 1\} \). Given this input, the AdaBoost builds a strong hypothesis \( H : \mathcal{X} \rightarrow \mathbb{R} \) which has the form of \( H(x) = \sum_{j=1}^{T} h_j(x) \) where each \( h_j(x) \) is a weak hypothesis. The sign of \( H(x) \) gives the classification, and \( |H(x)| \) measure of prediction confidence.

The strong hypothesis is built in \( T \) rounds. During training, AdaBoost maintains a distribution of weights \( D = \{d_i\} \) satisfying \( \sum_{i=1}^{m} d_i = 1 \) giving higher weights to examples that are “difficult”. In each round, AdaBoost selects a hypothesis that performs best on current distribution of weights, adds this hypothesis to the strong hypothesis and updates the distribution so that the currently selected hypothesis performs worst in the next round. See figure 4.1 (LEFT) for illustration of the boosting process.

Boosting assumes access to a set of weak classifiers, each of which splits the example space into a number of partitions. For each classifier, AdaBoost estimates a hypothesis of the form

\[
h(x) = \frac{1}{2} \log \frac{P_D(y = 1|x)}{1 - P_D(y = 1|x)}, \tag{4.1}
\]

where \( P_D(y = 1|x) \) is a probability estimated using current distribution of weights \( d_i \).
Boosting then selects the optimal hypothesis $h^*$ that minimizes

$$h^* = \arg\min_h Z_D(h) = \arg\min_h \sum_{i=1}^m d_i e^{-y_i h(x_i)}, \quad (4.2)$$

where $Z_D(h)$ is the exponential loss function. Once such a hypothesis is selected, AdaBoost updates the weights using $d_i \leftarrow d_i e^{-y_i h^*(x_i)}$ and normalizes them to a distribution.

**Bootstrapping as a sampling strategy**

Standard AdaBoost is not designed for large pools, and therefore, if $m$ is large, the steps 4.1 and 4.2 may be too time consuming or even not feasible. Therefore, some form of approximation is needed. Bootstrapping is a process that approximates the pool of size $m$ by an active set of size $n$, where $n \ll m$. The hypothesis estimation as well as the selection of the best hypothesis is then performed considering the active set only. The crucial question is how to approximate the pool correctly. We address this question by introducing the so-called sampling strategy, which is a process that takes the distribution of weights $D$ of size $m$ and approximates it by a distribution $\hat{D}$ where at most $n$ elements have non-zero value. The approximated distribution must satisfy $\sum_{i=1}^m \hat{d}_i = 1$. Notice that in general, $d_i \neq \hat{d}_i$ since both of these weights have to sum-up to one.

Every realization of the sampling strategy results in a different distribution $\hat{D}$ and thus different weights $\hat{d}_i$. It follows that $\hat{d}_i$ is a random variable with expectation $E[\hat{d}_i]$ and variance $\text{Var}[\hat{d}_i]$. If $E[\hat{d}_i] = d_i$ we consider the estimated weights unbiased. Suppose we have a hypothesis $h$ with a corresponding exponential loss of $Z_D(h)$ on distribution $D$. When bootstrap is employed, this value is estimated on the active set $\hat{D}$ according to the following equation:

$$Z_{\hat{D}}(h) = \sum_{i=1}^m \hat{d}_i e^{-y_i h(x_i)} \quad (4.3)$$

where the estimated weights are linearly combined by the constant coefficients $e^{-y_i h(x_i)}$. Therefore, if the estimated weights $\hat{d}_i$ are unbiased then the estimated exponential loss is unbiased as well. If $E[Z_{\hat{D}}(h)] = Z_D(h)$ then the entire strategy is unbiased. $\text{Var}[Z_{\hat{D}}(h)]$ will be referred to as a variance of error estimate. The terms bias and variance are illustrated in figure 4.2.

**Conditions for the optimal sampling strategy.** Our objective is to find the sampling strategy that selects $n$ unique samples out of the pool of size $m$, is unbiased and with minimal variance of error estimate. Such strategy would guarantee that the approximate hypothesis $\hat{h}$ is as close as possible to the optimal one $h^*$, which would be trained on the entire pool.
4.3. Integration of bootstrapping and boosting

Figure 4.2: Illustration of bias and variance of a bootstrapping strategy.

4.3.1 Known sampling strategies

Several bootstrapping strategies have been used in the context of object detection. Here we formalize the most popular approaches within our framework.

**Trimming (T).** This technique selects $n$ samples from the pool with the highest weights. The weights of the selected samples are then normalized so that $\sum \hat{d}_i = 1$. This strategy was introduced in [Friedman 00], where $n$ was set so that a predefined fraction ($90 - 99\%$) of the total weight mass was used for training. If $n < m$ then the weights are biased $E[\hat{d}_i] \neq d_i$ with zero variance.

**Unique Uniform Sampling (UUS).** This strategy selects $n$ unique samples from the pool with a probability of selecting sample $i$ equal to $P(i) = \frac{1}{m}$. The weights of the selected samples are then normalized so that $\sum \hat{d}_i = 1$. The estimated weights are in general biased (due to normalization) and have high variance since it is likely to disregard examples that carry significant mass of the distribution. Despite this it is often used in practice [Sochman 05, Wu 04, Xiao 03].

**Weighted Sampling (WS).** Selects $n$ samples with replacement from the pool. Sample $i$ is selected with probability $P(i) = d_i$ and assigned weight $\hat{d}_i = \frac{1}{n}$. WS is unbiased, since $E[\hat{d}_i] = \frac{Nd_i}{n} = d_i$. Possible implementation: the example weights are represented as intervals arranged on a unit line segment. Next we generate $n$ random points on this line segments the positions of which determine the index of the selected sample. Since the example weight are approximated by repeated selection of the sample, this strategy does not guarantee $n$ unique samples in the active set. This strategy was proposed in [Fleuret 08].
4.3.2 Proposed sampling strategies

In this section, we propose new sampling strategies based on WS. These strategies are designed to reduce the variance of error estimate by quasi-random sampling and guarantee selection of \( n \) unique samples in the active set.

Quasi-random Weighted Sampling (QWS). QWS reduces variance of the error estimate by pseudo-random sampling [Press 92]. The weights are represented as intervals and arranged to a unit line segment. The line segment is split into \( n \) equal intervals. Within each interval, one random number is generated the position of which determines the index of the selected sample. This process maintains the weights unbiased but significantly reduces their variance. Intuitively, each sample can be selected at most \( p \)-times, where \( p \) is number of intervals the sample is covering. For standard WS, each sample can be selected at most \( n \)-times. Since \( p \ll n \), QWS significantly reduces the variance of the estimated weights. However, QWS selects a unknown number of unique samples.

Quasi-random Weighted Sampling + Trimming (QWS+). The QWS+ is a generalization of Trimming and QWS, parameterized by \( k \), the number of samples with largest weights in the pool that are always selected (trimmed). For \( k = n \), QWS+ coincides with trimming. For \( k = 0 \), QWS+ becomes a QWS. For any \( 0 \leq k < n \), the QWS+ strategy is unbiased, since weighted sampling is unbiased and calculations of exponential loss on the trimmed set is exact. Parameter \( k \) is set to minimize the variance of error estimate.

Intuitively, QWS+ selects \( k \) most dominant samples that would be selected by QWS with probability at least 50\% and assigns them their own weights. The remaining weight mass is distributed amongst the remaining \( l = n - k \) samples selected by QWS. Trimming of the most dominant samples reduces the variance of their estimated weight and hence the variance of the entire distribution.

In the following theorem, we show the optimal setting of the parameter \( k \). We use the following notation. Index in parenthesis denotes \( (i) \)-th sample with largest weight. \( S = \{X, D\} \) represents the pool augmented with the distribution of weights that shall be called the weighted pool. \( S_k \) is the weighted pool without \( k \) samples with largest weights, \( D_k = 1 - \sum_{i=1}^{k} d_{(i)} \) is the weight mass of \( S_k \), and \( Z_{QWS+}(S, h) \) is the exponential loss of hypothesis \( h \) on weighted pool \( S \) when approximated by sampling strategy QWS+.

Theorem: The optimal size of trimming is the largest \( k = n - l \) that satisfies the condition \( \frac{1}{l} \leq -\frac{\delta_{(k)}}{D_k} (2 + \frac{\delta_{(k)}}{D_k}) \) if variance of exponential loss of \( h \) changes "slowly" (defined below).
4.3. Integration of bootstrapping and boosting

**Proof:** First, we express the exponential loss of QWS+ strategy as two components (trimming and weighted sampling) parameterized by $k$:

$$Z_{QWS+}(S, h) = \sum_{i=1}^{k} d_i e^{-y_i h(x_i)} + Z_{QWS}(S_k, h)$$  \hspace{1cm} (4.4)

$$\approx \sum_{i=1}^{k} d_i e^{-y_i h(x_i)} + Z_{WS}(S_k, h).$$  \hspace{1cm} (4.5)

For the purpose of the proof, we assume that $Z_{QWS}(S_k, h) \approx Z_{WS}(S_k, h)$ which results in simplified analysis. The difference between sampling with and without replacement is in this case small since all $k$ dominant samples were already removed by trimming.

Weighted sampling selects $l = n - k$ samples with replacement from $S_k$. Sample $i$ is selected with probability $P(i) = \frac{d_i}{D_k}$ and assigned weight $\hat{d}_i = \frac{D_k}{l}$. The random variable $Z_{WS}$ is thus a re-scaled sum of $l$ random variables corresponding to each sample:

$$Z_{WS} = \frac{D_k}{l} \sum_{s=1}^{l} Z_{WS}^s,$$  \hspace{1cm} (4.6)

where random variables $Z_{WS}^s$ attain the value $e^{-y_{i(s)} h(x_{i(s)})}$ with probability $\frac{d_i}{D_k}$, $i(s)$ is the index chosen in step $s$.

Next, we express the variance of the exponential loss. From equation 4.5 we can see that variance of $Z_{QWS+}$ can be approximated by the variance of $Z_{WS}$ since a trimming has variance equal to zero. Variance of $Z_{WS}$ is further substituted from equation 4.6 yielding the form:

$$\text{Var}[Z_{QWS+}] \approx \text{Var}[Z_{WS}] = \text{Var}\left[\frac{D_k}{l} \sum_{s=1}^{l} Z_{WS}^s\right] = \frac{D_k^2}{l} \sum_{s=1}^{l} \text{Var}[Z_{WS}^s].$$  \hspace{1cm} (4.7)

After substituting $\sum_{s=1}^{l} \text{Var}[Z_{WS}^s]$ with $c_k$, we obtain a simplified equation

$$\text{Var}[Z_{QWS+}] \approx \frac{D_k^2}{l} c_k.$$  \hspace{1cm} (4.8)

Our goal is not to find conditions for $k$ that minimize the variance. We assume that $c_k$ changes slowly as a function of $k$, i.e. $\frac{c_k}{c_{k-1}} \approx 1$. In the following, we express the
optimality condition that variance of exponential loss for $k$ must be smaller than for $k - 1$:

$$
\frac{D^2_k}{l} c_k \leq \frac{D^2_{k-1}}{l+1} c_{k-1} = \frac{(D_k + d_{(k)})^2}{l+1} c_{k-1}, \quad \frac{c_k}{c_{k-1}} \approx 1
$$

$$
\frac{l + 1}{l} \leq \frac{(D_k + d_{(k)})^2}{D_k^2}
$$

$$
\frac{1}{l} \leq \frac{d_{(k)}}{D_k} (2 + \frac{d_{(k)}}{D_k}) \approx \frac{2d_{(k)}}{D_k},
$$

which concludes the proof. In practice, the optimal $k$ is found iteratively. Samples are trimmed until the equation 4.10 is satisfied.

### 4.3.3 Properties of sampling strategies

This sub-section comparatively evaluates the existing and proposed sampling strategies on synthetic data. In particular, the impact of the sampling strategy on the bias and variance is investigated.

**Estimation of weights.** This experiment visualizes the estimation of weights for the discussed strategies. A pool of size $m = 15$ with randomly assigned weights $d_i$ was randomly generated. The active set of size $n = 5$ was sampled 1000 times, each realization outputs a set of estimated weights $\hat{d}_i$. For each estimated weight and strategy, the expected value and the variance was computed as shown in figure 4.3. The circles in the figure denote the expectation of the estimated weight. The variance of the estimated weights is depicted by the red error bars. Notice that the expectation of Trimming and UUS is biased, the remaining strategies are not. Note the reduction of variance by Quasi-random Weighted Sampling strategies (QWS, QWS+). Furthermore, QWS+ completely eliminates variance on large examples which leads to the minimal variance of all tested strategies.
4.3. **Integration of bootstrapping and boosting**

![Figure 4.4](image)

**Figure 4.4:** The variance and bias of the sampling strategies. The dashed line shows the exponential loss of the optimal hypothesis. See text for more details.

**Estimation of exponential loss.** In this experiment, we created a pool $S$ containing $m = 95,000$ examples with distribution of weights after the 100th iteration of training a face detector (details later). Using this entire training set, the optimal hypothesis $h^*$ was found and its exponential loss $Z_D(h^*)$ was measured. The pool was then sampled by each strategy to obtain 1000 active sets of size $n = 500$. Next, the optimal hypothesis was tested on each active set to obtain its approximated exponential loss $\hat{Z}_D(h^*)$. The expectation and variance of $\hat{Z}_D(h^*)$ is shown in figure 4.4 (a). Weighted sampling reduces significantly the variance of $\hat{Z}_D(h^*)$ and keeps mean unbiased. Intuitively, the exponential loss is dominated by examples with high weights, weighted sampling forces these examples to be included in every realization of the active set, and hence the exponential loss in different trials is similar. Notice that the variance for QWS and QWS+ is further reduced with respect to WS.

**Training of hypothesis.** In this experiment we use the same sets as defined in the previous paragraph. The optimal hypothesis $h^*$ was first trained on full pool and the corresponding exponential loss $Z_D(h^*)$ was measured. The hypothesis was trained by domain partitioning approach [Schapire 99]. Next, for each strategy and realization of active set, we trained an approximate hypothesis $\hat{h}$. We estimated the training exponential loss $Z_D(\hat{h})$ and testing exponential loss $Z_D(\hat{h})$. Figure 4.4 (b, c) shows the mean and variance of the training and testing exponential loss. Note, that UUS sampling overfits the training data more than any other strategies, i.e. training loss is lower and testing loss is higher than the optimal loss value. Furthermore, the variance of UUS is the highest of all the strategies. The variance is significantly reduced for WS. QWS and QWS+ further reduce the variance, but the difference with respect to WS is subtle. This apparently insignificant difference in the variance reduction proves to be of importance, when full detector training in evaluated.
Face database | Details
---|---
Frontal ($S_{\text{FRONT}}$) | 3,085 frontal faces collected from www.betaface.com, annotated by a 3rd party face detector + manual check.
Profile ($S_{\beta \text{PROF}}$) | 3,384 manually annotated profile faces from 8 movies, $\beta$ indicates in-plane rotation.
Background ($S_{\text{NONE}}$) | 3,310 images that were manually checked to contain no faces.

Table 4.1: Data sets used for training of a face detector.

4.4 Application to face detection

This section applies the proposed learning method to training of several face detectors (frontal, profile, specific). We build on WaldBoost [Sochman 05] algorithm, which uses UUS strategy.

**Training details.** We have collected three training sets shown in table 4.4 which contain frontal, profile and multi-view faces as well as images depicting background. The training sets consist of tightly cropped face-patches of size $28 \times 28$ pixels. The patches are described using three types of features: (i) Haar-like wavelets [Jones 03], (ii) Local Binary Patterns (LBP) [Ojala 02], and (iii) Histogram of Orientated Gradients (HOG) [Dalal 05] projected into one dimension using weighted Linear Discriminant Analysis [Laptev 06]. We generate a feature set of approximately 12,000 Haar, 12,000 LBP and 400 HOG features differing in their localization within the patch, scale and aspect ratio. The resulting detectors are tested on standard CMU-MIT[1] data sets and compared with other approaches using Receiving Operating Characteristic (ROC) and Precision and Recall Curves (PRC).

4.4.1 Frontal face detector

This experiments trains a frontal face detector and investigates the influence of relative active set size on the detector performance. We used the pool of 10,000 positive examples and 56 million background patches. The positive examples were generated from $S_{\text{FRONT}}$ using shift and in-plane rotation of the original face-patches. The negative patches were sampled from $S_{\text{NONE}}$ using a sliding window. This pool was approximated by active sets of three different sizes $n \in \{100, 1,000, 10,000\}$. For each size and sampling strategy (UUS, WS, QWS, QWS+), a strong classifier was trained up to the length of 200 features. Trimming was also tested but its performance was much worse than any other strategy.

Figure 4.5 shows the resulting PRC curves on CMU-MIT data set. The performance of all sampling strategies decreases with smaller active set size. For large active sets \((n = 10000)\) there is almost no difference in performance between the strategies. For smaller active sets \((n = 1000)\) the UUS sampling performs significantly worse than the others. WS is in this case slightly worse than QWS and QWS+. For extremely small active sets \((n = 100)\) the difference increases. This clearly demonstrates the benefit of QWS and QWS+. The difference between QWS and QWS+ is in this case on a noise level only. The active set size has impact on training efficiency. While training with \(n = 10000\) takes 3 hours, training with \(n = 100\) requires 1 hour on the same machine.

This experiment demonstrates that the sampling strategy does not play a crucial role in case of relatively large active sets, but a significant difference in performance can be achieved when the active set is relatively small with respect to the pool. QWS and QWS+ significantly outperformed UUS and WS in that case.

### 4.4.2 Profile face detector

This experiment trains a profile face detector and compares it with state-of-the-art approaches. The pool of 41,000 positive and 424 million negative patches was generated from \(S^{0}_{\text{PROF}}, S^{15}_{\text{PROF}}, S^{30}_{\text{PROF}}\) and \(S^{\text{NONE}}\). Two left-profile detectors of length 2000 were trained with QWS+ and UUS strategy. The right profile detectors were obtained by left-to-right flipping of the model.

The resulting ROC curves are displayed in the left panel of figure 4.6. QWS+ has detection rate by 5% higher than UUS for 100 false positives. On the same level of false positives, our detector performs better than the results in [Levi 04] and [Jones 03] and slightly worse than [Schneiderman 04]. The average number of evaluated weak classifiers is 5.6 for QWS+ and 7.2 for UUS which shows positive influence of weighted...
4.4.3 Specific face detector

Google face search was used to collect 650 images returned for query “Borat”\(^2\). This set contained 350 true positives (images depicting Borat), and 300 false positives (images containing other faces or background). This set was randomly split into training set and testing set. Using the training set, a specific face detector was trained. The resulting PRC is presented in the right panel of figure 4.6. Notice that also in this task the performance of QWS+ is superior to UUS. The specific face detector successfully finds approximately 80% of faces of Borat with 90% precision.

4.5 Conclusions

In this chapter, we developed an algorithm for training of object detectors from large labeled data sets. In particular we focused on boosting and bootstrapping and developed a tight fusion of these two approaches. One of the novelties is the formulation of bootstrapping as a weighted sampling strategy and formulation of commonly used strategies within this framework.

Two improvements of standard weighted sampling (WS) are introduced. We designed quasi-random weighted sampling (QWS) which reduces the variance of error estimate

\(^2\)A fictional character portrayed by a British comedian Sasha Barron Cohen.
but does not guarantee unique samples. Next, we introduced a new strategy based on quasi-random weighted sampling + trimming (QWS+) which fixes this drawback. We provided the theoretical proof of the variance minimization of QWS+.

The performance of the learning method was demonstrated on the task of generic face detection (frontal and profile). Various characteristics of the learning process were improved: reduction of the required active set; better precision/recall curves and speed of the resulting classifier. The developed face detector operates in real-time on QVGA images and therefore is applicable to tracking scenarios. This detector will be later used in chapter 6 when our long-term tracking system will be applied to faces.

We trained a specific face detector from manually checked images returned by Google face search. This shows that training a specific face detector is feasible, however, the need of manual preparation of the training set is not practical for long-term tracking scenarios. In long-term tracking, the training of the specific detector has to be done online and from unlabeled data.
Chapter 4. Detection: supervised bootstrap
Chapter 5

Learning: unsupervised bootstrap

This chapter investigates long-term tracking from the perspective of machine learning, the third component of our system. In particular, we study how to improve an offline trained object detector during tracking. Our goal is to design a learning approach that would be suitable for such a scenario as well as general enough to be applicable to other learning problems.

The chapter is structured as follows. Section 5.1 introduces the problem as online learning from labeled and unlabeled data. Section 5.2 develops a novel learning paradigm that we call P-N learning. Section 5.3 applies the P-N learning to training an object detector from a single example and a video stream. The proposed algorithm is evaluated on a number of challenging sequences. The chapter is concluded in section 5.4.

5.1 Introduction

Consider an object detector and a video stream. The object detector is trained offline from a limited number of labeled examples and the video stream is unconstrained. The goal of this chapter is to improve the offline detector by online processing of the video stream. At every time instance, we run the detector on the current frame, estimate its errors and update the detector so that it does not make such errors in the future. We refer to this process as unsupervised bootstrap.

The ability to improve an object detector by online processing of an unlabeled video stream has a large number of applications. Our motivation is long-term tracking where the online learned detector can be used to re-initialize a frame-to-frame tracker after its failure. This aspect will be studied in chapter 6.

There are several challenges that have to be tackled in order to enable the unsupervised bootstrap: (i) large variability – the learning must deal with arbitrarily complex video
Chapter 5. Learning: unsupervised bootstrap

streams where the object significantly changes appearance, moves in and out of the camera view, the stream contains background clutter, fast camera motions or motion blur; (ii) real-time performance – every learning step has to be done immediately after accepting a new frame; (iii) robustness – the learning should never degrade the classifier, if the video stream does not contain relevant information, the detector performance should not degrade.

To tackle all these challenges, we rely on the richness of information contained in the video. Consider for instance a single patch denoting the object location in a single frame. This patch defines not only the single appearance of the object, but also the surrounding of the patch defines what the object is not. When tracking the patch, one can discover different appearances of the same object as well as more appearances of the background. All of this from a single example. This is in contrast to standard machine learning approaches, where the training examples are considered to be independent [Blum 98]. This opens interesting questions how to exploit this rich information in learning.

To exploit the information in the video, we propose a new learning paradigm called P-N learning. The detector is evaluated on every frame of the video stream with the aim to find misclassified examples. These misclassified examples are estimated by two types of complementary ”experts”: (i) P-expert – an expert on positive examples, is estimates when the object detector missed the object, and (ii) N-expert – an expert on negative examples, is able to estimate when a detector made false alarm. The estimated errors augment a training set of the detector, and the detector is retrained in supervised manner to avoid these errors in the future. As any other process, also the P-N experts are making errors themself. However, if the probability of expert error is within certain limits (which will be analytically quantified), the errors are mutually compensated which leads to stable learning.

5.2 P-N learning

This section formalizes the P-N learning without considering any specific application. We assume a set of labeled and unlabeled examples and our goal is to train a binary classifier. The section is split into three parts. Subsection 5.2.1 formalizes the P-N learning and shows its relationship to supervised bootstrapping. Subsection 5.2.2 analyzes the stability of the P-N learning. The conditions that guarantee improvement of the classifier are inferred. Finally, subsection 5.2.3 performs several experiments that validate the proposed theory.
5.2. P-N learning

5.2.1 Formalization

The P-N learning is formalized as semi-supervised learning [Chapelle 06]. Let \( x \) be an example from a feature-space \( \mathcal{X} \) and \( y \) be a label from a space of labels \( \mathcal{Y} = \{-1, 1\} \). A set of examples \( \mathcal{X} \) will be called an unlabeled set. A pair \((X, Y)\) will be called a labeled set, where \( Y \) is a set of labels. The input to the P-N learning is a labeled set \((X_l, Y_l)\) and an unlabeled set \( X_u \), where \( l \ll u \). The task of P-N learning is to learn a classifier \( f : \mathcal{X} \rightarrow \mathcal{Y} \) from labeled set \((X_l, Y_l)\) and bootstrap its performance by unlabeled set \( X_u \). Classifier \( f \) is a function from a family \( \mathcal{F} \) parameterized by \( \theta \). The family \( \mathcal{F} \) is subject to implementation and is considered fixed in training, the training therefore corresponds to estimation of the parameters \( \theta \).

The P-N learning consists of four blocks: (i) classifier to be learned, (ii) training set – a collection of labeled training examples, (iii) supervised training – a method that trains a classifier from training set, and (iv) P-N experts – functions that estimate errors of the classifier and augment the training set with labeled examples. See figure 5.1 for illustration.

The training process is initialized by inserting the labeled examples \((X_l, Y_l)\) to the training set. The training set is then passed to supervised learning which trains a classifier, i.e. estimates the initial parameters \( \theta^0 \). The learning process then proceeds iteratively. In iteration \( k \), the classifier trained in iteration \( k - 1 \) classifies the entire unlabeled set, \( y_u^k = f(x_u | \theta^{k-1}) \) for all \( x_u \in X_u \). The classification is analyzed by the P-N experts that estimate, which example have been classified incorrectly. These examples are added with changed labels to the training set. The iteration finishes by retraining the classifier, i.e. estimation of \( \theta^k \). The process iterates until convergence or other stopping criterion.

Idea. The crucial element of P-N learning is the estimation of the classifier errors. The key idea is to treat the estimation of false positives independent from estimation
Chapter 5. Learning: unsupervised bootstrap

of false negatives. For this reason, the unlabeled set is split into two parts based on the current classification and each part is analyzed by an independent expert. 

**P-expert** analyzes examples classified as negative, estimates false negatives and adds them to training set with positive label. In iteration \( k \), P-expert outputs \( n^+(k) \) positive examples. **N-expert** analyzes examples classified as positive, estimates false positives and adds them with negative label to the training set. In iteration \( k \), the N-expert outputs \( n^-(k) \) negative examples. The P-expert influences the classifier in positive (growing) sense and increases the classifier generality. The N-expert influences the classifier in negative (pruning) sense and increases the classifier discriminability. These two forces are working in parallel and independently from each other.

**Relation to supervised bootstrap.** To put the P-N learning into context, let us consider that the labels of set \( X_u \) are known. Under this assumption it is straightforward to design P-N experts that identify misclassified examples and add them to the training set with correct labels. Such a strategy corresponds to supervised bootstrap as discussed in chapter 4. A classifier trained using supervised bootstrap focuses on the decision boundary and often outperforms a classifier trained on randomly sampled training set [Sung 98]. The same idea of focusing on the decision boundary underpins the P-N learning with the difference that the labels of the set \( X_u \) are unknown. P-N learning can be therefore viewed as a generalization of standard bootstrap to unlabeled case where labels are not given but rather estimated using the P-N experts. As any other process, also the P-N experts make errors, and estimate the labels incorrectly. Such errors then propagate through the training, which will be now theoretically analyzed.

### 5.2.2 Stability

This section analyses the impact of the P-N learning on the classifier performance. For the purpose of the analysis, let us consider that the ground truth labels of \( X_u \) are known and therefore it is possible to measure the errors made by the classifier. Next, consider a classifier that initially classifies the unlabeled set at random and then corrects its classification according to the output of the P-N experts. The performance of such a classifier is characterized by a number of false positives \( \alpha(k) \) and a number of false negatives \( \beta(k) \), where \( k \) indicates the iteration of training. The goal of the P-N learning is to reduce these errors to zero.

In iteration \( k \), the P-expert outputs \( n^+_c(k) \) positive examples which are correct (positive based on ground truth), and \( n^+_f(k) \) positive examples which are false (negative based on ground truth), which forces the classifier to change \( n^+(k) = n^+_c(k) + n^+_f(k) \) negatively classified examples to positive. Similarly, the N-experts outputs \( n^-_c(k) \) correct negative examples and \( n^-_f(k) \) false negative examples, which forces the classifier to change \( n^-(k) = n^-_c(k) + n^-_f(k) \) examples classified as positive to negative. The number
of false positive and false negative errors of the classifier in the next iteration thus becomes:

\[
\alpha(k+1) = \alpha(k) - n_c^-(k) + n_f^+(k) \tag{5.1a}
\]
\[
\beta(k+1) = \beta(k) - n_c^+(k) + n_f^-(k). \tag{5.1b}
\]

Equation \ref{5.1a} shows that false positives \(\alpha(k)\) decrease if \(n_c^-(k) > n_f^+(k)\), i.e. number of examples that were correctly relabeled to negative is higher than the number of examples that were incorrectly relabeled to positive. Similarly, the false negatives \(\beta(k)\) decrease if \(n_c^+(k) > n_f^-(k)\).

**Quality measures.** In order to analyze the convergence of the P-N learning, a model needs to be defined that relates the quality of the P-N experts to the absolute number of positive and negative examples output in each iteration. The quality of the P-N experts is characterized by four quality measures:

- **P-precision** – reliability of the positive labels, i.e. the number of correct positive examples divided by the number of all positive examples output by the P-expert, 
  \(P^+ = n_c^+/(n_c^+ + n_f^+)\).

- **P-recall** – percentage of identiﬁes false negative errors, i.e. the number of correct positive examples divided by the number of false negatives made by the classifier, 
  \(R^+ = n_c^-/\beta\).

- **N-precision** – reliability of negative labels, i.e. the number of correct negative examples divided by the number positive examples output by the N-expert, 
  \(P^- = n_c^-/(n_c^- + n_f^-)\).

- **N-recall** – percentage of recognized false positive errors, i.e. the number of correct negative examples divided by the number of all false positives made by the classifier, 
  \(R^- = n_c^-/\alpha\).

Given these quality measures, the number of correct and false examples output by P-N experts at iteration \(k\) have the form:

\[
n_c^+(k) = R^+ \beta(k), \quad n_f^+(k) = \frac{(1 - P^+)}{P^+} R^+ \beta(k) \tag{5.2a}
\]
\[
n_c^-(k) = R^- \alpha(k), \quad n_f^-(k) = \frac{(1 - P^-)}{P^-} R^- \alpha(k). \tag{5.2b}
\]

By combining the equation \ref{5.1a}, \ref{5.1b}, \ref{5.2a} and \ref{5.2b} we obtain:

\[
\alpha(k+1) = (1 - R^-) \alpha(k) + \frac{(1 - P^+)}{P^+} R^+ \beta(k) \tag{5.3a}
\]
\[
\beta(k+1) = \frac{(1 - P^-)}{P^-} R^- \alpha(k) + (1 - R^+) \beta(k). \tag{5.3b}
\]
Chapter 5. Learning: unsupervised bootstrap

Figure 5.2: The evolution of errors of the classifier depends on the quality of the P-N experts, which is defined in terms of eigenvalues of matrix $M$. The errors converge to zero (LEFT), are at the edge of stability (MIDDLE) or are growing (RIGHT).

After defining the state vector $\vec{x}(k) = [\alpha(k) \beta(k)]^T$ and a $2 \times 2$ matrix $M$ as

$$M = \begin{bmatrix} 1 - R^- & \frac{(1-P^-)R^+}{P^-} \\ \frac{(1-P^-)R^-}{P^-} & 1 - R^+ \end{bmatrix}$$

(5.4)

it is possible to rewrite the equations as

$$\vec{x}(k+1) = M\vec{x}(k).$$

This is a recursive equation that correspond to a discrete dynamical system. The system models the error propagation during training, i.e. from one iteration of P-N learning to another. Our goal is to show, under which conditions the error in the system drops.

Based on the stability criteria from control theory [Zhou 96, Ogata 09], the state vector $\vec{x}$ converges to zero if both eigenvalues $\lambda_1, \lambda_2$ of the transition matrix $M$ are smaller than one. Note that the matrix $M$ is a function of the expert quality measures. Therefore, if the quality measures are known, it is possible to check whether the error during training converges to zero or not. Experts which corresponding matrix $M$ has both eigenvalues smaller than one will be called error-canceling. Figure 5.2 illustrates the evolution of error of the classifier when the first eigenvalue is zero and the second eigenvalue attains values (i) $\lambda_2 < 1$, (ii) $\lambda_2 = 1$, and (iii) $\lambda_2 > 1$.

The pragmatic reason for developing the P-N learning theory was the observation, that it is relatively simple to design a large number of experts that correct specific errors made by the classifier. The combined influence of the experts was, however not understood. P-N learning gives us guidelines how to combine a number of weak experts so that the overall learning is stable. Interestingly, P-N learning does not put constraints on the quality of individual experts. Even experts with low precision might be used as long as the matrix $M$ has eigenvalues smaller than one, it is therefore possible to use various (even weak) information sources.
5.2.3 Experiments

In this experiment, a classifier is trained on a video sequence using simulated P-N experts. Our goal is to analyze the learning performance as a function of the expert quality measures.

Experiment setup. The analysis was performed on sequence CAR (see Appendix B). In the first frame, a classifier was trained using affine warps of the initial patch and background patches from the surrounding of the object. Details about the classifier and the training are described in chapter 7. Next, a single run over the sequence was performed. In every frame, the classifier was evaluated, the simulated experts identified errors and the classifier was updated. After every update, the classifier was tested on the entire sequence to measure its performance using f-measure. See sub-section 5.3.3 for definition of the measure. The performance was then drawn as a function of the number of processed frames and the quality of the P-N experts.

The P-N experts are characterized by four quality measures, $P^+, R^+, P^-, R^-$. To reduce this 4D space of parameters, we analyze the learning at equal error rate. The parameters are set to $P^+ = R^+ = P^- = R^- = 1 - \epsilon$, where $\epsilon$ represents error of the expert. The transition matrix then becomes $M = \epsilon I$, where $I$ is a 2x2 matrix with all elements equal to 1. The eigenvalues of this matrix are $\lambda_1 = 0, \lambda_2 = 2\epsilon$. Therefore the P-N learning should be improving the performance if $\epsilon < 0.5$.

In this experiment, the error is varied in the range $\epsilon = 0:0.9$. The experts were simulated as follows. In frame $k$ the classifier generates $\beta(k)$ false negatives. P-experts relabel $n^+_c(k) = (1 - \epsilon) \beta(k)$ of them to positive which guarantees $R^+ = 1 - \epsilon$. In order to satisfy the requirement precision $P^+ = 1 - \epsilon$, the P-expert relabels additional $n^+_f(k) = \epsilon \beta(k)$ background samples to positive. Therefore the total number of examples relabeled to positive in iteration $k$ is $n^+ = n^+_c(k) + n^+_f(k) = \beta(k)$. The N-experts were generated analogically.

The performance of the detector as a function of number of processed frames is depicted in figure 5.3. Notice that if $\epsilon \leq 0.5$ the performance of the detector increases with more processed frames. In general, $\epsilon = 0.5$ will give unstable results although in this sequence it leads to improvements. Increasing the noise-level further leads to sudden degradation of the classifier. These results are in line with the theory.

The error-less P-N learning ($\epsilon = 0$) is analyzed in more detail. In this case all classifier errors are identified and no miss-labeled examples are added to the training set. Three different classifiers were trained using: (i) P-experts, (ii) N-experts, and (iii) P-N experts. The classifier performance was measured using precision, recall and f-measure and the results are shown in figure 5.4. Precision (LEFT) is decreased by P-experts since only positive examples are added to the training set, these cause the classifier to be too generative. Recall (MIDDLE) is decreased by N-experts since these add only negative examples and cause the classifier to be too discriminative. F-measure (RIGHT)
Figure 5.3: Performance of a detector as a function of the number of processed frames. The detectors were trained by synthetic P-N experts with guaranteed level of error. The classifier is improved up to error 50% (BLACK), higher error degrades it (RED).

Figure 5.4: Performance of detectors trained by error-less P-expert, N-expert and P-N expert measured by precision (LEFT), recall (MIDDLE) and f-measure (RIGHT).
5.3. Learning an object detector from a video sequence

This section applies the P-N learning to bootstrapping a scanning window-based object detector from a video sequence. The section is split into three parts. Subsection 5.3.1 specifies the learning problem using the P-N learning terminology. Subsection 5.3.2 develops appropriate P-N experts. Finally, subsection 5.3.3 performs quantitative evaluation on a number of challenging sequences. The learning approach will be described without considering implementation details, these are given in chapter 6.

5.3.1 Problem specification

The input to the learning is a single frame where the object of interest is denoted by an initial bounding box and a video sequence. The learning is performed by sequential processing of the video sequence in a frame-by-frame fashion. One iteration of the P-N learning corresponds to processing of one frame of the sequence. The output of the learning is a binary classifier that separates appearances of the object from appearances of the background that appeared in the video sequence. Figure 5.5 illustrates the scenario.

The training examples (both labeled and unlabeled) correspond to image patches. These image patches are sampled at locations and scales determined by a scanning grid on which the detector operates. The labeled data $X_l$ are extracted from the first frame. Patches that are overlapping with the initial bounding box are positive, patches that are non-overlapping are negative. The unlabeled data $X_u$ are extracted from the remaining video sequence.
The learning is initialized in the first frame by supervised training of so called Initial detector. The learning then proceeds by sequential processing of the video sequence. In every iteration, the P-N learning performs the following steps: (i) evaluation of the detector on the current frame, (ii) estimation of the detector errors using the P-N experts, (iii) update of the detector by labeled examples output by the experts. The detector obtained at the end of the sequence is called the Final detector. We perform the learning in one pass through the video sequence which is analogical to processing live video stream.

5.3.2 P-N experts

This section presents our P-N experts developed for bootstrapping a scanning window-based detector from a video sequence. Consider that our goal is to train a detector of a motorbike. In every iteration of the P-N learning, the detector is evaluated on the current frame. One possible output of the detector is depicted in figure 5.6 (a). Note that the detector performed two false positives and one false negative errors. The goal of the P-N experts is to identify these errors. In particular, P-experts should identify false negatives and N-experts should identify false positives as shown in figure 5.6 (b).

To introduce the P-N experts, consider figure 5.7 (a) that shows three frames of a video sequence overlaid with a scanning grid. Every bounding box in the grid defines an image patch, which label is represented as a colored dot (b,c). The detector considers every patch to be independent. Therefore, there are \(2^N\) possible label combinations in a single frame, where \(N\) is the number of bounding boxes in the grid. Figure 5.7 (b) shows one such labeling. The labeling indicates, that the object appears on several location in a single frame and there is no temporal continuity in the motion. In natural videos is such labeling not credible and therefore it can be inferred that the detector made a mistake at least on several location. On the other hand, if the detector outputs
5.3. Learning an object detector from a video sequence

Figure 5.7: Illustration of a scanning grid applied to three consecutive frames (a) and corresponding spatio-temporal volume of labels with unacceptable (b) and acceptable (c) labeling. Red dots correspond to positive labels.

classification depicted in (c) the labeling is acceptable since the object appears at one location in a single frame and these locations constitute a smooth trajectory in time.

As we have just shown, it is fairly easy to estimate unlikely behavior of the detector when observing the detector responses in the context of a video volume. We exploited our prior knowledge about motion of an object which casts constraints on the labeling of the video volume. In other words, every single patch influences labels of other patches. Such a property will be called structure and the data that have this property are structured. This is in contrast to majority of existing learning algorithms in semi-supervised learning, which assume that the unlabeled examples are independent [Blum 98].

The key idea of the P-N experts is to exploit the structure in data to estimate the detector errors. Our approach to model the structure is based on simple rules such as: (i) overlapping patches have the same label, (ii) patches within a single image can have at most one positive label, (iii) patches that are connected by a trajectory have the same label, etc. Based on these rules the P-N experts are built.

The P-expert exploits the temporal structure in the video volume and assumes that the object moves on a smooth trajectory. The P-expert remembers the location of the object in the previous frame and estimates the object location in current frame using a frame-to-frame tracker. If the detector labeled the current location as negative (i.e. made a false negative error), the P-expert generates a positive example from the current location and performs update of the detector.

The N-expert exploits the spatial structure in the video volume and assumes that the object can appear at a single location in a single frame only. The N-expert analyzes all responses of the detector in the current frame and the response produced by the tracker and selects the one that is the most confident. Patches that are not overlapping with the maximally confident patch are labeled as negative and update the detector. The maximally confident patch re-initializes the location of the tracker.
Figure 5.8: Illustration of the P-expert and the N-expert and their error compensation.

**Error compensation.** The figure 5.8 depicts a sequence of tree images, the object to be learned is a car within the yellow bounding box. The car is tracked from frame to frame by a tracker. The tracker represents the P-expert that outputs positive training examples. Notice that due to occlusion of the car, the 3rd example is incorrect. N-expert identifies maximally confident patch (denoted by a red star) and labels all other patches as negative. Notice that the N-expert is discriminating against another car, and in addition corrected the error made by the P-expert in the 3rd frame. This shows that if the experts use an independent source of information (i.e. temporal and spatial structure in data) they have potential to correct their own errors.
5.3.3 Experiments

This experiment analysis the performance of the P-N learning on 10 sequences shown in figure 5.9. The performance is evaluated using precision $P$, recall $R$ and f-measure $F$. $P$ is the number of true positives divided by number of all detections, $R$ is the number true positives divided by the number of object occurrences that should have been detected. $F$ combines these two measures as $F = 2PR/(P + R)$. Next, the quality of the P-N experts is measured using $P^+, R^+, P^- and R^-$ averaged over all iterations. A detection is considered as true positive if its overlap with ground truth bounding box is larger than 50%. The overlap is defined as the ratio between intersection and union of two bounding boxes.

Table 5.1 (3rd column) shows the resulting scores of the Initial detector. This detector has high precision for most of the sequences with exception of sequence 9 and 10. Sequence 9 is very long (9928 frames) there is a significant background clutter and objects similar to the target (cars). Recall of the Initial detector is low for the majority of sequences except for sequence 5 where the recall is 73%. This indicates that in this sequence the appearance of the object does not vary significantly. The scores of the Final detector are displayed in the 4th column of the table 5.1. The recall of the detector was significantly increased with little drop of precision. In sequence 9, even the precision was increased from 36% to 90%, which shows that the false positives of the Initial detector were estimated by N-expert and corrected. The most significant increase of the performance is for sequences 7-10 which are the most challenging of the whole set. The Initial detector fails here but for the Final detector the f-measure is in the range of 25-83%. This demonstrates the benefit of P-N learning.

The last three columns of Table 5.1 report the quality of P-N experts. Both experts have precision higher than 60% except for sequence 10 which has P-precision just 31%. Recall of the experts is in the range of 2-78%. The last column shows the corresponding eigenvalues of matrix $M$. Notice that all eigenvalues are smaller than one. This demonstrates that the proposed experts work across different scenarios and lead to improvement of the Initial detector. The larger these eigenvalues are, the less the P-N learning improves the performance. For example in sequence 10 one eigenvalue is 0.99 which reflects poor performance of the P-N constraints. The target of this sequence is an animal, which changes its pose throughout the sequence. The Median-Flow tracker is not very reliable in this scenario, but still P-N learning exploits the information provided by the tracker and improves the detector.
Chapter 5. Learning: unsupervised bootstrap

Figure 5.9: Sample images from evaluation sequences with objects marked. See Appendix B for more details.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Frames</th>
<th>Initial detector precision / recall / f-measure</th>
<th>Final detector precision / recall / f-measure</th>
<th>P-expert</th>
<th>N-expert</th>
<th>Eigenvalues $\lambda_1, \lambda_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. David</td>
<td>761</td>
<td>1.00 / 0.01 / 0.02</td>
<td>1.00 / 0.32 / 0.49</td>
<td>1.00 / 0.08</td>
<td>0.99 / 0.17</td>
<td>0.92 / 0.83</td>
</tr>
<tr>
<td>2. Jumping</td>
<td>313</td>
<td>1.00 / 0.01 / 0.02</td>
<td>0.99 / 0.88 / 0.93</td>
<td>0.86 / 0.24</td>
<td>0.98 / 0.30</td>
<td>0.70 / 0.77</td>
</tr>
<tr>
<td>3. Pedestrian 1</td>
<td>140</td>
<td>1.00 / 0.06 / 0.12</td>
<td>1.00 / 0.12 / 0.22</td>
<td>0.81 / 0.04</td>
<td>1.00 / 0.04</td>
<td>0.96 / 0.96</td>
</tr>
<tr>
<td>4. Pedestrian 2</td>
<td>338</td>
<td>1.00 / 0.02 / 0.03</td>
<td>1.00 / 0.34 / 0.51</td>
<td>1.00 / 0.25</td>
<td>1.00 / 0.24</td>
<td>0.76 / 0.75</td>
</tr>
<tr>
<td>5. Pedestrian 3</td>
<td>184</td>
<td>1.00 / 0.73 / 0.84</td>
<td>0.97 / 0.93 / 0.95</td>
<td>0.98 / 0.78</td>
<td>0.98 / 0.68</td>
<td>0.32 / 0.22</td>
</tr>
<tr>
<td>6. Car</td>
<td>945</td>
<td>1.00 / 0.04 / 0.08</td>
<td>0.99 / 0.82 / 0.90</td>
<td>1.00 / 0.52</td>
<td>1.00 / 0.46</td>
<td>0.48 / 0.54</td>
</tr>
<tr>
<td>7. Motocross</td>
<td>2665</td>
<td>1.00 / 0.00 / 0.00</td>
<td>0.92 / 0.32 / 0.47</td>
<td>0.96 / 0.19</td>
<td>0.84 / 0.08</td>
<td>0.92 / 0.81</td>
</tr>
<tr>
<td>8. Volkswagen</td>
<td>8576</td>
<td>1.00 / 0.00 / 0.00</td>
<td>0.92 / 0.75 / 0.83</td>
<td>0.70 / 0.23</td>
<td>0.99 / 0.09</td>
<td>0.91 / 0.77</td>
</tr>
<tr>
<td>9. Car Chase</td>
<td>9928</td>
<td>0.36 / 0.00 / 0.00</td>
<td>0.90 / 0.42 / 0.57</td>
<td>0.64 / 0.19</td>
<td>0.95 / 0.22</td>
<td>0.76 / 0.83</td>
</tr>
<tr>
<td>10. Panda</td>
<td>3000</td>
<td>0.79 / 0.01 / 0.01</td>
<td>0.51 / 0.16 / 0.25</td>
<td>0.31 / 0.02</td>
<td>0.96 / 0.19</td>
<td>0.81 / 0.99</td>
</tr>
</tbody>
</table>

Table 5.1: Performance analysis of P-N learning. The Initial detector is trained on the first frame only. The Final detector is obtained by P-N learning after one pass through the sequence. The last three columns display internal statistics of the training process.

5.4 Conclusions

P-N learning, a novel approach for processing of labeled and unlabeled examples, has been proposed. The underlying assumption of the learning process is that the unlabeled data are structured. The structure of the data is exploited by positive and negative experts that restrict the labeling of the unlabeled data. These experts provide a feedback about the performance of the classifier which is iteratively improved in a bootstrapping fashion. We have formulated conditions under which the P-N learning guarantees improvement of the classifier. The conditions have been validated on synthetic and real data. The P-N learning has been applied to the problem of learning of an object detector from a single example and an unlabeled video sequence. We have proposed experts that exploit the spatio-temporal properties of a video and demonstrated that they lead to significant improvement of the detector for variety of objects and conditions.
Chapter 6

Tracking-Learning-Detection (TLD)

This chapter describes and extensively evaluates the long-term tracking system developed in the thesis. We build on the algorithms developed in earlier chapters, namely, adaptive object tracker (chapter 3), real-time object class detector (chapter 4), and a learning method (chapter 5). These components are assembled in a system, which we refer to as TLD.

The chapter is structured as follows. Section 6.1 introduces the main idea behind simultaneous tracking, learning and detection. Section 6.2 formalizes the TLD framework. Section 6.3 describes our implementation. Section 6.4 performs a set of comparative experiments with state-of-the-art approaches. Section 6.5 applies the TLD framework to long-term tracking of human faces. Finally, section 6.6 performs a set of qualitative experiments and discusses the pros and cons.

6.1 Introduction

Consider a video stream and a single bounding box defining the object of interest in one frame. The goal of long-term tracking is to track the object “forever”: every time, the object appears in the camera view, a long-term tracker should draw a bounding box around it. As we have discussed in chapter 2, the long-term tracking problem is closely related to frame-to-frame tracking and tracking-by-detection.

Frame-to-frame tracking assumes that the object moves on a smooth trajectory. Trackers are able adapt to changes of the object appearance, however, they typically fail if the object gets fully occluded or disappears. In contrast, tracking-by-detection assumes that the object model is known in advance. The detectors never fail due to occlusion or disappearance, however, the object model is fixed which means that unexpected appearances cannot be detected (false positives) and cluttered background may generate false detections (false positives). Obviously, both of these assumptions are too restrictive for the long-term tracking problem.
Chapter 6. Tracking-Learning-Detection (TLD)

Figure 6.1: Illustration of a tracker (a), detector (b), and a tractor (c). Dotted line shows ground truth trajectory, gray bar represents full occlusion, thick line is the trajectory of a tracker, red dots are responses of a detector.

In a number of tracking problems, the objects occasionally reappear with previously observed appearance. This has been already used to reduce drift of frame-to-frame tracking [Dowson 05, Rahimi 08], closing loops in SLAM [Newman 06] or tracking of people [Ramanan 05]. The idea of this chapter is to build online an object detector that represents all appearances observed so far. This detector shall run in parallel with an object tracker and correct or re-initialize it when necessary. Tracking is then understood not only as a way to determine the location of the object, but also as a way to provide training examples for the online trained object detector. Any algorithm, that combines an object tracker with an online learned detector will be referred to as tractor. See figure 6.1 for comparison of a tracker, a detector and a tractor.

The idea to augment any tracking algorithm using an online trained detector is general and can be applied to a wide range of tracking and detection algorithms. Their integration is, however, not straightforward. In the following section, we describe our way of integration, which we call the TLD framework.

6.2 The TLD framework

The TLD framework is designed for long-term tracking of unknown objects in unconstrained environments. The framework also enables tracking of objects the class of which is known in advance. This section describes the TLD framework on the highest level. The block diagram is shown in figure 6.2.

6.2.1 Components

The framework consists of four components: tracker, learning, detector and integrator. These components have the following characteristics:
1. **Tracker** is an exploratory and error-prone component of TLD. The tracker estimates frame-to-frame object motion and is adaptive in order to handle appearance and illumination changes. It is not assumed that the tracker is correct all the time. Automatic detection of tracking failures is an important, but not required feature of the tracker.

2. **Learning** is an analyzing component that maintains an *object model*. The object model represents the object of interest as a collection of examples. Learning constantly analyzes the output of the tracker and the detector, estimates errors performed by the detector and updates the object model to avoid these errors in the future.

3. **Detector** is a stabilizing component of the system that detect the appearances represented in the constantly updated object model. The detector is either build entirely online, or the online information is integrated with prior information about the object class. The detector must enable efficient incremental update.

4. **Integrator** is a component that merges the hypotheses from the detector and the tracker and outputs the *final hypothesis* about the object state.

The TLD framework distinguishes two modes: (i) initialization, and (ii) run-time.

### 6.2.2 Initialization

The initialization requires the first frame and the corresponding object state indicated by a bounding box. In addition, the framework may accept images depicting the object from multiple views or background images where the object is not present. The following operations are then performed:
• **Initialization of the tracker:** involves setting the initial state of the object and building of the tracker’s model (e.g. extraction of a template or training of a classifier).

• **Initialization of the object model:** involves inserting the given example(s) of the object and examples of the background into the object model.

• **Initialization of the detector:** involves training an object detector to detect the appearances represented in the object model. We shall refer to the resulting detector as the *Initial detector*.

After initialization, the TLD framework is prepared to process the video stream frame-by-frame.

### 6.2.3 Run-time

At each time instance, the framework accepts a video frame and passes it in parallel to the tracker, the detector and the learning component. The tracker estimates the object motion based on its previous state and outputs a single hypothesis. The detector returns a number of hypotheses about the location of the target object. The tracker’s and detector’s hypotheses are passed to the integrator, which merges them into the final state that is then output from the system. Outputs of the tracker, the detector and the integrator are analyzed by the learning block, which estimates errors and updates the detector to avoid these errors in the future.

### 6.3 Implementation of TLD

This section describes our implementation of the TLD framework, which we refer to as **TLD1.0**. We start by defining the object representation in section 6.3.1 and the object model in section 6.3.2. The following sections discuss the individual components of the framework. Section 6.3.3 introduces our detector, which is based on a cascaded classifier and enables integration of an offline learned detector developed in chapter 4. Section 6.3.4 mentions the adaptation of the Median-Flow tracker developed in chapter 3. Section 6.3.5 discusses the integration of the detector and the tracker. Finally, section 6.3.6 discusses the realization of the learning component that is based on the P-N learning developed in chapter 5.

#### 6.3.1 Object representation

**Object state.** At any time instance, the object state is defined by a bounding box or indicates that the object is not visible. The bounding box has a fixed aspect ratio.
(given by the initial bounding box) and is parameterized by its location and scale. Other parameters such as in-plane rotation are not considered. Spatial similarity of two bounding boxes is measured using overlap, which is defined as a ratio between intersection and union of the two bounding boxes.

**Object appearance.** A single instance of the object’s appearance is represented by an image patch \( p \). The patch is sampled within the object bounding box and then is re-sampled to a normalized resolution (typically 15x15 pixels) regardless of the aspect ratio. The similarity between two patches \( p_i, p_j \) is defined as

\[
S(p_i, p_j) = 0.5(\text{NCC}(p_i, p_j) + 1),
\]

where NCC is a Normalized Correlation Coefficient. The similarity ranges from 0 to 1.

**Object trajectory.** A sequence of object states defines a trajectory of an object in video volume as well as the corresponding trajectory in the appearance space. Note that the trajectory is fragmented as the object may not be always visible. See figure 6.3 for illustration.

### 6.3.2 The object model

The object model \( M \) is a dynamic data structure that represents the object appearances and its surrounding observed so far. It is a collection of positive and negative patches, \( M = \{p_1^+, p_2^+, \ldots, p_m^+, p_1^-, p_2^-, \ldots, p_n^-\} \), where \( p^+ \) and \( p^- \) represent the object and background patches, respectively. Positive patches are ordered according to the time when the patch was added to the collection. \( p_1^+ \) is the first positive patch added to the collection, \( p_m^+ \) is the last positive patch added the collection.
We define several similarity measures which are used throughout the system to indicate how much an arbitrary patch \( p \) resembles the object appearances represented in the model \( M \):

1. Similarity with the positive nearest neighbor, \( S^+(p, M) = \max_{p_i^+ \in M} S(p, p_i^+) \).

2. Similarity with the negative nearest neighbor, \( S^-(p, M) = \max_{p_i^- \in M} S(p, p_i^-) \).

3. Relative similarity, \( S^r = \frac{S^+}{S^+ + S^-} \). Relative similarity ranges from 0 to 1, higher values mean more confident that the patch depicts the object.

4. Conservative similarity, \( S^c = \frac{S^+}{S^+_\theta + S^-} \), similar as the relative similarity, however considers only first \( \theta \% \) of positive example in the object model. For \( \theta = 1 \) is the conservative similarity identical to relative similarity. The lower \( \theta \) the more conservative the measure becomes. For any setting \( S^c \leq S^r \). In our experiments, we use \( \theta = 0.5 \).

The Relative similarity is used to define the Nearest Neighbor (NN) classifier: a patch \( p \) is classified as positive if \( S^r(p, M) > \theta \). Parameter \( \theta \) enables tuning the nearest neighbor classifier either towards precision or recall. The classification margin is defined as \( S^r(p, M) - \theta \). The margin indicates the confidence of the classification.

**Model update.** To integrate a new labeled patch to the object model, the following strategy is used. The patch is first classified by the NN classifier and added to the collection only if the classification is incorrect. This strategy leads to a significant reduction of accepted patches [Aha 91] at the cost of coarser representation of the decision boundary. Therefore we alter this strategy by adding also patches where the classification margin is smaller than \( \lambda \). With larger \( \lambda \), the model accepts more patches which leads to better representation of the decision boundary. In our experiments we use \( \lambda = 0.1 \) which compromises the accuracy of the representation and the speed by which is the collection growing. Exact setting of this parameter is not critical.

### 6.3.3 The object detector

The object detector is an algorithm that efficiently localizes the appearances represented in the object model. The detector scans the input image by a scanning-window and for each patch decides about presence or absence of the object.

**Scanning-window grid.** We generate all possible scales and shifts of an initial bounding box with the following parameters: scales step 1.2, horizontal step 10% of width, vertical step 10% of height, minimal bounding box size 20 pixels. This setting produces around 50 000 bounding boxes for a QVGA image (240x320), the exact number...
6.3. Implementation of TLD

Cascaded classifier. As the number of bounding boxes to be evaluated is large, the classification of individual patches has to be efficient. A straightforward approach of directly evaluating the NN classifier is problematic as it involves the search for two nearest neighbors (positive and negative) in a high dimensional feature space. To speed up the process the classifier is structured into three stages: (i) patch variance, (ii) ensemble classifier, and (iii) nearest neighbor. Each stage either rejects the patch in question or passes it to the next stage. This cascaded architecture is common in face detection [Viola 01] where it enabled real-time performance. Figure 6.4 shows the block diagram of our detector.

Patch variance. Patch variance is the first stage of our detector. This stage rejects all patches, for which the gray-value variance is smaller than a threshold. We set the threshold to 50% of the variance of the initial patch. Our implementation exploits the fact that variance of a patch \( p \) can be expressed as \( \mathbb{E}(p^2) - \mathbb{E}^2(p) \), and that the expected value \( \mathbb{E}(p) \) can be measured in constant time using integral images [Viola 01]. This stage typically rejects more than 50% of non-object patches (sky, street, etc) and is essential for the real-time performance.

Ensemble classifier. Ensemble classifier is the second stage of our detector. The input to the ensemble is an image patch that was not rejected by the first stage. The ensemble consists of \( n \) base classifiers. Each base classifier \( i \) performs a number of pixel comparisons on the patch resulting in a binary code \( x \), which indexes to an array of posteriors \( P_i(y|x) \), where \( y \in \{-1, 1\} \). The posteriors of individual base classifiers are averaged. The patch is classified as positive if the average posterior is larger than 50%. Figure 6.5 shows the block diagram of the ensemble classifier.

Pixel comparisons. Every base classifier is based on a set of pixel comparisons. Similarly as in [Lepetit 06, Ozuysal 07, Calonder 10], the pixel comparisons are generated offline and stay fixed in run-time. The pixel comparisons are used to convert an image patch to a binary code. First, the image is convolved with a Gaussian kernel with standard deviation of 3 pixels to increase the robustness to shift and image noise. Next,
Chapter 6. Tracking-Learning-Detection (TLD)

Figure 6.5: The block diagram of our ensemble classifier.

Figure 6.6: Conversion of a patch to a binary code using a set of pixel comparisons.

Figure 6.7: Pixel comparisons measured within individual base classifiers. The rectangles correspond to a normalized patch. Squares correspond to pixel locations, lines show which pixels are compared.
the predefined set of pixel comparison is stretched to the patch. Each comparison returns 0 or 1 and these measurements are concatenated into a binary code \( x \). Figure 6.6 illustrates the process.

**Generating pixel comparisons.** The vital element of ensemble classifiers is the independence of the base classifiers [Breiman 01]. The independence of the classifiers is in our case enforced by measuring different pixel comparison by each base classifier. First, we discretize the space of pixel locations within a normalized patch and generate all possible horizontal and vertical pixel comparisons. Comparisons of zero length are not considered. Next, we permute the comparisons and split them into the base classifiers. As a result, every classifier is guaranteed to be based on a different set of features and all the features together uniformly cover the entire patch. This is in contrast to other approaches [Lepetit 06, Ozuysal 07, Calonder 10], where every pixel comparison is generated independently of other pixel comparisons. Figure 6.7 shows the pixel comparisons used in our implementation.

**Posterior probabilities.** Every base classifier \( i \) maintains a distribution of posterior probabilities \( P_i(y|x) \). The distribution has \( 2^d \) entries, where \( d \) is the number of pixel comparisons. We use 13 comparison, which gives 8192 possible codes that index to the posterior probability. The probability is estimated as \( P_i(y|x) = \frac{#p}{#p + #n} \), where \( #p \) and \( #n \) correspond to number of positive and negative patches, respectively, that were assigned the same binary code. 10 base classifiers is used in our implementatin. This is mainly motivated by the requirement on the real-time performance. If the speed is not an issue the number of base classifiers can be increased [Breiman 01] in order to increase the performance of the ensemble.

**Initialization and update.** In the initialization stage, all base posterior probabilities are set to zero, i.e. vote for negative class. During run-time the ensemble classifier is updated as follows. The labeled example is classified by the ensemble and if the classification is incorrect, the corresponding \( #p \) and \( #n \) are updated which consequently updates \( P_i(y|x) \).

**Nearest neighbor classifier.** After filtering the patches by the first two stages, the last stage is left with several bounding boxes that are not decided yet (\( \approx 50 \)). The last stage employs the NN classifier based on the online model. A patch is classified as the object if \( S^r > \theta \). Setting of \( \theta \) enables to tune the classifier either towards precision or recall. We use a conservative setting of \( \theta = 0.6 \), which leads to a minimal number of false positive detections. The positively classified patches represent the responses of the object detector.

### 6.3.4 The object tracker

The tracking component of TLD1.0 is based on the Median-Flow tracker developed in chapter 3 which was augmented with failure detection. The Median-Flow tracker
represents the object by a bounding box and estimates its motion between consecutive frames. Internally, the tracker estimates displacements of a number of points within the object’s bounding box, estimates their reliability, and votes with 50% of the most reliable displacements for the motion of the bounding box using median. A grid of $10 \times 10$ points is used. The motion of each individual point is estimated using the pyramidal implementation of Lucas-Kanade tracker [Bouguet 99]. The pyramidal Lucas-Kanade tracker uses 2 levels of the pyramid and represents the points by $10 \times 10$ patches.

**Failure detection.** Let $d_i$ denote the displacement of a single point of the Median-Flow tracker and $d_m$ be the median displacement. A failure of the tracker is declared if Median Absolute Deviation (MAD) is larger than a threshold, $\text{median}(|d_i - d_m|) > 10$ pixels. This heuristic is able to reliably identify most failures caused by fast motion or fast occlusion of the object of interest. In that case, the individual displacement become scattered around the image and the MAD rapidly increases. If the failure is detected, the tracker does not return any bounding box.

### 6.3.5 The integrator

The integrator is a function that combines the responses of the tracker and the detector into a single response output by the system. If neither the tracker nor the detector output a bounding box, the object is declared as not visible. Otherwise the integrator outputs the maximally confident bounding box, measured using the Conservative similarity $S^c$. The integrator is illustrated in figure 6.8.

**Smoothing the trajectory.** The object trajectory obtained by taking the maximally confident bounding box has one disadvantage: the trajectory tends to jitter. This is caused by the detector, which has often multiple responses close to the tracker, which might overrule the non-jittering tracker. Therefore we further modified the integrator...
as follows. If the tracker’s bounding box is defined and the maximally confident detection is in its vicinity (overlap $> 0.8$), the tracker bounding box is averaged with all detections that are in the tracker’s vicinity. If the maximally confident detection is far from the tracker (overlap $< 0.8$), the tracker is re-initialized.

### 6.3.6 The learning component

The task of the learning component is to initialize the object detector in the first frame and update the detector in run-time using the P-expert and the N-expert.

**Initialization**

In the first frame, the learning component trains the Initial detector. The detector is trained using labeled examples that are generated as follows. The positive training examples are synthesized from the initial bounding box. First, we select 10 bounding boxes from the scanning grid that are closest to the initial patch. For each of the bounding box, 20 warped versions are generated. The parameters of the warping are drawn randomly from a uniform distribution of shift $\pm 1\%$, scale change $\pm 1\%$ and in-plane rotation $\pm 10^\circ$. The warped patches are added with Gaussian noise ($\sigma = 5$). The result is 200 synthetic positive patches. Negative patches are collected from the surrounding of the initial patch, no synthetic negative examples are generated. If the application requires fast initialization, the training examples are sub-samples. The labeled training patches are then update the object model as discussed in subsection 6.3.2 and the ensemble classifier as discussed in subsection 6.3.3. After the initialization the Initial detector is ready for run-time and to be updated using the P-expert and the N-expert.

**The P-expert**

The goal of the P-expert is to discover new appearances of the object and thus increase the generality the object detector. Section 5.3 suggested a P-expert that exploits the fact that the object moves on a trajectory. In the TLD system, the object trajectory is generated by a combination of the tracker, the detector and the integrator. This combined process traces a discontinuous trajectory, which is by no means correct all the time as any of the components can fail. The challenge of the P-expert is to estimate reliable parts of the trajectory and use it to generate positive training examples.

To estimate the reliable parts of the trajectory, the P-expert relies on the object model. Consider an object model represented as colored points in a feature space. Positive examples are represented by red dots connected by a directed curve suggesting their order, negative examples are black. Using the conservative similarity $S^c$, one can
define a subspace in the feature space, where $S^c$ is larger than a threshold. We refer to this subspace as the core of the object model.

The P-expert estimates the reliable parts of the trajectory as follows. The trajectory becomes reliable as soon as it enters the core and remain reliable until is re-initialized or the tracker declares its own failure. Any other trajectory is not considered by the P-expert. The reliable trajectory generates positive examples that are then added to the object model. See figure 6.9 for illustration.

In every frame, the P-expert outputs a decision about the reliability of the current location output by the integrator. If the current location is reliable, the P-expert generates a set of positive examples that update the object model and the ensemble classifier. First, we select 10 bounding boxes from the scanning grid that are closest to the initial patch. For each of the bounding box, 10 warped versions are generated. The parameters of the warping are drawn randomly from a uniform distribution of shift $\pm 1\%$, scale change $\pm 1\%$ and in-plane rotation $\pm 5^\circ$. The warped patches are added with Gaussian noise ($\sigma = 5$). This results in 100 synthetic positive examples for the ensemble classifier. For efficiency reasons, we consider only 10 patches for update of the object model.

The N-expert

The N-expert generates negative training examples with the aim to discriminate the detector against background clutter. The key assumption of the N-expert is that the object can occupy at most one location in the image. The N-expert is applied at the same time as P-expert, i.e. if the object location is reliable. In that case, patches that are far from current bounding box (overlap $< 0.2$) are all labeled as negative. For the update of the object detector and the ensemble classifier, we consider only those patches that were not rejected by the first two stages of the cascade.
6.4 Quantitative evaluation

This section reports on a set of quantitative experiments comparing the TLD1.0 with relevant algorithms. The first two experiments evaluate the TLD1.0 on benchmark data sets that are commonly used in the literature. In particular, the experiment in section 6.4.1 extends the results reported in [Santner 10]. The experiment in section 6.4.2 extends the results from [Yu 08]. In both cases, a saturated performance is achieved. Section 6.4.3 therefore introduces a more challenging data set and performs further evaluation.

Every experiment in this section adopts the following evaluation protocol. A tracker is initialized in the first frame of a sequence and tracks the selected object up to the end of the sequence. The produced trajectory is then compared to ground truth. The particular evaluation measure is specified in every experiment. TLD1.0 has been compared with 11 comparable algorithms on 19 benchmark sequences. See Appendix for more details about the algorithms and the sequences.

6.4.1 Comparison 1: CoGD

TLD1.0 was compared with results reported in [Yu 08], which reports on performance of 5 trackers: (i) Incremental Visual Tracking (IVT) [Ross 07], (ii) Online Discriminative Features (ODF) [Collins 05], (iii) Ensemble Tracking (ET) [Avidan 07], (iv) Multiple Instance Learning (MILTrack) [Babenko 09], and (v) Co-trained Generative and Discriminative tracker (CoGD) [Yu 08]. The evaluation was performed on 6 sequences that include full occlusions and disappearance of the object. CoGD [Yu 08] clearly dominated on these sequences as it is capable of re-detection of the object. The performance was accessed using the Number of successfully tracked frames, i.e. the number of frames where overlap with a ground truth bounding box is larger than 50%. Frames where the object was occluded were not counted. For instance, for a sequence of 100 frames where the object is occluded in 20 frames, the best possible score is 80 frames.

TLD1.0 achieved the maximal possible score in all sequences and matched the performance of CoGD [Yu 08]. It was reported in [Yu 08] that CoGD runs at 2 frames per second, and requires several frames (typically 6) for initialization. In contrast, TLD1.0 requires just a single frame and runs at 20 frames per second. Table 6.1 shows the results.

This experiment demonstrates that neither the generative trackers (IVT [Ross 07]), nor the discriminative trackers (ODF [Collins 05], ET [Avidan 07], MILTrack [Babenko 09]) are able to handle long-lasting full occlusions or disappearance of the object. CoGD is evaluated in detail on more challenging data in section 6.4.3.
Table 6.1: The number of successfully tracked frames – TLD1.0 in comparison to results reported in [Yu 08]. Bold numbers indicate the best score; a dash indicates that the result was not reported. TLD1.0 achieved the best possible performance.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Frames</th>
<th>Occlusion</th>
<th>IVT (Ross 07)</th>
<th>ODF (Collins 05)</th>
<th>ET (Avidan 07)</th>
<th>MILTrack (Babenko 09)</th>
<th>CoGD (Yu 08)</th>
<th>TLD1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>761</td>
<td>0</td>
<td>17</td>
<td>-</td>
<td>94</td>
<td>135</td>
<td>759</td>
<td>761</td>
</tr>
<tr>
<td>Jumping</td>
<td>313</td>
<td>0</td>
<td>75</td>
<td>313</td>
<td>44</td>
<td>313</td>
<td>133</td>
<td>313</td>
</tr>
<tr>
<td>Pedestrian 1</td>
<td>140</td>
<td>0</td>
<td>11</td>
<td>6</td>
<td>22</td>
<td>101</td>
<td>136</td>
<td>140</td>
</tr>
<tr>
<td>Pedestrian 2</td>
<td>338</td>
<td>93</td>
<td>33</td>
<td>8</td>
<td>118</td>
<td>37</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Pedestrian 3</td>
<td>184</td>
<td>30</td>
<td>50</td>
<td>5</td>
<td>53</td>
<td>49</td>
<td>154</td>
<td>154</td>
</tr>
<tr>
<td>Car</td>
<td>945</td>
<td>143</td>
<td>163</td>
<td>-</td>
<td>10</td>
<td>45</td>
<td>802</td>
<td>802</td>
</tr>
</tbody>
</table>

Table 6.2: Recall – TLD1.0 in comparison to results reported in [Santner 10]. Bold numbers indicate the best score; a dash indicates that the result was not reported. TLD1.0 scored best in 9/10 sequences.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Frames</th>
<th>OnlineBoost (Grabner 06)</th>
<th>OnlineRF (Saffari 09)</th>
<th>FragTrack (Adam 06)</th>
<th>MILTrack (Babenko 09)</th>
<th>Prost (Santner 10)</th>
<th>TLD1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Girl</td>
<td>452</td>
<td>24.0</td>
<td>-</td>
<td>70.0</td>
<td>70.0</td>
<td>89.0</td>
<td>93.1</td>
</tr>
<tr>
<td>David</td>
<td>502</td>
<td>23.0</td>
<td>-</td>
<td>47.0</td>
<td>70.0</td>
<td>80.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Sylvester</td>
<td>1344</td>
<td>51.0</td>
<td>-</td>
<td>74.0</td>
<td>74.0</td>
<td>73.0</td>
<td>97.4</td>
</tr>
<tr>
<td>Face occlusion 1</td>
<td>858</td>
<td>35.0</td>
<td>-</td>
<td>100.0</td>
<td>93.0</td>
<td>100.0</td>
<td>98.9</td>
</tr>
<tr>
<td>Face occlusion 2</td>
<td>812</td>
<td>75.0</td>
<td>-</td>
<td>48.0</td>
<td>96.0</td>
<td>82.0</td>
<td>96.9</td>
</tr>
<tr>
<td>Tiger</td>
<td>354</td>
<td>38.0</td>
<td>-</td>
<td>20.0</td>
<td>77.0</td>
<td>79.0</td>
<td>88.7</td>
</tr>
<tr>
<td>Board</td>
<td>698</td>
<td>-</td>
<td>10.0</td>
<td>67.9</td>
<td>67.9</td>
<td>75.0</td>
<td>87.1</td>
</tr>
<tr>
<td>Box</td>
<td>1161</td>
<td>-</td>
<td>28.3</td>
<td>61.4</td>
<td>24.5</td>
<td>91.4</td>
<td>91.8</td>
</tr>
<tr>
<td>Lemming</td>
<td>1336</td>
<td>-</td>
<td>17.2</td>
<td>54.9</td>
<td>83.6</td>
<td>70.5</td>
<td>85.8</td>
</tr>
<tr>
<td>Liquor</td>
<td>1741</td>
<td>-</td>
<td>53.6</td>
<td>79.9</td>
<td>20.6</td>
<td>83.7</td>
<td>91.7</td>
</tr>
<tr>
<td>Mean</td>
<td>-</td>
<td>42.2</td>
<td>27.3</td>
<td>58.1</td>
<td>64.8</td>
<td>80.4</td>
<td>92.5</td>
</tr>
</tbody>
</table>

6.4.2 Comparison 2: PROST

TLD1.0 was compared with the results reported in [Santner 10], which reports on performance of 5 algorithms: (i) Online Boosting (OB) [Grabner 06], (ii) Online Random Forrest (ORF) [Saffari 09], (iii) Fragment-based Tracking (FT) [Adam 06], (iv) Multiple Instance Learning (MILTrack) [Babenko 09] and (v) PROST [Santner 10]. The evaluation was performed on 10 sequences, which include partial occlusions and pose changes. The performance was reported using two measures: (i) Recall - number of true positives (50% overlap) divided by the sequence length, and (ii) Localization error - average distance between the predicted and the ground truth bounding box centers.

TLD1.0 estimates the scale of an object. However, the algorithms compared in this experiment perform tracking in single scale only. In order to make a fair comparison, the scale estimation was not used in this experiment.

Table 6.2 shows the performance measured by Recall. TLD1.0 scored best in 9/10 outperforming by more than 12% the second best. Table 6.2 shows the performance measured by Localization error. TLD1.0 scored best in 7/10 being 1.6 times more accurate than the second best.
6.4. Quantitative evaluation

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Frames</th>
<th>OnlineBoost</th>
<th>OnlineRF</th>
<th>FragTrack</th>
<th>MILTrack</th>
<th>PROST</th>
<th>TLD1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Girl</td>
<td>452</td>
<td>43.3</td>
<td>-</td>
<td>26.5</td>
<td>31.6</td>
<td>19.0</td>
<td>18.1</td>
</tr>
<tr>
<td>David</td>
<td>502</td>
<td>51.0</td>
<td>-</td>
<td>46.0</td>
<td>15.6</td>
<td>15.3</td>
<td>4.0</td>
</tr>
<tr>
<td>Sylvester</td>
<td>1344</td>
<td>32.9</td>
<td>-</td>
<td>11.2</td>
<td>9.4</td>
<td>10.6</td>
<td>5.9</td>
</tr>
<tr>
<td>Face occlusion 1</td>
<td>858</td>
<td>49.0</td>
<td>-</td>
<td>6.5</td>
<td>18.4</td>
<td>7.0</td>
<td>15.4</td>
</tr>
<tr>
<td>Face occlusion 2</td>
<td>812</td>
<td>19.6</td>
<td>-</td>
<td>45.1</td>
<td>14.3</td>
<td>17.2</td>
<td>12.6</td>
</tr>
<tr>
<td>Tiger</td>
<td>354</td>
<td>17.9</td>
<td>-</td>
<td>39.6</td>
<td>8.4</td>
<td>7.2</td>
<td>6.4</td>
</tr>
<tr>
<td>Board</td>
<td>698</td>
<td>-</td>
<td>154.5</td>
<td>154.5</td>
<td>51.2</td>
<td>37.0</td>
<td>10.9</td>
</tr>
<tr>
<td>Box</td>
<td>1161</td>
<td>-</td>
<td>145.4</td>
<td>145.4</td>
<td>104.5</td>
<td>12.1</td>
<td>17.4</td>
</tr>
<tr>
<td>Lemming</td>
<td>1336</td>
<td>-</td>
<td>166.3</td>
<td>166.3</td>
<td>14.9</td>
<td>25.4</td>
<td>16.4</td>
</tr>
<tr>
<td>Liquor</td>
<td>1741</td>
<td>-</td>
<td>67.3</td>
<td>67.3</td>
<td>165.1</td>
<td>21.6</td>
<td>6.5</td>
</tr>
<tr>
<td>Mean</td>
<td>-</td>
<td>32.9</td>
<td>133.4</td>
<td>78.0</td>
<td>46.1</td>
<td>18.4</td>
<td>10.9</td>
</tr>
</tbody>
</table>

Table 6.3: Localization error (pixels) – TLD1.0 in comparison to results reported in [Santner 10]. Bold numbers indicate the best score; a dash indicates that the result was not reported. TLD1.0 scored best in 7/10 sequences.

6.4.3 Comparison 3: TLD data set

The previous experiments show that TLD1.0 performs well on benchmark sequences where the recall is in the range 90 - 100. We consider these sequences as saturated and introduce new, more challenging ones.

The TLD data set consists of 10 sequences. The sequences 1-6 have been used in experiment 6.4.1 and include: David, Jumping, Pedestrian 1, Pedestrian 2, Pedestrian 3 and Car. The sequences 7-10 are new and include: Motorbike, Volkswagen, Car Chase and Panda. These sequences are long and contain all the challenges outlined in the chapter. All sequences were manually annotated with ground truth. In every frame, the object is defined by a bounding box or it is indicated that the object is not visible. More than 50% of occlusion or more than 90 degrees of out-of-plane rotation was annotated as "not visible". See Appendix B for more details. The TLD data set is available online at the website of the TLD project.

Five tracking algorithms are compared on the TLD data set: (1) Online Boosting (OB) [Grabner 06], (2) Semi-Supervised Online Boosting (SOB) [Grabner 08], (3) Beyond Semi-Supervised Online Boosting (BSOB) [Stalder 09], (4) Multiple Instance Learning (MILTrack) [Babenko 09], and (5) Co-trained Generative and Discriminative tracker (CoGD) [Yu 08]. Binaries for trackers (1-3) are available in the Internet.

Tracker (4,5) were kindly evaluated directly by their authors.

The performance is evaluated using precision $P$, recall $R$ and f-measure $F$. $P$ is the number of true positives divided by number of all responses, $R$ is the number true positives divided by the number of object occurrences that should have been detected. $F$ combines these two measures as $F = 2PR/(P + R)$. Since this experiment compares

---

1. cmp.felk.cvut.cz/tld
various trackers for which the default initialization (defined by the ground truth) might not be optimal, it was allowed to initialize the object arbitrarily. For instance, when tracking a motorbike racer, some algorithms might perform better when tracking only a part of the racer. Every trajectory was therefore normalized. A transformation that mapped the initializing bounding box to the ground truth bounding box was found (shift, aspect and scale) and this transformation was applied to every bounding box on the trajectory. The normalized trajectory was directly compared to ground truth using overlap and true positive was considered if the overlap was larger than 25%. The earlier used threshold 50% was found to be too restrictive in this case. Sequences Motocross and Volkswagen were evaluated by the MILTrack [Babenko 09] only up to the frame 500 as the implementation required loading all images into memory in advance. Since the algorithm failed during this period the remaining frames were considered as failed.

Table 6.4 show the performance as measured by f-measure. The last row shows a weighted average performance (weighted by number of frames in the sequence). TLD1.0 achieved the best performance of 81% significantly outperforming the second best approach that achieved 22%, other approaches range between 13-15%. The performance is broken down to precision in table 6.5 and recall in table 6.6. This experiment demonstrates that TLD1.0 significantly outperforms state-of-the-art approaches on challenging data.
6.5. Long-term tracking of faces

This section adopts the TLD1.0 system to tracking of human faces that we call the Face-TLD. We consider the same block structure of the system as outlined in figure 6.2 with the only modification in the object detector, where the ensemble classifier is replaced by a generic object detector developed in chapter 4. Our goal is to investigate whether the information about the object class helps in the long-term tracking or not.

### 6.5.1 Sitcom episode

This experiment compares the TLD1.0 with Face-TLD on a sitcom episode It Crowd (see appendix B). The episode is 22 minutes long (35471 frames) and contains a number of characters. For speed purposes the original frames were downsampled to resolution $320 \times 176$ pixels. Both systems were initialized on a face of one character (Roy) at his first appearance and automatically tracked the face up to the end of the sequence.

---

**Table 6.5:** Precision – performance on the TLD data set. Bold numbers indicate the best score. TLD1.0 achieved the precision of 82%, the second best achieved 80%.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Frames</th>
<th>OB (Grabner 06)</th>
<th>OSB (Grabner 08)</th>
<th>BOSB (Stalder 09)</th>
<th>MILTrack (Babenko 09)</th>
<th>CoGD (Yu 08)</th>
<th>TLD1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>761</td>
<td>0.41</td>
<td>0.35</td>
<td>0.32</td>
<td>0.15</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Jumping</td>
<td>313</td>
<td>0.47</td>
<td>0.25</td>
<td>0.17</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Pedestrian 1</td>
<td>140</td>
<td>0.61</td>
<td>0.48</td>
<td>0.29</td>
<td>0.69</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Pedestrian 2</td>
<td>338</td>
<td>0.77</td>
<td>0.85</td>
<td>1.00</td>
<td>0.10</td>
<td>0.72</td>
<td>0.89</td>
</tr>
<tr>
<td>Pedestrian 3</td>
<td>184</td>
<td>1.00</td>
<td>0.41</td>
<td>0.92</td>
<td>0.69</td>
<td>0.85</td>
<td>0.99</td>
</tr>
<tr>
<td>Car</td>
<td>945</td>
<td>0.94</td>
<td>1.00</td>
<td>0.99</td>
<td>0.23</td>
<td>0.95</td>
<td>0.92</td>
</tr>
<tr>
<td>Motocross</td>
<td>2665</td>
<td>0.33</td>
<td>0.13</td>
<td>0.14</td>
<td>0.05</td>
<td>0.93</td>
<td>0.89</td>
</tr>
<tr>
<td>Volkswagen</td>
<td>8576</td>
<td>0.39</td>
<td>0.04</td>
<td>0.02</td>
<td>0.42</td>
<td>0.79</td>
<td>0.80</td>
</tr>
<tr>
<td>Carchase</td>
<td>9928</td>
<td>0.79</td>
<td>0.80</td>
<td>0.52</td>
<td>0.62</td>
<td>0.95</td>
<td>0.86</td>
</tr>
<tr>
<td>Panda</td>
<td>3000</td>
<td>0.95</td>
<td>1.00</td>
<td>0.99</td>
<td>0.36</td>
<td>0.12</td>
<td>0.58</td>
</tr>
<tr>
<td>mean</td>
<td>26850</td>
<td>0.62</td>
<td>0.50</td>
<td>0.39</td>
<td>0.44</td>
<td>0.80</td>
<td>0.82</td>
</tr>
</tbody>
</table>

**Table 6.6:** Recall – performance on the TLD data set. Bold numbers indicate the best score. TLD1.0 achieved recall of 81%, the second best achieved 18%.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Frames</th>
<th>OB (Grabner 06)</th>
<th>OSB (Grabner 08)</th>
<th>BOSB (Stalder 09)</th>
<th>MILTrack (Babenko 09)</th>
<th>CoGD (Yu 08)</th>
<th>TLD1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>761</td>
<td>0.29</td>
<td>0.35</td>
<td>0.24</td>
<td>0.15</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Jumping</td>
<td>313</td>
<td>0.05</td>
<td>0.13</td>
<td>0.14</td>
<td>1.00</td>
<td>0.99</td>
<td>1.00</td>
</tr>
<tr>
<td>Pedestrian 1</td>
<td>140</td>
<td>0.14</td>
<td>0.33</td>
<td>0.10</td>
<td>0.69</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Pedestrian 2</td>
<td>338</td>
<td>0.12</td>
<td>0.71</td>
<td>0.02</td>
<td>0.12</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>Pedestrian 3</td>
<td>184</td>
<td>0.33</td>
<td>0.33</td>
<td>0.46</td>
<td>0.81</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Car</td>
<td>945</td>
<td>0.59</td>
<td>0.67</td>
<td>0.56</td>
<td>0.25</td>
<td>0.96</td>
<td>0.97</td>
</tr>
<tr>
<td>Motocross</td>
<td>2665</td>
<td>0.00</td>
<td>0.03</td>
<td>0.00</td>
<td>0.02</td>
<td>0.30</td>
<td>0.77</td>
</tr>
<tr>
<td>Volkswagen</td>
<td>8576</td>
<td>0.02</td>
<td>0.04</td>
<td>0.01</td>
<td>0.04</td>
<td>0.06</td>
<td>0.96</td>
</tr>
<tr>
<td>Carchase</td>
<td>9928</td>
<td>0.03</td>
<td>0.04</td>
<td>0.12</td>
<td>0.04</td>
<td>0.04</td>
<td>0.70</td>
</tr>
<tr>
<td>Panda</td>
<td>3000</td>
<td>0.35</td>
<td>0.17</td>
<td>0.17</td>
<td>0.40</td>
<td>0.12</td>
<td>0.63</td>
</tr>
<tr>
<td>mean</td>
<td>26850</td>
<td>0.09</td>
<td>0.10</td>
<td>0.10</td>
<td>0.11</td>
<td>0.18</td>
<td>0.81</td>
</tr>
</tbody>
</table>
Chapter 6. Tracking-Learning-Detection (TLD)

Figure 6.10: Evaluation of Face-TLD on a sitcom episode “IT crowd”. (TOP-LEFT) The initial frame. The entire sequence (22 minutes) was then processed automatically.

The TLD1.0 tracked correctly beginning of the episode, but failed to detect the character in the second half. The overall recall was of 37% and precision of 70%. The Face-TLD was able to track the target throughout the entire episode leading to recall of 54% and precision of 75%. The introduction of the face detector increased the recall by 17%. Both approaches processed the episode at 20 frames per second a laptop. Figure 6.10 shows several frames from the episode and the online model.

6.5.2 Surveillance footage

This section performs a quantitative comparison on sequence Surveillance (see appendix B). The sequence consists of 500 frames depicting interior of a shop with multiple people captured at 1 frame per second. The sequence cannot be tracked by pure face detector as there are multiple faces which occlude one another. Moreover, frame-to-frame tracking are difficult to apply because the frame-to-frame motion is large and the subjects move in and out of the camera view.

The TLD1.0 was again compared to Face-TLD. The TLD1.0 achieved recall of 12% and precision of 57%, the Face-TLD achieved recall of 35% and precision of 79%. The introduction of face detector increased the recall by 23%. Figure 6.11 illustrates the scenario. This experiment demonstrates, that both TLD1.0 and Face-TLD are applicable to surveillance scenarios for tracking of faces. Furthermore, it shows that using a face detector increases the performance of the TLD system.
6.6 Qualitative analysis

This section discusses the strengths and weaknesses of TLD1.0 with respect to the challenges outlined in the chapter. The performance is illustrated on several snapshots from run-time of the system, where the following marking is used:

- **Yellow rectangle** - the bounding box output of the TLD1.0,
- **Gray dots** - detections output by the second stage of our detector,
- **Red dots** - detections output by the whole detector,
- **Blue dots** - the reliable points used by the Median-Flow tracker,
- **Patches to the left** - negative examples in the online model,
- **Patches to the right** - positive examples in the online model.
- Top left corner depict close-up of the object.

Figure 6.11: Evaluation of Face-TLD on sequence Surveillance. (LEFT) Responses of generic face detector (red), detections approved by online learned model (black), ground truth trajectory of the subject (blue). (RIGHT) The surveillance scenario.
6.6.1 Strengths

Scale changes. TLD1.0 is robust to scale changes. Median-Flow estimates scale changes even when the target is partially out of the frame. Detector localizes the object in multiple scales. In sequence Volkswagen, the object of interest changes scale in the range from 20x20 to 100x100 pixels. The output of TLD1.0 is illustrated in figure 6.12.

Illumination changes. TLD1.0 is invariant to smooth changes in illumination. Median-Flow adapts the tracked templates and the detector is based on illumination invariant pixel comparisons and NCC. In sequence David, a face is tracked from a dark room to full illumination. Figure 6.13 shows the results.

Appearance changes. TLD1.0 deals with significant appearance changes. Median-Flow handles well changes of appearance caused by pose change or articulations. The detector localizes all appearances observed in the past. In sequence Motocross, TLD1.0 learned all appearances of a motorbike from the rear view. However, it did not learn the side view. The detector was therefore not able to re-initialize a trajectory in that case. See figure 6.14 for illustration.
Partial occlusions. TLD1.0 deals with partial occlusions. Median-Flow tracker estimates reliable points within the bounding box and filters out parts of the object that are occluded. In sequence Face occlusion 2, the object of interest becomes partially occluded, but TLD1.0 is able to track these changes successfully as illustrated in figure 6.15.

Full occlusions and disappearances. The main power of TLD1.0 is the ability to re-detect the target after full occlusion or disappearance of the object from the scene. Figure 6.16 shows the re-detection in sequence Car. Note that the appearance of the car after occlusion is different from the initial appearance.

Similar targets. TLD1.0 is discriminative. If the object of interest is surrounded by objects of similar appearance or background clutter, the N-expert labels them as negative and inserts them to the online model. These negative examples then prevent the detector from confusing the object of interest with other objects. For instance in figure 6.17 the object of interest is a helmet of a football player. Notice that various appearances of the same helmet appear in positive examples, whereas different helmets occur in negative examples. This is best visible when zooming in on a display.
6.6.2 Weaknesses

While TLD1.0 has demonstrated a significant improvement in comparison to other approaches, there are situations which TLD1.0 does not handle well.

**Out-of-plane rotations.** In case of out-of-plane rotation, Median-flow drifts away from the target. The tracker typically stays away until a detector re-initializes its position to previously seen appearance. For instance, figure 6.18 shows a sequence of an object performing out-of-plane rotation. From frame 1 to 168, the object performs out of plane rotation and the Median-Flow drifts. As the drift is slow, the failure of the tracker is not identified and the system is learning new incorrect appearances. In frame 176 the object re-appears in previously seen appearance and the trajectory is correctly re-initialized. Notice that the incorrect data produced by the drift did not prevent the tracker from correct re-initialization of the trajectory.

**No previously seen appearances.** Particularly challenging scenarios for TLD1.0 are scenes when the object never re-appears in previously observed appearance. For instance, figure 6.19 shows an example when tracking a face in a Soccer sequence from [Kwon 10]. The target object is tracked for a couple of frames but then fails due to occlusion combined with pose and expression changes. The target is never re-detected as the object never re-appears with previously observed appearance.
6.6. Qualitative analysis

Figure 6.18: Out-of-plane rotations are challenging for TLD1.0. The sequence appeared in [Leichter 09].

Figure 6.19: TLD1.0 and sequences where the object never re-appear in a similar view. The sequence appeared in [Kwon 10].
Chapter 7

Discussion

This chapter discusses the contributions of the thesis, reviews recent developments, and proposes possible avenues for future research.

7.1 Contributions

In this thesis, we have proposed models and methods for real-time long-term tracking. In particular, we focused on tracking of a priori unknown objects as well as tracking of human faces. Our approach was demonstrated on number of challenging videos, and significantly outperformed state-of-the-art. The particular contributions are summarized below.

In chapter 3, we studied the long-term tracking problem from the perspective of frame-to-frame tracking. We accepted that frame-to-frame (closed-loop) tracking is not a good model for this scenario as it leads to inevitable failures. Rather than trying to avoid these failures by directly designing a better frame-to-frame tracker, we proposed a novel measure that indicates the reliability of a tracker. The measure is based on the well known forward-backward consistency assumption. We demonstrated that the proposed measure provides complementary information to appearance-based NNC and SSD. Furthermore, we used the error measure to improve frame-to-frame tracking itself. We showed that template tracking can be improved if the template is decomposed into independently tracked parts which are weighted based on their reliability and integrated using median estimator. The result is a novel template-based tracker (Median-Flow) which is robust to partial occlusions and appearance changes and outperforms comparable approaches.

In chapter 4, we studied the long-term tracking problem from the perspective of tracking-by-detection. We developed a novel learning method for supervised training of an object detector from a large data set. In particular, we focused on learning methods that
combine bootstrapping and boosting. The theoretical contribution is the formalization of bootstrapping and boosting in a unified framework. Within this framework, we designed the optimal combination of the two approaches. The approach formulates bootstrapping as a weighted sampling where the weights driven by boosting. The resulting combination demonstrated a significant improvement in terms of efficiency (both in training and testing) as well as the classifier accuracy in contrast to ad hoc combinations. The learning method has been applied to training a face detectors (frontal and profile), which operate at video frame-rate on QVGA images. Such detectors are relevant to all long-term tracking scenarios, where the target object is a face. The learning method does not make any face-specific assumptions and can be applied to any other visual classes.

In chapter 3 we investigated the task of learning during long-term tracking. We have demonstrated that an accurate object detector can be trained from a single example and an unlabeled video stream using the following strategy: (i) evaluate the detector, (ii) estimate its errors, and (iii) update the detector. The main novelty of the method is the estimation of the detector errors, which is guided by two types of rules, which we call $P$-expert and $N$-expert, respectively. $P$-expert estimates only false negatives and improves the detector generality. $N$-expert estimates only false positives and increases the detector discriminability. Estimation of the detector’s errors independently based on their type enabled not only simpler design of the experts, but also mutual compensation of their errors. The theoretical contribution is the formalization of this learning process as a discrete dynamical system, which allowed us to specify conditions, under which the learning guarantees improvement of the detector. We demonstrated, that the experts can be designed when considering spatio-temporal relationships in the video, which opens possibilities for indefinite learning of object detectors from unlabeled video streams.

In chapter 6 we proposed a novel tracking framework (TLD) that decomposes the long-term tracking task into three sub-tasks: tracking, learning and detection. The aim of the framework is to track an unknown object immediately after initialization, learn its appearance on the fly and re-detect the object whenever it re-appears in the video stream. Building on the components developed in chapters 3, 4 and 5, we showed how implement the TLD framework and how to achieve real-time performance and stability. An extensive quantitative evaluation on benchmark sequences demonstrated saturated performance. Therefore, we introduced a new data set, ground truth and evaluation protocol and showed a significant improvement over state-of-the-art approaches. Finally, we applied TLD to tracking of human faces and demonstrated how to incorporate an offline trained detector to further improve the long-term tracking.
7.2 Recent development

Based on the algorithms developed in this thesis, several improvements have been proposed in [Vojir 10]: (i) new types of P-experts based on the CamShift [Bradski 98] tracker, (ii) automatic discovery of the object shape in order to improve the tracking, (iii) learning of the ensemble classifier combined with bagging.

The implementation of TLD has been demonstrated at Computer Vision and Pattern Recognition, 2010. The system has been running for more than 8 hours, tracking various objects in real-time. Figure 7.1 shows the setup of the demo and several snapshots taken automatically from the demo camera.

The face detector developed in chapter 4 has been ported to RAVL\(^1\) (Recognition And Vision Library) that is being developed at Centre for Vision, Speech and Signal Processing, at the University of Surrey. Moreover, the system has been adapted to multi-view face detection and has been used in BAE/OmniPerception Project on Face Recognition.

The demo applications of TLD1.0 and Face-TLD have been made cubically available and are accessible on the website of the author. Currently, a release of the source code is being considered.

\(^1\)http://ravl.sourceforge.net/
7.3 Future work

In this section we outline possible directions for further research.

**Tracking.** In chapter 7 we have demonstrated that a relatively simple tracker coupled with a learning method and detector, significantly outperforms competitive approaches. On the other hand, we observed that if the tracker fails too quickly (e.g. due to out-of-plain rotation), the detector is not learned sufficiently in order to re-initialize the tracker. Therefore one promising way to proceed is to strengthen the tracking itself, e.g. by running multiple trackers in parallel [Kwon 10] each of which would be based on different features and motion models. Such an approach would lead to a set of P-experts that would train the detector more quickly.

**Detection.** The detector used in our system is based on a scanning window and global representation of the object and as such is prone to occlusions. However, as we have reviewed in chapter 2 a number of detectors is based on local representation [Lowe 04] where partial occlusions are not an issue. Another possible direction is to base the detector on Dominant Orientation Templates [Hinterstoisser 10] in order to handle better the non-textured objects.

**Learning.** In chapter 5 we proposed a learning method (P-N Learning) which processes a video stream in one pass, considering only one frame at a time. Frames observed so far have not been used. While our motivation was mainly speed, this is no longer an issue for multi-threaded architecture. One can imagine a second thread analysing the already processed frames, thus providing more training data for the detector. In ideal case, the detector should reflect all the information received up to current time.

**Tracking multiple objects.** A particularly promising direction is to adapt the TLD for tracking of a large number of small patches. One could imagine a scenario when hundreds of points are tracked simultaneously, each of which has the property to learn its appearance on the fly, detects its disappearance and re-initialize its own trajectory when it becomes visible. Such an approach would be applicable in a number of applications. Preliminary experiments are promising for long-term tracking of non-rigid and fast moving objects. See figure 7.2 for illustration.

**Faster implementation.** While current implementation of TLD1.0 is running in real-time on QVGA images, for larger images the frame-rate drops since the detector has to evaluate a larger number of windows. GPU implementation is a potential way to increase the speed as the scanning window approach can be parallelized. Implementation of pieces of code using SSE instruction set is also a promising direction.

**Apply P-N Learning to other problems.** In chapter 5 P-N Learning was applied to data from a video stream with a specific spatio-temporal structure. The structure is, however, present in many other problems as well. One possible way is to exploit
search engines such as Google Image Search. Search for a particular object (e.g. a dog) returns a set of images where the object is likely to appear (P-expert). A search for other object categories returns a set of images where the object is typically not present (N-expert).

**Sophisticated experts.** The P-N experts used in this thesis are relatively simple, as they were describing motion of a single object in a video stream. Apart from the assumption that the object is unique in the image and the object moves on a trajectory, the experts did not take any other assumption. In more complex scenarios, e.g. when tracking multiple object in parallel, one can came up with more complex rules to describe the problem. Here we are motivated by the philosophy that "the more you know, the less you need to learn" [Tommasi 09].
Appendix A

Compared algorithms

1. **FT**: Fragment-based Tracker [Adam 06], a static template tracker which represents the object by a set of parts. In presence of partial occlusions, the method have demonstrated better performance than Mean-Shift [Comaniciu 03].

2. **IVT**: Incremental Visual Tracking [Ross 07], a particle filter that incrementally builds a PCA-based model of the object.

3. **ET**: Ensemble Tracking [Avidan 07], a mean shift-based tracker that adapts an discriminative model classifying pixels.

4. **ODF**: Online Discriminative Features [Collins 03], a mean shift-based tracker that adapts color projections to separate the object from background.

5. **OB**: Online Boosting Tracker [Grabner 06], an approach similar to ODF, but the classification is performed on bounding box level.

6. **ORF**: Online Random Forests [Saffari 09], an approach similar to OB but more robust with respect to noise.

7. **SOB**: Semi-supervised Online Boosting [Grabner 08], an extension of OB that internally trains 2 classifiers.

8. **BSOB**: Beyond Semi-supervised Online Boosting [Stalder 09], an extension of SOB that trains 3 classifiers to simultaneously increase adaptability and stability.

9. **MILTrack**: Multiple Instance Learning tracker [Babenko 09], an approach similar to OB but with a modified, drift-resistant updating strategy.

10. **CoGD**: Co-trained Generative and Discriminative [Yu 08], a particle filter that co-trains a pair of classifiers.

11. **PROST**: Parallel Robust Online Simple Tracking [Santner 10], a method based on three complementary trackers: template, optical flow and random forest.
Appendix B

Sequences used for evaluation

The TLD1.0 system has been quantitatively evaluated on 21 sequences specified in table B.1 and shown in figure B.1.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequence name</th>
<th>Frames</th>
<th>First appeared in</th>
<th>Moving camera</th>
<th>Partial occlusion</th>
<th>Full occlusions</th>
<th>Out-of-plane rotation</th>
<th>Illumination change</th>
<th>Scale change</th>
<th>Similar objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David</td>
<td>761</td>
<td>D. Ross et al., IJCV’08</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>Jumping</td>
<td>313</td>
<td>Q. Yu, ECCV’08</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>Pedestrian 1</td>
<td>140</td>
<td>S. Avidan, PAMI’07</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>Pedestrian 2</td>
<td>338</td>
<td>Q. Yu et al., ECCV’08</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>Pedestrian 3</td>
<td>184</td>
<td>Q. Yu et al., ECCV’08</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>Car</td>
<td>945</td>
<td>Q. Yu et al., ECCV’08</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>7</td>
<td>Girl</td>
<td>502</td>
<td>S. Birchfield, CVPR’89</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>Silverster</td>
<td>1,344</td>
<td>D. Ross et al., IJCV’08</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>Face occlusion 1</td>
<td>885</td>
<td>A. Adam et al., CVPR’08</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>10</td>
<td>Face occlusion 2</td>
<td>812</td>
<td>B. Babenko et al., CVPR’09</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>11</td>
<td>Tiger</td>
<td>354</td>
<td>B. Babenko et al., CVPR’09</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>12</td>
<td>Board</td>
<td>698</td>
<td>J. Santner et al., CVPR’10</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>13</td>
<td>Blob</td>
<td>1,161</td>
<td>J. Santner et al., CVPR’10</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>14</td>
<td>Lemming</td>
<td>1,336</td>
<td>J. Santner et al., CVPR’10</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>15</td>
<td>Lucius</td>
<td>1,741</td>
<td>J. Santner et al., CVPR’10</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>16</td>
<td>Motocross</td>
<td>2,665</td>
<td>Z. Kala et al., CVPR’10</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>17</td>
<td>Volkswagen</td>
<td>8,576</td>
<td>Z. Kala et al., CVPR’10</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>18</td>
<td>Car Chase</td>
<td>9928</td>
<td>Z. Kala et al., CVPR’10</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>19</td>
<td>Panda</td>
<td>2,000</td>
<td>Z. Kala et al., CVPR’10</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>20</td>
<td>IT Crowd</td>
<td>35,471</td>
<td>Z. Kala et al., CVPR’10</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>21</td>
<td>Surveillance</td>
<td>500</td>
<td>Z. Kala et al., CVPR’10</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table B.1: Description of sequences used for evaluation. A horizontal line separates the standard and the introduced sequences. The red color indicates sequences from the TLD data set.
Figure B.1: Snapshots from the sequences used for evaluation. A horizontal line separates the standard and the introduced sequences.
Bibliography


[Belhumeur 97] P Belhumeur, J Hespanha & D Kriegman. Eigenfaces vs. Fish-
erfaces: recognition using class specific linear projection. IEEE
Transactions on Pattern Analysis and Machine Intelligence, 1997.

[Bibby 08] C Bibby & I Reid. Robust real-time visual tracking using pixel-wi-

[Bibby 10] C Bibby & I Reid. Real-time Tracking of Multiple Occluding Ob-
jects using Level Sets. Computer Vision and Pattern Recognition,
2010.

[Birchfield 98] S Birchfield. Elliptical head tracking using intensity gradients
and color histograms. Conference on Computer Vision and Pat-
tern Recognition, 1998.

[Black 98] M J Black & A D Jepson. Eigentracking: Robust matching and
tracking of articulated objects using a view-based representation.
International Journal of Computer Vision, vol. 26, no. 1, pages

[Blum 98] A Blum & T Mitchell. Combining labeled and unlabeled data
with co-training. Conference on Computational Learning Theory,
page 100, 1998.

Feature Tracker Description of the algorithm. Technical Report,

[Bradski 98] G R Bradski. Computer vision face tracking for use in a percep-

pages 123–140, 1996.

[Breiman 01] L Breiman. Random forests. Machine Learning, vol. 45, no. 1,

optical flow estimation based on a theory for warping. European

Long term arm and hand tracking for continuous sign language


Bibliography


