CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY IN PRAGUE

PhD THESIS

ISSN 1213-2365

Linear Predictors for Real-time Object Tracking and Detection

David Hurych

hurycd1@cmp.felk.cvut.cz

CTU–CMP–2014–04

April 14, 2014

Thesis Advisor: Tomáš Svoboda, Karel Zimmermann

Czech Science Foundation Project P103/10/1585 and CTU Prague project SGS11/125/OHK3/2T/13,

Research Reports of CMP, Czech Technical University in Prague, No. 4, 2014

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Linear Predictors for Real-time Object Tracking and Detection

A dissertation presented to the Faculty of Electrical Engineering of the Czech Technical University in Prague to meet the requirements for the Ph.D. degree in study programme No. P 2612 - Electrical Engineering and Information Technology, branch No. 3902V035 - Artificial Intelligence and Biocybernetics, by

David Hurych

Prague, April 2014

Supervisor: **Doc. Ing. Tomáš Svoboda, Ph.D.**
Supervisor-Specialist: **Ing. Karel Zimmermann, Ph.D.**

Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
fax: +420 224 357 385, phone: +420 224 357 465
http://cmp.felk.cvut.cz

Acknowledgments

I would like to express my sincere gratitude to my supervisor Doc. Ing. Tomáš Svoboda, Ph.D. and to my supervisor-specialist Ing. Karel Zimmermann, Ph.D. It has been a great pleasure to work with them and to learn from their experience. Their patience and support guided me through my Ph.D. study. Also I would like to thank to my colleagues and friends in the Center for Machine Perception with whom I had the pleasure to work and discuss.

Abstract

The thesis is focused on fast visual object tracking and detection methods. We use the linear regression to learn a predictor, which forms a mapping from the image data directly to the tracking parameters. We improve this popular and widely used concept by learning specific regression functions, which approximate non-linear regression functions and run extremely fast during tracking. Our method allows to estimate even a complex non-rigid object deformation from the image data much faster than in real-time. We apply our method on various problems including real-time object detection. We also show, how to use the learnable tracker for instant object tracking. Shrinkage of the learning time to minimum is necessary, therefore only a few training samples may be used. The tracker is incrementally learned during the tracking procedure in order to improve its robustness. A method is proposed for automatic selection of only the useful training samples for incremental learning. Next we demonstrate, that the predictor learned to track one image patch is also able to track many other image patches. We show how to use the learned predictor for fast detection of these trackable patches. Thanks to the efficiency of the proposed non-linear regression functions we may combine any sequential decision process (e.g. object detector) with our tracker to run in cooperation in real-time. The regression functions are jointly learned with a state of the art sliding window object detector on the same set of features collected from training data. The combination of both methods significantly improves the detection rates, thanks to the ability to align the detection window on deformed object instances. Since the combined detector is able to localize the object position and deformation from its close neighborhood, we speed up the detection process by a search space reduction.

Contents

Keywords: object tracking, object detection, regression, predictors, machine learning, real-time, computer vision

Notation and commonly used symbols

$\mathbf{a}$	small and bold font denotes a vector
$\mathtt{A}$	capital typewriter font denotes a matrix
A	capital italic font denotes a set
r_j	denotes a regressor operating with a j-th feature
γ	denotes a predictor, which is a set of regressors
γ_s	denotes a sequential predictor, which is a sequence of predictors
a^i	upper index usually denotes a particular training sample number
$(\dots)^T$ $\begin{pmatrix} \vdots \end{pmatrix}$	space or coma separated values in parentheses denote a vector
$[\dots]$	brackets denote a composition of scalars, vectors and matrices into a new matrix
$\Vert \dots \Vert$	euclidean norm of a vector
$\Vert \dots \Vert_F$	Frobenius norm of a matrix

1 Introduction

We focus on real-time visual object tracking and detection. The goal of object tracking is to obtain the motion trajectory of some object. The input data is a video sequence, captured by a digital camera. Apart from the trajectory we may also be interested in more complex information about the objects' pose, e.g. the 3D pose and orientation in space or a relative position of some objects' parts (like human limbs) etc. This object pose information may be obtained either by an object detector from every image separately or by a tracker in a sequence of images. Object tracking is closely connected to the task of object detection that is why we focus on both and we explore the posibilities of combining these methods for mutual benefit.

There are numerous applications, where an object tracking is necessary. Popular application is *augmented reality*, where objects or the whole scene needs to be tracked in order to be augmented. In movie postproduction industry object tracking is needed for various tasks, like deletion of undesired objects from the scene or object appearance modification, automatic video cut, etc. Human face tracking is also very frequent application as a part of face recognition systems or human-computer interaction. In many security systems motion tracking is used to monitor some area. Another common application is tracking cars on streets, which enables automatic traffic monitoring or estimating the speed of cars. For a robot navigation it is necessary to track and model objects at close to avoid the collision, or tracking objects is needed for robotic arm performing some tasks, etc.

Visual object tracking is difficult task for various reasons. The key factors are the lightning conditions in the scene, the geometrical complexity of the tracked object, the objects' texture, speed of the motion, camera framerate and many more. For example tracking of a specific well textured planar object in good lightning conditions is an easy task and the tracking results may be very accurate. Objects with a simple and known geometry may be tracked precisely also in 3D and not only in 2D. An example can be seen on Figure 1.1, where a wired model of a house is fitted into the scene. It fits very well to the scene geometry thanks to estimated homography of the tracked mousepad. A more difficult task is to track a generic object – lets say a human face. It is not possible to learn a specific tracker for every human face in the planet. But it is possible to take advantage of machine learning methods and to train a generic tracker from some representative set of human faces images. See Figure 1.1, where a learned generic face tracker is used to augment different human faces with glasses and another generic deformable tracker is used to track human eye opening.

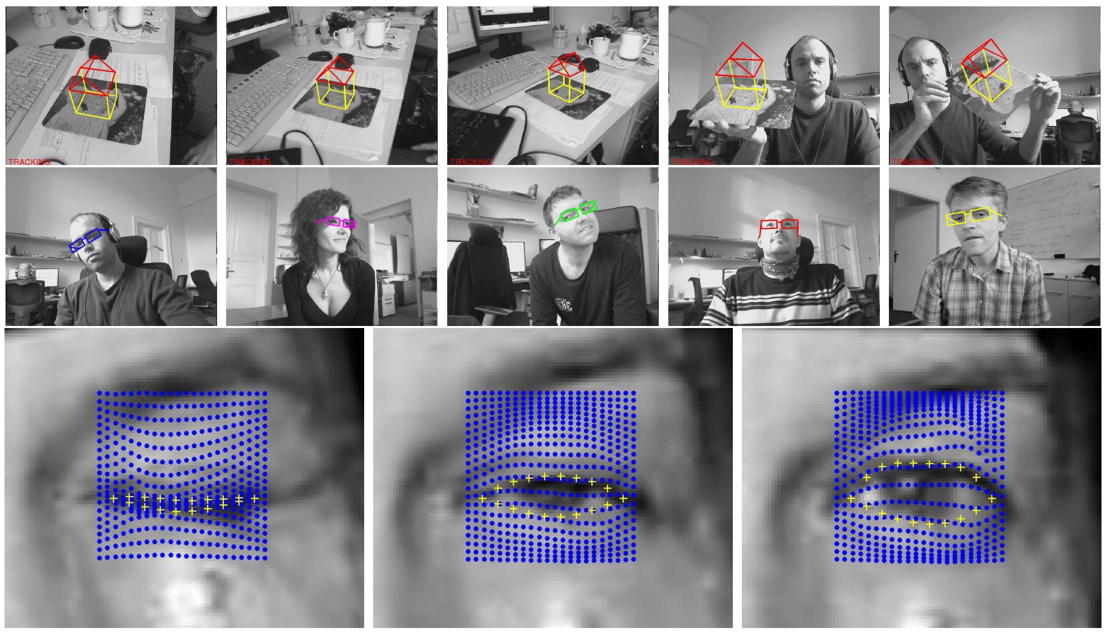


Figure 1.1: Examples of our object tracking results. **First row**: object specific planar tracking used to estimate homography, which allows to compute 3D position of the object and augment it with an artifitial object. **Second row:** object generic tracker is used to estimate 3D position of human head. Note, that the same tracker is used on different people. **Third row:** deformable object generic tracker estimates fully parameterized deformation of a human eye lids during opening.

Object tracking essentially means estimation of object pose in an image given the pose in a previous image. By object pose we mean any type of object motion, deformation, which we want to estimate. Let us define the image data as a mapping $I : \mathbb{R}^{2n} \rightarrow \mathbb{R}^n$ which takes a set of n two dimensional pixels and to each one of them assigns an intensity value. The number of input pixels n into the image function will vary according to current needs. The change of object pose between previous image J and current image I may be described by some transformation $\rho(X, \mathbf{t})$ of each object point in homogeneous coordinates $\forall (x_i, y_i, 1)^T \in X, i \in \{1, 2, \ldots, n\}$. The transformation is parameterized by a set of parameters $\mathbf{t}$ and a set of object points X, called a *support set*, on which the transformation is applied. For example the change of object 2D position, change in in-plane rotation and scale may be parameterized by a vector of parameters $\mathbf{t}^i = (\Delta x, \Delta y, \Delta\alpha, \Delta s)^T$. With these parameters we may build a transformation ρ described by a matrix

$$\mathtt{W}^i = \begin{bmatrix} \Delta s \cos \Delta\alpha & -\Delta s \sin \Delta\alpha & \Delta x \\ \Delta s \sin \Delta\alpha & \Delta s \cos \Delta\alpha & \Delta y \\ 0 & 0 & 1 \end{bmatrix}, \tag{1.1}$$

and the support set X^j are transformed from a pose in an image J to a pose in an image I as folows $X^i = \mathtt{W}X^j$. The object pose in a current image I^i given an initial pose in arbitrary preceeding image I^0 is then given by a composition of transformations $(\ldots((\rho_0 \circ \rho_1) \circ \rho_2) \circ \cdots \circ \rho_i)$. In our example, the composition would be simply a multiplication of consecutive transformation matrices $\mathtt{W} = \mathtt{W}^0\mathtt{W}^1 \ldots \mathtt{W}^i$ The goal of *object tracking* is to estimate the transformation vector $\mathbf{t} = (t_1, t_2, \ldots, t_p)^T$ with p parameters. In order to do this, we first need to extract some image features f_i computed over their support sets X^i which are placed somewhere in the image by a transformation ρ. Then the parameters vector $\mathbf{t}$ is computed from the image data according to

$$\mathbf{t} = \gamma(f_1(I(\rho(X_1, \mathbf{t}^{i-1}))), f_2(I(\rho(X_2, \mathbf{t}^{i-1}))), \ldots, f_k(I(\rho(X_k, \mathbf{t}^{i-1})))). \qquad (1.2)$$

Our task is to design the function $\gamma : \mathbb{R}^k \to \mathbb{R}^p$, which will be from now on called *predictor*. The predictor is a regression function, which is learned from training data. Before learning itself we need to select appropriate image features $f_i : \mathbb{R}^n \to \mathbb{R}, i \in \{1, \ldots, k\}$. f_i is a mapping assigning a feature value to n pixels in the pose specified by parameters $\mathbf{t}^{i-1}$ estimated in the previous image. Each feature f_k works with its own subset of the support set $X_k \in X$. In the simplest case the feature may be directly the image intensity value in one particular pixel. From many possible ways of designing γ we use machine learning methods which learns γ from annotated training samples. Feature functions should be selected specifically for each type of tracked object according to its visual appearance and with respect to the required computational complexity. The advantage of predictors is their low computational complexity during runtime and high tracking precision.

Object detection is a process of localizing a learned object of interest in each image separately. No initial position is given and no ordering of images is required, but otherwise the task is very similar to the object tracking. I.e. the goal is to make a decision about the object presence or absence in every possible pose of the object in an image. Let $\Omega \subset \mathbb{R}^p$ be a set of discretized pose parameters vectors, defining possible object poses in the image. Then an object detector $\Lambda : \mathbb{R}^k \to \{\mathtt{TRUE}, \mathtt{FALSE}\}$ makes a decision about the object presence for all the poses $\forall \mathbf{t}^i \in \Omega$ as

$$\Lambda(f_1(I(\rho(X_1, \mathbf{t}^i))), f_2(I(\rho(X_2, \mathbf{t}^i))), \ldots, f_k(I(\rho(X_k, \mathbf{t}^i)))). \qquad (1.3)$$

The result of object detection in a particular image is a set of poses $U \subset \{\Omega \cup \emptyset\}$ with the `TRUE` votes for object presence.

1.1 Contributions of the Thesis

In our work we have contributed to state of the art visual object tracking and detection. Mainly we focused on machine learning methods which took advantage of off-line learning from annotated training samples. Once learned, these methods achieve high precision in tracking and good detection rates, and keep the fast performance during runtime. We have achieved a real-time performance in tracking of non-rigid objects and estimating a fully parameterized deformation of the object. We also contribute to real-time object detection, by improving the *sequential decision process* (SDP). Our object detector is built as SDP and is combined with learned predictors. It is able to detect also the deformed object instances in real-time. As a part product of a positive detection we get the objects' deformation parameters. Here we present a summary of our contributions:

- Since the predictor is learned from some training set, it has a limited generalization on object appearences, which were not seen in the training set. It is therefore necessary to incrementally re-learn the predictor for new object appearances. We propose an approach for automatic selection of the additional training samples for incremental learning during tracking and a fast loss-of-track detection method in Chapter 3. This method was published and presented in 2010 on INSTICC VISAPP international conference [1]. Next we propose an efficient algorithm for instant object detection and tracking in high resolution images by combining a predictor with incremental learning together with Ferns based object detector. This approach was published and presented in 2011 on INSTICC VISAPP international conference [2].

- We propose to learn a piecewise linear predictor which approximates nonlinear regression function, but keeps the fast performance of linear predictor during runtime. We test this approach on real-time estimation of non-rigid object deformation in Chapter 5. This work has not yet been published and the plan is to submit it in the year 2014.

- Apart from widely used approach *tracking by detection* we also show how the learned predictor may be used for object detection in Chapter 6. We show, that a learned tracker may be initialized in multiple poses in the image and the object exact location is then estimated by a voting procedure. This method was published and presented in 2011 on international Computer Vision Winter Workshop [3].

- The closely related tasks of object detection and tracking may be efficiently combined in a joint learning scheme in order to achieve better precision. In the first experiments we combine a sliding window cascade classifier with interleaved 2D alignment in Chapter 7. This approach was published and presented in 2011 on IEEE ICARA international conference [4].

- We propose to combine a GentleBoost [5] object detector with jointly learned predictors on common set of features in order to create a real-time *deformable object detector* in Chapter 8. The detector is a sequential decision process, which is evaluating features in defined order and may decide about the object presence or absence after each evaluated feature. After each feature evaluation both the detectors' confidence the information about the current object pose are updated. The estimated information about object pose is immediatelly used to align the following features which better fit the objects' pose. We have published and presented this work in 2012 on IEEE ACCV international conference [6] and extended version in 2014 in IEEE TPAMI [7].

The rest of the thesis is divided into two main parts. In the first part (Chapters 2-5) we focus on the object tracking and the problem of changing object appearance, validation of the correct trackers' pose estimate and with speeding-up the tracker for real-time tracking of deformable objects. The second part of the thesis (Chapters 6-8) focuses on combining the tracker and the object detector in order to achieve better detection results and keep the real-time performance.

2 Object Tracking

2.1 State of the Art

Many different methods have been developed for object tracking. Probably the most popular and successful are methods based on *gradient descent* [8][9], which align a template to an image by minimizing their difference. We describe Lucas-Kanade gradient descent optimisation briefly in 2.1.1 and in detail in Appendix C. A lot of effort has been devoted to this method. It has many followers and many different derivations have been developed. In section 2.1.1 we describe the similarities and differences between the Lucas-Kanade and linear predictors.

Another very popular approach for object tracking are *Linear regression-based methods* [10][11][12]. Linear regression is used to find the mapping between image intensities (features) and the space of pose parameters used for tracking in the least squares sense. The tracking model formed by this linear mapping is called a linear predictor. Linear predictors were used for face tracking in [13][14], for acurate perspective patch rectification in [12], and object tracking by an optimal sequence of linear predictors has been proposed in [11]. We describe linear predictors in detail in section 2.1.2. The advantage of linear predictors is their extremely fast performance and good precision of prediction. The disadvantage is that linear predictors need a learning phase before tracking. Another disadvantage is that the prediction fails for input data, which were not seen in the training phase. But the predictor could be incrementally trained during tracking [12] and with optimisation of the learning process and incremental learning these problems could be reduced.

Using object detectors for tracking is a method which naturally comes to mind. Detectors localise the object in every frame and we may keep the sequence of positions (object states) as trajectory. There are numerous robust, tried-and-tested object detectors which may be used such as boosted cascade of classifiers [15], histogram of oriented gradients [16], tracking by support vector machine in [17], etc. *Tracking by detection* has many advantages. It naturally handles loss and re-appearance of the object and fast motion. These detectors are often robust enough to handle sudden changes in object appearance, because they are trained on large datasets, which cover many different appearances of that type of object. The problem is that to train these detectors we need large training set and plenty of time for learning. Another problem is the time efficiency - many object detectors are not fast enough to process a videosequence in real-time [18], although a lot of effort has been devoted to speeding up the detection process (such as using the integral image instead of the original image). The detector needs to process the whole image in different scales, while standard tracking methods usualy work on the close neighborhood of the last object position. Another approach would be to match object region descriptors like SIFT [19], SURF [20] or MSER [21]. These descriptors don't need to be trained in advance. Descriptors are simply computed over the whole image and they are matched with the stored object descriptors. These descriptors are usualy robust to scale change and rotation and are fast enough to be computed in real-time, but may fail if the appearance of the object changes. Lately, Hinterstoisser et al. in [22] proposed fast method for tracking by detection, which is demonstrated on on-line SLAM application (simultaneous localization and mapping). Each feature point (object) in the scene is charaterized by a set of mean patches, where each mean is computed by warping the patch centered on the point over a small range of poses. This approach is fast enough to be run on-line because they developed very fast method for computing these mean patches. In the detection phase they match directly the patches of incoming points to the precomputed database of mean patches. This saves time, because they don't need to compute the mean patch for each incoming feature point before matching. The feature point pose is represented by a homography. After each successful matching of feature point, they use combination of inverse compositional algorithm [23] and linear predictors [10] to obtain a fine estimate of homography. On-the-fly learned object detector and tracker are combined in [24]. Both are not pre-trained and they learn new object appearances from consecutive images in a video. The object detector is a cascade classifier: combination of (i) thresholding the patch variance, (ii) the ensemble classifier estimating a set of posteriors from binary codes and finally (iii) a nearest neighbor classifier. The *median flow tracker* collects new object appearances in every frame as well as image patches of changing background. These examples are used to incrementally learn the object detector and tracker. When the tracker is lost, or the object runs out of image, the detector tries to recover its position and restarts the tracker. The detector is evaluated in every frame of the video and a set of two experts tries to identify the misclassified examples (P-N learning). The P-expert tries to recognize when the detector missed the object and the N-expert tries to recognize a false positive detection. The resulting samples from P-N learning are used to re-learn the detector. In [24] the 2D position and scale of the object are estimated during tracking.

Kernel-based tracking (*mean shift*) [25][26][27] is an iterative tracking method, which measures and minimizes the statistical dissimilarity between the model and the target candidates. It is often modeled by the object patch colour distribution described by some kernel function. Shortly described, in each iteration the mean shift algorithm estimates the kernel probability density function in the position given by last iteration. Then the algorithm estimates the gradient of the density function and the model is shifted in the direction of gradient ascent. Thanks to the statistical model of the object, mean shift tracking is robust to partial occlusions, clutter, camera or object rotations, scale change and change in camera position. The disadvantage is, that for different object scales we need to have different kernels. Also, as many other tracking methods, it may converge to a local maxima instead of a global maxima. Success rate of this method depends mainly on the ability to reliably estimate the probability density function. If the model is not sufficiently discriminative, than the tracker will drift and fail. This is of course a common problem of all tracking methods.

2.1.1 Lucas-Kanade vs. Linear Predictors

We explain why we choose to work with predictors rather than Lucas-Kanade algorithm. For sake of simplicity, we represent the object by a rectangular patch and as the individual features we use directly the image intensity values within this patch. Therefore we need only one support set X corresponding to the image patch 2D coordinates. The image patch is usually called *template* $T(\mathbf{x}_i)$, where $\mathbf{x}_i \in X$ is a particular pixel coordinate in 2D. The initial tracking parameters $\mathbf{t}_0$ from the previous image encode the previous object pose. The transformed support set from previous image is $X^{'} = \rho(X, \mathbf{t}_0)$. Now the goal of tracking in the current image is to find the motion parameters vector $\mathbf{t}$ that minimizes

$$\sum_{i=1}^{|X|} \left[I\left(\rho\left(\mathbf{x}_i^{'}, \mathbf{t}\right)\right) - T\left(\mathbf{x}_i\right) \right]^2, \tag{2.1}$$

where $\mathbf{x}_i \in X$ and corresponding $\mathbf{x}_i^{'} \in X^{'}$. The minimization is performed with respect to $\mathbf{t} \in \Omega$. Minimizing the expression (2.1) is a non-linear optimization. To optimize the expression in (2.1), the Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{t}$ is known (in the beginning it is initialized as $\mathbf{t} \leftarrow \mathbf{t}_0$) and then iteratively solves for increments to parameters $\mathbf{\Delta t}$, i.e. the equation (2.1) becomes:

$$\sum_{i=1}^{|X|} \left[I\left(\rho\left(\mathbf{x}_i^{'}, \mathbf{t} + \mathbf{\Delta t}\right)\right) - T\left(\mathbf{x}_i\right) \right]^2 \tag{2.2}$$

and the minimization is performed with respect to $\mathbf{\Delta t} \in \Omega$, which is added to $\mathbf{t}$ after each iteration

$$\mathbf{t} \leftarrow \mathbf{t} + \mathbf{\Delta t}. \tag{2.3}$$

Equations (2.2) and (2.3) are iteratively computed until the norm $\|\mathbf{\Delta t}\| \le \epsilon$, where ϵ is some threshold, or until some number of maximum iterations has been reached. When performing some algebraic manipulation with (2.2) we get a solution for one iteration of the Lucas-Kanade algorithm as follows

$$\mathbf{\Delta t} = \sum_{i=1}^{|X|} \left(\underbrace{\left[\nabla I \frac{\partial \rho}{\partial \mathbf{t}}\right]^T \left[\nabla I \frac{\partial \rho}{\partial \mathbf{t}}\right]}_{\text{hessian}} \right)^{-1} \left[\nabla I \frac{\partial \rho}{\partial \mathbf{t}}\right]^T \left[T\left(\mathbf{x}_i\right) - I\left(\rho\left(\mathbf{x}_i^{'}, \mathbf{t}\right)\right) \right], \tag{2.4}$$

where ∇I denotes the image gradient. One disadvantage of Lucas-Kanade algorithm is that it needs to recompute the image hessian inverse and the jacobian of transformation

$\frac{\partial\rho}{\partial\mathbf{t}}$ in every iteration. The roles of image and template are switched in [23] and therefore the template hessian inverse multiplied by $\left[\nabla T\frac{\partial\rho}{\partial\mathbf{t}}\right]^T$ may be precomputed. But this is only possible for some simple transformations ρ the jacobian of which may be precomputed [23]. In general the image (or template) gradient, inverse hessian and the transformation jacobian need to be recomputed in every iteration. For more detailed Lucas-Kanade algorithm description and derivation, please refer to Appendix C.

In contrast the *predictor* γ solves for $\mathbf{t}$ directly in a closed form. The predictor estimates the parameters by multiplication of a learned regression matrix H and a vector of image intensities as follows

$$\gamma : \mathbf{t} = \mathtt{H}I(X^{'}). \tag{2.5}$$

The regression matrix H needs to be learned from some training samples before the tracking starts. The advantage of linear predictor is that it does not need to iteratively compute the image gradients, inverse hessian or transformation jacobian. On the other hand Lucas-Kanade algorithm does not require any learning stage before the tracking starts.

The predictor learning stage may be advantageous in the case of object generic tracking. A typical example is a face tracking task. The tracker may be trained on large datasets and after that it is able to track every human face and is very robust to changes in appearance. When the predictor is trained it is very efficient during runtime. On the other hand the learning stage is restrictive in case of the need to track some object immediately, when there were no sample images available for training. This case is basically the only disadvantage of predictors when compared to Lucas-Kanade algorithm. In Chapter 4 we deal with this problem of immediate object tracking by linear predictors. Apart from this case, predictors are faster in performance equally precise in parameters estimation and easilly to adapt for new object appearances. In the following chapter we show how to learn the predictor and how to further improve its performance.

2.1.2 Linear Predictors

Linear predictors estimate the tracking parameters directly from observed image features. Such approach requires a supervised learning stage, where pairs of transformation parameters $\mathbf{t}^i$ and corresponding observed intensities $I(X^{'})$ are collected, where $X^{'} = \rho(X, \mathbf{t}^i)$ and a mapping $\gamma : \mathbb{R}^k \rightarrow \mathbb{R}$, which minimizes the sum of squares of errors on training data, is estimated as

$$\gamma^* = \arg\min_{\gamma} \sum_{\mathbf{t}^i \in \Omega} \|\gamma\left(I\left(\rho\left(X, \mathbf{t}^i\right)\right)\right) - \mathbf{t}^i\|^2, \tag{2.6}$$

In the tracking stage, the learned mapping $\gamma^*(I(X^{'}))$ directly estimates motion parameters without the necessity of on-line optimization of the criterial function. One can replace the pseudoinverse operation (see equation (2.4)) by matrix H learned on a set of synthesized examples. Mapping γ then transforms to a linear function between intensities $I\left(X'\right)$ and transformation parameters $\mathbf{t}$,

$$\mathbf{t} = \gamma^*\left(I\left(X'\right)\right) = \mathtt{H}I\left(X'\right). \tag{2.7}$$

In the tracking procedure the transformation parameters $\mathbf{t}$ are simply computed as a linear function of the object intensities Suppose, that the coordinates in X are perfectly aligned on the current patch position in the image and vector $\mathbf{t}$ contains parameters for identity transformation. With m small random perturbations of parameters in $\mathbf{t}$ on some predefined *ranges* we get the matrix $\mathtt{T} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_m]$. The ranges specify the magnitude of allowed parameters differences on which we expect the predictor to work (in a single estimation step). Next the matrix $\mathtt{L}$ with image intensity values (stored columnwise with matrix $\mathtt{T}$) is created as $\mathtt{L} = [I(\rho(X, \mathbf{t}_1)), I(\rho(X, \mathbf{t}_2)), \ldots, I(\rho(X, \mathbf{t}_m))]$. Than the learning task may be formulated as

$$\begin{aligned} \mathtt{H}^* &= \arg\min_{\mathtt{H}} \|\mathtt{HL} - \mathtt{T}\|_F^2 & (2.8) \\ &= \arg\min_{\mathtt{H}} \operatorname{trace}\left(\mathtt{HL} - \mathtt{T}\right)\left(\mathtt{HL} - \mathtt{T}\right)^T & (2.9) \\ &= \arg\min_{\mathtt{H}} \operatorname{trace}\left(\mathtt{HLL}^T\mathtt{H}^T - 2\mathtt{HLT}^T + \mathtt{TT}^T\right). & (2.10) \end{aligned}$$

We take the derivative of (2.10) with respect to $\mathtt{H}$ and set it equal to zero

$$\begin{aligned} 2\mathtt{H}^*\mathtt{LL}^T - 2\mathtt{TL}^T &= 0 & (2.11) \\ \mathtt{H}^*\mathtt{LL}^T &= \mathtt{TL}^T & (2.12) \\ \mathtt{H}^* &= \mathtt{TL}^T\left(\mathtt{LL}^T\right)^{-1} & (2.13) \\ \mathtt{H}^* &= \mathtt{TL}^+. & (2.14) \end{aligned}$$

The training set construction is demonstrated in Figure 2.1. The parameter estimation procedure is extremly fast and has been used by numerous authors. For example Cootes et al. [13] use linear predictors for human face model parameters estimation. The model is composed from deformable shape model and appearance model. All parameters for shape and appearance are estimated by one regression function. Their approach proved to be very robust for tracking non-rigid objects, such as human faces [13].

2.1.3 Increasing the Precision and Complexity Optimization

The *complexity* C of a predictor γ is represented by the number of used feature values. The more complex is the regression function, the more precise parameters estimation is achievable, but more computational time is consumed. We may see, that computational complexity may be lowered by lowering the number of used template pixels. But how to select the best subset of pixels $X^* \subseteq X$ (for the given complexity C lower than size of X) which minimizes the error on the training data? A heuristic algorithm was

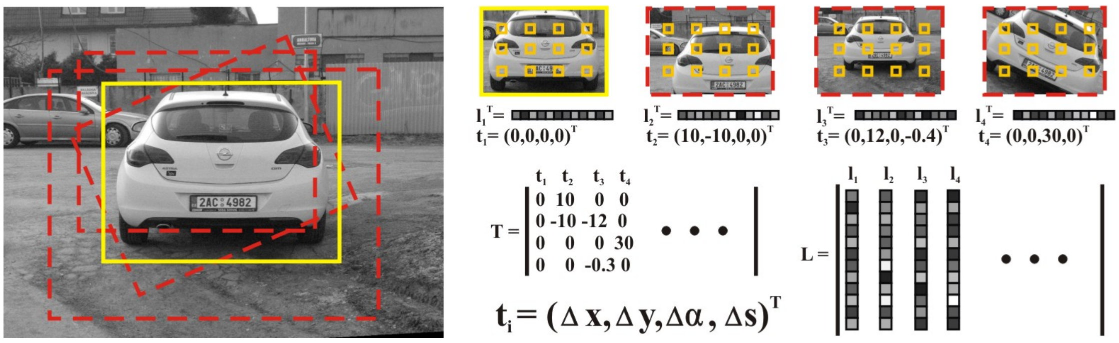


Figure 2.1: Creation of training matrices by random perturbations of the training image. The ground truth position in the left image is given by the yellow rectangle. For example to train a linear predictor which estimates 2D motion $\Delta x, \Delta y$, in-plane rotation $\Delta\alpha$ and scale change Δs we multiply transform the original image with known motion parameters. In each sample we collect the image features in support set points (small yellow rectangles) and store them in a vector $\mathbf{l}_i$. Vectors $\mathbf{l}_i$ and known parameters vectors $\mathbf{t}_i$ form training matrices $\mathtt{L}$ and $\mathtt{T}$.

proposed in [11], see Appendix B. For large ranges, it is usually very difficult to achieve a good precision of prediction even with the complexity corresponding to the size of the whole object patch [28]. A *sequential predictor* γ_s is proposed in [11], where it is experimentally shown, that γ_s with lower complexity achieves higher precision than γ with higher complexity. γ_s was also used in [14] for human face tracking.

The γ_s is in fact a sequence of predictors, which estimate warp parameters one after another thus improving the result of previous predictor estimation and lowering the overall error of estimation. γ_s tracks according to Equation 2.15

$$\begin{aligned}
\mathbf{t}^{(1)} &= \mathtt{H}^{(1)} I\left(X^{(1)}\right) \\
\mathbf{t}^{(2)} &= \mathtt{H}^{(2)} I\left(\rho\left(X^{(2)}, \mathbf{t}^{(1)}\right)\right) \\
\mathbf{t}^{(3)} &= \mathtt{H}^{(3)} I\left(\rho\left(X^{(3)}, \mathbf{t}^{(1)} \circ \mathbf{t}^{(2)}\right)\right) \\
&\vdots \\
\mathbf{t}^{(k)} &= \mathtt{H}^{(k)} I\left(\rho\left(X^{(k)}, \mathbf{t}^{(1)} \circ \cdots \circ \mathbf{t}^{(k-1)}\right)\right),
\end{aligned} \tag{2.15}$$

where the final warp parameters vector is

$$\mathbf{t} = \mathbf{t}^{(1)} \circ \mathbf{t}^{(2)} \circ \cdots \circ \mathbf{t}^{(k)} \tag{2.16}$$

and the operator $\circ$ is a composition of transformation parameters vectors. The *complexity of* γ_s is simply the sum of cardinalities of particular support sets in the sequence.

The *sequential learnable linear predictor* is than defined as

$$\gamma_s = |\{\mathtt{H}^{(1)}, X^{(1)}\}, \{\mathtt{H}^{(2)}, X^{(2)}\}, \ldots, \{\mathtt{H}^{(k)}, X^{(k)}\}| \tag{2.17}$$

for k predictors in the sequence. To see how the γ_s is learned, we need to introduce the *uncertainty region* of the predictor γ, as it is called in [11][28]. The uncertainty region is the smallest p-dimensional region Γ (for p parameters in vector $\mathbf{t}$) within which all the prediction errors from the ranges lie:

$$\Gamma(\gamma) = \{\Delta\mathbf{t} | \Delta\mathbf{t} = \mathbf{t} \circ^{-1} \gamma(I(\rho(X, \mathbf{t}))), \forall \mathbf{t} \in R(\gamma)\}, \tag{2.18}$$

where $R(\gamma)$ is the p-dimensional range of predictor γ and $\Delta\mathbf{t}$ is the error vector. Now we want to show how is the γ_s learned. The first predictor is trained to work on the original range specified by the user. The predicotr, which is second in the sequence has got new range of allowed parameters changes, which is equal to the uncertainty region Λ of the preceding predictor. It was shown in [11], that range given by the uncertainty region is inside of (thus smaller than) the original range. The support sets of predictors in the sequence are independent and may differ. Typically a predictor which is subsequent in the sequence needs less pixels in the support set X (less features to evaluate), because it needs to operate on a smaller range than the preceding predictor.

An alternative approach to least-squares learning of the regression matrix has been also proposed in [11]. The authors used *minmax* learning method to learn an optimal sequence of predictors. The learning is composed from two steps. First a set of predictors is learned by minmax optimization and then a sequence of predictors creating an optimal γ_s is selected. The selection of optimal sequence of predictors is formulated as finding the cheapest path in the graph and is solved using the Dijkstra algorithm.

The varying object appearance usually lowers the precision of tracking parameters estimation. An interesting approach for updating the regression matrix and adapt it for varying object appearance was proposed in [28]. See Appendix A for details.

2.1.4 Incremental On-line Learning and Adaptive Tracking

Jepson et al. in [29] proposed WSL tracker - an *adaptive* appearance model for object tracking, which deals with partial occlusion and change in object appearance. The image appearance is expressed in terms of the complex-valued coefficients of a steerable pyramid. It is a wavelet-based model, which allows to maintain a natural measure of the *stability* of the observed image structure during tracking. Stable properties of appearance are weighted more heavily for motion estimation, while unstable properties are proportionately downweighted. The generative model for appearance is formulated as a mixture of three components: a stable component that is learned with a relatively long time-course, a two-frame transient component, and an outlier process. The stable model identifies the most reliable structure for motion estimation, while the two-frame constraints provide additional information when the appearance model is being initialized or provides relatively few constraints. The parameters of the model are learned efficiently with an on-line version of the EM algorithm. This approach is robust and works well with slowly changing object appearance, but with a high computational overhead and may be hardly used for real-time tracking.

One of the current issues of tracking is the choise of object *generic* or object *specific* appearance (or even shape) models for tracking. *Generic* model should be able to track all objects of some class (cars, faces, books, etc), which requires usually large training datasets to cover most of the possible appearances for particular class. Object *specific* model is trained to track particular object instance of some class. Object specific model requires less training data and may be even trained from the more general generic model [30][31]. One of the comparisons is given by Gross et al. in [32]. The authors compare the performance of person generic with person specific active appearance models (AAMs). AAMs [13] are generative parametric models commonly used to model faces. Depending on the task at hand AAMs can be constructed in different ways. For example, we might build an AAM to model the variation in appearance of a single person across pose, illumination and expression. Such a person specific AAM might be useful for interactive user interface applications that involve head pose estimation, gaze estimation, or expression recognition. Alternatively, we might attempt to build an AAM to model any face, including unseen subjects which were not seen in the training set. The most common use of such a generic AAM would be face recognition. Gross et al. [32] argue that Person Specific AAMs perform substantially better than generic AAMs. Lee et al. [31] proposed method for on-line learning of person specific probabilistic appearance manifold when using the prior off-line learned person generic model. Once the person specific model was built on-line, the algorithm may use it for more accurate and robust face tracking. The advantage of this approach is, that appearance sub-manifolds may be used for face recognition as weel as for tracking. Method tracks well under difficult conditions and handles well the sudden change in appearance. The main disadvantage is, that we first need prior off-line learned generic model for one object class (e.g. human faces).

For template-based trackers the adaptation means continuous update of the tracked template. Tracking systems with *naive updates* update the template after every tracking step [9]. Sub-pixel errors inherent to each match are stored in each update and these errors gradually accumulate resulting in the template drifting off the feature, but usually naive update is better than no update. Matthews et al. in [30] propose a *strategic update* approach, which trades off mismatch error and drift. It is a simple and effective extension of the naive update. There are two template models kept during the tracking. The *updating template* is used for an initial alignment and the template from the first frame is than used in the error correction phase after alinment. If the size of correction is too large, the algorithm acts conservatively by preventing the updating template to be updated from the current frame. Matthews et al. also show that this template update can be extended to templates with linear appearance variation (derived in [33]). It is shown in [30], that this method for *template update* may be used to convert the *person generic* AAM to *person specific* AAM. This conversion proved to be very useful, because there is far less appearance variation in the person specific AAM and therefore it requires far fewer appearance parameters to provide the same representational power.

Lately some authors want to get rid of the exhaustive off-line learning stage, which usually consumes a lot of time and needs large labeled datasets to work with. Purely on-line learning has been proposed by Ellis et al. in [34], where bank of displacement linear predictors spatially localised over the object are on-line learned and the appearance model of the object is learned on-the-fly by clustering sub-sampled image templates. The templates are clustered using the medoidshift algorithm. The clusters of appearance templates allow to identify different views or aspects of the target and also allow to choose the bank of predictors most suitable for current appearance. The algorithm also evaluates the performance of particular predictors. When the performance of some predictor is too low, it is thrown away and new predictor is learned on-line to replace it. Similar approach was proposed by Ellis et al. in [35] with different appearance model. A probabilistic model of appearance is incrementally constructed by partitioning templates into components. The model is represented as a weighted mixture of components. Again the partitioning represents views or aspects of the target and is used to choose the right bank of predictors for tracking. Both approaches [34][35] seem to be robust and outperform the Grabners discriminative tracker [36] and Dowson's simultaneous modeling and tracking (SMAT) algorithm [37]. Also the TLD tracker proposed in [24], already mentioned in the beginning of section 2.1, is learned from a single image and during tracking it collects new samples of object appearance and both the detector and tracker are incrementally learned.

2.1.5 Incremental Learning of Predictors

It is not possible to train the predictor γ for all object appearances in advance before tracking. It would be useful to have a method for incremental learning of γ for unseen object appearances. For incremental learning using equation (2.11) we would need to keep both training matrices $\mathtt{L}, \mathtt{T}$ in memory and they would grow with adding new training examples. But the main difficulty would be the pseudoinverse computation of growing matrix $\mathtt{L}$. To avoid this, we may use a method for incremental least-squares update of the matrix $\mathtt{H}$, which has been proposed by Hinterstoisser et al. in [12]. Let us suppose, that we have regression matrix $\mathtt{H}_m$, which was trained using m training examples. The authors of [12] introduce matrices $\mathtt{Y}_m = \mathtt{T}_m\mathtt{L}_m^T$ and $\mathtt{Z} = \mathtt{L}_m\mathtt{L}_m^T$ and matrix $\mathtt{H}_{m+1}$ may be written as

$$\begin{aligned}
\mathtt{H}_{m+1} &= \mathtt{Y}_{m+1}\mathtt{Z}_{m+1}^{-1} \qquad (2.19)\\
&= \mathtt{T}_{m+1}\mathtt{L}_{m+1}^T\left(\mathtt{L}_{m+1}\mathtt{L}_{m+1}^T\right)^{-1}\\
&= \left[\mathtt{T}_m|\mathbf{t}_{m+1}\right]\left[\mathtt{L}_m|\mathbf{l}_{m+1}\right]^T\left(\left[\mathtt{L}_m|\mathbf{l}_{m+1}\right]\left[\mathtt{L}_m|\mathbf{l}_{m+1}\right]^T\right)^{-1}\\
&= \left(\mathtt{T}_m\mathtt{L}_m^T + \mathbf{t}_{m+1}\mathbf{l}_{m+1}^T\right)\left(\mathtt{L}_m\mathtt{L}_m^T + \mathbf{l}_{m+1}\mathbf{l}_{m+1}^T\right)^{-1}\\
&= \left(\mathtt{Y}_m + \mathbf{t}_{m+1}\mathbf{l}_{m+1}^T\right)\left(\mathtt{Z}_m + \mathbf{l}_{m+1}\mathbf{l}_{m+1}^T\right)^{-1}
\end{aligned}$$

where $\mathbf{t}_{m+1}$ and $\mathbf{l}_{m+1}$ are concatenated to $\mathtt{T}_m$ and $\mathtt{L}_m$, respectively to form $\mathtt{T}_{m+1}$ and $\mathtt{L}_{m+1}$. Thus only by storing the *constant size* matrices $\mathtt{Y}_m$ and $\mathtt{Z}_m$ and updating them as

$$\mathtt{Y}_{m+1} \leftarrow \mathtt{Y}_m + \mathbf{t}_{m+1}\mathbf{l}_{m+1}^T \tag{2.20}$$

$$\mathtt{Z}_{m+1} \leftarrow \mathtt{Z}_m + \mathbf{l}_{m+1}\mathbf{l}_{m+1}^T, \tag{2.21}$$

it becomes possible to incrementally learn the predictors without the need to store all training examples. Since the computation of $\mathtt{H}$ has to be done for many locations in each incoming image and matrix $\mathtt{Z}_m$ is usually large, it is necessary to avoid the computation of the inverse $\mathtt{Z}_m^{-1}$ for every new training example. Sherman-Morrison formula is applied to $\mathtt{Z}_{m+1}^{-1}$ and we obtain

$$\mathtt{Z}_{m+1}^{-1} = \left(\mathtt{Z}_m + \mathbf{l}_{m+1}\mathbf{l}_{m+1}^T\right)^{-1} \tag{2.22}$$

$$= \mathtt{Z}_m^{-1} - \frac{\mathtt{Z}_m^{-1}\mathbf{l}_{m+1}\mathbf{l}_{m+1}^T\mathtt{Z}_m^{-1}}{1 + \mathbf{l}_{m+1}^T\mathtt{Z}_m^{-1}\mathbf{l}_{m+1}}. \tag{2.23}$$

Therefore we need to store $\mathtt{Z}_m^{-1}$ instead of $\mathtt{Z}_m$. Than incremental learning of one new training example means updating matrices $\mathtt{Y}_m$ and $\mathtt{Z}_m^{-1}$ using equations (2.20) and (2.22) respectively and than updating the regression matrix $\mathtt{H}$ using equation (2.19). This approach was used in [12] for incremental learning of new patch appearances, while the predictor estimates the full perspective transformation parameters.

Incremental learning is recommendable for every tracking system, which has the ability to do so. The only problem is, how to choose the aditional training examples automatically? New examples could be of course added manually. The disadvantage of supervised incremental learning is the need of human interaction.

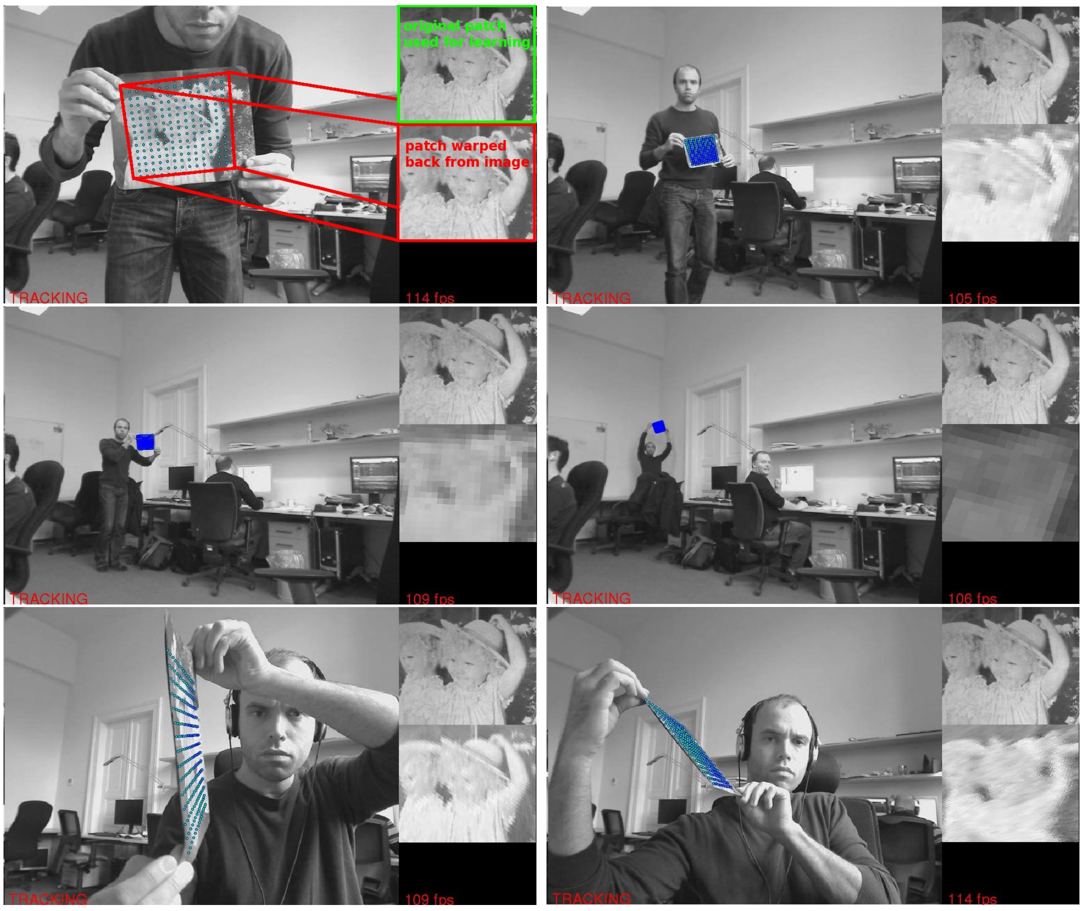


Figure 2.2: Here we see another result of our work. The tracker learned from a single image is marked by green rectangle. There is also the tracked patch warped back placed under the original patch for visual comparison (marked by red rectangle). Thanks to the incremental learning running during tracking the tracker is able to handle acute angle changes and signifficant changes in appearance caused by lower resolution of the tracked patch.

3 Automatic Selection of Training Examples for Incremental Learning

Learnable visual trackers have recently proved their wide applicability in object tracking in video. The tracking poses essentially two main challenges: i) adapting to changing appearance, ii) detecting tracker failure – loss of track. This chapter addresses both issues but contributes mainly to the adaptation problem. We propose to solve the adaptation problem by an incremental learning, which accommodates changing appearance whilst tracking. We also suggest a fast method for tracking validation (i.e. loss-of-track detection) which uses the same model as for tracking and does not need any additional object model. The predictor needs only a very short (seconds) offline learning stage before the tracking starts. The tracking itself is then tremendously efficient, much faster than real-time.

Tracker adaptation and loss-of-track detection have been active topics for many years. Jepson et al. [29] proposed WSL tracker (3 components - Wandering, Stable and Lost) - an *adaptive* appearance model which deals with partial occlusion and change in object appearance. It is a wavelet-based model, which allows to maintain a natural measure of the *stability* of the observed image structure during tracking. This approach is robust and works well with slowly changing object appearance. However, a high computational overhead precludes real-time applications. Lim et al. [38] propose an algorithm for incremental learning and adaptation of low dimensional eigenspace object representation with update of the sample mean and eigenbasis. Their approach appears to be robust to sudden illumination changes and does not need offline learning phase before tracking however, the algorithm speed does not fit our needs.

For template-based trackers the adaptation means continuous update of the tracked template. Tracking systems with *naive updates* update the template after every tracking step [9]. Sub-pixel errors inherent to each match are stored in each update and these errors gradually accumulate resulting in the template drifting off the feature. Despite this drawback, naive update is still usually better choice than no update at all. Matthews et al. in [30] propose a *strategic update* approach, which trades off mismatch error and drift. It is a simple but effective extension of the naive update. There are two template models kept during the tracking. The *updating template* is used for an initial alignment and the template from the first frame is than used in the error correction phase after alignment. If the size of correction is too large, the algorithm acts conservatively by preventing the template to be updated from the current frame.

Recently, some authors wanted to bypass an exhaustive off-line learning stage. Purely on-line learning has been proposed by Ellis et al. in [34], where a bank of local linear

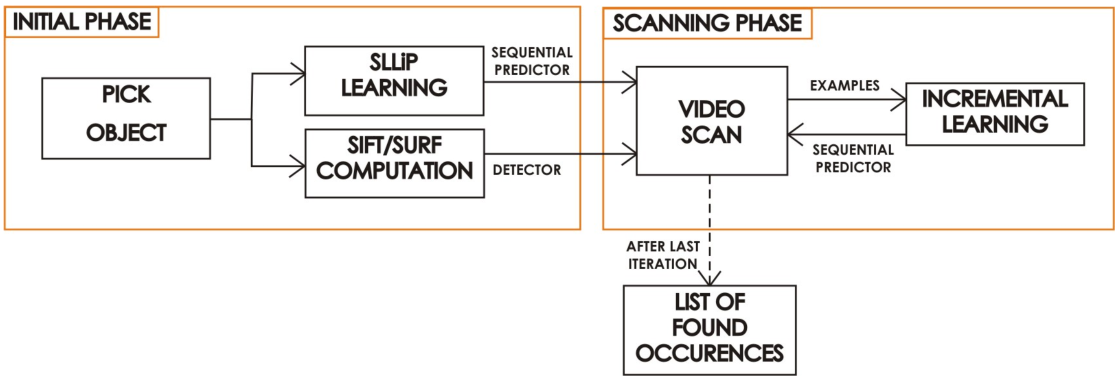


Figure 3.1: Video browsing procedure.

predictors, spatially disposed over the object, are on-line learned and the appearance model of the object is learnt on-the-fly by clustering sub-sampled image templates. The templates are clustered using the medoidshift algorithm. The clusters of appearance templates allow to identify different views or aspects of the target and also allow to choose the bank of predictors most suitable for current appearance. The algorithm also evaluates the performance of particular predistors. When the performance of some predictor is too low, it is discarded and a new predictor is learned on-line as a replacement. In comparison to our work, we do not throw away the predictors in sequence, but we incrementally train them with new object appearances in order to improve their performance.

Our learnable and adaptive tracking method, coupled with a sparsely applied SIFT [19] or SURF [20] based detector, is applied for faster than real-time linear video browsing. The goal is to find all object occurrences in a movie. One of possible solutions of video browsing task would be to use a general object detector in every frame. As it appears [39], [40], it is preferable to use a combination of an object detector and a tracker in order to speed up the browsing algorithm and also to increase the true positive detections. We indeed aim at processing rates higher than real-time which would allow almost interactive processing of lengthy videos. Our yet preliminary Matlab implementation can search through videos up to eight times faster than the real video frame rate.

3.1 Learning, Tracking, Validation and Incremental Learning

User initiates the whole process by selecting a rectangular patch with the object of interest in one image. This sample patch is artificially perturbed and a sequential predictor is learned [41]. Computation of a few SIFT or SURF object descriptors completes the initial phase of the algorithm, see Figure 3.1. The scanning phase of algorithm combines predictor based tracking, its validation, and a sparse object detection. The predictor is incrementally re-trained for new object appearances. Examples for the incremental learning are selected automatically with no user interaction.

The scanning phase starts with the object detection running every $n-$th frame (typ-

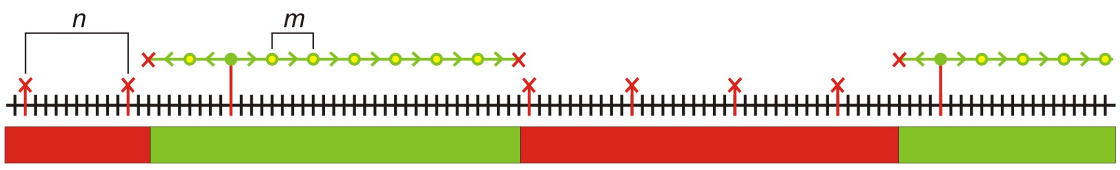


Figure 3.2: A typical video scan process. Vertical red lines depict frames, where the object detection was run. Red cross means negative detection or tracking failure. Green line shows backward and forward object tracking. Green circle means positive object detection and yellow circle depicts successful validation.

ically with the step of 20 frames) until the first object location is found. The tracker starts from this frame on the detected position both in backward and forward directions. Backward tracking scans frames which were skipped during the detection phase and runs until the loss-of-track or until it reaches the frame with last found occurrence of the object. Forward tracking runs until the loss-of-track or end of sequence. The detector starts again once the track is lost. Tracking itself is validated every $m-$th frame (typically every 10 frames). The scanning procedure is depicted on Figure 3.2.

One object sample represents only one object appearance. The predictor is incrementally re-trained as more examples become available from the scanning procedure. The next iteration naturally scans only images where the object was not tracked in the preceding iterations.

Training examples for incremental learning are selected automatically. The most problematic images-examples are actually the most useful for incremental training of the predictor. In order to evaluate the usefulness of a particular example we suggest a stability measure. The measure is based on few extra predictions of the predictor on a single frame. It means, that we let the sequential predictor track the object in a single static image and we observe the predictors' behavior. See Section 3.1.3 for more details.

The sequential linear predictor validates itself. Naturally, an object detector may be also used to validate the tracking. For example well trained face detector will do the same or better job when used to validate human face tracking. Motivation for using the sequential predictor for validation is its extreme efficiency, and robust performance. For more details about the tracking validation, see section 3.1.2.

3.1.1 Incremental Learning of Sequential Linear Predictor

We extend the sequential linear predictors γ_s proposed by Zimmermann et al. [41] in order to predict not only translation but also the affine deformation of the object. Next extension is the incremental learning of new object appearances. The predictor essentially estimates motion and deformation parameters directly from image intensities. It requires an offline learning stage before the tracking starts. The learning stage consists of generating exemplars and estimation of regression functions. We use two γ_s - first for 2D motion estimation (2 parameters) and second for affine warp estimation (4 parameters). We have experimentally verified that, especially for low number of training examples, this configuration is more stable than using just one γ_s to predict all 6 parameters at once. Using smaller training set decreases the necessary learning time which is important for the foreseen applications. Because of speed we opted for least squares learning of sequential predictors similarly as suggested by Zimmermann et al. in their any-time learning algorithm [41].

Let denote the translation parameters vector $\mathbf{t}_t = (\Delta x, \Delta y)^T$ estimated by the first sequential predictor, and the affine warp is parameterized by the parameters vector $\mathbf{t}_a = (\alpha, \beta, \Delta s_x, \Delta s_y)^T$ which is estimated by the second sequential predictor. The affine transformation $A(\mathbf{t}_a)$ parameterized by vector $\mathbf{t}_a$ is defined by a 2×2 matrix $\mathtt{A}$ computed as follows

$$\mathtt{A} = \mathtt{R}_\alpha \mathtt{R}_{-\beta} \mathtt{S} \mathtt{R}_\beta \ , \tag{3.1}$$

where $\mathtt{R}$ are standard 2D rotation matrices parameterized by the angles α, β and $\mathtt{S}$ is the scale matrix

$$\mathtt{S} = \begin{bmatrix} 1 + \Delta s_x & 0 \\ 0 & 1 + \Delta s_y \end{bmatrix} . \tag{3.2}$$

Than the image point $\mathbf{x} = [x, y]^T$ is transformed between two consecutive images using estimated parameters accordingly

$$\begin{aligned} \mathbf{x}' &= \mathtt{A}\mathbf{x} + \mathbf{t}_t \\ &= \mathtt{R}_\alpha \mathtt{R}_{-\beta} \mathtt{S} \mathtt{R}_\beta \mathbf{x} + \mathbf{t}_t, \end{aligned} \tag{3.3}$$

Tracking, learning and incremental learning will be explained for sequential predictor γ_s with general parameters vector $\mathbf{t}$. Equations are valid for both sequential predictors, which we use. Predictors in sequence estimate the parameters one after each other, thus each improving the result of previous predictor estimation and lowering the error of estimation. γ_s tracks according to Equation (2.15), let us rewrite this equation here

$$\begin{aligned} \mathbf{t}^{(1)} &= \mathtt{H}^{(1)} I\left(X^{(1)}\right) \\ \mathbf{t}^{(2)} &= \mathtt{H}^{(2)} I\left(\rho\left(X^{(2)}, \mathbf{t}^{(1)}\right)\right) \\ \mathbf{t}^{(3)} &= \mathtt{H}^{(3)} I\left(\rho\left(X^{(3)}, \mathbf{t}^{(1)} \circ \mathbf{t}^{(2)}\right)\right) \\ &\vdots \\ \mathbf{t}^{(k)} &= \mathtt{H}^{(k)} I\left(\rho\left(X^{(k)}, \mathbf{t}^{(1)} \circ \cdots \circ \mathbf{t}^{(k-1)}\right)\right), \end{aligned} \tag{3.4}$$

In this context the transformation ρ corresponds to Equation 3.3 and is parameterized by both vectors $\mathbf{t}_a$ and $\mathbf{t}_t$. $I(X)$ is a vector of image intensities collected at image coordinates X. The final result of prediction is the vector $\mathbf{t}$ which combines results of all predictions in the sequence. Operation $\circ$ means composition of two consecutive vectors with tracking parameters. Two affine parameters vectors $\mathbf{t}_a^{(i-1)}, \mathbf{t}_a^{(i)}$

are composed to one using $\circ$ as $A\left(\mathbf{t}_a^{(i-1)}\right)\mathbf{t}_a^{(i)}$ and two translation vectors $\mathbf{t}_t^{(i-1)}, \mathbf{t}_t^{(i)}$ are composed as $A\left(\mathbf{t}_a^{(i-1)}\right)\mathbf{t}_t^{(i)} + \mathbf{t}_t^{(i-1)}$. The model of a sequential predictor γ_s is $\gamma_s = \left|\left\{\mathtt{H}^{(1)}, X^{(1)}\right\}, \left\{\mathtt{H}^{(2)}, X^{(2)}\right\}, \ldots, \left\{\mathtt{H}^{(k)}, X^{(k)}\right\}\right|$. Matrices $\mathtt{H}^{(i)}$ are linear regression matrices which are learned from training data.

In our algorithm, each γ_s is learned from one image only and it is incrementally relearned after each video scan. A few thousands of training examples are artificially generated from the first image using random perturbations of parameters in vector $\mathbf{t}$, transforming the support set accordingly and collecting the image intensities. Each regression matrix in the sequential predictor is trained using the least squares method defined earlier in Equation 2.19. The initial learning phase takes 5 or 6 seconds on a standard PC.

More images (around 400) are selected for incremental learning from all images gathered during last scanning iteration. From each of the additional exemplars 10 training examples are generated. This procedure provides additional 4000 training examples after each video scan iteration. It is worth to note that this process is completely automatic, no user interaction is required. Incremental learning comprises update of regression matrices $\mathtt{H}^i$ according to Equations (2.19), (2.20) and (2.23).

3.1.2 Validation by Voting

To validate the tracking (i.e. detecting loss-of-track) we use the same sequential linear predictor as for tracking. We utilize the fact that the predictor is trained to point to the center of this object when initialized in a close neighborhood. On the contrary, when initialized on the background, the estimation of 2D motion is expected to be random.

We initialize the predictor several times on a regular grid (validation grid - depicted by red crosses in Figure 3.3) in the close neighborhood of current position of the tracker. The close neighborhood is defined as 2D motion range, for which the predictor was trained. In our case the range is $\pm(patch\_width/4)$ and $\pm(patch\_height/4)$. The validation grid is deformed according to estimated parameters. Then we observe the 2D vectors, which should point to the center of the object, i.e. current tracker position in the image. When all (or sufficient number of) the vectors point to the same pixel, which is also the current tracker position, we consider the tracker to be on its track. Otherwise, when the 2D vectors are pointing to some random directions, we say that the track is lost, see Figure 3.3.

A threshold value is needed in order to recognize if the sum of votes, which point to the center of object, is big enough to pass the validation. The threshold is set automatically from examples collected during the video scan. At first iteration, when no threshold is available, first few (tens) validations are done by the object detector and γ_s simultaneously. When the detector votes for positive validation, also the current sum of votes is taken as positive example. Negative examples (sums of votes) are collected by placing the validation grid on other parts of the image, where the object does not appear. Gaussian distributions are fitted into positive and negative examples and the

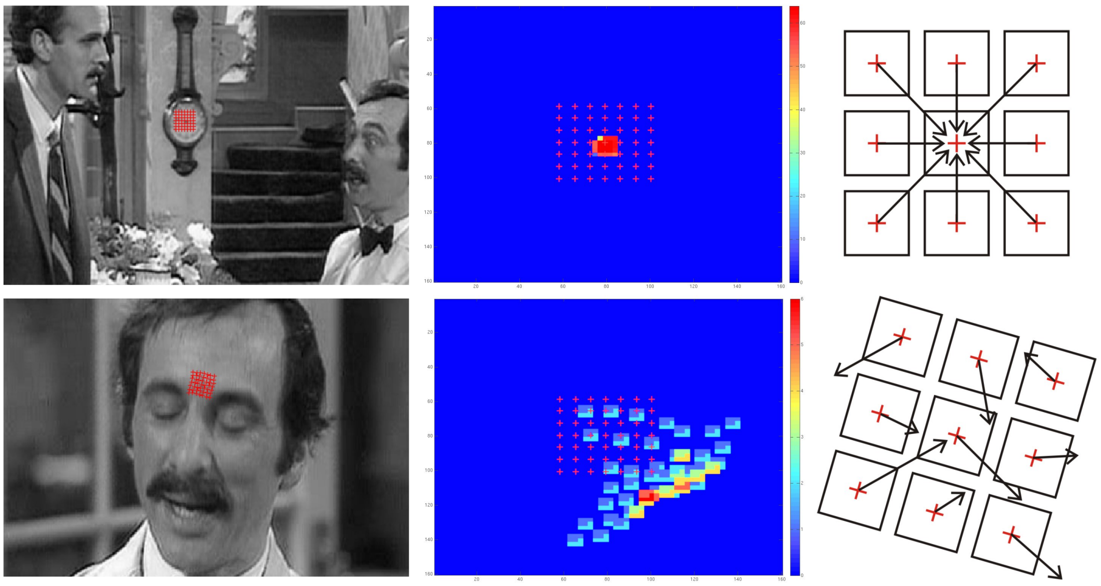


Figure 3.3: Example of predictor validation. The first row shows successful validation of clock tracking. Second row shows loss-of-track caused by a sudden scene change just after a video cut. Red crosses depict pixels, where the predictor was initialized - validation grid. Right column of pictures illustrates the idea of validation using linear predictors and the middle column shows the collected votes for the center of the object in normalized space.

classical Bayes threshold is found. Both negative and positive cases are considered to appear equally likely. In subsequent iterations, the additional training examples are also used for threshold update.

3.1.3 Stability Measure and Examples Selection for the Incremental Learning

Selecting only *relevant* examples for training may speed up the learning as well as increase the performance. Clearly relevant examples are those which contain the object but were not included in the previous training examples. The predictor has of course problems to handle new object appearances and it is likely, that it will loose the track. It is reasonable to presume, that these new difficult (and useful) examples should appear near frames, where the loss-of-track was detected. We need to examine the object occurrences, which appeared near loss-of-track frames, in order to capture the most interesting examples for incremental learning. We propose the *stability measure* for evaluation of these object occurrences.

When we let the predictor track object on a single frame, we would expect the tracker to stay still in objects' position with no additional change of parameters. However, due to

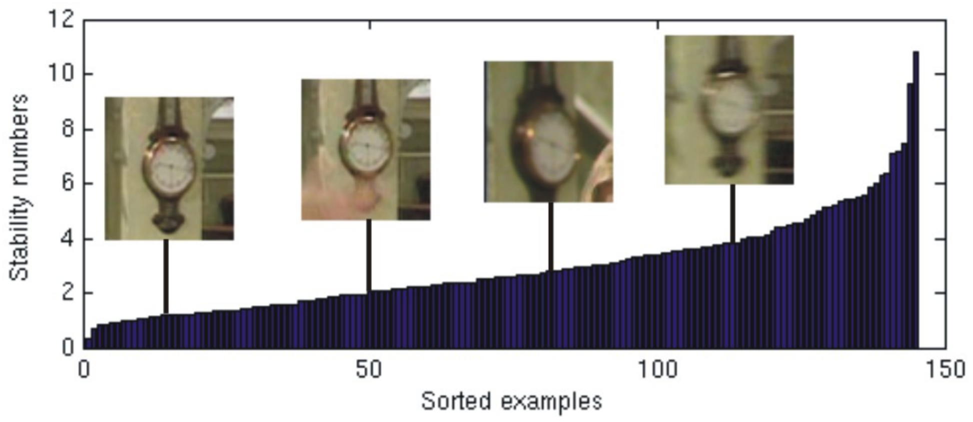


Figure 3.4: Blue bars depict sorted stability numbers. The left most clock image was used for predictor training. The other occurrences obtained during tracking were automatically evaluated as more difficult examples for the tracker. Clearly, the higher stability measure, the more difficult case for the predictor.

inherent noise in the data the predictor predicts non-zero parameters even when initiated on the correct position. The parameters changes are accumulated and their sum-of-squares is computed after 10 tracking steps. Let $\mathbf{t}$ be the vector of parameters estimated during tracking and $\mathbf{p}_i$ vector of parameters obtained in $i-$th step of this single frame tracking. The *stability number* s for current frame is computed as $s = \sum_{i=1}^{10} \| \mathbf{t} - \mathbf{p}_i \|^2$. Clearly, the higher value the more difficult example, see Figure 3.4. Parameters changes in both vectors are made relative to particular ranges, in order to obtain stability number, which is not dependent on different parameters units. Using this stability number we may evaluate how useful (difficult) is the examined object occurrence.

We use this stability number to select a fixed number of additional training examples from each interval obtained during one video scan. Each interval is a continuous subsequence of images from the whole video sequence (one interval is depicted as a green line in Figure 3.2).

We search for the best additional training examples near the borders of each interval. We go through fixed number of images from the start of the interval forwards and backwards from the end of the interval, while computing the stability number on tracker positions. Finally, the algorithm selects the examples with high stability number for incremental learning. Tracker positions in these images have also passed validation and we expect them to be well aligned to the object. The procedure of examples selection is depicted in Figure 3.5.

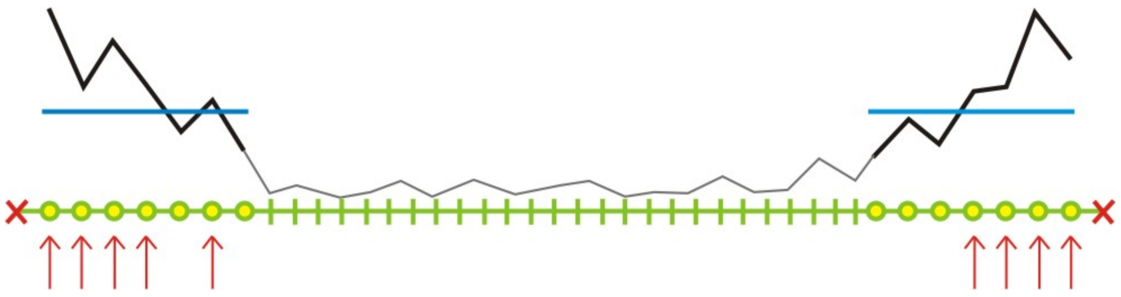


Figure 3.5: Illustration of examples selection for incremental learning. Green line depicts one interval - subsequence of video frames, where the object was found during scanning procedure. Only few images near the beginning and end of the interval are examined. Yellow circles mean successful validation. The black curve depicts computed stability measure on particular frames. The examples with stability number above the blue line are considered as useful for incremental learning. Selected examples are marked by red arrows.

3.2 Experiments

Real sequences used in experiments includes an episode from Fawlty Towers series (33 minutes, 720×576), and Groundhog Day movie (1 hour 37 minutes, 640×384). Several objects were tested, see Figure 3.6. The ground truth data for the Groundhog Day were kindly provided by Josef Šivic and they are the same as in [42]. We have manually labeled ground truth for two tested objects in Fawlty Towers. Third tested sequence captures a human moving in front of the camera (2 minutes 50 seconds, 640×480), see Figure 3.7. Matlab implementation of the algorithm was used for all experiments. SIFT and SURF object detectors are publicly available MEX implementations. Mostly, the standard *precision* and *recall* are used to evaluate the results. Let TP denote the *true positives*, FP denote the *false positives* and FN denote the *false negatives*. Than the precision and recall are computed accordingly

$$precision = \frac{TP}{TP+FP} \tag{3.5}$$

$$recall = \frac{TP}{TP+FN}. \tag{3.6}$$

The experiments are organized as follows. The first experiment (Section 3.2.1) shows the effect of incremental learning on resulting precision and recall. In Section 3.2.2 we evaluate the overall performance of the algorithm. Next we compare tracking validation by SIFT and by γ_s. Finally Table 3.3, Table 3.4 and Table 3.5 show comparison of SIFT detection in every frame with one iteration of our algorithm.

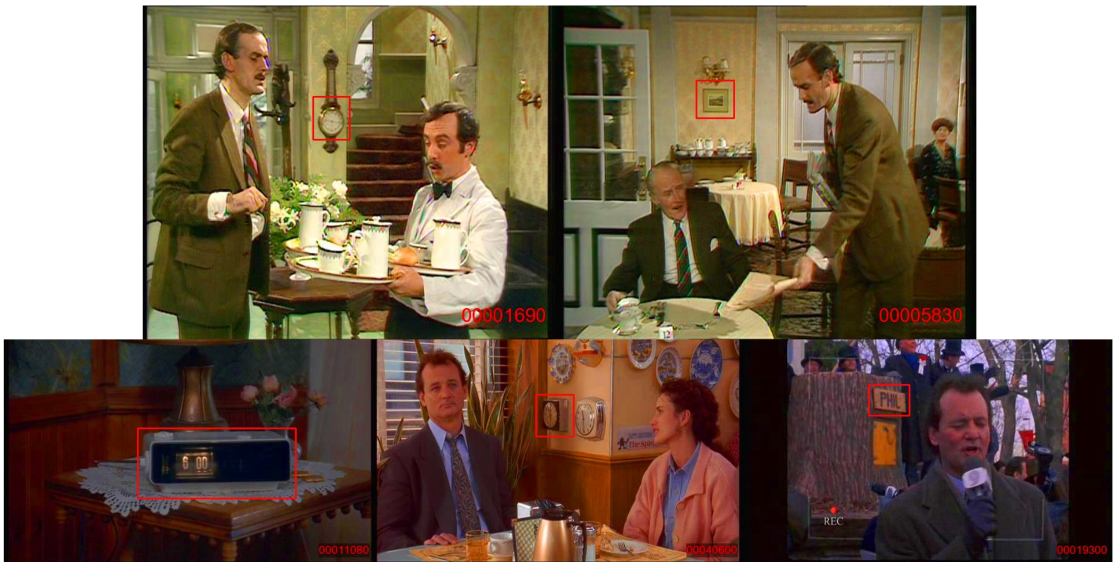


Figure 3.6: First row of pictures shows 2 tested objects from Fawlty Towers series and second row shows 3 tested objects from Groundhog Day movie.

	precision	recall	cumulative time
iter_0	0.86	0.61	13 min 42 sec
iter_1	0.81	0.63	23 min 18 sec
iter_2	0.81	0.64	32 min 21 sec

Table 3.1: Incremental learning evaluation for the clock object from the Fawlty towers episode. The video scan was running 76 frames per second in average.

3.2.1 Incremental Learning Evaluation

This experiment shows the improvement gained by the automatic incremental learning. At first we run one iteration of video scan using sequential predictor trained on one image only (in Table 3.1 denoted as *iter_0*). Next, we evaluate results after first and second incremental learning (*iter_1* and *iter_2*). Two objects were tested. First was the picture object in Fawlty Towers video (see Figure 3.6 top right image). The SURF based detector was used for picture detection with step $n = 20$ and sequential predictor for validation with step $m = 10$. Incremental learning improves the recall while keeping high precision, see Table 3.1.

Second tested object was a human face (see Figure 3.7). In this case the object was difficult to track with γ_s learned only from one image, because the appearance of the face changed significantly during the sequence. The lighting conditions were challenging and the human face undergoes various rotations and scale changes. We have chosen this sequence in combination with the face detector (instead of SIFT/SURF) to see how

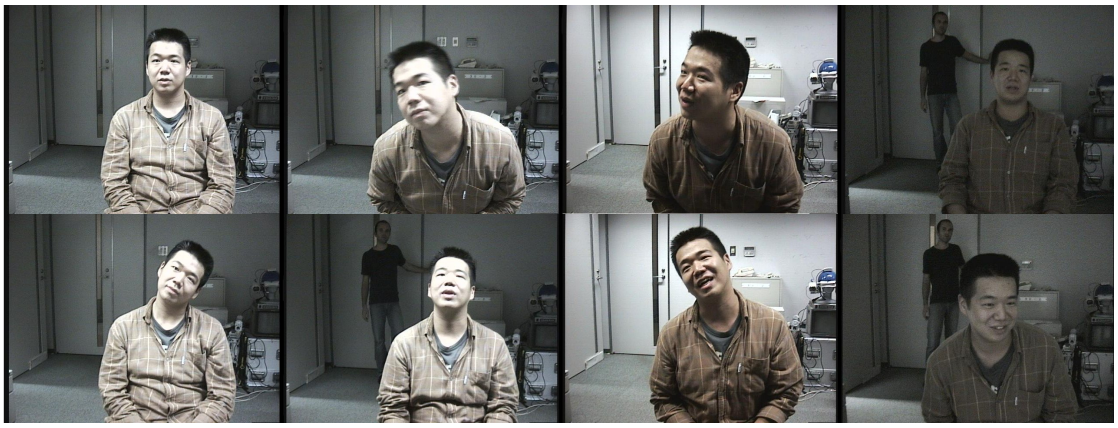


Figure 3.7: Here you may see examples of human face data used in experiment. All images are extracted from one video sequence. Note significant deformations and variations in illumination.

	precision	recall	cumulative time
iter_0	0.99	0.70	4 min 5 sec
iter_1	0.98	0.79	5 min 2 sec
iter_2	0.98	0.81	5 min 27 sec

Table 3.2: Incremental learning evaluation for human face. The first iteration of video scan was running 21 frames per second in average. Browsing time was increased by using the face detector instead of SURF.

the incremental learning helps to improve tracking results on complex non-rigid object. In this case incremental learning also improved the performance of the tracker. See Table 3.2 for results.

The high precision obtained in the face experiment was caused by flawless face detection, which did not return any false positive. You may see a few images of γ_s tracker aligned on human face on Figure 3.8.

3.2.2 Results of Detection and Tracking

One iteration of the algorithm in Fawlty Towers series runs 3−times faster than real-time and more than 8−times faster for the Groundhog Day movie. The detector was run every $n = 20$ frames and validation every $m = 10$ frames while tracking. In the sequence with human face the browsing time was almost twice the real-time, even for detection step $n = 40$. It was caused by the face detector which runs much slower than SURF. The difference in browsing times in Fawlty Towers and Groundhog Day is caused mainly by the different video resolution. Processing of higher resolution images and more complex scenes is slown down by the object detector. Even shorter browsing times may

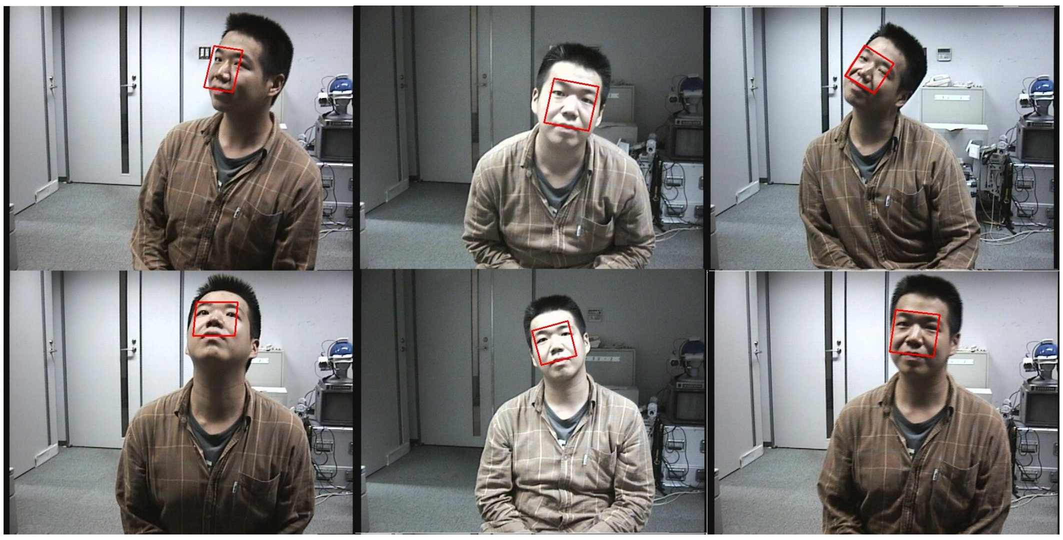


Figure 3.8: Examples of face tracking results. Red rectangle depicts γ_s tracker aligned on human face.

be achieved by increasing the detection interval n. Selecting the right interval depends on our expectation of the shortest time interval, where the object may appear. Reasonable values for detection interval are between 20 and 60 frames. Increasing the validation interval m to more than 10 generates more false positives and since the validation runs very fast, it is not necessary to validate with a bigger step.

Next we compare the performance of predictor validation with SIFT validation. The average time of one SIFT validation was 179 milliseconds and average time of one predictor validation was 33 milliseconds. The resulting recall of video browsing for clock object was 0.58 with SIFT and 0.61 with predictor validation, while the precision was 0.9 for SIFT validation and 0.86 for γ_s validation. Recall for the picture object was 0.94 with SIFT validation and 0.95 with predictor, while the precision was 0.84 for both. Predictor validation gives comparable precision and recall in much shorter time, which also saves time in the whole scanning iteration. Tables 3.3, 3.4 and 3.5 show the results for 5 tested objects obtained in one scanning iteration. The results of the video browsing algorithm are compared to the results produced by the SIFT detector only.

The SURF detection on every frame was tested too, but the results contained large number of false positives. Recall was comparable to SIFT detector, but precision was very low. We are using the SURF detector because it runs much faster than SIFT, but we need to use predictor validation after every positive detection because of the large number of false positives.

Table 3.3, Table 3.4 and Table 3.5 show comparison of SIFT detection in every frame with one iteration of our algorithm. The results demonstrate that even after a single iteration the algorithm gives results comparable with the SIFT detection only. Any

	clock (FT)	**picture**
SIFT detector on every frame - without tracking		
browsing time	32 h. 56 m.	33 h. 29 m.
scanning speed (fps)	0.4	0.4
obtained occurrences	2440	2140
true positives	2411	1996
false positives	29	144
precision	0.99	0.93
recall	0.43	0.89
SURF detect., γ_s track. and valid.		
browsing time	13 m. 42 s.	11 m. 40 s.
scanning speed (fps)	61	71
obtained occurrences	4026	2520
true positives	3462	2131
false positives	564	389
precision	0.86	0.85
recall	0.61	0.95

Table 3.3: Comparison of SIFT object detection only and one iteration of our algorithm on Fawlty Towers - clock and picture.

	alarm clock	**clock** (GhD)
SIFT detector on every frame - without tracking		
browsing time	48 h. 43 m.	48 h. 13 m.
scanning speed (fps)	0.8	0.8
obtained occurrences	1888	855
true positives	1811	801
false positives	77	54
precision	0.96	0.94
recall	0.37	0.29
SURF detect., γ_s track. and valid.		
browsing time	16 m. 46 s.	12 m. 48 s.
scanning speed (fps)	144	189
obtained occurrences	1345	2034
true positives	1125	1520
false positives	220	514
precision	0.84	0.75
recall	0.23	0.55

Table 3.4: Comparison of SIFT object detection only and one iteration of our algorithm on Groundhog Day - alarm clock and clock.

	PHIL sign
SIFT detector on every frame - without tracking	
browsing time	48 h. 7 m.
scanning speed (fps)	0.8
obtained occurrences	2597
true positives	2293
false positives	304
precision	0.88
recall	0.72
SURF detect., γ_s track. and valid.	
browsing time	15 m. 15 s.
scanning speed (fps)	159
obtained occurrences	4038
true positives	2361
false positives	1677
precision	0.58
recall	0.74

Table 3.5: Comparison of SIFT object detection only and one iteration of our algorithm on Groundhog Day - PHIL sign.

subsequent scanning iterations would improve the results, as shows the experiment in Section 3.2.1.

3.3 Summary

We have shown that an incremental learning of sequential predictor significantly improves its robustness. It increases the recall while keeping high precision. Proposed method for collecting additional training examples is completely automatic and requires no user interaction. Stability number well describes the condition of the tracker on particular image and prooves to be good criteria for training examples selection. Validation by clustering of γ_s responses works reliably and very fast.

When coupled with a sparsely applied object detector the system can search for objects through videos several times faster than real-time. The complete system for video browsing works very well with simple objects. Performance for more complex 3D objects (when using SIFT/SURF for detection) are not yet entirely satisfactory. It is mainly the detector that hinders the recognition rate. The tracker itself may be incrementally learned for new appearances of the object and it works better with every iteration. This was verified on face tracking where a robust face detector was applied.

4 Instant Object Detection and Tracking

This chapter focuses on the problem of real-time object detection and tracking in a sequence of high-resolution omnidirectional images. The idea of combining a detector and fast alignment by a tracker has already been used in several approaches [43, 12]. The frame rate of commonly used detectors naturally depends on both the scene complexity and image resolution. For example, the speed of ferns [44], SURF [20] and SIFT [19] detectors depends on the number of evaluated features, which is generally proportional to the scene complexity (e.g. number of harris corners) and image resolution. The speed

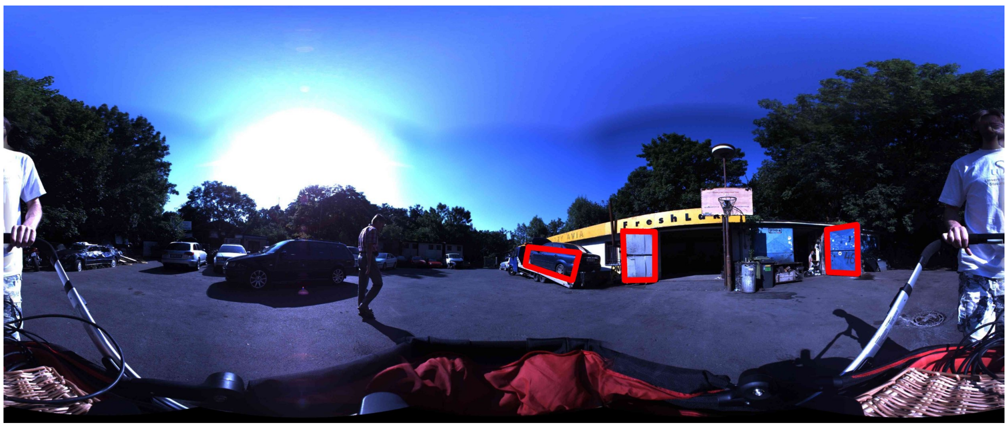


Figure 4.1: Omnidirectional high resolution image (12 Mpx) captured by Ladybug 3 camera. Three objects are marked.

of Waldboost [45] (or any cascade detector) depends on the number of computations performed in each evaluated sub-window. In contrast, most of the trackers are independent of both the scene complexity and image resolution. This guarantees stable frame rate however, once the tracker is lost it may never recover the object position again. Adaptive trackers can follow an object which is far from the training set and cannot be detected by the detector. We propose to combine a detector and a tracker to benefit from robustness (ability to find an object) of detectors and locality (efficiency) of trackers.

Ferns-based detector (also used by [12] for 10 fps tracking-by-detection) is one of the fastest object detectors because of the low number of evaluated binary features on detected harris corners. The speed makes the ferns detector ideal for the purpose of object detection in large images.

One of the most popular template trackers is the KLT tracker [9], which uses the Lucas-

Kanade *gradient descent* algorithm [8]. The algorithm has become very popular and has many derivations [46]. The gradient descent is a fast algorithm yet, it has to compute the image gradient, the Jacobian and inverse Hessian of the modeled warp in every frame. For some simple warps, the Jacobian may be precomputed [47], [48]. One may also get a rid of the inverse Hessian computation by switching the roles of the template and image [46]. Nevertheless we always need to compute the image gradients and in general case also the Jacobian and inverse Hessian of the warp. An alternative for template tracking are *regression-based* methods [10], [11]. They avoid the computation of image gradient, Jacobian and inverse Hessian by learning a regression matrix from training examples. Once learned they estimate the tracking parameters directly from the image intensities. If the regression function is linear, it is called *linear predictor*. The training phase is the biggest disadvantage of linear predictors, because the tracking cannot start immediately. Nevertheless, the regression matrix (function) may be estimated only from one image in a short time (few seconds). The training examples are generated by random warpings of the object template and collecting image intensities. This regression matrix may be updated by additional training examples during tracking [12].

Recently, it has been shown [44], [12], that taking advantage of the learning phase, greatly improves the tracking speed and makes the tracker more robust with respect to large perspective deformations. A learned tracker is able to run with fragment of processing power and estimates object position in complicated or not yet seen poses. However, once the tracker gets lost it may not recover the object position.

To fulfill the real-time requirements, we propose a combination of a robust detector and a very efficient tracker. Both, the detector and the tracker, are trained from image data. The tracker gets updated during the tracking. The tracker performance is extremely fast and as a result of that, faster than real-time tracking allows for multiple object tracking.

4.1 Related Work

We use a similar approach to [12], who also use a fern object detector and a linear predictor with incremental learning for homography estimation. The detector is used for object localization and also for a rough estimation of patch transformation. The initial transformation is further refined by the linear predictor, which predicts full 2D homography. The precision of the method is validated by inverse warping of the object patch and correlation-based verification with the initial patch. The detector is run in every frame of the sequence of 0.3 Mpx images processing 10 frames per second (fps). This approach however, would not be able to perform in real-time on 12 Mpx images. We use the fern detector to determine tentative correspondences and we run RANSAC on detected points to estimate the affine transformation. After a positive detection we apply the learned predictor in order to track the object for as many frames as possible. [12] use an iterative version of linear predictor similar to the one proposed by [10], while we use the sequential predictor γ_s instead. γ_s proved [11] to be faster than the iterative version, while keeping the high precision of the estimation. Our tracker is incrementally updated during tracking [12, 43]. We validate the tracking by the updated tracker itself (see Section 4.2.2), which is more precise, than correlation-based verification by a single template in case of varying object appearance.

Recently [49] used adaptive linear predictors for real-time tracking. Adaptation is done by growth or reduction of the tracked patch during tracking and update of the regression matrices. However, this approach is not suitable for our task, because of the need to keep in memory the large matrix with training examples, which is needed for computation of the template reduction and growth. This training matrix grows with additional training examples collected for on-line learning, which is undesirable for long-term tracking.

[43] use linear predictors in the form of locally weighted projection regressors (LWPR) as a part of self-tuning particle filtering (ISPF) framework. They approximate a non-linear regression by a piece-wise linear models. In comparison we use γ_s similar to [41], which uses the result of previous predictors in sequence as the starting point for another predictor in a row. In [43] the partial least-squares is used for data dimension reduction. We use a subset of template pixels spread over the object in regular grid, which proved to be sufficient for dimensionality reduction, while keeping the high precision and robustness of tracking.

In Section 4.2 you find the formal descriptions of used ferns detector and sequential predictor tracker and in Section 4.2.3 the outline of our algorithm. In Section 4.3 we present the general evaluation of our algorithm. A detailed evaluation of the detector and tracker are given in Sections 4.3.1 and 4.3.2. In the last two sections of this chapter we discuss the computational times of the algorithm.

4.2 Theory

The method combines a fern-based detector and a tracker based on sequential linear predictors. Both the detector and the tracker are trained from the image data. The tracker has its own validation and is incrementally re-learned as the tracking goes. The detector locates the object in case the tracker gets lost.

4.2.1 Ferns-based Object Detector

Object is modeled as a spatial constellation of detected harris corners on one representative image. In a nutshell: the fern detector first estimates similarity between harris corners detected in the current frame and harris corners on the model. The thresholded similarity determines tentative correspondences, which are further refined by RANSAC selecting the largest geometrically consistent subset (i.e. set of inliers). In our approach object was modeled as a plane. Since we observed that the estimation of full homography transformation was often ill-conditioned, because of both insufficient number of detected corners and non-planarity of the object, the RANSAC searches for the affine transformation, which showed to be more robust.

Detailed description of the similarity measure is in [44]. In the following, we provide just short description for the sake of completeness. The similarity measures probability

$p(\mathbf{V}(\mathbf{v}), \mathbf{w})$ that the observed appearance of the neighbourhood $\mathbf{V}(\mathbf{v})$ of the detected corner $\mathbf{v}$ corresponds to the model corner $\mathbf{w}$. The appearance is represented as a sequence of randomly selected binary tests, i.e. given the corner $\mathbf{v}$ and sequence of n point pairs $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots (\mathbf{x}_n, \mathbf{y}_n)\}$, the appearance of the $\mathbf{v}$ is encoded as binary code $V_k(\mathbf{v}) = I(\mathbf{v} + \mathbf{x}_k) > I(\mathbf{v} + \mathbf{y}_k)$, where $I(\mathbf{v} + \mathbf{x}_k)$ is the image intensity.

On one hand, it is insuficient to model probabilities of binary tests independently, i.e. assuming that $p(\mathbf{V}(\mathbf{v}), \mathbf{w}) = \prod_{k=1}^{n} p_k(V_k(\mathbf{v}), \mathbf{w})$. On the other hand, modeling $p(\mathbf{V}(\mathbf{v}), \mathbf{w}) = p(V_1(\mathbf{v}), \dots, V_n(\mathbf{v}), \mathbf{w})$ is ill-conditioned, since we would have to estimate probability in 2^n bins, where n is usually equal to several hundreds. Therefore, we divide the sequence of n binary tests into $N = n/m$ subsequences with length $m \approx 8 - 11$. Subsequences are selected by N membership functions $I(1) \dots I(N)$ and we denote $h_k = \text{card}(I_k)$, $k = 1 \dots N$. Finally, we consider these subsequences to be statistically independent and model the probability as:

$$p(\mathbf{V}(\mathbf{v}), \mathbf{w}) = \prod_{k=1}^{N} p_k(V_{I_k(1)}(\mathbf{v}), \dots, V_{I_k(h_k)}(\mathbf{v}), \mathbf{w}) \tag{4.1}$$

The proposed detector requires an off-line training phase, within which the subsequent probabilities are estimated. Once the probabilities are pre-computed, we use them on-line to determine the tentative correspondences. In the following both phases are detailed. *Offline training phase:* First n binary tests are randomly selected and divided

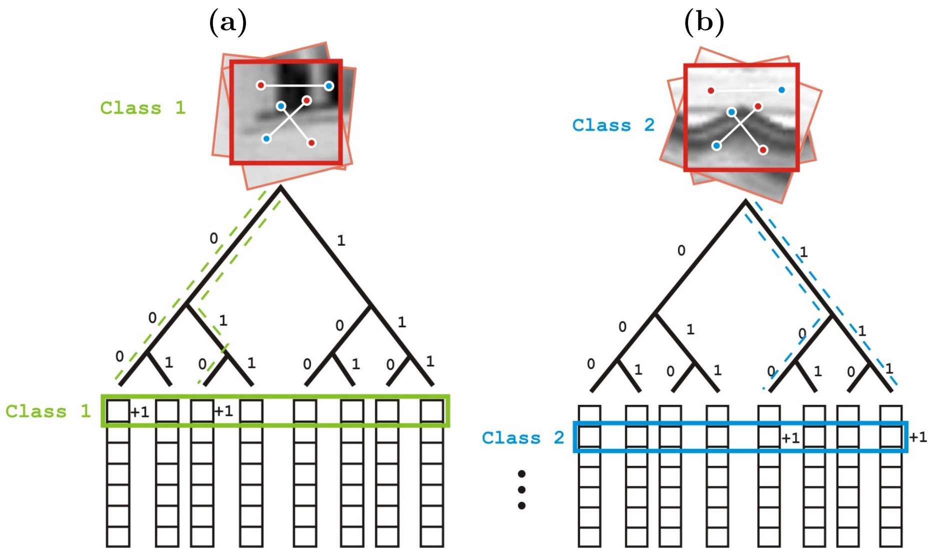


Figure 4.2: Ferns learning procedure. **(a)** Random perturbations of patch around particular harris corner go through the binary tests and increase the corresponding bins values for particular class. **(b)** Another harris corner random perturbations and *the same* binary tests fill the probabilities for *another* class.

into N subsequences, see Figure 4.3. The model is estimated from one sample image, where harris corners are detected within delineated object border. Appearance of each corner's neighbourhood is modeled by N h_k-dimensional binary hyper-cubes, with 2^{h_k} bins, representing joint probability $p_k(V_{I_k(1)}(\mathbf{v}), \ldots, V_{I_k(h_k)}(\mathbf{v}), \mathbf{w})$. To estimate values of the probability, each corners neighbourhood is L-times perturbated within the range of local deformations we want to cope with. For each perturbed training sample and each subsequence, binary tests are evaluated and correspoding bin is increased by $1/L$, see Figure 4.2. Note that different harris corners are modeled via different probabilities but the same binary tests, which allows significant improvement in the online running phase, since the computational complexity of the similarity computation is almost independent of the number of harris corners on the model. *Online running phase:* Given

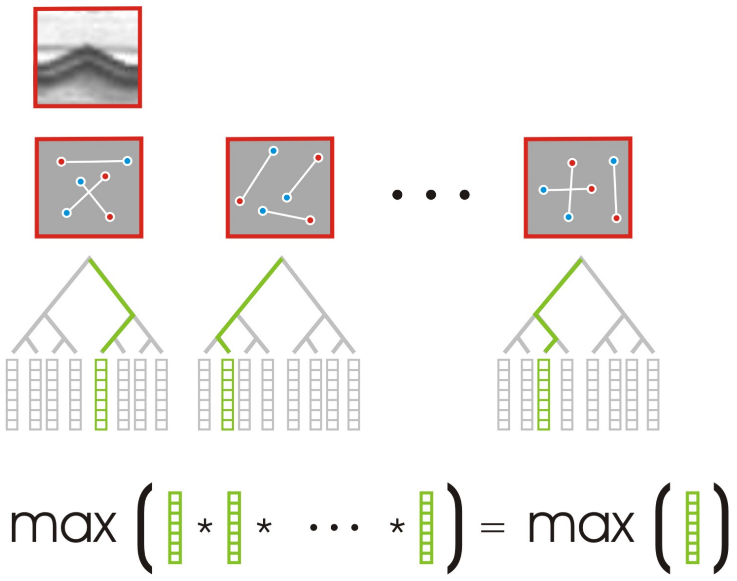


Figure 4.3: Detection procedure for one patch depicted on top. The patch goes through a series of binary tests. Each set of tests results in one probability distribution over the learned classes. The probability distributions are multiplied according to Equation 4.1. The patch is assigned to the class with the highest probability.

an input image, harris corners are detected. For each corner $\mathbf{v}$, binary tests are evaluated and similarity to each model corner is computed via Equation 4.1, see Figure 4.3. Similarities higher than a chosen threshold determine tentative correspondeces. Eventually, RANSAC estimates affine transformation between model and the given image. Confidence of the detection is equal to the number of inliers.

4.2.2 Sequential Linear Predictors

We use basically the same sequential predictor as was used in the previous Chapter 3. The $2-\gamma_s$ tracker is used to track the desired objects. The main difference is in the transformation estimation. Here the first sequential predictor also estimates 2D translation of the object patch, but the second one estimates homography transformation parameters (8 parameters) instead of affine transformation parameters as in the previous Chapter. The homography is parameterized by position of 4 patch corners. Knowing the corners position and having the reference coordinates, we compute the homography transformation for the whole patch. First the translation is roughly estimated by first sequential predictor and than a precise homography refinement is done.

We denote the translation parameters vector $\mathbf{t}^t = (\Delta x, \Delta y)^T$, estimated by the first γ_s, and the homography parameters vector $\mathbf{t}^a = (\Delta x_1, \Delta y_1, \dots, \Delta x_4, \Delta y_4)^T$, estimated by the second sequential predictor which represents the motion of 4 object corners $\mathbf{c}_i = (x_i, y_i)^T, i = 1, \dots, 4$. The object point $\mathbf{x} = (x, y)^T$ from previous image is transformed to corresponding point $\mathbf{x}'$ in current image accordingly

$$\mathbf{p} = \mathtt{A}\left(\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} + \begin{bmatrix}\mathbf{t}^t\\0\end{bmatrix}\right) \tag{4.2}$$

$$\mathbf{x}' = (p_x/p_z, p_y/p_z)^T, \tag{4.3}$$

where $\mathbf{p}$ are homogeneous coordinates. The 3×3 homography matrix $\mathtt{A}$ is computed from 4-point correspondences via least squares method. The used correspondences are the shifted object corners $\mathbf{c}_i + \mathbf{t}^t$ from previous image and the current corners positions $\mathbf{c}_i + \mathbf{t}^t + \left(\mathbf{t}^a_{2i-1}, \mathbf{t}^a_{2i}\right)^T, i = 1, \dots, 4$ estimated by the $2-\gamma_s$ tracker.

Each sequential predictor estimates the parameters according to Equation (2.15). The transformation ρ is here defined by Equation 4.2. Each regression matrix in the sequential predictor is trained using the least squares method defined earlier in Equation 2.19. Here again we use an incremental learning of regression matrices as it was defined in Chapter 2 in Equations (2.19), (2.20) and (2.23). The only difference in incremental learning with respect to the one in Chapter 3 is that the local perturbations, which generate additional training samples, are made by homography transformation.

The tracking procedure needs to be validated in order to detect the loss-of-track. When the loss-of-track occurs, the object detector is started instead of tracking. To *validate* the tracking we use the first sequential predictor, which estimates 2D motion of the object. We utilize the fact that the predictor is trained to point to the center of learned object when initialized in a close neighborhood. On the contrary, when initialized on the background, the estimation of 2D motion is expected to be random. We initialize the predictor several times on a regular grid (validation grid - depicted by red crosses in Fig. 4.4) in the close neighborhood of current position of the tracker. The close neighborhood is defined as 2D motion range (of the same size as the maximal parameters perturbation used for learning), for which the predictor was trained. In our case the range is $\pm(patch\_width/4)$ and $\pm(patch\_height/4)$. We let the γ_s vote for the

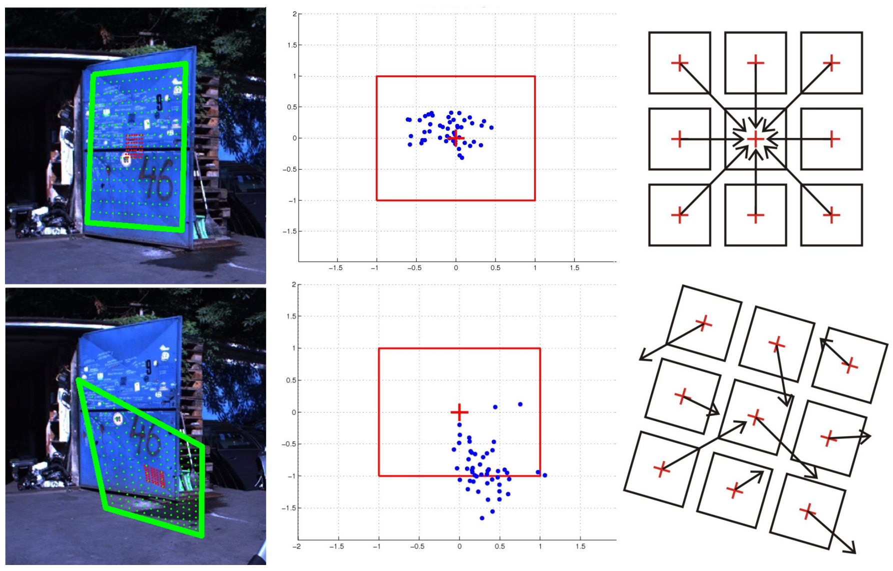


Figure 4.4: Validation procedure demonstrated in two situations. The first row shows successful validation of tracked *blue_door*, the second row shows loss of track caused by a bad tracker initialization. First column shows the tracker position marked by green. The third column depicts the idea of validation - i.e. a few initializations of the tracker (marked by red crosses) around its current position and the collection of votes for object center. When the votes point to one pixel, which is also the current tracker position (or close enough to the center), the tracker is considered to be well aligned on the object. When the votes for center are random and far from current position the loss-of-track is detected. In the second column we see the collected votes (blue dots), the object center (red cross) and the motion range (red rectangle) normalized to $< -1, 1 >$, for which was the γ_s trained.

object center from each position of the validation grid and observe the 2D vectors, which should point to the center of the object, in the case, when the tracker is well aligned on the object. When all (or sufficient number of) the vectors point to the same pixel, which is also the current tracker position, we consider the tracker to be on its track. Otherwise, when the vectors point to some random directions, we say that the track is lost, see Fig. 4.4. The same approach for tracking validation was suggested in [1]. The next section describes in detail the algorithm used in our system, which combines the ferns detector and $2-\gamma_s$ tracker.

4.2.3 The Algorithm

Our algorithm combines the ferns detector and $2-\gamma_s$ tracker together. In order to achieve real-time performance, we need to run the detector only when absolutely necessary. The detection runs when the object is not present in the image or the tracker loses its track.

As soon as the object is detected, the algorithm starts tracking and follows the target as long as possible. Since tracking requires only fragment of computational power, computational time is spared for other tasks. The on-line incremental update of the regressors helps to keep longer tracks. When the validator decides that the track is lost, the detector is started again until next positive detection is achieved. To lower the number of false detections to minimum, we run the validation after each positive response of the detector. The pseudo-code shown in algorithm 4.5 should clarify the whole process.

4.3 Experimental Results

The foreseen scenario for the use of our method is a visual part of mobile rescue robot navigation system. The operator selects one or more objects in the scene and the robot (carying a digital camera) should navigate itself through some space, by avoiding tracked obstacles to localized object of interest. The experiments simulate the foreseen use. Several objects were selected in one frame of particular sequence and from this starting frame they were tracked and detected.

Three types of experiments were performed. First we run the ferns detector itself in every frame without tracking. Second we run the $2 - \gamma_s$ tracker with validation without the recovery by detector. And finally, we run the combination of both. In all experiments were both the detector and the tracker trained from a single image. The detector and the tracker perform best on planar objects, because of the modeled 2D homography transformation. We tested our algorithm also on non-planar objects (*lying human, crashed car*) to see the performance limits and robustnes of our solution, see Section 4.3.3. Algorithm was tested on 8 objects in 4 videosequences. The ladybug camera provides 8 fps of panoramic images captured from 6 cameras simultaneously. Five cameras are set horizontaly in a circle and the sixth camera looks upwards, see Fig. 4.6. The panoramic image is a composition of these 6 images and has resolution of 5400×2248 pixels (12 Mpx). Fig. 4.1 and Fig. 4.7 show examples of the composed scenes and tested objects. Appearance changes for few selected objects are depicted in Fig. 4.8. Notice the amount of non-linear distorsion caused by the cylindrical projection. The objects of interest are relatively small in comparison to the image size. In average the object size was 400×300 pixels. The ground-truth homography for each object was manually labeled in each frame. For evaluation of the detection/tracking performance we provide ROC curves for each tested object. The ROC curve illustrates *false positive rate* versus *false negative rate.*

- *False positive (FP)* is a positive vote for an object presence in some position, but the object was not there.
- *False negative (FN)* is a negative vote for an object presence in some position, where the object actually was present.

```
1: Select object
2: $model\_fern$ ← learn fern detector
3: $model\_tracker$ ← learn $2 - \gamma_s$ tracker
4: $lost$ ← true
5: $i \leftarrow 0$
6: while next image is available do
7:   get next image
8:   $i \leftarrow i + 1$
9:   if $lost$ then
10:     $detected$ ← detect object
11:     if $detected$ then
12:       initialize tracker
13:       estimate homography
14:       $valid$ ← validate position
15:       if $valid$ then
16:         $lost$ ← false
17:         continue
18:       end if
19:     end if
20:   else
21:     track object
22:     if $i$ mod $5 == 0$ then
23:       $valid$ ← validate position
24:       if $valid$ then
25:         $model\_tracker$ ← update tracker
26:       else
27:         $lost \leftarrow true$
28:         continue
29:       end if
30:     end if
31:   end if
32: end while
```

Figure 4.5: Detection and Tracking

In ROC diagrams we want to get as close to the point $(0, 0)$ as possible. Each point in the curve of ROC diagram is evaluated for one particular *confidence threshold* c. In our system the *confidence* r for one detection is given by the number of affine RANSAC inliers after positive detection. The tracker keeps the confidence from last detection until the loss-of-track. With growing confidence we get less false positives, but also more false negatives (we may miss some positive detections). For one particular c we compute the

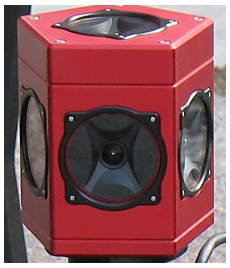


Figure 4.6: Ladybug 3 with 5 cameras placed horizontaly in circle and one camera looking upwards for capturing omnidirectional images.

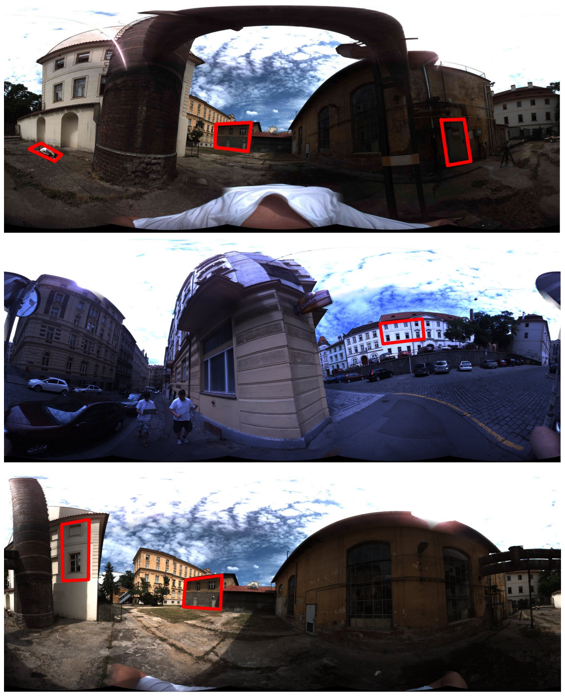


Figure 4.7: Example images with tracked objects marked by red rectangles.

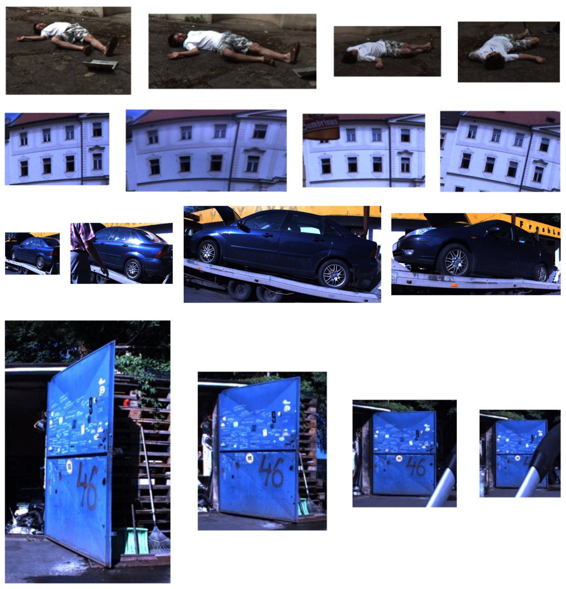


Figure 4.8: Four of eight tested objects under different view angles and appearances.

diagram coordinates as follows:

$$\mathrm{FP}(c) = \sum_{j=1}^{n} (\mathrm{FP}, \text{where } r_j > c)/n \tag{4.4}$$

$$\mathrm{FN}(c) = \sum_{j=1}^{n} (\mathrm{FN}, \text{where } r_j > c)/n, \tag{4.5}$$

where n is a number of frames in sequence. To draw the whole ROC curve we compute the coordinates for a discrete number of confidences from interval $< 0, 1 >$ and use linear interpolation for rest of the values.

In Fig. 4.9 we show three different ROC curves. Each curve corresponds to one method used to search for the object position in sequences. In order to make the evaluation less dependent on a particular object, we computed mean ROC curves over all tested objects for different methods. The green curve depicts the performance of the tracker itself, run on every object from the first frame until the loss-of-track without the recovery by the detector. The blue curve shows results obtained by the fern detector itself run on every frame of all sequences. And finally the red curve shows results, when our algorithm was used. We may observe, that our algorithm performance is better (curve is the closest to point $(0, 0)$) than both individual methods. The separate ROC curves for individual objects may be seen in Fig. 4.10 and Fig. 4.12. The experiments are organised as follows. The ferns detector is evaluated in Section 4.3.1, the performance of tracker is examined in Section 4.3.2. The algorithm 4.5, which combines both is evaluated in Section 4.3.3. And finally in Section 4.3.4 we provide computation times of all main parts of the algorithm.

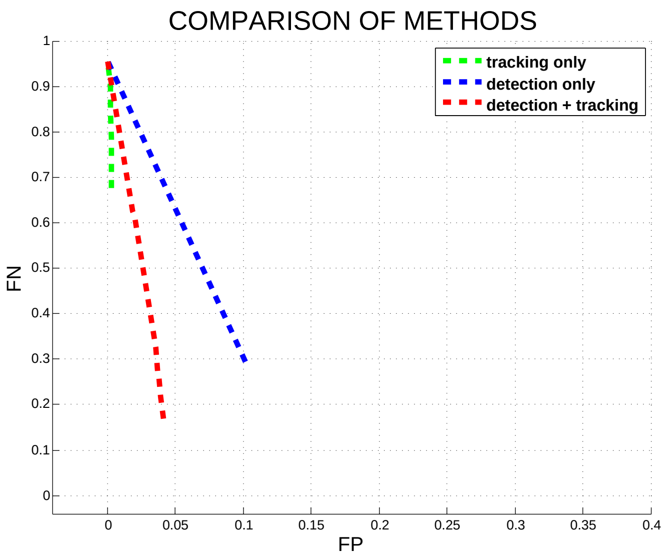


Figure 4.9: Each curve corresponds to results of one method computed as mean ROC curve over all objects.

4.3.1 Detector Evaluation

Using only the detector (tracking by detection) would be too slow for desired real-time performance in sequence of large images. Nevertheless we evaluate the performance of the detector itself to see how the adition of γ_s tracker lowers the false positive and false negative rate (see Section 4.3.3).

In this experiment the detector was run with slightly different set of parameters than in the experiment which combines it with the tracker. This was necessary in order to achieve the best detection performance. For example here it was not possible to aditionally validate the positive detection by the validator. So we needed to increase the threshold for number of RANSAC inliers necessary for positive detection to lower the number of false positives.

It was also necessary to adjust the detector parameters according to expected object scale and rotation changes. In average the detector was searching for the object in 3 scales and it was able to detect objects under ± 20 degrees rotation. In Fig. 4.10, the ROC curves are depicted for detector run in every frame for different objects. The results show that some objects were detected in almost all cases correctly, while some other objects, like the *door*, with poor results. *Door* was the most complicated object for harris corners-based detector, since only 21 keypoints were detected over the object, which were spread mostly in the central part of the door. That is why there was almost always a low number of inliers comming out of the RANSAC algorithm. This object was lately successfuly tracked by the tracker. Another complicated object was the *car*, due to its reflective surface and vast visual angle changes. Finally, the *human* lying

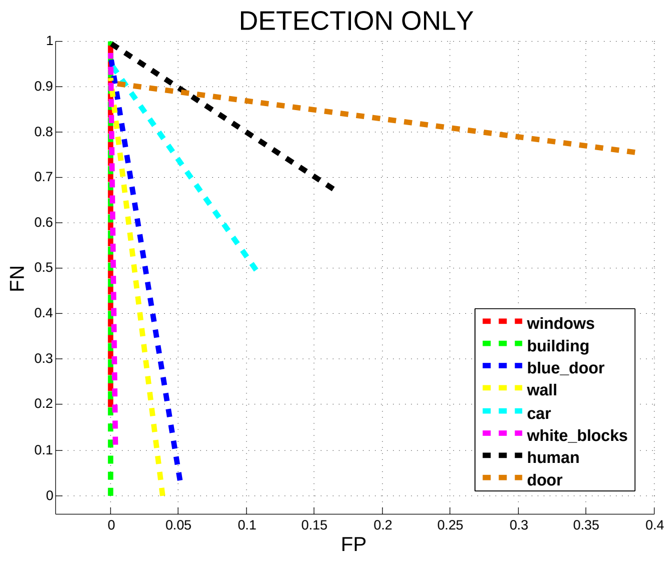


Figure 4.10: Each curve corresponds to detection results for one object.

on the floor was also a challenging object due to its non-planarity. As you will see in Section 4.3.3, the integration of tracking to the algorithm lowers the number of FP and FN and significantly speeds up the algorithm, see Section 4.3.4.

4.3.2 Tracker Evaluation

This experiment shows performance of the tracker without the recovery by the detector. The tracker is composed of 2 sequential predictors (for translation and homography). Each sequential predictor has 3 predictors in sequence with support set sizes $|X_1| = 225$, $|X_2| = 324$ and $|X_3| = 441$. The support set coordinates were spread over the object in regular grid. The tracker was incrementaly learned and validated during tracking until it lost its track or until the end of sequence. The tracking was manually initialized always in the first image of sequence (different form training image), where the object appeared. Some objects were tracked through the whole sequence. Some objects were lost after few frames, when there was fast motion right in the beginning. In Fig. 4.11 you may see the lengths of successful tracking until the first loss-of-track. In case of partial occlusion the tracker sometimes jitters or even fails. Nevertheless, when it is incrementally learned, it is able to handle the occlusion as a new object appearance. Incremental learning itself is very helpful for increasing the robustness of the tracker [12], [1]. The estimation of homography is very precise for planar objects.

Tracked objects appear in images as patches in resolutions varying from 211×157 (33 127 pixels) to 253×919 (232 507 pixels). Both sequential predictors work only with the subset of all patch pixels (same subset size for all objects). When tracking, each γ_s needs to read only 990 intensity values, which is given by the sum of support set sizes

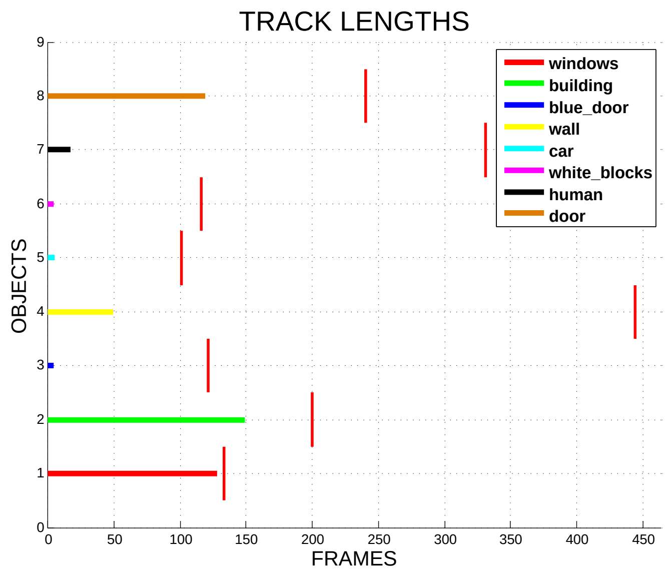


Figure 4.11: Each horizontal line depicts the length of track for one object until the first loss-of-track. The red vertical lines show the last frame of particular subsequence, where the object was fully or partially visible.

of predictors in sequence. This brings another significant speed-up for the learning and tracking process.

4.3.3 Detector and Tracker evaluation

Final experiment evaluates the performance of algorithm described in Section 4.2.3. The combination of the detector and the tracker improves the performance of the algorithm (lowers FP and FN), as may be seen in Fig. 4.12 and Fig. 4.9. This is caused by their complementarity in failure cases. Tracker is very robust even under extreme perspective deformations, while the detector is not able to recognize these difficult object poses. On the other hand the detector is robust to partial occlusion, where the tracker usually fails and needs to be recovered and re-learned. In comparison with the detector (see Fig. 4.10), our algorithm in average significantly improves the results. Only few objects, which were perfectly detected by the detector (e.g. *white blocks* and *blue door*) have a little worse results with our algorithm. This was caused by the tracking validation, which was running not every frame, but only every 5 frames, which means, that the tracker was lost a few frames just before loss-of-track detection by validation and received a few FPs and FNs. This small error could be eliminated by running the validation in every frame. The extreme efficiency of sequential predictors allows tracking much faster than real-time, which provides enough computational time for validation and incremental learning of the tracker. Running validation after each positive detection allows us to make the ferns detector more sensitive. We lower the threshold which specifies the number of neccessary

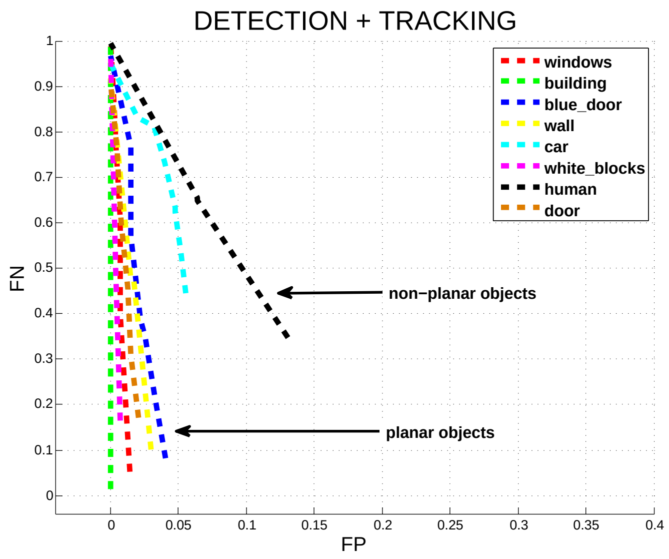


Figure 4.12: Each curve corresponds to results of one object detection and tracking. The ROC curves fit more to the left side of the diagram. This is caused by the high confidence of detections and tracking. The high confidence is actually valid, because of the very low number of false positives, as we may observe.

inliers, which allows more true positive, but also more false positive detections. After each detection, which has small number of inliers, we initialize the tracker in detected pose, let the tracker vote for homography and run the validation. Validation eliminates possible false positive detections and let pass the true positives.

The most difficult object for our algorithm was the *crashed car*, the appearance of which was changing signifficantly during the sequence, due to its reflective surface, non-planar shape and vast changes in visual angle. Detection and tracking of *lying human* was successful in high percentage of detected occurences and low FP and FN. But the precision of homography estimation was quite poor as expected, because of its non-planar geometry. Nevertheless the incremental learning kept the tracker from loosing its track too often. The robust detector and incremental learning of the tracker allows for tracking of more complex (non-planar) objects, but high precision homography estimation can not be expected. Planar or semi-planar objects were detected and tracked with high accuracy.

4.3.4 Computation Times

The algorithm was run on standard PC with 64 bit, 2.66 GHz CPU. The object detector was implemented in language C and run in the system as MEX. The RANSAC, $2-\gamma_s$ tracker and the rest of the algorithm were implemented in Matlab. When we pretermit the high resolution images, the PC memory requirements were minimal. The computation times for 12 Mpx image sequences are following:

(implementation in language C)

- detector learning: 2 sec for 200 classes, i.e. 10 ms per class (50 ferns with depth 11 and 500 training samples per class).
- detection: 0.13 ms for evaluation of 1 harris point with 50 ferns and 200 classes. The computational time changes linearly with the number of classes. For one image with 5350 harrises, which passed the quality threshold, it took 0.7 sec. Usually we run the detector in 3 scales.

(implementation in Matlab)

- learning of the γ_s trackers: 6 sec for the translation tracker with 1500 training samples and 9 sec for the homography tracker with 3500 training samples.
- tracking: 4 ms per image. This computational time is summed for both γ_s trackers.
- validation: 72 ms per one validation. In our experiments, the validation was run every 5 frames during tracking.
- incremental learning: 470 ms together for 10 samples for the translation tracker and 10 samples for the homography tracker. Incremental learning was triggered every 5 frames after successful validation.

Average amount of harris points in one image was around 50000, from which around 5300 passed the harris quality threshold [9] and were evaluated by ferns detector. The use of object detector is neccessary, but its runtime needs to be reduced to a minimum because of the high computational time. The tracker runs very fast, which allows for multiple object tracking, incremental learning and tracking validation.

4.4 Summary

In this chapter we combined ferns-based object detector and $2-\gamma_s$ tracker in an efficient algorithm suitable for real-time processing of high resolution images. The amount of streamed data is huge and we need to avoid running the detector too often. That is why we focused on updating the $2-\gamma_s$ model during tracking, which helped to keep the track even when the object appeared under serious slope angles and with changing appearance. In comparison with the detector run on every frame, our algorithm runs not only much faster, but also lowers the number of false positives and false negatives.

5 Deformable Object Tracking

In this chapter we focus on real-time deformable object tracking. Non-rigid deformations are difficult to model. We can't model the deformation by independent motion of every point of the object, because such a high dimensional model would prone to overfitting. We tackle the problem of low dimensional modelling by the *principal component analysis* (PCA).

impossible to manually design a transformation, which simulates the deformation realistically. The individual object points form a grid $(x_i, y_i)^T \in G$, where $\forall i \in 1, \dots, n$. The grid is a subset of image pixels and it is hard to parameterize their motion, eventhough we may see, that the motion of individual pixels is not independent. When we forget about the mutual pixel motion dependency, we may parameterize the transformation by the displacement of each point in the grid G by two numbers. The number of parameters of such a transformation would be very high. The probability of tracking failure grows with the number of estimated parameters. Also every additional parameter increases the number of pose space dimensions and the number of necessary training samples grows exponentially.

The individual object points in an image form a grid $(x_i, y_i)^T \in G$, where $\forall i \in 1, \dots, n$. Each grid point brings two parameters to the model. Nevertheless when we have got enough annotated training samples, we may extract the pixel deformation dependencies. We compute the PCA over all the training samples, which allows us to identify the principal components of the deformation and parameterize it with only a few parameters. This way we achieve a significant dimensionality reduction of the pose space. The last problem arises with the ground truth annotation of the training samples. We need to annotate every point of the deformed grid on every training sample. This would be extremely difficult and time consuming, since usually a few thousands of training samples are necessary for learning of an object generic tracker. To overcome this limitation we annotate only a few keypoints in each training sample and we use the displacement of these keypoints to drive an automatic flexible deformation of the remaining points in the grid, see Figure 5.1. We test our approach on human face deformation and a deformation of a human eyelid when blinking, as may be seen on Figure 5.1. To deform the grid we use the *thin plate splines* [50] method.

For the sake of completeness we derive the equations for thin plate splines for elastic deformation, which we use to deform the grids on training data. If you are not interested in the thin plate splines method, please skip the text and continue reading right after the Equation 5.6.

First we compute the mean position of m keypoints from all training samples anno-

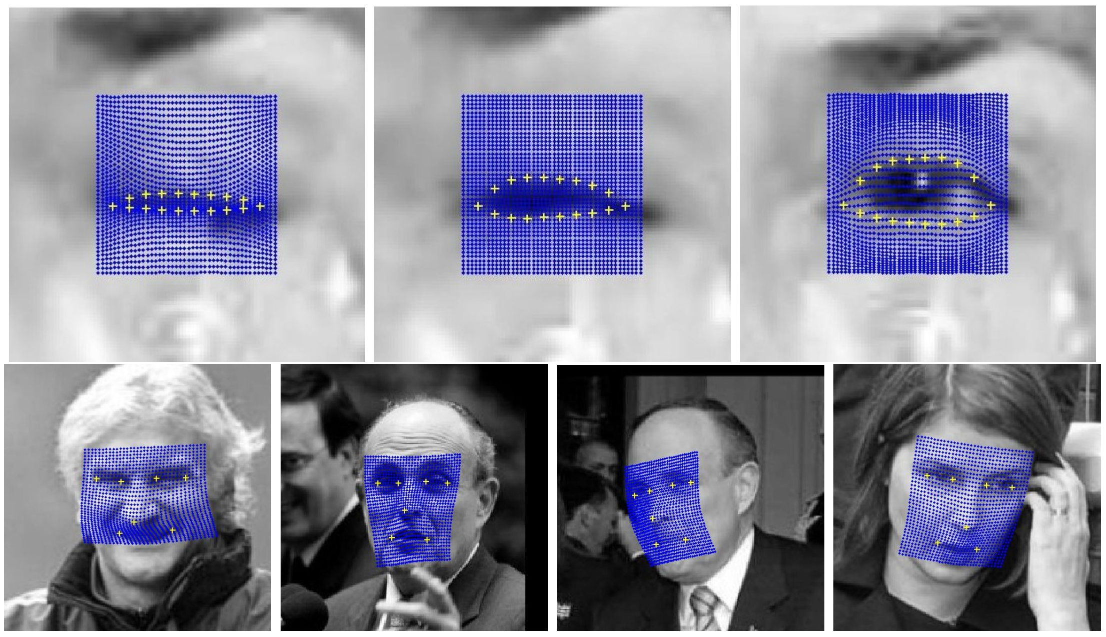


Figure 5.1: Generating ground truth points deformation. Yellow markers are the control points used in the thin plate splines method for the whole grid deformation. The blue dots correspond to the deformed grid points $(x_i^{'}, y_i^{'})^T \in G^{'}$.

tations $P_1 = (x_1, y_1)^T, \ldots, P_m = (x_m, y_m)^T$. Lets denote the same keypoints positions after the displacement in one sample as $P_1^{'} = (x_1^{'}, y_1^{'})^T, \ldots, P_m^{'} = (x_m^{'}, y_m^{'})^T$. Next we define the distance between two keypoints as $d_{ij} = \|P_i - P_j\|$ and a kernel function $k(d) = d^2 \ln(d^2)$. We will also need matrices

$$\mathtt{K} = \begin{bmatrix} 0 & k(d_{12}) & \ldots & k(d_{1m}) \\ k(d_{21}) & 0 & \ldots & k(d_{2m}) \\ \vdots & \vdots & \ddots & \vdots \\ k(d_{m1}) & k(d_{m2}) & \ldots & 0 \end{bmatrix}, \tag{5.1}$$

$$\mathtt{A} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & y_m \end{bmatrix}, \tag{5.2}$$

$$\mathtt{B} = \left[\begin{array}{c|c} \mathtt{K} & \mathtt{A} \\ \hline \mathtt{A}^T & \mathtt{O} \end{array}\right], \tag{5.3}$$

where $\mathtt{O}$ is 3×3 zero matrix. The elastic transformation coefficients for computing the new x coordinate of arbitrary grid point are obtained as

$$\begin{pmatrix} \mathbf{w}_x \\ a_{1x} \\ a_{xx} \\ a_{yx} \end{pmatrix} = \mathtt{B}^{-1} \begin{pmatrix} x_1^{'} \\ \vdots \\ x_m^{'} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{5.4}$$

and the coefficients for computation of the new y coordinates are

$$\begin{pmatrix} \mathbf{w}_y \\ a_{1y} \\ a_{xy} \\ a_{yy} \end{pmatrix} = \mathtt{B}^{-1} \begin{pmatrix} y_1^{'} \\ \vdots \\ y_m^{'} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{5.5}$$

where $\mathbf{w}_x = (w_{1x}, w_{2x}, \dots, w_{mx})^T$ and similarly for $\mathbf{w}_y$. Now we may finally define the elastic transformation of each point of the grid $(x_i, y_i)^T \in G$ to the transformed point $(x_i^{'}, y_i^{'})^T \in G^{'}$ via thin plate splines method as

$$\begin{pmatrix} x_i^{'} \\ y_i^{'} \end{pmatrix} = \begin{pmatrix} a_{1x} + a_{xx}x_i + a_{yx}y_i + \sum_{j=1}^{m} w_{jx} k\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right) \\ a_{1y} + a_{xy}x_i + a_{yy}y_i + \sum_{j=1}^{m} w_{jy} k\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right) \end{pmatrix}. \tag{5.6}$$

According to the non-linear Equation (5.6) we generate the deformed grid (depicted at Figure 5.1). The parameters for every image sample are the 2D displacement vectors between the regular grid and the deformed grid. The number of parameters is two times the cardinality of the set G. To avoid the prediction of so many parameters at once we perform the principal component analysis (PCA) to linearize (5.6) and for dimensionality reduction. First we compute the mean parameters vector from all the training samples and store them in one vector

$$\mathbf{m} = (m_1, m_2, \ \dots \ , m_{2n})^T = \frac{1}{p} \sum_{i=1}^{p} \left(x_1^i, y_1^i, x_2^i, y_2^i, \ \dots \ , x_n^i, y_n^i\right)^T,$$

where the upper index i denotes the training sample number and not the exponent. Then all the deformed points from each grid G_i (from each of p training samples, $\forall i \in \{1, \dots, p\}$) with subtracted parameters means are put as a single column vector into the matrix $\mathtt{P}$ of size $2n \times p$, as follows

$$\mathtt{P} = \begin{bmatrix} x_1^1 - m_1 & x_1^2 - m_1 & \dots & x_1^p - m_1 \\ y_1^1 - m_2 & y_1^2 - m_2 & \dots & y_1^p - m_2 \\ x_2^1 - m_3 & x_2^2 - m_3 & \dots & x_2^p - m_3 \\ y_2^1 - m_4 & y_2^2 - m_4 & \dots & y_2^p - m_4 \\ \vdots & \vdots & \ddots & \vdots \\ x_n^1 - m_{2n-1} & x_n^2 - m_{2n-1} & \dots & x_n^p - m_{2n-1} \\ y_n^1 - m_{2n} & y_n^2 - m_{2n} & \dots & y_n^p - m_{2n} \end{bmatrix}. \tag{5.7}$$

Then the PCA is performed via the *singular value decomposition* of the parameters matrix as $[\mathtt{U}\,\mathtt{S}\,\mathtt{V}^T] \leftarrow svd(\mathtt{P})$. The orthogonal matrix with the left singular vectors $\mathtt{U}$ may be used to project the original parameters matrix $\mathtt{P}$ to another matrix $\mathtt{T} = \mathtt{U}^T\mathtt{P}$. In order to reduce the number of parameters we take only the first g columns of matrix $\mathtt{U}$ and store them in another matrix $\mathtt{U}' = [\mathbf{u}_1, \dots, \mathbf{u}_g]$. Now we may get the reduced matrix with projected parameters accordingly $\mathtt{T}' = \mathtt{U}'^T\mathtt{P}$. The approximation of the matrix $\mathtt{P} \approx \tilde{\mathtt{P}}$ may be computed according to $\tilde{\mathtt{P}} = \mathtt{U}'\mathtt{T}'$. The original matrix $\mathtt{P}$ of size $2n \times p$ has been reduced to size $g \times p$ of new training matrix $\mathtt{T}'$. In practise g is a small number ranging from 1 to 10 depending on the deformation complexity. In our experiments we use $g = 1$ for the eye opening and $g = 4$ for the facial deformation estimation. The transformation ρ of the $j-$th grid point is then defined as

$$\rho : \begin{pmatrix} x_j' \\ y_j \end{pmatrix} = \begin{bmatrix} u_{2j-1,1} & \dots & u_{2j-1,g} \\ u_{2j,1} & \dots & u_{2j,g} \end{bmatrix} \mathbf{t} + \begin{pmatrix} m_{2j-1} \\ m_{2j} \end{pmatrix}. \tag{5.8}$$

I.e. we take two rows of the matrix $\mathtt{U}'$ corresponding to the j-th grid point coordinates, multiply it by the vector of parameters $\mathbf{t}$ and add the corresponding mean parameters values.

5.1 Learning of the Piecewise Linear Predictor

Given the preprocessed training data we need to train a sequential predictor γ_s, which will predict the object deformation in real-time from the image data. Here $m-$th predictor γ in the sequence is basically a set of jointly learned regressors r_{mj}. Each regressor is a function of one particular feature $f_j \in F_m$, where F_m is a set of features used by the $m-$th predictor. As image features we use the differences of two image pixels. To the $j-$th feature area assigned two selected grid coordinates $\{P_{j,1}, P_{j,2}\} \subset G$, where the feature value v is computed with the feature function $f_j : v = I(\rho(P_{j,1}, \mathbf{t})) - I(\rho(P_{j,2}, \mathbf{t}))$. The set of regressors (one predictor) are learned to lower the residual alignment error $(\mathbf{t}^i - \mathbf{a}^i)$ of the preceding predictor in sequence, where $\mathbf{t}^i$ is one column of the training matrix $\mathtt{T}'$ and $\mathbf{a}^i$ is the accumulated parameter vector estimated by all the previous predictors in the sequence for $i-$th sample. We rewrite the learning Equation (2.6) for

	bin 1	...	**bin U**
param. 1	$\omega = 0.621, \lambda = -0.045$	...	$\omega = 0.573, \lambda = 0.122$
param. 2	$\omega = 0.544, \lambda = 0.194$	...	$\omega = 0.250, \lambda = -0.174$

Figure 5.2: Example of a piecewise affine regressor r_{mj} for two tracking parameters. Each cell contains the learned slope and intercept parameters of the fitted affine function into one bin.

the $m-$th predictor in sequence over its individual regressors as

$$\arg\min_{r_{mj}} \sum_{i=1}^{p} \left\| \sum_{j=1}^{J} \left(r_{mj}(f_{mj}(I^i, \mathbf{a}^i)) \right) - (\mathbf{t}^i - \mathbf{a}^i) \right\|^2, \tag{5.9}$$

where $\mathbf{t}^i$ is the correct parameters vector and J is the number of features in set F_m used by one predictor in sequence. The non-linear regression usually yields higher precision than the linear one, yet we need to keep the fast performance. That is why we approximate fitting a non-linear function by binarization of each features' space (dividing the feature space into U bins), where into each bin $u \in \{1, \ldots, U\}$ we fit a linear function. We also experiment with fitting three different functions: piecewise linear, affine and constant:

$$t_k^i = \omega_{ju} f_j \left(I^i, \mathbf{a}^i \right), \tag{5.10}$$

$$t_k^i = \omega_{ju} f_j \left(I^i, \mathbf{a}^i \right) + \lambda_{ju}, \tag{5.11}$$

$$t_k^i = \lambda_{ju}. \tag{5.12}$$

where $k \in \{1, \ldots, g\}$ is the index of a particular deformation parameter. Each regressor r_{mj} is a series of g piecewise functions of type (5.10) or (5.11) or (5.12) working with a $j-$th feature value. We may write the the regressor r_{mj} as a table with the piecewise functions coefficients for individual bins in columns and tracked parameters in rows, see Figure 5.2. The optimal coefficients of group of regressors solving (5.9) are estimated by a least squares method. Let us denote $\mathtt{L} = \left[\mathbf{l}^1\mathbf{l}^2 \ldots \mathbf{l}^p\right]$ the matrix of training samples, where each vector $\mathbf{l}^i = (\ \ldots, f_{j-1}(I^i, \mathbf{a}^i), f_j(I^i, \mathbf{a}^i), f_{j+1}(I^i, \mathbf{a}^i),\ \ldots)^T$ contains the values of features $f_j, \forall j \in J$ of training sample i. The second training matrix with the ground truth deformation parameters $\mathtt{T}^{'}$ stored columnwise according to $\mathtt{L}$ is already prepared. The least squares solution of (5.9) may be written in a compact form as $\mathtt{T}^{'}\mathtt{L}^{+}$, where $^{+}$ denotes the Moore-Penrose pseudo inverse of a matrix [51]. The same equation is used for learning of all three types of tested piecewise linear functions with any number of bins. The only difference is in the composition of vectors of training samples $\mathbf{l}^i$ in the training matrix $\mathtt{L}$. Normally each row of matrix $\mathtt{L}$ corresponds to the values of a single feature over all training examples. We extend the number of rows corresponding to each feature to U (the number of bins), where the feature value fills only the position of the corresponding bin in each training example. Lets suppose, for example, that we want to partition the feature space into three bins. Than the row of $\mathtt{L}$ corresponding to one feature f_j over all training examples expands into $3 \times p$ matrix

$$\mathtt{\Omega}_j = \begin{bmatrix} 0 & f_j(I^2, \mathbf{a}^2) & & 0 \\ f_j(I^1, \mathbf{a}^1) & 0 & \dots & 0 \\ 0 & 0 & & f_j(I^p, \mathbf{a}^p) \end{bmatrix}. \tag{5.13}$$

When we want to use the intercept parameter λ_{ju} in the regression functions (5.11) and (5.12), we also need the expansion

$$\mathtt{\Lambda}_j = \begin{bmatrix} 0 & 1 & & 0 \\ 1 & 0 & \dots & 0 \\ 0 & 0 & & 1 \end{bmatrix}. \tag{5.14}$$

The training matrices used for learning of the regression functions (5.10), (5.11) and (5.12) are extended as follows

$$\mathtt{L}_{(5.10)} = \begin{bmatrix} \mathtt{\Omega}_{j_1} \\ \mathtt{\Omega}_{j_2} \\ \vdots \end{bmatrix}, \quad \mathtt{L}_{(5.11)} = \begin{bmatrix} \mathtt{\Omega}_{j_1} \\ \mathtt{\Lambda}_{j_1} \\ \mathtt{\Omega}_{j_2} \\ \mathtt{\Lambda}_{j_2} \\ \vdots \end{bmatrix}, \quad \mathtt{L}_{(5.12)} = \begin{bmatrix} \mathtt{\Lambda}_{j_1} \\ \mathtt{\Lambda}_{j_2} \\ \vdots \end{bmatrix}. \tag{5.15}$$

The individual regressors learned with different piecewise functions are visualized in Figure 5.3. The piecewise constant function (5.12) is significantly faster in runtime than the other two. In order to estimate the motion contribution from one feature value, we just need to read a constant value from particular bin. For a reasonable number of bins the function (5.12) quickly reaches the alignment precision of functions (5.10) and (5.11), see section 5.2.2.

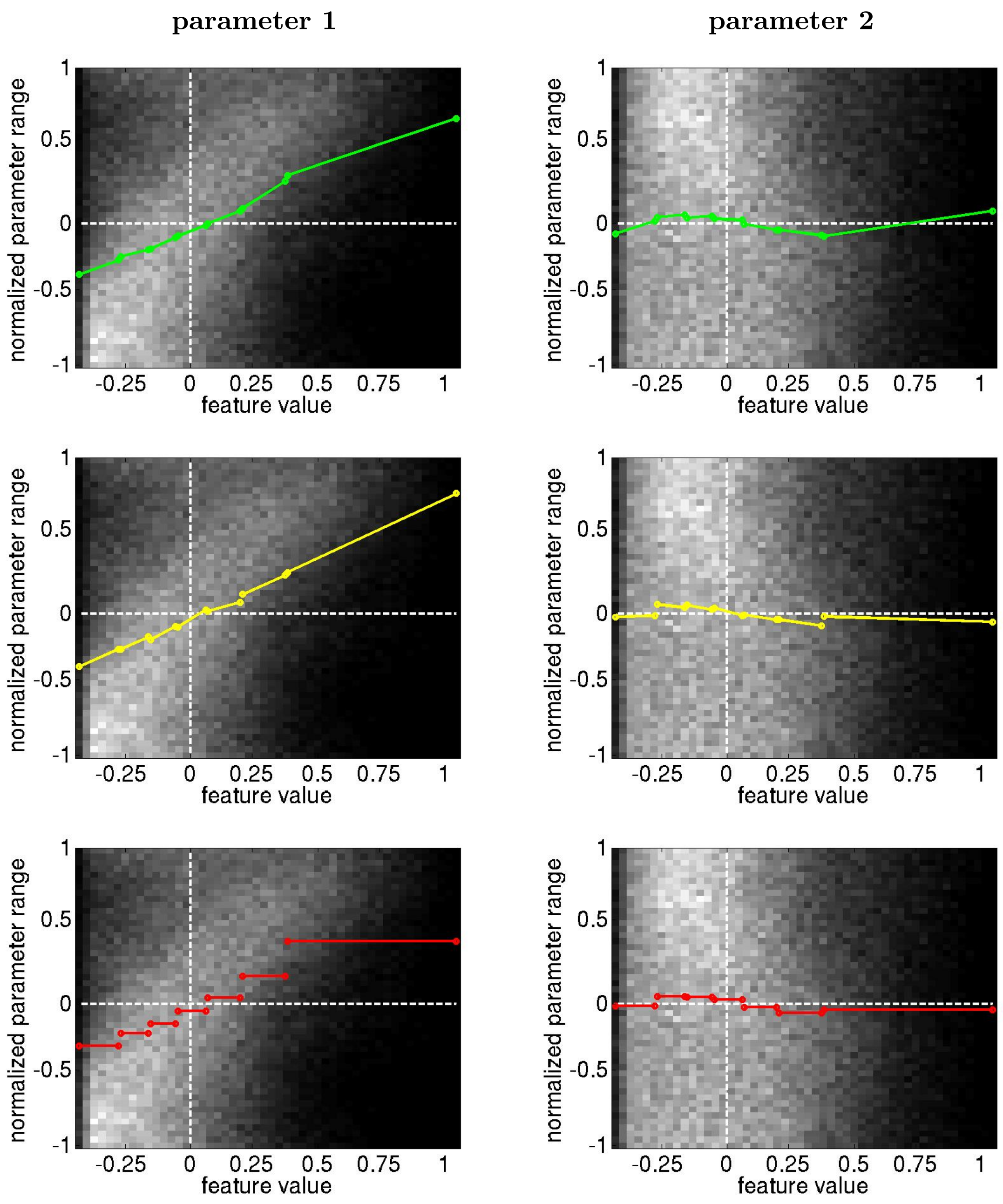


Figure 5.3: Fitted piecewise functions into the values of one feature collected over all training samples. The grey value heatmap is a 2D histogram showing the distribution of training samples in the space. Green - piecewise affine function, yellow - piecewise linear function, red - piecewise constant function. Notice, that the first parameter change is well encoded by the feature value, while the second parameter is not. Thanks to the non-linearity of the fitted functions we may exploit at least some dependency of the second tracking parameter on this feature value.

5.1.1 Additional Training Samples Generation

The object generic tracker requires relatively high number of training samples. The more instances of a particular object (e.g. more different people for face and eye tracking) the better the tracker generalizes for object instances, which were not seen during the training process.

Ideally every pose in the pose space (combination of tracking parameters) should be seen in every instance of the object. But this is usually not the case. We work with limited amount of training images, where there are only a few observation conditions for one object instance (few images with different poses and appearances of one human) or in most cases only one observation for one instance. In order to cover the pose space when the number of training samples is lower than necessary, we may generate additional training samples from the existing ones by random perturbation of the initial vector of parameters. This way we create multiple starting positions for every training sample which increases the number of the training samples and helps to cover the pose space with training samples more densely. When applied carefully, this approach is very efficient and helps to lower both the training and testing error.

Also when the tracker is learned strictly from the annotated data without the artificially generated training samples, the tracking gets into trouble in situations, where it is initialized in the current image with parameters vector different from zeros. Lets say we initialize the tracker with the right parameters for appearing object deformation and the parameters update should be a zero vector. Then the tracker may estimate non-zero parameters update, since it has not yet seen this type of appearance paired with the zero parameters vector in the training set, see Figure 5.4. This is valid also for other non-zero parameters initializations, which is used during the continuous tracking, where in the current image is the tracker initialized with the parameters vector estimasted in the previous image. When generating the artificial perturbations, we want the generator to perform conservatively and we need to respect the parameters ranges. We take the well aligned tracker with the ground truth parameters vector $\mathbf{t}^i$ and then we add a perturbation vector $\Delta\mathbf{t}$ the values of which range from plus to minus half of the parameters range. The starting position for each generated training sample is $\mathbf{a}^i = \mathbf{t}^i + \Delta\mathbf{t}$ This way we generate multiple training samples (cca 10) from each training image.

5.1.2 Features Selection

An important part of the learning process is the feature selection. For every predictor in the sequence we generate a pool of features, from which we greedily select the ones, which lower the training error the most and then we relearn the predictor jointly over all the number of C selected features. The feature selection process is a little bit different from the support set selection algorithm B.1 proposed earlier. Let us extend the feature selection algorithm from [11] (see Appendix B) here for our feature functions f_j, where F^* denotes the resulting set of selected features.

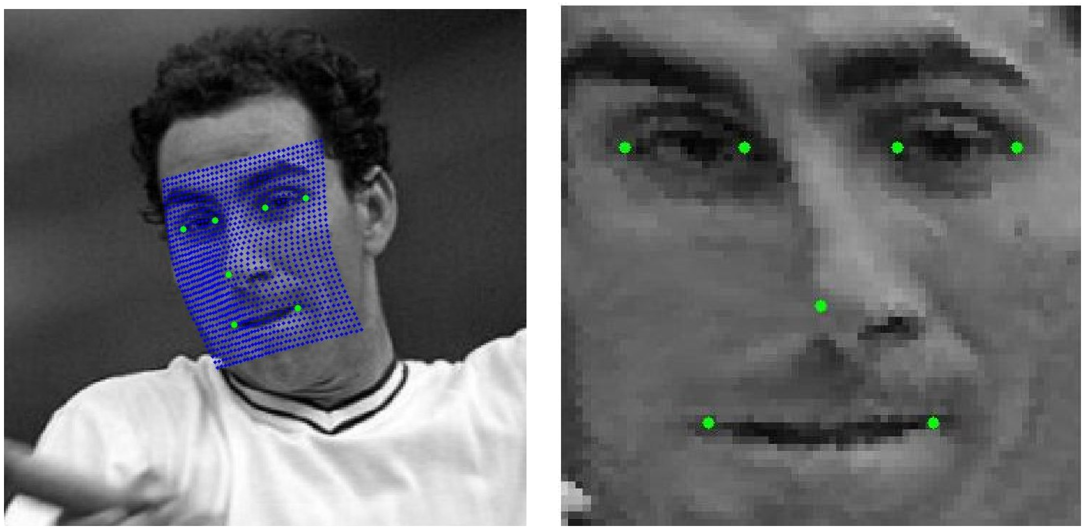


Figure 5.4: Left - well aligned tracker with the deformed grid (deformed by 4 tracking parameters). Right - rectified image using the estimated deformation. As you may see, even when the tracker is well aligned, the appearance of the object looks different from the real face images. The difference in visual appearance is caused partially by the imprecisely simulated deformation and partially by the self-occlusion of some parts of the object - missing visual data, e.g. when it is covered by the nose. That is why generating artificial image perturbations is helpful even when we have enough training data.

5.2 Experiments

For our experiments we use two datasets with manually annotated ground truth data. The first one contains consecutive images of opening eyes. There are 32 different people captured and the dataset consists of 7638 images. We will call this dataset *EyeData* and we model the deformation of each eyelid during the eye opening process. Each eye in the dataset was manually labelled with 8 points positioned on the eyelids edges. The images come from a high framerate camera (420 fps), where the process of eye opening is well visible. The images are in low resolution, very noisy and suffer from strong compression artefacts. The average width of each eye is only about 16 pixels. See Figure 5.6 with a few samples from EyeData. In order to obtain more annotated points we fit a 2D spline function into the upper eyelid and lower eyelid edges separately and along the spline we generate 10 points The second dataset contains individual images of human faces from the *Labeled Faces in the Wild* dataset [52] with 12007 images and 7-point ground truth annotations per image. We will call this dataset *FaceData*, see Figure 5.7 with a few samples. In this dataset we model the nonrigid deformation of human face caused by out-of-plane rotation a different facial expressions.

We flip each image along the vertical axis to get twice as much data. Then we split

```
1: $\mathcal{F} \leftarrow$ generate random features // *Fill the feature pool*
2: $F^* \leftarrow \{\emptyset\}$ // *Initialize the set of selected features*
3: $\hat{\mathtt{T}} \leftarrow \mathtt{T}' - \left[\mathbf{a}^1,\ \ldots,\ \mathbf{a}^p\right]$
4: init $\mathtt{T}_{\text{est}}$ and fill with zeros

5: for $i \leftarrow 1; i \leq C; i \leftarrow i+1$ do
6:     $\varepsilon_{\text{min}} \leftarrow \infty,$
7:     for $j \leftarrow 1; j \leq |\mathcal{F}|, j \leftarrow j+1$ do
8:         $f_j \in \mathcal{F}$ // *Take the $j-th$ feature from the pool*
           // $\mathbf{t}^1_{est}, \ldots, \mathbf{t}^p_{est}$ *are columns of* $\mathtt{T}_{est}$
9:         $\mathtt{L} \leftarrow \left[f_j(I^1, \mathbf{a}^1 + \mathbf{t}^1_{\text{est}}), \ldots, f_j(I^p, \mathbf{a}^p + \mathbf{t}^p_{\text{est}})\right]$
10:        $\mathtt{H} \leftarrow \hat{\mathtt{T}}\mathtt{L}^+$
11:        $\mathtt{T}_{\text{tmp}} \leftarrow \mathtt{HL}$
12:        if $\left(\varepsilon_{min} > \|\hat{\mathtt{T}} - \mathtt{T}_{\text{tmp}}\|_F\right)$ then
13:            $f_{j_{\text{min}}} \leftarrow f_j$
14:            $\varepsilon_{\text{min}} \leftarrow \|\hat{\mathtt{T}} - \mathtt{T}_{\text{tmp}}\|_F$
15:            $\mathtt{T}^* \leftarrow \mathtt{T}_{\text{tmp}}$
16:        else
17:            continue
18:        end if
19:    end for
20:    $F^* \leftarrow F^* \cup \{f_{j_{\text{min}}}\}$
21:    $\mathcal{F} \leftarrow \mathcal{F} \backslash \{f_{j_{\text{min}}}\}$
22:    $\mathtt{T}_{\text{est}} \leftarrow \mathtt{T}_{\text{est}} + \mathtt{T}^*$
23:    $\hat{\mathtt{T}} \leftarrow \hat{\mathtt{T}} - \mathtt{T}^*$
24: end for
```

Figure 5.5: Feature selection algorithm for one predictor

both datasets into training and testing parts. No person from the testing set was present in the training set. Each individual is present either in the training set or in the testing set. The training part of each dataset consists from 70% of the images and the testing part contains the remaining 30%.

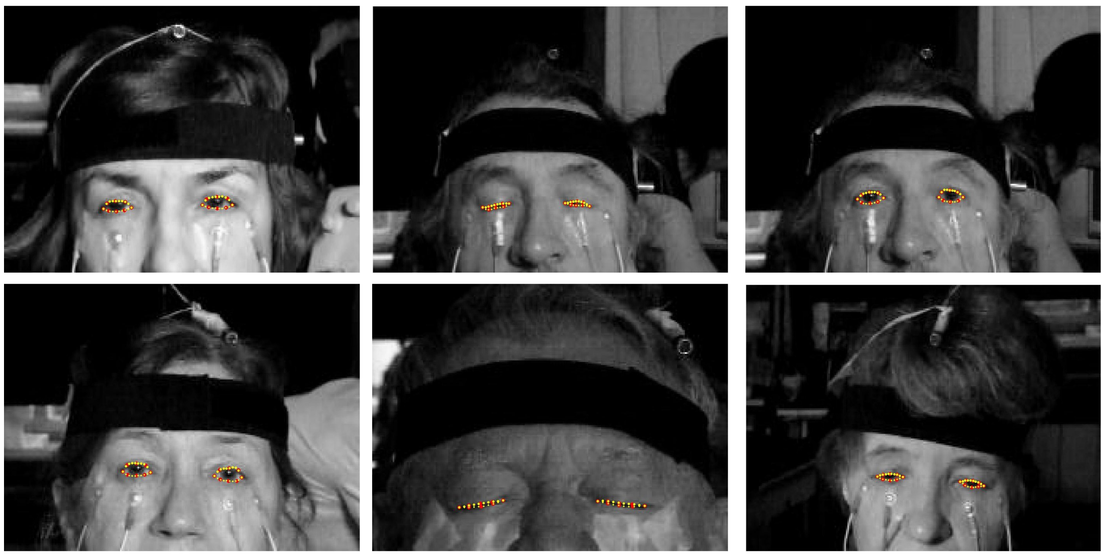


Figure 5.6: Example images from the EyeData dataset with the manually annotated points (red) and with the points generated from the fitted spline function (yellow) which are used as the keypoints driving the thin plate splines elastic deformation of the grid during initial training data preparation. Lower row - example images from the FaceData dataset with the manually annotated points (red) which were directly used as keypoints for the elastic deformation of the grid.

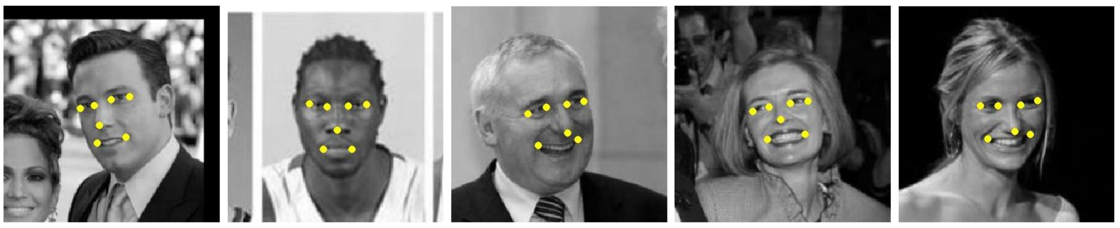


Figure 5.7: Example images from the FaceData dataset (lfw [52]) with the manually annotated points (yellow) which were directly used as keypoints for the elastic deformation of the grid. The dataset consists of images of frontal faces automatically collected from the internet by a face detector.

To evaluate the trackers' performance in each experiment we compute the *mean regression error* (MRE) of one predictor in sequence over either the training data or the testing data. The MRE over the training set for M predictors in sequence and J regressors in each predictor is computed as

$$\mathrm{MRE} = \frac{1}{p}\sum_{i=1}^{p}\left\|\sum_{m=1}^{M}\left(\sum_{\forall f_j \in F_m} r_{mj}(f_j(I^i, \mathbf{a}^i))\right) - \mathbf{t}^i\right\|^2 \tag{5.16}$$

and similarly for the testing set. $\mathbf{t}^i$ denotes the ground truth parameters vector. We use $|F_m| = 10$ features per predictor in all our experiments. In the following experiments we draw several graphs, where there is a discrete variable on the $x-$axis and continuous variable MRE on the $y-$axis. Since we draw many results in each graph, we will draw the results as continuous lines instead of bars for better clarity.

5.2.1 Feature Space Partitioning

Here we evaluate the performance of the γ_s learned with different numbers of bins, which divide each feature space. To generate the results in this experiment we used the piecewise constant functions (5.12) as regressors. The more bins we use, the better precision we may get. But the number of training samples is limited and we need to prevent overfitting. In this experiment we learned the sequential predictors on the EyeData training set with 10 generated samples from each training image. In Figure 5.8 you may see the dependence of MRE on the groving number of predictors in the sequence for different numbers of bins used during learning. With the groving number of predictors in sequence, the testing error continually decreases. Also notice in Figure 5.8 that the training error continually decreases with the growing number of bins. But the testing error shows, that the best performing are γ_s with 9 bins per feature, while γ_s with higher number of bins are overfitted. Another Figure 5.9 better shows the overfitting with higher number of bins than 9.

5.2.2 Piecewise Affine, Linear, Constant Functions Evaluation

In this experiment we compare the performance of different functions (5.10)-(5.12) used in regressors. In Figure 5.10 we may see the results of individual piecewise functions for different numbers of predictors in sequence and different numbers of bins. When we compare the best performing sequential predictors of each type we may see, that the lowest testing error is achieved by the piecewise affine functions. For number of bins higher than 3 the piecewise constant function performs surprisingly well. When we use more than 7 predictors in sequence we get better results than the piecewise linear functions. The piecewise constant function is a perfect choise in the case, when low computational complexity is needed, since the parameters estimation by one predictor costs just J *additions*, while with the piecewise affine it would be J *multiplications* plus J *additions*. When the highest tracking precision is necessary we would use the piecewise affine functions.

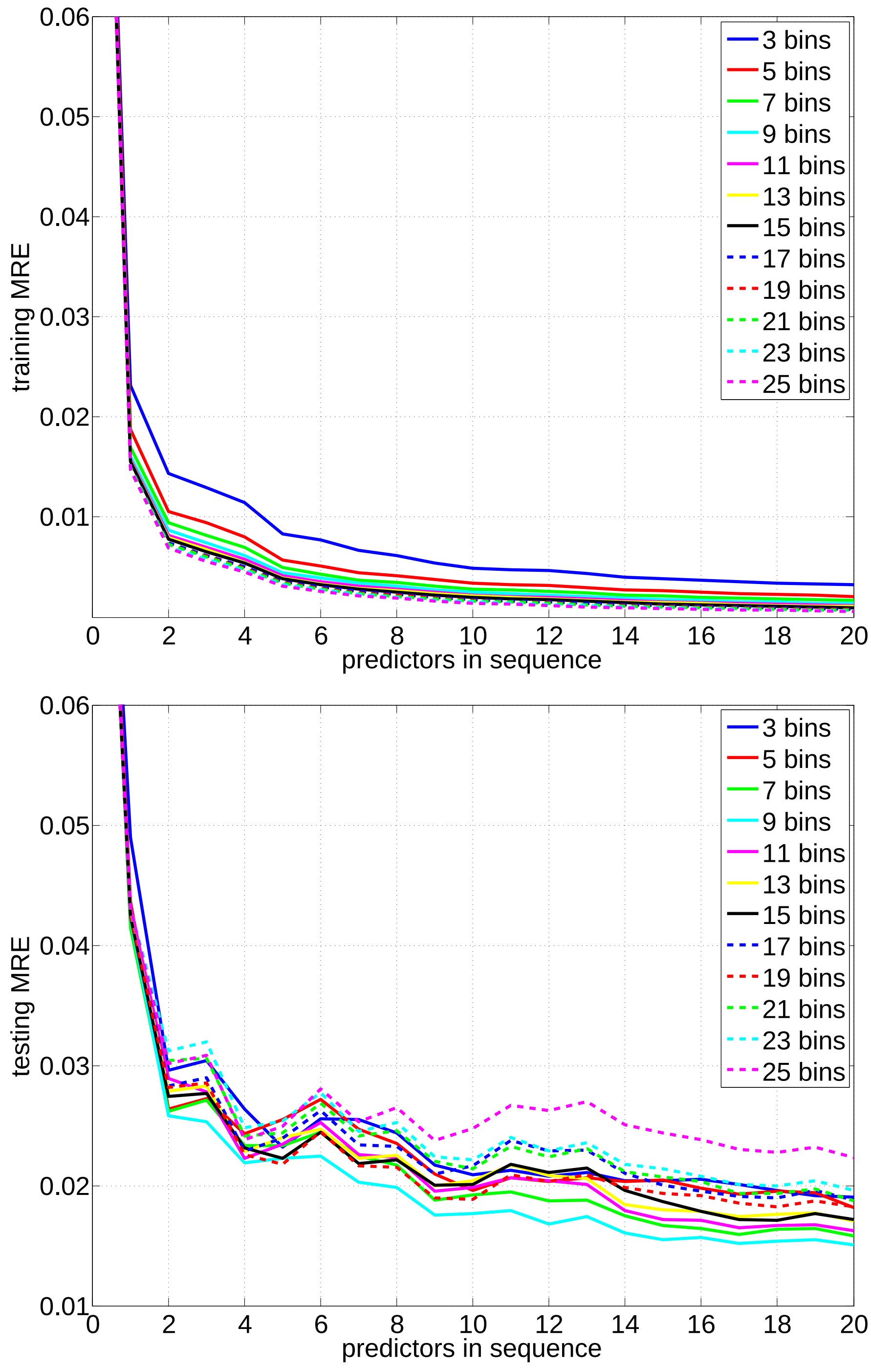


Figure 5.8: The mean regression error over the training and testing parts of the EyeData in dependence on the groving number of predictors in γ_s. Each curve connects sequential predictors learned on the same number of bins.

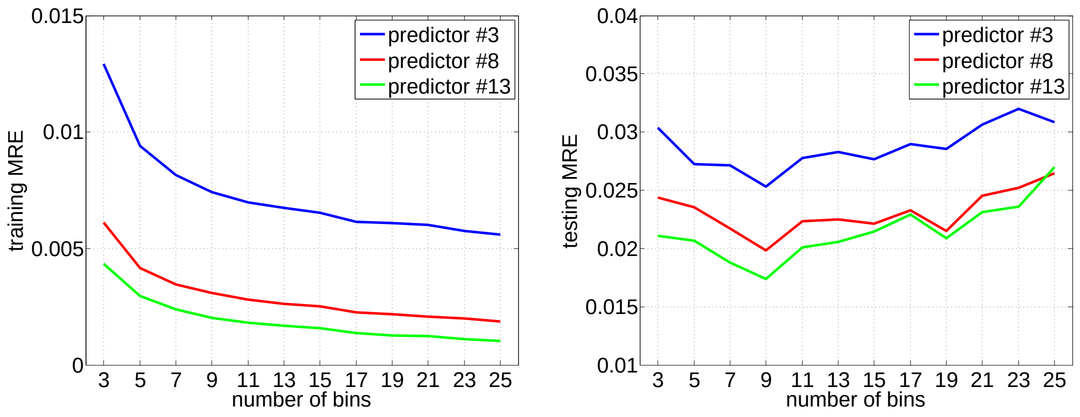


Figure 5.9: The training and testing errors in dependence of the growing number of bins. Three sequential predictors are tested - the blue curve corresponds to γ_s with 3 predictors, the red with 8 predictors and green to γ_s with 13 predictors. We may see, that the training error of all sequential predictors is decreasing, but the testing error shows, that for more than 9 bins every γ_s becomes overfitted.

5.2.3 Additional Training Samples Generation

Generating the additional training samples is usefull only in case, when we are able to generate the deformation from existing samples realistically. When we have enough training samples, the additional samples generation has only a small effect. Yet with the low number of training samples, the artificially generated samples help a lot. See Figure 5.11, where for the γ_s with 8 predictors in sequence the artificially generated samples on EyeData helped to reduce the testing error almost about 50%. Piecewise constant functions (5.12) were used as regressors in this experiment. In the case of EyeData the original number of training samples was, including the flipped images, 10694 and with the artificially generated samples 106940. The original training set size for the FaceData was 16810 and with artificially generated samples 168100.

5.2.4 Features Selection

This experiment shows that the feature selection process significantly increases the precision of tracking. The same testing error is reached with much lower number of predictors in sequence, thus with less evaluated features, than with randomly generated features. Of course the greedy feature selection process requires additional learning time which brings a trade-off between the learning time and tracking precision. In our experiments each predictor in the sequence uses 10 features selected out of 2000 in the feature pool. In Figure 5.12 you may see, that the feature selection algorithm signifficantly lowers the testing errors on both datasets. Again the piecewise constant regressors were used in this experiment.

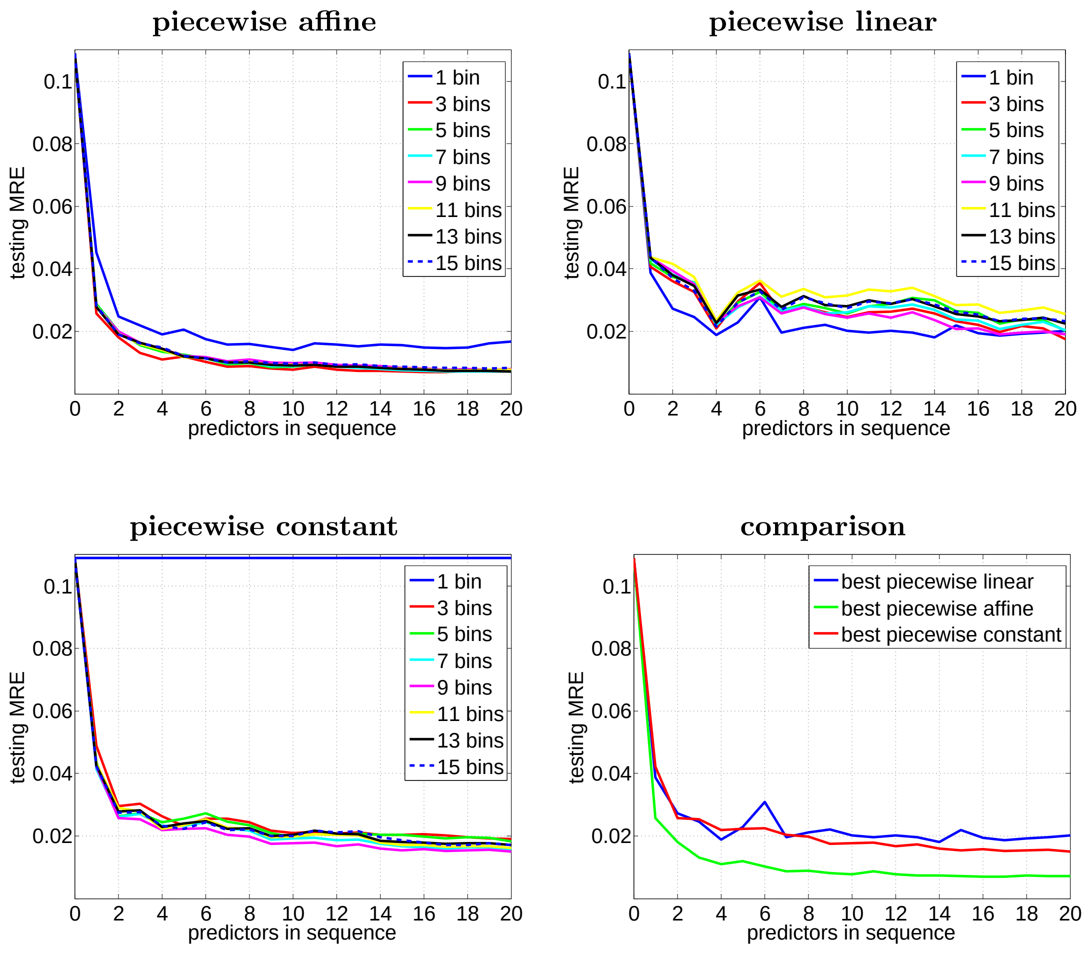


Figure 5.10: EyeData testing errors for different regressors, numbers of predictors in sequence and numbers of bins.

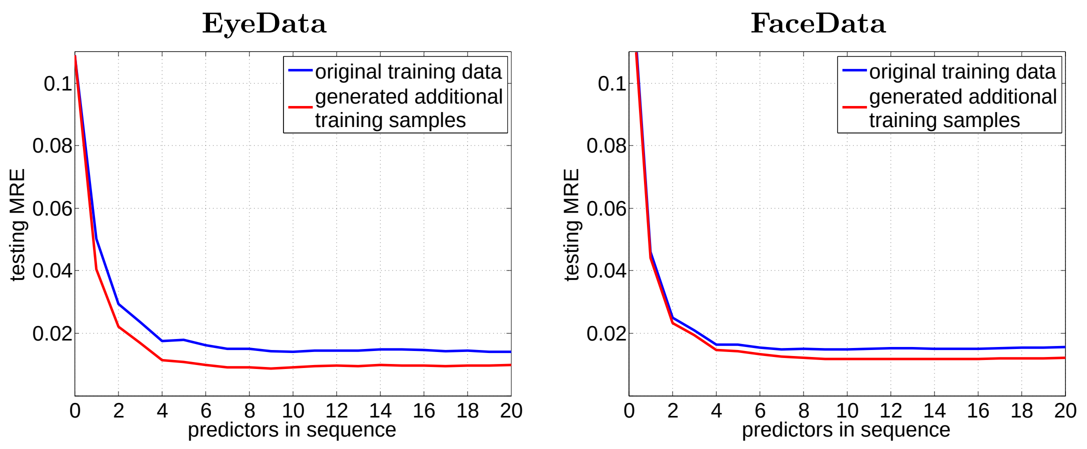


Figure 5.11: Testing errors for both datasets are compared when no artificial training samples were generated (blue curves) and when we generated 10 samples from each training sample (red curves).

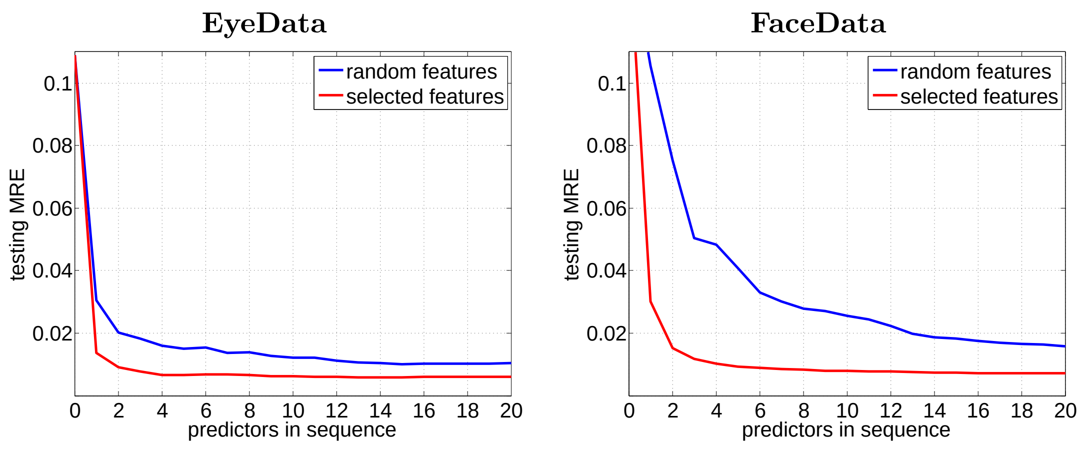


Figure 5.12: Testing errors for both datasets are compared when random features were used (blue curves) and when we used our feature selection algorithm 5.5 (red curves).

5.3 Summary

In this chapter we proposed a method for real-time deformable object tracking. The PCA was used to reduce the dimensionality of the pose space. We proposed to learn the piece-wise affine, linear or constant regressors, which approximate non-linear regression and allow for more precise parameters estimation. Also an artificial training samples generator was presented, which helped to cover the pose space with training samples. And finaly a greedy feature selection algorithm was proposed. Thanks to all the improvements, the precision and speed of the tracker were increased, which was also experimentally verified on two datasets.

6 Detection of LP-trackable Patches

Many computer vision techniques (e.g. 3D reconstruction, simultaneous localization and mapping) and applications (e.g. surveillance, robot's visual navigation) require real-time, reliable and accurate tracking algorithms. The most natural approach to tracking is scanning around the last known position for the maximum response of a criterion function. Since an exhaustive scanning through the whole image is time consuming, the search space is often restricted. If it is known, that the last position is not too far from the current position, a local estimation methods, like steepest descend methods [53, 54] or regression-based methods [55] are often used. Unfortunately, locality of such methods is unavoidable. Each method has a limited range within which it works. The range is usually determined by the maximal inter-frame object displacement. A detector is essentially needed for any tracking application in order to resolve cases when the track is lost. For example the Lucas-Kanade tracker usually re-starts on features that are good to track [9].

Linear predictor is one of the simplest yet powerful regression-based tracking method. Linear predictor (LP) is a linear regression function which maps observed image intensities to motion parameters. It has been shown in [11] that LPs outperform steepest descend method in both the size of the basin-of-attraction and the speed of tracking. However, the off-line learning stage limits their practical usage as a general tracker in an open world environment. To avoid this drawback we propose to pre-train a database of LPs and equip each LP by a detector that finds trackable patches. A naive solution, which would check the neighbourhoods of all image points for LP convergence, would make the usage of LPs prohibitively time consuming. We propose a way how to build the LP-structure-based detector, which detects all LP trackable points, rejects most of the other points and preserves computational cost comparable with standard appearance-based detectors.

The rest of the chapter is organized as follows: Section 6.1 introduces state-of-the-art in linear predictors, Section 6.3 explains the functionality and construction of the detector and finally Section 6.4 shows experimental evaluation of the method.

6.1 Related Work

In this chapter the predictor γ will estimate only the two translation parameters. Given the initial position parameterized by a 2D parameters vector $\mathbf{s}_0 \in S$, where S is the set of all 2D coordinates in the current image I, a tracker estimates motion[1] $\mathbf{t}$ of the object by some function γ:

$$\mathbf{t} = \gamma(I(\rho(X, \mathbf{s}_0))). \tag{6.1}$$

where $X \subset S$ is the set of pixels called *support set* (pixels coordinates spread over the object) and ρ is the transformation, which uses the motion parameters vector and applies it to the object support set points $\rho : \forall \mathbf{x}_i \in X,\ \mathbf{x}_i^{'} = \mathbf{x}_i + \mathbf{s}_0$. The most common way of tracking is repeated minimization of some image similarity (criterion) function $f(\mathbf{t}; I, \mathbf{s}_0)$ given an image I and previous object position $\mathbf{s}_0$

$$\mathbf{t}^* = \operatorname*{argmin}_{\mathbf{t}} f(\mathbf{t}; I, \mathbf{s}_0) = \gamma(I(\rho(X, \mathbf{s}_0))), \tag{6.2}$$

where $\mathbf{t}^*$ is the estimate of the object's motion. Criterion $f(\mathbf{t}; I, \mathbf{s}_0)$ includes implicit or explicit model of object's possible appearances and optionally some relation to $\mathbf{s}_0$. Criterion f could be for example obtained as a similarity function as well as a classifier or foreground/background probability ratio learned from training examples.

By *optimization-based tracking* we understand an on-line optimization technique solving problem (6.2). While some approaches [56, 17, 57, 20] exhaustively search for object in a subset of object positions S with a classifier approximating $f(\mathbf{t}; I, \mathbf{s}_0)$, another approaches [53, 54, 58, 9] use a gradient optimization of some criterion.

Regression-based tracking methods attempt to model explicitly the relationship between image observations and the optimal motion $\mathbf{t}^*$ without the necessity of defining criterion $f(\mathbf{t}; I, \mathbf{s}_0)$. They learn function γ (used in equation (6.1)) in a supervised way from synthesized training data [55, 59, 60]. We outline main principle of the regression-based methods.

The regression-based methods [59, 55, 60] estimate the motion $\mathbf{t}$ *directly* from locally observed intensities on the support set X instead of optimizing the criterion function $f(\mathbf{t}; I, \mathbf{s}_0)$. Such an approach requires a learning stage. Pairs of motions $\mathbf{t}$ and corresponding vector of image intensities $I(\rho(X, \mathbf{t}))$, observed in coordinates X moved by a vector $\mathbf{t}$, are collected and a mapping γ minimizing some error on these examples is estimated, see Figure 6.1,

$$\gamma^* = \operatorname*{argmin}_{\gamma} \sum_{\mathbf{t}^i \in \Omega} \|\gamma\Big(I\big(\rho(X, \mathbf{t})\big)\Big) - \mathbf{t}\|, \tag{6.3}$$

where $\Omega \in S$ is a set of 2D poses in a close neighborhood of the true object pose.

Notice that Lucas-Kanade tracker [53] solves a similar optimization task in each frame, where it needs to compute the image gradient, Jacobian of the warp and pseudo-inverse of the Hessian, see Appendix C for details.. This process can be replaced by using a regression matrix $\mathtt{H}$ learned on a set of synthesized examples. Matrix $\mathtt{H}$ forms a linear mapping between intensities $I(\rho(X, \mathbf{t}))$ and motion $\mathbf{t}$,

$$\mathbf{t} = \gamma\left(I(\rho(X, \mathbf{t}))\right) = \mathtt{H}\left(I(\rho(X, \mathbf{t})) - I(X)\right), \tag{6.4}$$

[1]For simplicity, we talk about *2D position* and *2D motion* but the method also generalizes to more complex transformations.

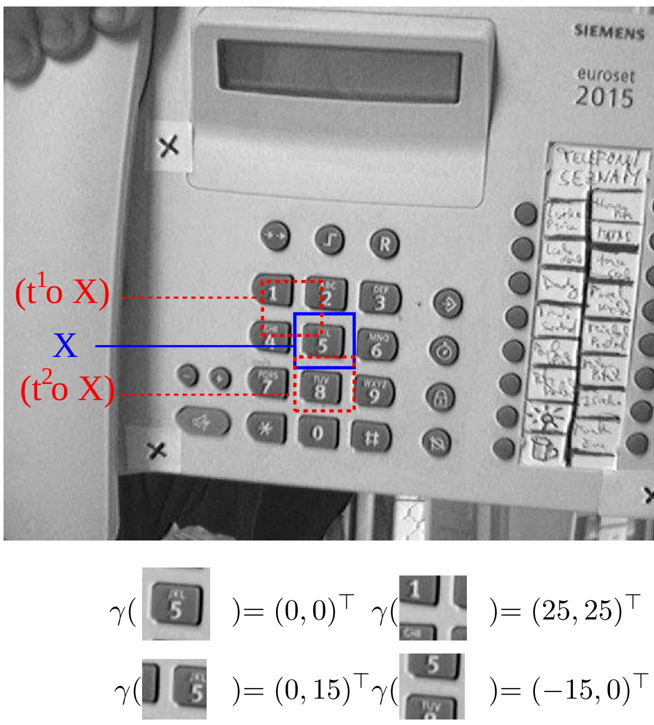


Figure 6.1: **Learning of a linear mapping** between image intensities and motion parameters. The synthetically created training examples (image patches) are collected in a close neighborhood of the object's position under known set of motions. A linear mapping γ between these image samples and motion parameters is than computed using the least squares method.

In the following, the least squares learning of the linear predictor γ is described.

Let us suppose we are given template $\mathbf{J} = I(X)$ and collected training pairs $(\mathbf{I}^i = I(\rho(X, \mathbf{t}^i)),\ \mathbf{t}^i),\ i \in \{1 \dots d\}$ of observed vectors of intensities $\mathbf{I}^i$ and corresponding motion parameters $\mathbf{t}^i$, which aligns the object with the current frame, see Figure 6.1. Then the *training set* is an ordered pair $(\mathtt{I}, \mathtt{T})$, such that $\mathtt{I} = [\mathbf{I}^1 - \mathbf{J}, \mathbf{I}^2 - \mathbf{J}, \dots \mathbf{I}^d - \mathbf{J}]$ and $\mathtt{T} = [\mathbf{t}^1, \mathbf{t}^2, \dots \mathbf{t}^d]$. Given the training set, LP's coefficients minimizing the square of Euclidean error on the training set are computed as follows:

$$\mathtt{H}^* \quad = \quad \mathtt{T}\underbrace{\mathtt{I}^\top(\mathtt{I}\mathtt{I}^\top)^{-1}}_{\mathtt{I}^+} = \mathtt{T}\mathtt{I}^+. \tag{6.5}$$

Since the regression method is very effective it is widely applied in tracking. In particular, Cootes et al. [59, 61] estimate the parameters of Active Appearance Model (AAM) - i.e., deformable model with the shape and appearance parameters projected into a lower dimensional space by the PCA. In [59] a linear predictor (6.4) learned by the least squares method (6.5) estimates all parameters of the AAM. Since the linearity holds only for a small range of parameters, the solution is iterated. Iterations are computed with the same matrix but the length of the optimization step is locally optimized.

This approach was later adapted by Jurie et al. [55] for tracking of rigid objects. Unlike Cootes et al. [59], Jurie's linear predictors estimate local 2D translations only. The global motion is estimated from local motions by the Ransac algorithm, showing the method to be very efficient and robust. Williams et al. [60] extended the approach to the non-linear motion predictors learned by Relevance Vector Machine [62] (RVM). Agarwal and Triggs [63] used RVM to learn the linear and non-linear mapping for tracking of 3D human poses from silhouettes. Another extension was suggested by Zimmermann et al. [11] who proposed an optimal way to concatenate several regression functions into a sequential predictor. Different learning techniques have also been proposed, e.g. Drucker et al. [64] search for the regression function that has at most certain deviation from the actually obtained poses. Zhou et al. [65] proposed greedy learning for additive regression functions, using weak regressor formed of a linear combination of binary functions.

6.2 Contribution

We show that an LP allows to track many points it has not been trained for, for examples see Figure 6.2. Notice, that appearance of the set of LP trackable patches are very discrepant from the set of training patches. Such points could not be detected by any standard detector trained on the appearance of training examples – simply because the trackability rather stems from the structure of the LP than from the appearance of the training samples.

We propose an efficient detector of LP trackable points which (i) does not require any time consuming learning (ii) detects all trackable points and (iii) has computational costs comparable with standard appearance based detectors. The detector construction is described in the following section.

6.3 Detector of LP Trackable Points

Different forms of LPs provide different sensitivity to object appearance - the more degrees of freedom, the more general the LP is, but the longer learning is needed. Two following forms of LPs are studied:

$$\begin{aligned}\textit{Basic LP:}\quad \mathbf{t} &= \mathtt{H}I(\rho(X,\mathbf{t})), && (6.6)\\ \textit{Extended LP:}\quad \mathbf{t} &= \mathtt{H}(I(\rho(X,\mathbf{t})) - I(X)). && (6.7)\end{aligned}$$

The *basic LP* uses directly the vector of observed image intensities $I(\rho(X,\mathbf{t}))$ whereas the *extended LP* substracts the object template $I(X)$ from $I(\rho(X,\mathbf{t}))$. The extended version allows the tracker to be more variable, as will be seen in Section 6.4. Note, that

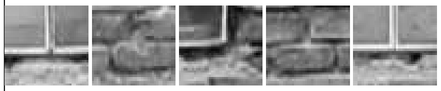
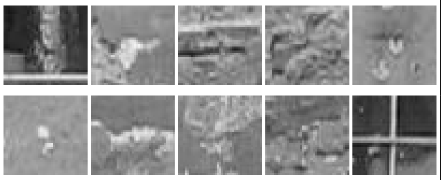
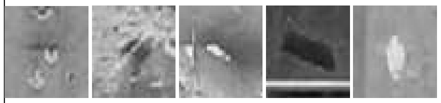
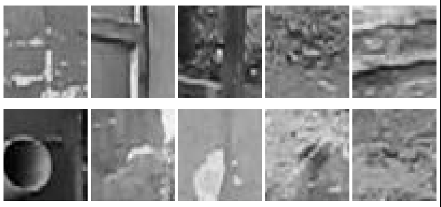
	All training patches	Some of the trackable patches
LP1		
LP2		

Figure 6.2: **Examples of patches** used for learning (middle column) and some of *trackable patches* (right column) for 2 different LPs are shown for visual comparison. You may notice, that the trackable patches, which were found by our detector, are not visually similar to the patches used for training.

although we speak about *Linear* predictors (because of historical reasons), huge class of possible non-linear extensions is at hand (e.g. polynomials can be treated as linear combinations of monomials).

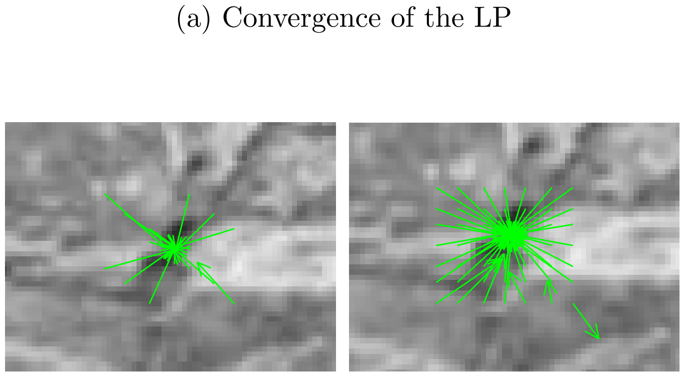
The simplest way to detect the a trackable point would be to build a naive detector which evaluates LP's convergence (see Figure 6.3a) at every single point (rotation and/or scale) in the image. Let us consider that we want to check convergence of a simple LP with coefficients (regression matrix) $\mathtt{H}$ and support set X. Convergence of the LP means that each row $\mathbf{h}_i^\top$ of $\mathtt{H}$ and corresponding element t_i^j of every local perturbation $\mathbf{t}^j$ from the considered neighbourhood $R = \{\mathbf{t}^1, \mathbf{t}^2, \ldots, \mathbf{t}^n\}$ has to satisfy

$$\forall j \quad \mathbf{h}_i^\top I(\rho(X, \mathbf{t}^j)) = t_i^j + \mathbf{h}_i^\top I(X) + \Delta^j \tag{6.8}$$

for reasonably small prediction errors Δ^j, where $I(\rho(X, \mathbf{t}^j))$ is a vector of intensities collected in the support set X transformed by local perturbation $\mathbf{t}^j$. For the sake of simplicity, row index i is further omitted.

This approach is, however, reasonably applicable just for LPs in the basic form and it can easily become prohibitively time consuming. Therefore, we propose sufficiently fast detection method of LP trackable points, applicable for more general forms of LPs, e.g. Equation (6.7) or other [66].

Main idea is based on the fact that checking the convergence of an LP on some region is almost equivalent to checking whether the mean and variance of the prediction errors



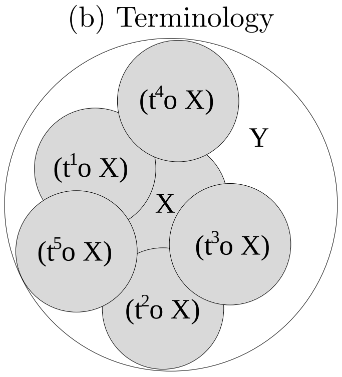


Figure 6.3: **(a) Convergence of the LP** from a neighbourhood of a single point (predictions are depicted by green arrows). Arrows point from the LP's initial position to the predicted object center. Left - depicts only a few of used votings for better visualization. Right - shows all the votings from the neighborhood R used in our algorithm. When all the arrows point to one pixel (or close enough), which is also the currently evaluated point, than the point is trackable. When the arrows point to some random directions, the point is not trackable by particular LP. **(b) Terminology:** X is the set of 2D coordinates, called support set. $(\rho(X, \mathbf{t}^j))$ is the support set transformed by local motion perturbation $\mathbf{t}^j$ and Y is union of all perturbations of the support set, i.e. $Y = \bigcup_{\mathbf{t}^j \in R}(\rho(X, \mathbf{t}^j))$.

Δ^j are close to zero. We first introduce the set of all pixels

$$Y = \bigcup_{\mathbf{t}^j \in R} (\rho(X, \mathbf{t}^j)), \tag{6.9}$$

used in linear system (6.8), see Figure 6.3b. Then the Equation (6.8) is rewritten to the intensity independent form as follows:

$$\forall j \quad \mathbf{h}^\top I(\rho(X, \mathbf{t}^j)) = \mathbf{f}^{j\top} I(Y) = t^j + \mathbf{h}^\top I(X) + \Delta^j, \tag{6.10}$$

where $\mathbf{f}^j$ consists of suitable permutations of elements of $\mathbf{h}^\top$ and zeros. Since $I(Y)$ contains all elements of $I(X)$ we can express prediction error in the following form

$$\Delta^j = [\mathbf{f}^{j\top} I(Y)] - [t^j + \mathbf{h}^\top I(X)] = \mathbf{w}^{j\top} I(Y) - t^j. \tag{6.11}$$

If the point is *trackable* then the prediction errors Δ^j has distribution $\mathcal{F}(\mu, \sigma)$ with both mean $\mu(\Delta)$ and variance $\sigma^2(\Delta)$ *close to zero* which is further denoted as $\mu(\Delta), \sigma^2(\Delta) \approx 0$. If $\mu(\Delta)$ or $\sigma^2(\Delta)$ gets far from zero, then there exist local perturbation(s) around the selected point, which cannot be compensated by the LP. It means, that to reject the hypotheses that the point is trackable by a given LP, it is not necessary to check all the linear equations in system (6.10), but we can easily check *necessary but not sufficient condition*:

$$\mu = \frac{1}{n}\sum_j \Delta^j = \underbrace{\Big(\frac{1}{n}\sum_j \mathbf{w}^{j\top}\Big)}_{\mathbf{w}^\top} I(Y) - \underbrace{\Big(\frac{1}{n}\sum_j t^j\Big)}_{b} = \mathbf{w}^\top I(Y) - b \approx 0, \tag{6.12}$$

the computational cost of which is the same as the computational cost of just one equation of the linear system (6.10). This condition will be insufficient for example in the case, where all LP predictions has got flipped signs or in the case where $I(Y)$ is constant (i.e. gradient is zero). While the first case is very rare, the second case is quite often. Image areas with constant (or almost constant) intensity function are inherently not trackable and can be eliminated in advance.

Similarly the variance $\sigma^2(\Delta)$ can be expressed as the following sparse quadratic form:

$$\begin{aligned}
\sigma^2 &= \mathcal{E}((\Delta^j)^2) - \mathcal{E}(\Delta^j)^2 = \frac{1}{n}\sum_j \big(\mathbf{w}_j^\top I(Y) - t^j\big)^2 - \mu^2 \\
&= I^\top(Y)\underbrace{\Big(\frac{1}{n}\sum_j \mathbf{w}^j\mathbf{w}^{j\top}\Big)}_{\mathtt{A}} I(Y) - \underbrace{\Big(\frac{2}{n}\sum_j \mathbf{w}^{j\top}t^j\Big)}_{\mathbf{b}^\top} I(Y) + \underbrace{\Big(\frac{1}{n}\sum_j (t^j)^2\Big) - \mu^2}_{c} \\
&= I(Y)^\top \mathtt{A} I(Y) - \mathtt{b}^\top I(Y) + c \approx 0,
\end{aligned} \tag{6.13}$$

where $\mathtt{A}$ is a sparse, positive-semidefinite and symmetric matrix, $\mathbf{b}^\top$ is a vector and c is a scalar. Sparsity of $\mathtt{A}$, which is crucial for the computational efficiency, depends on the set of considered local perturbations R. We observed that around 80% of its elements are equal to zero. All the coefficients needed for μ-test, i.e. Equation (6.12), and σ-test, i.e. Equation (6.13), are off-line directly created from the elements of $\mathtt{H}$, the on-line detection means only a few hundreds of scalar multiplications per point.

The point trackability is determined by the resulting μ and σ, which should not be bigger than corresponding threshold values Θ_μ and Θ_σ. If none of the values (μ or σ respectively) is bigger than threshold (Θ_μ or Θ_σ respectively), than the point is trackable by a particular LP, othervise it is not trackable. Note, that in our experiments 99.8% of not trackable points were already filtered by the μ-condition and the σ-condition was usually evaluated on only a few points. Note, that the same detector can be constructed for the basic LP by omitting term $\mathbf{h}^\top I(X)$ in Equation (6.11).

6.4 Experiments

Experiments were performed on two video sequences, 1548 frames and 2595 frames long. Moving camera captured walls with windows, see Figure 6.4 for few example frames. Ground-truth inter-frames homographies were computed from manually labeled 4-point correspondences in both sequences, in order to be able to detect the loss-of-track of tested

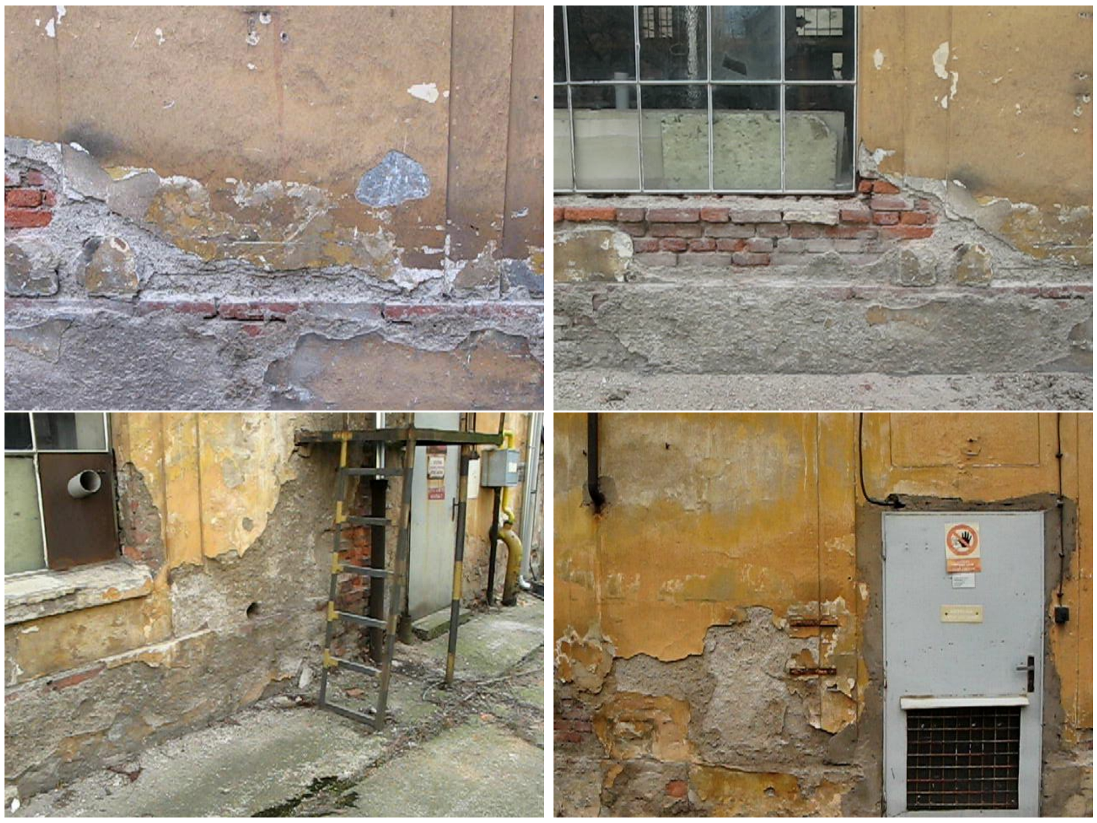


Figure 6.4: **Example images** from tested sequences. The sequences contained mainly 2D translations with small scale changes and rotations. The tracked objects were mainly planar walls, windows or doors.

LPs. The LPs used in our experiments predicted 2D motion only. The tested sequences contain mainly 2D motion with small scale changes and small in-plane rotations to see the robustness of tested LPs. We demonstrate the results of experiments by graphs of average track length L on horizontal axis for particular number of trackable patches N on vertical axis. Changing the thresholds $[\Theta_\mu, \Theta_\sigma]$ for detector changes the number and quality of detected and tracked points, which than generates various points in these $NL-$diagrams. We evaluate the results for one (Figs. 6.6 and 6.6) up to six LPs (Fig. 6.5) trained on different patches for various $[\Theta_\mu, \Theta_\sigma]$ thresholds.

Comparison of the basic and extended linear predictor: μ and σ conditions are in practice evaluated with respect to some thresholds Θ_μ and Θ_σ. The higher thresholds the more points are accepted but the lower is the average length of the track because of the worse local convergence of the LP. Figure 6.5 shows the threshold combinations for basic LPs (in blue) and extended LPs (in red). The results show the average LP performance

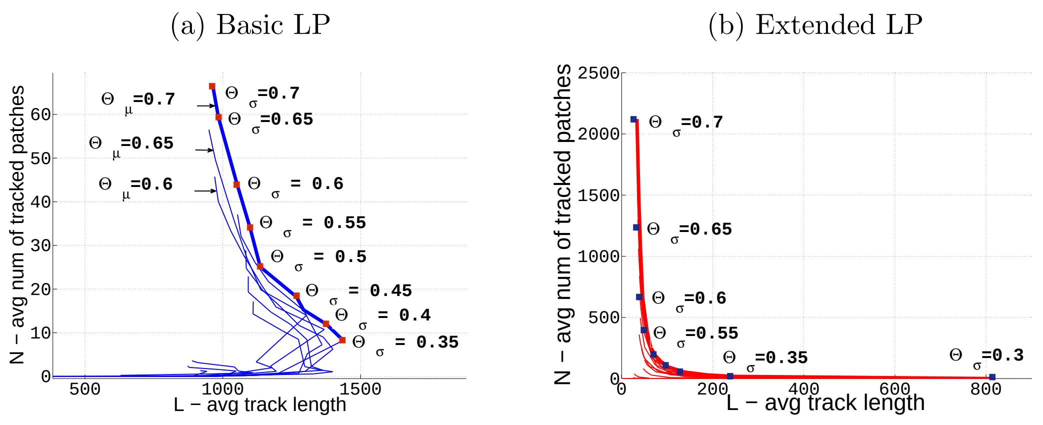


Figure 6.5: **Comparison of the (a) Basic LP and (b) Extended LP:** Lines connect threshold combinations with fixed Θ_μ, pareto-optimal threshold combinations are emphasized by the thick lines. Both predictor versions have different properties. The basic LP is able to track lower number of patches for more frames, while the extended LP is able to track a lot of patches, but looses the track more frequently.

computed from six distinct LPs over 2 sequences. Lines connect combinations with fixed Θ_μ. The ideal LP with ideal detector would have all threshold pairs on the most right vertical line, since it would allow to track all the points for as long as possible. The pareto-optimal threshold combinations, which are emphasized by the thick line show the best performance, which may be obtained with particular set of patches for two tested sequences. Both predictor types have different properties. The extended LP is alowes to track a high number of patches for a smaller number of frames (when averaged) than the basic LP, which tracks a lower number of patches for higher number of frames. The higher adaptibility of the extended LP on possible patch appearances, caused by the template $I(X)$ substraction in equation 6.7. On the other hand the extended LP has zero mean on image regions with constant intensities (zero image gradient), which requires to compute the time consuming σ-test in more image points, than for the basic LP. So for the extended LP, it usually takes a little longer to detect the trackable points.

Using one or more patches to train the LP: On Figure 6.6 you may see the comparison of performance of LP trained for one patch (object) and the same LP, which was incrementally trained for another two patches. Adding a few training examples improves the LP's *generality*.

The numbers of patches for learning of the LPs used in our experiments vary from one to five. Some LPs (used in all experiments) were trained on patches, which appear in the tested sequences, but most of them were trained on completely different patches and objects, which do not appear in the tested sequences. The aditional training patches

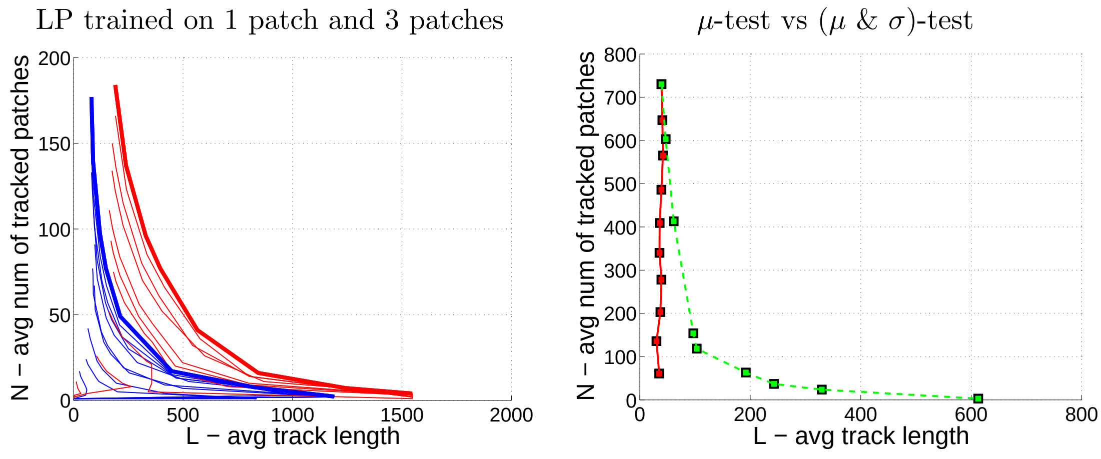


Figure 6.6: **Left:** Single versus multiple patch learning. Comparison of the performance of single LP trained for one image patch (blue) and the same LP which was incrementally trained for another two patches. The aditional two training patches improved the predictor performance although the right number of training patches and the criteria for their selection is still not solved. **Right:** NL-diagram of the points detected only by the μ-condition (in red) and points detected by both conditions with pareto-optimal thresholds (in green) for single LP.

were selected manually and generally contained some distinctive visual features (e.g. harris corners or well textured areas). The automatic selection of training patches (and their number) for the most general LP is clearly an important issue and it will be the focus of our future work.

Importance of the σ-test: In this experiment, we show the importance of the σ-test. We compare NL-diagram of the points detected only by the μ-condition (i.e., with $\Theta_\sigma = \infty$) and points detected by both conditions with pareto-optimal thresholds, showing that σ-test yields important improvement. Results are summarized in Figure 6.6.

6.5 Summary

We have designed a simple method for detection of LP trackable points. Depending on the μ and σ thresholding, one linear predictor may successfully track around 100 different patches, for which the LP was not trained. The detector is very efficient and does not slow down the learing phase, because it is constructed directly from the learned LP. An interesting opened question is how to select the proper patches in the off-line training phase. For the moment we select examples manually.

7 Cascade Object Detector with Sliding Window Alignment

State of the art *real-time* object detectors use a sliding window to evaluate different positions in image. With the growing image resolution grows the number of pixels, which need to be evaluated often in more scales, in which the object might appear, see Figure 7.1. Due to the framerate requirements we must evaluate only a sparser grid of pixels, so the sliding window moves with some step larger than one. We simply rely that the detector is robust enough to deal with this small perturbations in position. The sparsity, however, causes weaker responses on positive samples, see Figure 7.2(a). We observed, that if a sliding window on the sparse grid is aligned prior to the detection, higher response in terms of both *confidence* (i.e. the output of the classifier) and *score* (i.e. number of multiple detections) is achieved, see Figure 7.2(b). This approach consequently decreases the number of False Negatives (FNs) for a fixed detection threshold.

Unfortunately, windows containing negative samples are aligned in the way which increases the chance of detector's positive response increasing thus also the number of False Positives (FPs). Nevertheless, we show that if the alignment is invoked after few detection stages in the classifier cascade, i.e. only on the surviving fraction of detection windows, significantly better results are achieved.

7.1 Related Work

Among other detection methods (SVM, Ferns, SIFT, etc.) we choose cascade of AdaBoost classifiers. In comparison with other object detectors it is more universal and achieves impressive results in many real-world applications. An extensive overview of boosting algorithms and AdaBoost extensions may be found in [67], [68] and [69].
Adaptive boosting (AdaBoost) is a machine learning algorithm that creates one strong classifier from a large number of user-provided weak classifiers. AdaBoost is adaptive in the sense that subsequent classifiers are tweaked in favor of those instances misclassified by previous classifiers. Freund and Schapire's AdaBoost [70] algorithm for classification has attracted much attention in the machine learning community. Brieman [71] [72] showed, that the AdaBoost algorithm can be represented as a steepest descent algorithm in function space which is called functional gradient descent. Friedman et al. [73] explained the AdaBoost algorithm from a statistical perspective. They showed that AdaBoost algorithm is a Newton method for optimizing a particular exponential loss function. They propose Real AdaBoost (based on [74]) in which each weak estimator returns a class probability estimate to construct real-valued contributions. They also

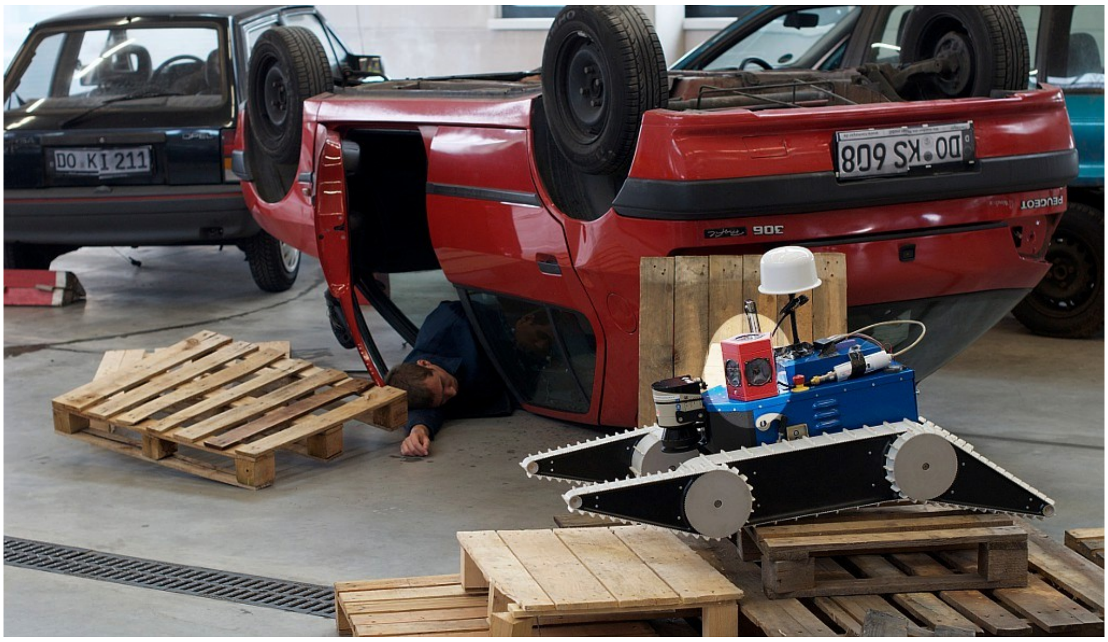


Figure 7.1: Our rescue robot with Ladybug3 omnidirectional digital camera on board. The camera has six 2Mpix chips and we want, among other tasks, to detect cars in five of them in real-time.

propose a Gentle AdaBoost algorithm. The Gentle AdaBoost is a modified version of Real AdaBoost by using Newton stepping instead of exact optimization at each step of classifier learning. In this work we use a Cascade of Gentle AdaBoost detectors. Ratsch et al. [75] find that AdaBoost asymptotically achieves a hard margin distribution, i.e. the algorithm concentrates its resources on a few hard-to-learn patterns, which is a sub-optimal strategy in the noisy case. They propose several regularization methods to achieve *soft margins* to deal with the outliers in training data. Some authors [76] [68] modified the loss function to use AdaBoost for regression. Perrotton et al. [77] use Gentle Adaboost with a different families of descriptors and soft cascade structure [78]. They also modify the way of building the weak classifiers in order to build multi-view object detector. At each boosting level they choose multiple weak classifiers allowing to encode multiple object views (appearances). [79] present a On-line Multi-Class Linear Programming Boost, where they try to maximize the multi-class soft-margin of the samples. To solve the on-line linear programming problem, they perform a variant of convex programming.

The *classifier cascade* [80][18][81][82][83] evaluates a candidate window by applying a sequence of learned classifiers. To accept evaluated sample, all the classifiers must vote for positive detection. This gives the opportunity for rejecting many candidates in early stages making thus the detector very effective. Also we might use the knowledge that

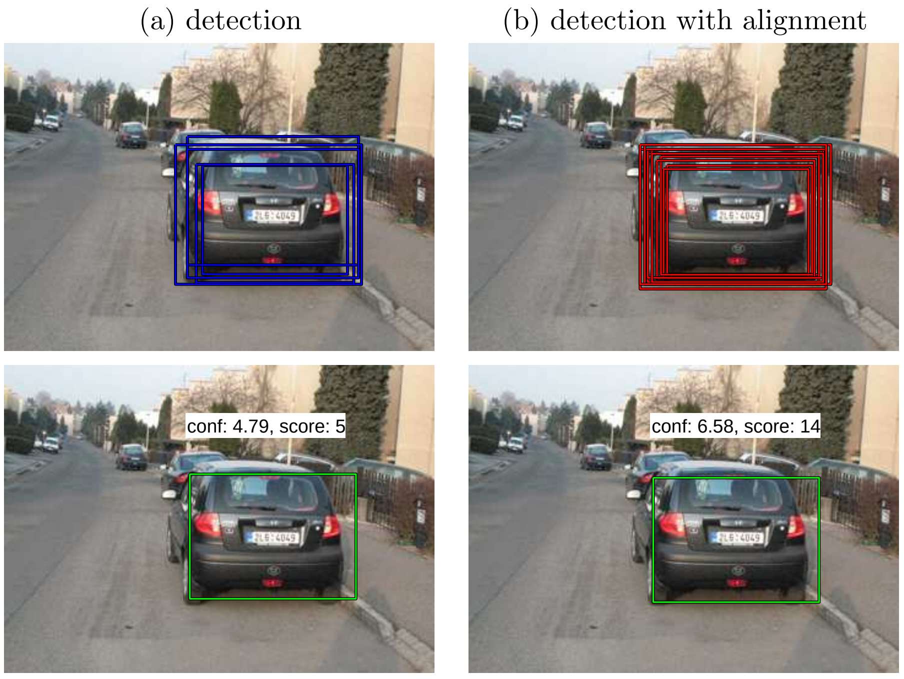


Figure 7.2: **Fundamental idea:** First line shows detected windows, second line shows results after non-maxima-suppression.

last stages are evaluated much less often, and place the more time consuming classifiers close to the end of the cascade.

In our experiments we test two different motion estimators. First is a sequential linear predictor γ_s similar to Zimmermann et al. [11]. The sequential predictor is learned from multiple training images to estimate 2D motions of specified class of objects. As a second motion estimator we use Ferns classifier similar to [44], where different classes correspond to discrete 2D motions. Linear predictors are widely used for tracking [10] [13] [34]. Incremental on-line learning was proposed by Hinterstoisser et al. [12]. Recently, linear predictors were used by Holzer et al. [49] for adaptive template tracking. The tracked template was enlarged or reduced on-line, when necessary (e.g. due to partial occlusion). Very similar method to the original [11] is [84], where they also use a sequence of pose regressors to estimate object's position, only they replace the linear predictors in sequence with a random Ferns regressors. Hinterstoisser et al. [12] use Ferns to detect planar objects and linear predictors to estimate the homography transformation for precise alignment. Recently, Villamizar et al. [85] used boosted random Ferns for rotation invariant object detection.

We did not find any similar work, which would try to improve the performance of the detector by aligning the detection window, as we do. We try to reduce the search space, get higher confidence on true positives, lower confidence on false positives and reject the false positives in an earlier detection stage. From the point of view of search space reduction our work may be compared with Felzenschwalb [18], where they use a partial hypothesis pruning with a sequence of thresholds. They show, that the sequence of thresholds provides a theoretical guarantee on the performance of cascade method. They use a part-based model and achieve another speed-up by detecting the root part of the object first and finding the best configuration of the remaining parts later, instead of detecting every part separably.

7.2 Theory

We assume a detector which comprises a cascade of 25 Gentle AdaBoost classifiers [73]. Each classifier in the cascade evaluates a particular example (window candidate). The example is accepted, i.e. classified as a positive match if and only if all the classifiers in the cascade accept it. Each classifier has its own fixed position in the cascade. We call this position the *cascade stage*.

To align the sliding window we test two different methods. The sequential linear predictor [11], which is a sequence of linear mappings between intensities and motions. Second motion estimator is Ferns [44] which is a randomized tree-like classifier learned to distinguish several discretized positions. Both motion estimators as well as the detector and are learned off-line on multiple training images and work with rectangular *Haar features*. The Haar features are computed as a difference of sums of image intensities over two (or more) image areas. This type of features became popular thanks to Viola-Jones face detector [15]. Rectangular Haar features are sensitive to horizontal and vertical structures in the image, which is in our case suitable to the car detection task for our experiments. The rectangular feature shape may be computed very efficiently using the integral image (summed area table). The motion estimators, adapted to work with Haar features, are able to localize the object from its local neighborhood (learned motion range). But when they are initialized on some random patch in the image, their response is unpredictable. The motion estimators are learned on positive examples only. They have no information about the background, or other objects in the scene, unlike the detector, which is trained on both positive and negative examples. The detector and motion estimator may be combined in several different ways. We might run the sliding window alignment just before every detection, but this would significantly slow down the entire detection procedure, because of the large number of evaluated windows in image. It is around 250 000 candidate windows for resolution 640×480 and detection step 3 pixels in both directions. To keep the detection fast we let the first few classifiers in the cascade to filter out most of evaluated positions, than we align the sliding window and evaluate the remaining classifiers on a changed position, see Figure 7.3.

We observed that the cascade stage after which the alignment is invoked (we further address to this stage as *the alignment stage*) is crucial: while running the alignment at the very beginning or the very end of the detection process yields negligible improvement, an alignment invoked after approximately $50-60\%$ of detection stages yields significantly

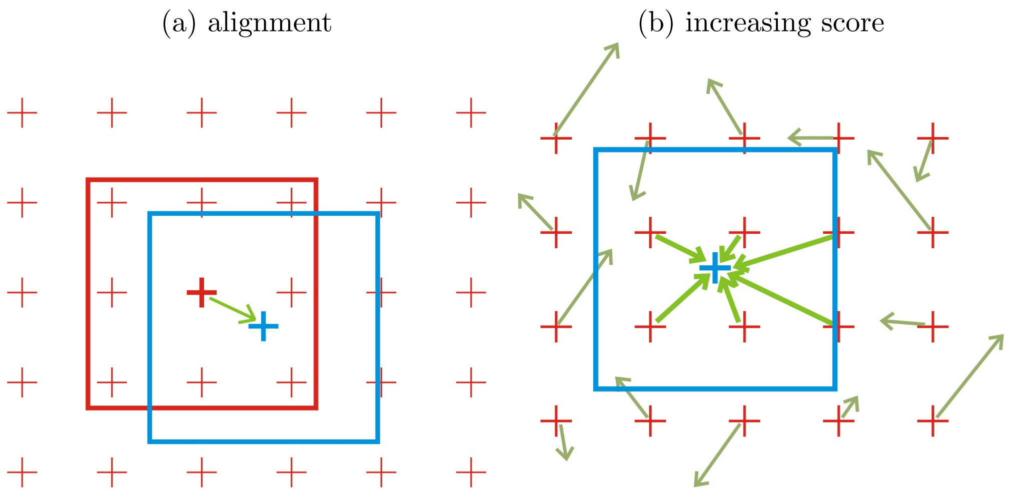


Figure 7.3: **(a) Alignment:** Sliding window is centered only on a sparse grid of pixels (red crosses). First few stages of the detector cascade are evaluated on this initial position (red rectangle). Than the motion estimator votes for object center position, where the detection window is moved (blue rectangle). Here the remaining stages of the detector cascade are evaluated and the result of detection is assigned to the aligned position (blue cross). **(b) Increasing score:** When we use the sliding window alignment we get higher score (number of overlapping detections) on positive sample (blue rectangle), see Figure 7.2(b). Higher score on positive sample is caused by sliding window alignment on its position from multiple initial positions around it.

better results. In addition to that, the detection speed is preserved, since the alignment is computed only on less than 0.05%[1] of rectangles.

In general, we observe that the higher the alignment stage, the fewer the false positives (FPs) but the more the false negatives (FNs). To explain this phenomena, we define *detection complexity* of a negative sample as the last stage at which the sample is accepted by the cascade of classifiers. If a negative sample is accepted by all detection stages, i.e. it is false positive, the detection complexity is equaled to the number of cascade stages.

First we discuss why is the number of FPs a decreasing function of the alignment stage, see Figure 7.4 (left). Aligning of a simple negative sample (i.e. sample with a low complexity), often yields more FPs than aligning of a hard negative sample (i.e. sample with a high complexity). The reasons are two-fold: (i) there are many more simple negative samples than hard ones[2] and (ii) most of the hard negative samples are often already aligned in the worst possible way and further alignment makes them often less complex.

[1]Actually, for alignment stage 15 it is $0.6^{15} = 0.00047$, because the rejection rate per stage is 60%.

[2]Simply because the cascade is trained in the way, that the first stage rejects 60% negative samples, the second stage rejects 36% (i.e. $0.6^2 \cdot 100$) of negative samples etc.

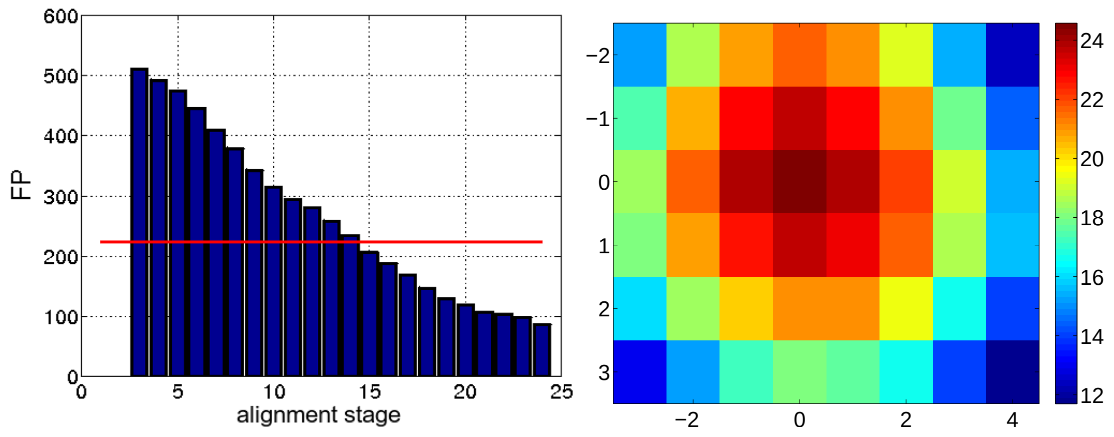


Figure 7.4: **Left: *FP* as a function of the alignment stage:** Graph shows that FP are decreasing function of the alignment stage. Number of FP of the not aligned detector is denoted by red line. This graph is valid for one particular confidence, i.e. for one point on ROC curves (see Figure 7.5). Similar graphs may be drawn for other confidence values. **Right: Average positive samples rejection stage** (i.e. FN only) as a function of displacement from the correct position $(0, 0)$.

We also observed that only a minority of positive samples were rejected in the first stages due to their bad alignment. Figure 7.4 (right) shows the average rejecting stage on positive samples as a function of the spatial alignment (similar effect can be observed in scale). We can observe that the smallest rejection stage is around 12 - far from the early decision stages. This in conjunction with the diminishing FP amount (shown in Figure 7.4), explains why early alignment does not pay off. Second, we justify that the number of FNs is an increasing function of the alignment stage. The higher the alignment stage, the fewer not-well-aligned positive samples survive and therefore the fewer positive samples are correctly aligned and the detector has the fewer chances to detect them with sufficiently high confidence. This means that too late alignment does not help neither. The above mentioned claims are experimentally validated in Section 7.3.

7.3 Experiments

The system is evaluated on the car rear detection problem, see Figure 7.6 for a few examples. We have created a dataset of 1800 images of cars with manually labeled bounding boxes. We divided the dataset into 900 training and 900 testing images. Training images were used for learning of the detector as well as for learning of both motion estimation methods. We have used another set of images not containing any cars as the negative samples for the detector learning. The results, we present, were obtained by evaluation of the method on the testing images only.

The detector consists of a cascade of 25 Gentle AdaBoost classifiers. The alignment is trained within the range of 6 pixels (no scale, rotation or even non-rigid alignment is considered). The alignment is estimated either by the γ_s (with three predictors in sequence) running on approximately 1000 randomly selected haar features, or by the Ferns classifier (consisting of 50 randomized trees with depth 11) distinguishing classes corresponding to discretized translations with 1 pixel accuracy.

We evaluate detector (blue), detector with the linear predictor (red) and detector with the Ferns (green) with the alignment stage equal to 5, 10, 15 and 20. Figure 7.5 shows ROC curves for this four cases. To demonstrate the influence of the alignment stage, the corresponding points (with the same confidence threshold) on different ROC curves are outlined by black asterisk and black triangle symbol.

As we can see, Figure 7.5(a), the early alignment yields

- **fewer FNs**, because most of not-well-aligned positive samples are correctly aligned before they are rejected,
- **more FPs**, because the earlier the alignment the higher the FPs as shown in Figure 7.4.

Too late alignment, Figure 7.5(d), works the other way around. The positive samples, which pass up to the high cascade stage are of two kinds. Either easy samples, which don't need to be well aligned for positive detection or hard samples, which are very sensitive to precise alignment, which brings

- **more FNs**, because in the high cascade stage the hard positive samples may be paradoxically damaged by the motion estimator, which also has some small estimation error on positive samples. Even one pixel error in alignment causes final rejection of those hard samples and rises the FNs.
- **fewer FPs**, because more potential FPs are incorrectly aligned. I.e. the motion estimator drifts away and FPs are rejected.

The best trade-off of above mentioned effects is achieved with the alignment stage in between 10-15, see Figure 7.5(b,c). We can see that the Ferns outperforms the sequential predictor, especially in higher alignment stages. In addition to that, when looking at Ferns ROC curves, one can notice that the Ferns alignment exhibits very desirable property of incorrectly aligning hard negative samples, which yields good results even for the high alignment stage. We assume, that this could be caused by significantly higher degrees of freedom of the Ferns with respect to the sequential predictor. On the other hand motion estimation by Ferns is more time costly (approximately 3 times slower) than by the sequential predictor.

The question of time complexity is also important. In a real-time application it would not be possible to align every candidate window. Although the alignment is very fast (0.5 ms per one window by γ_s implemented in Matlab), the candidate windows are simply too many (around 250 000 per image). While after lets say 15 stages of the cascade we

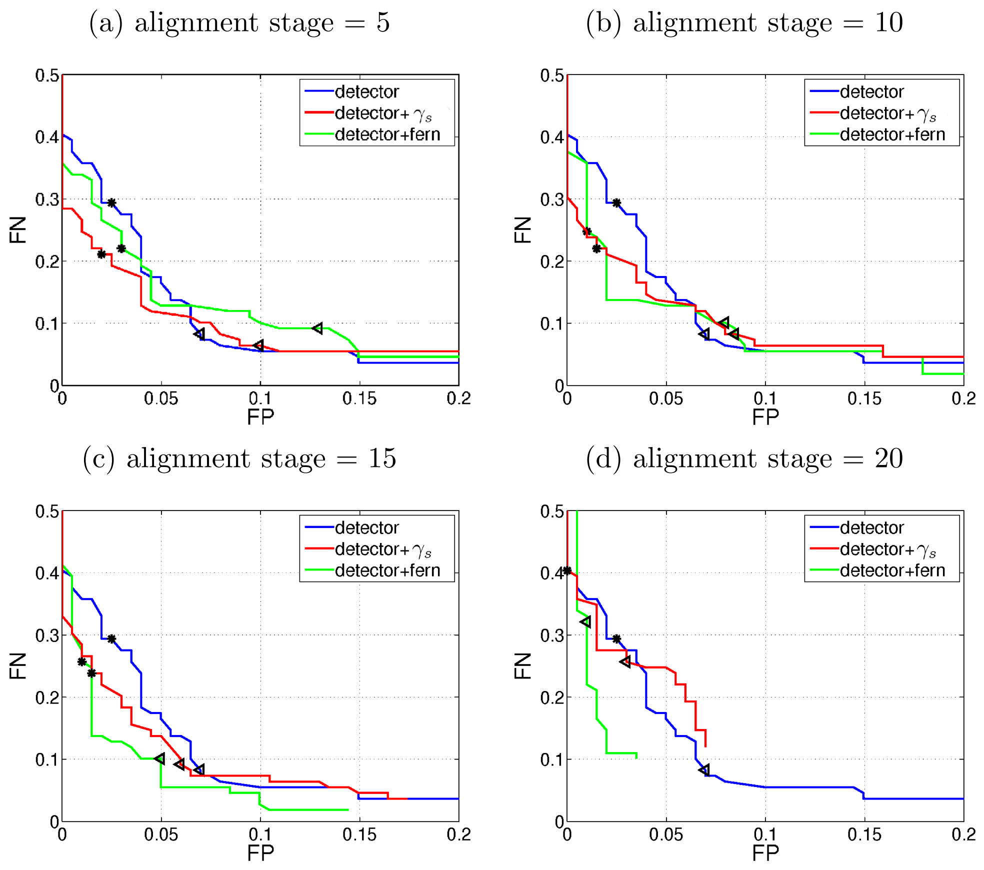


Figure 7.5: **ROC curves** for different *alignment stages*. Particular points of ROC curves are computed for different confidence thresholds. With growing confidence we are getting more to the top left corner of the ROC curves. Points with the same confidence threshold are depicted by *star* and *triangle*. The closer we get to point $(0, 0)$ the better.

have only some $0.6^{15} \cdot 250000 \approx 118$ windows left, since in average we reject in every stage 60% of candidates. For this small number of windows, the motion estimator is well applicable in real-time.

7.4 Summary

In this chapter we focused on improving the detectors based on cascade of classifiers. We found out, that aligning the detection window in the middle of the cascade improves the detector performance, while keeping the efficiency. Sliding window alignment before the detection, or after first few stages of the detection process lowers the FNs, but increases the FPs. The number of FPs is higher thanks to many potential FP window candidates,

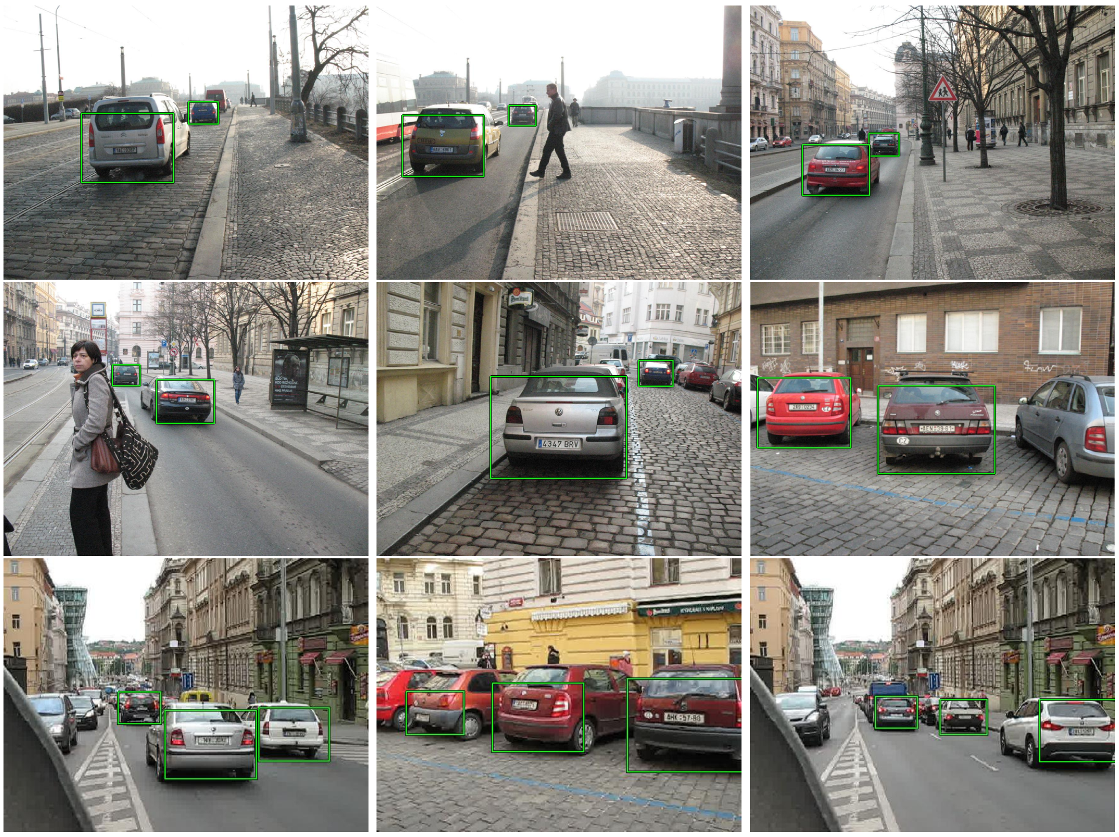


Figure 7.6: Examples of testing images.

which are sometimes aligned by the motion estimator to position, which makes it even more difficult (similar to object of interest) for the detector to reject it. On the other hand alignment in one of the last detection stages yields more FNs, where only the already well aligned candidate windows pass the earlier stages of the detector and the motion estimator would be no more useful. While in the middle stages the tracker significantly lowers the number of both FPs and FNs and at the same time needs to evaluate only a small portion of all candidate windows. We have experimentally verified our method on a practical problem of car detection.

8 Non-Rigid Object Detection with Local Interleaved Sequential Alignment (LISA)

Detection of objects with appearance altered by pose variations (including non-rigid deformations and viewpoint changes) is more difficult than the detection of objects in a single pose [86, 87]. If the detection time is constrained, exhaustive search over the space of possible poses with a single pose detector is intractable.

An ample amount of work has been devoted to the detection of objects deformed by pose variations. Many approaches partition the positive training samples into clusters with similar poses, see Figure 8.1b. Some of them [88, 89] first estimate the pose cluster and then use pose-specific classifier to decide about the object presence. Others [90, 91] estimate pose cluster simultaneously with the detection process. A fine partitioning of the pose space is desirable to achieve good detection performance. However, the finer the partitioning, the fewer training samples fall into each cluster and therefore immense training sets are often needed [90]. In contrast to these approaches, recent work [86] uses simple pose estimators which align some detection features. These pose estimators help to detect objects in an arbitrary pose without training set partitioning.

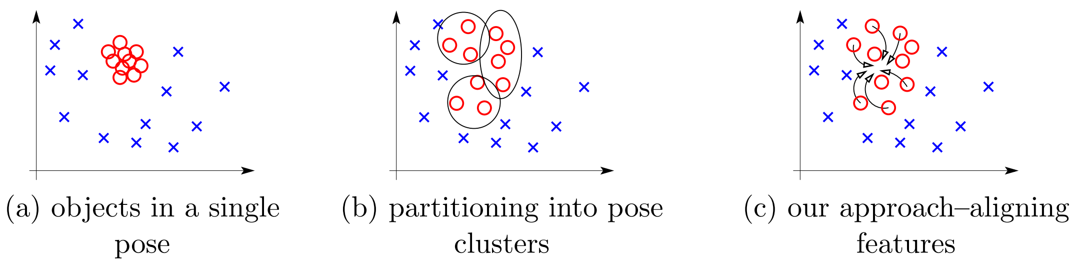

(a) objects in a single pose
(b) partitioning into pose clusters
(c) our approach–aligning features

Figure 8.1: Simplified sketches of positive (red circles) and negative (blue crosses) samples in 2D feature space. (a) objects in a single pose exhibit smaller scatter than objects deformed by pose variations, (b) scattered samples are often partitioned into pose clusters with a small number of training samples, (c) our approach aligns features during the detection to compensate for object deformation consequently making positive samples less scattered.

Our feature alignment remedies the partitioning of the training set. In contrast to [86] which finds simple local deformations (e.g. inplane rotations) and aligns each feature independently, we rather estimate a *global non-rigid alignment* of all the features. Importantly, the alignment is estimated solely from the features used for detection by pre-trained regressors. The alignment estimation is reduced to reading a value from a look-up table which costs negligible time. On the other hand, our approach requires annotated data for learning. Nevertheless, the use of our method does not prevent the use of [86], thus the frameworks are complementary.

In our system the features are evaluated sequentially; each one reveals a certain amount of object deformation, see Figure 8.2. Features are *successively aligned* to the observed deformation which makes the positive class less scattered and easier to detect, see Figure 8.1(c). The training set is not partitioned and the number of necessary training samples remains relatively low, even for large deformations.

We demonstrate the sequential alignment idea on a Sequential Decision Process (SDP) similar to Waldboost [45], where the successive nature of feature evaluation allows for efficient application of the estimated alignment. In the SDP a classifier cumulatively estimates a *confidence* about the object presence or absence in a given detection window. Once the confidence is sufficiently low, the window is rejected. We extend the SDP by exploiting the same features that were used for the confidence computation to estimate the alignment. The alignment is then applied on the subsequent features and the process continues with more appropriately aligned features. In consequence, both the confidence and the alignment are estimated more efficiently as it is then easier to distinguish the well aligned positive samples from the background and to estimate the alignment from a closer neighbourhood. The process continues until the rejection or acceptance is reached as in the classical SDP. Note that the confidence and alignment updates are encoded by the same feature values, therefore in comparison with the classical SDP the computational complexity of SDP with alignment is almost preserved. We call the proposed scheme Sequential Decision Process with *Locally Interleaved Sequential Alignment* (SDP with *LISA*).

The contribution is threefold: (i) we show that features evaluated in the sliding window detection process also contain knowledge about the correct alignment of the evaluated window on the observed object deformation; (ii) we propose very efficient piecewise linear regression functions which are jointly learned with the classifier. This facilitates estimating the alignment during the detection process; (iii) we show that the estimated alignment speeds up the detection process by reducing the search space and improves the detection rates.

8.1 Related Work

Great progress has been made in the detection of objects under varying poses and deformations [86, 16, 92, 93]. The predominant strategy is to combine a collection of classifiers, each dedicated to a single pose or deformation [87, 94, 95, 96, 97, 98]. To train multiple classifiers, either the training data need to be separated into disjoint clusters [90, 95], or the features in training samples need to be registered to lie in correspondence [86, 96], or both strategies are combined [87].

The clustering of training data imposes the need to collect large amounts of data for

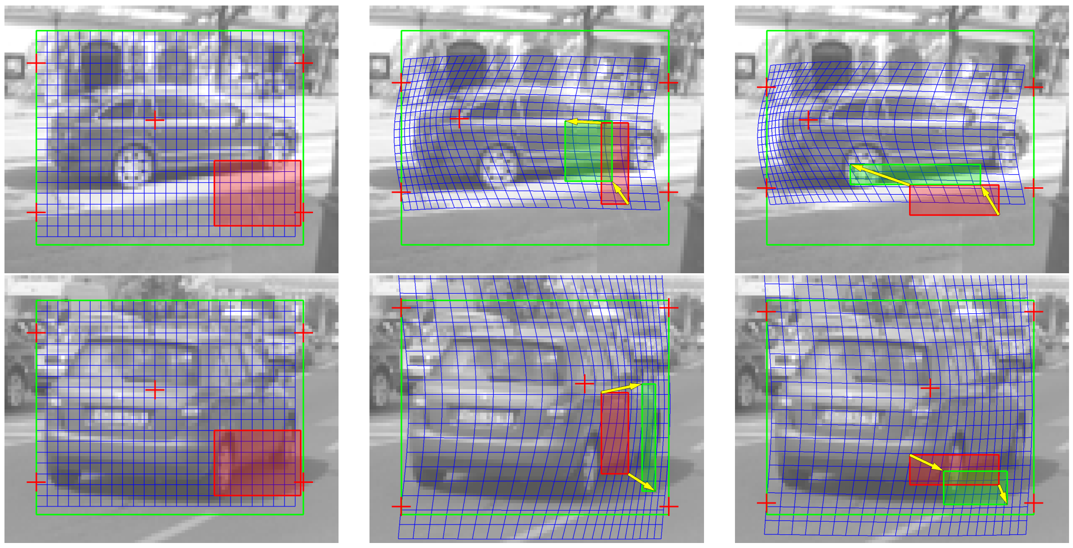


Figure 8.2: **Local Interleaved Sequential Alignment (LISA):** Left: The first feature always has the same position. The deformation of the local coordinate system is outlined by the blue mesh. Features evaluated during the detection process contribute to both (i) confidence and (ii) non-rigid deformation. The second column shows how features are aligned after 30 evaluated features. A non-aligned feature position is delineated by the red rectangle; the aligned feature is green. The last column shows alignment of the last evaluated feature.

learning each classifier separately. Some authors try to reduce the amount of training data by sharing some features across multiple views [95, 94]. To avoid the clustering of training data, some methods [95, 96, 97, 98] align the object features in each training sample (before or during the training process) to lie in correspondence. This task usually requires a precise labelling of object features correspondence.

Other methods avoid the necessary labelling and try to align the features automatically [87, 86] before their evaluation. Usually, the feature positions and low level deformations are estimated first, e.g. by computing the dominant edge orientation in some part of the detection window and using pose-indexed features [86, 96]. The automatic feature alignment keeps the training set less scattered, and improves the detection rates but lacks interpretation. We model the alignment for a specific class of objects, and as a side product of the detection we obtain a parametrized alignment of the whole model.

Detection of deformable objects

Recently, Ali et al. [86] proposed to use pose indexed features coupled with dominant edge orientation estimation, in different scales and positions in the detection window, for feature alignment. By the feature alignment, they forgo the need to train a collection of detectors for different object poses and learn a single deformable detector. The dominant edge orientation is partitioned into 8 bins in 14 poses (1 pose in the largest scale, 4 in smaller scale and 9 in the smallest scale) and needs to be computed for every candidate position in order to estimate the features poses. For alignment estimation they need to evaluate additional $8 \cdot 14 = 112$ features apart from the detection features which is considerably more than the average number of features needed for classification in our system. We interleave object detection and alignment by regression. The detector runs on increasingly better aligned features, which consequently groups together the training samples in the feature space, as shown in Figure 8.1(c). Thanks to the design of regression functions and the re-use of detection features, a negligible number of additional computations are needed (reading the alignment parameters from a look-up table and moving the features) and the computational complexity grows with the dimension of the alignment space only very gently. The approach of [86] does not require the training data labelling while our method does. However, our method allows having the pose space not discretized.

In general, SDP with LISA outperforms the standard SDP's [45, 70, 67, 68, 69, 78] in both the speed and detection performance. In [87] the authors argue that the sliding window-based object detectors work best when trained on examples that come from a single coherent group with well aligned features, e.g. frontal faces. Our improvement in performance is caused by the ability to locally align on the displaced, rotated, and deformed object instances. The gain in speed is obtained by the search space reduction. From the point of view of 2D translation search space reduction (sparser detection grid) has already been shown for a cascade of classifiers in [4] and for SDP in [6]. Here we show that we can effectively reduce a high dimensional search space of non-rigid deformation.

Recent deformable part-based object detectors [18, 95, 96] achieve excellent detection rates, but are far from the real-time performance. The computational complexity of part-based object detectors is given governed by the detection of model parts and by the estimation of globally optimal parts configuration. Model parts are usually detected by an SVM-based [18, 95] or AdaBoost [96] classifier. The root-part of the object is detected first [18, 96], which allows reducing the search space for the remaining parts. After estimating the candidate positions of the parts, the globally optimal configuration of the model parts (or for multiple models for multiple views [95]) is found by using the dynamic programming. Our method may be compared from the computational complexity point of view with [16]. Their classifier is an SVM working with HoG features which similar to [95] for *one-view model* as well as for the *root part* of the part-based model in [18]. In [16] N denotes the number of possible locations in multiple scales and V the number of evaluated features in each subwindow. For an SVM-based detector, *VN multiplications* are needed. Since the SDP with LISA may learn to compensate 2D translation [6], the number of poses N may be significantly reduced to M and we may run the SDP with LISA on a sparser detection grid. Here we compensate non-rigid deformation of high dimensionality and $M \ll N$. Our method requires *$4VM$ additions* in the worst case (all features being evaluated without the early rejection), where 4 is the number of additions needed to transform each evaluated feature according to estimated alignment (see Section 8.5 for details). For performance comparison with [18, 95] and the running times of different methods see Section 8.7.

The Patchwork of Parts [99] builds a statistical model for detection of objects with multiple parts. The detection process loops through the image locations (positions and scales) and evaluating the classifier for every part (class) in every location. Model deformations are modelled as shifts of object parts which are than recombined using a patchwork operation. The algorithm is able to classify 100 subwindows per second which is not fast enough a real-time performance.

Search space reduction

A speeding-up of the original part based model [18] was addressed in [100]. The authors argue that the cost of detection is dominated by the cost of matching each part of the model to the image and not by the cost of computing the optimal configuration of parts. They propose to learn a multi-resolution hierarchical part based model. The parts are tested sequentially and image locations are discarded as soon as a partial detection score falls under some threshold. The resulting algorithm achieves almost the same detection accuracy as the original algorithm and runs twice as fast.

The search space reduction for detection speed-up has been approached also in *Efficient Subwindow Search* (ESS) [101]. ESS is reducing the search space by a branch and bound algorithm. It defines multiple sets of rectangles (sets of candidate windows) in the image. After evaluation of all the features in the image the algorithm computes the upper bound (highest possible detection score) that the score function could take on any of the rectangles in each set. The authors propose an efficient scheme for going hierarchically through all the possible rectangles (scales and translations) without the need to exhaustively evaluate the detection score for all possible rectangles. Many rectangle sets are rejected as soon as the upper bound is under some acceptance threshold. The disadvantage is the need for evaluating all the features in the image first. This is well applicable for the approaches which use a bag of features or some shared low level features, usually for multiview and part-based object detection [97, 95]. After the features evaluation, the detectors need to evaluate a non-trivial score function (usually SVM-based classifiers) of all the features which fall into each particular rectangle, and here ESS brings significant speed-up [97]. A sliding window-based SDP does not need to evaluate all the features in all the positions and scales thanks to the early rejection stages. Here, ESS would actually slow down the process by the necessary evaluation of all features first. Also, in the SDP the rejection thresholds are already known in advance for each stage and no other bounds need to be computed. From the search space reduction point of view, ESS reduces the number of candidate window translations as well as scales, but does not take into account other object deformations.

The recently proposed Crosstalk Cascades [102] assume that adjacent subwindows responses at nearby locations and scales are correlated. As soon as the classification score for some location reaches some threshold, all points in the detection grid in the close neighbourhood start to be evaluated as well (excitatory cascades). When the ratio of score in the current position and in at least one position in the close neighbourhood goes under some stage specific threshold the evaluation of remaining stages in actual position stops (inhibitory cascades). Training the classifier needs perturbing each training sample

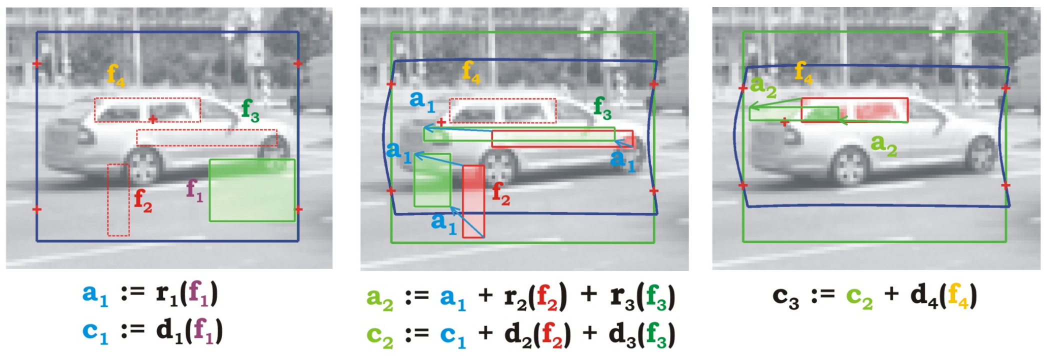


Figure 8.3: **Classification:** Three steps of SDP with LISA are depicted. Left: In the initial position, only feature 1 is evaluated. From its value the first alignment $\mathbf{a}_1$ and confidence c_1 are computed. Middle: the alignment $\mathbf{a}_1$ is applied on features 2 and 3. Note that the applied alignment $\mathbf{a}_2$ is updated by contribution of two regressors, not just one. Also note that the alignment $\mathbf{a}_1$ was not applied on feature 4. Right: the last feature is moved from its initial position by the accumulated alignment $\mathbf{a}_2$.

by small 2D shifts to ensure the correlation of classifier answers at nearby positions. This perturbing corrupts the performance of the cascaded detector. We approach the problem from an opposite direction. We are aligning the detection window to a pose at which it fits the object better. It does not require corrupting the training samples by adding shifts to positive samples. The detection of well aligned samples is easier and needs to evaluate fewer features to reach the decision. From the point of view of search space reduction, the Crosstalk Cascades reduce the 2D sliding window translation and scale as well as [101]. Nevertheless, they still need to look at every position in the detection grid but the number of evaluated features is reduced. Our method yields the detection grid reduction because our detector is able to move and deform.

Non-rigid alignment estimation

We mention only few of the most relevant papers in this area, since our method focuses more on improvement of *object detection*, than on precise alignment estimation.

State of the art methods specialized in alignment of deformable models are now able to cope with quite large object deformations [103, 104, 105]. In [103] authors train a cascade of regressors for non-rigid face deformation estimation. Every regressor in the cascade is a *Fern*. Each Fern separates the training set into subsets (one for each leaf), where within one subset there are samples with a similar type of deformation. Therefore each Fern basically divides the space of possible deformations. This makes the alignment task easier for regressors trained on subsets in each leaf and allows coping with larger deformations. On the other hand, it is necessary to cluster the training data to learn the regressors. In comparison, our method does not need to cluster the training data for learning. We transform only the features which is faster than inversely transforming the whole image patch as in [103].

In the Boosted Appearance Model (BAM) [105] the authors propose to train a classifier which recognizes well aligned deformable model from those not well aligned. The classifier is then used in a criterial function which includes a parametrized shape of the object. The goal is to find the shape parameters that maximize the score of the learned strong classifier. The problem is solved iteratively by gradient ascent optimization. The follow-up to BAM is Boosted Ranking Model [104] (BRM). In BRM given 2 image patches a classifier is trained to recognize the better aligned image patch from the worse aligned. The shape in both BAM and BRM is a set of 2D points and the displacement of each of the points is parametrized by a 2D vector. The number of parameters is reduced by projection to a low dimensional space via PCA similarly to our approach. In comparison, we model the shape deformation by deforming the whole grid which covers the object patch. This allows us to efficiently transform the features instead of inversely transforming the whole image patch. We obtain a transformation for every pixel in the grid and not only for the control points of the shape as in [103, 104].

A Sequence of Learnable Linear Predictors for learning a fixed sequence of linear predictors (weak regressors) that estimate the object alignment was proposed in [11]. Each predictor in the sequence is learned on the *estimation error* of the previous predictor. A similar idea was later proposed in [84], only they use Ferns instead of linear predictors as the weak regressors. Our work uses weak regressors proposed in [6]. Instead of using simple linear functions we propose a non-linear regression and approximate it by piecewise linear (or piecewise constant) functions which improves the alignment and keeps the fast performance of linear predictors.

8.2 SDP with LISA Classification

We divide the classification process into K stages. In each stage k, only one feature is evaluated. The value of this feature contributes to the confidence and the alignment. Contributions are determined by (i) a detection function $d_k : \mathbb{R} \to \mathbb{R}$, which maps the feature value to a contribution to the confidence, and (ii) a regression function $r_k : \mathbb{R} \to \mathbb{R}^m$, which assigns an m-dimensional contribution to the alignment vector $\mathbf{a}$ using the same feature value. Both the confidence and the alignment are accumulated from evaluated features. Then there is a threshold $\theta_k \in \mathbb{R}$ (estimated during the learning), which allows to reject windows with the so far accumulated confidence lower than θ_k. In each stage, the feature can potentially be aligned. This is determined by a binary value q_k, which is estimated by boosting during the training stage. If q_k is `TRUE`, this will invoke aligning of the feature, while $q_k =$ `FALSE` means that the non-aligned feature will be used.

The alignment estimated from a single feature may be inaccurate, therefore it must be accumulated over multiple features. We keep the last *valid* alignment, denoted as $\mathbf{a}_\omega$, where index ω corresponds to the stage at which the alignment was estimated. Besides that, we also accumulate alignment updates from all evaluated features. This alignment is updated in each stage k and we denote it by $\mathbf{a}_k$. Hence, there are two alignments: (i)

```
1: Initialize $\mathbf{a}_0 = \mathbf{0}, c_0 = 0, k = 1, \omega = 0$.
2: while $k \leq K$ do
3:     if $q_k =$ TRUE then    // use alignment
4:         Estimate the value of feature $v = f_k(I, \mathbf{a}_\omega)$
           with alignment $\mathbf{a}_\omega$.
5:     else
6:         Estimate the value of feature $v = f_k(I, \mathbf{0})$    without alignment.
7:     end if
8:     Update confidence $c_k \leftarrow c_{k-1} + d_k(v)$.
9:     if $c_k < \theta_k$ then
10:        reject the window and break,
11:    end if
12:    Estimate alignment $\mathbf{a}_k = \mathbf{a}_{k-1} + r_k(v)$.
13:    if $z_k =$ TRUE then    // alignment is valid
14:        update $\omega \leftarrow k$
15:    end if
16:    $k \leftarrow k + 1$
17: end while
18: Accept the window.
```

Figure 8.4: **Classification algorithm:** Classification of a single window by SDP with LISA

accumulated up to stage k denoted by $\mathbf{a}_k$ and (ii) valid, which is applied on features, denoted by $\mathbf{a}_\omega$. The stage at which the index ω is updated is determined by a binary value $z_k =$ TRUE; z_k is again estimated by boosting during the training stage).

We define a feature function $f_k : (\mathcal{I} \times \mathbb{R}^m) \rightarrow \mathbb{R}$ as a mapping which assigns a feature value to a window with image data $I \in \mathcal{I}$ and m-dimensional alignment vector $\mathbf{a} \in \mathbb{R}^m$. For the sake of simplicity, we refer to the feature function as the *feature* and to image data in the sliding window as the *window*. Based on the above introduced notation, we define the strong classifier as a collection:

$$H = [f_1, q_1, d_1, \theta_1, r_1, z_1, \ldots, f_K, q_K, d_K, \theta_K]. \tag{8.1}$$

The classification Algorithm (Figure 8.4) summarizes how the SDP with LISA decides about object presence or absence in a given window I with the given strong classifier H. See also Figure 8.3 for illustration. In the algorithm, we denote c_k as the confidence and $\mathbf{a}_k$ as the alignment, both accumulated up to stage k.

8.3 Joint Learning of SDP with LISA

The expected output from learning is the strong classifier H, Equation (8.1), inputs are *training* and *validation* sets. At the beginning of each training stage, the training set with the following structure is available:

$$\mathcal{T} = \{(I^1, \mathbf{t}^1, y^1),\ \ldots,\ (I^p, \mathbf{t}^p, y^p),\ (I^{p+1}, y^{p+1}),\ \ldots,\ (I^N, y^N)\},$$

where $I^1, \ldots, I^p$ are positive image data, $I^{p+1}, \ldots, I^N$ are negative image data, $y^1 \ldots y^N$ are labels such that $y^1 = \cdots = y^p = 1$, $y^{p+1} = \cdots = y^N = -1$ and $\mathbf{t}^1 \ldots \mathbf{t}^p$ are correct alignments of positive data. The validation set $\mathcal{V}$ and the testing set $\mathcal{W}^1$ have the same structure.

The learning algorithm uses the following notation: $[[\Psi]]$ is a binary function equal to 1 if a statement Ψ is `TRUE` and 0 otherwise, and $[\Upsilon\ \Xi]$ is concatenation of Υ and Ξ. We introduce the error of the strong classifier H on validation data $\mathcal{V}$ denoted as $E(H, \mathcal{V}) = \sum_i [[H(I_i) \neq y_i]],\ \forall (I_i, y_i) \in \mathcal{V}$. For the sake of completeness we define: $E(\emptyset, \mathcal{V}) = \infty$. To simplify the notation, we also denote a weak classifier $\mathtt{wc}_k$ to be the following foursome $\mathtt{wc}_k = [f_k\, q_k\, d_k\, \theta_k]$.

The joint learning of SDP and LISA, see Figure 8.5, successively builds a strong classifier H. The current stage is denoted by the lower index k, the training samples are indexed by the upper index i. Since we allow the alignment to be accumulated over multiple stages without direct application on features, we also keep the index ω of the last valid alignment.

The algorithm (Figure 8.5) first constructs two weak classifiers: $\widehat{\mathtt{wc}}_k$, that use features either aligned by $\mathbf{a}_{k-1}$ or not aligned at all, and $\mathtt{wc}_k$, that use features either aligned by $\mathbf{a}_\omega$ or not aligned at all (lines: 4-5). Then the validation errors of $[H_k\ \mathtt{wc}_k]$ (strong classifier H_k concatenated with $\mathtt{wc}_k$) and $[H_k\ \widehat{\mathtt{wc}}_k]$ (strong classifier H_k concatenated with $\widehat{\mathtt{wc}}_k$) are compared, and the one with the lower error is selected and denoted as H_k (lines: 6-11). If the alignment a_{k-1} is used (i.e. $\widehat{\mathtt{wc}}_k$ is used and $\widehat{q}_k = \mathtt{TRUE}$), then ω is set to $k-1$, which makes a_{k-1} to be the valid alignment from now on. After that, we jointly re-learn regression functions $r_{\omega+1} \ldots r_k$ to estimate the alignment from features $f_{\omega+1} \ldots f_k$, i.e. those features which have not yet been used for the alignment (lines 12-13). Please note, that in case where the alignment a_{k-1} is used, the re-learning reduces on the learning of the new regression function r_k, which will be used in the following stages.

Eventually, training weights are updated in line 14 and training data $\mathcal{T}$ are updated (line 15) by the new negative samples. Negative samples are collected as the false positive detections of the current strong classifier H_k. The algorithm continues until the validation error starts to increase.

The rest of the chapter continues as follows: Learning of weak classifiers and alignment regressors for the group of features is described in section 8.4. The non-rigid deformation model is summarized in Section 8.5. Implementation details are explained in Section 8.6.

```
1: input: $\mathcal{T}$, $\mathcal{V}$, $\mathcal{F}$
2: $k = \omega = 1$, $H_0 = \emptyset$, $w^i = 1/N, i = 1 \dots N$.
3: while $E(H_k, \mathcal{V}) \leq E(H_{k-1}, \mathcal{V})$ do

     //Build two weak classifiers: $\texttt{wc}_k$ that use $\mathbf{a}_\omega$ and $\widehat{\texttt{wc}}_k$ that use $\mathbf{a}_{k-1}$.
4:   $\texttt{wc}_k$ = learn_weak_cls($\mathcal{T}, \mathcal{F}, H_k, \mathbf{a}^1_{k-1} \dots \mathbf{a}^N_{k-1}$)
5:   $\widehat{\texttt{wc}}_k$ = learn_weak_cls($\mathcal{T}, \mathcal{F}, H_k, \mathbf{a}^1_\omega \dots \mathbf{a}^N_\omega$)

     //Comparison of validation errors determine, whether to start using $\mathbf{a}_{k-1}$ from
     stage $k$ or not.
6:   if $E([H_k \ \texttt{wc}_k], \mathcal{V}) > E([H_k \ \widehat{\texttt{wc}}_k], \mathcal{V})$ then
7:     $H_k \leftarrow [H_k \ \widehat{\texttt{wc}}_k]$
8:     if $\widehat{q}_k = \texttt{TRUE}$ then $\omega \leftarrow k - 1$ end if
9:   else
10:    $H_k \leftarrow [H_k \ \texttt{wc}_k]$
11:  end if

     //Learn regressors estimating alignment from features $f_{\omega+1} \dots f_k$.
12:  $[r_{\omega+1} \dots r_k] \leftarrow$ learn_regressors($\mathcal{T}, f_{\omega+1}, \dots, f_k$)

     //Update regressors $r_{\omega+1} \dots r_k$ in the strong classifier
13:  $H_k = \Big[ H_\omega \ [\texttt{wc}_{\omega+1} \ r_{\omega+1} \ z_{\omega+1}] \dots [\texttt{wc}_k \ r_k] \Big]$

     //Update weights of training samples
14:  $w^i = \exp(-y^i H_k(I^i)), \ i = 1 \dots N$

     //Collect new negative samples as FPs of $H_k$
15:  $\mathcal{T} \leftarrow$ update_negative_samples($H_k$)
16:  $[\mathbf{a}^1_\omega \dots \mathbf{a}^n_\omega \ \mathbf{a}^1_k \dots \mathbf{a}^N_k] \leftarrow$ update_alignment($\mathcal{T}$)
17:  $k \leftarrow k + 1$
18: end while
```

Figure 8.5: **Learning of SDP with LISA:**

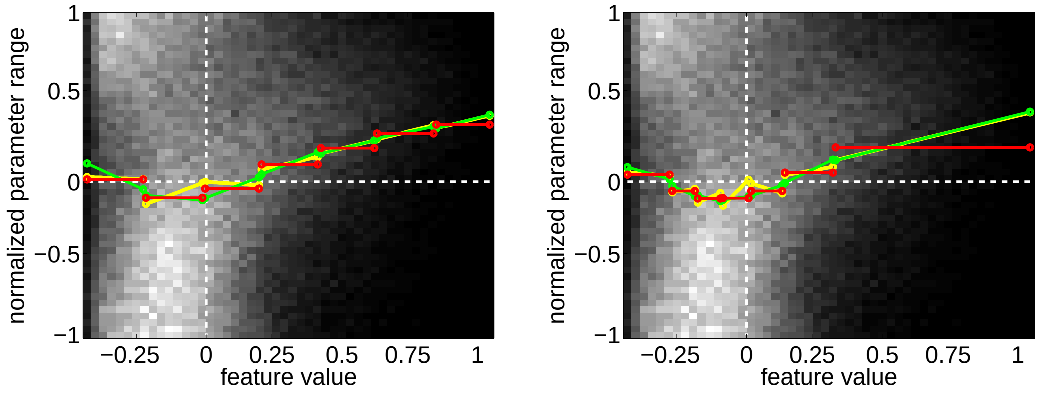


Figure 8.6: **Examples of tested piecewise linear regression functions:** The density of the training data for one feature (depicted as grayscale heatmap) with fitted regression functions. The left image corresponds to *non-proportional partitioning* and the right image to the *proportional partitioning* of the feature space. Green is a piecewise affine function (i), yellow corresponds to a piecewise linear function (ii) and red is a piecewise constant function (iii).

8.4 Learning of Weak Classifiers and Alignment Regressors

8.4.1 Learning of Weak Classifiers

In our approach, the weak classifier d_k is a piecewise constant function of feature $f_k \in \mathcal{F}$ dividing feature values into U bins with sizes proportional to the training data density[2]. To simplify the notation, we define a bin assigning function $\delta_j : \mathbb{R} \rightarrow \mathbb{N}$ which assigns corresponding bin u to feature value $v = f_j\left(I^i, \mathbf{a}^i_\omega\right)$ for the i-th training sample and j-th feature. Given training samples weights w^i, the constant response κ_{ku} of d_k in bin u is computed as follows:

$$\kappa_{ku} \quad = \quad \arg\min_{\kappa} \sum_{i \in I_u} w^i(\kappa - y^i)^2 = \frac{\sum_{i\in I_u} w^i y^i}{\sum_{i\in I_u} w^i} \tag{8.2}$$

where $I_u = \{i \,|\, \delta(f_j(I^i, \mathbf{a}^i_\omega)) = u\}$ is the set of training samples indexes, which fell to bin u.

In the weak classifier estimation the same procedure is performed for each bin and each feature from the feature pool. Finally, we use the feature (and corresponding classifier d_k) which yields the lowest weighted error. Such approach is coincident with Gentleboost technique [5]. Rejection thresholds θ_k and $\widehat{\theta}_k$ are set in order to preserve the required maximum number of false negatives (FN) per learning stage. The FN limit is defined by the user to achieve the required running time similarly to [78].

[1]Sets $\mathcal{T}$ and $\mathcal{V}$ are used during the learning phase and testing set $\mathcal{W}$ is used for experimental evaluation.
[2]except the size of border bins which are $[-\infty, \texttt{min\_value}]$ and $[\texttt{max\_value}, +\infty]$

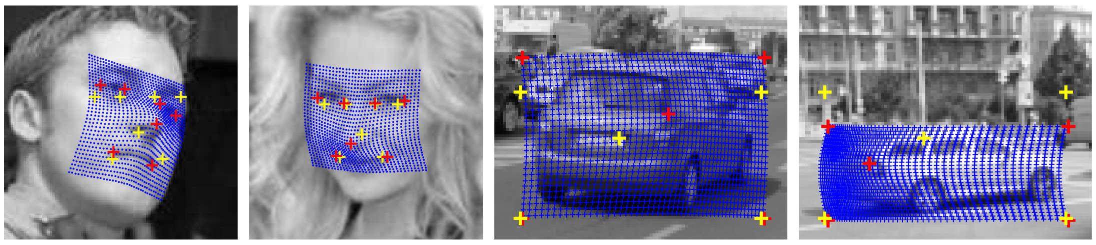


Figure 8.7: Example data from the LFW and CSV datasets. The red crosses are ground truth labels. Yellow crosses depict the *mean of labelled positions for each dataset* and the blue points are the deformed grid. The deformed grid is obtained by thin plate splines non-rigid deformation using the correspondences between the yellow and red points.

8.4.2 Learning of Regressors

As already noted in section 8.2, the use of a regressor, learned on a single feature, may inaccurately align some positive samples and cause the lower detection rate. During the learning we do not immediately apply the estimated alignment on the feature, but we wait for the right number of features, for which jointly learned regressors yield better alignment and consequently lower the validation error of the detector.

In the learning algorithm (Figure 8.5, line 12), regressors $r_{\omega+1}, \dots, r_k$ are jointly learned to compensate the residual alignment error $\Delta \mathbf{t}^i = (\mathbf{t}^i - \mathbf{a}^i_\omega)$ of preceding regressors $r_1, \dots, r_\omega$. $\mathbf{t}^i$ is the vector of ground truth parameters of alignment. We search for regressors $r_{\omega+1}, \dots, r_k$ which are the solution of the following problem:

$$\operatorname*{arg\,min}_{\tilde{r}_{\omega+1}\dots\tilde{r}_k} \sum_{i=1}^{p} \left\| \Big(\sum_{j=\omega+1}^{k} \tilde{r}_j(f_j(I^i, \mathbf{a}^i_\omega)) \Big) - \Delta \mathbf{t}^i \right\|_F^2. \tag{8.3}$$

For simplicity we explain only one dimensional alignment estimation, i.e. with Δt^i being only a scalar for each sample instead of a vector. The higher dimensional alignment is learned for each dimension independently, therefore the following equations are valid for multiple alignment parameters estimated by each regressor. To solve the problem (8.3) we propose to learn a piecewise affine function by the least squares method. The feature space is divided into U bins, where each bin gives an affine response, see green lines in Figure 8.6.

The response of a regression function $r(v)$ is then computed as follows:

$$r(v) = \varphi_{j,\delta(v)} v + \lambda_{j,\delta(v)}, \tag{8.4}$$

where $\varphi_{j,\delta(v)}$ and $\lambda_{j,\delta(v)}$ are scalar coefficients. We considered (i) full affine function with $\varphi_{j,\delta(v)} \in \mathbb{R}, \lambda_{j,\delta(v)} \in \mathbb{R}$, (ii) linear function $\varphi_{j,\delta(v)} \in \mathbb{R}, \lambda_{j,\delta(v)} = 0$ and (iii) constant function with $\varphi_{j,\delta(v)} = 0, \lambda_{j,\delta(v)} \in \mathbb{R}$. Since we experimentally verified that all three functions yield similar results for a sufficient number of bins (see experimental evaluation in section 8.7.4), we used the piecewise constant function to speed up the alignment estimation process. Therefore problem (8.3) is reduced to search for coefficients $[\lambda_{\omega+1,1} \dots \lambda_{k,U}]$. We substitute Equation (8.4) with $\varphi_{j,\delta(v)} = 0$ into problem (8.3) to obtain the corresponding least squares problem:

$$\underset{\tilde{\lambda}_{\eta,u} \in \mathbb{R}}{\arg\min} \sum_{i=1}^{p} \left\| \sum_{j=\omega+1}^{k} \tilde{\lambda}_{j,\delta(f_j(I^i,\mathbf{a}_\omega^i))} - \Delta t^i \right\|^2. \tag{8.5}$$

To write the solution of (8.5) in a compact form, we further introduce a binary matrix

$$[\mathbb{A}]_{i,(ju)} = \left\langle \begin{array}{ll} 1 & \text{if } \delta(f_j(I^i, \mathbf{a}_\omega^i)) = u \\ 0 & \text{otherwise} \end{array} \right. \tag{8.6}$$

where index i determines the row and indexes (ju)[3] determine the column. We also introduce vector $\tilde{\lambda} = [\tilde{\lambda}_{\omega+1,1} \dots \tilde{\lambda}_{k,U}]^\top$, which is concatenation of all unknown lambdas from all bins and all features. Finally, we form the vector $\mathbf{\Delta g} = [\Delta t^1 \dots \Delta t^p]^\top$ with all the residual alignments. The solution of problem (8.5) is then

$$\lambda = \underset{\tilde{\lambda}}{\arg\min} \left\| \mathbb{A}\tilde{\lambda} - \mathbf{\Delta g} \right\|^2 = \mathbb{A}^+ \mathbf{\Delta g}, \tag{8.7}$$

where $\mathbb{A}^+$ denotes Moore-Penrose pseudo-inverse [51] of matrix $\mathbb{A}$.

Two types of feature space partitionings were tested: (i) non-proportional partitioning, which divides the space into bins of equal sizes and (ii) proportional partitioning, which divides the space into bins of sizes inverse proportional to the training data density, where each bin contains the same number of training samples. See Figure 8.6 for example of partitioning into 7 bins with all three tested functions fit into the training data of one feature.

8.5 Non-rigid Deformation Model

We work with two types of alignments. The first is a simple two dimensional displacement and the second is a non-rigid deformation parametrized via PCA [106]. We define the feature as function $P : (\mathcal{I} \times \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{N}) \to \mathbb{R}$, the value of which is computed from image $I \in \mathcal{I}$ on the position specified by its left-upper corner $\alpha \in \mathbb{R}^2$ and right-bottom corner $\beta \in \mathbb{R}^2$ with type $\xi \in \mathbb{N}$. We experimented with HoG features [102] (where ξ denotes orientation of edges) and Haar features [107] (where ξ stands for the feature type).

The alignment encoding the two dimensional displacement is given by two dimensional vector $\mathbf{a} = (\Delta x,\ \Delta y)^T$. Then, feature function $f(I, \mathbf{a})$ of the feature specified by $\{\alpha, \beta, \xi\}$ aligned by the two dimensional displacement $\mathbf{a}$ is

$$f(I, \mathbf{a}) = P(I, \alpha + \mathbf{a}, \beta + \mathbf{a}, \xi). \tag{8.8}$$

[3] (ju) denotes a linear combination of indexes j and u.

The non-rigid alignment deforms the position of every corner point α (resp. β) by m-dimensional vector $\mathbf{a} = [a_1 \dots a_m]^T$ as follows:

$$\alpha(\mathbf{a}) = \alpha + a_1\mathbf{w}_\alpha^1 + \cdots + a_m\mathbf{w}_\alpha^m, \tag{8.9}$$

where $\mathbf{w}_\alpha^1 \dots \mathbf{w}_\alpha^m$ are 2D Eigenvectors corresponding to deformations modelled in point α. Eigenvectors of the non-rigid deformation are obtained by PCA. Training of PCA is detailed in the next paragraph. Feature function $f(I, \mathbf{a})$ of the feature specified by $\{\alpha, \beta, \xi\}$ aligned by the non-rigid deformation $\mathbf{a}$ is

$$f(I, \mathbf{a}) = P(I, \alpha(\mathbf{a}), \beta(\mathbf{a}), \xi). \tag{8.10}$$

To train the PCA, we use the position of a few manually selected keypoints in each bounding box from the training set, see the red points in Figure 8.7. Given these keypoints, we compute the elastic transformation by thin plate splines transformation [50] of an orthogonal pixel grid within the bounding box for each training sample, see blue grids in Figure 8.7. The elastic transformation assigns a two-dimensional displacement vector to each pixel from the orthogonal grid in each bounding box. Finally, we concatenate these displacements for each training sample into one column vector and project them into the lower dimensional space by PCA.

The alignment may be applied either by deforming the features and placing them in the right position in the image or by inversely deforming the image. The latter would require the image deformation after each applied alignment update. Unfortunately, this is unthinkable for the real-time performance of the detector as we would need to transform the image (or part of it) multiple times for each candidate window. Hence we transform the features.

We need to keep the features in a rectangular shape to take advantage of the fast evaluation on integral images. Therefore the deformation of each feature is only approximated by anisotropic scaling, see Figure 8.2 for example. The non-rigid transformation is applied on the upper left and lower right corner of each feature. This gives us the correct positions of both corners for each feature and determines the new width and height of each feature, see Figure 8.2. This feature transformation costs 8 additions per feature and is very efficient. This type of feature transformation is used in the learning as well as in the detection process.

8.6 Implementation Details

Both the weak classifier d_k and the weak regressor r_k in stage k are implemented as a single lookup table (see Table 8.1 for an example); both confidence and alignment updates are read by a single look-up. The only additional cost during the classification for the alignment is its application to the features positions. For a rigid alignment by translation, application of the alignment means only two scalar additions per evaluated stage since both corners move identically and the feature's height and width are precomputed. In non-rigid deformations, we precompute an alignment lookup table, where

bins of feature f_k	d_k	r_k	
$(-\infty, -0.42)$	-1	0	0
$\langle -0.42, -0.27)$	0	0.05	-0.37
$\langle -0.27, -0.19)$	-0.32	-0.07	-0.39
$\langle -0.19, -0.11)$	0.91	-0.09	0.02
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 8.1: A part of a lookup table encoding the confidence and 2D translation updates for one feature in stage k. Bin sizes and values of the first regression column correspond to the function shown in red in the right image in Figure 8.6.

a_1	$\dots$	a_m	f_1		$\dots$	f_k	
			α_1	β_1		α_k	β_k
0		0	(3,7)	(5, 25)		(51, 61)	(12, 18)
0	$\dots$	0.1	(4,7)	(6, 26)	$\dots$	(51, 61)	(11, 18)
$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$

Table 8.2: A part of a lookup table encoding corner positions of particular features for all possible alignments.

each alignment vector $\mathbf{a} = [a_1 \dots a_m]$ is assigned with both position corners α_i, β_i for all features f_i, see Table 8.2. To speed-up the m-dimensional indexing, we compute a 1-dimensional index from $a_1 \dots a_m$ by bit-shifting. As a consequence, application of the non-rigid alignment costs four additions plus one access to the lookup table per stage. The size of the lookup table is reasonable. We usually use $m = 3$ and feature corners positions are integer values, that can be encoded by one byte. Denoting: D to be the number of discrete values of a_j and F to be the total number of features, the size of the alignment lookup table is $4FD^m$ (e.g. for $D = 100$, $m = 3$ and $F = 300$, the size is 1.2GB).

8.7 Experiments

The results of the experiments demonstrate the importance of (non-rigid and rigid) LISA for SDP. Section 8.7.1 and 8.7.2 describes experiments with non-rigid LISA on *Annotated Faces in the Wild dataset* (AFW [108]) and *Car Semi-profile View dataset* (CSV). Our negative data consist mostly of Google street-view images without cars and faces in total amount of 15`Gpxl`. Experiments with rigid LISA are detailed in [6]. Section 8.7.4 evaluates performance of different regression functions.

We apply our feature alignment method on SDP similar to Waldboost [45]. We refer to our method applied on SDP as SDP+LISA, reimplementation of the alignment method proposed by Ali et al. [86] applied on SDP is referred to as SDP+Ali [86]. Besides

	SDP	SDP+LISA	SDP+Ali	SDP+LISA+Ali	Zhu	DPM
AFW	0.129	**0.041**	0.081	**0.047**	0.069	0.115
CSV h.-view	0.038	**0.001**	0.042	0.003	–	**0.001**

Figure 8.8: **Experiment summary:** Comparison of FN rates for fixed number of FP per 1Mpxl equal to 10^{-2}. Results corresponds to ROC curves in Figures 8.10 and 8.12

Method	SDP+LISA	SDP+Ali [86]	Zhu [95]	DPM [98]
Running time on VGA	33ms	41ms	17.2s	10.5s

Figure 8.9: **Running time:** Comparison of running time on Intel core i7 Q700, 1.7GHz. Methods SDP+LISA, SDP+Ali [86], Zhu [95] run on single core, DPM [98] uses 4 cores. We used publicly available MATLAB/MEX implementation of DPM [98] and Zhu [95], while SDP+LISA and SDP+Ali [86] is measured on our C++ implementation with image resolution known in advance and the nearest neighbour image rescaling (ROC curves correspond to the linear interpolation when rescaling images). Time reported in [86] for AdaBoost+Ali is 30ms for 120×190 images and probably only one scale. Nevertheless, we reimplemented their feature alignment method and use it in our SDP, which is significantly faster than the AdaBoost used in [86] due to early rejections.

that we also evaluate SDP+LISA+Ali [86]. This method allows to combine non-aligned features with features aligned by either the Ali [86] method, LISA method, or by both methods simultaneously. The alignment method of each particular feature is determined by boosting during the training stage. In addition to that, we also show baseline given by pre-trained, publicly available models: (i) Deformable Part based Models (DPM) [98] on a CSV and AFW dataset and (ii) Zhu's and Ramanan's [95] face detector on AFW dataset. In section 8.7.4 we also compare the precision of our alignment to the one of Zhu [95]. Section 8.8 discusses advantages, drawbacks, and limitations of the proposed method.

Ground truth annotations contain positions of several manually annotated keypoints. AFW has 7 keypoints and CSV has only 3 keypoints (upper left, lower right for bounding box and one vertical edge point), see Figure 8.7. All experiments are conducted with HoG features. Detection rates are summarized in Figure 8.8. Section 8.7.3 justifies the choice of HoG over Haar features by comparing detection rates on CSV dataset.

In all experiments where sequential decision process is involved, detected windows are filtered by Non Maxima Suppression (NMS). NMS is set to suppress all detections which have mutual coverage (union of bounding boxes divided by their intersection) bigger than 0.6. Criterion for correct detection is that the detected bounding box and ground truth bounding box have mutual coverage bigger than 0.3.

8.7.1 AFW Dataset Results

The AFW dataset is a publicly available dataset of face images obtained by random sampling of Flicker images. We use ground truth data which specify positions of 7 manually chosen keypoints (2 for each eye, 1 for the nose and 2 for the mouth corners). Note that a considerable amount of publicly available annotations have very low accuracy of keypoint positions (errors corresponding to 15% of the face size are not an exception). Even though such annotations make it difficult to train any accurate regression function (especially in L_2-norm), we use them directly. Since our approach does not contain any decision tree, which could split frontal and profile images, we focused only on frontal images captured within the range of approximately ± 45 degrees (in-plane and out-of-plane rotations).

Figure 8.10 shows ROC curves of SDP, SDP+LISA, SDP+Ali [86], SDP+LISA+Ali [86] trained on the first part of the AFW dataset. We can see that LISA outperforms Ali's [86] method. However, Ali's [86] method still yields significant improvement with respect to the pure SDP. It is also worth emphasizing that the SDP+Ali [86] method only needs annotated bounding boxes, while SDP+LISA also needs annotated keypoints to learn the regressors estimating non-rigid deformation. We can also see that the SDP+LISA+Ali [86] method, which combines features aligned by Ali [86] and LISA methods, has almost the same results as the SDP+LISA method. For comparative purposes the results of publicly available pre-trained models of Zhu's and Ramanan's [95] detector and Felzenszwalb's DPM detector [98] are shown. We do not retrain their detector and use publicly available model *p146_small* and the same DPM model from [95] using 10 mixtures learned for faces and kindly provided to us by Xiangxin Zhu. Unlike our detector, Zhu's and Ramanan's detector is designed to detect high-resolution faces only (bigger than 80×80 while our detector works with 40×40 pixels). To make the comparison fair, we evaluated all methods only on faces which are bigger than 80×80.

Since AFW was captured by random sampling of Flicker images, training and testing sets are independent. Nevertheless, we also show that if we train on a subset of the BIOID dataset and a small fraction of the LFW dataset (faces already detected by Viola-Jones detector), we achieve almost the same results (see Figure 8.11 for the comparison of SDP and SDP+LISA trained on different datasets).

8.7.2 CSV Dataset Results

The CSV dataset consists of 1600 images of cars taken from a semi-profile view ranging from almost pure rear view to the almost pure side view. Images are taken from a human view angle. We use 1200 images for training (training and validation sets) and 400 for testing. As the ground truth three points were marked: upper left, lower right, and a rear vertical edge point. The rear edge point corresponds to the imaginary intersection of the lower side windows line and the rear edge. The bounding box has a fixed aspect ratio. The parameters which were estimated by the regressors were selected by PCA,

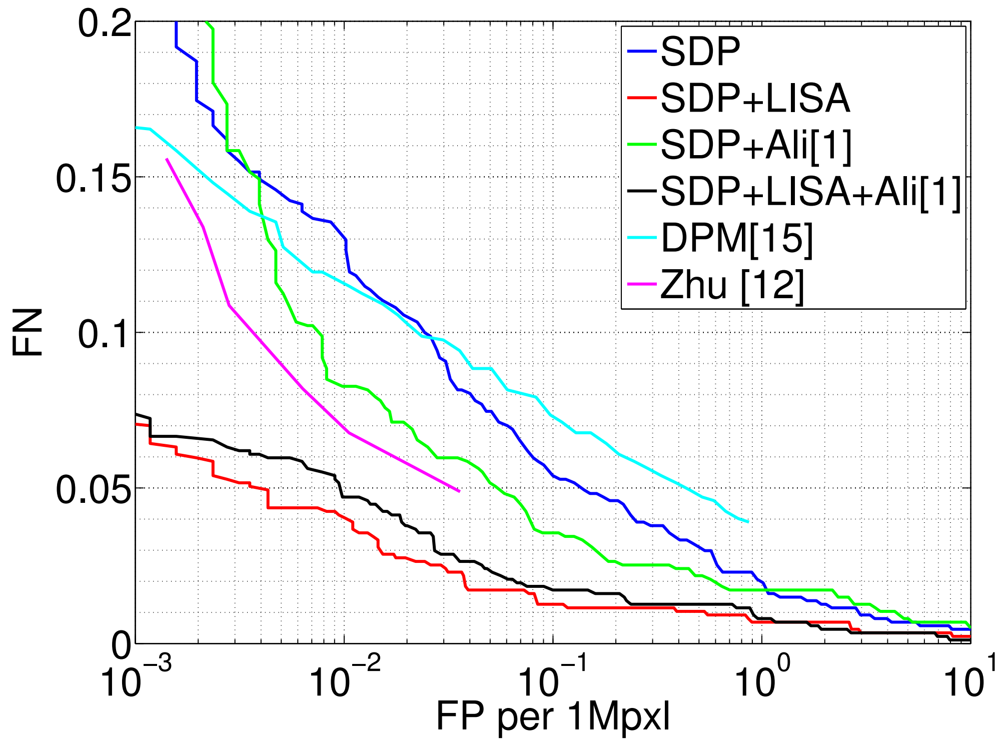


Figure 8.10: **AFW-AFW ROC curves:** Comparison of different methods on the AFW dataset. SDP, SDP+LISA, SDP+Ali [86], SDP+LISA+Ali [86] trained on the first part of AFW. All methods were tested on the second part of AFW (images were captured by random sampling of Flicker images, therefore training and testing sets are independent). False positives are measured per 1 `Mpxl` of background data, false negatives per dataset.

as described in section 8.5, for both modelled non-rigid deformations in the AFW and CSV datasets.

Figure 8.12 shows ROC curves of SDP, SDP+LISA, SDP+Ali [86], SDP+LISA+Ali [86] and publicly available DPM [98] pre-trained from VOC 2007 data. Figure 8.12 shows corresponding ROC curves. LISA outperformed Ali's [86] method. Actually, Ali's [86] method did not yield any significant improvement in the detection rate, because the deformations in this dataset probably could not be well modelled by the pose estimators of [86] (mainly estimating dominant edge orientation) since there is almost no in-plane rotation present in the CSV dataset. We can also see that DPM slightly outperforms SDP+LISA in human view-point images, however we still preserve real-time performance, see running time summary in Figure 8.9.

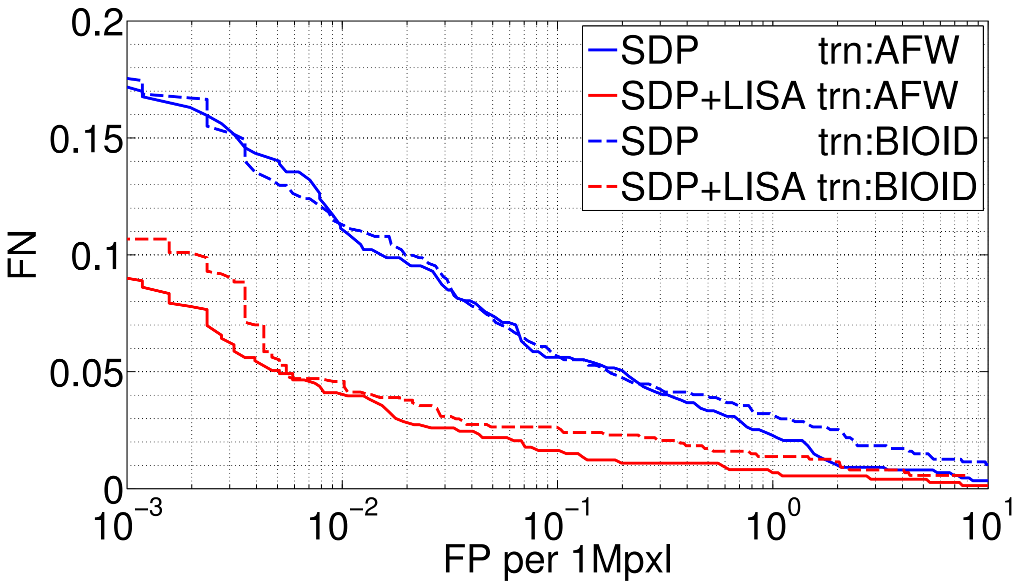


Figure 8.11: **AFW-BIOID ROC curves:** Comparison of (i) SDP, SDP+LISA trained on first part of AFW (solid lines) and (ii) SDP, SDP+LISA trained on BIOID dataset (office environment images) and small fraction of LFW (dashed line). All methods were tested on the second part of AFW. False positives are measured per 1 `Mpxl` of background data, false negatives per dataset.

8.7.3 Haar vs. HoG Features

We also demonstrate the influence of the choice of feature type. Figure 8.13 shows ROC curves of SDP and SDP+LISA methods evaluated on the CSV dataset captured from the human-view angle for (a) Haar features and (b) HoG features. While the relative improvement coming from using LISA is preserved, HoG features exhibit much better detection rates than Haars.

8.7.4 Regression Functions Evaluation

In this experiment we evaluate the performance of piecewise regression functions in all three variants of equation (8.4): (i) affine function with ξ and λ, (ii) linear function with ξ only and (iii) constant function with λ only (used in LISA). We learn the regression functions for different numbers of bins in combination with two types of feature space partitionings.

Here the regression functions are learned separately from the detector on their own features. In line 12 of the Learning Algorithm in Figure 8.5 multiple regressors are jointly learned at once. The number of jointly learned regressors is estimated automatically by observing the error on validation data. Five regressors are jointly learned on average. Therefore for performance evaluation, we also jointly learn 5 regressors.

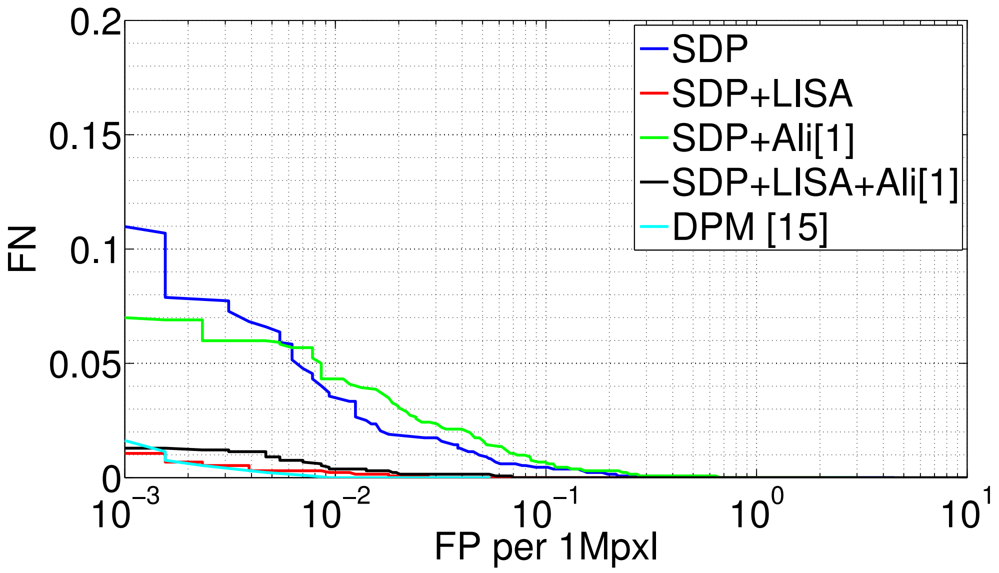


Figure 8.12: **CSV ROC curves:** False positives are measured per 1 `Mpxl` of background data, false negatives per dataset.

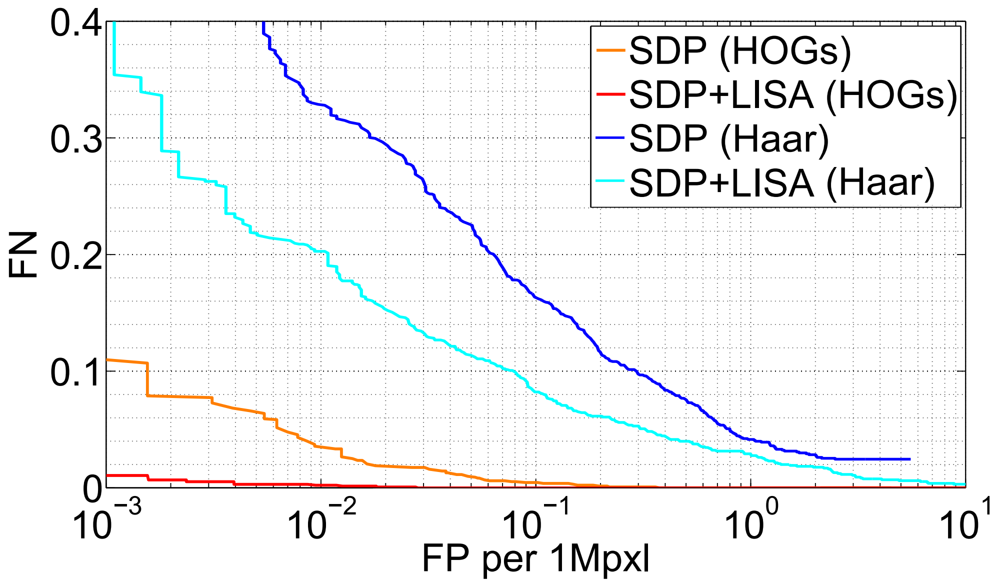


Figure 8.13: **ROC curves (Haar vs HoG features):** False positives are measured per 1 `Mpxl` of background data, false negatives per dataset.

As a criterion for selection of the best performing function and feature space partitioning we use the *mean regression error* (MRE) of the alignment parameters (selected

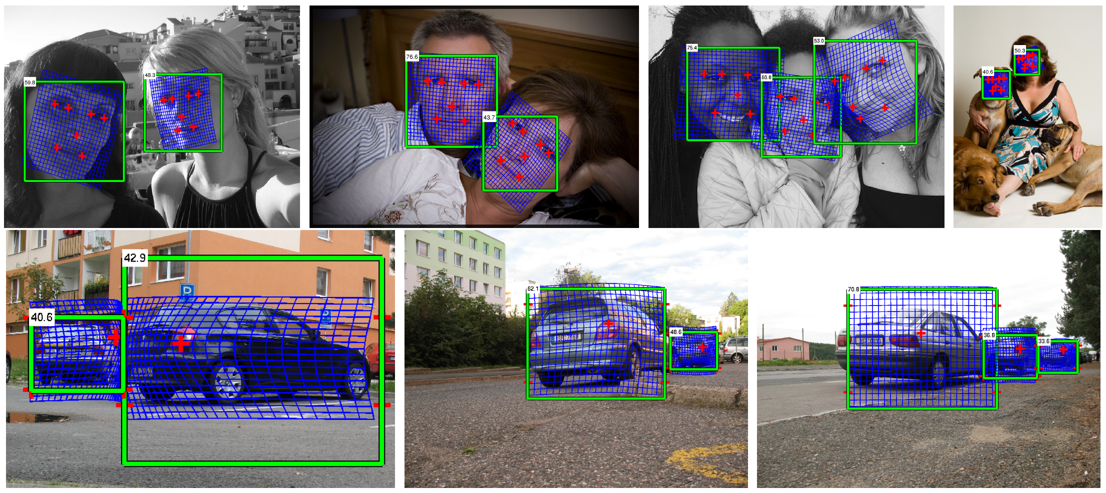


Figure 8.14: **Detections:** with the SDP+LISA detector. Upper row shows some faces from AFW testing set. The last image in the upper row shows FP detection. The bottom row shows semi-profile cars from robot view angle on CSV testing images.

by PCA) estimation with respect to the parameters computed from the deformed grids computed from ground truth annotations, depicted in Figure 8.7. Each features' regressor estimates all $m = 3$ alignment parameters. In this section we denote r_j^k a part of the $j-$th regressor, which estimates single alignment parameter k. The MRE of the positive samples in the testing set is then computed as follows

$$\mathrm{MRE} = \sum_{i=1}^{p} \left(\sum_{k=1}^{m} \left(\sum_{j=1}^{5} r_j^k(v_j^i) - \Delta t^{ki} \right)^2 /m \right) /p, \tag{8.11}$$

where p is the number of positive image samples, v_j^i is the $j-$th feature value of sample i and t^{ki} is the samples' $k-$th ground truth parameter.

The resulting MREs on testing data for all variants of tested regression functions dependent on the number of bins are depicted in Figure 8.15. The MREs are normalized by the initial MRE of the testing set when no alignment is applied. The results in Figure 8.15 correspond to non-rigid alignment estimation on the testing part of the LFW dataset. The results for other datasets are similar.

The important observation here is that MREs of the third function (piecewise constant - dark and light red bars in Figure 8.15) decrease very quickly with the growing number of bins. At approximately 15 bins it reaches the testing error of the first two types of functions with the slope parameter ξ. Also the proportional partitioning variants (bars in light tones) perform better than the non-proportional ones (bars in dark tones). In our algorithm we use the *piecewise constant function* with the *proportional partitioning* (light red bars). It achieves a good alignment precision and is extremely fast to evaluate.

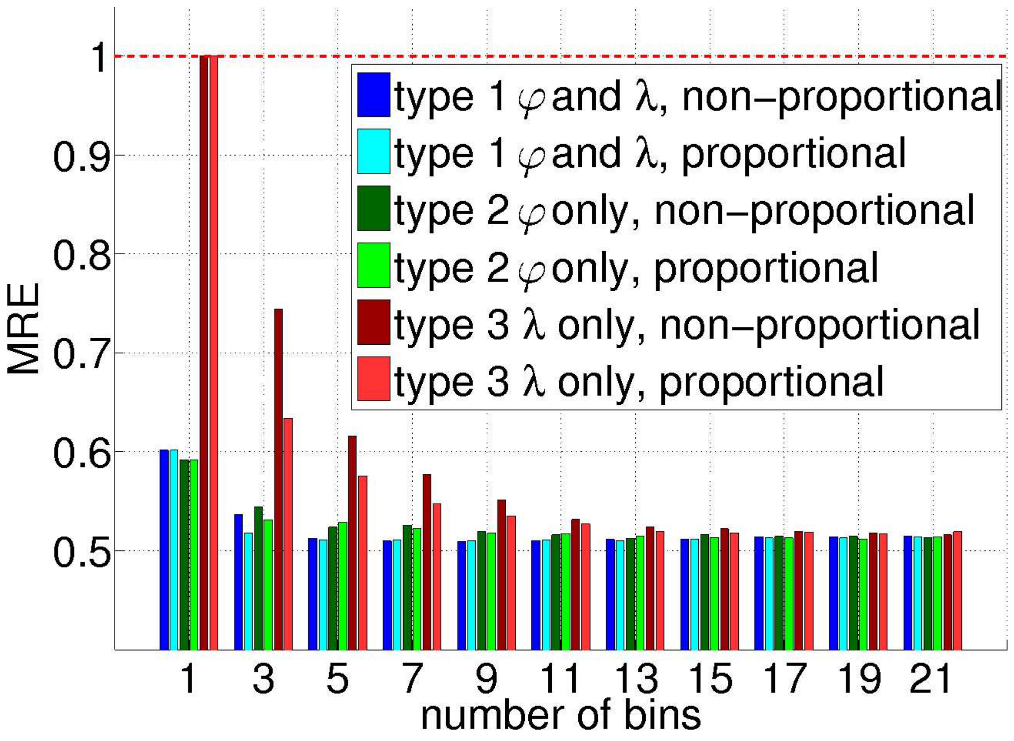


Figure 8.15: **Mean Regression Error (Equation. 8.11) as a function of the number of bins for LFW alignment learned on 5 features.** Please note that piecewise constant function (dark and light red bars) quickly reaches the testing error of the first two functions, which use the slope parameter ξ. Also note that the proportional partitioning of the feature space yields better results than the non-proportional one. MREs are normalized, with 1 being the testing set error when no alignment is estimated.

In the last experiment we evaluate the precision of estimated alignment and we compare our results to [95] on facial features. Alignment precision is evaluated on a testing part of the AFW dataset. Our model was trained on the training part of the AFW dataset. The same model was used to generate red curve in Figure 8.10. The publicly available model *p146_small* from [95] works with faces larger than 80×80. That is why we made a selection of images with faces appearing in larger resolution. We took the positive detections of [95], which corresponded to one of 7 frontal face models (out of 13). The remaining are side-view models, which do not contain all the facial features necessary for comparison with our model. From this selection we made an intersection of true positive detections of both our method and the one of [95] in order to evaluate the alignment on the exact same images. A total of 338 images from the testing set were selected for this experiment. The computed errors are Euclidean distances of estimated

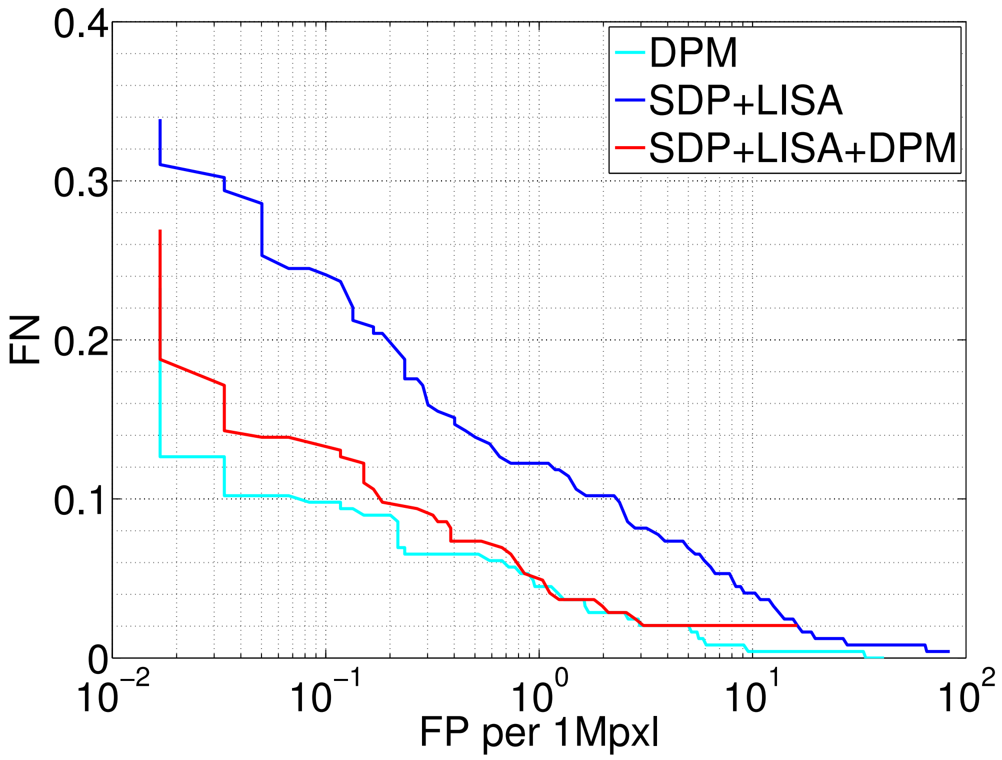


Figure 8.16: **SDP+LISA+DPM pipeline:** Comparison of SDP+LISA, SDP+LISA+DPM and DPM on a hard face dataset with wide range of poses, illuminations and occlusions.

facial features positions from ground truth facial features positions relative to face size (to compensate for different face scales). 7 facial features were used: 4 eye corners, 1 nose tip and 2 mouth corners. The resulting mean error of [95] is 0.0513, median 0.0441 with variance 0.0012. I.e. for a face of size 100×100 pixels, there is a mean error of 5.13 pixels for each facial feature. The resulting mean error of our method is 0.0472, median 0.0410 with variance 0.0009. For the same face size, we achieve a lower mean error of 4.72 pixels for each facial feature.

8.8 Summary

The proposed local interleaved sequential alignment improves the sequential decision process. The main competitors are the SDP itself and local pose estimators proposed by Ali [86], which may be combined with the SDP as well.

The SDP is favorable for its high detection speed, see running time comparison in Figure 8.9. On the other hand, it is known that the greedy learning suffers from lower generalization when compared to SVM based approaches like [95, 98]. Notice that in the previous experiments we used publicly available models of [95, 98] to show the baseline. When the DPM with 4 components [98] and SDP+LISA are trained on the same dataset and tested on a harder face dataset with wide range of poses, illuminations and occlusions, then the detection rate of DPM is indeed better, see Figure 8.16. To achieve both high speed of SDP+LISA and high detection rate of DPM, we propose a combined SDP+LISA+DPM pipeline. The SDP+LISA step reduces the number of possible sub-windows and the strong but slow DPM runs only on the remaining small fraction of all sub-windows. In this experiment, the SDP+LISA leaves only 15 sub-windows per 1MPxl image in average for additional DPM evaluation, while only 2% of true positives are rejected. Remaining 15 sub-windows are finally evaluated by the DPM in a negligible time.

LISA is based on regression functions, which sequentially compensate deformation of the object in the evaluated sub-window. The advantage of the regression functions is that the computational complexity grows only linearly with the dimensionality of the pose space. However, accurate regression is usually possible only for a limited range of local deformations.

Main drawbacks of the proposed method are: (i) inherently limited generalization of SDP methods, (ii) limited range of deformations and (iii) keypoint annotations needed for learning. The main advantages are: (i) high detection speed, (ii) better detection rate than other SDP methods, (iii) global object deformation is estimated as a side-product of the detection process.

9 Conclusions

In this thesis we focused on real-time object detection and tracking. We improved the performance of the standard methods, while keeping their fast performance during runtime. The summarized contribution follows here:

- We proposed an automatic selection of training samples for incremental learning, which was suitable for situations, when no user input was allowed and when the computational time was limited. When the time is limited, we can not incrementally train the predictor from every single image. The regression function may also be overfitted without the proper selection of the additional training samples. Our method delivers only the useful additional training samples for the incremental learning and requires no user input. The slowly changing object appearance is successfuly re-learned. Still in some situations the algorithm may fail, mainly because of sudden appearance changes. The object position needs to be validated before it is incrementally learned. We proposed a method for fast tracker position validation. When the appearance suddenly changes the validation detects the loss-of-track. This approach proved to be very useful in practical applications, when it was combined with a robust Ferns classifier or SURF keypoint descriptor used as an object detector.

- We showed how to learn a regressor which approximated non-linear regression function and kept the computational complexity of a linear predictor. This method was used for the real-time non-rigid object tracking. It allows to achieve a better precision in motion parameters estimation with less evaluated features. We also proposed a tracker learning scheme for objects undergoing a non-rigid deformation. We tested our approach on two tasks. First we estimated the deformation of human faces under different facial expressions and second we tracked the deformation of eye lids during blinking.

- We have proposed a method for the detection of LP-trackable points. We found out that multiple image patches may be tracked by a predictor learned only from a single image patch. By observing the trackers' local convergence properties we may determine the trackability of an image patch by this tracker. We proposed an efficient filter built from the learned predictor which performed in real-time. This filter may be used to find the LP-trackable points in the scene. The advantage of this approach is that no learning of the predictor is necessary. The trackable points may be tracked immediately. This method may be also used as a preprocessing step for more complex object detection methods. Lets say, that we want to detect human faces. Then we may also learn a predictor for the task of face tracking and create an image filter. The filter quickly delivers small amount of potential candidate positions for some more complex face detectors.

- In the final part of the thesis we focused on combination of learned regressors with an object detector (or generally with any sequential decision process). We worked with sliding window detector since it was the state of the art approach in the real-time object detection. We proposed to align the sliding window, or even individual features, during the detection process. First we tried to combine the linear regressors and the non-linear Ferns with a cascade of classifiers. This early work was a proof of concept which showed, that combining a sliding window alignment with a standard object detection methods pays off. We have achieved better false positive and false negative rates. Finally we proposed an efficient scheme for joint learning of the gentleBoost object detector and the speeded-up regressors for non-rigid object detection. The detectors' confidence and the information about correct alignment was accumulated during the evaluation of individual features. The updated alignment was immediatelly applied on the following features. Thus the features were better and better aligned on the object. This approach significantly improved the false positive and the false negative rates and it also speeds-up the detection process. Thanks to the local sliding window alignment the detector did not need to search through the pose spase so densely.

A Linear Predictors with Appearance Parameters

Encoding the object appearance variation is a desirable property of each tracking system. Cootes in [13] used linear predictors to model the human face shape and appearance at the same time. One regression matrix was used to estimate both shape and appearance parameters together. An interesting approach was proposed by Zimmerman in [28], where the object appearance was used to update the regression matrix $\mathtt{H}$. In [28], such a linear predictor is called *parameter-sensitive learnable linear predictor* (PLLiP).

Besides the regression matrix $\mathtt{H}$, the PLLiP has got another regression matrix $\mathtt{G}$ - so called appearance encoder. The matrix $\mathtt{G}$ is trained in an unsupervised way. Besides the training matrices $\mathtt{T}, \mathtt{L}$ another training matrix $\mathtt{J} = [\mathbf{j}_1, \dots, \mathbf{j}_m]$ with m object appearances is introduced. The matrix $\mathtt{J}$ is similar to $\mathtt{L}$, and vectors of training examples intensities $\mathbf{j}_1, \dots, \mathbf{j}_m$ are collected on the *expected* uncertainty region of predictor. It means that the size of uncertainty region needs to be estimated before learning of PLLiP. The ordered triplet $\mathtt{L}, \mathtt{J}, \mathtt{T}$ forms the training set for PLLiP. With regressor $\mathtt{H} = (\mathtt{H}_0, \dots, \mathtt{H}_d)$ and given number d of appearance parameters in the vector $\theta = (\theta_1, \dots, \theta_d)^T$, the PLLiP tracks according to

$$\mathbf{t} = (\mathtt{H}_0 + \theta_1 \mathtt{H}_1 + \cdots + \theta_d \mathtt{H}_d)\, I(X)\,, \tag{A.1}$$

where the appearance parameters vector θ is computed as

$$\theta = \mathtt{G}\mathbf{j}. \tag{A.2}$$

The dimensions of matrices $\mathtt{G}$ and $\mathtt{H}$ are $d \times n$ and $p(d+1) \times n$ respectively, where n is the number of pixel coordinates in support set and p is the number of tracked parameters. The tracking is than ordered triplet $X, \mathtt{H}, \mathtt{G}$ - support set, regresor of the predictor, regresor of appearance encoder. The prediction error $e(\mathtt{H}, \mathtt{G})$ of the system $X, \mathtt{H}, \mathtt{G}$ is

$$e(\mathtt{H}, \mathtt{G}) = \sum_{i=1}^{m} \left\| \left(\mathtt{H}_0 + \underbrace{\left(\mathbf{g}_1^T \mathbf{j}_i\right)}_{\theta_1} \mathtt{H}_1 + \cdots + \underbrace{\left(\mathbf{g}_d^T \mathbf{j}_i\right)}_{\theta_d} H_d \right) I(X) - \mathbf{t}_i \right\|, \tag{A.3}$$

where d is a number of appearance parameters. Optimal tracking system with respect to training set $\mathtt{L}, \mathtt{J}, \mathtt{T}$ is than

$$(\mathtt{H}^*, \mathtt{G}^*) = \arg\min_{\mathtt{H},\mathtt{G}}\; e(\mathtt{H}, \mathtt{G})\,. \tag{A.4}$$

```
1: Randomly initialize matrix $\mathtt{G}^0$
2: while $\frac{e^k - e^{k-1}}{e^k} \geq \epsilon$ do
3:     Find $\mathtt{H}^k = \arg\min_{\mathtt{H}} e\left(\mathtt{H}, \mathtt{G}^{k-1}\right)$.
4:     Find $\mathtt{G}^k = \arg\min_{\mathtt{G}} e\left(\mathtt{H}^k, \mathtt{G}\right)$.
5:     Recompute error $e^k = e\left(\mathtt{H}^k, \mathtt{G}^k\right)$.
6: end while
7: $\mathtt{H}^* = \mathtt{H}^k$, $\mathtt{G}^* = \mathtt{G}^k$, stop.
```

Figure A.1: Learning with appearance parameter.

The goal of the learning algorithm is to find matrices $\mathtt{H}^*$ and $\mathtt{G}^*$. This is an unconstrained optimization problem, where bilinear function is fitted in the least squares sense into a high dimensional data. Zimmerman [28] shows that this problem has a closed-form solution in $\mathtt{H}$ (respectivelly $\mathtt{G}$) if $\mathtt{G}$ (respectivelly $\mathtt{H}$) is fixed. Therefore the solution is searched by an exact line-search method, which successively computes more and more exact solutions, see algorithm A.1. The number ϵ is some chosen threshold for desirable minimal error of learning. The description of the PLLiP given here is not sufficient for perfect understanding of the learning phase (searching for matrices $(\mathtt{H}^*\mathtt{G}^*)$). Detailed description of PLLiP learning and experiments with parameter sensitive predictor can be found in [28].

Tracking with appearance parameters makes the estimation much more robust and lowers the estimation error [28]. Since each appearance parameter adds another degree of freedom to the system, exponentially more training examples must be added to the learning process. Also the tracking system may fail for appearances which were not seen in the training set. Anyway according to our experience from experiments the usage of a few (1 or 2) appearance parameters for tracking by linear predictors is recommendable.

B Greedy Support Set Selection

A greedy least-squares algorithm for the support set selection was proposed in [11]. Suppose, that the support set coordinates in X cover the whole object template, i.e. has the maximal complexity C_{max}. The algorithm starts with empty support set X^* and pixels coordinates are selected and added to X^* one after each other. The currently selected pixel coordinate $\mathbf{x}_i \in X$ is the one that minimizes error ε on training data $\varepsilon = \|\mathtt{T}_{est} - \mathtt{T}\|_F$ for some precomputed matrix $\mathtt{T}$ of randomly perturbed parameters, where $\mathtt{T}_{est}$ is matrix of parameters estimated by linear predictor trained for the support set $X^* \cup \{\mathbf{x}_i\}$, see algorithm B.1. The output of the algorithm is a support set X^*. It was experimentally shown in [11], that using X^* yiealds lower error on training data, than random selection of support set pixels. The disadvantage of proposed method for support set selection is the high computational cost of the algorithm.

Such a subset selection method, which does not lower the precision of prediction, we may lower the computational complexity of the linear predictor, which is the main contribution of this algorithm. It would be also intersting to try to select template pixels which are the most discriminative for particular parameter estimation.

Another approach for lowering the predictor complexity was used in [13], where the principal component analysis has been applied to the training set of image intensities and from all principal components were used only those with the high eigenvalues. After PCA the eigenvectors with small eigenvalues were ommited thus reducing the dimensionality of data. For 100 face images, each with approximately 10 000 intensity values for the facial region, only 55 parameters from the appearance model were used after PCA. These 55 parameters explained 95% of the appearance variation. This approach slightly increases the time complexity of the tracking, because PCA needs to be applied in each tracking step, while the greedy algorithm for support set selection [11] precomputes the support set before the tracking phase.

```
1: Initialize $X^* \leftarrow \{\emptyset\}, A \leftarrow \{1, 2, \ldots, C_{max}\}$
2: for $i \leftarrow 1; i \leq C; i \leftarrow i + 1$ do
3:     Set $B \leftarrow A, a_{\min} \leftarrow -1, \varepsilon_{\min} \leftarrow \infty,$
4:     for $j \leftarrow i; j \leq |A|, j \leftarrow j + 1$ do
5:         $X_{\text{tmp}} \leftarrow X^* \cup \mathbf{x}_b, b \in B$
6:         $B \leftarrow B \backslash \{b\}$
7:         $\mathtt{L} \leftarrow I\left(\mathtt{W}\left(X_{tmp}, \mathtt{T}\right)\right)$, $\mathtt{H} \leftarrow \mathtt{TL}^{+}$, $\mathtt{T}_{\text{est}} \leftarrow \mathtt{HL}$
8:         if $(\varepsilon_{min} > \|\mathtt{T} - \mathtt{T}_{\mathtt{est}}\|)$ then
9:             $a_{\min} \leftarrow b, \varepsilon_{\min} \leftarrow \|\mathtt{T} - \mathtt{T}_{\text{est}}\|$
10:        else
11:            continue
12:        end if
13:    end for
14:    $X^* \leftarrow X^* \cup \{\mathbf{x}_{a_{\min}}\}$
15:    $A \leftarrow A \backslash \{a_{\min}\}$
16: end for
```

Figure B.1: Support set selection algorithm

C Gradient Descent and Lucas-Kanade

Image alignment consists of moving, and possibly deforming, a template to minimize the difference between the template and an image. Since the first use of image alignment in the Lucas-Kanade optical flow algorithm [8], it has become one of the most widely used techniques in computer vision. Besides the optical flow, some of its other applications include tracking [9][22], medical image registration [109], and face coding [110][111]. Since the Lucas-Kanade algorithm was proposed in 1981 image alignment has become one of the most widely used techniques in computer vision. Numerous algorithms have been proposed and a wide variety of extensions have been made to the original formulation. Great work has been done by Simon Baker, Iain Matthews, Ralph Gross, Takahiro Ishikawa, et. al. in [23][112][33][113][114], where the overview of image alignment methods and most of the extensions of Lucas-Kanade algorithm are described.

The goal is to align a template image $T(\mathbf{x})$ taken in time t to an input image $I(\mathbf{x})$ taken in time $t+1$. The template T is extracted sub-region (e.g. window 7×7 pixels) from the image at time t, where $\mathbf{x} = (x, y)^T$ is a column vector containing the pixel coordinates. Let $\mathtt{W}(\mathbf{x}; \mathbf{p})$ denote set of allowed warps, parameterized by the vector of parameters $\mathbf{p} = (p_1, p_2, \ldots, p_n)^T$.. The warp $\mathtt{W}$ takes the pixel $\mathbf{x}$ in the coordinate frame of the template T and maps it to the sub-pixel location $\mathtt{W}(\mathbf{x}; \mathbf{p})$ in the coordinate frame of the image I. For a small template T it is reasonable to use affine warp [9] (6 parameters in vector $\mathbf{p}$) or homography [22] etc. In general, the number of parameters in $\mathbf{p}$ may be arbitrarily large and $\mathtt{W}$ can be arbitrarily complex.

The goal of the Lucas-Kanade algorithm is to minimize the sum of squared error computed over all pixels $\mathbf{x}$ of the patch T:

$$\sum_{\mathbf{x}} \left[I\left(\mathtt{W}\left(\mathbf{x}; \mathbf{p}\right)\right) - T\left(\mathbf{x}\right) \right]^2, \tag{C.1}$$

the minimization is performed with respect to $\mathbf{p} \in \Omega$ where Ω is $n-$dimensional space of parameters. Minimizing the expression in equation (C.1) is a non-linear optimization. To optimize the expression in (C.1), the Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{p}$ is known and then iteratively solves for increments to the parameters $\boldsymbol{\Delta}\mathbf{p}$, i.e. the equation (C.1) becomes:

$$\sum_{\mathbf{x}} \left[I\left(\mathtt{W}\left(\mathbf{x}; \mathbf{p} + \boldsymbol{\Delta}\mathbf{p}\right)\right) - T\left(\mathbf{x}\right) \right]^2 \tag{C.2}$$

and the minimization is performed with respect to $\boldsymbol{\Delta}\mathbf{p} \in \Omega$. We will call this approach *aditional* (to distinguish it from *compositional* defined later) because of the aditive parameters update

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{\Delta p}. \tag{C.3}$$

Equations (C.2) and (C.3) are iteratively computed until the norm $\|\mathbf{\Delta p}\| \leq \epsilon$, where ϵ is some threshold, or until some number of iterations has been reached.

Derivation of the Lucas-Kanade algorithm

The Lucas-Kanade algorithm is in fact a Gauss-Newton gradient descent non-linear optimization. The non-linear expression in equation (C.2) is linearized by performing a first order Taylor expansion on $I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}+\mathbf{\Delta p}\right)\right)$:

$$\sum_{\mathbf{x}} \left[I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)\right) + \nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}} \mathbf{\Delta p} - T\left(\mathbf{x}\right) \right]^2 , \tag{C.4}$$

where $\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$ is the gradient of image I evaluated at $\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)$. We neglected the high order terms of the Taylor expansion, so the equation (C.4) is only an approximation of equation (C.2). The term $\frac{\partial \mathtt{W}}{\partial \mathbf{p}}$ is the Jacobian of the warp. Minimizing the expression in equation (C.4) is a least squares problem and has a closed from solution which can be derived as follows. The partial derivative of the expression in equation (C.4) with respect to $\mathbf{\Delta p}$ is:

$$2\sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}}\right]^T \left[I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)\right) + \nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}} \mathbf{\Delta p} - T\left(\mathbf{x}\right) \right] , \tag{C.5}$$

where $\nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}}$ is called the steepest descent [23]. Setting this expression to equal zero and solving gives the closed form solution for the minimum of the expression in equation (C.4) as:

$$\mathbf{\Delta p} = \mathtt{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}}\right]^T \left[T\left(\mathbf{x}\right) - I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right], \tag{C.6}$$

where H is the $n \times n$ (Gauss-Newton approximation to the) Hessian matrix:

$$\mathtt{H} = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}}\right]^T \left[\nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}}\right] . \tag{C.7}$$

Equation (C.6) then expresses the fact that the parameter updates $\mathbf{\Delta p}$ are the steepest descent parameter updates multiplied by the inverse of the Hessian matrix. We will call this approach *forward aditional* in order to distinguish it from *inverse compositional* defined later in C. The Lucas-Kanade algorithm then consists of iteratively applying equations (C.6) and (C.3). Denoting $\mathtt{G} = \nabla I \frac{\partial \mathtt{W}}{\partial \mathbf{p}}$ and combining the equations (C.6) and (C.7) we obtain

$$\mathbf{\Delta p} = \sum_{\mathbf{x}} \left(\mathtt{G}^T\mathtt{G}\right)^{-1} \mathtt{G}^T \left[T\left(\mathbf{x}\right) - I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right], \tag{C.8}$$

where $\left(\mathtt{G}^T\mathtt{G}\right)^{-1}\mathtt{G}^T = \mathtt{G}^+$ is the pseudoinverse of matrix $\mathtt{G}$, which needs to be recomputed in every step. For some simple warps such like the translations and the affine warps the Jacobian $\frac{\partial \mathtt{W}}{\partial \mathbf{p}}$ can sometimes be constant [23] and precomputed, which saves computational time. In general, however, not only the image gradient ∇I and Hessian but also the Jacobian must be recomputed in every iteration. Hager in [47] and Dellaert in [48] show some classes of warps, for which the Jacobian can be precomputed.

Selecting the points to track

An issue in tracking is the selection of features (image points) which are well suited for the particular tracking algorithm. A method for selecting good features for Lucas-Kanade tracking algorithm was proposed in [9]. The proposed method is based on Harris corner detector principle. It computes the local autocorrelation function, which measures local appearance changes between two slightly translated image patches. Let X be the set of pixels positions distributed over the current patch centered on image point (x, y). With the small 2D motion vector $(\Delta x, \Delta y)$ the autocorrelation function for point (x, y) defined as

$$c(x,y) = \sum_{(x,y)\in X} \left[I(x_i, y_i) - I(x_i + \Delta x, y_i + \Delta y)\right]^2 . \tag{C.9}$$

We may then approximate the shifted image $I(x + \Delta x, y + \Delta y)$ by Taylor expansion:

$$I(x_i + \Delta x, y_i + \Delta y) \approx I(x_i, y_i) + \left[I_x(x_i, y_i)\, I_y(x_i, y_i)\right] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}, \tag{C.10}$$

where I_x and I_y are partial derivatives. Combining equations (C.9) and (C.10) we obtain

$$\begin{aligned} c(x,y) &= \sum_X \left(\left[I_x(x_i, y_i)\, I_y(x_i, y_i)\right] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}\right)^2 \qquad \text{(C.11)} \\ &= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} \sum_X I_x^2 & \sum_X I_x I_y \\ \sum_X I_x I_y & \sum_X I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\ &= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} C(x,y) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}. \end{aligned}$$

Shi and Thomasi [9] showed that good measure for detecting the corner is

$$min(\lambda_1, \lambda_2) > \epsilon,$$

where λ_1, λ_2 are eigenvalues of matrix $C(x,y)$ and ϵ some chosen threshold value. When both eigenvalues are close to zero, than the local autocorelation function is flat and no distinctive feature is present. When one of eigenvalues is small and other is big, than we found an edge, which is also not a good feature to track because of the aperture problem [115]. But if both eigenvalues are big enough, than there is a corner at the image point (x, y) which is a distinctive feature for the local neighborhood.

In the original Lucas-Kanade paper [8] the authors suggest, that for small template and small motion between 2 consecutive images it is sufficient to estimate only translation parameters. In [9] the authors argued and experimentaly verified, that better tracking results were achieved, when after 2D motion estimation the affine transformation of the patch was estimated as a second tracking step. Another observation from [9] is that smaller templates are better for tracking, because they suffer less for depth discontinuity. On the other hand it is more difficult to estimate the affine transformation for small templates. Small template (with a low number of sampled pixels) contains less information, than some larger template. When both templates undergo the same transformation we will get more precise estimation when using the larger template than the smaller one.

Dealing with the Fast Motion

Until now, we suggested that the template motion is small and it is sufficient to work with some small neighborhood of the previous feature position. But what if the motion between two consecutive images is too large? Than standard Lucas-Kanade tracking algorithm fails. The solution is to use pyramidal implementation of Lucas-Kanade tracker, which makes the tracking much more robust for large motion, see [116]. The trick is to start tracking in subsampled images, where the image and template resolutions are lower and by lowering the resolution the motion size is reduced too. After the LK algorithm convergence in the low-res image we use another image with higher resolution, where the template position is refined by another few LK iterations. This way we continue till the original image and template size and the final 2D motion is composed during passing through pyramid floors . The idea of subsampling both image and template is reasonable, when we track *good features* [9]. For a complete derivation of the pyramidal algorithm see [116].

Inverse Compositional Algorithm

Aditive parameters update specified in section C may be replaced *compositional* parameters update derived in [23]. This approach to parameters update was used for example in [117] or [110]. The compositional algorithm minimizes the equation

$$\sum_{\mathbf{x}} \left[I\left(\mathtt{W}\left(\mathtt{W}\left(\mathbf{x};\boldsymbol{\Delta}\mathbf{p}\right);\mathbf{p}\right)\right) - T\left(\mathbf{x}\right)\right]^2, \tag{C.12}$$

This equation is minimized with respect to $\boldsymbol{\Delta}\mathbf{p}$ in each iteration and then update the estimate of warp as

$$\mathtt{W}\left(\mathbf{x};\mathbf{p}\right) \leftarrow \mathtt{W}\left(\mathbf{x};\mathbf{p}\right) \circ \mathtt{W}\left(\mathbf{x};\boldsymbol{\Delta}\mathbf{p}\right), \tag{C.13}$$

i.e. the compositional approach iteratively solves for an incremental warp $\mathtt{W}\left(\mathbf{x};\boldsymbol{\Delta}\mathbf{p}\right)$ rather than an additive update to the parameters $\boldsymbol{\Delta}\mathbf{p}$. The compositional and additive approaches are proved to be equivalent in [23].

The disadvantage of Lucas-Kanade tracking method is the fact, that even if we have a simple warp, the Jacobian of which may be precomputed, we still need to recompute the Hessian (C.7) in every iteration as has been pointed out e.g. in [48][47][117]. The key to avoid time consuming Hessian computation is to use the *inverse compositional algorithm* [23]. The main idea is to switch the roles of image and the template. A restricted version of the inverse compositional algorithm was proposed for homographies in [48]. The inverse compositional algorithm minimizes expression

$$\sum_{\mathbf{x}} \left[T\left(\mathtt{W}\left(\mathbf{x};\boldsymbol{\Delta}\mathbf{p}\right)\right) - I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right]^2 \tag{C.14}$$

with respect to $\boldsymbol{\Delta}\mathbf{p}$ and than updates the warp

$$\mathtt{W}\left(\mathbf{x};\mathbf{p}\right) \leftarrow \mathtt{W}\left(\mathbf{x};\mathbf{p}\right) \circ \mathtt{W}\left(\mathbf{x};\boldsymbol{\Delta}\mathbf{p}\right)^{-1}. \tag{C.15}$$

The only difference from the update in the forwards compositional algorithm in equation (C.13) is that the incremental warp $\mathtt{W}\left(\mathbf{x};\boldsymbol{\Delta}\mathbf{p}\right)$ is inverted before it is composed with the current estimate. Performing the Taylor expansion on (C.14) gives

$$\sum_{\mathbf{x}} \left[T\left(\mathtt{W}\left(\mathbf{x};\mathbf{0}\right)\right) + \nabla T \frac{\partial \mathtt{W}}{\partial \mathbf{p}} \boldsymbol{\Delta}\mathbf{p} - I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right]^2. \tag{C.16}$$

The least-squares solution to this problem is

$$\boldsymbol{\Delta}\mathbf{p} = \mathtt{H}^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathtt{W}}{\partial \mathbf{p}}\right]^T \left[I\left(\mathtt{W}\left(\mathbf{x};\mathbf{p}\right)\right) - T\left(\mathbf{x}\right)\right], \tag{C.17}$$

where the I in former Hessian matrix is replaced replaced with T:

$$\mathtt{H} = \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathtt{W}}{\partial \mathbf{p}}\right]^T \left[\nabla T \frac{\partial \mathtt{W}}{\partial \mathbf{p}}\right]. \tag{C.18}$$

Now there is nothing in Hessian that depends on $\mathbf{p}$, it is constant across iterations and can be precomputed.

There are many extensions of the Lucas-Kanade algorithm. The possible linear appearance variation was used in [111] and derived in [33]. It is recomendable to use the appearance model, since it highly increases the robustness of tracking, but an off-line learning phase is needed with more than one training image to capture enough appearance variation. The choise of the error function is discussed (L2 norm, robust error function) in [112], adition of priors on the warp and appearance parameters [113], etc. Different gradient descent approximations are tested in [23], specifically Gauss-Newton (traditionally used), Newton, Steepest-Descent, Gauss-Newton Diagonal, Newton Diagonal and Levenberg-Marquardt. The result is, that *Gauss-Newton* and *Levenberg-Marquardt* are the most efficient and successful in convergence rate. The linear and quadratic subset of template pixels was proposed by Benhimane in [118] to improve the tracking performance by reducing the computational time and improvement of convergence properties. This pixel subset selection method is very useful for tracking large patches, for small templates it has no signifficant.

Bibliography

[1] D. Hurych and T. Svoboda, “Incremental learning and validation of sequential predictors in video browsing application,” in *VISIGRAPP 2010: International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, vol. 1, May 2010, pp. 467–474.

[2] D. Hurych, K. Zimmermann, and T. Svoboda, “Fast learnable object tracking and detection in high-resolution omnidirectional images,” in *Proceedings of VISAPP, International Conference on Computer Vision Theory and Applications*, Setubal, Portugal, March 2011, pp. 521–530.

[3] ——, “Detection of unseen patches trackable by linear predictors,” in *Computer Vision Winter Workshop*, February 2011, pp. 107–114.

[4] K. Zimmermann, D. Hurych, and T. Svoboda, “Improving cascade of classifiers by sliding window alignment in between,” in *IEEE International Conferencer on Automation, Robotics and Applications*, December 2011, pp. 196–201.

[5] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: a statistical view of boosting,” *Annals of Statistics*, vol. 28, pp. 337–407, 1998.

[6] K. Zimmermann, D. Hurych, and T. Svoboda, “Exploiting features – locally interleaved sequential alignment for object detection,” in *IEEE Asian Conference on Computer Vision*, November 2012, pp. 196–201.

[7] ——, “Non-rigid object detection with local interleaved sequential alignment (lisa),” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 731–743, April 2014.

[8] B. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *International Journal of Computer Vision*, August 1981, pp. 674–679.

[9] J. Shi and C. Tomasi, “Good features to track,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593 – 600.

[10] F. Jurie and M. Dhome, “Hyperplane approximation for template matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 996–1000, July 2002.

[11] K. Zimmermann, J. Matas, and T. Svoboda, "Tracking by an optimal sequence of linear predictors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 677–692, 2009.

[12] S. Hinterstoisser, S. Benhimane, N. Navab, P. Fua, and V. Lepetit, "On-line learning of patch perspective rectification for efficient object detection," in *Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[13] T. Cootes, G. Edwards, and C. Taylor, "Active appearance models," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 23, no. 6, pp. 681–685, June 2001.

[14] D. Hurych, T. Svoboda, J. Trojanova, and Y. US, "Active shape model and linear predictors for face association refinement," in *The IEEE International Workshop on Visual Surveillance held at IEEE International Conference on Computer Vision*, Kyoto, Japan, 2009, pp. 1193–1200.

[15] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[16] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 1–8.

[17] S. Avidan, "Support vector tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 8, pp. 1064–1072, 2004.

[18] P. Felzenszwalb, R. Girshick, and D. McAllester, "Cascade object detection with deformable part models," in *IEEE Conference on Computer Vision and Pattern Recognition*, june 2010, pp. 2241–2248.

[19] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[20] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *IEEE European Conference on Computer Vision*, Graz Austria, May 2006, pp. 346–359.

[21] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo form maximally stable extremel regions," in *Brittish Machine Vision Conference*, London, UK, 2002, pp. 384–393.

[22] S. Hinterstoisser, O. Kutter, N. Navab, P. Fua, and V. Lepetit, "Real-time learning of accurate patch rectification," in *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, Florida, USA, 2009, pp. 2945 – 2952.

[23] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework, part 1," Carnegie-Mellon University, Robotics Institute, Tech. Rep. CMU-RI-TR-02-16, July 2002.

[24] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learning-detection.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.

[25] C. D., R. V., and M. P., “Real-time tracking of non-rigid objects using mean shift,” in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2. Hilton Head Island, SC: IEEE Computer Society, 2000, pp. 142–149.

[26] A. Leung and S. Gong, “Mean-shift tracking with random sampling,” in *British Machine Vision Conference*, 2006, pp. 729–738.

[27] C. D., R. Visvanathan, and P. Meer, “Kernel-based object tracking,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May 2003, pp. 564–575.

[28] K. Zimmermann, “Fast learnable methods for object tracking,” in *Ph.D. Thesis.* Czech Technical University in Prague, 2008.

[29] A. Jepson, D. Fleet, and T. El-Maraghi, “Robust on-line appearance models for visual tracking,” in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2008, pp. 415–422.

[30] I. Matthews, T. Ishikawa, and S. Baker, “The template update problem,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 810–815, 2004.

[31] K.-C. Lee and D. Kriegman, “On-line learning of probabilistic appearance manifolds for video-based recognition and tracking,” in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2005, pp. 852–859.

[32] R. Gross, I. Matthews, and S. Baker, “Generic vs. person specific active appearance models,” *Image and Vision Computing*, vol. 23, no. 11, pp. 1080–1093, November 2005.

[33] S. Baker, R. Gross, and I. Matthews, “Lucas-kanade 20 years on: A unifying framework, part 3,” Carnegie-Mellon University, Robotics Institute, Tech. Rep. CMU-RI-TR-03-35, November 2003.

[34] L. Ellis, J. Matas, and R. Bowden, “On-line learning and partitioning of linear displacement predictors for tracking,” in *British Machine Vision Conference*, September 2008, pp. 33–42.

[35] L. Ellis, N. Dowson, J. Matas, and R. Bowden, “Linear predictors for fast simultaneous modelling and tracking,” in *International Conference on Computer Vision, Workshop on Non-rigid Registration and Tracking Through Learning*, October 2007, pp. 1–8.

[36] H. Grabner, M. Grabner, and H. Bischof, “Real-time tracking via on-line boosting,” in *British Machine Vision Conference*, vol. 1, 2006, pp. 47–56.

[37] N. Dowson and R. Bowden, “N-tier simultaneous modelling and tracking for arbitrary warps,” in *British Machine Vision Conference*, vol. 2, 2006, pp. 569–578.

[38] D. Ross, J. Lim, R. Lin, and M. Yang, “Incremental learning for robust visual tracking,” *International Journal of Computer Vision*, vol. 77, no. 1-3, pp. 125–141, May 2008.

[39] A. Yilmaz, O. Javed, and M. Shah, “Object tracking: A survey,” *ACM Computing Surveys (CSUR)*, vol. 38, no. 4, pp. 13–36, December 2006.

[40] E. Murphy-Chutorian and M. Trivedi, “Head pose estimation in computer vision: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 607–626, April 2009.

[41] K. Zimmermann, T. Svoboda, and J. Matas, “Anytime learning for the NoSLLiP tracker,” *Image and Vision Computing, Special Issue: Perception Action Learning*, vol. 27, no. 11, pp. 1695–1701, October 2009.

[42] J. Sivic and A. Zisserman, “Efficient visual search of videos cast as text retrieval,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 591–606, April 2009.

[43] M. Li, W. Chen, K. Huang, and T. Tan, “Visual tracking via incremental self-tuning particle filtering on the affine group,” in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2010, pp. 1315–1322.

[44] M. Özuysal, M. Calonder, V. Lepetit, and P. Fua, “Fast keypoint recognition using random ferns,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 448–461, March 2010.

[45] J. Šochman and J. Matas, “Waldboost - learning for time constrained sequential detection,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 150–157.

[46] S. Baker and I. Matthews, “Lucas-kanade 20 years on: A unifying framework,” *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, 2004.

[47] G. D. Hager and P. N. Belhumeur, “Efficient region tracking with parametric models of geometry and illumination,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 10, pp. 1025–1039, 1998.

[48] F. Dellaert and R. Collins, “Fast image-based tracking by selective pixel integration,” in *IEEE International Conference on Computer Vision, Workshop of Frame-Rate Vision*, 1999, pp. 1–22.

[49] S. Holzer, S. Ilic, and N. Navab, "Adaptive linear predictors for real-time tracking," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2010, pp. 1807–1814.

[50] F. L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, no. 6, pp. 567–585, 1989.

[51] R. Penrose, "A generalized inverse for matrices," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, no. 3, pp. 406–413, 1955.

[52] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, 2007.

[53] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision." in *International Joint Conference on Artificial Intelligence*, 1981, pp. 674–679.

[54] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," in *International Journal of Computer Vision*, vol. 56, 2004, pp. 221–255.

[55] F. Jurie and M. Dhome, "Real-time robust template matching," in *British Machine Vision Conference*, 2002, pp. 123–131.

[56] M. Özuysal, V. Lepetit, F. Fleuret, and P. Fua, "Feature harvesting for tracking-by-detection," in *European Conference on Computer Vision*, vol. 3953. Springer, 2006, pp. 592–605.

[57] H. Grabner and H. Bischof, "On-line boosting and vision," in *IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2006, pp. 260–267.

[58] L. Vacchetti, V. Lepetit, and P. Fua, "Stable real-time 3D tracking using online and offline information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1385–1391, 2004.

[59] T. Cootes, G. Edwards, and C. Taylor, "Active appearance models," in *European Conference on Computer Vision*, vol. 2, London, UK, 1998, pp. 484–498.

[60] O. Williams, A. Blake, and R. Cipolla, "Sparse bayesian learning for efficient visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1292–1304, 2005.

[61] D. Cristinacce and T. Cootes, "Feature detection and tracking with constrained local models," in *British Machine Vision Conference*, 2006, pp. 929–938.

[62] M. E. Tipping, “Sparse bayesian learning and the relevance vector machine,” *Journal of Machine Learning Research*, vol. 1, pp. 211–244, 2001.

[63] A. Agarwal and B. Triggs, “Recovering 3d human pose from monocular images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 1, pp. 44–58, 2006.

[64] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik, “Support vector regression machines,” *Advances in Neural Information Processing Systems*, vol. 9, pp. 156–161, 1996.

[65] S. K. Zhou, B. Georgescu, X. S. Zhou, and D. Comaniciu, “Image based regression using boosting method,” in *IEEE International Conference on Computer Vision*, vol. 1, Washington, DC, USA, 2005, pp. 541–548.

[66] K. Zimmermann, T. Svoboda, and J. Matas, “Adaptive parameter optimization for real-time tracking,” in *Workshop on Non-rigid Registration and Tracking Through Learning*, Rio de Janeiro, Brazil, 2007.

[67] R. Schapire, “The boosting approach to machine learning: An overview,” in *Lecture Notes in Statistics, Nonlinear Estimation and Classification*, vol. 171, 2003, pp. 149–171.

[68] D. L. Shrestha and D. P. Solomatine, “Experiments with adaboost.rt, an improved boosting scheme for regression,” *Neural Computation*, vol. 18, pp. 1678–1710, 2006.

[69] P. Bühlmann and T. Hothorn, “Boosting algorithms: Regularization, prediction and model fitting,” *Statistical Science*, vol. 22, no. 4, pp. 477–505, 2007.

[70] Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm,” in *Machine Learning*, 1996, pp. 148–156.

[71] L. Breiman, “Arcing classifiers (with discussion),” *Annual Statistics*, vol. 26, pp. 801–849, 1998.

[72] ——, “Prediction games and arcing algorithms,” *Neural Computation*, vol. 11, pp. 1493–1517, 1999.

[73] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: a statistical view of boosting,” *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[74] R. E. Schapire and Y. Singer, “Improved boosting algorithms using confidence-rated predictions,” *Machine Learning*, vol. 37, pp. 297–336, 1999.

[75] G. Rätsch, T. Onoda, and K.-R. Müller, “Soft margins for adaboost,” *Machine Learning*, vol. 42, no. 3, pp. 287–320, 2001.

[76] D. Solomatine and D. Shrestha, “Adaboost.rt: a boosting algorithm for regression problems,” in *International Joint Conference on Neural Networks*, July 2004, pp. 1163–1168.

[77] X. Perrotton, M. Sturzel, and M. Roux, “Implicit hierarchical boosting for multi-view object detection,” in *IEEE Conference on Computer Vision and Pattern Recognition*, june 2010, pp. 958 –965.

[78] L. Bourdev and J. Brandt, “Robust object detection via soft cascade,” in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2005, pp. 236–243.

[79] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischof, “Online multi-class lpboost,” in *IEEE Conference on Computer Vision and Pattern Recognition*, june 2010, pp. 3570 –3577.

[80] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, “Multiple kernels for object detection,” in *IEEE International Conference on Computer Vision*, 2009, pp. 606–613.

[81] H. Harzallah, F. Jurie, and C. Schmid, “Combining efficient object localization and image classification,” in *IEEE International Conference on Computer Vision*, 2009, pp. 237–244.

[82] C. Lampert, “An efficient divide-and-conquer cascade for nonlinear object detection,” in *IEEE Conference on Computer Vision and Pattern Recognition*, june 2010, pp. 1022 –1029.

[83] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan, “Fast human detection using a cascade of histograms of oriented gradients,” in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 1491 – 1498.

[84] P. Dollar, P. Welinder, and P. Perona, “Cascaded pose regression,” in *IEEE Conference on Computer Vision and Pattern Recognition*, june 2010, pp. 1078 –1085.

[85] M. Villamizar, F. Moreno-Noguer, J. Andrade-Cetto, and A. Sanfeliu, “Efficient rotation invariant object detection using boosted random ferns,” in *IEEE Conference on Computer Vision and Pattern Recognition*, june 2010, pp. 1038 –1045.

[86] K. Ali, F. Fleuret, D. Hasler, and P. Fua, “A real-time deformable detector,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 225–239, 2012.

[87] B. Babenko, P. Dollár, Z. Tu, and S. Belongie, “Simultaneous learning and alignment: Multi-instance and multi-pose learning,” in *Workshop on Faces in Real-Life Images: Detection, Alignment, and Recognition held at European Conference on Computer Vision*, 2008.

[88] P. Viola and M. Jones, "Fast multi-view face detection," in *Mitsubishi Elestric Research Laboratories - Technical Report*, August 2003.

[89] J. Zhang, S. Zhou, L. McMillan, and D. Comaniciu, "Joint real-time object detection and pose estimation using probabilistic boosting network," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[90] C. Huang, H. Ai, Y. Li, and S. Lao, "Vector boosting for rotation invariant multi-view face detection," in *International Conference on Computer Vision*, 2005, pp. 446–453.

[91] S. Li and Z. Zhang, "Flatboost learning and statistical face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1112–1123, 2004.

[92] A. Mohan, C. Papageorgiou, and T. Poggio, "Example-based object detection in images by components," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 4, pp. 349–361, 2001.

[93] P. Viola, M. Jones, and D. Snow, "Detection pedestrians using patterns of motion and appearance," in *International Conference on Computer Vision*, 2003, pp. 734–741.

[94] A. Torralba, K. Murphy, and W. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 854–869, 2007.

[95] X. Zhu and D. Ramanan, "Face detection, pose estimation, and landmark localization in the wild," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 2879–2886.

[96] F. Fleuret and D. Geman, "Stationary features and cat detection," *Journal of Machine Learning Research*, vol. 9, pp. 2549–2578, 2008.

[97] I. Kokkinos, "Rapid deformable object detection using dual-tree branch-and-bound," in *Advances in Neural Information Processing Systems*, 2011, pp. 2681–2689.

[98] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[99] Y. Amit and A. Trouvé, "Pop: Patchwork of parts models for object recognition," *International Journal of Computer Vision*, vol. 75, no. 2, pp. 677–692, 2007.

[100] M. Pedersoli, A. Vedaldi, and J. Gonzàlez, "A coarse-to-fine approach for fast deformable object detection," in *IEEE Conference on Computer Vision and Pattern Recognition.* IEEE Computer Society, 2011, pp. 1353–1360.

[101] C. H. Lampert, M. B. Blaschko, and T. Hoffmann, "Efficient subwindow search: A branch and bound framework for object localization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2129–2142, 2009.

[102] P. Dollár, R. Appel, and W. Kienzle, "Crosstalk cascades for frame-rate pedestrian detection," in *European Conference on Computer Vision*, 2012, pp. 645–659.

[103] X. Cao, Y. Wei, F. Wen, and J. Sun, "Face alignment by explicit shape regression," in *IEEE Conference on Computer Vision and Pattern Recognition.* IEEE Computer Society, 2012, pp. 2887–2894.

[104] H. Wu, X. Liu, and G. Doretto, "Face alignment via boosted ranking model," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[105] X. Liu, "Generic face alignment using boosted appearance model," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[106] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 6, pp. 559–572, 1901.

[107] P. Viola and M. Jones, "Robust real-time face detection," in *IEEE International Conference on Computer Vision*, vol. 2, 2001, pp. 747–757.

[108] M. Koestinger, P. Wohlhart, P. M. Roth, and H. Bischof, "Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization," in *IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011.

[109] G. Christensen and H. Johnson, "Image consistent registration," *IEEE Transactions on Medical Imaging*, pp. 568–582, July 2001.

[110] S. Baker and I. Matthews, "Equivalence and efficiency of image alignment algorithms," in *IEEE Conference on Computer Vision and Pattern Recognition.* IEEE Computer Society, 2001, pp. 1090–1097.

[111] I. Matthews and S. Baker, "Active appearance models revisited," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 135–164, November 2004.

[112] S. Baker, R. Gross, T. Ishikawa, and I. Matthews, "Lucas-kanade 20 years on: A unifying framework, part 2," Carnegie-Mellon University, Robotics Institute, Tech. Rep. CMU-RI-TR-03-01, February 2003.

[113] S. Baker, R. Gross, and I. Matthews, "Lucas-kanade 20 years on: A unifying framework, part 4," Carnegie-Mellon University, Robotics Institute, Tech. Rep. CMU-RI-TR-04-14, February 2004.

[114] S. Baker and I. Matthews, “Lucas-kanade 20 years on: A unifying framework, part 5,” Carnegie-Mellon University, Robotics Institute, Tech. Rep. CMU-RI-TR-04-64, November 2004.

[115] B. Horn and B. Schunck, “Determining optical flow,” Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Tech. Rep., 1981.

[116] J. Bouguet, “Pyramidal implementation of the lucas kanade feature tracker description of the algorithm,” Intel Corporation Microprocessor Research Labs, Tech. Rep., 2000.

[117] H.-Y. Shum and R. Szeliski, “Construction of panoramic image mosaics with global and local alignment,” *International Journal of Computer Vision*, vol. 36, no. 2, pp. 101–130, July 2000.

[118] S. Benhimane, A. Ladikos, V. Lepetit, and N. Navab, “Linear and quadratic subsets for template-based tracking,” in *IEEE Conference on Computer Vision and Pattern Recognition*, September 2007, pp. 1–6.