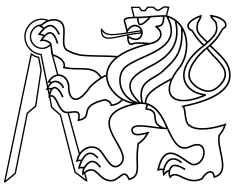
CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY

PhD THESIS

ISSN 1213-2365

Fast Learnable Methods for Object Tracking

Karel Zimmermann

zimmerk@cmp.felk.cvut.cz

CTU–CMP–2008–09

October 30, 2008

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/zimmermann/Zimmermann-TR-2008-09.pdf

Thesis Advisor: Jiří Matas and Tomáš Svoboda

I gratefully acknowledge Czech Academy of Sciences project 1ET101210407, which supported my research. Partial support of CTU project No. 0608813 and EC project FP6-IST-027787 DIRAC is also acknowledged.

Research Reports of CMP, Czech Technical University in Prague, No. 9, 2008

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

Fast Learnable Methods for Object Tracking

A Dissertation Presented to the Faculty of the Electrical Engineering of the Czech Technical University in Prague in Partial Fulfillment of the Requirements for the Ph.D. Degree in Study Programme No. P2612 - Electrotechnics and Informatics, branch No. 3902V035 - Artificial Intelligence and Biocybernetics, by

Karel Zimmermann

Prague, April 2008

Supervisor: **Doc. Dr. Ing. Jiří Matas**

Supervisor-Specialist: **Ing. Tomáš Svoboda, 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

Abstract

We propose a learning approach to tracking. The learning procedure explicitly minimizes the computational complexity of the tracking process subject to a user-defined probability of failure (loss-of-lock) and precision. In particular, we studied tracking methods estimating the object position by a learned linear mapping between intensities and motion. This mapping is called Learned Linear Predictor (LLiP). We extended the predictor to the Sequence of LLiPs (SLLiP). The proposed learning approach finds the SLLiP with the lowest computational complexity of the tracking subject to a user predefined range, i.e., region of motions within which the sequential predictor operates, and accuracy. Note, that since the individual predictor is a special case of the sequential one, the optimal sequence is superior to a single predictor.

Since variances of the object appearance, caused for example by non-rigid deformations or uneven illumination, can often make the accurate motion estimation impossible, we also extended the predictors to Parameter sensitive LLiPs (PLLiPs). PLLiP is a sub-space of the space of all individual predictors. In the tracking, the current object appearance is encoded into the parameters (coordinates) which select the most convenient predictor from the PLLiP sub-space. Both the sub-space and the appearance encoding are simultaneously optimized.

Robustness of the tracker is achieved by modeling the object as a collection of local motion predictors. Object motion is estimated by the outlier-tolerant Ransac algorithm from local predictions. The object tracker is formed by a Number of Sequences of Learned Linear Predictors (NoSLLiP).

Efficiency of the tracker stems from (i) the simplicity of the local predictors and (ii) from the fact that all design decisions — the number of local predictors used by the tracker, their computational complexity (i.e., the number of observations the prediction is based on), locations as well as the number of Ransac iterations are all subject to the optimization (learning) process. All the time-consuming operations are performed during the learning stage — the tracking is reduced to only a few hundreds of integer multiplications in each step.

The proposed approaches were verified on publicly-available sequences with approximately 12000 frames with ground-truth besides. Experiments demonstrate superiority in frame rates and robustness compared to the SIFT detector, Lucas-Kanade tracker, and other trackers.

Acknowledgments

I would like to express my thanks to my colleagues at CMP who I had the pleasure of working with, especially to Alexander Shekhovtsov and Prof. Václav Hlaváč who also read my thesis and provided me valuable comments. I am greatly indebted to both of my advisors for guiding me throughout my research. Their friendly support, patience and theoretical and practical help have been paramount to the successful completion of my PhD study. I also would like to thank to my family and to my friends for all their support that made it possible for me to finish this thesis.

I gratefully acknowledge Czech Academy of Sciences project 1ET101210407, which supported my research. Partial support of IGS project CTU0608813 and EC project FP6-IST-027787 DIRAC is also acknowledged.

Contents

Keywords: tracking by learning, regression, computer vision

Table of Definitions

Notation

h	scalar value
$\mathbf{h}$	column vector[1]
$\mathtt{H}$	matrix
$h(.), \mathbf{h}(.), \mathtt{H}(.)$	functions which map domain "." to scalars, vectors or matrices, respectively.
H	set
$\mathcal{H}$	set of functions or set of sets.
$\mathbf{h}^i$	column vector denoting i-th column of $\mathtt{H}$
$\mathbf{h}_j^\top$	row vector denoting j-th row of $\mathtt{H}$
$h_j^i = [\mathtt{H}]_{i,j}$	$[i,j]$-th element of matrix $\mathtt{H}$
h_j	j-th element of vector $\mathbf{h}$
$\mathtt{H}^k$	approximation of $\mathtt{H}$ in k-th iteration
$\mathtt{h}^*, \mathtt{H}^*$	optimal $\mathbf{h}$ or $\mathtt{H}$
$\mathbf{0} = [0 \dots 0]^\top$	vector of zeros ($\mathbf{0}_m$ denotes $m \times 1$ vector of zeros)
$\mathbf{1} = [1 \dots 1]^\top$	vector of ones ($\mathbf{1}_m$ denotes $m \times 1$ vector of ones)
$\mathtt{0} = [\mathbf{0} \dots \mathbf{0}]$	matrix of zeros
$\mathtt{1} = [\mathbf{1} \dots \mathbf{1}]$	matrix of ones
$\mathtt{H} \in H$	means that $\mathtt{H}$ is element of set H
$\mathtt{H} \succeq 0$	means that $\mathtt{H}$ is from a positive semi-definite cone
$\forall$, $\exists$	universal and existentional quantifiers
$\otimes$	Kronecker product
$\bullet$	Entry-wise product (sometimes called Hadamard product)
$\setminus$	set minus
$\circ$	warp, e.g., $(\mathbf{t} \circ X)$ is set of pixels X transformed by warp with parameters $\mathbf{t}$
$\det \mathtt{H}$	determinant of matrix $\mathtt{H}$
$\text{trace}\,(\mathtt{H}) = \sum_i h_i^i$	trace of matrix $\mathtt{H}$
$\mathtt{H}^+ = \mathtt{H}^\top(\mathtt{H}\mathtt{H}^\top)^{-1}$	Pseudo-inverse of $\mathtt{H}$
$\|\mathtt{H}\|_F = \sqrt{\sum_{i,j} \|h_j^i\|^2}$	Frobenius norm of $\mathtt{H}$
$\|\mathbf{h}\|_2 = \sqrt{\sum_i h_i^2}$	Euclidean norm of $\mathbf{h}$
$\|\mathbf{h}\|_1 = \sum_i \|h_i\|$	Manhattan norm of $\mathbf{h}$
$\|\mathbf{h}\|_\infty = \max_i \|h_i\|$	Infinity norm of $\mathbf{h}$
$\|h\|, \|H\|$	Absolute value of h, number of elements in set H.

[1] We usually denotes vectors by small bold letters e.g., $\mathbf{h}, \mathbf{g}$ however, vectors of image intensities are denoted by bold capital letters, e.g., $\mathbf{I}, \mathbf{J}$ because $\mathbf{i}, \mathbf{j}$ would look strange.

Commonly used symbols and abbreviations

LS	Least Squares (method)
MM	Minimax (method)
SSD	Sum of Square Differences
MSE	Mean Square Error
PCA	Principal Component Analysis
AAM	Active Appearance Model
LLiP	Learned Linear Predictor (Definition 9)
SLLiP	Sequential Learned Linear Predictor (Definition 10)
MM SLLiP	SLLiP constructed from MM learned LLiPs (Section 4.2)
LS SLLiP	SLLiP constructed from LS learned LLiPs (Section 4.3)
PLLiP	Parameter sensitive Learned Linear Predictor (Definition 14)
X	Support set.
φ, φ^+	Predictor, λ-minimal predictor.
$\hat{\varphi}$	Regressor.
Φ	Sequential predictor.
ω, ω^+	The set of achievable predictors, the set of λ-minimal predictors.
Ω, Ω^+	The set of sequential predictors, the set of sequential predictors created from λ-minimal predictors.
c, C	Complexity of the predictor, set of considered complexities.
R_r, r, R	Range, radius of range of the predictor, set of considered ranges.
Λ_λ, λ	Uncertainty region, radius of uncertainty region of the predictor.
ϵ	Prediction error of the predictor.
$\mathtt{H}$	Coefficients of linear predictor/regressor.
$\mathbf{t}$	Vector of pose parameters.
$\mathbf{I}, \mathbf{J}$	Vectors of image intensities.
$\mathtt{I}, \mathtt{J}, \mathtt{T}$	Matrices column-wise formed from $\mathbf{I}, \mathbf{J}, \mathbf{t}$, respectively.
$\boldsymbol{\theta}$	Vector of appearance parameters.
$\mathbb{R}, \mathbb{R}^+$	The set of real numbers, real positive numbers.
$\mathbb{N}, \mathbb{N}^+$	The set of integral numbers, integral positive numbers.

1 Introduction

Visual tracking is the process of repeated estimation of the state of an object (e.g., position) in an image given state(s) in previous frame(s). Tracking has many applications such as surveillance, 3D object modeling, augmented reality, medical imaging and others. Since many applications have real-time requirements, very low computational complexity is a highly desirable property. Our primary objective is to find a very fast tracking method with defined precision and robustness.

A natural formulation of tracking is search for the state which optimize a similarity criterion function. For example Lucas and Kanade [37, 3] use the steepest descent optimization to minimize the sum of square differences between the template and image data, see Figure 1.1. Other approaches [41, 2, 27] scan the image by a learned classifier, which evaluates the similarity criterion. However, there are also regression-based methods [33, 19, 55] which do not require any criterion function and estimate the object state directly from the observed intensities by a learned regression mapping. In principle, they collect training examples — pairs of observed intensities and corresponding states and use machine learning techniques to learn the regression function. In tracking, the regression method is initialized by the previous state or, if available, by the state derived from a dynamic model. Learned regression function estimates actual object state directly from the intensities observed around the initial location.

Our main contribution is related to the regression-based tracking methods. Rather than proposing a special learning procedure for a special type of the regression function, we present an optimal concatenation of different regression functions into a sequence. We claim that it is typically much easier to estimate the actual state if the method is initialized in a close neighbourhood of the actual state. Accepting this assumption, it is better to exploit a less complex regression function for coarse state estimation and use the newly obtained state for the initialization of another regression function. The coarse estimate of the state allows to the following regression function operate within a smaller range of the states. Such a function naturally achieves a higher precision with a reasonable complexity. Similarly, another ancestors again refine from the precision of previously estimated states. Hence, instead of a very complex regression function, we use a sequence of simpler functions concatenated in the way that each of the functions in the sequence compensate only errors of its predecessor and thus refines from the already computed coarse estimation of the state.

Since the computational time of tracking (i.e., the overall complexity of the used regression method) is usually an issue, the learning is formulated as a minimization of the complexity subject to a user predefined accuracy and robustness. Note, that a

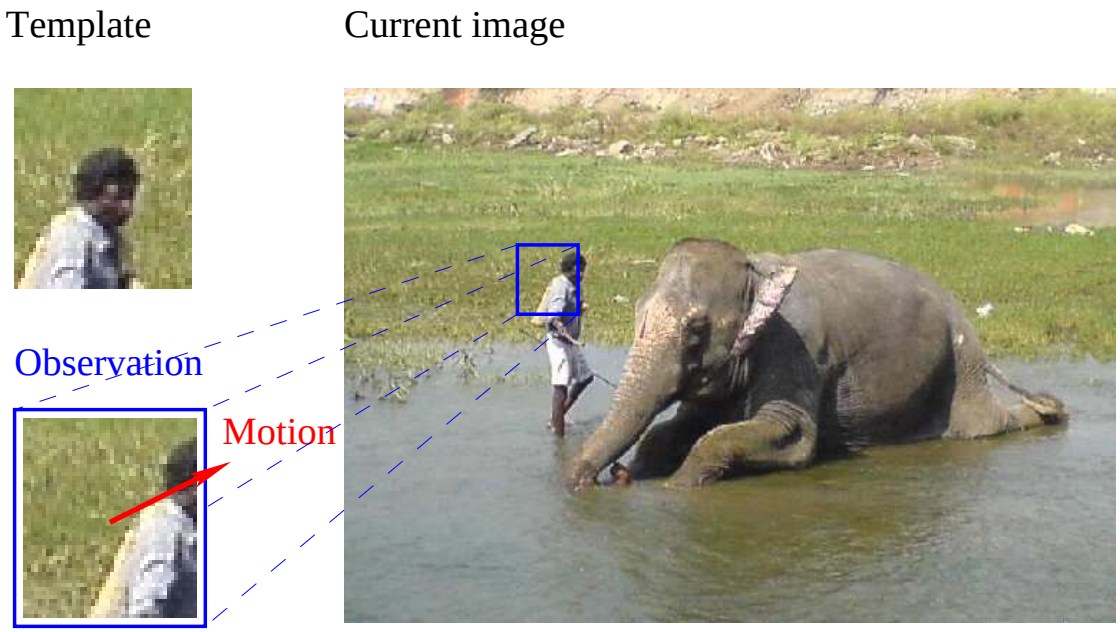


Figure 1.1: **Visual tracking** is the process of repeated estimation of state of an object given an image and state(s) in previous frame(s).

single regression function is a special case of the sequence. Since the globally optimal solution is found, the sequence is superior to a single regression function. Arbitrary regression function allows for such concatenation, but we observed that the sequence of linear functions achieved a very high precision at a very low computational complexity. Focusing on the sequences of the linear functions, we achieved more then real-time tracking algorithm, which estimates the object state using only a fraction of processing power of an ordinary computer.

In the next chapter the state-of-the-art tracking approaches are overviewed. Contributions of the thesis with respect to the state-of-the-art and the thesis outline are summarized afterwards in terms of the established terminology. List of publications related to the thesis is on the last page of the work.

2 State-of-the-art

In this chapter, we summarize state-of-the-art and define the tracking more formally. Given previous state(s) S_0 and current image I, the tracking method estimates the current state of the object by some algorithm $\varphi(\mathbf{I}, S_0)$ as follows:

$$\mathbf{s} = \varphi(\mathbf{I}, S_0). \tag{2.1}$$

The *state* of an object is vector of *essential* and *auxiliary* parameters. Essential parameters are parameters, the estimation of which is required by the application and the quality of which is evaluated. Auxiliary parameters are used internally to improve the quality of the essential parameters estimation. For example in medical applications, local deformations of a beating heart are usually much more essential than the global position however, the estimation of the global position is the necessary condition making the algorithm works. Therefore the global position is an auxiliary parameter, which must be estimated as well. In human tracking applications, the user is usually interested in human positions and the current appearance is just and auxiliary parameter. Lucas-Kanade tracker [37] does not contain any auxiliary parameters however, it is often extended to the tracking followed by a template update algorithm, then the current template becomes auxiliary part of the state of the tracker. Condensation-based tracking algorithms [31] model probability distribution of the state. These distributions are the auxiliary parameters and their maxima usually corresponds to the essential parameters. Kalman filter uses state, which consists of position, velocity, acceleration etc. and the essential parameters are computed as a linear combination of the state.

Especially, if no previous states S_0 are considered mapping φ corresponds to *detection* [53, 36, 38, 48] or *segmentation* [49, 57] problems. In detection problem, it is usually searched for the object position and the scale or rotation are auxiliary parameters. Segmentation is such mapping φ, which assigns non-zero labels to all pixels belonging to the object(s) and zero labels to the pixels belonging to the background. The state in the segmentation problem thus consists of all image pixels.

Note, that some tracking methods exploit detection and segmentation techniques. For example, the segmentation often successfully serves as a preprocessing step of the tracking on a static background. Nevertheless, the segmentation and detection problems are beyond the scope of this work and we further focus on tracking methods and do not investigate them in detail.

2.1 Taxonomy of tracking methods

The most common way of tracking is repeated minimization of some criterion function $f(\mathbf{s}; \mathbf{I}, S_0)$ over the space of object states $\mathbf{s} \in S$, given image $\mathbf{I}$ and previous state(s) S_0

$$\mathbf{s}^* = \underset{\mathbf{s} \in S}{\operatorname{argmin}} f(\mathbf{s}; \mathbf{I}, S_0) = \varphi(\mathbf{I}, S_0), \tag{2.2}$$

where $\mathbf{s}^*$ is estimate of the current state of the object. Criterion $f(\mathbf{s}; \mathbf{I}, S_0)$ includes some implicit or explicit model of object possible appearances and optionally some relation to 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. There are two different approaches to tracking:

- **Optimization-based tracking** is an online optimization technique solving problem (2.2). While some approaches [41, 2, 27, 4] exhaustively search a subset of object poses S with a classifier approximating $f(\mathbf{s}; \mathbf{I}, S_0)$ for $\mathbf{s} \in S$, another approaches [37, 3, 51, 44] use a gradient optimization of a criterion approximating $f(\mathbf{s})$.
- **Regression-based tracking** methods attempt to model explicitly a relationship between observations and state $\mathbf{s}^*$ without any necessity of defining $f(\mathbf{s}; \mathbf{I}, S_0)$. They learn function $\varphi(\mathbf{I}, S_0)$ from equation (2.1) in a supervised way from synthesized training data [33, 19, 55].

In the following, the state $\mathbf{s} = (\mathbf{t}, \boldsymbol{\theta})$ is defined as a vector of object *pose parameters* $\mathbf{t}$ (e.g., position, scale, articulation) and *appearance parameters* $\boldsymbol{\theta}$ (e.g., illumination or current template). If velocities or accelerations are required, it can be computed from the pose parameters in previous frame(s).

In many applications, essential parameters are a subset of the pose parameters, the appearance parameters are often auxiliary. We therefore start with methods estimating only the pose parameters: Section 2.2 describes optimization-based tracking and Section 2.3 describes regression-based tracking methods. Most of these methods can be extended to include the appearance parameters. Such extensions are outlined in Section 2.4. Section 2.5 discusses different features for tracking. Section 2.6 introduces tracking with a known model of motion dynamics.

2.2 Optimization-based tracking

In the following two sections, the state of an object is assumed to be given exclusively by its pose $\mathbf{s} = \mathbf{t}$. It means that the appearance is considered to be fixed. Extension of these methods to the tracking of the objects with variable appearance is detailed in Section 2.4.

Optimization-based tracking methods search for the object pose by optimizing a criterion f in problem (2.2). Choice of the criterion function has a crucial influence on the robustness and accuracy. In particular, the criterion is required to

- *distinguish* the object from its background, e.g., there must be no background pattern evaluated by the same or even smaller value of the criterion,
- allow for *fast optimization*, e.g., it should have a simple gradient or low computational cost.

The most natural optimization-based approach to tracking is scanning the image for the maximum response of the criterion function, we call such optimization methods *scanning-based*. Since the scanning the whole image might be time consuming the search space is usually restricted. If the motion is small enough and the criterion function has simple gradients, a gradient optimization is usually faster and more accurate than the exhaustive search. Such methods are called *gradient-based* methods.

2.2.1 Gradient-based optimization

The most common gradient-based optimization method is the Lucas-Kanade tracker [37, 44, 3], which simplifies problem (2.2) to

$$\mathbf{t}^* = \underset{\mathbf{t}\in S(\mathbf{t}_0)}{\operatorname{argmin}} f(\mathbf{t};\mathbf{I},\mathbf{t}_0) = \underset{\mathbf{t}\in S(\mathbf{t}_0)}{\operatorname{argmin}} \|\mathbf{I}(\mathbf{t}\circ X) - \mathbf{J}(X)\|_2^2, \tag{2.3}$$

where $\mathbf{I}(\mathbf{t}\circ X)$ is a vector of intensities observed in pixels $X = \{\mathbf{x}_1 \dots \mathbf{x}_c\}$ warped by pose parameters $\mathbf{t}$ and $\mathbf{J}(X)$ is a vector of intensities observed on the object, called *template*. Object is assumed to be at the pose $\mathbf{t}^*$ for which the sum of square intensity differences (SSD) between template $\mathbf{J}(X)$ and image data $\mathbf{I}(\mathbf{t}^*\circ X)$ is minimal. Since an exhaustive search might be time consuming and search space is limited by the previous state $\mathbf{t}_0$, the Newton-Raphson gradient descent optimization method is exploited. The optimization is performed over the space of translations [37], affine transformations [44], general warpings [3] or 3D motion parameters [61]. In order to solve problem (2.3), the intensity function $\mathbf{I}(\mathbf{t}\circ X)$ is locally approximated by the first order Taylor expansion

$$\mathbf{I}(\mathbf{t}\circ X) \approx \mathbf{I}(X) + \underbrace{\left.\frac{\partial \mathbf{I}(\mathbf{t}\circ X)}{\partial \mathbf{t}}\right|_{\mathbf{t}_0}}_{\mathtt{G}} \cdot \mathbf{t} = \mathbf{I}(X) + \mathtt{G}\mathbf{t}, \tag{2.4}$$

where

$$\mathtt{G} = \frac{\partial \mathbf{I}(X)}{\partial X}\frac{\partial(\mathbf{t}\circ X)}{\partial \mathbf{t}},$$

where $\frac{\partial \mathbf{I}(X)}{\partial X} = \left[\frac{\partial \mathbf{I}(\mathbf{x}_1)}{\partial \mathbf{x}_1} \dots \frac{\partial \mathbf{I}(\mathbf{x}_c)}{\partial \mathbf{x}_c}\right]^\top$ is $(c \times 2)$ matrix of image gradients and $\frac{\partial(\mathbf{t}\circ X)}{\partial \mathbf{t}}$ is the Jacobian of the warp. For example, in the simplest case of translation the following holds:

$$\frac{\partial(\mathbf{t}\circ X)}{\partial \mathbf{t}} = \frac{\partial(\mathbf{t}+X)}{\partial \mathbf{t}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and the Jacobian is omitted.

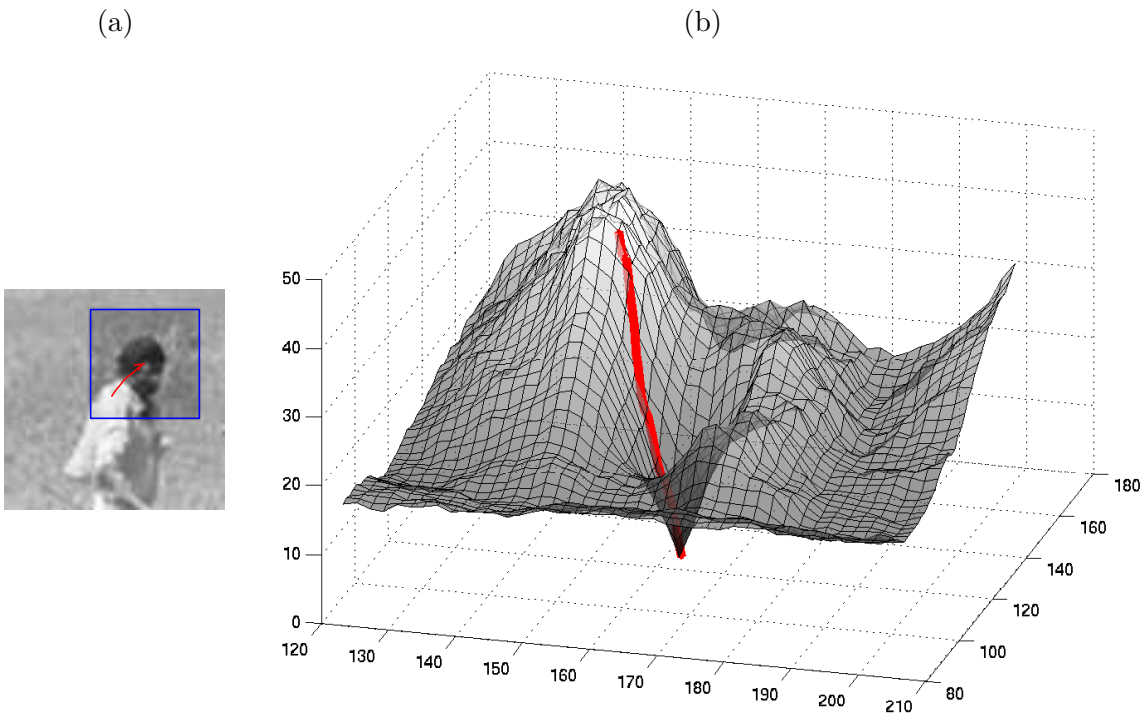


Figure 2.1: **Lucas-Kanade tracker:** (a) Template 40×40 pixels (outlined by blue line), (b) Dissimilarity function and 25 iterations (red) of a Lucas-Kanade tracker. The red path in the left image corresponds to the red optimization path.

Substituting approximation (2.4) to problem (2.3) the following closed-form solution is obtained:

$$\mathbf{t}^* \approx \operatorname*{argmin}_{\mathbf{t} \in S} \|\mathbf{I}(X) + \mathtt{G}\mathbf{t} - \mathbf{J}(X)\|_2^2 = \operatorname*{argmin}_{\mathbf{t} \in S} (\mathbf{I}(X) - \mathbf{J}(X) + \mathtt{G}\mathbf{t})^\top (\mathbf{I}(X) - \mathbf{J}(X) + \mathtt{G}\mathbf{t}) =$$

$$= \operatorname*{argmin}_{\mathbf{t} \in S} \underbrace{\Big(\mathbf{I}(X) - \mathbf{J}(X)\Big)^\top \Big(\mathbf{I}(X) - \mathbf{J}(X)\Big) + 2\Big(\mathbf{I}(X) - \mathbf{J}(X)\Big)^\top \mathtt{G}\mathbf{t} + \mathbf{t}^\top \mathtt{G}^\top \mathtt{G}\mathbf{t}}_{e(\mathbf{t})},$$

$$\frac{\partial e(\mathbf{t})}{\partial \mathbf{t}} = 0 + 2\mathtt{G}^\top \Big(\mathbf{I}(X) - \mathbf{J}(X)\Big) + 2\mathtt{G}^\top \mathtt{G}\mathbf{t}^* = 0,$$

$$\mathtt{G}^\top \mathtt{G}\mathbf{t}^* = \mathtt{G}^\top \Big(\mathbf{J}(X) - \mathbf{I}(X)\Big),$$

$$\mathbf{t}^* = (\mathtt{G}^\top \mathtt{G})^{-1} \mathtt{G}^\top \Big(\mathbf{J}(X) - \mathbf{I}(X)\Big) = \mathtt{G}^+ \Big(\mathbf{J}(X) - \mathbf{I}(X)\Big), \tag{2.5}$$

where $\mathtt{G}^+$ denotes the pseudoinverse of $\mathtt{G}$.

Since $\mathbf{I}(\mathbf{t} \circ X)$ has been locally approximated, the solution $\mathbf{t}^*$ is inexact and the process is iterated, see Figure 2.1. In general, the local linear approximation of $\mathbf{I}(\mathbf{t} \circ X)$ means local quadratic approximation of criterion $\|\mathbf{I}(\mathbf{t} \circ X) - \mathbf{J}(X)\|_2^2$ by a parabolic function $e(\mathbf{t})$ with Hessian $\mathtt{G}^\top \mathtt{G}$. Motion parameters $\mathbf{t}^*$ minimizing $e(\mathbf{t})$ determine both the steepest descent direction and the step length in each iteration. Note that $\mathbf{t}^*$ is thus estimated by a sequence of linear functions of image-template differences.

This approach has been shown computationally inexpensive and sufficiently robust, therefore many extensions have been recently proposed, see [3] for details. Second-order minimization of SSD was proposed by Benhimane and Malis [6]. Dowson and Bowden [21] experimentally show that mutual information criterion (MI) slightly outperform common SSD. Recently, Benhimane et al. [5] extended the SSD gradient optimization approach by a preprocessing stage, where a sub-template assuring theoretical convergence in the first iteration is selected. Brooks et al [13] combine SSD and MI; a learned performance profile is used to select the approximate number of pixels to process in each optimization step. As a result, the same quality with significantly less computational complexity is achieved. Comaniciu and Meer [16, 15, 17] propose a gradient based approach called meanshift, where pose $\mathbf{t}$ is iteratively estimated as a center of gravity of locally evaluated values of $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$.

2.2.2 Scanning-based optimization

Scanning based methods resembles object detection. However, instead of scanning the whole image the search is restricted on the basis of information from previous frames. Since these method are beyond the scope of this work, we provide, for the sake of completeness, only a short overview.

Nister et al. [40] proposed scanning-based method optimizing SSD criterion function. The criterion is the same as in Lucas and Kanade tracking approach [37], but the optimization method is different. Instead of the steepest descent minimization, Nister et al. evaluate values of SSD criterion only in Harris corners [30], which allows for very fast detection. Such approach avoids the problem of convergence to the local minimum, but it is more sensitive to image blur, which often reduce the number of detected Harris corners.

Felzenszwalb and Huttenlocher [24] search for the pose of an articulated structure. The criterion, to be minimized, is defined by the following equation:

$$\underbrace{\sum_{i=1}^{n} f_i(t_i)}_{\text{Similarity}} + \underbrace{\sum_{i,j \in E}^{n} g_{ij}(t_i, t_j)}_{\text{Bending energy}},$$

where t_i are components of the pose parameters and $E \subset \{1 \dots n\}^2$ is the set of modeled connections. If $E \equiv \{1 \dots n\}^2$ the articulated structure is the complete graph and exhaustive search in n-dimensional space is an NP-hard problem [12]. In order to avoid the time consuming search, the articulated structure is constrained to be a tree. Tree

structure allows dynamic programming to search through the pose space in $\mathcal{O}(h^2 \cdot n)$ time, where h is a number of discretized locations to be searched through. However, this $\mathcal{O}(h^2 \cdot n)$ algorithm is not practical in most cases, because the number of possible locations is usually huge. Restricting the bending energy (second term) to be the Mahalanobis distance, a minimization algorithm that runs in $\mathcal{O}(h' \cdot n)$ rather than $\mathcal{O}(h^2 \cdot n)$ time, is obtained.

Besides such explicitly defined criterions, the criterion function to be optimized is often represented by some kind of a learned classifier. Özuysal et al. [41] exhaustively search for maximum response of a set of Randomized Trees (RT) which stands for criterion f. Each non-leaf vertex is associated with a simple randomly initialized binary test (e.g., a comparison of two intensities) which decides whether the observed patch belongs to the left or right sub-tree. During a learning stage, each training patch is send throughout the RT by evaluating the vertex tests. Given a set of labeled training patches associated with leaves, object/background probabilities in each particular leaf are estimated. In the online tracking stage classified patches are again send throughout RT. If more than one RT is used, the probabilities from all reached leaves are combined together in order to approximate $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$. Besides the RT classifier, Adaboost [27] or SVM [2] classifiers are also often used. Grabner et al.t [27] exploit that in tracking, contrary to detection, it is sufficient to distinguish the object from its background locally. Both local background and object appearance are assumed to change, therefore the classifier is re-learned from newly collected background-foreground examples in each frame. Thus another criterion function $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$, which allows for the best local object/background separation, is optimized in each particular frame.

Performance of the scanning-based method is significantly influenced by the density of scanning grid, because a wrong choice of the density can cause either insufficient or time-consuming searching. On the other hand, gradient-based approaches suffer from all typical problems of local optimization: convergence to a local minimum, unknown number of required iterations and unknown basin of convergence.

2.3 Regression-based tracking

In order to avoid problems of the optimization based tracking, some authors [19, 33, 55] get rid of the cost function $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$ and estimate the location $\mathbf{t}$ directly from locally observed intensities. Such approach requires a learning stage, where pairs of motions $\mathbf{t}$ and corresponding observed intensities $\mathbf{I}(\mathbf{t} \circ X)$ are collected and a mapping $\varphi\colon \mathbf{I} \to \mathbf{t}$ minimizing the error on these examples is estimated, see Figure 2.2,

$$\varphi^* = \operatorname*{argmin}_{\varphi} \sum_{\mathbf{t}} \|\varphi\Big(\mathbf{I}\big(\mathbf{t} \circ X\big)\Big) - \mathbf{t}\|. \tag{2.6}$$

In the tracking stage, the learned mapping $\varphi^*(\mathbf{I})$ directly estimates motion parameters without necessity of an online optimization of any criterion function.

Noticing that Lucas-Kanade tracker [37] solves a similar optimization task in each frame, one can replace $\mathtt{G}^+$ in equation (2.5) by matrix $\mathtt{H}$ learned on a set of synthesized

examples. Equation (2.5) then transforms to the linear mapping between intensities $\mathbf{I}(X \circ \mathbf{t})$ and motion $\mathbf{t}$,

$$\mathbf{t} = \varphi\Big(\mathbf{I}(X)\Big) = \mathtt{H}\Big(\mathbf{I}(X) - \mathbf{J}(X)\Big), \tag{2.7}$$

where $\mathtt{H}$ is the matrix of some learned coefficients. In the tracking procedure, the motion parameters $\mathbf{t}$ are simply computed as a linear function $\mathtt{H}(\mathbf{I}(X) - \mathbf{J}(X))$ of the object intensities. We call such method *learned linear predictor* (LLiP). In the following, the learning of LLiP is described.

Let us suppose we are given an image template $\mathbf{J} = \mathbf{J}(X)$ and collected training pairs $(\mathbf{I}^i,\ \mathbf{t}^i)$ $(i = 1 \dots d)$ of observed intensities $\mathbf{I}^i$ and corresponding motion parameters $\mathbf{t}^i$, which aligns the object with current frame. Then *training set* is 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, LLiP coefficients minimizing the square of Euclidean error on the training set are found as follows:
First the learning task is formulated and rewritten to more convenient form:

$$\begin{aligned}
\mathtt{H}^* &= \operatorname*{argmin}_{\mathtt{H}} \sum_{i=1}^{d} \|\mathtt{H}(\mathbf{I}^i - \mathbf{J}) - \mathbf{t}^i\|_2^2 = \operatorname*{argmin}_{\mathtt{H}} \|\mathtt{HI} - \mathtt{T}\|_F^2 = \\
&= \operatorname*{argmin}_{\mathtt{H}} \operatorname{trace}(\mathtt{HI} - \mathtt{T})(\mathtt{HI} - \mathtt{T})^\top = \\
&= \operatorname*{argmin}_{\mathtt{H}} \operatorname{trace}(\mathtt{HII}^\top\mathtt{H}^\top - 2\mathtt{HIT}^\top + \mathtt{TT}^\top).
\end{aligned}$$

Next its derivative is equaled to zero:

$$\begin{aligned}
2\mathtt{H}^*\mathtt{II}^\top - 2\mathtt{TI}^\top &= 0 \\
\mathtt{H}^*\mathtt{II}^\top &= \mathtt{TI}^\top \\
\mathtt{H}^* &= \mathtt{T}\underbrace{\mathtt{I}^\top(\mathtt{II}^\top)^{-1}}_{\mathtt{I}^+} = \mathtt{TI}^+.
\end{aligned} \tag{2.8}$$

Since the method is very fast and simple it has various applications in tracking approaches. In particular, Cootes et al. [18, 19, 20] 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 [18] a linear predictor (2.7) learned by the LS method (2.8) estimates all parameters of the AAM. Since the linearity holds only for a small range of the 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. [33] for tracking of rigid objects. Unlike Cootes et al. [19], 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. [55] extended the approach

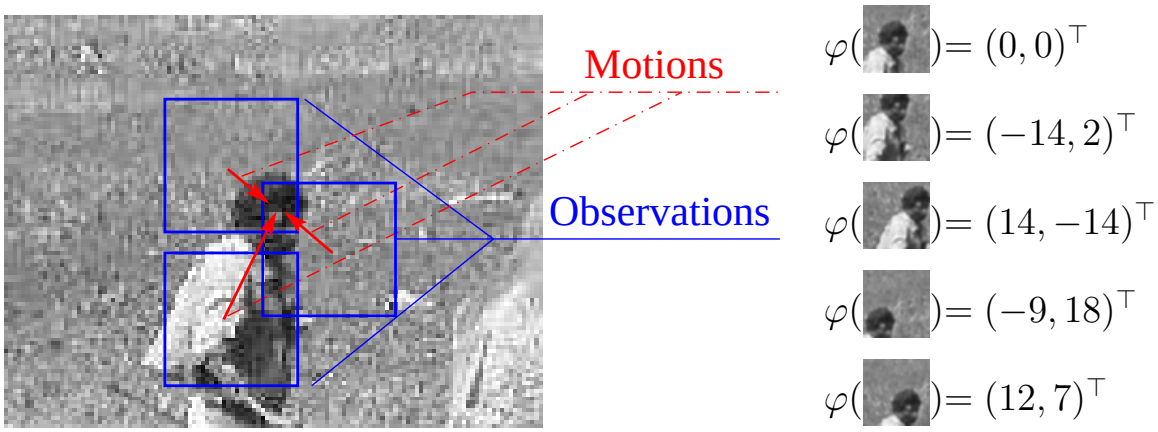


Figure 2.2: **Learning of linear mapping** between intensities and motions from a set of synthetically perturbed examples.

to the non-linear motion predictors learned by Relevance Vector Machine [50] (RVM). Agarwal and Triggs [1] used RVM to learn the linear and non-linear mapping for tracking of 3D human poses from silhouettes.

Drucker et al.[22] search for the regression function that has at most ϵ deviation from the actually obtained poses $\mathbf{t}^i$. It means, that they do not care about errors as long as they are less than ϵ, but any larger deviations are inacceptable. Especially if the function has the following linear form $\mathbf{h}^\top \mathbf{I}^i$, the learning is formulated as the following quadratic programming problem:

$$\begin{aligned} \mathbf{h}^* &= \operatorname*{argmin}_{\mathbf{h}} \|\mathbf{h}\|^2 \\ &\text{subject to: } \mathbf{t}^i - \mathbf{h}^\top \mathbf{I}^i \leq \epsilon \\ &\qquad\qquad \mathbf{h}^\top \mathbf{I}^i - \mathbf{t}^i \leq \epsilon \end{aligned} \tag{2.9}$$

The method, called Support Vector Regression Machine, is similar to the Support Vector Machine [52] and allows also the extension for nonlinear kernels. Detailed description may be found in [47].

Zhou et al. [59] proposed greedy learning for additive regression function:

$$\varphi(\mathbf{I}(X)) = \sum_{i=1}^{c} \varphi_i(\mathbf{I}(X)), \tag{2.10}$$

where $\mathbf{I}(X)$ are some image features. The learning consists of c-steps, within each of them *weak regressor* $\varphi_i(\mathbf{I}(X))$ minimizing the training error is estimated.

Let us emphasize two special cases:

- If the weak regressor is linear function, the additive regressor $\varphi(\mathbf{I}(X))$ simplifies to linear mapping (2.7) and the problem has closed-form solution (2.8).

- If the weak regressor has the form of piece-wise constant function with L discontinuities, the additive regressor (2.10) corresponds to the L-nary regression tree with the same decision stumps in each level.

Zhou et al. [59] use the weak regressor formed of a linear combination of binary functions. They constrained the coefficients of the linear combination to have the same absolute values. Such constraint allows to find a closed-form solution in each of c learning greedy steps. Bissacco et al. [7] extended the learning technique for the L-nary regression trees and showed that it outperforms [59].

Since we mainly contribute to these methods, we briefly summarize our contribution also here. Rather than proposing a special learning procedure for a special type of the regression function, we present an optimal way to concatenate different regression functions into a sequence. Our main idea follows the fact that the intensities (features) of pixels located close to the searched pose are usually more convenient for precise pose estimation than some other intensities.

Let us suppose, we are given a class of regression functions. Each of the functions operates within a different range of poses and has different precisions and computational complexities. Given predefined range and precision, we want to design a regression-based tracking method. The simplest thing one can do is to select a function with sufficient range and precision. Of course, such function need not even exists and if so, it could have a very high computational complexity. The other possibility is to select a sequence functions. The first function provides a coarse estimate of the pose. The following function is consequently allowed to operate within a smaller range of poses. If it is true, that the intensities of pixels located close to the searched pose are more convenient for the pose estimation, such function naturally achieves a higher precision with a reasonable complexity. Similarly, another ancestors again refines from the precision of previously estimated poses.

In continuation of that, we define learning as a searching for a sequence with the lowest computational complexity subject to predefined precision and range. We learn the optimal sequence of regression functions, which is in general superior to a single function. Since LLiPs are easy to operate and allow for good precision on low computational complexity, we demonstrate the method on the Sequences of LLiPs (SLLiP). Note, that the linear predictor can be naturally extended to an arbitrary linear combination of non-linear mappings by data lifting, therefore the linearity is not too much restricting condition.

2.4 Tracking objects with variable appearance

If we had a perfect geometrical model of the object and its environment, every observed camera image would be explainable only by pose parameters of objects in the

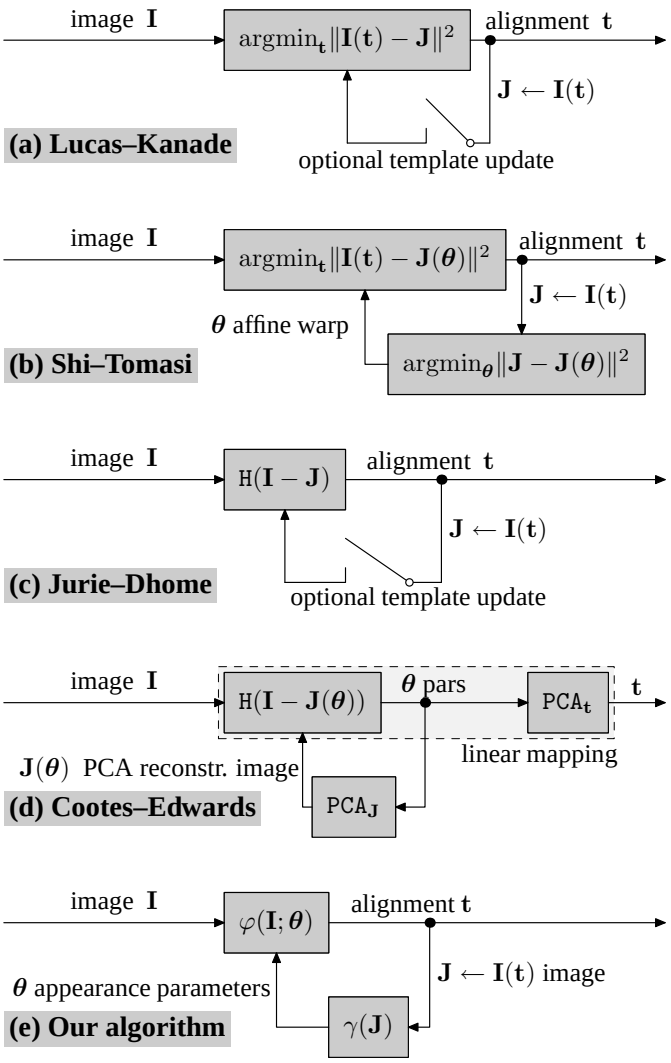


Figure 2.3: **Tracking methods** compensating changes of the object appearance.

scene. However, such model would probably require immense number of parameters and its estimation would be intractable, therefore only the most essential pose parameters are explicitly considered. The omitted degrees of freedom cause changes in the object appearance, that cannot be compensated by these parameters, which often results in the loss-of-lock. Therefore, we further introduce appearance parameters $\boldsymbol{\theta}$, which compensate these appearance changes implicitly. It is even not clear in advance, which parameters belong to the pose and which to the appearance. Usually the appearance parameters influence template $J(\boldsymbol{\theta})$ intensities and pose parameters influence positions of object pixels $\mathbf{t} \circ X$, but it is not the rule. Usually, pose is assumed to change faster

than object appearance[1], therefore the pose parameters are estimated the first — the appearance parameters are then estimated from the image aligned by the pose parameters.

Let us first assume that no training set is available in advance. In such a case we have usually no prior information about the possible changes of the object appearance. Therefore the template is either not updated at all or updated from the last known position in terms of partial or whole template replacement by the aligned image, see Figure 2.3a. Since the whole template replacement often results in drifting and consequently to the loss-of-lock as well, Matthews et al. [39], allow only slow changes of objects appearance. Dowson and Bowden [21] continuously watch a dissimilarity measure, once the model appearance is too different from the observed object, the new template is initialized and the old one is saved for future usage. Shi and Tomasi [44] propose to use Lucas-Kanade translation tracker [37], with the template optionally updated by an affine transformation estimated again by the gradient optimization, see Figure 2.3b. Although the affine transformation can be optimized at once with the translation as the 6-dimensional gradient optimization problem, they did not do it arguing that their joint optimization provides worse results than successive optimization of the translation and the affine warp.

There are also methods [8, 19, 29] exploiting machine learning techniques. They allow only a limited parameter-driven changes of the model appearance. Black and Jepson [8], similarly to the Lucas-Kanade tracker [37], also minimize the SSD function. They model possible object appearances by a linear combination of eigen-templates learned by the PCA method [25] from a set of training images. During the tracking stage, the object is aligned and the closest template $\mathbf{J}(X)$ to the currently observed appearance is reconstructed as a linear combination $\mathbf{J}(\boldsymbol{\theta}) = \sum_i \Theta_i \mathbf{J}^i$ of eigen-templates $\mathbf{J}^i$.

Scanning-based methods like [2, 27] adaptively re-learn classifier on currently collected examples of both object and background appearances. Yuan et al. [56] recently proposed a parameter sensitive classifier which searches for the object pose. Given a training set of possible object/background appearances the classifier controlled by a vector of appearance parameters $\boldsymbol{\theta}$ is learned. During the detection/tracking stage the current appearance parameters of the searched object are unknown, therefore both the pose and parameter space is searched through, which is hardly usable for a real-time application.

Cootes et al. [18, 19] proposed a paradigm where both the motion and appearance are projected by the PCA to the lower dimensional space of some parameters. Given an image the learned linear mapping with coefficients $\mathtt{H}$ estimates parameters $\boldsymbol{\theta}$. The parameters are back-projected by the PCA, in order to simultaneously estimates the motion and update the template, see Figure 2.3d. While in [19] the parameters are optimized by the steepest gradient descent, in [18] a linear predictor is used. Note, that in our work, we generalize the tracking paradigm as shown in Figure 2.3e.

[1] A particular counter-example is a sudden change of an object illumination (e.g., switching off the light). Nonetheless, we are interested in the scenes, where both pose and appearance change smoothly and the iterative estimation is affordable.

2.5 Tracking in feature space

Instead of object intensities $I(X)$ an arbitrary set of features (e.g., histogram) could be used. A set of used features has crucial influence on the tracker robustness and accuracy. Some features are very accurate (e.g., intensities or Haar features [28]), some have large basin of attraction (e.g., mean), others are invariant to projective transformation (e.g., moment invariants) or to an arbitrary permutation of pixels (e.g., color histogram). In general, the more independent features used, the higher the probability of successful tracking, but the higher the computational complexity. Since the time consuming algorithms do not allow for real-time tracking, only a convenient subset of features, which allows the object to be well locally distinguished from its background, must be used.

The object is distinguished from its background by a criterion f, the discriminability of which is proportional to the difference between object and background probabilities. Since the feature selection is a combinatorial problem, authors usually propose a heuristic maximizing the difference or ratio of the object/background probabilities.

Grabner and Bishof [27] scan the image by Ada-boost classifier. Object and background appearances are assumed to change, therefore a set of used features is re-optimized and the Ada-boost is re-built in every frame. Collins et al. [14] use mean-shift [16] tracking algorithm with tuning of features in order to maximize their discriminability. Each feature is a weighted linear combination of RGB values. The weights are the subject of the tuning. Zhou et al. [59] proposed a greedy least-square feature selection algorithm for regression functions. Jepson et al. [32] use EM-algorithm to adapt the model appearance parameteres over time. Appearance is assumed to be generated by three independent clusters of parameters - stable, transient and outlier

2.6 Modeling motion dynamics

Until now, we assumed that the state of an object is not far away from its last known state, which allowed for the gradient optimization or regression based tracking methods. We explicitly assumed that motion prior probability is uniformly distributed over the range within which the method is able to operate, e.g., basin of attraction of gradient optimization methods or range of regression methods. Prior was assumed to be zero out of this range prior.

If motions are too fast then inter-frame distances are too long and the range of tracking methods is insufficient. If the object state is well predictable just from dynamics, a motion model could be used for initial guess. However, motion model is often insufficient. The range could be optionally enlarged by multiple initialization, but it costs time and consequently affects tracking accuracy.

If no motion model is available, no initial guess can be computed and the range have to be estimated adhoc. Therefore a motion model, which allows prior estimation is often used. The prior distribution determines both the initial guess and the range. Common tracking approaches [43, 57, 60] model the object dynamics by a linear motion model and use the Kalman filter [34] for the range and the initial guess estimation.

Besides of such explicit motion dynamics modeling there is also another alternative, which models time continuity constraints implicitly. Instead of computing the prior, the feature vector is simply extended by visual features computed on previous image(s) [58] or on difference image [7].

Instead of making hard decision about the object presence in each frame, it is also possible to approximate probability distribution of the essential parameters by a multi-modal distribution such as the mixture of Gaussians. These distributions are the state of an object and the essential parameters are estimated in each frame as for example maxima of these distributions. Such representation is robust to partial occlusions or noise and avoids falling into a local minimum.

One of the most common approach is the condensation technique [31], where state space of the essential parameters is in each frame sampled according to their probability distributions estimated from previous frames. The approximation is frame to frame updated from the samples of the distribution and from a motion model. Another probabilistic approach was presented by Sminchisescu et al. [46, 45], where Bayesian mixture of experts (BME), learned by EM-algorithm [50], directly estimates probability of motion parameters. In contrast to the other approaches, BME can deal with multi-modal distribution, which often arises in a case of 3D tracking due to ambiguities [1]. Patras and Hancock [42] extended the method to incorporate also relevance of the current observation.

3 Contribution of the thesis

Our contributions is related to the regression-based tracking methods like [19, 33, 55, 58], where motion is directly estimated from the observed intensities by a learned mapping called *predictor*. Specifically, our contributions are the following:

1. **Optimal concatenation of the predictors**. The more complex the predictor, the more precise pose estimation is achievable. Increasing the complexity, however, often suffers from diminishing returns and very complex predictors are prone to overfitting. We follow a simple assumption, that it is at least usually easier to estimate the actual state, if the predictor is initialized in its close neighbourhood. Accepting this assumption, it is better to exploit a less complex predictor for coarse state estimation and use the newly obtained state for the initialization of another predictor. The coarse estimate of the state allows to the following regression function operate within a smaller range of the states and achieve a higher precision with a reasonable complexity. Hence, instead of the learning a very complex predictor, we use a sequence of rather simple predictors concatenated in the way that each of the predictor compensates only errors of its predecessor and thus refines from the previous predictions. While single predictor operates on a fixed set of intensities (features), the sequential predictor allows for higher precision because the set of the intensities is estimated successively as the actual state accuracy increases. We learn the optimal sequence of predictors, which is in general superior to a single predictor.

2. **Explicit minimization of the computational complexity.** In contrast to the state-of-the-art approaches, where learning of the predictor is formulated as a minimization of motion estimation error on a given training set, we formulated the learning as a minimization of the tracking time given user predefined precision and robustness. We proposed learning based on dynamic programming, which estimates the optimal sequence of *linear* predictors. Note, that the linear predictor can be naturally extended to an arbitrary mapping formed as a linear combination of kernel functions by data lifting, therefore the linearity is not too much restricting condition.

3. **Anytime learning.** Since our learning procedure of the optimal sequential predictor might be time consuming, an *anytime* learning algorithm is also proposed. Anytime learning delivers, after a short initialization period, some sequential predictor with defined precision and robustness, but higher computational complexity.

While the tracking is allowed to start with this predictor, the learning continues in a separate background thread and continuously improves the predictor.

4. **Tracking of objects with variable appearance**. We extended the proposed method for objects, which exhibit nontrivial appearance changes caused for example by non-rigid deformations or variable illumination. We introduce some abstract appearance parameters, which encodes actual object appearance. These parameters tune the predictor to estimate the pose of the object with actual appearance. Proposed tracking system consists of the *tracker* - a parameter-tunable linear predictor and *appearance encoder* - a single predictor, both connected to the feedback scheme. Tracking is defined as an image alignment estimated by the tracker followed by an appearance encoding, which tunes the tracker for the current object appearance. Simultaneous learning of both predictors is shown to be a bilinear fitting problem, the minimum of which is searched by a line-search algorithm. In contrast to Cootes et al. [18], who projects both the motion and appearance parameters by the PCA to the space of lower dimensional parameters and use the learned linear predictor to predict these lower dimensional parameters, our appearance parameters has no explicit meaning, because they are automatically determined during the unsupervised learning stage in order to minimize the training error.

5. **Adaptive real-time optimization of range and accuracy**. Performance of learned sequential predictors is characterized by range, i.e., region of predictable motions, accuracy and speed. Any sub-sequence of the sequential predictor can be viewed as another sequential predictor with some range, accuracy and speed. In general the subsequent linear predictor is faster. If the subsequence does not include some final predictors, lower accuracy is achieved. If the predictors from the beginning are ignored, the subsequence has a smaller range. Range and accuracy are dynamically adjusted in order to achieve the most accurate tracking subject to a predefined maximal probability of tracker failure (loss-of-lock). In tracking a virtuous circle emerges: minimal state covariance in the following frame estimated by the Kalman filter determines both the length of the sequence of predictors to be used in the current frame and the initial range in the following frame. Since the initial range is typically smaller the higher accuracy is achieved. Such process repeats until the highest possible accuracy is reached.

6. **Large set of tracking data with ground truth.** In order to verify the tracker we also created publicly-available sequences[1] with approximately 12000 ground-truthed frames.

[1] `http://cmp.felk.cvut.cz/~zimmerk/linTrack`

3.1 Thesis outline

The rest of the thesis is organised as follows:

- Chapter 4 describes the optimal sequential predictor. In particular:
  - Section 4.1 introduces predictors, sequential predictors and proves some of their elementary properties.
  - Section 4.2 defines the optimal sequential predictor and describes efficient learning, which does not allow restriction of the learning time.
  - Section 4.3 proposes not as efficient but anytime learning algorithm, which, after a short initialization procedure, delivers a sub-optimal sequential predictor with defined precision.
- Chapter 5 describes RANSAC-based tracking of the objects modeled by more than one predictor.
  - Section 5.1 proposes an online optimization of the locations, where the predictors are attached to the model.
  - Section 5.2 optimizes ratio between the number of RANSAC iterations and the number of used predictors subject to a user predefined frame-rate.
- Chapter 6 experimentally verifies the tracker consisting of a few optimal sequential predictors on the publicly-available ground truthed sequences.
- Chapter 7 extends the tracking approach for objects with variable appearance.
- Chapter 8 introduces adaptive real-time optimization of range and accuracy.

4 Learning to estimate motion

4.1 Predictors, properties and terminology

In this chapter, we define the following terms predictor, sequential predictor and the optimal sequential predictor and show how to learn it. Let us suppose that the object state is given by object pose parameters (e.g., position)[1]. In each frame we update the object state by current motion parameters estimated by the predictor from a subset of object pixels. The subset of the object pixels is called the *support set* $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_c\}$. The intensities observed on the support set X are collected in the *observation vector* $\mathbf{I}(X)$.

Ideally, a predictor would use a support set minimizing the prediction error. However, the problem has combinatorial complexity and we discuss it later in Section 4.4; let us assume for now that a support set has been selected.

We denote $(\mathbf{t} \circ X)$ the support set transformed by a motion with parameters $\mathbf{t}$. For example, if the considered motion is a 2D translation, then $(\mathbf{t} \circ X) = (X + \mathbf{t}) = \{(\mathbf{x}_1 + \mathbf{t}), \ldots, (\mathbf{x}_c + \mathbf{t})\}$. There is a mapping from parameters $\mathbf{t}$ to observations $\mathbf{I}(\mathbf{t} \circ X)$, which is usually not invertible. We therefore search for a mapping approximating a set of motions $\mathbf{t}$ which could have generated the observation $\mathbf{I}(\mathbf{t} \circ X)$. This mapping, called a *regressor*, assigns a p-vector of motion parameters to a c-vector of observation. Regressors $\hat{\varphi}$ are completly characterized by their complexity, range and uncertainty region:

Definition 1. Complexity $c(\hat{\varphi})$ *of regressor* $\hat{\varphi}$ *is a value proportional to the computational cost of the regressor. It is equal to the size of a support set for linear regressor.*

Definition 2. Range $R(\hat{\varphi})$ *of the regressor* $\hat{\varphi}$ *is a set of motion parameters*[2]*.*

Definition 3. Uncertainty region *of the regressor* $\hat{\varphi}$ *is the smallest region*

$$\Lambda(\hat{\varphi}) = \left\{ \Delta\mathbf{t} \mid \Delta\mathbf{t} = \mathbf{t} \circ \hat{\varphi}\Big(\mathbf{I}(\mathbf{t} \circ X)\Big),\ \forall \mathbf{t} \in R(\hat{\varphi}) \right\}. \tag{4.1}$$

[1] In general, object could be represented by more than one predictor. Such representation allows for robust object pose estimation by RANSAC and we discuss this extension in Section 5. For now, let us suppose that only one predictor is associated with the object.

[2] Note, that this is not the range in usuall mathematical meaning

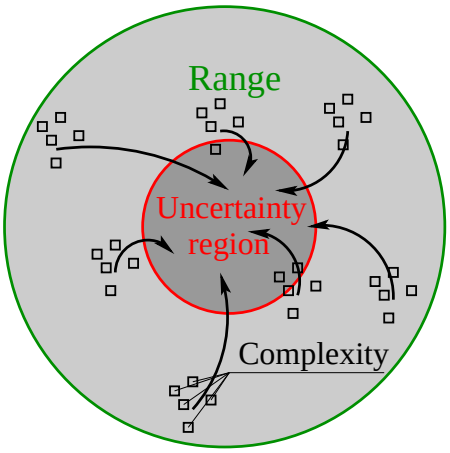


Figure 4.1: **Definitions:** Range, accuracy and complexity. An example with 2D translations.

The uncertainty region is the smallest region within which all the prediction errors from the range $R(\hat{\varphi})$ lie, see Figure 4.1.

In order to simplify the learning procedure we select only a class (e.g., circles or squares) $\{\Lambda_\lambda\}_{\lambda\in\mathbb{R}}$ parameterizable by one scalar parameter $\lambda \in \mathbb{R}$ such that

$$\forall\, \lambda_1, \lambda_2 \in \mathbb{R} :\ \lambda_1 < \lambda_2 \ \Rightarrow\ \Lambda_{\lambda_1} \subset \Lambda_{\lambda_2}\,. \tag{4.2}$$

Ranges R_r are selected from the same class of regions and parameterized by the same parameter $r \in \mathbb{R}$. According to equation (4.2), parameter λ (and r) are proportional to the area of the region, therefore we sometimes refer to it as an area of the region and use notation $\lambda(\hat{\varphi})$ (and $r(\hat{\varphi})$) to denote corresponding values of $\Lambda(\hat{\varphi})$ and $R(\hat{\varphi})$ respectively. An extension to regions parameterizable by more than one parameter is discussed later.

4.1.1 Predictors

Definition 4. Predictor $\varphi(c, r, \lambda)$ *is an ordered* 4*-tuple* $(\hat{\varphi}, X, R_r, \Lambda_\lambda)$*, where* X *is the support set,* $c \approx |X|$ *is complexity (for linear case* $|X| = c$*),* $\hat{\varphi}$ *is the regressor,* R_r *is range and* Λ_λ *is the uncertainty region.*

Even though we defined the predictor as 4-tuple, we parameterize all predictors by the three parameters: complexity c, range r and uncertainty region λ, the regressor $\hat{\varphi}$ is omitted. It actually says, that two predictors with the same (c, r, λ) and different regressors $\hat{\varphi}_1, \hat{\varphi}_2$ are equivalent.

In order to assure that increase in complexity does not reduce the prediction abilities, we further restrict ourselves to the class of support sets satisfying that every support set contains all support sets with lower complexity. This is assured by the successive support set construction algorithm described in Section 4.4. This is, however, still insufficient

assumption. It is also required that regressors must be able to ignore values of some input pixels. In the following definition we define the class of such regressors.

Definition 5. *Let denote $\mathcal{F}^c$ some class of regressors $\hat{\varphi}\colon \mathbb{R}^c \to \mathbb{R}^p$ with the same support set X. Let $\mathcal{F} = \left\{\mathcal{F}^1 \cup \mathcal{F}^2 \cup \cdots \cup \mathcal{F}^c \ldots\right\}$. $\mathcal{F}$ is called* domain independent class *if:*

$$\forall\, c\, \forall\, \hat{\varphi}_1 \in \mathcal{F}^c\ \exists \hat{\varphi}_2 \in \mathcal{F}^{c+1}\ \textit{such that}\ \forall\, \mathbf{I} \in \mathbb{R}^c\, \forall\, u \in \mathbb{R}\ \ \hat{\varphi}_2([\mathbf{I}, u]) = \hat{\varphi}_1(\mathbf{I}).$$

This is for example satisfied for the following class of regressors

$$\mathcal{F}^1 = \{a_1 \cdot x_1 \,|\, a_1 \in \mathbb{R}\}\,,$$
$$\mathcal{F}^2 = \{a_1 \cdot x_1 + a_2 \cdot x_2 \,|\, a_1, a_2 \in \mathbb{R}\}\,,$$
$$\mathcal{F}^c = \{\sum_{i=1}^{c} a_i \cdot x_i \,|\, a_i \in \mathbb{R}, i = 1 \ldots c\}$$

parameterized by coefficients $a_1, a_2 \in \mathbb{R}$, because it can ignore an arbitrary input x_1, x_2 by setting the corresponding coefficient to zero. On the contrary, the class following is not domain independent class

$$\mathcal{F}^1 = \{a \cdot x_1 \,|\, a \in \mathbb{R}\}\,,$$
$$\mathcal{F}^2 = \{a \cdot (x_1 + x_2) \,|\, a \in \mathbb{R}\}\,,$$
$$\mathcal{F}^c = \{a \cdot \sum_{i=1}^{c} x_i \,|\, a \in \mathbb{R}\}$$

parameterized only by one coefficient $a \in \mathbb{R}$. In general, the class of polynomials of an arbitrary order parameterized by all of their coefficients is an example of the domain independent class.

Note that not all good properties of the predictors are simultaneously achievable. It is clear that there is no ideal predictor which would simultaneously have (very) low complexity, (very) large range and (very) small error. We denote the achievable subset of predictors in (c, r, λ) space by ω, see Figure 4.3 for an example. Predictors lying on the border of ω are very important, because it will be shown later that optimal sequential predictors are exclusively formed from these predictors.

Definition 6. λ-minimal predictors $\varphi^+(c, r)$ *are predictors having the minimal achievable λ for a given range r and complexity c.*

$$\varphi^+(c, r) \in \operatorname*{argmin}_{\varphi} \{\lambda \mid \varphi(c, r, \lambda) \in \omega\}\,. \tag{4.3}$$

Note that λ-minimal predictors are the predictors lying on the boundary of ω, see Figure 4.3c.

Simple consequence of Definition 5 is, that more complex predictors can do everything that the simpler can. This is shown in two following propositions which summarize properties of λ-minimal predictors. The propositions are not crucial for the understanding of the learning procedure however, we later use them to prove that the learning algorithm can be simplified.

Proposition 1 *The uncertainty region of a λ-minimal predictor is a nonincreasing function of the complexity c.*

Proof: We prove that the uncertainty region cannot increase with complexity (see for example Figure 4.3a). Let us suppose, we are given two λ-minimal predictors with regressors $\hat{\varphi}_1^+ \in \mathcal{F}^c, \hat{\varphi}_2^+ \in \mathcal{F}^{c+1}$. Since λ-minimal predictors are predictors with minimum uncertainty region, their regressors have to satisfy:

$$\hat{\varphi}_1^+ \in \operatorname*{argmin}_{\hat{\varphi}_1 \in \mathcal{F}^c} \lambda(\hat{\varphi}_1)\,, \tag{4.4}$$

$$\hat{\varphi}_2^+ \in \operatorname*{argmin}_{\hat{\varphi}_2 \in \mathcal{F}^{c+1}} \lambda(\hat{\varphi}_2)\,. \tag{4.5}$$

We prove that regressor with higher complexity $\hat{\varphi}_2^+$ has the uncertainty region smaller or equal to the uncertainty region of regressor with smaller complexity $\hat{\varphi}_1^+$, i.e., $\lambda(\hat{\varphi}_1^+) \geq \lambda(\hat{\varphi}_2^+)$. This fact is shown by contradiction, therefore we assume that

$$\lambda(\hat{\varphi}_1^+) < \lambda(\hat{\varphi}_2^+). \tag{4.6}$$

Since we know that $\hat{\varphi}_1^+ \in \mathcal{F}^c$, then according to the Definition 5, there exists some $\hat{\varphi}_3^+ \in \mathcal{F}^{c+1}$ such that $\forall\, \mathbf{I} \in \mathbb{R}^c\ \forall\, u \in \mathbb{R}\ \ \hat{\varphi}_3^+(\mathbf{I}) = \hat{\varphi}_1^*([\mathbf{I}, u])$. It also implies that $\lambda(\hat{\varphi}_3^+) = \lambda(\hat{\varphi}_1^+)$. Hence according to the assumed inequality (4.6)

$$\lambda(\hat{\varphi}_1^+) = \lambda(\hat{\varphi}_3^+) < \lambda(\hat{\varphi}_2^+).$$

This leads us to the contradiction, because there exists regressor $\hat{\varphi}_3^+$, which has smaller uncertainty region than $\hat{\varphi}_2^+$ and therefore $\hat{\varphi}_2^+$ could not be the optimal solution of problem (4.5) and consequently $\hat{\varphi}_2^+$ could not be the regressor of any λ-minimal predictor with complexity $c+1$. ■

Note, that Proposition 1 is valid for arbitrary predictors which are optimal with respect to some criterion. For example, if we had been dealing with predictors minimizing mean Euclidean prediction error, say e, then the minimal e would have been a nonincreasing function of the complexity, as well.

Proposition 2 *Uncertainty region of λ-minimal predictor is a nondecreasing function of the range.*

Proof: Given two λ-minimal predictors:

$$\varphi_1^+ = \varphi^+(c, r_1) = \underset{\varphi}{\operatorname{argmin}} \{\lambda \mid \varphi(c, r_1, \lambda) \in \omega\},$$

$$\varphi_2^+ = \varphi^+(c, r_2) = \underset{\varphi}{\operatorname{argmin}} \{\lambda \mid \varphi(c, r_2, \lambda) \in \omega\},$$

such that $r_2 > r_1$, we prove that the predictor with larger range r_2 has larger or at most the same uncertainty region as a predictor with smaller range r_1,
ie $r_2 > r_1 \Rightarrow \lambda(\varphi_2^+) \geq \lambda(\varphi_1^+)$.

The implication is proved by contradiction. We assume $r_2 > r_1$ and $\lambda(\varphi_2^+) < \lambda(\varphi_1^+)$. Since $R_{r_1} \subset R_{r_2}$, the predictor φ_2 can also predict every motion from range r_1. Consequently, we can define a new predictor $\varphi_1' = (\hat{\varphi}_2^+, X, R_{r_1}, \Lambda_{\lambda_2})$ operating on range r_1 with

$$\lambda(\varphi_1') = \lambda(\varphi_2^+) < \lambda(\varphi_1^+). \tag{4.7}$$

This is in contradiction with the fact that φ_1^+ is λ-minimal predictor, because we have just found another predictor φ_1', which has a smaller uncertainty region. ■

4.1.2 Sequential predictor

It directly follows from the Proposition 1 that the higher is the complexity the better is the prediction. However, increasing the complexity has diminishing returns, see for example Figure 4.3. For large ranges, it is usually very difficult to achieve a good prediction even with the complexity corresponding to the cardinality of the complete template. In order to overcome this limitation we develop a *sequential predictor* $\Phi = (\varphi_1 \dots \varphi_m)$, see Figure 4.2, which estimates vector of motion parameter $\mathbf{t}$ in m steps as follows:

$$\mathbf{t}_1 = \hat{\varphi}_1\big(\mathbf{I}(X_1)\big),\ \mathbf{t}_2 = \hat{\varphi}_2\big(\mathbf{I}(\mathbf{t}_1 \circ X_2)\big),$$

$$\mathbf{t}_3 = \hat{\varphi}_3\big(\mathbf{I}(\mathbf{t}_2 \circ \mathbf{t}_1 \circ X_3)\big),\ \dots,\ \mathbf{t}_m = \hat{\varphi}_m\Big(\mathbf{I}\big(\big(\bigcirc_{i=1}^{m-1} \mathbf{t}_i\big) \circ X_m\big)\Big),$$

$$\mathbf{t} = \bigcirc_{i=1}^{m} \mathbf{t}_i.$$

The first vector of motion parameters $\mathbf{t}_1$ is estimated directly by predictor φ_1 from the intensities observed in support set X_1 . This predictor has a known uncertainty region λ_1 within which all its predictions lie. Therefore the successive predictor φ_2 is learned only on the range $r_2 \approx \lambda_1$ corresponding to this uncertainty region, which is usually significantly smaller than range r_1 of the first predictor. The smaller range yields the smaller uncertainty region. The advantage is that the predictors in the sequence are more and more specific, which consequently allows the prediction to be very accurate for reasonably textured regions. It is experimentally shown that the sequential predictor,

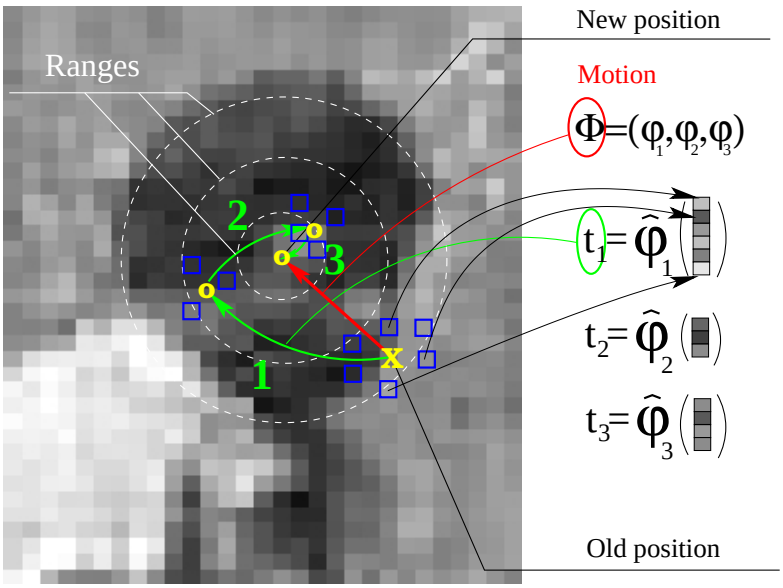


Figure 4.2: **Sequential predictor** $\Phi = (\varphi_1 \dots \varphi_m)$. Vector of motion parameters $\mathbf{t}$ (denoted by red arrow) is estimated successively by m different predictors $\varphi_1 \dots \varphi_m$. Both the length of the sequence and particular predictors are subject of the optimization.

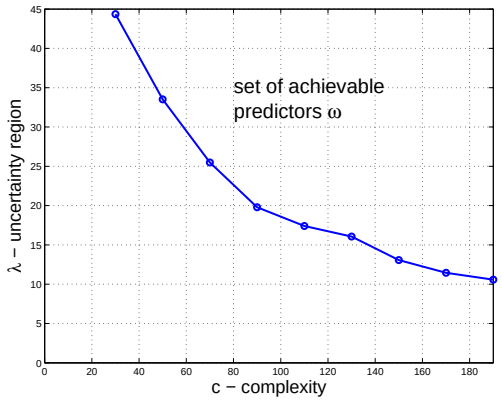
which is superior to a single predictor, yields significantly lower complexity and a higher precision.

Obviously, we consider only those sequential predictors which satisfy $R(\hat{\varphi}_{i+1}) \supseteq \Lambda(\hat{\varphi}_i),\ i = 1 \dots m-1$. The range of each particular predictor must accommodate the uncertainty region of its predecessor at least. The uncertainty region of the sequential predictor is understood as the uncertainty region of the last predictor and its range as the range of the first predictor.

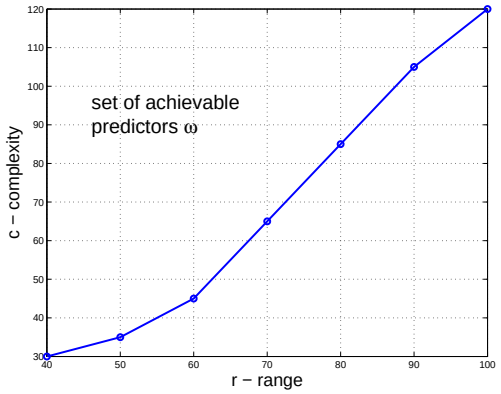
Definition 7. Sequential predictor *is an m-tuple* $\Phi = (\varphi_1(c_1, r_1, \lambda_1), \dots, \varphi_m(c_m, r_m, \lambda_m))$ *of predictors* $\varphi_i \in \omega$ *such that* $R(r_{i+1}) \supseteq \Lambda(\lambda_i), i = 1 \dots m-1$. *Uncertainty region of the sequential predictor* Φ *is* λ_m *and its range is* r_1.

4.2 Learning optimal sequential predictors

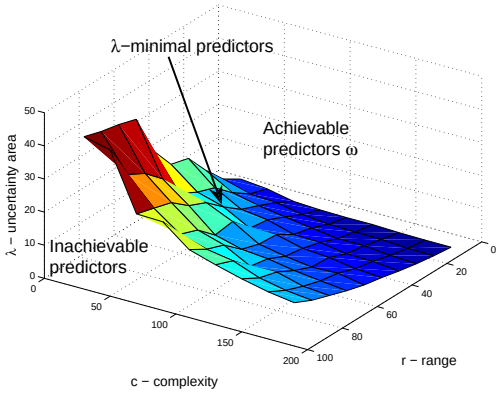
In the previous section, we defined the predictor and the sequential predictor. In this section, we first define the optimal sequential predictor and show that it can be created exclusively from the λ-minimal predictors (Definition 6). Section 4.2.1 describes learning of the λ-minimal predictor, given a training set. In Section 4.2.2 a set of λ-minimal



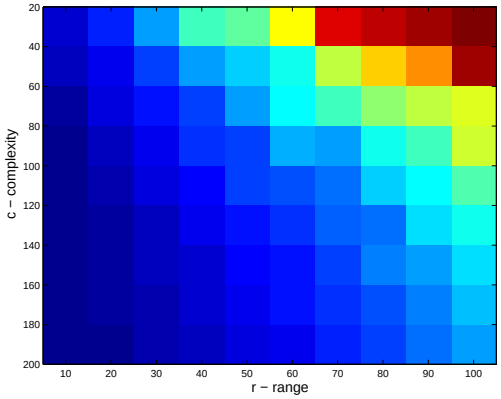
(a) λ-lower bound as a function of complexity



(b) Complexity as a function of range.



(c) Set of achievable predictors in (c, r, λ)-space.



(d) Rolled-out λ-lower bound.

Figure 4.3: **Set of achievable predictors.** Color codes the size of the uncertainty region.

predictors with different complexities a ranges is learned; selection of an optimal sequence of the predictors from the set is formulated as a search for the cheapest path in a graph.

Definition 8. *The* optimal sequential predictor *is*

$$\Phi^* = \underset{\Phi\in\Omega, m\in\mathbb{N}^+}{\operatorname{argmin}} \left\{ \sum_{i=1}^{m} c_i \mid r_1 \geq r_0, \lambda_m \leq \lambda_0 \right\}, \tag{4.8}$$

where Ω is the set of all sequential predictors, r_0 is predefined range and λ_0 is predefined uncertainty region and $\mathbb{N}^+$ is the set of positive integral numbers.

Proposition 3 *There is at least one optimal sequential predictor created exclusively from the λ-minimal predictors.*

Proof: The proposition is proved by showing that any non-λ-minimal predictor can be replaced by a λ-minimal predictor of the same complexity. It is then clear, for every non λ-minimal predictor φ_i from the optimal sequence, there exists a λ-minimal predictor φ_i^+ with the same complexity, such that the following holds:

$$\Lambda(\varphi_i^+) \subset \Lambda(\varphi_i) \subset R(\varphi_i) \subset R(\varphi_i^+).$$

Therefore φ_i^+ can replace φ_i. ■

Therefore, we consider only predictors with the smallest uncertainty region λ, i.e., predictors lying on the λ-lower bound defined by (4.3). In that way the λ-lower bound, 2D manifold in (c, r, λ)-space (Figure 4.3c), is rolled out to the (c, r)-space (Figure 4.3d). Task (4.8) reduces to

$$\Phi^* = \operatorname*{argmin}_{\Phi \in \Omega^+} \sum_{i=1}^{m} c_i, \tag{4.9}$$

where Ω^+ is the set of sequential predictors created only by the λ-minimal predictors, equation (4.3).

The procedure of linear λ-minimal predictor learning is carried out by linear programming in Section 4.2.1. In Section 4.2.2, a sequence of the λ-minimal predictors creating the optimal sequential predictor Φ^*, equation (4.9), is selected from a set of learned λ-minimal predictors. The problem is formulated as searching of the cheapest path in a graph.

4.2.1 Learning linear λ-minimal predictor $\hat{\varphi}+$

Linear predictor

In order to estimate a predictor satisfying equation (4.3), the regressor $\hat{\varphi}$ needs to be specified in detail. We restrict ourselves to Learned *Linear* Predictors:

Definition 9. Learned Linear Predictor *(LLiP) is predictors with linear regressor $\hat{\varphi}_L$ defined as follows:*

$$\mathbf{t} = \hat{\varphi}_L(\mathbf{I}) = \mathtt{H}\mathbf{I}, \tag{4.10}$$

where $\mathtt{H}$ *is a* $2 \times c$ *matrix.*

Similarly to this, we define sequence of linear predictors:

Definition 10. *Sequential Linear Predictor (SLLiP) is Sequential Predictor formed from LLiPs. Note, that the time required for motion estimation by LLiP is determined by the size of its support set, therefore complexity of SLLiP is $|X|$.*

Although we will further work with linear predictors, the method allows a natural extension to an arbitrary class of functions formed of a linear combination of kernels

function by *data lifting*. Polynomial mapping of a given order is an example. In that case, all monomials are considered as further observed intensities. It allows the learning procedure to deal with non-linear mappings via linear mappings with higher dimension.

Training set construction

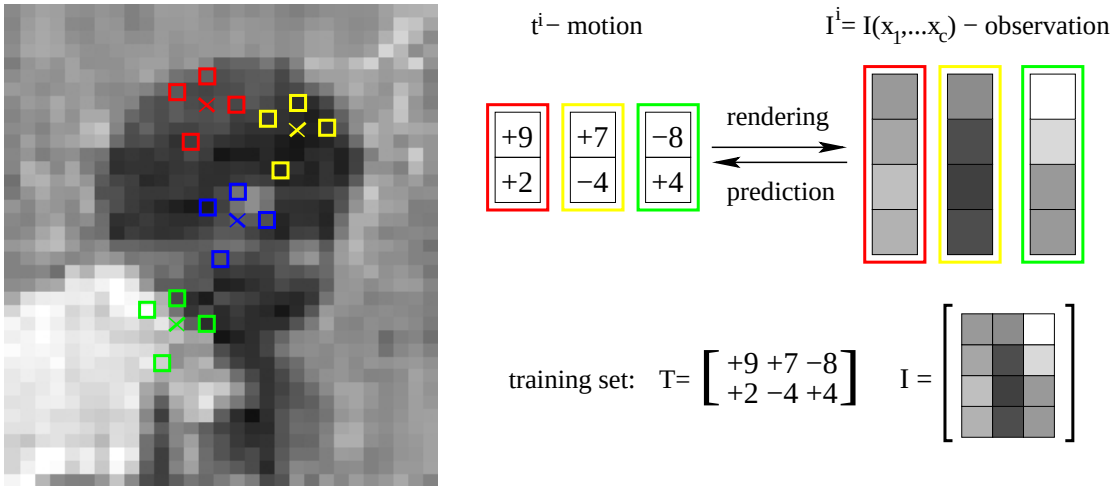
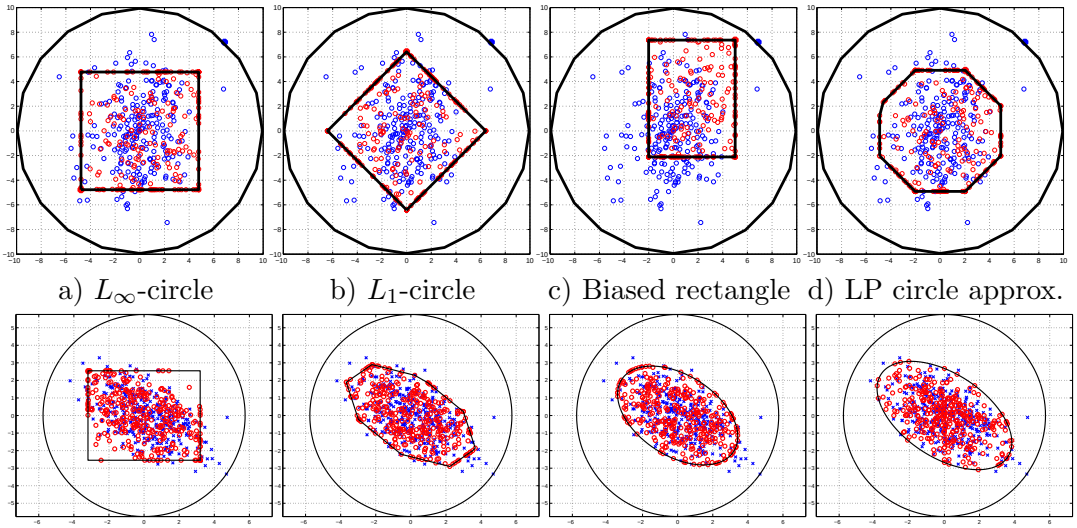


Figure 4.4: **Training set construction:** Image is perturbed by the motion parameters from the predefined range. Pairs of observed intensities $\mathbf{I}^i$ and corresponding motions $\mathbf{t}^i$ creates the training set.

Let us suppose we are given a reference point, a support set and a predefined range of motion within which the regressor is assumed to operate. We perturb the support set by the motion with parameters $\mathbf{q}^i$ randomly (uniformly) generated inside the range. Each motion $\mathbf{q}^i$ warps the support set X to a set X^i, where a vector of intensities $\mathbf{I}^i$ is observed, see Figure 4.4. Given the observed intensity, we search for a mapping assigning motion $\mathbf{t}^i = -(\mathbf{q}^i)$, which warps the support set X^i back to the original support set X. These examples are stored in matrices $\mathtt{I} = \left[\mathbf{I}^1 \dots \mathbf{I}^d\right]$ and $\mathtt{T} = \left[\mathbf{t}^1 \dots \mathbf{t}^d\right]$. The ordered triple $(\mathtt{I}, \mathtt{T}, \mathcal{X})$ of such matrices and ordered d-tuple of support sets $\mathcal{X} = \left\{X^1 \dots X^d\right\}$ is called a *training set*.

Learning linear regressor given a training set

Let us suppose, we are given a training set $(\mathtt{I}, \mathtt{T}, \mathtt{X})$. While Jurie and Dhome [33] obtain $\mathtt{H}$ by the least squares method $\mathtt{H} = \mathtt{T}\mathtt{I}^{+} = \mathtt{T}\mathtt{I}^{\top}(\mathtt{I}\mathtt{I}^{\top})^{-1}$, we search for λ-minimal predictor (Definition 6), i.e., the predictor with the smallest uncertainty region, equation (4.3). As we mentioned before, the uncertainty region is assumed to be from a class parameterizable by one scalar parameter λ. In the following, we show how to find λ-minimal predictor for the class of squares and rectangles. See appendix for other uncertainty region classes (e.g., circles or ellipses).



a) L_∞-circle b) L_1-circle c) Biased rectangle d) LP circle approx.

e) LP rectangle f) LP ellipse approx. g) SDP ellipse h) SDP LS ellipse

Figure 4.5: **Examples of uncertainty region classes:** Points correspond to the pose estimation errors $\Delta\mathbf{t}$. Errors of the predictor learned by the LS method are in blue and by minimax method in red. Uncertainty regions and ranges are black. Only a),b) and e) are described in this section, see Appendix A for detailed description of the other classes.

Restricting to the square-shaped uncertainty regions centered in the origin of the coordinate system (see figure 4.5-a) and parameterized by parameter λ, equation (4.3), defining the λ-minimal predictor, simplifies as follows

$$\begin{aligned} \mathtt{H}^* &= \underset{\mathtt{H}}{\operatorname{argmin}}(\max_i \|\mathtt{H}\mathbf{I}^i - \mathbf{t}^i\|_\infty) = \underset{\mathtt{H},\lambda}{\operatorname{argmin}}\{\lambda \mid \forall_i |\mathtt{H}\mathbf{I}^i - \mathbf{t}^i| < \mathbf{1}\lambda\} = \\ &= \underset{\mathtt{H},\lambda}{\operatorname{argmin}}\ \lambda \\ &\quad \text{subject to}: -\lambda \leq (\mathtt{H}\mathbf{I}^i)_k - \mathbf{t}^i_k \leq \lambda, \\ &\qquad\qquad i = 1 \ldots d,\ k = 1, 2. \end{aligned} \tag{4.11}$$

We reformulate problem (4.11) as a linear program

$$\min_{\mathbf{x}}\{\mathbf{c}^\top\mathbf{x} \mid \mathtt{A}\mathbf{x} \leq \mathbf{b}\}, \tag{4.12}$$

where

$$\mathbf{x} = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_p \\ \lambda \end{bmatrix}, \ \mathbf{c} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \ \mathtt{A} = \begin{bmatrix} \mathtt{I}^\top & \mathtt{0} & \dots & \mathtt{0} & -\mathbf{1} \\ \mathtt{0} & \mathtt{I}^\top & \dots & \mathtt{0} & -\mathbf{1} \\ \vdots & & \ddots & & \vdots \\ \mathtt{0} & \dots & \mathtt{0} & \mathtt{I}^\top & -\mathbf{1} \\ & & & & \\ -\mathtt{I}^\top & \mathtt{0} & \dots & \mathtt{0} & -\mathbf{1} \\ \mathtt{0} & -\mathtt{I}^\top & \dots & \mathtt{0} & -\mathbf{1} \\ \vdots & & \ddots & & \vdots \\ \mathtt{0} & \dots & \mathtt{0} & -\mathtt{I}^\top & -\mathbf{1} \end{bmatrix}, \ \mathbf{b} = \begin{bmatrix} \mathbf{t}_1 \\ \vdots \\ \mathbf{t}_p \\ \\ -\mathbf{t}_1 \\ \vdots \\ -\mathbf{t}_p \end{bmatrix},$$

where $\mathbf{h}_i$ is column vector corresponding to the i-th row of matrix $\mathtt{H}$.

Since the computation of each component of the predicted parameters can be considered as an independent task, estimation of each row of $\mathtt{H}$ is solved separately[3]. Hence, the task splits into p independent linear problems, where each of them determines one row $\mathbf{h}_j^{T*}$ of matrix $\mathtt{H}$. The problem is solved as follows:

$$\mathbf{h}_j^{T*} = \underset{\mathbf{h}_j}{\operatorname{argmin}} \max_i \left\{ \|\mathbf{h}_j^\top \mathbf{I}^i - t_j^i\|_\infty \right\} = \underset{\mathbf{h}_j^\top, \lambda_j}{\operatorname{argmin}} \{\lambda_j \mid \forall_i \, |\mathbf{h}_j^\top \mathbf{I}^i - t_j^i| < \lambda_j\}.$$

Denoting

$$\mathbf{x}_j = \begin{bmatrix} \mathbf{h}_j \\ \lambda_j \end{bmatrix}, \ \mathbf{c} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \ \mathtt{A} = \begin{bmatrix} \mathtt{I}^\top & \mathbf{0} & -\mathbf{1} \\ -\mathtt{I}^\top & -\mathbf{1} & \mathbf{0} \end{bmatrix}, \ \mathbf{b}_j = \begin{bmatrix} \mathbf{t}_j \end{bmatrix},$$

linear programming problem (4.12) is obtained. The shape of such uncertainty region is a rectangle, which is in 2D space parameterized by two parameters - length of its sides, see Figure 4.5c. Since we want to work with uncertainty regions parameterizable by one parameter, it could be considered as a square with the side equal to the longer rectangle side. Result is the same as if the square shape is assumed in advance and the learning is significantly faster.

If L_∞ in problem (4.11) is replaced by L_1 the uncertainty region is L_2 hyper-cube (square in 2D) rotated by 45^o and the problem is solved alike, see Figure 4.5b. The combination of L_∞ and L_1 norms allows to work also with L_2 circle approximation, see Figure 4.5d. Note that we can also work with elliptic regions as shown in Figure 4.5d-g. In order to adjust a trade-off between robustness of minimax solution and accuracy of LS solution, it is also possible to formulate the criterion as a weighted sum of LS error and minimax error, which can be shown to be a semi-definite problem, see Figure 4.5h. Detailed description of such uncertainty region extensions can be found in Appendix A.

[3]This is significantly faster than computation of one larger problem.

4.2.2 Learning optimal sequential predictor Φ^*

In this section, we describe selection of the optimal sequence of predictors from a set of learned λ-minimal predictors. We assume that we are able to estimate the λ-minimal predictors (4.3), e.g., the linear predictors as shown in previous section. Set of λ-minimal predictors $\varphi^+(c, r)$ for some discretized values of complexities $c \in C$ and ranges $r \in R$ is denoted by ω^+. Note, that ω^+ is actually a subset of the set of all possible λ-minimal predictors however, for the sake of simplicity we use the same notation. Figure 4.6a shows uncertainty region $\lambda(c, r)$ (size coded by color) of the λ-minimal predictors as a function of complexity $c \in C$ (vertical axis) and range $r \in R$ (horizontal axis).

Given the set ω^+, desired range r_0 and uncertainty region λ_0, we search for an ordered subset of ω^+ that forms the optimal sequential predictor Φ^*, which minimizes the complexity. Since the desired range r_0 of the sequential predictor is the range $r_1 = r_0$ of the first predictor in the sequence, the first predictor must lie in the corresponding (usually the most right) column. For this range, the predictors with the different complexities are available in that column. The higher the complexity is, the smaller the uncertainty region, see Figure 4.6a, where the size of uncertainty region decreases with the complexity for each particular range. Selection of a particular complexity c_1 determines the first λ-minimal predictor $\varphi^+(c_1, r_1)$ in the sequence. The size of corresponding uncertainty region $\lambda(\varphi^+(c_1, r_1))$ determines an admissible range r_2 of the following predictor, which has to be at least as large as the uncertainty region according to its definition, i.e., $r_2 \geq \lambda(c_1, r_1)$. The following proposition shows that it is sufficient to consider only the smallest possible range.

Proposition 4 *Range r_i of a λ-minimal predictor $\varphi^+(c_i, r_i)$ in an optimal sequence of λ-minimal predictors has to be as tight as possible to the uncertainty region λ_{i-1} of its predecessor, i.e., asymptotically, in a continuous case $r_i = \lambda_{i-1}$.*

Proof: The uncertainty region is a nonincreasing function of complexity, according to Proposition 1

$$c_2 > c_1 \Rightarrow \lambda(\varphi^+(c_2, r)) \leq \lambda(\varphi^+(c_1, r)).$$

However, a λ-minimal predictor whose complexity can be decreased without uncertainty region increase, cannot be part of an optimal sequence. We therefore consider only a λ-minimal predictor, whose uncertainty region is a decreasing function of complexity, i.e., for which the following holds:

$$c_2 > c_1 \Rightarrow \lambda(\varphi^+(c_2, r)) < \lambda(\varphi^+(c_1, r))$$

and consequently:

$$\lambda(\varphi^+(c_2, r)) \geq \lambda(\varphi^+(c_1, r)) \Rightarrow c_2 \leq c_1. \tag{4.13}$$

Hence, the uncertainty region is strictly decreasing function of the complexity. Putting this together with Proposition 2, which claims that the uncertainty region is a nondecreasing function of range, we prove that the complexity is a nondecreasing function of

the range for every fixed $\lambda_0 = \lambda(\varphi^+(c_1, r_1)) = \lambda(\varphi^+(c_2, r_2))$, because:

$$r_2 > r_1 \underset{\text{Prop.2}}{\Rightarrow} \begin{array}{l} \lambda(\varphi^+(c_1, r_2)) \geq \lambda(\varphi^+(c_1, r_1)) \\ \lambda(\varphi^+(c_2, r_2)) \geq \lambda(\varphi^+(c_2, r_1)) \\ \lambda(\varphi^+(c_1, r_1)) = \lambda(\varphi^+(c_2, r_2)) \end{array} \Rightarrow$$

$$\Rightarrow \begin{array}{l} \lambda(\varphi^+(c_1, r_2)) \geq \lambda(\varphi^+(c_2, r_2)) \\ \lambda(\varphi^+(c_1, r_1)) \geq \lambda(\varphi^+(c_2, r_1)) \end{array} \underset{\text{Eq.(4.13)}}{\Rightarrow} c_2 \leq c_1$$

Since the complexity is a nondecreasing function of the range, considering larger range $r_i > \lambda_{i-1}$ than necessarily leads only to increasing of the complexity c_i. Taking into account that this would necessarily increased the complexity of the resulting sequential predictor, the smallest possible range $r_i = \lambda_{i-1}$ must be used. ■

Note, that if only the smallest possible ranges are considered, then the constructed graph has at most $|C| \cdot |R|$ edges. On the contrary, without the Proposition 4 the constructed graph would have $|C| \cdot |R|^2$ edges.

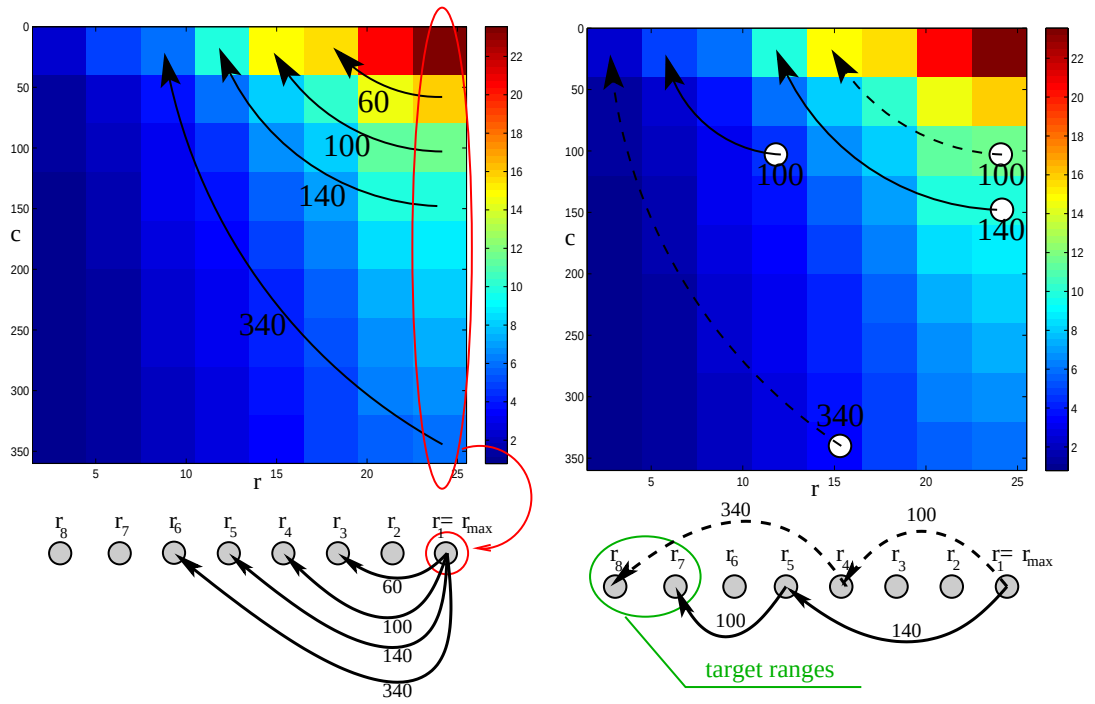
Arrows in Figure 4.6a show the smallest possible ranges for the predictors with different complexities. A sequence with the last predictor with uncertainty region λ_m smaller than λ_0 can be constructed, see for example the two sequences in Figure 4.6b. Furthermore, we search for the sequence consisting of predictors converging to the sufficiently small uncertainty regions with the lowest complexity.

We formulate the previous problem as the search for the cheapest path in the graph $\mathcal{G} = (V \equiv R, E \subseteq R \times R, \alpha \colon E \to C)$, where R is the set of considered ranges and C is the set of considered complexities and operator α assigns a cost to each edge, see Figure 4.6. It means, each range is associated with a vertex and a set of edges starting from this range, which stand for predictors with different complexities. Edge cost equals its complexity. We construct the graph by adding forward edges for each particular range.

The Dijkstra algorithm [10] searches for the cheapest path to the ranges with predictors with sufficiently small uncertainty regions, depicted by red circles in Figure 4.7. These predictors are called *target* predictors and their ranges are called *target* ranges. The solution is a sequence of predictors associated with edges on the cheapest path to a target range plus its cheapest target predictor. If more than one target range exists then there are more possible solutions and the cheapest solution is selected. The solution is the optimal sequential predictor (4.9). The method is summarized in Algorithm 1.

The optimal path is depicted in Figure 4.7a. For instance, the optimal sequence for the example in Figure 4.7a, where $r_0 = 100, \lambda_0 = 2$ is created as $\Phi^* = (\varphi^+(140, 25), \varphi^+(100, 12), \varphi^+(100, 5))$ and corresponding uncertainty regions are $(10, 4.5, 2)$.

Note that due to simplicity, we focus on the one-variable parameterized uncertainty regions. Extension of the proposed method to the more than one-variable parameterized uncertainty regions is straightforward. For w-dimensional parameterization of the uncertainty region shape is ω^+ $w + 1$-dimensional .



(a) Edges from the first vertex (range). (b) Two paths to target ranges.

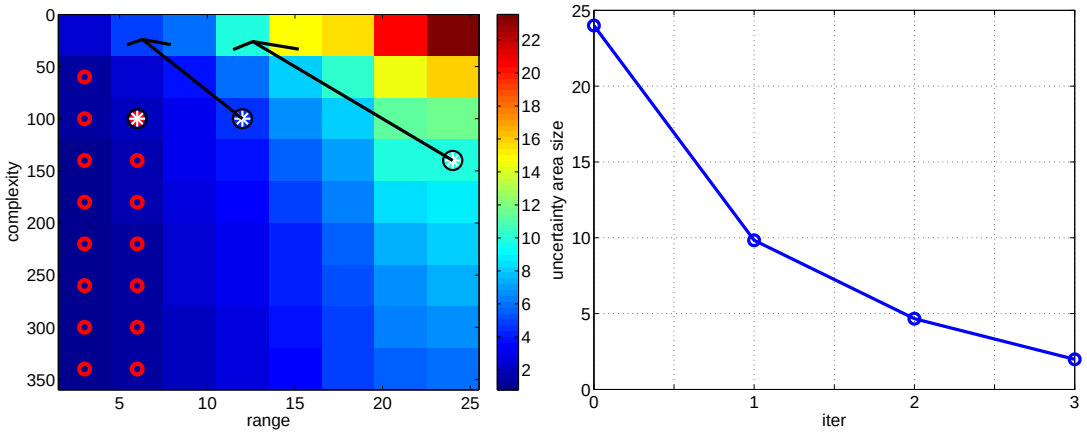
Figure 4.6: **Selection of the optimal sequential predictor:** (a) Construction of a graph $\mathcal{G}$ from a set of λ-minimal predictors ω^+. Different complexities of the first predictor lead to different uncertainty regions and therefore different ranges of the second predictor. Edges from the range r_0 of the first predictor, depicted by black arrows, show the ranges of the second predictor corresponding to the complexities of the first predictor. Cost of edges correspond to the complexities. (b) Two paths to the target ranges (solid line denotes the optimal path).

1. Estimate set of λ-minimal predictors $\omega^+ = \{\varphi^+(c,r) \mid c \in C,\, r \in R\}$.
2. Construct graph $\mathcal{G} = (V \equiv R, E \subseteq R \times R, \alpha : E \to C)$:

   for each $r \in R$ and each $c \in C$,
   a) Find the smallest possible following range v^* achievable by $\varphi^+(c,r)$: $v^* = \text{argmin}_{v \in R} \{v \mid \lambda(c,r) < v\}$
   b) $E = E \cup (r, v^*)$ and $\alpha(r, v^*) = c$

   end
3. Dijkstra($\mathcal{G}$, r_0) $\Rightarrow$ Compute the cheapest paths from r_0 to each range $r \in R$ by Dijkstra algorithm.
4. $\rho(r)$ denotes complexity of the cheapest path to range r. Consequently, $\varphi_t^+(c_t, r_t) = \text{argmin}_{\varphi^+(c,r) \in \omega_T} \{\rho(r) + c\}$ is the last predictor.
5. The optimal sequential predictor is created from the sequence of predictors associated with the edges of the cheapest path to r_t and the last predictor $\varphi_t^+(c_t, r_t)$.

Algorithm 1 Estimation of the optimal sequence from a set of λ-minimal LLiPs.



(a) Path with the lowest complexity. (b) Iterations.

Figure 4.7: **The optimal sequential predictor:** (a) size of uncertainty regions (coded by colors) as a function of complexity c (vertical axis) and range r (horizontal axis). The optimal path from the predefined range r_0 to a predictor with sufficiently small uncertainty region (red circles). (b) size of uncertainty region after each iteration (iter = 0 corresponds to the range $r_0 = 25$).

4.3 Anytime Learning

4.3.1 Introduction

In the previous section, we formulated the learning of SLLiP as an optimization problem where time of tracking (computational complexity) is minimized given a predefined precision of motion predictors. Although a globally optimal sequence is delivered, the learning might be prohibitively time consuming for large problems.

In the following, a new *anytime* learning approach is proposed which, after a very short initialization period, provides a solution with predefined precision. The solution is continuously improved, i.e., the SLLiPs with lower complexity and defined precision allowing for faster tracking are continuously delivered. The anytime learning searches through the space of SLLiPs and successively constructs SLLiPs from LLiPs of different complexities. In order to make the searching process efficient, the Branch & Bound [35] searching approach is used.

If no constraint on the learning time is imposed then the anytime learning algorithm finds the globally optimal solution with respect to a certain class of predictors. If time consuming learning is not affordable then the tracking can start immediately after a short initialization period. Since the SLLiP tracking requires only a fraction of processing power of an ordinary PC, the learning can continue in a parallel background thread.

Until now, the uncertainty region and range were assumed to be simple compact regions (e.g., circles). Consequently the optimal sequential predictor can be exclusively constructed from LLiPs learned by minimax. A disadvantage of this approach is the

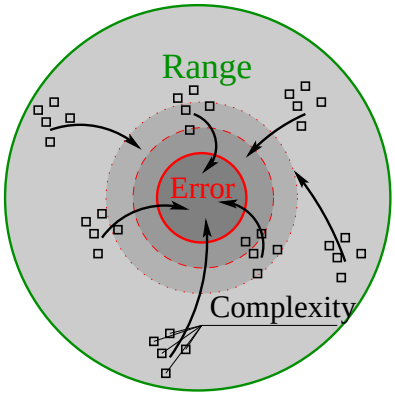


Figure 4.8: **Definitions:** Range, complexity and prediction error of learned linear predictor.

minimax learning, which consumes most of the learning time. Furthermore this learning process is unfortunately not able to provide an early solution with a predefined precision.

From now on, we relax the compactness condition of uncertainty regions and ranges. The space of possible SLLiPs is immense and its searching through inherently requires intractable number of operations of an informed or uninformed search algorithm. Therefore, we restrict the set of considered SLLiPs Ω (respectively LLiPs ω). Since the learning time is an issue and we mainly want to get rid of time consuming minimax learning, we focus on the SLLiPs constructed from LLiPs learned by the Least Squares method, i.e., minimizing prediction error (Figure 4.8) instead of the uncertainty region.

Definition 11. Prediction error *of predictor* $\varphi = (\mathtt{H}, X, R)$[4] *for range* R *is*

$$\epsilon(\varphi) = E\Big(\|\mathbf{t} - \mathtt{H}\mathbf{I}(\mathbf{t} \circ X)\|_2^2\Big), \ \forall \mathbf{t} \in R(\varphi),$$

where $E(.)$ *denotes the expectation value with respect to* $\mathbf{t}$ *uniformly distributed on* $R(\varphi)$[5].

Given training set $(\mathtt{I}, \mathtt{T}, \mathcal{X})$, the predictor $\varphi = (\mathtt{H}^*, X, R)$ with minimum prediction error has regressor $\mathtt{H}^*$ computed by the LS method as follows

$$\mathtt{H}^* = \underset{\mathtt{H} \in \mathbb{R}^{p \times c}}{\operatorname{argmin}} \|\mathtt{H}\mathtt{I} - \mathtt{T}\|_F^2 = \mathtt{T}\mathtt{I}^+. \tag{4.14}$$

The LS method is essentially computationally simpler with respect to linear programming used for the minimax solution[6].

[4]Previous definition of predictor also included uncertainty region, it is further omitted, because the uncertainty region is not further used.

[5]In practice, the error is the mean value of square Euclidean error of all predictions from the range.

[6]Nevertheless, the proposed learning algorithm can be used to find the globally optimal solution with

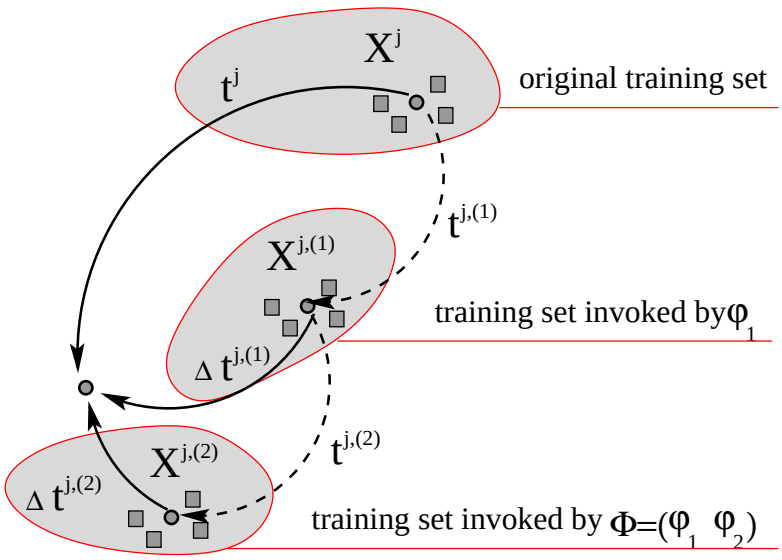


Figure 4.9: **Training set invoked by (sequential) predictor:** Observed intensities $\mathbf{I}^j \in \mathtt{I}$ are transformed by predictor φ_1 into motion parameters $\mathbf{t}^{j,(1)}$ which are as close as possible to the desired motions $\mathbf{t}^i \in \mathtt{T}$. We warp the original motions $\mathbf{t}^j \in \mathtt{T}$ to the new motion parameters $\Delta\mathbf{t}^{j,(1)} = (\mathbf{t}^{j,(1)})^{-1} \circ \mathbf{t}^j$ which need to be further compensated, and we also warp each support set $X^j \in \mathcal{X}$ by motion parameters $\mathbf{t}^{j,(1)}$ obtaining the new support set $X^{j,(1)} = \mathbf{t}^{j,(1)} \circ X^j$. Newly observed intensities $\mathbf{I}^{j,(1)} = \mathbf{I}(X^{j,(1)})$ create in conjunction with $\Delta\mathbf{t}^{j,(1)}$ and $X^{j,(1)}$ training set *invoked by predictor* φ_1.

4.3.2 Anytime learning algorithm

In this section, we describe method estimating the optimal sequential predictor given a training set $(\mathtt{I}, \mathtt{T}, \mathcal{X})$ generated on image $\mathcal{I}$. Ideally, the predictor, say φ_1, learned according to equation (4.14) would transform intensities $\mathbf{I}^j \in \mathtt{I}$ exactly to motions $\mathbf{t}^j \in \mathtt{T}$. However, such predictor usually does not exist. Therefore the observed intensities are transformed into motion parameters $\mathbf{t}^{j,(1)}$ which are as close as possible to the desired motions $\mathbf{t}^i \in \mathtt{T}$, in sense of equation (4.14). We warp the original motions $\mathbf{t}^j \in \mathtt{T}$ to the new motions $\Delta\mathbf{t}^{j,(1)} = (\mathbf{t}^{j,(1)})^{-1} \circ \mathbf{t}^j$ which need to be further compensated, see Figure 4.9. And we also warp each support set $X^j \in \mathcal{X}$ by motion parameters $\mathbf{t}^{j,(1)}$ obtaining the new support set $X^{j,(1)} = \mathbf{t}^{j,(1)} \circ X^j$. Denoting $\mathbf{I}^{j,(1)} = \mathbf{I}(X^{j,(1)})$ the intensities observed

respect to arbitrary ω. For example, ω could be a set of LLiPs learned by the minimax method, then the result of learning would be the same as of the algorithm proposed in Section 4.2. Note that the globally optimal solution found with respect to the restricted ω is not guaranteed to provide globally optimal solution with respect to the set of *all* possible LLiPs.

on these newly obtained support sets, we form the new training set $(\mathtt{I}^1, \mathtt{T}^1, \mathcal{X}^1)$, where

$$\mathtt{I}^1 = \left[\mathbf{I}^{1,(1)} \dots \mathbf{I}^{d,(1)}\right], \mathtt{T}^1 = \left[\Delta\mathbf{t}^{1,(1)} \dots \Delta\mathbf{t}^{d,(1)}\right], \mathcal{X}^1 = \left[X^{1,(1)} \dots X^{d,(1)}\right]. \tag{4.15}$$

We refer to it as to training set *invoked by predictor* φ_1 and denote it $\mathcal{T}(\varphi_1, c)$, where c denotes the size of support set used in the training set. Similarly to this we also define training set *invoked by sequential predictor* Φ as $\mathcal{T}(\Phi, c)$.

Note that given a training set $\mathcal{T}(\Phi, c_1)$, we can simply generate a training set $\mathcal{T}(\Phi, c_2)$ for the predictor with a lower complexity $c_2 < c_1$ by removing corresponding number of pixels from $\mathcal{X}$ and corresponding number of rows from matrix $\mathtt{I}$. We refer to this process as the training set *restriction*.

In order to simplify the problem, we further work with a discretized set of complexities C. The optimal sequence of predictors is found by searching through the set of all considered SLLiPs, which involve $\sum_{i=1}^{m} |C|^i$ elements, where $|C|$ denotes the size of C and m is maximum length of SLLiP. In order to make the searching process efficient, the Branch & Bound [35] (BB) searching approach is used. Sequential predictors are successively constructed from the LLiPs of different complexities: In the first level, we learn LLiPs for all complexities in C according to equation (4.14). They correspond to the SLLiPs of the length equal to one. One of these SLLiPs, say Φ, is expanded in the next iteration. The expansion means that Φ is successively extended by LLiPs with different complexities $c \in C$ learned on training set invoked by itself $\mathcal{T}^i(\Phi, c)$. This process creates $|C|$ new SLLiPs, which may or may not be expanded in further iterations. Once a SLLiP with a sufficiently small prediction error is found, all other partially constructed SLLiPs with a higher complexity are terminated, i.e., they will never be expanded. The smallest complexity c^* of the found solution is saved and once a SLLiP reaches a higher complexity it is automatically terminated. The learning process is summarized in Algorithm 2; see also Figure 4.10, which demonstrates six iterations of the algorithm on a toy example with $C = \{20, 300\}$, range equal to 40% of the object size and the predefined error 10% of the object size.

Note that the selection strategy $\mathcal{S}$ which selects a SLLiP from Ω (step 4), may influence the learning behaviour. However, if Algorithm 2 satisfies condition $\Omega = \emptyset$ in step 6, Φ^* is an optimal SLLiP with respect to the set of considered LLiPs ω. In our implementation, we first use the strategy which expands the SLLiP with the highest complexity. This strategy usually finds a solution Φ^* in a few iterations. This solution is of a high complexity, but the prediction error is guaranteed and the tracking is allowed to start. The strategy is switched and the SLLiPs with the average complexity are preferably expanded. Once a solution is reached, it can be immediately used for tracking with a lower performance. If the learning continues, the SLLiP can be in future replaced by better solutions.

The stoping condition (step 6) could be also optionally replaced for example by a maximum number of iterations, maximum running time, maximum depth of the constructed graph or an arbitrary intersection of these conditions. However, such replacement might influence the optimality of the found Φ^*.

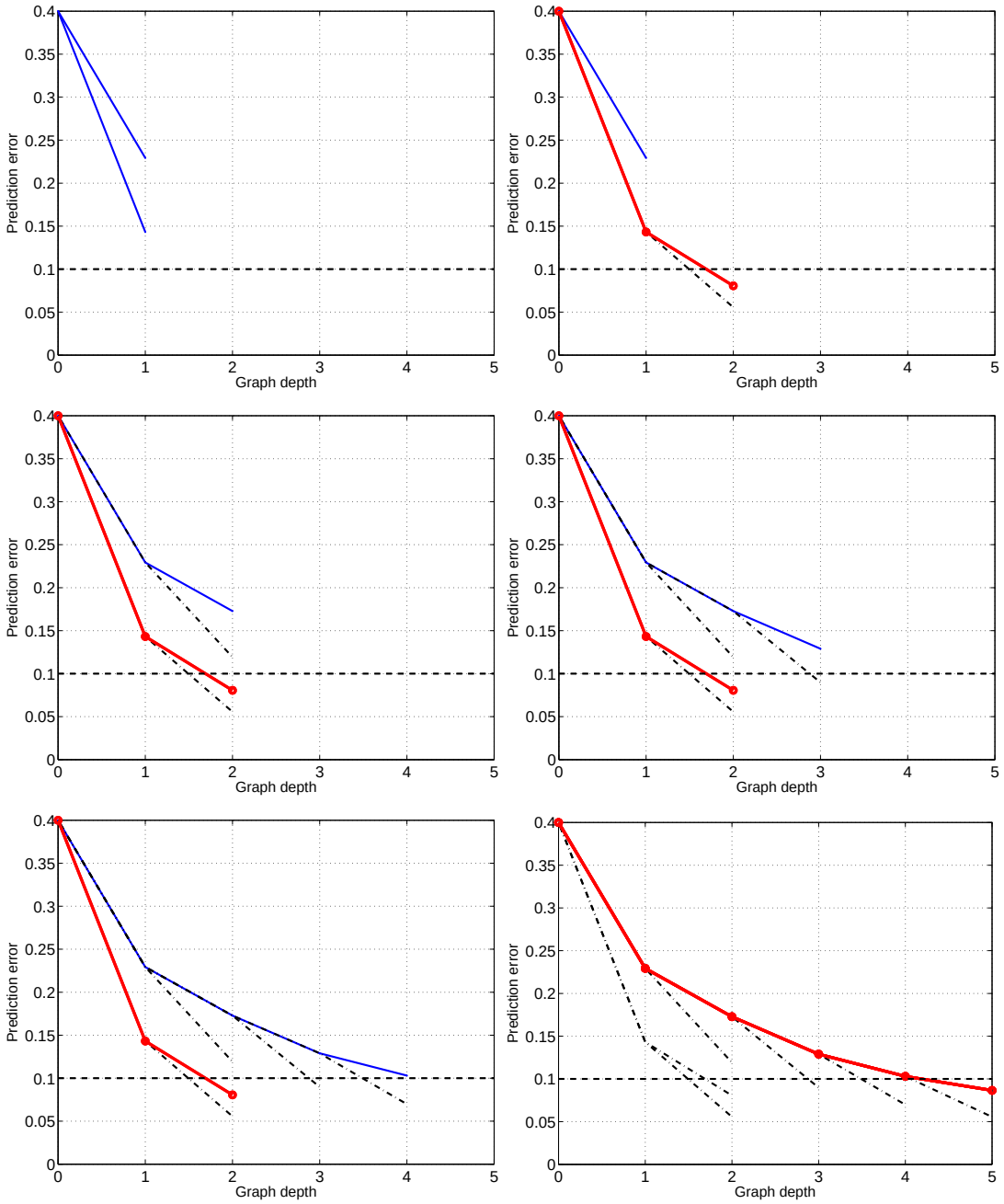


Figure 4.10: **Demonstration of Algorithm 2** on toy example with $C = \{20, 300\}$, $|C| = 2$, $R = 0.4$ and $\epsilon_0 = 0.1$. Blue denotes a set of current SLLiPs Ω, black denotes terminated SLLiPs and red delineates solution Φ^* with the lowest complexity so far.

Input:

Range R within which SLLiP is expected to operate.
Set of considered complexities C.
Predefined accuracy ϵ_0.
Training image $\mathcal{I}$.
Support set X.

---

1. Set:

   $c^* = \infty$ (complexity of the simplest admissible SLLiP found) and
   $i = 0$ (number of current iteration).

2. Generate training sets $\mathcal{T}^0(c)$, $\forall c \in C$ on the predefined range R^a.

3. Initialize set Ω of learned active SLLiPs as a set of LLiPs learned on $\mathcal{T}^0(c)$, $\forall c \in C$ according equation (4.14).

4. $\Phi = \mathcal{S}(\Omega)$, $\Omega = \Omega \setminus \Phi$ (Select and remove Φ according to a strategy $\mathcal{S}$.)

5. For each $c \in C$: (expand Φ)
   a) Generate training set $\mathcal{T}^i(\Phi, c)$ invoked by Φ.
   b) Learn LLiP φ for $\mathcal{T}^i(\Phi, c)$ according to equation (4.14).
   c) $\Phi' = (\Phi, \varphi)$, $\Omega = \Omega \cup \Phi'$ (add the new SLLiP to Ω)
   d) If $e(\Phi') \leq \epsilon_0$ and $c(\Phi') < c^*$ then $\Phi^* = \Phi'$ and $c^* = c(\Phi')$ (replace solution)
   e) For $\forall \Phi'' \in \Omega$ with $c(\Phi'') > c^*$, do $\Omega = \Omega \setminus \Phi''$ (terminate the SLLiPs with higher complexity)

   end

6. If $\Omega = \emptyset$ stop otherwise $i = i + 1$ and goto 4.

Output:

Optimal SLLiP Φ^*

[a] We generate only $\mathcal{T}^0(c_{max})$ where $c_{max} = \max C$ is maximum complexity of C, the other training sets with the lower complexity are constructed by its restriction.

Algorithm 2 Anytime learning for SLLiP.

4.4 Selection of support set for efficient tracking

Until now, we assumed that the support set was given. Since the support set selection, which minimize an error on a training set has combinatorial complexity, we propose a heuristic method. The only condition on the proposed heuristic is that every selected support set of complexity c contains also support set of complexity $c-1$, which consequently assures monotonicity of the λ-bound as shown in Section 4.1.

Let us suppose we are given training set $(\mathtt{I}, \mathtt{T}, \mathcal{X})$ with support set covering the whole object. We define a support set *selection* vector $\mathbf{u} \in \{0,1\}^b$, which determines the support set selected from a b-pixel template; used pixels marked by ones, unused pixels marked by zeros, respectively. The prediction error of a predictor operating on the support set selected by $\mathbf{u}$ is

$$e(\mathbf{u}) = \left\| \mathtt{T} - \mathtt{T}\left(\mathtt{I}(\mathbf{u},:)\right)^{+} \mathtt{I}(\mathbf{u},:) \right\|_F^2, \tag{4.16}$$

where $\mathtt{I}(\mathbf{u},:)$ is a submatrix of $\mathtt{I}$ with rows selected by $\mathbf{u}$. Given a desired complexity c, the optimal solution of problem,

$$\mathbf{u}^* = \underset{\substack{\mathbf{u} \in \{0,1\}^b, \\ \|\mathbf{u}\|_1 = c}}{\operatorname{argmin}} \; e(\mathbf{u}), \tag{4.17}$$

is usually intractable because the problem has combinatorial complexity. Therefore we propose the following greedy LS algorithm for the support set selection problem, which searches for a solution convinient for efficient tracking.

1. Let $\mathbf{u} = \mathbf{0}$ is the selection and $\mathtt{L}, \mathtt{T}$ are given training examples.
2. Repeat c-times:
$$j^* = \underset{j=1\ldots b}{\operatorname{argmin}} \{e(\mathbf{u} + \delta_j)\},$$
$$\mathbf{u}(j^*) = 1$$
where δ_j is b-vector of zeros with "1" at the position j.

Algorithm 3 Greedy LS support set selection algorithm.

4.5 Discussion

In this section, we summarize and discuss previously described learning and tracking methods for SLLiPs. Learning described in Section 4.2 considers only ranges and uncertainty regions being from a class of connected regions, usually parameterizable by the one scalar parameter. Under this assumption, the optimal SLLiP with defined range and uncertainty region is found. It is shown that the optimal SLLiP must be formed from LLiPs computed by minimax method. Theoretically, it is also possible to construct the SLLiP from LLiPs learned by any other method. Such SLLiP is not optimal anymore,

but the learning might be faster. For example LLiPs learned by the least square method (LS) allows for very fast learning in contrast to the minimax. However, LLiP learned by the LS method has a very large uncertainty region and its ancestor must inherently have a very large range. The range of the following LLiP may optionally cover only a $p\%$ of cases. As a result of such strategy, $(1-p)^m\%$ of training examples will not converge for the SLLiP of length m. Since the error density of LS learned LLiPs is usually from normal distrubtion, even $p = 99\%$ significantly decreases the range.

Learning algorithm described in Section 4.3 successively constructs SLLiPs from LLiPs with a different complexities. Once a SLLiP with sufficiently small error is achieved it can be immediately used for tracking, while the learning continues in a separate background thread. There are no explicit assumptions on the range or the uncertainty region, however, in order to make the search space reasonably large, the set of considered LLiPs must be restricted. Estimated SLLiP is then optimal with respect to the restricted set of LLiPs. Since the learning time is an issue, we restrict ourselves to LLiPs learned by LS, which is known to be very fast. If we restricted ourselves to LLiPs learned by minimax, the estimated SLLiP would be the same as the SLLiP found by the method from Section 4.2, but the learning would be longer, because the evidence that the regions has total ordering would not be taken into account. Of course, the anytime property of the learning would be preserved.

We further refer to the SLLiPs learned according to Section 4.2 as MM SLLiPs and SLLiPs learned according to Section 4.3 are called LS SLLiPs. Carefull validation of accuracy and robustness on ground truthed sequences is described in Chapter 6, where objects are tracked by number of SLLiPs.

4.6 Summary

We proposed very fast tracking by learning approach, where object motion is estimated by a sequence of learned linear predictors (SLLiP). Defining computational complexity of tracking as a number scalar multiplication necessary for the motion estimation, the learning is formulated as the computational complexity minimization subject to predefined accuracy and range.

Two different learning approaches were proposed: First one estimates globally optimal SLLiP using dynamic programming. Nevertheless for some applications this learning might be prohibitively time consuming, therefore anytime learning algorithm, which can continuously provides improved SLLiPs during the learning procedure, is also proposed. Result of the learning is extremely fast SLLiP with defined properties. Note that, a non-optimized C++ implementation of an average sequential predictor takes only $30\mu s$ on an average machine (1xK8 3200+ MHz).

In the following chapter, the object is modeled by a set of SLLiPs which independently estimates its local motions. Object motion is robustly determined by the RANSAC with online and offline tuned parameters. Such tracker is than compared to state-of-the-

art trackers [33, 36, 33] on approximately 12000 ground truthed frames, showing its superiority in accuracy, robustness and frame-rate.

We encourage the reader to download a MATLAB implementation of the proposed learning and tracking and additional materials at `http://cmp.felk.cvut.cz/~zimmerk/linTrack`.

5 Tracking objects with a known geometrical model

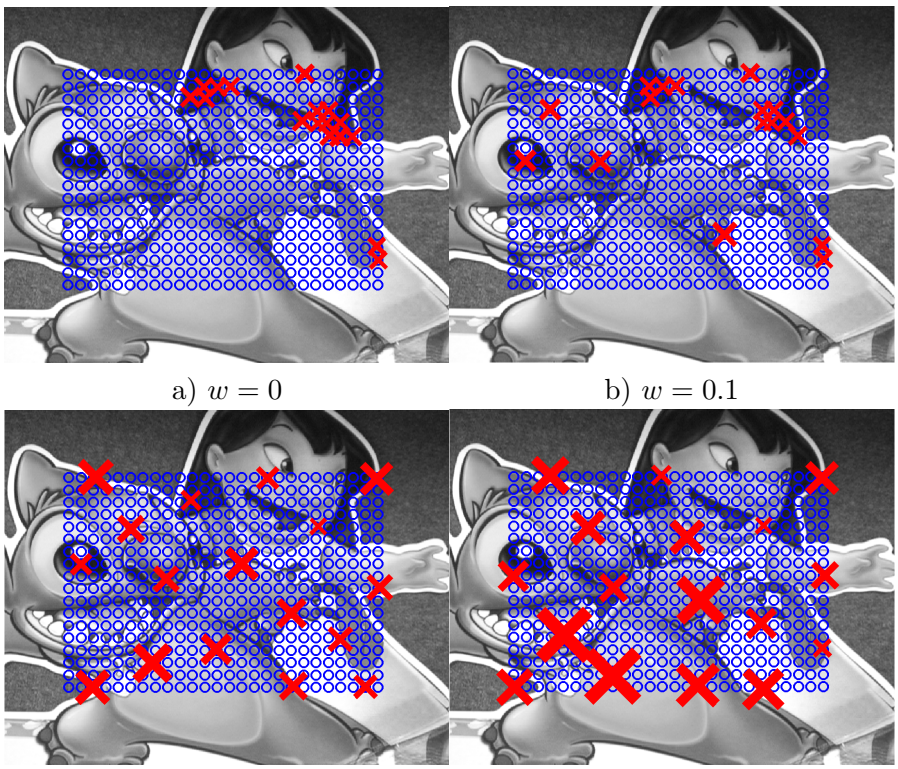
If the object is represented only by a single SLLiP, the robustness to partial occlusions/noise and the dimensionality of predicted motions are limited. Therefore we represent the object by Number of SLLiPs (NoSLLiP tracker). Such representation requires a geometrical model of the object. Since the geometrical model estimation is beyond the scope of this work, we mainly work with planar or piece-wise planar objects, the accurate geometrical model of which could be manually estimated with negligible effort.

Besides of the geometrical model estimation, many other questions must be answered: In particular, how many SLLiPs should be used, where should be attached to the model and how should be particular motion contributions combined. In our approach, we follow the most common way of robust motion estimation based on the RANSAC. Since we do not make any assumptions about the object pose, we learn SLLiPs equally distributed on the object. During the tracking stage a set of active SLLiPs, maximizing a trade-off between coverage and quality, is automatically selected and used for motion estimation. This method is introduced in Section 5.1. It is also not clear, how many SLLiPs should be used and how many RANSAC's iterations should be computed. Section 5.2 describes method estimating the ratio between number of SLLiPs and number of RANSAC's iterations, which maximize a probability of successful tracking.

5.1 Online selection of active predictor set

Let us suppose, that a set of SLLiPs evenly distributed on the object is available. In the following we describe how to select a subset of SLLiPs, which assures both a reasonable coverage of the object and quality of SLLiPs. It is not possible to find the set of regions suitable for object tracking independently on the object position, because if the object changes its pose some points can disappear and the global motion estimation can easily become ill-conditioned. In this section, we present an online method which automatically selects a subset of n predictors, called *active predictor set*, from all visible predictors.

To optimize the distribution of SLLiPs across the surface, we define *coverage measure* $r(Z)$ and *quality measure* $q(Z)$ of the set of SLLiP's reference points Z. Note, that we have no theoretical justification for these definitions and we do not claim, that this is the only right way how to define it. We provide only one possible definitions which might not be convienient for some applications.



a) $w = 0$ b) $w = 0.1$

c) $w = 0.5$ d) $w = 1$

Figure 5.1: **Influence of the weight on the active predictor set:** Blue circles correspond to the all learned predictors, red crosses to the selected predictors. Size of crosses corresponds proportionally to the complexity.

Definition 12. Coverage measure *is*

$$r(Z) = \sum_{\mathbf{z} \in Z} d(\mathbf{z}, Z \setminus \mathbf{z}), \tag{5.1}$$

where distance between point $\mathbf{z}$ *and set* Z *is defined as the distance from the closest element of the set*

$$d(\mathbf{z}, Z) = \min_{\mathbf{y} \in Z} \|\mathbf{z} - \mathbf{y}\|. \tag{5.2}$$

Ideally, for optimal robustness to occlusion the coverage measure would be maximized. In practice, particular SLLiPs differ by their complexities. Complexity corresponds to the suitability of SLLiP neighbourhood for motion estimation. We have experimentally shown that: the lower the complexity, the higher the robustness. Therefore, we derive the quality measure from complexity $c(\mathbf{z})$.

Definition 13. Quality measure *is*

$$q(\mathbf{z}) = |c(\mathbf{z}) - \max_{\mathbf{y} \in Z} c(\mathbf{y})|. \tag{5.3}$$

To find a suitable subset Z of predictors from all visible predictors $\widetilde{Z}$ we seek to optimize the weighted sum of the coverage r and quality q:

$$f(Z) = w\frac{r(Z)}{r(\widetilde{Z})} + (1-w)\frac{q(Z)}{q(\widetilde{Z})}, \tag{5.4}$$

where $w \in [0;1]$ is the coverage weight. Algorithm 4 selects a set of active SLLiPs, gGiven predefined number of SLLiPs n.

1. Let $\widetilde{Z}$ be the set of visible predictors and $Z = \emptyset$ a subset of selected reference points.
2. Select $\mathbf{z}^* = \arg\max_{\mathbf{z} \in \widetilde{Z} \setminus Z} f(\mathbf{z} \cup Z)$
3. $Z = \mathbf{z}^* \cup Z$ and $\widetilde{Z} = \widetilde{Z} \setminus \mathbf{z}^*$
4. if $|Z| = n$ end, else goto 2

Algorithm 4 Selection of active set of SLLiPs.

Figure 5.1 shows results obtained for $w = \{0, 0.1, 0.5, 1\}$. If $w = 0$, n predictors with the highest quality are selected and SLLiPs are stacked in one corner. Conversely, $w = 1$ causes that SLLiPs are equally spread across the object.

5.2 Object motion estimation

Objects are modeled as a spatial constellation of optimal SLLiPs (henceforward just predictors), which estimates 2D translation. Object motion is estimated from these local translations by RANSAC algorithm. We understand tracking as a time-limited task, where the object pose needs to be estimated from a camera image before the next image comes. There is a trade-off between the time spent with the local motion estimation and the global motion estimation. While there are n local motions estimated by n predictors, the global motion is estimated by h iterations of RANSAC. The longer the time spent with each particular step the higher the probability of successful tracking. We address the following question: Given the frame-rate and the computational costs of different operations at a specific computer, how many predictors should be used and how many RANSAC's iterations should be performed in order to maximize the probability of successful tracking?

The probability of a successful pose estimation in h-iterations of the RANSAC method is

$$P_R(k,h) = 1 - \left(1 - \left(\frac{k}{n}\right)^v\right)^h, \tag{5.5}$$

where n is the number of tracked points, k is the number of successfully tracked points, and v is the minimal number of the points needed for the pose estimation. Note, that $\frac{k}{n}$ is the percentage of the successfully tracked points (inliers). Number of successfully tracked points k in not known in advance, it is a random quantity with binomial distribution,

$$P_k(k) = P_{\text{bin}}(n,k) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{5.6}$$

where p is the probability of successful tracking of each particular reference point. Hence, the probability of successful tracking is

$$\begin{aligned} P_{\text{success}}(n,p,h) &= \sum_{k=1}^{n} P_R(k,h) P_k(k) = \sum_{k=1}^{n} P_R(k,h) P_{\text{bin}}(n,k) \\ &= \sum_{k=1}^{n} \left[1 - (1 - \left(\frac{k}{n}\right)^m)^h\right] \binom{n}{k} p^k (1-p)^{n-k}. \end{aligned}$$

In the rest, we assume that p is a constant value that has been estimated, e.g., online as a mean number of inliers or measured on training data. $P_{\text{success}}(n,p,h)$ is therefore replaced by $\hat{P}_{\text{success}}(n,h)$. The case where p is not fixed is discussed later. Given

- the maximum time t we are allowed to spend in pose estimation,
- time t_0 of one RANSAC iteration and
- times $t_1, \ldots, t_n$ required for local motion estimation or reference points $1, \ldots n$,

we formulate the following constrained optimization task

$$(n^*, h^*) = \{\arg\max_{n,h} \hat{P}_{\text{success}}(n,h) \mid h t_0 + \sum_{i=1}^{n} t_i \leq t\} \tag{5.7}$$

Since, the probability $\hat{P}_{\text{success}}(n,h)$ is a monotonously increasing function in all variables, the maximum has to be located on the boundary $\{[n,h] \mid h t_0 + \sum_{i=1}^{n} t_i = t\}$ of the constrained set. Consequently, problem (5.7) can be rewritten as the unconstrained one-dimensional problem as follows

$$n^* = \arg\max_n \hat{P}_{\text{success}}(n, \frac{t - \sum_{i=1}^{n} t_i}{t_0}) = \arg\max_n \overline{P}_{\text{success}}(n). \tag{5.8}$$

We are not able to proof analytically concavity of this function but it is experimentally shown that $\overline{P}_{\text{success}}(n)$ is a concave function. If interested in a real-time application, Golden mean optimization is a natural choice. The probability evaluation is very simple and the computational time can be practically neglected.

5.3 Summary

In this chapter the object were modeled by Number of optimally learned SLLiPs (NoSLLiP) from the previous chapter. While SLLiPs estimates local motions, object motion is determined by the RANSAC. In the proposed method, the number of SLLiPs and the number of RANSAC iterations are automatically determined, given a time limit induced by a predefined frame-rate. Since all the time-consuming operations are performed in the learning stage the NoSLLiP tracking requires only a few hundreds multiplications yielding extremely efficient motion estimation. Note that, a non-optimized C++ implementation of an average sequential predictor takes only $30\,\mu s$[1].

[1] We believe that in a future, most of the computations could be significantly accelerated using a GPU hardware.

6 Experiments - motion estimation

In this chapter, properties of SLLiP tracking and learning algorithms are demonstrated. In Section 6.1 some preliminary results on challenging sequences are demonstrated. In Section 6.2 robustness and accuracy is evaluated on ground truthed sequences. In particular, Section 6.2.1 describes ground truthed data, Section 6.2.2 compares SLLiPs learned by algorithms from Sections 4.3, 4.2, and Section 6.2.3 compares SLLiP to the state-of-the-art approaches. In Section 6.3 additional properties are summarized.

6.1 Preliminary qualitative evaluation

In the first experiment tracker is qualitatively evaluated on real sequences with planar and 3D rigid objects, which exhibit oblique views, motion blur, partial occlusions and significant scale changes. We encourage the reader to look also at video-sequences available at `http://cmp.felk.cvut.cz/linTrack`. Tracking of various objects with partial occlusions and motion blur is shown in Figure 6.1. Green/blue circles outline inliers/outliers, red arrows shows local motion estimated by SLLiPs. In some image also support set is outlined by blue points. Tracking of objects with variable set of active predictors is demonstrated in Figure 6.2 and 6.3. Active set of visible SLLiPs is estimated by Algorithm 3. Yellow numbers denotes IDs of particular SLLiPs. Although we mainly work with planar objects in order to avoid problems arising from inaccurate 3D reconstruction, SLLiPs are attachable to arbitrary 3D model, see for example Figure 6.3.

6.2 Quantitative evaluation of robustness and accuracy

6.2.1 Ground truthed data

The quantitative evaluation of robustness and accuracy of SLLiPs is conducted on sequences with 3 different objects (MOUSEPAD, TOWEL and PHONE), where ground truth positions of the object corners in total number 11963 frames were manually labeled[1], see Figure 6.4 for some examples. Accuracy is measured in each corner as a percentage; the displacement error is related to the current size of the object upper edge. Robustness is measured by the number of loss-of-locks, defined as the cases where the accuracy was

[1]These ground truthed sequences are available at `ftp://cmp.felk.cvut.cz/pub/cmp/data/lintrack`

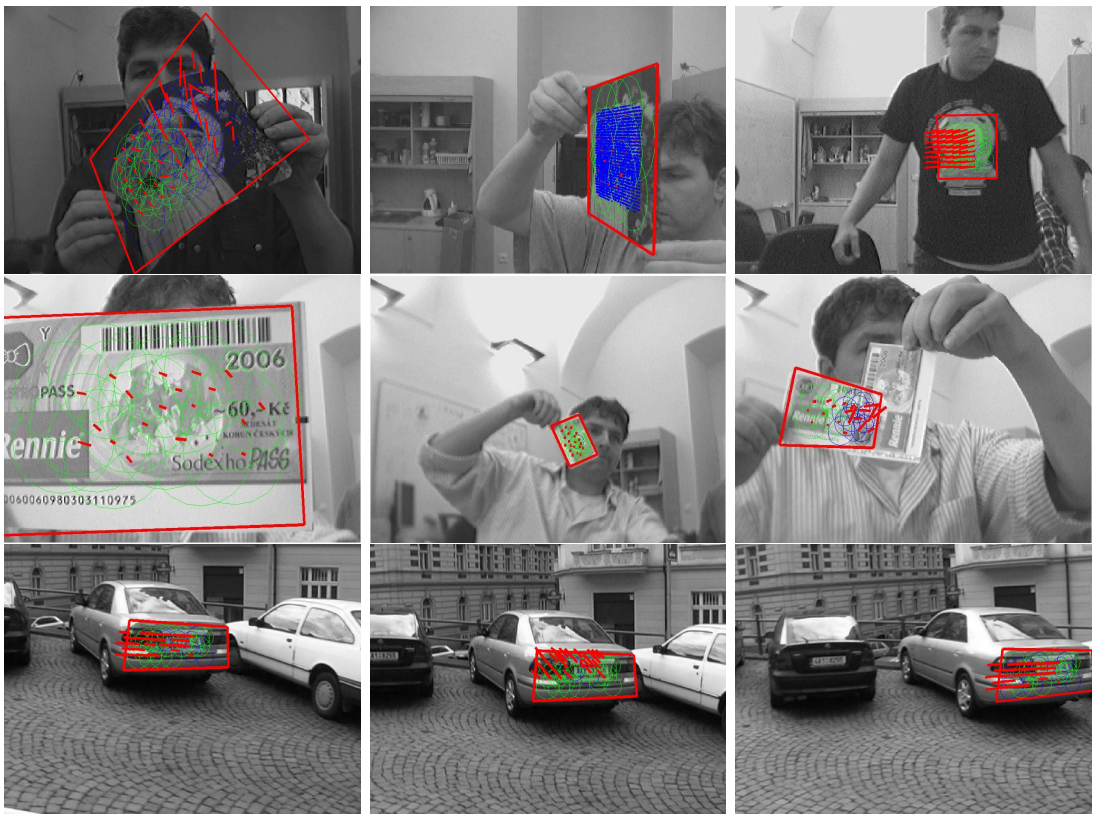


Figure 6.1: **Robustness to partial occlusions and fast motion:** Green/blue circles outline inliers/outliers, red arrows shows local motion estimated by SLLiPs. Support set outlined by blue points.

worse than 25%. In loss-of-lock frames, the tracker was reinitialized from the ground truth and the accuracy did not contribute to the total accuracy statistic.

6.2.2 Comparison of LS and MM SLLiPs

Some of the successfully tracked frames, which include oblique views, motion blur and significant scale changes, are presented in Figure 6.4. The results are summarized in Table 6.1.

In the first row, results of NoSLLiP tracker with SLLiPs learned by anytime algorithm (LS) proposed in Section 4.3 with no time constraint are presented. Second row contains results for SLLiPs learned by minimax algorithm (MM) proposed in Section 4.2. The minimax learning minimize the size of a compact region within which all prediction errors lie (uncertainty region) instead of the square of Euclidean error, therefore higher robustness is achieved, but the learning is 10-20 times longer due to time consuming

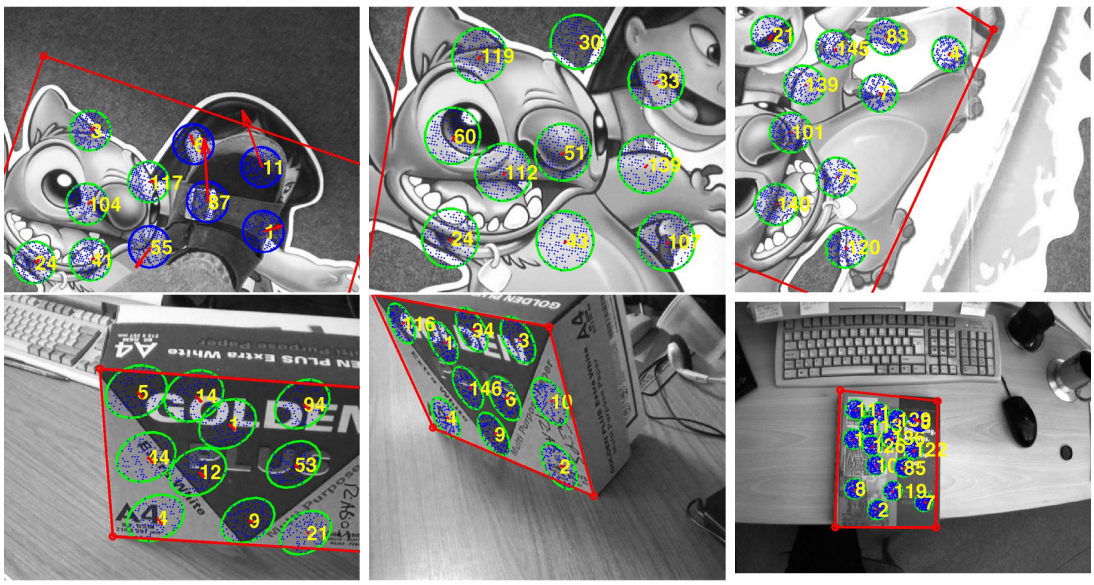


Figure 6.2: **Tracking with variable set of active predictors:** Yellow numbers denote ID of particular SLLiPs. Blue points represent support set, green circles highlight inliers, red arrows outline local motion estimated by SLLiPs.

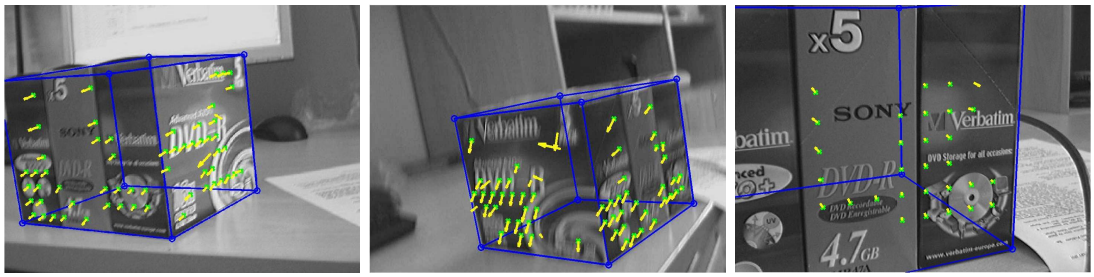


Figure 6.3: **3D tracking:** Variable set of active predictors and motion blur.

linear programming. Tracking accuracy of MM and LS methods varies with data; the accuracy is similar for MOUSEPAD and TOWEL sequences, but PHONE object contains similar repetitive structure (buttons) which make the LS accuracy significantly worse.

We also compare predefined uncertainty region λ_0 of MM SLLiP, predefined prediction error ϵ_0 of LS SLLiP and true error distribution on ground truthed data (MOUSEPAD sequence). We learned 35 SLLiPs with different ranges covering the mousepad. MM SLLiPs are learned to achieve uncertainty region $\lambda_0 = 5\%$, LS SLLiPs are learned for prediction error $\epsilon_0 = 3\%$. Both the uncertainty region and the prediction error are relative to SLLiPs range. λ_0, ϵ_0 were chosen experimentally in order achieve the

Figure 6.4: **Ground truthed sequences:** Left column shows images used for training. The middle and right columns demonstrate some successfully tracked frames with strong motion blur from the testing sequences. Blue rectangle delineates the object. Percentage values in corners are current corner speeds related to the current size of the object upper edge.

object	SLLiP learning	learning time [sec]	processing [fps]	loss-of-locks	mean-error [%]
MOUSEPAD	LS	11	27.6	17/6935	[1.4, 1.3, 1.1, 1.1]
MOUSEPAD	MM	310	18.9	13/6935	[1.3, 1.8, 1.5, 1.6]
TOWEL	LS	16	33.3	5/3229	[1.6, 1.8, 1.1, 1.5]
TOWEL	MM	310	21.8	2/3229	[3.0, 2.2, 1.4, 1.9]
PHONE	LS	21	25.6	55/1799	[7.3, 7.1, 10.6, 6.5]
PHONE	MM	310	16.8	20/1799	[1.2, 1.8, 2.6, 1.9]

Table 6.1: **Comparison of robustness and accuracy of LS SLLiPs and MM SLLiPs:** NoSLLiP tracker with SLLiPs learned by anytime algorithm (LS) proposed in Section 4.3 and minimax algorithm (MM) proposed in Section 4.2.

best performance of SLLiPs. Lower values result in higher complexity and consequent overfitting. The error is evaluated on those frames, where inter-frame motion is smaller than the learning range of SLLiPs. Figure 6.5 shows error distribution of MM SLLiPs (blue solid line) and LS SLLiPs (red solid line). λ_0 is denoted by blue dot-dashed line and ϵ_0 by red dot-dashed line. In approximately 8% of cases, MM SLLiPs errors are higher, which is presumably caused mainly by the limited ground truth accuracy and partly by the insufficient training set.

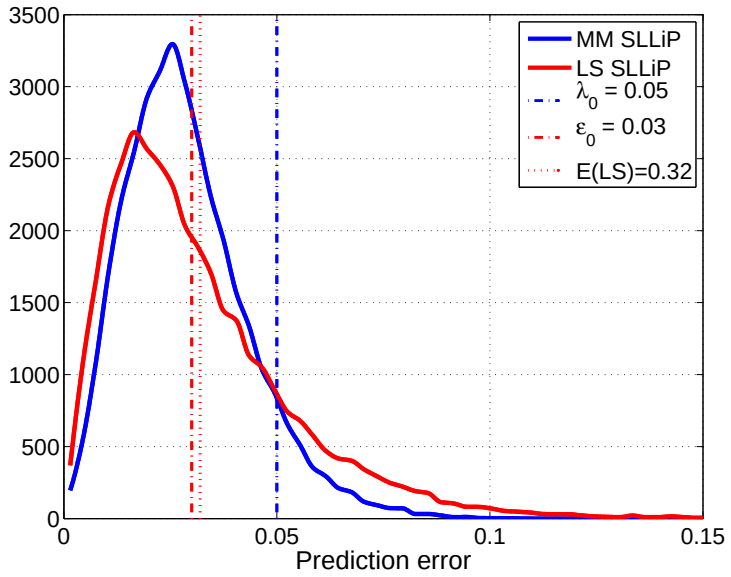


Figure 6.5: **Accuracy analysis:** Comparison of predefined uncertainty region λ_0 of MM SLLiPs (blue dot-dashed line), predefined prediction error ϵ_0 of LS SLLiPs (red dot-dashed line) and true error distribution (blue and red solid lines) on MOUSEPAD sequence.

6.2.3 Comparison of SLLiPs to the state-of-the-art

Table 6.2 compares the NoSLLiP tracker to the state-of-the-art Lowe's SIFT detector [36] (method: SIFT)[2], Lucas-Kanade tracker [37] (method: LK tracker) and Jurie's LS LLiP tracker [33] (method: LLiP LS). All these local motion estimators were combined with the RANSAC, to keep test conditions as similar as possible. SIFT tracking mainly fails in frames with strong motion blur or in frames where the object was very far from the camera. LK tracker, which estimates the local motion at Harris corners, provided quite good results on the frames where the object was far from camera, but its basin of

[2]We use implementation of the SIFT detector downloaded from `http://www.cs.ubc.ca/~lowe/keypoints/`

method	processing [fps]	loss-of-locks	mean-error [%]
SLLiP LS	27.6	17/6935	[1.4, 1.3, 1.1, 1.1]
SLLiP MM	18.9	13/6935	[1.3, 1.8, 1.5, 1.6]
SIFT [36]	0.5	281/6935	[1.6, 1.2, 1.5, 1.4]
LK (IC) tracker [37]	2.6(25)	398/6935	[2.3, 2.2, 2.5, 2.5]
LLiP LS [33]	24.4	1083/6935	[5.9, 6.0, 6.7, 6.7]
LLiP LS [33] half-range	24.2	93/6935	[3.1, 2.3, 2.7, 4.0]

Table 6.2: **Comparison of robustness and accuracy of MM SLLiP, LS SLLiP, LK, LLiP trackers and SIFT detector:** All trackers implemented in MATLAB, SIFT detector implemented in C++. Comparison on MOUSEPAD sequence. Frame-rate of IC algorithm is estimated based on comparison published in [3].

attraction was in many frames insufficient for correct motion estimation, failing for fast motions.

According to the detailed speed comparison published in [3], 6-parameter optimization by Inverse Compositional (IC) algorithm implemented in MATLAB runs approximately ten times faster than the optimization by Forward Additive algorithm used in LK tracker. However, the basin of attraction and sensitivity to noise are the same. Since SLLiP tracker is also implemented in MATLAB, the achieved frame-rates of LK, IC and SLLiP are comparable. Concerning computational complexity of LK, IC, and SLLiP: Computational complexity of one iteration computed on n-pixel template and p-vector of pose parameters by LK is $\mathcal{O}(p^2n + p^3)$ and by IC is $\mathcal{O}(pn + p^3)$. SLLiPs exploit only a small subset of pixels. Since we experimentally verified that approximately $\sqrt{n}$ pixels is used from n-pixel template, the computational complexity of one iteration of SLLiP (i.e., LLiP) is $\mathcal{O}(\sqrt{n}p)$.

Jurie's tracker is a LLiP tracker with the support set equal to the whole template learned by LS method for the same reference points and ranges as optimal SLLiPs. Since a single LLiP tracker does not allow sufficient accuracy on the same range, very high loss-of-lock ratio and low accuracy are reported. If the half-range is used, the higher accuracy is achieved, but the number of loss-of-locks is still significantly higher than with NoSLLiP tracker, mainly due to long inter-frame motions.

6.3 Additional experiments

6.3.1 Empirical properties of MM SLLiPs

In this section, empirically observed properties of LLiPs and SLLiPs are demonstrated.

LLiPs

We summarize our empirically observed properties of the parameters such as complexity, range, uncertainty region, prediction error or length of the optimal sequence. Relation among the complexity, range and uncertainty region of LLiP is given by the manifold in (c, r, λ)-space, see Figure 6.6a for an example. It shows that reasonable complexities are from 40 to 300 pixels. The maximum considered complexity should be maximal possible complexity of the SLLiP (experimentally shown about 2000 pixels) however, we observed that 300 pixels are in most of the cases sufficient, because uncertainty region improvements for the higher complexities are negligible. Range is limited by the size of the object: radius of the support set plus radius of the range must be smaller or equal to the size of the object. If this assumption is violated, some of the support pixels can get out of the object (on the background) and the predictor output is not guaranteed. We usually choose radius of the range 1-2 times higher than the radius of the support set. Depicted manifold shows that the single LLiP with 300 support pixels decreases

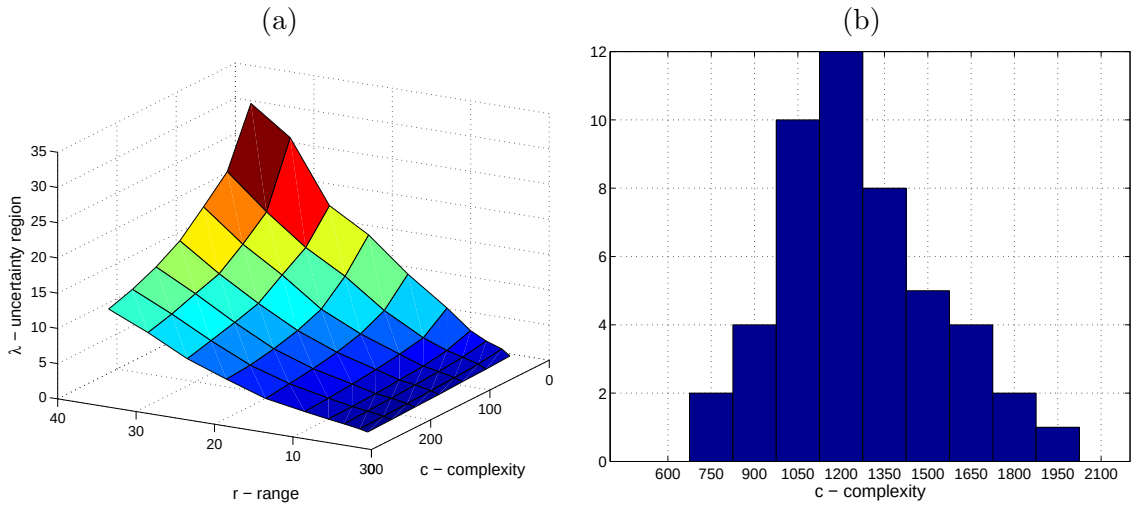


Figure 6.6: **Properties of MM SLLiPs:** (a) Relation among complexity, range and uncertainty region of LLiP. Color codes the size of the uncertainty region. (b) Histograms of SLLiPs complexities.

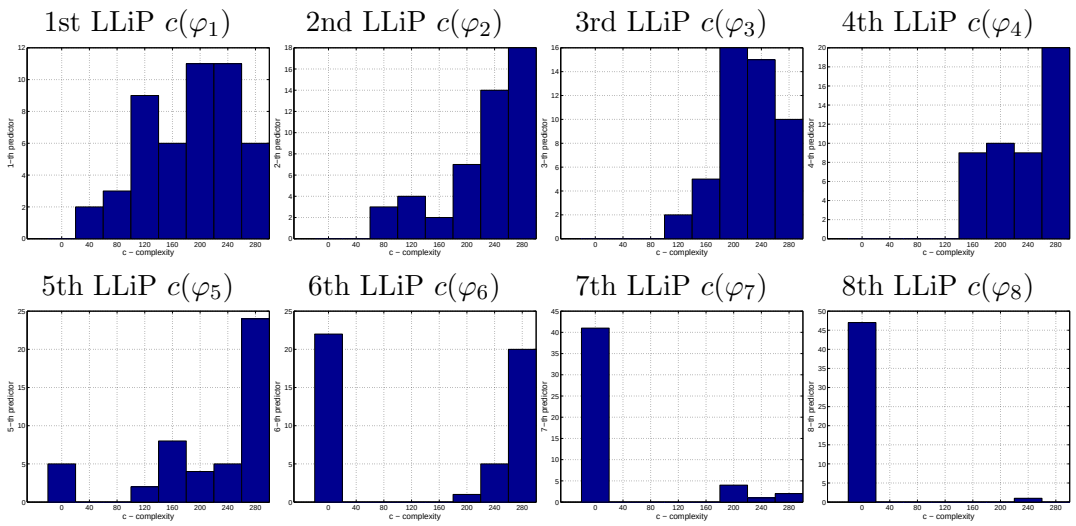


Figure 6.7: **Histograms of LLiPs complexities:** Complexities $c(\varphi_1)$-$c(\varphi_8)$ of LLiPs at positions 1–8 of the optimal SLLiPs.

range approximately 2.5 times. Note, that the shape of the manifold depends on the texture and the complexity of the training examples. For example, if blured images are not included in the training set, LLiP with complexity 300 decreases range more than 5 times. In extremal case of the texture with the constant intensity, the manifold is a plane with the uncertainty region equal to the range. In the other hand, the most convenient texture is the texture, which linearly codes the motion, but this is usually not the case.

SLLiPs

Length of the optimal sequence also depends on the texture. In most of the cases, the optimal sequential predictor consists of 5-8 LLiPs. Complexity of the optimal sequential predictor varies from 700 to 2000 pixels, see Figure 6.6b. Histograms of the complexities of LLiPs in particular positions in the optimal sequence are depicted in Figure 6.7. It shows that further located LLiPs are more likely to have either the maximal complexity or zero complexity (i.e., they are unused).

Figure 6.8 shows some typical patches their (c, r, λ)-space and corresponding SLLiPs. Complexity distribution along the optimal sequence $c(\varphi_i)$-distribution is depicted in the third column. Complexity $c(\Phi)$ of the optimal sequential predictor Φ with $\lambda(\Phi) = 2$ is depicted in the last column. The first patch (first row) required SLLiP with relatively high complexity $c(\Phi) = 1680$, because weak vertical gradients are present.

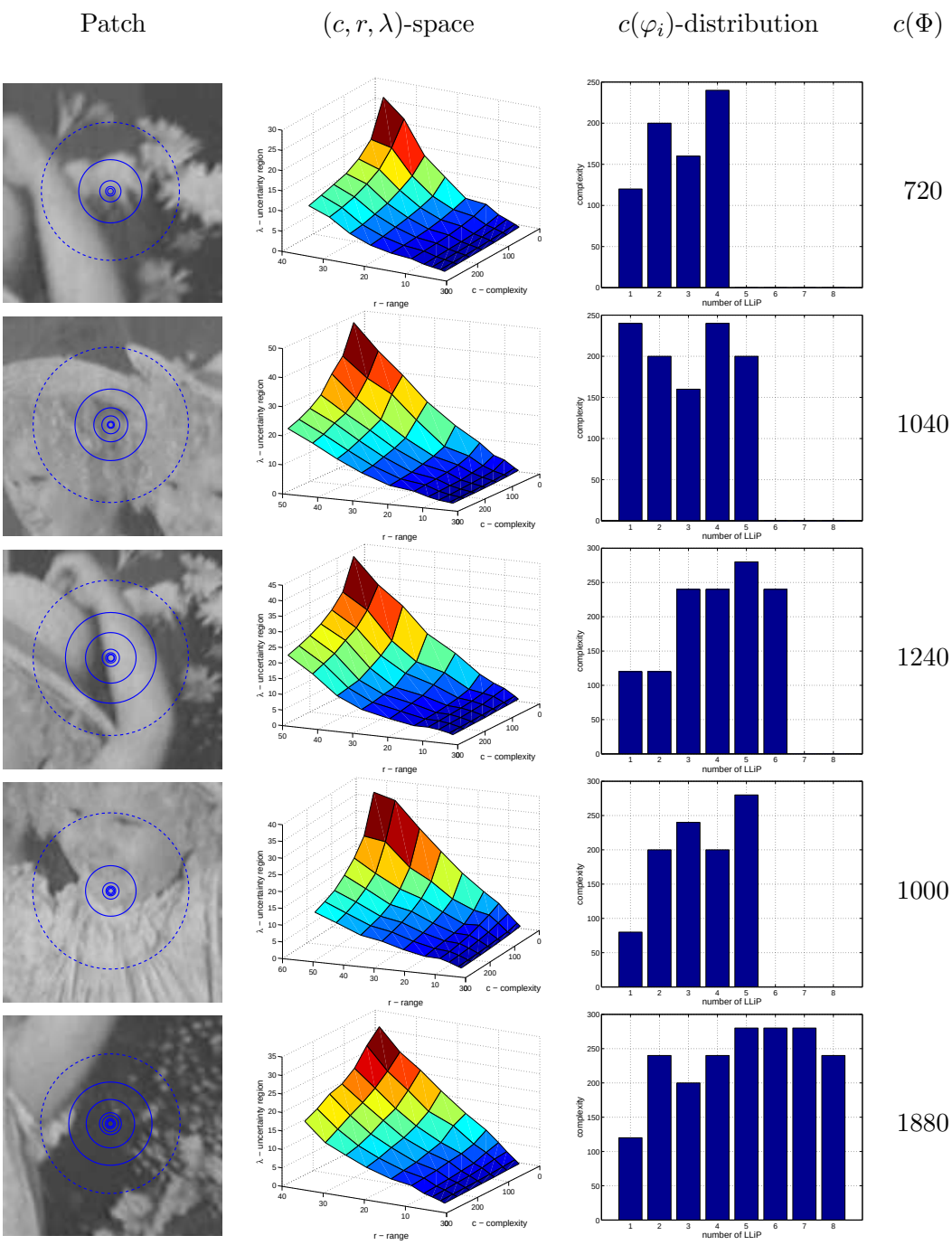
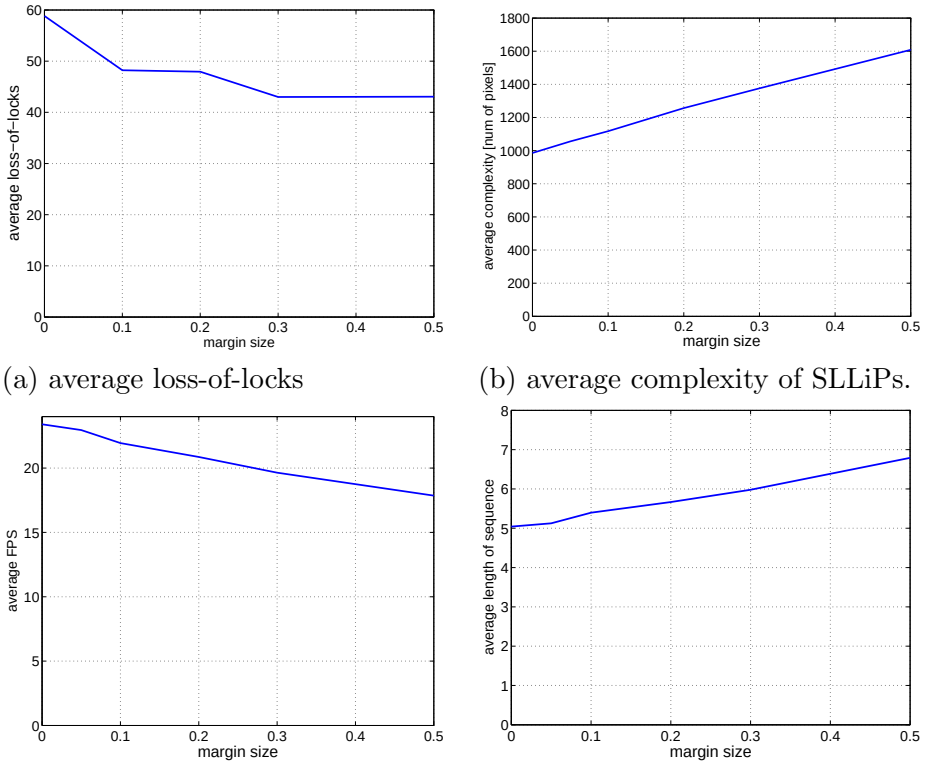


Figure 6.8: **Examples of patches, their (c, r, λ)-space and complexities of LLiPs in the optimal SLLiP:** The most left column contains image data, ranges (blue dashed circles) and corresponding uncertainty regions (blue solid circles). Middle column shows relation among complexity, range and uncertainty region in (c, r, λ)-space. The most right column shows complexity distribution along the optimal sequence. Complexity $c(\Phi)$ of the optimal sequential predictor Φ with $\lambda(\Phi) = 2$.

6.3.2 Influence of robustness margin of MM SLLiPs



(a) average loss-of-locks

(b) average complexity of SLLiPs.

(c) average frame-rate

(d) average number of LLiPs in SLLiP

Figure 6.9: **Robustness analysis:** The higher the margin, the higher the robustness to noise but also the higher the complexity of SLLiPs

We defined SLLiP as a sequence of LLiPs satisfying that the range of every predictor is at least as large as the uncertainty region of its predecessor, i.e., $\forall\, r_{i+1} \geq \lambda_i$, $i = 1 \dots m-1$. We showed in Proposition 4, that the complexity minimization in the learning stage results in equality $\forall\, r_{i+1} = \lambda_i$, $i = 1 \dots m-1$. As a result, whenever the testing data are corrupted by noise, the prediction might not be within the range of the following predictor, which might consequently causes a divergence of the SLLiP. For practical applications, a margin assuring robustness to the noise, is required. In other words, we require $\forall\, r_{i+1} \geq \lambda_i + \gamma$, $i = 1 \dots m-1$ for a non-negative number γ. We claim the higher is the margin γ the higher is the robustness against noise but simultaneously also the higher the complexity of the optimal SLLiPs.

Quantitative robustness evaluation is performed by computing the average number of loss-of-locks as a function of the margin (Figure 6.9a), average complexity of SLLiPs as a function of the margin (Figure 6.9b) and average frame-rate as a function of the margin (Figure 6.9c). The experiment is conducted on a selected sub-sequence with

mousepad (frames 3500-6000), where only each second frame is processed in order to increase inter-frame motion and consequently to achieve a statistically important number of loss-of-locks. The sequence is processed with SLLiPs learned for 6 different margins $[0, 0.05, 0.1, 0.2, 0.3, 0.5]$. Number of loss-of-locks (Figure 6.9a) and frame-rate (Figure 6.9c) are evaluated as an average over 20 processing of the sequence with different starting frame. Complexity (Figure 6.9b) and length of LLiP sequence (Figure 6.9d) are computed as an average over set of 48 learned SLLiPs.

6.3.3 Support set selection by greedy LS algorithm

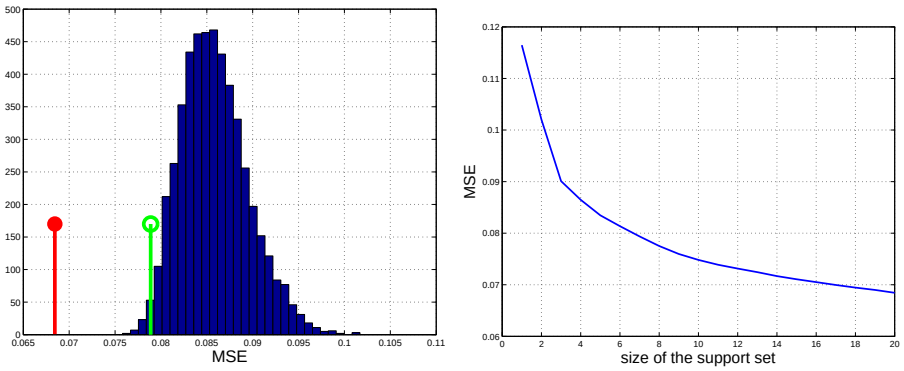


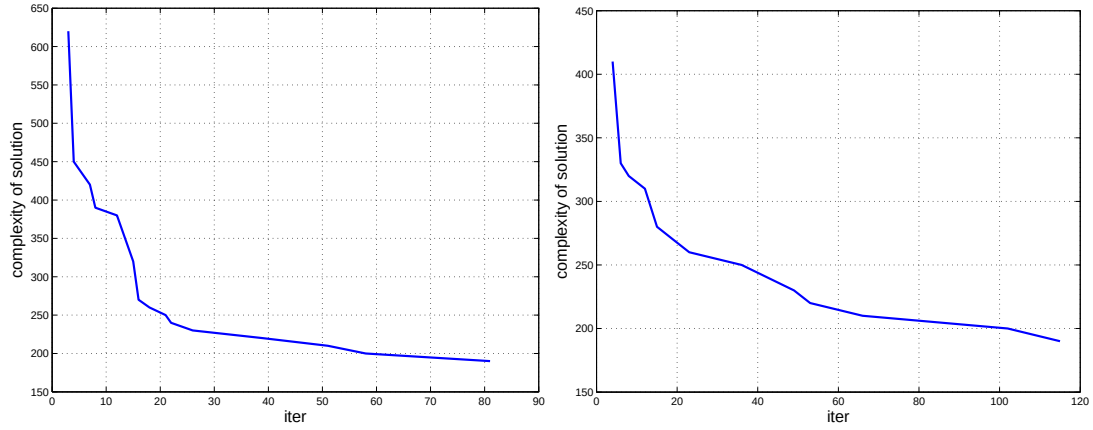
Figure 6.10: **Comparison of greedy LS support set selection and random selection.** Histogram of prediction errors of predictors with the randomly constructed support sets. (a) $C = 20$, $E_{99} = 0.078$ (green empty circle denotes 99%-quantile), $\hat{E} = 0.068$ (red filled circle denotes MSE of the proposed method), (b) MSE as a function of complexity during the incremental construction of the support set.

We compare relative mean square error (MSE) achieved by the greedy LS support set selection algorithm (Algorithm 3) and the MSE achievable by a random support set selection. Figure 6.10a shows the MSE histogram of predictors operating on a randomly selected support sets of the size of 20 pixels. The 99% left quantile of the histogram is depicted by empty green circle. It shows that usage of a randomized sampling instead of the Algorithm 3 would require a prohibitively high number of iterations to achieve the error achievable by the greedy LS algorithm. On the other hand, the error is improved only by 10% with respect to expected value. For low complexities ($C \approx 20$) the algorithm takes only a few seconds, however, the learning time increases exponentially with the complexity.

Figure 6.10b shows MSE as a function of complexity during the incremental construction of the support set. It demonstrates that 99% left quantile of randomly selected support sets is achieved with less than one-half of the support set size. The mean is achievable with less than one-quarter of the support set size.

6.3.4 Anytime learning algorithm: trade-off between complexity learned SLLiPs and learning time

Anytime learning procedure might be time consuming if a set of considered LLiPs is too large. Since a long learning time might not be acceptable for some types of applications, either the set of considered LLiPs or the maximum number of learning iterations have to be restricted. However, the constraint on maximum number of iterations of Algorithm 2 affects the optimality of the found SLLiP. We therefore show average complexity of the best so far found solution as a function of iterations. Figure 6.11 presents this function for the different sets of considered LLiPs. One can see that the learning time could be 2-3 times decreased without significant increase of the solution complexity.



(a) $|C| = 3$ (b) $C = |4|$

Figure 6.11: **Average complexity as a function of iterations of Algorithm** 2: (a) and (b) differs by the size of the set of considered complexities $|C|$. The higher $|C|$ the larger space of considered LLiPs.

Note that there is also another option besides of premature interruption of the learning procedure. The anytime learning algorithm, after a short initialization procedure, provides a solution - SLLiP with higher complexity but defined precision, which can be immediately used for tracking. Since the tracking requires only a fraction of the processing power of an ordinary PC, the learning need not be necessarily terminated and it might be allowed to run in a parallel background thread continuously providing better and better SLLiPs. This principle theoretically allows to start the tracking procedure immediately without any learning using for example Lucas-Kanade tracker and collect training examples automatically. Once pa training set is constructed the learning procedure can run in a parallel thread providing the SLLiPs which continuously replace worse LK trackers. Similar idea based in simple LLiPs was demonstrated in [23].

7 Simultaneous learning of motion and appearance

7.1 Introduction

Visual tracking is often formulated as an iterative motion estimation with possible update of the object appearance. The optional on-line appearance update may allow for longer tracks but also makes the procedure prone to a failure. We propose an algorithm that learns both motion prediction and appearance parametrization from training data.

In Section 2.4, we explained that Cootes et al. [18] proposed a paradigm where both the motion and appearance are projected by the PCA [25] to the lower dimensional space of some parameters. Given an image, the learned linear mapping `H` estimates the parameters which are used for both the template update and motion estimation by the PCA back-projection, see Figure 7.1d. This approach is iterated until covergence is reached.
Observing that

- the mapping between input image $\mathbf{I}$ and the output motion parameters $\mathbf{t}$ is linear, see dashed line in Figure 2.3d denoting the direct route, and
- the mapping between input image $\mathbf{I}$ and appearance parameters $\boldsymbol{\theta}$ is also linear,

we generalized the tracking approach to Figure 2.3e. While the *tracker* φ aligns the model with the current image, the *appearance encoder* γ encodes a current appearance into parameters $\boldsymbol{\theta}$, which adjust the tracker for the current object appearance. Since the learning process estimates both mappings simultaneously, the learned appearance encoder γ projects, in contrast to the PCA, the current object appearance to a manifold, the coordinates of which are the most convenient for the tracker adjustment. In the rest we describe tracking and learning of the *tracking system* depicted in Figure 2.3e.

We restrict φ and γ to be from a class of mappings formed as a linear combination of kernel functions. Consequently, learning of φ, γ minimizing prediction error of φ is a bilinear fitting problem, which is solved by an exact line-search algorithm [26]. Note, that there are two kinds of parameters: motion $\mathbf{t}$, which we want to estimate and appearance $\boldsymbol{\theta}$, which has only the auxiliary meaning. The appearance parameters are not included in the training set and the learning process is unsupervised with respect to them. Their role is automatically determined during the unsupervised learning.

Section 7.2 describes a training set construction, given a labeled sequence. The problem is defined in Section 7.3 and solved in Section 7.4. Experiments demonstrating an improvement to the state-of-the-art and limitations are presented in Section 7.5.

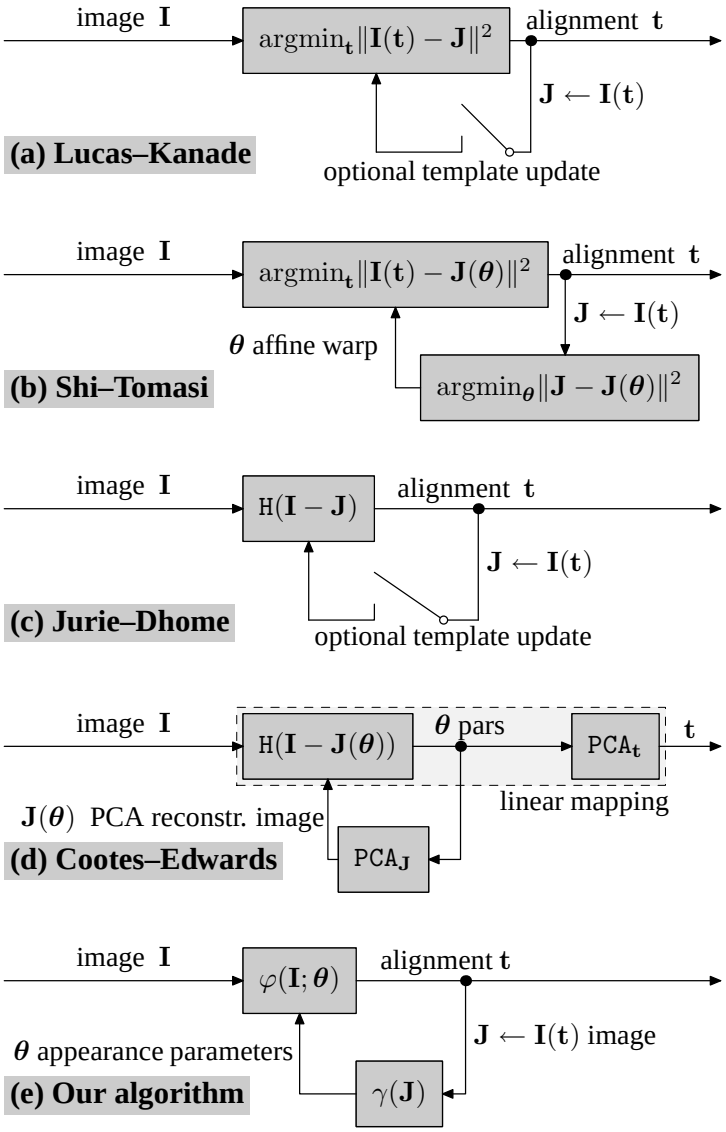


Figure 7.1: **Tracking methods** compensating changes of the object appearance.

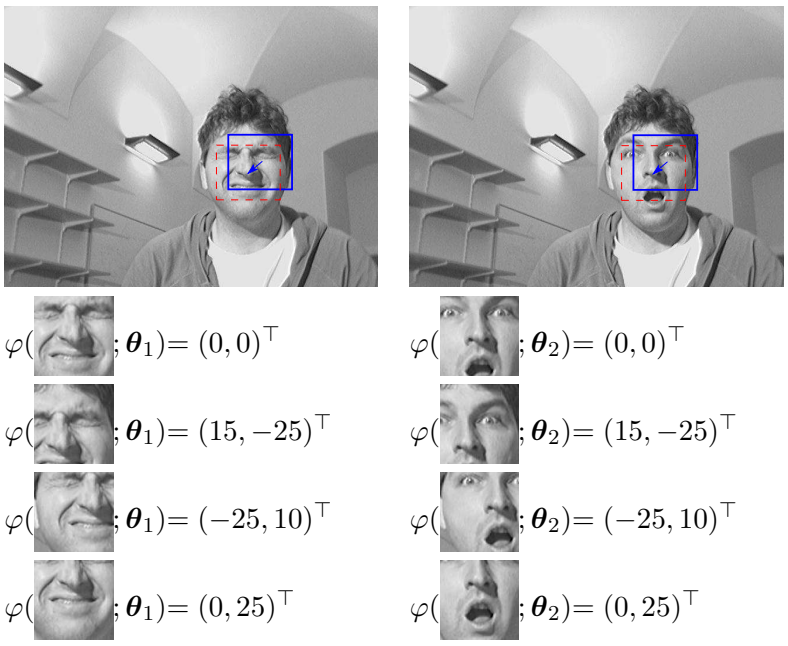


Figure 7.2: **Training set construction:** Image is perturbed by the motion parameters within a predefined range in order to create a set of synthesized examples of observed intensities $\mathbf{I}^i$ and motions $\mathbf{t}^i$. Red dashed square denotes image data observed at translation $\mathbf{t} = (0,0)^\top$, blue square denotes image data observed at translation $\mathbf{t} = (15,-25)^\top$.

7.2 Training set construction

Let us suppose we are given a ground truthed sequence of images, where the position of the object reference point is known. Given a predefined range of motions R within which the tracker is assumed to operate, the training set consists of:

- *tracker input* images $\mathbf{I}^i$, randomly (uniformly) generated within R,
- corresponding *tracker output* motion parameters $\mathbf{t}^i$ (e.g., translations),
- and *appearance encoder input* images $\mathbf{J}^i$, i.e., the images aligned by the tracker with a limited accuracy.

We perturb neighbourhood of reference point in each particular frame of the sequence by motion parameters $\mathbf{t}^i$ randomly generated inside the range, creating the set of synthesized examples of intensities $\mathbf{I}^i$, see Figure 7.2. These examples are column-wise stored in matrices $\mathtt{I} = \left[\mathbf{I}^1 \dots \mathbf{I}^d\right]$ and $\mathtt{T} = \left[\mathbf{t}^1 \dots \mathbf{t}^d\right]$.

The alignment estimated by the tracker is never perfect. Let us assume for a moment, that the accuracy of the alignment by the tracker is known in advance[1]. We perturb neighbourhood of reference points within the range determined by the tracker accuracy creating the set of images $\mathtt{J} = \left[\mathbf{J}^1 \dots \mathbf{J}^d\right]$ entering to the appearance encoder. Ordered triple $(\mathtt{I}, \mathtt{J}, \mathtt{T})$ of such matrices is called a *training set*.

7.3 Optimization problem definition

In our approach, we restrict φ and γ to be from a class of polynomial mappings. As mentioned in Section 4.1.1, the polynomial mapping corresponds to LLiP, therefore appearance encoder γ is further considered to be LLiP with regressor

$$\boldsymbol{\theta} = \gamma(\mathbf{J}) = \mathtt{G}\mathbf{J}, \tag{7.1}$$

which encodes current object appearance from image $\mathbf{J}$ previously aligned by the tracker into parameters $\boldsymbol{\theta}$, to be used in the following frame.

The tracker φ is not a simple polynomial mapping but it is parameterized by the appearance parameters, therefore Parameter Sensitive LLiP (PLLiP) is defined.

Definition 14. Parameter sensitive LLiP *is LLiP with regressor* $\mathtt{H} = (\mathtt{H}_0, \dots, \mathtt{H}_m)$*, which computes motion parameters* $\mathbf{t}$ *from observed image* $\mathbf{I}$ *given current appearance parameters* $\boldsymbol{\theta} = (\theta_1 \dots \theta_m)^\top$ *as follows*

$$\mathbf{t} = \varphi(\mathbf{I}; \boldsymbol{\theta}) = (\mathtt{H}_0 + \theta_1\mathtt{H}_1 + \dots + \theta_m\mathtt{H}_m)\mathbf{I}, \tag{7.2}$$

Definition 15. Tracking system *is ordered pair* $(\mathtt{H}, \mathtt{G})$ *corresponding to tracker (PLLiP) and appearance encoder (LLiP) connected according to Figure 7.1.*

Definition 16. Prediction error $e(\mathtt{H}, \mathtt{G})$ *of tracking system* $(\mathtt{H}, \mathtt{G})$ *on training set* $(\mathtt{I}, \mathtt{J}, \mathtt{T})$ *is*

$$e(\mathtt{H}, \mathtt{G}) = \sum_{i=1}^{d} \left\| \Big(\mathtt{H}_0 + \underbrace{(\mathbf{g}_1^\top \mathbf{J}^i)}_{\theta_1}\mathtt{H}_1 + \dots + \underbrace{(\mathbf{g}_m^\top \mathbf{J}^i)}_{\theta_m}\mathtt{H}_m\Big)\mathbf{I} - \mathbf{t}^i \right\|_2^2. \tag{7.3}$$

[1] In practice, the accuracy has to be estimated by iterating the learning process. However, for the sake of simplicity, it is assumed to be known.

Definition 17. Optimal tracking system *with respect to training set* $(\mathtt{I}, \mathtt{J}, \mathtt{T})$ *is tracking system*

$$(\mathtt{H}^*, \mathtt{G}^*) = \underset{\mathtt{H},\mathtt{G}}{\operatorname{argmin}}\, e(\mathtt{H}, \mathtt{G}). \tag{7.4}$$

The matrices $\mathtt{H}, \mathtt{G}, \mathtt{I}, \mathtt{J}, \mathtt{T}$ have dimensions $(p(m+1)\times n), (m\times n), (n\times d), (n\times d), (n\times p)$, respectively, where

- p is the number of tracked motion parameters (e.g., 2D-translation $\Rightarrow p = 2$),
- n is the number of used pixels,
- m is the number of appearance parameters,
- d is number of training examples.

Computational complexity of the tracking procedure corresponds to the number of all elements in the matrices $(\mathtt{H}, \mathtt{G})$ since the motion in particular frame requires approximately such number of multiplications and additions.

Definition 18. Complexity $C(\mathtt{H}, \mathtt{G}) \in \mathbb{R}$ *of tracking system* $(\mathtt{H}, \mathtt{G})$ *is*

$$C(\mathtt{H}, \mathtt{G}) = p(m+1)n + mn. \tag{7.5}$$

7.4 Learning the tracking system

In this section, we propose an iterative algorithm, which finds optimal tracking system with respect to a training set, i.e., it solves problem (7.4). This is an unconstrained optimization problem, where bilinear function is fitted in the least squares sense into a high dimensional data.

We show later that problem (7.4) 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*[2], which successively computes more and more exact solutions of (7.4) for fixed $\mathtt{H}$ and $\mathtt{G}$).

In the very beginning of the learning process, the appearance encoder $\mathtt{G}$ is randomly initialized. Naturally, it does not provide any reasonable appearance parameters and its influence is negligible. Given random matrix $\mathtt{G}^0$, minimum of criterion function (7.3) over $\mathtt{H}$ has a closed-form solution $\mathtt{H}^0$. Given $\mathtt{H}^0$, the minimum over $\mathtt{G}$ has also a closed-form solution $\mathtt{G}^1$. In that manner, the error is iteratively minimized until a stopping condition is satisfied. In our implementation the learning stops if a relative error difference is smaller than some threshold ϵ. Number of appearance parameters m, desired complexity

[2] An exact line-search method finds the length of step which minimizes a criterion in a descent direction, see [26] for details.

C (or equivalently number of pixels n), learning threshold ϵ and a training set $(\mathtt{I}, \mathtt{J}, \mathtt{T})$ (optionally range of the tracker) are the only inputs to the learning procedure, which is summarized in Algorithm 5.

1. Randomly initialize matrix $\mathtt{G}^0$ and set number of current iteration $k = 1$.
2. Find $\mathtt{H}^k = \operatorname{argmin}_{\mathtt{H}} e(\mathtt{H}, \mathtt{G}^{k-1})$.
3. Find $\mathtt{G}^k = \operatorname{argmin}_{\mathtt{G}} e(\mathtt{H}^k, \mathtt{G})$.
4. Recompute error $e^k = e(\mathtt{H}^k, \mathtt{G}^k)$.
5. If $\frac{e^k - e^{k-1}}{e^k} < \epsilon$ stop, otherwise $k = k + 1$ and goto 2.

Algorithm 5 Exact line-search algorithm for problem (7.4)

In order to derive closed-form solution of step 2, we rewrite criterion function (7.3) as follows:

$$\begin{aligned} e(\mathtt{H}, \mathtt{G}) &= \Big\| \mathtt{H}_0 \mathtt{I} + [\mathtt{H}_1 \dots \mathtt{H}_m] \Big((\mathbf{1}_m \otimes \mathtt{I}) \bullet (\mathtt{GJ} \otimes \mathbf{1}_n) \Big) - \mathtt{T} \Big\|_F^2 \\ &= \left\| \mathtt{H} \left[(\mathbf{1}_{m+1} \otimes \mathtt{I}) \bullet \left(\begin{bmatrix} \mathbf{1}_d^\top \\ \mathtt{GJ} \end{bmatrix} \otimes \mathbf{1}_n \right) \right] - \mathtt{T} \right\|_F^2 = \left\| \mathtt{HA} - \mathtt{T} \right\|_F^2, \end{aligned} \tag{7.6}$$

where

$$\mathtt{A} = (\mathbf{1}_{m+1} \otimes \mathtt{I}) \bullet \left(\begin{bmatrix} \mathbf{1}_d^\top \\ \mathtt{GJ} \end{bmatrix} \otimes \mathbf{1}_n \right) \tag{7.7}$$

is $(m+1)n \times d$ matrix. If the training set has

$$d \geq (m+1)n \tag{7.8}$$

independent samples. The closed-form solution of step 2 is

$$\mathtt{H}^k = \mathtt{TA}^+, \tag{7.9}$$

where $\mathtt{A}^+$ denotes pseudo-inverse of $\mathtt{A}$. Note, that condition (7.8) may not assure a reasonable behaviour on testing data. It sets the lower bound. However, we usually generate five or ten times more training examples to make the tracker robust.

In order to derive closed-form solution of the step 3, we rewrite the criterion function (7.3) as follows:

$$e(\mathtt{H}, \mathtt{G}) = \left\| \mathbf{g}^\top \mathtt{B} - \mathtt{C} \right\|_F^2, \tag{7.10}$$

where

$$\begin{aligned}
\mathbf{g}^\top &= [\mathbf{g}_1^\top \dots \mathbf{g}_m^\top], \\
\mathtt{B} &= \begin{bmatrix} \mathtt{I} \bullet \mathbf{h}_{11}^\top \mathtt{J} \otimes \mathbf{1}_n & \dots & \mathtt{I} \bullet \mathbf{h}_{1p}^\top \mathtt{J} \otimes \mathbf{1}_n \\ \vdots & \ddots & \vdots \\ \mathtt{I} \bullet \mathbf{h}_{m1}^\top \mathtt{J} \otimes \mathbf{1}_n & \dots & \mathtt{I} \bullet \mathbf{h}_{mp}^\top \mathtt{J} \otimes \mathbf{1}_n \end{bmatrix}, \\
\mathtt{C} &= \left[\mathbf{t}_1^\top - \mathbf{h}_{01}^\top \mathtt{J} \dots \mathbf{t}_p^\top - \mathbf{h}_{0p}^\top \mathtt{J} \right],
\end{aligned}$$

where $\mathbf{h}_{mp}^\top$ denotes a p-th row of $\mathtt{H}_m$. The closed-form solution of step 3 is

$$\mathbf{g}^\top = \mathtt{C}\mathtt{B}^+ \text{ reshaped to } \mathtt{G}^k. \tag{7.11}$$

Since the matrix $\mathtt{B}$ has dimension $mn \times dp$, its pseudo-inverse requires

$$dp \geq mn \tag{7.12}$$

Notice, that this condition is clearly weaker than the condition (7.8) for every $p \geq 1$, therefore it need not be considered any further.

7.4.1 Convergence of Algorithm 5

We observed, that the proposed algorithm converges to the solutions, which are associated with the same criterion value. If criterion (7.4) was a convex or quasi-convex function it would be simple to prove that Algorithm 5 would converge to a global minimum. However, the criterion is neither convex nor quasi-convex. We propose conjecture, that every local minimum of the criterion is simultaneously global. Using MAPLE, we analytically prove that the conjecture is true, but we were not able to generalize it for arbitrary number of variables. In this proof, we equal the criterion gradient to zero and symbolically solve the algebraic system. The solution consists of a few manifolds but only one of them has semidefinite Hessian, i.e., creates local minima. This solution is substituted to the criterion showing that the criterion is constant along it.

In Section 7.5.3 the convergence is verified experimentally. It is shown that every local minima of criterion (7.4) within which the algorithm is trapped is global, i.e., the same criterion value is achieved every time and the solution is independent of the algorithm initialization.

7.5 Experiments

Three experiments are presented in this section. The first experiment (Section 7.5.1) compares *simple tracking system* which consists only from a single LLiP and the proposed *parameter sensitive* tracking system. It shows that the parameter sensitive system achieves significantly smaller prediction error if the nature of the appearance changes

allows some reasonable parameterization. The second experiment (Section 7.5.2) demonstrates some interesting properties of the appearance encoder. The third experiment (Section 7.5.3) shows the convergence properties of Algorithm 5.

7.5.1 Results on real sequences

In this experiment, we compare prediction error of simple tracking system and parameter sensitive tracking system of the same complexity. Note, that the simple tracking system is a special case of the parameter sensitive tracking system. Just for clarity we state that: simple tracking system estimates the motion parameters by LLiP with regressor $\mathtt{H}_s$. It is learned given a training set $(\mathtt{I}_s, \mathtt{T}_s)$ as follows:

$$\mathtt{H}_s = \mathtt{T}_s \mathtt{I}_s^+. \tag{7.13}$$

and its complexity

$$C_s = p_s n_s \tag{7.14}$$

is number of elements in $\mathtt{H}_s$.

Training/testing errors and complexities for different objects are presented in Table 7.1. The first three objects CUP, BASIL and SIBIL are taken from a British sitcom Fawlty Towers, see the first three rows in Figure 7.3. The other three object demonstrate variability of the appearance changes which can be cope by the tracker. In particular, fourth object HEAD 1 is a human head, where different expressions and illuminations influence its appearance, see fourth and fifth row in Figure 7.3. The fifth object HEAD 2 is human head, the appearance of which changes due to out-of-plane rotations, see third row of Figure 7.4. The object FLOWER is a flower with appearance strongly affected by non-trivial variable illumination and non-rigid deformations, see sixth row in Figure 7.3.

Notice that in all cases the prediction error of PLLiP was significantly smaller than the error LLiP despite of its lower complexity. Loss-of-lock events are denoted by "X". We used 5000-20000 training examples generated from 5-50 training images. Testing is performed on the sequences with 100-400 frames. The tracker estimated only translation ($p = 2$) in the range within the radius of 20 pixels, other degrees of freedom were represented by 1-5 appearance parameters. Non-optimized Matlab implementation of the learning procedure performing 10-40 iterations requires about 20-300 seconds on an average machine (1xK8 3200+ MHz). Note, that the learning time mainly depends on the number of training examples and the number of appearance parameters. The motion estimation time in the tracking procedure is negligible (say smaller than 1ms) in contrast to the time required for image capturing and its visualisation.

We first discuss results from the Fawlty Towers sequences, where only one appearance parameter is used and the complexity of the parameter sensitive tracking system is two times smaller than the complexity of the simple tracking system. Despite of the lower complexity the error is smaller about approximately 30%. In the CUP experiment (first

[1]Medical data presented in the last row are provided by Dr. Utz Kappert, Department of Cardiac Surgery, Heart Center Dresden University Hospital at the University of Technology Dresden.

Figure 7.3: **Objects**[1] **with variable appearance** caused by illumination, non-rigid deformations and self-occlusions.

Object	Parameter sensitive				Simple		
	$\text{Error}_{\text{train}}$	$\text{Error}_{\text{test}}$	C	m	$\text{Error}_{\text{train}}$	$\text{Error}_{\text{test}}$	C_s
CUP	5.1	5.3	605	1	7.2	X	1352
BASIL	4.9	5.0	605	1	7.0	7.8	1352
SIBIL	4.9	7.4	605	1	7.0	X	1352
HEAD 1	2.7	3.4	1859	2	6.4	8.0	1922
HEAD 2	5.0	5.5	1248	2	7.5	8.5	1300
HEAD 2	4.3	4.5	1716	3	7.3	8.2	1860
FLOWER	3.3	3.6	2873	5	4.0	4.3	3362

Table 7.1: **Comparison of PLLiPs and LLiPs:** Prediction error is presented in pixels and the complexity follows definitions (7.5) and (7.14).

row in Table 7.1), the simple tracker lost the target during the fast motion (see blured image in Figure 7.3 4th row, 3rd column). In the BASIL experiment (second row in Table 7.1), both trackers succeeded in tracking however, the parameter sensitive tracker was more accurate. In the third experiment (third row in Table 7.1), the trackers learned on BASIL were used for tracking of SIBIL. While the simple tracker lost BASIL in the very beginning of the testing sequence the parameter sensitive tracker succeeded in tracking. Of course, the achieved error was higher.

The other three experiments demonstrates appearance changes which can be efficiently compensated by the appearance encoder. While in the HEAD 1 experiment (fourth row in Table 7.1), the error achieved by the proposed tracker is more than two times smaller, in the remaining experiments the improvement is not as significant. In the HEAD 2 experiment (fifth and sixth row in Table 7.1), small out-of-plane rotations are presumably interchangeable with translations, which consequently do not allow a unique interpretation of the observed image data. The rotation may be misinterpreted as a translation, and the image entering to the appearance encoder is not correctly aligned. Incorrect alignment yields incorrect appearance parameters and the error increases. Still, even in the worst case, the adaptive tracker did not produce worse results than the static one. In the FLOWER experiment (seventh row in Table 7.1), the difference between the parameter sensitive and static tracker is the smallest (about 20%) because the appearance changes are chaotic. Many mutual self-occlusions and shadows are casted by waving leaves, which does not allow for a reasonable parameterization.

7.5.2 Properties of the appearance encoder

This experiment demonstrates, appearance parameters estimated by the learned appearance encoder reasonably represent reasons, which caused the appearance changes. First row of Figure 7.4 shows a sequence, where human head turns around. Appearance parameters obtained by parameter encoder are depicted in the second row of Figure 7.4.

Notice, that these parameters naturally correspond almost exactly to the 3D rotation, without any explicit knowledge about that evidence. It also demonstrates that if two frames are close each other in a time line, the corresponding parameters are also close each other, which consequently means that parameters do not suddenly change its value without a reasonable change of the object appearance.

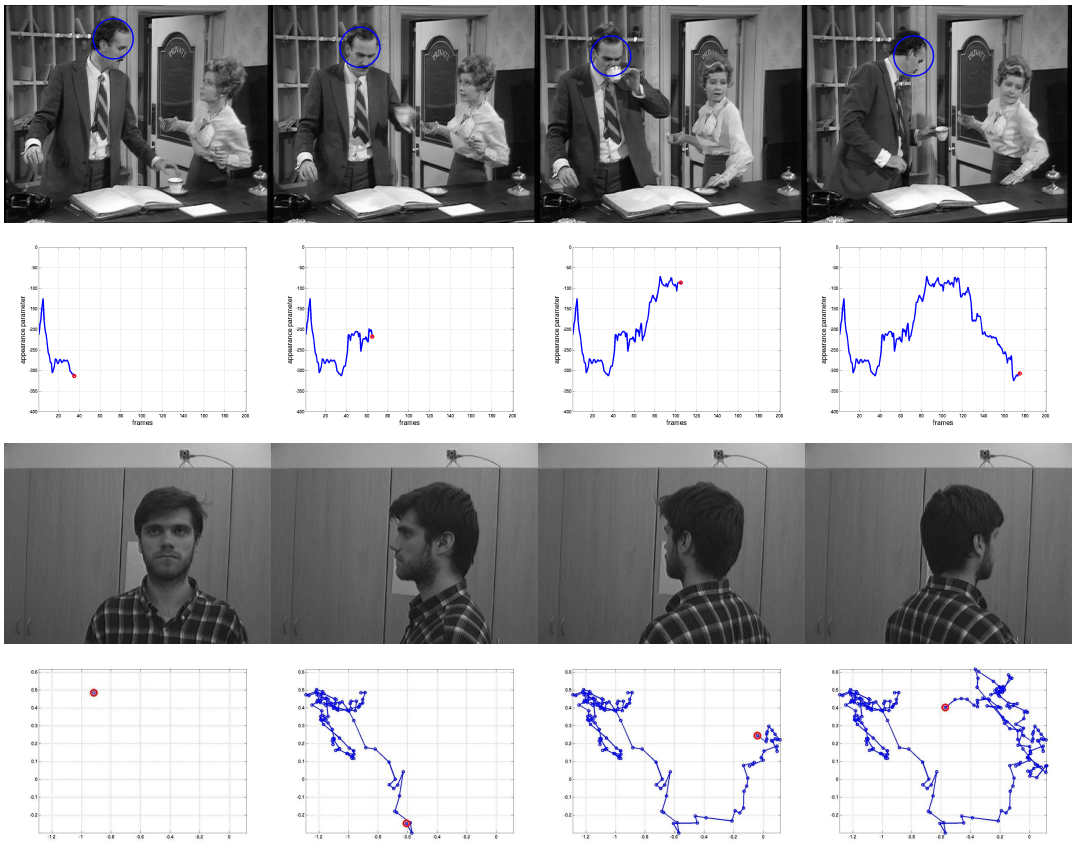
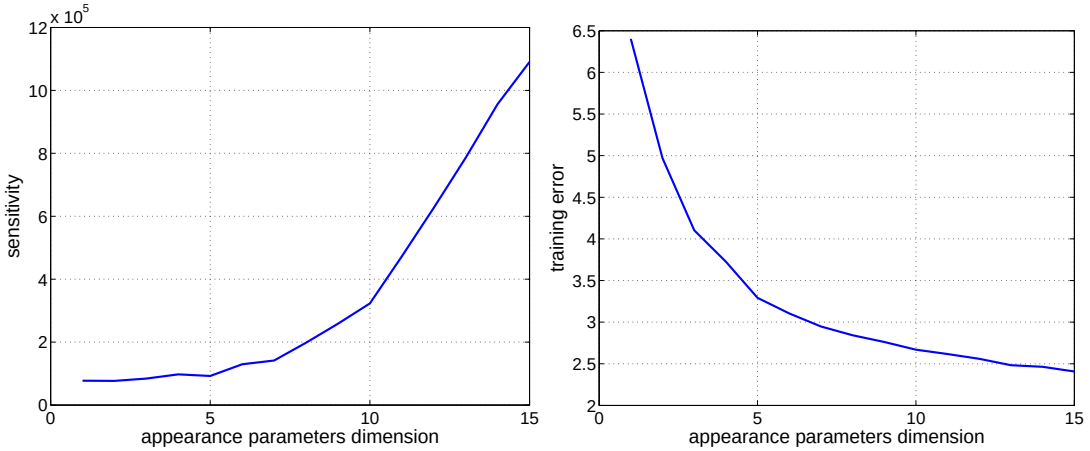


Figure 7.4: **Values of appearance parameters:** Pose parameters are only translations, the other degrees of freedom are modeled as appearance changes. Selected frames from sequences (first and third row) and the corresponding output of one-dimensional (second row) and two-dimensional (fourth row) appearance encoder. The red circle denotes the current appearance parameters, the blue line shows history.

We further study sensitivity of the tracker with respect to appearance variances. Let assume the learned tracker with coefficients $\mathtt{H}(\boldsymbol{\theta}) = \mathtt{H}_0 + \theta_1^i \mathtt{H}_1 + \cdots + \theta_m^i \mathtt{H}_m$ and a testing sequence as given. We define the tracker sensitivity as

$$\sum_i \|\Delta \mathtt{H}(\boldsymbol{\theta}^i)\|_F^2,$$

where $\boldsymbol{\theta}$ are appearance parameters in frame i and $\Delta\mathtt{H}(\boldsymbol{\theta}^i)$ is difference in tracker coefficients in two consecutive frames. In Figure 7.5a, the sensitivity is depicted as a function of appearance parameters dimensionality. We can see that sensitivity is proportional to the dimensionality. Note that the lower sensitivity means the higher robustness. However, prediction error is a decreasing function of the dimensionality, see Figure 7.5b. It means that there is a trade-off between the sensitivity and the error.



(a) Prediction sensitivity (b) Prediction error

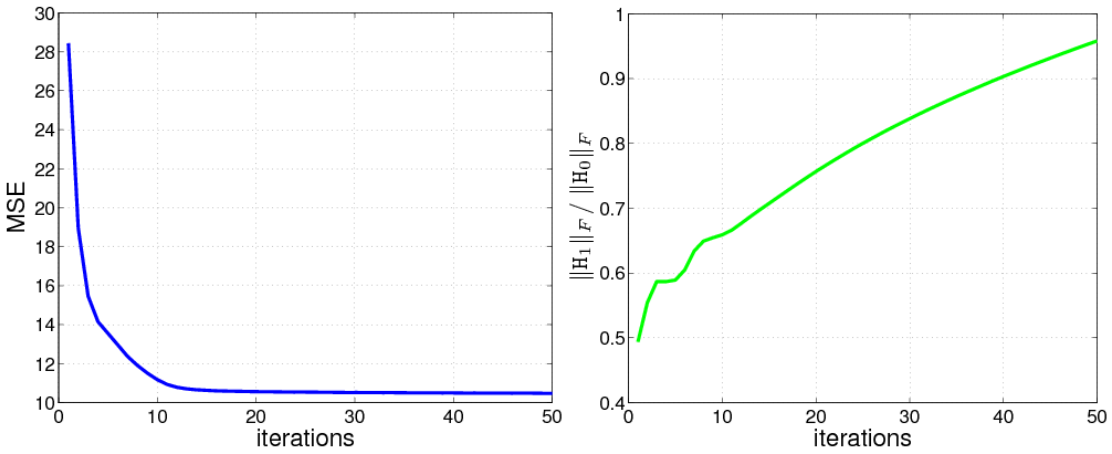
Figure 7.5: **Sensitivity analysis:** Trade-off between the sensitivity and error.

The learning procedure starts at randomly initialized state $\mathtt{G}^0$. Since appearance encoder $\mathtt{G}^0$ does not provide any reasonable appearance parameters, the parameter sensitive part of the tracker $(\mathtt{H}_1 \dots \mathtt{H}_m)$ is useless and ratio $\|\mathtt{H}_1\|_F^2 / \|\mathtt{H}_0\|_F^2$ is very low. During the learning process, significance of the appearance encoder $\mathtt{G}^k$ increases, which mostly results in an increase of the parameter-sensitive part of the tracker. In order to demonstrate it, we outline the ratio $\|\mathtt{H}_1\|_F^2 / \|\mathtt{H}_0\|_F^2$ of parameter sensitive $\mathtt{H}_1$ and non-sensitive $\mathtt{H}_0$ part of the tracker $\mathtt{H}$ as a function of iterations, see Figure 7.6.

7.5.3 Convergence of Algorithm 5

We study convergence of Algorithm 5 in this section. It is experimentally shown that every local minima of criterion (7.4) within which the algorithm is trapped is global, i.e., the same criterion value is achieved every time.

In the experiment, Algorithm 5 was one thousand times randomly initialized from uniform distribution and 150 iterations on 5000 training examples generated from sequence Head 1 were computed. Covariance of achieved Mean Square Errors (MSE) is 3.4×10^{-3}. See Figure 7.7 for twenty convergence examples. From detail depicted in Figure 7.7b, we conclude that 20 iterations are for most of the initializations sufficient.



(a) Mean square error (b) Ratio $\|\mathtt{H}_1\|_F^2/\|\mathtt{H}_0\|_F^2$

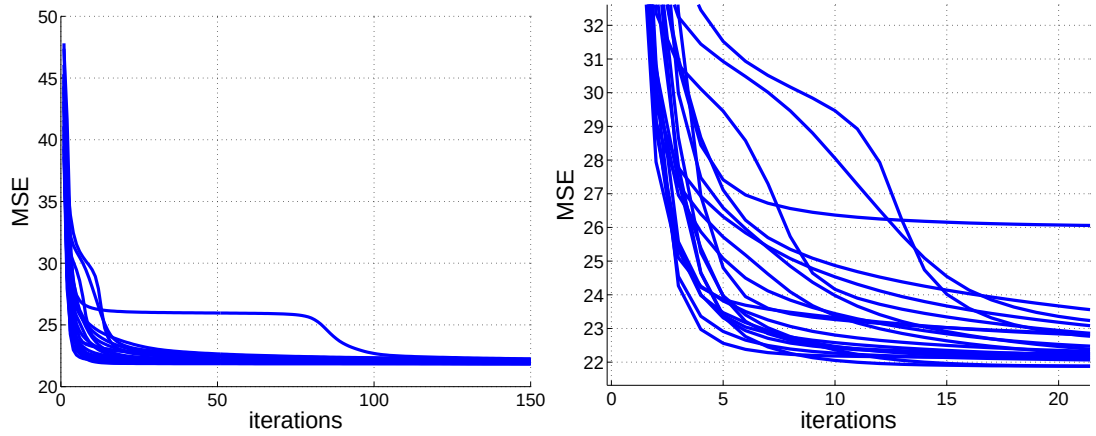
Figure 7.6: **Iterative optimization of** $\mathtt{H}, \mathtt{G}$: The learning procedure starts at randomly initialized state $\mathtt{H}^0, \mathtt{G}^0$. Since $\mathtt{G}^0$ does not provide any reasonable appearance parameters, the parameter sensitive part of tracker $(H_1 \dots H_m)$ is useless. During the learning process, significance of the appearance encoder $\mathtt{G}^k$ increases, which mostly results in increase of a sensitive part of the tracker. In order to demonstrate it, we outline ratio of parameter non-sensitive and sensitive part of the tracker $\mathtt{H}$ as a function of iterations. Note, that the ratio also converges to a steady value, but it takes significantly higher number of iterations.

7.6 Summary

We proposed a learnable tracking procedure suitable for objects with significantly varying appearance. The proposed system consists of tracker φ, which aligns a model with the current observation and appearance encoder γ, which transforms the observed appearance to the parameters adjusting the tracker behavior. Restricting φ and γ to be from a polynomial class, the simultaneous learning of φ and γ is formulated as Least Squares (LS) bilinear fitting problem. The problem is solved by an exact line-search algorithm, the convergence of which is geometrically verified.

Since the learning process estimates both mappings simultaneously, the learned parameter encoder γ projects the current object appearance to a manifold. The coordinates of the manifold are the most suitable for the tracker adjustment. As a result, we are able to track the objects without any necessity of a geometrical explanation of the observed appearance changes.

We have experimentally verified the method on three challenging sequences which exhibit strong variable illumination, non-rigid deformations and self-occlusions. The results are compared to the parameter insensitive tracking system, which is shown to be a special case of our approach. We demonstrate that the error of the parameter insensitive



(a) convergence of Algorithm 5 (b) detail

Figure 7.7: **Covergence analysis of the learning procedure:** Twenty convergence examples of randomly initialized algorithm.

tracking system with the same computational complexity is significantly higher 20-100%. Since the tracking means only a few multiplication, the time required for the motion estimation ($\leq$ 1ms) is negligible with respect to the time required for its visualisation ($\approx$ 10ms).

8 Adaptive parameter optimization for real-time tracking

8.1 Introduction

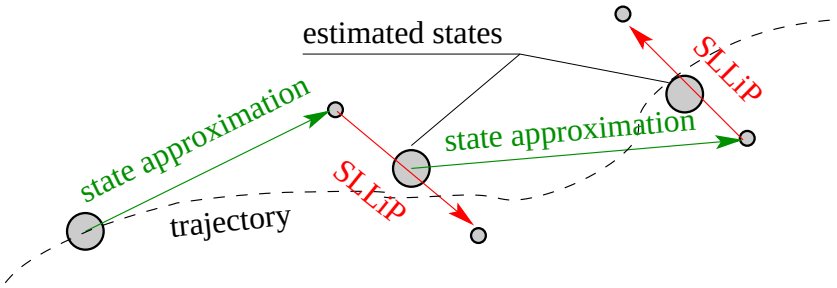


Figure 8.1: **Tracking with the motion model:** the state is first approximated from its previous state(s) by a motion dynamic model and then predicted from image data by SLLiP. Estimated state and predictor parameters are estimated by Kalman filtering.

In real-time tracking application, frames are coming in very short fixed time intervals. A new frame is available each 30 ms (or even faster). However, a very precise motion estimation in a current frame may require a longer time. Consequently, we face the decision, whether the incoming frame should be rejected and the longer time should be spent with the current frame or whether the accuracy in the current frame is sufficient and we should start to process the incoming frame. Such decision has the crucial influence on the resulting robustness and accuracy of the tracking procedure. In this chapter, we study how to adaptively set up the time and range of the SLLiP in order to minimize the tracking error, subject to a loss-of-lock[1] probability constraint.

Let us suppose, we are given a SLLiP $\Phi = (\varphi_1, \ldots, \varphi_m)$, which estimates object motion by a sequence of m different LLiPs $\varphi_i, i = 1 \ldots m$. SLLiP estimates object motion within the range of its first LLiP and with the accuracy of its last LLiP. Any subsequence of the SLLiP can be viewed as another faster SLLiP with smaller range and/or accuracy. Hence, the number of ignored LLiPs in the beginning of the sequence determines the range and the number of ignored LLiPs in the end of the sequence determines accuracy.

Let us also suppose that a motion model, which defines state transition, is available.

[1] Loss-of-lock is understand as the event where tracker error is larger than its range.

It means that the motion model approximates the object state in the following frame just from its current and previous states (i.e., without any image data).

Given such model, the current state of the object is estimated as follows (see also Figure 8.1): (i) state is approximated by the motion model from the dynamic and previous states (green arrow), (ii) the state is re-estimated by the SLLiP initialized from the approximation (red arrow), (iii) The state is refined from both estimates by the Kalman filter [34]. Notice that the motion model error in the following frame is proportional to the time spent by SLLiP in the current frame, because the longer time spent, the more available frames ignored and the less predictable the object state in the next processed frame.

In order to set up the length of the sub-sequence, we derive covariance of the state error in the following frame as a function of the number of used LLiPs, see dark green line in Figure 8.2. Then the number of LLiPs which minimize the motion model error is used for the motion estimation. Such setting minimizes the error in the future frame and leads to minimum error of a steady solution. Furthermore, the estimated probability distribution of the motion model error allows us to determine the following range, i.e., the number of ignored LLiPs in the beginning of the SLLiP.

Since the true motion model error in the following frame is not known in advance, we define the probability of loss-of-lock event as the probability that the range of the first used LLiP is smaller than the motion model error in the following frame estimated by the Kalman filter [34], see brown probability distribution in Figure 8.2. Next the first used LLiP in the sequence is determined as the LLiP assuring that the probability of loss-of-lock is smaller then a predefined upper bound. More formally: we set the number of used LLiPs and the first LLiP in each frame in order to minimize state error of a steady solution, subject to an upper bound of the loss-of-lock probability.

8.2 Problem definition

Since the number of used LLiPs directly corresponds to the processing time of one frame, we replace the number of LLiPs by time t_k spent in k-th frame; similarly first not-ignored LLiP is replaced by range r_k. Let us suppose, we are able to express the probability distribution $\mathcal{P}_{k+1}(\mathbf{q}; t_k, r_k)$ of the state estimation error $\mathbf{q}$ in frame $k+1$ for values of time and range (t_k, r_k) used in k-th frame.

If (t^*_{k-1}, r^*_{k-1}) is a setting used in a previous frame, then probability of loss-of-lock in the current frame is function of the initial range r_k of currently used SLLiP:

$$P_{\text{loss-of-lock}}(r_k) = 1 - \int_{-r_k}^{r_k} \mathcal{P}_{k+1}(\mathbf{q}; t^*_{k-1}, r^*_{k-1}) \mathrm{d}\mathbf{q}, \tag{8.1}$$

see for example Figure 8.3a.

During the tracking, we minimize trace of the covariance matrix of the motion error distribution in the following frame $\operatorname{trace}\left(\operatorname{cov}_{\mathbf{q}}\left(\mathcal{P}_{k+1}(t_k, r_k)\right)\right)$. This measure comprises contributions of the SLLiP accuracy in the current frame and the motion model error

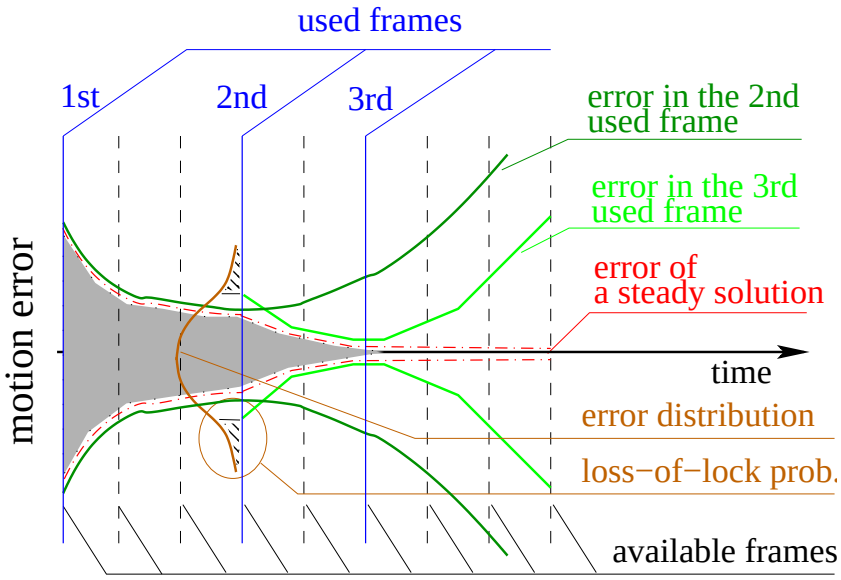


Figure 8.2: **Adaptive optimization of the range and accuracy:** In each frame, the number of used LLiPs and the first LLiP in a SLLiP are estimated in order to minimize error in the following frame given a loss-of-lock probability constraint. Such process also minimize error of a steady solution.

in the next frame. We estimate parameters (t_k^*, r_k^*) which minimize the error in the following frame and satisfy constraint that $P_{\text{loss-of-lock}}(r_k) < \delta$, which leads to the following constrained optimization problem:

$$\begin{aligned}(t_k^*, r_k^*) \quad &= \quad \underset{t_k, r_k}{\operatorname{argmin}} \left\{ \operatorname{trace} \left(\operatorname{cov}_{\mathbf{q}} \left(\mathcal{P}_{k+1}(\mathbf{q}; t_k, r_k) \right) \right) \right\} \\ &\text{subject to:} \quad 1 - \int_{-r_k}^{r_k} \mathcal{P}_k(\mathbf{q}; t_{k-1}^*, r_{k-1}^*) \, \mathrm{d}\mathbf{q} < \delta.\end{aligned} \tag{8.2}$$

The smaller the range r_k^*, the more accurate SLLiP with the fixed time however, the higher $P_{\text{loss-of-lock}}(r_k^*)$. Therefore trace $\left(\operatorname{cov}_{\mathbf{q}} \left(\mathcal{P}_{k+1}(\mathbf{q}; t_k, r_k) \right) \right)$ is non-decreasing function of r_k and we set up r_k^* as tight as possible to its upper bound, i.e., $P_{\text{loss-of-lock}}(r_k^*) = \delta$. In consequence, problem (8.2) is decomposed to two separated tasks:

1. Find the smallest r satisfying the loss-of-lock constraint

$$r_k^* = \underset{r_k}{\operatorname{argmin}} \left\{ r_k \middle| \int_{-r_k}^{r_k} \mathcal{P}_k(\mathbf{q}; t_{k-1}^*, r_{k-1}^*) \, \mathrm{d}\mathbf{q} > 1 - \delta) \right\} \tag{8.3}$$

2. Find the time t minimizing error in the next frame for the fixed r_k^*

$$t_k^* = \underset{t_k}{\operatorname{argmin}} \left\{ \operatorname{trace} \left(\operatorname{cov}_{\mathbf{q}} \left(\mathcal{P}_{k+1}(\mathbf{q}; t_k, r_k^*) \right) \right) \right\}, \tag{8.4}$$

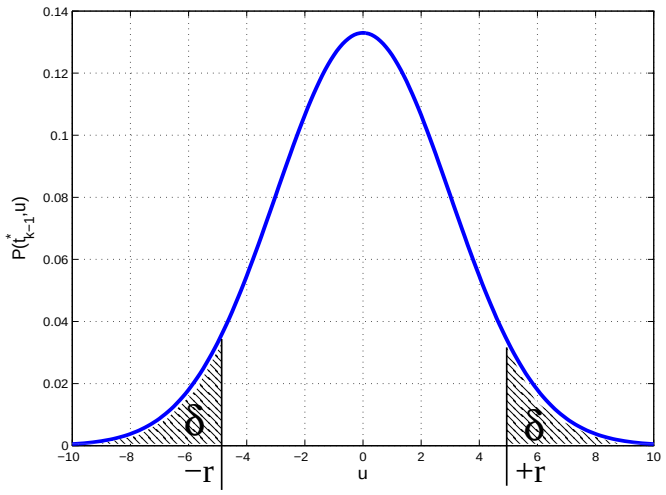


Figure 8.3: **Probability distribution of the motion model error:**, range denoted by r and probability of loss-of-lock event denoted by δ.

8.3 Estimation $\mathcal{P}_{k+1}(t_k, r_k)$ using Kalman filter

In this section, we derive covariance of the probability distribution $\mathcal{P}_k$ by adopting the Kalman filter principle [34]. As we have shown, problem (8.2) can be decomposed into two independent optimization problems (8.3, 8.4).

Because of simplicity, we start with (8.4) and fix range $r_k = r_k^*$. The estimation of r_k^* is detailed later.

We further follow the idea of tracking by Kalman filter [9, 54]. The Kalman filter uses the state of an object, which is a little bit more general than we have been using until now. State $\mathbf{s}$ includes parameters we are interested in (e.g., pose $\mathbf{t}$, illumination $\boldsymbol{\theta}$) and some auxiliary parameters, which serve for state approximation from the object dynamics (e.g., object velocity). The parameters we are interested in are called measurement and we denote them by $\mathbf{z}$. The measurement $\mathbf{z} = \mathtt{M}\mathbf{s}$ is a linear function of state parameters $\mathbf{s}$ in general, but usually matrix $\mathtt{M}$ only selects a subset of the state parameters.

State of an object is estimated in two steps:

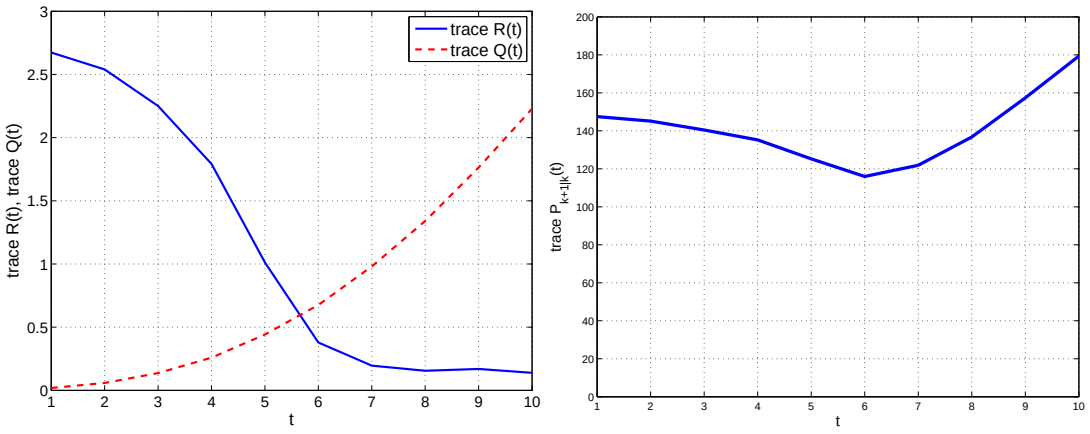
- First the state in frame k is approximated by a linear dynamic model

$$\mathbf{s}_{k|k-1} = \mathtt{A}\mathbf{s}_{k-1|k-1} \tag{8.5}$$

  from a previous state $\mathbf{s}_{k-1|k-1}$, where $\mathtt{A}$ is an $n \times n$ matrix, learned from a training data.

- After that, measurement $\mathbf{z}_{k|k}$ is estimated by SLLiP Φ initialized at $\mathbf{z}_{k|k-1} = \mathtt{M}\mathbf{s}_{k|k-1}$,

$$\mathbf{z}_{k|k}(t_k) = \Phi(\mathbf{z}_{k|k-1}; t_k, r_k^*). \tag{8.6}$$



(a) (b)

Figure 8.4: **Motion model error** trace $(\mathtt{Q}(t_k))$, **SLLiP error** trace $(\mathbf{R}(t_k))$ **and state estimation error in the next frame** trace $\left(\mathtt{P}_{k+1|k}(t_k)\right)$. (a) trace $(\mathtt{Q}(t_k))$ and trace $(\mathtt{Q}(t_k))$ as functions of time t_k spent in current frame. (b) trace $\left(\mathtt{P}_{k+1|k}(t_k)\right)$ as a function of t_k.

The motion model error $\mathbf{q}_k$ and the SLLiP error $\mathbf{r}_k$ are random variables, which are further assumed to be from Gaussian distributions $\mathbf{q}_k \sim \mathcal{N}(0, \mathtt{Q}(t_k)), \mathbf{r}_k \sim \mathcal{N}(0, \mathtt{R}(t_k))$ with covariance matrices $\mathtt{Q}(t_k), \mathtt{R}(t_k)$. Notice, that in contrast to a common Kalman filter tracking approach, both error covariances are the functions of time spent in the k-th frame. trace $(\mathtt{R}(t_k))$ is a decreasing function of time, because the longer time we spend the better is the prediction. trace $(\mathtt{Q}(t_k))$ is increasing function of time, because the longer time we spend, the smaller is the frame-rate and the harder is to approximate the state by the linear dynamic model, see Figure 8.4a.

The state $\mathbf{s}_k$ is approximated by

$$\mathbf{s}_{k|k}(t_k) = \mathbf{s}_{k|k-1} + \mathtt{K}_k(t_k)(\mathbf{z}_{k|k}(t_k) - \mathtt{M}\mathbf{s}_{k|k-1}), \tag{8.7}$$

where $\mathtt{K}_k(t_k)$ weights contributions of prediction by the dynamic model (8.5) and by the tracker (8.6).

Covariance of a state estimation error in frame k based on measurements in frame k is

$$\mathtt{P}_{k|k}(t_k) = E\left\{(\mathbf{s}_k - \mathbf{s}_{k|k})(\mathbf{s}_k - \mathbf{s}_{k|k})^\top\right\}. \tag{8.8}$$

For the sake of simplicity, we utilize the following results of common Kalman filter derivation [34] without any further explanation:

- Kalman gain $\mathtt{K}_k(t_k)$ minimizing trace $\left(P_{k|k}(t_k)\right)$ in frame k is

$$\mathtt{K}_k(t_k) = \mathtt{P}_{k|k-1}\mathtt{M}^\top(\mathtt{M}\mathtt{P}_{k|k-1}(t_k)\mathtt{M}^\top + \mathtt{R}(t_k))^{-1}. \tag{8.9}$$

- Covariance of the state estimation error in frame k is

$$\mathtt{P}_{k|k}(t_k) = (\mathtt{I} - \mathtt{K}_k(t_k)\mathtt{M})\mathtt{P}_{k|k-1}(t_k). \tag{8.10}$$

- Approximation of the covariance of the state estimation error in frame $k+1$ is

$$\mathtt{P}_{k+1|k}(t_k) = \mathtt{A}\mathtt{P}_{k|k}(t_k)\mathtt{A}^\top + \mathtt{Q}(t_k). \tag{8.11}$$

Since we measure the precision by covariance matrix of the state estimation error, problem (8.4) is therefore replaced by

$$t_k^* = \underset{t_k}{\operatorname{argmin}} \left\{ \operatorname{trace}\left(\mathtt{P}_{k+1|k}(t_k)\right) \right\}. \tag{8.12}$$

In order to find a global minimum of $\operatorname{trace}\left(\mathtt{P}_{k+1|k}(t_k)\right)$, we express $\mathtt{P}_{k+1|k}(t_k)$ as a function independent of $\mathtt{K}_k(t_k)$. Substituting to equation (8.11) for $\mathtt{P}_{k|k}(t_k)$ from equation (8.10) and substituting for $\mathtt{K}_k(t_k)$ from equation (8.9) we obtain

$$\mathtt{P}_{k+1|k}(t_k) = \mathtt{A}\left(\mathtt{I} - (\mathtt{P}_{k|k-1}\mathtt{M}^\top(\mathtt{M}\mathtt{P}_{k|k-1}(t_k)\mathtt{M}^\top + \mathtt{R}(t_k))^{-1})\mathtt{M}\right)\mathtt{P}_{k|k-1}(t_k)\Big)\,\mathtt{A}^\top + \mathtt{Q}(t_k). \tag{8.13}$$

Trace of this covariance matrix, see for example Figure 8.4b, has usually one minimum determining the time t_k^* we are allowed to spend. As a side product, $\mathtt{P}_{k+1|k}(t_k^*)$ is also delivered, which allows estimation of the initial range r_{k+1}^* used in the following frame. According to equation (8.3) we search for the smallest range satisfying loss-of-lock probability constraint:

$$P_{\text{loss-of-lock}}(r_{k+1}) = 1 - \int_{-r_{k+1}}^{r_{k+1}} \mathcal{N}(\mathbf{q}, \mathbf{0}, \mathtt{P}_{k+1|k}(t_k^*))\,\mathrm{d}\mathbf{q} < \delta. \tag{8.14}$$

Since $P_{\text{loss-of-lock}}(r_{k+1})$ is a decreasing function of r_{k+1}, range r_{k+1}^* such that $P_{\text{loss-of-lock}}(r_{k+1}^*) = \delta$ is the range solving problem (8.3). Hence we search for the solution r_{k+1}^* of the following equation

$$\int_{-r_{k+1}^*}^{r_{k+1}^*} \mathcal{N}(\mathbf{q}, \mathbf{0}, \mathtt{P}_{k+1|k}(t_k^*))\,\mathrm{d}\mathbf{q} = 1 - \delta. \tag{8.15}$$

In order to avoid time consuming integration over at least 2-dimensional space, we approximate the solution of problem (8.15) by $r_{k+1}^* \approx q \cdot \operatorname{trace}\left(\mathtt{P}_{k+1|k}(t_k^*)\right)$, which for example in 1D space for $q = 3$ results in $\delta \approx 0.2\%$.

Starting the tracking procedure in the following frame at a smaller range causes smaller state covariance allowing to estimate the motion more accurately, which again leads to decreases of the following state covariance. This process leads to a steady solution of $\lim_{k\to\infty}(\mathtt{K}_k, t_k, r_k, \mathtt{P}_{k|k})$ which is later experimentally shown in experiments.

8.4 Algorithm

Tracker Φ, dynamic model $\mathtt{A}$ and their covariance matrices $\mathtt{Q}(t), \mathtt{R}(t)$ need to be estimated from a training data during a learning stage. Learning of SLLiP Φ was described in Section 4, learning of matrix $\mathtt{A}$ is outlined in Section 8.5.

Proposed tracking algorithm consists of two stages:

- **Time update**, where state $\mathbf{s}_{k|k-1}$ and covariance $\mathtt{P}_{k|k-1}$ are approximated from the motion dynamics and time t_k^* minimizing covariance $\mathtt{P}_{k+1|k}$ in the next frame is estimated.

- **Measurement update**, where measurement parameters are predicted by SLLiP $\Phi(\mathbf{s}_{k|k-1}; t_k^*, r_k^*)$ with parameters t_k^*, r_k^*. The final state approximation $\mathbf{s}_{k|k}$ is refined from both predictions.

The method is summarized by Algorithm 6, see also Figure 8.5 where its first 4 steps are visualized. In each step, functions trace $(\mathtt{R}(t_k))$, trace $(\mathtt{Q}(t_k))$, trace $\left(\mathtt{P}_{k+1|k}(t_k)\right)$ and trace $\left(\mathtt{P}_{k|k}(t_k^*)\right)$ are depicted. Initial range is delineated by the green color and the final time by the red color.

1. Initialize covariance $\mathtt{P}_{0|0}$, learn a tracker Φ and dynamic model $\mathtt{A}, \mathtt{M}, \mathtt{Q}(t), \mathtt{R}(t)$. Let $k = 1$
2. Get frame k
3. Time update equations:
   - $\mathbf{s}_{k|k-1} = \mathtt{A}\mathbf{s}_{k-1|k-1}$
   - $t_k^* = \operatorname{argmin}_{t \in \mathbb{R}} \left\{\text{trace}\left(\mathtt{P}_{k+1|k}\right)(t_k)\right\}$
   - $\mathtt{P}_{k|k-1}(t_k^*) = \mathtt{A}\mathtt{P}_{k-1|k-1}(t_k^*)\mathtt{A}^\top + \mathtt{Q}(t_k^*)$
4. Measurement update:
   - $\mathtt{K}_k(t_k^*) = \mathtt{P}_{k|k-1}\mathtt{M}^\top(\mathtt{M}\mathtt{P}_{k|k-1}(t_k^*)\mathtt{M}^\top + \mathtt{R}(t_k^*))^{-1}$
   - $\mathbf{z}_{k|k}(t_k^*) = \Phi(\mathtt{M}\mathbf{s}_{k|k-1}; t_k^*, r_k^*)$
   - $\mathbf{s}_{k|k} = \mathbf{s}_{k|k-1} + \mathtt{K}(t_k^*)(\mathbf{z}_{k|k}(t_k^*) - \mathtt{M}\mathbf{s}_{k|k-1})$
   - $\mathtt{P}_{k|k}(t_k^*) = \mathtt{I} - \mathtt{K}_k(t_k^*)\mathtt{M}\mathtt{P}_{k|k-1}(t_k^*)$
   - $r_{k+1}^* = q \cdot \text{trace}\left(\mathtt{P}_{k+1|k}(t_k^*)\right)$, where q is a parameter.
5. $k = k + 1$ repeat from 2.

Algorithm 6 Proposed method of tracking

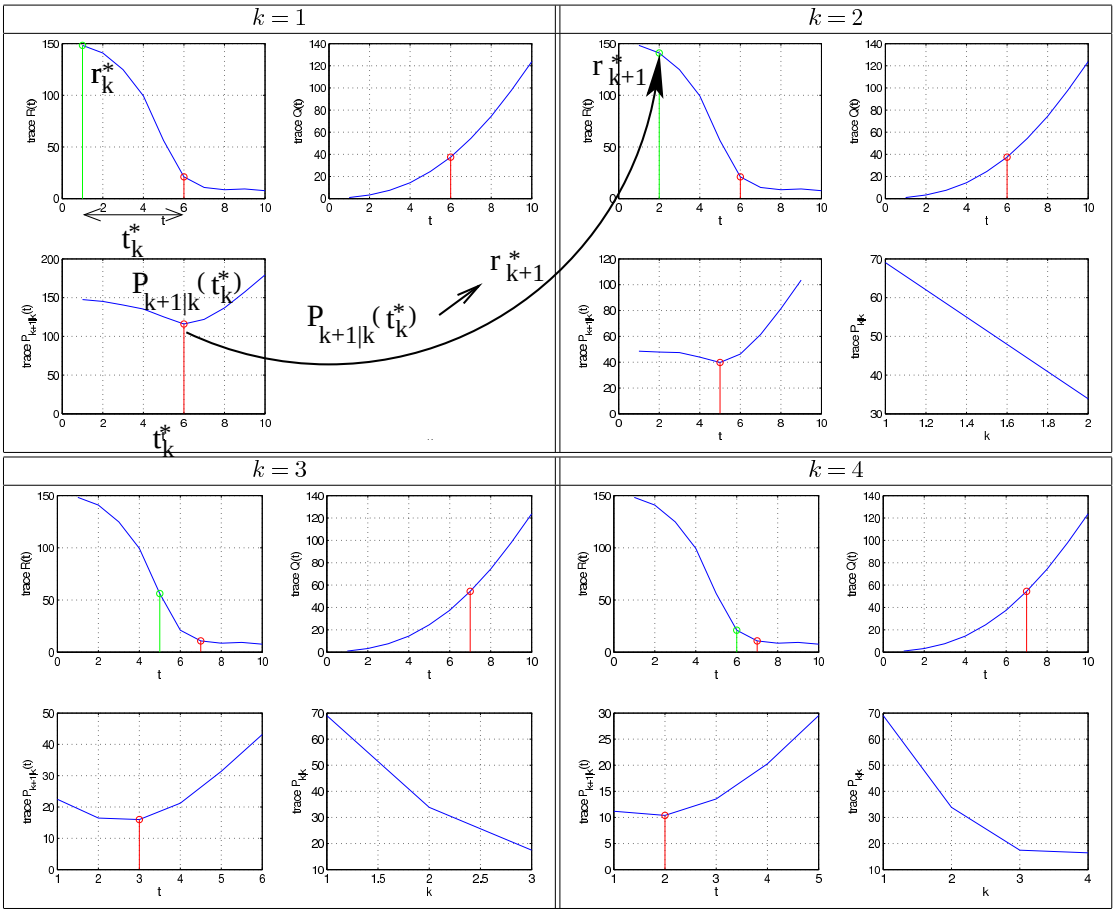


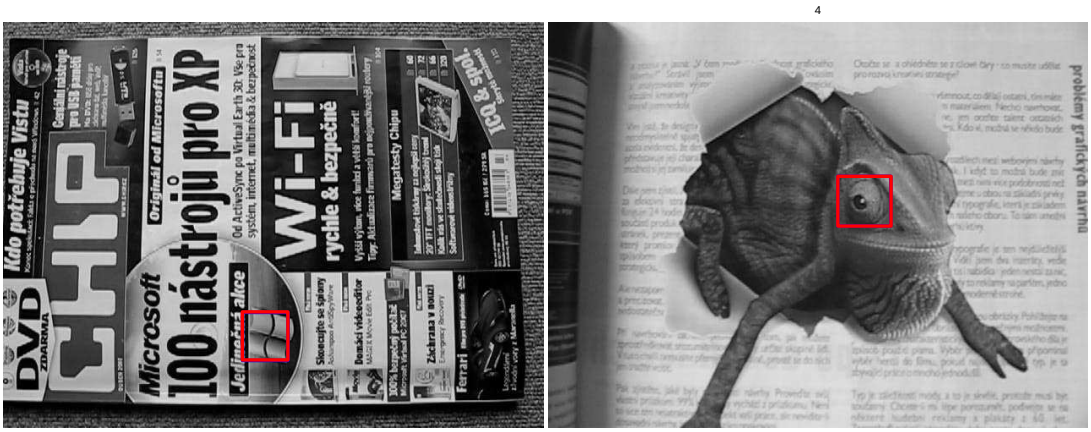
Figure 8.5: **Visualization of Algorithm 6**. Upper-left: trace $(\mathtt{R}(t_k))$ as a function of t, Upper-right: trace $(\mathtt{Q}(t_k))$ as a function of t, Lower-left: trace $\left(\mathtt{P}_{k+1|k}(t_k)\right)$ as a function of t and its minimum, Lower-right: trace $\left(\mathtt{P}_{k|k}(t_k^*)\right)$ as a function of k. Initial range r is denoted by green color, final time t is denoted by red color.

8.5 Experiments

Experiment in Section 8.5.1 shows some results achieved by Algorithm 6 on the tracking of a planar patch shown in Figure 8.6a. Section 8.5.2 shows results on a different sequence Figure 8.6b, achieved if the state covariance matrix is dynamically updated.

8.5.1 Comparison with non updated tracker

We verify our method on a sequence with length of $d = 300$ frames. Tracked object is a planar 50×50-pixels patch, see Figure 8.6a. We learned an optimal linear sequential predictor estimating its 2D translation in a range of 25 pixels.



(a) (b)

Figure 8.6: **Tracked patches** in experiments 8.5.1, 8.5.2.

Considering state vector $\mathbf{x} = \begin{bmatrix} p_x & p_y & v_x & v_y \end{bmatrix}^\top$, where p_x, p_y are positions and v_x, v_y are velocities, linear dynamic model is estimated from a training sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_m$ as follows:

$$\mathtt{A} = [\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_d] \, [\mathbf{x}_0, \mathbf{x}_1 \ldots \mathbf{x}_{d-1}]^+ ,$$

where $+$ denotes pseudo-inverse operation. Also $\mathtt{R}(t)$ and $\mathtt{Q}(\mathtt{t})$ for some discretized values $t \in T \subset \mathbb{R}$ are computed for a training sequence.

We conclude that the proposed method converges to a steady solution with trace $\left(\mathtt{P}_{k|k}\right)$ almost $2\times$ smaller than a common Kalman tracking approaches, see Figure 8.7. Note

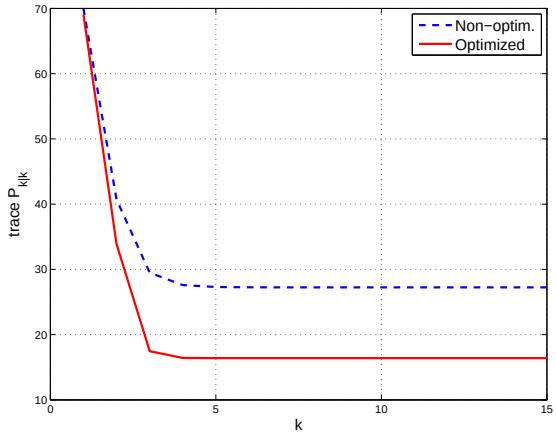


Figure 8.7: **State error comparison** of the online optimized SLLiP and non-optimized SLLiP in realtime tracking. Since the motion is fixed the state error covariance converges to a steady solution.

that, the visualization (Figure 8.5) of the first 4 steps of the Algorithm 6 was obtained during this experiment.

8.5.2 Online $\mathtt{Q}(t)$ updating

The covariance matrix of dynamic model prediction error $\mathtt{Q}(t)$ characterizes the ability of the equation (8.5) to predict the motion of target. Consequently, the value of trace $(\mathtt{Q}(t))$ significantly influences precision of tracking: If the linear model is precise then the algorithm is allowed to spend longer time with fine motion estimation. However, if the linear model fails, for example because of saccadic motion of the target, the method automatically gives up the precision in order to avoid loss-of-lock. We therefore extend Algorithm 6 to the method which automatically updates $\mathtt{Q}(t)$. The results on a testing sequence with 1323 frames, where the first 400 are well predictable by equation (8.5) and in the rest saccadic unpredictable motions are present, are shown in Figure 8.8a, where position p_x is depicted as a function of frames, the sizes of blue circles correspond to trace $\left(P_{k|k}\right)$.

We also conclude that an unpredictable motion causes an increase in frame-rate and inevitably a decrease in precision. Frame-rate, which directly corresponds to the time spent with each particular frame, and trace $\left(P_{k|k}\right)$ are visualized in Figure 8.8b.

8.6 Summary

We proposed adaptive optimization of range and accuracy of the previously proposed SLLiP tracker. The task is formulated as a maximization of tracking precision subject to loss-of-lock probability. We showed, that the proposed process leads to the steady solution of all variables and achieved accuracy is two times higher than the accuracy of the SLLiP with non-optimized range and accuracy.

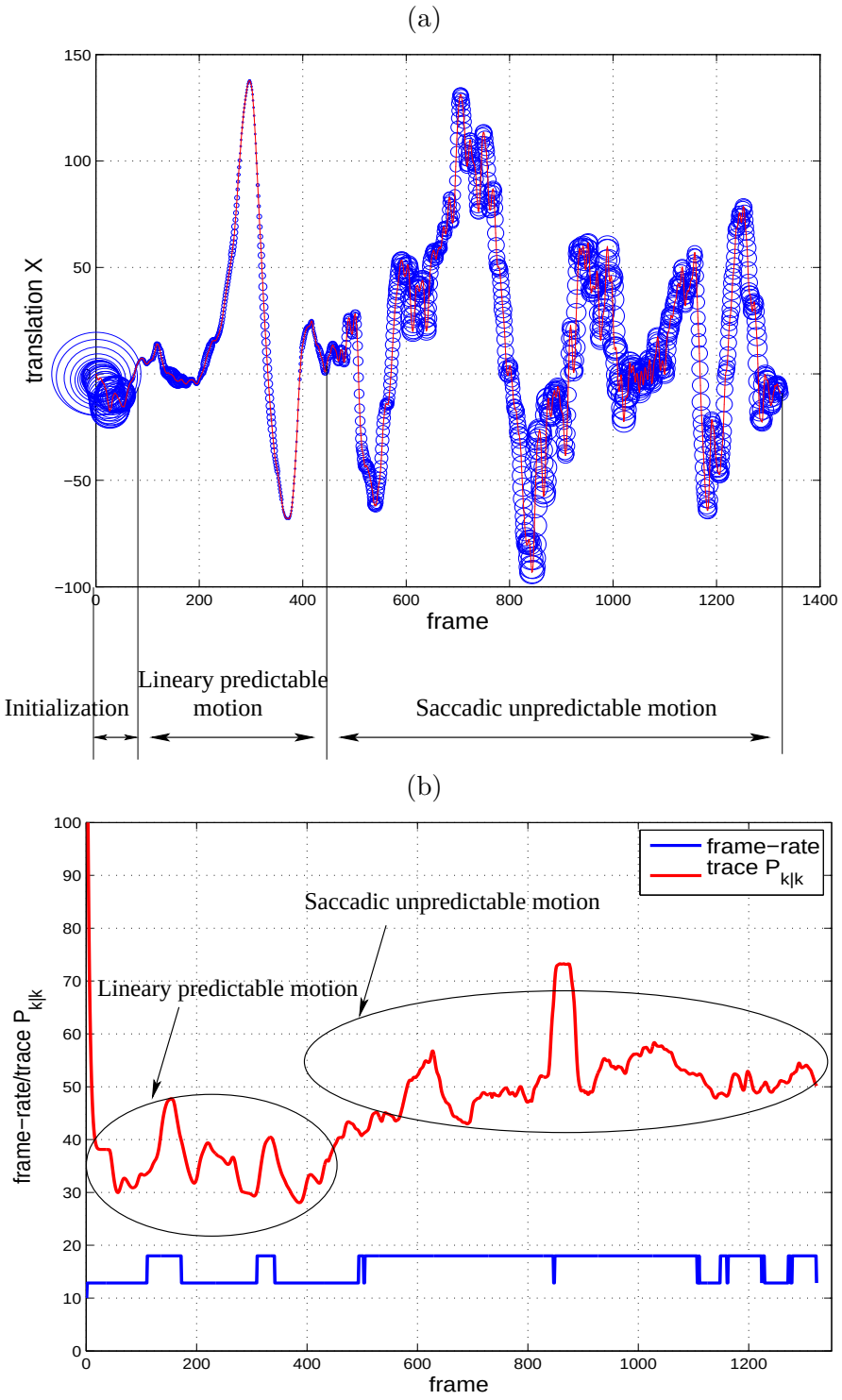


Figure 8.8: **State estimation error during predictable and unpredictable motions:** If the error of the motion model increases, e.g., because of saccadic motion of the object, the method automatically gives up the precision in order to avoid loss-of-lock event. (a) Positions as a function of frames, size of blue markers is proportional to trace $\left(\mathtt{P}_{k|k}(t_k^*)\right)$. (b) Covariance trace $\left(\mathtt{P}_{k|k}(t_k^*)\right)$ and frame-rate as a functions of frames.

9 Conclusions

We proposed and designed several regression-based tracking methods distinguished by accuracy, range, frame-rate, learning time and the type of the object, they are applicable to. In the following, we list the advantages and limitations of the proposed tracking and learning approaches and discuss their suitability for particular tasks.

Proposed regression based methods:

- **Learned Linear Predictors (LLiP).** We studied and extended tracking approaches based on Learned Linear Predictors (LLiPs), i.e., linear mapping between intensities and motion. LLiPs are typically learned from synthetized examples by the Least Square method. Such learning is fast ($\approx$ seconds) however, the range and accuracy are limited. We therefore studied more complex structures, which at the cost of longer learning achieve better performance.

- **Sequential Learned Linear Predictors (SLLiP).** LLiPs are extended to Sequences of LLiPs (SLLiPs). Learning of the optimal SLLiP is formulated as computational complexity minimization subject to predefined accuracy and range. Two different methods are proposed: (i) describes the optimal SLLiPs learned by the minimax method (MM SLLiP), and (ii) is extension for sequences of LLiPs learned by the Least Square method (LS SLLiP), which after a very short initialization period provides a sub-optimal solution. It is experimentally demonstrated that MM SLLiPs have higher robustness, i.e., approximately 30–50% lower number of loss-of-lock events, however the achieved framerate is lower (by about $\approx 30\%$) and the learning time is significantly higher ($\approx$ 30-times) in contrast to LS SLLiPs. Since for both methods approximately 20–30 frames per second are achievable by a MATLAB implementation, much higher framerate is probably achievable with a careful implementation and the framerate difference is not an issue. MM SLLiPs are the most suitable for applications, where long learning is affordable, else anytime learning of LS SLLiPs allows to setup the trade-off between learning time and performance.

  Note, that the proposed approaches were carefully verified on publicly-available sequences with approximately $12,000$ ground-truthed frames. Superiority in framerate and robustness with respect to the SIFT detector, Lucas-Kanade tracker and Jurie's tracker is experimentally demonstrated.

- **Parameter-sensitive Learned Linear Predictors (PLLiP).** If the object appearance changes due to for example non-rigid deformations or uneven illumination, accurate motion estimation by LLiPs or SLLiPs is often impossible. We therefore extended LLiP to Parameter sensitive LLiP (PLLiP). We learn the whole sub-space of LLiPs and encode the current object appearance into the coordinates of this subspace. PLLiP might be theoretically extended to sequences, but we did not studied this issue. PLLiPs are therefore experimentally compared to LLiPs showing that PLLiPs with the same computational complexity achieve 50% lower error than LLiPs.

Proposed optional improvements

We also proposed a few optional improvements of LLiPs, SLLiPs and PLLiPs:

- **Tracking by the number of predictors.** It is well known, that if the object is modeled by more than one predictor and motion is estimated by RANSAC, higher robustness and accuracy are often achievable. We estimate the optimal ratio between the number of predictors and number of RANSAC iteration. Online optimization of the locations, where predictors are attached to the model is also proposed.

- **Adaptive optimization of range and accuracy for SLLiPs.** Any subsequence of a SLLiP can be viewed as another SLLiP with higher speed and smaller range and/or accuracy. We propose method, where the range and accuracy of the SLLiP are automatically adjusted in order to achieve the most accurate tracking given a loss-of-lock probability constraint. Using a Kalman filter, it is automatically determined whether the motions are well predictable by the motion model or not and the most suitable range and accuracy are used. It is experimentally shown that the online optimized SLLiP achieves 40% higher accuracy than the non optimized one.

- **Support set selection** is the combinatorial problem, therefore greedy LS algorithm is proposed. Such improvement increases accuracy about 10% in comparison to random selection, however, for larger support sets the method is very time consuming, because the learning time depends exponentially on the complexity.

All regression-based tracking methods commonly suffers from the limitation by the size and speed of tracked object. If the object is too small and the inter-frame motion is unexpectedly fast, the support pixels may get on the previously unseen background pixels and cause tracker failure. Nevertheless, we see the future of the fast regression-based methods in mutliple regression evaluated by a classifier.

We encourage the reader to download MATLAB implementations and videos from `ftp://cmp.felk.cvut.cz/~zimmerk/linTrack`. Ground truthed sequences are available at `ftp://cmp.felk.cvut.cz/pub/cmp/data/linTrack`.

A Appendix

A.1 Quasi-Euclidean norm for circle and ellipse approximation

For the sake of completness, we also show how to find a λ-bound predictor for circular and elliptic uncertainty region. Because of simplicity, we further considers only 2D space of motion parameters (e.g., translation). A circle can be approximated by equilateral octagon, see Fig. 4.5-d, by combining the preceding results. Defining the quasi-Euclidean norm

$$\|\mathbf{y}\|_E = \min\{\sqrt{2}\|\mathbf{y}\|_1, \|\mathbf{y}\|_\infty\},$$

the estimation of the linear regressor minimizing the size of the octagon-shaped uncertainty area is as follows

$$\mathtt{H}^* = \operatorname*{argmin}_{\mathtt{H}} \max_i \|(\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i)\|_E =$$

$$= \operatorname*{argmin}_{\mathtt{H},\lambda} \{\lambda \mid \forall_i \;\; \min\{\sqrt{2}\sum_j^2 |\mathbf{h}_j^\top \mathbf{I}^i - \mathbf{t}_j^i|,\; |\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i|\} \;< \mathbf{1}\lambda\}.$$

Variables in the corresponding LP problem (4.12) are equal to

$$\mathtt{A} = \begin{bmatrix} \mathtt{I}^\top & \mathtt{I}^\top & -\mathbf{1}\sqrt{2} \\ -\mathtt{I}^\top & \mathtt{I}^\top & -\mathbf{1}\sqrt{2} \\ \mathtt{I}^\top & -\mathtt{I}^\top & -\mathbf{1}\sqrt{2} \\ -\mathtt{I}^\top & -\mathtt{I}^\top & -\mathbf{1}\sqrt{2} \\ \mathtt{I}^\top & \mathtt{0} & -\mathbf{1} \\ -\mathtt{I}^\top & \mathtt{0} & -\mathbf{1} \\ \mathtt{0} & \mathtt{I}^\top & -\mathbf{1} \\ \mathtt{0} & -\mathtt{I}^\top & -\mathbf{1} \end{bmatrix}, \; \mathbf{x} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \lambda \end{bmatrix}, \; \mathbf{c} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \; \mathbf{b} = \begin{bmatrix} \mathbf{t}_1 + \mathbf{t}_2 \\ \mathbf{t}_1 - \mathbf{t}_2 \\ \mathbf{t}_2 - \mathbf{t}_1 \\ -\mathbf{t}_1 - \mathbf{t}_2 \\ \mathbf{t}_1 \\ -\mathbf{t}_1 \\ \mathbf{t}_2 \\ -\mathbf{t}_2 \end{bmatrix}.$$

Sometimes the prediction errors are not distributed uniformly in the uncertainty area due to a strong edge in some direction. While the motion estimation orthogonal to the edge direction is well conditioned, the motion estimation along the edge is ill-conditioned. In these cases, the smallest uncertainty region approximately corresponds to an ellipse (see Figure 4.5f) with main half-axis in the direction of the strong edge.

Let us assume, we are able to find the shape of ellipse. Given the shape (2×2 matrix $\mathtt{Q}$), all the corresponding ellipses are parameterized by their radius λ as follows

$$Q = \{\mathbf{y} \mid \mathbf{y}^\top \mathtt{Q}^\top \mathtt{Q}\mathbf{y} < \lambda\}. \tag{A.1}$$

Radius is minimized similarly to the case above

$$\mathtt{H}^* = \operatorname*{argmin}_{\mathtt{H}} \max_i \|\mathtt{Q}(\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i)\|_E$$

and the corresponding LP problem (4.12) has the following coefficients

$$\mathtt{A} = \begin{bmatrix} \mathtt{Q}\otimes\mathtt{I}^\top & -\mathbf{1} \\ -\mathtt{Q}\otimes\mathtt{I}^\top & -\mathbf{1} \\ (\mathbf{q}_1^\top+\mathbf{q}_2^\top)\otimes\mathtt{I}^\top & -\mathbf{1}\sqrt{2} \\ (\mathbf{q}_1^\top-\mathbf{q}_2^\top)\otimes\mathtt{I}^\top & -\mathbf{1}\sqrt{2} \\ (-\mathbf{q}_1^\top+\mathbf{q}_2^\top)\otimes\mathtt{I}^\top & -\mathbf{1}\sqrt{2} \\ -(\mathbf{q}_1^\top+\mathbf{q}_2^\top)\otimes\mathtt{I}^\top & -\mathbf{1}\sqrt{2} \end{bmatrix},\ \mathbf{x} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \lambda \end{bmatrix},\ \mathbf{c} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} \mathtt{T}^\top\mathbf{q}_1 \\ \mathtt{T}^\top\mathbf{q}_2 \\ -\mathtt{T}^\top\mathbf{q}_1 \\ -\mathtt{T}^\top\mathbf{q}_2 \\ \mathtt{T}^\top\mathbf{q}_1+\mathtt{T}^\top\mathbf{q}_2 \\ \mathtt{T}^\top\mathbf{q}_1-\mathtt{T}^\top\mathbf{q}_2 \\ \mathtt{T}^\top\mathbf{q}_2-\mathtt{T}^\top\mathbf{q}_1 \\ -\mathtt{T}^\top\mathbf{q}_1-\mathtt{T}^\top\mathbf{q}_2 \end{bmatrix},$$

where $\otimes$ denotes Kronecker (matrix direct) product. Since the shape ($\mathtt{Q}$) optimization would not be the linear task, it is necessary to find it in a different way. We do believe that the inversion of the covariance matrix of LS solution errors is a good approximation.

A.2 Semidefinite programming for minimization of elliptic uncertainty region

Computation of an optimal linear regressor using the true elliptic uncertainty region (Figure 4.5g), instead of the approximation (Figure 4.5f), is formulated as semidefinite programming problem. Let us define ellipse Q in a different way

$$Q = \{\mathbf{y} \mid \mathbf{y}^\top\mathtt{Q}\mathbf{y} < 1\}$$

than in previous section. The Q has volume

$$\mathrm{vol}(Q) = V\det(\mathtt{Q}^{-1}) \approx -\mathrm{trace}\,\mathtt{Q}, \tag{A.2}$$

where V denotes volume of the unit sphere in corresponding space (e.g., in $\mathbb{R}^2$ $V=\pi$). Equation (4.3) is reformulated for elliptic uncertainty region as follows

$$\min_{Q,\mathtt{H}} \left\{\mathrm{vol}(Q) \;\mid\; \forall_i \left(\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i\right) \in Q\right\}. \tag{A.3}$$

Approximating ellipsoid Q according to equation (A.2) we can write

$$(\mathtt{Q}^*,\mathtt{H}^*) = \operatorname*{argmin}_{\mathtt{Q},\mathtt{H}} \left\{-\mathrm{trace}\,\mathtt{Q} \;\mid\; \forall_i (\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i)^\top\mathtt{Q}(\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i) \le 1\right\}. \tag{A.4}$$

Using Schur complement [11], following equivalence can be proved

$$(\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i)^\top\mathtt{Q}(\mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i) \le 1 \iff \begin{bmatrix} \mathtt{Q}^{-1} & \mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i \\ \mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i & 1 \end{bmatrix} \succeq 0,\ \mathtt{Q} \succeq 0, \tag{A.5}$$

where $\succeq 0$ denotes that a matrix is from a positive semi-definite cone. Using equivalence (A.5), constraining conditions in equation (A.4) are replaced and the following semidefinite problem is obtained

$$(\mathtt{Q}^*, \mathtt{H}^*) = \underset{\mathtt{Q},\mathtt{H}}{\operatorname{argmin}} \left\{ \operatorname{trace} \mathtt{Q} \mid \forall_i \begin{bmatrix} \mathtt{Q} & \mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i \\ \mathtt{H}\,\mathbf{I}^i - \mathbf{t}^i & 1 \end{bmatrix} \succeq 0,\ \mathtt{Q} \succeq 0 \right\}. \tag{A.6}$$

In this optimization task, 2×2 matrix $\mathtt{Q}$ stands for parameter λ in equation (4.3). Hence, the uncertainty region is parameterized in the four dimensional space instead of the one dimensional. In order to retain reasonable computational complexity of learning, we need to parameterize the shape only by one parameter by fixing the shape $\mathtt{Q}$ of the obtained ellipsoid Q for each particular simple predictor like we did in Section A.1.

Different methods have been proposed for various classes of uncertainty regions. Results of the best three methods are depicted in Figure 4.5e-g.

Bibliography

[1] A. Agarwal and B. Triggs. Recovering 3d human pose from monocular images. *IEEE Transactions Pattern Analysis Machine Intelligence*, 28(1):44–58, 2006.

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

[3] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[4] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *9th European Conference on Computer Vision*, Graz Austria, May 2006.

[5] S. Benhimane, A. Ladikos, V. Lepetit, and N. Navab. Linear and quadratic subsets for template-based tracking. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, 2007.

[6] S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *International Conference on Intelligent Robots Systems*. Japan, 2004.

[7] A. Bissacco, M. Yang, and S. Soatto. Fast human pose estimation using appearance and motion via multi-dimensional boosting regression. In *Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[8] M. J. Black and A. D. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. In B. F. Buxton and R. Cipolla, editors, *4th European Conference on Computer Vision*, volume 1064 of *Lecture Notes in Computer Science*, pages 329–342. Springer, 1996.

[9] A. Blake and M. Isard. *Active Contours*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[10] B. Bollobás. *Modern Graph Theory*. Springer, New York, 1998.

[11] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[12] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

[13] R. Brooks, T. Arbel, and D. Precup. Fast image alignment using anytime algorithms. In M. M. Veloso, editor, *IJCAI*, pages 2078–2083, 2007.

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

[15] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *International Conference on Computer Vision*, volume 2, pages 1197–1203. IEEE Computer Society, 1999.

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

[17] V. Comaniciu, D. Ramesh and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–575, 2003.

[18] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. In *5th European Conference on Computer Vision-Volume II*, pages 484–498, London, UK, 1998. Springer-Verlag.

[19] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, June 2001.

[20] D. Cristinacce and T. Cootes. Feature detection and tracking with constrained local models. In 17^{th} *British Machine Vision Conference, Edinburgh, England*, pages 929–938, 2006.

[21] N. Dowson and R. Bowden. N-tier simultaneous modelling and tracking for arbitrary warps. In *British Machine Vision Conference*, volume 2, pages 569–578, 2006.

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

[23] L. Ellis, N. Dowson, J. Matas, and R. Bowden. Linear predictors for fast simultaneous modelling and tracking. In *Proceedings of 11th IEEE International Conference on Computer Vision, workshop on Non-rigid registration and tracking through learning*, Rio de Janeiro, Brazil, October 2007. IEEE computer society.

[24] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, 2005.

[25] K. Fukunaga. *Statistical Pattern Recognition*. Academic Press, New York, NY, USA, 1990.

[26] N. I. M. Gould and S. Leyffer. An introduction to algorithms for nonlinear optimization. In *Frontiers in Numerical Analysis*, pages 109 – 197, Berlin, 2003. Springer Verlag.

[27] H. Grabner and H. Bischof. On-line boosting and vision. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 260–267, Washington, DC, USA, 2006. IEEE Computer Society.

[28] A. Haar. Zur theorie der orthogonalen funktionensysteme. *Annals of Mathematics*, 69:331–371, 1910.

[29] G. D. Hager and P. N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, 1998.

[30] C. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, 1988.

[31] M. Isard and A. Blake. Condensation-conditional denstity propagation for visual tracking. *IJCV*, 29(1):5–28, 1997.

[32] A. D. Jepson, D. J. Fleet, and T. F. El-Maraghi. Robust online appearance models for visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1296–1311, 2003.

[33] F. Jurie and M. Dhome. Real time robust template matching. In *British Machine Vision Conference*, pages 123–131, 2002.

[34] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME - Journal of Basic Engineering, 82(Series D)*, pages 35–45, 1960.

[35] A. Land and A. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

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

[37] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Journal of Computer Vision and Artificial Intelligence*, pages 674–679, 1981.

[38] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, September 2004.

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

[40] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. *cvpr*, 01:652–659, 2004.

[41] M. Özuysal, V. Lepetit, F. Fleuret, and P. Fua. Feature harvesting for tracking-by-detection. In *9th European Conference on Computer Vision*, volume 3953 of *Lecture Notes in Computer Science*, pages 592–605. Springer, 2006.

[42] I. Patras and E. Hancock. Regression tracking with data relevance determination. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, 2007.

[43] Y. Satoh, T. Okatani, and K. Deguchi. A color-based tracking by kalman particle filter. In *International Conference on Pattern Recognition*, volume 3, pages 502–505, Washington, DC, USA, 2004. IEEE Computer Society.

[44] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593 – 600, 1994.

[45] C. Sminchisescu, A. Kanaujia, Z. Li, and D. Metaxas. Discriminative density propagation for 3d human motion estimation. In *Conference on Computer Vision and Pattern Recognition*, volume 1, pages 390–397, Washington, DC, USA, 2005. IEEE Computer Society.

[46] C. Sminchisescu, A. Kanaujia, and D. Metaxas. Discriminative density propagation for visual tracking. *IEEE Transactions Pattern Analysis Machine Intelligence*, 2007.

[47] A. Smola and B. Schölkopf. A tutorial on support vector regression. Technical report, ESPRIT Working Group in Neural and Computational Learning II (NeuroCOLT2), 1998.

[48] J. Šochman and J. Matas. Waldboost - learning for time constrained sequential detection. In C. Schmid, S. Soatto, and C. Tomasi, editors, *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 150–157, Los Alamitos, USA, June 2005. IEEE Computer Society.

[49] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *IEEE Computer Vision and Pattern Recognition*, pages 246–252, June 1999.

[50] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[51] 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*, 26(10):1385–1391, 2004.

[52] V. N. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer-Verlag, New-York, Inc., November 1999.

[53] P. Viola and M. Jones. Robust real-time face detection. In *8th IEEE International Conference on Computer Vision*, volume 2, pages 747–757, 2001.

[54] G. Welch and G. Bishop. An introduction to the kalman filter. In *Special Interest Group for Computer Graphics*, 2001.

[55] O. Williams, A. Blake, and R. Cipolla. Sparse bayesian learning for efficient visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1292–1304, 2005.

[56] Q. Yuan, A. Thangali, V. Ablavsky, and S. Sclaroff. Parameter sensitive detectors. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, 2007.

[57] J. Zhong and S. Sclaroff. Segmenting foreground objects from a dynamic textured background via a robust kalman filter. In *International Conference on Computer Vision*, volume 1, pages 44–52, 2003.

[58] S. K. Zhou, B. Georgescu, D. Comaniciu, and J. Shao. Boostmotion: Boosting a discriminative similarity function for motion estimation. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1761–1768, 2006.

[59] S. K. Zhou, B. Georgescu, X. S. Zhou, and D. Comaniciu. Image based regression using boosting method. In *Proceedings of the 10th IEEE International Conference on Computer Vision*, volume 1, pages 541–548, Washington, DC, USA, 2005. IEEE Computer Society.

[60] Z. Zhu, Q. Ji, and K. Fujimura. Combining kalman filtering and mean shift for real time eye tracking under active ir illumination. volume 4, pages 318–321, 2002.

[61] K. Zimmermann, T. Svoboda, and J. Matas. Multiview 3D tracking with an incrementally constructed 3D model. In *Third International Symposium on 3D Data Processing, Visualization and Transmission*, page 9, Chapel Hill, USA, June 2006. University of North Carolina.

List of publications related to the thesis

Sections	**Publication**
4.1, 4.2	K. Zimmermann, J. Matas, T. Svoboda (40-30-30). Tracking by an Optimal Sequence of Linear Predictors. *Transaction on Pattern Analysis Machine Intelligence*, IEEE computer society, accepted, Washington, DC, USA, 2008.
4.3	K. Zimmermann, T. Svoboda, J. Matas (33-33-33). Anytime learning for the NoSLLiP tracker. *Image and Vision Computing*, Elsevier, submitted, 2008.
7	K. Zimmermann, T. Svoboda and J. Hilton, (50-30-20). Simultaneous learning of motion and appearance. *The 1st International Workshop on Machine Learning for Vision-based Motion Analysis, In conjunction with the 10th European Conference on Computer Vision*, Maresille, France, 2008.
8	K. Zimmermann, T. Svoboda and J. Matas, (33-33-33). Adaptive parameter optimization for real-time tracking. *Proceedings of 11th IEEE International Conference on Computer Vision, workshop on Non-rigid registration and tracking through learning*, isbn: 978-1-4244-1631-8, issn: 1550-5499, IEEE computer society, Rio de Janeiro, Brazil, 2007.
5	J. Matas, K. Zimmermann, T. Svoboda and A. Hilton, (25-25-25-25). Learning Efficient Linear Predictors for Motion Estimation. *5th Indian Conference on Computer Vision, Graphics and Image Processing*, Springer-Verlag, pages: 445-456, LNCS: 4338, issn: 0302-9743, isbn: 978-3-540-68301-8, Madurai, India, 2006