CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY

Habilitation thesis

Learning models and regression for visual tracking and detection

Tomáš Svoboda

svobodat@fel.cvut.cz

September 17, 2012

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

Contents

1 Introduction

Learning models from visual exemplars proved to be a very successful strategy in computer vision. Results of methods which learn from images clearly outperform results achieved by using hand-crafted models. This is true in many areas including object recognition, image retrieval, or object tracking to name just a few. Perhaps even more surprisingly, some very successful methods actually do not only learn models from training data but also automatically select the most appropriate features. With just a small exaggeration the only input required is a pile of positive examples. Discriminatively trained classifiers are often able to collect negative examples automatically.

1.1 Thesis overview

The thesis is organized by topics and it is also approximately sorted by the time the work has been performed. If not stated otherwise, a chapter corresponds to a paper published in an international conference or in an impacted scientific journal. Each chapter begins with a short paragraph explaining about its origin. The relevant state of the art and conclusion sections are part of all chapters. The state of the art parts have not been significantly updated and may thus not include some recent contributions.

The part I of the thesis discusses approaches which use multiview data, assume articulated object or both. The use of multiple cameras that see scene from convenient viewpoints makes many computer vision problems conditioned better or generally solvable at all. A typical example is the estimation and fitting of a human articulated model. We define number of parts and the tree-like connectivity between them. The probabilistic method we proposed estimates the dimensions of the parts and joint angles. The input is a set of segmented multiview images. Neither learning on annotated data nor manual input is needed. Having said this, we actually do propose a method for estimating an articulated human model from a single image, which is described in the subsequent chapter. The monocular method however requires intensive training on annotated data. As the full depth cannot be estimated from a single image the resulting model is more 2D than a full 3D model despite the fact that foreshortening is modeled. In the closing chapter if the first part we show that multiple cameras allow for visual modeling and gradient based tracking of a rigid object.

The part II is about predicting the object motion by learnable regressors. The term *prediction* is a sort of product of its time. The term was coined in order to emphasize the striking efficiency progress against the estimators usual at that time. Even a non-optimal implementation in Matlab[1] runs much faster than the real time–camera

[1] Matlab is sometimes considered as unacceptably slow for computer vision applications.

frame rate. The main computation complexity is transferred to the learning phase which learns decisive all possible object appearances ideally in advance and learns simultaneously the simplest possible predictor–estimator. The optimality criterion may vary. Two main are discussed: maximal error on a training set and fixed complexity of the resulting predictor.

The part III of the thesis describes some more recent progress of the learnable regressors and their application to several computer vision problems. We step beyond the static object model and introduce incremental learning. The algorithm allows for updating the visual model as the prediction goes. The efficient predictor is coupled with a simple detector and applied in mobile robotics in object localization in high-resolution images.

1.2 What did not fit in

The thesis discussed work I did with my students at CTU in Prague during last few years. Some work has not been included notably the work I did during my post-doc stay at ETH Zürich. I list the non-included topics for sake of completeness: the work on geometry and omnidirectional vision [21,89], automated video editor [20], multicamera systems [86,22,36], image based object retrieval [78,79,77], self-calibration [88,85], Kalman filter [17,105], and also a very recent work on recognition of free-form 3D objects [69]. I also do not include the very recent progress on predictors witch I still consider to be very much an ongoing work despite the starting publication [100]. Our collaboration with the Honeywell Research on face association problem for visual surveillance [44] has also been omitted.

1.3 The personal story and acknowledgments

I was introduced to the fascinating filed of computer vision by Václav Hlaváč in the time I was pursuing my master degree. I learned a lot when I was a research student in his lab. He also sent me abroad to the Delft University for gaining some international experiences. I spent wonderful years in the Center for Machine Perception (CMP) when pushing the limits in studying geometry in computer vision. My effort, co-advised by Tomáš Pajdla and Václav Hlaváč eventually led to a PhD on Panoramic vision geometry. During my PhD study I also spent four months at INRIA Rhône-Alpes where I witnessed a very friendly environment of a top European research lab headed by Roger Mohr. Finishing my PhD thesis would have been much harder without staying three months at Walter Kropatsch's group in TU Wien where I had a privilege of writing and thinking without the usual interrupts. After PhD, I went to ETH Zürich, a top world university where I spent three fruitful years as a post-doc at Luc Van Gool's lab. I switched also the research topic and did some work on image based retrieval and tracking. Besides these topics I fell somehow in love with multicamera systems which was enabled by the steep raise of digital cameras at the beginning of 2000's. I am indebted to Vladimír Mařík, head of the Department of Cybernetics, who accepted me back to the department after my return from the

post-doc. While working at CMP, I was lucky enough being advisor-specialist of PhD study of Karel Zimmermann. Chapters 2, 4, 5, 6, and 7 stem from our joint work. During the years, I collaborated with several bright master students. One of the most successful series is presented in chapter 3. Chapters 8 and 9 represent joint work with my PhD student David Hurych.

I have been recently participating on establishing a new computer science study program Open Informatics[2]. I am honored to serve the board of the program.

I am truly indebted to Václav Hlaváč who stood near to me for most of my scientific and teaching career. He also carefully read this thesis for which I am really thankful. I am grateful he convinced me co-writing a book [87]. He is also my role model as a team leader. Tomáš Pajdla was at the beginning of my scientific work. He has been influencing me a lot. Later in my career, I have had the pleasure of having many discussions with Jiří Matas. His unrelenting willingness to push the limits of knowledge has been very stimulating.

[2]`http://oi.fel.cvut.cz/`

Part I

Articulated models and multiple views

2 Probabilistic Estimation of Articulated Body Model from Multiview Data

The chapter describes our work on the estimating articulated human model from segmented multiview sequences. The human body parts are modeled as ellipsoids. Number of parts and their connectivity are the only inputs specified. Physical dimensions and articulation parameters are estimated from data. The work has been published as a conference paper [103].

An optimization algorithm and statistical description of articulated body model estimation is proposed. The optimization algorithm fits the model into segmented multiview images. The input of our algorithm is a sequence of segmented images captured by several cameras and a structure of the articulated model. The output of the optimization procedure is the shape and the motion of the articulated model. The optimization runs over all cameras and all images in the sequence. We focus on the description and optimization of probability distribution of the model parameters given segmented multiview sequence. The performance of the algorithm is demonstrated on real sequences of a walking human.

2.1 Introduction

The acquisition of articulated body model, often called motion capture, has numerous applications. The gait analysis, computer animations or ergonomics study are only few examples. The motion capture problem is often solved by using commercially available marker-based systems. A set of markers is attached to important positions on the human body. The 3D coordinates of the markers are computed via magnetic or optical tracking. Such systems are expensive and using markers is uncomfortable for patients and may affect their natural behaviour. Therefore computer vision researches have studied markerless motion capture where no special hardware devices are needed.

A setup for markerless motion capture typically consists of several calibrated [88] cameras encircling a working volume. A static background of the scene is modeled from images of the empty scene. The position of a human body is found by the background subtraction [84] which data allows efficient computation of a volumetric model. Motion parameters of the human body are then found by fitting of an articulated model to multiview observations or directly to the volumetric model.

Some methods fit model directly to the silhouettes [10,33]. The criterial function describes correspondence of the model projection with silhouettes (e.g., Magnor and Theobalt [57] uses simple XOR function). An alternative way is to reconstruct 3D shape first and then fit the model to it. Mikic et al. [62] fit cylindrical model into the carved voxel volume. The space carving requires a relatively high number of cameras.

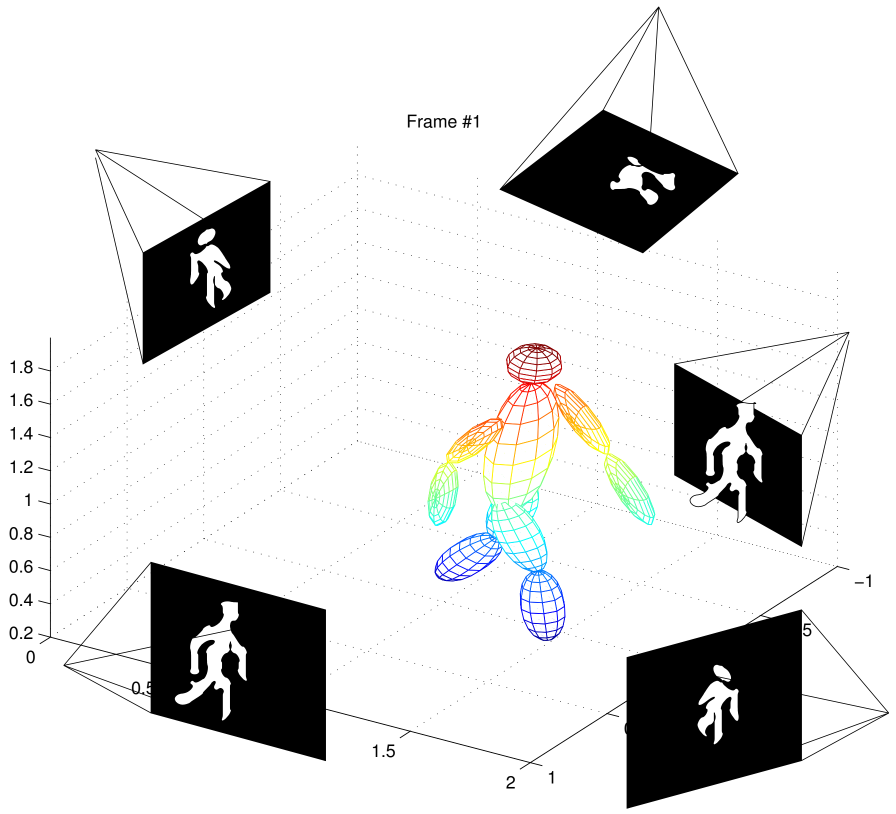


Figure 2.1: A setup for markerless motion capture consists of several cameras encircling the working volume. Articulated ellipsoidal model is used for human body approximation. The parameters are retrieved by optimization, in which the coverage of the silhouettes by the model projection is maximized. All parameters are computed fully automatically without any manual intervention. The dimensions are in meters.

Therefore Plankers and Fua [70] combine fitting to the silhouettes with the stereo reconstruction.

All mentioned methods have a lot variations in complexity of the model or criterial function, but the crucial problem is the non-linear, high-dimensional minimization. Mikic et al. [62] solve the high-dimensional minimization by a hierarchical decomposition. They fit head first, torso second, and limbs finally. This approach, of course, does not assure that global minimum is found. Urtasun and Fua [91] reduce the dimension of the search-space by PCA. However, this limits the variety of movements which can be tracked. Deutscher and Reid [19] propose the particle filter which is efficient for non-linear search.

The current state-of-the-art approaches achieve good results for a human body model tracking but require a careful model initialization. The initialization is often done by assigning initial parameters by hand or by requiring a certain pose of the captured human.

Moreover, the static parameters evaluation (e.g. the length of the arm) requires the optimization over the whole sequence at once. In contrast with previous works, we propose an approach with a weak model but we describe correctly the decomposition of the criterial function over the time to obtain static parameters, which is close to global minimum over the whole sequence. Hence, our approach needs no manual initialization and exploits all available information since it optimizes over all data.

2.2 Human body model and parameters

We decided to use an articulated body model created from ellipsoids, see Figure 2.1. The ellipsoidal model approximates the shape of the real human body and allows a very fast projection to the images which is advantageous in the optimization. Each rigid part is represented by one ellipsoid. The structure of the model follows the anatomy of an average human. Movement of a joint causes movement of all succeeding rigid parts. We apply the Hartenberg-Denavit notation [18] for open kinematic chains which is a frequent solution in robotics.

We distinguish two main types of human body model parameters: shape (e.g. length of the arm) and motion (e.g. angle between the upper and the lower arm). The motion parameters are naturally different for each frame of the multiview sequence. However, we assume the shape parameters to be constant. i.e. the person, remain the same throughout the whole sequence. Parameters of the model given multiview sequence $Z = \{Z_1, \dots, Z_n\}$ are $\theta = \{\mathbf{m}_1, \dots, \mathbf{m}_n, \mathbf{s}\}$, where n is the number of frames and Z_i is a particular multiview frame (i.e., the set of images from all cameras in time i). In the case of ellipsoidal model, shape parameters are sizes of ellipsoids, and the motion parameters are mutual positions and orientations of ellipsoids.

Let us consider for the moment that shape parameters $\mathbf{s}$ are known. The Estimation of motion parameters $\mathbf{m}_i$ in frame i is based on the posterior probability maximization. The posterior is calculated from the projection of the ellipsoidal model to the cameras. The probability of parameters, given images Z_i, is inversely proportional to the sum of distances between border of silhouettes (i.e. segmented images) and projected ellipsoids. The probability is maximized by the standard Gauss-Newton method.

A problem arises when the shape parameters $\mathbf{s}$ are unknown. Clearly, the optimization of all parameters θ over the whole sequence is technically intractable because the number of variables of $p(\theta|Z)$ is proportional to the number of frames in the sequence Z. Therefore we propose an algorithm which finds the maximum of $p(\theta|Z)$ without the necessity to optimize over all of the parameters at once. The optimization method is independent on the choice of the model structure (i.e. the derivations are provided without any explicit knowledge of the criterial function). The ellipsoidal model is used only for experimental results and can be replaced simply by a more sophisticated model of an arbitrary articulated structure.

2.3 Statistical framework

In our particular case, we are looking for the most probable shape $\mathbf{s}$ and motion $\mathbf{m}_1, \ldots, \mathbf{m}_n$ parameters, compactly called $\theta = \{\mathbf{m}_1, \ldots, \mathbf{m}_n, \mathbf{s}\}$, given a multiview segmented sequence Z of the length n and also the structure of articulated model. The multiview sequence $Z = \{Z_1, \ldots, Z_n\}$ consists of multiview frames Z_i which are set of images from all of the cameras in time i. The probability of parameters is given by the *sequence posterior probability* $p(\theta|Z)$. The sequence posterior optimization over all parameters at once is technically intractable. Therefore we decompose sequence posterior into a multiplicative form which is suitable for optimization.

In order to split the optimization task into the particular subtasks we need to accept a few constraints on parameters independence. We expect a human motion to be a Markov process. The motion parameters $\mathbf{m}_i$ in frame i are considered to be dependent only on the multiview frame Z_i, shape parameters $\mathbf{s}$ and preceding motion parameters $\mathbf{m}_{i-1}$.

Under these constraints, the sequence posterior is decomposed to

$$p(\theta|Z) = p(\mathbf{m}_1|\mathbf{s}, Z_1)p(\mathbf{m}_k, \mathbf{s}|Z_k) \prod_{i \neq k} p(\mathbf{m}_i|\mathbf{m}_{i-1}, \mathbf{s}, Z_i), \tag{2.1}$$

where k denotes a keyframe (which is an arbitrary frame of sequence). The full derivation can be found in Appendix A. The sequence posterior is multiplication of probabilities of parameters in the appropriate frames, which are called *frame posterior probabilities.* The algorithm can optimize $p(\mathbf{m}_k, \mathbf{s}|Z_k)$ to obtain the optimal value of $\mathbf{s}$ with respect to the observation Z_k. Given the shape parameters, we can optimize motion parameters of the whole sequence frame by frame from $p(\mathbf{m}_1|\mathbf{s}, Z_1)$ to $p(\mathbf{m}_n|\mathbf{m}_{n-1}, \mathbf{s}, Z_n)$. These shape parameters are optimal with respect to the frame k, but it is not clear whether these parameters will be optimal for the sequence posterior. Nevertheless, since the frame posterior is the probability of parameters in the given frame, we would expect it to have the maximum near the optimal value of shape parameters with respect to the sequence posterior.

Not all the frames provide the same information about the shape parameters. Two examples of different frames of arm-like object projection to the camera are depicted in Figure 2.2. The shape parameter $\mathbf{s}$ is the ratio of semi-axes of the two-ellipsoidal model in this case. The first frame does not provide any information about the shape parameter $\mathbf{s}$ because the frame posterior is uniformly distributed. In contrast, a different image of the same object allows the estimation of the true value of shape parameter $\mathbf{s}$. It is hard to say in advance what is the most informative frame. We propose trying the optimization of shape parameters in a several different frames. Given individual hypothesis of shape parameters, we are able to evaluate the sequence posterior and decide, which value is optimal. More formal description is provided in the section 2.4.

Individual frame posteriors in the (2.1) are derived in Appendix B. There are two different frame posterior:
with *not given* shape parameters

$$p(\mathbf{m}_i, \mathbf{s}|Z_i) \propto p(Z_i|\mathbf{m}_i, \mathbf{s})p(\mathbf{s}), \tag{2.2}$$

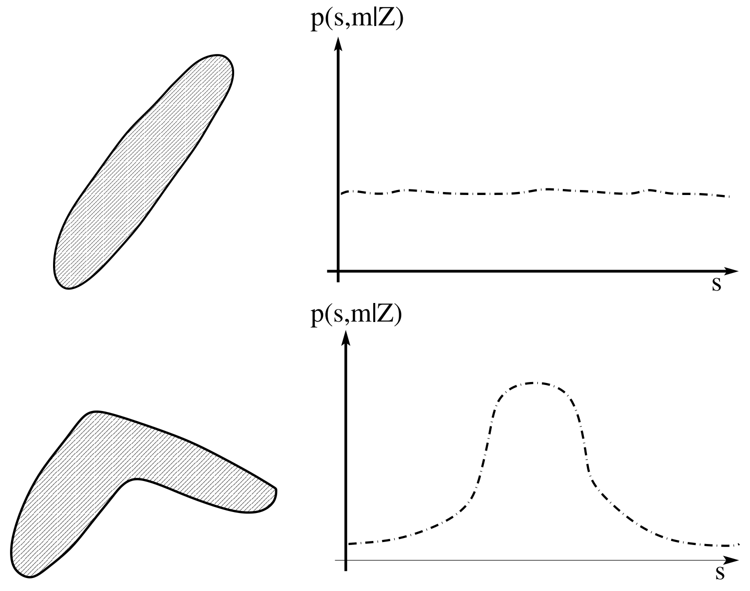


Figure 2.2: Two views of an arm-like object which consists of two rigid parts. However, the upper view provides no evidence of that fact. This non-usefulness is reflected by the uniform distribution of the shape parameters. The bottom row shows a much more useful frame which is reflected by the distribution.

and with *given* shape parameters

$$p(\mathbf{m}_i|\mathbf{m}_{i-1}, \mathbf{s}, Z_i) \propto p(Z_i|\mathbf{m}_i, \mathbf{s})p(\mathbf{m}_i|\mathbf{m}_{i-1}). \tag{2.3}$$

If preceding motion parameters are unknown, then the prior $p(\mathbf{m}_i|\mathbf{m}_{i-1})$ has an uniform distribution and equation (2.3) reduces to the simpler form without this prior. The $p(Z_i|\mathbf{m}_i, \mathbf{s})$, usually called *likelihood*, is proportional to the coverage of segmented images Z_i by model with given parameters $(\mathbf{m}_i, \mathbf{s})$. The likelihood can accommodate arbitrary appearance information like color histogram, edges, etc.

In this work, however, we do not use any appearance information and likelihood is proportional to the distance between border of silhouettes (segmented images) and projected ellipsoidal model[1]. The $p(\mathbf{m}_i|\mathbf{m}_{i-1})$ and $p(\mathbf{s})$ are prior probabilities of motion $\mathbf{m}_i$ and shape $\mathbf{s}$ parameters, respectively. The prior of motion parameters given preceding motion parameters combines temporal coherence constraints with natural motion limitations. The shape prior, which represents the probability of the shape, is considered to have a Gaussian distribution with the mean and covariance matrix proportional to the natural shapes.

2.4 Optimization

The sequence posterior is expressed as multiplication of the frame posterior probabilities in (2.1), in which only one of them (no matter which) is not given shape

[1]More details about the model can be found in Appendix C.

parameters in advance. Theoretically, we could obtain shape parameters by optimization $p(\mathbf{m}_k, \mathbf{s}|Z_k)$ in a arbitrary frame k. However, as argued in the preceding section, the approximation of true value is successful only in the most informative frames.

The shape parameter optimization is performed by Algorithm 1. The vector of shape parameters $\mathbf{s}$ is called hypothesis. Given the hypothesis, motion parameters are evaluated frame by frame by maximizing the frame posteriors. The probability of the hypothesis is calculated by (2.1).

First, the algorithm chooses the set of keyframes, frames in which it expects to be the most informative. The selection follows the expected type of motion and the sampling frequency (frames per second). The keyframes should (sparsely) cover one period of the motion at least. The first shape hypothesis is the shape which maximize the prior $p(\mathbf{s})$. Next shape hypothesis is obtained by optimization in the keyframes over the shape and motion parameters.

Second, the motion parameters, given each of hypothesized shape parameters, are evaluated. Finally, the sequence posterior of each hypothesis (i.e. shape and appropriate motion parameters) is calculated and the most probable hypothesis is selected.

```
Algorithm 1 - the sequence posterior maximization.

  1. $\mathbf{K}$ is set of keyframe indexes, $\mathbf{H}$ is the set of shape parameters hypothesis.
  2. Set the shape parameters to the mean of the prior $p(\mathbf{s})$ (i.e., the most apriori
     probable values).
  3. for each keyframe $i \in \mathbf{K}$:
       • optimize the shape and motion parameters $(\mathbf{m}^*, \mathbf{s}^*) = \arg\max p(\mathbf{m}_i, \mathbf{s}|Z_i)$
         (2.2)
       • save the $\mathbf{s}^*$ as hypothesis $\mathbf{H} = \mathbf{H} \cup \mathbf{s}^*$.
  4. for each shape hypothesis $\mathbf{s}_j \in \mathbf{H}$ and each frame $i$ do:
       • optimize only motion parameters $\mathbf{m}^* = \arg\max p(\mathbf{m}_i|\mathbf{m}_{i-1}, \mathbf{s}, Z_i)$ (2.3).
  5. Evaluate the sequence posterior (2.1) for each $\mathbf{s}_j \in \mathbf{H}$ with appropriate motion
     parameters and choose the most probable hypothesis. $k$ is the index of the
     appropriate keyframe.
```

To make the proposed algorithm working, we must accept some constraints which assure that we, at least asymptotically, reach the true value of shape parameters. We have not yet mentioned any constraints on the character of the frame posterior probability. Let us denote $\bar{\mathbf{s}}$ true (unknown) shape parameters which maximize the sequence posterior

$$\bar{\mathbf{s}} = \arg\max p(\theta|Z).$$

Let us denote $\mathbf{s}^*_k$ the shape parameters which maximize the frame posterior in a frame k. Clearly, if the $\mathbf{s}^*_k = \bar{\mathbf{s}}$ then the maximization of the frame posterior in an arbitrary frame provides the true value of shape parameters. However, this is too hard constraint. Nevertheless, since the frame posterior is the probability of parameters in the given frame, we should expect it to have the maximum near the $\bar{\mathbf{s}}$.

By the maximization of the second term $p(\mathbf{m}_k, \mathbf{s}|Z_k)$ of (2.1), we can find the maximum of the frame posterior probability $\mathbf{s}_k^*$. This value is not equal to the optimal value of $\bar{\mathbf{s}}$ but we expect to be close to it. By the optimization of the different frames we obtain different values $\mathbf{s}_k^*$. We can evaluate the sequence posteriors given these two different shape parameters by the optimization in the remaining frames. Now we express the probability that after the maximization of shape parameters in K different frames, we obtain at least one vector of shape parameters which is closer to the $\bar{\mathbf{s}}$ then some small ϵ.

More formally, the maxima of frame posteriors are considered to have a Gaussian distribution with the mean $\bar{\mathbf{s}}$ and a covariance σ. The covariance, which corresponds to the model and the likelihood selection, is considered to be as small as possible. Then

$$e = \bar{\mathbf{s}} - \mathbf{s}_k^* = \mathbf{N}(0, \sigma)$$

is of Gaussian distribution too. Note, that we can find ML estimation of σ from different values of $\mathbf{s}_k^*$.

The probability that the maximum $\mathbf{s}_k^*$ of the frame posterior in frame k is closer then ϵ to the true value $\bar{\mathbf{s}}$ is

$$p_{\epsilon,\sigma} = \int_{-\epsilon}^{\epsilon} \mathbf{N}(e, 0, \sigma) de.$$

If we find the $\mathbf{s}_k^*$ in the K frames independently then the probability that the maximum $\mathbf{s}_k^*$ of the frame posterior in at least one of these K frames is closer then ϵ to the true value $\bar{\mathbf{s}}$ is

$$p_{\epsilon,\sigma}(K) = 1 - (1 - p_{\epsilon,\sigma})^K.$$

This function rapidly goes to the 1 in K for reasonable values of ϵ and σ. At the end of the Algorithm 1,we obtain K different values of s_k^* $k \in \mathbf{K}$ which are used to find a ML estimation of σ. One of shape parameters provides the maximal value of the sequence posterior. The $p_{\epsilon,\sigma}(K)$ is the probability that this shape parameters are closer than ϵ to the true value of $\bar{\mathbf{s}}$.

2.5 Experiments

Simulated data

We clarify basic principle in simple example shown in Figure 2.3 in the first experiment. We fit a human arm model (i.e. open kinematic chain of two ellipsoids) into an observation sequence (6 frames). We decided for parameters $\theta_i = (\varphi_i, a)$, where φ_i (motion parameter) is an angle between two main axes and a (shape parameter) is the length of the main semi-axis of the first ellipsoid. The length of the main semi-axis of the second ellipsoid is $1 - a$ and all of the other parameters are fixed. The unknown

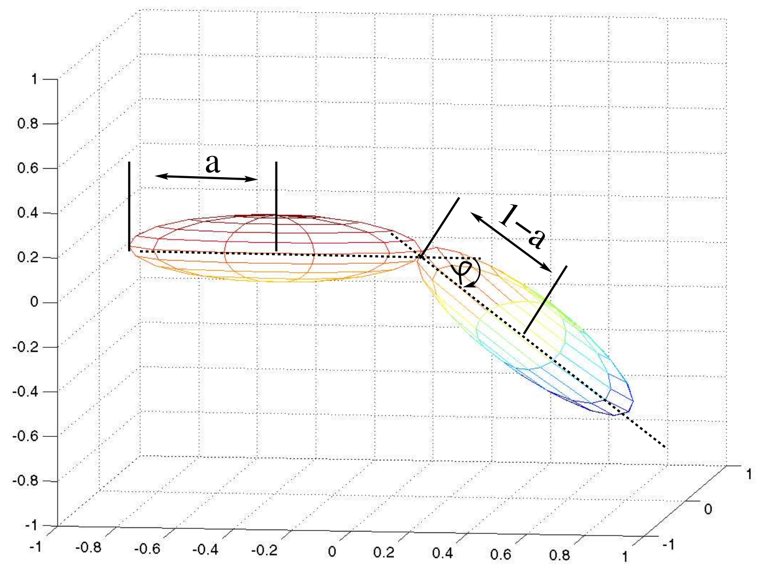


Figure 2.3: Human arm model, $\theta_i = (\varphi_i, a)$.

parameters of the model are $a = 1 - a = 0.5$, and the motion parameters changes frame by frame $\varphi_1 = 0\,\text{rad}$, $\varphi_2 = 0.1\,\text{rad}$, $\varphi_2 = 0.2\,\text{rad}$, ...

According to **Algorithm 1**,we obtain the shape parameter independently in each keyframe (note, that all frames are keyframes in this simple example). A distribution of shape parameter a and input sequence are depicted in Figure 2.4. Shape parameters computed for different frames are, of course, different. Each instance of shape parameters is called hypothesis k. Given the hypothesis, we computes motion parameters of the model for whole sequence and probability of this hypothesis. Values of normalized sequence probabilities for each hypothesis are depicted in Figure 2.5a.

We can see, that first frames do not provide any information about the shape parameters because the posterior is equally distributed (see Figure 2.4). Thus, we obtain $a = 0.18$ from the first frame and the corresponding incorrect vector of motion parameters. Since the third frame shape starts appearing and the correct shape and motion parameters are calculated. True and the most probable values of the angle φ^* are in Figure 2.5b. The most probable shape parameter $a = 0.53$. The approximation is inaccurate due to the simulated noise.

Real sequence

We used the multiview segmented sequence[2] of human gait captured by seven cameras. The length of the sequence is about 200 frames. We select each twentieth frame as

[2]Data have been provided by Lars Mündermann from Stanford University.

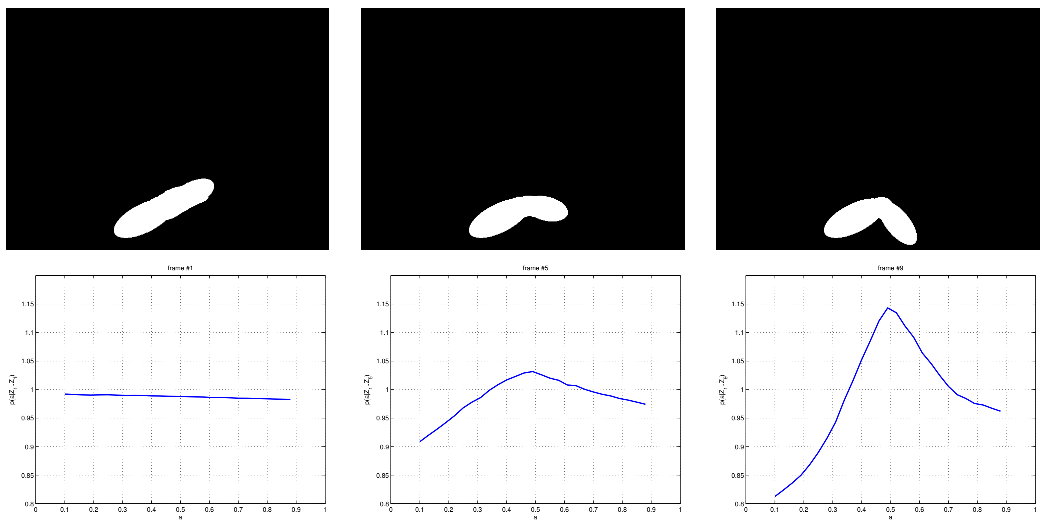


Figure 2.4: The first row shows the observation sequence (silhouettes) in the frames 1,4,6. The second row shows the appropriate sequence probabilities $p(a, \varphi_1^*|Z_1), p(a, \varphi_1^* \dots \varphi_4^*|Z_1 \dots Z_4), p(a, \varphi_1^* \dots \varphi_6^*|Z_1 \dots Z_6)$.

a keyframe (i.e., the 10 keyframes were used). Parameters estimated by our method matched well with the ground truth which was acquired by a complicated, manually intervened, process. All of the shape parameters (length of limbs and sizes of body) were calculated correctly. Motion parameters were computed correctly up to frames about number 120 (see Figure 2.6). The ellipsoidal model is too weak to correctly distinguish arm angles when arms are close to the body and therefore almost invisible in segmented views. It should be noted however, that this is a problem of the weak model not the method itself. Correct motion parameters produce higher (worse) value of the criterial function. After the problematic frame, arms are distinguishable from the body and the motion parameters return to the correct values.

2.6 Conclusions

We proposed a method of human body model fitting into segmented multiview data, which optimizes both shape and motion parameters over the whole sequence. The main contribution is the probabilistic description which, under reasonable constraint, provides a solution near to global optimum. Moreover, we are able to quantify the probability that we have actually reach the global minimum.

2.7 Appendix A

We will show how to decompose sequence probability into a multiplicative form, which is usable for the optimization. The Bayes rule application on sequence posterior

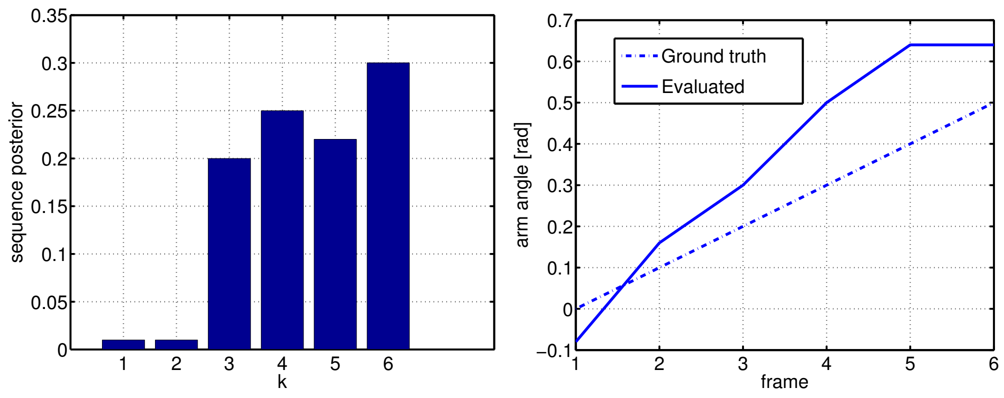


Figure 2.5: Left: Visualization of the probability of a different hypothesis. Right: φ values associated with the most probable explanation and the ground truth.

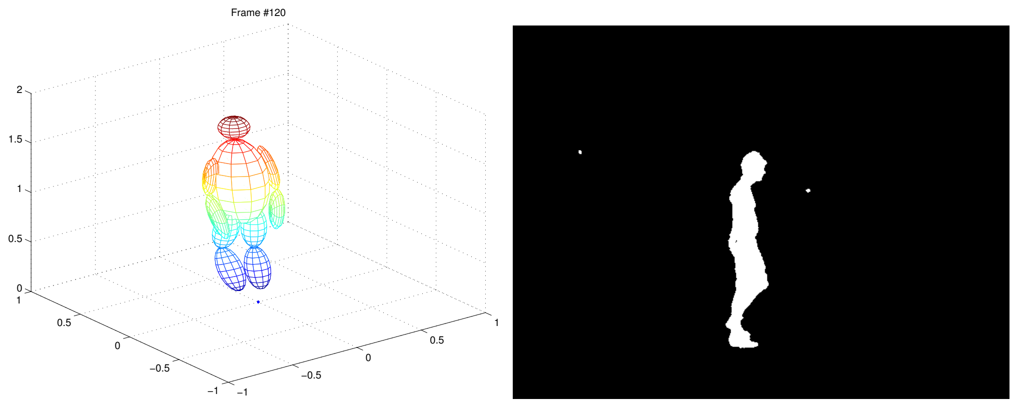


Figure 2.6: Frame 120 - incorrect motion parameters of the arm due to occlusion of arms by the body. Human arm bends to the opposite side to cover inaccuracy of the model.

probability $p(\theta|Z)$ provides

$$p(\mathbf{m}_1, \dots, \mathbf{m}_n, \mathbf{s}|Z_1, \dots, Z_n) =$$

$$p(\mathbf{m}_k, \mathbf{s}|Z_1, \dots, Z_n) p(\mathbf{m}_1, \dots, \mathbf{m}_{k-1}, \mathbf{m}_{k+1}, \dots, \mathbf{m}_n|\mathbf{m}_k, \mathbf{s}, Z_1, \dots, Z_n).$$

The motion and static parameters $(\mathbf{m}_k, \mathbf{s})$ in the frame k are dependent only on the multiview image Z_k. Therefore the first term is reduced to

$$p(\mathbf{m}_k, \mathbf{s}|Z_1, \dots, Z_n) = p(\mathbf{m}_k, \mathbf{s}|Z_k).$$

The Bayes rule is similarly applied for the second term to obtain similar decomposition

$$p(\mathbf{m}_1, \dots, \mathbf{m}_{k-1}, \mathbf{m}_{k+1}, \dots, \mathbf{m}_n|\mathbf{m}_k, \mathbf{s}, Z_1, \dots, Z_n) =$$

$$p(\mathbf{m}_1|\mathbf{s}, Z_1)p(\mathbf{m}_2, \ldots, \mathbf{m}_{k-1}, \mathbf{m}_{k+1}, \ldots, \mathbf{m}_n|\mathbf{m}_k, \mathbf{s}, Z_1, \ldots, Z_n)$$

This expression again consists of two terms, where the second is decomposed by similar chain application of Bayes rule to the form

$$p(\mathbf{m}_2, \ldots, \mathbf{m}_{k-1}, \mathbf{m}_{k+1}, \ldots, \mathbf{m}_n|\mathbf{m}_k, \mathbf{s}, Z_1, \ldots, Z_n) =$$

$$\prod_{i \neq k} p(\mathbf{m}_i|\mathbf{m}_{i-1}, \mathbf{s}, Z_i).$$

Substituting these results to the first equation we obtained the wanted decomposition of the sequence posterior

$$p(\theta|Z) = p(\mathbf{m}_1|\mathbf{s}, Z_1)p(\mathbf{m}_k, \mathbf{s}|Z_k) \prod_{i \neq k} p(\mathbf{m}_i|\mathbf{m}_{i-1}, \mathbf{s}, Z_i).$$

2.8 Appendix B

In preceding Appendix A, we express the sequence probability as the multiplication of the frame posterior probability. Here, we derive how to compute different posterior probabilities.

The frame posterior probability is derived from the generalized Bayesian rule. The joint probability $p(\mathbf{m}_i, \mathbf{m}_{i-1}, Z_i, \mathbf{s})$ can be expressed in different forms based on a different order of decompositions

$$p(\mathbf{m}_i, \mathbf{m}_{i-1}, Z_i, \mathbf{s}) = p(\mathbf{m}_i, \mathbf{s}|\mathbf{m}_{i-1}, Z_i)p(\mathbf{m}_{i-1}|Z_i)p(Z_i)$$

$$p(\mathbf{m}_i, \mathbf{m}_{i-1}, Z_i, \mathbf{s}) = p(Z_i|\mathbf{m}_i, \mathbf{m}_{i-1}, \mathbf{s})p(\mathbf{m}_i, \mathbf{s}|\mathbf{m}_{i-1})p(\mathbf{m}_{i-1}).$$

Therefore

$$p(\mathbf{m}_i, \mathbf{s}|\mathbf{m}_{i-1}, Z_i) = \frac{p(Z_i, \mathbf{s}|\mathbf{m}_i, \mathbf{m}_{i-1}, \mathbf{s})p(\mathbf{m}_i, \mathbf{s}|\mathbf{m}_{i-1})p(\mathbf{m}_{i-1})}{p(\mathbf{m}_{i-1}|Z_i)p(Z_i)}.$$

We do not model any prior knowledge about the probability of the multiview image Z_i and expect it uniformly distributed, i.e. $p(Z_i) = const.$ The motion parameters $\mathbf{m}_{i-1}$ in the frame $(i-1)$ are independent on the image Z_i in the frame i, therefore $p(\mathbf{m}_{i-1}|Z_i) = p(\mathbf{m}_{i-1})$. There are no explicit knowledge about the shape and motion parameters and therefore are considered to be independent too $p(\mathbf{m}_i, \mathbf{s}|\mathbf{m}_{i-1}) = p(\mathbf{m}_i|\mathbf{m}_{i-1})p(\mathbf{s})$. By substituting these expressions and ignoring the constant, we obtain exactly

$$p(\mathbf{m}_i, \mathbf{s}|\mathbf{m}_{i-1}, Z_i) \propto p(Z_i|\mathbf{m}_i, \mathbf{s})p(\mathbf{m}_i|\mathbf{m}_{i-1})p(\mathbf{s}).$$

Similarly, from different forms of the joint probability $p(\mathbf{m}_i, \mathbf{m}_{i-1}, Z_i, \mathbf{s})$, we derive motion $\mathbf{m}_i$ and shape $\mathbf{s}$ joint probability given preceding motion $\mathbf{m}_{i-1}$ and images Z_i. Comparison of equations

$$p(\mathbf{m}_i, \mathbf{m}_{i-1}, Z_i, \mathbf{s}) = p(\mathbf{m}_i|\mathbf{m}_{i-1}, Z_i, \mathbf{s})p(\mathbf{s}|\mathbf{m}_{i-1}, Z_i)p(\mathbf{m}_{i-1}, Z_i),$$

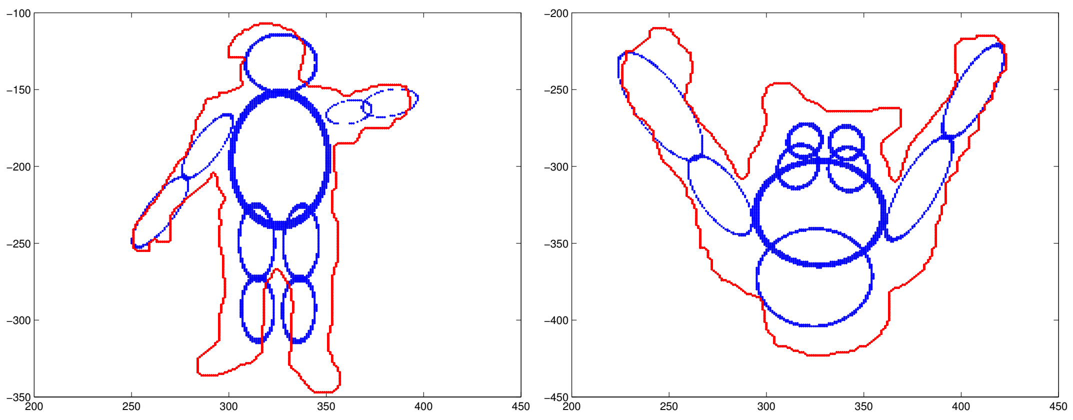


Figure 2.7: The projection of ellipsoidal model to the individual cameras (blue color) and the border of segmented image (red color)

$$p(\mathbf{m}_i, \mathbf{m}_{i-1}, Z_i, \mathbf{s}) = p(Z_i|\mathbf{s}, \mathbf{m}_i, \mathbf{m}_{i-1})p(\mathbf{m}_i|\mathbf{s}, \mathbf{m}_{i-1})p(\mathbf{s}, \mathbf{m}_{i-1})$$

provides

$$p(\mathbf{m}_i|\mathbf{m}_{i-1}, Z_i, \mathbf{s}) = \frac{p(Z_i|\mathbf{s}, \mathbf{m}_i)p(\mathbf{m}_i|\mathbf{m}_{i-1})p(\mathbf{s}, \mathbf{m}_{i-1})}{p(\mathbf{s}|\mathbf{m}_{i-1}, Z_i)p(\mathbf{m}_{i-1}, Z_i)}.$$

Considering independences mentioned above,

$$p(\mathbf{m}_i|\mathbf{m}_{i-1}, Z_i, \mathbf{s}) = \frac{p(Z_i|\mathbf{s}, \mathbf{m}_i)p(\mathbf{m}_i|\mathbf{m}_{i-1})p(\mathbf{s})p(\mathbf{m}_{i-1})}{p(\mathbf{s})p(\mathbf{m}_{i-1})p(Z_i)}.$$

Simplification of this expression provides

$$p(\mathbf{m}_i|\mathbf{m}_{i-1}, \mathbf{s}, Z_i) \propto p(Z_i|\mathbf{m}_i, \mathbf{s})p(\mathbf{m}_i|\mathbf{m}_{i-1}).$$

2.9 Appendix C

The frame posterior probability (2.2), (2.3) consists of the likelihood and priors. The priors are probabilities expressing knowledge given in advance and are considered to have a simple distribution (i.e. uniform or Gaussian). The likelihood expresses the probability that the image was induced by parameters $(\mathbf{m}_i, \mathbf{s})$. Such probability can be expressed in many ways. We chose it proportional to the coverage of borders of silhouettes by model projection in the individual cameras. The model projection and borders of silhouettes are depicted in the Figure 2.7. The sum of distances of pixels from the model projection over all cameras is inversely proportional to the frame posterior probability. The distance of the pixel from the model projection is considered to be the distance of the pixel from the nearest ellipse. The Euclidean distance between point $\mathbf{x}$ and ellipse requires solving of quartic equation which may have up four solutions, requiring the one with the minimum distance to be determined [73]. To avoid the complexity of evaluating the true Euclidean distance, we proposed an approximation measure [102].

3 Pictorial structures

This chapter deals with learning and fitting an articulated model to single images. I started the work with my master students [27], see also [61], further evolved and combined with graph-cut for an unsupervised learning of body part detectors [71]. The unsupervised learning was quite novel at that time. We achieved promising results, however, the work has not been published as we got overrun by contributions of V. Ferrari et al. [30], later also [29].

An articulated model of the human body is assembled from individual parts (head, torso, limbs, etc). A human body model is represented as a collection of the parts arranged in a deformable configuration [28]. Single body parts are detected using color characteristics that are gained from training examples. We advance the standard algorithm by detecting shapes of multiple scales. The method is accelerated by assuming vertical symmetry of a human body. A sliding-window human detector [16] is applied in order to reduce the state space. We propose a method for unsupervised learning of color appearance of the human body parts. This approach makes the detection (using matching of the pictorial structures) more robust. The resulting method for the human body modeling integrates the fast human detector, the pictorial structures matching and image segmentation based on graph cuts.

3.1 Probability model of a pictorial structure

The human body is represented as a collection of parts which are arranged in a deformable configuration. Local visual properties are encoded in models for individual parts, deformable configuration is represented by spring-like connections between certain pairs of parts. Parts correspond to significant rigid parts of the human body, connections correspond to joints. Figure 3.1 shows the model of the human body.

The model is described as an undirected graph $G = (V, E)$. The vertices $V = \{v_1, v_2 ... v_n\}$ correspond to individual parts and $(v_i, v_j) \in E$ only when a joint between vertices v_i and v_j exists. An instance of the object is given with a configuration $L = \{l_1, l_2, ... l_n\}$, where $l_i = (x_i, y_i, s_i, \theta_i)$ specifies a location of a part v_i. x_i, y_i are coordinates of the center of the part in the picture, θ_i is the orientation of the part and s means foreshortening of the part only for its longer side. It is caused with a scaled orthographic projection.

The best match is found as the one which minimizes the energy function of observing an object in the location L given image I:

$$L^* = \arg\min_L \Big(\sum_{i=1}^{n} m_i(l_i) + \sum_{(v_i, v_j) \in E} d_{ij}(l_i, l_j) \Big), \tag{3.1}$$

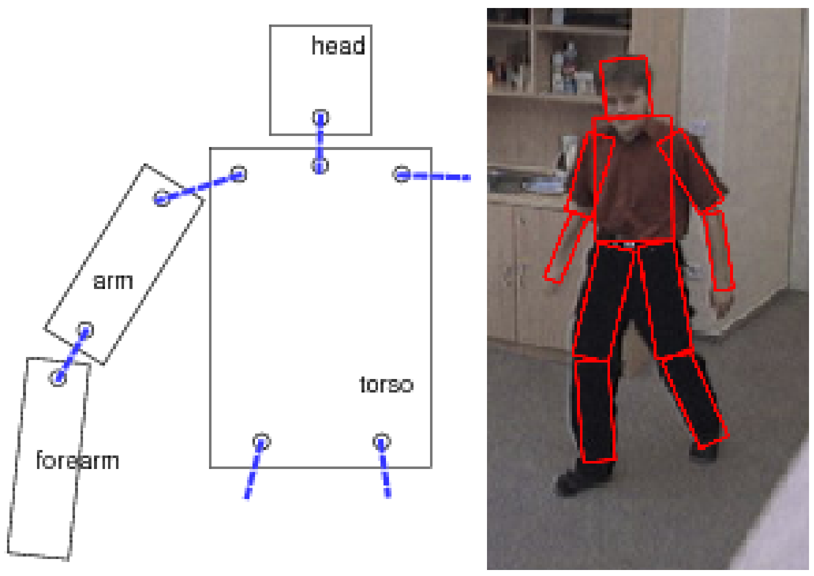


Figure 3.1: The model of the human body. The body is assembled of rectangular parts, which are connected with flexible joints in deformable configuration. Size of the rectangles is deformed by the parameter scale and the foreshortening.

where n is number of parts, $m_i(l_i)$ measures the degree of mismatch, when part v_i is placed the location l_i in the picture, $d_{ij}(l_i, l_j)$ measures degree of deformation of the model, when a part v_i is placed at location l_i and part v_j is placed on location l_j in the image.

As suggested in [28], we do not minimise directly the function (3.1), but maximize the posterior probability

$$p(L|I,\theta) \propto \left(\prod_{i=1}^{n} p(I|l_i,u_i) \prod_{(v_i,v_j)\in E} p(l_i,l_j|c_{ij}) \right) , \tag{3.2}$$

where $p(I|l_i, u_i)$ is the probability of observing picture, u_i are learnable parameters, and $p(l_i, l_j|c_{ij})$ is the probability of the mutual position of two parts. The negative logarithm of (3.2) is equal to the energy function (3.1)

The entire task can be divided into three independent partial tasks according to equation (3.2):

1. Computing of probability $p(I|l_i, u_i)$ and learning of its parameters from training examples.

2. Computing of probability $p(l_i, l_j|c_{ij})$ and learning of its parameters from training examples.

3. Efficient searching of the state space and finding the match, which maximize the probability (3.2). The MAP estimate is used.

3.2 Detector of parts, color based segmentation

The color based segmentation [72] suggests to segment individual parts using information about their color. The projection of one part is approximated as a rectangle. Width of the rectangle at a certain scale is fixed. The maximal length of the part corresponds to a length of a limb seen in the picture, when it is oriented perpendicularly to the optical axis. The length of the projection can vary and it is modeled by foreshortening. A human body is modeled by the projection of a part as a rectangle parametrized by parameters (x, y, s, θ), where (x, y) are coordinates of the centre in an image, $s \in \langle 0, 1\rangle$ is the amount of foreshortening and θ is an angle of an orientation of the part. Color histograms of foreground and background are learned from training examples. Histograms are three-dimensional in RGB space. Probability, that a pixel of the given colour belongs to the given part, is evaluated with a quadratic logistic regression classifier [51]. Number of foreground pixels in a given bin of a histogram serves as the number of positive examples; number of background pixels in that bin as negative examples. The quadratic logistic regression classifier is a special example of generalised linear classifiers,

$$\operatorname{logit}(p) = \ln\left(\frac{p}{1-p}\right) = \mathbf{X}^\top \cdot \beta, \tag{3.3}$$

where $\beta = [\beta_0, \beta_1, ..., \beta_n]^\top$ is a vector of $n+1$ parameters, which must be determined[1], and $\mathbf{X} = \left[1, r, g, b, r^2, g^2, b^2, rg, rb, gb\right]^\top$ are variables. r, g, b mean values of colors and can obtain only values from a set $\{1, 2, \ldots, 8\}$ - it corresponds to the number of bins. According to (3.3):

$$p = \frac{e^{\mathbf{X}^\top \cdot \beta}}{1 + e^{\mathbf{X}^\top \cdot \beta}}\,. \tag{3.4}$$

Each pixel in the image has its probability, that it is a component of a given part. To avoid thresholding of this data, we use directly probabilities for next processing, because it can be difficult to determine threshold.

Parts are represented by rectangles consisting of a foreground rectangle and surrounding area, which shows Figure 3.2. Each pixel in the image has its probability, that it is a component of the given part. The probability of observing part in the location l_i in the given image is computed:

$$p(I|l_i, u_i) = e^{-[(area_1 - count_1) + count_2]/s}, \tag{3.5}$$

where $count_1$ is the sum of pixels masked with a part weighted with probabilities, that they belong to the foreground, $count_2$ is the sum of pixels masked with a surrounding of a part weighted with probabilities, that they belong to the foreground. In the smaller rectangle, misclassified pixels are summed $(area_1 - count_1)$ (probabilities that they

[1]We use the `gmlfit` function in Matlab.

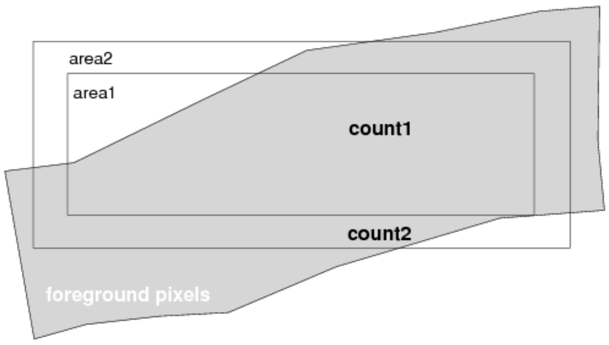


Figure 3.2: Model of one rectangular part with the surrounding area.

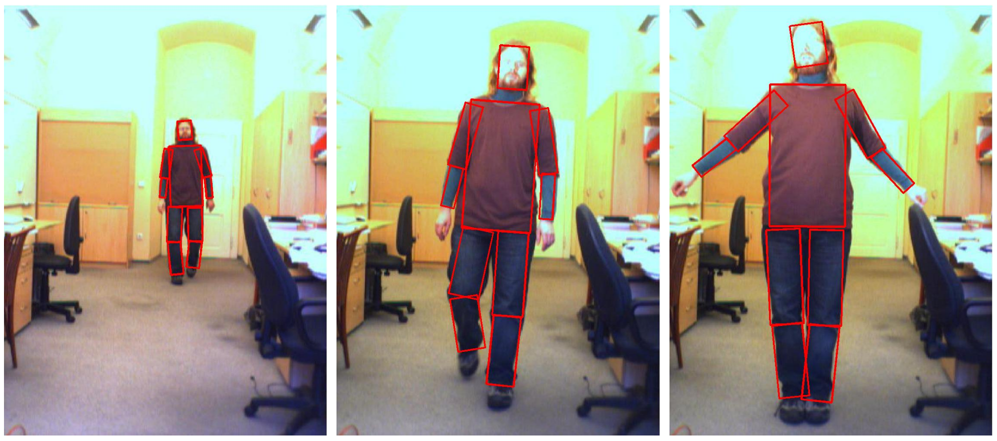


Figure 3.3: Some of interactively labelled training examples with multi-scale height of a person. The person has a symmetric clothing, which is used to speed up the algorithm. Note the shin/calf foreshortening at the mid image.

belong to the background). Division with the scale s is used because of normalisation of results.

3.3 Learning process

The color model of parts and the human model are learned from labeled training images. A height of a person is changed, when she or he moves from/to a camera. Figure 3.3 shows some labeled training examples with various heights of a person.

We suppose that the foreshortening parameter of a human torso varies only gently, because the persons stay only in an upright position in the training data. The scale value of a training example is computed from the height of the torso. The foreshortening of the human torso is set to 1. The proportions of other parts are recomputed from the scale of the whole human body. The scale of one training example is com-

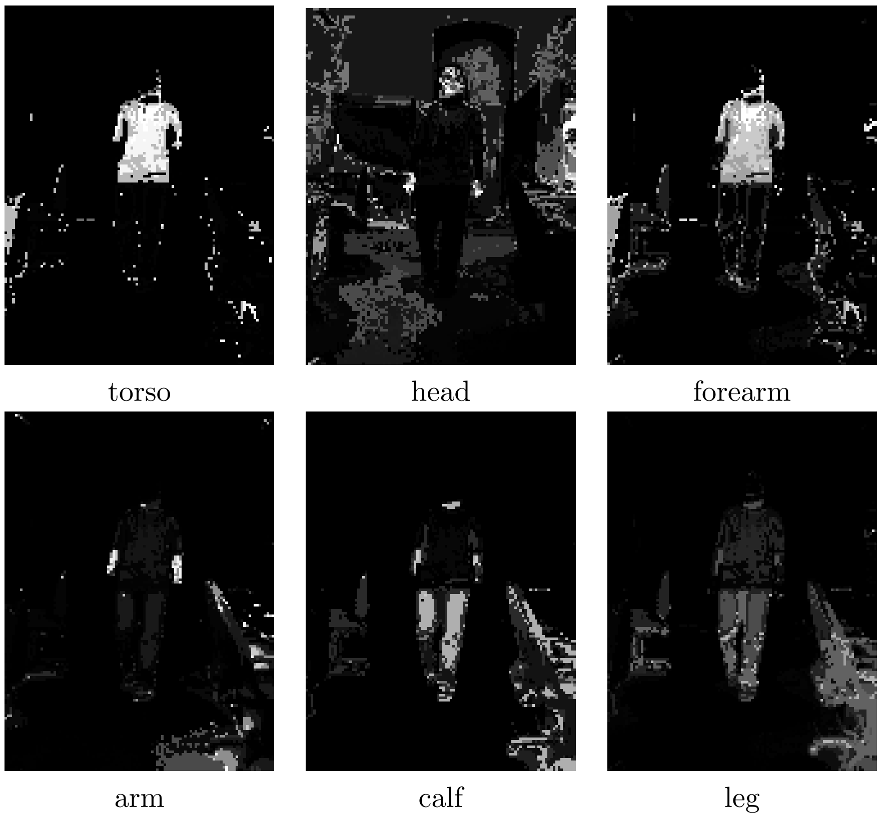


Figure 3.4: Example of color based segmentation for each part. The color of a pixel represents the probability that pixel belongs to each part. White corresponds to the probability 1, black to probability 0.

puted as the nearest value to predefined scales values set. A color model is learned for each scale.

To speed up the algorithm, we expect that the human body and the cloth color are symmetric along the vertical axis. We also presume that the part location probabilities are the same for left and right body part. It means that we compute probabilities for 6 body parts (torso, head, arm, forearm, calf, thigh) instead of 10 parts (torso, head, left/right arm, left/right forearm, left/right calf, left/right thigh). Figure 3.4 shows the probability of the fact that each pixel of image belongs to each part. Figure 3.5 shows the probability of the location of some parts in the position of the discretized state space.

Sometimes it is impossible to distinguish parts only by their colours. Many false detections are found. If the segmentation is wrong, it is almost impossible to find parts like the head. Location of these parts can be located by using the pictorial structures. A deformable human body model is learned from all training examples over all scales, because the pose of the human body is the same when it is far or near to the camera.

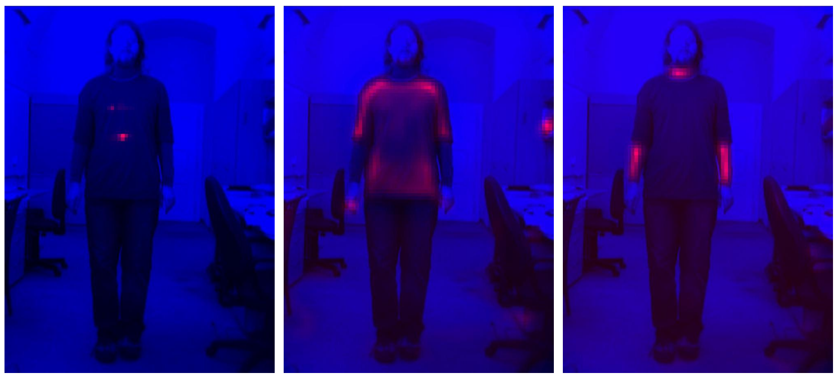


Figure 3.5: Probabilities of parts location in a particular position of the discretized state space. From left; torso, arm and forearm. The degree of red represents the probabilities.

Considering multiple scales slows down the computation. The probabilities must be computed separately for each scale value. The maximum a posteriori estimate (MAP) selects the best configuration from all the position and scales.

3.4 Unsupervised learning of the appearance

The articulated model represents a general human body. However, the human appearance may vary significantly. In order to make the pictorial modeling reasonably applicable, we need an unsupervised learning of the appearance. We propose to use an image segmentation using the minimal cut of a graph. The Graph Cut [9] algorithm finds the optimal foreground and background labeling as an energy minimization problem. The minimum cut can be computed very efficiently by max the flow algorithms[2].

Learning of appearance of body parts is divided into 4 steps:

1. A human body is detected in an image using the human detector based on the HoG descriptors [98].
2. A selected area is segmented by the Graph Cut algorithm. The color appearance of the whole body is learned from the segmentation result.
3. A configuration of a human body in the detection window (step 1) is found by the pictorial structure matching.
4. A color appearance of each body part is learned using the results of the pictorial structure matching.

[2]In our work we use the implementation of the Graph Cut available on `http://www.csd.uwo.ca/faculty/olga/code.html` and the MATLAB wrapper available on `http://www.wisdom.weizmann.ac.il/~bagon/matlab.html`.

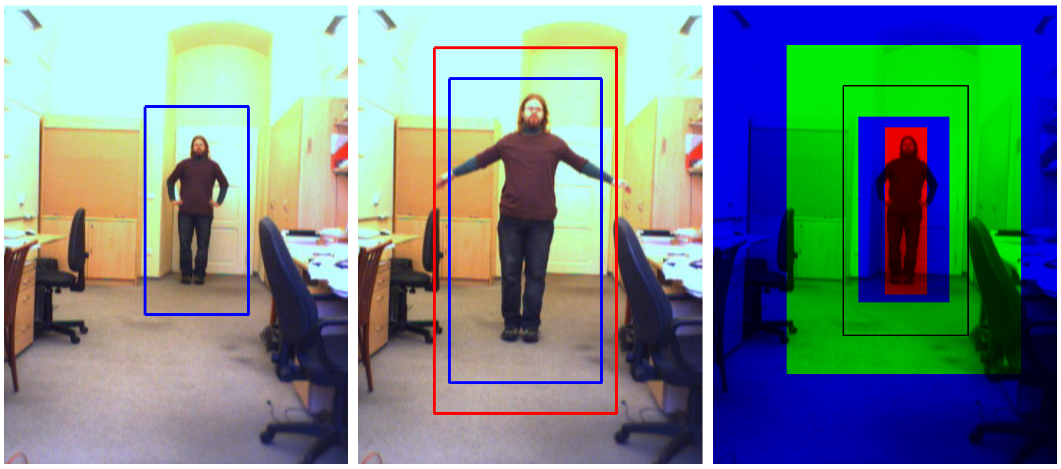


Figure 3.6: The human body detection. Left: The blue rectangle is the selected detection window. Middle: Blue depicts the original detection. Red is the resized detection window. Forearms fit to the detection window now. Right: Areas for computing color histogram of foreground and background. Red pixels are used for learning of foreground color histogram. Green pixels are for learning of background color histogram. Blue pixels are not used. Black rectangle is the detection window.

The human body is detected by the fast human detector using HoG descriptors [98] and the detection window is selected by the averaging of all detections. Figure 3.6 shows the detected human body in the image. Forearms may go outside the detection window. Therefore the detection window is expanded by multiplying its height and width by a constant greater than 1. We experimentally set the expansion to 1.2. Figure 3.6 shows the original and resized detection window.

The probabilities that a pixel belongs either to the foreground or the background are computed after the human body is detected in the image. A normalised color histogram is used to estimate the probabilities that a pixel with RGB color belongs to the foreground and the background. There are two computed color histograms, one for the foreground and one for the background. The RGB color was discretized into $8 \times 8 \times 8$ bins color histograms computation.

The image is divided into three areas. The color histogram of the foreground is learned from the first area, the color histogram of the foreground is learned from the second area and third area is not used to learn any color histogram. We suppose that the foreground pixels are inside the detection window. However, background pixels may be inside the detection window, too. Therefore we learn the color histogram of foreground from the inside the detected area, but not from the inside area near to the border of the detection window. We suppose that background pixels with similar color values as background pixels inside of the detection window are near the border of detection window. Therefore the color histogram of the background is computed from pixels near the border of the detection window. Figure 3.6 (right) shows areas

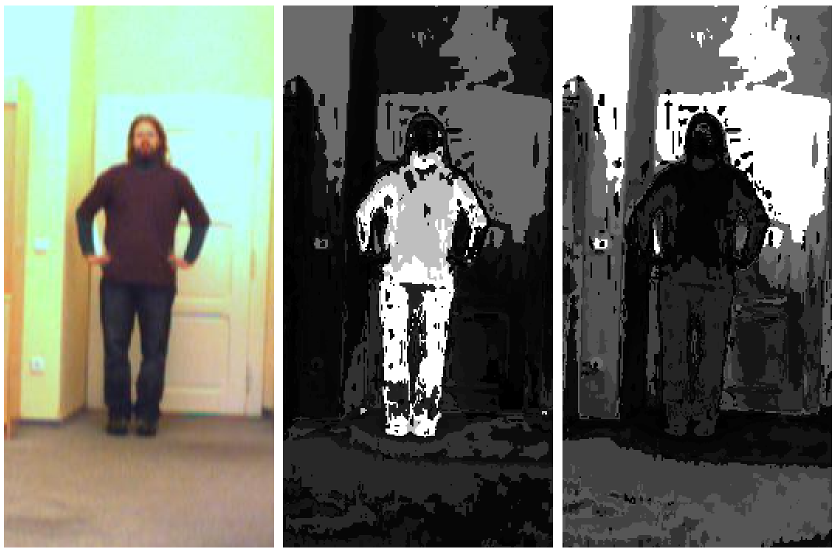


Figure 3.7: Initialization of the Graph Cut algorithm. Left: the original detection window. Middle: the probabilities that pixels belong to the foreground. Right: probabilities that pixels belong to the background. Black corresponds to 0, white to the maximal value.

assigned to learn the color histogram.

Figure 3.7 shows the probabilities initialization that pixels belong to the foreground and background in a detection window. The Graph Cut segmentation can start after learning the color histograms. Each pixel outside the detection window is assigned to the background.

New color histograms of the foreground and background pixels are computed from the Graph Cut segmentation result. The Graph Cut refines the initial rectangular segmentation. The quadratic logistic regression classifier is learned from the color histograms. Contrary to the hand-labeled examples approach, only one classifier for all the parts is learned. Figure 3.8 shows probabilities that pixels belong to the foreground in the detection window computed by the quadratic logistic regression classifier.

Next step to the unsupervised learning of the appearance of human body parts is the search of a human body configuration in the detection window. The used method based on the pictorial structures matching is very similar to the one used for hand-labeled exemplars, however, with the following little differences:

- Only the deformable human body model is learned by labeling of training examples.
- The color model is learned from the Graph Cut segmentation of the actual image. It is the same for all parts of human body. It means that each body part does not have own color model. There is only one color model.
- The value of multiple scale is estimated from the size of the detection window.

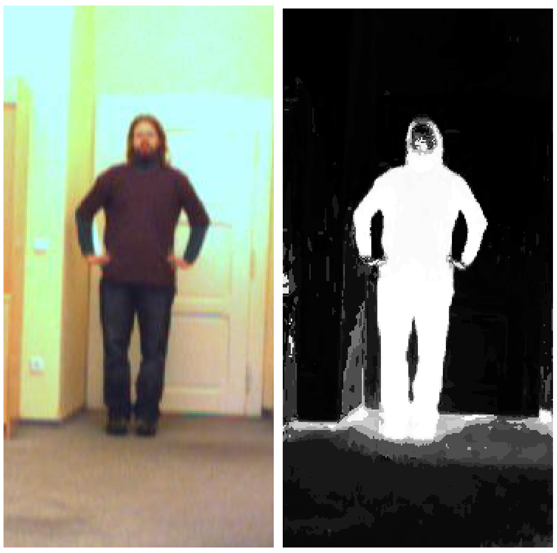


Figure 3.8: Probabilities of pixel belonging to the foreground. Left is the original detection window, right is the probability of each pixel belong to the foreground.

In practice we use 3 closest values of multiple scales, because the estimation of high of human body does not be to precise.

- The posterior probability is sampled instead of using MAP.

Sampling from the posterior probability is used because MAP estimation could miss Sometimes the probability distribution, which is computed with the matching algorithm, has more than one peak. This can be caused by several reasons. For example, the self-occlusion of a human body. The Graph Cut segmentation may not be precise, because some parts of the human body can be segmented like background pixels. A stochastic method is used and it is sampled from the posterior probability according to the Monte Carlo method. That means that more probable locations in the space are sampled more frequently. Some locations have a minimal chance that they will be sampled. For more details about the sampling procedure refer to [27]. Figure 3.9 shows some selected samples from the posterior probability. One best sample is not selected as opposed to [27]. For each pixel of the image the frequency of the occurrence from the samples (of each part of the human body) is computed. The result of this operation is a probability map (one for each part of human body), in which each pixel matches the frequency of the occurrence of the part. The frequency map is normalised to the interval from 0 to 1. Bottom row of Figure 3.9 shows the frequency map for the torso and head. For other parts, the frequency map is similar. The last step of the learning of a human body appearance is to learn the color models of each part. A color histogram of the foreground and background is computed for each part of a human body. The counts of color values are weighted by the frequency of the occurrence map of each part, when foreground histograms are computed. The inverted

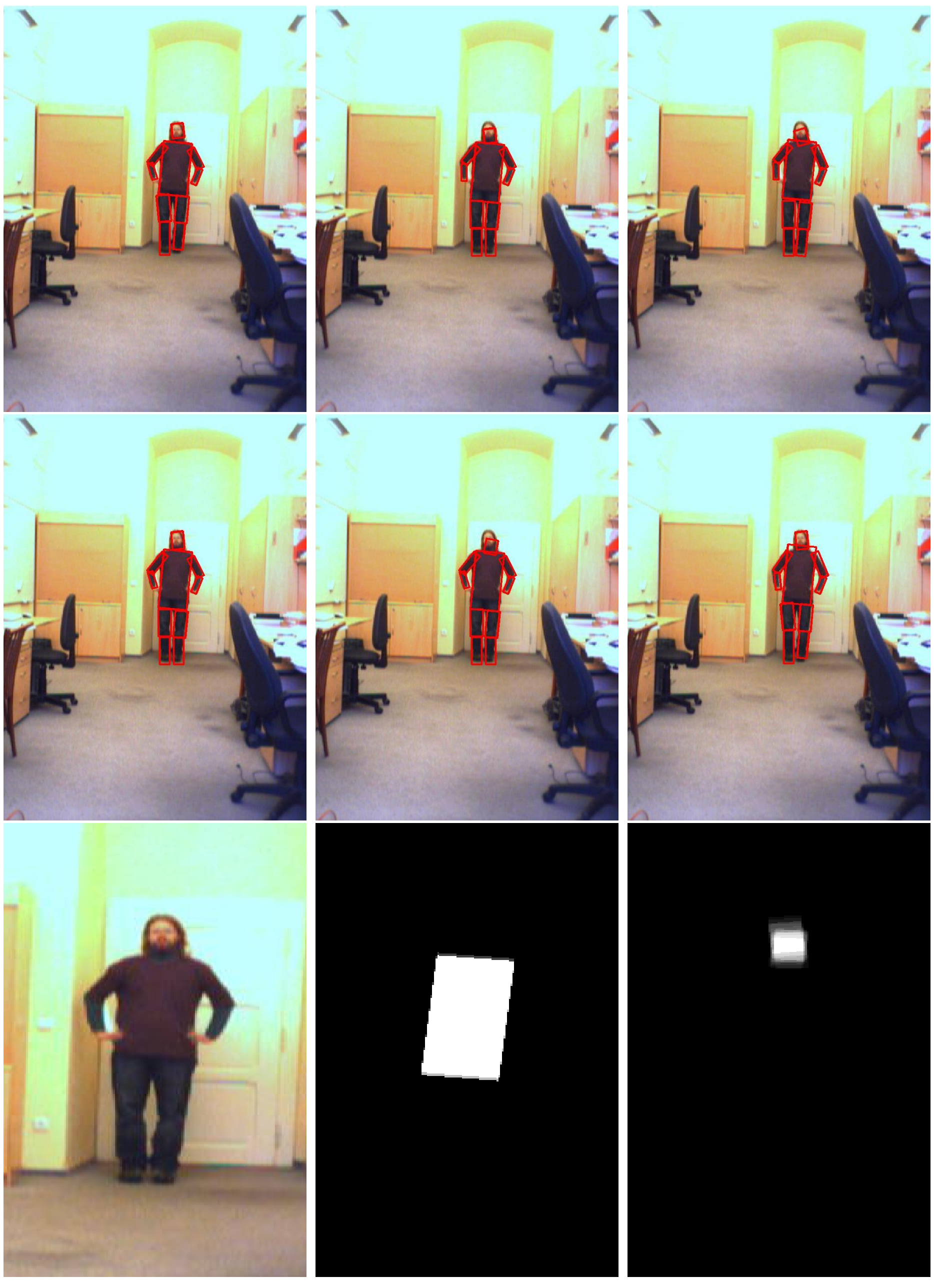


Figure 3.9: Examples of sampling from the posterior probability. The torso is sampled the same often in this case. The position of the head is different in each sample. The bottom row shows Frequency of occurrence of the parts normalised to the interval from 0 to 1. Left: original image. Middle: frequency of the occurrence of the torso. Right: the head occurrence frequency. The torso is sampled in a similar position in all the samples.

frequency occurrence map is used for computing of histograms of the background. The quadratic logistic regression classifiers are computed from the foreground as well as the background histograms for each body part.

4 Multiview 3D tracking with a 3D model constructed incrementally

This chapter essentially derives an extension of a gradient based technique used in 2D image onto 3D models. The work has been published as conference paper [104].

We propose a multiview tracking method for rigid objects. Assuming that the part of an object is visible in at least two cameras, a partial 3D model is reconstructed in terms of a collection of small 3D planar patches of an arbitrary topology. The 3D representation, recovered fully automatically, allows to formulate tracking as the gradient minimization in the pose space (translation, rotation). As the object moves, the 3D model is incrementally updated. A virtuous circle emerges: tracking enables composition of the partial 3D model; the 3D model facilitates and robustifies the multiview tracking.

We demonstrate experimentally that the interleaved track-and-reconstruct approach tracks successfully a 360 degrees turn-around and a wide range of motions. Monocular tracking is also possible after the model is constructed. Using more cameras, however, increases stability in critical poses and moves significantly. We demonstrate how to exploit the 3D model to increase stability in the presence of uneven and/or changing illumination.

4.1 Introduction

Existing multiview approaches mostly represent objects as blobs. A blob representation assumes that the appearance of an object does not change significantly when the object rotates. The global object position is sought and the methods do not attempt to recover the *orientation* of the object [31,63].

Most *model-based tracking* methods use 3D models prepared offline. An overview of such methods was published by Lepetit et al. [53]. Vacchetti et al. [92] propose a tracker based on matching with keyframes. The method demonstrates impressive results on out-of-plane rotation data. Still, it cannot track complete turn of the object and needs the offline manual selection of keyframes which are essential for its stability. Muñoz et al. [64] suggest a method that track even deformable objects. Their model is composed of small textured planar patches, a set of shape bases, and a set of texture bases. The tracking procedure needs a reference image and optimizes over local shape deformations, color/texture changes and overall motion. Results on real data show successful tracking only of small variations in object pose and negligible local deformations.

Several approaches build elaborated 3D models from multiple views. The methods rely heavily on carefully constructed and expensive setup and require special scene arrangement since they are based on scene/object segmentation [10,49,60,95]. Würmlin et al. [95] propose dynamic 3D point samples for streaming 3D video. This point based representation somehow resembles our model. However, the method does not track object and needs many cameras and very precise pixel-wise motion segmentation.

We propose a combined method that tracks objects in 3D and constructs a point based appearance model simultaneously. Our primary interest is in object tracking and detection. Our model is rather simple, a set of 3D points associated with 3D orientation and albedo. Despite its simplicity, the model is rich enough for recovering orientation of the object. The tracking can follow a complete 360 degree turn of object. Rothganger et al. [75] also compose a 3D model from small planar patches. The patches are reconstructed from multiview correspondences. Objects are photographed an object from several viewpoints, corresponding image patches are found by affine covariant feature matching. Finally, patches are reconstructed in 3D. In fact, it would be possible to use this model in our tracking. Any complete off-line built model [66] could be used, too.

Cobzas and Jagersand [12] propose a monocular, registration-based, 3D camera tracking of the planar 3D patches. The 3D planar patches are estimated from tracks. Although the formulation of the tracking resembles our method, there are several differences. The patch based model is initialized at the beginning of the sequence (in about 100 frames) by using a standard 2D patch based tracker. Then the algorithm switches to tracking and refines the model using 3D model-based tracking. Cobzas et al estimate the camera pose, assuming a rigid scene. Unlike our method which models illumination changes, Cobzas et al. assume constant illumination and intensity of observed points. Our method builds the model from the very beginning of the sequence. Tracked objects change their position and orientation w.r.t. to light sources. In this case, constant pixel intensities cannot be assumed even for Lambertian surfaces and our method reflects this.

4.2 3D tracking

An object O is modelled as a triplet (X, α, N) where X is a set of 3D points, $\alpha\colon X \to \mathcal{R}$ assigns albedo and $N\colon X \to \mathcal{S}^2$ a normal to each point $\mathbf{x} \in X$, where $\mathcal{S}^2$ is a sphere. While tracking, intensity $T(\mathbf{x})$ of point $\mathbf{x}$ in a given frame is predicted from its albedo $\alpha(\mathbf{x})$ and an estimated illumination as detailed in section 4.3.

Assuming object rigidity, the motion of points $\mathbf{x} \in X$ between two time instances t_1 and t_2 is

$$\mathbf{x}^{t_2} = \mathtt{R}\mathbf{x}^{t_1} + \mathbf{d},$$

where $\mathtt{R}$ represents rotation and $\mathbf{d}$ translation. When the rotation is small [40] (e.g. between two consecutive video frames), the motion equation simplifies to

$$\mathbf{x}^{t} = (\mathtt{I} + \mathtt{D})\mathbf{x}^{t-1} + \mathbf{d}, \tag{4.1}$$

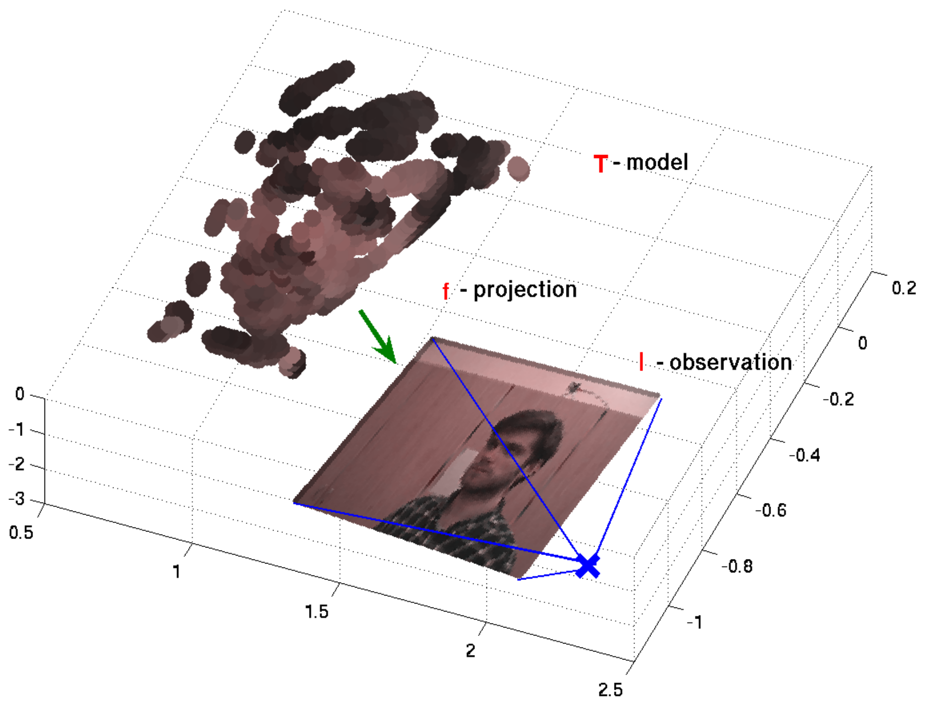


Figure 4.1: Model (template) T is projected by projection function f and compared to the current observation I.

where the rotation matrix $\mathtt{R}$ is replaced by the antisymmetric matrix $\mathtt{D}$ and an identity matrix $\mathtt{I}$. The matrix $\mathtt{D}$ is defined by three parameters $\mathbf{u} = [D_1, D_2, D_3]^T$;

$$\mathtt{D} = \begin{bmatrix} 0 & D_3 & -D_2 \\ -D_3 & 0 & D_1 \\ D_2 & -D_1 & 0 \end{bmatrix}.$$

Tracking in 3D is defined as the process of finding motion parameters $\mathtt{D}, \mathbf{d}$ minimizing the following image dissimilarity

$$\sum_{\mathbf{x}\in X} \Big[T(\mathbf{x}^{t-1}) - I\Big(f(\mathbf{x}^t)\Big)\Big]^2, \tag{4.2}$$

where $I : \mathcal{R}^2 \to \mathcal{R}$ assigns intensity to each pixel, $T : X \to \mathcal{R}$ assigns intensity to each 3D point. The projection function $f : \mathcal{R}^3 \to \mathcal{R}^2$ maps 3D points to image coordinates and depends on internal and external parameters of the camera, see Appendix A for details.

Substituting from equation (4.1) for $\mathbf{x}^t$ in the dissimilarity function (4.2) and simplifying notation by setting $\mathbf{x}^{t-1} = \mathbf{x}$, a cost function in six unknowns is obtained

$$J(\mathbf{u},\mathbf{d}) = \sum \Big[T\left(\mathbf{x}\right) - I\big(f\left(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d}\right)\big)\Big]^2, \tag{4.3}$$

where the sum is over all $\mathbf{x} \in X$ as in (4.2); starting from (4.3) the summation range is omitted for brevity. We seek motion parameters $\mathbf{u}$ and $\mathbf{d}$ that minimize dissimilarity $J(\mathbf{u},\mathbf{d})$. At the minimum, the partial derivatives with respect to all variables must be zero:

$$\frac{\partial J(\mathbf{u},\mathbf{d})}{\partial \mathbf{d}} = \mathbf{0}, \quad \frac{\partial J(\mathbf{u},\mathbf{d})}{\partial \mathbf{u}} = \mathbf{0},$$

which yields the following two vector equations

$$\sum \Big[T(\mathbf{x}) - I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big)\Big] \frac{\partial I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big)}{\partial \mathbf{d}} = \mathbf{0}, \tag{4.4}$$

$$\sum \Big[T(\mathbf{x}) - I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big)\Big] \frac{\partial I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big)}{\partial \mathbf{u}} = \mathbf{0}, \tag{4.5}$$

There is no closed-form solution for $(\mathbf{u},\mathbf{d})$. We therefore apply Newton-Raphson minimization, approximating $I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big)$ by its first-order Taylor expansion

$$I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big) \approx I\big(f(\mathbf{x})\big) + \mathbf{g}^T(\mathtt{D}\mathbf{x} + \mathbf{d}), \tag{4.6}$$

where

$$\mathbf{g}^T = I'^T\big(f(\mathbf{x})\big)f'(\mathbf{x}); \tag{4.7}$$

$I' : \mathcal{R}^2 \to \mathcal{R}^2$ is the gradient of image I and $f' : \mathcal{R}^3 \to \mathcal{R}^{2\times 3}$ is the Jacobian of the projection function f.

Differentiating the linear approximation (4.6) leads to

$$\begin{aligned} \frac{\partial I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big)}{\partial \mathbf{d}} &\approx \mathbf{g}, && (4.8) \\ \frac{\partial I\big(f(\mathbf{x} + \mathtt{D}\mathbf{x} + \mathbf{d})\big)}{\partial \mathbf{u}} &\approx \frac{\partial \mathbf{g}^T\mathtt{D}\mathbf{x}}{\partial \mathbf{u}}. && (4.9) \end{aligned}$$

Applying the approximations (4.8), (4.9), equations (4.4), (4.5) are simplified to

$$\begin{aligned} \sum \big[T(\mathbf{x}) - I\big(f(\mathbf{x})\big) - \mathbf{g}^T\mathtt{D}\mathbf{x} - \mathbf{g}^T\mathbf{d}\big]\mathbf{g} &= \mathbf{0}\,. && (4.10) \\ \sum \big[T(\mathbf{x}) - I\big(f(\mathbf{x})\big) - \mathbf{g}^T\mathtt{D}\mathbf{x} - \mathbf{g}^T\mathbf{d}\big]\frac{\partial \mathbf{g}^T\mathtt{D}\mathbf{x}}{\partial \mathbf{u}} &= \mathbf{0}, && (4.11) \end{aligned}$$

Simple algebraic manipulations confirms that the following two identities hold

$$\begin{aligned} \mathbf{g}^T\mathtt{D}\mathbf{x} &= (\mathbf{g} \times \mathbf{x})^T\mathbf{u}, \\ \frac{\partial \mathbf{g}^T\mathtt{D}\mathbf{x}}{\partial \mathbf{u}} &= (\mathbf{g} \times \mathbf{x})\,, \end{aligned}$$

where $\times$ is the cross product. Equations (4.11) and (4.10) can be compactly represented as a system of six linear equations $\mathtt{A}$.

$$\mathtt{A}\begin{bmatrix} \mathbf{u} \\ \mathbf{d} \end{bmatrix} = \mathbf{b}\,, \tag{4.12}$$

where

$$\begin{aligned} \mathtt{A} &= \sum \begin{bmatrix} (\mathbf{g} \times \mathbf{x})(\mathbf{g} \times \mathbf{x})^T & (\mathbf{g} \times \mathbf{x})\mathbf{g}^T \\ \mathbf{g}(\mathbf{g} \times \mathbf{x})^T & \mathbf{g}\mathbf{g}^T \end{bmatrix}, && (4.13) \\ \mathbf{b} &= \sum [T(\mathbf{x}) - I(f(\mathbf{x}))] \begin{bmatrix} (\mathbf{g} \times \mathbf{x}) \\ \mathbf{g} \end{bmatrix}. && (4.14) \end{aligned}$$

Assuming regular $\mathtt{A}$, the solution approximately minimizing equation $J(\mathbf{u}, \mathbf{d})$ is

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{d} \end{bmatrix} = \mathtt{A}^{-1}\mathbf{b}\,. \tag{4.15}$$

The 6×6 matrix $\mathtt{A}$ consists of four 3×3 sub-matrices and is block-wise symmetric. Unknown motion parameters $\mathbf{d}$, $\mathbf{u}$ are both 3×1 column vectors and $\mathbf{b}$ is a 6×1 column vector.

At least six points are required for rank$(\mathtt{A}) = 6$. In practice, many more points are visible. If the object is weakly textured back-projected image derivatives $\mathbf{g}$ may get close to zero and matrix $\mathtt{A}$ becomes nearly singular. Texture properties needed for reliable tracking of the object are discussed in [80]. Unlike [80], we optimize over the whole object not just over a small patch.

Newton-Raphson iterations are carried out until convergence or a maximum number of steps N. Experiments showed the process converged usually in $8 - 10$ iterations. Convergence may require more iterations when the motion is fast, so N was set to 20.

The tracking method was derived for an intensity image and a single camera. Extension to RGB tracking is straightforward. The single sum in solution (4.13, 4.14) is replaced by summations over all visible points, cameras and all RGB channels.

4.3 Compensation of illumination

Intensity recorded during the model acquisition depends, besides the object shape and reflectance, on light sources. We treat the intensity as albedo. As the object moves, the set of light sources visible from a point and their photometric angles change. When modeling these effects we assume:

- cast shadows can be ignored,
- the light sources are distant,
- no specular reflectance.

Under these assumptions, intensities of all points with identical normals will be scaled by a common matrix (for grayscale images only scalar is considered). We adopted a simple method for estimation of the matrix, which performed well in experiments. The method clusters the points X into n groups $G_1, \ldots, G_n$ according to their normals and compensates for the illumination of i-th cluster in each optimization step (4.15) by a color correction matrix

$$\mathtt{E}_i^* = \arg\min_{\mathtt{E}_i} \sum_{\mathbf{x}\in G_i} \|\mathtt{E}_i I(f(\mathbf{x})) - T(\mathbf{x})\|_2^2. \tag{4.16}$$

Let us denote

$$F(\mathtt{E}_i) = \sum_{\mathbf{x}\in G_i} \|\mathtt{E}_i I(f(\mathbf{x})) - T(\mathbf{x})\|_2^2 =$$

$$\sum_{\mathbf{x}\in G_i} I^T(f(\mathbf{x}))\mathtt{E}_i^T \mathtt{E}_i I(f(\mathbf{x})) - 2T^T(\mathbf{x})\mathtt{E}_i I(f(\mathbf{x})) + T^T(\mathbf{x})T(\mathbf{x})\,.$$

Minimization yields the following matrix equation

$$\frac{\partial F(\mathtt{E}_i)}{\partial \mathtt{E}_i} = \sum_{\mathbf{x}\in G_i} -2T(\mathbf{x})I^T(f(\mathbf{x})) + 2\mathtt{E}_i^* I(f(\mathbf{x}))I^T(f(\mathbf{x})) = \mathtt{0} \tag{4.17}$$

and its least square solution is

$$\mathtt{E}_i^* = \left[\sum_{\mathbf{x}\in G_i} I(f(\mathbf{x}))I^T(f(\mathbf{x}))\right]^{-1} \sum_{\mathbf{x}\in G_i} T(\mathbf{x})I^T(f(\mathbf{x})). \tag{4.18}$$

4.4 Tracking-modeling algorithm

The minimal configuration able to build the model must include at least one stereo pair. For tracking, a single camera is sufficient.

If no model is available from a previous tracking-modeling session, the processing starts with a stereo-based reconstruction [50] of the visible part of the object. Albedo of each point is determined from the average of intensities at its projections onto images used for 3D reconstruction. The reconstructed points are clustered and replaced by points on fish-scales [76]. Fish-scales are small oriented planar patches obtained by local clustering of the cloud of points. Small clusters of points are replaced by ellipses with half-axes corresponding to the two main eigenvectors of their covariance matrix. The third eigenvector defines the surface normal. Note that the computation of fish-scale representation is much simpler then a complete surface triangulation. Still the fish-scales are experimentally shown to be sufficient representation for 3D tracking. Knowledge of surface orientation at each points allows:

- Efficient visibility calculations for convex objects.
- Compensation of illumination effects.

Once the partial model is known, it can be used for pose estimation. If observed motion in the image indicates that a part of the image moves consistently with points currently in the model, stereo is invoked again and newly reconstructed patches are merged into the model. The complete algorithm is summarized in Figure 4.2.

Note, that the system never knows when the model is completed, because another consistently moving rigid part of the object can appear later. The system only detects that no reconstruction is currently needed.

1. Capture images
2. If needed, invoke **stereo reconstruction** and merge it to the model.
3. **Estimate the pose** of the object by iterating least square solution (4.15).
4. **Update matrices** $\mathtt{E}_1, \ldots, \mathtt{E}_n$ and for all i and each $\mathbf{x} \in G_i$ recompute object intensity $T(\mathbf{x}) \leftarrow \mathtt{E}_i T(\mathbf{x})$. **goto 1**.

Figure 4.2: Tracking-Modeling Algorithm.

4.5 Experiments

The sequences were captured in an office. We used four firewire cameras with resolution of 640×480 pixels connected to Linux operated computers. The acquisition was TCP/IP synchronized and the setup was calibrated. The total cost of the setup (without computers) is less than 500 dollars. Calibration is easy since a free software for automatic (self)calibration exists [88].

Two different sequences were used. In the *human* sequence, a person makes a variety of motions. The individual walks around, shakes and tilts his head. The camera setup consists of two narrow-baseline cameras for stereo reconstruction and two other cameras spanning approximately a half-circle.

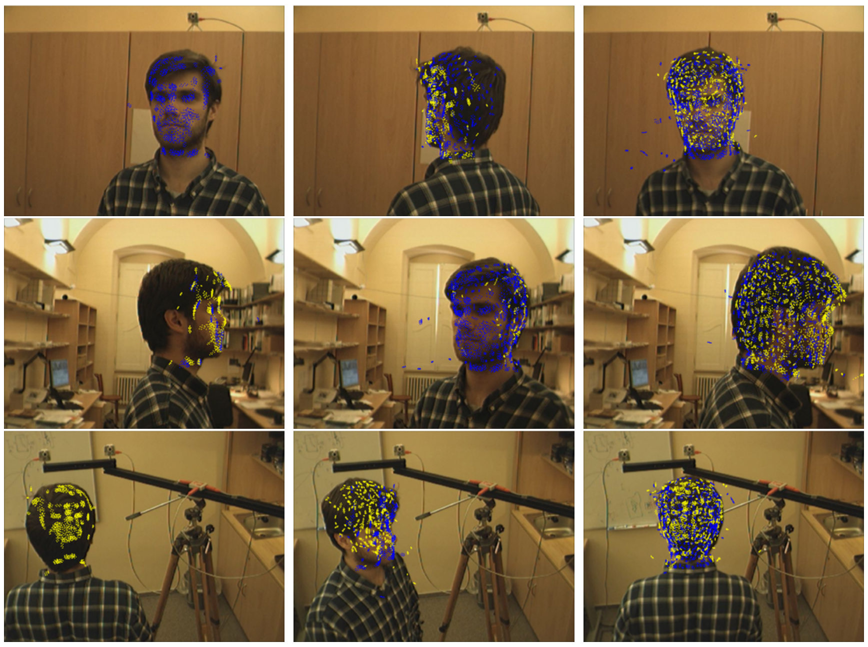
The *book* sequence poses slightly different challenges. The book is a relatively thin object and in some poses the dominant planes (front and back cover) are invisible. The camera setup consists of three cameras located near each other. Two of them are used for stereo, all of them are used for tracking. The model of the book is incrementally constructed from a stereo pair and tracked in all cameras.

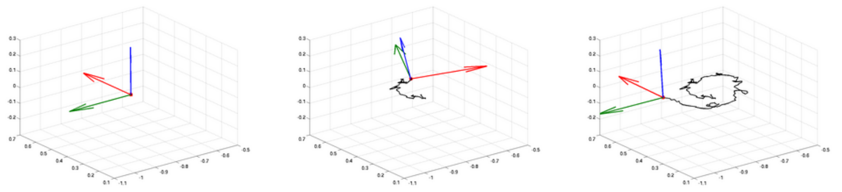
Objects are tracked successfully in both sequences and their shapes are correctly reconstructed. We performed experiments to assess the accuracy and robustness of multiview and monocular tracking. Section 4.5.1 shows that the accuracy of multiview tracking is sufficient for incremental model construction without additional alignment. Section 4.5.2 compares monocular and polynocular tracking. We show that monocular tracking often estimates poses which are incorrect but look correct in the tracking camera. Robustness is tested in section 4.5.3 on the book sequence where the tracking survives even in frames where dominant planes are absent. Experiments showing illumination compensation are described in section 4.5.4. Tracking speed is considered in section 4.5.5. Experiments in sections 4.5.4,4.5.5 are conducted with illumination compensation.

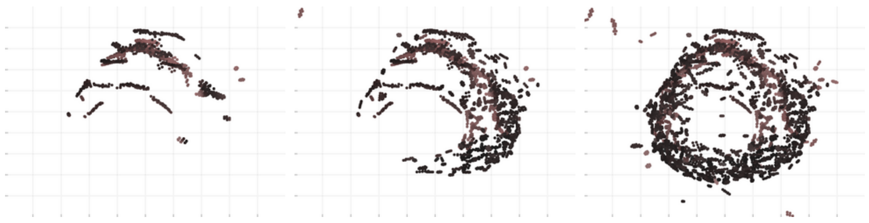
In Figures 3-5, projections of visible points are depicted in blue and invisible in yellow. Readers are encouraged to zoom-in the Figures in the electronic version of the document and watch the accompanying video sequences.

4.5.1 Interleaved tracking and model construction

The first experiment demonstrates the interleaved tracking operation with the model construction. The process starts with a partial reconstruction in the first frame, see the left-most column of Figure 4.3. The tracker is initialized using this partial model. As the human is turning around Fig.4.3(b), the model, is augmented by adding further partial reconstructions Figs. 4.3(c,d). Once the 360 turn is finished, the model is complete and further reconstruction are not required.


a) Multiview tracking; blue points are visible, yellow invisible (occluded) projections.


b) Corresponding poses and path recorded.


c) Incremental construction of the model as seen from top.

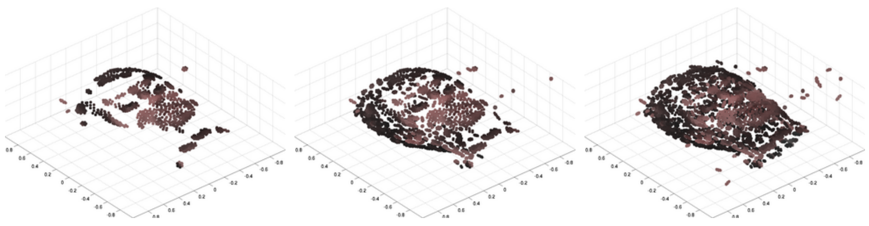

d) Incremental construction of the model, a random view.

Figure 4.3: **Incremental model construction from partial 3D reconstructions and registered by 3D tracking**. Rows 1-3: Different views with projected model. Row 4: Position and orientation in 3D space. Rows 5-6: incrementally constructed 3D model. Columns correspond to frames 1, 100 and 310.

The 3D model is only a side product of the tracking. Its visual appearance cannot match models created with specialized stereo algorithms or visual-hull based algorithms.

4.5.2 Monocular model-based 3D tracking

In the case of monocular tracking, a 3D model and its initial position are considered to be known in advance (e.g. we use the model from previous experiment). The head was successfully tracked over 630 frames, despite the fact that both 3D translation and out-of-plane rotation were present in the sequence. Tracking results are shown in The projected model poses seem correct in images from the tracking camera. However, since only a single camera was used, the recovered 3D position is inaccurate, see row 2 in Figure 4.4. Naturally, the more cameras are used for the optimization, the more accurate 3D pose becomes. Results from the same sequence with the object tracked by all cameras are depicted in the last row of Figure 4.4.

4.5.3 Robustness against critical poses

A thin object like a book used in the experiment, may easily appear in poses which are inherently challenging for the tracking algorithm. If only the book back is visible, the tracking may get unstable. Even during multiview tracking it may happen that most of the object is visible only in a single camera. We call such poses critical.

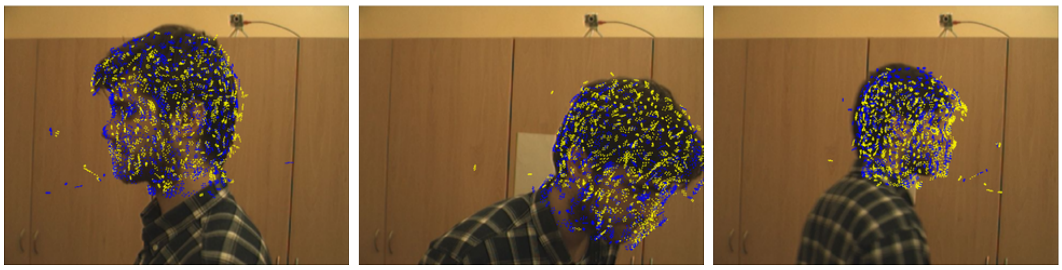
In a critical pose, the book has to be tracked virtually from the single view. The position of the model does not correspond to the projection in the cameras, in which only a small fraction of the book is observable. After the object leaves the critical pose, the model converges to the true position, see Figure 4.5.

4.5.4 Compensation of illuminance effects

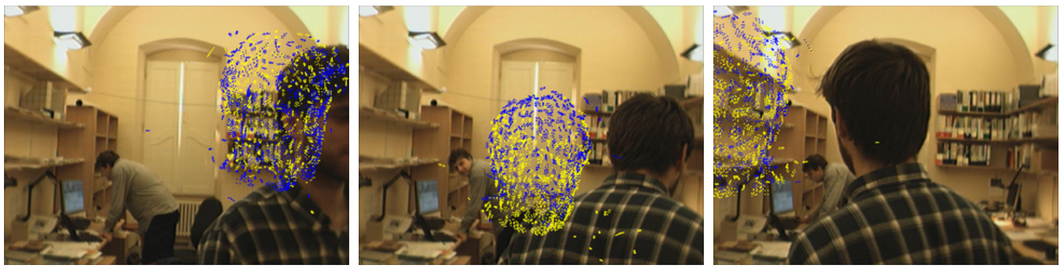
The model points are clustered in 14 equally distributed clusters according to their normals. Each cluster is associated with illuminance constant E_i which changes during the tracking to best fit the observed data.

Figure 4.7-left shows a view with a projected model. Gray levels of particular fish-scales correspond to the values of illuminance constants. Higher values corresponds to the recently illuminated points and vice versa. One can see that in this case light sources were located on the left side of the object which corresponds to the reality.

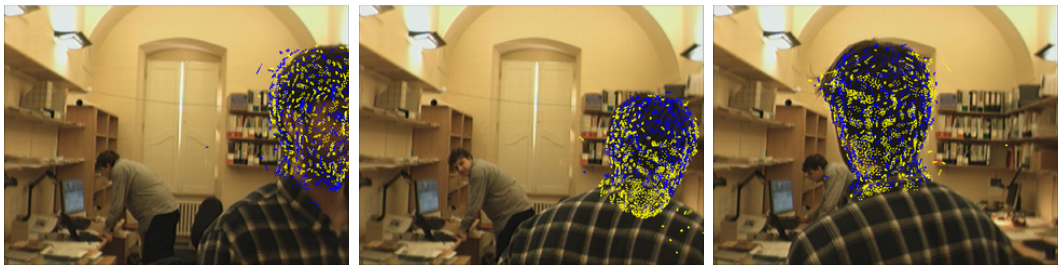
The office has several light sources placed on opposite walls and oriented to the irregularly arched ceiling. Corresponding changes of the illuminance constant E_6 during 360 turn are shown at Figure 4.7-right. Two significant changes during the turn corresponding the light sources are clearly visible. The function of illumination changes is not smooth because during the turn, fish-scales visibility in particular cameras changes and in different times different sets of fish-scales are used for the compensation of illumination effects. Another reason is local inaccuracy of tracking caused by image discretization. Tracking trajectories as well as illumination changes could be smoothed using a motion model, but in our experiments only the output of the optimization is used.



Tracking camera, in monocular tracking, this is the only one used for optimization. Results of monocular tracking projected.



Monocular tracking results as projected to a camera which is approximately orthogonal to the tracking one.



Polynocular tracking. The same camera as above. Note the essentially more consistent 3D pose.

Figure 4.4: **Comparison of monocular (rows 1-2) and polynocular (row 3) tracking**. **Monocular:** Row 1: view from the tracking camera, Row 2: observing camera (shows that, accuracy in orthogonal direction is low). **Polynocular:** Row 3: The same camera with the projected model from multiview tracking.

4.5.5 Speed evaluation

The speed of the algorithm shown in Figure 4.2 was tested on the sequence introduced in the first two experiments (i.e. 4 cameras, RGB images). Slightly-optimized implementation in Matlab runs cca 1.8 s/frame on an AMD-64b Linux running machine. We show experimentally that the tracking of the same sequences in grayscale is successful as well as in RGB. Since one of the most important property of the tracking is the framerate, we increase it three times by considering only a grayscale

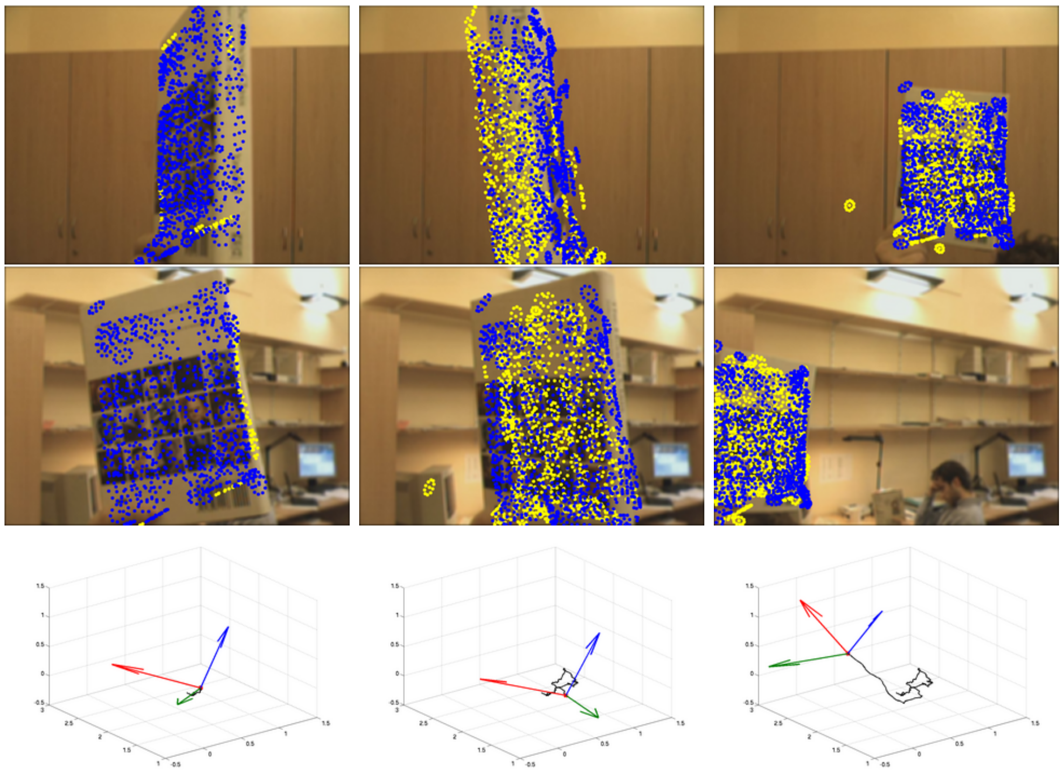


Figure 4.5: **Book tracking**: Rows 1-2: different cameras with projected model, row 3: shows the position and orientation in a 3D space; columns correspond to frames 55, 205 and 265. The second column shows the book in a critical position in which the dominant plane is visible only in one camera.

model/sequence.

Tracking of a grayscale sequence takes approximately 800 ms/frame. Typically, multiple cameras are connected to different computers. Hence, all the contributions to the $\mathtt{A}, \mathbf{b}$ from equation (4.13, 4.14) can be computed independently on the particular computers. Using such a system, a frame rate of 5 frames per second can be achieved with the current Matlab implementation.

4.6 Conclusions

We proposed a fully automatic approach of multiview/monocular 3D object tracking interleaved with incremental model construction. Neither the model nor the initialization are needed to be known in advance. We formulated tracking as a gradient-based method minimizing dissimilarity of the observed image and projected 3D point intensities. We showed that the fish-scale 3D model [76] is accurate enough to support the stable 3D tracking.

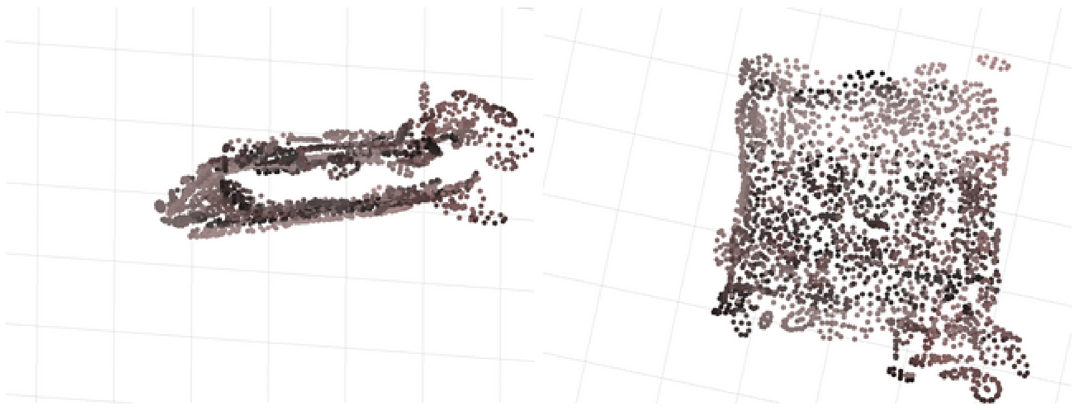


Figure 4.6: **Book Model**: Different views of the book model. Small non-planarity in one corner is the reconstructed hand holding the book.

We experimentally demonstrated that the proposed interleaved approach, successfully tracks a complete 360 turn and a wide range of motion without a need for pre-prepared 3D model. A 3D model is delivered as a side product. We demonstrated the robustness of our method on a sequence with a thin object where the dominant plane was often tracked only from one view.

We showed that the monocular tracking is possible if the model is available. The model projection to the tracking camera often looks correct, projections to other cameras reveals 3D inaccuracies. Still, the monocular tracking can provide results acceptable for some applications. Using more cameras significantly increases stability and accuracy in critical poses and moves. Exact 3D pose may be necessary in many application ranging from virtual reality, human computer interfaces to visual surveillance.

Appendix A

A 3D point $\mathbf{x}$ is projected to 2D image (pixel) coordinates $\mathbf{p}$ as

$$\begin{bmatrix} \lambda\mathbf{p} \\ \lambda \end{bmatrix} = \mathtt{P} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix},$$

where $\mathtt{P}$ is 3×4 camera matrix [40] and $\lambda \in \mathcal{R}$. Let the camera matrix be parametrized as

$$\mathtt{P} = \begin{bmatrix} \mathbf{m}_1^T & t_1 \\ \mathbf{m}_2^T & t_2 \\ \mathbf{m}_3^T & t_3 \end{bmatrix} \tag{4.19}$$

the function $f\colon \mathcal{R}^3 \to \mathcal{R}^2$ projecting 3D point to the camera coordinates is

$$f(\mathbf{x}) = \begin{bmatrix} \frac{\mathbf{m}_1^T\mathbf{x}+t_1}{\mathbf{m}_3^T\mathbf{x}+t_3} \\ \frac{\mathbf{m}_2^T\mathbf{x}+t_2}{\mathbf{m}_3^T\mathbf{x}+t_3} \end{bmatrix}. \tag{4.20}$$

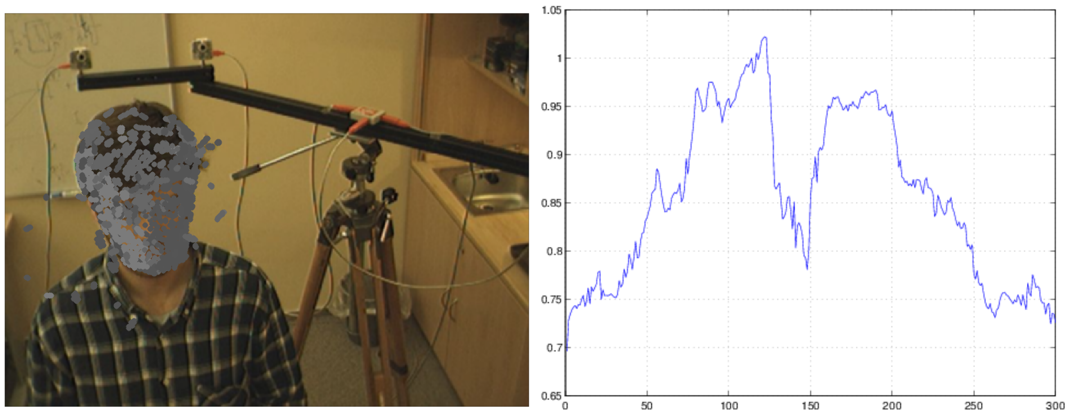


Figure 4.7: **Left**: The image with projected model. Colors correspond to the computed illuminance $\mathtt{E}_i$ of each particular cluster. **Right**: Values of $\mathtt{E}_6$ during the the 360 turn.

Differentiating f with respect to $\mathbf{x}$ we obtain $f' \colon \mathcal{R}^3 \rightarrow \mathcal{R}^{2\times 3}$ Jacobian matrix function, which consists of elements

$$f'_{pq} = \frac{m_{pq}(\mathbf{m}_3^T\mathbf{x} + t_3) - m_{3q}(\mathbf{m}_1^T\mathbf{x} + t_p)}{(\mathbf{m}_3^T\mathbf{x} + t_3)^2} \tag{4.21}$$

where $m_{pq}, p = 1\ldots 2, q = 1\ldots 3$ is q-th elements of $\mathbf{m}_p^T$.

Part II

Learnable regressors for motion prediction

5 Tracking by the optimal sequence of linear predictors

This chapter represents one of the main contribution of the thesis. We propose to learn a sequence of linear regressors on artificially distorted variants of a sample image. The learning minimizes the worst case error on training data. The work has been published in a major scientific journal [101].

We propose a learning approach to tracking explicitly minimizing the computational complexity of the tracking process subject to user-defined probability of failure (loss-of-lock) and precision. The tracker is formed by a Number of Sequences of Learned Linear Predictors (NoSLLiP).

Robustness of NoSLLiP is achieved by modeling the object as a collection of local motion predictors — the object motion is estimated by the outlier-tolerant RANSAC algorithm from local predictions.

Efficiency of the NoSLLiP 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 time-consuming operations are performed during the learning stage - tracking is reduced to only a few hundreds integer multiplications in each step. On PC with 1xK8 3200+, a predictor evaluation requires about 30 microseconds.

The proposed approach is verified on publicly-available sequences with approximately 12000 frames with ground-truth. Experiments demonstrates, superiority in frame rates and robustness with respect to the SIFT detector, Lucas-Kanade tracker and other trackers.

5.1 Introduction

Visual tracking is the process of repeated estimation of the pose of an object (e.g., position) in an image given its pose(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 a search for a pose which optimizes a similarity criterion function. For example, Lucas and Kanade [56,3] use the steepest descent optimization to minimize the sum of square differences between the template and

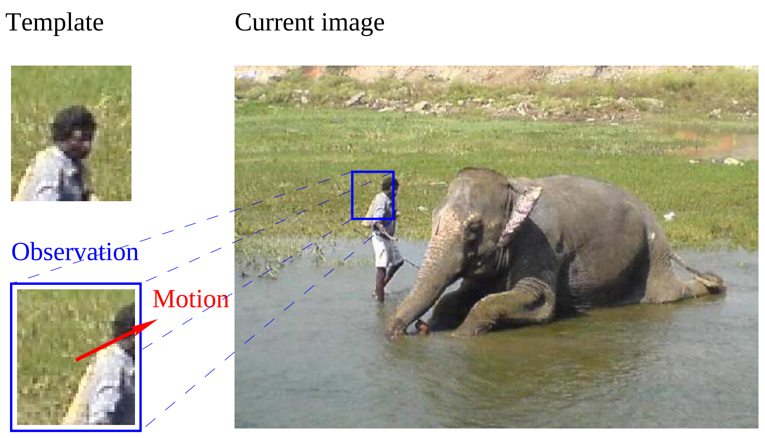


Figure 5.1: Tracking: We define visual tracking as the process of repeated estimation of pose of an object given an image and pose(s) in previous frame(s).

image data, see Figure 5.1. Other approaches [68,2,35] scan the image by a learned classifier, which evaluates the similarity criterion. Regression-based methods [48,14, 94], which do not require any criterion function, estimate the object pose directly from the observed intensities by a learned regression mapping. Methods proceed by collecting training examples—pairs of observed intensities and corresponding poses—and use machine learning techniques to learn the regression function. In tracking, the regression method is initialized by the previous pose or, if available, by the pose derived from a dynamic model. Learned regression function estimates actual object pose directly from the intensities observed around the initial location.

The more complex the regression function, the more precise pose estimation is achievable. Increasing the complexity, however, often suffers from diminishing returns and very complex functions are prone to overfitting. We follow a simple assumption that it is easier to estimate the actual state if the method is initialized in the close neighbourhood of searched pose. 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 function. The coarse estimate of the state allows the consecutive regression functions to operate within a smaller range of poses and to achieve a higher precision with reasonable complexity. Hence, instead of the learning sophisticated predictor, we use a sequence of simple regression functions concatenated so that each of the functions compensate only errors of its predecessor and thus refines the previous estimations. While a single regression function operates on a fixed set of intensities (features), the sequence of functions allows for higher precision because the set of the intensities is updated successively as the actual pose accuracy increases. We learn the optimal sequence of regression functions.

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 single regression function is a special case of a sequence. Since the globally optimal solution is found, the sequence is superior to a single regression function. Any arbitrary regression function allows concatenating, but we observed that the sequence of linear functions achieves high precision with a low computational cost. Focusing on sequences of linear functions we achieved an algorithm which estimates the object pose using only a fraction of processing power of an ordinary computer.

5.2 The State-of-the-art

The most common approach to tracking is repeated optimization of some criterion function $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$ over the space of object poses $\mathbf{t} \in S$, given image $\mathbf{I}$ and previous pose $\mathbf{t}_0$

$$\mathbf{t}^* = \operatorname*{argmin}_{\mathbf{t} \in S} f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0), \tag{5.1}$$

where $\mathbf{t}^*$ is the estimate of the current pose of the object. Criterion $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$ includes some implicit or explicit model of possible object appearances and optionally some relation to $\mathbf{t}_0$. Criterion f could be e.g. obtained as a similarity function or a classifier or foreground/background probability ratio learned from training examples. We call these methods *optimization-based tracking*.

Optimization-based tracking is an online optimization process solving problem (5.1). While some approaches [68,2,35,4] exhaustively scan a subset of object poses S with a classifier approximating $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$, another approaches [56,3,92,80] use a gradient optimization of a criterion approximating $f(\mathbf{t})$.

Unlike optimization-based tracking, *regression-based tracking* methods attempt to model explicitly a relationship between observations and state $\mathbf{t}^*$ without any necessity of defining $f(\mathbf{t}; \mathbf{I}, \mathbf{t}_0)$. They learn a mapping $\varphi(\mathbf{I}, \mathbf{t}_0)$ in a supervised way from synthesized training data [48,14,94].

Tracking methods based on exhaustive scanning can operate within a small range of poses or over the whole image. On the other hand, tracking methods based on the gradient optimization or regression estimate the object pose only locally within a certain range of poses. We understand these local methods as complementary to the scanning based methods, since every pose in a scanned grid can be optionally preprocessed by such local method.

Tracking based on the gradient optimization does not require any learning procedure, however, it suffers from problems of local optimization: convergence to a local minimum, unknown number of required iterations and unknown basin of convergence. In the state-of-the-art, we further focus on regression-based tracking.

Regression-based tracking approaches [14,48,94] estimate 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 5.2,

$$\varphi^* = \operatorname*{argmin}_{\varphi} \sum_{\mathbf{t}} \|\varphi\Big(\mathbf{I}(\mathbf{t} \circ X)\Big) - \mathbf{t}\|. \tag{5.2}$$

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

Noticing that Lucas-Kanade tracker [56] solves a similar optimization task in each frame, one can replace the pseudo-inverse operation by matrix $\mathtt{H}$ learned on a set of synthesized examples. Mapping φ then transforms to the linear function between intensities $\mathbf{I}(X \circ \mathbf{t})$ and motion $\mathbf{t}$,

$$\mathbf{t} = \varphi\Big(\mathbf{I}(X)\Big) = \mathtt{H}\Big(\mathbf{I}(X) - \mathbf{J}(X)\Big), \tag{5.3}$$

where $\mathtt{H}$ is the matrix of some learned coefficients. In the tracking procedure, 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 align the object with current frame. Then the *training set* is an ordered pair $(\mathtt{I}, \mathtt{T})$, such that $\mathtt{I} = [\mathbf{I}^1 - \mathbf{J}, \mathbf{I}^2 - \mathbf{J}, \dots \mathbf{I}^d - \mathbf{J}]$ and $\mathtt{T} = [\mathbf{t}^1, \mathbf{t}^2, \dots \mathbf{t}^d]$. Given the training set, 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 set equal to zero:

$$\begin{aligned}
2\mathtt{H}^*\mathtt{II}^\top - 2\mathtt{TI}^\top &= 0 \\
\mathtt{H}^*\mathtt{II}^\top &= \mathtt{TI}^\top \\
\mathtt{H}^* &= \mathtt{T}\underbrace{\mathtt{I}^\top(\mathtt{II}^\top)^{-1}}_{\mathtt{I}^+} = \mathtt{TI}^+.
\end{aligned} \tag{5.4}$$

Since the method is very fast and simple, it has various applications in tracking approaches. In particular, Cootes et al. [13,14,15] 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. They use a linear predictor (5.3) learned by the LS method (5.4) to estimate all parameters of the AAM. Since the linearity holds only for a small range of parameters, the solution is iterated. Iterations are computed with the same matrix but the length of the optimization step is locally optimized.

This approach was later adapted by Jurie et al. [48] for tracking of rigid objects. Unlike Cootes et al. [14], Jurie's linear predictors estimate the 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. [94] extended

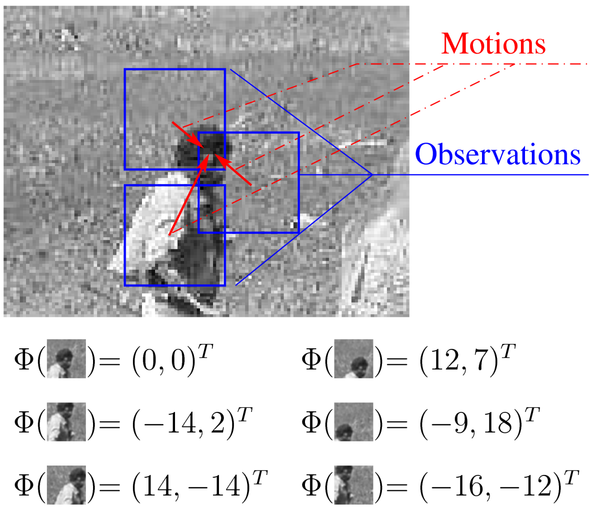


Figure 5.2: Learning linear mapping between intensities and motion in advance. The mapping is learned by a LS method from a set of synthetically perturbed examples.

the approach to the non-linear translation predictors learned by Relevance Vector Machine [90] (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. [24] search for the regression function that has at most ϵ deviation from the actually obtained poses $\mathbf{t}^i$. Their method, called Support Vector Regression Machine, is similar to the Support Vector Machine [93] and allows also the extension for nonlinear kernels. Detailed description may be found in [82].

Zhou et al. [97] proposed greedy learning for additive regression function:

$$\varphi(\mathbf{I}(X)) = \sum_{i=1}^{c} \varphi_i(\mathbf{I}_i(X)), \tag{5.5}$$

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.

Zhou et al. [97] 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. [6] extended the learning technique for the L-nary regression trees and showed that it outperforms [97].

5.3 Contribution

We contribute to the regression-based methods. 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 the 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.

We further extend the method for tracking of the objects modeled by a set of sequential predictors. While each predictor estimates local motion independently, object motion is determined by the RANSAC from these local motions. We optimize the ratio between the number of RANSAC iterations and the number of used predictors subject to a user predefined frame-rate. Since we do not make any assumptions about the object pose, visibility and suitability of the predictors, the set of used predictors must be optimized online. Therefore we learn a set of predictors equally distributed on the object and select an *active* subset which optimize trade-off between coverage and quality in each frame separately.

5.4 Method overview

Because of robustness, the object is locally represented as a set of compact regions. Position of each compact region is determined by its *reference point*, e.g., the geometrical mean of pixels in the region. Since we do not make any apriori assumptions, which positions are the most suitable for the motion estimation, we learn the SLLiPs for evenly distributed reference points on the object. During the learning stage, which is outlined in Algorithm 2, the globally optimal SLLiPs are estimated for all reference points.

Sections 5.5-5.7 describe learning of the individual optimal SLLiP. Section 5.5 introduces definitions. Section 5.6 formulates the learning task as an optimization problem. In this section, we also show that an optimal SLLiP can be created exclusively from LLiPs learned by a minimax method. Hence, the learning is compound: firstly a set of LLiPs is learned by minimax optimization (Section 5.6.1) and then a sequence of LLiPs creating an optimal SLLiP is selected (Section 5.6.2). An efficient heuristic for support set selection which minimizes error on training data is described in Section 5.7.

```
1. Select a set of reference points.
2. For each reference point on the object:
   a) For some discretized values of parameters (ranges and complexities):
      • Generate examples of (observation, motion) pairs.
      • Learn a set of LLiPs by the minimax method (Section 5.6.1).
   b) Select LLiPs creating an optimal SLLiP, (Section 5.6.2).
3. Compute the optimal balance between a number of SLLiPs and RANSAC
   iterations (Section 5.8.2).
```

Algorithm 1 NoSLLiP learning

NoSLLiP tracker, summarized in Algorithm 1, first selects a set of SLLiPs considering the trade-off between the quality and coverage of a visible part of the object (Section 5.8.1). These SLLiPs are used for local motion estimation in the particular frame. The global motion is determined by RANSAC, given the set of local motions. The trade-off between time spent with the local and global motion estimation is also considered and optimized in Section 5.8.2. The proposed method is experimentally verified on synthetic and real data with ground truth in Section 5.9.

```
1. Select a set of active SLLiPs (Section 5.8.1).
2. Estimate local motions by the selected SLLiPs.
3. Estimate object motion from the local motions by RANSAC (Section 5.8.2).
4. Capture the next frame and goto 1.
```

Algorithm 2 NoSLLiP tracking

5.5 Predictors, properties and terminology

In this chapter, we define the predictor and the sequential predictor and show their fundamental properties, which are further used for learning. Let us suppose that the object state is given by object pose parameters (e.g., position)[1]. In each frame,

[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 5.8. For now, let us suppose that only one predictor is associated with the object.

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* $X = \{\mathbf{x}_1, \dots, \mathbf{x}_c\}$. 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 5.7; 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 $(\mathbf{t} \circ X) = (X + \mathbf{t}) = \{(\mathbf{x}_1 + \mathbf{t}), \dots, (\mathbf{x}_c + \mathbf{t})\}$. 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 completely 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*

$$\Lambda(\hat{\varphi}) = \Big\{ \Delta\mathbf{t} \mid \Delta\mathbf{t} = \mathbf{t} \circ \hat{\varphi}\Big(\mathbf{I}(\mathbf{t} \circ X)\Big),\ \forall \mathbf{t} \in R(\hat{\varphi}) \Big\}. \tag{5.6}$$

The uncertainty region is the smallest region within which all the prediction errors from the range $R(\hat{\varphi})$ lie, see Figure 5.3.

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

$$\forall\, \lambda_1, \lambda_2 \in \mathbb{R} :\ \lambda_1 < \lambda_2 \ \Rightarrow\ \Lambda_{\lambda_1} \subset \Lambda_{\lambda_2}\,. \tag{5.7}$$

Ranges R_r are selected from the same class of regions and parametrized by the same parameter $r \in \mathbb{R}$. According to equation (5.7), 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 parametrizable by more than one parameter is discussed later.

5.5.1 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.*

[2]Note, that this is not the range in its usual mathematical meaning.

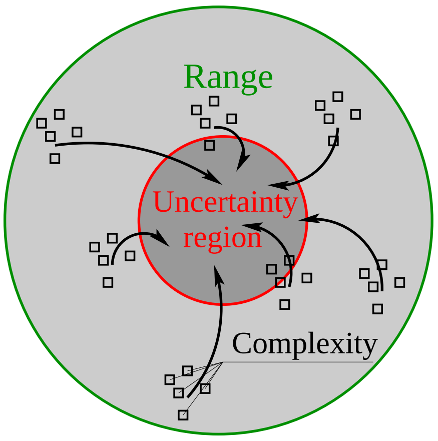


Figure 5.3: Definitions: Range, accuracy and complexity. An example with 2D translations.

Even though we defined the predictor as a 4-tuple, we parametrize 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 5.7. 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

$$\mathcal{F}^1 = \{a_1 \cdot x_1 \,|\, a_1 \in \mathbb{R}\}\,,$$

$$\mathcal{F}^2 = \{a_1 \cdot x_1 + a_2 \cdot x_2 \,|\, a_1, a_2 \in \mathbb{R}\}\,,$$

$$\mathcal{F}^c = \{\sum_{i=1}^{c} a_i \cdot x_i \,|\, a_i \in \mathbb{R}, i = 1 \ldots c\}$$

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

$$\mathcal{F}^1 = \{a \cdot x_1 \,|\, a \in \mathbb{R}\}\,,$$

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

$$\mathcal{F}^c = \{a \cdot \sum_{i=1}^{c} x_i \,|\, a \in \mathbb{R}\}$$

parametrized only by one coefficient $a \in \mathbb{R}$. In general, the class of polynomials of an arbitrary order parametrized 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 5.4 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*.*

$$\varphi^+(c, r) \in \operatorname*{argmin}_{\varphi} \{\lambda \mid \varphi(c, r, \lambda) \in \omega\}\,. \tag{5.8}$$

Note that λ-minimal predictors are the predictors lying on the boundary of ω, see Figure 5.4c.

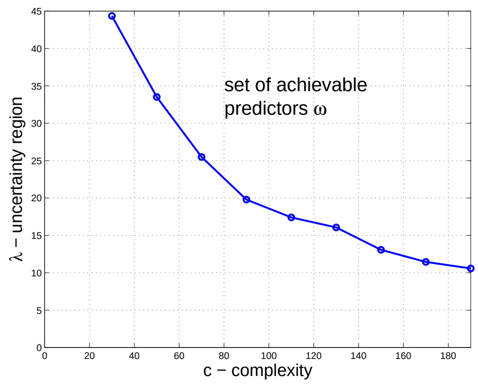
Simple consequence of Definition 5 is that more complex predictors can do everything that the simpler can. This is shown in two following propositions which summarize properties of λ-minimal predictors. The propositions are not crucial for the understanding of the learning procedure however, we later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

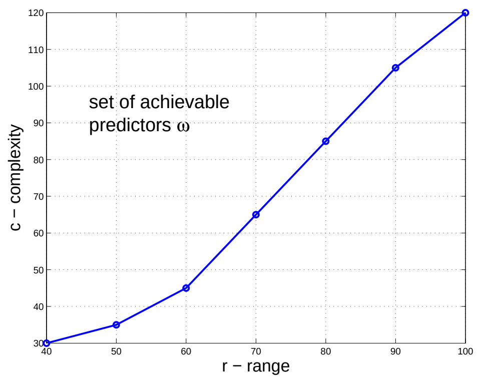
Proof: We prove that the uncertainty region cannot increase with complexity (see for example Figure 5.4a). Let us suppose, we are given two λ-minimal predictors with regressors $\hat{\varphi}_1^+ \in \mathcal{F}^c, \hat{\varphi}_2^+ \in \mathcal{F}^{c+1}$. Since λ-minimal predictors are predictors with minimum uncertainty region, their regressors have to satisfy:

$$\hat{\varphi}_1^+ \in \operatorname*{argmin}_{\hat{\varphi}_1 \in \mathcal{F}^c} \lambda(\hat{\varphi}_1)\,, \tag{5.9}$$

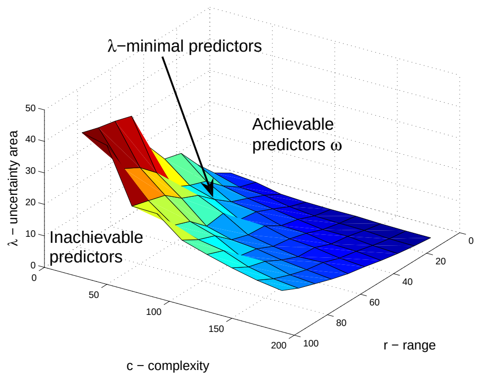
$$\hat{\varphi}_2^+ \in \operatorname*{argmin}_{\hat{\varphi}_2 \in \mathcal{F}^{c+1}} \lambda(\hat{\varphi}_2)\,. \tag{5.10}$$



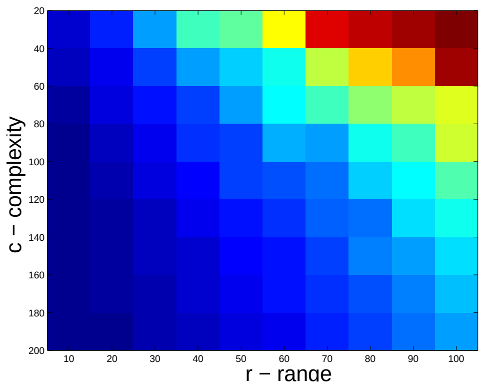
(a) λ-lower bound as a function of complexity.



(b) Complexity as a function of range.



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



(d) Rolled-out λ-lower bound.

Figure 5.4: Set of achievable predictors. Color codes the size of uncertainty region.

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

$$\lambda(\hat{\varphi}_1^+) < \lambda(\hat{\varphi}_2^+). \tag{5.11}$$

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 (5.11)

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

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 (5.10) 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 the 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:

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

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

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

$$\lambda(\varphi_1') = \lambda(\varphi_2^+) < \lambda(\varphi_1^+). \tag{5.12}$$

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. ■

5.5.2 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 5.4. 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* $\Phi = (\varphi_1 \ldots \varphi_m)$, see Figure 5.5, which estimates vector of motion parameter $\mathbf{t}$ in m steps as follows:

$$\begin{aligned}
\mathbf{t}_1 &= \hat{\varphi}_1(\mathbf{I}(X_1)), \\
\mathbf{t}_2 &= \hat{\varphi}_2(\mathbf{I}(\mathbf{t}_1 \circ X_2)), \\
\mathbf{t}_3 &= \hat{\varphi}_3(\mathbf{I}(\mathbf{t}_2 \circ \mathbf{t}_1 \circ X_3)), \\
&\vdots \\
\mathbf{t}_m &= \hat{\varphi}_m\Big(\mathbf{I}\big((\bigcirc_{i=1}^{m-1} \mathbf{t}_i) \circ X_m\big)\Big), \\
\mathbf{t} &= \bigcirc_{i=1}^{m} \mathbf{t}_i.
\end{aligned}$$

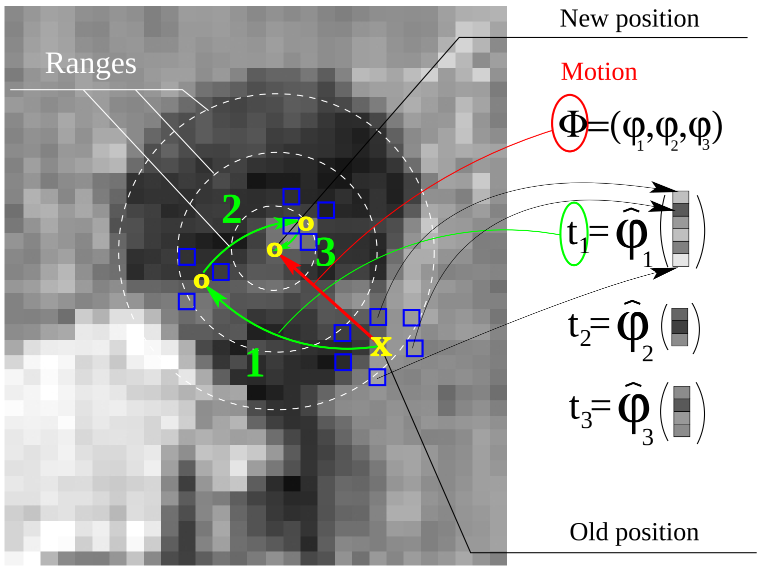


Figure 5.5: *Sequential predictor* $\Phi = (\varphi_1 \dots \varphi_m)$ estimates the vector of motion parameters $\mathbf{t}$ (denoted by the red arrow) in m steps by m different predictors $\varphi_1 \dots \varphi_m$. Particular predictors and the number of steps is the subject of the learning.

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, which is superior to the 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 *of order* m *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.

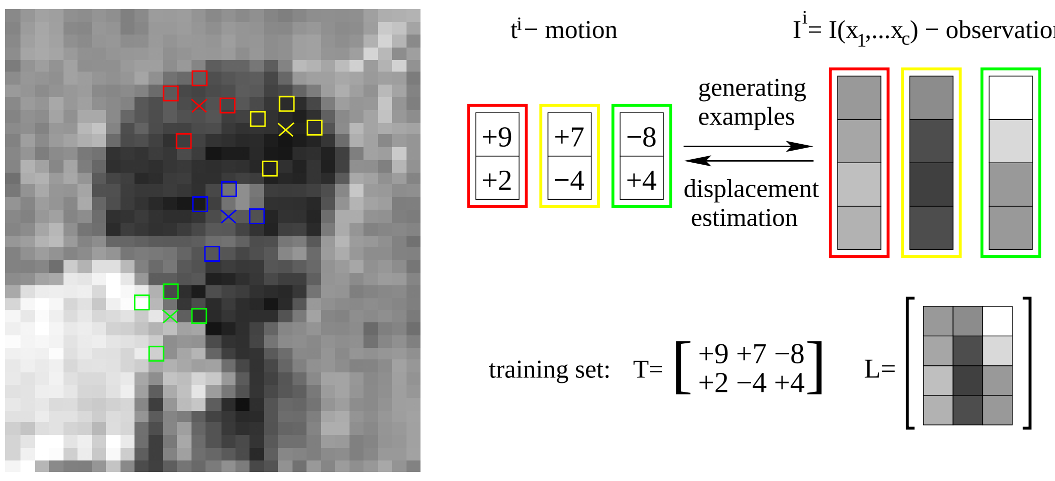


Figure 5.6: Image is perturbed by the motion parameters included in the range r, creating the set of synthesized examples of observed intensities $\mathbf{I}^i$ and motions $\mathbf{t}^i$.

5.6 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 5.6.1 describes learning of the λ-minimal predictor, given a training set. In Section 5.6.2, a set of λ-minimal 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*

$$\Phi^* = \operatorname*{argmin}_{\Phi \in \Omega, m \in \mathbb{N}^+} \left\{ \sum_{i=1}^{m} c_i \mid r_1 \geq r_0, \lambda_m \leq \lambda_0 \right\}, \tag{5.13}$$

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 one optimal sequential predictor at least 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. ■

We consider only predictors with the smallest uncertainty region λ, i.e., predictors lying on the λ-lower bound defined by (5.8). In that way, the λ-lower bound, 2D manifold in (c, r, λ)-space (Figure 5.4c), is rolled out to the (c, r)-space (Figure 5.4d). Task (5.13) reduces to

$$\Phi^* = \operatorname*{argmin}_{\Phi \in \Omega^+, m \in \mathbb{N}^+} \left\{ \sum_{i=1}^{m} c_i \mid r_1 \geq r_0, \lambda_m \leq \lambda_0 \right\}, \tag{5.14}$$

where Ω^+ is the set of sequential predictors created only by the λ-minimal predictors, equation (5.8).

The procedure of linear λ-minimal predictor learning is carried out by linear programming in Section 5.6.1. In Section 5.6.2, a sequence of the λ-minimal predictors creating the optimal sequential predictor Φ^*, equation (5.14), is selected from a set of learned λ-minimal predictors. The problem is formulated as seeking he cheapest path in a graph.

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

Linear predictor

In order to estimate a predictor satisfying equation (5.8), the regressor $\hat{\varphi}$ needs to be specified in detail. We restrict ourselves to Learned *Linear* Predictors (LLiP), i.e., predictors with the linear regressor:

The linear regressor $\hat{\varphi}_L$ is a linear mapping defined as

$$\mathbf{t} = \hat{\varphi}_L(\mathbf{I}) = \mathtt{H}\mathbf{I}, \tag{5.15}$$

where $\mathtt{H}$ is a $2 \times c$ matrix. Similarly to this, sequential linear predictor is the Sequence of LLiPs (SLLiP). Note, that the time required for motion estimation by LLiP is determined by the size of its support set, therefore $c = |X|$.

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 kernel functions by *data lifting*. A 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 in a higher dimension.

Training set construction

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 5.6. 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 $\mathtt{I} = \left[\mathbf{I}^1 \ldots \mathbf{I}^d\right]$ and $\mathtt{T} = \left[\mathbf{t}^1 \ldots \mathbf{t}^d\right]$. The ordered triple $(\mathtt{I}, \mathtt{T}, \mathcal{X})$ of such matrices and ordered d-tuple of support sets $\mathcal{X} = \left\{X^1 \ldots X^d\right\}$ 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 [48] 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 (5.8). As we mentioned before, the uncertainty region is assumed to be from a class parametrizable 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).

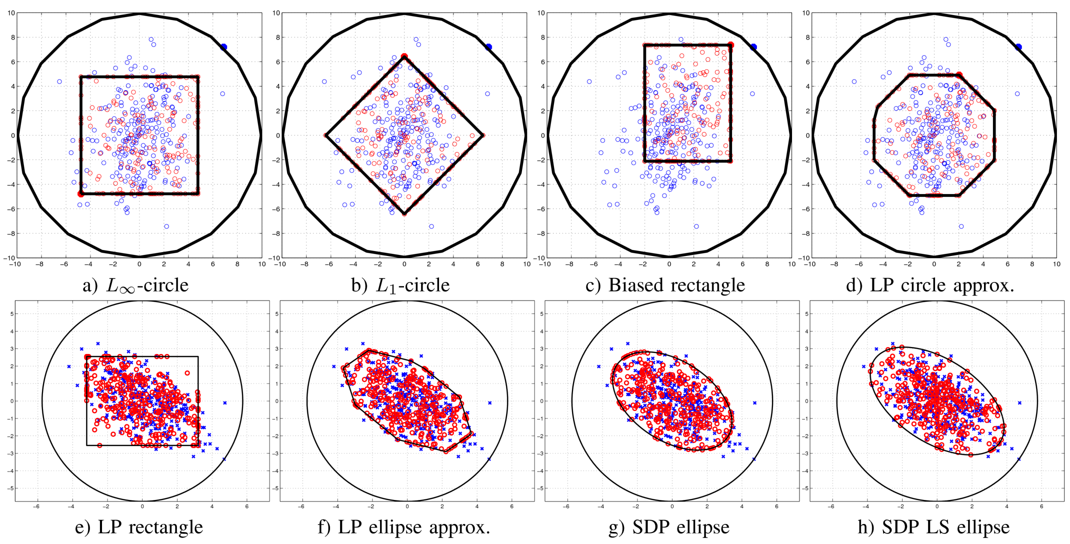


Figure 5.7: **Different classes of uncertainty regions:** Points correspond to prediction errors $\Delta\mathbf{t}$ of 2D translation on a training set. Errors of the predictor learned by LS method are in blue and by the minimax method in red. Uncertainty regions and ranges are black. Only a), b) and e) are described in the paper, see [99] for detailed description of the other classes.

Restricting to the square-shaped uncertainty regions centered in the origin of the coordinate system (see figure 5.7-a) and parametrized by parameter λ, equation (5.8), defining the λ-minimal predictor, simplifies as follows

$$\begin{aligned}
\mathtt{H}^* &= \operatorname*{argmin}_{\mathtt{H}}(\max_i \|\mathtt{H}\mathbf{I}^i - \mathbf{t}^i\|_\infty) = \\
&= \operatorname*{argmin}_{\mathtt{H},\lambda}\{\lambda \mid \forall_i |\mathtt{H}\mathbf{I}^i - \mathbf{t}^i| < \mathbf{1}\lambda\} = \\
&= \operatorname*{argmin}_{\mathtt{H},\lambda}\ \lambda \qquad (5.16)\\
&\quad \text{subject to}: -\lambda \leq\ (\mathtt{H}\mathbf{I}^i)_k - \mathbf{t}^i_k\ \leq \lambda,\\
&\qquad\qquad i = 1 \ldots d,\ k = 1, 2.
\end{aligned}$$

We reformulate problem (5.16) as a linear program

$$\min_{\mathbf{x}}\{\mathbf{c}^\top\mathbf{x} \mid \mathtt{A}\mathbf{x} \leq \mathbf{b}\}, \qquad (5.17)$$

where

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

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:

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

Denoting

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

the linear programming problem (5.17) is obtained. The shape of such uncertainty region is a rectangle, which is in 2D space parametrized by two parameters - length of

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

its sides, see Figure 5.7c. Since we want to work with uncertainty regions parametrizable 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 (5.16) 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 5.7b. The combination of L_∞ and L_1 norms allows to work also with L_2 circle approximation, see Figure 5.7d. Note that we can also work with elliptic regions as shown in Figure 5.7d-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 5.7h. Detailed description of such uncertainty region extensions can be found in [99].

5.6.2 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 (5.8), 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 5.8a 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^+, predefined range r_0 and uncertainty region λ_0, we seek the ordered subset of ω^+ that forms the optimal sequential predictor Φ^*, which minimizes the complexity. Since the predefined 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 5.8a, where the size of uncertainty region decreases with the complexity for each particular range. The 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 as large as the uncertainty region at least 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 the complexity of which can be decreased without uncertainty region increase, cannot be part of the 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:

$$\lambda(\varphi^+(c_2, r)) \geq \lambda(\varphi^+(c_1, r)) \Rightarrow c_2 \leq c_1. \tag{5.18}$$

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:

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

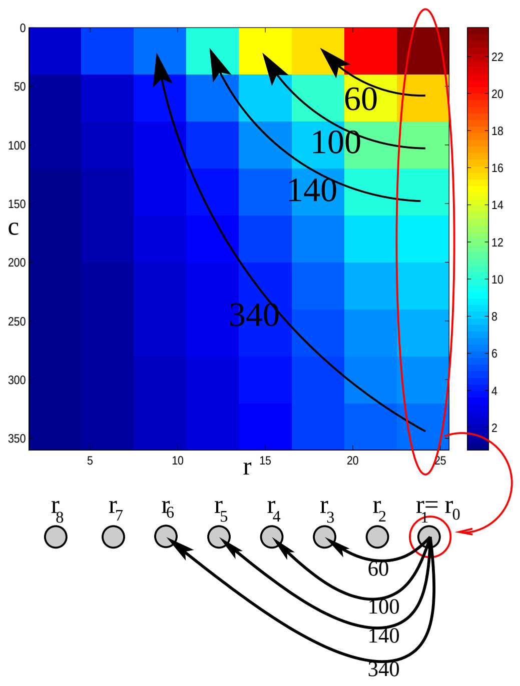
$$\Rightarrow \begin{array}{c} \lambda(\varphi^+(c_1, r_2)) \geq \lambda(\varphi^+(c_2, r_2)) \\ \lambda(\varphi^+(c_1, r_1)) \geq \lambda(\varphi^+(c_2, r_1)) \end{array} \underset{\text{Eq.(5.18)}}{\Rightarrow} c_2 \leq c_1$$

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. ■

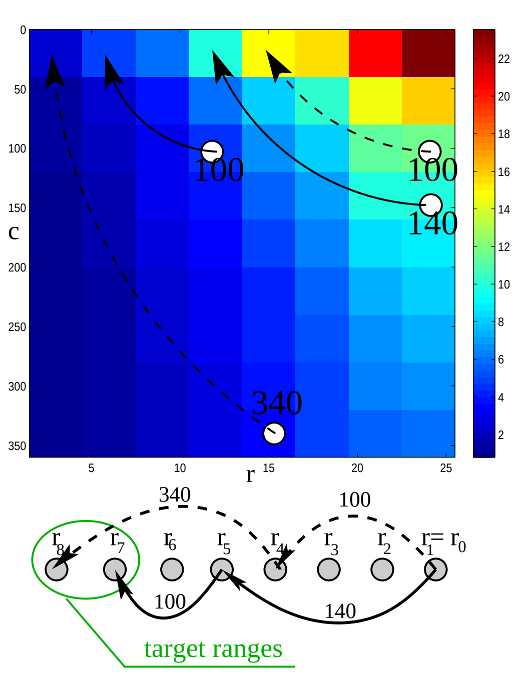
Note, that if only the smallest possible ranges are considered, then the constructed graph has at most $|C| \cdot |R|$ edges. On the contrary, without the Proposition 4 the constructed graph would have $|C| \cdot |R|^2$ edges.

Arrows in Figure 5.8a 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 5.8b. 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 $\mathcal{G} = (V \equiv R, E \subseteq R \times R, \alpha\colon E \rightarrow C)$, 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 5.8. 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. Dijkstra algorithm [8] searches for the cheapest path to the ranges with predictors with sufficiently small uncertainty regions, depicted by red circles in Figure 5.9. 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



(a) Edges from the first vertex (range).



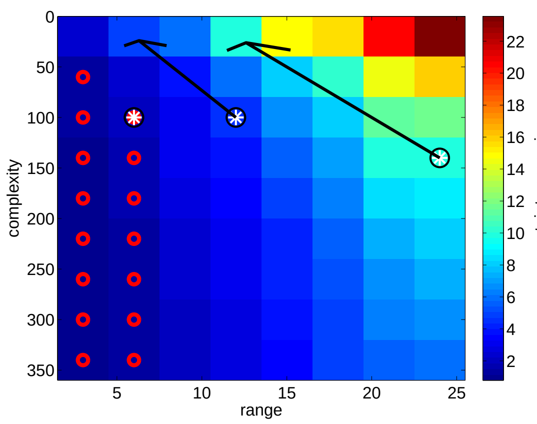
(b) Two different paths to target ranges.

Figure 5.8: (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. The cost of edges corresponds to the complexities. (b) Two paths to the target ranges (solid line denotes the optimal path).

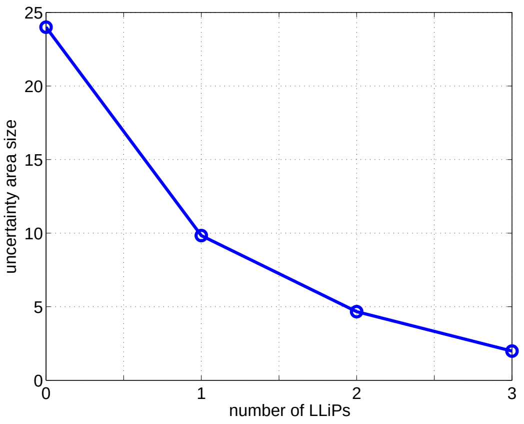
target range exists then there are more possible solutions and the cheapest solution is selected. The solution is the optimal sequential predictor (5.14). The method is summarized in Algorithm 3.

The optimal path is depicted in Figure 5.9a. For instance, the optimal sequence for the example in Figure 5.9a, 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 parametrized uncertainty regions. Extension of the proposed method to the more than one-variable parametrized uncertainty regions is straightforward. The uncertainty region shape is ω^+ $w + 1$-dimensional for w-dimensional parametrization.



(a) Path with the lowest complexity.



(b) Iterations.

Figure 5.9: (a) size of uncertainty regions (coded by colors) as a function of complexity c (vertical axis) and range r (horizontal axis) and the optimal path from the initial r_0 to a predictor with sufficiently small uncertainty region (red circles). (b) size of uncertainty region after each iteration (number of LLiPs = 0 corresponds to the range $r_0 = 25$).

```
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 $\varphi^+(c, r)$:
               $v^* = \operatorname{argmin}_{v \in R} \{v \mid \lambda(c, r) < v\}$
         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, $\varphi_t^+(c_t, r_t) = \operatorname{argmin}_{\varphi^+(c,r) \in \omega_T} \{\rho(r) + c\}$ 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 3 Estimation of the optimal sequence from a set of λ-minimal LLiPs.

5.7 Selection of support set for efficient tracking

Until now, we have 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 5.5.

Let us suppose we are given training set $(\mathtt{I}, \mathtt{T}, \mathcal{X})$ with the 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

$$e(\mathbf{u}) = \left\| \mathtt{T} - \mathtt{T}\left(\mathtt{I}(\mathbf{u},:)\right)^{+} \mathtt{I}(\mathbf{u},:) \right\|_F^2, \tag{5.19}$$

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,

$$\mathbf{u}^* = \operatorname*{argmin}_{\substack{\mathbf{u} \in \{0,1\}^b, \\ \|\mathbf{u}\|_1 = c}} e(\mathbf{u}), \tag{5.20}$$

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 convenient 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:
       $j^* = \operatorname*{argmin}_{j=1\dots b}\{e(\mathbf{u} + \delta_j)\},$
       $\mathbf{u}(j^*) = 1$
   where $\delta_j$ is $b$-vector of zeros with "1" at the position $j$.
```

Algorithm 4 Greedy LS support set selection algorithm.

Recently, an extension to LK tracker has been published by Benhimane et al. [5], where the most convenient (w.r.t. gradient optimization method) subset of pixels is selected during a training stage. According to the published experimental data, such improvement decreases error rate of no more than 20%. While Benhimane et al. optimize only the subset of pixels and preserves the gradient-based tracking, we optimize both the set of pixels and the motion estimation method.

5.8 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

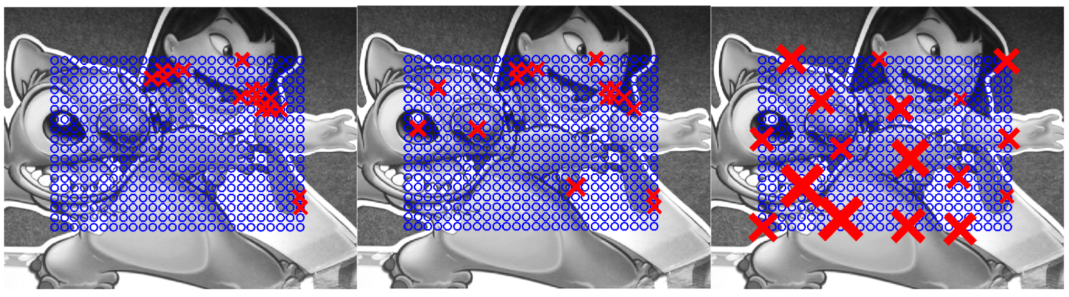


Figure 5.10: Object coverage by predictors for different weightings. Blue circles correspond to the all learned predictors, red crosses to the selected predictors. Size of crosses corresponds proportionally to the complexity.

represent the object by Number of SLLiPs (NoSLLiP tracker). Such a 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 5.8.1. It is also not clear, how many SLLiPs should be used and how many RANSAC's iterations should be computed. Section 5.8.2 describes method estimating the ratio between number of SLLiPs and number of RANSAC's iterations, which maximize a probability of successful tracking.

5.8.1 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 convenient for some applications.

Definition 9 Coverage measure *is*

$$r(Z) = \sum_{\mathbf{z}\in Z} d(\mathbf{z}, Z \setminus \mathbf{z}), \tag{5.21}$$

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

$$d(\mathbf{z}, Z) = \min_{\mathbf{y}\in Z} \|\mathbf{z} - \mathbf{y}\|. \tag{5.22}$$

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 10 Quality measure *is*

$$q(\mathbf{z}) = |c(\mathbf{z}) - \max_{\mathbf{y}\in Z} c(\mathbf{y})|. \tag{5.23}$$

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:

$$f(Z) = w\frac{r(Z)}{r(\widetilde{Z})} + (1-w)\frac{q(Z)}{q(\widetilde{Z})}, \tag{5.24}$$

where $w \in [0;1]$ is the coverage weight. Algorithm 5 selects a set of active SLLiPs, given 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 5 Selection of active set of SLLiPs.

Figure 5.10 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.

5.8.2 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

$$P_R(k,h) = 1 - \left(1 - \left(\frac{k}{n}\right)^v\right)^h, \tag{5.25}$$

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). The number of successfully tracked points k in not known in advance, it is a random quantity with binomial distribution,

$$P_k(k) = P_{\text{bin}}(n,k) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{5.26}$$

where p is the probability of the 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 of this chapter, 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

$$(n^*, h^*) = \{\arg\max_{n,h} \hat{P}_{\text{success}}(n, h) \mid ht_0 + \sum_{i=1}^{n} t_i \le t\} \tag{5.27}$$

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 $\{[n, h] | \; ht_0 + \sum_{i=1}^{n} t_i = t\}$ of the constrained set. Consequently, problem (5.27) can be rewritten as the unconstrained one-dimensional problem as follows

$$n^* = \arg\max_{n} \hat{P}_{\text{success}}(n, \frac{t - \sum_{i=1}^{n} t_i}{t_0}) = \arg\max_{n} \overline{P}_{\text{success}}(n). \tag{5.28}$$

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.

5.9 Experiments

Properties of SLLiP tracking and learning algorithms are experimentally verified. In Section 5.9.1 some preliminary results on challenging sequences are demonstrated. In Section 5.9.2 robustness and accuracy is evaluated on ground truthed sequences. In particular, Section 5.9.2 describes ground truthed data and Section 5.9.2 compares SLLiP to the state-of-the-art approaches. In Section 5.9.3 additional properties such as relation between robustness and speed relation between predefined and achieved accuracy are summarized.

5.9.1 Preliminary qualitative evaluation

In the first experiment NoSLLiP 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[4]. Tracking of various objects with partial occlusions and motion blur is shown in Figure 5.11. Green/blue circles outline inliers/outliers, red arrows show the local motion estimated by SLLiPs. In some images, also the support set is outlined by blue points. Tracking of objects with variable set of active predictors is demonstrated in Figure 5.12 and 5.13. Active set of visible SLLiPs is estimated by Algorithm 4. Yellow numbers denotes IDs of particular SLLiPs. Although we mainly work with planar objects in order to avoid problems arising from the inaccurate 3D reconstruction, SLLiPs are attachable to the arbitrary 3D model, see for example Figure 5.13.

[4]We encourage the reader to look also at video-sequences available at `http://cmp.felk.cvut.cz/demos/Tracking/linTrack`.

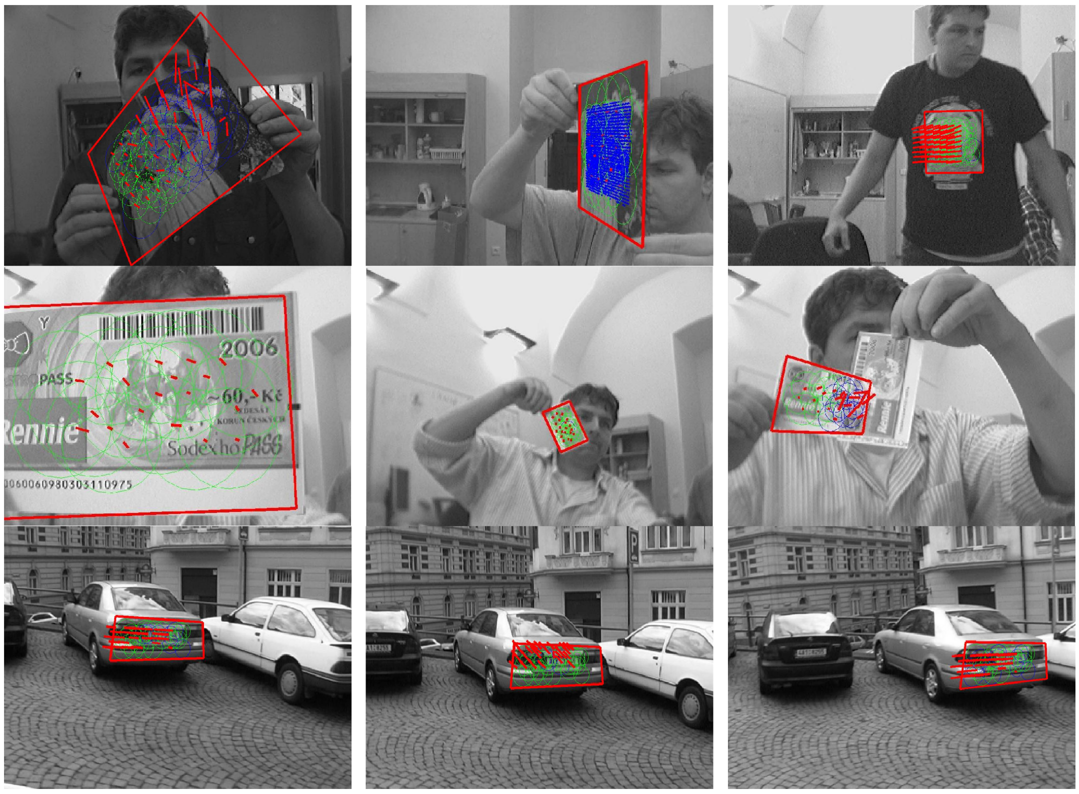


Figure 5.11: **Robustness to partial occlusions and fast motion:** Green/blue circles outline inliers/outliers, red arrows shows local motion estimated by SLLiPs. Support set outlined by blue points.

5.9.2 Quantitative evaluation of the robustness and accuracy

Ground truthed data

The quantitative evaluation of the robustness and accuracy of SLLiPs is conducted on sequences with 3 different objects: MOUSEPAD (MP), TOWEL and PHONE, where ground truth positions of the object corners in total number 11963 frames were manually labeled[5], see Figure 5.14 for some examples. Accuracy is measured by the average error in object corners. The error is expressed in percentage and normalized by the actual size of the object upper edge, in order to make the measure independent to the actual scale. The robustness is measured by the number of loss-of-locks, defined as the cases where the error was higher than 25% in one of the corners at least. In loss-of-lock frames, the tracker was reinitialized from the ground truth and the accuracy did not contribute to the total accuracy statistics.

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

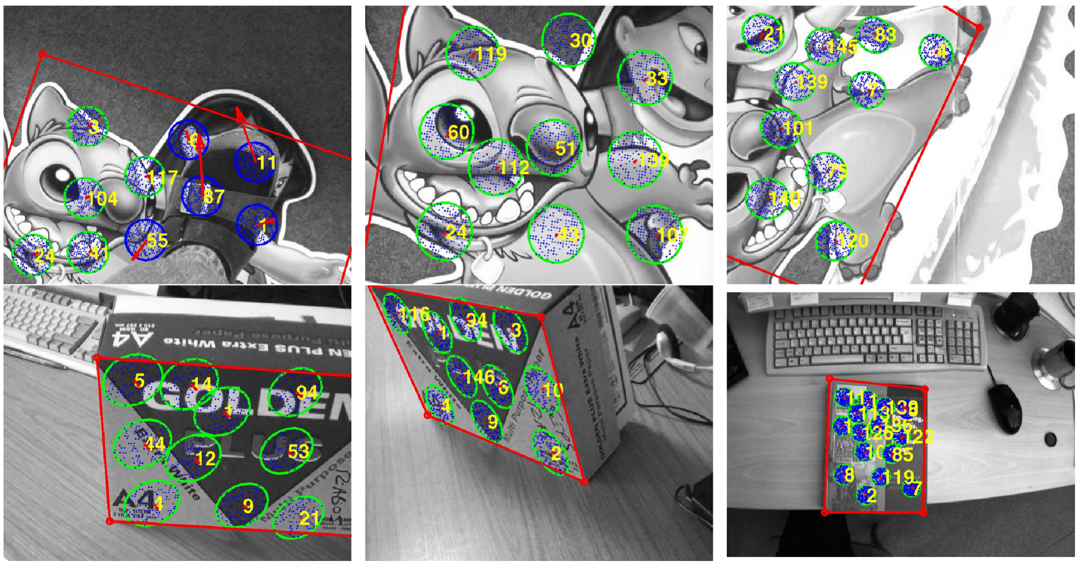


Figure 5.12: **Tracking with variable set of active predictors:** Yellow numbers denote ID of particular SLLiPs. Blue points represent the support set, green circles highlight inliers, red arrows outline a local motion estimated by SLLiPs.

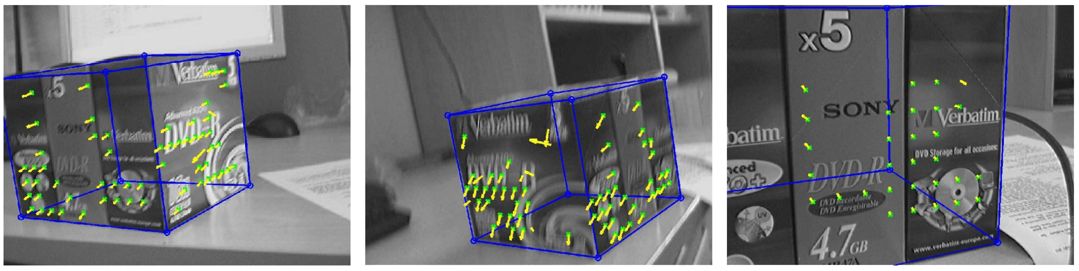


Figure 5.13: **3D tracking:** Variable set of active predictors and motion blur.

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

Table 5.1 compares the NoSLLiP tracker to the state-of-the-art Lowe's SIFT detector [55] (method: SIFT)[6], Lucas-Kanade tracker [56] (method: LK tracker) and Jurie's LLiP tracker learned by the Least Squares method [48] (method: LLiP LS). All these local motion estimators were combined with RANSAC, to keep test conditions as similar as possible. SIFT 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 the camera, but its basin of attraction was insufficient in many frames

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

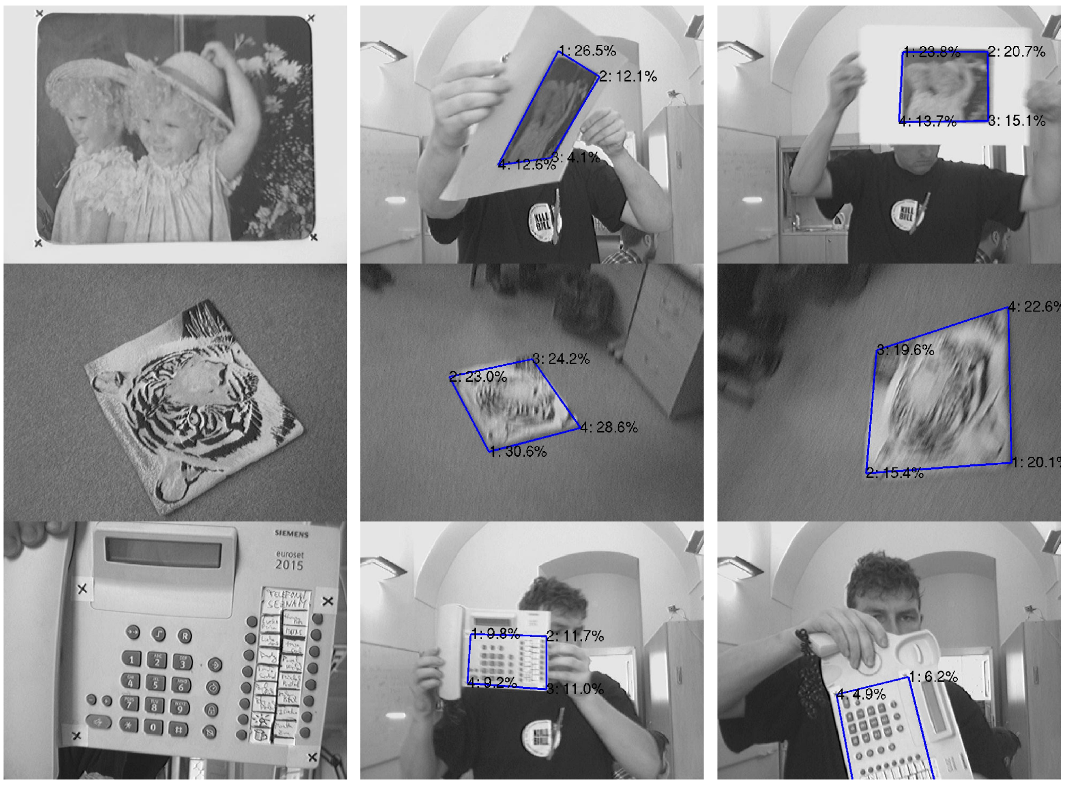


Figure 5.14: **Ground truthed sequences:** Left column shows images used for training. The middle and right columns demonstrate some successfully tracked frames with a strong motion blur from the testing sequences. The blue rectangle delineates the object. Percentage values in corners are current corner speeds related to the current size of the object upper edge.

Method	Object	Frame-rate [fps]	Loss-of-locks [-/-]	Error [%]
NoSLLiP	MP	18.9	13/6935	1.5
SIFT [55]	MP	0.5	281/6935	1.4
LK (IC) tracker [56]	MP	2.6 (25)	398/6935	2.4
LLiP LS [48]	MP	24.4	1083/6935	6.3
LLiP LS [48] 1/2	MP	24.2	93/6935	3.0
NoSLLiP	TOWEL	21.8	5/3229	2.1
NoSLLiP	PHONE	16.8	20/1799	1.8

Table 5.1: **Comparison of robustness and accuracy** of SLLiP, LK, LLiP trackers (MATLAB implementation) and SIFT detector (C++ implementation) on MP sequence. Frame-rate of IC algorithm is estimated based on comparison published in [3].

for the correct motion estimation. The tracking failed for fast motions, too.

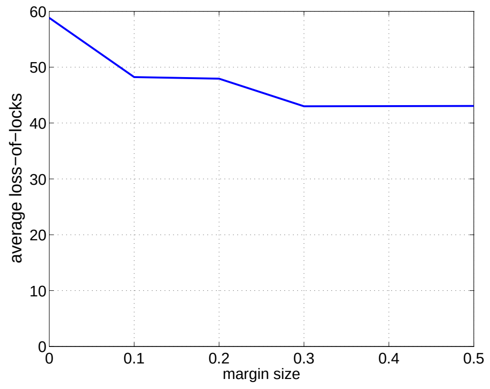
According to the detailed speed comparison published in [3], 6-parameter optimization by Inverse Compositional (IC) algorithm implemented in MATLAB runs approximately ten times faster than the optimization by Forward Additive algorithm used in LK tracker. However, the basin of attraction and sensitivity 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 verified experimentally 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 a sufficient accuracy on the same range, a 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.

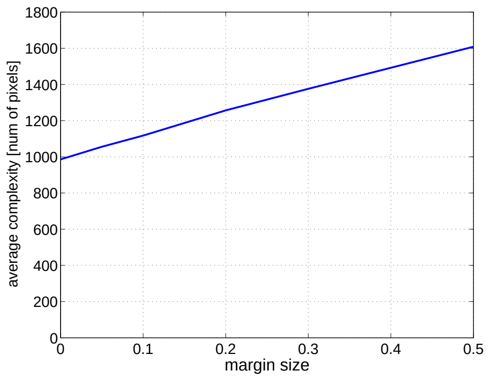
5.9.3 Additional experiments

Robustness analysis

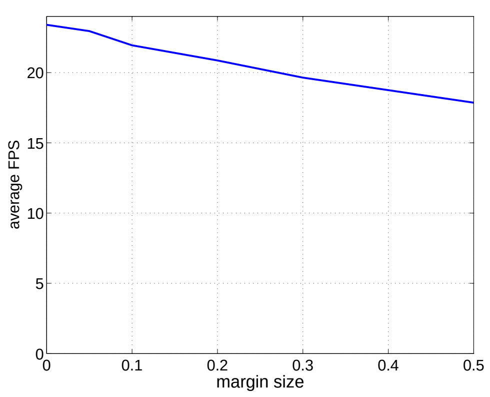
We defined SLLiP as a sequence of LLiPs satisfying that the range of every predictor is as large as the uncertainty region of its predecessor at least, i.e., $\forall\, r_{i+1} \geq \lambda_i$, $i = 1 \ldots m-1$. We showed in Proposition 4, that the complexity minimization in the learning stage results in equality $\forall\, r_{i+1} = \lambda_i$, $i = 1 \ldots m-1$. As a result, whenever



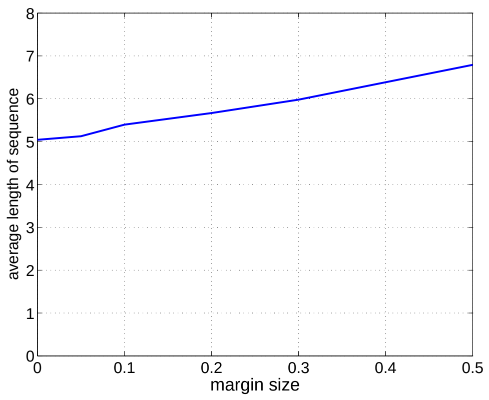
(a) Loss-of-locks



(b) Complexity of SLLiPs.



(c) Frame-rate



(d) Number of LLiPs in SLLiP

Figure 5.15: Robustness analysis: The higher the margin, the higher the robustness to noise but also the higher the complexity of SLLiPs.

the testing data are corrupted by noise, the prediction might not be within the range of the following predictor, which might consequently cause a divergence of the SLLiP. For practical applications, a margin assuring robustness to the noise, is required. We require $\forall\, r_{i+1} \geq \lambda_i(1+\gamma)$, $i = 1 \ldots m-1$, 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.

A quantitative robustness evaluation is performed by computing the average number of loss-of-locks as a function of the margin (Figure 5.15a), the average complexity of SLLiPs as a function of the margin (Figure 5.15b) and average frame-rate as a function of the margin (Figure 5.15c). The test sequence based on the ground truthed sequence was intentionally made more challenging. The experiment is conducted on a selected mousepad sub-sequence (frames 3500–6000), where only the each second frame is processed in order to increase the 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 $[0, 0.05, 0.1, 0.2, 0.3, 0.5]$. The number

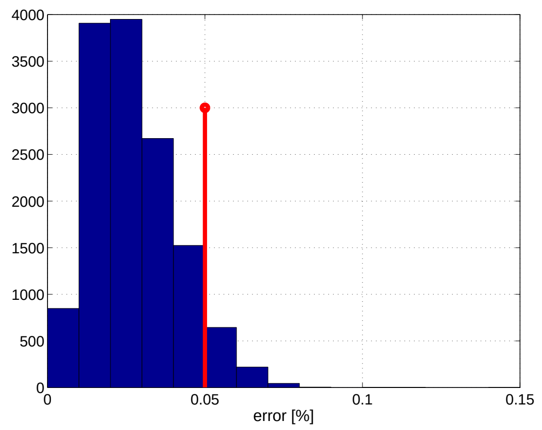


Figure 5.16: Comparison of desired and real accuracy.

of loss-of-locks (Figure 5.15a) and the frame-rate (Figure 5.15c) are evaluated as an average over 20 processing of the sequence with the different starting frame. The complexity (Figure 5.15b) and the length of LLiP sequence (Figure 5.15d) are computed as an average over the set of 48 learned SLLiPs.

Accuracy analysis of minimax learning

In this experiment we compare the uncertainty region λ_0 required in learning and the real distribution of SLLiP errors. We learned 48 SLLiPs covering the mousepad with desired accuracy 5% of the range size. The accuracy is evaluated on those frames, in which the inter-frame motion is smaller than the learning range of SLLiPs. Figure 5.16 shows histogram of displacement errors with λ_0 denoted by the red line at 0.05. In approximately 10% of cases, the errors are higher. This is presumably caused mainly by the limited ground truth accuracy and partly by the image noise. Note, that the optimal SLLiP is guaranteed to converge into λ_0 for all training examples,

Support set selection by greedy LS algorithm evaluation

We compare the mean square error (MSE) achieved by the greedy LS support set selection algorithm (Algorithm 4) and the MSE achievable by a random support set selection. Figure 5.17a 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 4 would require a prohibitively high number of iterations to achieve at least comparable results with the proposed greedy LS algorithm.

Figure 5.17b shows MSE as a function of the 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.

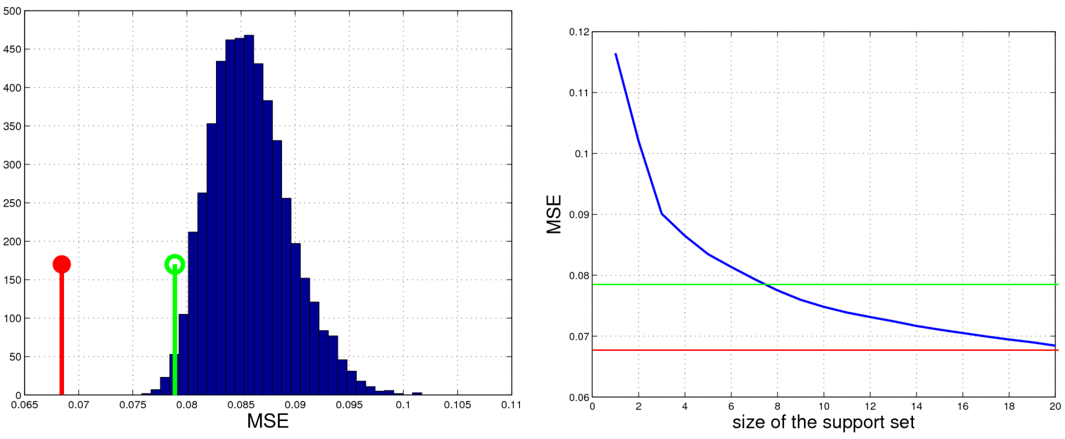


Figure 5.17: Histogram of MSEs of predictors with the randomly constructed support sets. (a) $C = 20$, $E_{99} = 0.0789$ (green empty circle denotes 99%-quantile), $\hat{E} = 0.0684$ (red filled circle denotes MSE of the proposed method), (b) MSE as a function of the complexity during the incremental construction of the support set.

5.10 Discussion

Tracking for detection

Due to the very high performance of the proposed predictor, there is a possibility coupling it with a detector. The detection performed in a pose-grid [2,35,68] could be replaced by the prediction followed by detection performed in a sparser pose-grid. The efficiency of the method depends on the ratio of detectability and predictability radii and times needed for the detection and prediction, respectively.

Tracking in feature space

Instead of the intensities the arbitrary set of features can be used. There is a large set of linear features, i.e. the features computed as a linear combination of the observed intensities. Since the linear prediction from linear features would be only a linear combination of a linear combination, which is again the linear combination, there is almost no reason use the linear features. In the other words, if any linear combination of the intensities is necessary, it is automatically included in the regressor coefficients during the learning stage in the optimal manner. The particular counter example are Haar features [37], which allow for a faster direct computation from the integral image.

Non-linear regression

If the linear mapping is insufficient, the method allows a natural extension to an arbitrary class of mappings formed as a linear combination of kernel functions by *data lifting*. For example, in the polynomial mapping particular monomials are considered as further observed intensities. It allows the learning procedure to deal with higher-dimensional linear mappings instead of the non-linear ones. The prediction by a non-linear predictor is in general computationally more complex than by a linear predictor. We have experimentally shown that no substantial improvement is achievable with the non-linear one. We use only the linear predictor.

Confidence measure

We do not propose any confidence measure for tracking with a single sequential predictor. In general, every standard confidence measure can be used, e.g., SSD or a learned classifier. If an object is modeled by a set of predictors, RANSAC determines the number of outliers as a side product of the pose estimation. The number of outliers provides a measure of confidence of the estimated pose. Since we did not investigate this issue in detail, the tracker failure is reported if 50% of outliers is reached.

5.11 Conclusions

We proposed a learning approach to tracking that explicitly minimizes computational complexity of the tracking process subject to user-defined probability of failure (loss-of-lock) and precision. In our approach, the object is modeled by a set of local motion predictors estimating translations. Object motion is estimated from these translations by RANSAC. Local motion predictors, their locations and number as well as the number of RANSAC iterations are subject of the optimization. Since the tracker is formed by a Number of Sequences of Learned Linear Predictors, we refer to it as NoSLLiP tracker.

In experiments, the NoSLLiP tracker was tested on approximately 12 thousands frames with a labeled ground truth, showing that the NoSLLiP tracker achieves a significantly smaller number of loss-of-locks than SIFT detector, LK tracker or Jurie's tracker. Since all the time-consuming computations are performed in the off-line 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 $30\mu s$.

We encourage the reader to download a MATLAB implementation of the proposed methods of learning and tracking and an additional material from `http://cmp.felk.cvut.cz/demos/Tracking/linTrack/`.

6 Anytime learning for sequential predictors

This part corresponds to journal article [107]. It is, like the previous chapter about sequential predictors. It essentially provides an alternative to the min-max estimation presented in the previous chapter.

We propose an anytime learning procedure for the Sequence of Learned Linear Predictors (SLLiP) tracker. Since learning might be time-consuming for large problems, we present an anytime learning algorithm which, after a very short initialization period, provides a solution with defined precision. As SLLiP tracking requires only a fraction of the processing power of an ordinary PC, the learning can continue in a parallel background thread continuously delivering improved, i.e. faster, SLLiPs with lower computational complexity and the same precision.

The proposed approach is verified on publicly-available sequences with approximately 12 thousands ground-truthed frames. The learning time is shown to be twenty times smaller than standard SLLiP learning based on linear programming, yet its robustness and accuracy is similar. Superiority in the frame-rate and robustness in comparison with the SIFT detector, Lucas-Kanade tracker and Jurie's tracker is also demonstrated.

6.1 Introduction

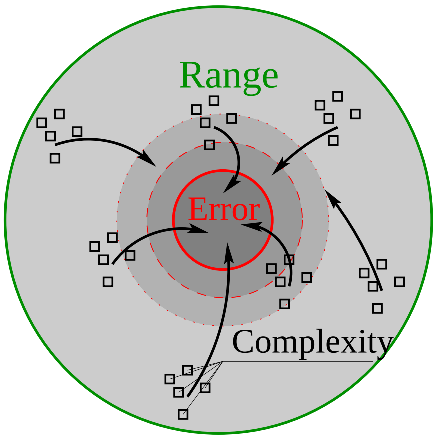
The main contribution of this chapter is a new *anytime* learning approach which, after a very short initialization period, provides a solution with predefined precision. The solution is continuously improved, i.e. the SLLiPs with lower complexity and defined precision allowing for faster tracking are 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 a bound [52] searching approach is used.

If no constraint on the learning time is imposed, the anytime learning algorithm finds a globally optimal solution with respect to a certain class of predictors. If time consuming learning is not acceptable, 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.

We consider only linear predictors, nevertheless, the method is easily extended to an arbitrary polynomial class by the data lifting. For instance, particular monomials can be considered as additional features.

Abbreviation	Meaning
LK	Lucas-Kanade tracker [56]
LS	Least Squares
MM	Minimax
LLiP	Learned Linear Predictor
SLLiP	Sequential LLiP
NoSLLiP	Number of SLLiPs
MM SLLiP	SLLiP learned by [101]
LS SLLiP	SLLiP anytime learning

(a) Table of used abbreviations.



(b) Definitions.

Figure 6.1: (a) Table of used abbreviations. (b) Definitions: The range, complexity and prediction error of a learned linear predictor.

6.2 Problem formulation

In this section, we introduce formal definitions of LLiP and SLLiP and formulate their learning as a constrained optimization problem. Let us suppose we are given an image $\mathcal{I}$ of an object to be tracked. Object motion is robustly determined by RANSAC from local motions of some points on the object. These points are called *reference points* and their motion is estimated from their neighbourhoods. For motion predictors, it is not necessary to use all neighbourhood pixels, because sufficient precision is achievable even with smaller number of pixels. Therefore only a selected subset of pixels $X = \{\mathbf{x}_1 \dots \mathbf{x}_c\}$, called *support set*, is used. LLiP estimates motion of the reference point from the intensities observed on the support set. These intensities are stored in the *observation vector* denoted $\mathbf{I}(X)$.

We denote $(\mathbf{t} \circ X)$ the support set warped by a motion with parameters $\mathbf{t}$. For example, if the considered motion is a 2D translation, then $(\mathbf{t} \circ X) = (X + \mathbf{t}) = \{(\mathbf{x}_1 + \mathbf{t}), \dots, (\mathbf{x}_c + \mathbf{t})\}$. There is a mapping (rendering) 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 the set of motions $\mathbf{t}$ which could have generated the observation $\mathbf{I}(\mathbf{t} \circ X)$. This mapping assigns a p-vector of motion parameters to a c-vector of the observation.

Definition 11 Linear predictor *(LLiP) is an ordered pair* $\varphi = (\mathtt{H}, X)$*, which assigns p-vector of motion parameters* $\mathbf{t} = \mathtt{H}\mathbf{I}(X)$ *to c-vector of observations* $\mathbf{I}(X)$*, where* $\mathtt{H} \in \mathcal{R}^{p \times c}$.

All predictors φ are characterized by the following parameters, see also Figure 6.1:

Definition 12 Complexity $c(\varphi) = |X|$ *of predictor* φ *is the cardinality of the predictor's support set* X.

Definition 13 Range $R(\varphi)$ *of the predictor* φ *is a set of motion parameters.*

Definition 14 Error *of predictor* $\varphi = (\mathtt{H}, X)$ *for range* $R(\varphi)$ *is* $\lambda(\varphi) = E\Big(\|\mathbf{t} - \mathtt{H}\mathbf{I}(\mathbf{t} \circ X)\|_2^2\Big)$, $\forall \mathbf{t} \in R(\varphi)$, *where* $E(.)$ *denotes the expectation value with respect to* $\mathbf{t}$ *uniformly distributed on* $R(\varphi)$[1]*.*

The predictor complexity approximately corresponds to the number of multiplications and sums necessary for motion estimation. It is clear that there is no ideal predictor which would simultaneously have a (very) low complexity, (very) large range and (very) small error. It is easy to see that the higher the complexity, the better the prediction. However, as the complexity increases towards the complete template, the improvements become less and less significant. In general, for large ranges it is very difficult to achieve a good prediction with any complexity. In order to overcome this limitation, we developed a *sequential predictor* $\Phi = (\varphi_1 \dots \varphi_m)$. Since the sequential predictor is provably superior to a single monolithic predictor, it allows a lower complexity for the higher precision. A vector of motion parameters $\mathbf{t}$ is predicted in m steps as follows:

$$\begin{aligned}
&\mathbf{t}_1 = \mathtt{H}_1\big(\mathbf{I}(X_1)\big),\ \mathbf{t}_2 = \mathtt{H}_2\big(\mathbf{I}(\mathbf{t}_1 \circ X_2)\big),\\
&\mathbf{t}_3 = \mathtt{H}_3\big(\mathbf{I}(\mathbf{t}_2 \circ \mathbf{t}_1 \circ X_3)\big),\ \dots,\ \mathbf{t}_m = \mathtt{H}_m\Big(\mathbf{I}\big((\bigcirc_{i=1}^{m-1} \mathbf{t}_i) \circ X_m\big)\Big),\\
&\mathbf{t} = \bigcirc_{i=1}^{m} \mathbf{t}_i,
\end{aligned} \tag{6.1}$$

The first vector of motion parameters $\mathbf{t}_1$ is directly predicted from intensities observed *at locations defined by the support set* X_1. The second predictor estimates motion parameters $\mathbf{t}_2$ from intensities $\mathbf{I}(\mathbf{t}_1 \circ X_2)$ observed on the its support set warped by $\mathbf{t}_1$, and so on. The advantage is that each predictor in a sequence is more and more specific, using a smaller range which corresponds to the accuracy of the preceding predictor.

Definition 15 Sequential predictor *(SLLiP) is an m-tuple* $\Phi = (\varphi_1, \dots, \varphi_m)$ *of predictors* $\varphi_i \in \omega$, $i = 1 \dots m$, *where* ω *is a set of predictors.*

The set of predictors ω can include all possible predictors, or its convenient subset. Because of the computational complexity of the learning process, only the predictors with $\mathtt{H}$ minimizing their prediction error for a given support set and training set, will be considered further.

Definition 16 *The* optimal sequential predictor *is a sequential predictor*

$$\Phi^* = \operatorname*{arg\,min}_{\Phi \in \omega^m} \left\{ \sum_{i=1}^{m} c(\varphi_i) \ \middle|\ \lambda(\varphi_m) \le \lambda^* \right\}, \tag{6.2}$$

where λ^* *is predefined prediction error,* c *is predictor complexity,* ω *is a set of predictors and* $\omega^m = \omega \times \omega \cdots \times \omega$ *is a set of sequential predictors of length* m.

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

6.3 Anytime learning of SLLiP

We define learning as a search for the optimal SLLiP subject to a predefined prediction error (Definition 16). In general, the learning procedure consists of two steps: support set selection and SLLiP optimization. The support set selection is a combinatorial problem, the solution of which might be time consuming [58,97]. Since we are interested in applications where the learning time is an issue, a randomly selected support set is used instead. The SLLiP optimization is also simplified by restricting ω to be a class of LLiPs with the minimal prediction error (Definition 14). Nevertheless, the proposed learning algorithm can be used to find the globally optimal solution with 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 [101] (previous chapter).

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. In Section 6.3.1, a training set construction from a single image is described. Section 6.3.2 presents SLLiP learning.

6.3.1 Training set construction

Given a predefined range of motions, within which the tracker is assumed to operate, we perturb the support set by 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 6.2. Given the observed intensities, we search for a mapping assigning motion $\mathbf{t}^i = (\mathbf{q}^i)^{-1}$, which warps X^i as close as possible to the original support set X. These examples are stored in matrices $\mathtt{I} = \left[\mathbf{I}^1 \ldots \mathbf{I}^d\right]$ and $\mathtt{T} = \left[\mathbf{t}^1 \ldots \mathbf{t}^d\right]$. The ordered triple $(\mathtt{I}, \mathtt{T}, \mathtt{X})$ of such matrices and ordered d-tuple of support sets $\mathcal{X} = \left\{X^1 \ldots X^d\right\}$ composes a *training set.*

6.3.2 Learning algorithm

In this section, we describe the method searching for the optimal sequential predictor given a training set $(\mathtt{I}, \mathtt{T}, \mathcal{X})$ generated on image $\mathcal{I}$. Since we restricted the set of considered LLiPs ω to the set of LLiPs minimizing prediction error λ, the LLiP learned from the training set is $\varphi = (\mathtt{H}^*, X)$, where

$$\mathtt{H}^* = \operatorname*{arg\,min}_{\mathtt{H} \in \mathcal{R}^{p \times c}} \|\mathtt{H}\mathtt{I} - \mathtt{T}\|_F^2 = \mathtt{T}\mathtt{I}^+ \tag{6.3}$$

and X is the support set aligned with the object.

Ideally, the predictor learned according to Equation (6.3) would transform intensities $\mathtt{I}$ to motions $\mathtt{T}$. However, such predictor usually does not exist. Therefore the observed intensities are transformed into motion parameters $\mathtt{T}^{(1)}$ which are as close as possible to the desired motions $\mathtt{T}$. We warp each support set $X^i \in \mathcal{X}$ by motion parameters $\mathtt{T}^{i,(1)}$ obtaining the new support set $X^{i,(1)} = \mathtt{T}^{i,(1)} \circ X^i$. Denoting $\mathtt{I}^{i,(1)}$ the intensities observed on these newly obtained support sets $X^{i,(1)}$, we form the new

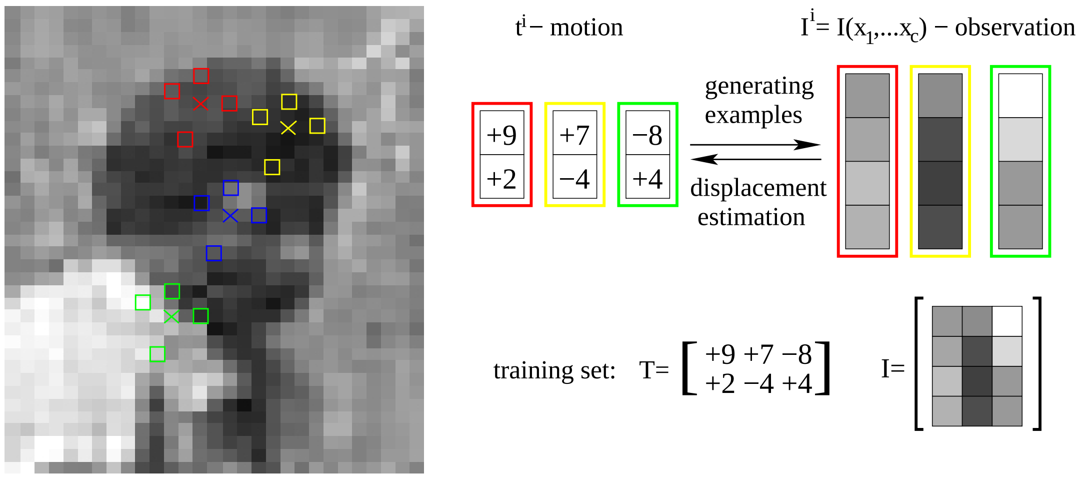


Figure 6.2: An image patch is perturbed by the motion parameters within a predefined range in order to create a set of synthesized examples of observed intensities $\mathbf{I}^i$ and motions $\mathbf{t}^i$.

training set $(\mathtt{I}^1, \mathtt{T}, \mathcal{X}^1)$. 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, we define training set *invoked by sequential predictor* as $\mathcal{T}(\Phi, c)$.

As already mentioned, the size of the support set influences the prediction error. Removing some pixels from the support set necessarily results in error increase[2]. Given a training set $\mathcal{T}(\Phi, c_1)$ we can simply generate a training set $\mathcal{T}(\Phi, c_2)$, $c_2 < c_1$ for the predictor with a lower complexity c_2 by removing corresponding number of pixels from $\mathcal{X}$ and corresponding number of rows from matrix $\mathtt{I}$. We refer to this process as training set *restriction*[3].

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 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 and bound [52] 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 (6.3). They correspond to the SLLiPs of the length equal to one. One of these SLLiPs, Φ, is expanded in the next iteration. The expansion means that Φ is successively extended by LLiPs with different complexities learned on training set invoked by itself $\mathcal{T}^i(\Phi, c)$. This process creates $|C|$ new SLLiPs, which could be expanded in further iterations. Once a SLLiP with a sufficiently small prediction error (feasible solution) is found, all other partially

[2]Proof of this claim is detailed in [101] (previous chapter).

[3]Since the support set is selected randomly, its restriction is random as well. If, for instance, the greedy construction [58,97,101] had been used, then the order of the selection would have provided the importance measure of the support pixels. The lastly selected pixels would have been removed firstly in the restriction.

constructed SLLiPs with a higher complexity are terminated, i.e., they will never be expanded. The smallest complexity c^* of the feasible solution is saved and once any SLLiP reaches a higher complexity it is automatically terminated.

The learning process is summarized in Algorithm 1; see also Figure 6.3, 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 set to 10% of the object size. In the first iteration, two LLiPs with complexities 20 and 300 are learned, denoted φ_1 and φ_2. Obviously, the LLiP with the higher complexity achieves lower prediction error $\lambda(\varphi_2) = 0.15$. Since no solution has been found, φ_2 is expanded in the second iteration, i.e. we learn two further LLiPs, denoted φ_{21} and φ_{22}, on the training set invoked by φ_2. Since both newly constructed SLLiPs $\Phi_1 = (\varphi_2, \varphi_{21})$ and $\Phi_2 = (\varphi_2, \varphi_{22})$ achieve sufficiently low prediction error, i.e. smaller than $\lambda^* = 0.1$, the one with the lower complexity, i.e., $c(\Phi_1) = 300 + 20 = 320$, is selected and the other one, Φ_2, is terminated. Φ_1 could be immediately used for tracking, while the learning can continue: in the third iteration, φ_1 is expanded. Since $c(\varphi_1\varphi_{12}) = 300 + 300 = 600 > c(\Phi_2) = 320$, this SLLiP is terminated. In the remaining iterations the not terminated SLLiP is further expanded till the solution, SLLiP Φ_3 consisting of 5 LLiPs, is reached. Since the complexity $c(\Phi_3) = 5 \times 20 = 100$ is smaller than $c(\Phi_1) = 320$, Φ_1 is replaced by Φ_3. And since there are no more SLLiPs to expand, Φ_3 is accepted as the final solution.

Of course, it is likely that better solution exist, consisting from the LLiPs with complexities not constrained to $C = \{20, 300\}$, but this is just a toy example demonstrating the learning process. In practice we work with $|C| \in \{10 \dots 15\}$.

Note that the selection strategy $\mathcal{S}$ which selects a SLLiP from Ω (step 4), may influence the learning behavior. However, if Algorithm 1 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 can start. Then 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 stopping 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^*.

```
Input:
  • Range $R$ within which SLLiP is expected to operate.
  • Set of considered complexities $C$.
  • Predefined accuracy $\lambda^*$.
  • 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 $\Omega$ of learned active SLLiPs as a set of LLiPs learned on
   $\mathcal{T}^0(c)$, $\forall c \in C$ according Equation (6.3).
4. $\Phi = \mathcal{S}(\Omega)$, $\Omega = \Omega \setminus \Phi$ (Select and remove $\Phi$ according to a strategy $\mathcal{S}$.)
5. For each $c \in C$: (expand $\Phi$)
   a) Generate training set $\mathcal{T}^i(\Phi, c)$ invoked by $\Phi$.
   b) Learn LLiP $\varphi$ for $\mathcal{T}^i(\Phi, c)$ according to Equation (6.3).
   c) $\Phi' = (\Phi, \varphi)$,  $\Omega = \Omega \cup \Phi'$ (add the new SLLiP to $\Omega$)
   d) If $c(\Phi') < c^*$ 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 $\Phi^*$
```

[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 1 - anytime learning of SLLiP.

6.4 Experiments

The proposed method is verified on real sequences with planar objects. The object is represented as a Number of SLLiPs (NoSLLiP), which estimates local translations at a few points on the object. Object motion, i.e. a homography, is determined from

object	SLLiP learning	learning* time [sec]	processing [fps]	loss-of-locks	mean-error [%]
MP	LS	11	27.6	17/6935	$[1.4, 1.3, 1.1, 1.1]$
MP	MM [101]	310	18.9	13/6935	$[1.3, 1.8, 1.5, 1.6]$
TOWEL	LS	16	33.3	2/3229	$[1.6, 1.8, 1.1, 1.5]$
TOWEL	MM [101]	310	21.8	5/3229	$[3.0, 2.2, 1.4, 1.9]$
PHONE	LS	21	25.6	55/1799	$[7.3, 7.1, 10.6, 6.5]$
PHONE	MM [101]	310	16.8	20/1799	$[1.2, 1.8, 2.6, 1.9]$

Table 6.1: Comparison of robustness and accuracy of NoSLLiP learned by anytime algorithm (LS) and minimax algorithm (MM) proposed in [101].
*Learning time is an average time required per one SLLiP.

these local translations by the RANSAC. Note that although we work with planar objects in order to avoid problems of 3D reconstruction, the proposed trackers could be attached to a 3D model with a reasonable texture, as was shown in [101].

The quantitative evaluation of the robustness and accuracy of SLLiPs is conducted on sequences with 3 different objects (MOUSEPAD-MP, TOWEL and PHONE), where ground truth positions of the object corners in total number 11963 frames were manually labeled[4]. 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 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. 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 Algorithm 1 with no time constraint (method LS) are presented. Second row contains results for SLLiPs learned by minimax [101] (method MM). The minimax learning minimize the size of a compact region within which all predictions lie (uncertainty region) instead of the square of Euclidean error, therefore higher robustness is achieved, but the learning is 10-20 times longer. Tracking accuracy of MM and LS methods varies with data; the accuracy is similar for MP and TOWEL sequences, but PHONE object contains similar repetitive structure (buttons) which make the LS accuracy significantly worse.

Robustness of the predictor is given by the shape of the error distribution, because the higher the probability of large errors the higher the probability of the predictor failure in the next frame due to its initialization out of its range. Shape of MM SLLiP distribution (blue solid line) and LS SLLiP distribution (red solid line) are shown in Figure 6.5. We observe that LS SLLiPs are more likely to have higher errors in difficult cases however, the accuracy in easier cases is higher. In addition to this we can also compare predefined uncertainty region λ_0 of MM SLLiP, predefined prediction

[4]These sequences in conjunction with the ground truth are available at `ftp://cmp.felk.cvut.cz/pub/cmp/data/lintrack/index.html`

method	processing [fps]	loss-of-locks	mean-error [%]
SLLiP LS	27.6	17/6935	$[1.4, 1.3, 1.1, 1.1]$
SLLiP MM [101]	18.9	13/6935	$[1.3, 1.8, 1.5, 1.6]$
SIFT [55]	0.5	281/6935	$[1.6, 1.2, 1.5, 1.4]$
LK tracker [56]	2.6	398/6935	$[2.3, 2.2, 2.5, 2.5]$
LLiP LS [48]	24.4	1083/6935	$[5.9, 6.0, 6.7, 6.7]$
LLiP LS [48] half-range	24.2	93/6935	$[3.1, 2.3, 2.7, 4.0]$

Table 6.2: Comparison of robustness and accuracy of SLLiP, LK, SIFT and LLiP tracker on MP sequence.

error ϵ_0 of LS SLLiP and true error distribution on ground truthed data (MOUSEPAD sequence). In this experiment we learned 35 SLLiPs with different ranges covering the mousepad. MM SLLiPs are learned to achieve uncertainty region $\lambda_0 = 5\%$ (blue dot-dashed line), LS SLLiPs are learned for prediction error $\epsilon_0 = 3\%$ (red dot-dashed line). Both the uncertainty region and the prediction error are relative to SLLiPs range. λ_0, ϵ_0 were chosen experimentally in order achieve the best performance of SLLiPs. Lower values result in a higher complexity and a consequent over-fitting. The error is evaluated on those frames, in which the inter-frame motion is smaller than the learning range of SLLiPs.

Table 6.2 compares the NoSLLiP tracker to the state-of-the-art Lowe's SIFT detector [55] (method: SIFT)[5], Lucas-Kanade tracker [56] (method: LK tracker) and Jurie's LS LLiP tracker [48] (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 the camera, but its basin of attraction was in many frames insufficient for correct motion estimation, failing for fast motions.

Since we work with a non-optimized implementation of the LK tracker, the presented frame-rate in this experiment could not serve for a speed comparison. Nonetheless the SLLiP computational complexity is clearly smaller than the complexity of the LK tracker. 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.

The learning procedure proposed in Algorithm 1 might be time consuming if the set of considered LLiPs is too large. Since a long learning time might not be acceptable for

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

some types of applications, either the set of considered LLiPs or the maximum number of iterations have to be restricted. However, the constraint on maximum number of iterations affects the optimality of the found SLLiP, and therefore we show average complexity of the best found solution as a function of iterations in Algorithm 1. Figure 6.6 presents this function for different sets of considered LLiPs. One can see that the learning time could be decreased 2-3 times without a significant increase of the solution complexity.

Note that there is also another option besides premature interruption of the learning procedure. The anytime learning algorithm, after a short initialization procedure, provides a solution - SLLiP with higher complexity but predefined precision. Having this SLLiP the tracking can immediately start. Since the tracking requires only a fraction of the processing power of an ordinary PC, the learning need not to 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 a 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 [26].

6.5 Conclusions

We proposed a fast learning algorithm for the SLLiP learnable tracker. Unlike the minimax learning procedure, the new algorithm has the anytime property and outputs progressively faster SLLiPs satisfying a user defined accuracy and range. The learning process very quickly returns a SLLiP which is slow, but satisfies the user-defined conditions on accuracy and range. During tracking, the learning is run in a background thread and gradually improves the SLLiP tracker.

The method was quantitatively evaluated on approximately 12 thousands labeled frames with three different planar objects. The performance and robustness superiority of the SLLiP tracker in comparison with Lucas-Kanade tracker [56], SIFT detector [55] and Jurie's LLiP tracker [48] was demonstrated. We encourage the reader to download sequences, ground-truth data and a MATLAB implementation which is available at `http://cmp.felk.cvut.cz/demos/Tracking/linTrack`.

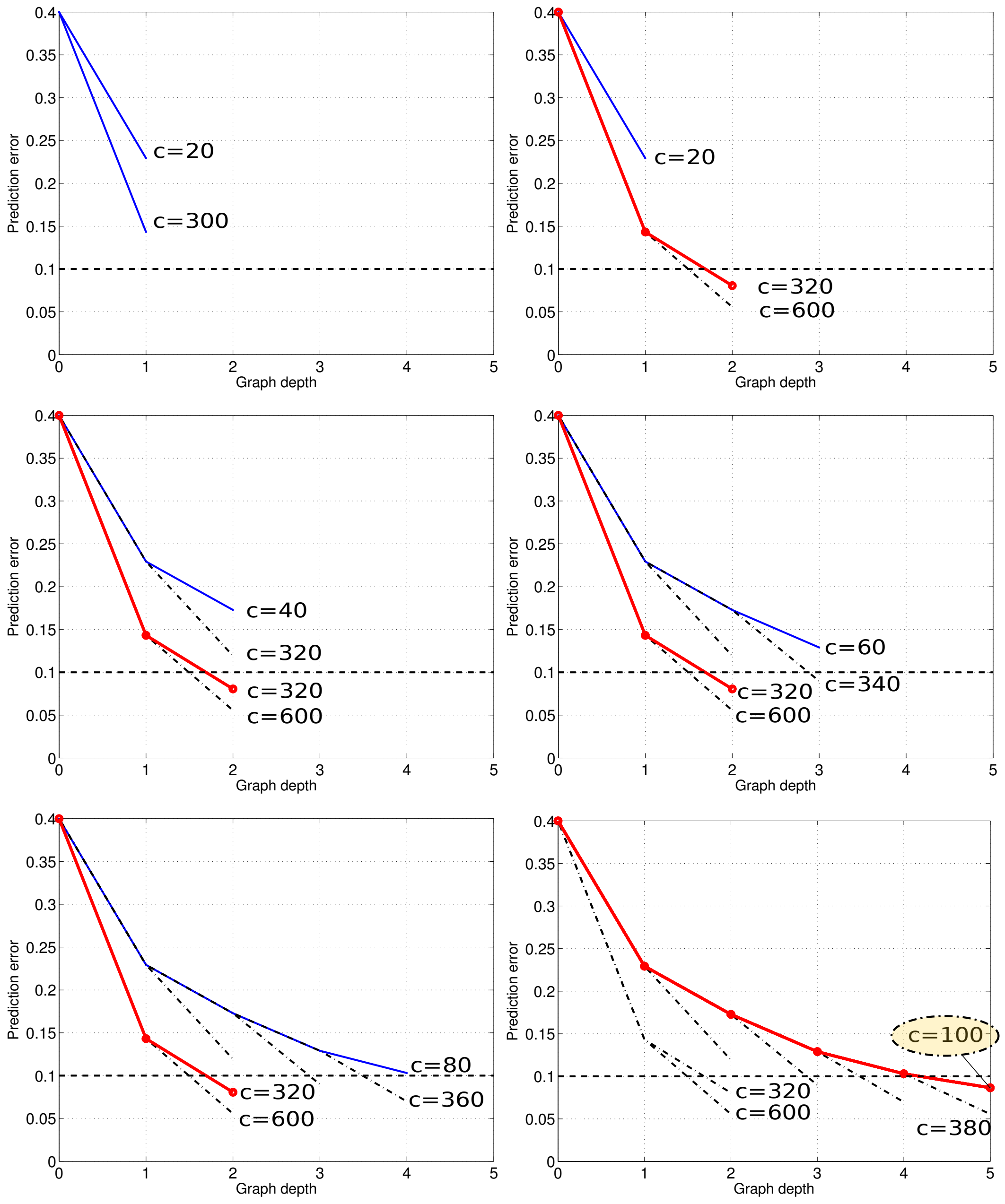


Figure 6.3: Demonstration of progress of Algorithm 1 for a toy example with $C = \{20, 300\}$, $|C| = 2$, $R = 0.4$ and $\lambda^* = 0.1$. Blue denotes a set of current SLLiPs Ω, black denotes terminated SLLiPs and red delineates solution Φ^* with the lowest complexity so far.

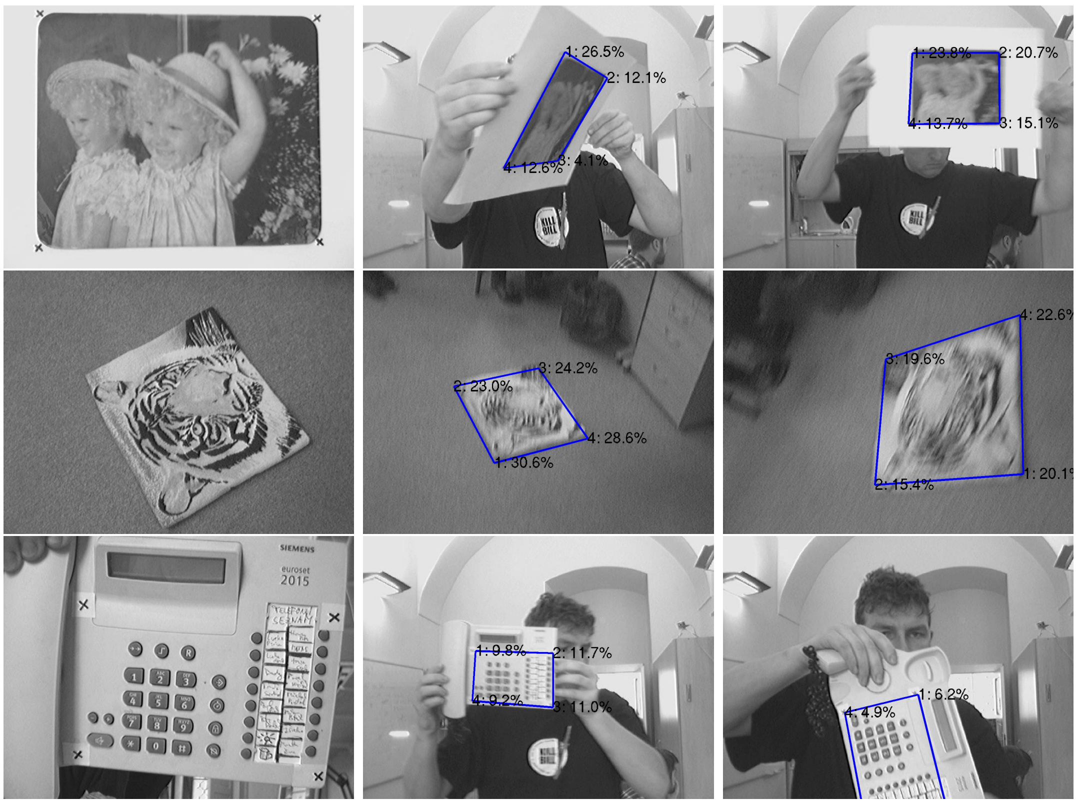


Figure 6.4: The left column shows images used for training. The middle and right columns demonstrate some successfully tracked frames with a strong motion blur from the testing sequences. The blue rectangle delineates the object. Percentage values in corners are current corner speeds related to the current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

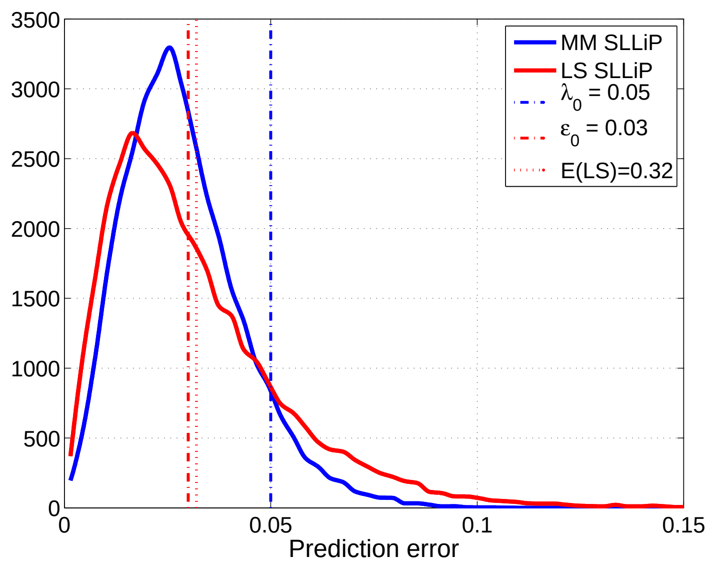


Figure 6.5: **Accuracy analysis:** Comparison of the predefined uncertainty region λ_0 of MM SLLiPs (blue dot-dashed line), the predefined prediction error ϵ_0 of LS SLLiPs (red dot-dashed line) and the true error distribution (blue and red solid lines) on the MOUSEPAD sequence.

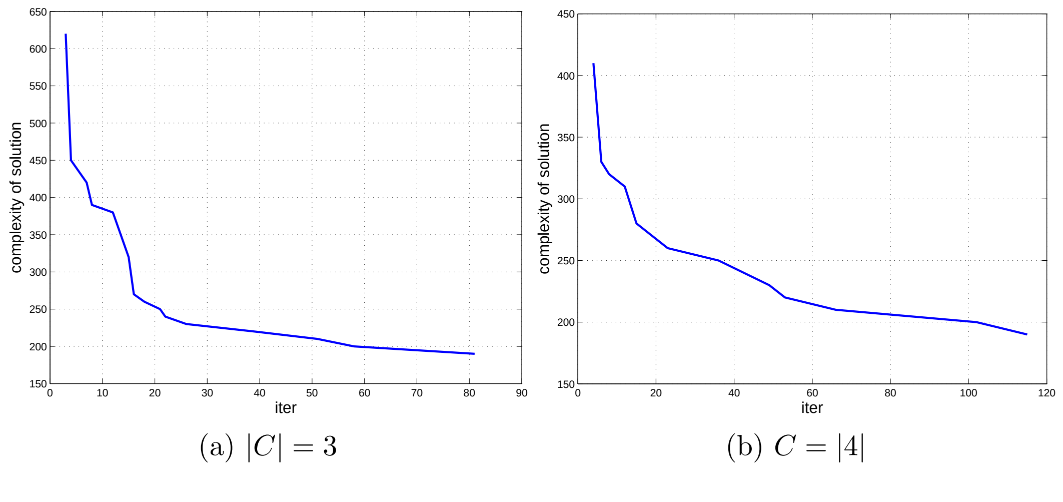


Figure 6.6: Average SLLiP complexity as a function of iterations of Algorithm 1. (a) and (b) differ by the size of the set of considered complexities $|C|$. The higher the value of $|C|$, the larger the space of considered LLiPs.

7 Simultaneous learning of motion and appearance

This part corresponds to [106] and describes an attempt to accommodate varying appearance of an object.

A new learning method for motion estimation of objects with a significantly varying appearance is proposed. The varying object appearance is represented by a low dimensional space of appearance parameters. The appearance mapping and motion estimation method are optimized simultaneously. Appearance parameters are estimated by unsupervised learning. The method is experimentally verified by a tracking application on sequences which exhibit a strong varying illumination, non-rigid deformations and self-occlusions.

7.1 Introduction

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

The most natural approach to tracking is alignment of a *template* image $\mathbf{J}$ with image data $\mathbf{I}$ by an online optimization of a criterion function like the sum of square errors [56] $\|\mathbf{I} - \mathbf{J}\|^2$ or mutual information [23], see Figure 7.1a. Since the tracker has usually no prior information about the possible changes of the object appearance, the template is either not updated at all or updated from the last known position in terms of a partial or a whole template replacement by the aligned image [23,26,59]. However, such *hard* appearance update often results in drifting and consequently to the loss-of-lock.

If the changes of the object appearance could be explained by a reasonably small number of parameters $\boldsymbol{\theta}$, e.g. affine transformation [80], than the template is updated by another online optimization, which searches for the best template warp, see for example Figure 7.1b. If such appearance parametrization describes well all possible appearance variations and the motions are slow enough to initialize the optimization within the basin of attraction, the system works usually well. However, the appearance changes are often caused by a non-rigid deformation or non-trivial illumination change, which do not allow a simple parametrization.

Facing these problems, many authors [7,11,14,38,48] propose a learning stage, where the motions and/or possible object appearances are learned. Jurie and Dhome [48] suggest to learn a linear mapping between observed image intensities and corresponding motion $\mathbf{t}$. During the learning stage, a training image is perturbed by random motion parameters and the least square method estimates coefficients $\mathtt{H}$ of the searched

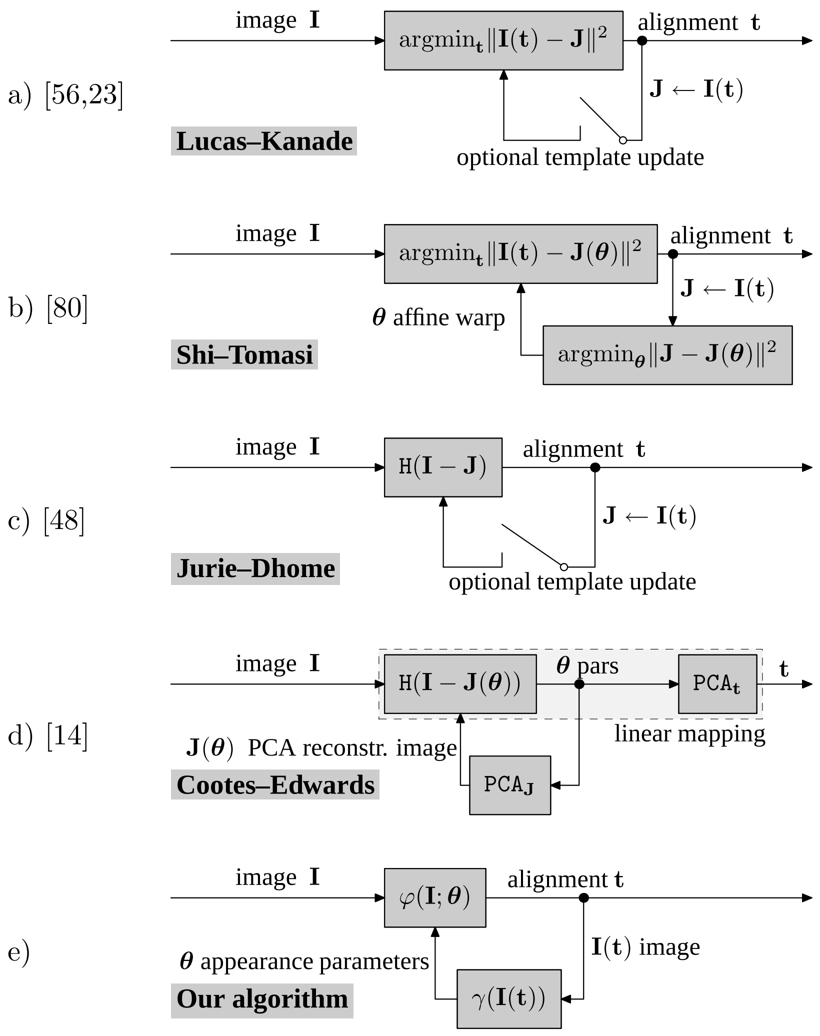

a) [56,23]
image I
$\text{argmin}_{\mathbf{t}}\|\mathbf{I}(\mathbf{t}) - \mathbf{J}\|^2$
alignment t
$\mathbf{J} \leftarrow \mathbf{I}(\mathbf{t})$
optional template update
Lucas–Kanade
b) [80]
image I
$\text{argmin}_{\mathbf{t}}\|\mathbf{I}(\mathbf{t}) - \mathbf{J}(\boldsymbol{\theta})\|^2$
alignment t
$\mathbf{J} \leftarrow \mathbf{I}(\mathbf{t})$
$\boldsymbol{\theta}$ affine warp
$\text{argmin}_{\boldsymbol{\theta}}\|\mathbf{J} - \mathbf{J}(\boldsymbol{\theta})\|^2$
Shi–Tomasi
c) [48]
image I
$\mathtt{H}(\mathbf{I} - \mathbf{J})$
alignment t
$\mathbf{J} \leftarrow \mathbf{I}(\mathbf{t})$
optional template update
Jurie–Dhome
d) [14]
image I
$\mathtt{H}(\mathbf{I} - \mathbf{J}(\boldsymbol{\theta}))$
$\boldsymbol{\theta}$ pars
$\mathtt{PCA}_{\mathbf{t}}$
t
linear mapping
$\mathbf{J}(\boldsymbol{\theta})$ PCA reconstr. image
$\mathtt{PCA}_{\mathbf{J}}$
Cootes–Edwards
e)
image I
$\varphi(\mathbf{I};\boldsymbol{\theta})$
alignment t
$\mathbf{I}(\mathbf{t})$ image
$\boldsymbol{\theta}$ appearance parameters
$\gamma(\mathbf{I}(\mathbf{t}))$
Our algorithm

Figure 7.1: State-of-the-art summary and the proposed approach.

linear mapping. During the tracking stage, the motion is estimated as the linear function of the difference image $(\mathbf{I} - \mathbf{J})$ between the image data and template, see Figure 7.1c. Jurie's approach avoids the problems of the basin of attraction, but it allows only the hard template update.

Cootes et al. [14] realized that once a training set is available, it is reasonable to use it also for the learning of the appearance. They proposed a paradigm where both the motion and appearance are projected by the PCA [32] to the lower dimensional space of some parameters. Given an image the learned linear mapping with coefficients $\mathtt{H}$ estimates the parameters which are used for both the template update and motion estimation by PCA back-projection, see Figure 7.1d.
Observing that

- the mapping between the input image $\mathbf{I}$ and the output motion parameters $\mathbf{t}$ is linear, see dashed line in Figure 7.1d 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 7.1e. 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 7.1e.

Section 7.2 describes acquisition of a training set, i.e., the set of the examples consisting of the intensities incoming into the tracker, the motion parameters to be estimated by the tracker and the intensities incoming into the appearance encoder. In Section 7.3, we define the criterion function to be minimized as the squared Euclidean distance (i.e. L_2-norm). In Section 7.4 the learning of φ and γ minimizing the criterion is described. Roughly speaking, the learning is the least squares bilinear function fitting problem, the minimum of which is searched by an exact line-search algorithm [34]. Section 7.5 presents experiments demonstrating an improvement to the state-of-the-art. The conclusions are in Section 7.6.

7.2 Training set construction

Let the object position be represented by a *reference point* (e.g., center of gravity of object pixels). Let us suppose that we are given a ground truthed sequence of images, in which the position of the object reference point is denoted. Given a predefined range of motions R within which the tracker is assumed to operate (e.g., radius of the desired basin of attraction), 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 neighborhood 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 columnwise 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 provided by the tracker is known in advance[1]. We perturb neighborhood of reference points within the range determined

[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.

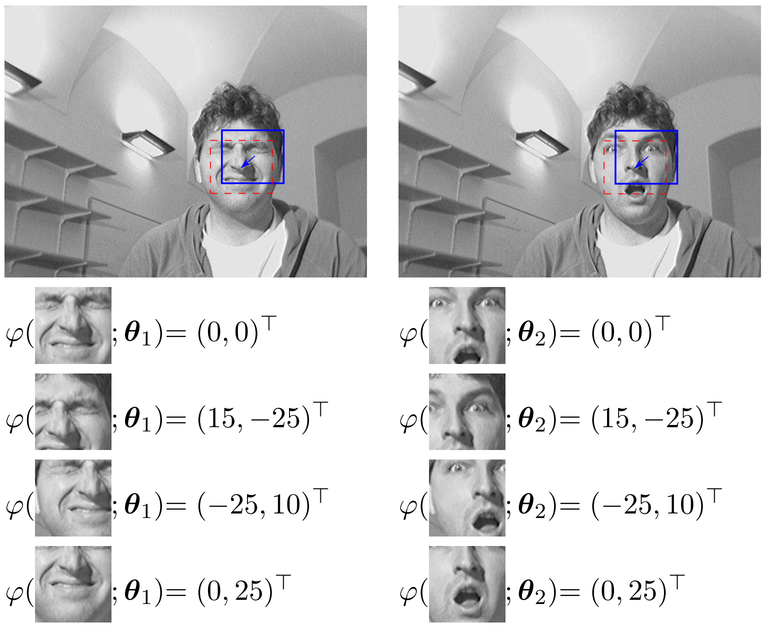

$\varphi(\;;\boldsymbol{\theta}_1) = (0,0)^\top$
$\varphi(\;;\boldsymbol{\theta}_2) = (0,0)^\top$
$\varphi(\;;\boldsymbol{\theta}_1) = (15,-25)^\top$
$\varphi(\;;\boldsymbol{\theta}_2) = (15,-25)^\top$
$\varphi(\;;\boldsymbol{\theta}_1) = (-25,10)^\top$
$\varphi(\;;\boldsymbol{\theta}_2) = (-25,10)^\top$
$\varphi(\;;\boldsymbol{\theta}_1) = (0,25)^\top$
$\varphi(\;;\boldsymbol{\theta}_2) = (0,25)^\top$

Figure 7.2: 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 $\mathbf{t} = (15,-25)^\top$. Reference point is the tip of the nose. Different appearances are encoded by different appearance parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$.

by the tracker accuracy creating the set of images $\mathtt{J} = \left[\mathbf{J}^1 \ldots \mathbf{J}^d\right]$ 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 this section the following notation is used:
Given a matrix $\mathtt{H}$:

- $\mathbf{h}_j^\top$ is row vector corresponding to j-th row,
- $\mathbf{h}^i$ is column vector denoting its i-th column,
- $h_j^i = [\mathtt{H}]_{i,j}$ denotes element at coordinates $[i,j]$,
- $\mathtt{H}^k$ is the approximation of $\mathtt{H}$ in k-th iteration,
- $||\mathtt{H}||_F^2 = \sum_{i,j} |h_j^i|^2$ is the squared Frobenius norm of $\mathtt{H}$,
- $\mathbf{1}_m = [1\ldots 1]^\top$ is $m \times 1$ unit vector,
- $\otimes$ is Kronecker product,

- ● is entry-wise product, sometimes called Hadamard product.

Because of simplicity, we restrict φ and γ to be from a class of linear mappings. Note that *data lifting* makes the same method work for any mapping formed as the linear combination of arbitrary intensity functions. For example, arbitrary polynomial mapping is equivalent to the linear mapping on the embedded space of monomials. It means, that we just transform the training set to the space of monomials and the rest of the learning process is the same.

Definition 17 Learned Linear Predictor *(LLiP) is ordered pair* $(\mathtt{H}, X)$ *which computes motion parameters* $\mathbf{t}$ *from the vector of image intensities observed on the set of pixels* X *as follows:*

$$\mathbf{t} = \mathtt{H}\mathbf{I}(X).$$

We further refer to the set of pixels as to the support set *and the linear mapping is called* regressor*.*

Note that the support set selection problem is beyond the scope of this text and we select it randomly.

We extend the LLiP to the form allowing parametrization by the appearance parameters, therefore Parameter Sensitive LLiP (PLLiP) is defined as follows:

Definition 18 Parameter sensitive Learned Linear Predictor *(PLLiP) is LLiP with the regressor* $\mathtt{H} = (\mathtt{H}_0, \dots, \mathtt{H}_m)$*, which computes motion parameters* $\mathbf{t}$ *from the observed image* $\mathbf{I}$ *given current appearance parameters* $\boldsymbol{\theta} = (\theta_1 \dots \theta_m)^\top$ *as follows*

$$\mathbf{t} = (\mathtt{H}_0 + \theta_1\mathtt{H}_1 + \dots + \theta_m\mathtt{H}_m)\,\mathbf{I}, \tag{7.1}$$

Referring to Figure 7.1e, we define *appearance encoder* to be the LLiP with regressor

$$\boldsymbol{\theta} = \gamma(\mathbf{J}) = \mathtt{G}\mathbf{J}, \tag{7.2}$$

which encodes current object appearance from previously aligned image $\mathbf{J}$ into parameters $\boldsymbol{\theta}$.
Tracker is PLLiP with the regressor

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

parametrized by the appearance parameters $\boldsymbol{\theta}$.

Definition 19 Tracking system *is the ordered pair* $(\mathtt{H}, \mathtt{G})$ *corresponding to tracker (PLLiP) and appearance encoder (LLiP) connected according to Figure 7.1e.*

Definition 20 Prediction error $e(\mathtt{H}, \mathtt{G})$ *of the tracking system* $(\mathtt{H}, \mathtt{G})$ *on the training set* $(\mathtt{I}, \mathtt{J}, \mathtt{T})$ *is*

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

Definition 21 *The* optimal tracking system *with respect to training set* $(\mathtt{I}, \mathtt{J}, \mathtt{T})$ *is the tracking system*

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

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 22 Complexity $C(\mathtt{H}, \mathtt{G}) \in \mathbb{R}$ *of the 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 seeks the optimal tracking system with respect to a training set, i.e., it solves problem (7.4). This is an unconstrained optimization problem, where the bilinear function is fitted into a high dimensional data in the least squares sense. We show later that problem (7.4) has a closed-form solution in $\mathtt{H}$ (respectively $\mathtt{G}$) if $\mathtt{G}$ (respectively $\mathtt{H}$) is fixed. Therefore the solution is sought by an *exact line-search method*[2], which successively minimizes criterion (7.4) along the direction $\mathtt{H}$ 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 a random matrix $\mathtt{G}^0$, the minimum of the 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 $\mathtt{G}^1$. The error is iteratively minimized in that manner, until a stopping condition is satisfied. The learning stops in our implementation if a relative error difference is smaller than some threshold ϵ. The number of appearance parameters m, the desired complexity C (or equivalently number of pixels n), the learning threshold ϵ and the training set $(\mathtt{I}, \mathtt{J}, \mathtt{T})$ (the range of the tracker, optionally) are the only inputs to the learning procedure, which is summarized in Algorithm 7.4.

[2]An exact line-search method finds the length of step which minimizes a criterion in a descent direction, see [34] for details.

Exact line-search algorithm for problem (7.4)

1. Randomly initialize matrix $\mathtt{G}^0$ and set the 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 step 2.

In order to derive the closed-form solution of step 2 (Algorithm 7.4), we rewrite criterion function (7.3) as follows:

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

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 (Algorithm 7.4) is

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

where $\mathtt{A}^+$ denotes the pseudo-inverse of $\mathtt{A}$. Note, that condition (7.8) may not assure a reasonable behavior 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 the closed-form solution of the step 3 (Algorithm 7.4), 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 (Algorithm **??**) is

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

Since the matrix $\mathtt{B}$ has the 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 $p \geq 1$. Thus it need not be considered any further.

We observed, that the proposed algorithm converges to the solutions, which has the same criterion value. If the criterion (7.4) was a convex or quasi-convex function it would be simple to prove that Algorithm 7.4 converges to a global minimum. However, the criterion is neither convex nor quasi-convex. We propose a conjecture that every local minimum of the criterion is simultaneously global. Using MAPLE, we analytically proved that the conjecture is true, but we were not able to generalize it for arbitrary number of variables. In this proof, we equaled the criterion gradient to zero and symbolically solved the algebraic system. The solution consists of a few manifolds but only one of them has a semidefinite Hessian, i.e., it 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 converges is the global one, 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 the *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 a significantly smaller prediction error if the nature of the appearance changes allows some reasonable parametrization. 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 7.4.

7.5.1 Results on real sequences

In this experiment, we compare the prediction error of the simple tracking system and the parameter sensitive tracking system of the same complexity. Note, that the simple tracking system is the special case of the parameter sensitive tracking system. We state that the 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

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

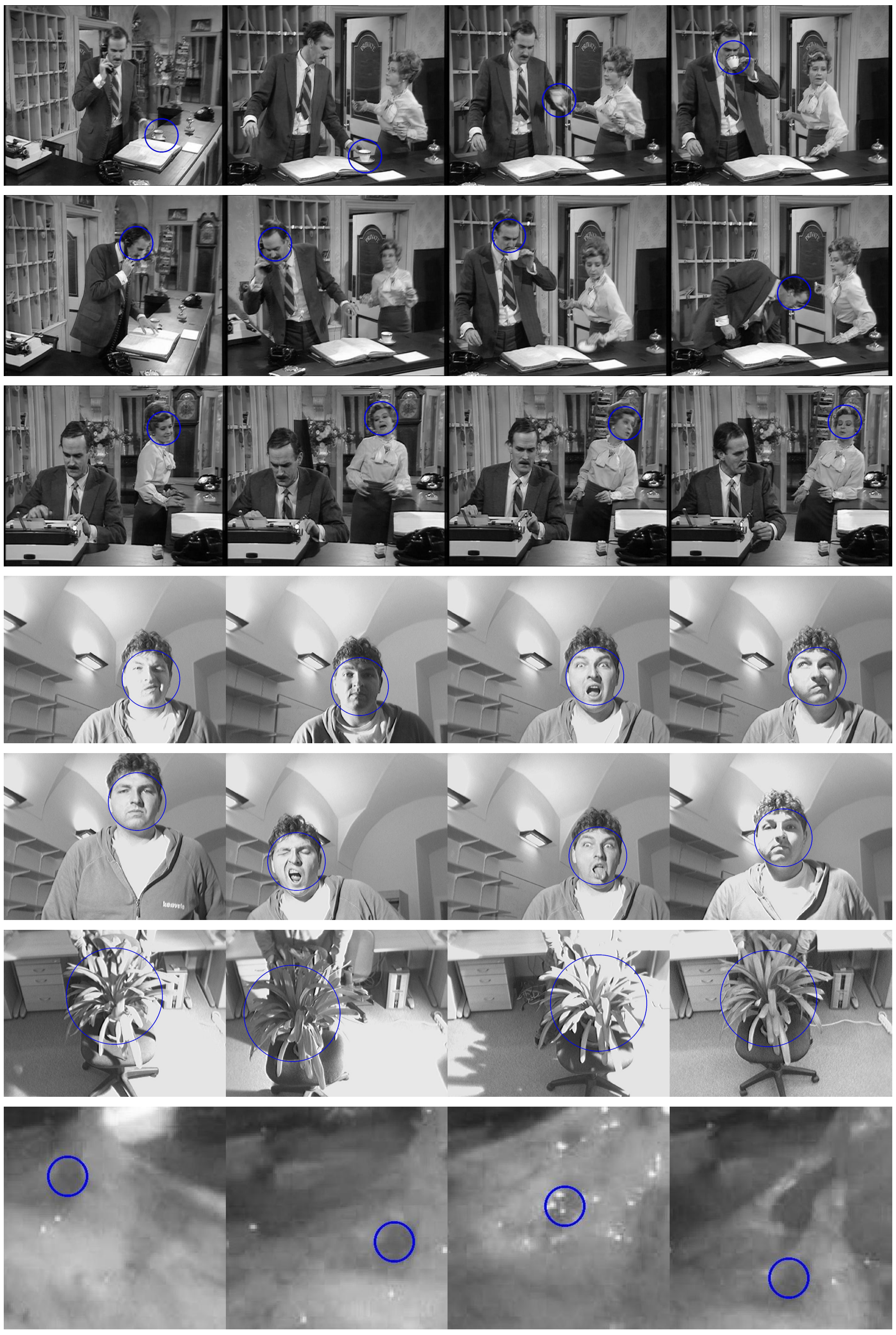


Figure 7.3: **Objects with variable appearance** caused by the illumination, non-rigid deformations and self-occlusions. Medical data presented in the last row are provided by Dr. Utz Kappert, Department of Cardiac Surgery, Heart Center Dresden University Hospital at the University of Technology Dresden.

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

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 coped by the tracker. In particular, the 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 a 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 the appearance strongly affected by non-trivial variable illumination and non-rigid deformations, see sixth row in Figure 7.3.

Notice that the prediction error of PLLiP was in all cases 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 was performed on the sequences with 100-400 frames. The tracker estimated only the translation ($p = 2$) in the range within the radius of 20 pixels, other degrees of freedom were represented by 1-5 appearance parameters. The non-optimized Matlab implementation of the learning procedure performing 10-40 iterations required about 20-300 seconds on an average machine (1xK8 3200+ MHz). Note, that the learning time depends mainly on the number of training examples and on the number of appearance parameters. The motion estimation time in the tracking procedure is negligible (say smaller than 1 ms) in contrast to the time required for the image capturing and its visualization.

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 the lower complexity, the error is smaller approximately about 30%. In the CUP experiment (first row in Table 7.1), the simple tracker lost the target during the fast motion (see the blurred image in Figure 7.3 1st row, 3rd column). In the BASIL experiment (second row in Table 7.1 and in Figure 7.3), both trackers succeeded in tracking. However, the parameter sensitive tracker was more accurate. In the third experiment (third row in Table 7.1 and in Figure 7.3), 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 demonstrated the 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

Object	**Parameter sensitive**				**Simple**		
	Error$_{\text{train}}$	Error$_{\text{test}}$	C	m	Error$_{\text{train}}$	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:** The prediction error is presented in pixels and the complexity follows definitions (7.5) and (7.14).

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 parametrization.

7.5.2 Properties of the appearance encoder

This experiment demonstrates relationship between the current object appearance and appearance parameters estimated by the learned appearance encoder. First and second rows of Figure 7.4 show four frames from the BASIL sequence and the corresponding values of the one-dimensional appearance parameter. The lowest values are associated with the head profile, middle values correspond to the front view and the highest values correspond to the partial head occlusion by the white cup. Third row of Figure 7.4 shows a sequence, in which the human head turns around. Appearance parameters obtained by the parameter encoder are depicted in the second row of Figure 7.4. Notice, that these parameters correspond naturally to the out-of-plane rotation almost exactly, without any explicit knowledge about that evidence.

7.5.3 Convergence of Algorithm 7.4

We study the convergence of Algorithm 7.4 in this section. In order to demonstrate the convergence, we experimentally show that every local minima of criterion (7.4) within which the algorithm is stopped is global, i.e., the same criterion value is achieved every time.

In the experiment, Algorithm 7.4 was one thousand times randomly initialized from the uniform distribution and 150 iterations on 5000 training examples generated from sequence HEAD 1 were computed. The covariance of achieved values of the criterion function (7.3), i.e., Mean Square prediction Errors (MSE) is 3.4×10^{-3}. See Figure 7.5

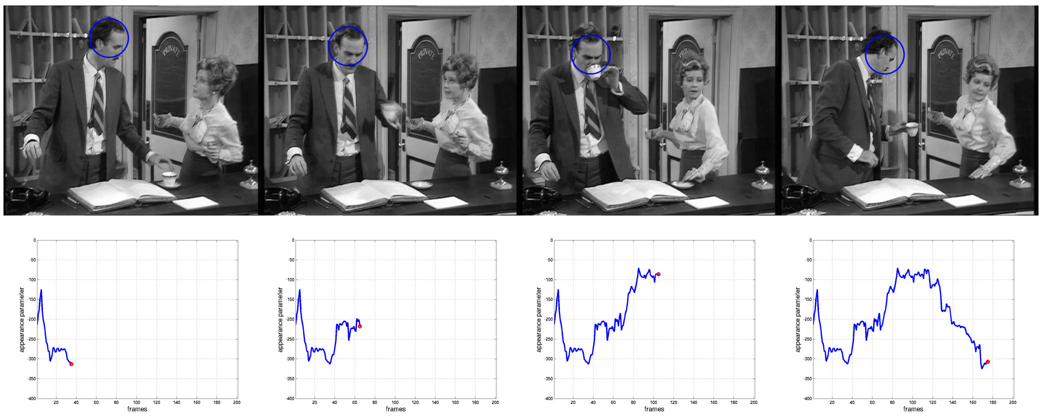
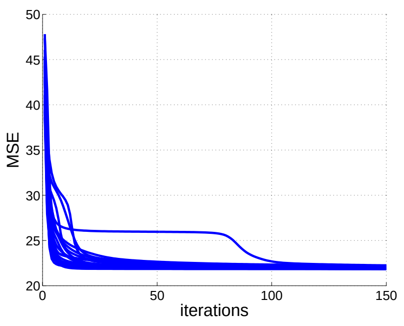
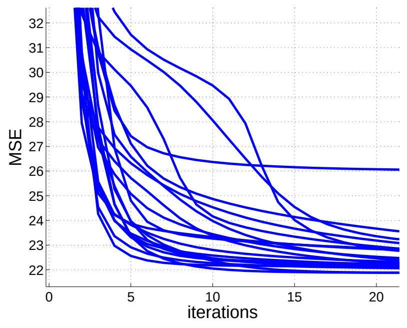


Figure 7.4: **Values of appearance parameters:** Pose parameters are only translations, the other degrees of freedom are modeled as appearance changes. Selected frames from sequences (first row) and the corresponding output of one-dimensional (second row) appearance encoder. The blue line shows the course of the value, the red dot denotes the current value corresponding to the image (tracking state) above.



(a) convergence of Algorithm 7.4



(b) detail

Figure 7.5: **Convergence analysis of the learning procedure:** Twenty convergence examples of randomly initialized algorithm.

for twenty convergence examples. From the detail depicted in Figure 7.5b, we conclude that 20 iterations are for most of the initializations sufficient.

7.6 Conclusions

We proposed a learnable tracking procedure suitable for objects with a significantly varying appearance. The method was experimentally verified on the challenging sequences, which exhibit strong variable illumination, non-rigid deformations and self-occlusions. The results were compared to the parameter insensitive tracking system, which was shown to be a special case of our approach. We demonstrated that on the same or lower complexity a smaller prediction error is achieved.

Part III

Advanced predictors and their applications

8 Incremental learning

This part mainly corresponds to [43] but some ideas appeared already in [106]. It advances the sequential linear predictors proposed in the previous part II in several ways. It introduces a more complex motion model, proposes a validation procedure and applies the on-line incremental learning in order to accommodate drifting appearance. It also discusses a possible method for the automatic selection of suitable training examples for the incremental learning by introducing a stability measure. The advanced predictor is combined with an object detector and it is applied in fast object-driven videobrowsing.

8.1 Introduction

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

Tracker adaptation and loss-of-track detection have been active research topics for many years. Jepson et al. [46] proposed WSL tracker (3 components - Wandering, Stable and Lost) - an *adaptive* appearance model which deals with partial occlusion and change in object appearance. It is a wavelet-based model, which allows to maintain a natural measure of the *stability* of the observed image structure during tracking. This approach is robust and works well with slowly changing object appearance. However, a high computational overhead precludes real-time applications. Ross et al. [74] propose an algorithm for incremental learning and adaptation of low dimensional eigenspace object representation with update of the sample mean and eigenbasis. Their approach appears to be robust to sudden illumination changes and does not need off-line learning phase before tracking however. The algorithm speed is lower than our needs.

For template-based trackers the, adaptation means continuous update of the tracked template. Tracking systems with *naive updates* modify the template after every tracking step [80]. Sub-pixel errors inherent to each match are stored in each update. These errors gradually accumulate resulting in the template drifting off the feature. Despite

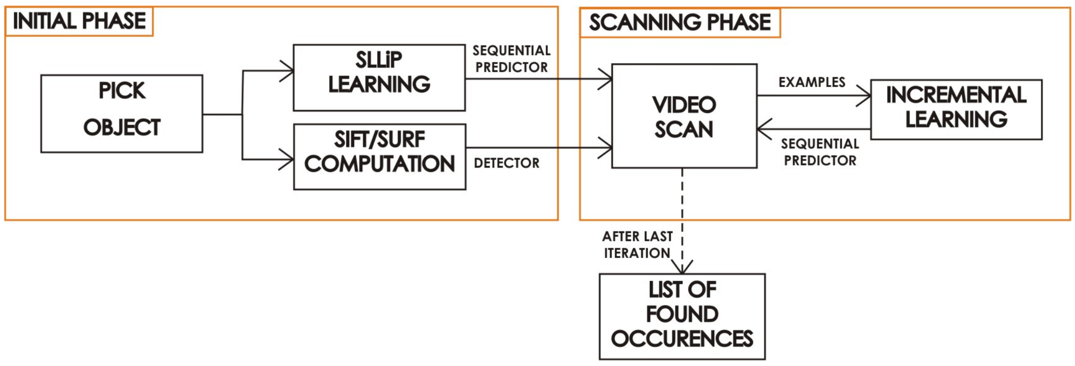


Figure 8.1: Video browsing procedure.

this drawback, the naive update is usually a better choice than no update at all. Matthews et al. in [3] propose a *strategic update* approach, which trades off mismatch error and drift. It is a simple but effective extension of the naive update. There are two template models kept during the tracking. The *updating template* is used for an initial alignment and the template from the first frame is then used in the error correction phase after the alignment. If the size of correction is too large, the algorithm acts conservatively by preventing the template to be updated from the current frame.

Recently, some authors wanted to bypass the exhaustive off-line learning stage. A purely on-line learning has been proposed by Ellis et al. in [25], where a bank of local linear predictors (LLiPs), spatially disposed over the object, are on-line learned. The appearance model of the object is learnt on-the-fly by clustering the sub-sampled image templates. The templates are clustered using the medoidshift algorithm. The clusters of appearance templates allow to identify different views or aspects of the target and also allow to choose the bank of LLiPs most suitable for current appearance. The algorithm also evaluates the performance of particular LLiPs. When the performance of some predictor is too low, it is discarded and a new predictor is learned on-line as a replacement. In comparison to our work, we do not throw away the predictors in sequence, but we incrementally train them with new object appearances in order to improve their performance.

Our learnable and adaptive tracking method, coupled with a sparsely applied SIFT [55] or SURF [4] based detector, is applied for the faster than real-time linear video browsing. The goal is to find all object occurrences in a movie. One of possible solutions of video browsing task would be to use a general object detector in every frame. As it appears [96], [65], it is preferable to use a combination of an object detector and a tracker in order to speed up the browsing algorithm and also to increase the true positive detections. We indeed aim at processing rates higher than real-time which would allow almost interactive processing of lengthy videos. Our yet preliminary Matlab implementation can search through videos up to eight times faster than the real video frame rate.

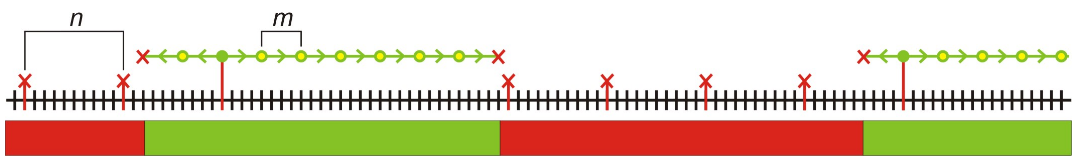


Figure 8.2: A typical video scan process. Vertical red lines depict frames, where the object detection was run. The red cross means negative detection or the tracking failure. The green line shows backward and forward object tracking. Green circle means positive object detection and yellow circle depicts successful validation.

8.2 Learning, tracking, validation and incremental learning

The user initiates the whole process by selecting a rectangular patch with the object of interest in one image. This sample patch is artificially perturbed and a sequential predictor is learned [107]. Computation of a few SIFT or SURF object descriptors completes the initial phase of the algorithm, see Figure 8.1. The scanning phase of algorithm combines predictor based tracking, its validation, and a sparse object detection. The predictor is incrementally re-trained for new object appearances. Examples for the incremental learning are selected automatically with no user interaction.

The scanning phase starts with the object detection running every $n-$th frame (typically with the step of 20 frames) until the first object location is found. The tracker starts from this frame on the detected position both in backward and forward directions. Backward tracking scans frames which were skipped during the detection phase and runs until the loss-of-track or until it reaches the frame with last found occurrence of the object. Forward tracking runs until the loss-of-track or end of sequence. The detector starts again once the track is lost. Tracking itself is validated every $m-$th frame (typically every 10 frames). The scanning procedure is depicted on Figure 8.2.

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

Training examples for incremental learning are selected automatically. The most problematic images-examples are actually the most useful for incremental training of the predictor. In order to evaluate the usefulness of a particular example, we suggest a stability measure. The measure is based on few extra predictions of the predictor on a single frame. It means, that we let the sequential predictor track the object in a single static image and we observe the predictors' behavior. See Section 8.2.3 for more details.

The sequential linear predictor validates itself. Naturally, the object detector may be also used to validate the tracking. For example, a well trained face detector will perform the same or better job when used to validate human face tracking. The motivation for using the sequential predictor for validation is its extreme efficiency, and robust performance. For more details about the tracking validation, see section 8.2.2.

8.2.1 Incremental learning of sequential learnable linear predictor

We extend the min-max learning of the Sequential learnable linear predictors (SLLiP) by Zimmermann et al. [107] in order to predict not only translation but also the affine deformation of the object. Next extension is the incremental learning of new object appearances. The predictor essentially estimates motion and deformation parameters directly from image intensities. It requires an off-line learning stage before the tracking starts. The learning stage consists of generating exemplars and estimation of regression functions. We use 2 SLLiPs - first for 2D motion estimation (2 parameters) and second for affine warp estimation (4 parameters). We have experimentally verified that, especially for a low number of training examples, this configuration is more stable than using just one SLLiP to predict all 6 parameters at once. Using smaller training set decreases the necessary learning time which is important for the foreseen applications. Because of speed, we opted for least squares learning of SLLiPs similarly, as suggested by Zimmermann et al. in their any-time learning algorithm [107].

Let denote the translation parameters vector $\mathbf{t}_t = [\Delta x, \Delta y]^T$ estimated by the first SLLiP, and the affine warp is parametrized by the parameters vector $\mathbf{t}_a = [\alpha, \beta, \Delta s_x, \Delta s_y]^T$ which is estimated by the second SLLiP. The 2×2 affine warp matrix $\mathtt{A}$ is computed as

$$\mathtt{A} = \mathtt{R}_\alpha \mathtt{R}_{-\beta} \mathtt{S} \mathtt{R}_\beta \, , \tag{8.1}$$

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

$$\mathtt{S} = \begin{bmatrix} 1 + \Delta s_x & 0 \\ 0 & 1 + \Delta s_y \end{bmatrix} . \tag{8.2}$$

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

$$\begin{aligned} \mathbf{x}' &= \mathtt{A}\mathbf{x} + \mathbf{t}_t \\ &= \mathtt{R}_\alpha \mathtt{R}_{-\beta} \mathtt{S} \mathtt{R}_\beta \mathbf{x} + \mathbf{t}_t, \end{aligned} \tag{8.3}$$

Tracking, learning and incremental learning will be explained for SLLiP with the general parameters vector $\mathbf{t}$. Equations are valid for both SLLiPs, which we use. SLLiP is simply a sequence of linear predictors. Predictors in this sequence estimate the parameters one after each other (see equation 8.4), thus each improving the result of previous predictor estimation and lowering the error of estimation. SLLiP tracks according to

$$\begin{aligned} \mathbf{t}_1 &= \mathtt{H}_1 I\left(X_1\right) \\ \mathbf{t}_2 &= \mathtt{H}_2 I\left(\mathbf{t}_1 \circ X_2\right) \\ \mathbf{t}_3 &= \mathtt{H}_3 I\left(\mathbf{t}_2 \circ X_3\right) \\ &\vdots \\ \mathbf{t} &= \bigcirc_{(i=1,\ldots,k)} \mathbf{t}_i, \end{aligned} \tag{8.4}$$

where I is the current image and X is a set of 2D coordinates spread over the the object patch—it is called *support set*. $I(X)$ is a vector of image intensities collected at image coordinates X. The operation $\circ$ means transformation of support set points using Equation 8.3, i.e. aligning the support set to fit the object using parameters estimated by the previous predictor in the sequence. Final result of the prediction is vector $\mathbf{t}$ which combines results of all predictions in the sequence. The model θ_s for SLLiP is formed by the sequence of predictors $\theta_s = |\{\mathtt{H}_1, X_1\}, \{\mathtt{H}_2, X_2\}, \ldots, \{\mathtt{H}_k, X_k\}|$. Matrices $\mathtt{H}_1, \mathtt{H}_2, \ldots, \mathtt{H}_k$ are linear regression matrices which are learned from training data.

In our algorithm, the SLLiP is learned from one image only and it is incrementally (re-)learned after each video scan. A few thousands training examples are artificially generated from the first image using random perturbations of parameters in vector $\mathbf{t}$, warping the support set accordingly and collecting the image intensities. The column vectors of collected image intensities are stored in matrix $\mathtt{D}_i$ and perturbed parameters in matrix $\mathtt{T}_i$ columnwise. Each regression matrix in SLLiP is trained using the least squares method $\mathtt{H}_i = \mathtt{T}_i\mathtt{D}_i^T\left(\mathtt{D}_i\mathtt{D}_i^T\right)^{-1}$. The initial learning phase takes 5 or 6 seconds on a standard PC.

More images (around 400) are selected for incremental learning from all images gathered during the last scanning iteration. From each of the additional exemplars, 10 training examples are generated. This procedure provides additional 4000 training examples after each particular video scan. It is worth to note that this process is completely automatic, no user interaction is required. Incremental learning comprises update of regression matrices $\mathtt{H}_i, i = 1, \ldots, k$. An efficient way of updating regression matrices was proposed by Hinterstoisser et al. in [41]. Each regression matrix $\mathtt{H}_i$ may be computed alternatively

$$\mathtt{H}_i = \mathtt{Y}_i\mathtt{Z}_i, \tag{8.5}$$

where $\mathtt{Y}_i = \mathtt{T}_i\mathtt{D}_i^T$ and $\mathtt{Z}_i = \left(\mathtt{D}_i\mathtt{D}_i^T\right)^{-1}$. Let denote $\mathtt{Y}_i^j, \mathtt{Z}_i^j$, where j indexes the training examples. New training example $\mathbf{d} = I(X)$ with parameters $\mathbf{t}$ is incorporated into the predictor as follows

$$\mathtt{Y}_i^{j+1} = \mathtt{Y}_i^j + \mathbf{t}\mathbf{d}^T \tag{8.6}$$

$$\mathtt{Z}_i^{j+1} = \mathtt{Z}_i^j - \frac{\mathtt{Z}_i^j\mathbf{d}\mathbf{d}^T\mathtt{Z}_i^j}{1 + \mathbf{d}^T\mathtt{Z}_i^j\mathbf{d}}. \tag{8.7}$$

After updating matrices $\mathtt{Y}_i$ and $\mathtt{Z}_i$, we may also update the regression matrices $\mathtt{H}_i$ using equation 8.5.

8.2.2 Validation by voting

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

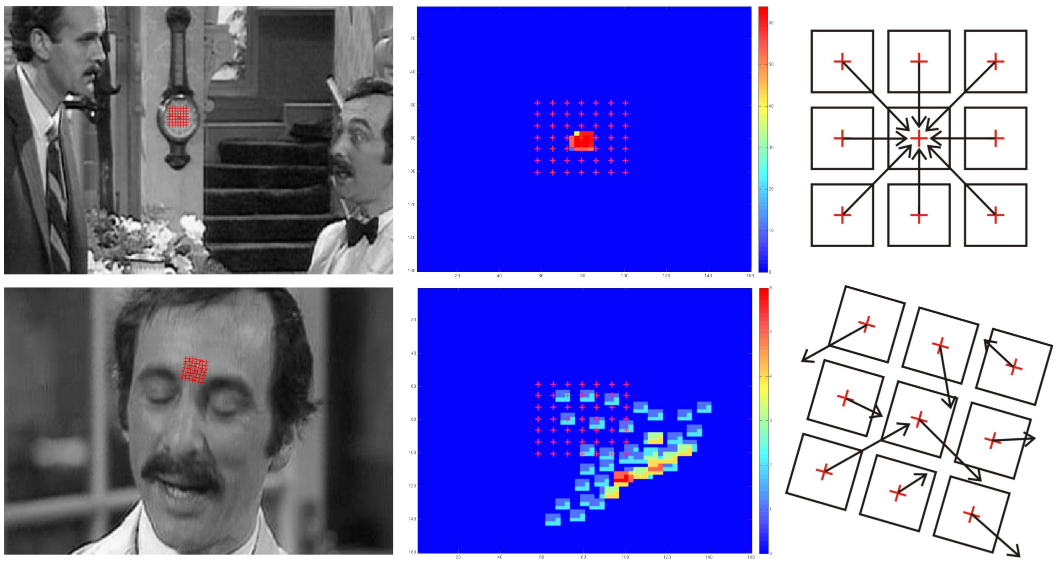


Figure 8.3: The example of predictor validation. The first row shows successful validation of the clock tracking. The second row shows the loss-of-track caused by a sudden scene change just after a video cut. Red crosses depict pixels, where the predictor was initialized - a validation grid. The right column of pictures illustrates the idea of validation using linear predictors and the middle column shows the collected votes for the center of the object in a normalized space.

We initialize the predictor several times on a regular grid (a validation grid - depicted by red crosses in Figure 8.3) in the close neighborhood of current position of the tracker. The close neighborhood is defined as a 2D motion range, for which the predictor was trained. In our case, the range is $\pm(patch\_width/4)$ and $\pm(patch\_height/4)$. The validation grid is deformed according to estimated parameters. Then we observe the 2D vectors, which should point to the center of the object, i.e. the current tracker position in the image. When all (or sufficient number of) the vectors point to the same pixel, which is also the current tracker position, we consider the tracker to be on its track. Otherwise, when the 2D vectors are pointing to some random directions, we say that the track is lost, see Figure 8.3.

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

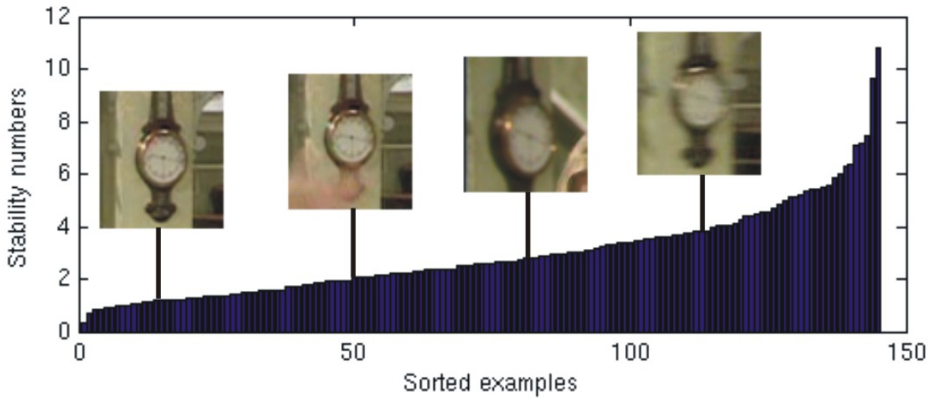


Figure 8.4: Blue bars depict sorted stability numbers. The left most clock image was used for predictor training. The other occurrences obtained during tracking were automatically evaluated as more difficult examples for the tracker. Clearly, the the higher stability measure, the more difficult case for the predictor.

additional training examples are also used for threshold update.

8.2.3 Stability measure and examples selection for the incremental learning

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

When we let the predictor track object on a single frame, we would expect the tracker to stay still in objects' position with no additional change of parameters. However, due to inherent noise in the data the predictor predicts non-zero parameters even when initiated on the correct position. The parameters changes are accumulated and their sum-of-squares is computed after 10 tracking steps. Let $\mathbf{t}$ be the vector of parameters estimated during tracking and $\mathbf{p}_i$ vector of parameters obtained in $i-$th step of this single frame tracking. The *stability number* s for current frame is computed as $s = \sum_{i=1}^{10} \parallel \mathbf{t} - \mathbf{p}_i \parallel^2$. Clearly, the higher value the more difficult example, see Figure 8.4. Parameters changes in both vectors are made relative to particular ranges, in order to obtain the stability number, which is not dependent on different parameters units. Using this stability number we may evaluate how useful (difficult) is the examined object occurrence.

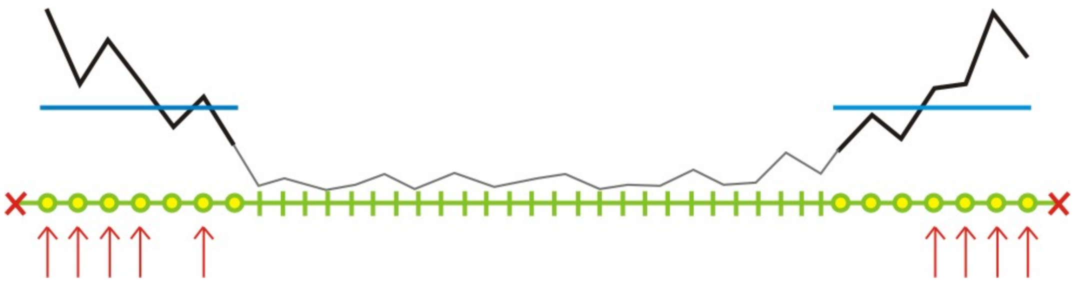


Figure 8.5: Illustration of examples selection for incremental learning. The green line depicts one interval - a subsequence of video frames, where the object was found during scanning procedure. Only few images near the beginning and end of the interval are examined. Yellow circles mean successful validation. The black curve depicts computed stability measure on particular frames. The examples with stability number above the blue line are considered as useful for incremental learning. Selected examples are marked by red arrows.

We use this stability number to select a fixed number of additional training examples from each interval obtained during one video scan. Each interval is a continuous subsequence of images from the whole video sequence (one interval is depicted as a green line in Figure 8.2).

We search for the best additional training examples near the borders of each interval. We go through fixed number of images from the start of the interval forwards and backwards from the end of the interval, while computing the stability number on tracker positions. Finally, the algorithm selects the examples with the high stability number for incremental learning. Tracker positions in these images have also passed validation and we expect them to be well aligned to the object. The procedure of examples selection is depicted in Figure 8.5.

8.3 Experiments

Real sequences used in experiments include an episode from Fawlty Towers series (33 minutes, 720×576 pixels), and Groundhog Day movie (1 hour 37 minutes, 640×384). Several objects were tested, see Figure 8.6. The ground truth data for the Groundhog Day were kindly provided by Josef Šivic and they are the same as in [81]. We have manually labeled ground truth for two tested objects in Fawlty Towers. The third tested sequence captures a human moving in front of a camera (2 minutes 50 seconds, 640×480 pixels), see Figure 8.7. Matlab implementation of the algorithm was used for all experiments. SIFT and SURF object detectors are publicly available MEX implementations [4,55]. Mostly, the standard *precision* and *recall* are used to evaluate the results. Let *TP* denote the *true positives*, *FP* denote the *false positives* and *FN*

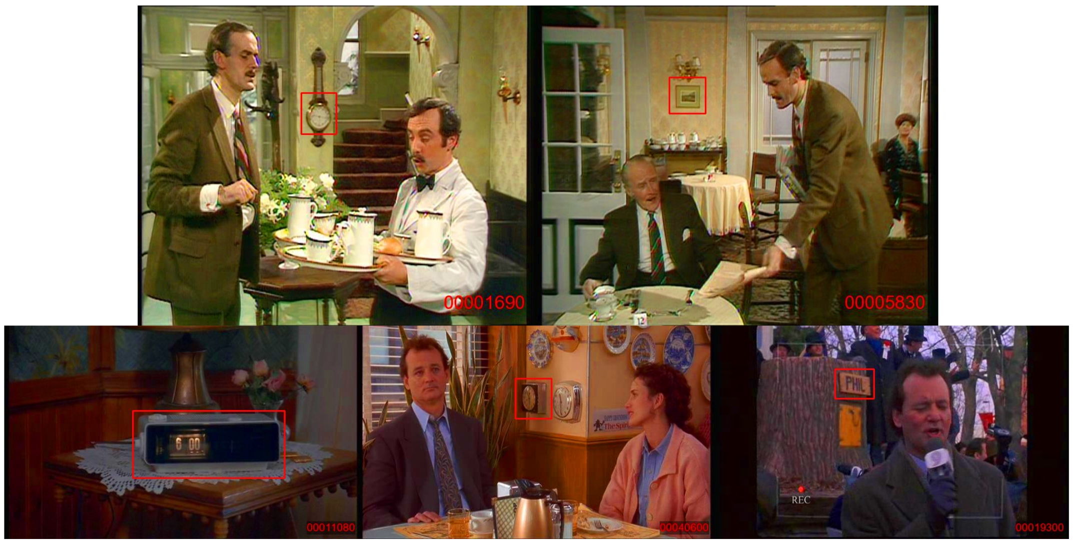


Figure 8.6: Tested objects are marked with the red rectangle.

denote the *false negatives*. Then the precision and recall are computed as:

$$precision = \frac{TP}{TP + FP}, \quad (8.8)$$

$$recall = \frac{TP}{TP + FN}. \quad (8.9)$$

The experiments are organized as follows. The first experiment (Section 8.3.1) shows the effect of the incremental learning on the resulting precision and recall. In Section 8.3.2, we evaluate the overall performance of the algorithm. Next we compare tracking validation by SIFT and by SLLiP. Finally, Table 8.3, Table 8.4, and Table 8.5, show comparison of SIFT detection in every frame with one iteration of our algorithm.

8.3.1 Incremental learning evaluation

This experiment shows the improvement gained by the automatic incremental learning. At first, we run one iteration of video scan using sequential predictor trained on one image only (in Table 8.1 denoted as **iter__0**. Next, we evaluate results after the first and the second incremental learning iteration (**iter__1** and **iter__2**). Two objects were tested. First was the picture object in Fawlty Towers video (see Figure 8.6 top right image). The SURF based detector was used for the picture detection with the step $n = 20$ and the sequential predictor for the validation with step $m = 10$. Incremental learning improves the recall while preserving a high precision, see Table 8.1.

The second tested object was a human face (see Figure 8.7). In this case, the object was difficult to track with SLLiP learned only from one image, because the appearance of the face changed significantly during the sequence. The lighting conditions were challenging and the human face underwent various rotations and scale

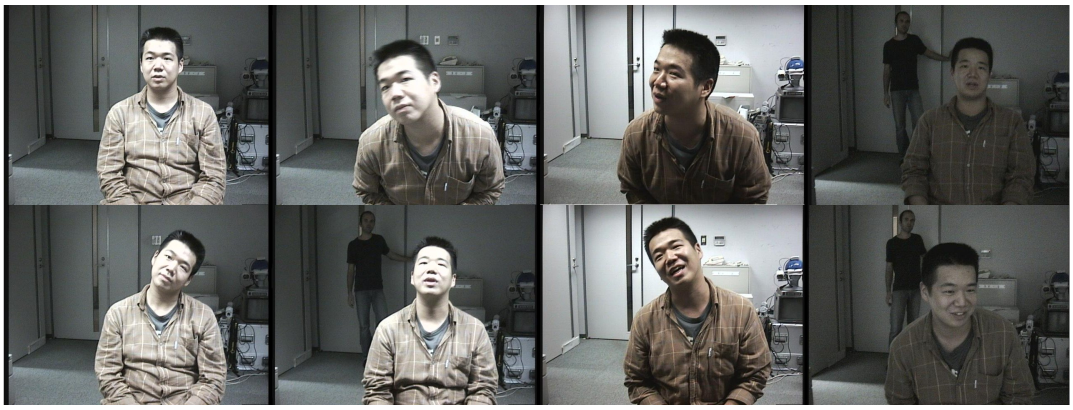


Figure 8.7: Few examples of the human face data used in experiment. All images are extracted from one video sequence. Note the significant deformations and variations in illumination.

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

Table 8.1: Incremental learning evaluation for the clock object from the Fawlty towers episode. The video scan was running 76 frames per second in average.

changes. We have chosen this sequence in combination with the face detector (instead of SIFT/SURF) to see how the incremental learning helps to improve tracking results on a complex non-rigid object. In this case incremental learning also improved the performance of the tracker. See Table 8.2 for results.

The high precision obtained in the face experiment was caused by flawless face detection, which did not return any false positive. A few images of SLLiP tracker aligned on human face can be seen on Figure 8.8.

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

Table 8.2: Incremental learning evaluation for the human face. The first iteration of video scan was running 21 frames per second in average. The browsing time was increased using the face detector instead of SURF.

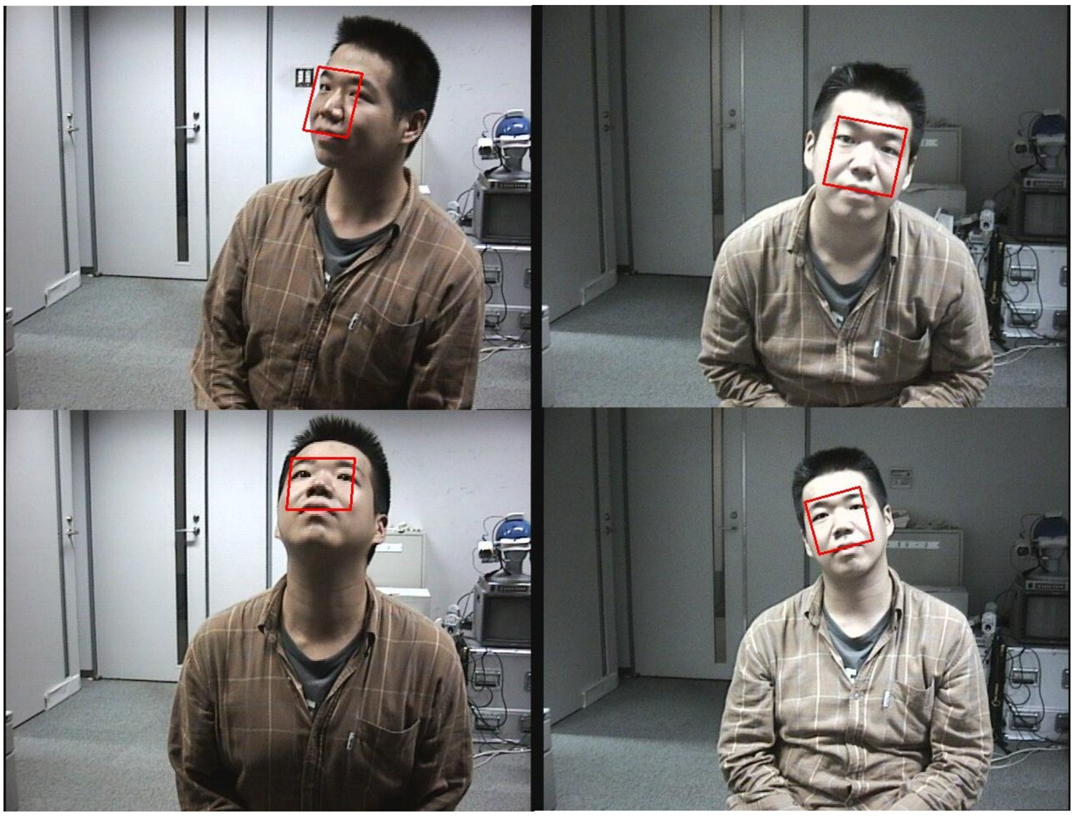


Figure 8.8: Examples of face tracking results. The red rectangle depicts SLLiP tracker aligned on human face.

8.3.2 Results of detection and tracking

One iteration of the algorithm in Fawlty Towers series runs 3 times faster than real-time and more than 8times faster for the Groundhog Day movie. The detector was run every $n = 20$ frames and validation every $m = 10$ frames while tracking. In the sequence with human face the browsing time was almost twice slower than the real-time, even for detection step $n = 40$. It was caused by the face detector which runs much slower than SURF. The difference in browsing times in Fawlty Towers and Groundhog Day is caused mainly by the different video resolution. Processing of higher resolution images and more complex scenes is slowed down by the object detector. Even shorter browsing times may be achieved by increasing the detection interval n. Selecting the right interval depends on our expectation of the shortest time interval, where the object may appear. Reasonable values for detection interval are between 20 and 60 frames. Increasing the validation interval m to more than 10 generates more false positives and, since the validation runs very fast, it is not necessary to validate with a bigger step.

Next we compare the performance of the predictor validation with SIFT validation. The average time of one SIFT validation was 179 milliseconds and the average time of

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

Table 8.3: Comparison of SIFT object detection only and one iteration of our algorithm on Fawlty Towers—clock and picture.

one predictor validation was 33 milliseconds. The resulting recall of the video browsing for the clock object was 0.58 with SIFT and 0.61 with the predictor validation, while the precision was 0.9 for SIFT validation and 0.86 for SLLiP validation. The recall for the picture object was 0.94 with SIFT validation and 0.95 with the predictor, while the precision was 0.84 for both. The predictor validation gives comparable precision and the recall in much shorter time, which also saves time in the whole scanning iteration. Tables 8.3, 8.4 and 8.5 show the results for 5 tested objects obtained in one scanning iteration. The results of the video browsing algorithm are compared to the results produced by the SIFT detector only.

SURF detection on every frame was tested too, but the results contained a large number of false positives. The recall was comparable to SIFT detector, but the precision was very low. We are using the SURF detector because it runs much faster than SIFT, but we need to use the predictor validation after every positive detection because of the large number of false positives. The results show that the algorithm gives results comparable with SIFT detection even after a single iteration only.

8.4 Conclusion

We have shown that the incremental learning of the sequential predictor significantly improves its robustness. It increases the recall while keeping the high precision. The proposed method for collecting additional training examples is completely automatic and requires no user interaction. The stability number well describes the condition of the tracker on a particular image and proves to be a good criterion for the training

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

Table 8.4: Comparison of SIFT object detection only and one iteration of our algorithm on Groundhog Day - alarm clock and clock.

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

Table 8.5: Comparison of SIFT object detection only and one iteration of our algorithm on Groundhog Day - PHIL sign.

examples selection. Validation by clustering SLLiP responses works reliably and very fast.

When coupled with a sparsely applied object detector, the system can seek objects through videos several times faster than real time despite the current rather preliminary Matlab implementation. The complete system for video browsing works very well with simple objects. Performance for more complex 3D objects (when using SIFT/SURF for detection) have not been entirely satisfactory yet. It is mainly the detector that hinders the recognition rate. The tracker itself may be incrementally learned for new appearances of the object and it works better with every iteration. This was verified on the face tracking where a robust face detector was applied. We plan to extend the sequential predictor in a way that would allow its application as a detector.

9 Application – object localization in high-resolution image sequences

This chapter advances further the ideas proposed in the previous chapter. It introduces the full homography motion model which was dictated by larger objects. It also improves the detection phase by applying a fern based detector. Work has been published as [45].

9.1 Introduction

This chapter focuses on the problem of the real-time object detection and tracking in a sequence of high-resolution omnidirectional images. The idea of combining a detector and a fast alignment by a tracker has been already used in several approaches [41,54]. The frame rate of commonly used detectors naturally depends on both the scene complexity and image resolution. For example, the speed of ferns [67], SURF [4] and SIFT [55] detectors depends on the number of evaluated features, which is generally proportional to the scene complexity (e.g., the number of Harris corners) and the image resolution. The speed of Waldboost [83] (or any cascade detector) depends on

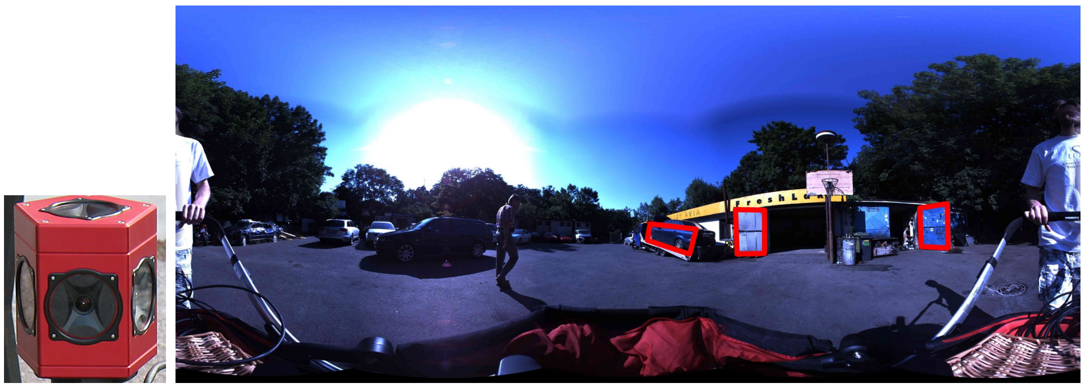


Figure 9.1: Omnidirectional high resolution image (12 Mpx) captured by Ladybug 3 camera (left). Three objects are marked.

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

The ferns-based detector (also used by [41] for 10 fps tracking-by-detection) is one of the fastest object detectors because of the low number of evaluated binary features on detected Harris corners. The speed makes the ferns detector ideal for the purpose of object detection in large images.

Recently, it has been shown [67], [41], that taking advantage of the learning phase, greatly improves the tracking speed and makes the tracker more robust with respect to large perspective deformations. A learned tracker is able to run with a fragment of the processing power and it estimates the object position in complicated or not yet seen poses. However, once the tracker gets lost, it may not recover the object position.

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

We use a similar approach to [41], which also uses the fern object detector and the linear predictor with incremental learning for homography estimation. The detector is used for the object localization and also for the rough estimation of patch transformation. The initial transformation is further refined by the linear predictor, which predicts the full 2D homography. The precision of the method is validated by the inverse warping of the object patch and by the correlation-based verification with the initial patch. The detector is run in every frame of the sequence of 0.3 Mpx images processing 10 frames per second (fps). This approach however, would not be able to perform in real-time on 12 Mpx images. We use the fern detector to determine tentative correspondences and we run RANSAC on detected points to estimate the affine transformation. After a positive detection, we apply the learned predictor in order to track the object for as many frames as possible. The approach [41] uses the iterative version of linear predictor similar to the one proposed by [47], while we use SLLiP version. The SLLiP proved [101] to be faster than the iterative version, while keeping the high precision of the estimation. Our tracker is incrementally updated during tracking [41,54]. We validate the tracking by the updated tracker itself (see Section 9.2.2), which is more precise, than correlation-based verification by a single template in case of varying object appearance.

Recently, [42] used adaptive linear predictors for real-time tracking. Adaptation is performed by growth or reduction of the tracked patch during tracking and by the regression matrices. However, this approach is not suitable for our task, because it needs to keep the large matrix with training examples in the update memory, which is needed for computation of the template reduction and growth. This training matrix grows with additional training examples collected for on-line learning, which is undesirable for long-term tracking.

Li et al. [54] use linear predictors in the form of locally weighted projection regressors (LWPR) as a part of self-tuning particle filtering (ISPF) framework. They approximate a non-linear regression by a piece-wise linear models. In contrast we use a sequence of learnable linear predictors (SLLiP) similar to [107], which uses the result of previous predictors in sequence as a starting point for another predictor in a row. The partial least-squares is used for data dimension reduction in [54]. We use a subset of template pixels spread over the object in a regular grid, which proved to be sufficient for dimensionality reduction, while keeping the high precision and robustness of tracking.

The rest of this chapter is organized as follows. In Section 9.2 the formal descriptions of used ferns detector and sequential predictor tracker and the outline of our algorithm are described. In Section 9.3 we present the general evaluation of our algorithm. A detailed evaluation of the detector and the tracker are given in Sections 9.3.1 and 9.3.2. In the last two sections we discuss, the computational times of the algorithm.

9.2 Detector and 2D homography predictor

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

9.2.1 Ferns-based detector

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

A detailed description of the similarity measure is in [67]. In the following, we provide just short description for the sake of completeness. The similarity measures probability $p(\mathbf{V}(\mathbf{v}), \mathbf{w})$ that the observed appearance of the neighbourhood $\mathbf{V}(\mathbf{v})$ of the detected corner $\mathbf{v}$ corresponds to the model corner $\mathbf{w}$. The appearance is represented as a sequence of randomly selected binary tests, i.e. given the corner $\mathbf{v}$ and the sequence of n point pairs $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots (\mathbf{x}_n, \mathbf{y}_n)\}$. The appearance of the $\mathbf{v}$ is encoded as the binary code $V_k(\mathbf{v}) = I(\mathbf{v} + \mathbf{x}_k) > I(\mathbf{v} + \mathbf{y}_k)$, where $I(\mathbf{v} + \mathbf{x}_k)$ is the image intensity.

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

$$p(\mathbf{V}(\mathbf{v}), \mathbf{w}) = \prod_{k=1}^{N} p_k(V_{I_k(1)}(\mathbf{v}), \dots, V_{I_k(h_k)}(\mathbf{v}), \mathbf{w})\,. \tag{9.1}$$

The proposed detector requires an off-line training phase, within which the subsequent probabilities are estimated. Once the probabilities are pre-computed, we use them to determine the tentative correspondences on-line. In the following, both phases are detailed.

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

Online running phase: Given an input image, Harris corners are detected. For each corner $\mathbf{v}$, binary tests are evaluated and the similarity to each model corner is computed using Equation 9.1. Similarities higher than a chosen threshold determine tentative correspondences. Eventually, RANSAC estimates affine transformation between the model and the given image. Confidence of the detection is equal to the number of inliers.

9.2.2 Sequential linear predictors for homography

We extend the anytime learning of the Sequential Learnable Linear Predictors (SLLiP) by [107] in order to predict not only translation but also the full homography transformation. The next extension is the incremental learning of new object appearances, also used by [41]. The predictor essentially estimates deformation parameters from image intensities directly. It requires a short offline learning stage before the tracking starts. The learning stage consists of generating exemplars and estimation of regression functions. We use a simple cascade of 2 SLLiPs - the first for 2D motion estimation (2 parameters) and the second for the homography estimation (8 parameters). The homography is parametrized by the position of 4 patch corners. Knowing the corners position and having the reference coordinates, we compute the homography transformation for the whole patch. We have experimentally verified that this 2-SLLiP configuration is more stable than using just one SLLiP to predict homography. The translation is roughly estimated by the first SLLiP and then a precise homography refinement is done. Because of speed, we opted for least squares learning of SLLiPs similarly, as suggested in [107].

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

$$\mathbf{p} = \mathtt{A}\left(\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} + \begin{bmatrix}\mathbf{t}^t\\0\end{bmatrix}\right), \tag{9.2}$$

$$\mathbf{x}' = [p_x/p_z, p_y/p_z]^T, \tag{9.3}$$

where $\mathbf{p}$ are homogeneous coordinates. The homography matrix $\mathtt{A}$ is computed from 4-point correspondences, between shifted object corners $\mathbf{c}_i + \mathbf{t}^t$ from previous image and current corners positions $\mathbf{c}_i + \mathbf{t}^t + [\mathbf{t}^a_{2i-1}, \mathbf{t}^a_{2i}]^T, i = 1, \ldots, 4$ estimated by the 2-SLLiP tracker. The parameters $\mathbf{t}^t$ and $\mathbf{t}^a$ are estimated by the same way as in the previous chapter for the affine model.

In our algorithm, the 2 SLLiPs are learned from one image only and they are incrementally learned during tracking. A few thousands of training examples are artificially generated from the training image using random perturbations of parameters in vector $\mathbf{t}$, warping the support set accordingly and collecting the image intensities. The column vectors of collected image intensities $I(X)$ are stored in matrix $\mathtt{D}_i$ and perturbed parameters in matrix $\mathtt{T}_i$ columnwise. Each regression matrix in SLLiP is trained using the least squares method $\mathtt{H}_i = \mathtt{T}_i\mathtt{D}_i^T\left(\mathtt{D}_i\mathtt{D}_i^T\right)^{-1}$.

The tracking procedure needs to be validated in order to detect the loss-of-track. When the loss-of-track occurs, the object detector is started instead of tracking. To *validate* the tracking we use the first SLLiP, which estimates 2D motion of the object. We utilize the fact that the predictor is trained to point to the center of learned object when initialized in a close neighborhood. On the contrary, when initialized on the background, the estimation of 2D motion is expected to be random. We initialize the predictor several times on a regular grid (validation grid - depicted by red crosses in Fig. 9.2) in the close neighborhood of the tracker current position. The close neighborhood is defined as a 2D motion range (of the same size as the maximal parameters perturbation used for learning), for which the predictor was trained. In our case, the range is $\pm(patch\_width/4)$ and $\pm(patch\_height/4)$. We let the SLLiP vote for the object center from each position of the validation grid and observe the 2D vectors, which should point to the center of the object, in the case, when the tracker is well aligned on the object. When all (or sufficient number of) the vectors point to the same pixel, which is also the current tracker position, we consider the tracker to be on its track. Otherwise, when the vectors point to some random directions, we say that the track is lost, see Fig. 9.2. The same approach for tracking validation was suggested in [43].

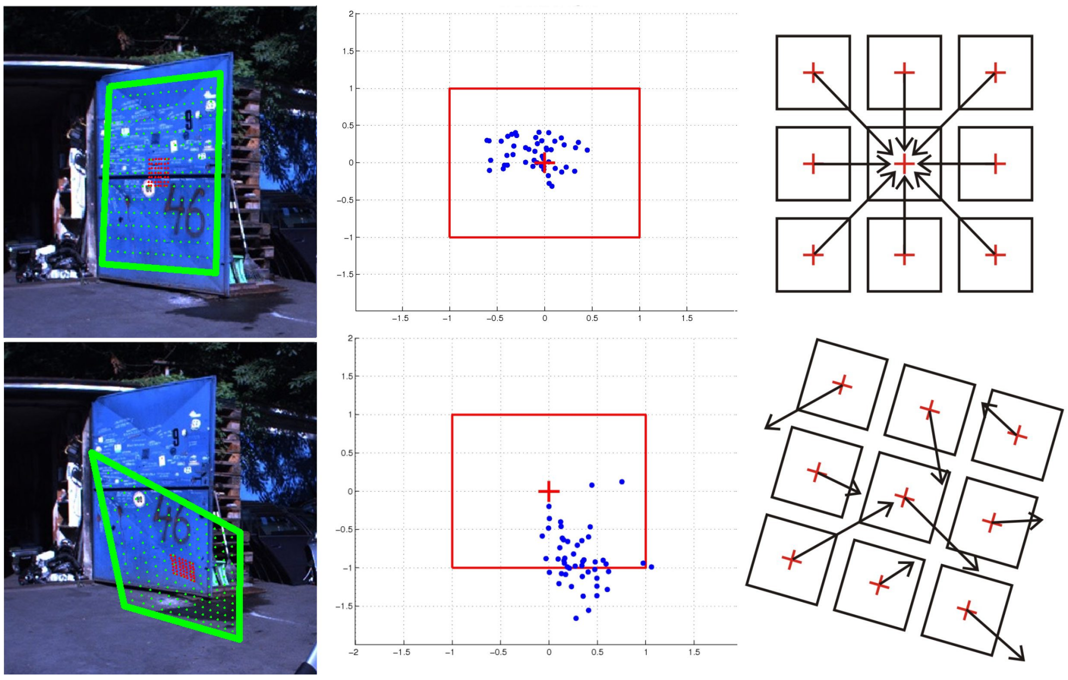


Figure 9.2: Validation procedure demonstrated in two situations. The first row shows successful validation of tracked *blue_door*, the second row shows loss of track caused by a bad tracker initialization. The first column shows the tracker position marked by green. The third column depicts the idea of validation, i.e., a few initializations of the tracker (marked by red crosses) around its current position and the collection of votes for the object center. When the votes point to one pixel, which is also the current tracker position (or close enough to the center), the tracker is considered to be well aligned on the object. When the votes for center are random and far from current position, the loss-of-track is detected. In the second column, we see the collected votes (blue dots), the object center (red cross) and the motion range (red rectangle) normalized to $< -1, 1 >$, for which was the SLLiP trained.

Algorithm 2 In order to achieve real-time performance, we need to run the detector only when absolutely necessary. The detection runs when the object is not present in the image or the tracker loses its track. As soon as the object is detected, the algorithm starts tracking and follows the target as long as possible. Since tracking requires only fragment of computational power, computational time is spared for other tasks. The on-line incremental update of the regressors helps to keep longer tracks. When the validator decides that the track is lost, the detector is started again until next positive detection is achieved. To lower the number of false detections to minimum, we run the validation after each positive response of the detector.

```
Select object
$model\_fern \leftarrow$ learn fern detector
$model\_tracker \leftarrow$ learn $2-$SLLiP tracker
$lost \leftarrow$ true
$i \leftarrow 0$
while next image is available do
  get next image
  $i \leftarrow i + 1$
  if $lost$ then
    $detected \leftarrow$ detect object
    if $detected$ then
      initialize tracker
      estimate homography
      $valid \leftarrow$ validate position
      if $valid$ then
        $lost \leftarrow$ false
        continue
      end if
    end if
  else
    track object
    if $i$ mod $5 == 0$ then
      $valid \leftarrow$ validate position
      if $valid$ then
        $model\_tracker \leftarrow$ update tracker
      else
        $lost \leftarrow true$
        continue
      end if
    end if
  end if
end while
```

9.3 Experiments

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

Three types of experiments were performed. First, we run the ferns detector itself in every frame without tracking. Second, we run the 2-SLLiP tracker with the validation without the recovery by detector. And finally, we run the combination of both. In all experiments both the detector and the tracker were trained from a single image. The detector and the tracker perform best on planar objects, because of the modeled 2D homography transformation. We tested our algorithm also on non-planar objects (*lying human, crashed car*) to see the performance limits and robustness of our solution, see Section 9.3.3. Algorithm was tested on 8 objects in 4 video sequences. The Ladybug camera provides 8 fps of panoramic images captured from 6 cameras simultaneously. Five cameras are set horizontally in a circle and the sixth camera looks upwards, see Fig. 9.1. The panoramic image is a composition of these 6 images and has a resolution of 5400×2248 pixels (12 Mpx). Fig. 9.1 and Fig. 9.3 show examples of the composed scenes and tested objects. Appearance changes for few selected objects are depicted in Fig. 9.4. Notice the amount of non-linear distorsion caused by the cylindrical projection. The objects of interest are relatively small in comparison to the image size. In average the object size was 400×300 pixels. The ground-truth homography for each object was manually labeled in each frame. We provide ROC curves for each tested object for evaluation of the detection/tracking performance. The ROC curve illustrates *false positive rate* versus *false negative rate.*

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

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

$$\mathrm{FP}\,(c) = \sum_{j=1}^{n} \left(\mathrm{FP}, \text{where } r_j > c\right)/n \tag{9.4}$$

$$\mathrm{FN}\,(c) = \sum_{j=1}^{n} \left(\mathrm{FN}, \text{where } r_j > c\right)/n, \tag{9.5}$$

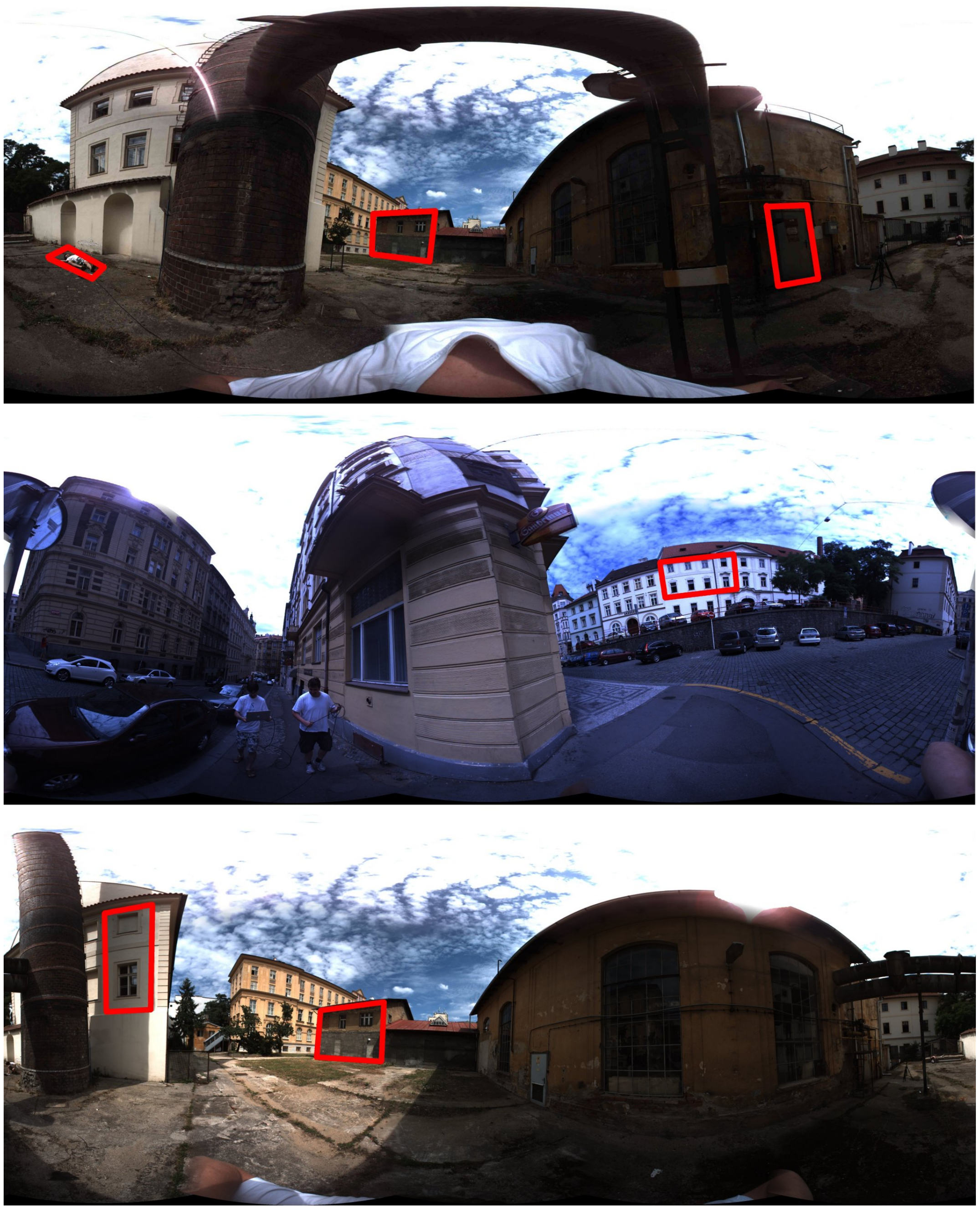


Figure 9.3: Example images with tracked objects marked by red rectangles.

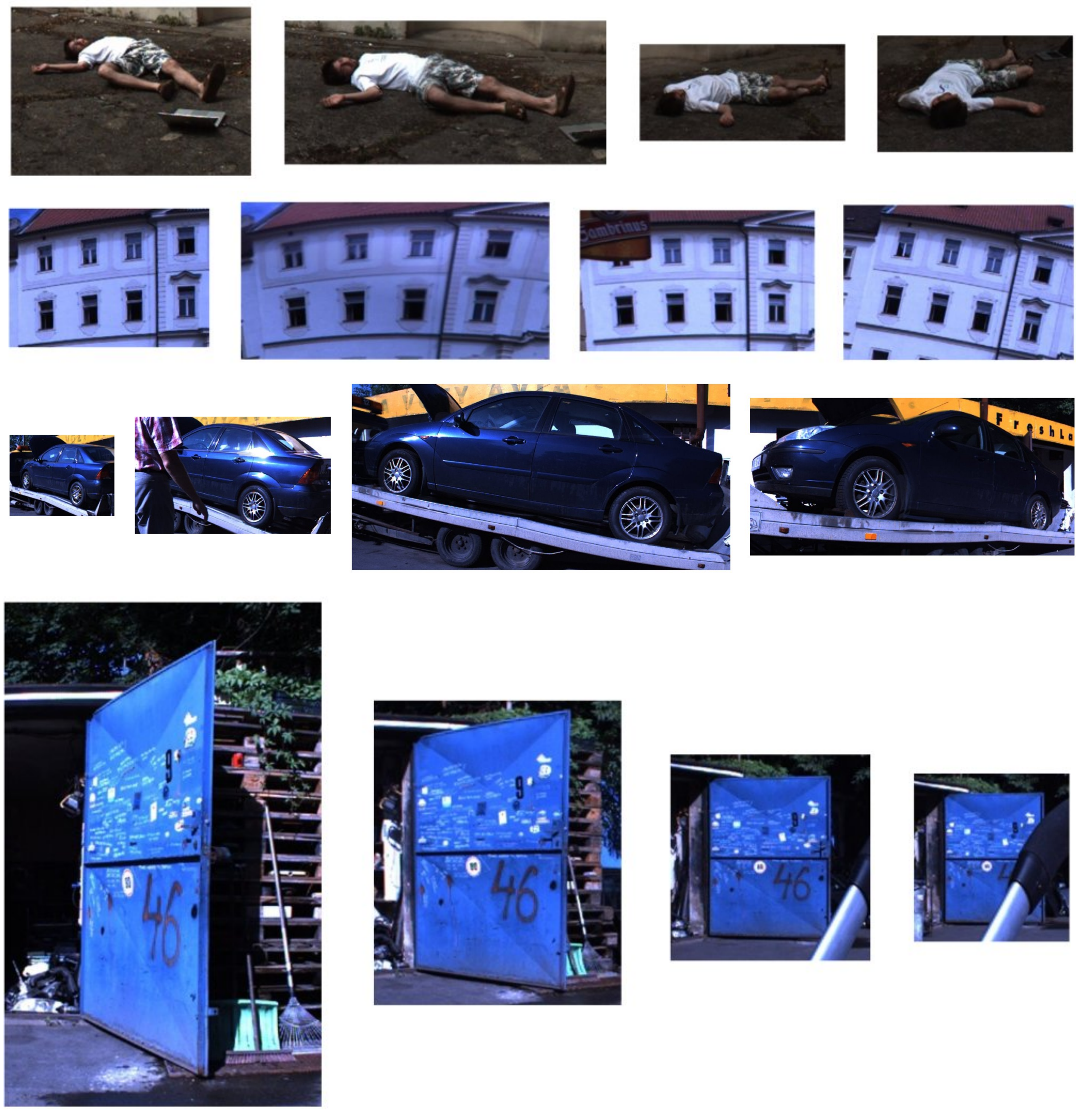


Figure 9.4: Four of eight tested objects under different view angles and appearances.

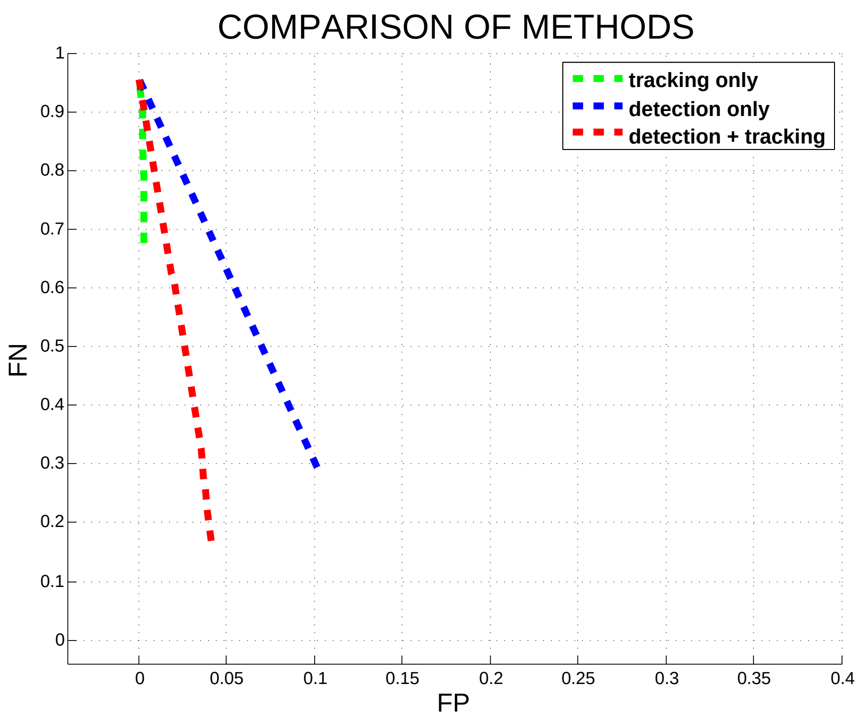


Figure 9.5: Each curve corresponds to results of one method computed as mean ROC curve over all objects.

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

In Fig. 9.5, we show three different ROC curves. Each curve corresponds to one method used to search for the object position in sequences. In order to make the evaluation less dependent on a particular object, we computed mean ROC curves over all tested objects for different methods. The green curve depicts the performance of the tracker itself, run on every object from the first frame until the loss-of-track without the recovery by the detector. The blue curve shows results obtained by the fern detector itself run on every frame of all sequences. And finally, the red curve shows results, when our algorithm was used. We may observe, that our algorithm performance is better (the curve is the closest to point $(0, 0)$) than both individual methods. The separate ROC curves for individual objects may be seen in Fig. 9.6 and Fig. 9.8.

The experiments are organised as follows. The ferns detector is evaluated in Section 9.3.1, the performance of tracker is examined in Section 9.3.2. The algorithm 1, which combines both is evaluated in Section 9.3.3. And finally in Section 9.3.4, we provide computation times of all main parts of the algorithm.

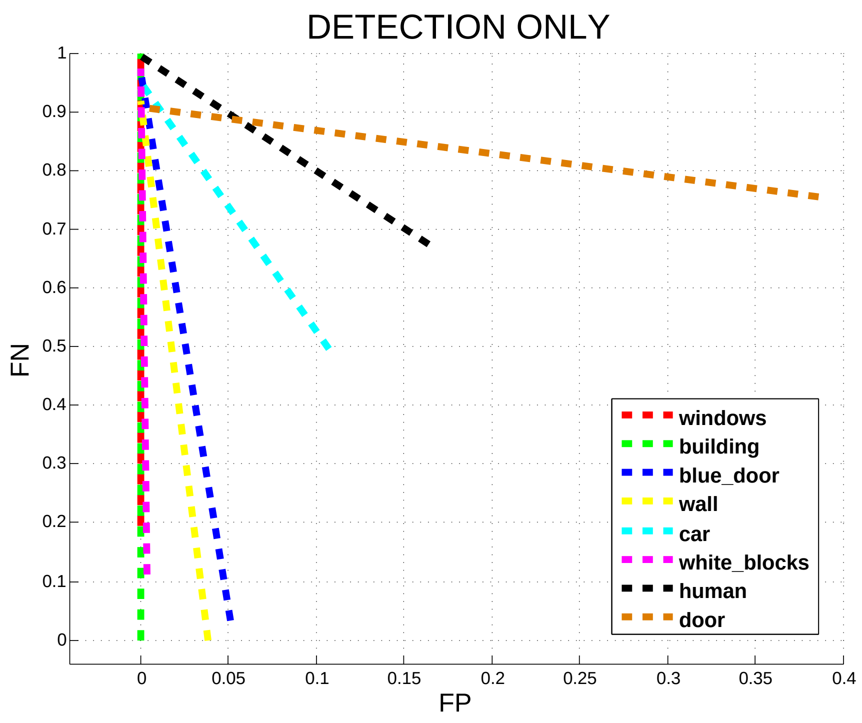


Figure 9.6: Each curve corresponds to detection results for one object.

9.3.1 Detector evaluation

Using only the detector (tracking by detection) would be too slow for desired real-time performance in a sequence of large images. Nevertheless, we evaluate the performance of the detector itself to see how the addition of SLLiP tracker lowers the false positive and the false negative rate (see Section 9.3.3).

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

It was also necessary to adjust the detector parameters according to the expected object scale and rotation changes. In average, the detector was searching for the object in 3 scales and it was able to detect objects under ± 20 rotation degrees. In Fig. 9.6, the ROC curves are depicted for the detector run in every frame for different objects. The results show that some objects were detected in almost all cases correctly. However some other objects, like the *door*, were detected with poor results. The *Door* was the most complicated object for Harris corners-based detector, since only 21 keypoints were detected over the object, which were spread mainly in the central part of the door. That is why there was almost always a low number of inliers provided by the RANSAC algorithm. This object was successfully tracked by the tracker lately. Another complicated object was the *car*, due to its reflective surface and vast visual angle changes. Finally, the *human* lying on the floor was also a challenging object due to its non-planarity. As can be seen in Section 9.3.3, the integration of tracking to the algorithm lowers the number of FP and FN and significantly speeds up the algorithm, see Section 9.3.4.

9.3.2 Tracker evaluation

This experiment shows performance of the tracker without the recovery by the detector. The tracker is composed of 2 SLLiPs (for translation and homography). Each SLLiP has 3 predictors in a sequence with support the set sizes $|X_1| = 225$, $|X_2| = 324$ and $|X_3| = 441$. The support set coordinates were spread over the object in a regular grid. The tracker was incrementally learned and validated during tracking until it lost its track or until the end of sequence was reached. The tracking was manually initialized always in the first image of a sequence (different from the training image), where the object appeared. Some objects were tracked through the whole sequence. Some objects were lost after few frames, when there was a fast motion right in the beginning. In Fig. 9.7, you may see the lengths of successful tracking until the first loss-of-track. In case of partial occlusion, the tracker sometimes jitters or even fails. Nevertheless, when it is incrementally learned, it is able to handle the occlusion as a new object appearance. Incremental learning itself is very helpful for increasing the robustness of the tracker [41], [43]. The estimation of the homography is very precise for planar objects.

Tracked objects appear in images as patches in the resolutions varying from 211×157 (33 127 pixels) to 253×919 (232 507 pixels). Both SLLiPs work only with the subset of all patch pixels (same subset size for all objects). While tracking, each SLLiP needs to read only 990 intensity values, which is given by the sum of support set sizes of predictors in sequence. This brings another significant speed-up to the learning and tracking process.

9.3.3 Detector + tracker evaluation

The final experiment evaluates the performance of algorithm described in Section 9.2. The combination of the detector and the tracker improves the performance of the algorithm (lowers FP and FN), as may be seen in Fig. 9.8 and Fig. 9.5. This is caused by their complementarity in failure cases. Tracker is very robust even under extreme perspective deformations, while the detector is not able to recognize these difficult object poses. On the other hand the detector is robust to partial occlusion, where the tracker usually fails and needs to be recovered and re-learned. In comparison with the detector (see Fig. 9.6), our algorithm significantly improves the results in average. Only few objects, which were perfectly detected by the detector (e.g. *white blocks* and *blue door*) have a little worse results with our algorithm. This was caused by the tracking validation, which was running not every frame, but only every 5 frames. This means, that the tracker was lost just a few frames before loss-of-track detection by

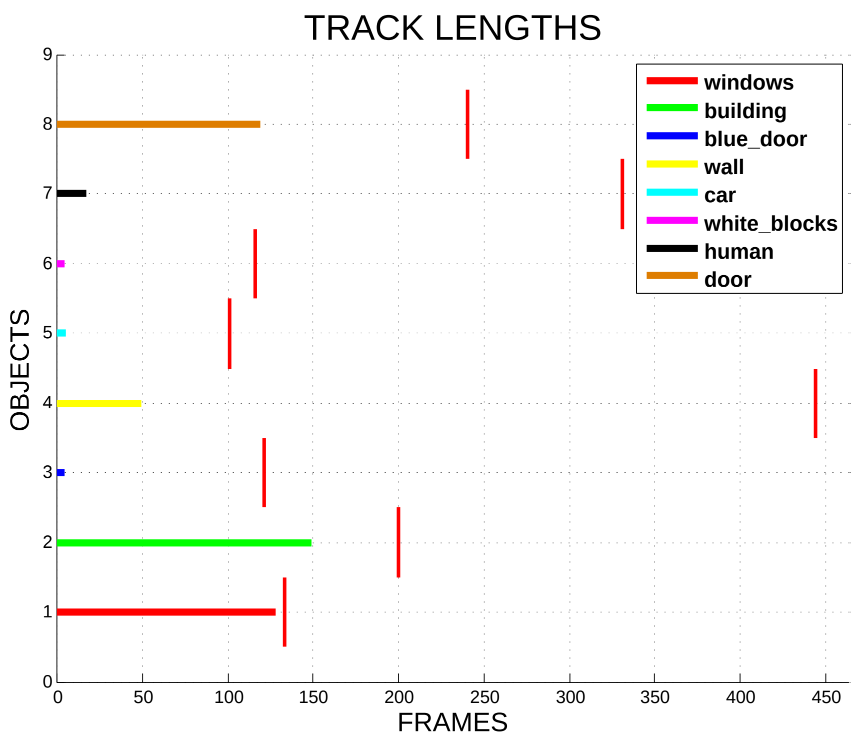


Figure 9.7: Each horizontal line depicts the length of the track for one object until the first loss-of-track. Red vertical lines show the last frame of a particular subsequence, in which the object was fully or partially visible.

validation and received a few FPs and FNs. This small error could be eliminated by running the validation in every frame. The extreme efficiency of sequential predictors allows tracking much faster than real-time, which provides enough computational time for validation and incremental learning of the tracker. Running validation after each positives detection allows to make the ferns detector more sensitive. We lower the threshold which specifies the number of necessary inliers, which allows more true positive, but also more false positive detections. After each detection, which has a small number of inliers, we initialize the tracker in the detected pose, let the tracker vote for homography and run the validation. Validation eliminates possible false positive detections and let the true positives pass.

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

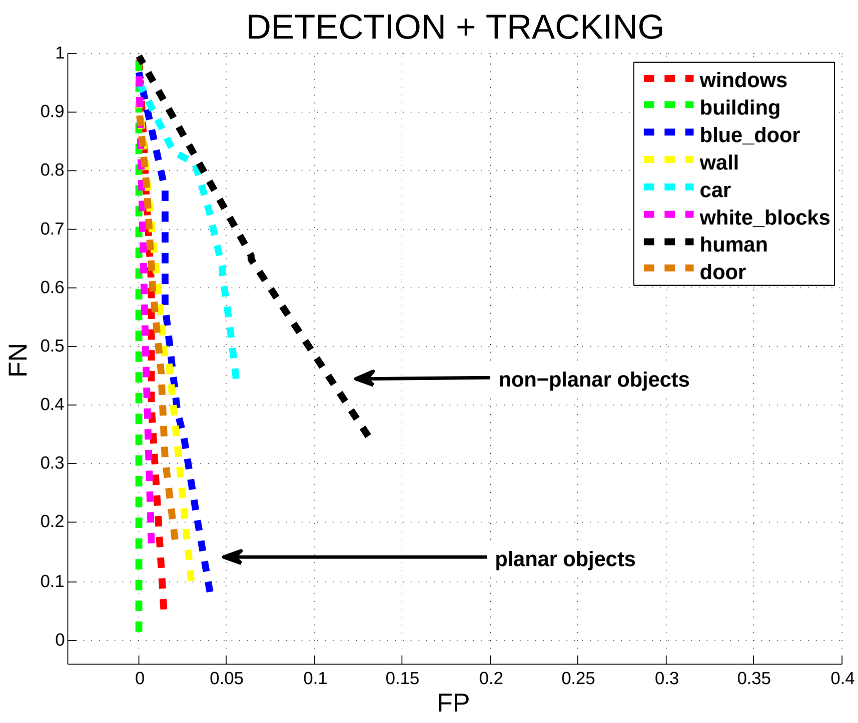


Figure 9.8: Each curve corresponds to results of one object detection and tracking. The ROC curves fit more to the left side of the diagram. This is caused by the high confidence of detections and tracking. The high confidence is actually valid, because of the very low number of false positives.

detected and tracked with the high accuracy.

9.3.4 Computation times

The algorithm was run on standard PC with 64 bit, 2.66 GHz CPU. The object detector was implemented in language C and run in the system as MATLAB MEX. The RANSAC, 2-SLLiP tracker and the rest of the algorithm were implemented in Matlab. The computation times for 12 Mpx image sequences are following:

(implementation in language C)

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

(implementation in Matlab)

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

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

9.4 Conclusion and future work

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

In our future work, we want to focus on incremental learning of both the detector and the tracker. The detector is robust to partial occlusion, since it works with Harris corners [39] sparsely placed around the object unlike the patch-based tracker. On the other hand, the tracker is more robust to object appearance changes and keeps tracking even the significantly distorted objects, which the detector fails to detect. This gives the opportunity to deliver the training examples for the detector in cases where it fails, while the tracker holds and vice-versa. We would like to develop a suitable strategy for mutual incremental learning.

10 Conclusions

10.1 Contributions of the thesis

Chapter 2 describes our work on estimating the articulated human model from segmented multiview sequences. The human body parts are modeled as ellipsoids. Number of parts and their connectivity are the only inputs specified. Physical dimensions and articulation parameters are estimated from data. Our work on the multiview approach though relatively successful has been mostly suspended after the Dagstuhl seminar in 2006[1] where I met several scientists from biomechanics. We achieved reasonable accuracy when using very simple and cheap digital cameras. However, most of biomechanics applications require significantly higher accuracy achievable by using costly high-speed cameras which we did not possess at the time.

Chapter 3 reports our contribution on learning and fitting an articulated model to single images. We proposed a graph-cut based step for unsupervised learning of appearance. The work was actually quite novel at that time. We achieved promising results however, the work has not been published as we got overrun by contributions of V. Ferrari et al. [30].

Chapter 4 proposes a multiview tracking and modeling method for rigid objects. Assuming that a part of the object is visible in at least two cameras, a partial 3D model is reconstructed in terms of a collection of small 3D planar patches of an arbitrary topology. The 3D representation, recovered fully automatically, allows to formulate tracking as a gradient minimization in the pose (translation, rotation) space. As the object moves, the 3D model is incrementally updated. A virtuous circle emerges: tracking enables composition of the partial 3D model; the 3D model facilitates and robustifies the multiview tracking. We demonstrated experimentally that the interleaved track-and-reconstruct approach tracks successfully a 360 degrees turn-around and a wide range of motions. Monocular tracking is also possible after the model is constructed. Using more cameras, however, increases significantly the stability in critical poses and moves. We demonstrated how to exploit the 3D model to increase stability in the presence of uneven and/or changing illumination.

Chapter 5 proposes an algorithm for learning a sequence of linear regressors on artificially distorted variants of sample data. The learning procedure minimizes the worst error on training data. An object is modeled by a set of sequential predictors. The motion of the object is estimated by a RANSAC based procedure. The number of RANSAC iterations and number of used predictors is optimized subject to required frame rate.

Chapter 6 addresses the problem of computationally expensive min-max learning and suggests a branch and bound method based on least square estimation. It is

[1]Human Motion, `http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=06241`

an any time method which, after a very short initialization period, it can provide a solution at any time of learning.

Chapter 7 proposes a learning method for motion estimation of objects with significantly varying appearance. Varying object appearance is represented by a low dimensional space of appearance parameters. The appearance mapping and motion estimation method are optimized simultaneously. Appearance parameters are estimated by unsupervised learning.

Chapter 8 advances the sequential linear predictors–regressors in several ways. It introduces a more complex motion model, suggests a validation procedure and applies an on-line incremental learning in order to accommodate drifting appearance. It also discusses a possible method for automatic selection of suitable training examples for the incremental learning by introducing a stability measure. The advanced predictor is combined with an object detector and applied in fast object-driven video browsing.

The final Chapter 9 introduces a full homography motion model and combined with an object detector applies the combination on finding images in high resolution cameras.

10.2 Future plans

In the future, I would like to continue doing research in machine learning for visual processing. Though being already popularized be recent advancements, I believe the area of weakly supervised learning from large data is still largely unexplored. The machine perception of the unstructured world is still in an early stage.

I plan to be very active in teaching as I have always been. I will strive introducing recent developments in computer science into the teaching process. I consider teaching and research as tightly connected.

References

[1] Ankur Agarwal and Bill Triggs. Recovering 3d human pose from monocular images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):1–15, 2006.

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

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

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision*, volume I of *LNCS*, pages 404–417. Springer, May 2006.

[5] Selim Benhimane, Alexander Ladikos, Vincent Lepetit, and Nassir Navab. Linear and quadratic subsets for template-based tracking. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, 2007.

[6] A. Bissacco, M.H. 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.

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

[8] Béla Bollobás. *Modern Graph Theory.* Springer, New York, 1998.

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

[10] J. Carranza, C. Theobalt, M. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. *ACM Transaction on Computer Graphics*, 22(3), July 2003.

[11] Marco La Cascia, Stan Sclaroff, and Vassilis Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. *IEEE Transactions of Pattern Analysis Machine Intelligence*, 22(4):322–336, 2000.

[12] Dana Cobzas and Martin Jagersand. 3D SSD tracking from uncalibrated video. In *Workshop on Spatial Coherence for Visual Motion Analysis (SCVMA), in conjunction with ECCV 2004*, 2004.

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

[14] Timothy Cootes, Gareth Edwards, and Christopher Taylor. Active appearance models. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 23(6):681–685, June 2001.

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

[16] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition - CVPR*, pages 886–893, 2005.

[17] Patrick de la Hamette, Paul Lukowicz, Gerhard Tröster, and Tomáš Svoboda. Fingermouse: A wearable hand tracking system. In Peter Ljungstrand and Lars Erik Holmquist, editors, *UBICOMP2002 Adjunct Proceedings*, volume 1, pages 15–16, Göteborg, Sweden, September-October 2002. TeknologTryck, Elektroteknologsektionen Chalmers, Göteborg.

[18] J Denavit and R.S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Transactions ASME Journal of Applied Mechanics*, 22:215–221, 1955.

[19] Jonathan Deutscher and Ian Reid. Articulated body motion capture by stochastic search. *International Journal of Computer Vision*, 61(2):185–205, 2005.

[20] Petr Doubek, Indra Geys, Tomáš Svoboda, and Luc Van Gool. Cinematographic rules applied to a camera network. In Peter Sturm, Tomáš Svoboda, and Seth Teller, editors, *Omnivis2004, The fifth Workshop on Omnidirectional Vision, Camera Networks and Non-Classical Cameras*, pages 17–29, Prague, Czech Republic, May 2004. Czech Technical University.

[21] Petr Doubek and Tomáš Svoboda. Reliable 3d reconstruction from a few catadioptric images. In R. Benosman and E.M. Mouaddib, editors, *Proceedings of the IEEE Workshop on Omnidirectional Vision 2002*, pages 71–78, Los Alamitos, CA, June 2002. IEEE Computer Society.

[22] Petr Doubek, Tomáš Svoboda, and Luc Van Gool. Monkeys — a software architecture for ViRoom — low-cost multicamera system. In James L. Crowley, Justus H. Piater, Markus Vincze, and Lucas Paletta, editors, *3rd International Conference on Computer Vision Systems*, number 2626 in LNCS, pages 386–395, Berlin, Germany, April 2003. Springer.

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

[24] Harris Drucker, Chris Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. *Advances in Neural Information Processing Systems*, 9:156–161, 1996.

[25] L. Ellis, J. Matas, and R. Bowden. On-line learning and partitioning of linear displacement predictors for tracking. In *Proceedings of the 19th British Machine Vision Conference*, pages 33–42, September 2008.

[26] Liam Ellis, Nicholas Dowson, Jiří Matas, and Richard Bowden. Linear predictors for fast simultaneous modeling and tracking. In *Proceedings of 11th IEEE International Conference on Computer Vision, workshop on Non-rigid registration and tracking through learning*, October 2007.

[27] Lukáš Fajt. Pictorial structural models, learning and recognition in image sequences. MSc Thesis K333–27/07, CTU–CMP–2007–04, Department of Cybernetics, Faculty of Electrical Engineering Czech Technical University, Prague, Czech Republic, January 2007.

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

[29] V. Ferrari, Marin M., , and A. Zisserman. Pose search: retrieving people using their pose. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2009.

[30] V. Ferrari, M. Marin-Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2008.

[31] Francois Fleuret, Richard Lengagne, and Pascal Fua. Fixed point probability field for complex occlusion handling. In *IEEE International Conference on Computer Vision*, 2005.

[32] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Computer Science and Scientific Computing. Academic Press, San Diego, California, USA, 2nd edition, 1990.

[33] D. Gavrila and L. Davis. 3-D model-based tracking of humans in action: a multi-view approach. In *In Proceedings of Computer Vision and Pattern Recognition*, pages 73–80, 1996.

[34] 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.

[35] Helmut Grabner and Horst 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.

[36] Markus Gross, Stephan Wuermlin, Martin Naef, Edouard Lamboray, Christian Spagno, Kunz Andreas, Esther Koller-Meier, Tomas Svoboda, Luc Van Gool, Silke Lang, Strehlke Kai, Andrew Vande Moere, and Oliver Staadt. Blue-c: A spatially immersive display and 3D video portal for telepresence. *ACM Transactions on Graphics (Siggraph 2003)*, 22(3):819–827, July 2003.

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

[38] Gregory D. Hager and Peter 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.

[39] C. Harris and M. Stephen. A combined corner and edge detection. In M. M. Matthews, editor, *Proceedings of the 4th ALVEY vision conference*, pages 147–151, University of Manchester, England, September 1988. on-line copies available on the web.

[40] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University, Cambridge, 2nd edition, 2003.

[41] Stefan Hinterstoisser, Selim Benhimane, Nassir Navab, Pascal Fua, and Vincent Lepetit. Online learning of patch perspective rectification for efficient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 1–8, 2008.

[42] Stefan Holzer, Slobodan Ilic, and Nassir Navab. Adaptive linear predictors for real-time tracking. In *Conference on Computer Vision and Pattern Recognition (CVPR), 2010 IEEE*, pages 1807–1814, June 2010.

[43] David Hurych and Tomáš Svoboda. Incremental learning and validation of sequential predictors in video browsing application. In José Braz Paul Richard, editor, *VISIGRAPP 2010: International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, volume 1, pages 467–474, Setubal, Portugal, May 2010. Institute for Systems and Technologies of Information, Control and Communication. CD-ROM.

[44] David Hurych, Tomáš Svoboda, Jana Trojanová, and Yadhunandan US. Active shape model and linear predictors for face association refinement. In *The Ninth IEEE International Workshop on Visual Surveillance 2009, In conjunction with the 12th IEEE International Conference on Computer Vision, 2009*, pages 1193–1200, Piscataway, USA, October 2009. IEEE Computer Society. CD-ROM.

[45] David Hurych, Karel Zimmermann, and Tomáš Svoboda. Fast learnable object tracking and detection in high-resolution omnidirectional images. In Leonid Mestetskiy and Jose Braz, editors, *Proceedings of VISAPP 2011 International Conference on Computer Vision Theory and Applications*, pages 521–530, Setúbal, Portugal, March 2011. INSTICC - Institute for Systems and Technologies of Information, Control and Communication, INSTICC-Institute for Systems and Technologies of Information, Control and Communication. The Best Student Paper Award.

[46] A.D. Jepson, D.J. Fleet, and T.F. El-Maraghi. Robust on-line appearance models for visual tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 415–422, 2008.

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

[48] Frédéric Jurie and Michel Dhome. Real time robust template matching. In *British Machine Vision Conference*, pages 123–132, 2002.

[49] Takeo Kanade, P. J. Narayanan, and Peter W. Rander. Virtualized reality: Concepts and early results. In *In Proc. of IEEE workshop on the Representation on Visual Scenes*, 1995.

[50] Jana Kostková and Radim Šára. Stratified dense matching for stereopsis in complex scenes. In Richard Harvey and J. Andrew Bangham, editors, *BMVC 2003: Proceedings of the 14th British Machine Vision Conference*, volume 1, pages 339–348, London, UK, September 2003. British Machine Vision Association.

[51] Michael H. Kutner. *Applied linear statistical models.* The McGraw-Hill Companies, Inc., New York, NY, 5th edition, 2004.

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

[53] Vincent Lepetit and Pascal Fua. Monocular model-based 3D tracking of rigid objects. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.

[54] Min Li, Wei Chen, Kaiqi Huang, and Tieniu Tan. Visual tracking via incremental self-tuning particle filtering on the affine group. In *Conference on Computer Vision and Pattern Recognition (CVPR), 2010 IEEE*, pages 1315–1322, June 2010.

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

[56] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Conference on Artificial Intelligence*, pages 674–679, August 1981.

[57] Marcus Magnor and Christian Theobalt. Model-based analysis of multi-video data. *Proc. IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI-2004),* Lake Tahoe, USA, pages 41–45, March 2004.

[58] Jiří Matas, Karel Zimmermann, Tomáš Svoboda, and Adrian Hilton. Learning efficient linear predictors for motion estimation. In *Proceedings of 5th Indian Conference on Computer Vision, Graphics and Image Processing*, number 4338 in LNCS, pages 445–456, December 2006.

[59] Iain Matthews, Takahiro Ishikawa, and Simon Baker. The template update problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):810–815, 2004.

[60] W. Matusik, C. Buehler, R. Raskar, L. McMillan, and S. Gortler. Image–based visual hulls. In *Proceedings of ACM SIGGRAPH 2000*, 2000.

[61] Ondřej Mazaný. Articulated 3D human model and its animation for testing and learning the algorithms of multi-camera systems. MSc Thesis K333–25/07, CTU–CMP–2007–02, Department of Cybernetics, Faculty of Electrical Engineering Czech Technical University, Prague, Czech Republic, January 2007.

[62] Ivana Mikic, Mohan Trivedi, Edward Hunter, and Pamela Cosman. Human body model acquisition and tracking using voxel data. *International Journal of Computer Vision*, 53(3):199–223, 2003.

[63] Anurag Mittal and Larry S. Davis. M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene using region-based stereo. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *The seventh European Conference on Computer Vision, ECCV2002*, number 2350 in LNCS, pages 18–36. Springer, May 2002.

[64] Enrique Muñoz, José M. Buenaposada, and Luis Baumela. Efficient model-based 3D tracking of defomable objects. In Bill Freeman, Luc Van Gool, and Subhasis Chaudhuri, editors, *10th International Conference on Computer Vision — ICCV*, volume I, pages 877–882. IEEE Computer Society Press, October 2005.

[65] E. Murphy-Chutorian and M.M. Trivedi. Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):607–626, April 2009.

[66] David Nistér. Automatic passive recovery of 3d from images and video. In *Second International Symposium on 3D Data Processing, Visualization and TRansmission (3DPVT04)*, 2004. Invited paper.

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

[68] Mustafa Özuysal, Vincent Lepetit, François Fleuret, and Pascal 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.

[69] Tomáš Petříček and Tomáš Svoboda. Area-weighted surface normals for 3D object recognition. In *21st IEEE International Conference on Pattern Recognition*, 2012.

[70] R. Plankers and P. Fua. Articulated soft objects for multi-view shape and motion capture. *PAMI: Pattern Recgnition and Machine Inteligence*, 25(10), 2003.

[71] Rostislav Prikner. Pictorial structural models for human detection in videos. MSc Thesis CTU–CMP–2008–11, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, May 2008.

[72] Deva Ramanan, D. A. Forsyth, and Andrew Zisserman. Strike a pose: Tracking people by finding stylized poses. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 271–278, Washington, DC, USA, 2005. IEEE Computer Society.

[73] P. L. Rosin. Analysing error of fit functions for ellipses. *Pattern Recognition Letters*, 17(14):1461–1470, 1996.

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

[75] F. Rothganger, Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 3D object modeling and recognition using affine-invariant patches and multi-view spatial constraints. In *International Conference on Computer Vision and Pattern Recognition*, volume 2, pages 272–277, 2003.

[76] Radim Šára and Ruzena Bajcsy. Fish-scales: Representing fuzzy manifolds. In Sharat Chandran and Uday Desai, editors, *Proc. 6th International Conference on Computer Vision*, pages 811–817, New Delhi, India, January 1998. IEEE Computer Society, Narosa Publishing House.

[77] Hao Shao, Tomáš Svoboda, Vittorio Ferrari, Tinne Tuytelaars, and Luc Van Gool. Fast indexing for image retrieval based on local appearance with re-ranking. In *IEEE International Conference on Image Processing*, Los Alamitos, CA, USA, September 2003. IEEE.

[78] Hao Shao, Tomáš Svoboda, Tinne Tuytelaars, and Luc Van Gool. Hpat indexing for fast object/scene recognition based on local appearance. In Erwin M. Bakker, Thomas S. Huang, Michael S. Lew, Nicu Sebe, and Sean Xian Zhou, editors,

International Conference on Image and Video Retrieval, number 2728 in LNCS, pages 71–80, Berlin, Germany, July 2003. Springer.

[79] Hao Shao, Tomáš Svoboda, and Luc Van Gool. ZuBuD — Zürich buildings database for image based recognition. Technical Report 260, Computer Vision Lab, Swiss Federal Institute of Technology, April 2003. http://www.vision.ee.ethz.ch/showroom/zubud.en.html.

[80] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[81] J. Sivic and A. Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):591–606, April 2009.

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

[83] Jan Šochman and Jiří Matas. Waldboost - learning for time constrained sequential detection. In Cordelia Schmid, Stefano Soatto, and Carlo 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.

[84] Chris Stauffer and W.E.L Grimson. Adaptive background mixture models for real-time tracking. In *IEEE Computer Vision and Pattern Recognition*, volume 2, pages 2242–2252, June 1999.

[85] Tomas Svoboda. A software for complete calibration of multicamera systems. In Amir Said and John G Apostopoulos, editors, *Image and Video Communications and Processing, Proceedings od SPIE–IS&T Electronic Imaging*, volume SPIE 5685, pages 115–128, Bellingham, WA and Springfield, VA, January 2005. IS&T—The Society for Imaging Science and Technology and SPIE—The International Society for Optical Engineering, SPIE and IS&T. Invited paper.

[86] Tomáš Svoboda, Hanspeter Hug, and Luc Van Gool. V**i**Room — low cost synchronized multicamera system and its self-calibration. In Luc Van Gool, editor, *Pattern Recognition, 24th DAGM Symposium*, number 2449 in LNCS, pages 515–522, Berlin, Germany, September 2002. Springer.

[87] Tomáš Svoboda, Jan Kybic, and Hlaváč Václav. *Image Processing, Analysis and Machine Vision — A MATLAB Companion.* Thomson, Toronto, Canada, 1st edition, September 2007.

[88] Tomáš Svoboda, Daniel Martinec, and Tomáš Pajdla. A convenient multicamera self-calibration for virtual environments. *PRESENCE: Teleoperators and Virtual Environments*, 14(4):407–422, August 2005.

[89] Tomáš Svoboda and Tomáš Pajdla. Epipolar geometry for central catadioptric cameras. *International Journal of Computer Vision*, 49(1):23–37, August 2002.

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

[91] R. Urtasun and P. Fua. 3d tracking for gait characterization and recognition. In *8th European Conference on Computer Vision*, pages 17–22, 2004.

[92] Luca Vacchetti, Vincent Lepetit, and Pascal Fua. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1385–1391, 2004.

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

[94] Olivier Williams, Andrew Blake, and Roberto Cippola. Sparse bayesian learning for efficient visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1292–1304, August 2005.

[95] Stephan Würmlin, Edouard Lamboray, and Markus Gross. 3D video fragments: Dynamic point samples for real-time free-viewpoint video. *Computers and Graphics*, 28(1):3–14, 2004.

[96] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4), 2006.

[97] Shaohua Kevin Zhou, Bogdan Georgescu, Xiang Sean Zhou, and Dorin 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.

[98] Qiang Zhu, Shai Avidan, Mei-Chen Yeh, and Kwang-Tin Cheng. Fast human detection using a cascade of histograms of oriented gradients. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, pages 1491–1498, 2006.

[99] Karel Zimmermann. *Fast learnable methods for object tracking.* PhD Thesis CTU–CMP–2008–09, Czech Technical University, November 2008.

[100] Karel Zimmermann, David Hurych, and Tomáš Svoboda. Improving cascade of classifiers by sliding window alignment in between. In G. Sen Gupta, Donald Bailey, Serge Demidenko, and Dale Carnegie, editors, *Proceedings of the Fifth International Conference on Automation, Robotics and Applications*, pages 196–201, Private Bag 11 222, Palmerston North, New Zealand, December 2011. School of Engineering and Advanced Technology, Massey University, Massey University. CD-ROM, `http://dx.doi.org/10.1109/ICARA.2011.6144881`.

[101] Karel Zimmermann, Jiří Matas, and Tomáš Svoboda. Tracking by an optimal sequence of linear predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):677–692, April 2009.

[102] Karel Zimmermann and Tomáš Svoboda. Approximation of euclidean distance between point from ellipse. Research Report CTU–CMP–2005–23, CMP, Center for Machine Perception, Faculty of Electrotechnical Engineering, Czech Technical University, Prague, Czech Republic, August 2005.

[103] Karel Zimmermann and Tomáš Svoboda. Probabilistic estimation of articulated body model from multiview data. In Peter Kneppo and Jiří Hozman, editors, *IFMBE Proceedings EMBEC'05, 3rd European Medical and Biological Engineering Conference*, pages 1–6, Prague, Czech Republic, November 2005. International Federation for Medical and Biological Engineering.

[104] Karel Zimmermann, Tomáš Svoboda, and Jiří Matas. Multiview 3D tracking with an incrementally constructed 3D model. In *Third International Symposium on 3D Data Processing, Visualization and Transmission*, Chapel Hill, USA, June 2006. University of North Carolina.

[105] Karel Zimmermann, Tomáš Svoboda, and Jiří Matas. Adaptive parameter optimization for real-time tracking. In *Proceedings of 11th IEEE International Conference on Computer Vision, workshop on Non-rigid registration and tracking through learning*, Madison, USA, October 2007. Omnipress.

[106] Karel Zimmermann, Tomáš Svoboda, and Jiří Matas. Simultaneous learning of motion and appearance. In *The 1st International Workshop on Machine Learning for Vision-based Motion Analysis, In conjunction with the 10th European Conference on Computer Vision 2008*, Grenoble, France, October 2008. INRIA Rhone-Alpes.

[107] Karel Zimmermann, Tomáš Svoboda, and Jiří Matas. Anytime learning for the NoSLLiP tracker. *Image and Vision Computing, Special Issue: Perception Action Learning*, 27:1695–1701, October 2009.