Accurate Interpolation in Appearance-Based Pose Estimation

Erik Jonsson and Michael Felsberg*

Computer Vision Laboratory
Dept. of Electrical Engineering, Linköping University
erijo@isy.liu.se, mfe@isy.liu.se

Abstract. One problem in appearance-based pose estimation is the need for many training examples, i.e. images of the object in a large number of known poses. Some invariance can be obtained by considering translations, rotations and scale changes in the image plane, but the remaining degrees of freedom are often handled simply by sampling the pose space densely enough. This work presents a method for accurate interpolation between training views using local linear models. As a view representation local soft orientation histograms are used. The derivative of this representation with respect to the image plane transformations is computed, and a Gauss-Newton optimization is used to optimize all pose parameters simultaneously, resulting in an accurate estimate.

1 Introduction

Object recognition and pose estimation can be done in several ways. In the bag-of-features approach, local coordinate frames are constructed around points of interest [5], [9], and features from each local frame vote for a certain object and pose hypothesis. In the model-based approach [2], [11], a geometrical model is fitted to the observed image. This approach is often very accurate, but requires a good initial guess and a manually constructed 3D model. Global appearance-based methods extract features from the appearance of the entire object and match these to training views in memory. Ever since [10], [7], the most common approach seems to be using PCA.

In this paper, we use an appearance-based method using full object views, but avoid PCA due to the global nature of this representation. The main goal is to maximize the accuracy of the pose estimate by interpolating between a limited number of training views. The interpolation method is based on representing the views with *channel-coded orientation* [3], [4], and optimizing all pose parameters (including position, rotation and scale in the image plane) simultaneously using a Gauss-Newton method. The method requires an initial guess, which in a real system could be obtained using your favorite fast but inaccurate bag-of-features approach.

* This work has been supported by EC Grant IST-2003-004176 COSPAL. This paper does not represent the opinion of the European Community, and the European Community is not responsible for any use which may be made of its contents.

B.K. Ersbøll and K.S. Pedersen (Eds.): SCIA 2007, LNCS 4522, pp. 1–10, 2007.

The motivation for using full object views is two-fold. The first reason is that once we have formed an initial object hypothesis, it makes sense to use as much image information as possible in order to get an accurate estimate. The second reason is that using full views, we can focus on the interpolation and view representation, and ignore other aspects like how to choose interest points and construct local frames in a bag-of-features approach. This makes it easier to compare different view representations. Similar interpolation techniques as proposed here should however be possible to integrate also in a bag-of-features framework.

In contrast to model-based methods, our approach requires no knowledge of 3D geometry in the system, and is in no way specific to 3D pose estimation. The training set could consist of any parameterized image set, e.g. a robotic arm in different configurations etc.

2 Algorithm

2.1 Pose Estimation

The appearance of an object is determined by the object state $\mathbf{p} = [\theta, \phi, s, \alpha, x, y]$. The parameters s, α, x, y represent the scale, rotation and position of the object in the image plane and will be referred to as the *image parameters* $\mathbf{p}_{\text{img}}$. The two auxiliary angles θ and ϕ cover all pose variations not explained by rotation in the image plane and will be referred to as the *pose angles* $\mathbf{p}_{\text{pose}}$.

During training, we learn the appearance of the object given (θ, ϕ) using canonical image parameters. The result of the learning can be seen as a function $\mathbf{f}$ that maps the pose angles to a predicted feature vector:

$$\hat{\mathbf{c}} = \mathbf{f}(\theta, \phi) \ . \tag{1}$$

During operation of the system, we maintain a current hypothesis of the object state, and cut out an image patch around the current (x, y) with rotation α and size s. This can be formalized by a function

$$\mathbf{c} = \mathbf{g}(s, \alpha, x, y) \tag{2}$$

producing an observed feature vector from the image given certain image parameters. The pose estimation problem is now to find an object state $\mathbf{p}_*$ which minimizes the difference between the observed and predicted feature vectors:

$$\mathbf{p}_* = \arg\min_{\mathbf{p}} \|\mathbf{r}(\mathbf{p})\|^2 \tag{3}$$

where

$$\mathbf{r}(\mathbf{p}) = \mathbf{f}(\theta, \phi) - \mathbf{g}(s, \alpha, x, y) \ . \tag{4}$$

This can be solved using your favorite optimization method. We use a Gauss-Newton method, with a simple backtracking line search [8]. The update step direction $\mathbf{s}$ is given by

$$\mathbf{J}\mathbf{s} = -\mathbf{r} \ , \tag{5}$$

where $\mathbf{J}$ is the Jacobian of $\mathbf{r}$:

$$\mathbf{J} = [\mathbf{f}',\ -\mathbf{g}'] = \left[\mathbf{f}'_\theta, \mathbf{f}'_\phi, -\mathbf{g}'_s, -\mathbf{g}'_\alpha, -\mathbf{g}'_x, -\mathbf{g}'_y\right] \tag{6}$$

The derivative of $\mathbf{g}$ with respect to transformations in the image plane depends on the choice of view representation and will be discussed in Sect. 3. The derivative of $\mathbf{f}$ can be approximated by a local linear approximation of the training manifold, discussed in Sect. 2.3.

In each step of the iterations, we measure $\mathbf{g}(\mathbf{p}_{\text{img}})$ directly in the query image, i.e. we cut out a new patch using the current $\mathbf{p}_{\text{img}}$ and extract a new feature vector from the image. A faster but less accurate option would be to keep the original feature vector and Jacobian, and use them as a linear approximation of $\mathbf{g}$ throughout the entire solution procedure.

2.2 Geometrical Interpretation of Gauss-Newton

To fully understand the method, it is useful to have a geometrical image in mind. The output of the functions $\mathbf{f}$ and $\mathbf{g}$ define two manifolds in feature space. The first manifold contains all expected appearances of the object, learned from training examples, and the second one contains all observable feature vectors at different positions in the query image. The objective is to find one point on each manifold such that the distance between the two points is minimal.

What the Gauss-Newton method does is to approximate each manifold with its tangent plane and find the points of minimal distance on these hyperplanes. Let $\mathbf{f}(\mathbf{p}_{\text{pose}} + \mathbf{s}_{\text{pose}}) \approx \mathbf{f}(\mathbf{p}_{\text{pose}}) + \mathbf{f}'(\mathbf{p}_{\text{pose}})\mathbf{s}_{\text{pose}}$ and $\mathbf{g}(\mathbf{p}_{\text{img}} + \mathbf{s}_{\text{img}}) \approx \mathbf{g}(\mathbf{p}_{\text{img}}) + \mathbf{g}'(\mathbf{p}_{\text{img}})\mathbf{s}_{\text{img}}$. The minimum-distance points are given by the over-determined equation system

$$\mathbf{f}(\mathbf{p}_{\text{pose}}) + \mathbf{f}'(\mathbf{p}_{\text{pose}})\mathbf{s}_{\text{pose}} = \mathbf{g}(\mathbf{p}_{\text{img}}) + \mathbf{g}'(\mathbf{p}_{\text{img}})\mathbf{s}_{\text{img}}\ , \tag{7}$$

which is solved by (5) with $\mathbf{s} = [\mathbf{s}_{\text{pose}}\ \mathbf{s}_{\text{img}}]$. If the linear approximation is good enough, we can expect good results even after a single iteration.

2.3 Local Linear Approximation

In this section, we describe how to approximate the value and derivative of $\mathbf{f}$ by a variety of *locally weighted regression* [1], [6]. To simplify the notation and avoid double subscripts, let $\mathbf{p} = [\theta, \phi]^{\mathrm{T}}$, and let $\mathbf{p}_0$ be the current guess of $\mathbf{p}$, i.e. we consider only the auxiliary pose angles. The system is given a set of training views with pose angles $\mathbf{p}_i$, from which features $\mathbf{c}_i$ are extracted. The learning consists simply of storing all these training samples $\{\mathbf{p}_i, \mathbf{c}_i\}$. In operation mode we need the value and derivative of $\mathbf{f}$ at the current hypothesis $\mathbf{p}_0$, which is computed by fitting a linear model to the training samples closest to $\mathbf{p}_0$. The basic strategy is to weight all training samples according to their distance to $\mathbf{p}_0$:

$$w_i = K(\|\mathbf{p}_i - \mathbf{p}_0\|)\ . \tag{8}$$

Here K is a smooth Gaussian-looking weighting kernel with compact local support; in our case a second-order B-spline. We then solve the weighted least-squares problem

$$\min_{\mathbf{A},\mathbf{b}} \sum_i w_i \|(\mathbf{A}(\mathbf{p}_i - \mathbf{p}_0) + \mathbf{b} - \mathbf{c}_i\|^2 \ . \tag{9}$$

This produces an interpolation using neighboring points only. From the Taylor expansion of $\mathbf{f}$, we can identify $\mathbf{b}$ and $\mathbf{A}$ as the approximated function value and derivative respectively:

$$\mathbf{f}(\mathbf{p}) \approx \mathbf{f}(\mathbf{p}_0) + \mathbf{f}'(\mathbf{p}_0)(\mathbf{p} - \mathbf{p}_0) \approx \mathbf{b} + \mathbf{A}(\mathbf{p} - \mathbf{p}_0) \ . \tag{10}$$

If the training views are irregularly distributed in (θ, ϕ)-space, the number of samples included within the support of K is arbitrary and may even be zero. In contrast, if the weighting kernel is large, the linear approximation may be poor. Instead of using a fixed kernel, we could always select the k nearest neighbors, but without any sample weighting this would produce a discontinuity when set of neighbors changes. Since we expect $\mathbf{f}$ to be a smooth function, and since we are going to use the approximation in an iterative optimization, it is important that our approximation is also smooth.

Our method does something in between, by using a weighting kernel that is scaled according to the nearest training samples. We let $K(r)$ be scaled such that $K(r) = 0$ iff $r > 1$ and sort the training samples by their distance to $\mathbf{p}_0$, such that $\|\mathbf{p}_i - \mathbf{p}_0| \ \leq \|\mathbf{p}_{i+1} - \mathbf{p}_0\|$ for all i. We now weight our samples according to

$$w_i = K\left(\beta \|\mathbf{p}_i - \mathbf{p}_0\| / \|\mathbf{p}_k - \mathbf{p}_0\|\right) \ . \tag{11}$$

If $\beta = 1$, this gives zero weight to the k'th sample and non-zero weight to all samples strictly closer to $\mathbf{p}_0$. However, if there are several samples with the same distance to $\mathbf{p}_0$ as $\mathbf{p}_k$, all these samples will get zero weight as well, which may produce too few active samples for a reliable model fitting. This is solved by choosing β slightly larger than 1, giving at least k active samples. Each w_i is now a continuous function of $\mathbf{p}_0$, and $\mathbf{A}, \mathbf{b}$ depend continuously on the w_i's. This ensures that our approximation responds continuously to changes in $\mathbf{p_0}$.

3 View Representation

Given the pose estimation procedure in Sect. 2.1, we wish to find a good description of each view in terms of a feature vector $\mathbf{c}$. To make the representation depend continuously on the input image, we avoid too complicated things like region detection etc. What we want is a simple non-linear transformation of the view. In order to be invariant to lighting, we use local orientation instead of image intensity. For robustness against occlusion and background clutter, we avoid global view representations like PCA or DCT. One simple option could be to simply downsample the gradient magnitude. Note however that in order for the interpolation between training views to be successful, the same edge of an object must be visible within the receptive field of one pixel in the downsampled

image for several views (see Fig. 1). Since our training views are rather coarsely spaced, we would need very heavy smoothing or downsampling, which would destroy much image information. Instead, we use *channel coded orientation* or *soft orientation histograms* as view representation.

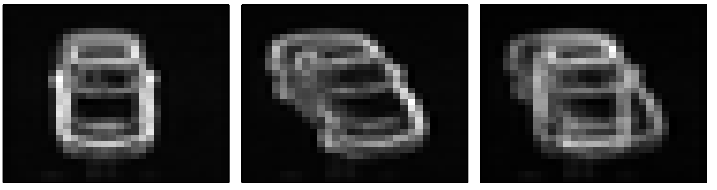

Fig. 1. Top: Gradient magnitude of two adjacent training views and an interpolated intermediate view. Since the spatial resolution of the feature representation is too large, linear interpolation does not produce the expected features of intermediate views.

3.1 Channel Coding

Given a scalar-valued feature image $z(x, y)$ and a weight function $w(x, y)$, the *channel coded feature map* is a 3D array $\mathbf{c}$ with elements

$$c_{ijk} = \int_{\mathbb{R}^2} B_{ijk}(x, y, z(x, y))w(x, y) \, \mathrm{d}x \, \mathrm{d}y \ , \tag{12}$$

where

$$B_{ijk}(x, y, z) = B_1(x - \tilde{x}_i)B_2(y - \tilde{y}_j)B_3(z - \tilde{z}_k) \tag{13}$$

are smooth, localized but overlapping basis functions, causing each pixel to smoothly contribute to several channels. Each channel c_{ijk} measures the presence of the feature value $\tilde{z}_k$ around image position $(\tilde{x}_i, \tilde{y}_j)$. The points $(\tilde{x}_i, \tilde{y}_j, \tilde{z}_k)$ are called the *channel centers*. We can think of it as first representing the feature image as a set of points (x, y, z) in a 3D *spatio-featural* space, and then downsampling this space in all three dimensions simultaneously.

In our case, the feature z is local orientation taken modulo π, such that we do not distinguish between positive and negative edges. The gradient magnitude is used as weight $w(x, y)$. If the kernels are chosen as rectangular and non-overlapping, we get a simple 3D histogram. If we create a binary weight $w_{\mathrm{B}}(x, y)$ by thresholding $w(x, y)$ at 10% of the maximum value and use first order (linear) B-spline [12] as basis function, we something similar to the SIFT descriptor [5]. By increasing the overlap and smoothness of the basis functions, we expect to get a smoother behavior. In the evaluations, second order B-spline kernels will be used.

The expected advantage comes from the fact that we can use a coarse spatial resolution but still maintain much useful information. For example, we can represent the presence of multiple orientations in a region without averaging them together. The low spatial resolution and the smoothness of the basis functions makes it more likely that the view representation transforms smoothly between the training poses, which makes it suitable for view interpolation.

3.2 Derivatives of the Channel Coding

In updating the image parameters iteratively according to Sect. 2.1, the derivatives of the view representation with respect to the image parameters are required. For simplicity of notation, we ignore the weight function $w(x, y)$ for a moment, and rewrite (12) in vector notation:

$$c_{ijk} = \int_{\mathbb{R}^2} B_{ijk}(\mathbf{x}, z(\mathbf{x}))\mathrm{d}\mathbf{x} \ . \tag{14}$$

The weights will be considered again in Sect. 3.3. We consider now a certain channel coefficient c_{ijk}, and to further simplify the notation, the integral limits and indices ijk will be dropped. We are interested in what happens when the feature image is rotated, scaled and translated:

$$c = \int B(\mathbf{x}, z(\mathbf{A}\mathbf{x} + \mathbf{b}))\ \mathrm{d}\mathbf{x} \tag{15}$$

where

$$\mathbf{A} = e^s\mathbf{R} = e^s \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} b_x \\ b_y \end{pmatrix} \tag{16}$$

Substituting $\mathbf{u} = \mathbf{A}\mathbf{x} + \mathbf{b}$ where $\mathbf{u} = [u, v]^{\mathrm{T}}$ gives $\mathbf{x} = \mathbf{A}^{-1}(\mathbf{u} - \mathbf{b})$ and

$$c = |\mathbf{A}^{-1}| \int B(\mathbf{A}^{-1}(\mathbf{u} - \mathbf{b}), z(\mathbf{u}))\, \mathrm{d}\mathbf{u} \ . \tag{17}$$

We now want to differentiate (17) with respect to α, s, b_x, b_y, and start with α. For compactness, we leave out the arguments to B and its derivatives. These arguments are always as in (17). First note that $|\mathbf{A}^{-1}|$ is constant with respect to α. Since everything is smooth and well-defined, we can replace the order of the integration and differentiation.

$$\frac{\mathrm{d}c}{\mathrm{d}\alpha} = |\mathbf{A}^{-1}| \int \frac{\mathrm{d}}{\mathrm{d}\alpha} \left[B(\ldots)\right] \mathrm{d}\mathbf{u} = |\mathbf{A}^{-1}| \int B'_{\mathbf{x}}(\ldots) \frac{\mathrm{d}\mathbf{A}^{-1}}{\mathrm{d}\alpha}\mathbf{u}\ \mathrm{d}\mathbf{u} \tag{18}$$

where

$$\frac{\mathrm{d}\mathbf{A}^{-1}}{\mathrm{d}\alpha} = e^{-s} \begin{bmatrix} -\sin\alpha & \cos\alpha \\ -\cos\alpha & -\sin\alpha \end{bmatrix} \tag{19}$$

$$B'_{\mathbf{x}} = [B'_x,\ B'_y] \tag{20}$$

The differentiation with respect to $\mathbf{b}$ proceeds similarly. We get

$$\frac{\mathrm{d}c}{\mathrm{d}\mathbf{b}} = |\mathbf{A}^{-1}| \int \frac{\mathrm{d}}{\mathrm{d}\mathbf{b}} \left[B(\ldots)\right] \mathrm{d}\mathbf{u} = -|\mathbf{A}^{-1}| \int B'_{\mathbf{x}}(\ldots)\mathbf{A}^{-1}\ \mathrm{d}\mathbf{u} \tag{21}$$

In differentiating with respect to s, $|\mathbf{A}^{-1}|$ is no longer constant. The product rule gives us

$$\frac{\mathrm{d}c}{\mathrm{d}s} = \frac{\mathrm{d}|\mathbf{A}^{-1}|}{\mathrm{d}s} \int B(\ldots)\,\mathrm{d}\mathbf{u} \;+\; |\mathbf{A}^{-1}| \int \frac{\mathrm{d}}{\mathrm{d}s}\left[B(\ldots)\right]\,\mathrm{d}\mathbf{u} = \tag{22}$$

$$= -\,2|\mathbf{A}^{-1}| \int B(\ldots)\,\mathrm{d}\mathbf{u} \;+ |\mathbf{A}^{-1}| \int B'_{\mathbf{x}}(\ldots)\frac{\mathrm{d}\mathbf{A}^{-1}}{\mathrm{d}s}\mathbf{u}\,\mathrm{d}\mathbf{u} = \tag{23}$$

$$= -\,|\mathbf{A}^{-1}| \int 2B(\ldots) + B'_{\mathbf{x}}(\ldots)\mathbf{A}^{-1}\mathbf{u}\,\mathrm{d}\mathbf{u} \tag{24}$$

Setting $s = 0, \alpha = 0, \mathbf{b} = 0$ gives $\mathbf{A}^{-1} = \mathbf{I}$, and the derivatives in (18), (21) and (24) become

$$\frac{\mathrm{d}c}{\mathrm{d}b_x} = -\int B'_x(\mathbf{u}, z(\mathbf{u}))\,\mathrm{d}\mathbf{u} \tag{25}$$

$$\frac{\mathrm{d}c}{\mathrm{d}b_y} = -\int B'_y(\mathbf{u}, z(\mathbf{u}))\,\mathrm{d}\mathbf{u} \tag{26}$$

$$\frac{\mathrm{d}c}{\mathrm{d}s} = -\int 2B(\mathbf{u}, z(\mathbf{u})) + uB'_x(\mathbf{u}, z(\mathbf{u})) + vB'_y(\mathbf{u}, z(\mathbf{u}))\,\mathrm{d}\mathbf{u} \tag{27}$$

$$\frac{\mathrm{d}c}{\mathrm{d}\alpha} = \int vB'_x(\mathbf{u}, z(\mathbf{u})) - uB'_y(\mathbf{u}, z(\mathbf{u}))\,\mathrm{d}\mathbf{u} \tag{28}$$

By computing these derivatives for each channel and stacking them into vectors, we get $\mathbf{g}'_x, \mathbf{g}'_y, \mathbf{g}'_s, \mathbf{g}'_\alpha$ required in (6). Note that if the basis functions are B-splines, all terms in the above integrals are just piecewise polynomial functions with the same support, so the amount of computation required to evaluate each of the derivatives is in the same order of magnitude as computing the channel-coded feature map itself.

3.3 Weighted Data

In the previous section, the weights from (12) were not considered. By introducing these weights again, the results are similar. Since the weights are defined for each pixel in the feature image, they transform with the features, i.e. (15) becomes in the weighted case

$$c = \int B(\mathbf{x}, z(\mathbf{A}\mathbf{x} + \mathbf{b}))w(\mathbf{A}\mathbf{x} + \mathbf{b})\,\mathrm{d}\mathbf{x}\ . \tag{29}$$

After the variable substitution, we have

$$c = |\mathbf{A}^{-1}| \int B(\mathbf{A}^{-1}(\mathbf{u} - \mathbf{b}), z(\mathbf{u}))w(\mathbf{u})\,\mathrm{d}\mathbf{u} \tag{30}$$

In this expression, the weighting function is independent of the transformation parameters $\alpha, s, \mathbf{b}$ and is left unaffected by the differentiation. The complete expressions for the derivatives in the weighted case are just (25)-(28) completed with the multiplicative weight $w(\mathbf{u})$ inside the integrals.

3.4 Normalization

The method has shown to work better if the channel vectors are normalized using $\tilde{\mathbf{c}} = \mathbf{c}/\|\mathbf{c}\|$, where $\|\cdot\|$ is the L_2 norm. In this case, we should change the derivatives from previous section accordingly. From the quotient rule, we have

$$\frac{\mathrm{d}\tilde{\mathbf{c}}}{\mathrm{d}\alpha} = \|\mathbf{c}\|^{-2}\left(\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\alpha}\|\mathbf{c}\| - \mathbf{c}\frac{\mathrm{d}\|\mathbf{c}\|}{\mathrm{d}\alpha}\right) = \|\mathbf{c}\|^{-1}\left(\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\alpha} - \tilde{\mathbf{c}}\tilde{\mathbf{c}}^{\mathrm{T}}\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\alpha}\right) \tag{31}$$

where we have used that $\frac{\mathrm{d}\|\mathbf{c}\|}{\mathrm{d}\alpha} = \tilde{\mathbf{c}}^{\mathrm{T}}\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\alpha}$. The derivatives with the respect to the other variables are derived analogously.

4 Experiments

The method is evaluated on a set of objects scanned with a turn-table, producing training images like in Fig. 2. Views are available for every 5° in both the θ and ϕ domain. The method was trained on views 20° apart, and evaluated on all intermediate views. This gives 50 training views and 629 evaluation views.

In the first experiment, we assume that the image parameters are known, and optimize the pose angles $[\theta, \phi]$. The iterations were initialized at the closest training view, which is similar to what can be expected from an inaccurate object detector. The best parameter settings were found by an exhaustive search. The results using different varieties around the best option are shown in Table 1. As we see, the performance is rather insensitive to parameter variations in this order of magnitude.

In the second experiment, all 6 pose parameters were optimized simultaneously. One problem here is that the set of angles $[\alpha, \theta, \phi]$ is ambiguous such that two distinct set of angles can represent the same pose. Because of this, we combine the pose angles and image rotation into a rotation quaternion and measure the error in the quaternion domain. An RMS quaternion error of 0.015 corresponds to around 2° error in each angle. The method was initialized using the

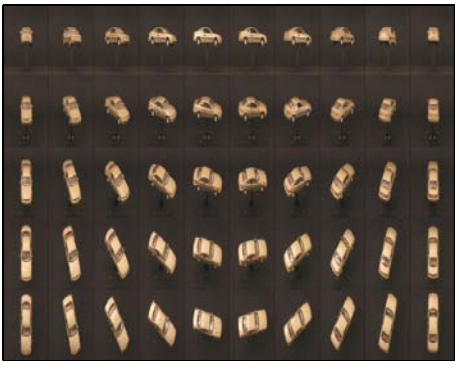

Fig. 2. Training views. Y-axis: θ, X-axis: ϕ.

Table 1. RMS error in degrees for pose angles only, around the manually selected option $8 \times 8 \times 6$ channels, 4 neighbors. Left: Varying spatial resolution. Middle: Varying number of orientation channels. Right: Varying number of neighbors.

$n_x \times n_y$	error
6x6	1.2
8x8	1.2
10x10	1.3
12x12	1.4
14x14	1.4

n_f	error
4	1.2
6	1.2
8	1.3
10	1.3

neighbors	error
3	1.3
4	1.2
5	1.2
6	1.2
7	1.3
8	1.6
9	1.9

Table 2. RMS error for all parameters (x,y,s error is in pixels) around the manually selected option $8 \times 8 \times 6$ channels, 4 neighbors. Left: Varying spatial resolution. Middle: Varying number of orientation channels. Right: Varying number of neighbors.

$n_x \times n_y$	x,y	s	q
6x6	3.5	6.8	0.016
8x8	3.5	6.2	0.016
10x10	4.2	6.7	0.017
12x12	4.1	7.0	0.017
14x14	4.5	7.6	0.018

n_f	x,y	s	q
4	3.55	6.6	0.015
6	3.5	6.8	0.016
8	3.5	7.0	0.015
10	3.6	7.5	0.016

neighbors	x,y	s	q
3	3.5	6.1	0.016
4	3.5	6.2	0.016
5	3.4	6.0	0.016
6	3.1	6.0	0.015
7	3.6	6.3	0.017
8	3.6	6.6	0.020

true image parameters and the closest pose parameters from the training set. The results for different options are shown in Table 2. Here s is the radius in pixels of a box containing the object, and the errors in x, y, s are measured in pixels. The size of the car in the images is around 300 pixels.

The current implementation runs at a few frames per second on an AMD Athlon 3800+.

5 Discussion

This paper has described an accurate interpolation method for view-based pose estimation using local linear models and Gauss-Newton optimization. Some varieties in the view representation have been compared in terms of fitness to this framework. However, the evaluation is in no way complete. There are several more parameters to play around with, e.g. the amount of overlap between basis functions, different soft tresholdings of the orientation image etc. It would also be interesting to perform a more detailed study on the performance of this representation compared to other approaches like PCA, wavelet representations etc.

The most critical fact is however that the evaluation dataset is too simple, without occlusion and difficult backgrounds. To verify that the method works in the real world, it has been run on hand-camera video sequences, but without

any quantitative error measures. Some video clips are available online [1]. A more rigorous evaluation should be performed on real image sequences with ground truth. In the near future, we plan to create datasets for this purpose and make them publicly available.

Recall that the full view setting was chosen mainly to be able to focus on the interpolation and feature representation. The goal of our future research is to transfer these techniques to methods based on local patches. This creates new problems, e.g. how to handle the fact that patches may be selected from different positions on the object in different views. Solving these problems will be challenging but hopefully rewarding in terms of greater performance.

References

1. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning. Artificial Intelligence Review 11, 11–73 (1997)
2. Comport, A., Marchand, E., Chaumette, F.: A real-time tracker for markerless augmented reality. In: Proc. The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 36–45 (2003)
3. Felsberg, M., Forssén, P.-E., Scharr, H.: Channel smoothing: Efficient robust smoothing of low-level signal features. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(2), 209–222 (2006)
4. Granlund, G.H.: An associative perception-action structure using a localized space variant information representation. In: Sommer, G., Zeevi, Y.Y. (eds.) AFPAC 2000. LNCS, vol. 1888, Springer, Heidelberg (2000)
5. Lowe, D.G.: Object recognition from local scale-invariant features. In: IEEE Int. Conf. on Computer Vision (September 1999)
6. Moore, A.W., Schneider, J., Deng, K.: Efficient locally weighted polynomial regression predictions. In: Proc. 14th International Conference on Machine Learning, pp. 236–244. Morgan Kaufmann, San Francisco (1997)
7. Murase, H., Nayar, S.K.: Visual learning and recognition of 3-d objects from appearance. International Journal of Computer Vision 14(1), 5–24 (1995)
8. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, Heidelberg (1999)
9. Obdrzalek, S., Matas, J.: Object recognition using local affine frames on distinguished regions. In: British Machine Vision Conf. (2002)
10. Pentland, A., Moghaddam, B., Starner, T.: View-based and modular eigenspaces for face recognition. In: CVPR (1994)
11. Cipolla, R., Drummond, T.: Real-time visual tracking of complex structures. IEEE Transactions on Pattern Analysis and Machine Intelligence,vol. 24(7) (July 2002)
12. Unser, M.: Splines: A perfect fit for signal and image processing. IEEE Signal Processing Magazine 16(6), 22–38 (1999)

[1] `http://www.cvl.isy.liu.se/Research/Object/ChannelCodedFeatureMaps/`