Face Tracking for Model-based Coding and Face Animation

Jörgen Ahlberg[1], Robert Forchheimer
Image Coding Group, Dept. of Electrical Engineering, Linköping University,
SE-581 83 Linköping, Sweden, E-mail: {ahlberg, robert}@isy.liu.se

ABSTRACT: We present a face and facial feature tracking system able to extract animation parameters describing the motion and articulation of a human face in real-time on consumer hardware. The system is based on a statistical model of face appearance and a search algorithm for adapting the model to an image. Speed and robustness is discussed, and the system evaluated in terms of accuracy.

I. INTRODUCTION

Tracking faces and facial features has many applications in communication, security, man-machine interfaces, and media creation. The application we address here is model-based coding and face animation; we want to extract animation parameters from a video sequence displaying a human face. The parameters should describe the motion and articulation of the face well enough to be used for synthesizing an animated face. Thus, we can enable very low bit-rate face-to-face communication and create animated faces for web services.

The advent of the MPEG-4 standard for face animation, enabling interoperability between different face tracking and animation systems, has stressed the need for face tracking methods. MPEG-4 provides a standardized way of representing, compressing and interpreting animation parameters, but how to generate the content is up to the user.

Even though face and facial feature tracking has been a topic for research for several decades, it is not a solved problem. Recently, many promising techniques have been proposed, of which some also show real-time performance. In this paper, we present a system based on statistical models of face appearance, and show results from a real-time tracker.

The paper is organized as follows. In Section II the concept of model-based coding is described, followed, in Section III, by a discussion on different kind of tracking systems for face animation. In the following sections we describe our tracker. In Section IV, the face model, its parameterization, and the model adaptation process (i.e., the tracking) are described. In Section V the training procedure and initial tracking experiments are described and some results shown, while Sections VI and VII treat how to improve the tracker in terms of speed and robustness. Section VIII contains an evaluation of the tracker's accuracy, and, finally, in Section X, our conclusions are drawn.

II. MODEL-BASED CODING

Since the major application of the techniques described in this document is model-based coding, an introduction to the topic will follow here. We briefly describe the principle of model-based coding, its history and terminology. For more details, see (Aizawa et al., 1993), (Li et al., 1994), (Pearson, 1995), or (Ström, 2002).

A. Basic Principle. The basic idea of model-based coding of video sequences is as follows: At the encoding side of a visual communication system, the image from the camera is analysed, using computer vision techniques, and the relevant object(s), for example a human face, is identified. A general or specific model is then adapted to the object—usually the model is a wireframe describing the 3D shape of the object.

Instead of transmitting the full image pixel-by-pixel, or by coefficients describing the waveform of the image, the image is handled as a 2D projection of 3D objects in a scene. To this end, parameters describing the object(s) are extracted, coded and transmitted. Typical parameters are size, position and shape. To achieve acceptable visual similarity to the original image, the texture of the object is also transmitted. The texture can be compressed by some traditional image coding technique, but specialized techniques can lower the bit-rate considerably in certain applications, as shown by Ström (1997).

1. J.Ahlberg is now with the Swedish Defence Research Agency, Div. of Sensor Technology, P.O.Box 1165, SE-581 11 Linköping, Sweden.

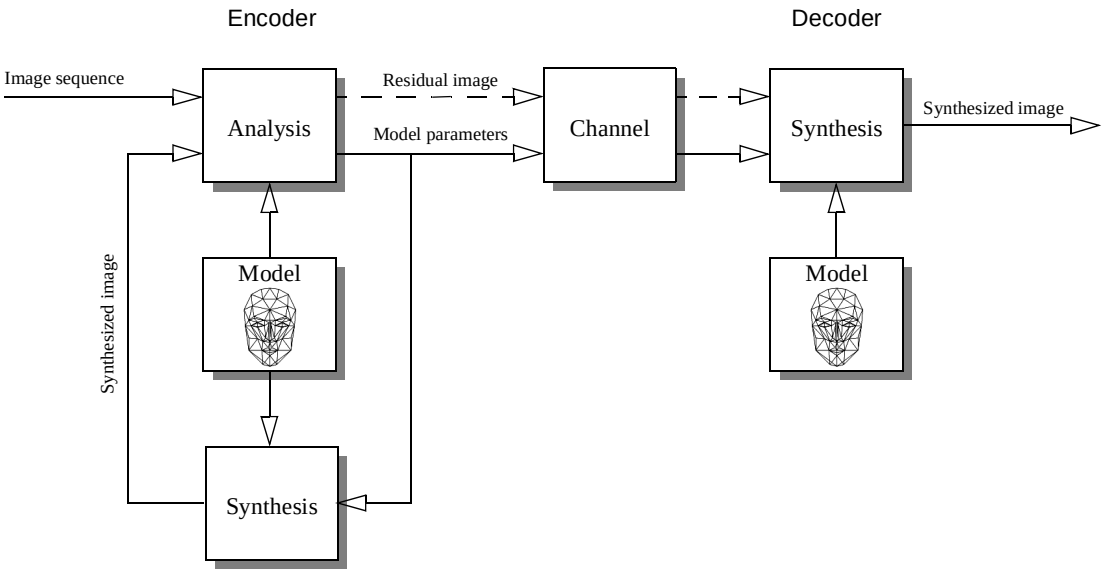


Figure 1: The principle of analysis-by-synthesis coding.

At the receiver side of the system, the parameters are decoded and the decoder's model is modified accordingly. The model is then synthesized as a visual object using computer graphics techniques, e.g., a wireframe is shaped according to the shape and size parameters and the texture is mapped onto its surfaces.

In the following frames, parameters describing the change of the model are transmitted. Typically, those parameters tell how to rotate and translate the model, and, in case of a non-rigid object like a human face, parameters describing motion of individual vertices of the wireframe are transmitted. This constitutes the largest gain of the model-based coding; the motion parameters can be transmitted at very low bit-rates.

A model-based coder can be implemented as an *analysis-by-synthesis* coder, in which case the encoder contains a feedback loop and a copy of the decoder, thus enabling it to compare the synthesized image to the original image. By minimizing the residual image, the encoder can optimize the transmitted parameters. This also permits the encoder to transmit a residual image, to improve image quality. The principle of analysis-by-synthesis coding is illustrated in Fig. 1.

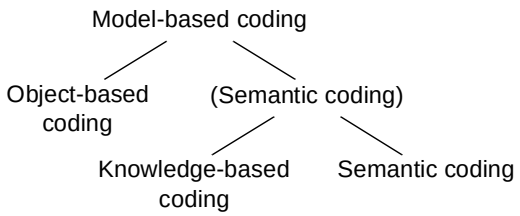


Figure 2: Terminology for model-based coding.

B. Terminology. Depending on the situation, the model-based coding system can either be equipped with a general model, e.g., a 3D wireframe being able to adapt to any object, or a more specific model, like a model of the human face, which can exploit the special characteristics of the specific class of objects. Using the terminology from Li et al. (1994), the first case is called *object-based* (or *object-oriented*) *coding*, while the latter is called *semantic coding*, see Fig. 2.

There can also be reasons to differ between two kinds of semantic coding. One is when we know in advance which specific model to use. The other is when each object, during the image analysis, is recognized as a member of a class, and the specific model corresponding to that class is selected automatically. The first case is sometimes called *knowledge-based coding*, thus reserving the term semantic coding for the highest level of intelligence in the coder (Harashima, 1989). However, the implicit criterion for a coder being a semantic coder is that the transmitted parameters are in some way semantically meaningful, like telling a face model to perform a specific facial action (e.g., wrinkle the nose) in contrast to moving vertices only.

In this paper, only the case where it is known in advance that the relevant object is a human face will be treated, and the transmitted parameters are to some degree semantically meaningful. This also relieves us from the general computer vision problem of understanding objects in images, and leaves us with the more specific problem of understanding facial images. This problem has shown to be quite challenging by itself.

C. History of Model-Based Coding. Before the introduction of model-based coding as we know it today, models for computerized facial animation were

created by Parke (1982) for computer graphics purposes. Parke suggested that his facial models might be used for data compression, but he did not pursue this further.

Another important precursor is the *Facial Action Coding System* (FACS) developed by Ekman and Friesen (1977) (probably inspired by the somewhat earlier work done by Hjortsjö (1969)). The goal of Ekman and Friesen was to investigate the relation between emotion and facial expression, and as a part thereof, the FACS was developed as a way to quantitatively measure the intensity of facial actions. Each minimal facial action, for example, lip stretch or nose wrinkle, was called an *Action Unit*, and by combining the about 50 different Action Units, every facial expression should be possible to analyse. The Action Units have later on become popular as parameters for animation of human faces, and was probably the system most widely used until the definition of the MPEG-4 Facial Animation Parameters.

The modern concept of model-based coding of human faces, or "coding through animation" was introduced by Forchheimer and Fahlander (1983). This was soon followed up by a system estimating Action Units from a video sequence (Forchheimer et al., 1984), using a simple wireframe face-model suitable for real-time animation known as Candide (Rydfalk, 1987). Candide is still a popular model, but nowadays facial animators typically use more advanced models. A modernized version of Candide, taking MPEG-4 compliancy into account, is available; see Section IV.

Model-based coding made a big leap when Aizawa (1987) and Welsh (1991) introduced texture-mapping in the synthesis part. Until then, only artificial textures, if any, had been used. The texture-mapping technique opened the door to photo realistic model-based coding. The works of Welsh and Aizawa also made the concept of model-based coding more popular as a research topic, as people became convinced that good-quality image communication could indeed be achieved this way.

The next major step in the history of model-based coding was the introduction of analysis-by-synthesis coding, as described above. This paved the way for robust video analysis.

In the nineties, model-based coding and face animation became increasingly popular. Intensive research has been done in face detection, face tracking, face expression analysis etc., exploiting advances in statistical models for computer vision. At the same time, the face models have been more and more advanced, often consisting of thousands of polygons and having advanced motion patterns to be able to synthesize natural looking animation. In 1999, the international standard MPEG-4 was ratified, including definitions for representation and coding of facial animation parameters.

Face recognition/analysis/modelling/animation are now very popular as research topics as well as areas for industrial and commercial development, one of the reasons being new media. When model-based coding was first thought of in the eighties, video telephones were the main intended application, while much research nowadays is motivated by animated faces in movies, computer games, and on the web. Current and future uses include a range of applications, such as human-computer interfaces, avatars, hearing disabled support, and virtual guides, salesmen, actors, and newsreaders. A survey of face animation on the web can be found in (Pandzic, 2001).

III. FACE TRACKING FOR ANIMATION

A tracking system estimates the rigid or non-rigid motion of an object through an image sequence. In the following, we discuss the two-frame situation, where we have a model of the object in the current frame, and the system should estimate how to transform the model to fit the object in the following frame.

A. Motion vs. Model-based Trackers. Tracking systems can be said to be either *motion-based* or *model-based*. A motion-based tracker estimates the displacements of pixels (or blocks of pixels) from one frame to another. The displacements might be estimated using optical flow methods (giving a dense optical flow field), block-based motion estimation methods (typically giving a sparse field but using less computational power), or motion estimation in a few image patches only (giving a few motion vectors only, but at very low computational cost).

The estimated motion field is used to compute the motion of the object model, using, for example, least square methods or extended Kalman filtering.

The motion estimation in such a method is consequently dependent on the pixels in two frames; the object model is used only for transforming the motion vectors to object model motion parameters. The problem with such methods is the so called *long sequence motion problem* or *drifting problem*, treated, for example, by Li et al. (1993). The basic idea is that any tracker of this kind will accumulate motion errors and gradually lose track of the object without the ability to recover.

A *model-based tracker*, on the other hand, uses a stored or generated texture of the object, and tries to change the object model's position (and possibly shape) parameters to fit the new frame. The motion estimation is thus dependent on the object model and the new frame—the old frame is not regarded at all. In such a tracker, the drifting problem does not exist,

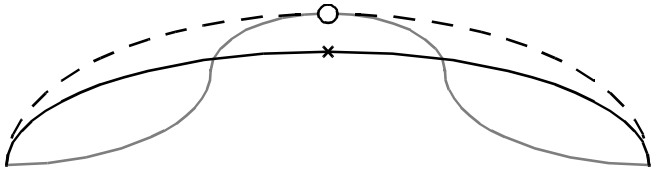


Figure 3: Tracking deformations or feature points.

instead, problems arise when the model is not strong or flexible enough to cope with the situation in the new frame. If the model cannot fulfil the high demands, the tracker will sooner or later lose track of the object. However, it still has the possibility to recover by itself.

B. Deterministic vs Statistical Model-based Trackers. One of the main issues when designing a model-based tracker is thus the texture model. An obvious approach is to capture a reference texture of the object in the beginning of the sequence. The texture could then be geometrically transformed according to the estimated motion parameters, so that changes in scale and rotation (and possibly non-rigid motion) can be handled. Since the actual texture is captured, the texture model is deterministic, object specific, and very accurate. Thus, trackers of this kind can be very precise, and systems working in real-time have been demonstrated by La Cascia et al. (2000), Wiles et al. (2001), Ström (2002).

A drawback, though, is the lack of flexibility in the texture model. This can cause problems with changing appearance due to variations in illumination, facial expression etc. Another drawback is that the initialization is very critical; if the captured texture was for some reason not representative of the sequence (due to partial occlusion, facial expression or illumination) or simply not the correct texture (if the object model was not correctly placed/shaped in the first frame) the tracker will not work well. Such problems can usually be solved by manual interaction, but can be hard to automate.

Another property is that the tracker does not know what it is tracking. This could be an advantage—the tracker can track different kind of objects—or a disadvantage. An example very relevant to our work is when the goal is to extract some higher level information from a human face, like facial expression or lip motion. We would then need a tracker that identifies and tracks specific facial features, for example, lip contours or MPEG-4 Facial Feature Points.

A slightly different approach is a *statistical model-based tracker.* Here, the texture model relies on previously captured textures combined with knowledge of which parts of the textures correspond to different facial features. When the model is transformed to fit the new frame, we will thus get information about the estimated positions of those specific facial features.

The texture model may be specific or general. A specific model could, for example, be trained on a data base containing images of one person only, resulting in an accurate model for this person. This model could be able to cope, to some degree, with the illumination and expression changes present in the data base. A more general texture model could be trained on a data base containing many different faces in different illuminations and with different facial expressions. Such a texture model would have a higher chance to enable successful tracking of a previously unseen face in a new environment, while a specific texture model presumably would result in better performance on the person and environment it was trained for. As a variation, adopted by La Cascia et al. (2000), the person-specific texture model (captured at the first frame) could be extended with a general model of illumination variation.

C. Tracking Deformations or Feature Points. The types of tracking systems discussed above can track *deformations* or *feature points*. A tracker based on feature points tries, in the rigid motion case, to estimate the position of a set of points, and from these points compute the position and pose of the object, as done by Ström (2002) and Wiles (2001).

In the non-rigid motion case, a tracker based on feature points would directly extract the points of interest, for example MPEG-4 Facial Feature Points for computation of MPEG-4 FAPs. The corresponding face synthesizer (face animation player, decoder, receiver) uses the point positions to compute the deformation of the model. If the synthesizer has a facial motion model that corresponds well to the motion of the tracked face, the deformations can be similar, but using different facial motion models, the output may look very different. Depending on the application, this may be good or bad; in a video-phone system, where the input and output images should be as similar as possible, it is obviously not preferable. On the other hand, when creating synthetic animated faces, like virtual characters, the motion model should produce the wanted appearance and behaviour of the animated face, and this might be altogether different from the face that generates the motion.

A *deformation-tracking* system will try to estimate a set of deformations in the face. We show the difference by an example; see Fig. 3. The gray line shows a contour in the image, for example the edge of the upper

lip. A deformation-tracking system representing the particular contour with an arc will estimate the contour with the black line. The face synthesizer would then render a face with the contour following the same arc. However, if the deformation tracker is used to estimate point positions, a feature point in the middle of the contour will be estimated at the cross, which is not correct.

A *point-tracking* system, with a feature point in the middle of the contour, will represent the contour with the point position marked with a circle. If the face synthesizer uses an arc as a motion model, it will render the contour marked by the dashed line. On the other hand, if the synthesizer does have a motion model that corresponds to the actual contour, a correct contour will be rendered.

In practice, the main drawback with point-tracking is that single points are usually hard to track robustly. There are certain points in a human face that are easy to track, as exploited by Petajan (2002) and Ström (2001), but unfortunately the easily trackable points are not the same as the interesting points.

The main drawback with deformation trackers is that they may give erroneous results when used for estimating parameters describing point positions, which is what computer graphics/animation people request.

IV. OUR FACE MODEL AND ITS PARAMETERIZATION

Our approach is to use the *active appearance model* (AAM) search algorithm to create a *statistical model-based deformation-tracking system*. The active appearance models were first introduced by Edwards et al (1998), and has been further treated by Cootes et al. (2001). To use the algorithm, we first need to decide a geometry for the face model, how its shape is controlled and how we use the model for warping images.

For the image warping, a triangular wireframe model where the control points are used as vertices is the straight-forward choice. The image warping can then be implemented as an ordinary texture mapping operation using affine transformations.

Since we intend to create a deformation-tracking system (in contrast to a point-tracking system) the model geometry needs to be sufficiently detailed to approximate the tracked deformations. However, we also want the model's complexity to be quite low due to the following reasons:

- The initial training will include the manual positioning of the control points in a large number of images. If the training should be performed within a reasonable amount of time, the number of control points needs to be kept fairly small (maybe a hundred).

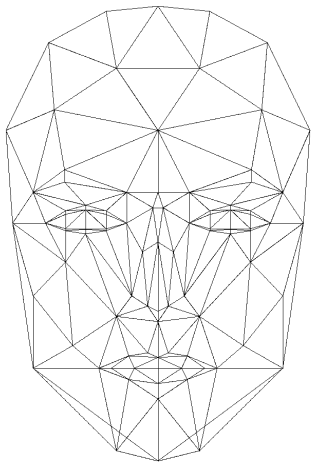


Figure 4: The Candide-3 face model.

- We want to keep the computational complexity down. Using an advanced face model with thousands of polygons and vertices could slow down the tracking.

From earlier work, the face model Candide by Rydfalk (1987) is available, including software and knowledge on how to handle it. Candide seems very suitable due to its low complexity (low number of polygons) and the fact that it covers the face only (not the entire head). However, it soon turned out that significant modifications were needed:

1. The eyes and (especially) the mouth were too crude to follow eyelid/lip contours. A few more vertices and polygons should be added.
2. The output from the tracking should preferably be easily expressed in terms of MPEG-4 Facial Animation Parameters (FAPs). Thus, (a subset of) the vertices of the model should correspond to the MPEG-4 Facial Feature Points (FFPs). To achieve this, vertices need to be added to the mouth, cheeks, eyes and nose.
3. When a correspondence between model vertices and MPEG-4 FFPs is set, FAPs and FAP Units should be defined as well.
4. The original Candide supported Action Units for dynamic deformations (animation), but to deform the model to fit a specific given face, static deformations changing the shape of the model are needed as well.

To implement the above changes, the Candide-3 model was created and is illustrated in Fig. 4. A complete documentation of the model, including file format and MPEG-4 correspondences is available in (Ahlberg, 2001) and as an appendix in (Ahlberg, 2002).

Note that we typically can use a much simpler model for tracking than we would use for animation. The Candide model is advanced enough for extracting the MPEG-4 FAPs, but for face animation, a more complex model is needed for visually pleasing results.

A. Controlling the Shape and Texture. The face model is a wireframe model with a texture mapped onto its surfaces. We use a statistical model of the texture and control the texture with a small set of texture parameters ξ. The texture is then generated as

$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{X}\xi \tag{1}$$

where $\bar{\mathbf{x}}$ is the mean texture and the columns of the matrix $\mathbf{X}$ are the *texture modes*, i.e., eigenfaces computed from *geometrically normalized face images* (Ström, 1997) or *shape-free face images* (Cootes et al., 2001, Craw et al., 1987). Thus, ξ is the vector of texture parameters, and the synthesized texture $\mathbf{x}$ is mapped on the wireframe model.

The shape of the Candide model is parameterized according to

$$\mathbf{s}_0 = \bar{\mathbf{s}} + \mathbf{S}\sigma + \mathbf{A}\alpha \tag{2}$$

where the resulting vector $\mathbf{s}_0$ contains the (x, y, z) coordinates of the vertices of the model. $\bar{\mathbf{s}}$ is the standard shape of the model, and the columns of the matrices $\mathbf{S}$ and $\mathbf{A}$ are the shape and animation modes respectively. Thus, σ and α contain shape and animation parameters. The difference between shape and animation modes is that the shape modes define deformations that differ individuals from each other, while the animation modes define deformations that occur due to facial expression.

Since we also want to perform global motion, i.e., pose change, we need six parameters for rotation and translation. Adopting the weak perspective projection (Aloimonos, 1990), we can replace the three translation parameters with 2D translation and a scaling factor, thus going directly from model coordinates to projected (screen) coordinates by replacing Eq. (2) with

$$\mathbf{s} = (z + 1)\mathbf{R}(\bar{\mathbf{s}} + \mathbf{S}\sigma + \mathbf{A}\alpha) + \mathbf{t} \tag{3}$$

where $\mathbf{R} = R(r_x, r_y, r_z)$ is a rotation matrix, z is the scale, and $\mathbf{t} = t(t_x, t_y)$ is the translation vector.

The geometry of our model is thus parameterized by the parameter vector

$$\mathbf{p} = [r_x, r_y, r_z, z, t_x, t_y, \sigma^{\mathrm{T}}, \alpha^{\mathrm{T}}]^{\mathrm{T}}. \tag{4}$$

The reason why $z + 1$ is used as a scaling factor (instead of just z) is to make the static (standard) shape correspond to an all-zeros parameter vector.

Note that the above parameterization differs from the original formulation of the AAMs. The original model is entirely a two-dimensional model, and thus there are no parameters for out-of-plane rotation (rotation around the x- and y-axes). Instead, such rotations are handled like any other 2D deformation, and are thus built-in in the shape modes. Other differences are that in the original formulation, there is no distinction between shape modes and animation modes, and that 3D coordinates are used in our case.

B. Fixing the Static Shape. When adapting a model to a video sequence, the shape parameters σ should only be changed in the first frame(s)—the head shape does not vary during a conversation—while the animation parameters α and the global parameters naturally change at each frame. Thus, during the tracking process we can assume that σ is known, and let the standard shape be person specific

$$\bar{\mathbf{s}}' = \bar{\mathbf{s}} + \mathbf{S}\sigma \tag{5}$$

and optimize $\mathbf{s}$ over the reduced parameter vector

$$\mathbf{p}' = [r_x, r_y, r_z, z, t_x, t_y, \alpha^{\mathrm{T}}]^{\mathrm{T}}. \tag{6}$$

In the following section we ignore the difference between $\mathbf{p}'$ and $\mathbf{p}$, since the process is identical.

C. Adapting the Model to the New Frame. Our goal is to find the optimal adaptation of the model to a frame in an image sequence, that is, to find the parameter vector $\mathbf{p}$ (or $\mathbf{p}'$) that minimizes a distance measure between the model and the frame. As initial value of $\mathbf{p}$ we use the parameter vector that adapts the model to the previous frame in the sequence, assuming that the motion from one frame to another is small enough. We reshape the model according to $\mathbf{s}(\mathbf{p})$, and map the input image $\mathbf{i}$ (the new frame) onto the model. We then reshape the model to the standard shape, $\bar{\mathbf{s}}$, and get the resulting normalized image as a vector

$$\mathbf{j}(\mathbf{i}, \mathbf{p}) = \mathbf{j}(\mathbf{i}, \mathbf{s}(\mathbf{p})). \tag{7}$$

Since our model is parameterized separately in shape and texture, we compute the texture parameters from the normalized input image according to

$$\xi(\mathbf{i}, \mathbf{p}) = \mathbf{X}^{\mathrm{T}}(\mathbf{j}(\mathbf{i}, \mathbf{p}) - \bar{\mathbf{x}}). \tag{8}$$

Inserting this in Eq. (1), we get

$$\mathbf{x}(\mathbf{i}, \mathbf{p}) = \bar{\mathbf{x}} + \mathbf{X}\mathbf{X}^{\mathrm{T}}(\mathbf{j}(\mathbf{i}, \mathbf{p}) - \bar{\mathbf{x}}) \tag{9}$$

and we compute the residual image

$$\mathbf{r}(\mathbf{i}, \mathbf{p}) = \mathbf{j}(\mathbf{i}, \mathbf{p}) - \mathbf{x}(\mathbf{i}, \mathbf{p}), \tag{10}$$

the error measure

$$e = \|\mathbf{r}(\mathbf{i}, \mathbf{p})\|^2, \tag{11}$$

and the *update vector*

$$\Delta\mathbf{p} = \mathbf{U}\mathbf{r}(\mathbf{i}, \mathbf{p}). \tag{12}$$

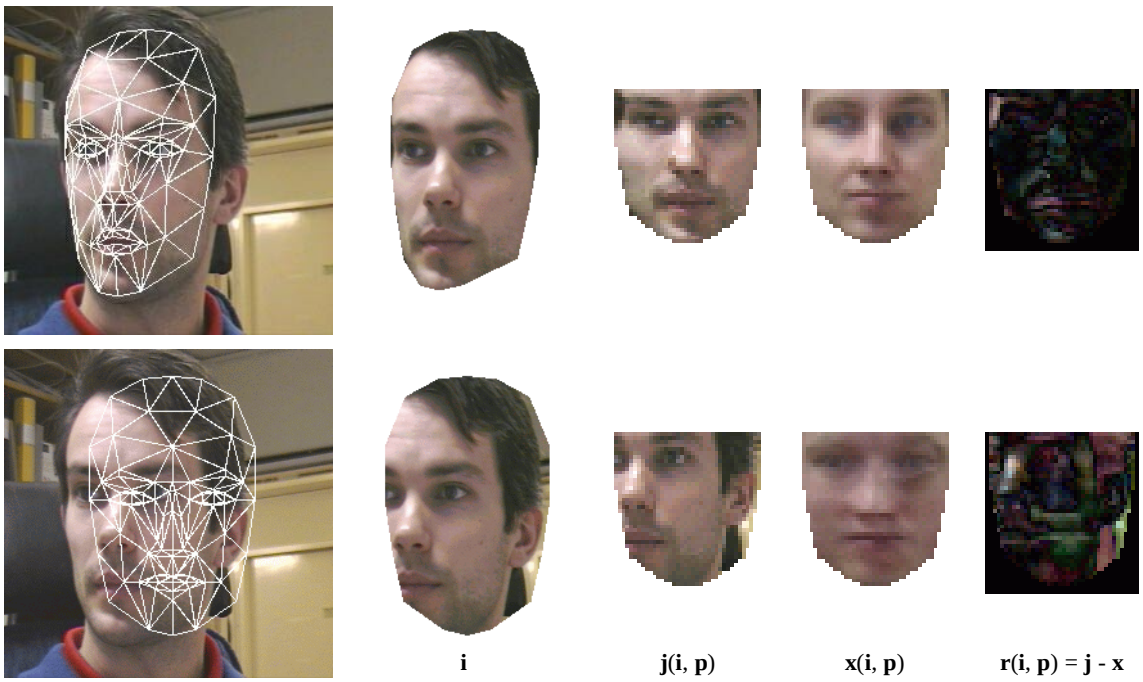


Figure 5: The model matching and texture analysis-synthesis process. a) A good and a bad (top and bottom row respectively) model adaptation is shown. b) The image mapped onto the model. c) The model is reshaped to the standard shape, producing the image vector **j**(**i**, **p**) according to Eq. (7). d) The normalized texture is approximated by the texture modes, producing the image vector **x**(**i**, **p**) according to Eq. (9). e) The residual image **r**(**i**, **p**) according to Eq. (10) is computed. The image vectors **j**(**i**, **p**) and **x**(**i**, **p**) are more similar the better the model adaptation is.

The update vector is computed by multiplying the residual image vector with an update matrix **U**, which is computed from training data according to the AAM search. The computation of **U** is briefly described in Appendix A, and the acquisition and processing of the training data is treated below.

The process described by Eqs. (7)-(10), from input image to residual, is illustrated in Fig. 5.

We compute the new error measure

$$e_k = \left\| \mathbf{r}\left(\mathbf{i}, \mathbf{p} + \frac{1}{2^{k-1}} \Delta \mathbf{p} \right) \right\|^2 \qquad (13)$$

for $k = 1$, and if $e_k < e$ we update

$$e_k \rightarrow e \qquad (14)$$

$$\mathbf{p} + k\Delta\mathbf{p} \rightarrow \mathbf{p} \qquad (15)$$

and iterate until convergence. We declare convergence when $e_k > e$ for $k = 1, 2, 3$.

D. Design Choices. As has been implicitly mentioned above, there are a number of design choices to make when building the model:

- *How to acquire the training data:* Choices include the number of persons to use, how many images on each person we should use, and if we should use images captured with similar or different illumination.

  Here we have chosen quite a large number of images per person (around 50) in order to capture different facial expressions and different head poses for each person. In contrast, we have quite few persons in the training set. The reason for not having more people in the data base is that it is a lot of manual work to record many persons, and also to adapt the wireframe model to each image.

- *How many texture modes should be computed, and in what format?* This is a compromise between speed and reconstruction quality—more texture modes will better reconstruct the image, and may generalize better, but the computational complexity of the analysis-synthesis grows approximately linearly with the number of eigenfaces. Experiments have been performed with 5, 10, and 20 texture modes, showing increasing accuracy given more texture modes. In the following, we use 10 texture modes, which seems to be the lowest we can go with reasonable tracking quality.

  Another question is what size should be used for texture modes. In the initial experiment, the normalized image was 40×42 pixels, which is large enough to represent a face in enough detail to follow its movements. However, some of the facial features will not be clearly trackable with precision; higher resolutions (80×84, 160×168) has shown to improve the precision.

A third choice is the colour format. In the initial experiment, the texture modes were computed in RGB, so that each pixel contributed with three coefficients to the training vector. Typically, colour variations provide much information about the face and its features, that in this way might be exploited. For example, Tzovaras et al. (1999) demonstrates improved performance in an eigenface-based face detector when using colour eigenfaces. The drawback is the increase in computational complexity.

- *What animation modes should we choose?* There are, at least, two different schemes to choose from; adapt the model to a set of training images, and perform a PCA on the point positions (when scale, translation and possibly rotation have been compensated for) to compute the principal deformations. The other scheme is to choose the deformations that we are interested in tracking depending on the application. For example, if the goal of our system is to do automated lip reading, we would need a set of deformations describing lip motion, but tracking eyelids and/or eyebrows would be unnecessary.

  In our case, we have chosen the six animation modes that seem to be most crucial for creating an animated face.

- *In how large and how many steps should the model parameters be perturbed?* To estimate the gradients needed to compute the update matrix U, the model parameters should be perturbed in small steps (see Appendix A). The step size and the number of steps (the design variables *h* and *K)* have been chosen to 0.01 and 40 respectively, purely by intuition. Details can be found in Ahlberg (2002).

V. TRAINING AND TESTING

A. Collecting and processing the training data.

When the design choices accounted for above are made, and training images are acquired, the training of the model can performed. The face model geometry has been adapted, by changing the parameter vector **p**, to $N_i = 330$ facial images with different head pose and with different facial expressions. Six persons were included in the training set. The model adaptation was originally done manually, but later semi-automatically. Two pictures of each person, with models adapted, are shown in Fig. 6. The same camera have been used for all images, and the variation in illumination is small.

The adapted models have, for each image, been normalized to a standard shape with the size 40×42 pixels, as Fig. 5 c), top row, and stored as a 5040-dimensional texture vector (1680 pixels and three colour components in each pixel). To reduce dependency on illumination, the texture vectors have been processed in the following ways:

- Each texture image has (before reshaping it into a vector) been mirrored in the y-axis, thus doubling the number of training textures.
- The DC level (the average value) has been removed from each texture vector.
- The norm of each vector has been set to one.

Note that this also requires the image to be analysed (**j**(**i**, **p**) in Eqs. (7)-(10)) to be processed in the same way (setting the DC to zero and the norm to one).

On the processed set of training vectors, a principal component analysis (PCA) has been performed to compute a set of texture modes. Initially, we chose to compute 10 texture modes.

As animation modes, six Action Units from Candide-3 have been chosen: *Jaw drop*, *Lip stretcher*, *Lip corner depressor*, *Upper lip raiser*, *Eyebrow lowerer*, and *Outer eyebrow raiser*.

Thus, we have six animation parameters in our vector α, and a total of $N_p = 12$ parameters in the model parameter vector, **p**, to be optimized.

For each image in the training set, each model parameter has been perturbed, in using a parameter step size $h = 0.01$ in K = 40 steps. For each perturbation, the image has been geometrically normalized and approximated using the texture modes. The

$$K/2 \cdot N_p \cdot N_i = 20 \cdot 12 \cdot 330 = 79\,200 \qquad (16)$$

residual images have been computed, and the update matrix **U** has been computed as described in Appendix A with parameters (N_i, N_p, *h*, *K*) = (330, 12, 0.01, 40).

The two translation parameters need to be treated slightly different. For the translation to be relative to the size of the model, the step size *h* has been changed to $h = 0.01 \cdot (z + 1)$ when computing the corresponding columns of **G**.

B. Initial tracking experiments. To test the system, two video sequences of a previously unseen person has been recorded, using the same camera as when capturing the training set. In the first sequence, the illumination was the same as in the training set, in the second one it differed somewhat. The Candide-3 model was manually adapted to the first frame of the sequence by changing the pose parameters and the static shape parameters (recall that the shape parameter vector σ is assumed to be known). The parameter vector **p** containing pose and animation parameters was then iteratively optimized for each frame. The results were quite satisfactory, and the resulting face model adaptations are shown in Fig. 7. As can be seen, the pose (rotation, scale, translation) is well estimated most of the time, and the mouth and eyebrow parameters behave well.

Figure 6: Two images from each of the six persons in the ICG training set.

The exception is from the 20[th] to the 40[th] frame (approximately) in the first sequence, where track is lost but eventually recovered. The reason is the fast head motion from frame 17 to 18, that results in the parameter vector being to far from the correct solution; how the model loses track is illustrated in Fig. 8. The recovery in frame 43 is shown in Fig. 9.

This initial test shows that the system is able to track a previously unseen person in a subjectively accurate way, but there are still quite a few issues to be addressed in the following sections:

- *Speed.* Can the system run in real-time? This is addressed in Section VI.
- *Robustness.* Can the system cope with varying illumination, facial expressions, and large head motion? Can situations like in Fig. 8 be avoided? The robustness issue is dealt with in Section VII.
- *Accuracy.* How accurate is the tracking? An evaluation method is described in Section VIII.

VI. IMPROVING THE SPEED: TOWARDS REAL-TIME TRACKING

The speed of the algorithm is of course critical for real-time applications, but an important observation is also that the algorithm works better if the frame rate is higher. When processing frames captured at a low frame rate, the algorithm suffers from being greedy and sometimes getting stuck in a local optimum. Typically, this results in the model not being able to follow fast moves, like when the mouth is closed too fast or when the head is moved away quickly. If the frame rate is higher, then the motion between each frame is smaller, and thus the initial $\mathbf{p}$ will be closer to the correct value. A few (around five to ten) frames per second seem to be sufficient for handling normal head motion.

A second observation is that when the motion between the frames is smaller, fewer iterations are usually needed, and the algorithm thus requires less time per frame.

When running the algorithm on live video, i.e., with input directly from a camera, the computation time is thus very critical. Assuming that the video input device can provide new frames as fast as the tracking

Test Sequence 1

Test Sequence 2

Figure 7: Tracking results (every 10:th frame shown).

system can process them, then a reduced computation time would increase the frame rate. As observed above, this would improve the performance of the tracking, and also decrease the average number of iterations needed per frame, which would increase the possible frame rate even more.

In conclusion, it is very important to optimize the algorithm as much as possible, and we have two principle ways of reducing the computation time; we can either reduce the computation time per iteration or reduce the number of iterations.

A. Reducing the Time per Iteration. Studying the algorithm, we find that there are three potentially time-consuming parts within each iteration:

- *The geometrical normalization:* The process of reshaping the incoming image to the standard face shape (corresponds to Eq. (7)). Using a brute force method, the image warping needs approximately 2 seconds, which is a huge amount of time for a real-time system.
- *The analysis-synthesis step*, corresponding to Eqs. (8)-–(11), is the projection of the normalized image onto the texture modes, reconstruction, and computation of the residual image and the summed residual error. This grows approximately linearly with the number of texture modes used.
- *The vector update computation*, corresponding to Eq. (12), is the computation of the update parameter vector from the residual images.

Speeding up any of these steps will of course speed up the entire algorithm. Below, three methods are studied: Using dedicated graphics hardware, pre-computation of barycentric coordinates, and using vector instructions.

Using dedicated graphics hardware: Almost all personal computers of today have a graphics card with specialized hardware for 3D graphics, including texture mapping. Observing the computation of the normalized texture $\mathbf{j}(\mathbf{i}, \mathbf{s})$, see Eq. (7), is essentially a texture mapping, it could be suitable to use the graphics card hardware for this computation. In fact, the graphics card will probably perform this operation so fast that the transfer of the geometrically normalized image from the graphics card memory to the main memory will take longer time than the geometrical normalization itself. Practical experiments show that the entire process takes a few milliseconds only.

Pre-computation of barycentric coordinates: The hardware in a graphics card is specialized for animating 3D mesh objects with texture mapped onto them and/or with different shadings and light sources. In such situations the texture coordinates are usually fixed, and the object coordinates are variable (animated).

In our situation, we have the opposite situation; the object coordinates (the normalized/destination shape) are fixed, and the texture coordinates are variable. This has an important implication: The pixel positions in the destination image are always the same in relation to the vertices of the standard-shaped wireframe model. Thus, these positions could be pre-computed, which should speed up the geometrical normalization significantly. Unfortunately, this can typically not be communicated to the graphics card, and the entire calculation has to be done in the CPU. This is not a major drawback, though, since the remaining computations can be performed very quickly in the CPU.

Using vector instructions: Exploiting the fact that modern CPUs have vector instructions for integer and floating point data, the remaining computations of the geometrical normalization together with the analysis-synthesis and the update vector computation can be performed in a total time of less than 3 milliseconds on a 500 MHz Pentium III. This time is of course dependent on the resolution, number, and format (colour or grayscale) of the texture modes. However, performing the full iterative model adaptation in 30 ms is quite reasonable. Details about the implementation can be found in Ahlberg (2002).

B. Reducing the Number of Iterations. The simplest way of reducing the number of iterations is to simply stop the computation when a certain maximum number of iterations are done. In a real-time scenario, capturing images from a camera, this does not necessarily degrade the performance of the tracker. If the alternative is to miss a captured frame, the result will probably improve.

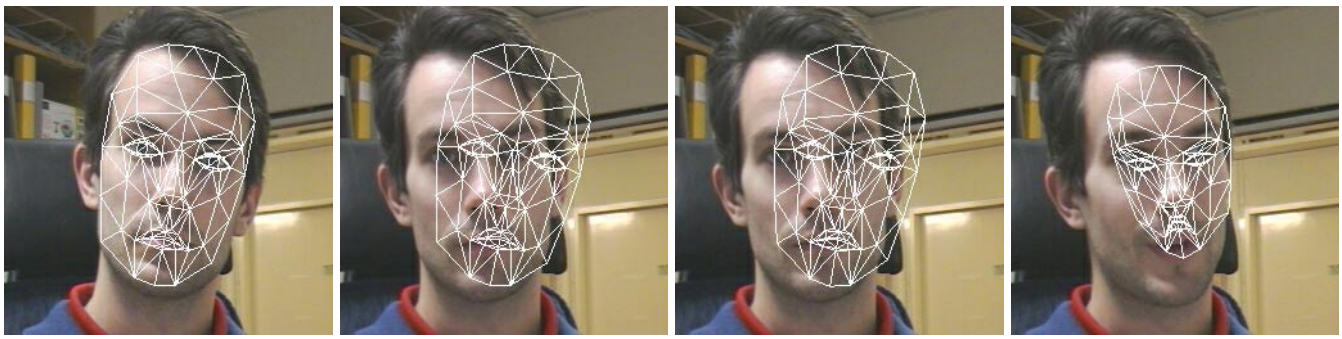
Another way of reducing the number of iterations would be to stop the process when the update vector is small enough. However, practical experiments show that none of these two methods give any dramatic improvements in terms of speed.

The number of iterations is usually smaller the shorter the distance the face has moved since the last frame. Thus, if the motion of the face could be predicted or estimated in a fast way, to give a better starting parameter vector, the number of iterations would decrease. For this to be of any use, the prediction/estimation must be faster than the time per iteration times the number of iterations earned.

For fast *motion prediction*, a simple linear filter can be applied to the model parameters from the previous frames, creating a prediction of the new model parameters. However, empirical tests using a constant speed model and a half-speed model did only show decreasing performance.

For fast *motion estimation*, a simple block matching procedure, using a few patches from the previous frame, can be used. Notice that this could also improve the robustness of the tracking; we will investigate this in Section VII.

C. A Real-time System. By using features of the hardware (the graphics card, the vector instructions of the CPU) in an efficient way and/or algorithm specific solutions, we can perform an active model iteration in a few milliseconds on consumer hardware. Typically, less than 10 iterations per frame are needed, which means that the facial feature tracking could theoretically run at about 35 Hz on the experimental platform. In practice, the video capture and transfer require some time as well, allowing the system to run at half that speed at best.



a b c d

Figure 8: Track lost due to fast head motion. a) The model adapted to frame 17. b) Frame 18 with the model still adapted to frame 17. c) The model (erroneously) adapted to frame 18. d) In frame 21 the model has lost track completely.

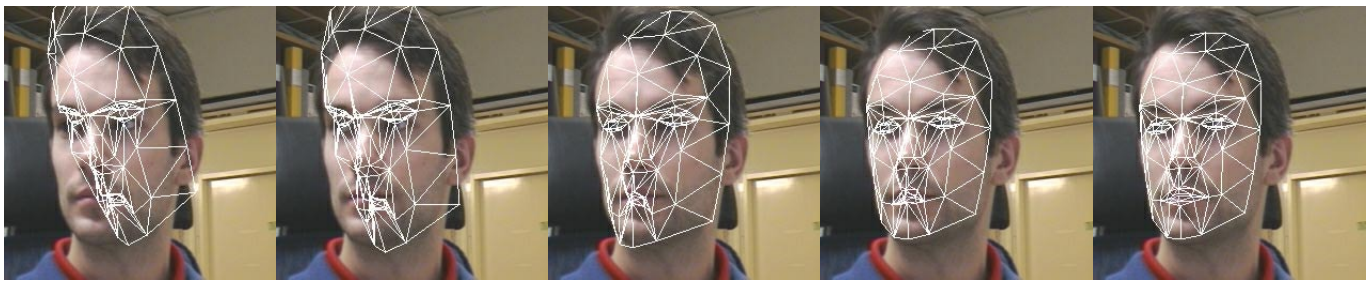


Figure 9: Tracking recovery. The images show the model adaptation in frame 43 after 0, 5, 10, 15, and 20 iterations.

Is then our main goal, to show that it is possible to implement a real-time model based coder on consumer hardware, reached? If the tracking was the only component needed, we would be there. However, depending on the system requirement, we need to add compression and transmission not only of the extracted animation parameters, but also of speech, and maybe facial texture, background, etc.

A minimal system, transmitting animation parameters only, is feasible. Converting the extracted parameters to MPEG-4 FAPs and compressing them using the MPEG-4 reference software was done in approximately five milliseconds per frame on the experiment platform. Assuming a minimal frame rate of 12.5 Hz (on a PAL system; when processing video captured by an NTSC camera, 15 Hz is more reasonable), we have 80 ms to spend on each frame. Spending 30 ms on the tracking, 5 ms on compression, and 5 ms on rendering the GUI, half of the time is available for other tasks. These other tasks would typically include video capturing, operating system tasks, and, in a communication scenario, decoding and visualization of the received face.

Also taking into account the tremendous development of consumer hardware—from the day of the initial experiment to the publishing of this paper, the CPU clock frequency of the “typical consumer hardware” was approximately quadrupled. Thus, our conclusion is that the techniques presented here should enable real-time model-based coding using current consumer hardware.

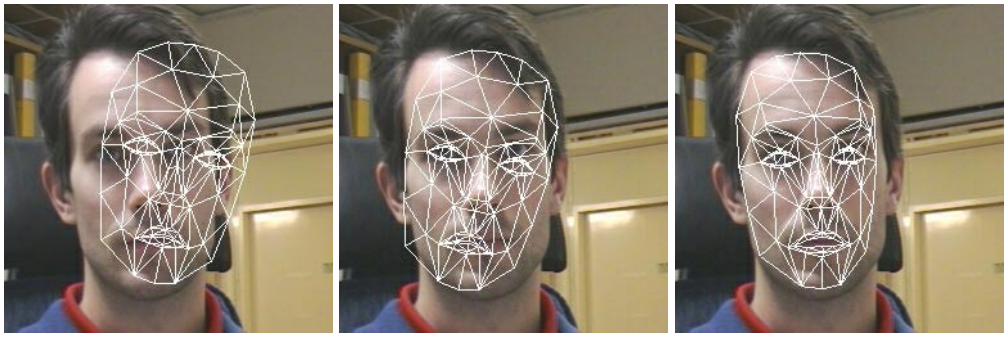
VII. IMPROVING THE ROBUSTNESS THROUGH MOTION ESTIMATION

To combine the strengths of model-based and motion-based trackers, a motion estimation algorithm could be added to our model-based tracker. The idea is to use a simple and fast block motion estimator to get a quick estimate of the global motion (at least the translation) of the face. The face model would then be translated according to this estimate, and the iterative model adaptation process start at this new position. Provided that the motion estimate is accurate, this would be advantageous in two ways:

- By doing a “jump” to a position closer to the correct position, the risk of getting stuck in a local (sub-optimal) optimum is reduced.
- Fewer iterations might be needed from the new position. If the block motion estimator is faster than the skipped iterations, the overall speed will improve.

There are a number of fast motion estimators available in the literature of video coding (Lundmark, 2001), using different difference measures, search algorithms, block shapes etc. We have chosen to implement the following simple estimator:

1. A single block is cut out of the old frame, converted to grayscale, and stored as reference block. The centre coordinate of the block is denoted **b**.
2. A search region around **b** in the new frame is converted to grayscale, subsampled, by a scaling factor



a b c

Figure 10: Using motion estimation to compensate for fast head motion. Frame 18 of Test Sequence 1 (cf. Fig. 8). a) The model still adapted to the previous frame. b) The model after motion compensation. c) The model adapted to the frame after motion compensation.

of two, twice, to create a three level scale hierarchy. The reference block is subsampled in the same way.

3. At the coarsest level, a full search within the search region is performed. The sum of absolute differences (SAD) is used as error measure, since it is quick to compute. The coordinate with the lowest SAD is stored in a variable **m** (where **m** = 0 indicates that the optimum was found at **b**).
4. The estimate is refined on the medium level, searching in a small window (3 × 3 pixels) around **m**, and then on the finest level. The face model is then translated according to the final **m**.

The choices to make are then the size of the block, the size of the search region, and which block to use as reference block. It has already been pointed out by Ström (2001) that the region between the eyes is suitable for tracking, being quite robust to changes in scale and rotation.

Running the tracker on Test Sequence 1 (that partly failed in the initial experiment) shows a good improvement when the motion estimation is used; the part with fast head motion is now correctly tracked, as shown in Fig. 10.

To introduce more tracking failures, the tracker was given every third frame only in order to simulate very fast head motion. This leads to tracking failure a few times during the sequence, but using motion estimation the whole sequence is correctly tracked.

As mentioned in the previous section, motion estimation could be used for increasing the speed. This requires of course that the motion estimation algorithm sufficiently reduces the number of iterations to compensate for the time which is spent on the motion estimate itself. Using a 64 × 64 pixel reference block and a 128 × 128 pixel search area, the motion estimation needs approximately 7 ms, that is, about as much as one iteration (when using 10 RGB texture modes of the size 40 × 42). In our test sequences, which requires only a few iterations per frame, it is not very realistic that the motion estimation could reduce the average number of iterations per frame as to compensate for the extra time. Measuring the number of iterations needed gave the results shown in Table I; apparently the motion estimation will slow down the tracking a little, but if more than 10 texture modes are used, or if they use higher resolution, the motion estimation should improve the speed.

a b c

Figure 11: Synthesizing test sequences. a) Frame 62 from Test Sequence 1 and 2 respectively. b) After tracking, the frame is synthesized by texture mapping frame 1 onto the model. c) The tracker is run on the synthetic sequences.

Table I: The average number of iterations per frame with and without motion estimation.

Iterations per frame	Sequence 1	Sequence 2
Without ME	3.04	2.38
With ME	2.21	1.86
Improvement	0.83	0.52

VIII. EVALUATION OF ACCURACY

Above, the evaluation of the tracking system has not been more formal than observing that it works "quite satisfactory". The problem with an objective evaluation is that the ground truth is not known—the only way to estimate the parameters is by using the tracker. There are other techniques for measuring face and facial feature motion, such as motion capture systems based on reflective markers or on the Polhemus sensor. However, such systems are typically somewhat inaccurate when measuring small motion, or they are expensive. In the case of marker based systems, the problem is the markers themselves; they will certainly influence the performance of the tracker described in this chapter.

A inexpensive solution which gives exact ground truth is adopted by Ström (2002), and will be used here as well. The idea is to create a *synthetic* test sequence. Using the Candide model, mapping a texture onto it, and then animating it according to some captured or semi-random motion, we get a video sequence. This may not look very life-like (we will only have a face, not an entire head, and there will be no background), but at least we know exactly the correct animation parameters. Furthermore, we can easily change the texture of the model, investigating the tracker's behaviour when tracking different persons.

In Fig. 11, a frame from the original test sequences and the corresponding frames from the synthetic test sequences are shown. The tracker has been applied to the two synthetic sequences, and the six estimated global motion parameters are compared to the ground truth. The results are plotted in Fig. 12, and the errors accounted for in Table II.

The average errors in rotation are typically less than 0.1 radians; as expected, the in-plane rotation (z-rotation) is estimated better than the out-of-plane rotation (x- and y-rotation). The average scale error for the two sequences is around 0.07, which at the resolution used for the synthetic sequences (512×512) corresponds to the model being two pixels larger/smaller. Note that the scale most of the time is estimated to be a bit smaller than the true value. This is explained by the fact that the black background influences the normalized face image near the borders, and the model thus tries to avoid the borders. In terms of translation, a parameter change of approximately 0.007 in the current scale and resolution corresponds to a one pixel motion. Consequently, the models were, on average, 3–4 pixels off track.

IX. CURRENT AND FUTURE WORK

The main problem with the tracking is the estimation of out-of-plane rotation and scale. Experience shows that these are correlated, and it is also clear that the

quality of the local parameters (mouth and eyebrows motion) reduces when the pose parameters are not correctly estimated. Typically, the pose parameter estimation fails due to the greedy search algorithm getting stuck in a local minimum. We are therefore currently developing a technique based on RANSAC (Fischler, 1981) to avoid local minima, combined with a texture consistency step to avoid the drifting that would otherwise be the result of estimating the pose by RANSAC only.

Another work item is to improve the detection of mouth closure. When animating a face from the estimated MPEG-4 FAPs together with recorded speech, it is very critical that the mouth is entirely closed at the correct points in time. For a human observer it is very disturbing if the mouth is still open by a pixel or two when a phoneme like [m] is pronounced, and it is also disturbing if the mouth stays closed for more than a very short time when a phoneme like [b] is pronounced. Since the current algorithm entirely disregard the pixel data between the lips, mouth closure is not robustly or accurately detected. Mouth-closure estimating add-ons are under investigation.

It has also turned out that the animation modes used for controlling the eyebrows are not very well suited for tracking. By pulling the parameters to somewhat extreme values, a few triangles will totally disappear (the three corner points being situated on a line), sometimes providing a local minimum for the parameter estimation. Preliminary results show that modifying these animation modes gives a noticeable improvement on eyebrow tracking.

Finally, the number of animation modes should be increased. In the initial test, six animation modes were chosen since that seemed an absolute minimum. Adding more animation modes would increase the complexity of the tracker linearly, but, as mentioned, the hardware development already have (and will even more) relax the demands on complexity.

X. CONCLUSION

We have described the design of a model-based face and facial feature tracker based on the active appearance models. Experiments have been performed to track a previously unseen person in two test sequences. The results are very encouraging; the tracker seems to work subjectively well, but can apparently fail due to fast head motion. Even if the tracker may recover spontaneously, methods for improving robustness are needed. We have thus investigated a way of improving the tracker using fast and simple motion estimation, and found that it makes the tracker significantly more robust.

The accuracy of the tracker has been measured by tracking synthetic sequences, thus having ground truth to compare to. The measurements indicate that the tracker is satisfactory accurate, however not perfect.

Methods for improving robustness and accuracy are under development.

ACKNOWLEDGEMENT

This work has been financed by the national Swedish project VISIT and the European 5:th framework programme project InterFace.

APPENDIX A. COMPUTING THE UPDATE MATRIX

When performing the model adaptation using the AAM search, we compute a residual image $\mathbf{r}(\mathbf{p})$ and an error measure

$$e(\mathbf{i}, \mathbf{p}) = \|\mathbf{r}(\mathbf{i}, \mathbf{p})\|^2 . \quad (17)$$

Our optimal parameter vector is thus

$$\mathbf{p}^*(\mathbf{i}) = \arg\min_{\mathbf{p}} e(\mathbf{i}, \mathbf{p}) \quad (18)$$

Taylor-expanding $\mathbf{r}$ around $\mathbf{p} + \Delta\mathbf{p}$, we can write

$$\mathbf{r}(\mathbf{i}, \mathbf{p} + \Delta\mathbf{p}) = \mathbf{r}(\mathbf{i}, \mathbf{p}) + \mathbf{G}\Delta\mathbf{p} + O(\Delta\mathbf{p}^2) \quad (19)$$

where

$$\mathbf{G} = \frac{\partial}{\partial \mathbf{p}} \mathbf{r}(\mathbf{i}, \mathbf{p}) . \quad (20)$$

Thus, given a $\mathbf{p}$ (and thus $\mathbf{r}(\mathbf{i}, \mathbf{p})$), we want to find the $\Delta\mathbf{p}$ that minimizes

$$e(\mathbf{i}, \mathbf{p} + \Delta\mathbf{p}) \approx \|\mathbf{r}(\mathbf{i}, \mathbf{p}) + \mathbf{G}\Delta\mathbf{p}\|^2 . \quad (21)$$

Minimizing Eq. (21) is a least squares problem with the solution

$$\Delta\mathbf{p} = -(\mathbf{G}^{\mathrm{T}}\mathbf{G})^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{r}(\mathbf{i}, \mathbf{p}) , \quad (22)$$

which gives us the update matrix $\mathbf{U}$ as the negative pseudo-inverse of the gradient matrix $\mathbf{G}$:

$$\mathbf{U} = -\mathbf{G}^{\dagger} = -(\mathbf{G}^{\mathrm{T}}\mathbf{G})^{-1}\mathbf{G}^{\mathrm{T}} . \quad (23)$$

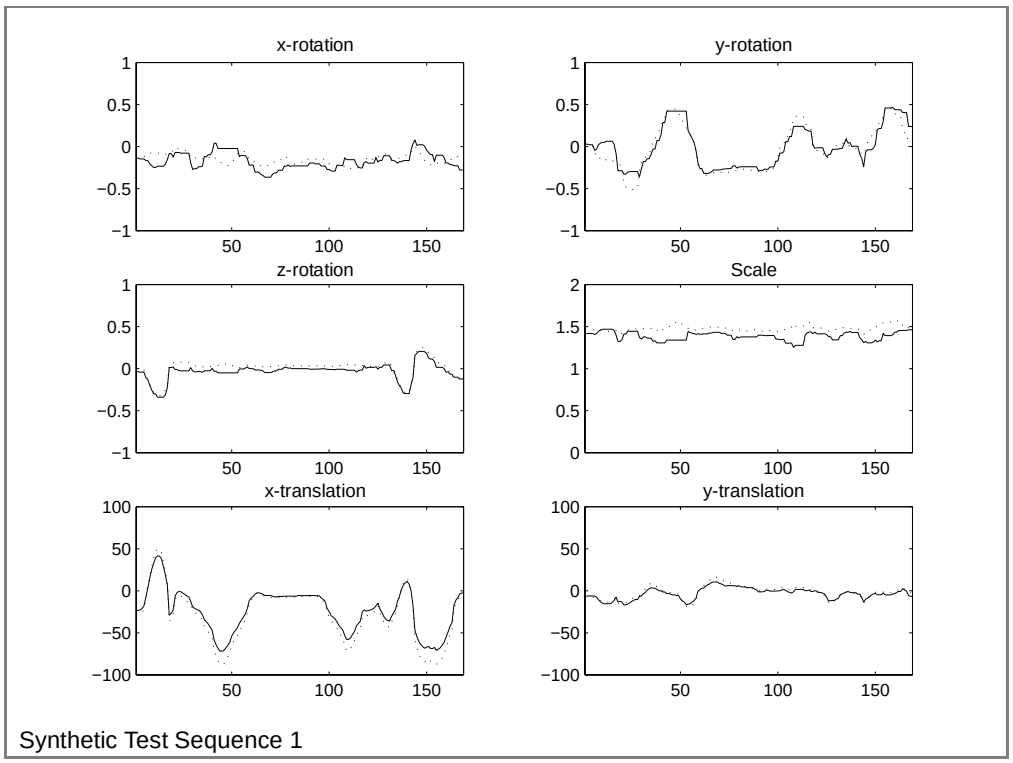
By observing that $\mathbf{G}$ is *similar even for different images* $\mathbf{i}$, we come to the conclusion that it can be estimated from training data in advance, as follows.

The j^{th} column in $\mathbf{G}$,

$$\mathbf{G}_j = \frac{\partial}{\partial \mathbf{p}_j} \mathbf{r}(\mathbf{i}, \mathbf{p}) \quad (24)$$

can be estimated using differences

$$\mathbf{G}_j \approx \frac{\mathbf{r}(\mathbf{i}, \mathbf{p} + h\mathbf{q}_j) - \mathbf{r}(\mathbf{i}, \mathbf{p} - h\mathbf{q}_j)}{2h} , \quad (25)$$



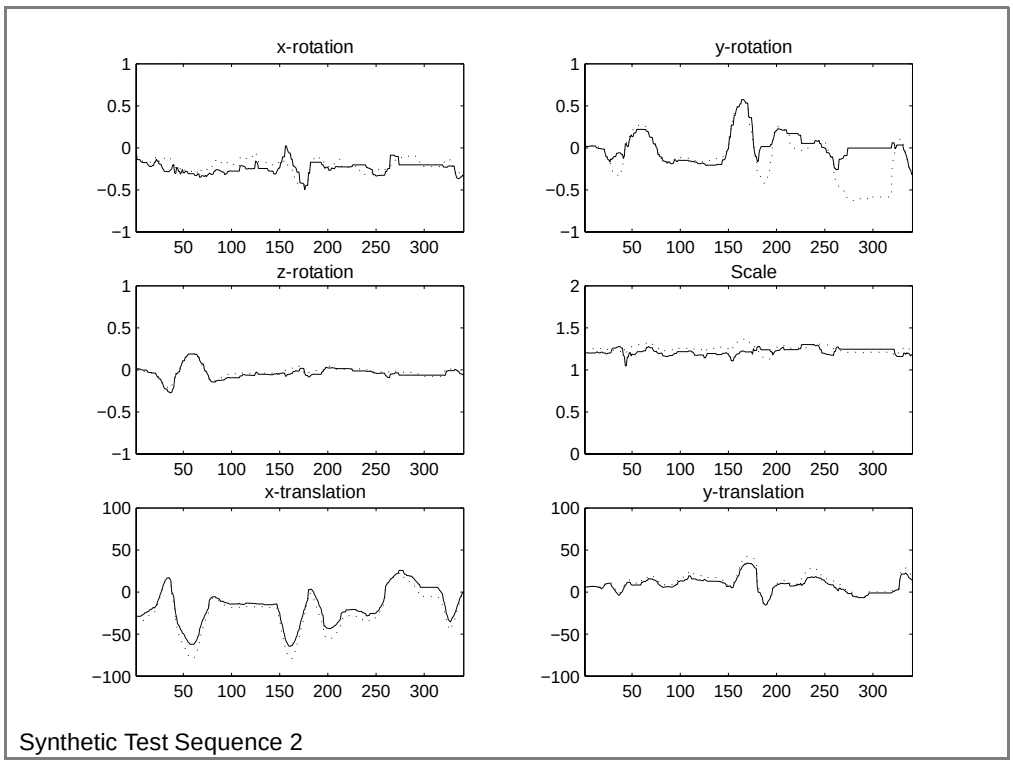


Figure 12: Tracking results on the synthetic test sequences. Frame numbers on x-axes. Solid lines: Estimated parameter values. Dotted lines: True parameter values.

where h is some suitable step size and $\mathbf{q}_j$ is a vector with all elements zero except the j^{th} element that equals one. Since we have already adapted the model to a large number N of training images $\mathbf{i}_n$, n = {1, …, N_i}, in order to compute texture modes, we have a set of corresponding parameter vectors $\mathbf{p}_n$. By estimating $\mathbf{G}_j$ for several step sizes and for all our training images with adapted models, and then averaging over all these, we get our final estimate of $\mathbf{G}_j$ as

$$\mathbf{G}_j \approx \frac{1}{NK} \sum_{n=1}^{N_i} \sum_{k=1}^{K/2} \frac{\mathbf{r}(\mathbf{i}_n, \mathbf{p}_n + kh\mathbf{q}_j) - \mathbf{r}(\mathbf{i}_n, \mathbf{p}_n - kh\mathbf{q}_j)}{2h} \quad (26)$$

Table II: Average errors in global parameter estimation.

Parameter	Sequence 1	Sequence 2
x-rotation	0.0789	0.0651
y-rotation	0.0706	0.1446
z-rotation	0.0433	0.0213
Scale	0.0942	0.0578
x-translation	0.0142	0.0232
y-translation	0.0186	0.0127

where K is the number of steps that we want to perturb the parameters in. The computation in Eq. (26) is performed for $j = \{1, ..., N_p\}$, where N_p is the number of parameters (degrees of freedom) we intend to use for the model adaptation. Deciding the values of h and K are design choices made from empirical tests.

Having thus estimated $\mathbf{G}$ and computed $\mathbf{U}$, we can search for the optimal parameter vector $\mathbf{p}^*(\mathbf{i})$ for a new image $\mathbf{i}$. For a starting value of $\mathbf{p}$, being close enough to $\mathbf{p}^*(\mathbf{i})$, we compute $\mathbf{r}(\mathbf{i}, \mathbf{p})$ and $e(\mathbf{i}, \mathbf{p})$. The update vector $\Delta\mathbf{p}$ is computed by multiplying the residual image with the update matrix:

$$\Delta\mathbf{p} = \mathbf{U}\mathbf{r}(\mathbf{i}, \mathbf{p}). \quad (27)$$

We compute a new parameter vector and a new error measure:

$$e' = e(\mathbf{i}, \mathbf{p} + \Delta\mathbf{p}). \quad (28)$$

REFERENCES

J. Ahlberg, Candide-3—un updated parameterised face, Report No. LiTH-ISY-R-2326, Dept. of Electrical Engineering, Linköping University, Sweden, 2001.

J. Ahlberg, Model-based coding—Extraction, Coding and Evaluation of Face Model Parameters, PhD Thesis No. 761, Dept. of Electrical Engineering, Linköping University, Sweden, 2002.

K. Aizawa, H. Harashima, and T. Saito, A model-based image coding system—construction of a 3-D model of a person's face, Proc Int Picture Coding Symposium (PCS), Stockholm, Sweden, 1987, paper 3.11.

K. Aizawa et al., "Human Facial Motion Analysis and Synthesis with Applications to Model-Based Coding," Motion Analysis and Image Sequence Processing, M.I. Sezan and R.L. Lagendijk (Editors), Kluwer Academic Publishers, 1993.

Y. Aloimonos, Perspective approximations, Image and Vision Computing 8 (1990), 177-192.

T.F. Cootes, G.J. Edwards, and C.J. Taylor, Active Appearance Models, IEEE Trans Pattern Anal 23 (2001), 681–684.

I. Craw, H. Ellis, and J. Sishman, Automatic extraction of face features, Pattern Recognition Letters 5 (1987), 183–187.

G.J. Edwards, T.F. Cootes, and C.J. Taylor, Interpreting Face Images using Active Appearance Models, Proc Int Conf on Automatic Face and Gesture Recognition, Nara, Japan, 1998, pp. 300–305.

P. Ekman and W.V. Friesen, Facial Action Coding System, Consulting Psychologist Press, Palo Alto, CA, USA, 1977.

M. Fischler and R. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Communication ACM 24 (1981), 381-395.

R. Forchheimer and O. Fahlander, Low bit-rate coding through animation, Proc Picture Coding Symposium (PCS), Davis, CA, USA, 1983.

R. Forchheimer, O. Fahlander and T. Kronander, A semantic approach to the transmission of face images, Proc Picture Coding Symposium (PCS), Cesson-Sevigne, France, 1984.

H. Harashima, K. Aizawa, and T. Saito, Model-based analysis-synthesis coding of video telephone images—conception and basic study of intelligent image coding, Trans IEICE E72 (1989), 452–458.

C.-H. Hjortsjö, Människans ansikte och det mimiska språket ("Man's Face and the Mimic Language," in Swedish), Studentliteratur, Lund, Sweden, 1969.

M. La Cascia, S. Sclaroff, and V. Athitsos, Fast, Reliable Head Tracking under Varying Illumination: An Approach Based on Registration of Texture-Mapped 3D Models, IEEE Trans Pattern Anal 22 (2000), 322-336.

H. Li, P. Roivanen, and R. Forchheimer, 3D Motion Estimation in Model-Based Facial Image Coding, IEEE Trans Pattern Anal 15 (1993), 545–556.

H. Li, A. Lundmark, and R. Forchheimer, Image Sequence Coding at Very Low Bitrates: A Review, IEEE Trans Image Process 3 (1994), 589–609.

A. Lundmark, Hierarchical Structures and Extended Motion Information for Video Coding, PhD Thesis No. 683, Dept. of Electrical Engineering, Linköping University, Sweden, 2001.

I.S. Pandzic, Life on the Web, Software Focus Journal 2 (2001), 52–59.

F. Parke, Parameterized models for face animation, IEEE Computer Graphics Applications Mag 12 (1982), 61–68.

D.E. Pearson, Development in Model-Based Video Coding, Proc IEEE 83 (1995), 892–906.

E. Petajan, "alterEGO: Video Analysis for Facial Animation", MPEG-4 Facial Animation—The Standard, Implementations, and Applications, I.S. Pandzic and R. Forchheimer (Editors), John Wiley & Sons Ltd, Chichester, England, 2002, pp. 269–276.

M. Rydfalk, CANDIDE, a parameterized face, Report No. LiTH-ISY-I-866, Dept. of Electrical Engineering, Linköping University, Sweden, 1987.

J. Ström, Reinitialization of a Model-Based Face Tracker, Proc Int Conf Augmented, Virtual Environments and 3-D Imaging (ICAV3D), Mykonos, Greece, 2001, pp. 128–131.

J. Ström, Model-Based Head Tracking and Coding, PhD Thesis No. 733, Dept. of Electrical Engineering, Linköping University, Sweden, 2002.

J. Ström et al., Very Low Bit Rate Facial Texture Coding, Proc Int Workshop Synthetic/Natural Hybrid Coding and 3-D Imaging (IWSNHC3DI), Rhodes, Greece, 1997, pp. 237–240.

D. Tzovaras, D. Koutsos, and M.G. Strintzis, Efficient Head Detection Based on Color Eigenfaces, Proc Int Workshop Synthetic/Natural Hybrid Coding and 3D Imaging (IWSNHC3DI), Santorini, Greece, September 1999, pp. 148–151.

B. Welsh, Model-Based Coding of Images, PhD dissertation, British Telecom Research Lab, 1991.

C.S. Wiles, A. Maki, and N. Matsuda, Hyperpatches for 3D Model Acquisition and Tracking, IEEE Trans Pattern Anal 23 (2001), 1391–1403.