MIT Media Laboratory, Perceptual Computing Technical Report #440
Appears in: Proceedings of ICCV'98, Bombay, India, January 4-7, 1998

Mixtures of Eigenfeatures for Real-Time Structure from Texture

Tony Jebara, Kenneth Russell and Alex Pentland
Perceptual Computing, M.I.T. Media Laboratory
Massachusetts Institute of Technology
20 Ames Street, Cambridge, MA 02139
{ jebara, kbrussel, sandy } @media.mit.edu

Abstract

We describe a face modeling system which estimates complete facial structure and texture from a real-time video stream. The system begins with a face tracking algorithm which detects and stabilizes live facial images into a canonical 3D pose. The resulting canonical texture is then processed by a statistical model to filter imperfections and estimate unknown components such as missing pixels and underlying 3D structure. This statistical model is a soft mixture of eigenfeature selectors which span the 3D deformations and texture changes across a training set of laser scanned faces. An iterative algorithm is introduced for determining the dimensional partitioning of the eigenfeatures to maximize their generalization capability over a cross-validation set of data. The model's abilities to filter and estimate absent facial components are then demonstrated over incomplete 3D data. This ultimately allows the model to span known and regress unknown facial information from stabilized natural video sequences generated by a face tracking algorithm. The resulting continuous and dynamic estimation of the model's parameters over a video sequence generates a compact temporal description of the 3D deformations and texture changes of the face.

1 Introduction

Several approaches have been proposed for the recovery of 3D structure from 2D imagery. These methods include shape from shading [1], structure from motion [2] [4], stereo techniques, geometric modeling and other variants. We investigate the use of an appearance based technique for the recovery process. This approach models the correlation between the texture of a face and its 3D structure from a training set of 3D laser scanned faces. The learning process uncovers statistical correlations between these two forms of data directly. Thus, we can rely on probabilistic learning to abstract the complex solutions often required for finding shape from low-level approaches.

One issue that arises when using statistical methods is that the interesting relationships in the data are typically difficult to uncover in the presence of unwanted variations and non-linearities. Large changes due to pose, illumination, and irrelevant features can easily dominate the data. Therefore, we propose normalizing these external factors by performing 3D alignment, segmentation and color correction on the faces. This preprocessing is applied to both scanned 3D head models and to the video stream using a 3D face tracking algorithm. Thus, we generate a new representation of the data by stabilizing it using vision and alignment. This limits its variations so that the interesting correlations between texture and structure can be stably recovered and modeled by statistical learning.

The statistical model we introduce is a mixture of feature eigenspaces. Traditionally, a single eigenspace is formed over the whole data set (of dimensionality M). However, if the number of samples (N) in this set is relatively small, the model may not have enough flexibility (at most N parameters) and the N samples may not be sufficient to adequately train it. Thus, we propose training a multitude of eigenspaces, each specializing over different dimensions of the data, and then merging their results. This allows us to have a more flexible model which has an intermediate complexity between that of the eigenspace (N) and the original representation (M).

To maximize the generalizability of the model we develop an iterative algorithm for partitioning the dimensions or the features given by the original representation. The method iterates by optimizing feature partitioning over a cross-validation data set and then computing the corresponding eigenspace models on the training set. This algorithm is demonstrated to converge to a modular feature decomposition which permits the mixture of eigenfeatures to generalize to the cross-validation set of faces. The consequent model is thus capable of superior generalization.

2 System Overview

The proposed system is depicted in Figure 1. Automatic face detection is used to find a face which is then tracked in 3D (at 30 Hz). A 3D normalization is used to warp the texture data of the image into a canonical configuration. This texture data is then processed by a statistical model of the correlations between facial texture and structure to estimate a full 3D facial model. The dashed line shows the possible feedback of the estimated 3D shape model to the 3D normal-

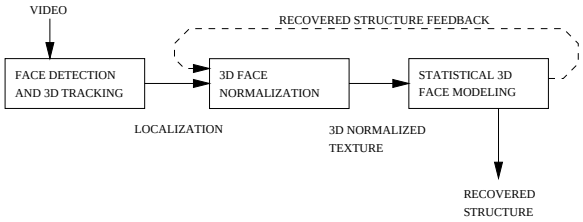


Figure 1: The Integrated System Components: Detection, Tracking, Normalization and Modeling

ization stage which could be made adaptive.

We begin by describing the pre-processing and representation of the data used for training. Our statistical model and its iterative optimization are then presented. We discuss the application of the statistical model to incomplete data and demonstrate its ability to filter known and estimate unknown components. A video tracking system is then discussed and its input to the modeling stage is explained. The combined modeling and tracking system is evaluated to show the resulting structural estimation from a video stream.

3 3D Data Preprocessing

3.1 Training Data

The system begins by off-line learning of the relationships between structure and texture from a set of N texture mapped 3D facial models which are obtained from a Cyberware laser range scanner. This data is to be vectorized so that it can be treated as a sample of a random vector and modeled probabilistically. In addition, we need to remove certain variations in the model so that its parameters only span deformations and texture changes. Thus, pose changes, illumination and irrelevant data must be factored out and modeling resources must be focussed on the correlation between texture and 3D structure.

The Cyberware data is comprised of a cylindrical depth map and an RGB texture map. A scalar value is specified at each pixel coordinate, (h, θ), to indicate the voxel's radial distance from the cylindrical axis. Three scalar values are specified to indicate the pixel's red, green and blue color components, (R, G, B). To represent this data as a vector, we simply rasterize it.

3.2 Alignment and Segmentation

For each of the models in the training database, the coordinates of the eyes, nose and mouth were manually selected as anchor points to align the 3D heads with a rigid transformation. Once aligned, the faces can be automatically cropped to remove most of the irrelevant data such as the hair and shoulders.

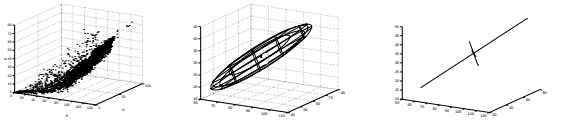
3.3 Color Correction

Color correction is performed to shift an RGB image into a canonical configuration so that illumination and camera chromatic variations are removed from the modeling process. This operation is referred to as histogram fitting for gray-scale images. Given a destination gray scale image, I_d, and a source gray scale image, I_s, one can compute a mapping for each of the intensities in I_s. We compute the cumulative density functions for each image $c_d(i)$ and $c_s(i)$ and map the intensities of I_s via the transfer function in Equation 1.

$$T_{s \to d}(i) = c_d^{-1}(c_s(i)) \tag{1}$$

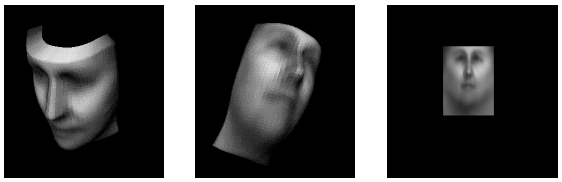
To apply this process to a color image, one can treat the R,G and B components independently as one dimensional distributions and compute histogram fitting on each. However, for skin colors, R,G and B components are not independent. Instead, we compute a Gaussian pdf over the RGB samples of the face. The eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ that span this distribution decorrelate the data (see Figure 2) and histogram fitting along these vectors under independence assumptions is more justifiable. Thus, the RGB values of each face are mapped into canonical distributions.

The average training face is shown after preprocessing in Figure 3 both in its cylindrical form as well as its 3D rendered form. The effects of alignment, hair, shoulders, global illumination and global chromatic changes have been reduced so that these do not consume significant modeling resources later on.



(a) RGB Data (b) Gaussian (c) Eigenvectors

Figure 2: The Eigenspace Basis for Histogramming



(b) Rendered (c) Rendered (d) Cylindrical

Figure 3: The Mean Face in Canonical Configuration

4 The Model

4.1 Mixture of Eigenfeatures

We now introduce our mixture of eigenfeatures statistical model and describe how it differs from a traditional eigenspace. Typically, principal components analysis is used in an unsupervised learning setting to find the dominant eigenvectors that best span the variations in the data. However, we note that PCA is sensitive to scaling of different features or dimensions of the data.

A data set is modeled with a Gaussian (which defines an eigenspace) in Figure 4(a). The eigenspace in Figure 4(a) was then recomputed over the same data after it was stretched vertically by a factor of 4 and generated the Gaussian in (b). Figure 4(c) displays the rescaled eigenvectors from each of the 2 Gaussians superimposed. Note how the eigenvectors rotated after we scaled the vertical dimension by a factor of 4.

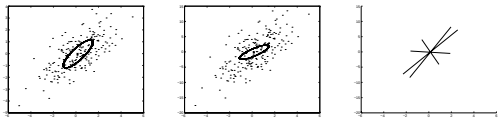


Figure 4: Eigenvector Rotation due to Scaling

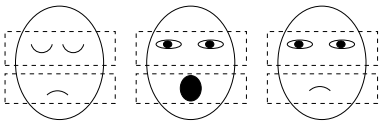


Figure 5: Modular Eigenspace Span

At the extreme, certain dimensions can be discarded (i.e. by scaling by zero) and the eigenspace will be computed only over a portion of the dimensionality of the original data.

Dimensional scaling helps reduce the impact of certain features or eliminates them altogether. This is often done manually as a pre-processing step to limit an eigenspace so that it only spans the interesting components of the data and to favor more significant dimensions. For instance, Moghaddam and Pentland [8] use modular eigenspaces to compute PCA only over small windows centered at the eyes, the nose and mouth instead of the whole face. This prevents the eigenvectors from wasting modeling resources to span irrelevant features such as hair. Eliminating features also alleviates the sparse data problem since fewer parameters (i.e. elements of a smaller covariance matrix) are to be estimated from N samples.

In addition, a modular eigenspace allows the combination of different instances of the modules to occur even though the training set did not contain any of them originally. Take for example, the training data consisting of two faces in Figure 5. One face has closed eyes and a closed mouth, and the other has open eyes and an open mouth. To generate a face with a closed mouth and open eyes would be straightforward using modular eigenspaces. However, this change in the data is not feasible in the original full space since no linear combination of the original 2 full dimensional vectors can generate it. Thus, the modular eigenspace can be a superset of the single eigenspace.

The mixture of eigenfeatures used is composed of "softly" modular eigen spaces. Dimensions are scaled for each module by a value between $[0, 1]$ to artificially weight different features and rotate the eigenvectors in their favor. In our training set $\{\mathbf{x}\}$ we have N M-dimensional vectors, i.e. $\mathbf{x}_i$ with $i\epsilon[1, N]$. Without of loss of generality, the training set is assumed to be zero-mean (simply by subtracting the average face in Figure 3). We form J eigenspaces by computing the PCA on $\{\mathbf{x}\}$ weighted by the vector $\mathbf{w}_j$ ($j\epsilon[1, J]$ is the label of the modular eigenspace). Each eigenspace is computed from a weighted covariance matrix as in Equation 2. The reconstruction of a face from the mixture of eigenfeatures is performed by summing the weighted approximations of each eigenspace as in Equation 3. We next introduce a technique for training such a model which will automatically recover the eigenvectors for each module as well as a set of partitioning weight vectors (i.e. the modular eigenspace's partition) which will be used to scale the dimensions of the training set.

$$\begin{aligned} V_j \Lambda^j {V_j}^T &= [\mathbf{e}_1^j ... \mathbf{e}_N^j] \mathrm{diag}([\lambda_1^j ... \lambda_N^j]) [\mathbf{e}_1^j ... \mathbf{e}_N^j]^T \\ &= \textstyle\sum_{i=1}^{N} (\mathrm{diag}(\mathbf{w}_j)\mathbf{x}_i)(\mathrm{diag}(\mathbf{w}_j)\mathbf{x}_i)^T \end{aligned} \quad (2)$$

$$\hat{\mathbf{x}}_i = (\sum_{j=1}^{J} \mathrm{diag}(\mathbf{w}_j))^{-1} (\sum_{j=1}^{J} V_j {V_j}^T \mathrm{diag}(\mathbf{w}_j)\mathbf{x}_i) \quad (3)$$

4.2 An Iterative Algorithm for Mixtures of Eigenfeatures

The mixture of eigenfeatures has two sets of parameters: the individual modular eigenspaces V_j and the weight vectors $\mathbf{w}_j$ which represent the domain or features spanned by each eigenspace. We propose an iterative algorithm which will converge the model to a local minimum of error by alternately taking steps to minimize the error. The algorithm alternates by updating the modular eigenvectors and then updating the feature weighting vectors of each module. If each of these constrained steps in the model configuration space increases likelihood (i.e. decreases error), then their iterative application will also increase likelihood and the system will converge to a local minimum.

An eigenspace model has the capacity to perfectly span the training data if all the eigenvectors are utilized. However, if the set of faces presented to the algorithm is split into a training set and a cross-validation set, we can check how well an eigenspace is performing on data that it has not encountered during training (and thus won't be perfectly spanned). Each eigenspace is constructed from the training samples and then evaluated on the cross validation samples. The external set of samples tests the eigenspace's ability to generalize to new and previously unseen instances of the random vector we are modeling. We wish to optimize the residual error as the mixture of eigenspaces attempts to span the cross validation (via Equation 3).

The iterative algorithm starts with in an initial configuration of the mixture of eigenfeatures model. This overall configuration is composed of a set of J covariance estimates of the data (or equivalently sets of eigenvectors V_j) and a corresponding set of J weight vectors $\mathbf{w}_j$ used to partition the pixels.

$$E_1 = \sum_{i \,\epsilon\, Training\ Data} (\mathbf{x}_i - \hat{\mathbf{x}}_i)^T (\mathbf{x}_i - \hat{\mathbf{x}}_i) \quad (4)$$

$$\frac{dE_1}{d\mathbf{V}_j} = \sum_{i \,\epsilon\, Training\ Data} (\mathbf{x}_i - \hat{\mathbf{x}}_i)^T \frac{\mathrm{d}\hat{\mathbf{x}}_i}{\mathrm{d}\mathbf{V}_j} = 0 \quad \forall\, j \quad (5)$$

In step 1 of the optimization, we wish to minimize the reconstruction error over the training data (Equation 5) by varying the eigenspaces V_j and holding the weight vectors $\mathbf{w}_j$ fixed. The best estimate of the V_j could be solved by taking the derivatives with respect to the error and setting them to zero. If we assume that the $\mathbf{w}_j$ are binary and do not overlap (i.e. not softly modular, or a winner-takes all approach) in the dimensional partitioning, the eigenvectors that minimize error can be computed from the covariance matrix given by Equation 2. This covariance matrix is the maximum likelihood estimate from the scaled segmented data under the assumption that the process being observed is a zero-mean Gaussian. This approximation is computationally efficient and is acceptable as long as the modular eigenspaces do not overlap significantly.

Next we optimize the error over the cross-validation data set by holding the V_j constant and varying $\mathbf{w}_j$. Ideally, this would be done by computing the partials of the error with respect to each $\mathbf{w}_j$ and setting them to zero as shown in Equation 7.

$$E_2 = \sum_{i \, \epsilon \, Cross \, Validation} (\mathbf{x}_i - \hat{\mathbf{x}}_i)^T (\mathbf{x}_i - \hat{\mathbf{x}}_i) \quad (6)$$

$$\frac{dE_2}{d\mathbf{w}_j} = \sum_{i \, \epsilon \, Cross \, Validation} (\mathbf{x}_i - \hat{\mathbf{x}}_i)^T \frac{\mathrm{d}\hat{\mathbf{x}}_i}{\mathrm{d}\mathbf{w}_j} = 0 \quad \forall \, j \quad (7)$$

However, if we again assume that the weight vectors are not softly modular, a simple, efficient heuristic approach can be used to calculate the weight vectors. To compute the update in the partitioning or the weighting vectors of the model, we use a winner-takes-all strategy. We first compute each modular eigenspace's individual reconstruction of the cross-validation data as in Equation 8. What we wish to do is to determine which modular eigenspace is best at reconstructing a given pixel coordinate in the cross-valiation set. When we find the modular eigenspace which does best at this pixel, we increase its weight vector (and decrease that of the other modular eigenspaces). This allows this modular eigenspace to dominate the mixture (Equation 3) at that pixel. Consequently, the reconstruction at that pixel will be closer to the true data, decreasing error or increasing likelihood.

$$\hat{\mathbf{x}}_i^j = \mathrm{diag}(\mathbf{w}_j)^{-1} V_j V_j^T \mathrm{diag}(\mathbf{w}_j) \mathbf{x}_i \quad (8)$$

$$\mathbf{w}_j(d) = \begin{cases} 1 \text{ if } j = \sup_{\mathcal{J}} \sum_{i=1}^{N} (\hat{\mathbf{x}}_i^{\mathcal{J}}(d) - \mathbf{x}_i(d))^2 \\ \epsilon \text{ otherwise} \end{cases} \quad (9)$$

Thus, during training, the algorithm assumes that the modular eigenspaces are not soft and that strict boundaries in the dimensions exist. In other words, the eigenspace which best approximates pixel d over the cross-validation data takes ownership of that pixel. We end up with a calculation of $\mathbf{w}_j$ in Equation 9. However, at each iteration, we re-inject soft modularity by using a regularizer and smoothing. Note that the modules are never set to 0 for any value of d but rather $\epsilon < 1$. The value ϵ is a small regularizer which allows the modules to always have some minimum support over the full M dimensional space (Equation 9). In addition, at each iteration, a small spatial blur operation is applied to the masks (i.e. the weight vectors) to insure soft mixing near the boundaries. The softness allow perceptually appealing reconstructions since no artifacts or edges are visible at the boundaries of the modular eigenspaces. We update the model iteratively until the system converges to a local minimum of error. To evaluate the performance of the mixture of the eigenfeatures, we use the approximation of Equation 3. The squared residual error between the $\hat{\mathbf{x}}_i$ and $\mathbf{x}_i$ is summed over the whole cross-validation set to determine the total error.

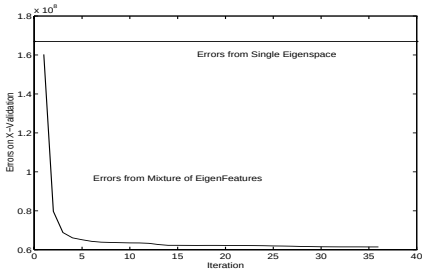


Figure 6: Cross-Validation Error over Iterations

4.3 Initialization and Convergence

The optimization can be initialized randomly as well as manually via a user-specified prior set of $\mathbf{w}_j$. We tested a random and manually specified partition involved segmenting the eyes, nose, mouth and face regions. We then looped the algorithm over a set of faces which were split into training and cross-validation sets. The performance on the cross-validation set is displayed in Figure 6. Note the rapid convergence which is achieved in less than 10 iterations (i.e. a few minutes). At each iteration, the algorithm decreases the cross-validation error until a final solution is obtained for $\mathbf{w}_j$ as well as the corresponding eigenfeatures. The algorithm converged to a modular space whose reconstruction outperformed that of the single eigenspace, a random set of modular eigenspaces and the manually generated set of modular eigenspaces. The modular eigenfeatures and their weight vectors or masks are displayed in Figure 7 at initialization ((a)-(d)) and convergence ((e)-(h)).

We used the masks to form 4 modular eigenspaces (of 20 eigenvectors each) and a single eigenspace (also of 20 eigenvectors) over the training set and crossvalidation set. These two were tested on an unobserved test set of faces to compute the squared residual reconstruction errors: 3.26e+08 for the single eigenspace and 1.97e+08 for the modular eigenspace which performed better. Figure 8 shows some of the eigenvectors (eigenheads) that were generated in both the single and one of the modular spaces.

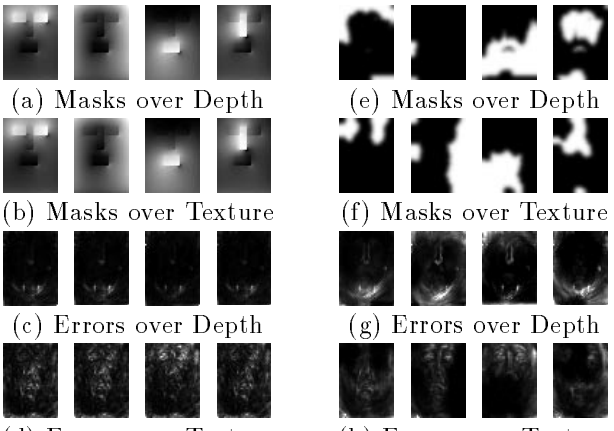


Figure 7: Initial and Final Solution

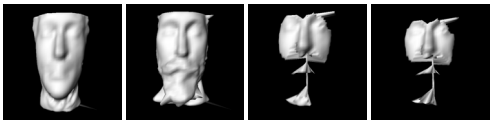


Figure 8: Single and Modular Eigenvectors (+ Mean)

5 Model Application and Evaluation

We now describe three useful applications of the model: compact representation, filtering and regression.

5.1 Compact Projection and Filtering

Recall that each 3D face can be represented as a set of coefficients for each of the eigenspaces requiring $J \times N_{pca}$ parameters where $N_{pca} < N$ is the number of eigenvectors we keep for each eigenspace. These coefficients are computed as in Equation 10 for each sample i using each module j.

$$c_i^j = {V_j}^T \mathrm{diag}(\mathbf{w}_j)\mathbf{x}_i \tag{10}$$

Reconstruction is performed by summing the eigenvectors scaled by these coefficients. Essentially, the equivalent of Equation 3 is evaluated. This process projects data into the mixture of eigenspaces and then reconstructs it. In so doing, the algorithm filters out unusual data that was not spanned by the original distribution and stores the data efficiently.

5.2 Regression

Having formed a statistical model of the joint space of structure and texture, it is now possible to condition on either one of the two to obtain a maximum likelihood estimate of the other. Thus, if only texture is observed, we can compute the probability distribution of the structure component of the random variable. In fact, any component of the data can be regressed from evidence in other dimensions. Typically, sophisticated non-linear local models are used to compute structure from 2D images. We propose a comprehensive statistical model which utilizes global properties of the face and merges various redundant cues to predict each depth map value. Thus, a simple global estimation of structure is favored over a complex local computation.

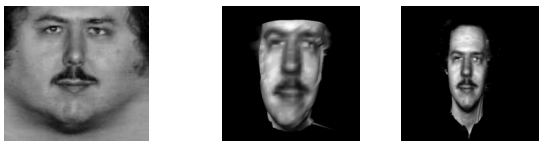


Figure 9: Regressing 3D Data from Observed Texture

$$p(x_k, x_u) = \frac{exp\{-\frac{1}{2}\begin{bmatrix} x_k \\ x_u \end{bmatrix}^T \begin{bmatrix} \Sigma_{kk} \Sigma_{uk} \\ \Sigma_{uk} \Sigma_{uu} \end{bmatrix}^{-1} \begin{bmatrix} x_k \\ x_u \end{bmatrix}\}}{(2\pi)^{D/2} \sqrt{\begin{vmatrix} \Sigma_{kk} \Sigma_{uk} \\ \Sigma_{uk} \Sigma_{uu} \end{vmatrix}}} \tag{11}$$

$$\begin{aligned} p(x_u|x_k) \propto \quad & exp\{ -\tfrac{1}{2}(x_u - \Sigma_{uk}\Sigma_{kk}^{-1}x_k)^T \times \\ & (\Sigma_{uu} - \Sigma_{uk}\Sigma_{kk}^{-1}\Sigma_{ku})^{-1} \times \\ & (x_u - \Sigma_{uk}\Sigma_{kk}^{-1}x_k) \} \end{aligned} \tag{12}$$

$$x^* = \begin{bmatrix} x_k \\ \hat{x_u} \end{bmatrix} = \begin{bmatrix} x_k \\ \Sigma_{uk}\Sigma_{kk}^{-1}x_k \end{bmatrix} \tag{13}$$

Given a zero-mean Gaussian pdf as in Equation 11 we can split the data into known (x_k) and unknown (x_u) components as shown. The conditional density of the unknown components given the known ones is displayed in Equation 12. The optimal linear estimate of the unknown data (x^*) is given by the conditional mean as in Equation 13. For a mixture of weighted eigenspaces, we can find the eigenspace regression for each space individually using Equation 13, reconstruct the scaled projections from there and sum them as in Equation 3.

Thus, from known data, we can estimate unknown facial components. This is demonstrated in Figure 9. Note the ability to regress unknown 3D facial structure from only the texture map of the 3D model. We shall now describe a system for automatically generating these canonical texture components of the data from a live video source. Ultimately, we should be able to apply a similar regression to a video image if we stabilize and generate a normalized facial texture. This texture could then be used to obtain a 3D model as shown above.

6 Tracking and Texture Stabilization

The face tracking system used to stabilize the 3D texture map is composed of two stages: an automatic detection system and a closed loop feedback tracking

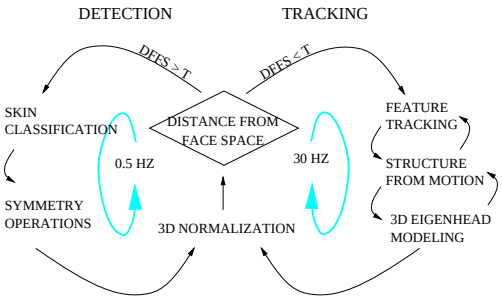


Figure 10: Face Tracking System

system as displayed in Figure 10. Note the fast face tracking loop and the slower face detection loop. The system switches between these two modes using eigenface measurements. If the object being tracked is a face, tracking continues. However, if the object being tracked is not face-like, reliable face detection is used to search the whole image for a new face (see [7]).

6.1 Facial Feature Detection

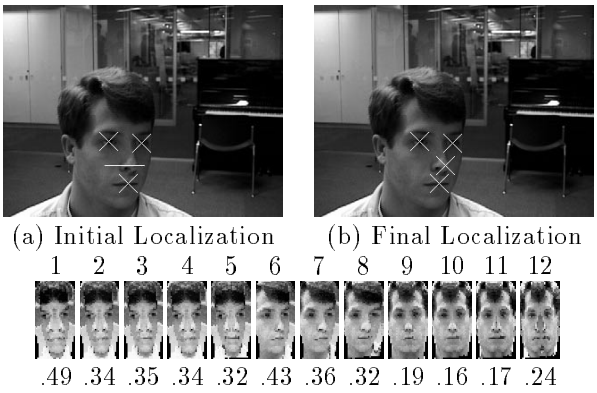
Human skin forms a dense manifold in RGB color space which makes it an easy feature to detect in images. We obtain multiple training samples of skin and compute a mixture of Gaussians on the RGB values (using EM). When a new image is acquired, the likelihood of each pixel is evaluated using the pdf to label the pixel as skin or non-skin. Then, a connected component analysis is used to group skin pixels into a large skin region.

Using the detected skin contour, a window can be defined which is expected to contain the eyes. We then use the dark symmetry transform [3] [9] [6] as an eye detector. This is an annular sampling region which detects perceptually significant edge configurations that enclose an object. Dark axial symmetry is computed from a phase and edge map by wave propagation and subsequently we compute dark radial symmetry. The peaks of dark radial symmetry are used as candidates for eye positions. Horizontal limb extraction is performed on the axial symmetry to find the mouth. Additionally, a coarse estimate for the nose's vertical location is found by searching for the strongest vertical gradient between the eyes and mouth.

At this stage, a variety of candidates have been detected as possible facial features. Several eye loci, mouth loci and many nose loci are generated. These candidates must be accurately evaluated to select the best localization.

6.2 Evaluating Localization via an Eigenspace of 3D Normalized Faces

The system considers each possible set of facial feature candidates that was generated previously. Each combination is evaluated to determine how face-like it is. This is done by aligning a 3D facial model (i.e. the average 3D Cyberware head) to the candidate facial features (which act as anchor points). The head is texture mapped and rotated into frontal view, projecting



(a) Initial Localization (b) Final Localization

(c) Mug-shots & DFFS for Trial Noses

Figure 11: Pruning Localizations

Figure 12: Initialized Correlation Trackers

a head-on mug-shot. This process (followed by the histogramming described earlier) generates a normalized mug-shot face. We can then use a 2D eigenspace of frontal faces to compute a 'distance-to-face-space'[8] and note how face-like that projection was. The more face-like a normalized mug-shot seems, the more likely the original 4 candidate anchor points corresponded to correct facial features (eyes, nose and mouth).

For instance, take Figure 11(a). The system has a candidate for two eyes, one mouth and 12 candidates for the nose (along a line). Thus, we attempt 12 different normalizations and eigenspace projections along the horizontal line across the nose's bottom (Figure 11(c)). The mug-shot with the minimal DFFS (# 10) corresponds to the best possible nose localization (Figure 11(b)).

6.3 2D Feature Tracking

We now describe the tracking algorithm which tracks the facial features as the face undergoes large 3D variations [7]. After having determined the locations of facial features in the image as explained above, it is now possible to define a number of windows on the face which will be used for template matching via SSD or normalizaed correlation [5]. Eight tracking windows are initialized on the nose, mouth corners and the eyes as shown in Figure 12. From frame to frame, a linear approximation of the behavior of the image patch under small translation, scaling and rotation perturbations can be used to recover the 2D motion of the patch.

However, the 8 trackers follow the face individually

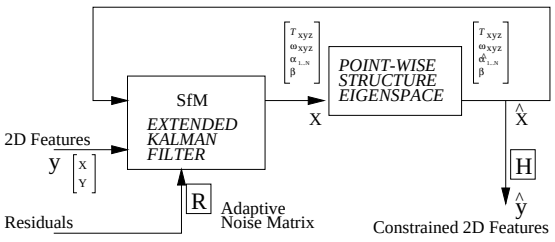


Figure 13: Constrained, Adaptive SfM

and are fundamentally spanning only 2D variations. So, they may stray off course under 3D changes and occlusion. What is desired is a global framework that overcomes some of the difficulties inherent in simple 2D tracking by coupling the individual trackers to a global 3D structure. Structure from motion is computed online as the 2D feature trackers follow the face and is then fed back to constrain their individual behavior and avoid feature loss.

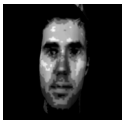
6.4 Structure from Motion

In [2], structure from motion has been reformulated into a stable recursive estimation problem and been shown to converge reliably. The SfM algorithm is implemented as an extended Kalman filter which recovers an internal state vector $\mathbf{x}$. This state vector contains the translation $(t_X, t_Y, t_Z\beta)$, focal length (β) and the incremental Euler angles $(\omega_X, \omega_Y, \omega_Z)$. The point-wise depth of each facial feature, $(\alpha_1, ..., \alpha_N)$ is also recovered in $\mathbf{x}$. The input to the Kalman filter is the 2D feature locations $(X_1, Y_1, ..., X_N, Y_N)$ at each time step. These form the observation vector, $\mathbf{y}$. Unlike some other formulations which are underdetermined at every time step, the above parameterization is well-posed and, when 7 or more 2D feature points are being tracked, a well-constrained solution can be found recursively.

For each frame, we also compute an appropriate weighting of the tracked features which depends on the current orientation and scale of the correlation trackers, their residual values and the spatial sensitivity characteristics of their corresponding texture [7]. Thus, the Kalman filter is adaptive and dynamically changes its confidence in the 2D feature tracking (via the noise covariance matrix R). In addition, the point-wise depth structure (α_i) is constrained by an eigenspace of structure formed by analyzing Cyberware depth data over the points we are tracking. So, the SfM is specialized and constrained to estimate facial structure as in Figure 13 [7].

6.5 System Integration and Feedback

We now go over the implementation details of the system integration and the feedback process. The system begins with the face detection loop and repeats until a face is detected (i.e. satisfies a threshold on distance from face-space). The facial features detected are eyes, nose and mouth. From these features, a set of template windows can be placed on the face. These



(a) Subject's 3D Scan

(b) Mean 3D Head

acquire the underlying texture and then begin tracking the face patch by minimizing the SSD error.

Simultaneously, at each iteration, the Kalman filter computes an estimate of the rigid 3D structure that could correspond to the motion recovered by the set of 2D SSD trackers. This global estimate is weighted using the noise characteristics and residuals of the individual 2D trackers. The EKF's 3D structural estimate is filtered using an eigenspace of 3D head shape. The filtered 3D structure, motion and focal length are used to projected feature points back onto the image to predict the constrained position of the 2D feature trackers in the next time step. The feedback from the adaptive Kalman filter maintains a sense of 3D structure and enforces a global collaboration between the separate 2D trackers.

In addition, at each iteration, the orientation of the face is computed and is used to warp the face image back into frontal view to compute distance to face space. If DFFS is below a threshold, tracking continues. Otherwise, the system reverts back to the initial detection stage.

7 Testing Tracking and Modeling

Figure 14 shows a real-time live video sequence where the subject was tracked stably (using an SGI O2). Initialization of the tracking took under 2 seconds and the user underwent translations, in and out-of-plane rotations and facial deformations which were tracked at 30Hz ((a)-(d)(i)). The system maintained a stabilized canonical cylindrical texture map ((a)-(d)(ii)) which was used to regress the 3D structure using the mixture of eigenfeatures. The dynamically estimated 3D structure is shown in frontal and side view ((a)-(d) (iii) & (iv)). For comparison, we also show the mean 3D face and the subject's Cyberware scanned face (the subject was scanned once into the training data) in Figure 7.

Note the model's ability to span deformations as the user opens his mouth in (b). The modularity of the eigenspace allows the mouth component to vary its local structure independently of other parts of the face. A single global eigenspace, in comparison, would require more training examples of possible combinations of mouths and faces to span the independent mouth deformation. The mouth's expression is also consistent with the slight smile in (a), the closed mouth in (c) and the mouth in (d). Some sensitivity to directional shading can be seen from the results. Note the out-of-plane rotation in (c) and in-plane rotation in (d) do not affect the estimation of structure too severely. The 3D structures estimated still bear the likeness of the subject throughout the sequence even though each structure was computed independently and no temporal integration of structural estimates is

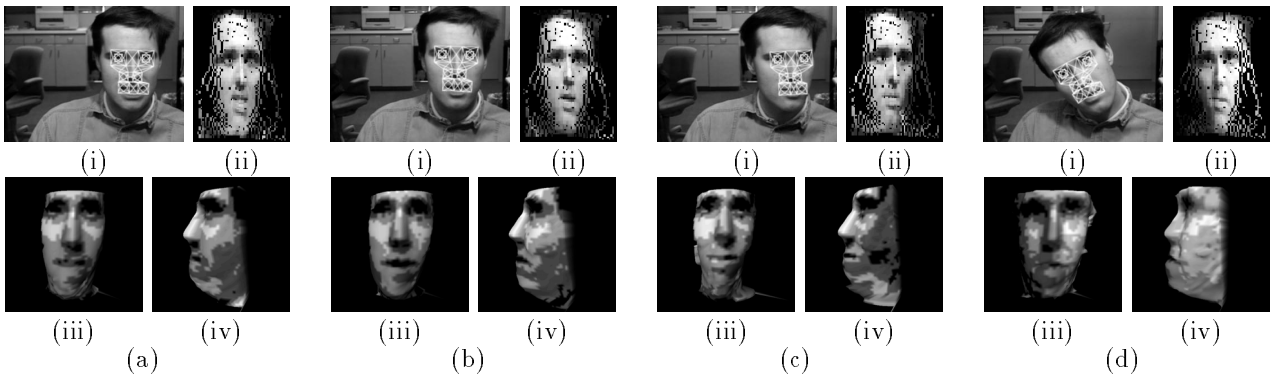


Figure 14: Real-Time Tracking and Structure Estimation from Video

being used. The independent 3D estimates of structure are being updated at 30Hz and are parametrized by 80 scalar values (the eigenspace coefficients); the 3D translation and rotation quaternions are also estimated. This permits the use of the technique for real-time very low bandwidth 3D rendering or as an input to a higher level face and gesture recognition system.

8 Future Work

Currently, we are investigating combining the structural estimate at each frame with estimates from previous frames (via a Kalman Filter). Thus, the structure and texture knowledge over the whole tracking sequence can be integrated to obtain a more accurate and temporally stable model. In addition, we are investigating feeding back the recovered facial structure into the normalization algorithm which can use it (instead of the average head's facial structure) to warp the face into canonical configuration. This could yield a higher fidelity reconstruction as the normalization and the 3D structural estimate simultaneously converge to the true face model from the video sequence. We are also accumulating a larger database which should yield better eigenvectors and more structural deformation and detail (only 30 Cyberware heads were used for the displayed results).

9 Conclusions

We have demonstrated a fully integrated face tracking and modeling system. The system utilizes a statistical model of the correlations between facial structure and texture to regress unknown components of the face from those that are currently visible in the video image. The face tracking system maintains a real-time stabilized cylindrical texture map of the face in the image. This is done by automatically detecting the facial features and tracking them using closed loop structure-from-motion feedback. A statistical modeling system based on a mixture of eigenfeatures is proposed and trained using an iterative algorithm. It is then used to regress the 3D structure from the given stabilized texture. This generates a full 3D model from the image frame which is a compact description of the facial structure and deformations and dynamically updates in real-time.

Acknowledgements

Thanks to Jill Smith at Headus, Jennifer Whitestone at Wright-Patterson Air Force Base, Brian Corner and Bob Kinney at Natick Army Labs and Tom Minka for their collaboration.

References

[1] J. Atick, P. Griffin, and N. Redlich. Statistical approach to shape from shading: Reconstruction of three-dimensional face surfaces from single two-dimensional images. *Neural Computation*, 8, 1997.

[2] A. Azarbayejani and A. Pentland. Recursive estimation of motion, structure and focal length. *IEEE Pattern Analysis and Machine Intelligence*, June 1995.

[3] M. Bolduc, G. Sela, and M. Levine. Fast computation of multiscalar symmetry in foveated images. In *Proceedings of the Conference on Computer Architectures for Machine Perception*, pages 2–11, 1995.

[4] D. DeCarlo and D. Metaxas. Deformable model-based face shape and motion estimation. In *International Conference on Automatic Face and Gesture Recognition*, pages 146–160, October 1996.

[5] G. Hager and P. Belhumeur. Real time tracking of image regions with changes in geometry and illumination. In *CVPR96*, pages 403–410, 1996.

[6] T. Jebara. *3D Pose Estimation and Normalization for Face Recognition.* McGill Centre for Intelligent Machines. McGill University, 1996. Bachelor's Thesis.

[7] T. Jebara and A. Pentland. Parameterized structure from motion for 3d adaptive feedback tracking of faces. In *CVPR97*, pages 144–150, 1997.

[8] B. Moghaddam and A. Pentland. Probabilistic visual learning for object detection. In *ICCV95*, pages 786–793, 1995.

[9] D. Reisfeld and Y. Yeshurun. Robust detection of facial features by generalized symmetry. In *ICPR92*, pages I:117–120, 1992.