Department of Electrical and Computer Engineering
Faculty of Sciences and Technology
University of Coimbra

Active Appearance Models for Facial Expression Recognition and Monocular Head Pose Estimation

Pedro Alexandre Dias Martins
pedromartins@isr.uc.pt

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

June 2008

Supervisor: Dr. Jorge Batista
Department of Electrical and Computer Engineering
University of Coimbra

Abstract

Human face images can show a great degree of variability in shape and texture. This appearance variations are caused by differences between individuals, deformations in facial expression, pose and illumination changes. Model based techniques represent a very promissing approach where a model representing an identity of interest is matched with unknown data. Face models are able to mimic shape and appearance variations from a representative training set describing face characteristics with a small set of parameters.

This thesis describes work with face models with respect to facial expression analysis and head pose estimation. It is composed by main three modules, the Active Appearance Models (AAM), the Facial Expression Analysis and Recognition (FEAR) and the Monocular Head Pose Estimation.

A model-based approach for the interpretation of face images, Active Appearance Models (AAM), is described in the first module. AAM is an adaptive template matching method where the variability of shape and texture is captured from a representative training set. Building shape, texture and combined models allow the generation of a parametrized face model that fully characterize with photorealistic quality the trained faces as well as unseen. Fitting an AAM model to a target image is a nonlinear optimization problem where the texture residuals are minimized by additive updates to the model parameters. The training stage consists on learning the correlations between texture residuals and model parameters in order to build an fast and efficient Steepest Descent (SD) algorithm. Several Monte Carlo simulations were performed, evaluating AAM fitting performance over the location of the initial estimate.

The FEAR module analyzes the discrimination power of the AAM for facial expression recognition proposes. Seven different expressions of several subjects,

representing the neutral face and the facial emotions of happiness, sadness, surprise, anger, fear and disgust are analysed. The proposed solution starts by describing the human face by an AAM model. Classification is performed using two approaches: a Linear Discriminant Analysis (LDA) combined with Mahalanobis distance and a multi-class Support Vector Machines (SVM). Comparative results were shown.

The third module is an automatic framework for the human head pose estimation from single view images. The 6DOF head pose was estimated using Pose from Orthography and Scaling with ITerations (POSIT) where an anthropometric 3D rigid model is used as an approximation of the human head, combined with Active Appearance Models (AAM) for facial features extraction and tracking. The overall performance of the proposed solution was evaluated comparing the results with a ground-truth data obtained by a pose planar approach. The results show that orientations and head location were, on average, found within 2° or 1cm error standard deviations, respectively.

Acknowledgements

In first, I would like to thank Professor Jorge Batista for accepting me as his master student.

To all my laboratory colleagues and those from the Computer Vision laboratory, thanks for the help they gave me.

This master thesis was carried out at the Institute of Systems and Robotics (ISR) on the Department of Electrical and Computer Engineering (DEEC) from the Faculty of Sciences and Technology from the University of Coimbra and this work was supported by the Portuguese *Fundação para a Ciência e a Tecnologia (FCT)* with the project Advance Interaction Using Facial Information (AI_FI) - POSC/EEA-SRI/61150/2004 (web site http://aifi.isr.uc.pt).

List of Abbreviations

AAM – Active Appearance Models
API – Application Programming Interface
AR – Augmented Reality
ASM – Active Shape Models
CoG – Center of Gravity
DOF – Degrees of Freedom
FACS – Facial Action Coding System
FEAR – Facial Expression Analysis Recognition
FEM – Finite Element Methods
FFT – Fast Fourier Transform
GN – Gauss-Newton
GPA – Generalized Procrustes Analysis
HCI – Human Computer Interface
HMM – Hidden Markov Models
KKT – Karush-Kuhn-Tucker
LDA – Linear Discriminant Analysis
LM – Levenberg Marquardt
MAP – Maximum *A-Posteriori*
MLR – Multivariate Linear Regression
PCA – Principal Component Analysis
PDM – Point Distribution Model
PCR – Principal Component Regression
PnP – Perspective-n-Point
POS – Pose from Orthography and Scaling
POSIT – Pose from Orthography and Scaling with ITerations

RBF – Radial Basis Function

ROI – Region Of Interest

RPY – Roll Pitch Yaw

SD – Steepest Descent

SOP – Scaled Orthographic Projection

SVD – Singular Value Decomposition

SVM – Support Vector Machines

VR – Virtual Reality

3DMM – 3D Morphable Models

Contents

List of Tables

List of Figures

Chapter 1

Introduction

For Human Computer Interface (HCI) applications face interaction is an important issue. Knowledge about face pose, i.e. position and orientation, and the ability to recognize identities, as well as the facial expression that they present, enables building smart interactive systems such as: facial expression recognition, mental state estimation, teleconference, knowledge about gaze direction, video compression, etc.

Facial expression recognition is an important example of face recognition techniques used in smart environments. However, there exist several difficulties to overcome. Human face images can show a great degree of variability in shape and texture. This appearance variations are caused by differences between individuals, deformations in facial expression, pose and illumination changes. Also, images of faces could have different backgrounds, resolutions, contrast, brightness, sharpness, and colour balance. Interpretation of such images requires the ability to understand a face, i.e. how a face looks like? which face properties change with facial expression? with pose variation? or from one person to another? The extracted information must be of some manageable size, since to process a full image requires a huge computation effort. In this work facial models were used to intirely describe faces. Generally, a face model is a system where a set of input parameters generate a face image output. This kind of models are able to entirely describe facial characteristics in a reduced model, extracting relevant face information without background interference.

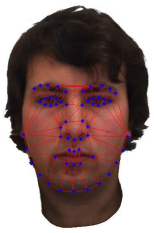
Model based techniques represent a very promissing approach where a model representing an identity of interest is matched with unknown data.

An accurate 3D head pose (position and orientation) estimation is another important issue. Reporting all the 6 Degrees of Freedom (DOF) of the head is important since provides analysis of movements in the head intrinsic coordinate system, revealing prefered directions of movement and even behavior analysis. Head pose also provides essential information, that combined with an eye tracking system, provides knowledge about where an individual is looking at, which is related to the focus of attention.

This thesis describe work with face models with respect to facial expression analysis and head pose estimation using computer vision techniques. The achievement of this work is a step through the near future smart environments, where people are recognized in order to remembering their preferences and characteristics, with the system being able to interpret facial gestures and reacts to it. These systems will provide the knowledge about the focus of attention that will lead to another step in Human Computer Interface (HCI), to answer the user needs in advance.

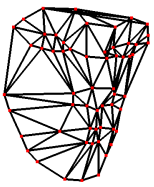
1.1 Thesis Overview

This thesis is structured in four main parts.



Part I: Active Appearance Models - Standart Active Appearance Models theory is described. Shape, texture and combined model building are analyzed in detail. Strategies to train a AAM to fit to new images are shown. Several results of model fitting are presented as well as the accuracy of the method.

Part II: FEAR Facial Expression Analysis Recognition - The representative power of AAM is used to facial expression recognition proposes. Seven different expressions, namely neutral expression, happiness, sadness, surprise, anger, fear and disgust were analysed. Two classifications approaches were compared. The first consists in projecting each expression instance, described by the AAM, into the Fisherspace and classifying-it using the Mahalanobis distance. The second consists also in projecting each appearance vector but into the hyperspace that maximize class separability using a multi-class Support Vector Machines (SVM). Comparative results are shown.



Part III: Monocular Head Pose Estimation - Refers to a single view way to estimate the human head pose (position and orientation) using the AAM model fitting as feature tracking. The 6 Degrees Of Freedom (DOF) are estimated using Pose from Orthography and Scaling with ITerations (POSIT) where a 3D anthropometric model is used as an approximation of the human head. Accuracy results from orientation and positioning are shown. Supported by an accurate pose estimation, an Augmented Reality (AR) application where a 3D virtual glasses are superimpose to the face, is proposed.

Part IV: Final Notes - Last part contains a general conclusion and discussion.

1.2 List of Publications

During the thesis the following conference proceedings where published,

- Facial Expression Recognition using Active Appearance Model,
  Pedro Martins, Joana Sampaio and Jorge Batista.
  VISAPP 2008 - International Conference on Computer Vision Theory and Applications

- Monocular Head Pose Estimation,
  Pedro Martins and Jorge Batista.
  ICIAR 2008 - International Conference on Image Analysis and Recognition

- Single View Head Pose Estimation,
  Pedro Martins and Jorge Batista.
  ICIP 2008 - IEEE International Conference on Image Processing

- Accurate Single View Model-Based Head Pose Estimation,
  Pedro Martins and Jorge Batista.
  FG 2008 - IEEE International Conference on Automatic Face and Gesture Recognition

1.3 Mathematical Notation

The mathematical notation used in this thesis is enumerated below

Vectors are represented as bold lower-case letters $\mathbf{v} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T$

Matrices are represented as bold capital letters $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

Scalar value is a small lower letter

$\mathbf{I}_n$ – is an Identity matrix, where n is the matrix dimension.

$\mathbf{1}$ – vector of ones, $\mathbf{1} = [1 \cdots 1]^T$

$\mathbf{T}$ – 3×3 transformation matrix

$\mathbf{A}^{\dagger}$ – is the pseudo-inverse matrix of $\mathbf{A}$

$tr(\mathbf{A})$ – is trace of matrix $\mathbf{A}$, i.e. $tr(\mathbf{A}) = \sum_{i=1}^{n} \mathbf{A}_{ii}$

$\Re^k$ – k-dimensional Euclidean space

$\mathbf{I}$ – represents an 2D image

$\mathbf{x} = [x, y]^T$ – two dimensional point

λ – eigenvalue

ϕ – eigenvector

ϕ_i – is used to represent the i^{th} eigenvector and λ_i his associated eigenvalue

Φ – is a matrix whose columns are eigenvectors

Λ – diagonal matrix of eigenvalues

$\overline{\mathbf{a}}$ – represents the average

σ – is the standard deviation

1.3.1 Active Appearance Models Related Nomenclature

n – number of landmark points in each shape

m – number of sampled pixels in texture

N_s – number of shapes in a training set

t_s – number of shape variation modes

t_g – number of texture variation modes

t_c – number of combined variation modes

Part I

Active Appearance Models

Chapter 2

Introduction

Active Appearance Models (AAM) is a statistical based template matching method, where the variability of shape and texture is captured from a representative training set. Principal Components Analysis (PCA) on shape and texture data allow to produce a parametrized face model that fully describe with photorealistic quality the trained faces as well as unseen.

Fitting an AAM model to a target face is a nonlinear optimization problem, where the difference in texture between the current model estimate and the target image covered by the model is minimized. Learning the correlations between texture residuals and model parameters allows to build an fast and efficient Steepest Descent (SD) algorithm based on a fixed Jacobian matrix. During this process, model parameters were repetitively being re-evaluated so that they better describe the target entity.

2.1 Related Work

Model-based deformable models that are able to fit to new data instances have great interest in computer vision. Several approaches exists. Active Contours or Snakes [28] which are energy minimizing curves, that deform according to internal and external forces. Internal forces keep the curve smooth while the external forces pull the curve towards the local image features like edges. In Active Blobs [48] the deformation is based on physical properties such as stiffness and elasticity modeled by Finite Element Methods (FEM). Active Blobs

deform a static texture whereas AAMs change both shape and texture during the fitting process. Active Shape Models (ASM) also known as Smart Snakes [56] are Point Distribution Models (PDM), i.e. landmark based methods, where the variability of shape is learned offline using Principal Component Analysis (PCA). The texture along normal scanlines is evaluated and combined with *apriori* knowledge about how shape deforms, driving the shape model to a fast and very accurate model fit.

Active Appearance Models (AAM) [58] are the natural evolution of ASM. They differs from ASM on the use of a complete model texture that covers the target. A full model of shape and texture is used to fit to new images. Several AAM extensions exist. Direct Active Appearance Models (DAM) [65] relies on noticing that one shape may contain many textures but no texture is associated with more than one shape. DAM predict shape directly from texture instance. In Constrained Active Appearance Models [54] the model matching is driven using a probabilistic framework, a Maximum *a-posteriori* formulation (MAP), allowing to include prior constraints on point positions. The Inverse Compositional Image Alignment [47], extension of the Lucas Kanade Image Alignment [8] and the Project Out [33] algorithm, transforms the AAM formulation in a true Steepest Descent (SD) method. The AAM original formulation considers that pose update is compositional, but the shape and appearance parameters update are additive. Baker and Matthews [33] showed that the shape parameters should be updated also by composition. AAM fitting is considered as an optimization of a set of image alignment warps (Piecewise Affine warps) using an efficient Gradient Descent solution for the Lucas Kanade Image Alignment. Adaptive AAM [7] [57] uses Quasi-Newton methods to update the AAM model Jacobian matrix (considered fixed on the standard formulation) improving fitting performances for individuals outside the training set.

Extensions to 3D where also proposed like 3DMM [10] or bi-temporal 3D AAMs [50]. Normally an increase in dimensionality, inevitably causes a rapid increase of data. Solutions to overcome this situations where proposed, where improvements on texture specificity [43] [22] for fast texture processing using wavelet texture [51] representation where made. Recently, Combined 2D+3D AAMs [27] used a hybrid solution that combines 2D and 3D concepts, where a

3D AAM is constrained by a 2D AAM search.

A occlusion robust [44] solution was also proposed. The solution is based on building the shape and texture model using using a Principal Components Analysis with missing data [20].

Chapter 3

Shape Model

In this section is described how to build a shape model. A shape is described as a vector of coordinates from a series of landmarks points. These shapes must be aligned to a common frame, therefor, a Generalized Procrustes Analysis (GPA) approach is used. By having the aligned data into a common mean, the dataset shape variation can be studied, analyzing its variance using a Principal Components Analysis (PCA). The shape will be modeled by a small set of parameters.

3.1 Landmarks

The shape is represented as a set of n landmarks points defined in $\Re^k$, usually in two or three dimensions. Shape is defined as the quality of the configuration of points which is invariant over the Euclidian Similarity transformation [55]. These landmark points are selected to match borders, vertexes, profile points, corners or other features that describe the shape.

Mathematically, a shape defined by n landmark points in k dimensional space is represented by a nk vector. In 2D images ($k = 2$), n landmarks $\{(x_i, y_i) : i = 1, \ldots, n\}$, define the $2n$ vector

$$\mathbf{x} = (x_1, y_1, x_2, y_2, \ldots, x_{n-1}, y_{n-1}, x_n, y_n)^T . \tag{3.1}$$

Note that no landmark connectivity information is given. Image 3.1 shows the landmark connectivity scheme used in this work. This is the same scheme

used in [34] using a total of 58 landmarks ($n = 58$) to represent the shape of a human face.

In the following sections were used images from the IMM annotated database [34].

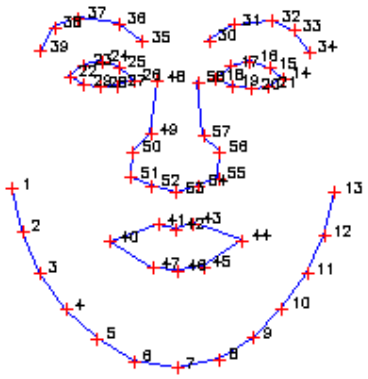


Figure 3.1: Landmarks connectivity scheme.

With N_s shape vectors $\mathbf{x}_j : j = 1, ..., N_s$, follows a statistical shape analysis on this data. In order to get statistical validity, is crucial that all shapes are represented on the same referential. The location, scale and rotation effects are removed, aligning all shapes into a common frame using the approach described below.

3.2 Align Shapes

The alignment of two shapes consists on finding the Similarity parameters (scale, rotation and translation) that best match one shape to another by minimizing a given metric. The classical solution of align two shapes is the *Procrustes* Analysis method. It align shapes with the same number of landmarks with one-to-one point correspondences, which is sufficient for the AAM standart formulation.

The alignment of multiple shapes is based on aligning pairs of shapes where one of them is a reference frame.

The alignment procedure only removes Euclidean Similarity information from the dataset, all the deformation caused by pose variation, identity and expression is retained for further statistical analysis.

3.2.1 Align Two Shapes - *Procrustes* Analysis

Aligning two shapes, $\mathbf{x}_1$ to $\mathbf{x}_2$, consists on finding the parameters of the transformation $\mathbf{T}$, i.e. scale, s, rotation, θ and translation, (t_x, t_y) that, when applied to $\mathbf{x}_1$ best aligns it with $\mathbf{x}_2$, minimizing the *Procrustes* distance metric

$$D_{Procrustes}(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{n}(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.2}$$

with respect to s, θ and (t_x, t_y).

The first step is to align their both shape centroids $\overline{\mathbf{x}}_1$ and $\overline{\mathbf{x}}_2$ by shifting the shapes to origin

$$\mathbf{x}_{1_c} = \mathbf{x}_1 - \overline{\mathbf{x}}_1, \qquad \mathbf{x}_{2_c} = \mathbf{x}_2 - \overline{\mathbf{x}}_2. \tag{3.3}$$

Notice that (t_x, t_y) parameters are given by $\overline{\mathbf{x}}_2$. Scale factor is normalized by applying an Isomorphic transformation

$$\hat{\mathbf{x}}_1 = \frac{\mathbf{x}_{1_c}}{||\mathbf{x}_{1_c}||}, \qquad \hat{\mathbf{x}}_2 = \frac{\mathbf{x}_{2_c}}{||\mathbf{x}_{2_c}||}. \tag{3.4}$$

Representing the aligned shapes w.r.t scale and translation as column vectors $\mathbf{X}'_1 = [\hat{x}_1|\hat{y}_1]_{n\times 2}$ and $\mathbf{X}'_2 = [\hat{x}_2|\hat{y}_2]_{n\times 2}$ and introducing the rotation matrix,

$$\mathbf{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.5}$$

the error difference, $\mathbf{E}$, after the rotation of $\mathbf{X}'_2$ can be written as

$$\mathbf{E} = \mathbf{R}\mathbf{X}'_2 - \mathbf{X}'_1. \tag{3.6}$$

Noticing that the trace of $\mathbf{E}\mathbf{E}^T$ equals the *Procrustes* distance in eq. 3.2, i.e.

$$tr(\mathbf{E}\mathbf{E}^T) = D_{Procrustes}(\mathbf{x}_1, \mathbf{x}_2) \tag{3.7}$$

minimizing the *Procrustes* distance is the same problem than minimizing the $tr(\mathbf{E}\mathbf{E}^T)$ with respect to rotation. The optimal rotation matrix that aligns $\mathbf{X}'_2$ to $\mathbf{X}'_1$, minimizing 3.7, is given by using [11] a Singular Value Decomposition (SVD) on matrix $\mathbf{X}'_2\mathbf{X}'_1$, where

$$SVD(\mathbf{X}'_2\mathbf{X}'_1) = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{3.8}$$

Algorithm 1 Procrustes Analysis

1: Compute the centroid of each shape: $(\overline{x}, \overline{y}) = \left(\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{1}{n}\sum_{i=1}^{n} y_i\right)$
2: Align both shapes to the origin: $(x_c, y_c) \rightarrow (x - \overline{x}, y - \overline{y})$
3: Normalize each shape by isomorphic scaling: $\hat{\mathbf{x}} = \frac{\mathbf{x}_c}{||\mathbf{x}_c||}$
4: Arrange shape matrices as $\mathbf{X}' = [\hat{x}|\hat{y}]_{n\times 2}$
5: Perform SVD$(\mathbf{X}_2'\mathbf{X}_1') = \mathbf{USV}^T$
6: The optimal rotation matrix is given by $\mathbf{R} = \mathbf{UV}^T$

and the solution is $\mathbf{R} = \mathbf{UV}^T$.

This approach is summarized in algorithm 1.

Other shape aligning approaches could also be used, for instance [55] after the aligning the shape centroids, seeks to minimize the sum of squared distance $|\mathbf{T}(\mathbf{x}_1) - \mathbf{x}_2|^2$ in the two dimensional Euclidian case

$$\mathbf{T} = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \tag{3.9}$$

where the solutions are

$$\begin{aligned} a &= \frac{\sum_{i=1}^{n} x_{1_i} \times x_{2_i} + \sum_{i=1}^{n} y_{1_i} \times y_{2_i}}{\sum_{i=1}^{n} x_{1_i} + \sum_{i=1}^{n} x_{2_i}} \\ b &= \frac{\sum_{i=1}^{n} x_{1_i} \times y_{2_i} - \sum_{i=1}^{n} y_{1_i} \times x_{2_i}}{\sum_{i=1}^{n} x_{1_i} + \sum_{i=1}^{n} x_{2_i}} \end{aligned} \tag{3.10}$$

3.2.2 Generalized *Procrustes* Analysis - GPA

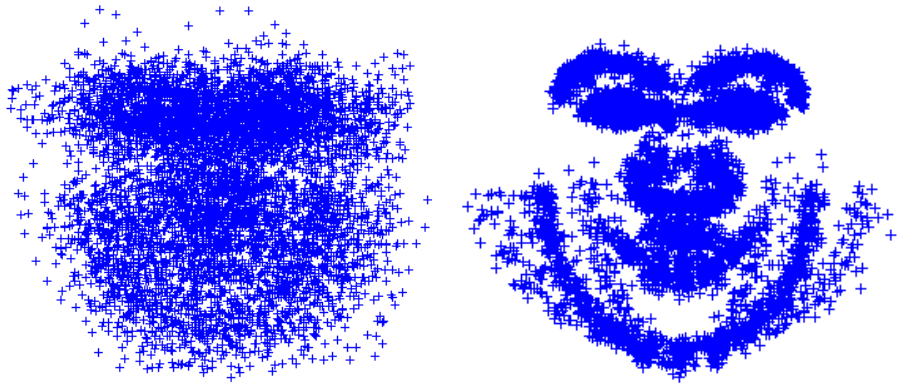
A Generalized *Procrustes* Analysis (GPA) consists in sequentially align pairs of shapes with *Procrustes* using the reference shape (the mean shape) and align the others to it. At the beginning any shape can be chosen to be the initial mean. After the alignment a new estimate for the mean is recomputed and again the shapes are aligned to this mean. This procedure is performed repeatedly until the mean shape don't change significantly within iterations. Algorithm 2 describes this iterative approach.

Normally this process converges in two iterations [11]. Image 3.2 shows results from this alignment procedure. The left image side shows the full training set raw data and the right side the correspondent aligned data.

Optionally, in order to improve the data linearity nature, the aligned distribution could be project into the tangent space but omitting this projection

Algorithm 2 Generalized *Procrustes* Analysis

1: Choose the first shape as an estimate for the mean shape $\rightarrow \overline{\mathbf{x}} = \mathbf{x}_1$
2: k=0
3: **repeat**
4: **for** Each Shape $\mathbf{x}_i, i = 1, \ldots, n$ **do**
5: Procrustes$(\mathbf{x}_i, \overline{\mathbf{x}})$
6: **end for**
7: k=k+1
8: Recompute new mean from the aligned shapes $\overline{\mathbf{x}}_k = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$
9: **until** mean converges $\rightarrow \overline{\mathbf{x}}_{k+1} \approx \overline{\mathbf{x}}_k$



(a) Raw data (b) *Procrustes* Alignment

Figure 3.2: Generalized *Procrustes* analysis results.

leads to very small changes [53].

3.3 Principal Components Analysis - PCA

The Principal Components Analysis (PCA) is a statistical technique that allows data dimension reduction. This procedure searches for directions in the data that has largest variance and subsequently project the data onto it. Mathematically is defined as an orthogonal linear transformation that projects data into a new coordinate system defined by the data variance axis. The dimension reduction is done by holding data that contribute more for the variance ignoring

remaining, less important characteristics.

Considering a dataset with N vectors: $\mathbf{x}_i : i = 1, \ldots, N$, where each $\mathbf{x}_i$ is a n dimensional vector. It is required that the number of samples is greater that the number of dimensions ($N > n$).

A PCA is performed by:

- Computing the N vectors average,

$$\overline{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i. \tag{3.11}$$

- The maximum likelihood estimation of the covariance matrix is given by

$$\mathbf{C} = \frac{1}{N-1} \sum_{i=1}^{N} (\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T. \tag{3.12}$$

  The eigenvectors, ϕ_i, and the associated eigenvalues of the covariance matrix are computed and ordered in such a way that $\lambda_i \geq \lambda_{i+1}$. The eigenvector which correspond to the highest eigenvalue, represents the direction of largest variation. The second eigenvalue corresponds to the largest variation in a direction orthogonal to the first. The subsequent eigenvectors correspond to directions orthogonal to the precedent of decreasing importance in terms of data variation. Some eigenvalues are very small, so they don't contribute much to the total variance and can be ignored. The data can be approximated as a linear combination of the most relevant eigenvectors that result in data compression.

- Setting Φ by holding the t most important eigenvalues (t is the number of modes of variation), any instance in the training set can be closer to the original data as

$$\mathbf{x} \approx \overline{\mathbf{x}} + \Phi \mathbf{b} \tag{3.13}$$

  where $\Phi = (\phi_1 | \phi_1 | \ldots | \phi_t)$ is an orthogonal matrix and $\mathbf{b}$ is a t dimensional vector that can be recovered by

$$\mathbf{b} = \Phi^{-1}(\mathbf{x} - \overline{\mathbf{x}}) = \Phi^T(\mathbf{x} - \overline{\mathbf{x}}). \tag{3.14}$$

The PCA is also known as Karhunen-Loeve transform (KLT) or the Hotelling transform.

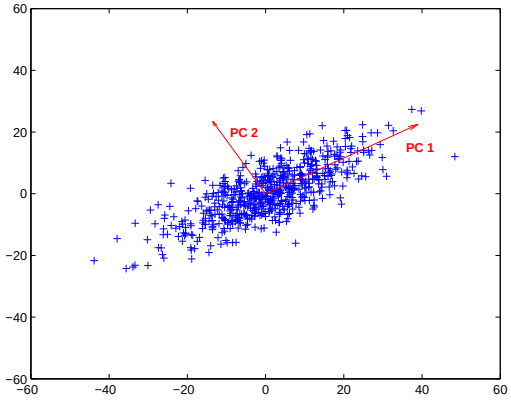


Figure 3.3: Principal components from 2D data.

Image 3.3 shows the principal components, PC1 and PC2, from a set of 2D data. Its shown that the principal componets form a new basis for data and each original point x_i can be calculated as the sum of the mean plus a linear combination of PC1 and PC2

$$x_i = \overline{x} + \sum_{i=1}^{t} \phi_i b_i. \tag{3.15}$$

3.3.1 Number of Modes of Variation

The number of modes of variation to hold, t, usually is chosen in such a way that the model represents a user defined variance of the total data. Each eigenvalue, λ_i, gives the variance of data in the direction of the correspondent eigenvector, ϕ_i, the total variance of data is given by the sum of all eigenvalues, $V_T = \sum_{i=1}^{N} \lambda_i$. The t most higher eigenvalues are chosen in order that

$$\sum_{i=1}^{t} \lambda_i \geq pV_T, \tag{3.16}$$

where p is a portion of the total variation, for instance 90% or 95%.

3.4 Statistical Shape Model

Applying a PCA on the previously aligned data, the statistical shape variation can be modeled with

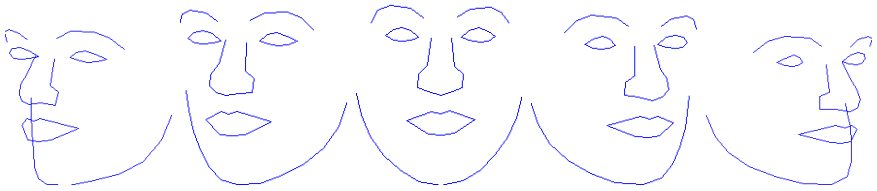
$$\mathbf{x} = \overline{\mathbf{x}} + \Phi_s \mathbf{b}_s \tag{3.17}$$

where new shapes $\mathbf{x}$, are synthesised by deforming the mean shape, $\overline{\mathbf{x}}$, using a weighted linear combination of eigenvectors of the covariance matrix, Φ_s. $\mathbf{b}_s$ is a vector of shape parameters which represents the weights. Φ_s holds the t_s most important eigenvectors that explain a user defined variance. It is possible to recover the shape parameters associated with each shape by

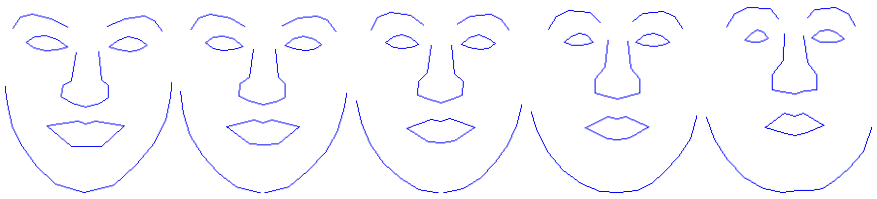
$$\mathbf{b}_s = \Phi_s^T(\mathbf{x} - \overline{\mathbf{x}}). \tag{3.18}$$

The $\mathbf{b}_s$ vector defines a set of deformable model parameters. Changing $\mathbf{b}_s$ elements leads to shape deformations.

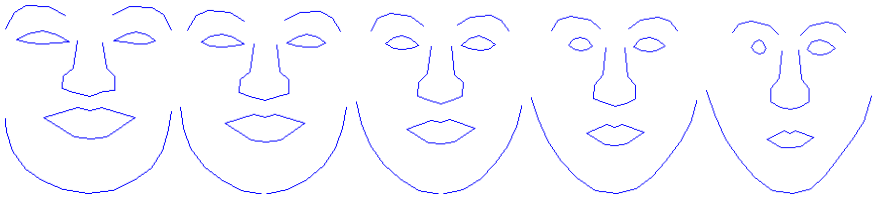
The variance of the parameter $b_{s_i} : i = 1, \ldots, t_s$ over the training set is given by λ_i. Limiting b_{s_i} between $\pm 3\sqrt{\lambda_i} = \pm 3\sigma_i$ is safe that the shape generated is similar to the ones in the training set. Image 3.4 shows the firstest five shape parameters varied between $[-3\sigma_i, +3\sigma_i]$. Holding 95% of the total variance of shape data, this model presents a total of 19 variation modes. The first variation mode, images 3.4(a-e), as expected it displays more information associated, causing a bigger movement between landmarks position. The lower significatives modes cause a more local variation. In image 3.5 is shown the decay of the total variance as function of the eigenvectors held. Note that only 30 dimensions are shown for clarity, existing a total of $2n = 116$.



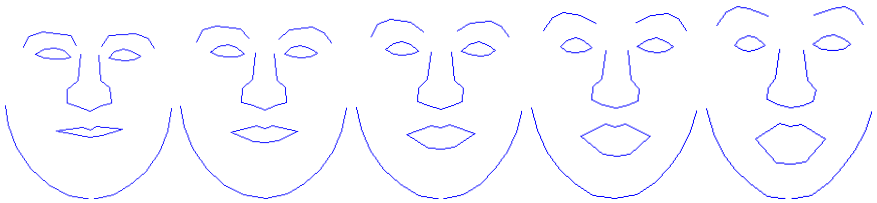
(a) $-3\sigma_1$ (b) $-1.5\sigma_1$ (c) $b_{s1} = 0$ (d) $+1.5\sigma_1$ (e) $+3\sigma_1$



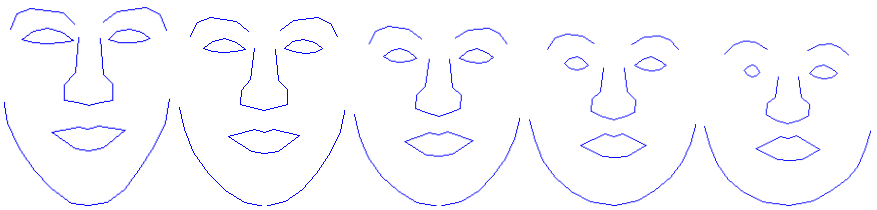
(f) $-3\sigma_2$ (g) $-1.5\sigma_2$ (h) $b_{s2} = 0$ (i) $+1.5\sigma_2$ (j) $+3\sigma_2$



(k) $-3\sigma_3$ (l) $-1.5\sigma_3$ (m) $b_{s3} = 0$ (n) $+1.5\sigma_3$ (o) $+3\sigma_3$



(p) $-3\sigma_4$ (q) $-1.5\sigma_4$ (r) $b_{s4} = 0$ (s) $+1.5\sigma_4$ (t) $+3\sigma_4$



(u) $-3\sigma_5$ (v) $-1.5\sigma_5$ (w) $b_{s5} = 0$ (x) $+1.5\sigma_5$ (y) $+3\sigma_5$

Figure 3.4: The first five shape variation modes.

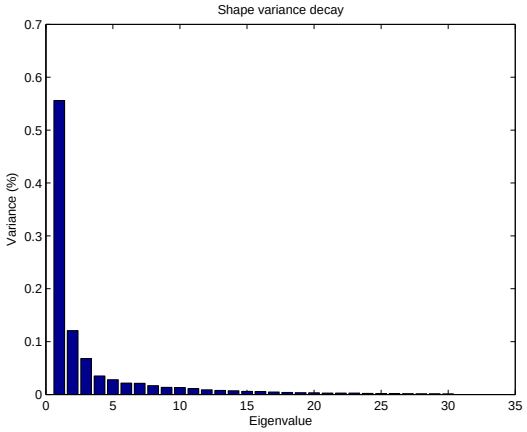


Figure 3.5: Shape variance decay as the number of eigenvalues increases.

Chapter 4

Texture Model

Building a full model from a face requires modeling shape and also the texture. In this section is described the procedures to build such a statistical texture model. Similarly to the shape model, where all the shapes are previously aligned into a common frame, the texture model requires the alignment of all texture samples to a reference texture frame also. The texture is mapped in a way that the control points from each sample match the control points of a suitable reference frame, the mean shape. Delaunay triangulation is used in the mean shape control points to establish triangles that will be used to map pixel intensities by barycentric coordinates (Piece-wise Affine Warping). A photometric normalization is used on the mapped texture samples and a statistical texture model is build using Principal Components Analysis, describing the texture in a condensed model.

4.1 Texture

In this work, the texture is defined as the pixel intensities over the modeled entity. For m pixels sampled, the texture is represented by the vector

$$\mathbf{g} = [g_1, g_2, \ldots, g_{m-1}, g_m]^T. \tag{4.1}$$

To improve texture specificity, RGB color information is used, sampling each one of the three color channels. For RBG color images the m value actually represents three times the number of sampling pixels.

Building a statistical texture model, requires warping each training image so that the control points match those of the mean shape. This procedure removes differences in texture due to shape changes, establishing a common texture reference frame.

4.2 Texture Mapping - Warping

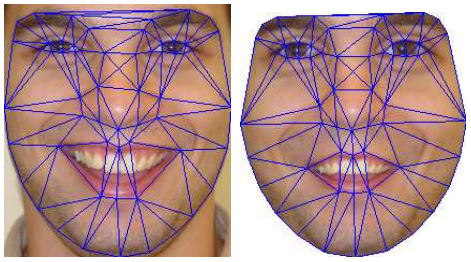
In texture mapping, the spacial configuration from one image is changed into another image. Formally $\mathbf{I} \in \Re^k \rightarrow \mathbf{I}' \in \Re^k$, where $k = 2$ in the case of 2D images. AAMs is a landmark based method where a set of control points $\{\mathbf{x}_1 \dots \mathbf{x}_n\}$ is mapped into $\{\mathbf{x}'_1 \dots \mathbf{x}'_n\}$. Each single point is represented by $\mathbf{x} = [x, y]^T$.

A mapping function, $\mathbf{f}$, can be written by

$$\mathbf{f}(\mathbf{x}_i) = \mathbf{x}'_i, \forall i = 1, \dots, n. \tag{4.2}$$

In practice, a reverse map is used, i.e. take $\{\mathbf{x}'_1 \dots \mathbf{x}'_n\}$ to $\{\mathbf{x}_1 \dots \mathbf{x}_n\}$ solving 'holes' problems.

The texture mapping is performed, using a piece-wise affine warp, i.e. partitioning the convex hull of the mean shape by a set of triangles using the Delaunay triangulation. Each pixel inside a triangle is mapped into the correspondent triangle in the mean shape using barycentric coordinates, see figure 4.1.



(a) Original (b) Warped texture

Figure 4.1: Texture mapping example.

4.2.1 Delaunay Triangulation

The triangulation of a set of points, in $\Re^2$, is a triangle network whose vertexes are the points and the triangles don't intercept themselves.

In a Delaunay triangulation, each triangle follows the Delaunay property, i.e. each triangle don't have their vertexes inside his *circumcircle* (unique circle that has all the three triangle vertexes). See figure 4.2. The Delaunay triangulation maximize the minimum angle of each triangle, trying to build as much equilateral triangles as possible.

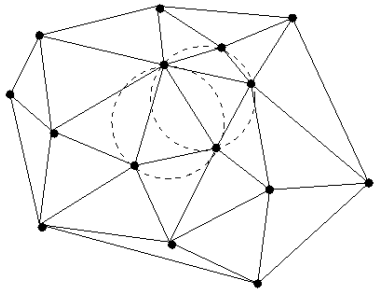


Figure 4.2: Delaunay Triangulation example.

For a set of points with concave nature, this kind of triangulation produces triangles outside the shape control points. For this cases it should be used a restrict Delaunay triangulation or Thin Plate Splines.

Figure 4.3 shows the Delaunay triangulation result on the mean shape control points. These control points will be the reference points since all the texture is processed on this normalized reference frame.

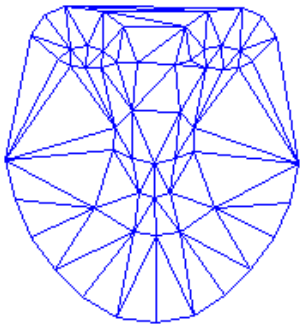


Figure 4.3: Mean shape Delaunay triangulation.

4.2.2 Barycentric Coordinates

With the mean shape Delaunay triangles, remains the question: how to known if a point belongs into a triangle? For that purpose, barycentric coordinates are used. Any point, $\mathbf{x} = [x, y]^T$, in a triangle can be defined as function of its vertexes in a way that

$$\mathbf{x} = \mathbf{f}(\mathbf{x}) = \mathbf{x}_1 + \beta(\mathbf{x}_2 - \mathbf{x}_1) + \gamma(\mathbf{x}_3 - \mathbf{x}_1) = \alpha\mathbf{x}_1 + \beta\mathbf{x}_2 + \gamma\mathbf{x}_3 \tag{4.3}$$

where $\mathbf{x}_1$, $\mathbf{x}_2$ e $\mathbf{x}_3$ are the three vertexes of a triangle, α, β e γ are real numbers that $\alpha + \beta + \gamma = 1$. The coefficients α, β e γ are named barycentric coordinates of $\mathbf{x}$ in relation to $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$.

A system of three equations with three unknowns, can be written as

$$\begin{cases} \alpha x_1 + \beta x_2 + \gamma x_3 = x \\ \alpha y_1 + \beta y_2 + \gamma y_3 = y \\ \alpha + \beta + \gamma = 1 \end{cases} \tag{4.4}$$

with solutions,

$$\begin{aligned} \alpha &= 1 - (\beta + \gamma) \\ \beta &= \frac{yx_3 - x_1y - x_3y_1 - y_3x + x_1y_3 + xy_1}{-x_2y_3 + x_2y_1 + x_1y_3 + x_3y_2 - x_3y_1 - x_1y_2} \\ \gamma &= \frac{xy_2 - xy_1 - x_1y_2 - x_2y + x_2y_1 + x_1y}{-x_2y_3 + x_2y_1 + x_1y_3 + x_3y_2 - x_3y_1 - x_1y_2} \end{aligned} . \tag{4.5}$$

To find if a point $\mathbf{x}$ belongs into a triangle with vertexes $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, its barycentric coordinates must be in range $0 \leq \alpha, \beta, \gamma \leq 1$.

4.2.3 Bilinear Interpolation

As mentioned earlier, in order to prevent holes, the texture mapping is performed using the reverse map with bilinear interpolation correction. The bilinear interpolation consists on two consecutive linear interpolations using the four pixel neighbors. Figure 4.4 highlights this procedure.

The interpolation value, $F(x, y)$, in (x, y) is function of its neighbors F_{11}, F_{12}, F_{21}, F_{22} and is given by

$$F(x, y) \approx F_{11}(1 - x)(1 - y) + F_{21}x(1 - y) + F_{12}(1 - x)y + F_{22}xy \tag{4.6}$$

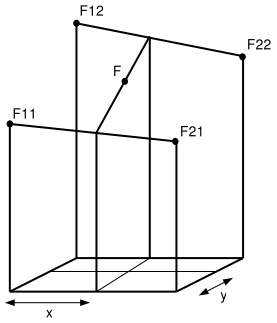


Figure 4.4: Bilinear Interpolation.

or in a matricial form

$$F(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}. \tag{4.7}$$

4.2.4 Piecewise Affine Warp

A Piecewise Affine Warp is the texture mapping procedure where each pixel from the image to sample, belonging to a specific triangle, is mapped into the respective destination triangle in the mean shape frame using barycentric coordinates with bilinear interpolation correction.

With the Delaunay triangulation result, an affine texture mapping is described in algorithm 3.

Algorithm 3 Piece-wise Affine Warp

1: **for** each pixel $\mathbf{x} = [x, y]^T$ inside the mean shape convex-hull **do**
2: Find the triangle, t, in which $\mathbf{x}$ belongs to $\Rightarrow \mathbf{x} \in t : 0 \leq \alpha_t, \beta_t, \gamma_t \leq 1$
3: Use 4.3 to get the relative position inside t'
4: Establish $\mathbf{I}'(\mathbf{x}) = \mathbf{I}(\mathbf{f}(\mathbf{x}))$ with bilinear interpolation correction
5: **end for**

Figure 4.5 displays texture mapping examples over an individual expression and pose variation. Since the convex nature of human face shape, the texture mapping should be a straightfull procedure, but across several pose variation the triangulation result goes outside the expected face. This problem is illustrated in image 4.6. As referred earlier, this problem can be overcome by using restrict Delaunay triangulation or Thin Plate Splines.

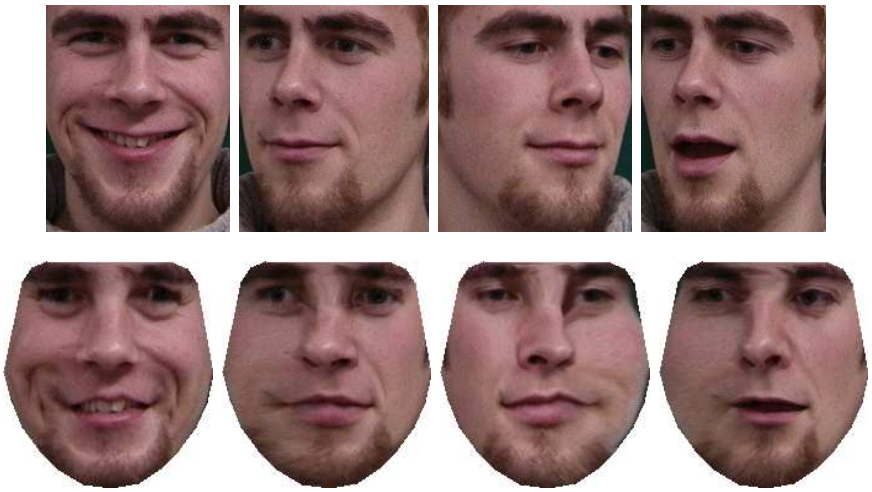


Figure 4.5: Texture mapping samples. At top, original images, at bottom warped images.

Hardware-Assisted Texture Mapping

In the AAM framework, the texture mapping is the module that requires the most computation effort. A solution based on computer graphics texture mapping was implemented using `OpenGL` [23] API, where this procedure is supported by hardware on modern graphics cards. See appendix B for details.

Table 4.1 shows a comparison of the texture mapping computation time between different implementations of Piece-wise Affine Warps. Two software approaches using `MatLab`, `C/C++` and a hardware-assisted `OpenGL` version were compared. The times shown are approximated.

Table 4.1: Texture mapping times.

	`MatLab`	`C/C++`	`OpenGL`
Time	2.7 s	200 ms	5 ms

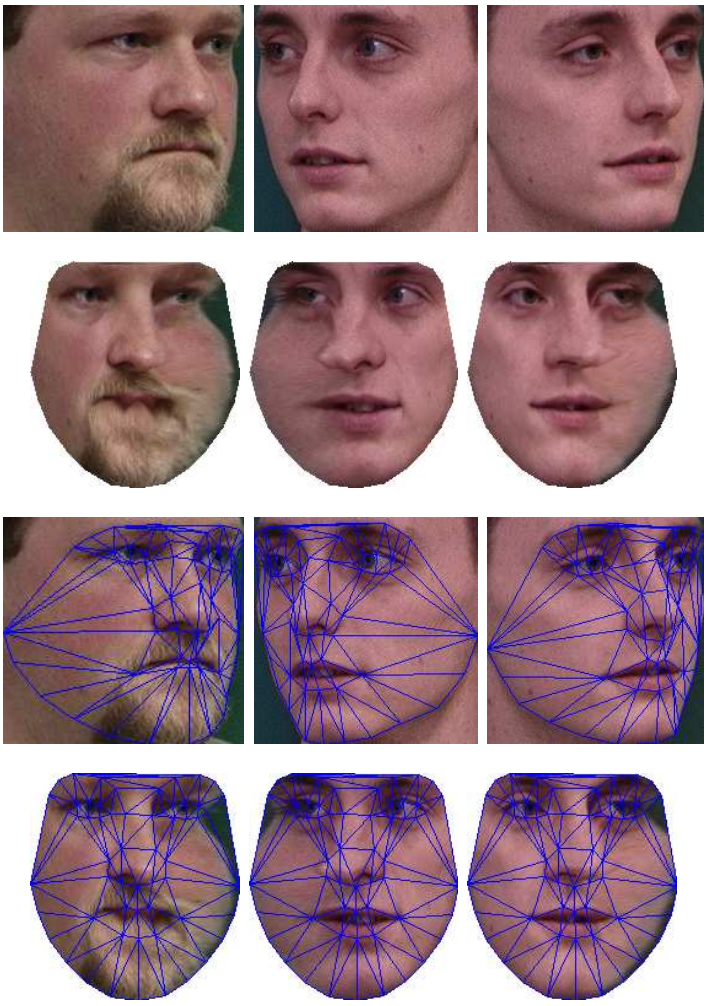


Figure 4.6: Failed warped textures that result from extreme pose variation.

4.3 Photometric Normalization

In the standart AAM formulation the influence of global lighting variations is reduced applying a scaling, α and offset, β in the texture samples,

$$\mathbf{g}_{norm} = (\mathbf{g}_i - \beta.\mathbf{1})/\alpha \tag{4.8}$$

where $\mathbf{1}$ is a vector of ones. The normalization process transform samples according to $\mathbf{g}_{norm}^T.\mathbf{1} = 0$ and $|\mathbf{g}_{norm}| = 1$.

Other possible solutions can be used. In [16], the sampled image, $\mathbf{I}$, is divided in blocks. For the central pixel of block B, the illumination interference

is normalized based on its mean μ_B and variance σ_B. Hence

$$\mathbf{I}_{norm}(i,j) = \frac{\mathbf{I}(i,j) - \mu_B}{\sigma_B}. \tag{4.9}$$

The effects of illumination differences in the current AAM framework are reduced by a histogram equalization independently in each of the three color channels [18] applied only over a rectangular face region. Figure 4.7 shows some examples of the normalized region changing the face pose.

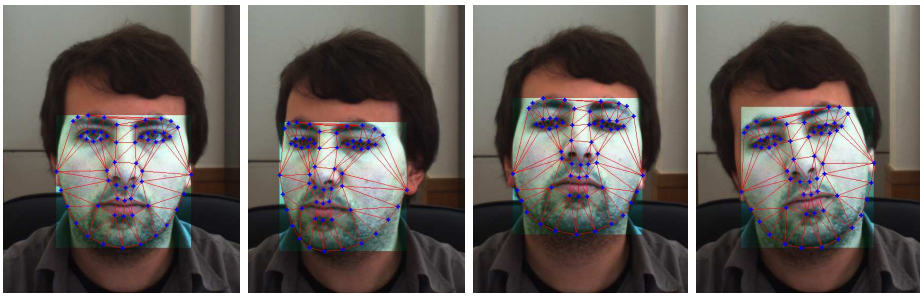


Figure 4.7: Face region histogram equalization.

4.4 Statistical Texture Model

A texture model can be obtained by applying a PCA on the normalized textures

$$\mathbf{g} = \overline{\mathbf{g}} + \Phi_g \mathbf{b}_g \tag{4.10}$$

where $\mathbf{g}$ is the synthesized texture, $\overline{\mathbf{g}}$ is the mean texture, Φ_g contains the t_g highest covariance texture eigenvectors and $\mathbf{b}_g$ is a vector of texture parameters. The texture parameters for a given sample can be retrieved by

$$\mathbf{b}_g = \Phi_g^T(\mathbf{g} - \overline{\mathbf{g}}). \tag{4.11}$$

Figure 4.8 shows the change of the firstest five texture parameters in $[-3\sigma_i, +3\sigma_i]$ interval. This texture model presents 63 modes of variation ($t_g = 63$) holding 95% of the total variation in the training set. In figure 4.9 the decay of texture variance function of eigenvectors is shown.

Similarly to shape analysis, a PCA is conducted in texture data to reduce dimensionality and data redundancy. Since the number of dimensions is greater than the number of samples, i.e is safe to say that there are more pixels than training image samples, $m >> N$, a low-memory PCA is used.

4.4.1 Low-Memory PCA

In a PCA analysis, when the number of dimensions is greater than the number of samples, it is required the computation of a covariance matrix of huge proportions ($m \times m$) that will be rank deficient. However, it is possible to perform an eigen analysis on a small matrix $N \times N$, requiring much less memory and computation effort, and obtain results statistically consistent. This approach is described in [55].

First, data matrix is build by subtracting the mean at each sample, obtaining

$$\mathbf{D} = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{g}_i - \overline{\mathbf{g}} & \cdots & \mathbf{g}_N - \overline{\mathbf{g}} \\ \vdots & & \vdots \end{pmatrix}_{m \times N}, \tag{4.12}$$

and the covariance matrix can be written as

$$\mathbf{C}_{m \times m} = \frac{1}{N-1} \mathbf{D}\mathbf{D}^T, \tag{4.13}$$

considering the matrix $N \times N$

$$\mathbf{C}'_{N \times N} = \frac{1}{N-1} \mathbf{D}^T\mathbf{D}. \tag{4.14}$$

If ϕ_i and λ_i are, respectively, eigenvectors and eigenvalues of $\mathbf{C}'$ with $i = 1, \ldots, N$ and $\lambda_i > \lambda_{i+1}$, by the Eckart-Young theorem, it can be proved that the first N eigenvectors of $\mathbf{D}\phi_i$ are eigenvectors of $\mathbf{C}$ with corresponded eigenvalues λ_i. The remaining $m - M$ eigenvectors of $\mathbf{C}$ have all null eigenvalues. Note that $\mathbf{D}\phi_i$ don't has necessarily normalized columns.

(a) $-3\sigma_1$ (b) $-1.5\sigma_1$ (c) $b_{g1} = 0$ (d) $+1.5\sigma_1$ (e) $+3\sigma_1$

(f) $-3\sigma_2$ (g) $-1.5\sigma_2$ (h) $b_{g2} = 0$ (i) $+1.5\sigma_2$ (j) $+3\sigma_2$

(k) $-3\sigma_3$ (l) $-1.5\sigma_3$ (m) $b_{g3} = 0$ (n) $+1.5\sigma_3$ (o) $+3\sigma_3$

(p) $-3\sigma_4$ (q) $-1.5\sigma_4$ (r) $b_{g4} = 0$ (s) $+1.5\sigma_4$ (t) $+3\sigma_4$

(u) $-3\sigma_5$ (v) $-1.5\sigma_5$ (w) $b_{g5} = 0$ (x) $+1.5\sigma_5$ (y) $+3\sigma_5$

Figure 4.8: Texture variation modes.

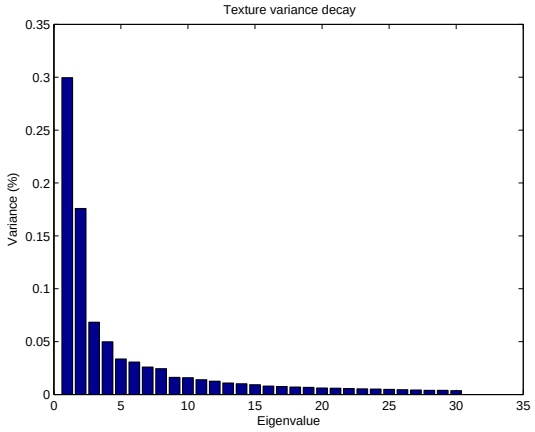


Figure 4.9: Texture variance decay as the number of eigenvalues increases.

Chapter 5

Combined Model

5.1 Introduction

This section describes how to combine both shape and texture models in a single statistical combined model. Since shape data and texture data differs in units, a way to relate the two is described. A face will be represented in a compact way, where a single vector of appearance parameters controls shape and texture. AAM model instances allow generation of faces from the training set as well as new faces.

5.2 Combined Model of Shape and Texture

The shape and texture from any training sample is described by the parameters $\mathbf{b}_s$ and $\mathbf{b}_g$. To remove correlations between shape and texture model parameters a third PCA is performed to the following data

$$\mathbf{b} = \begin{pmatrix} \mathbf{W}_s \mathbf{b}_s \\ \mathbf{b}_g \end{pmatrix} = \begin{pmatrix} \mathbf{W}_s \Phi_s^T (\mathbf{x} - \overline{\mathbf{x}}) \\ \Phi_g^T (\mathbf{g} - \overline{\mathbf{g}}) \end{pmatrix} \tag{5.1}$$

where $\mathbf{W}_s$ is a diagonal matrix of weights that measures the unit difference between shape and texture parameters. See section 5.2.1 for weight choice.

As result, using again a PCA to model statistical variation, Φ_c holds the t_c highest eigenvectors, and the combined model is obtained as

$$\mathbf{b} = \Phi_c \mathbf{c} \tag{5.2}$$

Both shape and texture parameters, $\mathbf{b}_s$ and $\mathbf{b}_g$ respectively, have zero mean, so the combined model has also zero mean.

Due to the linear nature of the model, it is possible to express shape, $\mathbf{x}$, and texture, $\mathbf{g}$, using the combined model by

$$\mathbf{x} = \overline{\mathbf{x}} + \Phi_s \mathbf{W}_s^{-1} \Phi_{cs} \mathbf{c} \tag{5.3}$$

$$\mathbf{g} = \overline{\mathbf{g}} + \Phi_g \Phi_{cg} \mathbf{c} \tag{5.4}$$

where

$$\Phi_c = \begin{pmatrix} \Phi_{cs} \\ \Phi_{cg} \end{pmatrix}, \tag{5.5}$$

and $\mathbf{c}$ is a vector of appearance controlling both shape and texture. Equation 5.6 allow retrieving the appearance parameters, $\mathbf{c}$, from a given sample.

$$\mathbf{c} = \Phi_c^T \mathbf{b} \tag{5.6}$$

Notice that, the number of modes of combined variation is less that the sum of shape and texture variation modes, i.e. $t_c \leq t_s + t_g$, due to the dimension reduction by the third PCA. Also, the rank of Φ_c never exceed the number of training images.

In the original AAM formulation [58], mainly for historical reasons, the combined model is obtained by performing three independent PCA. Concatenating shape and texture samples, properly weighted, in a unique observation matrix is possible to build the combined model through a single PCA [12].

5.2.1 Choosing Weights

The shape parameters, $\mathbf{b}_s$, have distance units, while the texture parameters, $\mathbf{b}_g$, have pixel intensity units, so they can't be compared directly. To overcome this problem a weight matrix, $\mathbf{W}_s$, that measures this unit difference is included.

A simple estimate to $\mathbf{W}_s$ is to uniformly weight with ratio, r, of the total variance in texture and shape [52], i.e. $\mathbf{W}_s$ is a diagonal matrix

$$\mathbf{W}_s = r\mathbf{I} = \begin{pmatrix} r & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & r \end{pmatrix} \tag{5.7}$$

with the weight ratio given by

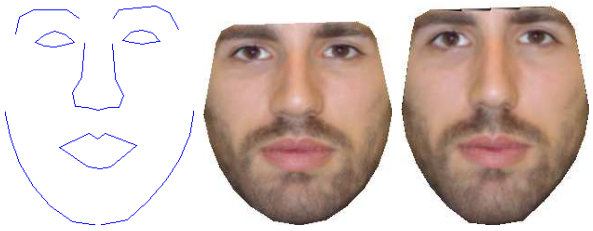
$$r = \frac{\sum_{i=1}^{N} \lambda_{g_i}}{\sum_{j=1}^{n} \lambda_{s_j}}, \tag{5.8}$$

where λ_s and λ_g are respectively the shape and texture eigenvalues.

5.3 Building an AAM Instance

An AAM instance is a synthetic image based on the combined statistical model in which the appearance vector, $\mathbf{c}$, controls both generated shape and texture. A face model with photorealistic quality is achieved.

An AAM instance is built by generating the texture in the normalized frame using eq. 5.4 and warping-it to the control points given by eq. 5.3. See figure 5.1.



(a) Shape control points (b) Texture in normalized frame (c) AAM instance

Figure 5.1: Building a AAM instance.

Figure 5.2 show a few examples of reconstructed AAM instances.

Figure 5.3 show the variation between $[-3\sigma_i, +3\sigma_i]$ in the five parameters of the combined model ($i = 1, \ldots, 5$). Holding 95% of the total combined variance the truncated model presents 15 modes of variation. Figure 5.4 shows the decay of the combined variance for each eigenvalue.

5.4 Combined vs Independent Model

Combined AAMs [58] [52] explores the intuitive relationship between the texture and shape parameters. Shape and texture is coupled in a single parametrization. A third PCA applied over shape and texture leads to a more compact and

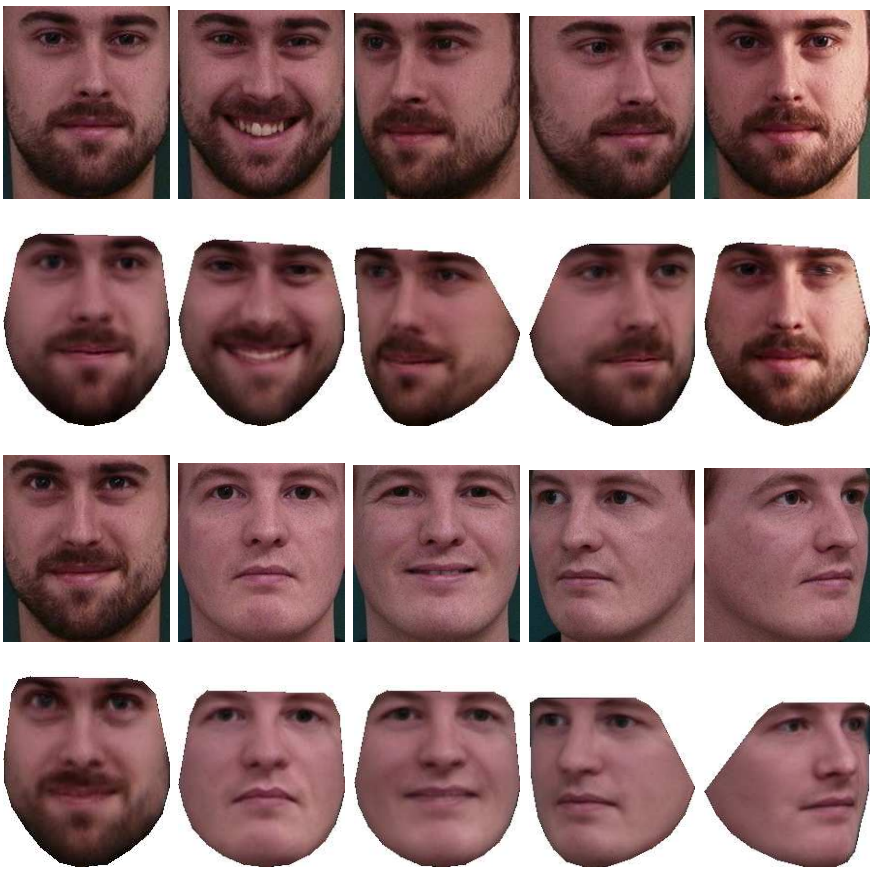


Figure 5.2: AAM instances. Original images are displayed in first and third rows. Reconstructed images are shown in second and fourth rows.

general model. The combined model formulation models the face using only a set of parameters, $\mathbf{c}$ (with $t_c < t_s + t_g$).

An independent model formulation [33] [27] process shape and texture parameters separately. An AAM independent instance is build in a similar way with respect to the combined approach. A texture generated in the mean shape by texture parametres, $\mathbf{b}_g$, is warped to the control points that result from the shape parameters, $\mathbf{b}_s$. A face is fully defined with both shape and texture parameters.

Combined AAMs have the advantage of requiring less parameters to represent the same visual phenomenon, with the same degree of accuracy, while the disadvantage that is no longer assumed that shape and texture parameters are

orthogonal, which restricts the fitting algorithm.

(a) $-3\sigma_1$ (b) $-1.5\sigma_1$ (c) $c_1 = 0$ (d) $+1.5\sigma_1$ (e) $+3\sigma_1$

(f) $-3\sigma_2$ (g) $-1.5\sigma_2$ (h) $c_2 = 0$ (i) $+1.5\sigma_2$ (j) $+3\sigma_2$

(k) $-3\sigma_3$ (l) $-1.5\sigma_3$ (m) $c_3 = 0$ (n) $+1.5\sigma_3$ (o) $+3\sigma_3$

(p) $-3\sigma_4$ (q) $-1.5\sigma_4$ (r) $c_4 = 0$ (s) $+1.5\sigma_4$ (t) $+3\sigma_4$

(u) $-3\sigma_5$ (v) $-1.5\sigma_5$ (w) $c_5 = 0$ (x) $+1.5\sigma_5$ (y) $+3\sigma_5$

Figure 5.3: Combined variation modes.

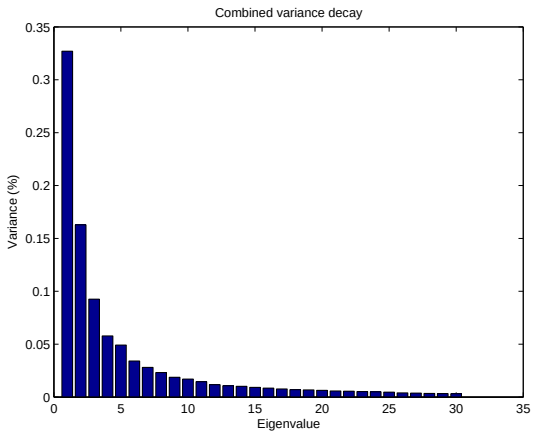


Figure 5.4: Combined model variance decay as the number of eigenvalues increases.

Chapter 6

Model Training

This section describe how to train a statistical combined model of shape and texture in order to fit entities in new images. The AAM fitting process seek to minimize texture residuals, i.e. the texture difference between the target image and the generated AAM model instance. This texture difference will drive the model on a additive parameter update scheme to better estimate a new model instance until convergence.

The training consists in learning the correlations between AAM model instances and texture residuals. In this stage, a set of experiences are required, perturbing a set of ground truth appearance parameters, that describe each model instance, by a known amount and recording the texture difference between the sampling image and the model. From this process results two huge matrices, one that holds the model perturbations and other matrix that keeps the texture residuals. With this information, two approaches to estimate a correction matrix could be applied: the Multivariate Linear Regression (MLR) and by differentiation, estimating the Jacobian matrix.

Knowing how to correct model parameters over textures residuals, a fast iterative method for matching the model to new instances can be found.

6.1 Model Training

An AAM search seeks to minimize the texture difference between a model instance and the beneath part of the target image that it covers. It can be treated

as an optimization problem where

$$\arg\min_{\mathbf{c}} \sum_{pixels} \left(\text{  } - \text{  } \right)^2 \tag{6.1}$$

or, more formally,

$$\arg\min_{\mathbf{c}} |\mathbf{I}_{image} - \mathbf{I}_{model}|^2 \tag{6.2}$$

updating the appearance parameters $\mathbf{c}$ and pose.

This nonlinear problem can be solved by learning offline how the model behaves due parameters change and the correspondent relations between the texture residual [58]. Additionally, are considered the similarity parameters for representing the 2D pose. To maintain linearity and keep the identity transformation at zero, i.e. in order to null parameters don't represent changes in pose, these parameters are redefined to

$$\mathbf{t} = (s_x, s_y, t_x, t_y)^T \tag{6.3}$$

where $s_x = (s\cos(\theta) - 1)$, $s_y = s\sin(\theta)$ represents a combined scale, s, and rotation, θ, and the remaining parameters t_x and t_y are the usual translations. The 2D similarity transformation matrix, in homogeneous coordinates, with this redefinition of parameters is

$$\mathbf{T_t} = \begin{pmatrix} 1+s_x & -s_y & t_x \\ s_y & 1+s_x & t_y \\ 0 & 0 & 1 \end{pmatrix}. \tag{6.4}$$

The complete model parameters including pose are given by concatenating the appearance parameters with the pose parameters

$$\mathbf{p} = (\mathbf{c}^T|\mathbf{t}^T)^T \tag{6.5}$$

where $\mathbf{p}$ is a $t_p = t_c + 4$ dimensional vector.

The initial AAM formulation uses the Multivariate Linear Regression (MLR) approach over the set of training texture residuals, $\delta\mathbf{g}$, and the correspondent model perturbations, $\delta\mathbf{p}$. The goal is to get the optimal prediction matrix, in the least square sense, satisfying the linear relation

$$\delta\mathbf{p} = \mathbf{R}\delta\mathbf{g}. \tag{6.6}$$

It is assumed that the correlation of texture difference and model parameters update is locally linear. Solving eq 6.6 involves perform a set s experiences, building the residuals matrices Δ_p and Δ_g

$$\Delta_p = \begin{pmatrix} \vdots & & \vdots \\ \delta p_1 & \dots & \delta p_s \\ \vdots & & \vdots \end{pmatrix}_{t_p \times s} \qquad \Delta_g = \begin{pmatrix} \vdots & & \vdots \\ \delta g_1 & \dots & \delta g_s \\ \vdots & & \vdots \end{pmatrix}_{m \times s} \tag{6.7}$$

where Δ_p holds, by columns, the model parameters perturbations and Δ_g holds the correspondent texture residuals. Computing texture residuals consists on subtracting a sampled version of the target image with the texture from the model, $\delta \mathbf{g} = \mathbf{g}_{image} - \mathbf{g}_{model}$. Sampling an image involves warping the texture that lies inside the current control points position into the reference mean shape frame. In algorithm 4 is described how to compute the residual matrices. Equation 6.6 can be extended to

$$\Delta_p = \mathbf{R}\Delta_g. \tag{6.8}$$

Remains the question of how the model parameters should be displaced. Table 6.1 shows the model perturbation scheme, suggested by [55], used in the s experiences to compute $\mathbf{R}$. In each experience, only one parameter was displaced while the remaining were set to zero. The percentage values in scale and translation are referred with respect of the size of the reference mean shape.

Table 6.1: Perturbation scheme.

Parameter $\mathbf{p}_i$	**Perturbation $\delta\mathbf{p}_i$**
$\mathbf{c}_i$	$\pm$ 0.25σ_i , $\pm$ 0.5σ_i
Scale	90%, 110%
θ	$\pm 5^o$, $\pm 10^o$
t_x, t_y	$\pm$ 5%, $\pm$ 10%

6.2 Multivariate Linear Regression

The standart AAM formulation uses the Multivariate Linear Regression (MLR) on the residuals matrices, Δ_p and Δ_g, in order to estimate, in a precomputation

Algorithm 4 Training Procedures

1: **for** Each image k **do**

2: **for** Each displacement experience j **do**

3: Get the aligned shape, $\mathbf{x}$, of image $\mathbf{I}_k$

4: Convert shape $\mathbf{x}_{2n\times 1}$ to format $\mathbf{x} = [x_{n\times 1}\ y_{n\times 1}]$

5: Recover Procrustes pose (s_x, s_y, t_x, t_y) and build transformation matrix $\mathbf{T}$, eq 6.4

6: Sample image $\mathbf{I}_k$ into $\mathbf{g}_{image}$ on the control points given by

$$\begin{bmatrix} x'_1 & \cdots & x'_n \\ y'_1 & \cdots & y'_n \\ 1 & \cdots & 1 \end{bmatrix}_{3\times n} = \mathbf{T}^{-1} \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ 1 & \cdots & 1 \end{bmatrix} \tag{6.9}$$

7: Set model perturbation $\delta\mathbf{p}$ (see table 6.1)

8: Displace model parameters with $\mathbf{p} = \mathbf{p} + \delta\mathbf{p}$

9: Generate AAM model instance with $\mathbf{p}$

10: Compute texture residual $\mathbf{r} = \delta\mathbf{g} = \mathbf{g}_{image} - \mathbf{g}_{model}$

11: Store in j column of Δ_p the model perturbation $\delta\mathbf{p}$

12: Store in j column of Δ_g the texture residual $\mathbf{r}$

13: **end for**

14: **end for**

step, the appropriate regression matrix $\mathbf{R}$ that solves the linear problem $\Delta_p = \mathbf{R}\Delta_g$.

Regarding that $m >> t_s > t_p$, one possible solution of eq 6.8 can be obtained by Principal Component Regression (PCR), projecting the large matrix Δ_g into a $k-$dimensional subspace, where $k \geq t_p$, which captures the major part of the variation. This is archived by finding the eigenvectors, $\Phi_{s\times s}$, and eigenvalues, $\Lambda_{s\times s}$, of matrix $\Delta_g^T\Delta_g$, obtaining the relationship

$$\Delta_g^T\Delta_g\Phi = \Lambda\Phi. \tag{6.10}$$

Since Φ is orthonormal, i.e. $\Phi^T\Phi = \mathbf{I}$, eq 6.10 can be written as

$$\Phi^T = \Lambda^{-1}\Phi^T\Delta_g^T\Delta_g. \tag{6.11}$$

Using Principal Components Analysis, it can be shown that Δ_g multiplied by the matrix of eigenvectors Φ is the matrix Φ_{pc} whose columns are the principal components of Δ_g

$$\Delta_g\Phi = \Phi_{pc} \tag{6.12}$$

or

$$\Phi_{pc}^T\Delta_g = \Phi^T. \tag{6.13}$$

In eq 6.13, Φ_{pc}^T projects Δ_g into a lower dimensional space. The right side of eq 6.8 is replaced by

$$\Delta_p = \mathbf{R}'\Phi^T \tag{6.14}$$

where $\mathbf{R}'$ represents the correlation between the parameter differences, Δ_p, and the texture differences, Δ_g, after projection Φ^T.

Regarding the right side of eq 6.8, eq 6.14 could be rewritten to

$$\mathbf{R}\Delta_g = \mathbf{R}'\Phi^T, \tag{6.15}$$

that combining 6.15 with 6.11 leads to

$$\mathbf{R} = \mathbf{R}'\Lambda^{-1}\Phi^T\Delta_g. \tag{6.16}$$

According to 6.14, $\mathbf{R}'$ can be expressed as

$$\mathbf{R}' = \Delta_p\Phi. \tag{6.17}$$

Finally, combining 6.16 and 6.17 the regression matrix can be computed by

$$\mathbf{R} = \Delta_p \Phi \Lambda^{-1} \Phi^T \Delta_g^T. \tag{6.18}$$

The computation of the regression matrix, $\mathbf{R}$, using a Multivariate Linear Regression solution is resumed in algorithm 5.

Algorithm 5 Computing the Regression matrix by Multivariate Linear Regression

1: Build residual matrices Δ_p and Δ_g using algorithm 4
2: Find eigenvectors Φ and eigenvalues Λ of matrix $\Delta_g^T \Delta_g$
3: The regression matrix is given by $\mathbf{R} = \Delta_p \Phi \Lambda^{-1} \Phi^T \Delta_g^T$

6.3 Jacobian

The multivariate linear regression was later replaced by a simpler approach [58], computing the gradient matrix, $\frac{\partial \mathbf{r}}{\partial \mathbf{p}}$, requiring much less memory and computational effort. This was the approach used in the current work.

The texture residual vector is defined as

$$\mathbf{r}(\mathbf{p}) = \mathbf{g}_{image}(\mathbf{p}) - \mathbf{g}_{model}(\mathbf{p}) \tag{6.19}$$

where the goal is to find the optimal update at model parameters to minimize

$$F(\mathbf{p}) = |\mathbf{r}(\mathbf{p})|^2 = \mathbf{r}^T \mathbf{r}. \tag{6.20}$$

Expanding the texture residuals, $\mathbf{r}(\mathbf{p})$, in Taylor series around $\mathbf{p}$ and holding the first order terms, results

$$\mathbf{r}(\mathbf{p} + \partial \mathbf{p}) \approx \mathbf{r}(\mathbf{p}) + \mathbf{J} \partial \mathbf{p} \tag{6.21}$$

where $\mathbf{J}_{m \times t_p} = \frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}}$ is the Jacobian matrix

$$\mathbf{J} = \frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial r_1}{\partial p_1} & & \frac{\partial r_1}{\partial p_{t_p}} \\ \vdots & \cdots & \vdots \\ \frac{\partial r_m}{\partial p_1} & & \frac{\partial r_m}{\partial p_{t_p}} \end{pmatrix}. \tag{6.22}$$

In this case

$$\begin{aligned} F(\mathbf{p}+\partial\mathbf{p}) &= (\mathbf{r}+\mathbf{J}\partial\mathbf{p})^T(\mathbf{r}+\mathbf{J}\partial\mathbf{p}) \\ &= \partial\mathbf{p}^T\mathbf{J}^T\mathbf{J}\partial\mathbf{p} + 2\partial\mathbf{p}^T\mathbf{J}^T\mathbf{r} + \mathbf{r}^T\mathbf{r} \end{aligned} \tag{6.23}$$

that by differentiating w.r.t $\mathbf{p}$ gives

$$\frac{1}{2}\frac{\partial F}{\partial \mathbf{p}} = (\mathbf{J}^T\mathbf{J})\partial\mathbf{p} + \mathbf{J}^T\mathbf{r}. \tag{6.24}$$

Setting the gradient to zero, results

$$\begin{aligned} (\mathbf{J}^T\mathbf{J})\partial\mathbf{p} &= -\mathbf{J}^T\mathbf{r} \\ \partial\mathbf{p} &= -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r} \end{aligned} \tag{6.25}$$

or

$$\partial\mathbf{p} = -\mathbf{R}\mathbf{r}. \tag{6.26}$$

The regression matrix is then given by

$$\mathbf{R} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T = \mathbf{J}^\dagger. \tag{6.27}$$

A Singular Value Decomposition (SVD) should be used on the computation on the Jacobian matrix pseudo-inverse[1], $\mathbf{R}$, improving the numerical stability.

Usually Steepest Descent (SD) approaches require the Jacobian evaluation for each iteration. Since the AAM framework works on a normalized reference frame, the Jacobian matrix can be considered fixed over the training set and its pseudo-inverse, $\mathbf{R}$, can be estimated once on the training phase.

Algorithm 6 describe the procedures to estimate the Jacobian matrix.

Algorithm 6 Computing the Regression matrix by differentiation

1: Build residual matrices Δ_p and Δ_g using algorithm 4
2: Compute the Jacobian matrix $\mathbf{J} = \dfrac{\partial \mathbf{r}}{\partial \mathbf{p}} = \Delta_g\Delta_p^\dagger$
3: The regression matrix is $\mathbf{R} = \dfrac{\partial \mathbf{r}}{\partial \mathbf{p}}^\dagger$

[1] If $SVD(A) = USV^T$ then $A^\dagger = VS^\dagger U^T$

Chapter 7

Model Fitting

Fitting an AAM model to an image is a nonlinear optimization problem where the texture residuals are minimized by updating the model parameters.

Several nonlinear solver methods can be used, such as Steepest Descent (SD) (Newton, Gauss-Newton (GN) [33], Levenberg Marquardt (LM) [48]), Powell's method, Genetic Algorithms, Simulated Annealing (SA) [17], Simplex, etc. In the previous section a regression matrix, $\mathbf{R}$, is estimated holding the information about how to correct the model parameters over texture errors. It will be shown that the model fitting procedure follows a Steepest Descent Gauss-Newton approach with a fixed Jacobian matrix that was estimated offline.

In this section is described how to fit an AAM model to a target image. Results from successful model fittings, as well as failures, will be showed. The model fitting procedure requires a rough estimate from the location of the face. The AdaBoost [62] method is used to deal with this problem. Several Monte Carlo simulations were performed, testing the quality of model fitting by perturbing initial location, comparing it with the ground truth training annotations.

7.1 Iterative Model Refinement

The model parameters are updated over texture residuals by the linear relation $\delta\mathbf{p} = \mathbf{R}\delta\mathbf{g}$. Regarding eq. 6.27 the model update becomes

$$\mathbf{p}_k = \mathbf{p}_{k-1} - \alpha(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\delta\mathbf{g} \tag{7.1}$$

which is a damped Gauss-Newtow modification of the Steepest Descent methods, where $\mathbf{J}$ is the Jacobian matrix and α is the damping factor. As described in section 6.3 the Jacobian matrix is fixed and is estimated once at the AAM training process which makes the process faster since there's no needed to compute the Jacobian for each iteration. To helps on convergence process a damping factor is also included.

Starting with a given estimate for the model, $\mathbf{p}_0$, and a rough estimate of the location of the face, an AAM model can be fitted following the algorithm 7. If no information is available, start $\mathbf{p}_0$ with zeros, which represents the mean face[1] AAM instance, and a face detector can be used to locate the face. In this work the AdaBoost [62] method its used. First the image is sampled over the location of the initial control points $[\mathbf{x}, \mathbf{y}]$ and an AAM model instance is build with the current estimate of the model parameters, $\mathbf{p}_k$. The texture residual vector, $\delta\mathbf{g}$, and the current error $E_0 = |\delta\mathbf{g}|^2 = \delta\mathbf{g}\delta\mathbf{g}^T$ are evaluated. The model parameters are updated using eq. 7.1 using a damped approach. The model is considered fitted if no improvement is made to the error or the maximum iterations value is reached.

Figure 7.1 and 7.2 shows successful AAM searches. The displayed optimization processes are done limiting the number of iterations to 15. Figure 7.3 shows samples of model fitting failure.

Assuming that the appearance parameters, $\mathbf{c}$, are independent gaussian distributions with zero mean with variance σ_i^2 that equals the λ_i eigenvalue from the combined model PCA(in the range of $\pm 3\sigma_i$ is covered 99.7% of the distribuition), a model is considered a failure if

$$-3\sigma_i \leq \mathbf{c}_i \leq 3\sigma_i. \tag{7.2}$$

The model fitting usually fails over images where the subject presents a large

[1] The mean AAM instance consists on the mean texture over the mean shape control points.

Algorithm 7 Iterative Model Refinement

1: $Iteration = 1$
2: **while** $Iteration < MaxIterations$ or no improvement is made to error E_0 **do**
3: Sample image at $(\mathbf{x}, \mathbf{y}) \rightarrow \mathbf{g}_{image}$
4: Build an AAM instance AAM($\mathbf{p}$) $\rightarrow (\mathbf{x}_{model}, \mathbf{y}_{model}, \mathbf{g}_{model})$
5: Compute residual $\delta\mathbf{g} = \mathbf{g}_{image} - \mathbf{g}_{model}$
6: Evaluate Error $E_0 = |\delta\mathbf{g}|^2 = \delta\mathbf{g}\delta\mathbf{g}^T$
7: Predict model displacements $\delta\mathbf{p} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\delta\mathbf{g}$
8: Set $\alpha = 1$
9: Update Model Parameters $\mathbf{p}_k = \mathbf{p}_{k-1} - \alpha\delta\mathbf{p}$
10: Update sample control points from $(\mathbf{x}_{model}, \mathbf{y}_{model})$ with similarity compositional pose correction $\rightarrow (\mathbf{x}_k, \mathbf{y}_k)$
11: Sample image at $(\mathbf{x}_k, \mathbf{y}_k) \rightarrow \mathbf{g}_{image_k}$
12: Compute residual $\delta\mathbf{g}_k = \mathbf{g}_{image_k} - \mathbf{g}_{model}$
13: Evaluate Error $E_k = \delta\mathbf{g}_k\delta\mathbf{g}_k^T$
14: **if** $E_k < E_0$ **then**
15: Accept model parametes, $\mathbf{p}_k$
16: Accept control points $(\mathbf{x}, \mathbf{y}) = (\mathbf{x}_k, \mathbf{y}_k)$
17: Update current error $E_0 = E_k$
18: **else**
19: Try $\alpha = 1.5, \alpha = 0.5, \alpha = 0.25, \alpha = 0.125$
20: **end if**
21: $Iteration = Iteration + 1$
22: **end while**

pose displacement, and the AAM model fit was proven to be very dependent on the initial estimate.

7.2 Initial Estimate Problem

Since the AAM requires a initial estimate to the location of the face, the better is this estimate, minor is the risk of being trap in a local minimum. Several methods for finding a face in images can be found [35]. A common method for finding the position of the face is based on color segmentation [9]. The human skin color distribution falls into a small area of the chromatic color space and can be model with a single gaussian. Applying a filter based on this gaussian, the pixels that have a similar color can be highlighted. By fitting an ellipse to this result a good approximation of the face can be found. Other solution for locate the face is to use the AAM model himself, i.e. a image to look for is downsampled and divided in subregions. AAM model fitting is performed in each subregion until the face is found.

Recently, robust appearance-based detection [63] approaches were developed. In this work face detection is achieved using a AdaBoost classifier that initializes the AAM fitting procedure.

7.2.1 Appearance-Based Face Detection

Haar-Like features are used to extract the information from the image. The detection of the objects is performed using these features as an input to an AdaBoost classifier.

Haar-Like Features

Each Haar-Like feature is represented by a template, its coordinate relative to the search window origin and the size of the feature. A subset of the features prototypes used is shown in figure 7.4. The Haar-Like features values are calculated as a weighted sum of two components: the pixel gray level values sum over the black rectangle and the sum over the whole feature area. Hundreds of features are used in a real and robust classifier. To reduce the computational time it was used the concept of Integral Image introduced in [62].

AdaBoost Classifier

The main goal is to find a very small number of these features that can be combined to form an effective classifier. To support this purpose, a weak learning algorithm is designed to select the single feature which best separates the positive and negative examples. For each feature, the weak learner determines the optimal threshold classification function, so that the minimum number of examples are misclassified.

A weak classifier $h_j(x)$ consists of a feature f_j, a threshold θ_j and a parity p_j indicating the direction of the inequality sign.

$$h_j(x) = \begin{cases} 1 & if\ p_j f_i(x) < p_j \theta_j \\ 0 & otherwise \end{cases} \tag{7.3}$$

In 7.3 the value 1 represents the detection of the object class and 0 represents a non-object. Each of these classifiers per si are not able to detect the object category. Rather, it reacts to some simple feature in the image that may be related to the object.

An approach to reduce the computational time of the detection task consists in using a cascade of classifiers. With the cascade of classifiers is possible to achieve an increased detection performance, while radically reducing the computational time. The key objective is to construct smaller, and therefore more efficient, boosted classifiers which reject many of the negative sub-windows, while detecting almost all positive instances. Simpler classifiers are used to reject the majority of sub-windows before more complex classifiers are called upon, in order to achieve low false positive rates.

Face Detection

The detection is done by sliding a search window through the frame image and checking whether an image region at a certain location is classified as a face.

Figure 7.5 shows the outputs of the detector were the face is enclosed inside the red rectangle. The AAM initial estimate consists on the mean shape scale adjusted to the AdaBoost detection.

7.2.2 Recovery From Lost Track

In certain occasions, especially when the movements of the head are particularly fast, the fitting algorithm, applied alone, loses track and is not able to recover. To overcome this problem, each time the AAM fitting fails, i.e. the condition 7.2 is satisfied, the appearance-based detection initialization process is called again.

7.3 Monte Carlo Simulations

Since the quality of the AAM model fitting is very dependent of the initial estimate location, some Monte Carlo simulations were performed, evaluating the fitting robustness procedure, perturbing the starting location.

The location is perturbed over x-axis and y-axis directions independently. Two different initial location references were used ranging from $[-50, 50]$ pixel units. One is the location given by the AdaBoost classifier, while other is the shape aligned with the Center of Gravity (CoG) of the ground truth annotations. Each AAM search experience start with the mean face model instance, i.e. $\mathbf{p}_0 = \mathbf{0}_{t_p \times 1}$, and the final fit control points are compared with the ground truth annotations.

To measure the model fitting accuracy several metrics were used. In the following equations, (x_i, y_i) represents the coordinates of the converged fitted shape and (x_{gt}, y_{gt}) is the ground truth shape annotation.

The point to point error is the average Euclidian distance from shape (x_i, y_i) to (x_{gt}, y_{gt}) and is defined as

$$D_{pt \to pt} = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(x_i - x_{gt})^2 + (y_i - y_{gt})^2}. \tag{7.4}$$

The point to border error is defined as the mean distance from the final shape (x_i, y_i) to (x_{gt}, y_{gt}) measured over a normal profile

$$D_{pt \to border} = \frac{1}{N} \sum_{i=1}^{N} distance(along\ the\ normal_{(x_i, y_i) \to (x_{gt}, y_{gt})}). \tag{7.5}$$

A graphical interpretation of the two previous metrics are illustrated in image 7.6.

The *Procrustes* distance, in eq. 3.2, gives the square root of the sum of squared differences of the landmarks positions in two shapes. It is used the averaged *Procrustes* distance, eq 7.6, as comparison metric to give an idea of the degree of deformation between two shapes. The final fitted shape, (x, y), is aligned with (x_{gt}, y_{gt}) using the *Procrustes* analysis, resulting $(x_{aligned}, y_{aligned})$. The error is measured using the Euclidian distance between the aligned shape and the ground truth shape.

$$D_{avgProcrustes} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(x_{i_{aligned}} - x_{gt})^2 + (y_{i_{aligned}} - y_{gt})^2}. \tag{7.6}$$

Texture is also evaluated, and the error between the final model instance and the target image is measured using

$$T_{error} = \frac{1}{m}\sum_{i=1}^{m}|\mathbf{g}_{image} - \mathbf{g}_{model}|^2. \tag{7.7}$$

It is shown the evolution of each evaluation metric over translation displacements using as starting reference the CoG, figure 7.7, and the AdaBoost method, figure 7.8. The vertical bars represent unit standard deviations. The results presented include fitted data even when model fitting fails. At a glance, the graphics show that is seems that the model fits within a large range of location perturbations using the AdaBoost approach. These experiments show that normally this method returns a better starting initial estimate compared with starting the AAM search with the CoGs aligned. As expected, the point to border error compared to point to point error returns better results.

Remembering that the training images have the size of 640×480 pixels, the AAM model used in these experiences was obtained holding 95% of the total variance of shape, texture and combined model. Normally the model fits sucessfuly in the range of near $[-40, 40]$ pixels units over x and y-axis.

Tables 7.1 and 7.2 shows respectively, the failure rate from the two reference initial estimates over translation perturbations. As mentioned earlyer, the model is considered a failure if any of the appearance parameters fails condition 7.2.

Table 7.1: Model fit failure rate with CoG initial estimate.

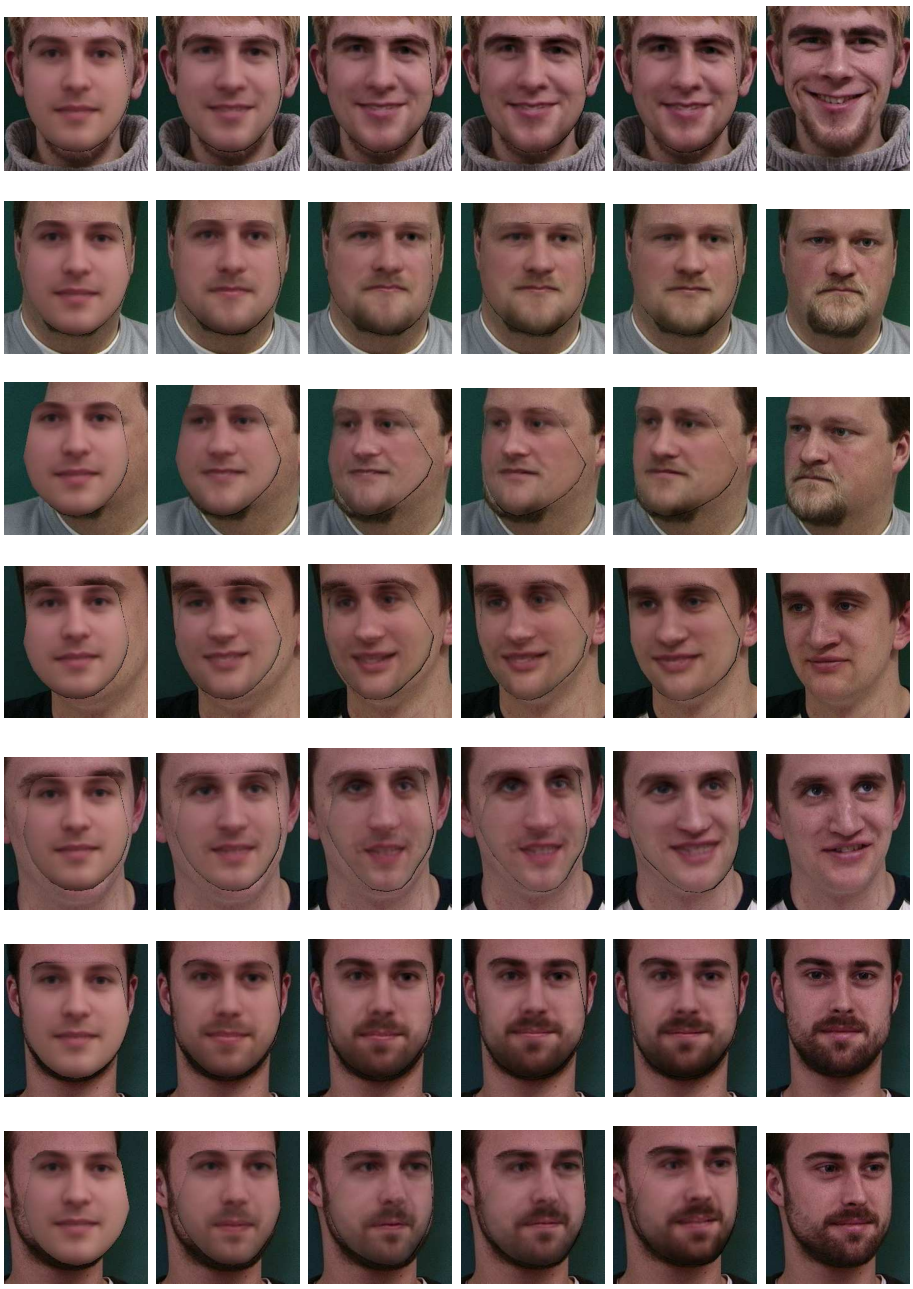
Tx	**Failure%**	**Tx**	**Failure%**	**Ty**	**Failure%**	**Ty**	**Failure%**
1	12,71	-1	13,56	1	13,56	-1	13,56
3	13,56	-3	12,71	3	12,71	-3	12,71
5	14,41	-5	13,56	5	13,56	-5	11,02
10	13,56	-10	15,25	10	15,25	-10	11,02
15	12,71	-15	14,41	15	14,41	-15	12,71
20	12,71	-20	15,25	20	15,25	-20	13,56
25	12,71	-25	18,64	25	18,64	-25	17,8
30	15,25	-30	22,03	30	22,03	-30	30,51
35	20,34	-35	26,27	35	26,27	-35	49,15
40	38,14	-40	34,75	40	34,75	-40	72,03
45	60,17	-45	49,15	45	49,15	-45	84,75
50	81,36	-50	63,56	50	63,56	-50	93,22

(Tx=0,Ty=0) → Failure=12,71%

Table 7.2: Model fit failure rate with AdaBoost initial estimate.

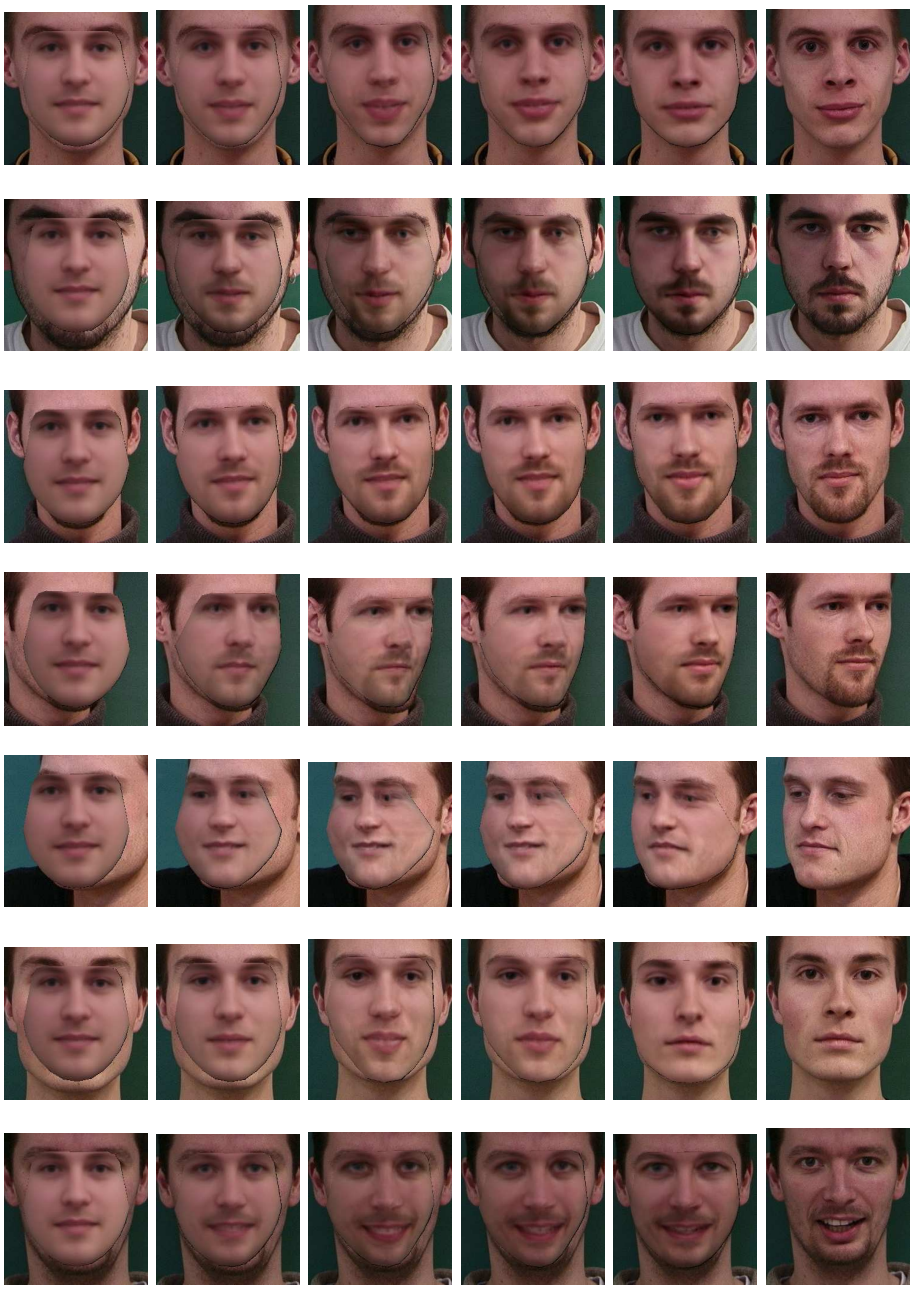
Tx	**Failure%**	**Tx**	**Failure%**	**Ty**	**Failure%**	**Ty**	**Failure%**
1	11,86	-1	12,71	1	12,71	-1	12,71
3	11,86	-3	13,56	3	12,71	-3	11,02
5	14,41	-5	11,02	5	11,86	-5	10,17
10	11,86	-10	14,41	10	12,71	-10	11,02
15	12,71	-15	14,41	15	11,86	-15	11,02
20	12,71	-20	15,25	20	12,71	-20	12,71
25	14,41	-25	17,8	25	11,02	-25	16,1
30	15,25	-30	18,64	30	14,41	-30	18,64
35	21,19	-35	21,19	35	20,34	-35	31,36
40	34,75	-40	33,05	40	38,14	-40	61,86
45	55,08	-45	51,69	45	66,1	-45	88,14
50	82,2	-50	67,8	50	83,05	-50	95,76

(Tx=0,Ty=0) → Failure=11,86%



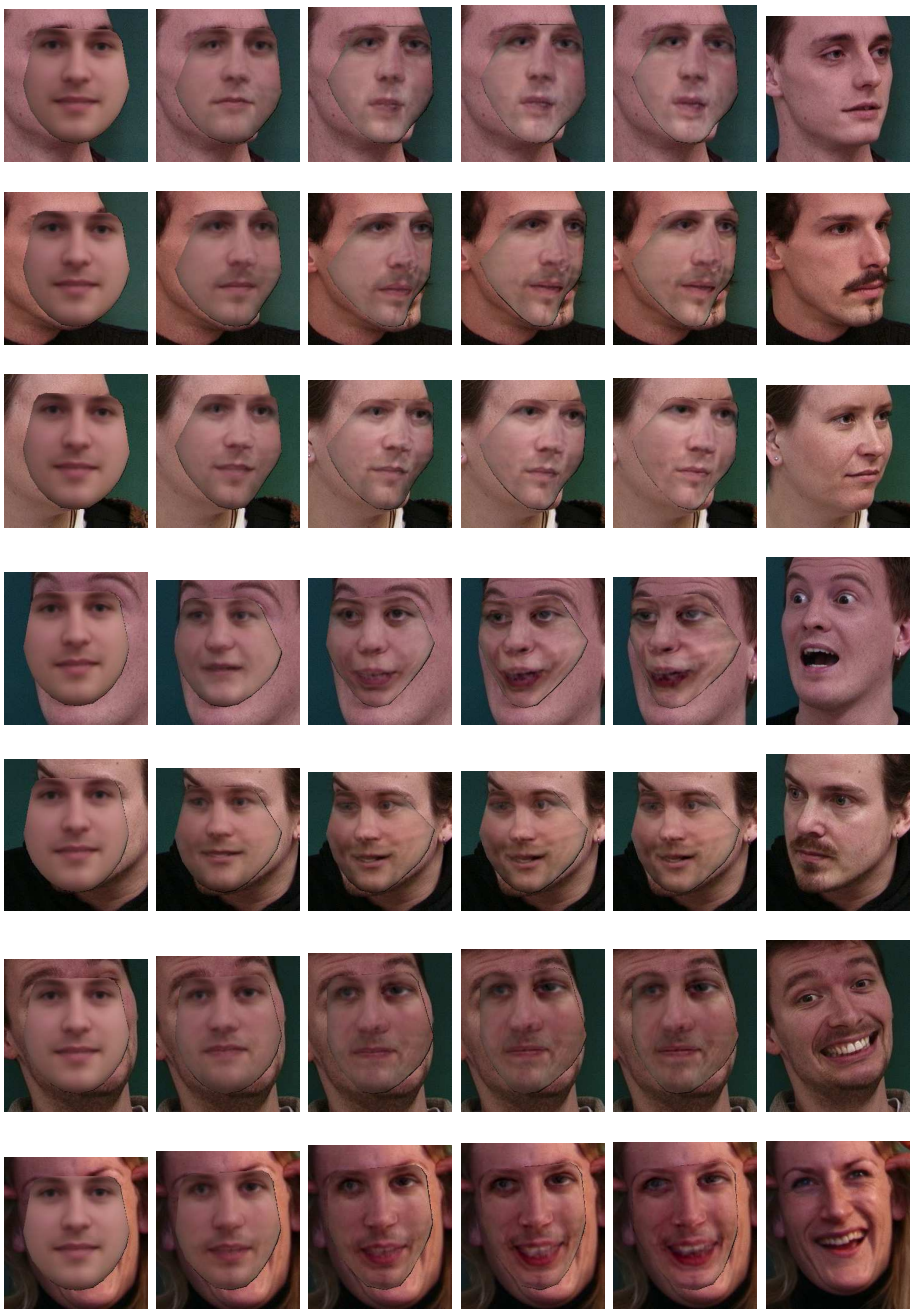
(a) 1^{st} (b) 2^{nd} (c) 3^{rd} (d) 5^{th} (e) Final (f) Original

Figure 7.1: Examples of AAM fitting procedure.



(a) 1^{st} (b) 2^{nd} (c) 3^{rd} (d) 5^{th} (e) Final (f) Original

Figure 7.2: More examples of AAM fitting procedure.



(a) 1^{st} (b) 2^{nd} (c) 3^{rd} (d) 5^{th} (e) Final (f) Original

Figure 7.3: Examples of model fitting failure.

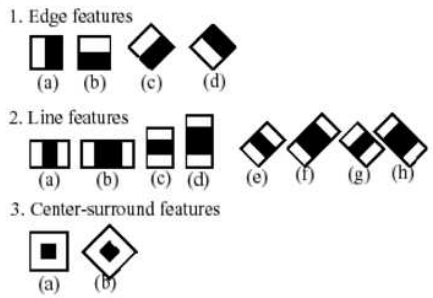


Figure 7.4: Haar-like features used. Courtesy from [5].

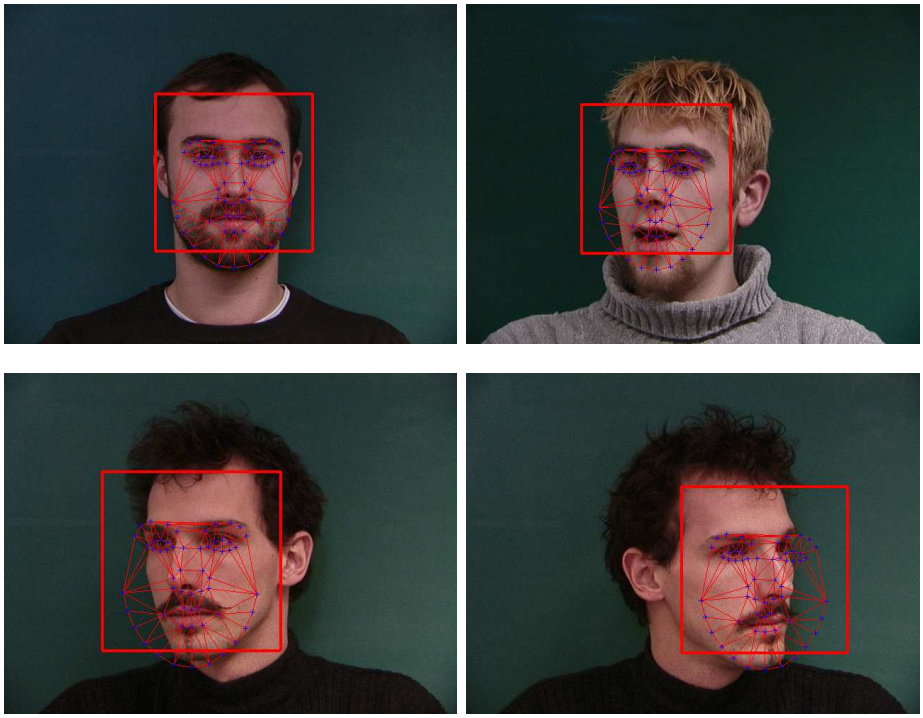


Figure 7.5: Initial estimate examples obtained by AdaBoost.

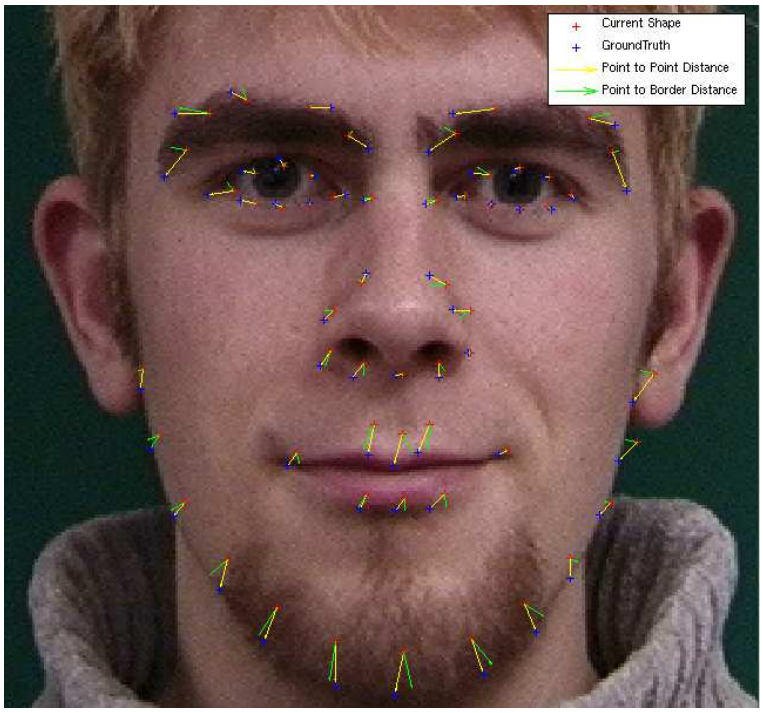


Figure 7.6: Landmarks error evaluation metrics.

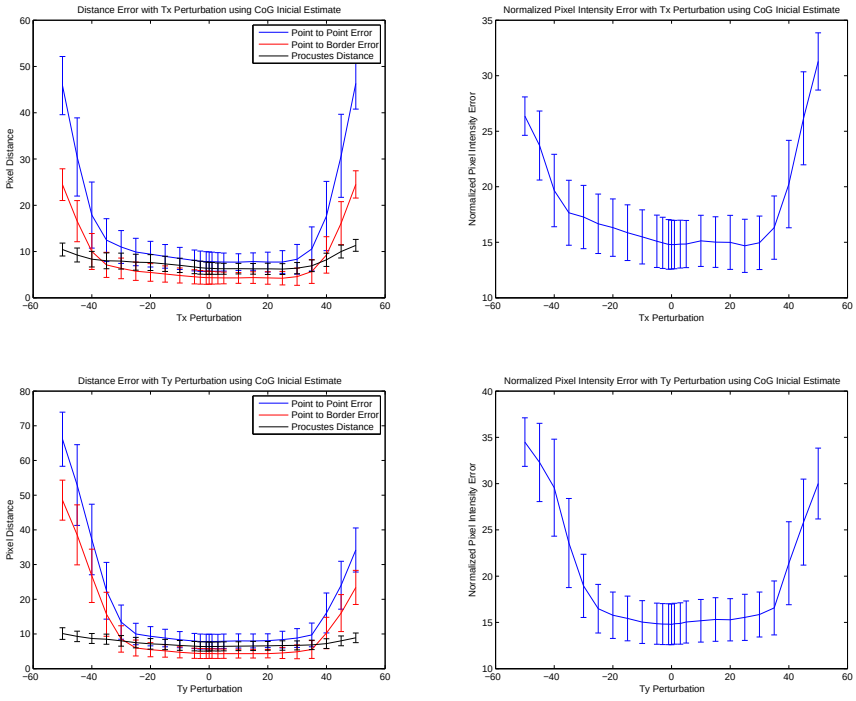


Figure 7.7: Monte Carlo simulation results for CoG initial estimate.

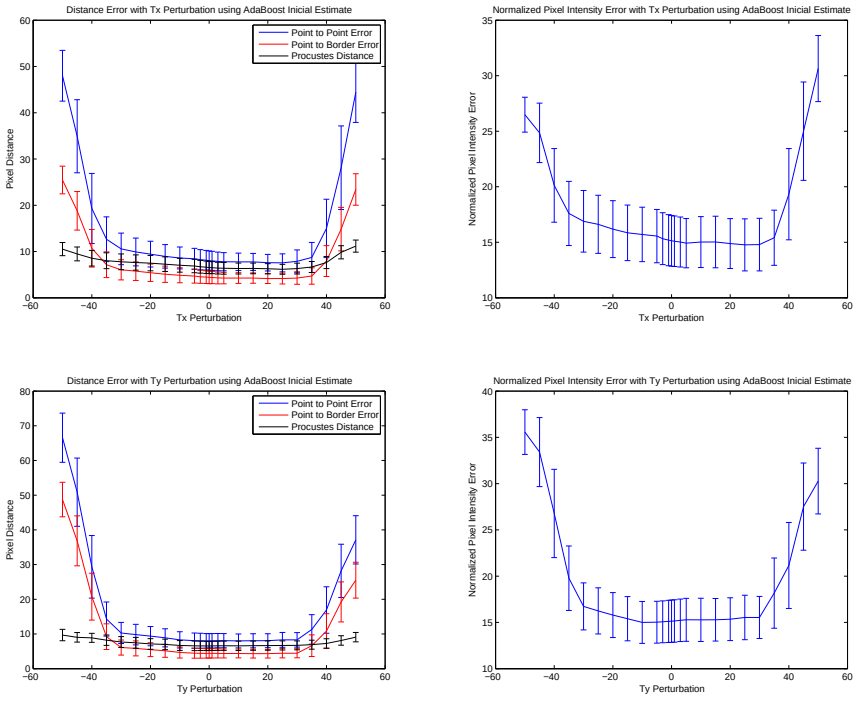


Figure 7.8: Monte Carlo simulation results for AdaBoost initial estimate.

Part II

FEAR: Facial Expression Analysis Recognition

Chapter 8

Introduction

Facial expressions recognition plays an important role in human communication since it has more influence than simpler audio information. Psychology studies, namely the Facial Action Coding System (FACS) [14] describe that there are six basic emotions universally recognized: joy, sadness, surprise, fear, anger and disgust. Notice that, these expressions are also compatible with MPEG-4 norm.

Facial expression recognition consist of three major steps, face detection, a mechanism for extracting facial expression information, and a mechanism to classify the information extracted according a set of categories [38].

An Adaboost [63] face detection is embedded in the Active Appearance Models (AAM) [58] framework where its used as initial estimate.

Model-based approaches for the interpretation of faces are used to extract relevant image information. AAM is an effective way to locate facial features, modeling both shape and texture and also the correlations between them from an observed training set. Statistical redundancy reduction techniques are used for compact coding, fully describing face characteristics in a reduced model. The discriminant features extracted from a target image consists on the AAM appearance parameters vector, $\mathbf{c}$.

In statistical supervised learning, Linear Discriminant Analysis [39] (LDA) is a classical solution that finds the basis vectors maximizing the interclass distances while minimizing the intraclass distances. Linear Discriminant Analysis is performed in order to extract the most discriminant features which max-

imizes class separability. The discriminating power of the AAM appearance parameters is analysed when projected in Fisherspace and the degree of similarity is measured in that subspace using Mahalanobis distances. Facial expression recognition was also achieved projecting the appearance parameters to the hyperplane that maximizes class separability using a multi-class Support Vector Machines (SVM) [1]. A series of experiments where conducted on a set of unknown images showing the faces of different subjects with facial expressions ranging neutral to intensely expressive. All tests were performed using still images without analyzing temporal evolution of facial expressions.

In this thesis part, chapters 9 and 10 refers to Linear Discriminant Analysis and Support Vector Machines based recognition, respectively. Chapter 11 discusses the results.

8.1 Related Work

Early work on automatic facial expression recognition by Ekman [14], introduced the Facial Action Coding System (FACS). FACS provided a prototype of the basic human expressions and allowed researchers to study facial expression based on an anatomical analysis of facial movement. A movement of one or more muscles in the face is called an action unit (AU) and all expressions can then be described by a combination of one or more of 44 AUs.

Pantic & Rothkrantz [38] identify three basic problems a facial expression analysis approach needs to deal with: face detection in a facial image or image sequence, facial expression data extraction and facial expression classification.

Most previous systems assume presence of a full frontal face view in the image or the image sequence being analyzed, yielding some knowledge of the global face location. To give the exact location of the face, Viola & Jones [63] use the Adaboost algorithm to exhaustively pass a search sub-window over the image at multiple scales for rapid face detection.

To perform data extraction, Littlewort [19] use a bank of 40 Gabor Wavelet filters at diferent scales and orientations to perform convolution. Pentland [15] extend their face detection approach to extract the positions of prominent facial features using eigenfeatures and PCA by calculating the distance of an image

from a feature space given a set of sample images via Fast Fourier Transform (FFT) and a local energy computation. Cohn [25] first manually localize feature points in the first frame of an image sequence and then use hierarchical optical flow to track the motion of small windows surrounding these points across frames. The displacement vectors for each landmark between the initial and the peak frame represent the extracted expression information.

In the final step of expression analysis, expressions are classified according to some scheme. The most prevalent approaches are based on the existence of six basic emotions (joy, sadness, surprise, fear, anger and disgust) as argued on FACS [14]. While much progress has been made in automatically classifying according to FACS [67], a fully automated FACS based approach for video is yet to be developed.

Chapter 9

Linear Discriminant Analysis Based Recognition

The facial expression recognition of seven different emotions, namely neutral expression, happiness, sadness, surprise, anger, fear and disgust is proposed. The classification procedure is performed by firstly describing a set of faces with the AAM model [12], using the extracted appearance vector as features. Afterwards, each appearance vector is projected into Fisherspace and the classification is performed using the Mahalanobis distance.

This section describes the projection of data into the Fisherspace, i.e. the use of a Linear Discriminant Analysis (LDA). Since LDA involves a eigenvector decomposition, a way to evaluate the data discrimination is required. For this propose a k-means based metric is used, measuring the degree of concentration on each class at several experiences, deciding how many LDA eigenvectors should be held.

To evaluate the recognition rate, an expression database was built, consisting in 21 individuals presenting seven different facial expressions each. Several AAM models, holding different variances, were built and evaluated. Using a leave-one-out cross-validation scheme, results in form of confusion matrices are shown.

9.1 Linear Discriminant Analysis - LDA

A Linear Discriminant Analysis (LDA) [39], also known as Fisher's Linear Discriminant (FLD), is a supervised learning technique that consists in optimizing the separability of the dataset observations according to the expression class they belong to. This linear transformation of data will allow subsequent classification of new images representing one of the expression categories.

Fisher-LDA considers maximizing the following objective function [64]

$$\arg\max_{\mathbf{w}} J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{9.1}$$

where $\mathbf{S}_B$ is the between-class scatter matrix and $\mathbf{S}_W$ is the within-class scatter matrix given by eqs 9.2 and 9.3 respectively.

$$\mathbf{S}_B = \sum_{i=1}^{n_c} N_i (\overline{\mathbf{x}}_i - \overline{\mathbf{x}})(\overline{\mathbf{x}}_i - \overline{\mathbf{x}})^T \tag{9.2}$$

$$\mathbf{S}_W = \sum_{i=1}^{n_c} \sum_{j=1}^{N_i} (\mathbf{x}_{i,j} - \overline{\mathbf{x}}_i)(\mathbf{x}_{i,j} - \overline{\mathbf{x}}_i)^T \tag{9.3}$$

In these expressions, n_c is the number of classes, $\mathbf{x}_{i,j}$ is the j^{th} sample in class i, $\overline{\mathbf{x}_i}$ is mean of class i, $\overline{\mathbf{x}}$ is the mean of all samples and N_i is the number of samples in class i.

An important property about the objective function, $\mathbf{J}(\mathbf{w})$ is that it is invariant w.r.t. rescalings of the vectors $\mathbf{w} \rightarrow \alpha\mathbf{w}$. Hence, $\mathbf{w}$ can always be chosen such that the denominator is simply $\mathbf{w}^T \mathbf{S}_W \mathbf{w} = 1$. For that reason, maximizing $J(\mathbf{w})$ can be transformed into the following constrained optimization problem,

$$\begin{array}{ll} \min_{\mathbf{w}} & -\frac{1}{2}\mathbf{w}^T \mathbf{S}_B \mathbf{w} \\ s.t. & \mathbf{w}^T \mathbf{S}_W \mathbf{w} = 1 \end{array} \tag{9.4}$$

corresponding to the Lagrangian,

$$\mathcal{L} = -\frac{1}{2}\mathbf{w}^T \mathbf{S}_B \mathbf{w} + \frac{1}{2}\lambda(\mathbf{w}^T \mathbf{S}_W \mathbf{w} - 1). \tag{9.5}$$

The Karush-Kuhn-Tucker (KKT) conditions leads to the following dual optimization problem,

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w} \quad \Rightarrow \quad \mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}. \tag{9.6}$$

Maximizing eq 9.1 leads to $\mathbf{S}_W^{-1}\mathbf{S}_B\mathbf{w} = \lambda\mathbf{w}$ which is an eigen-problem. The solution that maximizes the between-class variance and minimizes the within-class variance is given by the direction of the eigenvector associated to the maximum eigenvalue of $\mathbf{S}_W^{-1}\mathbf{S}_B$. Notice that $\mathbf{S}_W$ should be a full rank matrix. For that it is required more samples than dimensions modeled.

9.1.1 K-means

Clustering is the computational task to partition a given input into subsets of equal characteristics. These subsets are usually called clusters and ideally consist of similar objects that are dissimilar to objects in other clusters.

The most prominent and widely used clustering algorithm is the *Lloyd's* algorithm also referred as the k-means [49]. This algorithm requires the input set to be a set of points in the d-dimensional Euclidian space. Its goal is to find k cluster centers and partitioning of the points such that the sum of squared distances to the nearest center is minimized. The algorithm is a heuristic that converges to a local optimum.

Using $\|p, q\|$ to denote the Euclidian distance between p and q where $p, q \in \Re^d$, $\mathcal{C}(Q)$ is used to represent the centroid of a set of Q points, i.e. $\mathcal{C}(Q) = \frac{1}{|Q|}\sum_{q\in Q} q$. The k-means clustering problem consists on find a set $C = \{c_1, ..., c_k\}$ of k cluster centers and a partition set P into k clusters $C_1, .., C_k$ such that

$$\sum_{i=1}^{k}\sum_{p\in C_i} \|p, c_i\|^2 \tag{9.7}$$

is minimized.

K-means algorithm runs in iterations and is described in algorithm 8. Since the algorithm only converges to local optimum and the quality of the solution depends on the initial set centers, it is usually repeated several times with different sets of initial centers.

9.1.2 LDA Evaluation Metric

Similarly to a Principal Component Analysis, a LDA transformation also involves performing an eigenvector decomposition which reflects the importance of the data variance on the transformed subspace. The resulting data model

Algorithm 8 k-means

1: Set initial guesses for the k centers $\{c_1, ..., c_k\}$
2: **while** Until there are no changes in any $\{c_1, ..., c_k\}$ **do**
3: **for** Every $p \in P$ **do**
4: Compute its nearest center in $\{c_1, ..., c_k\}$
5: **end for**
6: Partition P into k sets $C_1, ..., C_k$ such that C_i contains all points whose nearest center is c_i
7: **for** Every cluster $C_i = 1$ to $C_k = 1$ **do**
8: Compute its centroid $c_i = \mathcal{C}(C_i)$
9: **end for**
10: **end while**

retains the features that maximize class separability while holding a portion of the data variance.

In order to evaluate the quality of the data discrimination, a metric based on applying a k-means clustering algorithm on the result of the LDA was analysed. Considering the ideal case where all the observations were completely separated after LDA, applying a clustering algorithm on the data, the obtained result will be n_c groups containing all faces with the same expression. Table 9.1 shows what would be the ideal clustering result for a dataset of n_s subjects per expression. Note that since there are seven expressions the k-means algorithm would be applied to get seven groups.

Table 9.1: k-means clustering result for ideal LDA.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	$\mathbf{n}_s$	0	0	0	0	0	0
Happ	0	$\mathbf{n}_s$	0	0	0	0	0
Sad	0	0	$\mathbf{n}_s$	0	0	0	0
Surp	0	0	0	$\mathbf{n}_s$	0	0	0
Ang	0	0	0	0	$\mathbf{n}_s$	0	0
Fear	0	0	0	0	0	$\mathbf{n}_s$	0
Disg	0	0	0	0	0	0	$\mathbf{n}_s$

The metric assigns a discrimination quality value to the clustering result. Its value is calculated by eq 9.8, i.e. summing the difference between the higher and the lower values of each row of this table returning the sum of the degree of concentration of each class.

$$D_{LDA_{concentration}} = \sum_{i=1}^{n_c} \max(row(i)) - \min(row(i)) \tag{9.8}$$

The higher the metric output, the better is the performed discrimination. By applying this metric to several LDA runs, with different number of features retained, the number of LDA eigenvectors that should be held to optimize the discrimination can be estimated.

9.1.3 Fisherspace Classification

Once defined the axes that maximize the data classes separability, it is possible to classify either training images or unseen faces. The classification for an unknown face image consists in two steps. The first step is projecting its appearance vector, $\mathbf{c}$, on the hyperspace that resulted from the training process. The second step is to estimate to each group does this projected point belongs to. This is done by using an adaptation of the nearest-neighbourhood algorithm, which takes into consideration not only the distance of the point to the centre of each group, but also its dispersion. The distance to each group is measured using the Mahalanobis distance [30] given by

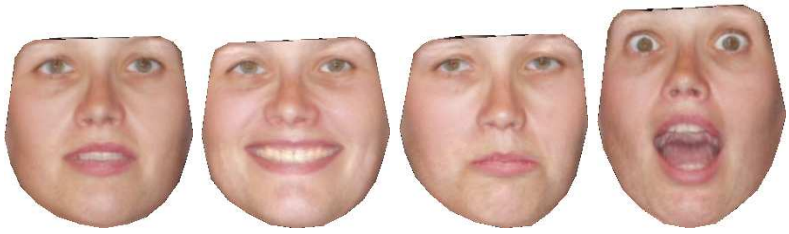
$$D_{Mahalanobis} = (\mathbf{c}_i - \overline{\mathbf{c}}_i)\Sigma^{-1}(\mathbf{c}_i - \overline{\mathbf{c}}_i), \tag{9.9}$$

where $\mathbf{c}_i$ is the vector of extracted appearance parameters, $\overline{\mathbf{c}}_i$ is the centroid of the class multivariate distribution and Σ is the within-class covariance matrix. Eq. 9.9 gives a scaled measure of an expression instance to a particular class.

9.2 Experimental Results

For the purpose of this work, an expression database was built, consisting in 21 individuals presenting seven different facial expressions each, namely neutral expression, happiness, sadness, surprise, anger, fear and disgust. The data set is therefore formed by a total of 147 colour images 640×480. Figure 9.1 shows

AAM model instances of a given subject for each one of the 7 facial expressions used. The shape model was built using 58 annotated landmarks, $N = 58$, and the texture model was generated sampling around 47000 pixels using also colour information, $m = 141000$.



(a) Neutral (b) Happy (c) Sad (d) Surprise

(e) Anger (f) Fear (g) Disgust

Figure 9.1: AAM instances of facial expressions used.

The dimensionality of the appearance vector, $\mathbf{c}$, changes as function of the retained variance on the three PCA of the training building process. Table 9.2 shows these dimension values, t_c, for held variances of the training set. Since more dimensional data not always produces better classification results, several experiences for each variance held dataset were performed.

9.2.1 Optimizing AAM and LDA Variation Modes

The influence of the number of retained variation modes in AAM and LDA on the final discrimination performance is analysed. A series of experiences were done, cross-validating the number of features held with several AAM face models and the number of LDA held modes, where the discrimination quality is evaluated using the k-means clustering algorithm.

As discrimination quality metric output depends on the result of a k-means clustering, these tests were repeated for 250 times. A histogram of the best LDA

Table 9.2: Retained variance and correspondent number of modes, t_c.

Variance(%)	t_c
95.0	17
97.0	29
98.0	42
99.0	70
99.5	97
99.9	133

features retained in each trial was created for every AAM model used. Figure 9.2 shows the result when 99.0% variation modes are retained in the dataset used.

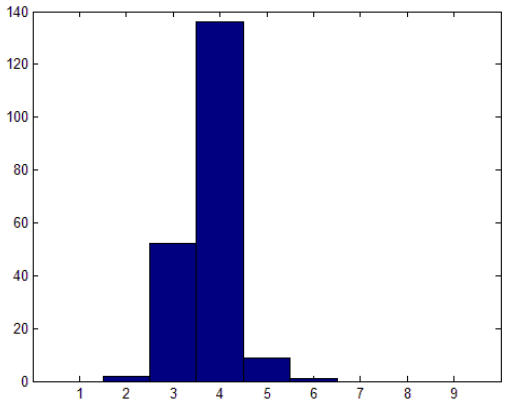


Figure 9.2: Histogram for best LDA modes on 250 trials.

These experiments shown that for the dataset used, four eigenvectors produce the best discrimination results, maximizing the separation of classes. This was also observed for 95.0% and 97.0% AAM variation modes.

9.2.2 Classifying 7 Expressions

A leave-one-out cross-validation scheme was used to evaluate the classification, where each one of the dataset observations is used once as the testing sample and the remaining data is used for training. Tables 9.3, 9.4 and 9.5 show results

of confusion matrices from a Linear Discriminant Analysis (LDA) combined with a Mahalanobis distance metric for 97%, 98% and 99% of held variance, respectively.

The better overall classification results come from the 99% held variance AAM model.

Table 9.3: LDA + Mahalanobis confusion matrix 97.0%.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	**19.0**	0	52.38	0	9.52	19.05	0
Happ	0	**90.48**	0	0	0	4.77	4.77
Sad	9.52	0	**61.90**	4.77	9.52	14.29	0
Surp	0	0	0	**80.95**	0	19.05	0
Ang	0	0	14.29	0	**33.33**	9.52	42.86
Fear	0	4.77	9.52	19.05	14.29	**52.38**	0
Disg	0	14.29	0	0	38.10	0	**47.62**

Overall recognition rate = 55%

Table 9.4: LDA + Mahalanobis confusion matrix 98.0%.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	**33.33**	0	61.90	0	0	4.76	0
Happ	0	**80.95**	0	0	0	9.52	9.52
Sad	0	4.76	**66.67**	0	19.05	9.52	0
Surp	0	0	0	**71.43**	0	28.57	0
Ang	0	4.76	4.76	0	**42.86**	14.29	33.33
Fear	0	4.76	9.52	19.05	9.52	**52.38**	4.76
Disg	0	9.52	0	0	38.10	4.76	**47.62**

Overall recognition rate = 56.5%

From the results, is shown that there is an effect of confusion between neutral and sad expressions and also between anger and disgust, suggesting appearance correlations between these two pairs of expressions. Images 9.3 and 9.4 display examples of training images where this effect is clearly visible. At a glace, even

Table 9.5: LDA + Mahalanobis confusion matrix 99.0%.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	**52.38**	0	42.86	0	4.76	0	0
Happ	0	**90.48**	4.76	0	0	4.76	0
Sad	4.76	4.76	**76.19**	0	4.76	4.76	4.76
Surp	0	0	0	**76.19**	0	23.81	0
Ang	4.76	0	9.52	0	**33.33**	23.81	28.57
Fear	0	9.52	4.76	14.29	4.76	**66.67**	0
Disg	0	23.81	4.76	0	33.33	4.76	**33.33**

Overall recognition rate = 61.2%

for a human, its not straightforward to decide which expression is shown. In fact, these results are not surprising. Neuroscience studies [29] have shown that in case of human emotion recognition, to perceive sad and anger expressions, an specific cognitive process is required.

9.2.3 Classifying 5 Expressions

Eliminating the confusion effect, i.e. excluding, both from the training and testing set the faces expressing sadness and anger, naturally leads to better classification results. Confusion matrices for same LDA and AAM conditions are expressed on tables 9.6, 9.7 and 9.8.

Now the better classification results, table 9.7, were extracted from the 98% AAM model.

Table 9.6: LDA + Mahalanobis confusion matrix 97.0%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**57.14**	0	0	42.86	0
Happ	0	**90.48**	0	9.52	0
Surp	0	0	**76.19**	23.81	0
Fear	9.52	4.76	23.81	**61.90**	0
Disg	0	9.52	0	4.76	**85.71**

Overall recognition rate = 74.3%

Table 9.7: LDA + Mahalanobis confusion matrix 98.0%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**90.48**	0	0	9.52	0
Happ	0	**85.71**	0	4.76	9.52
Surp	0	0	**71.43**	28.57	0
Fear	4.76	4.76	19.05	**66.67**	4.76
Disg	0	19.05	0	14.29	**66.67**

Overall recognition rate = 76.2%

Table 9.8: LDA + Mahalanobis confusion matrix 99.0%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**80.95**	9.52	4.76	4.76	0
Happ	9.52	**66.67**	0	14.29	9.52
Surp	4.76	0	**66.67**	28.57	0
Fear	4.76	9.52	28.57	**52.38**	4.76
Disg	9.52	23.81	4.76	9.52	**52.38**

Overall recognition rate = 63.8%

(a) Neutral (b) Sad

Figure 9.3: Correlation between neutral and sad expressions.

(a) Anger (b) Disgust

Figure 9.4: Correlation between anger and disgust expressions.

Chapter 10

Support Vector Machines Based Recognition

Like in the previous section, the problem of facial expression recognition is addressed here. For the same conditions, i.e. the same dataset with seven expressions and the AAM appearance parameters used as discriminant face features, a Support Vector Machines (SVM) [1] based recognition is proposed.

In this section each appearance vector is projected into the hyperspace that maximize class separability using a multi-class Support Vector Machines. Since SVM were originally designed for binary classification, extensions for multi-class classification were used, namely one-against-all approach.

Standard Support Vector Machines are briefly described as well as multi-class classification extensions. Similarly to the previous section, seven and five expressions were classified and evaluated using a leave-one-out scheme. Confusion matrices with the achieved results are shown.

10.1 Support Vector Machines - SVM

Support Vector Machines (SVM) are maximal margin hyperplane classification methods that relies on results from statistical learning theory to guarantee high generalization performance. SVM is capable to solve linear and nonlinear classification problems. In the nonlinear case a kernel function is used to map the

input data into the feature space, normally with higher dimensionality. The propose of SVM is to map this nonlinear data into a higher dimensional space, maybe infinite, and make them linearly separable in that space.

Consider the problem of separating the set of training instance-label pairs belonging to two separate classes,

$$\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_l, \mathbf{y}_l)\}, \quad \mathbf{x} \in \Re^n, \quad \mathbf{y} \in \{1, -1\}, \tag{10.1}$$

the separation hyperplane is given by

$$\mathbf{w}\mathbf{x} + b = 0 \tag{10.2}$$

where $\mathbf{w}$ is the input weight and b is the bias. Figure 10.1 shows a geometric interpretation for bidimensional data. The space is divided in two by an hyperplane defined in eq 10.2. $\mathbf{w}$ defines a direction normal to the hyperplane and $\mathbf{b}$ is the offset.

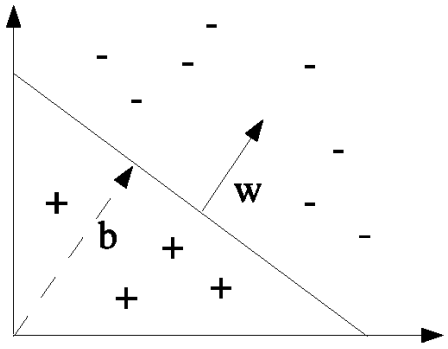


Figure 10.1: Separation hyperplane $(\mathbf{w}, b)$ for a bidimensional data.

The optimal hyperplane that maximize the class separability is determined by solving the following quadratic programing task,

$$\begin{aligned} \min_{\mathbf{w},\mathbf{b},\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ s.t. \quad & \mathbf{y}_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{10.3}$$

were the training vectors $\mathbf{x}_i$ are mapped into a higher dimensional space by the function ϕ. Then SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. Furthermore, $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ is called the kernel function. ξ are slack variables that measure misclassification errors and C is the penalty parameter for this error term.

10.1.1 Kernels

The use of functions that map the input data into a high dimensional feature space is the strategy to deal with nonlinear separable data, transforming the input data $\mathbf{x}$,

$$\mathbf{x} = (\mathbf{x}_i, \ldots, \mathbf{x}_l) \rightarrow \phi(\mathbf{x}) = (\phi(\mathbf{x}_i), \ldots, \phi(\mathbf{x}_l)). \tag{10.4}$$

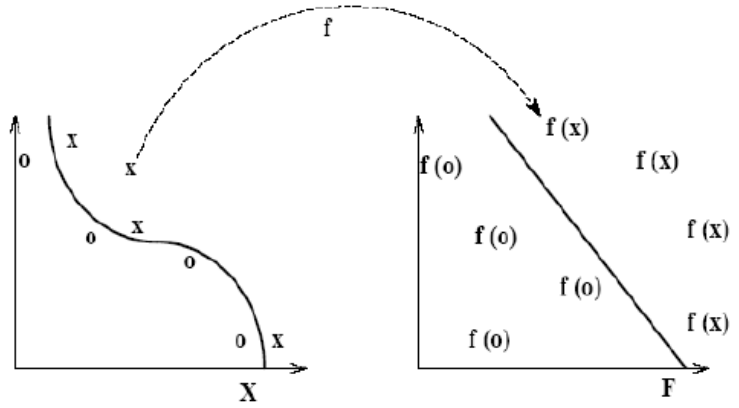


Figure 10.2: SVM nonlinear mapping.

A kernel is a function of the form $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ that allows to compute the dot product between features in the feature space without transform the data into that space, which enables operations to be performed in the input space rather than the potentially high dimensional feature space.

A kernel function, $\mathbf{K}$, should be a symmetric positive definite function, that satisfies the Mercer's conditions [1]. Table 10.1 shows the four kernels generally used.

Table 10.1: The four basic SVM kernels. γ, r, and d are kernel parameters.

Linear	$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
Polynomial	$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d, \gamma > 0$
Radial Basis Function (RBF)	$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}, \gamma > 0$
Sigmoid	$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$

10.1.2 Multi-Class Support Vector Machines

Support Vector Machines were originally designed for binary classification. Two approaches of achieve multi-class SVM exists [21]. One is by constructing and combining several binary classifiers while the other is by considering all data in one optimization formulation. For details on this last approach refer to [26]. One-against-all, one-against-one [2] and DAGSVM [3] are cascade of binary SVM classifiers methods that are briefly described in the following.

One-Against-All

In one-against-all method it constructs k SVM models where k is the number of classes. The i^{th} SVM is trained with all of the examples in the i^{th} class with positive labels, and all the other examples with negative labels. Thus given l training data $(x_1, y_1), \ldots, (x_l, y_l)$, where $x_i \in \Re^n, i = 1, \ldots, l$ and $y_i \in \{1, \ldots, k\}$ is the class of x_i, the ith SVM solves the following optimization problem

$$\begin{array}{ll} \min_{\mathbf{w}^i, \mathbf{b}^i, \xi^i} & \frac{1}{2}(\mathbf{w}^i)^T \mathbf{w}^i + C \sum_{j=1}^{l} \xi_j^i \\ s.t. & (\mathbf{w}^i)^T \phi(\mathbf{x}_j) + b^i \geq 1 - \xi_j^i, \text{ if } \mathbf{y}_j = i, \\ & (\mathbf{w}^i)^T \phi(\mathbf{x}_j) + b^i \leq -1 + \xi_j^i, \text{ if } \mathbf{y}_j \neq i, \\ & \xi_j^i \geq 0, j = 1, \ldots, l \end{array} \tag{10.5}$$

where the training data x_i are mapped to a higher dimensional space by the function ϕ and C is the penalty parameter.

Solving 10.5 leads k decisions functions

$$\begin{array}{c} (\mathbf{w}^1)^T \phi(\mathbf{x}) + b^1 \\ \vdots \\ (\mathbf{w}^k)^T \phi(\mathbf{x}) + b^k \end{array} \tag{10.6}$$

The final decision of $\mathbf{x}$ is the class that has the largest value of the decision function, i.e.

$$\text{decision } \mathbf{x} = \arg \max_{i=1,\ldots,k} ((\mathbf{w}^i)^T \phi(\mathbf{x}) + b^i). \tag{10.7}$$

One-Against-One

Other multi-class voting scheme is called one-against-one [2]. This method builds $k(k-1)/2$ classifiers were each one is trained on data from two classes. For

training data from the i^{th} and the j^{th} classes the following binary classification problem is solved

$$\begin{array}{ll} \min_{\mathbf{w}^{ij},\mathbf{b}^{ij},\xi^{ij}} & \frac{1}{2}(\mathbf{w}^{ij})^T\mathbf{w}^{ij} + C\sum_t \xi_t^{ij} \\ s.t. & (\mathbf{w}^{ij})^T\phi(\mathbf{x}_t) + b^{ij} \geq 1 - \xi_t^{ij}, \text{ if } \mathbf{y}_t = i, \\ & (\mathbf{w}^{ij})^T\phi(\mathbf{x}_t) + b^{ij} \leq -1 + \xi_t^{ij}, \text{ if } \mathbf{y}_t \neq i, \\ & \xi_t^{ij} \geq 0. \end{array} \tag{10.8}$$

The decision is based on testing all $k(k-1)/2$ classifiers on a "Max Wins" strategy, i.e. if sign $(\mathbf{w}^{ij})^T\phi(\mathbf{x}) + b^{ij})$ it means that $\mathbf{x}$ is in the i^{th} class, then the vote for the i^{th} class is added by one. Otherwise, the j^{th} class is increased by one. The predicted $\mathbf{x}$ is in the class with largest vote. In the case of draw, any class is selected. The final decision is the one that has the smaller index.

DAGSVM

The third method is the Direct Acyclic Graph Support Vector Machines (DAGSVM) [3] consists on the same training stage that the one-against-one method solving the optimization problem 10.8. However a rooted binary directed acyclic graph with $k(k-1)/2$ nodes internal nodes and k leaves. Each node is a binary SVM of i^{th} and j^{th} classes. Given a test sample $\mathbf{x}$, starting at the root node, the binary decision moves left or right depending on the output value, going through the tree reaching to the leaf node which indicates the predicted class.

10.2 Experimental Results

Using the extracted appearance vectors that describe each facial expression, a multiclass Support Vector Machines (SVM) [1] classification was performed. The appearance vectors are projected into the hyperspace that maximize class separability.

The facial expression SVM classification was achieved using a multi-class one-against-all voting scheme with a gaussian Radial Basis Function (RBF) kernel, $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2}$, where the features data are normalized by mapping-it into the unitary hypersphere. The kernel parameters, i.e. γ and the penalty C, were found by cross-validation ranging the values in the interval $[1^{-12}, \ldots, 1^{12}]$.

10.2.1 Classifying 7 Expressions

A leave-one-out cross-validation scheme was used to evaluate the classification. Tables 10.2, 10.3, 10.4 and 10.5 show the confusion matrices for 95%, 97%, 98% and 99% of held variance, respectively, using multi-class one-against-all SVM classification with a RBF kernel.

The better overall classification results come from the 95% held variance AAM model.

Table 10.2: SVM confusion matrix 95.0%.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	**50**	0	37.5	0	6.25	0	6.25
Happ	6.25	**81.25**	0	6.25	0	0	6.25
Sad	18.75	0	**56.25**	0	6.25	12.5	6.25
Surp	6.25	0	0	**68.75**	0	25	0
Ang	6.25	0	6.25	0	**37.5**	25	25
Fear	12.5	6.25	0	31.25	6.25	**37.5**	6.25
Disg	0	6.25	12.5	0	37.5	0	**43.75**

Overall recognition rate = 53.57%

Table 10.3: SVM confusion matrix 97.0%.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	**62.5**	0	31.25	0	6.25	0	0
Happ	6.25	**87.5**	0	0	0	0	6.25
Sad	25	0	**37.5**	0	0	25	12.5
Surp	0	0	0	**75**	0	25	0
Ang	0	6.25	12.5	0	**25**	18.75	37.5
Fear	18.75	6.25	6.25	25	12.5	**31.25**	0
Disg	6.25	12.5	6.25	0	37.5	0	**37.5**

Overall recognition rate = 50.89%

As in the previous section, the SVM classification results also show that there are appearance correlations between neutral and sad expressions and also

Table 10.4: SVM confusion matrix 98.0%.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	**50**	0	31.25	6.25	6.25	6.25	0
Happ	0	**93.75**	0	0	6.25	0	0
Sad	18.75	0	**50**	0	12.5	18.75	0
Surp	0	0	6.25	**62.5**	0	31.25	0
Ang	0	0	12.5	0	**25**	18.75	43.75
Fear	25	6.25	6.25	25	18.75	**18.75**	0
Disg	0	6.25	6.25	0	56.25	0	**31.25**

Overall recognition rate = 47.32%

Table 10.5: SVM confusion matrix 99.0%.

	Neut	**Happ**	**Sad**	**Surp**	**Ang**	**Fear**	**Disg**
Neut	**62.5**	0	25	0	0	12.5	0
Happ	0	**87.5**	0	0	6.25	0	6.25
Sad	31.25	0	**43.75**	0	12.5	12.5	0
Surp	0	0	6.25	**81.25**	0	12.5	0
Ang	18.75	0	0	0	**18.75**	12.5	50
Fear	18.75	6.25	6.25	25	12.5	**31.25**	0
Disg	0	6.25	6.25	6.25	50	0	**31.25**

Overall recognition rate = 50.89%

between anger and disgust.

10.2.2 Classifying 5 Expressions

Excluding images from sadness and anger, the five-class SVM classification results are shown is tables 10.6, 10.7, 10.8, 10.9, 10.10 and 10.11 with respect to 95%, 97%, 98%, 99%, 99.5% and 99.9% of AAM held variance respectively.

Now the better classification results, table 10.10, were extracted from the 99.5% AAM model.

Table 10.6: SVM confusion matrix 95.0%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**81.25**	6.25	0	12.5	0
Happ	6.25	**75**	0	0	18.75
Surp	0	0	**81.25**	18.75	0
Fear	12.5	6.25	37.5	**37.5**	6.25
Disg	0	18.75	0	6.25	**75**

Overall recognition rate =70%

Table 10.7: SVM confusion matrix 97.0%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**93.75**	0	0	6.25	0
Happ	6.25	**81.25**	0	0	12.5
Surp	0	0	**87.5**	12.5	0
Fear	12.5	6.25	25	**43.75**	12.5
Disg	0	12.5	6.25	0	**81.25**

Overall recognition rate =77.5%

Table 10.8: SVM confusion matrix 98.0%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**81.25**	0	0	18.75	0
Happ	6.25	**87.5**	0	0	6.25
Surp	0	6.25	**68.75**	25	0
Fear	18.75	6.25	25	**37.5**	12.5
Disg	0	12.5	6.25	12.5	**68.75**

Overall recognition rate = 68.75%

Table 10.9: SVM confusion matrix 99.0%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**87.5**	0	0	12.5	0
Happ	6.25	**93.75**	0	0	0
Surp	0	6.25	**75**	18.75	0
Fear	12.5	6.25	25	**50**	6.25
Disg	0	12.5	6.25	12.5	**68.75**

Overall recognition rate = 75%

Table 10.10: SVM confusion matrix 99.5%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**100**	0	0	0	0
Happ	6.25	**93.75**	0	0	0
Surp	0	6.25	**81.25**	12.5	0
Fear	12.5	6.25	31.25	**43.75**	6.25
Disg	0	12.5	6.25	0	**81.25**

Overall recognition rate = 80%

Table 10.11: SVM confusion matrix 99.9%.

	Neut	**Happ**	**Surp**	**Fear**	**Disg**
Neut	**93.75**	0	0	6.25	0
Happ	6.25	**93.75**	0	0	0
Surp	0	6.25	**68.75**	25	0
Fear	25	6.25	37.5	**25**	6.25
Disg	0	12.5	6.25	0	**81.25**

Overall recognition rate = 72.5%

Chapter 11

Conclusions

Standart AAM model formulation was used to describe face characteristics in a compact way. Two approaches were used to classify facial emotions. Several classes were separated using LDA, projecting each appearance vector into Fisherspace, and classification was performed based on Mahalanobis distance. Similarly the same facial expressions were classified projecting the AAM appearance parameters to the hyperplane that maximizes class separability using a multi-class one-against-all SVM.

Since holding more information on the AAM building process not always produces a better discrimination result, several experiences were conducted. While using a LDA based recognition, regarding that the number of LDA eigenvectors held is a parameter of great importance, k-means was used to give a good estimation for this value.

Naturally the larger the number of expressions used, worse is the overall successful classification rate since there are more classes to classify. With all the seven expressions, the LDA (61.2%) approach achieved an slight better overall recognition rate. Since the results emphasis appearance correlations between the two pairs of expressions neutral, sad and anger, disgust, that are comproved by psico-physics studies, the correlated expressions were removed. Classifying five expressions shows that SVM method achieves an overall recognition rate of 80%.

Part III

Monocular Head Pose Estimation

Chapter 12

Introduction

In many Human Computer Interface (HCI) applications such as face recognition systems, teleconference, knowledge about gaze direction, video compression, etc, an accurate head pose (position and orientation) estimation is an important issue.

This chapter deals with the problem of estimate the tridimensional orientation and position of faces (6DOF) using a non-intrusive solution from a single camera (monocular) video. The proposed approach is based on considering the human head as a rigid body structure. A statistical anthropometric 3D rigid model is used with Pose from Orthography and Scaling with ITerations (POSIT) [13] algorithm for head pose estimation. Since POSIT estimates pose by a set of 3D model points and 2D image projections correspondences, a way to extract facial characteristics and perform the associations is required. For that purpose, a model-based approach for the interpretation of face images, Active Appearance Models (AAM) [58], is used. The AAM fitting procedure provides an effective way to locate facial features on 2D images [6]. Given a sufficient representative training set, AAM has the key advantage of fitting an unseen person, modeling non-rigid deformations fully describing facial characteristics.

Monocular head pose estimation is achieved combining the tracking of facial features with POSIT, using for that purpose a 3D anthropometric head model of human head. Due to the AAM landmark-based nature, the 3D model / 2D correspondences registration problem is easily solved.

This thesis part is organized as follows: chapter 13 refers the POSIT algorithm, in chapter 14 the combined solution AAM plus POSIT is described and experimental results are shown. Finally in chapter 15 an Augmented Reality (AR) application that consists on a 3D virtual glasses augmentation using the monocular head pose estimation system.

12.1 Related Work

Human head tracking and pose estimation has received much attention in past years. Several approaches addressed to this problem were been proposed.

Birchfiled [9] uses a 2D ellipse to approximate the head position in image obtained using color histograms or image gradients. The advantage of such approach is that it is extremely fast. However, light changes and different color skins result in tracking failures. In [59], [46], partial head orientation (tilt or pan) is recovered. The accuracy of those systems is up to 15 degree rotation estimates.

More accurate systems use 3D geometrical models to represent the head as a rigid body [41]. Recent approaches, such as [32] , use a cylinder to approximate both head underlying geometry and texture. A linear combination of motion adaptive templates is used to match the model against current image. Since a cylinder is only a rough approximation to head geometry, those methods suffer from inaccuracies in estimating rotation, and have difficulties differentiating between small rotations and translations. [66] use Nurbs surface with texture to synthesize both appearance and pose.

From the monocular point of view, estimate the head pose, consists on recovering the camera position and relative orientation to a known set of 3D points. Numerous algorithms exist, given a set of 3D points and their matched 2D features, such that Pose from Orthography and Scaling with ITerations (POSIT) [13], Perspective-n-Point (PnP) [42], [4], [68]. In principle, any solution to the camera pose estimation algorithm that uses only the relative distances, can be used to track rigid body objects.

Recently, other solutions that estimate the head pose by template matching as been proposed. Methods such as Gabor Wavelet [61] or Support Vector

Machines [36]. The principal advantage of these is that they rely on locating the face in the image, but have the disadvantage of resulting on pour accuracy when compared to methods based on correspondences between a 3D model and 2D image projections [45], [37].

Chapter 13

Pose from Orthography and Scaling with ITerations

Pose from Orthography and Scaling with ITerations (POSIT) [13] is a fast and accurate iterative algorithm for finding the 6DOF pose (orientation and translation) of a 3D model or scene with respect to a camera given a set of 2D image and 3D object points correspondences.

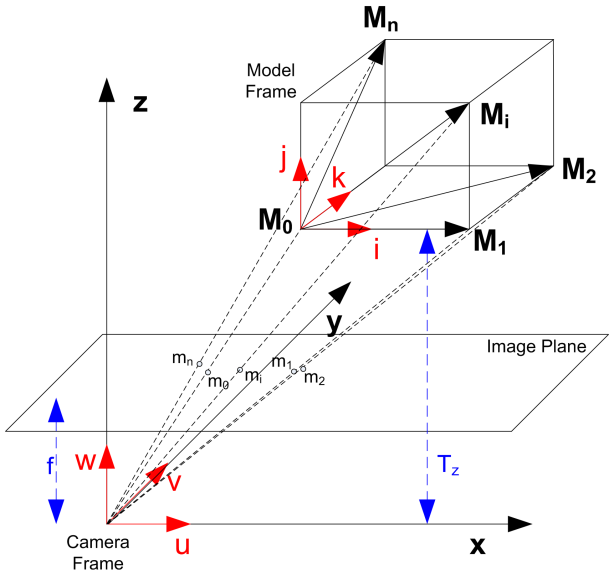


Figure 13.1: Perspective projections m_i for model points M_i.

Figure 13.1 shows the pinhole camera model, with its center of projection, O, and image plane at the focal length, f, (focal length and image center are assumed to be known).

A 3D model with feature points $M_0, M_1 \ldots, M_i, \ldots, M_n$ is positioned at camera *frustrum*. The model coordinate frame is centered at M_0. A point M_i has known coordinates (X_i, Y_i, Z_i) in the model frame and unknown coordinates in the camera frame. The image projections of M_i are known and called m_i, having image coordinates (u'_i, v'_i).

In a perspective projection model a 3D point (X, Y, Z) is projected in image plane by

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \mathbf{K} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left[\begin{array}{c|c} \mathbf{R} & \mathbf{T} \\ \hline \mathbf{0}_{1\times 3} & 1 \end{array}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{13.1}$$

where $\mathbf{R}$ is the rotation matrix representing the orientation of the camera frame with respect to the world frame. $\mathbf{T}$ is the translation vector from the camera center O to M_0 expressed in the camera frame. $\mathbf{0}$ is a matrix of zeros and $\mathbf{K}$ is the camera matrix with f the focal distance and (c_x, c_y) the principal point. $\mathbf{K}$ is supposed to be known.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \text{ and } \mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{13.2}$$

For using normalized image coordinates the following transformation is applied

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \mathbf{K}^{-1} \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix}, \tag{13.3}$$

eq 13.1 becomes

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{13.4}$$

and the projection matrix is now given by $\mathbf{P} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{T} \end{array}\right]$. Solving the Pose problem consists on finding $\mathbf{R}$ and $\mathbf{T}$ matrix that fully describe the 6 DOF.

Defining $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$, as

$$\mathbf{r}_1 = \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix}, \quad \mathbf{r}_2 = \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}, \quad \mathbf{r}_3 = \begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \end{bmatrix}, \tag{13.5}$$

eq 13.4 can be written as

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1^T & T_x \\ \mathbf{r}_2^T & T_y \\ \mathbf{r}_3^T & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{13.6}$$

or, dividing all projection matrix elements by T_z,

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1^T/T_z & T_x/T_z \\ \mathbf{r}_2^T/T_z & T_y/T_z \\ \mathbf{r}_3^T/T_z & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{13.7}$$

From the third equation on the previous system 13.7, gives

$$w_i = 1 + \frac{\mathbf{r}_3}{T_z}(X_i, Y_i, Z_i), \tag{13.8}$$

and applying its transpose on the remaining two equations

$$\begin{bmatrix} u & v \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_1/T_z & \mathbf{r}_2/T_z \\ T_x/T_z & T_y/T_z \end{bmatrix}. \tag{13.9}$$

Using n points, eq 13.9 is extended to

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ \vdots & \vdots \\ u_{n-1} & v_{n-1} \\ u_n & v_n \end{bmatrix} = \underbrace{\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_{n-1} & Y_{n-1} & Z_{n-1} & 1 \\ X_n & Y_n & Z_n & 1 \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} \mathbf{r}_1/T_z & \mathbf{r}_2/T_z \\ T_x/T_z & T_y/T_z \end{bmatrix} \tag{13.10}$$

where $\mathbf{M}$ is the model matrix that defines the structure of the 3D model used and (u_i, v_i) are the projected image points coordinates of (X_i, Y_i, Z_i).

Solving eq 13.10 for the pose parameters gives

$$\begin{bmatrix} \mathbf{r}_1/T_z & \mathbf{r}_2/T_z \\ T_x/T_z & T_y/T_z \end{bmatrix}_{4\times 2} = \mathbf{M}^{-1}_{4\times n} \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ \vdots & \vdots \\ u_{n-1} & v_{n-1} \\ u_n & v_n \end{bmatrix}_{n\times 2} . \tag{13.11}$$

It is straightforward to retrieve T_z, T_x, T_y, $\mathbf{r}_1$ and $\mathbf{r}_2$. Since the rotation matrix rows are orthogonal $\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$ and the pose becomes fully defined. Note that, at least 4 non coplanar correspondence points is required, otherwise the matrix $\mathbf{M}$ is singular. This approach is called Pose from Orthography and Scaling (POS) [13], i.e. finding pose for fixed values of w_i.

The perspective image (u_i, v_i) of a 3D world point is related to the image (u_i^w, v_i^w) produced by a scaled orthographic camera according to

$$\begin{aligned} u_i^w &= w_i u_i, \\ v_i^w &= w_i v_i. \end{aligned} \tag{13.12}$$

The term can be determined only if the camera pose is already known using eq 13.8. The POSIT (POS with ITerations) [13] algorithm algorithm starts by assuming that the perspective image points are identical to the scaled orthographic image points, so that $w_i = 1, i = 1, \ldots, n$. Under this assumption, the camera pose can be determined by solving a linear system of equations 13.11. This solution is only approximate since $w_i = 1$ is also an approximation. However, given a more accurate estimate of the object's pose, the accuracy of the w_i terms can be improved by reestimaing these terms using eq 13.8. The scaled orthographic perspective model is used iteratively in the process of computing the full perspective pose. This process is repeated until the pose converges. The steps of POSIT are described in algorithm 9. This method does not require an initial pose estimate, is very fast (it converges in about four iterations) and is robust with respect to image measurements and camera calibration errors.

The image registration problem, i.e. the image and model points correspondences, while using POSIT combined with AAM is a straightforward problem as it is described on the next chapter.

Algorithm 9 POSIT

Normalize image coordinates $u_i = u_i - \frac{c_x}{f}$, $v_i = v_i - \frac{c_y}{f}$

Compute model matrix pseudoinverse $\mathbf{M}^\dagger$

Assume $\mathbf{w}_i = 1$

loop

  Get Scaled Orthographic coordinates $(u_i, v_i) = w_i(u_i, v_i)$

  Compute $\begin{bmatrix} \mathbf{r}_1/T_z & \mathbf{r}_2/T_z \\ T_x/T_z & T_y/T_z \end{bmatrix} = \mathbf{M}^\dagger \begin{bmatrix} u_1 & v_1 \\ u_i & v_i \\ u_n & v_n \end{bmatrix}$

  Find T_z, T_x, T_y, $\mathbf{r}_1$ and $\mathbf{r}_2$

  $\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$

  Update $w_i = 1 + \frac{\mathbf{r}_3}{T_z}(X_i, Y_i, Z_i)$

  if $w_i - w_{i-1} < \Delta$ **then**

    Retrieve $\mathbf{R} = \begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \mathbf{r}_3^T \end{bmatrix}$ and $\mathbf{T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$

    Exit Loop

  end if

end loop

13.1 Scaled Orthographic Projection Model

Let the i^{th} row of $\mathbf{R}$ be denoted by $\mathbf{R}_i$ and using normalized image coordinates, the perspective image of a 3D point P in the world frame, (u, v), is

$$u = \frac{\mathbf{R}_1 P + T_x}{\mathbf{R}_3 P + T_z}, \quad v = \frac{\mathbf{R}_2 P + T_x}{\mathbf{R}_3 P + T_z}. \tag{13.13}$$

The weak perspective (also known as scaled orthographic) projection model [40], which makes the assumption that the depth of an object is small compared to the distance of the object from the camera and that visible scene points are close to the optical axis, has the assumption that $\mathbf{R}_3 \cdot P \approx 0$, since $\mathbf{R}_3$ is a unit vector in the world coordinate frame that is parallel to the camera's optic axis. The weak perspective image of a 3D point in the world frame is (u^w, v^w) where

$$u^w = \frac{\mathbf{R}_1 P + T_x}{T_z}, \quad v^w = \frac{\mathbf{R}_2 P + T_x}{T_z}. \tag{13.14}$$

Chapter 14

Head Pose Estimation

The full automatic framework for head pose extraction is composed by the two parts previously described. An AAM model fitting is performed on a subject leading to shape model landmarks location tracking over time. Notice that no temporal filter is used. In certain occasions the model fitting fails, especially when the movements of the head are particularly fast, and is assumed to be a failure if any of the appearance parameters don't follow condition 7.2. The recovery from lost track is overcome by reinitializing the appearance-based detection process each time the model fails.

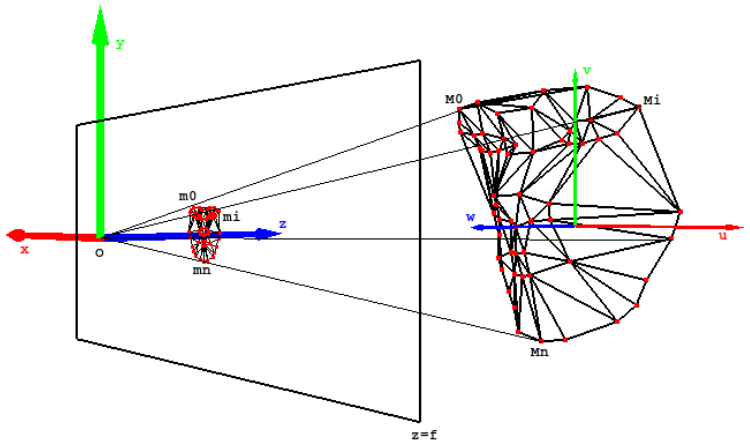
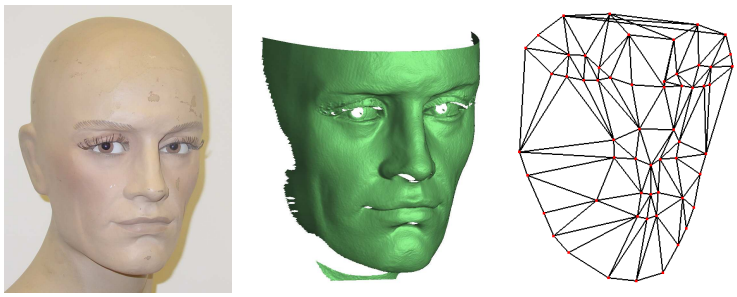


Figure 14.1: Anthropometric head used as POSIT 3D model.

14.1 Anthropometric 3D Model

The head pose estimation is performed using POSIT. As 3D model, an anthropometric 3D rigid model of the human head its used, see figure 14.1, since it is the best suitable rigid body model that describes the 3D face surface of several individuals. It was acquired by a frontal laser 3D scan of an physical model, selecting the equivalent 3D points of the AAM annotation procedure creating a sparse 3D model. Figure 14.2 illustrates this procedure.



(a) Physical anthropometric model (b) 3D laser scan data (c) Sparse OpenGL model

Figure 14.2: 3D anthropometric model acquisition. a) Physical model used. b) Laser scan data acquired c) OpenGL built model using the AAM shape features.

By tracking features in each video frame combined with the landmark-based nature of AAMs, the image/3Dmodel registration problem required for the use of POSIT is easily solved (one-to-one point correspondences).

14.2 Experimental Results

The orientation of the estimated pose is represented by the Roll, Pitch and Yaw (RPY) angles,

$$\mathbf{R}_{RPY}(\alpha, \beta, \gamma) = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_x(\gamma). \tag{14.1}$$

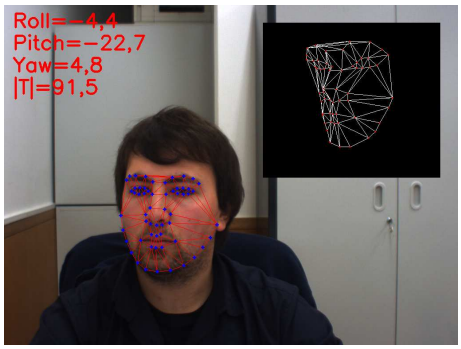
Solving RPY inverse kinematics comes,

$$\mathbf{R}_{R(\alpha)P(\beta)Y(\gamma)} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\alpha & c\beta s\gamma & c\alpha c\gamma \end{bmatrix} \tag{14.2}$$

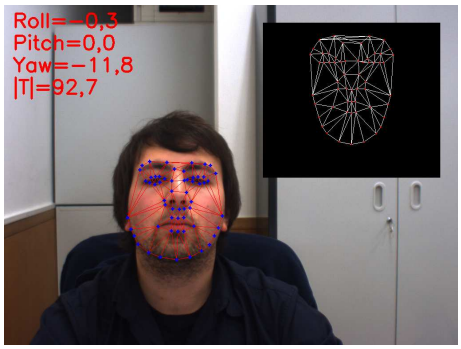
with the solutions,

$$\begin{aligned} \alpha = &\quad atan\left(\frac{r_{21}}{r_{11}}\right) \\ \beta = &\quad atan\left(\frac{-r_{31}}{r_{11}}\right) \times \cos(\alpha) + r_{21}\sin(\alpha) \\ \gamma = &\quad atan\left(\frac{r_{32}}{r_{33}}\right). \end{aligned} \tag{14.3}$$

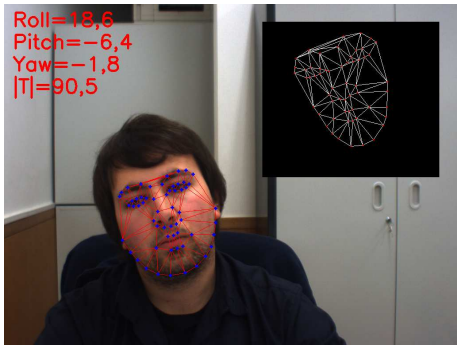
Figure 14.3 shows some examples of pose estimation where the pose is represented by an animated 3DOF rotational OpenGL model showed at images top right. This model, used only for display, follows the subject head rotations, ignoring translational effects.



(a) Pitch variation



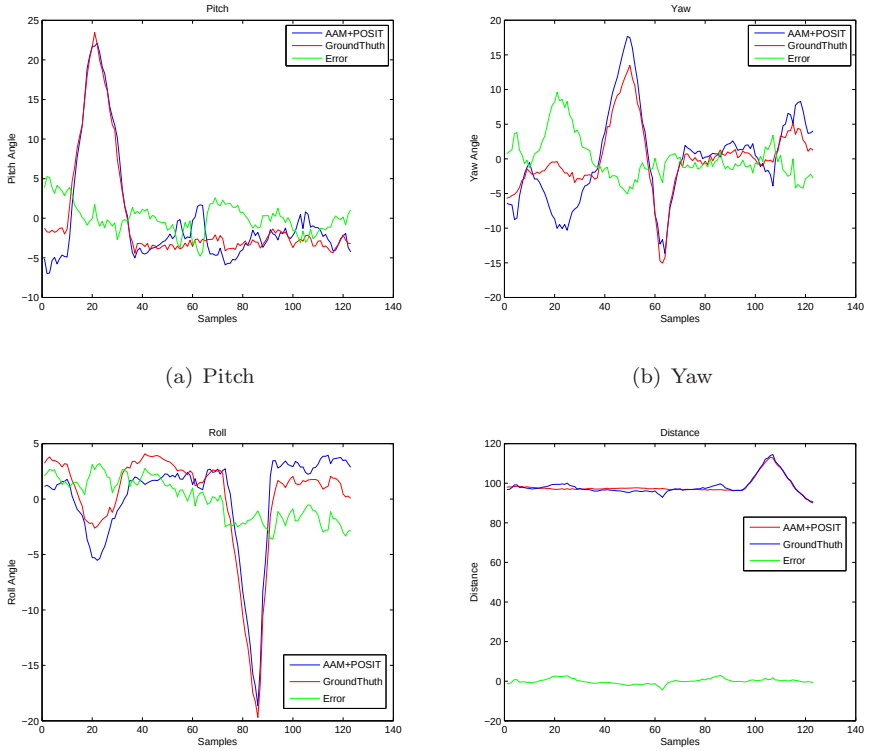
(b) Yaw variation



(c) Roll variation

Figure 14.3: Samples of pose estimation.

The evaluation of pose estimation accuracy is performed comparing the pose estimated with the one estimated from a planar checkerboard [31], used as ground truth reference values. Figure 14.4 presents results from the pose estimated during a video sequence where the subject performs several human head movements, ranging from yaw, pitch and roll head rotations of several degrees (during a total of 140 frames). The experience began by rotating head left, changing pitch angle, and recovering to frontal position, followed by a yaw angle, moving head up and down and again recovering to frontal position, and finally performing a head roll rotation. Near the end, after frame 95 the distance from camera is also changed. The individual parameters (Pitch, Yaw, Roll and distance) results are presented in figure 14.4-a, 14.4-b, 14.4-c and 14.4-d respectively. Figures 14.5, 14.6, 14.7, 14.8 and 14.9 show results for similar pose evaluation experiments.



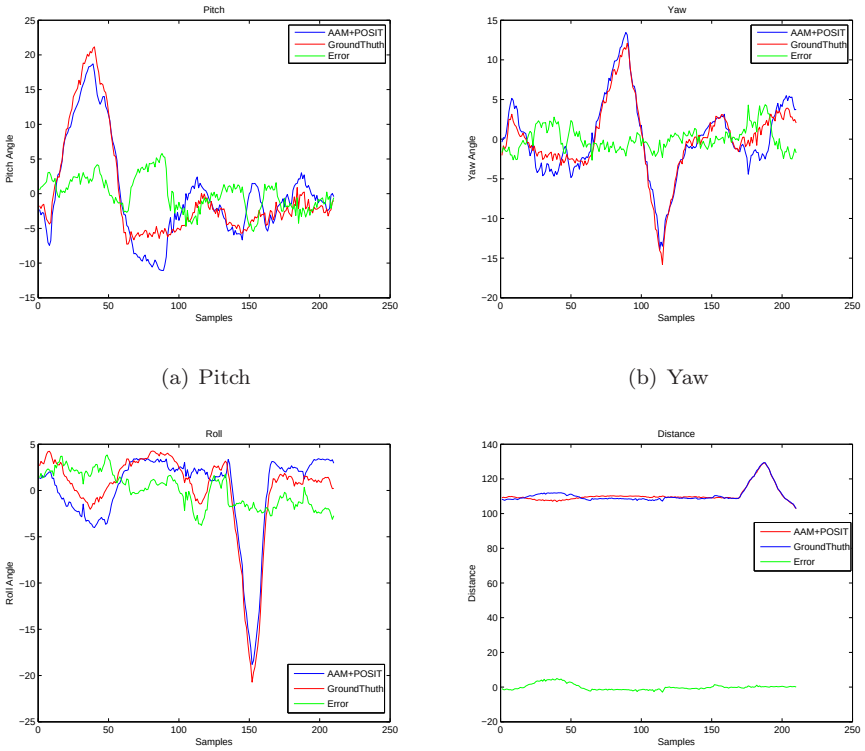
(a) Pitch

(b) Yaw

(c) Roll

(d) Distance

Figure 14.4: Pose evaluation, results from experiment 1.



(a) Pitch

(b) Yaw

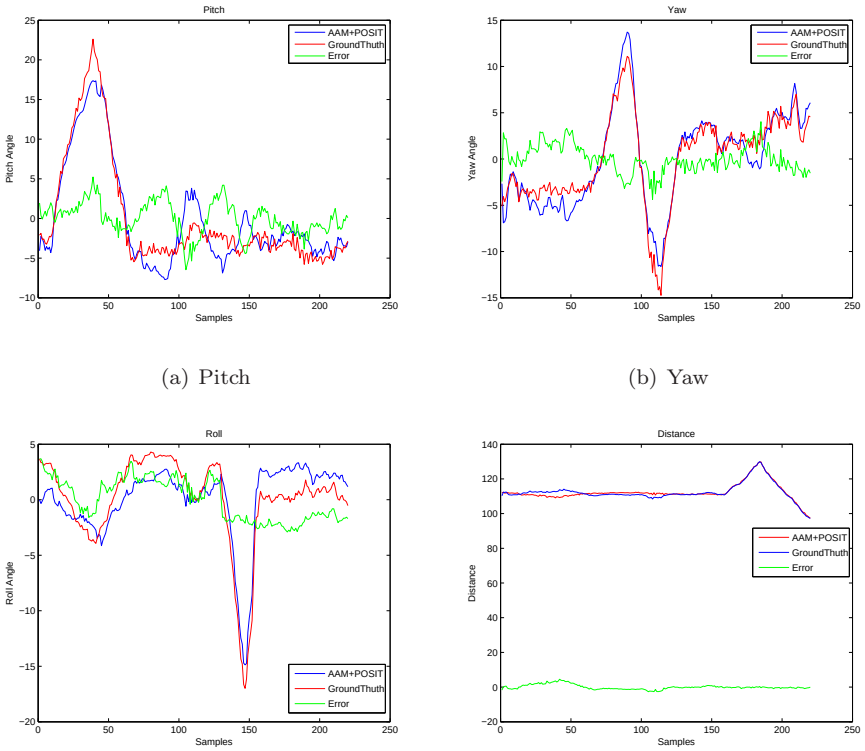
(c) Roll

(d) Distance

Figure 14.5: Pose evaluation, results from experiment 2.

The graphical results show correlations between Pitch and Yaw angles that result from the differences between the subject and the rigid 3D anthropometric model used. Table 14.1 displays the errors average standard deviations of the pose parameters for six similar performed experiences with different individuals.

The application with AAM model fitting plus POSIT pose estimation runs at 5 frames/s on 1024×768 images using a 3.4 GHz P4 Intel Processor under Linux OS. AAM is based on a 58 landmark shape points ($n = 58$), sampling 48178 pixels with color information ($m = 48178 \times 3 = 144534$) by OpenGL hardware-assisted texture mapping using a Nvidia GeForce 7300 graphics board.



(a) Pitch

(b) Yaw

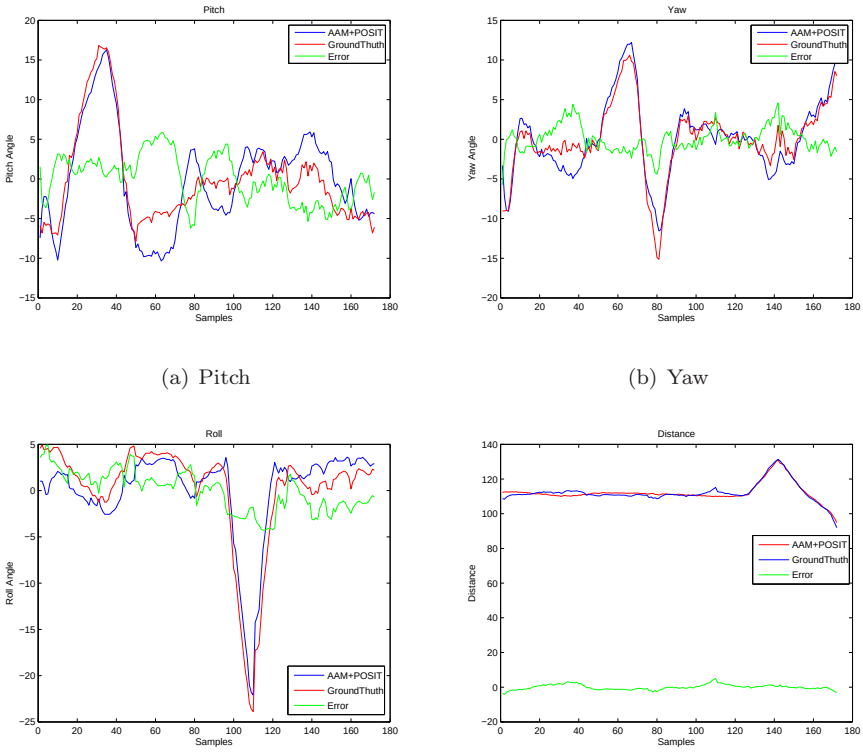
(c) Roll

(d) Distance

Figure 14.6: Pose evaluation, results from experiment 3.

14.3 Conclusions

This chapter describes a single view solution to estimate the head pose of human subjects combining AAM and POSIT. AAM extracts in each image frame the landmarks position. These selected features are tracked over time and used in conjunction with POSIT to estimate head pose. Since the solution requires the use of a 3D rigid model, a statistical anthropometric model is selected since is the most suitable one. One of the major advantage of using combined AAM plus POSIT is that it solves directly the correspondences problem, avoiding the use of registration techniques. An accurate pose estimation is achieved with average standard deviations about 2 degrees in orientation and 1 centimeter in distance and subjects exhibiting a normal expression.



(a) Pitch

(b) Yaw

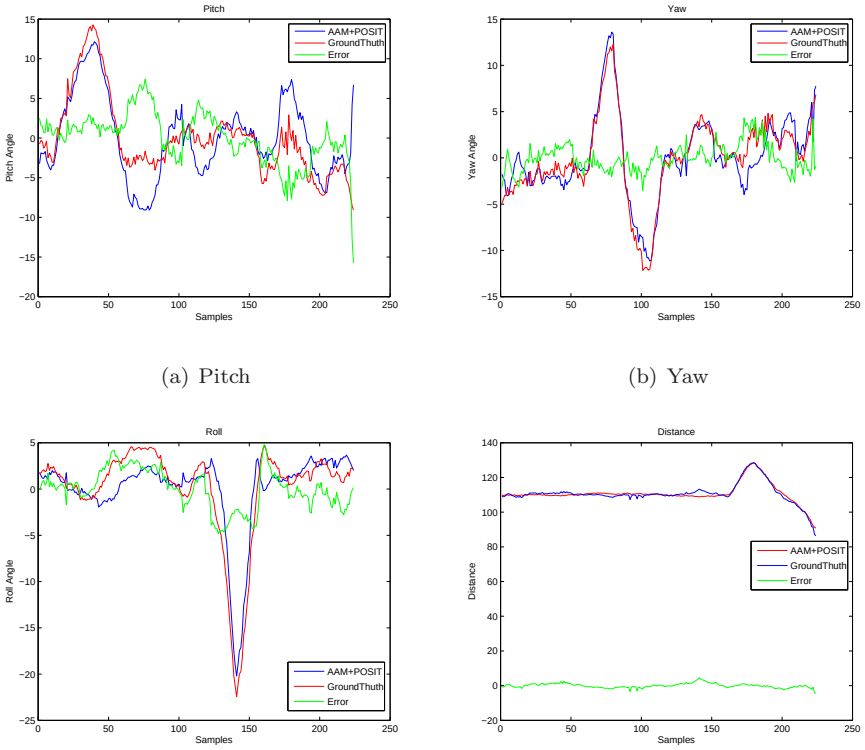
(c) Roll

(d) Distance

Figure 14.7: Pose evaluation, results from experiment 4.

Table 14.1: Error standard deviation. The angle parametes are in degrees and the distance in centimeters. Each column of the results shown are related to images 14.4 to 14.9 respectively.

Parameters	**Experiences error std**						**Avg std**
Roll	1.9175	1.8569	1.8715	2.1543	2.1389	1.6935	1.9388^o
Pitch	1.9122	2.4645	2.0985	2.9398	3.2278	2.8053	2.5747^o
Yaw	3.0072	1.4661	1.4673	1.6393	1.4884	1.1435	1.7020^o
Distance	1.2865	1.7224	1.3744	1.5041	1.2956	0.8475	$1.3384cm$



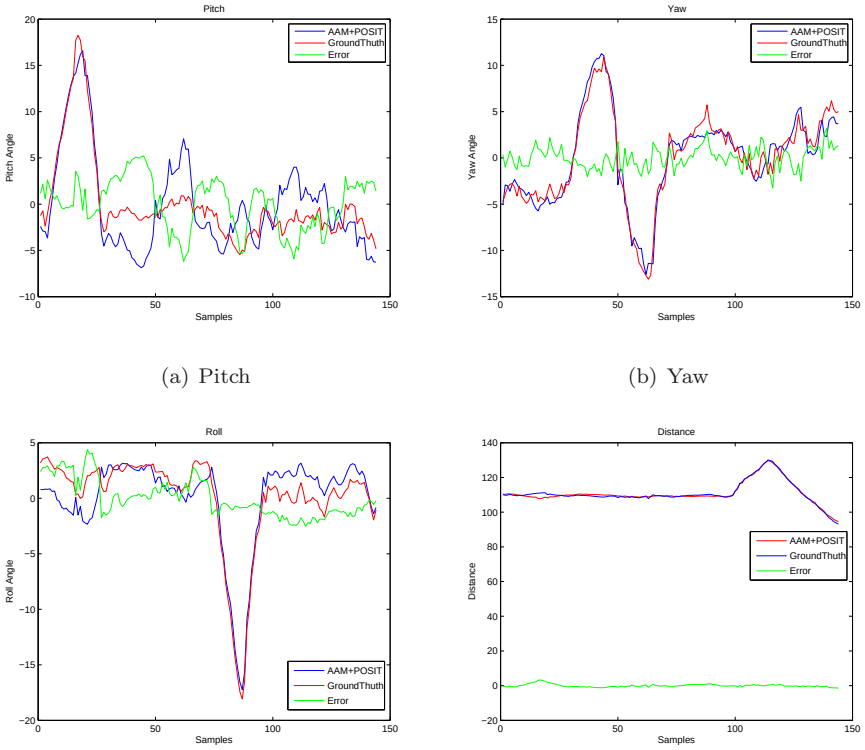
(a) Pitch

(b) Yaw

(c) Roll

(d) Distance

Figure 14.8: Pose evaluation, results from experiment 5.



(a) Pitch

(b) Yaw

(c) Roll

(d) Distance

Figure 14.9: Pose evaluation, results from experiment 6.

Chapter 15

Augmented Reality Application

This section describes an Augmented Reality (AR) application where a 3D virtual glasses is inserted into a subject face without the use of any kind of markers our fiducials. AR consists on combining real world information with computer generated data, where pose estimation (translation and orientation) is an crucial issue. As accurate is the pose estimation, better is the model backprojection on the target image.

In this application, the face pose is extracted using POSIT as a rigid anthropometric model being a 3D approximation of the subject face and the correspondent image features are tracked by the AAM fitting framework in each image frame, see in chapter 14.

15.1 What's Augmented Reality

Augmented Reality (AR) is the overlay of artificial computer graphics images on the physical world. Virtual data is combined with real world information. Unlike, in a Virtual Reality (VR) concept, all the information is purely computer generated graphics.

Combining real information with virtual one normally requires some *a priori* knowledge about the world or the use of markers. As better is the pose

estimation quality, better the augmented objects fit the marker. Figure 15.1 displays an AR example, consisting on a cube augmentation over a chessboard plane marker.

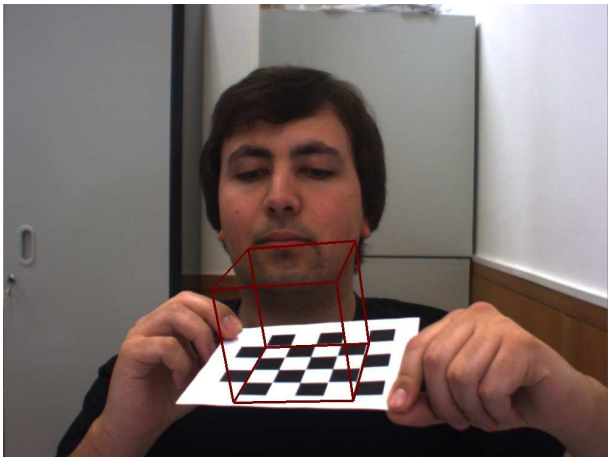


Figure 15.1: Augmented Reality example.

15.2 3D Glasses Augmentation

An AR system where virtual glasses are inserted on the subjects face is described in the following. To support this propose, an accurate human head pose estimation is required. Monocular pose estimation is achieved as described in chapter 14, where a statistical rigid anthropometric model is used in POSIT as an approximation of the subject 3D face surface. Facial features, i.e. shape landmarks, are retrieved using AAM fitting in each frame.

The monocular pose estimation system extracts the 6DOF pose from the head model, i.e. the six pose parameters, $(t_x, t_y, t_z, \alpha, \beta, \gamma)$, being $[t_x, t_y, t_z]^T$ the translation vector and (α =Roll, β =Pitch, γ =Yaw) the rotation angles using a RPY orientation representation. Mapping the camera captured image combined with the 6DOF animation of the rigid head model in a `OpenGL` application, a system where this rigid face model reacts to the user pose variation is achieved. Image 15.2 displays the 3D anthropometric model overlaid with the captured image.

Including 3D glasses model requires drawing-it with respect to the head

model. See figure 15.3. Similarly, the final AR application consists on animating only the glasses model. Image 15.4 shows several views of the subject with glasses augmentation with different poses.

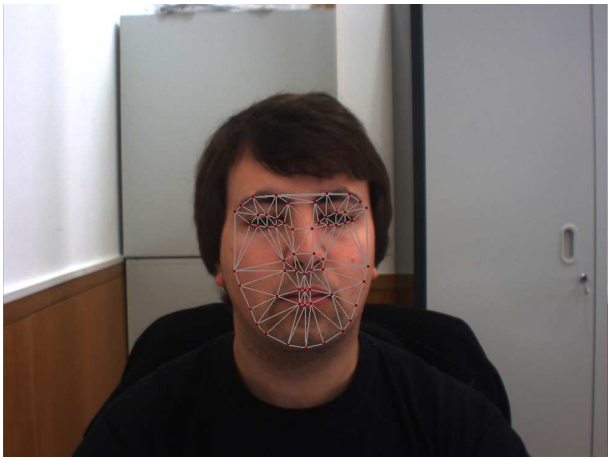


Figure 15.2: 3D anthropometric model overlaid with input image.

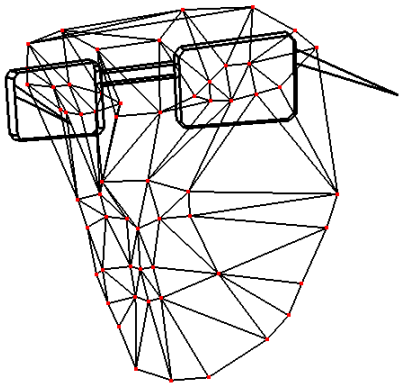


Figure 15.3: Glasses inserted on anthropometric 3D model.

The AR application runs at 3 frames/s on 1024×768 images with the same AAM model conditions than section 14.

Figure 15.4: 3D glasses augmentation.

Part IV

Final Notes

Chapter 16

Conclusion

Advanced issues on facial recognition and pose estimation requires basic face handling. Face model-based approaches were used to overcome the large variation in shape and texture that the human face presents, namely differences in identity, deformations by facial expression, pose and illumination changes.

Standard formulation of Active Appearance Models was described in detail. AAM has the capability of analyse the variation of different individuals on a training set and mimic this variation in the model fitting process. The shape of faces is modelled, where a large dataset of shape annotation are aligned into a common mean removing location, scale and rotation effects. Shape variation is then analyzed, producing a parametrized linear model. AAM texture model is an improvement to EigenFaces [60] approach, where affine texture mapping is used to remove texture differences due shape changes. Similarly to the shape models, after texture alignment, a Principal Component Analysis is used to model texture variation. Shape and texture models, regarding the nature of different parameters units, are combined in to single coupled model. Fitting an AAM model to a target image is a nonlinear optimization problem where the texture residuals are minimized by additive updates to the model parameters. The training stage consists in learning the correlations between AAM model instances and texture residuals. A fast and efficient damped Gauss-Newton Steepest Descent based on a fixed Jacobian matrix was achieved. However convergence is very dependent from the initial estimate. Monte Carlo simulations

show successful results on a range of $[-40, 40]$ unit pixels over x and y-axis on 640×480 images. AAM fitting initialization is performed using AdaBoost for face detection.

To facial recognition proposes, AAM model was used to describe face proprieties in a compact way, using the appearance parameters as classification features. Two approaches were used to classify facial emotions. Several classes were separated using a Linear Discriminant Analysis projecting each appearance vector into Fisherspace, and classification was performed based on Mahalanobis distance. Similarly the same facial expressions were classified projecting the AAM appearance parameters to the hyperplane that maximizes class separability using a multiclass one-against-all Support Vector Machines. Naturally the larger the number of expressions used, the worse is the overall successful classification rate. With all the seven expressions, the LDA (61.2%) approach achieved an slight better overall recognition rate. Since the results emphasis appearance correlations between the two pairs of expressions neutral, sad and anger, disgust, these expressions were removed. Classifying five expressions shows that SVM method achieves a better recognition rate, about 80%. A promissing method to overcome emotions appearance correlations, could be to include also a temporal facial dynamics analysis (using for instance Hidden Markov Models (HMM)).

A single view solution to estimate the head pose of human subjects combining AAM and Pose from Orthography and Scaling with ITerations, was proposed. AAM extracts in each image frame the landmarks position. These selected features are tracked over time and used in conjunction with POSIT to estimate head pose. Since the solution requires the use of a 3D rigid model, an anthropometric model is selected because is the most suitable one, statistically describing the face of a several individuals. One of the major advantage of using combined AAM plus POSIT is that it solves directly the correspondences problem, avoiding the use of registration techniques. An accurate pose estimation is achieved with average standard deviations about 2 degrees in orientation and 1 centimeter in distance on subjects exhibiting a normal expression, being accurate enough for the developed 3D glasses Augmented Reality application.

Bibliography

[1] *The Nature of Statistical Learning Theory*. Springer-Verlag, N.Y., 1995.

[2] *Pairwise classification and suportvector machines*. Advances in Kernel Methods - Support Vector Learning. Cambridge, MA, MIT Press, 1999.

[3] *Advances in Neural Information Processing Systems*. MIT Press, 2000.

[4] *Multiple view geometry in computer vision*. Cambridge University Press, 2000.

[5] Vadim Pisarevsky Adrian Kaehler, Adrian Kaehler. Learning-based computer vision with intels open source computer vision library. *Intel Technology Journal*, 2005.

[6] J. Ahlberg. An active model for facial feature tracking. *EURASIP Journal on Applied Signal Processing*, 2002.

[7] Monson H. Hayes Aziz Umit Batur. Adaptive active appearance models. *IEEE Transactions on Image Processing*, 2005.

[8] T.Kanade B. Lucas. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*.

[9] Stan Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[10] V. Blanz and T Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH'99*.

[11] F. L. Bookstein. Landmark methods for forms without landmarks: localizing group diferences in outline shape. In *Medical Image Analysis*.

[12] Mo Dang Bouchra Abboud, Frank Davoine. Facial expression recognition and synthesis based on an appearance model. *Signal Processing Image Communication*, 2004.

[13] D. DeMenthon and L.S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 1995.

[14] Friesen W.V. Hager J.C. Ekman, P. Facial action coding system. 1978.

[15] I. Essa and A. Pentland. Coding, analysis, interpretation and recognition of facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997.

[16] Peter H. N. de With Fei Zuo. Fast facial feature extraction using a deformable shape model with haar-wavelet based local texture attributes.

[17] N.; Duta N.; Carstensen J.M.; Fisker, R.; Schultz. A general scheme for training and optimization of the grenander deformable template model. In *Computer Vision and Pattern Recognition*.

[18] G. Schaefer G. Finlayson, S. Hordley and G. Y. Tian. Illuminant and device invariant color using histogram equalization. In *Pattern Recognition*.

[19] M. Stewart Bartlett G. Littlewort, I. Fasel and J. R. Movellan. Fully automatic coding of basic expressions from video. Technical report, UCSD INC MPLab, 2002.

[20] K. Ikeuchi H. Shum and R.Reddy. Principal componet analysis with missing data and its application to polyhedral object modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995.

[21] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 2002.

[22] C.J. Taylor I.M. Scott, T.F. Cootes. Improving appearance model matching using local image structure. In *Medical Imaging Processing*.

[23] Silicon Graphics Inc. Opengl - open source computer graphics library. *http://www.opengl.org/*.

[24] Intel. Opencv - open source computer vision library. *http://www.intel.com/technology/computing/opencv/index.htm*.

[25] J. J. Lien J. F. Cohn, A. J. Zlochower and T. Kanade. Feature-point tracking by optical flow discriminates subtle diferences in facial expression. In *Proceedings International Conference on Automatic Face and Gesture Recognition*.

[26] C. Watkins J. Weston. Multi-class support vector machines. In *Proceddings of ESANN99*.

[27] I. Matthews J. Xiao, S. Baker and T. Kanade. Real-time combined 2d+3d active appearance models. *IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[28] Michael Kass. Snakes: Active contour models. *International Journal of Computer Vision*, 1987.

[29] William D.S. Killgore and Deborah A. Yurgelun-Todd. Activation of the amygdala and anterior cingulate during nonconscious processing of sad versus happy faces. *NeuraImage*, 2003.

[30] W. J. Krzanowski. Principles of multivariate analysis. *Oxford University Press*, 1988.

[31] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends in Computer Graphics and Vision*, 2005.

[32] J. Isidoro M. La Cascia and S. Sclaroff. Head tracking via robust registration in texture map images. In *CVPR*.

[33] I. Matthews and S. Baker. Active appearance models revisited. *International Journal of Computer Vision*, 2004.

[34] Janusz Sierakowski Michael M. Nordsom, Mads Larsen and Mikkel Bille Stegmann. The imm face database - an annotated dataset of 240 face images. Technical report, Informatics and Mathematical Modelling, Technical Univesity of Denmark.

[35] Narendra Ahuja Ming-Hsuan Yang, David J. Kriegman. Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[36] J. Ng and S. Gong. Multi-view face detection and pose estimation using a composite support vector machine across the view sphere, 1999.

[37] Shay Ohayon and Ehud Rivlin. Robust 3d head tracking using camera pose estimation. In *International Conference of Pattern Recognition*.

[38] M. Pantic and L. J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[39] Joao P. Hespanha Peter N. Belhumeur and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997.

[40] Ramani Duraiswami Philip David, Daniel DeMenthon and Hanan Samet. Simultaneous pose and correspondence determination using line features. In *Computer Vision and Pattern Recognition*.

[41] Francoise Preteux and Marius Malciu. Model-based head tracking and 3d pose estimation. Technical report.

[42] Long Quan and Zhong-Dan Lan. Linear n-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1999.

[43] Stegmann M.B; Larsen R. Multi-band modelling of appearance. In *Image and Vision Computing*.

[44] I. Matthews R. Gross and S. Baker. Active appearance models with occlusion. *Image and Vision Computing*, 2006.

[45] Alex Waibel Rainer Stiefelhagen, Jie Yang. A modelbased gaze tracking system. In *IEEE International Joint Symposia on Intelligence and Systems*.

[46] Daniel B. Russakoff and Martin Herman. Head tracking using stereo. Technical report, 2002.

[47] I. Matthews S. Baker. Equivalence and efficiency of image alignment algoritms. In *Computer Vision and Pattern Recognition*.

[48] Stan Sclaroff and John Isidoro. Active blobs. In *International Conference on Computer Vision*.

[49] Gereon Frahling; Christian Sohler. A fast k-means implementation using coresets. In *Proceedings of the twenty-second annual symposium on Computational geometry*.

[50] M. B. Stegmann. Bi-temporal 3d active appearance modelling. In *IAVP - Image Analysis in Vivo Pharmachology, Roskilde, Denmark*.

[51] Mikkel B. Stegmann. *Generative Interpretation of Medical Images*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2004.

[52] Mikkel Bille Stegmann. Active appearance models theory, extensions & cases. Master's thesis, IMM Technical Univesity of Denmark, 2000.

[53] Mikkel Bille Stegmann and David Delgado Gomez. A brief introduction to statistical shape analysis. Technical report, Informatics and Mathematical Modelling, Technical Univesity of Denmark, 2002.

[54] T.F.Cootes and C.J.Taylor. Constrained active appearance models. In *International Conference on Computer Vision*.

[55] T.F.Cootes and C.J.Taylor. Statistical models of appearance for computer vision. Technical report, Imaging Science and Biomedical Engineering - University of Manchester, 2004.

[56] C.J.Taylor T.F.Cootes. Active shape models - smart snakes. In *British Machine Vision Conference*.

[57] C.J.Taylor T.F.Cootes. An algorithm for tuning an active appearance model to new data. In *British Machine Vision Conference*.

[58] G.J. Edwards T.F.Cootes and C.J.Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[59] Baback Moghaddam Trevor Darrell and Alex P. Pentland. Active face tracking and pose estimation in an interactive room. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

[60] M. Turk and A. Pentland. Face recognition using eigenfaces. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[61] V. uger, S. Bruns, and G. Sommer. Efficient head pose estimation with gabor wavelet networks, 2000.

[62] Paul Viola and Michael Jones. Rapid object detection using a boosted cascate of simple features. In *Computer Vision and Pattern Recognition*.

[63] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 2001.

[64] Max Welling. Fisher linear discriminant analysis. Technical report, Department of Computer Science, University of Toronto.

[65] S.Z.; HongJiang Zhang; QianSheng Cheng XinWen Hou; Li. Direct appearance models. In *Computer Vision and Pattern Recognition*.

[66] C. Stratelos X.L.C. Brolly. Model-based head pose estimation for air-traffic controllers. In *ICIP*.

[67] T. Kanade Y.-L. Tian and J. F. Cohn. Recognizing action units for facial expression analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[68] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

Appendix A

Pose Estimation from a Plane

In this appendix, a procedure to estimate the pose (orientation and translation) of a plane in a image, assuming a calibrated camera [31], is presented.

Knowing the camera camera matrix $\mathbf{K}_{3\times3}$, the transformation between the image plane and a world plane is given by the homography matrix.

The homograghy matrix can be decomposed by

$$\mathbf{H} = \mathbf{K}[\mathbf{R}_1|\mathbf{R}_2|\mathbf{T}] \tag{A.1}$$

where $\mathbf{R}_1$, $\mathbf{R}_2$ represents the first two columns of the rotation matrix $\mathbf{R}$ and $\mathbf{T}$ is the translation vector.

These can be retrieved from the product $\mathbf{W} = \mathbf{K}^{-1}\mathbf{H}$ using the following normalization [31]:

$$\mathbf{R}_1 = \frac{\mathbf{W}_1}{l}, \qquad \mathbf{R}_2 = \frac{\mathbf{W}_2}{l}, \qquad \mathbf{T} = \frac{\mathbf{W}_3}{l} \tag{A.2}$$

with

$$l = \sqrt{|\mathbf{W}_1||\mathbf{W}_2|}. \tag{A.3}$$

Defining the vectors $\mathbf{c}, \mathbf{p}$ and $\mathbf{d}$ as

$$\mathbf{c} = \mathbf{R}_1 + \mathbf{R}_2, \qquad \mathbf{p} = \mathbf{R}_1 \times \mathbf{R}_2, \qquad \mathbf{d} = \mathbf{c} \times \mathbf{p}, \tag{A.4}$$

the normalized columns of the rotation matrix are

$$\mathbf{R}'_1 = \frac{1}{\sqrt{2}}\left(\frac{\mathbf{c}}{|\mathbf{c}|} + \frac{\mathbf{d}}{|\mathbf{d}|}\right), \qquad \mathbf{R}'_2 = \frac{1}{\sqrt{2}}\left(\frac{\mathbf{c}}{|\mathbf{c}|} - \frac{\mathbf{d}}{|\mathbf{d}|}\right), \qquad \mathbf{R}_3 = \mathbf{R}'_1 \times \mathbf{R}'_2 \tag{A.5}$$

i.e. the rotation matrix that defines the orientation between image and world planes is given by

$$\mathbf{R} = [\mathbf{R}'_1|\mathbf{R}'_2|\mathbf{R}'_3]. \tag{A.6}$$

Appendix B

Hardware Assisted Texture Mapping

This appendix section describes a hardware assisted texture mapping details. Since the heavy nature of software texture warping, solutions based on hardware provide near real time capabilities on AAM framework. Texture mapping is a classical problem on computer graphics. Modern graphics cards provide hardware based solutions to this problem and a solution based on the open graphics library, `OpenGL` [23], API is given.

The AAM framework was implemented in Linux SO using mostly the Intel's computer vision library, `OpenCV` [24]. Texture mapping was performed by `OpenGL`. Since the machine state nature of `OpenGL`, a dedicated thread was used to support the warping process. Thread management was achieved using POSIX Threads.

To understand the listed code, consider the following C++ class,

```
class AAM{
private:
pthread_t glThreadID;

public:
//Performs Hardware-Assisted PieceWise Affine Texture Mapping
void glTextureMapping(IplImage*Image,CvMat*x,CvMat*y,CvMat*xw,CvMat*yw,
```

```
IplImage*WarpedImage);

AAM(); //AAM Constructor
~AAM(); //AAM Destructor
};
```

a pthread identifier is used as a private member. Texture mapping procedure is supported by a public method. This method receives the input image, control points to sample, (x, y), and the destination control points, (x_w, y_w). Each one of the `CvMat` structures represents n dimensional vectors. The output image is returned to WarpedImage. Image B.1 shows an illustrative scheme of the method. The `OpenGL` display routine procedures, `glDisplay()` is also showed.

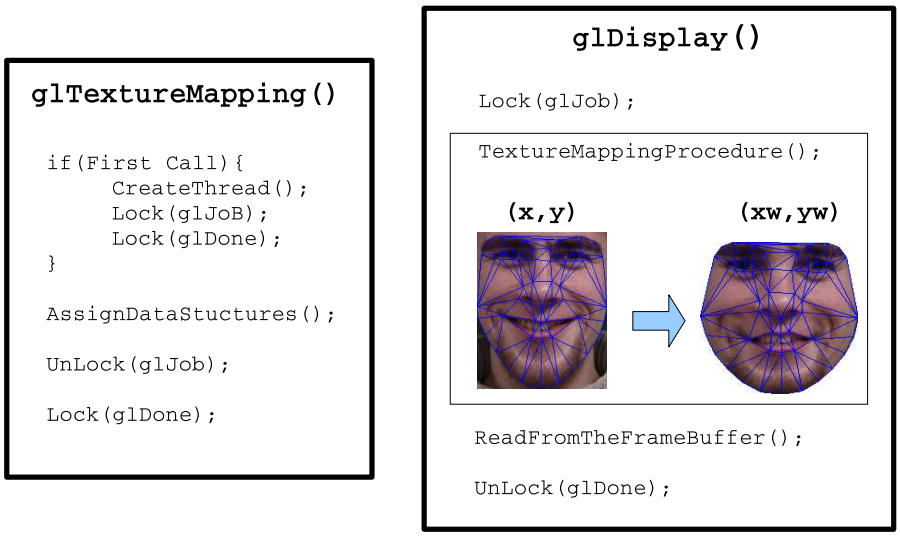


Figure B.1: Texture Mapping Procedure

The two threads works on a producer-consumer scheme. A routine call on `glTextureMapping()` sets `glDisplay()` to texture map the input image. Thread synchronization is achieved using two mutexs, `glJob` marking a pending job, i.e. a texture to be mapped, and `glDone` unblock when the job finishes. `OpenGL` is set to orthographic projection model and viewport is defined with the same size of the input image.

A first call on `glTextureMapping()` creates a thread, starting the `OpenGL` state machine, both mutexs are blocked and Delaunay triangulation result is

loaded. Source and destination control points are assigned and the synchronization mechanism is triggered.

`glDisplay()` starts loading the input image into a texture. The texture for each Delaunay triangle, is mapped from source, (x, y), to destination, (xw, yw), control points. The output image is then read from the framebuffer and returned.

The source code for the AAM texture mapping is shown in the following.

```
#include "AAM.h"
#include <GL/glut.h>
#include <stdio.h>
#include <signal.h>

//Initialize Mutexes
pthread_mutex_t glJob=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t glDone=PTHREAD_MUTEX_INITIALIZER;

//Texture Identifier
GLuint glTextureName;

//Auxiliar Texture Mapping Vars
IplImage*glImage=NULL,*glWarpedImage=NULL;
CvMat*glx=NULL,*gly=NULL,*glxw=NULL,*glyw=NULL;
CvMat*glDelaunayTriangles=NULL;

//OpenGL Display Routine
void glDisplay(void){
int k=0;

//Waits for Jobs
pthread_mutex_lock(&glJob);

//Assign Texture
glTexImage2D(GL_TEXTURE_2D,0,3,glImage->width,glImage->height,0,GL_RGB,
```

```
GL_UNSIGNED_BYTE, glImage->imageData);

glClear(GL_COLOR_BUFFER_BIT);

glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glBindTexture(GL_TEXTURE_2D, glTextureName);

glBegin(GL_TRIANGLES);
//Assign Texture for Each Triangle
for(k=0;k<glDelaunayTriangles->rows;k++){

glTexCoord2f(cvV32f(glx,cvM16i(glDelaunayTriangles,k,0))/glImage->width,
cvV32f(gly,cvM16i(glDelaunayTriangles,k,0))/glImage->height);
glVertex3f(cvV32f(glxw,cvM16i(glDelaunayTriangles,k,0)),
cvV32f(glyw,cvM16i(glDelaunayTriangles,k,0)), 0.0);

glTexCoord2f(cvV32f(glx,cvM16i(glDelaunayTriangles,k,1))/glImage->width,
cvV32f(gly,cvM16i(glDelaunayTriangles,k,1))/glImage->height);
glVertex3f(cvV32f(glxw,cvM16i(glDelaunayTriangles,k,1)),
cvV32f(glyw,cvM16i(glDelaunayTriangles,k,1)), 0.0);

glTexCoord2f(cvV32f(glx,cvM16i(glDelaunayTriangles,k,2))/glImage->width,
 cvV32f(gly,cvM16i(glDelaunayTriangles,k,2))/glImage->height);
glVertex3f(cvV32f(glxw,cvM16i(glDelaunayTriangles,k,2)),
cvV32f(glyw,cvM16i(glDelaunayTriangles,k,2)), 0.0);
}
glEnd();

glutSwapBuffers();

//Read From FrameBuffer
glReadPixels(0,0,glWarpedImage->width,glWarpedImage->height,GL_RGB,
GL_UNSIGNED_BYTE,glWarpedImage->imageData);
```

```
//Unblocks Mutex glDone. Setting Job Done
pthread_mutex_unlock(&glDone);
}

//OpenGL Idle Function
void glIdle(void){
glutPostRedisplay();
}

void glVisible(int vis){
if (vis == GLUT_VISIBLE)
glutIdleFunc(glIdle);
else
glutIdleFunc(NULL);
}

//Reshape Window Handler
void glReshapeWindow(GLsizei w,GLsizei h){
glViewport( 0, 0, w, h);

glMatrixMode (GL_PROJECTION);
glLoadIdentity();

//Using Orthographic Projection
glOrtho (0, w, 0, h, -5.0, 5.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

//Set Texture Mapping Parameters
void glInit(void){
```

```
//Clear The Background Color To Black
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

glShadeModel(GL_SMOOTH);

//Enable Texture Mapping
glEnable(GL_TEXTURE_2D);

glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

glGenTextures(1, &glTextureName);
glBindTexture(GL_TEXTURE_2D,glTextureName);

//scale linearly when image bigger than texture
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);

//scale linearly when image smalled than texture
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
}

//Signal function Handler
void glExit(int Signal){
//Release Mutexes
pthread_mutex_destroy(&glJob);
pthread_mutex_destroy(&glDone);

//Terminate glThread
pthread_exit(NULL);
}

//OpenGL Main Function
void* glTextureMappingMainLoop(void*Args){
int argc=1;
```

```
//Redirect SIGTERM to glExit() Funcion
signal(SIGTERM,glExit);

glutInit(&argc, NULL);

//Init OpenGL With Double Buffer in RGB Mode
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

//Init GL Window
glutInitWindowSize(glImage->width,glImage->height);

glutCreateWindow("Texture Mapping");

//Set Display Handler
glutDisplayFunc(glDisplay);

//Set Keyboard Handler
//glutKeyboardFunc(KeyboardHandler);

glutReshapeFunc(glReshapeWindow);
glutVisibilityFunc(glVisible);

//Set Texture Mapping Parameters
glInit();

//Hide glWindow
glutHideWindow();

//OpenGL Main Loop
glutMainLoop();

pthread_exit(NULL);
```

```
}

//Performs Hardware-Assisted PieceWise Affine Texture Mapping
void AAM::glTextureMapping(IplImage*Image,CvMat*x,CvMat*y,CvMat*xw,CvMat*yw,
IplImage*WarpedImage){

if(this->glThreadID==0){
//Init Mutex States
pthread_mutex_lock(&glJob);
pthread_mutex_lock(&glDone);

//Create Thread
if(pthread_create(&this->glThreadID,NULL,glTextureMappingMainLoop,NULL)){
printf("Can't Create OpenGL Texture Mapping Thread"); exit(0);};

//Get Mean Shape Delaunay Triangles
glDelaunayTriangles=this->MeanDelaunayTriangles;
}

//Assign Images
glImage=Image;
glWarpedImage=WarpedImage;

//Assign Control Points
glx=x;
gly=y;
glxw=xw;
glyw=yw;

//Unblocks Mutex -> Marks Having a Job
pthread_mutex_unlock(&glJob);

//Waits for glDisplay()
```

```
pthread_mutex_lock(&glDone);
}
```

The souce code for the class destructor is

```
//AAM Destructor
AAM::~AAM(){
if(this->glThreadID!=0){
pthread_kill(this->glThreadID, SIGTERM);
pthread_join(this->glThreadID,NULL);
}
}
```

Index