Introduction to Active Appearance Models: Understanding Theory and Algorithms

Ricardo D.C. Marín

October 2011

Abstract

Active Appearance Models learn and build a mathematical model for shape and appearance variance inside a database of images of an object in question. The method can be used for many computer vision tasks including face recognition and tracking in videos. In this document I present in a detailed format the theory behind such learning process, and provide guidelines for implementing the method from scratch by using a well known database of annotated face images, which is the *imm_ face_ database*[Nordstrom et al., 2004].

Contents

Chapter 1

Introduction

How do we know computationally if an image contains an specific object?. An instance of this problem is how can we be sure that an image contains a face. If we define video as a sequence of images (frames), going further we can add complexity to the problem and ask for a system that apart from identifying an object in a video, once found, the system is capable of describing how the object change its position, orientation and possibly color intensities along time. The first problem is known as *detection* or *recognition* while the second has been called *tracking*. Identification is performed by us humans as a cognition process involving our memory or what we have learned before, that is, we recognize an object because we have see it before, or then by inference or deduction (from a previous description) in a probabilistic fashion: we can't know for sure, or we may be wrong. To simulate the learning process computationally, usually we need to feed the system with information that we have previously well classified and described, that is, a database of training information from where we want to learn. The step from taking a collection of images to developing a mathematical model for a system to learn something from these training images samples is the crucial factor of every machine learning process for image interpretation.

Over the years, numerous mathematical approaches has been developed for providing solutions for detection and tracking, ones with bigger success than others for many factors including how well does the system respond to new instances not present in the training samples. As we will see through this document, of special interest for facial motion tracking is the method introduced in [Cootes et al., 2001] called ***Active Appearance Models (AAM)*** which expanded a previous work of the same authors [Cootes et al., 1995] called ***Active Shape Models (ASM)***. In AAM, we would like to create a mathematical model for describing shape and appearance of a class of objects within its different instances present in different images. To fully understand this sentence we need to know get some intuition on what we mean by a *mathematical model*, and what we call an

instance of a class of objects in an image.

A ***mathematical model*** is an abstraction of a system, or intuitively, of a real life situation or phenomena described in mathematical terms. As an example, the Newton's laws are a mathematical model for describing objects motion due to external forces. Since a model is a mathematical description, it uses mathematical concepts like functions, equations, parameters, sets, etc. To construct a model, first we abstractly separate the system in components and their inherent relationships. Intuitively, each component is translated to a mathematical concept as well as their relationships, in order to reach some field within mathematics from which we can reason and make conclusions about the system that otherwise (without this mathematical translation) couldn't be noticed or couldn't be asserted. The key point is that mathematics is an exact field and, having everything well defined, we gain control over the system. Since a single system could be described mathematically in different ways, a system may have more than one model.

Being itself a model, AAM's are a mathematical description of shape and appearance variance that exist for a class of objects in images. This document address the construction of such model. The main components of this system are shape and appearance, so we need to define and translate them in mathematical terms. Variance between different shapes and appearances need also to be treated as a component/relationship that need to be translated into mathematics. Since Principal Component Analysis (PCA) is a fundamental tool for describing variance in a collection of samples, it will help us with in this task.

1.1 Structure of the Report

This report is divided in 3 main chapters. Chapter 2 introduce the mathematical representation of shape. We study a space of shapes, and construct a PCA linear model for variance in a discrete set of aligned samples of this space. Similarly to shapes, appearance mathematical representation is discussed in chapter 3, as well as a PCA model for a our training images appearances. Since these two chapters construct independent parameters for shape and appearance, chapter 4 performs PCA method in the set of the already constructed model parameters of shape and appearance, to account for correlations between these two properties. The combined model description as well as computation of model instances is discussed there.

Algorithms that give insights on an implementation of combined AAM's are distributed along sections in all chapters. The code can be treated as pseudo-code with similarities to $C/C++$, but must be not totally treated or translated to this language. The database of face images *imm_face_database*[Nordstrom et al., 2004] is used through all the examples and implementation in this document.

Chapter 2

PCA Shape Model

2.1 Shape Space

Since we will be talking about shape, we need to address how to represent shape in images mathematically. The simplest approach is to use landmarks points positioned on the apparent edges or contours of the object. The landmarks should be positioned such that the object form is well captured and the quantity of landmarks should be taken into account for the sake of the description. Figure 2.1 shows different instances of a face (our object of interest) with annotated landmarks. To represent the shape of the face, landmarks are positioned on places that distinguish the image as a face: eyes, eyebrows, nose, lips and face contour. Notice that although we are using the face of the same person, his shape landmarks varies from image to image according to the expression (non rigid motion) or head rigid motion. This shape variance in our training database is fundamental for a robust model of shape, since we are going to make use of PCA to approximate all variance present on different faces and on different poses (rigid and non-rigid) for tracking motion.

To give some intuition over the statistical model that we are going to construct, let's think for a moment in terms of linear algebra. Given an image in the database, its corresponding ***shape*** s is defined as the set of n points positions $s = \{(x_i, y_i)\}_{i=1}^{n}$ in two dimensions (pixel coordinates)[1]. Instead of a set, a more useful representation is the concatenation of each point in a whole vector

$$s = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n) \in R^{2n.}$$

We assume that the landmarks are always sorted in the same way, thus if for example x_5 of shape s_1 correspond to the nose tip landmark, then

[1]In practice, coordinates x' and y' in $[0, 1]$ are used instead of pixel coordinates x and y in $[image\_width, image\_height]$. The simple conversion $x' = x/image\_width$, $y' = y/image\_height$ is used for this.

Figure 2.1: Landmark points for shape representation. Up non-rigid shape motion and down rigid shape motion.

x_5 of any other shape s_2 correspond also to this same nose tip landmark. Assuming further that all faces in our training database have the same number of landmarks that describe its shape, with this new representation we have obtained that all shapes are points of the space R^{2n}. However, not all points of R^{2n} correspond to shapes of our object, in a semantic way. What this means is that the above vector definition of shape needs to be refined since it doesn't take into account the relationship that exist inherently between landmark points. As an example technically the null vector 0 of R^{2n} with the above definition is a shape, but semantically it doesn't give us a shape with eyes, nose, etc. The shape definition thus needs to account for the type of object it describes. Nevertheless, this definition although incomplete is useful since it allow us to operate with shapes using linear algebra methods.

Thus shapes conforms a subset of R^{2n} space that we will denote by S. For account for intuition, let's assume that this set correspond to a sphere centered at the origin of coordinates of R^{2n} and with radius ρ, and therefore all our shapes satisfy the equation of the sphere $\sum_{i=1}^{n}(x_i^2 + y_i^2) = \rho^2$. Now for simplicity let's further assume our space R^{2n} is R^2. This is great, now all the points in the sphere not only satisfy $x^2 + y^2 = \rho^2$ but also they can be represented as $(\rho\cos\theta, \rho\sin\theta)$ for some $\theta \in R$. The interesting point in this simplification, is that the set of shapes (which we are assuming are points in the circumference of radius ρ) can be represented by selecting θ values and evaluating the vectorial function

$$f(\theta) = (\rho\cos\theta, \rho\sin\theta).$$

No matter which θ value in R we choose, by evaluating this function we will reach a shape, i.e a point in the circumference of radius ρ. Thus we can think of f as a "generator" of shapes. Analogously, the spherical coordinates function

$$f(\theta, \phi) = (\rho\sin\theta\cos\phi, \rho\sin\theta\sin\phi, \rho\cos\theta)$$

is a "generator" of points inside the sphere centered at the origin of R^3 and with radius ρ for every (θ, ϕ).

Let's check some properties of the generator functions we have seen. First, notice that being the functions f periodic we can restrict their domain to $\theta \in [0, 2\pi]$ for R^2 and $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ in R^3, and still be able to cover *all* points in the circumference or sphere. Second, observe that with this generators, our shapes can be represented with less parameters: in R^2 we just need one value θ instead of two (x, y), and in R^3 we need two values (θ, ϕ) instead of three (x, y, z). Third, notice how going from R^2 to R^3 makes more complex the generator function. Mathematically, with properly restrictions on the domain of the functions f, they form what are called parameterizations of the sphere. A parameterization of the sphere in any dimension such as R^{2n} can be achieved using a *stereographic projection*. Finally, a fundamental fact for our shape model, is that parameterizations permit us walk continuously on the set they describe; we will never reach a point outside of this set, we will never reach a no-shape point, and we are able reach shapes that are near other shapes continuously.

The assumption of S being a sphere of R^{2n} is far from realistic. In fact we can't know for sure how S is. Still, it is natural to ask whether or not a generator of shapes for S exist (not for our simple sphere), and if it exist, how can we find or construct one?. One of the main problems in finding answers to these questions relies precisely on the ambiguous definition of shape, since we can't say for sure (except in some simple cases) if a point in R^{2n} is a shape or not, since we have not well characterized the points of R^{2n} that describes the contours of our object.

Now let's try to get some intuition on S. A good observation is that if s is a shape in a particular image, then for any rotation matrix R of R^{2n}, the rotated shape $R{\cdot}s$ stills describes our object but in a different image, namely, the rotated initial image. Since the same reasoning applies for translation and scaling, we can say that S is invariant under affine transformations in R^{2n}. This implies that we can restrict our investigation of S to a subset $S' \subset S$ obtained by removing scaling, rotation and translation. The fact is that, from this new subset S', we can reconstruct the whole original shape set S, precisely by scaling, rotating and translating. Another observation is that S as well as S' are composed of infinitely many shapes, since we all humans have different face configurations. Given that in practice we can only have a finitely set of training image samples, we will get a finite subset DS of shapes. In other words, the set S is the population (infinite) but in practice we will have a (finite or discrete) sample set DS of this population. It is important to note that the samples we get from the training images have present rotation, translation and scaling among them, that is, $DS \subset S$. Since we want to restrict our investigation to S' we will need a method to remove scaling, rotation and translation of members of DS to obtain its equivalent discrete set $DS' \subset S'$. The common method for this is called *Procrustes Analysis*[2].

2.2 Shape Alignment with Procrustes Analysis

An affine transformation in R^2 can be totally described by a 2×2 scaling (f) rotation (θ) matrix

$$M(f,\theta) = \begin{pmatrix} f\cos\theta & -f\sin\theta \\ f\sin\theta & f\cos\theta \end{pmatrix}$$

and a translation vector

$$t = (t_x, t_y) \in R^2.$$

Since any shape $s \in S$ in our shape space is a collection of n points in R^2, we can use such an affine transformation in a point by point fashion to obtain a transformed shape

$$s_t = (x_{t_1}, y_{t_1}, x_{t_2}, y_{t_2}, \ldots, x_{t_n}, y_{t_n}).$$

Point by point transformation means that fixing the scaling f, the rotation θ and the translation vector t, we have for all $i = 1, 2, ..., n$:

$$M(f,\theta)\begin{pmatrix} x_i \\ y_i \end{pmatrix} + t = \begin{pmatrix} x_{t_i} \\ y_{t_i} \end{pmatrix}$$

[2]Procrustes is character from Greek mythology that stretched people to an iron bed by cutting their extremities.

Thus a transformation $T = (M(f, \theta), t)$ takes shapes s to transformed shapes s_t. We write this as $s_t = T \cdot s$, and its coordinates can be computed with the previous equation.

Taking a shape s_r as a constant reference, alignment of any other shape s with respect to s_r means to find a transformation T in order to minimize the difference between the transformed shape and the shape reference

$$E(T) = E(f, \theta, t) = |s_r - (T \cdot s)| = |s_r - s_t| \,. \tag{2.1}$$

This equation must be understood as the distance in R^{2n} between s_r and s_t.

Procrustes analysis refers indirectly to solving this problem, since when we apply a transformation that minimize (2.1) to a shape, we are removing scaling, rotation and translation with respect to the reference shape, like if we were fitting shape s to s_r. To find such transformation we can use a simplification[3] of an approach included in the original paper on ASM. First define $a_x = f \cos\theta$ and $a_y = f \sin\theta$. The method consist in solving the 4×4 linear system

$$\begin{pmatrix} X_2 & Y_2 & n & 0 \\ Y_2 & X_2 & 0 & n \\ Z & 0 & X_2 & Y_2 \\ 0 & Z & -Y_2 & X_2 \end{pmatrix} \begin{pmatrix} a_x \\ a_y \\ t_x \\ t_y \end{pmatrix} = \begin{pmatrix} X_1 \\ Y_1 \\ C_1 \\ C_2 \end{pmatrix} \tag{2.2}$$

[3]In general, a vector of weights $W \in R^{2n}$ is used to influence the motion of points in a shape. A simplification is done by treating all the weights equal to 1.

where all the constants can be computed as:

$$X_1 = \sum_{i=1}^{n} x_{r_i} \tag{2.3a}$$

$$Y_1 = \sum_{i=1}^{n} y_{r_i} \tag{2.3b}$$

$$X_2 = \sum_{i=1}^{n} x_i \tag{2.3c}$$

$$Y_2 = \sum_{i=1}^{n} y_i \tag{2.3d}$$

$$W = n \tag{2.3e}$$

$$Z = \sum_{i=1}^{n} x_i^2 + y_i^2 \tag{2.3f}$$

$$C_1 = \sum_{i=1}^{n} x_i \cdot x_{r_i} + y_i \cdot y_{r_i} \tag{2.3g}$$

$$C_2 = \sum_{i=1}^{n} y_{r_i} \cdot x_i - x_{r_i} \cdot y_i \tag{2.3h}$$

After solving (2.2) we will obtain the matrix $M(f, \theta)$ from a_x and a_y, and the translation t directly. To understand why this method works, since minimizing E is the same as minimizing E^2, we write explicitly an equation for $E^2(a_x, a_y, t_x, t_y)$ from (2.1):

$$E^2(a_x, a_y, t_x, t_y) = \sum_{i=1}^{n} \{ \begin{pmatrix} x_{r_i} \\ y_{r_i} \end{pmatrix} - (M(f, \theta) \begin{pmatrix} x_i \\ y_i \end{pmatrix} + t) \}^2 \tag{2.4}$$

and from calculus we know that to find the values (a_x, a_y, t_x, t_y) that minimize (2.4), we must find the derivatives of E^2 with respect to each of the parameters a_x, a_y, t_x, t_y and solve $\nabla E^2 = 0$. Making these calculations will lead us to (2.2). In summary, the next algorithm performs alignment of a shape s with respect to a reference s_r:

```
void procrustesAlignment(shape s, const shape sr){
  /* Compute the matrix elements*/
  double X1=0.0, X2=0.0, Y1 =0.0, Y2=0.0, Z=0.0, C1=0.0, C2=0.0;
  double W = (double)n;
  for(int i=0;i<n;i++){
    X1 += sr[2*i];
    Y1 += sr[2*i+1];
    X2 +=  s[2*i];
    Y2 +=  s[2*i+1];
    Z  +=  s[2*i]*s[2*i] + s[2*i+1]*s[2*i+1];
```

```
    C1 +=  s[2*i]*sr[2*i] + s[2*i+1]*sr[2*i+1];
    C2 +=  sr[2*i+1]*s[2*i] - sr[2*i]*s[2*i+1];
  }

  /* Arrange the matrix and vector of the linear system */

  double PAMatrix[4][4];
  PAMatrix[0][0] = X2; PAMatrix[0][1] = -Y2;
  PAMatrix[0][2] = W;  PAMatrix[0][3] = 0.0;
  PAMatrix[1][0] = Y2; PAMatrix[1][1] = X2;
  PAMatrix[1][2] = 0;  PAMatrix[1][3] = W;
  PAMatrix[2][0] = Z;  PAMatrix[2][1] = 0.0;
  PAMatrix[2][2] = X2; PAMatrix[2][3] = Y2;
  PAMatrix[3][0] = 0.0;PAMatrix[3][1] = Z;
  PAMatrix[3][2] =-Y2; PAMatrix[3][3] = X2;

  double b[4];
  b[0]=X1;
  b[1]=Y1;
  b[2]=C1;
  b[3]=C2;

  /* Solve the linear system: PAMATRIX*T = b */
  double T[4]; // T=(ax,ay,tx,ty)
  linear_solver(PAMatrix,b,T);

  /* Transform the shape s by T */
  double xr, yr;
  for(int i=0; i<n; i++){
     xr = T[0] * s[2*i] -  T[1] * s[2*i+1];
     yr = T[1] * s[2*i] +  T[0] * s[2*i+1];
     s[2*i]     = xr + T[2]; // xr + tx
     s[2*i + 1] = yr + T[3]; // yr + ty
  }
}
```

Until this point we have aligned a single shape s to a reference s_r. In practice we want to take all the samples shapes $s \in DS$ and align them to the same reference s_r. The reference could be any shape. A common approach consist in taking s_r as the mean shape of the set of samples DS. This lead us to the following algorithm to perform shape alignment in our samples database:

```
void shapesAligment(shape* shSamples){
 shape cmsh;  // current mean shape
 shape nmesh; // new mean shape
 double d;
 computeMeanShape(shSamples,&cmsh);
 bool convergence = false;
 do{
   /*Align all shapes with reference shape the current mean*/
   for(int k=0;k<nSamples;k++){
    procrustesAlignment(shSamples[k], cmsh);
   }

   /* Compute the new mean shape of the aligned shapes*/
   computeMeanShape(shSamples,&nmesh);
```

```
    /* Check for convergence, */
    d = EuclideanDistance(cmsh,nmesh);
    if(d < 0.1 ) convergence = true;
    else oldMeanShape = newMeanShape;
  }while(!convergence);
}
```

Figure 2.2 shows the result of applying the shapes alignment algorithm to the face database *imm_face_db* we are using in this document.

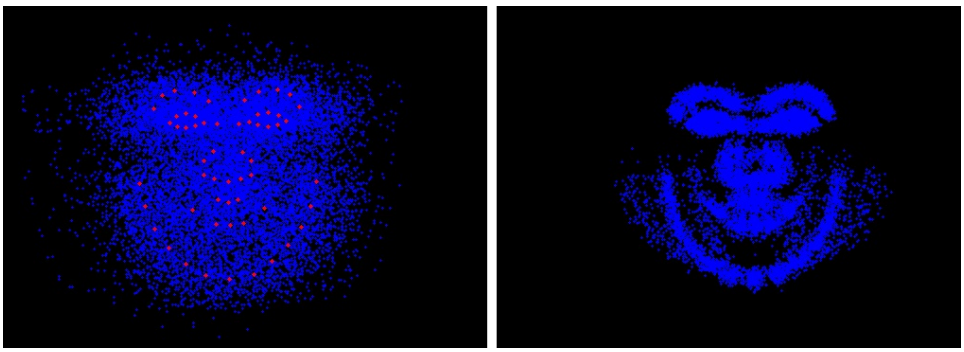

Figure 2.2: Raw and Aligned Samples. Left: Raw samples with the mean in red, set DS; Right: Aligned samples with respect to the mean, our set DS'

2.3 Shape Model

In the previous section we saw how we can construct the set of aligned shapes DS'. We now turn our interest in modeling the inherent variation in this set. In ASM, the approach used for this is PCA[4]. Intuitively we can think of PCA as a change of coordinates R^{2n}, where the the set of nv vectors $V = \{V_i\}_{i=1}^{nv}$, of the new reference frame are pointing in the directions of variance in the shape set DS'. The PCA method begins by computing the means of each of the coordinates x_i and y_i for all i, producing a mean vector of n components, each component being the mean of coordinate x_i or y_i. Setting a matrix Φ_s of $2n \times ns$ where each column is exactly one shape of DS' minus the computed mean vector, with ns the number of samples shapes in DS', PCA find the set V to be the eigenvectors of the covariance matrix of Φ_s, computed as

$$Cov(\Phi_s) = \frac{1}{ns}\Phi_s\Phi_s^T$$

We denote by $\bar{s}$ the mean shape of DS' as in the previous section and we proceed to define a matrix E_s which has as columns all the sorted eigenvectors V. Note that all V_i are in R^{2n}, so the matrix E_s is of order $2n \times nv$.

[4]A good reference for getting started with PCA is [Shlens, 2009]

The fundamental thing about PCA that we need to keep in mind for ASM is that any shape $s_k \in DS'$ can be approximated as:

$$s_k = \bar{s} + E_s \cdot b_k \tag{2.5}$$

for some vector $b_k \in R^{nv}$. We must see that in this equation the variable is b_k and not s_k, given that we know values for s_k, $\bar{s}$ and E_s, and we don't know who b_k would be. Actually that is the next step. For a given s_k in DS', if we take

$$b_k = E_s^T \cdot (s_k - \bar{s}) \tag{2.6}$$

we will see that replacing (2.6) in the right hand side of (2.5) we will reach s_k. This is not satisfactorily enough, since for knowing b_k we used the value of s_k, and vice-versa. What it is important is what happens when we consider the right hand side of (2.5) as a function of b, with values of b not necessarily obtained from (2.6):

$$G_s(b) = \bar{s} + E_s \cdot b \tag{2.7}$$

Let's get back for a moment to the discussion on shape space generators in section 2.1. We note that the function G_s is a pretty good candidate for generating shapes. For example for all $b = b_k$ obtained from (2.6), $G_s(b)$ is a shape, namely, $s_k \in DS'$. Also $G_s(0) = \bar{s}$ can be recognized as a shape (see Figure 2.2). And what if b is not taken from (2.6) nor is $b = 0$? Let's get some intuition taking a scaled by $f \in R$ canonical vector $b = f \cdot e_1 \in R^{nv}$. Evaluating in (2.7) we get

$$G_s(f \cdot e_1) = \bar{s} + f \cdot V_1$$

This equation suggest us that by varying the scalar factor f we are obtaining a perturbation of the $\bar{s}$ in the direction of V_1. The question is if $G_s(f \cdot e_1)$ which is a vector of R^{2n} is a shape or not. The reader should be familiar with this question from our discussion in 2.1. Since the shape space S is not well characterized, we cannot demonstrate rigorously that this vector is a shape. We however can plot $G_s(f \cdot e_1)$ as a set of n points in R^2. Figure 2.3 shows this for values of $f = -2, 2$. We see that unhappily we didn't obtain shapes for this values. So G_s generate shapes for some values of b, but not for all.

So, is it possible to determine entirely a parameter set of b's for which $G_s(b)$ is a shape?. Fortunately the answer is yes. Almost all the variance present in DS' on each principal direction V_i can be modeled by a scaled version of V_i, with scaling factors in $[-2.5\sqrt{\lambda_i}, 2.5\sqrt{\lambda_i}]$. For our previous discussion with $G_s(f \cdot e_1)$ this means that by varying f in $[-2.5\sqrt{\lambda_1}, 2.5\sqrt{\lambda_1}]$ we will probably get similar shapes (variant shapes) to shapes present in our set DS'. Figure 2.4 shows plots of $G_s(f \cdot e_1)$ for values of $f = -2\sqrt{\lambda_1}, 2\sqrt{\lambda_1}$.

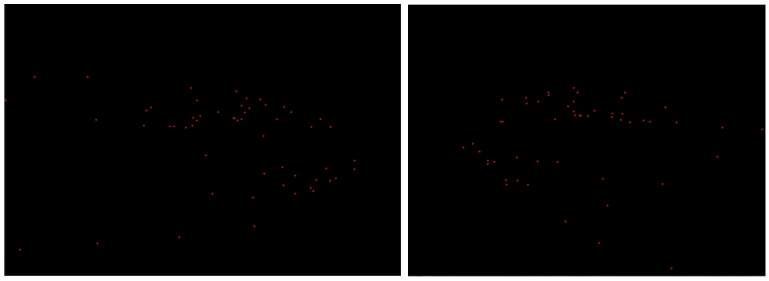

Figure 2.3: Plot of the vectors $G_s(f \cdot e_1)$: Left $f = -2$; Right $f = 2$. These are not face shapes.

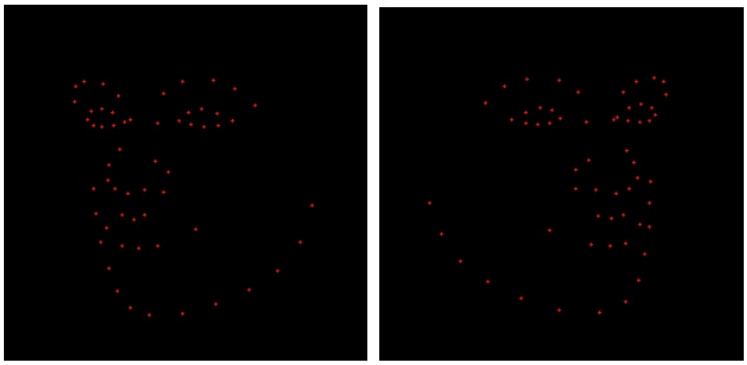

Figure 2.4: Plot of the vectors $G_s(f \cdot e_1)$: Left $f = -2\sqrt{\lambda_1}$; Right $f = 2\sqrt{\lambda_1}$. These are face shapes.

Therefore a possible parameter set for which $G_s(b)$ is a generator of shapes would be

$$P_s = \prod_{i=1}^{nv} [-2.5\sqrt{\lambda_i}, 2.5\sqrt{\lambda_i}] = \prod_{i=1}^{n} [-2.5\sigma_i, 2.5\sigma_i]. \tag{2.8}$$

We call the function $G_s(f \cdot e_i)$ for f in $[-2.5\sqrt{\lambda_i}, 2.5\sqrt{\lambda_i}]$ the *i-esim Mode of Variation* of our PCA model, since we just move from the mean shape in the direction of the $i - esim$ principal component (evaluating in (2.7)):

$$G_s(f \cdot e_i) = \bar{s} + f \cdot V_i$$

In our example we were using $b = f \cdot e_1$, so 2.4 correspond to the 1st mode of variation evaluated at two different values of f.

In summary a PCA shape model has been created from the function G_s defined in (2.7) restricted to the parameter set P_s defined in (2.8). Shapes generated by this model however are posed in the space S' where we have removed scaling, rotation and translation. To go back to the general space

S we simply use an affine transformation T. Our model thus correspond to two main components: G_s restricted to P_s and an affine transformation T. Note also that as with the generators functions discussed in section 2.1, no matter which value of b in P_s we take, we will reach a variant/similar shape in S' to the ones present in DS'. Here we emphasize however that this model is strictly dependent on DS', since our new shapes generated by G_s will be variants of the ones present in this set. For example, if we want a shape of a happy expression, we will need to feed the set DS' with samples of shapes of happy expressions. This is an implication of the fact that $G_s : P_s \rightarrow S'$ is not a surjective function, so not every $s \in S'$ has a parameter b for which $G_s(b) = s$. Finally suppose that we have an input image that is not present in our training database, but that contains an instance of the class of our shape model. The problem of finding the parameter b and the affine transformation T that best describe (approximate) this new instance shape is called *fitting*. A detailed approach for solving a fitting problem can be found on chapter 10, page 469 of [Sonka et al., 2007].

Chapter 3

PCA Appearance Model

3.1 Appearance Space

As with shape, in this section we address how to represent appearance in images mathematically. A simple approach for this is a list of pixel intensities values of a region of interest in the image sample. This list can be arranged in a high m dimensional vector of (r, g, b) pixel intensities

$$c = (r_1, g_1, b_1, r_2, g_2, b_2, \ldots, r_m, g_m, b_m) \in R^m$$

which we call *color vector*. The region of interest in our case will be the set of pixels positions which are inside of the shape that describe the object in the image. To be more precise, a pixel position $p \in R^2$ is inside the shape $s \in S$, if for every triangulation of the set of n points that define s, which are in R^2, we can find a triangle that contains p. Therefore our region of interest is composed of all points p which satisfy this property. In this way to find the color vector c associated to a shape vector s, we triangulate s and store in c the pixel intensities corresponding to pixels positions inside all the triangles of the triangulation. Figure 3.1 shows an example result of extracting the color vector (here visualized as pixels colors) of a raw image of the training samples. The color vector is the collection of all the pixels of the image at the right.

These color vectors c are however not well suited for what we want to define as *appearance*. Observe that because of the dependence of the region of interest with shape, the number of pixels positions m inside a region of interest can vary from one image sample to another. Thus we cannot use matrix algebra as we did with shapes for example, since we would have vectors in multiple different spaces. And this is not the only drawback. Imagine for example that we had two images with the same neutral expression of the same person but one in frontal view and the other in 45 degrees view. We would like our definition of appearance to be as *equal* as possible in these two images, since what have changed from one to the

Figure 3.1: Region of Interest: On the original image (left), annotated shape landmarks are triangulated (middle). The collection of pixels positions which are inside the triangulation compose our region of interest. Its corresponding list of pixel intensities compose the color vector which is here visualized as a set of pixels (right).

other is the view, and not the expression or the person face characteristics (race, age, eyes, eyebrows, etc.). Observe however that if we define the color vector to be appearance, appearance will be different for these two images. To see this, it is obvious that the two shapes will differ, since shape depends on the image you are seeing (remember that you manually annotate landmarks in the samples accordingly to the form of the class of objects in question). Now since the region of interest is dependent on the shape, the region of interest will change for these two images, and more importantly, the color vectors for these two images will differ because of this shape difference. Note also that repeating a similar experiment, now taking two frontal images of a neutral expression of the same person but under different illumination of the scene, again the vector color will differ from illumination variance.

Therefore we need to find a definition for appearance that takes into account these issues. In the literature however, the appearance definition doesn't treat the illumination difference problem inside this definition. The usual theory goes as defining what is appearance, and based on the constructed set of appearance samples, perform photometric normalization to remove illumination variance. Thus we will forget for now this issue and proceed to define appearance.

Basically the main problem that the color vector has for being a good appearance definition is the dependence on shapes, and the dimension m over-definition. If we want to erase the influence of shape variance over the color distribution vector, we would need to take a *unique* shape as a reference (therefore removing shape variance) on which all colors can be treated. The usual choice for a reference shape is the mean shape $\bar{s}$ (see section 2.3), shown in figure 3.2. Given an image in the database, its corresponding

appearance a is defined as the list of (r, g, b) pixel intensities inside $\bar{s}$

$$a = (r_1, g_1, b_1, r_2, g_2, b_2, \ldots, r_m, g_m, b_m) \in R^m$$

that results from warping the region of interest of the image shape s to the mean shape region. Observe that now m is constant, since we are taking colors always on the same reference shape $\bar{s}$. Also note that the only difference between a and the previously defined color vector c is that now color triplets are inside of $bars$ instead of s. Figure 3.3 shows an example result of obtaining the appearance of a single image.

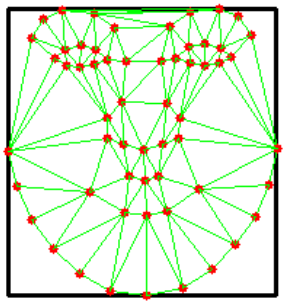

Figure 3.2: The Mean Shape $\bar{s}$: Red: Landmarks of the mean shape; Black: Bounding box of the landmarks points; Green: Delaunay triangulation of the landmarks set.

Figure 3.3: Appearance Example: The original image (left) is warped to the mean shape to obtain its appearance (Right).

Before giving implementation details on computing appearance vectors, we first make some further notations on what we need to use. We denote by TGL the triangulation associated to $\bar{s}$. Usually TGL is computed as a

Delaunay triangulation[1], storing a list of triangles by using the indexes of the tree points in $\bar{s}$ that conforms a triangle. Notice that TGL induce also a triangulation for any other shape s, since by definition all the shapes have their coordinates sorted in the same way, and thus their indexes are equal. Thus in practice, we need to compute TGL just one time for $\bar{s}$ and it will give us triangulations for all other shapes in our training database. Triangles of TGL in $\bar{s}$ will be denoted by $\bar{t} = (\bar{a}, \bar{b}, \bar{c})$, and its corresponding triangle in any other shape s will be denoted by $t = (a, b, c)$, with each coordinate being a point of the triangle. Later on we will be also using *barycentric coordinates* of a point inside a triangle. In general, given a pixel position $p = (p_x, p_y)$ its barycentric coordinates with respect to the triangle $(p1, p2, p3)$ are defined as the triplet $(\alpha, \beta, \gamma) \in R^3$ which satisfy

$$p = \alpha \cdot p_1 + \beta \cdot p_2 + \gamma \cdot p_3, \;\; \alpha + \beta + \gamma = 1.$$

To find such coordinates we solve the linear system

$$\begin{aligned} p_x &= \alpha \cdot p_{1_x} + \beta \cdot p_{2_x} + \gamma \cdot p_{3_x} \\ p_y &= \alpha \cdot p_{1_y} + \beta \cdot p_{2_y} + \gamma \cdot p_{3_y} \\ 1 &= \alpha + \beta + \gamma \end{aligned}$$

which can be explicitly solved as

$$\beta = \frac{p_y p_{3_x} - p_{1_x} p_y - p_{3_x} p_{1_y} - p_{3_y} p_x + p_{1_x} p_{3_y} + p_x p_{1_y}}{-p_{2_x} p_{3_y} + p_{2_x} p_{1_y} + p_{1_x} p_{3_y} + p_{3_x} p_{2_y} - p_{3_x} p_{1_y} - p_{1_x} p_{2_y}} \tag{3.1a}$$

$$\gamma = \frac{p_x p_{2_y} - p_x p_{1_y} - p_{1_x} p_{2_y} - p_{2_x} p_y + p_{2_x} p_{1_y} + p_{1_x} p_y}{-p_{2_x} p_{3_y} + p_{2_x} p_{1_y} + p_{1_x} p_{3_y} + p_{3_x} p_{2_y} - p_{3_x} p_{1_y} - p_{1_x} p_{2_y}} \tag{3.1b}$$

$$\alpha = 1 - \beta - \gamma \tag{3.1c}$$

An useful property of barycentric coordinates is that they allow us to assert if the point is inside of the triangle simply by checking whether we have $0 \leq \alpha, \beta, \gamma \leq 1$. Algorithm 3.1 returns true or false in the case the point is inside or outside some triangle of a triangulation TGL. When true the algorithm can be also used to store the corresponding triangle index in TGL where the point is and its respective barycentric coordinates, that is, weights.

```
bool pointInsideTriangle(point p, int* tId, double weights[]){
 point p1, p2, p3;
 float denom, alpha, beta, gamma;
 for(int i=0;i<TGL.nTri;i++){ // for every triangle
    //The i-esim triangle points
    p1 = TGL.triangles[i].p1;
```

[1]For computing triangulations, [Berg et al., 2008] is a good reference.

```
    p2 = TGL.triangles[i].p2;
    p3 = TGL.triangles[i].p3;

   //Compute barycentric coordinates of p with respect to triangle i
    denom=-p2.x*p3.y + p2.x*p1.y + p1.x*p3.y + p3.x*p2.y - p3.x*p1.y↩
         - p1.x*p2.y;

    beta = p.y*p3.x - p1.x*p.y - p3.x*p1.y - p3.y*p.x + p1.x*p3.y + ↩
        p.x*p1.y;
    beta /=denom;
    gamma= p.x*p2.y - p.x*p1.y - p1.x*p2.y - p2.x*p.y + p2.x*p1.y + ↩
        p1.x*p.y;
    gamma/=denom;
    alpha=1-beta-gamma;

   //check if point belogns to this triangle #i using barycentric ↩
       coordinates
    if(alpha>=0 && alpha<=1 && beta>=0 && beta<=1 && gamma>=0 && ↩
        gamma<=1){
      //store the triangle id and its barycentric coordinates
      (*tId) = i;
      weights[0]=alpha; weights[1]=beta; weights[2]=gamma;
      return true;
    }
  }
  return false;
}
```

Figure 3.4 illustrate the process of obtaining the appearance vector for a single image. In order to compute which are the pixels that conforms an appearance vector a_0 of an image that have shape s_0, we use triangulations of s_0 and $\bar{s}$ and perform warping in a piece-wise manner. For this, a backward warping to fill in the mean shape $\bar{s}$ with colors from the region of interest of s_0 is performed by iterating over the pixels of the bounding box of the mean shape, $\bar{p}$, and asking whether or not each of these pixels belongs to a triangle of TGL. If a pixel doesn't belong to any triangle, we continue with the next pixel. If it belongs to triangle $(\bar{a}, \bar{b}, \bar{c})$ of $\bar{s}$, we compute its barycentric coordinates

$$\bar{p} = \alpha \cdot \bar{a} + \beta \cdot \bar{b} + \gamma \cdot \bar{c}$$

and use them to find its relative position in the corresponding triangle (a, b, c) in s_0. In a piece-wise affine warping this correspond to the position p with

$$p = \alpha \cdot a + \beta \cdot b + \gamma \cdot c.$$

Since this new pixel position doesn't have necessarily integer coordinates, we cannot know directly which is the color on this position on the region of interest in s_0, so we use a bilinear interpolation method to compute its pixel color from the surrounding pixel colors. Given these four color values v_1, v_2, v_3, v_4, each with (r, g, b) components, the bilinear interpolated color value v that is positioned on the previously found pixel $p = (p_x, p_y)$ is

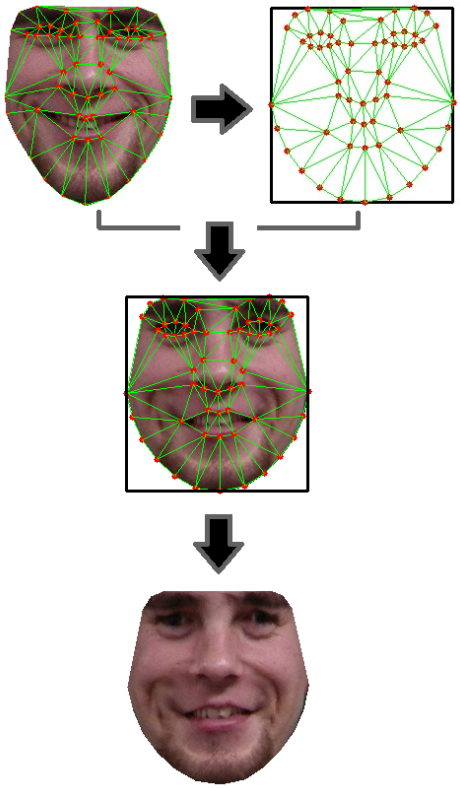

Figure 3.4: Obtaining Appearance Vectors: Using the triangulation of the sample and the mean shape triangulation, the region of interest of the sample is warped to the mean shape region to obtain the appearance vector.

computed as

$$v_r = v_{1_r}(1-p_x)(1-p_y) + v_{2_r}p_x(1-p_y) + v_{3_r}(1-p_x)p_y + v_{4_r}p_xp_y \quad (3.2a)$$
$$v_g = v_{1_g}(1-p_x)(1-p_y) + v_{2_g}p_x(1-p_y) + v_{3_g}(1-p_x)p_y + v_{4_g}p_xp_y \quad (3.2b)$$
$$v_b = v_{1_n}(1-p_x)(1-p_y) + v_{2_b}p_x(1-p_y) + v_{3_b}(1-p_x)p_y + v_{4_b}p_xp_y \quad (3.2c)$$

Note that for these equations to work, it is necessary to have colors normalized to $[0, 1]$, as well as pixel coordinates (p_x, p_y) in $[0, 1]$. The appearance vector a_0 is finally composed of all these interpolated values v.

Because of our definition of shape and its influence on the region of interest in which we perform the warping process, appearance takes into account the intensities of the eyes, eyebrows, nose, lips and face contour, etc. describing the colors configuration that would have the instance before any movement with respect to the mean shape have occurred. Note that all our appearance vectors are defined inside the mean shape. Similar to shapes, appearance vectors form a set $A \subset R^m$, and in practice for a

database of images we will have a discrete subset $DA \subset A$. These sets are in general very hard to characterize and again we don't have a method to decide which vectors of R^m are appearance vectors. As stated previously, once we have constructed our collection of appearance vectors DA, we would like to remove their dependence on illumination to obtain a modified collection of appearances DA'. These new normalized vectors will serve as our base for the construction of a PCA appearance model in a similar fashion as we did for shapes in section 2.3, but modeling now variance in appearance intensities rather than in shapes landmarks.

3.2 Appearance Model

In this section we turn our interest in modeling the spread of the data within DA', that is, appearance variance. Similarly to shapes, we use PCA to analyze our data set. We denote by $\bar{a}$ the mean appearance vector computed as the mean of all vectors in the set DA',

$$\bar{a} = \frac{1}{|DA'|} \sum_{a \in DA'} a.$$

We then construct the matrix Φ_a as having columns composed of the difference between appearance vectors $a \in DA'$ and the mean vector $\bar{a}$. Thus Φ_a has order $m \times ns$, with ns the number of samples in our set DA'. We find the principal directions of variance in DA' as the eigenvectors of the covariance matrix of Φ_a, computed as

$$Cov(\Phi_a) = \frac{1}{ns} \Phi_a \Phi_a^T$$

This time we retain a number of nu eigenvectors $U = \{U_i\}_{i=1}^{nu}$, which are sorted in decreasing order accordingly to their associated eigenvalues $\{\gamma\}_{i=1}^{nu}$, and construct the matrix E_a of $m \times nu$ which has as columns all these eigenvectors. Now every appearance $a_k \in DA'$ can now be approximated by

$$a_k = \bar{a} + E_a \cdot d_k \tag{3.3}$$

for some vector $d_k \in R^{nu}$. Again in this equation the only variable term that we don't have values for is d_k, which can be computed for a given a_k in DA' as

$$d_k = E_a^T \cdot (a_k - \bar{a}) \tag{3.4}$$

The vectors a_k and d_k can be seen as two different representations of appearance, one with pixel intensities and the other as a vector in the variance parameter space R^{nu}. Similar to our shape model, if we now take the right had side of (3.3) as a function of $d \in R^{nu}$, with values of d not necessarily obtained from (3.4), we obtain

$$G_a(d) = \bar{a} + E_a \cdot d \tag{3.5}$$

Since G_a can produce appearance vectors for some d's (equation (3.4) for example) but not for all $d \in R^{nu}$, in order for G_a to be exclusively a generator of appearance vectors, we restrict it to the set that give us parameters within the appearance variance captured from the set DA' by PCA, defined as

$$P_a = \prod_{i=1}^{nu} [-2.5\sqrt{\gamma_i}, 2.5\sqrt{\gamma_i}] \tag{3.6}$$

In this way, the *i-esim appearance mode of variation* is defined as the function $G_a(f \cdot e_i)$, for the $i - esim$ canonical vector $e_i \in R^{nu}$, and for f varying in $[-2.5\sqrt{\gamma_i}, 2.5\sqrt{\gamma_i}]$, since we move from the mean appearance in the direction of the $i - esim$ principal component U_i (evaluating in (3.5)):

$$G_a(f \cdot e_i) = \bar{a} + f \cdot U_i$$

Summarizing, our PCA appearance model is composed of the function G_a defined in (3.5), restricted to the set P_a defined in (3.6).

Chapter 4

PCA Shape + Appearance Model

4.1 Parameter Space

In the two previous chapters we have constructed models for shape and appearance separately. These models based on Principal Component Analysis (PCA) are totally described by equations (2.7) and (3.5):

$$G_s(b) = \bar{s} + E_s \cdot b; \quad b \in P_s = \prod_{i=1}^{nv} [-2.5\sqrt{\lambda_i}, 2.5\sqrt{\lambda_i}] \subset R^{nv} \tag{4.1a}$$

$$G_a(d) = \bar{a} + E_a \cdot d; \quad d \in P_a = \prod_{i=1}^{nu} [-2.5\sqrt{\gamma_i}, 2.5\sqrt{\gamma_i}] \subset R^{nu} \tag{4.1b}$$

G_s and G_a conforms what is known as *Independent AAMs*. In order to construct a model that accounts for both shape and appearance correlations, we make use of our training samples representations in the parameters spaces P_s and P_a (refer to equations (2.6) and (3.4)):

$$\begin{aligned} b_k &= E_s^T \cdot (s_k - \bar{s}); \quad s_k \in DS' \\ d_k &= E_a^T \cdot (a_k - \bar{a}); \quad d_k \in DA' \end{aligned}$$

and create the unified parameter samples

$$c_k = \begin{bmatrix} W_s b_k \\ d_k \end{bmatrix}; \quad c_k \in DC \subset R^{nv} \times R^{nu} \tag{4.2}$$

where W_s is a diagonal square matrix of $nv \times nv$ that accounts for the difference in unit measure between shape (distance) and appearance (intensities). A common definition of W_s is (see section 5.2.1 of [Cootes et al., 2000])

$$W_s = rI$$

where r is found to be the ratio of all variance in texture to all variance in shape

$$r = \frac{\sum_{i=1}^{nu} \gamma_i}{\sum_{i=1}^{nv} \lambda_i}$$

We will denote by DC to the collection of all the combined parameters c_k. By definition of the sets DS' and DA', DC turns out to be a discrete set of a population set of combined parameters, a parameter space that we denote by C. Also by definition of c_k, $C \subset R^{nv} \times R^{nu}$.

In summary we have go from shape and appearance spaces, to a parameter space C where we have a collection of training samples DC.

4.2 Combined Shape and Appearance Model

Similar to the already defined PCA models, we can now construct a third model that accounts for correlations between shape and appearance parameters. To model variance in DC, we will use PCA again. The first thing to note about the set DC is that, since both b_k and d_k has zero mean (remember that these parameters were found by PCA), DC has zero mean: $\bar{c} = 0$. To perform PCA on DC, we construct the matrix Φ_c as having columns equal to each sample c_k of DC, and so we find the directions of principal variance as the eigenvectors of the covariance matrix

$$Cov(\Phi_c) = \frac{1}{ns} \Phi_c \Phi_c^T$$

Here ns denote the number of samples in DC, which is the same number of samples in DS' and DA', that is, the number of training images in our database. We retain ne number of eigenvectors $N = \{N_i\}_{i=1}^{ne}$ of this matrix, sorted in decreasing order accordingly to its correspondent set of eigenvalues $\Lambda = \{\alpha\}_{i=1}^{ne}$. The matrix whose columns are N_i is denoted by E_c, which is of order $(nv + nu) \times ne$ by definition. Similar to the other PCA models, and since $\bar{c} = 0$, now every combined parameter $c_k \in DC$ can be approximated by

$$c_k = E_c \cdot h_k \tag{4.3}$$

for some $h_k \in R^{ne}$, which can be found by knowing the parameter c_k.

$$h_k = E_c^T \cdot c_k \tag{4.4}$$

Observe that we have gone from the parameter set DC to another set of parameters h_k defined by this equation. Also note that as before, we have constructed a generator function of combined parameters

$$G_c(h) = E_c \cdot h \tag{4.5}$$

with h restricted to the set

$$P_c = \prod_{i=1}^{ne} [-2.5\sqrt{\alpha_i}, 2.5\sqrt{\alpha_i}] \tag{4.6}$$

4.3 Combined Model Instance

Given any parameter $h \in P_c$ and the our combined model (4.5), how do we obtain its associated shape and appearance?, and how do we visualize them?. This is call a *combined model instance*. For this, we begin by simply evaluating (4.5) on this h to obtain a vector c:

$$c = G_c(h) = E_c \cdot h.$$

Since E_c is of order $(nv + nu) \times ne$ and $h \in R^{ne}$, we have that $c \in R^{nv+nu}$. We can then separate this vector as being composed of two vectors, one of R^{nv} and the other of R^{nu}:

$$c = \begin{bmatrix} wb \\ d \end{bmatrix}; \quad wb \in R^{nv}, \quad d \in R^{nu}$$

where wb represent the shape parameters, and d the appearance parameters. However since when we constructed our combined model we scaled the shape parameters by the weight matrix W (refer to (4.2)), we can compute our true shape parameters by un-weighting wb:

$$b = W^{-1} wb$$

and so $b \in R^{nv}$ are corrected to its units of measure. Having b and d, we can now use (4.1) to compute both shape and appearance

$$s = G_s(b) = \bar{s} + E_s \cdot b \tag{4.7a}$$

$$a = G_a(d) = \bar{a} + E_a \cdot d; \tag{4.7b}$$

From our construction of our appearance model, this appearance however is a warped version of a region of interest to the mean shape $\bar{s}$. To recover this region of interest we back-warp a to the found shape s in a similar fashion as described in section 3.1. Some examples results of this process are shown on figure 4.1.

Figure 4.1: Region of Interest Reconstruction: The appearance (middle) is warped back to the shape found by equations (4.7) to reconstruct the region of interest (right) of the original image(left).

Conclusion

Active appearance models combine mathematical representations of both shape and appearance to capture the inherent spread of data (variance) of these properties inside a collection of images. Independent models as well as combined models which takes into account correlations between shape and appearance parameters, can be used to represent and interpret images of a class of objects of our interest. The main mathematical tool for constructing such models was Principal Component Analysis, and a good understanding of linear algebra gave us some insights on what was happening intuitively behind the equations.

Extensions for 3D AAM shapes and appearance can be performed by using calibrated cameras and similar formulations to the exposed in this document. Applications of AAMs are face recognition, expressions recognition, vision based performance driven facial animation and tracking among others [Sonka et al., 2007]. The present document can be used as the start point for research and practice of AAMs, since by knowing this kind of models from its roots, we became able develop applications in a desired field while understanding the theory that can be used for further innovative research.

Acknowledgements

This research was supported by the European Union FP7 Integrated Project VERE (No. 257695), Instituto de Telecomunicacoes - IT and Fundacão para a Ciência e a Tecnologia - FCT, Portugal.

Bibliography

[Berg et al., 2008] Berg, M. d., Cheong, O., Kreveld, M. v., & Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, 3rd ed. edition.

[Cootes et al., 2000] Cootes, T., Taylor, C., & Manchester, M. P. (2000). Statistical models of appearance for computer vision.

[Cootes et al., 2001] Cootes, T. F., Edwards, G. J., & Taylor, C. J. (2001). Active appearance models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23, 681–685.

[Cootes et al., 1995] Cootes, T. F., Taylor, C. J., Cooper, D. H., & Graham, J. (1995). Active shape models: their training and application. *Comput. Vis. Image Underst.*, 61, 38–59.

[Nordstrom et al., 2004] Nordstrom, M. M., Larsen, M., Sierakowski, J., & Stegmann, M. B. (2004). *The IMM Face Database - An Annotated Dataset of 240 Face Images*. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby.

[Shlens, 2009] Shlens, J. (2009). *A Tutorial on Principal Component Analysis*. Technical report, Center for Neural Science, New York University.

[Sonka et al., 2007] Sonka, M., Hlavac, V., & Boyle, R. (2007). *Image Processing, Analysis, and Machine Vision 3rd Ed.* Thomson-Engineering.