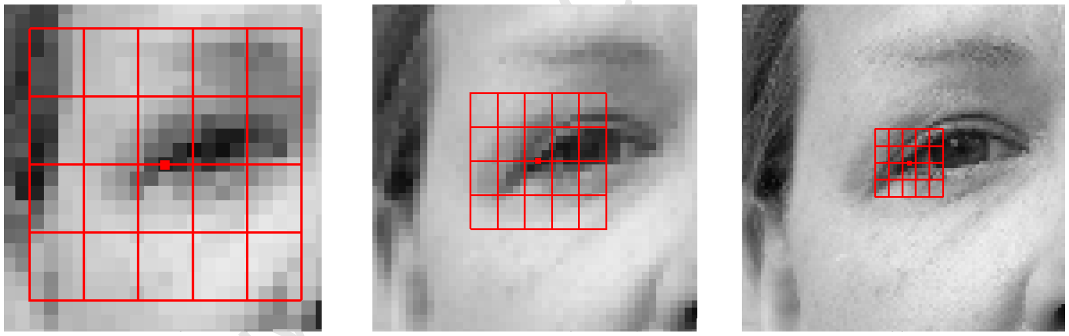
Multiview Active Shape Models with SIFT Descriptors

Stephen Milborrow



Supervised by Dr Fred Nicolls

Submitted to the
Faculty of Engineering, University of Cape Town,
for the Degree of Doctor of Philosophy

February 17, 2016
Minor Revision 6

Declaration

I, Stephen Milborrow, declare that this dissertation is my own work except where otherwise stated. It is being submitted for the degree of Doctor of Philosophy at the University of Cape Town. It has not been submitted before for any degree or examination at any other university.

Signature of Author ....

February 14, 2016, Cape Town, South Africa

Abstract

This thesis presents techniques for locating landmarks in images of human faces.

A modified Active Shape Model (ASM [21]) is introduced that uses a form of SIFT descriptors [68]. Multivariate Adaptive Regression Splines (MARS [40]) are used to efficiently match descriptors around landmarks. This modified ASM is fast and performs well on frontal faces.

The model is then extended to also handle non-frontal faces. This is done by first estimating the face's pose, rotating the face upright, then applying one of three ASM submodels specialized for frontal, left, or right three-quarter views. The multiview model is shown to be effective on a variety of datasets.

To the memory of Carolyn.

Contents

Chapter 1

Introduction

This thesis investigates methods for automatically finding landmarks in images of faces. A *landmark* is a distinguishable point present in most of the images under consideration. An example is the pupil of the left eye. Figure 1.1 shows a face with landmarks.

The overall intention is to introduce methods that perform well enough to be used in real-world applications, with software that is straightforward, portable, and easily maintainable. To this end, the methods investigated are essentially extensions to the well known Active Shape Model (ASM) of Cootes and Taylor [23].

We look for improvements to the ASM because, like all current automatic landmarking techniques, it's sometimes inaccurate and sometimes fails completely. Figure 1.2 shows the results of an ASM search that is only partially successful.

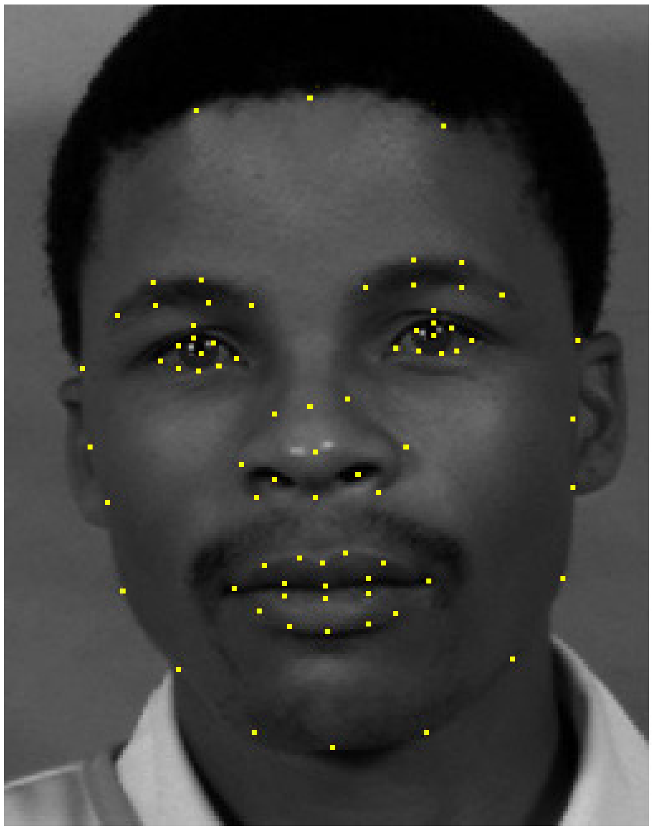


Figure 1.1: *A face with 77 landmarks.*

Face images in this document are from the MUCT set [81], except where noted.

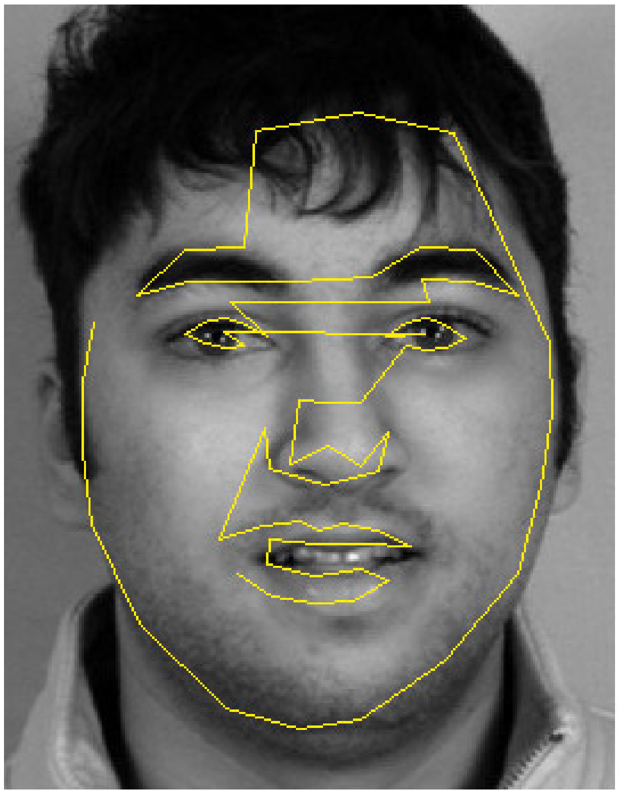


Figure 1.2:

Results of an unsatisfactory ASM search.

Note the misplacement of mouth and other landmarks.

1.1 New contributions

The main new contributions of this thesis are the integration of SIFT/HAT descriptors into ASMs and the way the model is extended to handle non-frontal faces. This section describes these contributions for readers already familiar with the field of automatic landmarking. (All terms and technology mentioned below will be described in detail in the body of the thesis.)

The first contribution (Chapter 8) replaces the one-dimensional gradient descriptors used in the classical ASM [23] with a form of the descriptors used in Lowe's SIFT algorithm [68]. These descriptors, called Histogram Array Transforms (HATs [83]), are like the descriptors used in the SIFT framework, but can be generated and tested against the image much more quickly than standard SIFT descriptors. Multivariate Adaptive Regression Spline (MARS [40]) models are used to match image descriptors against the model, rather than the Mahalanobis distances used by the classical ASM. The MARS models bring both speed and better predictions in this context.

This HAT-based ASM performs well on frontal faces, with close to state-of-the-art accuracies on the BioID data (Figure 8.18).

The second contribution (Chapter 9), extends the HAT-based model to handle faces with different poses, for example rotated or three-quarter views of faces. The face's pose is first estimated by applying face detectors specialized for frontal and three-quarter faces. These face detectors are also applied to rotated versions of the image. They are Viola-Jones detectors [114] optimized for specific poses and tuned to be generous with their detections—they usually give multiple detections for the face, with most detections from the detector that most closely matches the face's yaw.

Once the pose is estimated by aggregating the results from these detectors, the image is rotated so the face is upright. Finally one of three ASM submodels optimized for frontal, left, or right three-quarter views is applied.

This simple multiview model is fast but doesn't quite reach state-of-the-art accuracies (Figures 9.14 and 9.15). It performs well enough and with sufficient speed for practical applications on realistic data.

1.2 Not the problem

There are several related problems and techniques that are *not* part of this thesis:

(i) Finding the **overall position of face**. The automatic landmarker does a *local* search for facial features. Before it starts, the overall position of the face must be determined with a standard face detector.

(ii) Locating landmarks in features in **images other than faces**. Only faces are considered, although the software accompanying this thesis can build models for other objects, such as those in medical images or components on electronic circuit boards [79].

(iii) Finding landmarks in **low-quality images** of faces. This thesis focuses on the medium- to high-resolution faces typically processed by commercial applications like automatic and interactive retouchers or remodelers, and applications that apply virtual makeup or eye wear. Faces for these applications are typically of good quality, or at least not of bad quality. Very blurry or low-resolution faces are unusual in this environment, and aren't considered important. Note that the goal here is a practical target that isn't the same as trying to achieve competitive performance on the artificially difficult face datasets that have become popularly cited in academic papers in recent years.

(iv) **Slow landmarking**. Speed is important, because in the applications we have in mind the user must wait while the automatic landmarker does its work. The face and its landmarks should be located in less than a second, and preferably much less than that.

(v) **Recognizing the face**.

(vi) Detecting **the facial expression** e.g. smiling, frowning. There are many schemes to recognize the face or facial expression. Some of these schemes require landmarks. We are concerned here only with locating the landmarks, not really with the details of how they are used.

(vii) Using **color** during search. In this thesis we use only gray-level information.

(viii) Using **3D image information**. We restrict ourselves to a 2D approach.

(ix) Consolidating **multiple images** using motion clues, etc. We search for landmarks using a single image.

1.3 The Stasm software

The methods presented in this thesis are embodied in the open-source *Stasm* software, available at `http://www.milbo.users.sonic.net/stasm`. This thesis isn't about the software, but refers to it for concreteness and to demonstrate that the methods work.

Versions of Stasm and the associated documents are shown in the table below. When this thesis refers to Stasm it's referring to version 4.2, unless stated otherwise.

Version	**Technique**	**Documentation**
Stasm 1.5	2D gradient descriptors	Milborrow and Nicolls ECCV 2008 [82]
Stasm 3.1	2D gradients, optimized version of 1.5	e.g. Çeliktutan et al. EURASIP 2013 [15]
Stasm 4.0	HAT descriptors	Milborrow and Nicolls VISAPP 2014 [83]
Stasm 4.2	HAT descriptors, optimized version of 4.0	Chapter 8 of this thesis
multiview Stasm	HAT descriptors, multiple models	Chapter 9 of this thesis Milborrow and Nicolls ICCV 2013 [80]

1.4 Why this approach?

The methods proposed in this thesis were initially developed in 2011 and 2012 on a contractual basis for a commercial company. Based on the distribution of hundreds of thousands of faces already processed by their software in the field, the company was primarily interested in medium- to high-resolution faces, with prominence given to frontal faces although three-quarter views were also considered important.

The company had already independently evaluated packages for automatic landmarking, and had settled on Stasm mainly because of its landmarking performance on the faces of interest. Taking this into account and to manage uncertainty we decided on an "extend Stasm" approach—as opposed to say an approach of "go off for an indefinite period and come back with something that is much better" or "go off for several months and code up an algorithm in one of the papers that claim (but may not actually yield) better performance". It was less risky to extend the existing model than to adopt a radically new approach, even though the potential improvements in performance would be perhaps not as big. Another advantage of the chosen approach is that it allowed interim software releases to benefit end users without obliging them to wait for the entire project to be completed.

Stasm at that time used 2D gradient descriptors (Section 5.6). Given the many advances made in image descriptors since those were implemented, updating the descriptors was a clear way forward, with SIFT based techniques being of primary interest because of their established reputation. Also, improved performance on non-frontal faces was necessary, and by combining ASM submodels this improved performance could be achieved without a completely new type of model. The overall goal was simply to come up with a model that located landmarks adequately on the faces of interest.

This goal was easily met with the methods described in this thesis. And indeed, although some current competing models outperform these methods (Section 9.6.7), this usually doesn't translate into notably better fits in actual use on the faces of interest in the field.

1.5 About this document

The layout of this document is as follows. Chapters 2 to 5 give a detailed description of Active Shape Models from the ground up. Techniques for generating the start shape are also explored. Chapter 6 discusses how these models are evaluated and compared. These five chapters essentially form a textbook on ASMs.

A survey of the literature is given in Section 2.6. (This survey is deferred to after ASMs are introduced to make the review more meaningful to readers who are coming to the technology for the first time.) Additional reviews of the relevant literature appear in context as the various sub-technologies are described, primarily in Sections 8.1, 8.7, 8.12, and 9.1.

Chapter 7 reviews some aspects of landmarked face databases. Such databases are necessary for training and evaluating automatic landmarkers. This chapter describes the "MUCT" face database, created by the author and his colleagues to provide more diversity of lighting, age, and ethnicity than similar landmarked databases at the time it was created (2009). The MUCT data was used for training the models in this thesis because of these characteristics, and also because of the resolution of the faces and the quality of manual landmarking. Chapter 7 also presents an empirical study of uncertainty in manual landmarking, and discusses the "ideal" face database.

The aforementioned Chapters 2 to 7 essentially describe existing technology at the time this project was started (although they also present a few new ideas). The next two chapters present the principal new contributions of this thesis. Specifically, Chapter 8 introduces a model based on the new HAT descriptors, and Chapter 9 then extends the HAT model to handle non-frontal faces. Finally, Chapter 10 wraps things up.

Three appendices describe principal components, Mahalanobis distances, and some general principles of model building and evaluation.

Users of ASMs come to the technology with vastly differing backgrounds. In providing engineering documentation for this diverse audience over the history of this project, I've tried to explain concepts as directly and simply as possible, without assuming a great deal of prior knowledge. Some of this engineering documentation now appears in this thesis, giving the thesis a somewhat informal tone. In attempting to be direct and to reduce use of the passive voice, I use the word "we" (and "our") in a very broad sense that ranges all the way from "I" to "people in general".

The layout of this thesis is similar in some ways to my master's thesis [75]. Both discuss

ASMs and introduce new extensions to ASMs. In the current document the description of classical ASMs is expanded. This expansion (and the inclusion of the appendices) was motivated by queries I've received from users over the years. This document also spends less time on the details of parameter tuning, with the understanding that model parameters were selected by tuning on parameter-selection data.

Some of the text and diagrams have previously appeared in the papers listed in the table on page 13, in the Stasm documentation, and in email and documentation written for users of the software.

1.6 Acknowledgments

Every new idea in this thesis arose out of discussions with other people. I especially thank the engineers I have worked with at various companies around the world. A big thanks goes to my thesis supervisor Fred Nicolls for his support. The external examiners are thanked for their time and insights, which have had a pronounced impact on this document. I'd also like to thank my friends and mentors Rick Andrew, Ralph Carmines, Clayton Cramer, Darren Murray, Andrew Nunn, Micheal Pucci, Paul Vroomen, Steve West, and Brad Yearwood. A thanks goes out to the people who published the databases used in this project, and to those who allowed their faces to be used.

Chapter 2

Active Shape Models

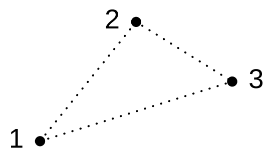
This chapter gives an overview of Active Shape Models (ASMs), followed by a literature review. The next three chapters will cover ASMs in more detail.

Occasionally we will refer the *classical* ASM described in e.g. Cootes and Taylor [23]. The classical ASM is characterized by a principal-component shape model and its use of Mahalanobis distances on one-dimensional gradient descriptors. What that all means will be explained in this and the next few chapters.

2.1 Shapes

For our purposes, a two-dimensional *shape* is simply an $n \times 2$ ordered set of points: a matrix of x,y coordinates (Figures 2.1 and 2.2). The position of a point relative to the other points is fixed. If you move a shape, it's still the same shape. If you expand or rotate it, it's still the same shape.

Edges between points aren't part of the shape but are often drawn to clarify the relationship or ordering between the points. The *centroid* (also simply called the *position*) of a shape is the mean of its point positions. A *centered* shape is a shape translated so it is positioned on the origin.



	x	y
Point1	**0.0**	**1.0**
Point2	**1.0**	**3.0**
Point3	**2.0**	**2.0**

x_1	**0.0**
y_1	**1.0**
x_2	**1.0**
y_2	**3.0**
x_3	**2.0**
y_3	**2.0**

Figure 2.1: ***Left*** *A simple shape, with three points*
Middle *The shape as a matrix*
Right *The shape as a vector*

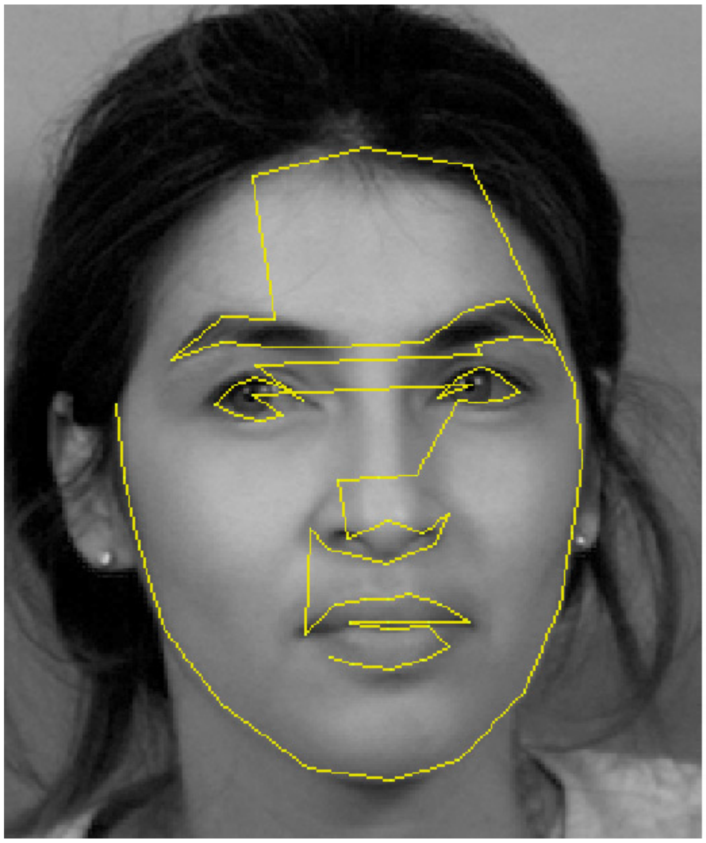


Figure 2.2: *A face shape.*

This is a manually landmarked face from the MUCT 77 point data [81].

2.2 Shape models

Shape models form one leg of the Active Shape Model. A *shape model* defines an allowable set of shapes. In this thesis, shape models have a fixed number of points and a matrix formula that specifies the relationship between the points.

A shape model learns allowable constellations of shape points from training shapes. The training shapes are manually landmarked face images. By *manually landmarked* we mean that somebody had to mark all the images by hand, by clicking on the screen. This is done before model building begins. There are various standard manually landmarked sets of faces.

A toy shape model

To gain some intuition about shape models, let's think about how we could create a model that allows us to generate plausible new shapes from a set of training shapes. A shape is considered plausible if it's near a training shape, inbetween, or around the training shapes.

To generate shapes, we could start with the mean shape and allow variations off this mean. (The *mean shape* is the average shape after aligning the training shapes.)

If we attached the points to the mean face with springs, we could generate different faces by stretching the springs. The more we stretch the springs the more the generated

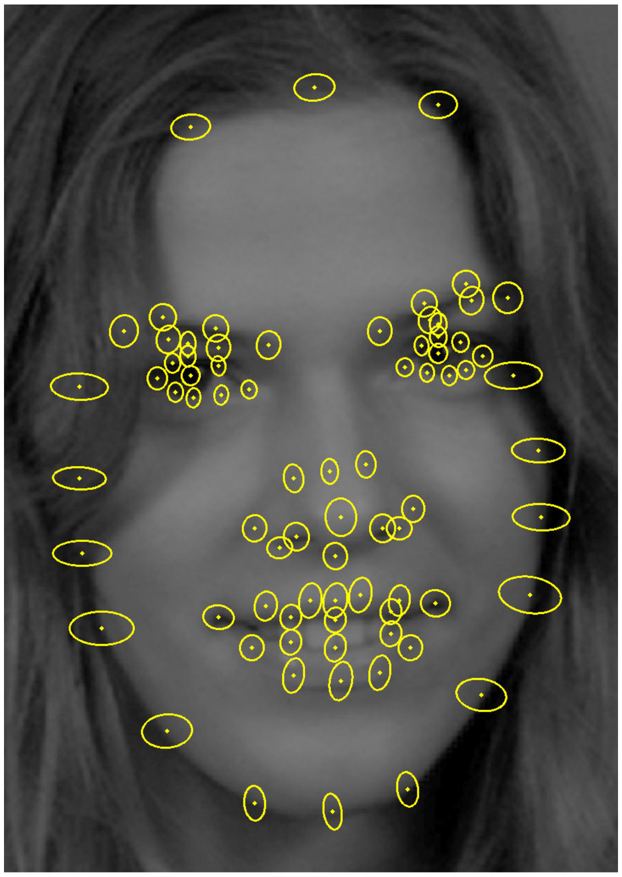


Figure 2.3: *Variance of points in the aligned training shapes. The points have been placed on a reference face.*

The ellipses are proportional to the standard deviations of the points (measured on a frontal subset of the MUCT shapes [81]).

Variation is caused by differences in face shapes and also by manual landmarking uncertainty. The biggest variation is seen in the points on the sides of the face. The nose tip also varies more than the points around it, possibly because of the difficulty of determining the position of the nose tip when manually landmarking the faces. The pupils are stable—because in this dataset most of the subjects look directly at the camera and pupils are easily located precisely during manual landmarking.

face would differ from the average face. Points that varied less in the training shapes would have stiffer springs than points that varied more: stiffness would have an inverse relationship to the variance of the point (Figure 2.3). A point whose spring has to be pulled very hard could indicate a problem—maybe the point is misplaced and the face shape isn't plausible.

This fanciful spring model treats points in isolation. In reality groups of points such as the points of an eye tend to move together. Groups of points are also related to each other in other ways—for example, if one eye is large, the other eye should also be large. To allow these types of relationships, we need to consider relationships between points, perhaps with some kind of "smart spring" that interconnects points. This starts getting unwieldy, so in practice instead of this crude spring approach we use shape models based on principal components. These models will be described in full in Chapter 3.

2.3 Overview of the Active Shape Model

This section gives an overview of Active Shape Models. The next few chapters will then describe in detail the two types of submodel of the ASM: the descriptor models and the shape model.

The ASM is first trained on a set of manually landmarked training images. Once training is complete, we can use the ASM to search for features on a face as follows.

```
input image of a face

1. Locate the face in the image with a face detector
2. Position the start shape on the face
3. repeat
4.         Suggest a new shape by descriptor matching around each shape point
5.         Adjust the suggested shape to conform to the Shape Model
6. until convergence

output shape giving the x,y coordinates of the face landmarks
```

Figure 2.4: *The ASM search algorithm.*

After locating the face in the image with a standard face detector, we generate a *start shape* by positioning the mean face shape on the face detector rectangle (Figure 5.1 on page 61 shows an example). The general idea then is (i) try to locate each landmark independently, then (ii) correct the locations if necessary by looking at how the landmarks are located with respect to each other. To do this, the ASM is constructed from two types of submodel:

(i) A *descriptor model* for each landmark, which describes the characteristics of the image around the landmark. The model specifies what the image is expected to "look like" around the landmark. During training, we build the descriptor model for the landmark by sampling the area around the landmark in the training images. During the search, we search around the tentative position of the landmark, and move the landmark to the position where the image surface best matches the descriptor model. This process, applied over all landmarks, generates tentative new positions for the landmarks, called the *suggested shape*.

(ii) A *shape model*, which defines the allowable relative position of the landmarks. During the search, the shape model adjusts the shape suggested by the descriptor models to conform to a legitimate face shape.

The algorithm iterates for a solution by alternating these submodels as shown in Figure 2.4. There are a variety of methods of testing for convergence [23, 75], but Stasm simply assumes convergence after iterating four times.

2.4 Interaction between the descriptor models and the shape model

The ASM algorithm combines the results of the weak descriptor matchers to build a stronger overall matcher. It's a shape-constrained feature detector: the shape model acts globally across all landmarks; each descriptor model acts locally in the image region around the facial feature of interest.

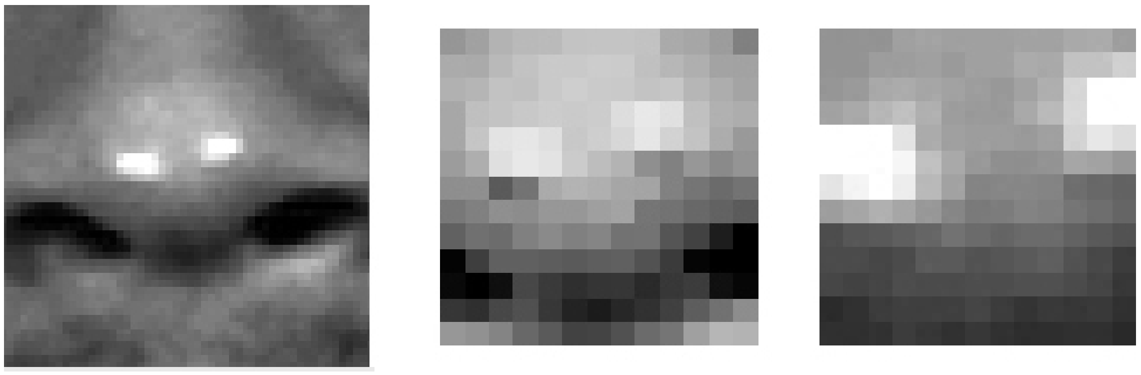


Figure 2.5: ***Left*** *The nose tip from Figure 1.1 on page 10.*
Middle 13×13 *region at half image scale.*
Right 13×13 *region at full scale.*

The shape model would be unnecessary if the descriptor models always positioned points correctly. But each descriptor matcher sees only a small portion of the image, so can't be completely reliable. For example, on the right of Figure 2.5, without wider context it's difficult to precisely determine the position of the tip of the nose. As you can imagine, the bright spots on the nose in this example don't make the matcher's job any easier.

Figure 2.6 illustrates a shape model at work. The left side of the figure shows four landmarks, one of which has been mispositioned by the descriptor matcher. It has been mispositioned on the collar (because the collar line is an edge much like a jaw line, at least in the small part of the image seen by the descriptor). The right side shows the landmarks after correction by the shape model. The mispositioned landmark is now correctly on the jawline.

Figure 2.7 shows a shape model operating on all points of a face shape during one iteration of an ASM search. In the left figure, some of the suggested points for the left[1] upper eyelid are in impossible positions—they are below the pupil. In the right figure, these points are in more sensible positions with respect to the surrounding points. Similar corrections can be seen elsewhere.

[1]In this thesis, *left* and *right* are with respect to the viewer, not the subject.

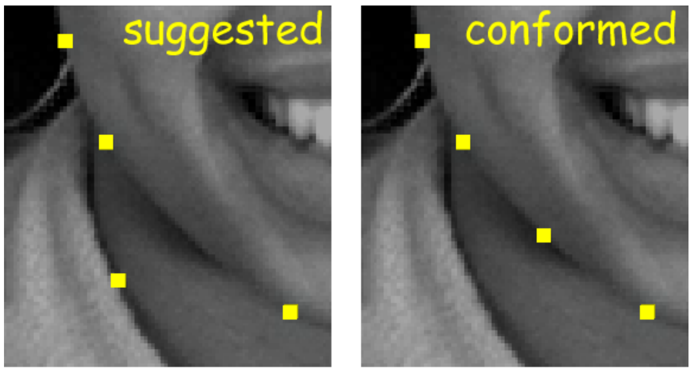


Figure 2.6: *Operation of the shape model.*

Left *Landmarks located by the descriptor models.*

Right *The same landmarks conformed to the shape model. The landmark that was snagged on the collar is now in a better position.*

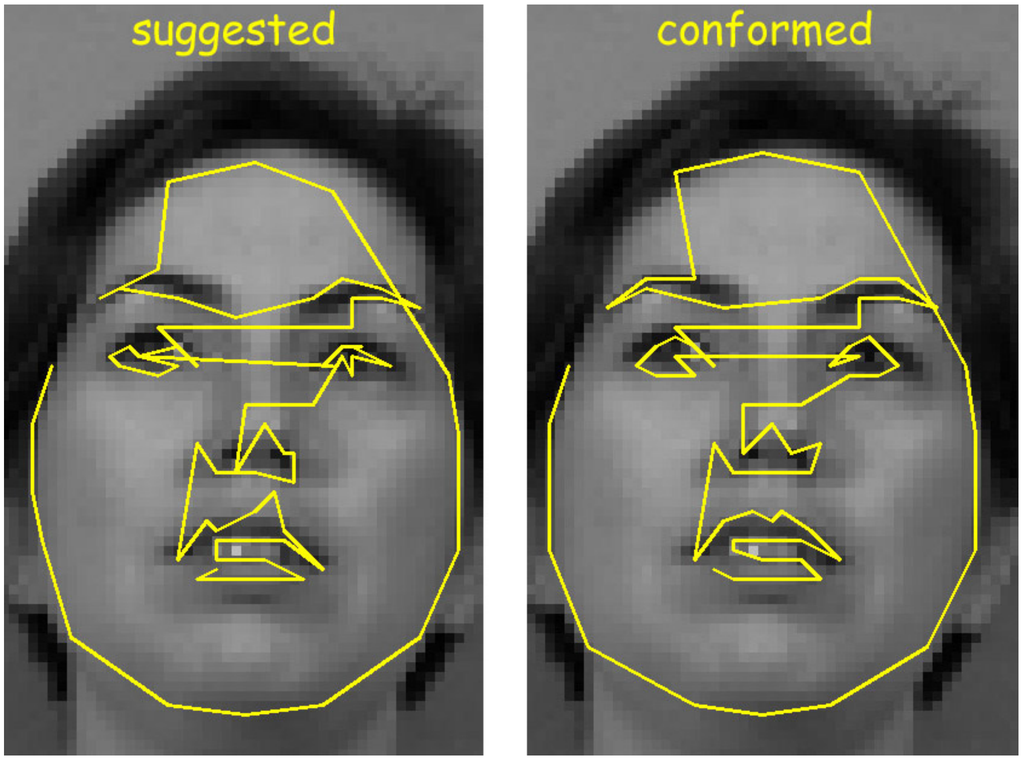


Figure 2.7: *Operation of the shape model.*

Left *A shape suggested by the descriptor models.*

Right *The shape after correction by the shape model.*

2.5 Multi-resolution search

An ASM search works best if we start off by approximately positioning the landmarks on a low resolution image, and then refine the search on higher resolution images. This multi-resolution approach converges more often to the correct shape than simply searching at full resolution, which is more prone to getting stuck in local minima.

Therefore, before the search begins we build an *image pyramid* [1]. Each image in the image pyramid is a down-scaled version of the image above it (Figure 2.8). We start the ASM search at the lowest resolution image in the pyramid, and repeat the search at each level, from coarse to fine resolution. The start shape for the first search (which is on the coarsest image) is the mean shape aligned to the global face detector. The start shape at subsequent levels is the best shape found by the search at the previous level. The shape model is the same across all pyramid levels (apart from scaling), but a separate descriptor model is needed for each point at each level.

Figure 2.10 on the following page shows the start shape, then one iteration of an ASM search at each level in a four-level pyramid.

There are a variety of methods for downscaling the pyramid images (e.g. Gonzalez and Woods [42]). Stasm uses bilinear interpolation with four pyramid levels, and halves the width of the image at each level.

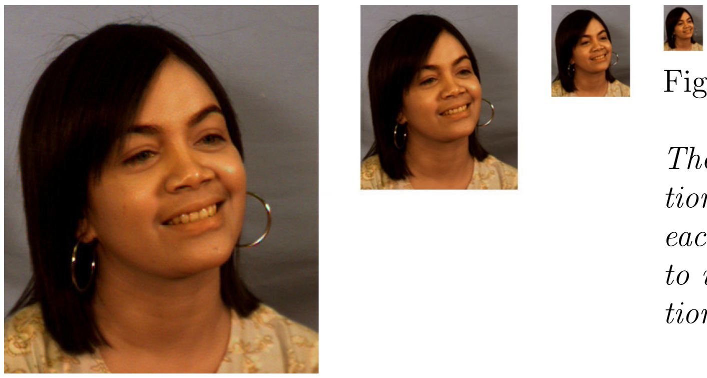


Figure 2.8: *An image pyramid.*

The left image is at full resolution. In this pyramid, the width of each image is half that of the image to its left (i.e. a quarter of the resolution).

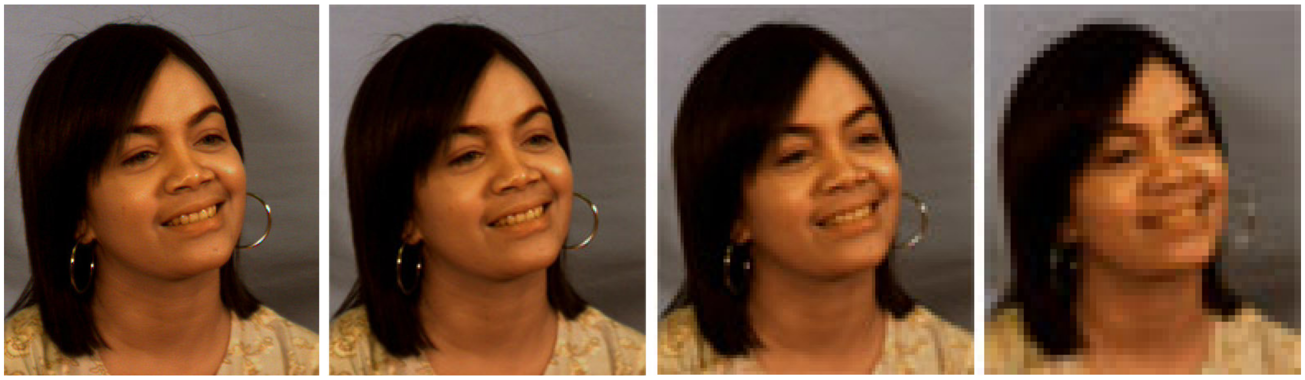


Figure 2.9: *The images in the above pyramid resized to the same size on the page. The loss of resolution in higher pyramid levels is evident.*

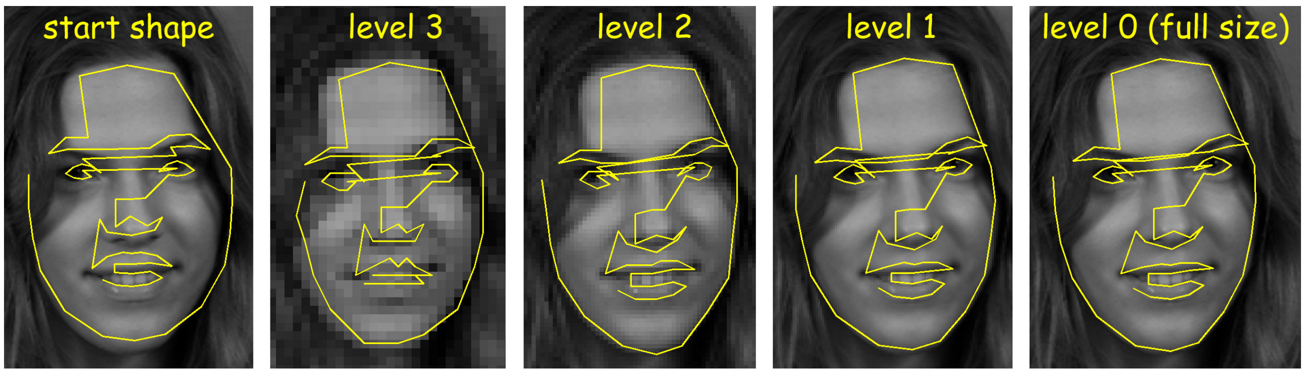


Figure 2.10: *The start shape and the ASM search at each level in a four-level pyramid. Each image shows the shape after correction by the shape model. Intermediate shapes aren't shown.*

2.6 Literature review

This section first presents a history of the ASM and its direct descendants. It then gives an overview of some of the current alternative approaches to facial landmarking. Additional overviews of the relevant literature appear in context later in this document as the various sub-technologies are described, primarily in Sections 8.1, 8.7, 8.12, and 9.1.

2.6.1 Predecessors of the ASM

ASMs belong to the class of models that, after a shape is situated near an image feature, interact with the image to warp the shape to the feature. A variety of such *deformable models* were developed before the ASM.

A classic example is the *snake*, also called the Active Contour Model (Kass et al. [56]). Snakes iteratively minimize the sum of "external" and "internal" energies. External energy is minimized as the snake molds itself to the desired image feature. Internal energy constrains the snake shape, typically by penalizing curviness.

A problem with snakes is that they can easily hitch onto the wrong image feature. Also, snakes don't like to bend and thus have difficulties locating sharply curved features. Researchers have modified snakes and/or used them in combined models to help deal with these issues. An example is the eye locater of Yuille et al. [123], which is a handcrafted deformable model that incorporates global information about eyes. The trouble here is that you have to manually design a new model for each application.

Another approach is to use finite element methods to represent prototype shapes (e.g. Pentland and Sclaroff [91]). This approach takes a single shape and treats it as if it were made of elastic material. However, the "modes of vibration" used to deform the shape may not represent the actual variations of the image feature being modeled.

Staib and Duncan presented a method of shape modeling based on Fourier descriptors [107]. (Fourier descriptors themselves were developed much earlier [92, 125].) Different shapes are generated by varying the parameters of a Fourier series defining the coordinates of the shape perimeter. Unfortunately these models cannot easily describe all shapes, especially sharp curves, and choosing parameters for the Fourier series is tricky.

2.6.2 Statistical shape models

Two independent fields of research led to the shape and descriptor models used in the ASM. One field, mostly independent of image analysis, was the development of statistical shape models [32, 57]. The central idea is that once you represent shapes as vectors (Section 3.1), you can apply standard statistical methods to them just like any other multivariate statistical object. These models learn allowable constellations of shape points from training examples and typically use principal components to build *Point Distribution Models*, which is another name for what we call shape models. These models have been used in diverse ways, for example for categorizing Iron Age brooches [105].

2.6.3 Locating image features

Feature location in images was the other field of research that led to the ASM. Initial research focused on *image segmentation*, which divides images into regions with similar characteristics. For example, we may want to separate grasslands from forests in satellite images. There are a host of techniques for doing this, often based on edge and line detection. These techniques are described in any image processing textbook (e.g. Gonzalez and Woods [42], Nixon and Aguado [86]).

For feature location, the most basic approach is to make a template of the average gray levels of the image feature in question. For example, we could make a template of the outer corner of the left eye in a frontal face. This template is then moved over the search image looking for matches. Variability of image textures and features make this simple technique inadequate, and a good deal of research has gone into developing *descriptors*, which are essentially an advanced form of template. For a general overview see e.g. Nixon and Aguado [86] or Prince [93].

2.6.4 Active Shape Models

Cootes and Taylor and their colleagues synthesized ideas from the fields of shape modeling and image processing to create the ASM. In 1991 they presented the first model for image analysis using a principal-component shape model [21]. The shape model can deform only in ways that are characteristic of the object—so if a beard, say, covers the chin, the shape model can "override the image" to approximate the position of the chin under the beard. Building on this approach they released a series of papers that culminated in what we call the classical ASM [23].

Apart from better performance at the time than other facial feature locaters, ASMs had the advantage that they didn't require a handcrafted model, although the training images had to be manually landmarked. ASMs are built by training on a set of reference images, a process which mechanically extracts important characteristics of the training set. In practice, hand tuning of parameters is still needed to get best results for a specific application.

2.6.5 Extensions to ASMs

Many extensions to the classical ASM have been proposed. This subsection summarizes some of the principal early developments.

Cootes et al. [22] presented a shape model that is a mixture of multivariate Gaussians, rather than assuming that the shapes come from a single multivariate Gaussian distribution.

Romdhani et al. [98] used Kernel Principal Component Analysis [103] and a Support Vector Machine [12]. This trains on 2D images, but models non-linear changes to face shapes as they are rotated in 3D. The idea is to project non-linearities into a higher-dimensional linear space.

Rogers and Graham [97] used robust least-squares techniques to minimize the residuals between the model shape and the suggested shape. Shape alignment in a classical ASM, in contrast, uses standard least-squares which is susceptible to outliers.

Van Ginneken et al. [41] took the tack of replacing the 1D gradient profiles used in the classical ASM with local descriptors calculated from "locally orderless images" [59]. Their method automatically selects the optimum set of descriptors. They also replace the standard ASM profile model search, which is based on Mahalanobis distances, with a k-nearest-neighbors classifier.

Y. Zhou et al. [131] presented the Bayesian Tangent Shape Model. This estimates shape and pose parameters using Bayesian inference after projecting the shapes into a tangent space.

My master's thesis [75] and associated paper [82] replaced the 1D profiles used in the classical ASM with 2D gradient descriptors. These descriptors are described in detail in Section 5.6.

Cootes, Edwards, and their colleagues took some of the ideas of the ASM significantly further with the development of the Active Appearance Model (AAM) [19, 35]. AAMs merge the shape and texture models into a single model of appearance. Texture is measured across the whole object, i.e. the whole face. An optimization technique that pre-calculates residuals during training is used when matching image texture to model texture.

The AAM has many descendants. A well-known example is the Constrained Local Model of Cristinacce [28].

2.6.6 Modern directions

Almost every method (and there are dozens) for locating face landmarks has its roots in the ASM, but researchers have now taken the technology in very new directions. Most of these new directions aren't of immediate relevance to us (since this thesis is mostly concerned only with extensions to ASMs), so this section just mentions some that were developed independently in parallel with the work in this thesis. More comprehensive surveys can be found in Saragih et al. [101] and Çeliktutan et al. [15].

In 2010, Dollár et al. [31] introduced Cascaded Pose Regression (CPR), primarily for estimating the pose of objects in images (where "pose" here means any set of systematic and parameterizable changes in the appearance of the object, such as the changing curves of a fish[2]). CPR takes an initial guess of the object's position and pose, and then successively refines it. Each refinement is carried out by a different regressor, which accumulates the results on "pose-indexed features" (meaning features whose output depends on both information from the image and the current estimate of pose).

[2]Elsewhere in this document we use "pose" in a slightly less general sense, to mean the angle from which the face was photographed, or equivalently, the head's *yaw*, *pitch*, and *roll* with respect to the camera. Figure 3.16 on page 44 illustrates these terms.

CPR became important for facial landmarking when it was adapted for that purpose in 2012 by Cao et al. [13]. Their method, in addition to using CPR, estimates adjustments to the shape as a whole, that is, it uses a model of appearance over all landmarks on the face (not per-landmark), without making use of an explicit shape model. The fitted face shape is a linear combination of the training shapes. This method is significantly faster than the 2011 method of Belhumeur et al. [7] on the BioID data [2], although with only a small increase in landmarking accuracy on that data.

The Supervised Descent Method (SDM) presented in 2013 by Xiong and de la Torre [120] has become very influential. This method optimizes the approach of Cao et al. by using HOG descriptors [29] and a highly efficient least-squares technique for inference: it learns a sequence of descent directions during training that makes optimization during the search for landmarks efficient and simple to implement.

The above-mentioned SDM has become the basis for many other methods, such as the 2015 models of Martinez and Valstar [71] and Tzimiropoulos [110]. These two methods will be considered in Section 9.6.7, where we compare results to the model proposed in this thesis.

It's possible that the SDM and its descendants will now be supplanted by alternative approaches such as that of Ren et al. [96]. Time will tell.

Chapter 3

Shape models

As part of its overview of ASMs, the previous chapter gave an overview of shape models (Section 2.2). Now in this chapter we go into the details of shape models. Descriptor models will be covered in a later chapter.

Recall that a shape model defines a set of allowable shapes, and that in an ASM the shape model converts shapes suggested by the descriptor models to plausible face shapes. Obviously our focus here is on faces, but nearly everything in this chapter also applies to other types of shapes, such as those used in medical images. (Perhaps the biggest difference in practice between models for faces and for medical images is that current training sets for medical images tend to be much smaller, raising some statistical and mathematical issues that don't bother us here.)

3.1 Shape space: shapes as vectors

Remember from Section 2.1 on page 16 that a *shape* with n points is just an $n{\times}2$ matrix of x,y coordinates related to each other in some invariant sense.

It's often convenient to represent a shape not as an $n{\times}2$ array of x,y coordinates, but as a $2n{\times}1$ vector: the x,y coordinates of the first point followed by the x,y coordinates of the remaining points:

$$\boldsymbol{x} = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n)^T. \tag{3.1}$$

This representation is used in the equations in this document and in our software. (Many authors use a different ordering convention: first all the x coordinates, then all the y coordinates. Either convention is fine.)

We can work with this vector as a mathematical object in multidimensional space, just like any other vector. The equations operating on this vector don't know or care that it was originally a shape consisting of pairs of x,y coordinates.

The $2n$-dimensional vector exists in what is called *shape space*. An entire shape is a single point in shape space. The training shapes are a swarm of points in shape space. Plausible face shapes exist in a subregion, or *manifold*, of the shape space.

3.2 Distances and sizes

The *distance between two shapes* $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, written as $|\boldsymbol{x}_1, \boldsymbol{x}_2|$, is the square-root of the sum of the squared distances between their corresponding points:

$$|\boldsymbol{x}_1, \boldsymbol{x}_2| \quad = \quad \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}_2) \cdot (\boldsymbol{x}_1 - \boldsymbol{x}_2)}\ . \tag{3.2}$$

The *size* of a shape is the square-root of the sum of the squared distances from its points to its centroid.

These measures are often normalized by dividing by the number of points in the shape. There are other measures of distance and size but we don't need them.

3.3 Transforming shapes

We can move a shape, rotate it, or change its size. In the language of linear algebra, these are *linear-translation*, *rotation*, and *scaling* transforms. A transform that does all three (but nothing else, such as shearing) is a *similarity* transform.

Now let's briefly focus on single points, rather than on shapes. The similarity transform $\boldsymbol{T}$ that rotates the point x,y by θ, scales it by s, and translates it by $x_{translate}$ and $y_{translate}$ is

$$\boldsymbol{T}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_{translate} \\ y_{translate} \end{pmatrix} + \begin{pmatrix} s\ \cos\theta & s\ \sin\theta \\ -s\ \sin\theta & s\ \cos\theta \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}. \tag{3.3}$$

The above equation requires two transformation matrices: one for translation and one for scaling and rotation. The 2×1 translation matrix is necessary because matrix multiplication can't shift the origin: zero multiplied by anything is still zero. However it's often more convenient to represent the entire transformation as a single matrix. We do this using the trick of *homogeneous coordinates*, by temporarily appending a dummy entry fixed at 1 to the two-dimensional x,y vector. We can then specify the above transform with a single 3×3 matrix:

$$\boldsymbol{T}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} s\ \cos\theta & s\ \sin\theta & x_{translate} \\ -s\ \sin\theta & s\ \cos\theta & y_{translate} \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3.4}$$

The above transform can be applied to a shape (as opposed to a single point) simply by applying it independently to each point in the shape.

3.4 Aligning a shape to another shape

A shape can be *aligned* to another shape by applying a transform that yields the minimum distance between the shapes. Appendix B in Cootes and Taylor [23] gives a method to align two shapes. It aligns the shape $\boldsymbol{x}_1$ to the shape $\boldsymbol{x}_2$ using a least-squares procedure: it calculates the transform $\boldsymbol{T}$ that minimizes the distance $|\boldsymbol{T}(\boldsymbol{x}_1) - \boldsymbol{x}_2|$. This kind of alignment is sometimes called *Procrustes alignment* (after a figure from Greek mythology).

Accurate alignment may be deemed more important for certain points than for others, and points can be weighted accordingly during alignment. But in this thesis, points either have a weight of one or zero—a point is either used or not-used (Section 3.13).

3.5 Aligning a set of shapes

During training we need to align not just two shapes but a set of shapes. By definition, alignment means that the total distance from the aligned shapes to the mean shape is minimized. The *mean shape* is the average of the aligned shapes.

If we knew the mean shape beforehand, we could simply align all the shapes to the mean shape and be done. Since we don't have this prior knowledge (it's a chicken and egg situation), we instead create an initial provisional mean shape from a *reference shape*, and iterate using the algorithm in Figure 3.1. The reference shape can be any shape in the set of shapes; the first shape is usually as good as any.

Line 6 of the algorithm constrains the mean shape to have approximately the same centroid and size as the reference shape. This step is necessary because without some constraints multiple shape alignment is ill-defined, in the sense that a set of shapes can be aligned but with their mean positioned anywhere. Also, we have to make sure that the algorithm doesn't minimize distances by making the shapes smaller and smaller. (Technically, shapes can be aligned by simply zeroing all the points.)

Figure 3.2 shows the reference shape for the Stasm frontal model, and the mean shape aligned to the reference shape.

Before alignment begins, it may be beneficial to position the reference shape on the origin and prescale its size to unity. However this isn't essential—the absolute position

```
input a set of unaligned shapes

1. Select a reference shape. (The reference shape
   doesn't change. It is typically the first shape.)
2. Generate the initial mean shape as a copy of the reference shape.
3. repeat
4.         Align all shapes to the mean shape.
5.         Recalculate the mean shape from the aligned shapes.
6.         Align the mean shape to the reference shape.
7. until convergence (the mean shape is stable)

output the mean shape and the set of aligned shapes
```

Figure 3.1: *Aligning a set of shapes.*

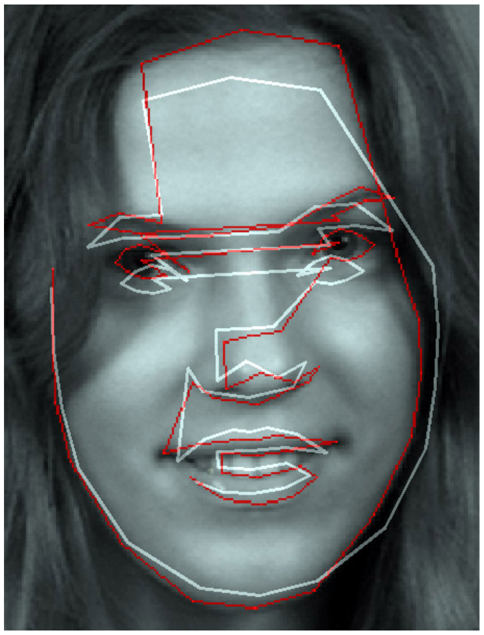


Figure 3.2:

The Stasm reference shape (red) and the mean shape (white).

The reference shape is the first shape in the training set.

and scale of the model space isn't important. It's also convenient if the reference shape (and hence the model mean shape) is approximately upright. For face shapes, "upright" can be defined to mean that the eyes in the face are horizontal.

Scaling and rotating shapes during alignment introduces artificial non-linearities [23, 108]. The effect of these can be minimized by projecting shapes into a tangent space, but tangent spaces aren't used in our software.

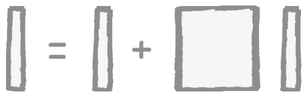
Analytic methods exist for aligning shapes [43], but the iterative algorithm is uncomplicated, and easily fast enough in practice. Only four iterations are needed to align several thousand training shapes when training the Stasm frontal model, taking well under a second. Cootes and Taylor [23] discuss some variations of the iterative algorithm.

3.6 Shape models

To create the shape model, we use a standard principal components approach (Appendix A) to generate a set of directions, or axes, along which the mean shape can be flexed in shape space to best represent the way the faces vary in the training set. This is done as described below. Figure 3.3 is an overview. Remember that we represent shapes in these equations as vectors in shape space (Section 3.1).

To generate the shape model, first the training shapes must be aligned as described in the previous section. The shape model then consists of the mean aligned shape $\bar{\boldsymbol{x}}$ and allowed distortions from the mean:

$$\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi b}\,, \qquad (3.5)$$



where $\hat{\boldsymbol{x}}$ is the generated shape. The hat on $\hat{\boldsymbol{x}}$ reminds us that it is generated by a model. The sketch on the right of the formula illustrates the shapes of the participants.

The mean shape $\bar{\boldsymbol{x}}$ is the average of the aligned training shapes $\boldsymbol{x}_i$, i.e.,

$$\bar{\boldsymbol{x}} \quad = \quad \frac{1}{n_{shapes}} \sum_{i=1}^{n_{shapes}} \boldsymbol{x}_i \ . \tag{3.6}$$

Each column of the square matrix $\boldsymbol{\Phi}$ is a unit-length eigenvector, a principal axis of the aligned shape space. That is, the matrix $\boldsymbol{\Phi}$ is the eigenmatrix of the covariance matrix $\boldsymbol{S}_{shapes}$ of the aligned training shapes. We will examine eigenvectors in more detail shortly.

The covariance matrix $\boldsymbol{S}_{shapes}$ is defined in the standard way, i.e.,

$$\boldsymbol{S}_{shapes} \quad = \quad \frac{1}{n_{shapes} - 1} \sum_{i=1}^{n_{shapes}} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \ . \tag{3.7}$$

The vector $\boldsymbol{b}$ in Equation 3.5 is a set of parameters that we can vary to generate different shapes. In other words, by varying the elements of the parameter vector $\boldsymbol{b}$ we generate different linear combinations of the principal axes in $\boldsymbol{\Phi}$, and thus generate different shapes.

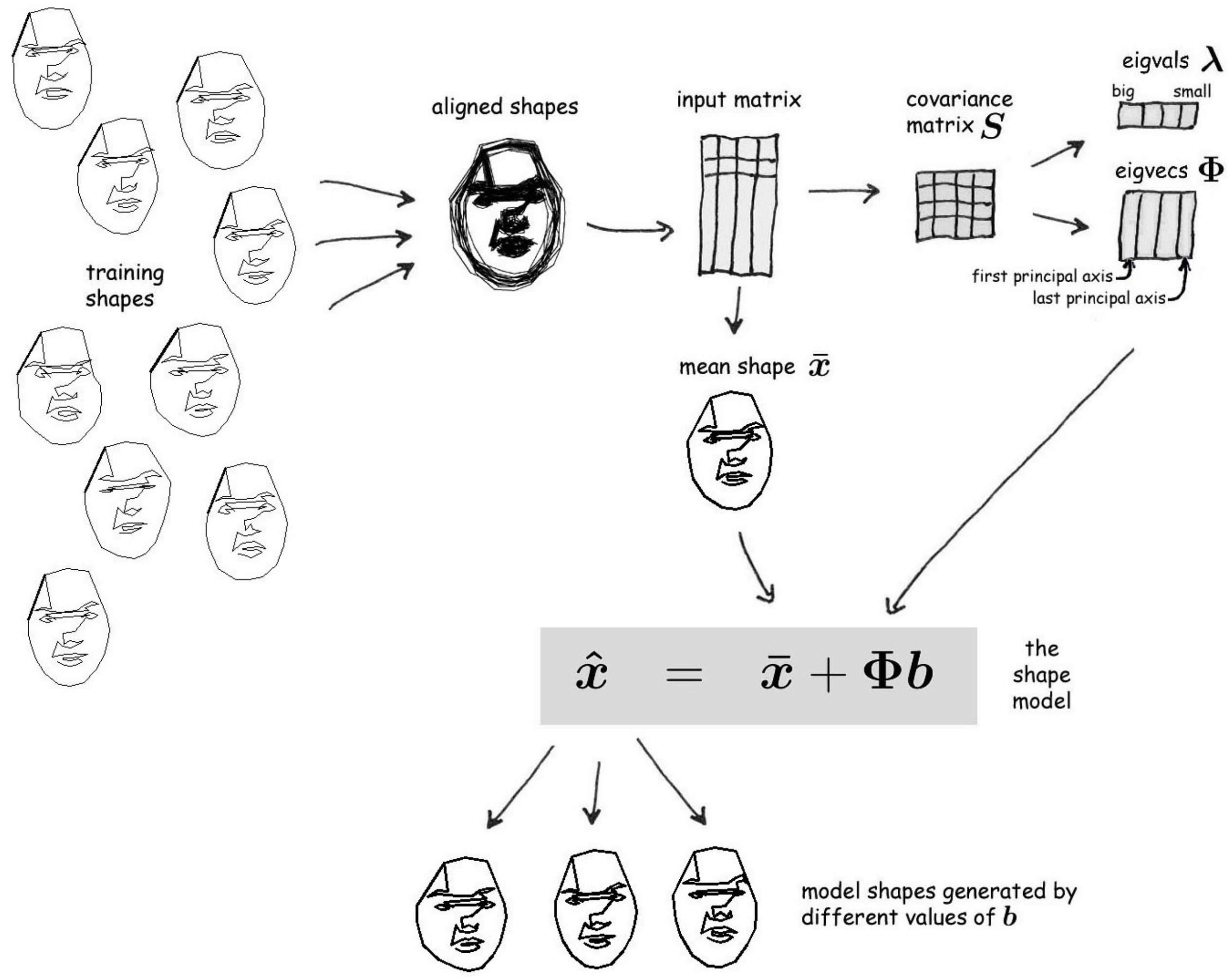

training shapes
aligned shapes
input matrix
covariance matrix $\boldsymbol{S}$
eigvals $\boldsymbol{\lambda}$
big
small
eigvecs $\boldsymbol{\Phi}$
first principal axis
last principal axis
mean shape $\bar{\boldsymbol{x}}$
$\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi b}$
the shape model
model shapes generated by different values of $\boldsymbol{b}$

Figure 3.3: *How the shape model is generated and used.*

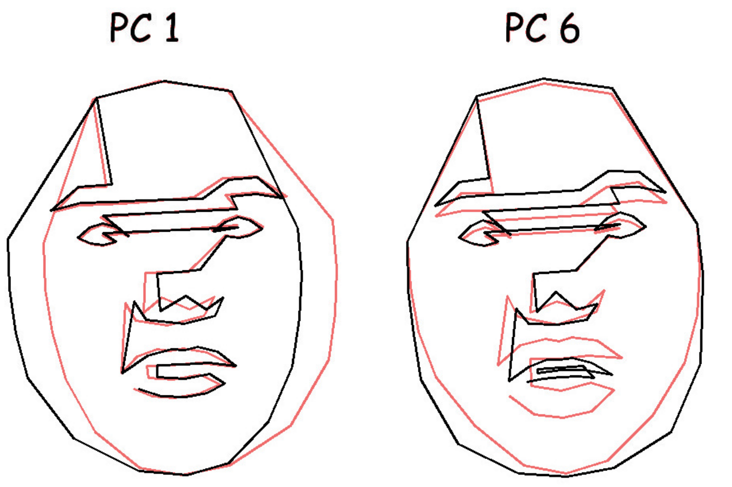


Figure 3.4: *Effect of principal components in the shape model. The figure shows the mean shape with a multiple of an eigenvector added (pink) or subtracted (black).*

PC 1 changes the size of the side of the face relative to the facial features (it mainly accounts for yaw).

PC 6 mainly affects the mouth, and to a lesser extent the jawline and the eyebrows.

3.7 Generating shapes from the shape model

As just mentioned, we can use the shape model Equation 3.5 to generate different shapes by varying the parameter $\boldsymbol{b}$. By keeping the elements of $\boldsymbol{b}$ within limits we ensure that generated faces are lifelike. The next section will consider eigenvectors in more detail, but first let's look at some simple examples.

Figure 3.4 shows faces generated by adding multiples of a single eigenvector to the mean shape. The left figure was generated using Equation 3.5 with b_1 set to $\pm 2\sqrt{\lambda_1}$ and with all other b_i's fixed at 0 (where b_1 is the first element of $\boldsymbol{b}$ and λ_1 is the largest eigenvalue).

If the distribution of the training shape points in shape space is Gaussian then so will be the distribution of b_i, and the black and pink shapes will delimit 95% of the variability along the first principal axis. This is because ± 2 standard deviations covers 95% of the area under the bell curve—the relationship between the eigenvalues and standard deviations will be reviewed in the next section.

The right figure was generated in a similar fashion from the sixth eigenvector. This eigenvector was chosen for illustration because for this set of training shapes it happens to have a fairly clear effect on the mouth. In general, the principal axes of shape models can't be interpreted so easily.

3.8 Mathematics of shape models

This section reviews some standard linear algebra and statistics necessary for a full understanding of the shape model. The next section gives an example which will help to make this mathematics more clear. Readers unfamiliar with Principal Component Analysis may want to first visit Appendix A, which gives an overview of principal components.

Eigenvectors

Because the eigenvectors are generated from the covariance matrix, they are the principal axes of the data. We can thus use the terms "eigenvector" and "principal axis" interchangeably here. Although "eigenvector" has a broader meaning, in this thesis all eigenvectors are principal axes.

By convention, the eigenvectors in the matrix $\mathbf{\Phi}$ are normalized to unit length and are ordered on their eigenvalues from left to right in the matrix. Thus the first column of $\mathbf{\Phi}$ corresponds to the largest eigenvalue of $\boldsymbol{S}_{shapes}$. This first column is the first eigenvector, or first principal axis, of the aligned and centered training shapes. The second column is the second principal axis, orthogonal in shape space to the first principal axis. (Remember that we are working in shape space, where we treat shapes as points in multidimensional space.)

Since the eigenvectors are unit length and orthogonal to each other, the eigenmatrix is an orthogonal matrix, and thus its inverse is its transpose.

The column space of the eigenmatrix is identical to that of the aligned and centered training shapes. Therefore any of the training shapes can be recreated by a linear combination of the eigenvectors. A training shape can be approximated using a linear combination of just the first few eigenvectors; as we use more eigenvectors we can approximate the shape more precisely.

Eigenvalues

A covariance matrix is symmetric and real (it has no imaginary components). Therefore it has a full set of real orthogonal eigenvectors. Furthermore a covariance matrix is at least positive-semidefinite (Section B.3.3). Therefore its eigenvalues are nonnegative.

Because the eigenmatrix is created from the covariance matrix, each eigenvalue is the variance of the data in the direction of the associated eigenvector (and therefore its square root is the standard deviation in that direction). Thus an eigenvalue is a measure of the importance of its eigenvector. Eigenvectors on the left of the matrix $\mathbf{\Phi}$ have large eigenvalues and point in directions that capture the bulk of variation in the training set. Eigenvectors on the right of the matrix have small eigenvalues and typically just represent noise in the sampled set.

3.9 Understanding the shape model

It's easier to gain an intuitive understanding of the shape model if we use rectangles instead of complicated face shapes. Figure 3.5 shows an example of such rectangles.

To specify the four points of any of the rectangles, we need eight numbers: four x- and four y-coordinates. The shape space is eight-dimensional. But the rectangles in this example are centered and symmetrical about the origin. Thus we can actually define a rectangle by just two parameters: its width and height.

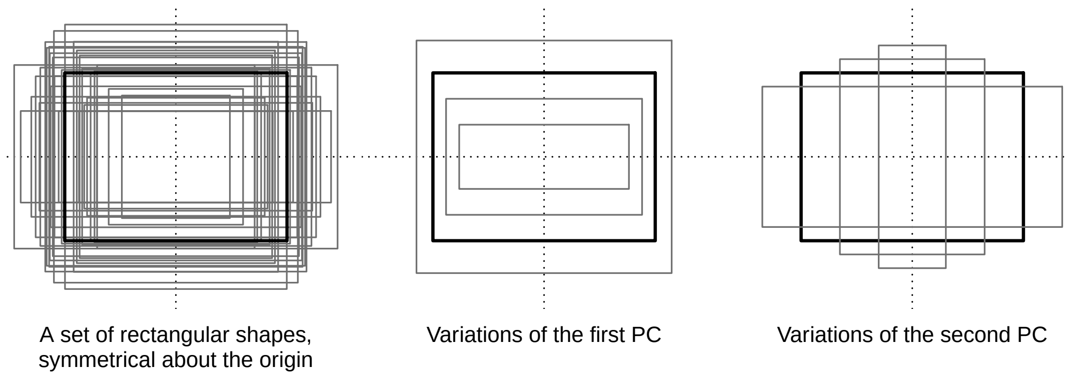

A set of rectangular shapes, symmetrical about the origin
Variations of the first PC
Variations of the second PC

Figure 3.5: *A simple four-point model.*
Left *A set of rectangular shapes, symmetrical about the origin, with their mean shape.*
Middle *Mean shape and variations by adding a multiple of the* ***first*** *eigenvector.*
Right *Variations created by adding a multiple of the* ***second*** *eigenvector.*

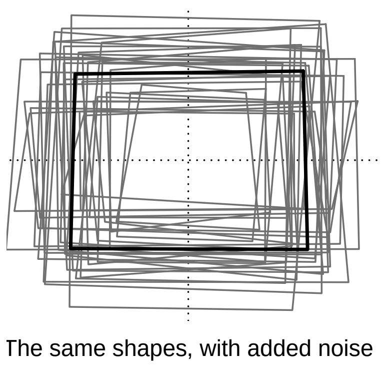


Figure 3.6: *The same set of shapes with Gaussian noise added to the points in the horizontal and vertical directions.*

This noise could for example represent uncertainty in the manually landmarked points (although exaggerated here).

Now we generate the mean shape and covariance matrix of the rectangles, and hence the shape model (Equation 3.5). The shape model $\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi b}$ becomes[1]

$$\hat{\boldsymbol{x}} = \begin{pmatrix} 6 & 6 \\ -6 & 6 \\ -6 & -6 \\ 6 & -6 \end{pmatrix} + b_1 \begin{pmatrix} 0.2 & 0.5 \\ -0.2 & 0.5 \\ -0.2 & -0.5 \\ 0.2 & -0.5 \end{pmatrix} + b_2 \begin{pmatrix} 0.5 & -0.2 \\ -0.5 & -0.2 \\ -0.5 & 0.2 \\ 0.5 & 0.2 \end{pmatrix} + \dots \qquad (3.8)$$

The sorted eigenvalues of the covariance matrix $\boldsymbol{S}_{shapes}$ are 13.6, 9.2, 0, 0, …, 0. There are eight eigenvalues altogether. The first two are important. The rest are zero (actually not exactly zero on a computer because of numerical error, but extremely small).

Thus the process has discovered that the shapes can be parameterized by just two parameters, b_1 and b_2 (as shown in the scree plot on the left of Figure 3.7). The

[1]The 8×1 vectors are rounded to one decimal place and reshaped into 4×2 matrices for clarity. The left column of each matrix shows x values; the right column shows y values.

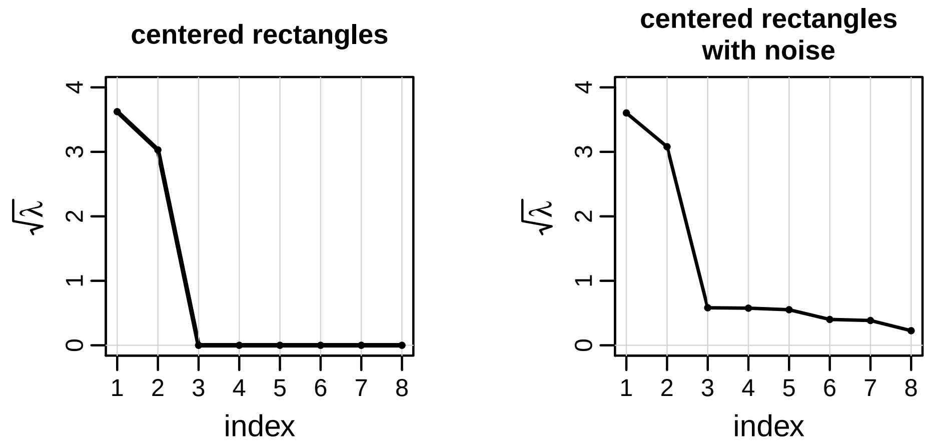


Figure 3.7: *Scree plots for the rectangle shapes.*

Scree plots show the ordered eigenvalues. An eigenvalue is the variance of the corresponding principal component. In the figures I have chosen to plot the square root of the eigenvalues to show standard deviations rather than variances, to emphasize the nonzero eigenvalues in the graph on the right.

Left ***The centered rectangles*** *(illustrated in Figure 3.5).*
All eigenvalues except the first two are zero. The first two principal components account for all variation in the shapes. The shapes can be specified with just two parameters.

Right ***The noisy centered rectangles*** *(illustrated in Figure 3.6).*
The spurious variability is captured by the 3rd and higher principal components.
We need more than two parameters to precisely specify the shapes, although two parameters suffice to specify their overall structure.

shapes exist in a two-dimensional subspace of the eight-dimensional shape space. The first parameter varies the contribution of the first eigenvector and, in this example, chiefly changes the overall size of the generated shape. The second parameter varies the contribution of the second eigenvector, which mainly adjusts the aspect ratio of the shape. (I artificially generated the training rectangles to have this simple overall-size and aspect-ratio interpretation. For an arbitrary set of centered rectangles, the eigenvectors generally won't have such a simple interpretation.)

What is the advantage of this shape model over the "obvious" model where the two parameters simply specify the width and height? The advantage is that the first principal axis, of all possible axes, captures as much variation as possible in this sample of shapes. If you had to specify a shape using just a single parameter, you would on the whole get closer to your desired shape by adding a multiple of the first principal axis to the mean shape, rather than by changing just the width or just the height. The second principal axis captures as much of the remaining variation as possible, with the constraint that the principal axes are orthogonal. Since in this example only two eigenvalues are non-zero, the second principal axis in fact captures all the remaining variation.

Now let's add some noise to the rectangle coordinates (Figure 3.6), and re-generate the shape model. We discover that the remaining eigenvalues are no longer zero, as can be seen on the right side of Figure 3.7. Linear combinations of the first two eigenvectors no longer suffice to generate the training shapes. Nevertheless the first two eigenvalues clearly account for most of the variance, and we can approximate the training shapes with just two eigenvectors.

For realistic face data there will be many eigenvalues and no abrupt cutoff point (as can be seen in the scree plot in Figure 3.8). We can capture as much variation of the input shapes as we want by retaining the appropriate number of eigenvectors. But what is the appropriate number? This brings us to the next section.

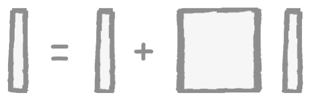
3.10 Constraining the shape model

In the shape model formula (Equation 3.5), generated shapes are determined by setting values of the vector $\boldsymbol{b}$. To ensure that the generated faces are lifelike, we must limit the number of eigenvectors and constrain the allowed values of $\boldsymbol{b}$.

3.10.1 Limiting the number of eigenvectors

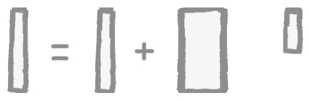
We want to retain what is essential in our set of training shapes, and so want to ignore sampling quirks, very unusual facial features, and manual landmarking mistakes and uncertainty. As is standard practice when using principal components, we therefore retain only the subset of the principal components corresponding to larger eigenvalues. That is, we keep only the leftmost eigenvectors in the eigenmatrix $\boldsymbol{\Phi}$. (As always, we assume the columns of $\boldsymbol{\Phi}$ are ordered on the eigenvalues, so the most important eigenvectors are on the left.) Pictorially,

$$\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \quad \boldsymbol{\Phi} \quad \boldsymbol{b}$$



(3.9)

becomes



(3.10)

The set of eigenvectors defines the set of directions in shape space along which the shapes can vary. By truncating the eigenmatrix we limit the model to the underlying intrinsic dimensionality of the population of shapes. (Or at least try to, since we can never be entirely sure of the properties of the population from the training set, which is just a sample of the population).

How do we determine the number of eigenvectors to retain? A textbook approach is to look for a knee in the scree plot, which would indicate a natural cutoff point. But scree diagrams for realistic shape models exhibit no such knee (Figure 3.8). Alternatively

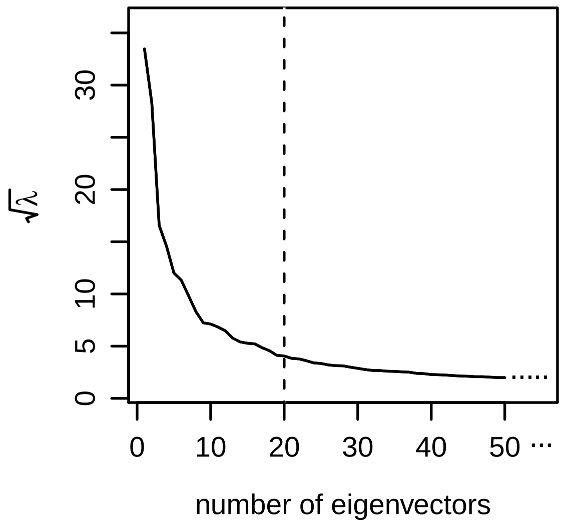


Figure 3.8: *Scree plot for the Stasm frontal model.*

Only 50 of the 154 eigenvalues are shown. (There are $77 \times 2 = 154$ *values because the shapes have 77 points with an x and y coordinate for each point.)*

The vertical dashed line shows the number of eigenvectors selected by empirical testing.

we could choose to explain a reasonable percentage of the variance in the training set, say 95%, and select the number of eigenvalues necessary to explain that proportion of variance.

But our end goal really is to accurately locate face landmarks. The amount of variance explained is just a means to that end. Thus we determine the number of eigenvectors empirically, that is, by choosing the number of eigenvectors that gives the best landmark accuracies on a parameter-selection set of faces. When tuning the Stasm frontal model, this approach selected 20 eigenvectors, which happen to explain 91% of the training set variance. The precise number of eigenvectors is in fact not critical in this application.

3.10.2 Constraining the shape parameter $\boldsymbol{b}$

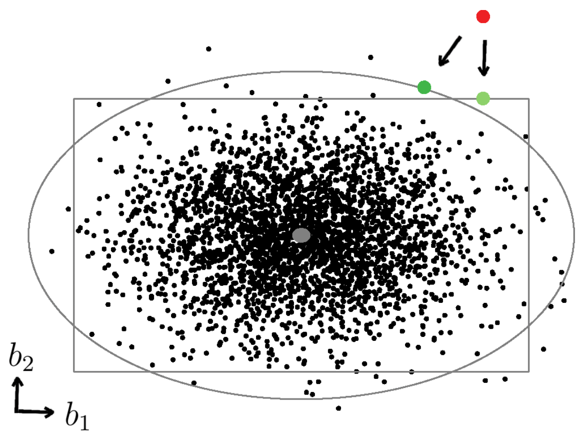
To ensure that the model faces are realistic, we must limit the range of values of the shape model parameter $\boldsymbol{b}$. For example, if we allowed an excessively large b_6 in the face on the right of Figure 3.4, the generated shape would have an unrealistically large mouth. Given an arbitrary $\boldsymbol{b}$, an easy way of applying limits is to clamp each element b_i of $\boldsymbol{b}$ independently. We clamp each b_i to a predetermined multiple m of its standard deviation

$$|b_i| \; < \; m\,\sigma_i \;, \tag{3.11}$$

where m is a clamping constant determined during training, and σ_i is the standard deviation of b_i measured on the training shapes. Since an eigenvalue λ_i is the variance of the corresponding principal component b_i, this is equivalent to

$$|b_i| \; < \; m\,\sqrt{\lambda_i} \;. \tag{3.12}$$

If m is 2, for example, then ± 2 standard deviations of b_i in the training data will be within the clamped b_i. So if b_i is Gaussian, then 95% of the values in the training data will less than the clamped b_i. But our end goal really is to accurately locate face landmarks and the allowed range of $\boldsymbol{b}$ is just a means to that end. Thus in our software m is chosen empirically on a parameter-selection set.



The black points were drawn from a two-dimensional normal distribution.

By Equation 3.12, points outside the gray rectangle are considered outliers.

By Equation 3.13, points outside the gray ellipse are considered outliers.

In this example, the rectangle and ellipse were generated with $m = 3$.

Figure 3.9: *Constraining **b**.*

The red point is an outlier. By clamping each dimension independently (Equation 3.12), we clamp the red point to the light green point on the gray rectangle. For this outlying point it so happens that we needed to clamp only in the vertical direction to bring the point onto the rectangle. The clamped point is still outside the ellipse.

More correctly, to match the ellipsoidal equi-probability contours of this Gaussian distribution, we should "shrink" the red point towards the center of the distribution i.e., towards to the nearest point on the ellipse as shown by the darker green point.

We can show only two dimensions on a printed page, but actually in a shape model the number of dimensions would be the selected number of eigenvectors (Section 3.10.1).

It should be mentioned that this simple clamping technique, where we clamp each b_i independently, theoretically isn't optimal. The distribution of shape coordinates in multidimensional space is approximately ellipsoidal (or at least more ellipsoidal than rectangular). Thus, as illustrated in Figure 3.9, $\boldsymbol{b}$ should be constrained to an ellipsoid

$$\frac{1}{n_{eigs}} \sum_{i=1}^{n_{eigs}} \frac{b_i^2}{\lambda_i} \;<\; m^2, \tag{3.13}$$

where n_{eigs} is the selected number of eigenvectors (Section 3.10.1) and once again m is a constant determined during training[2]. If m is 2, for example, the above inequality constrains the generated shape to within 2 standard deviations of the mean shape. To force the above inequality, we multiply $\boldsymbol{b}$ by a constant less than 1, shrinking it towards zero, and thereby shrinking the generated shape towards the center of the distribution.

Clamping each b_i independently would be optimal if the distribution were rectangular, not ellipsoidal. However, in practice simple clamping seems to work just as well as constraining to an ellipsoid ([75] Section 5.4.3).

[2]We mention that the sum term on the left is the squared Mahalanobis distance of $\boldsymbol{b}$ from the center $\mathbf{0}$ of the distribution. Since the principal components are independent there are no covariance terms. The squared Mahalanobis distance is thus $\sum b_i^2/\lambda_i = \boldsymbol{b}^T \boldsymbol{\Lambda} \boldsymbol{b}$ where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, ..., \lambda_t)$.

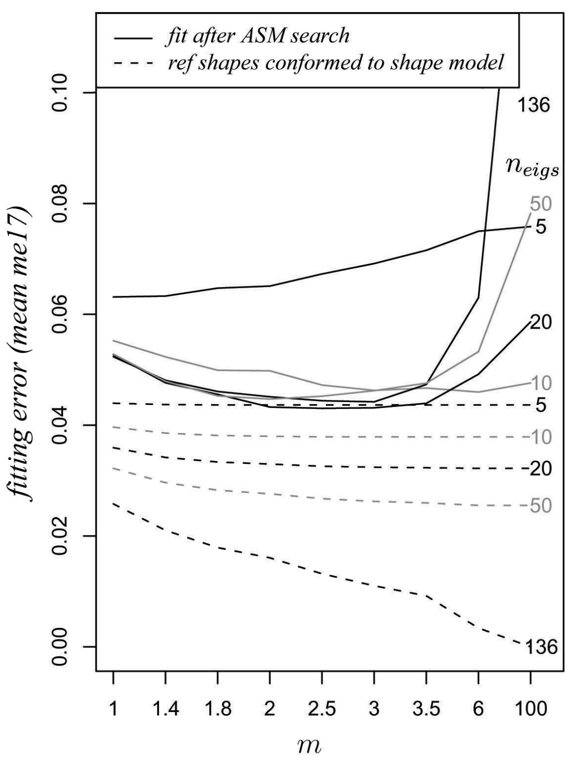


Figure 3.10: *Mean landmark fitting error for various values of m (the clamping constant, Equation 3.12) and n_{eigs} (the number of retained eigenvectors).*

The solid curves show the fit for landmarks positioned by an ASM over a sample of faces.

The dashed curves show how well the shape model acting alone can recreate the training shapes. It can't represent the training shapes well if m is too low or if an insufficient number of eigenvectors is retained for the model.

Gray and black are used to distinguish curves and have no special meaning. This graph shows error expressed as the mean `me17` *(Section 6.2.1). The graph is for a 68 point model in an old version of Stasm, and is based on Figure 5.3 in [75].*

3.10.3 Tuning the shape model parameters

Figure 3.10 shows how the landmark fitting error typically varies with the clamping constant m and the number of retained eigenvectors.

Shape constraints impose a lower error bound for automatic landmarking, because the constraints imposed by the shape model will usually adjust landmarks if we conform even a training shape to the model. We get close to the ideal of being able to exactly regenerate any training shape if we use a large m and a large set of eigenvectors (the dashed curve at the bottom right of the plot). But using such a flexible shape model won't optimally correct implausible suggested shapes during the search, and so doesn't give the best results when searching (as can be seen by the upward slopes of the solid curves on the right of the plot). Thus a smaller m and reduced set of eigenvectors works better, even though it doesn't allow us to exactly reproduce the training shapes.

3.11 Conforming a given shape to the shape model

We have examined in some detail how the shape model can generate shapes. Now we look at how we can adjust an arbitrary shape to conform to the shape model. Given an arbitrary suggested shape $\boldsymbol{x}$, we want to calculate the parameter $\boldsymbol{b}$ that allows Equation 3.5 to best approximate $\boldsymbol{x}$ with a model shape $\hat{\boldsymbol{x}}$.

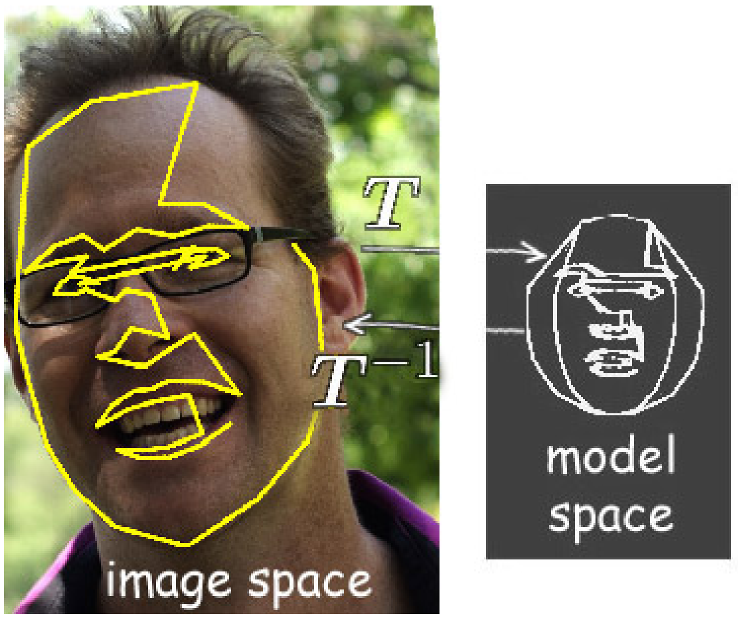


Figure 3.11: *Image space and model space.*

The face can be anywhere in the image. To use the shape model, we first must transform the shape from the image to the model space.

When we have finished manipulating the shape in model space, we invert the transform to position the shape back on the image.

3.11.1 Image space and model space

First a few remarks on image space and model space (Figure 3.11). The face we are interested can be anywhere in the image. The shape points have x,y pixel coordinates in *image space*, whereas a shape $\hat{\boldsymbol{x}}$ generated by the model has x,y coordinates in *model space*. The center and scale of the model space is determined during training by the set of training shapes, the position of the reference shape, and so on. (In both spaces we typically work with the shape as a $2n$-vector. Image space and model space can thus be considered as two different forms of shape space.)

To use the model, we must transform the face shape from the image space to the model space. The transform will depend on the position, orientation, and scale of the image shape. Once we have used the shape model, we apply the inverse transform to place the shape back on the image.

3.11.2 Conforming a given shape to the shape model

To approximate a suggested shape with a model shape, we seek the $\boldsymbol{T}$ and $\boldsymbol{b}$ that minimize the distance

$$|\boldsymbol{T}(\boldsymbol{x}_{suggested}),\ \hat{\boldsymbol{x}}| \tag{3.14}$$

$$= |\boldsymbol{T}(\boldsymbol{x}_{suggested}),\ \bar{\boldsymbol{x}} + \boldsymbol{\Phi}\boldsymbol{b}|\ , \tag{3.15}$$

where $\boldsymbol{T}$ is a similarity transform that maps from the image to the model space, $\boldsymbol{x}_{suggested}$ is the given suggested image shape, and $\hat{\boldsymbol{x}}$ is a model shape.

Figure 3.12 on the following page gives an iterative algorithm to do this. It maps the suggested shape to the model space, corrects wayward points by constraining $\boldsymbol{b}$, then maps the corrected model shape back to the image. The algorithm follows Section 4.8 in Cootes and Taylor [23]. Note that $\boldsymbol{T}$ in the notation here is $\boldsymbol{T}^{-1}$ in their notation.

```
input the suggested image shape $\boldsymbol{x}_{suggested}$

Zero the model parameters $\boldsymbol{b}$

repeat

    1. Generate a model shape $\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi b}$

    2. Calculate the $\boldsymbol{T}$ that maps the image shape $\boldsymbol{x}_{suggested}$ to the model space
       This is the transform $\boldsymbol{T}$ that minimizes $|\boldsymbol{T}(\boldsymbol{x}_{suggested}), \hat{\boldsymbol{x}}|$ as described in Section 3.4

    3. Transform the image shape to the model space: $\boldsymbol{x} = \boldsymbol{T}(\boldsymbol{x}_{suggested})$

    4. Update the model parameters $\boldsymbol{b} = \boldsymbol{\Phi}^T(\boldsymbol{x} - \bar{\boldsymbol{x}})$

    5. Constrain $\boldsymbol{b}$ as described in Section 3.10.2

until convergence ($\boldsymbol{T}$ and $\boldsymbol{b}$ are stable)

Transform the model shape back to the image space: $\boldsymbol{x}_{suggested} = \boldsymbol{T}^{-1}(\hat{\boldsymbol{x}})$

output new version of the image shape $\boldsymbol{x}_{suggested}$, conformed to the shape model
```

Figure 3.12: *Conforming an image shape to the shape model (iterative version). The shape $\boldsymbol{x}_{suggested}$ is in the image space, all other shapes are in the model space.*

```
input (i)  the suggested image shape $\boldsymbol{x}_{suggested}$
      (ii) the shape model parameters $\boldsymbol{b}$ from the previous ASM iteration

1. Generate a model shape $\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi b}$

2. Calculate the $\boldsymbol{T}$ that maps the image shape $\boldsymbol{x}_{suggested}$ to the model space
   This is the transform $\boldsymbol{T}$ that minimizes $|\boldsymbol{T}(\boldsymbol{x}_{suggested}), \hat{\boldsymbol{x}}|$ as described in Section 3.4

3. Transform the image shape to the model space: $\boldsymbol{x} = \boldsymbol{T}(\boldsymbol{x}_{suggested})$

4. Update the model parameters $\boldsymbol{b} = \boldsymbol{\Phi}^T(\boldsymbol{x} - \bar{\boldsymbol{x}})$

5. Constrain $\boldsymbol{b}$ as described in Section 3.10.2

6. Using the constrained $\boldsymbol{b}$ generate the conformed model shape $\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi b}$

7. Transform the model shape back to the image space: $\boldsymbol{x}_{suggested} = \boldsymbol{T}^{-1}(\hat{\boldsymbol{x}})$

output (i)  new version of the image shape $\boldsymbol{x}_{suggested}$, conformed to the shape model
       (ii) the shape model parameters $\boldsymbol{b}$, for use next time
```

Figure 3.13: *Conforming an image shape to the shape model (non-iterative version).*

```
input (i)  the suggested image shape $\boldsymbol{x}_{suggested}$
      (ii) the list of pinned point positions

1. Allow the descriptor models to suggest a shape as usual,
   but leave pinned points in place.

2. repeat

3.        Conform the shape to the shape model as usual (Figure 3.13).
          In general this will move the pinned points.

4.        $dist$ = mean distance between the points and their pinned positions

5         Force pinned points back to their pinned positions.

6. until convergence ($dist$ is less than half a pixel)

output conformed version of the image shape, with points pinned
```

Figure 3.14: *Conforming a shape with pinned points.*
This algorithm replaces Steps 4 and 5 in Figure 2.4.

The algorithm can be simplified slightly. In practice, we don't need to iterate the shape model, because iteration is built into the ASM search (Figure 2.4 on page 19). Instead of initializing the shape model parameter $\boldsymbol{b}$ to zero, we use its value from the previous iteration of the ASM (after setting it to zero the very first time). The iterative algorithm in Figure 3.12 simplifies to the non-iterative algorithm in Figure 3.13. This is the algorithm used in our software and empirically gives results as good as the iterative algorithm.

3.12 Pinned points and interactive landmark location

Sometimes we want to keep the positions of certain points fixed. Such *pinned* points shouldn't be moved by the ASM search. For example, in an interactive application the user may manually position some landmarks, and possibly iteratively interact with the automatic landmarker to correct points.

The body of the ASM search algorithm (Figure 2.4) can be modified to handle pinned points as shown in Figure 3.14.

3.13 Missing points

Sometimes a landmark is obscured in a training shape, perhaps by the side of the face or the nose (Figure 3.15). Minor complications caused by missing points ripple out to several places during model training. These complications are handled as described

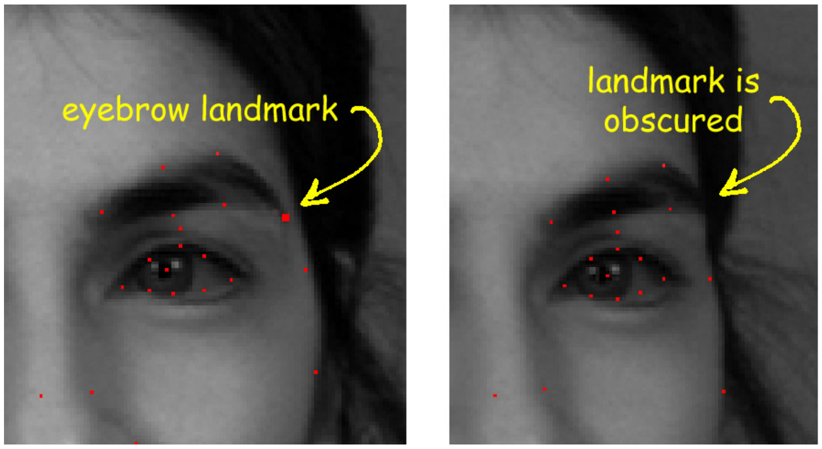


Figure 3.15: *Obscured landmarks.*

As the subject turns her head slightly, the landmark on the edge of the eyebrow is obscured by her face.

below. Readers uninterested in the mundane details of missing points can skip directly to Section 3.15

In our software, a missing point is indicated with coordinates (0,0). During processing, it's sometimes possible for the coordinates of a valid point to become zero (for example, when manipulating the shape with the shape model). Such a point shouldn't be treated as missing in further processing, so the software automatically jitters the point's x coordinate to 0.1.

To avoid unnecessary complications, the reference shape must have all points.

Missing points are also useful in other situations. For example we can conveniently pass around a set of pinned points (Section 3.12) as a shape with all points zero except the pinned points.

3.13.1 Ignoring missing points during training

When aligning two shapes (Section 3.4), missing points must be ignored when calculating the transform $\boldsymbol{T}$. This comes into play when aligning the set of shapes during training (Step 4 in Figure 3.1).

Missing points must be ignored when taking the mean of the points to calculate the mean shape (Step 5 in Figure 3.1)

Missing points must be ignored when calculating the covariance matrix of the aligned training shapes (Equation 3.7). So when calculating the covariance between two coordinates we use only complete pairs and divide by the number of complete pairs. Strictly speaking the resulting covariance matrix is only an *approximate* covariance matrix [48], because the number of observations isn't the same for all variables. An approximate covariance matrix may have some negative eigenvalues—but this isn't a concern when training frontal models, because in practice only a small fraction (less than ten percent) of the eigenvalues are negative, and then only slightly so, and besides we retain only the most important eigenvectors for the shape model.

See also Section 5.5, which discusses how missing points affect the descriptor models (as opposed to the shape models we are focusing on here).

3.13.2 Missing points during the search

We don't need it for applications in this thesis, but for completeness we mention that in applications that must conform a shape with missing points to the shape model, an extra step must be added to the algorithm in Figure 3.13. Before updating the model parameters $\boldsymbol{b} = \boldsymbol{\Phi}^{-1}(\boldsymbol{x} - \bar{\boldsymbol{x}})$ in Step 3, missing coordinates in $\boldsymbol{x}$ should be estimated. This can be done most simply by replacing them with points from the mean shape.

3.14 Imputing missing points for training three-quarter views

Non-frontal faces tend to have significant numbers of missing points. Chapter 9 will discuss models for non-frontal faces in detail. Here we discuss only how missing points affect training of non-frontal models. Note that we are talking here about shape models specialized to a limited set of face poses—for example a model specialized for right three-quarter views.

Figure 3.16 shows the terms used for face poses and some of their synonyms. Our main concern is with yaw. In-plane rotation (not illustrated) refers to rotations aligned to the image surface. For frontal faces, in-plane rotation and roll are the same. These terms will become more important in Chapter 9 which explores multiview models.

3.14.1 Frontal faces

In the training data for *frontal* faces, typically all or nearly all landmarks are visible. (Why not all? To allow the model to cope with faces that aren't strictly-frontal, we include some slightly yawed training faces, like the face in Figure 3.15, thus some landmarks may be obscured.)

In the subset of MUCT faces used for training the Stasm frontal model (the details are in Section 8.9 on page 107), the outer eyebrow landmarks (illustrated in Figure 3.15) are the most obscured of all points, but even they are missing in less than 2% of the faces. About 1% of some nose landmarks are also missing. We can simply ignore these points as described in Section 3.13.1.

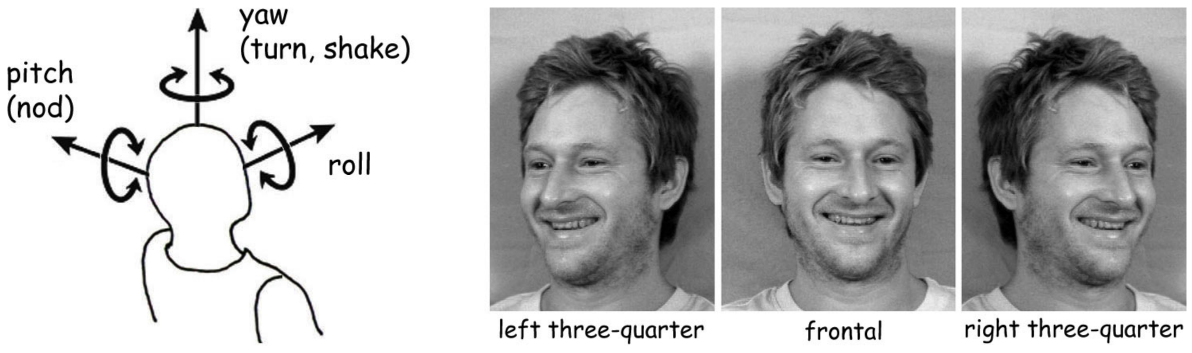


Figure 3.16: *Pose nomenclature.*

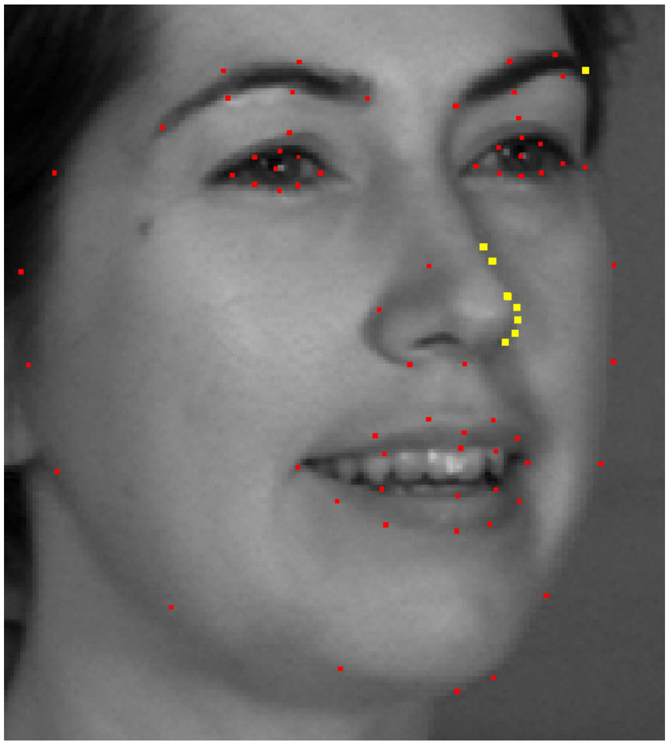


Figure 3.17: *Obscured landmarks.*

In this non-frontal view, the yellow landmarks are obscured by the nose or the side of the face.

In the convention used by the MUCT data, the landmarks along the right jaw aren't obscured, because by definition they are on the visible edge of the jaw. On the other hand, the landmarks on the nose and eyebrow are defined to be on a specific point on the face, which is obscured in this view.

3.14.2 Three-quarter faces

However, when building models for *three-quarter* views, a relatively large percentage of faces will have missing points (Figure 3.17). In the MUCT three-quarter faces for example, 81% of the faces are missing the outer-eyebrow landmark. Thus we need to handle training sets that have a large number of obscured points.

There are at least two potential ways of circumventing this issue:

(i) We can completely ignore faces with any missing points. But this seriously depletes the training set.

(ii) We can redefine the shapes for use in three-quarter faces by defining a smaller set of points, and build a model with shapes of say 60 points instead of 77 points. But in the larger context of the software that is using the ASM, it's usually more convenient to have the same set of points for all the faces. (For full side-views this becomes infeasible, but here we are concerned with frontal and three-quarter views.)

Instead of the above techniques, our software augments the set of training shapes and imputes missing points, as will now be described.

3.14.3 Augmenting the set of training shapes

We augment the central set of yawed training faces with faces that are slightly less yawed. This gives a moderate increase in the range of poses supported by the model, but more importantly it provides substantially more training data for the descriptor models.

This matters because for non-frontal faces we typically have much smaller training sets. (This is because we can't double the size of the training set by using mirrored faces, as we do with frontal sets. Also, as a practical matter if we are collecting high-resolution training faces from the internet it's much easier finding frontal faces than significantly yawed faces.)

3.14.4 Imputing points for training the model

Using the augmented set of shapes, we impute the missing points. We do this as follows. First we build an interim shape model, ignoring the missing points (as described in Section 3.13.1). This yields an interim mean shape.

Then for each shape with missing points, we impute the missing points. This is done by aligning the interim mean shape to the shape, then replacing missing points in the shape with points from the aligned mean shape. (Stasm version 4.1 imputed missing points by predicting their position using the interim shape model, but the additional complexity of that approach isn't needed.) No shape in this new training set has missing points, and we use this new set instead of the original set to build the final shape and descriptor models.

This approach seems to work well in practice. Other approaches are certainly possible. With larger training sets, such trickery may be unnecessary. We haven't done rigorous tests to determine the optimality of this approach. In particular, since all points are imputed before building the final model, imputed points are given the same status as other points in the final model. Instead, ASM search results may be better if during the search we ignored points that were missing in a high percentage of training shapes, and merely at the end of the search impute their positions using the shape model (as described in Section 3.13.2).

3.15 Points obscured by glasses and hair

Points can be obscured in other ways, for example by eye glasses or facial hair. When manually landmarking the MUCT faces, the human landmarkers estimated the position of such points—they aren't missing in the sense we have been using the word, because they aren't obscured by parts of the face. (Other ways of handling such points may also make sense.)

When training the shape model these obscured points are treated like any other point. However they are ignored when training the descriptor models (Chapter 5). This is because even though their positions are correct, or close to correct, the image texture at the point is essentially random (since for example the glasses may reflect anything).

3.16 Distributions in shape space

The shape model (Equation 3.5) assumes that the variation in each direction in shape space (after aligning the shapes) clusters around a single center (the mean shape), is symmetrical, and decreases monotonically from the center. Thus this variance can be adequately constrained simply by limiting $\boldsymbol{b}$. Therefore it's worthwhile examining normality (Gaussianity) of the shape coordinates. Normality is sufficient (although not necessary) for these assumptions, and under normality the shape model adequately describes shape variation.

Facial features are determined by a host of factors, mostly genetic and age-related. Thus by the Central Limit Theorem (and assuming additivity), we should expect approximate normality in the position of facial features in aligned faces photographed under similar conditions. However, variations in lighting, pose, and expression change the positions in ways that make application of the Central Limit Theorem less appropriate.

The Stasm training set for the frontal model has 6006 training shapes with 77 points per shape (Section 8.9 "Training the HAT ASM"). Thus we have $2 \times 77 = 154$ coordinates per shape. The 6006 cases are sparse in this high-dimensional space, as would be any feasible sample of shapes. It isn't possible to infer with reasonable confidence that a sparse multivariate sample like this is or isn't drawn from a normal distribution (e.g. Hastie et al. [45] Section 2.5). It's nevertheless worthwhile plotting the distribution of the points to look for obvious anomalies.

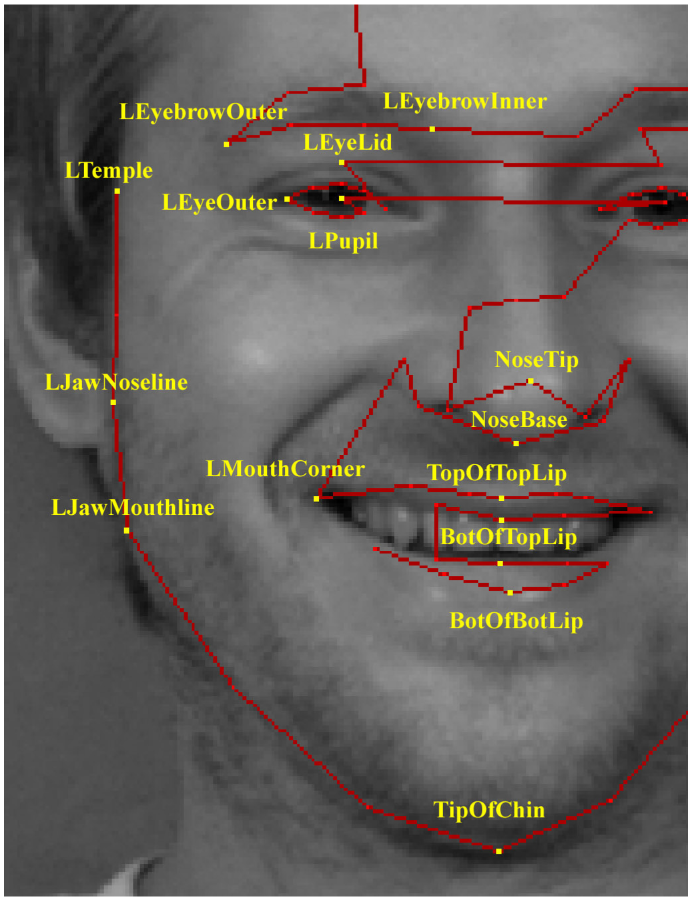


Figure 3.18: *Landmarks shown in Figures 3.19 and 3.20.*

Univariate distributions

We look first at the univariate distributions of some selected point coordinates in Figure 3.19. (Univariate normality is necessary but not sufficient for multivariate normality.) The points selected are shown in Figure 3.18. A limited set of points was chosen so the plots aren't too small on the printed page, whilst still being representative of all points.

Figure 3.19 shows that the marginal densities of the points appear approximately normal. We do see some rather odd shaped bells, but we see these also in the top row which shows reference samples drawn from a true normal distribution.

Joint distributions

What do the joint distributions look like? We can't plot a 154 dimensional cloud, but Figure 3.20 shows a scatter plots of some pairs of mouth coordinates. (Once again, bivariate normality is necessary but not sufficient for multivariate normality.)

From the figure we can see that some point clouds aren't symmetrical and ellipsoidal. An example is the subplot shaded gray. (There is nothing especially unique about this subplot; it's just a representative example. It plots `TopOfTopLipY` versus `BotOfBotLipY`, i.e., the y coordinates of the center of the top of the top lip and the bottom of the bottom lip, illustrated in Figure 3.18.) We expect the vertical position of mouth points to be correlated (since the points move together with the position of the mouth), and in the gray subplot we indeed see the correlation. But we also see quite a few outlying points on the lower left of the subplot, possibly because some mouths are open and some closed.

For visual calibration of the reader's eye, Figure 3.21 shows scatter plots of a sample of 1000 points draw from a multivariate normal distribution with the same covariance matrix (i.e, the empirical covariance matrix of the points in Figure 3.20). Note how the distributions of these true Gaussian points tend to be more symmetrical than in the previous graph.

Principal component distributions

Figure 3.22 shows scatter plots for the first eight principal components of the shapes. And once again for visual calibration of the reader's eye, Figure 3.23 shows scatter plots of a sample of 1000 points draw from a multivariate normal distribution with the same covariance matrix. Since these are principal components the correlations are zero.

There are a few anomalies in Figure 3.22. For example, the cloud in far right plot in the top row doesn't appear elliptical (an approximate ">" shape seems apparent).

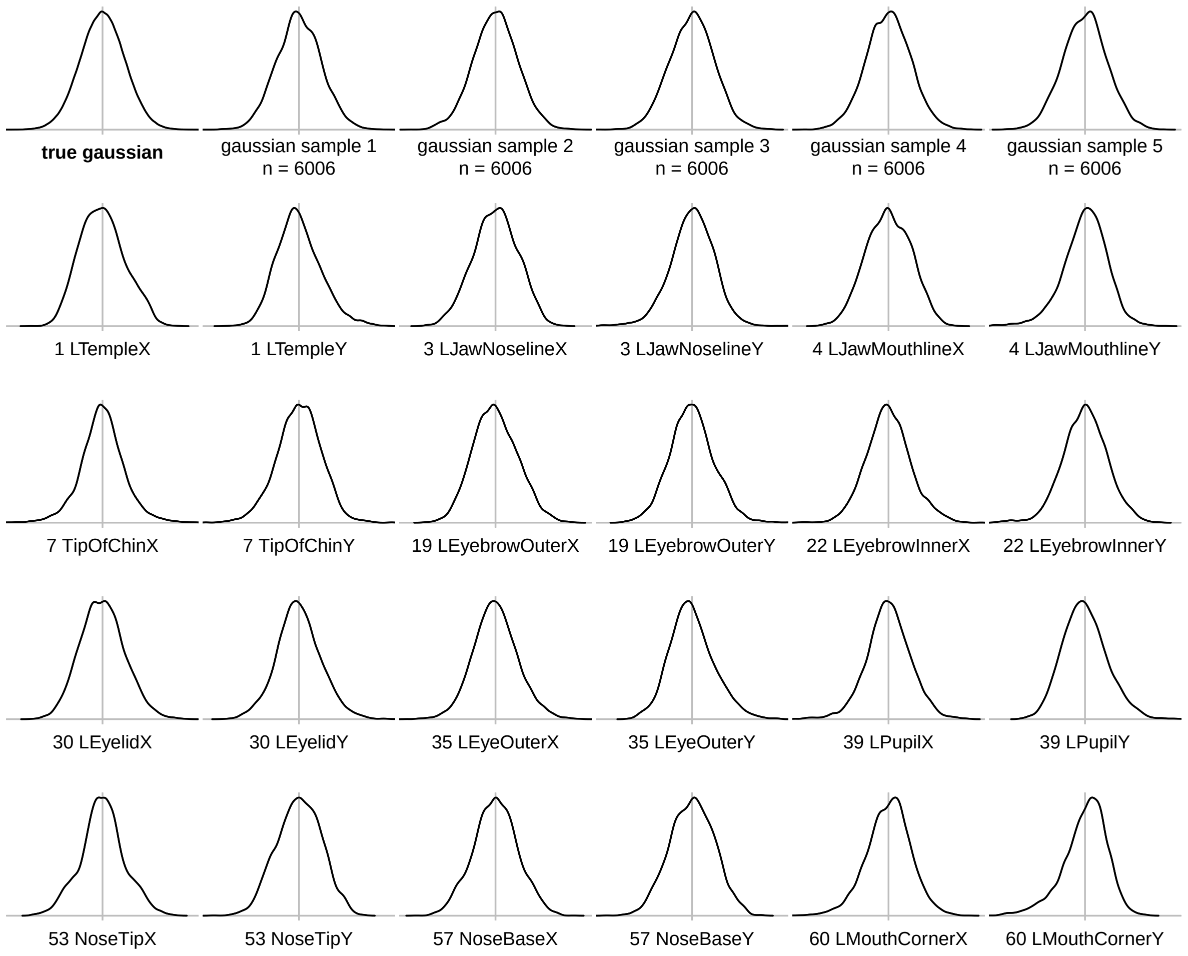


Figure 3.19: *Marginal densities of some landmarks of the 6006 aligned training shapes used for the Stasm frontal model.*

For visual calibration of the reader's eye, the first row shows a Gaussian bell curve, followed by densities of some samples drawn from a Gaussian distribution. A sample of 6006 observations was drawn, the same sample size as the rest of the plots. Note the dents in the bell in some of these samples, even though they are from a true Gaussian distribution.

The following rows show the densities of landmark positions from the aligned training shapes used in the Stasm shape model.

In all plots the width of the plot is ±3 standard deviations. Figure 3.18 shows the positions of these landmarks on a face. The suffixes X and Y indicate the x or y coordinate. Densities were estimated using the default settings of the R `density` *function* *`https://stat.ethz.ch/R-manual/R-devel/library/stats/html/density.html`.*

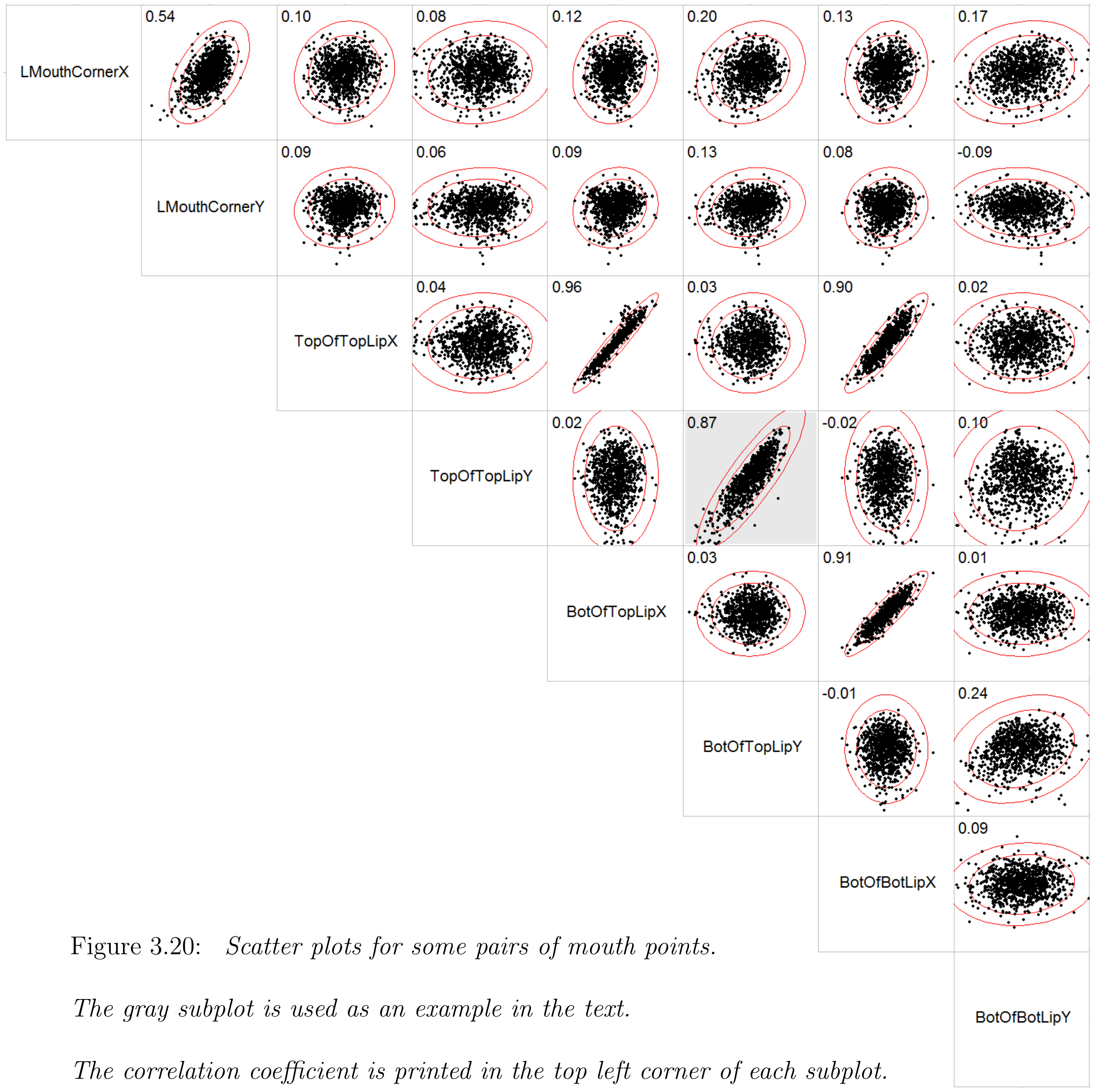


Figure 3.20: *Scatter plots for some pairs of mouth points.*

The gray subplot is used as an example in the text.

The correlation coefficient is printed in the top left corner of each subplot.

The red ellipses are at 2 and 3 standard deviations, assuming normality.

To limit overplotting, only 1000 of the 6006 Stasm training shapes are shown (although all shapes were used to calculate the printed correlation coefficients and ellipses).

Figure 3.18 shows the position of these landmarks on the face.

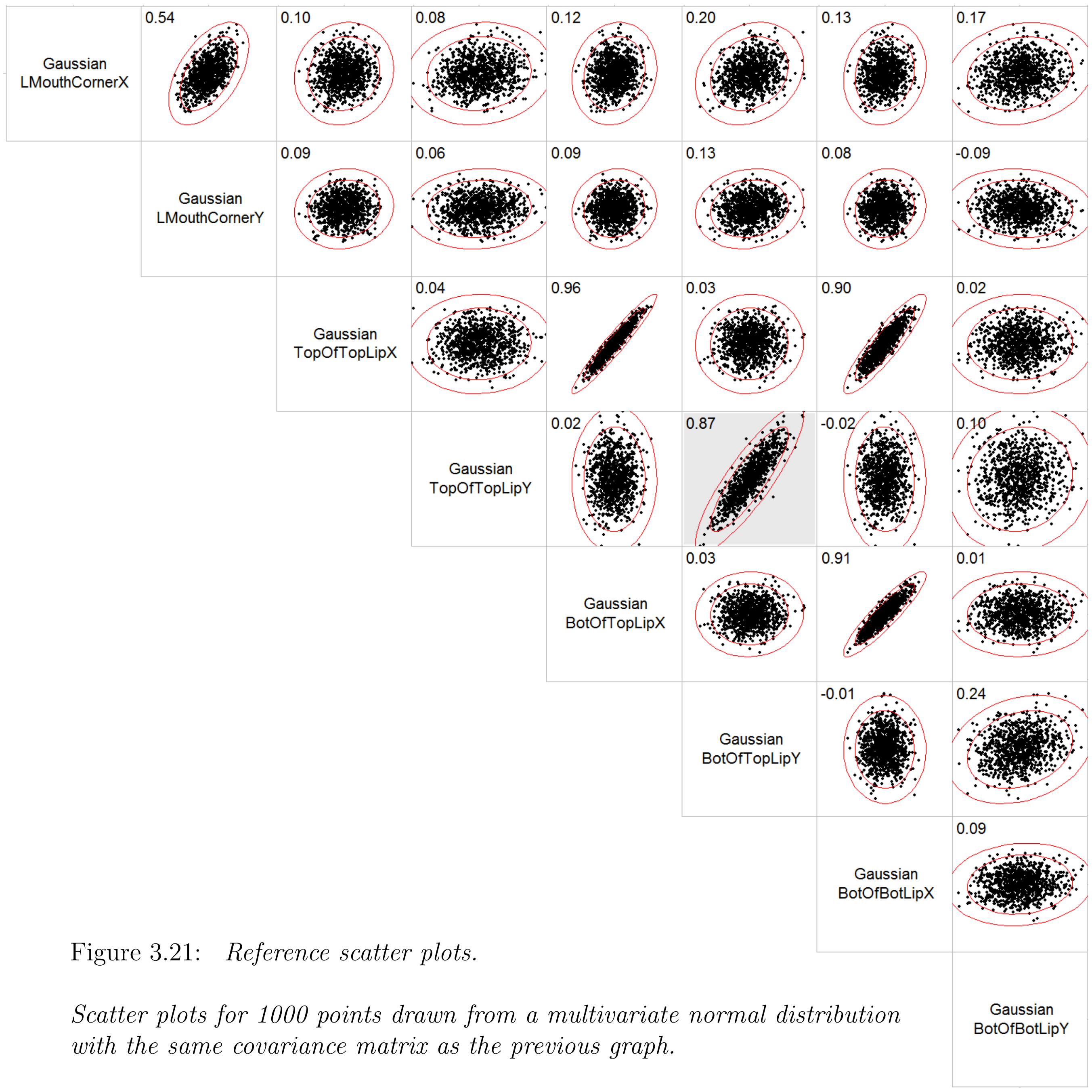


Figure 3.21: *Reference scatter plots.*

Scatter plots for 1000 points drawn from a multivariate normal distribution with the same covariance matrix as the previous graph.

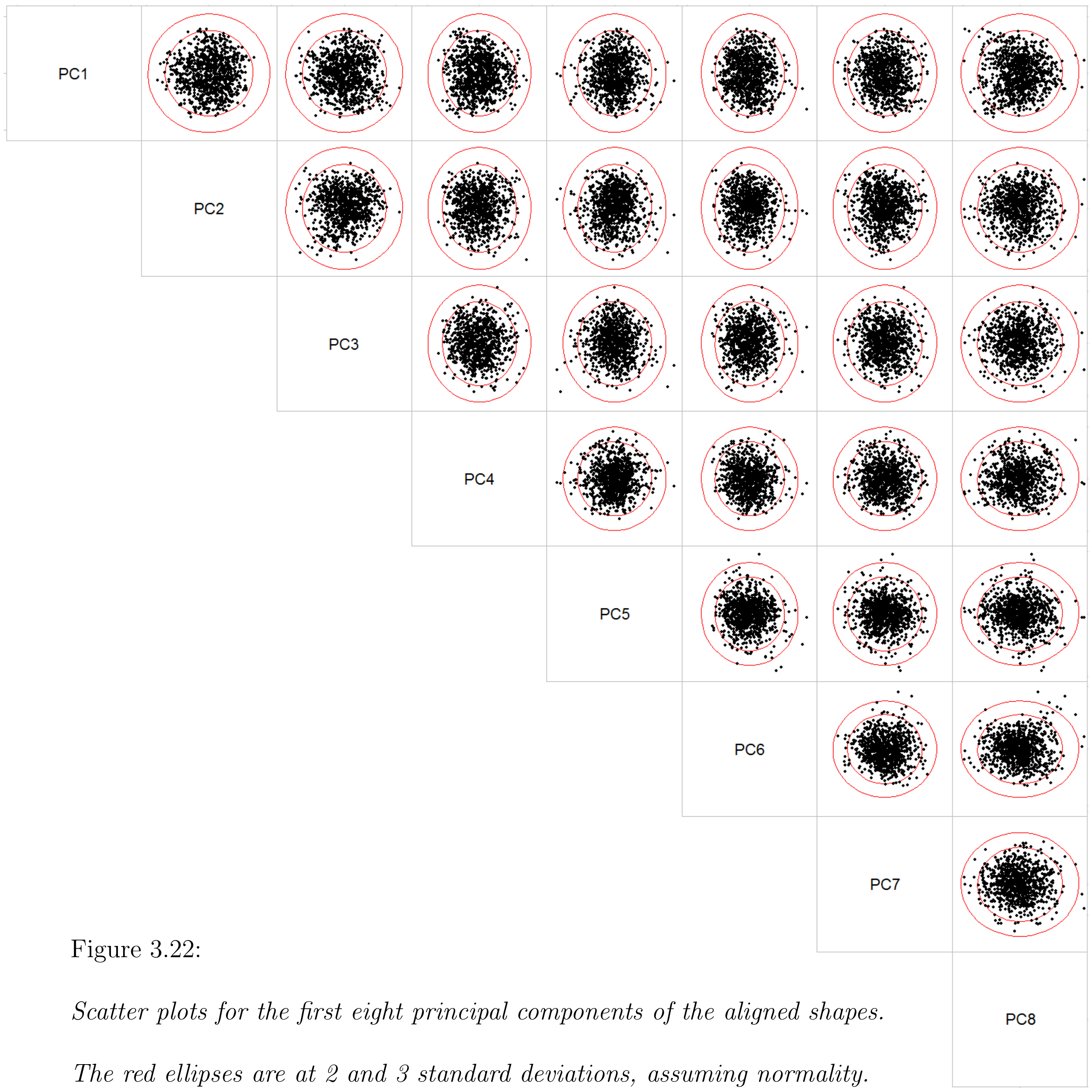


Figure 3.22:

Scatter plots for the first eight principal components of the aligned shapes.

The red ellipses are at 2 and 3 standard deviations, assuming normality.

To limit overplotting, only 1000 of the 6006 Stasm training shapes are shown (although all shapes were used to calculate the ellipses).

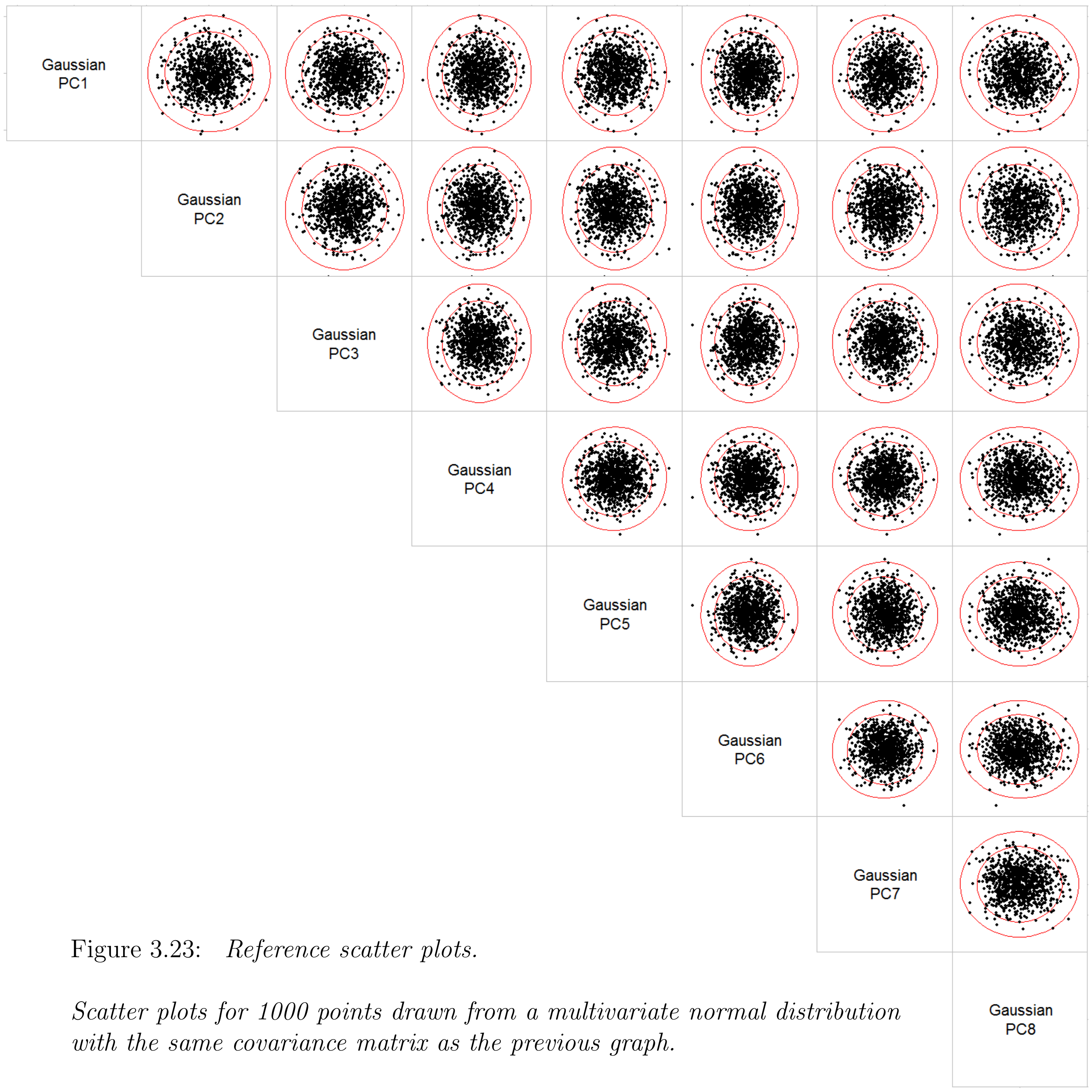


Figure 3.23: *Reference scatter plots.*

Scatter plots for 1000 points drawn from a multivariate normal distribution with the same covariance matrix as the previous graph.

Comments on the distributions

Summarizing, while it cannot be said that the shape data for the frontal model is Gaussian, it as at least somewhat Gaussian. Many alternatives to the classical shape model have been invented to handle deviations from normality. An example is the model of Kirschner and Wesarg [58], who replace the principal-component shape model with a model which minimizes combined image energy and shape energy, using "Distance From Feature Space" [85] as shape energy. Some other alternatives were mentioned on page 24 of this thesis, and Chapters 8 and 9 will mention some more modern approaches.

Chapter 4

Before the search begins

This chapter describes how we prepare the search image and position the start shape before the ASM search begins. These preliminaries don't seem to be explicitly described elsewhere in the literature (other than in the Stasm manual [78], and there only partially). Given their importance for accurate landmarking in our models they are described in some detail below. The ideas in this chapter will be extended when we consider multiview models in Chapter 9.

4.1 Adding a border to the image

Stasm uses the OpenCV face detector [66, 87] to find the face before the ASM search begins. This detector often fails if the face is near the edge of the image, which is frequently the case in cellphone "selfies" for example. To counteract this, our software adds a 10% border to each edge of the image (Figure 4.1).

With this border the face detector fail rate on the BioID set [2] goes from 3.3% down to 0.7%. However, this larger image does make face detection slower, because there is 44% more image area ($1.2 \times 1.2 = 1.44$).

4.2 Detecting the eyes and mouth

We need the position of the eyes and mouth to rotate the face upright and to position the start shape before the search begins.

The OpenCV eye and mouth detectors [14] used by Stasm aren't too trustworthy. They will sometimes make false positives or fail to detect eyes. This is probably true for any

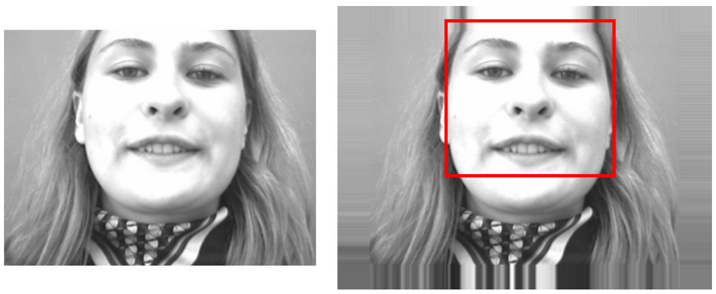


Figure 4.1:
left *Face is too near the edge— face detection fails.*
right *Artificial border added— face is now detected.*

(Face from the BioID set [2, 52].)

pure Viola-Jones eye or mouth detector—eyes and mouths are easily confused with other objects—and so some scaffolding is needed around the detectors for best results. To mitigate problems, we apply these detectors to only a limited area of the image, as shown in Figure 4.2. We search in the green rectangle for the left eye, but consider the detection valid only if the center of the detected eye is in the inner yellow rectangle. This helps minimize false positives on the eyebrows. If multiple eyes are detected, we choose the largest eye.

Similar considerations apply to the right eye and mouth (Figure 4.3). We also check that the left and right eye rectangles don't overlap by more than 10%, which would indicate a false positive.

The positions of the inner rectangles are fixed relative to the face detector rectangle. These positions are determined from the training data. If a different implementation of the face or feature detectors is used, the positions of the rectangles have to be redetermined.

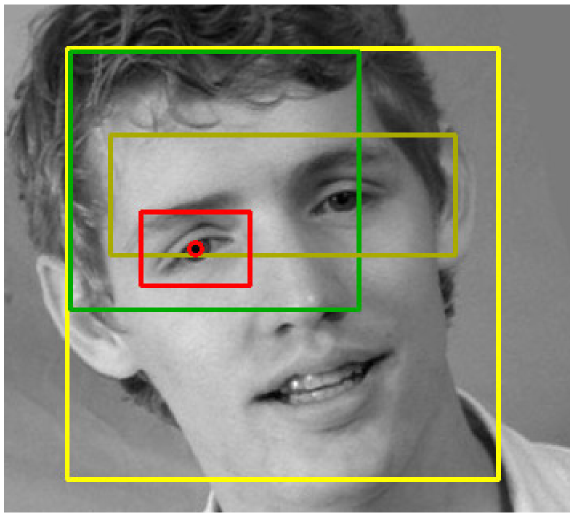


Figure 4.2: *Detecting the left eye.*

*The big rectangle is the **face detector rectangle**. Also shown is the **detected eye**.*

*We search for the left eye in the **left eye search area**, but consider it valid only if its center is in the **inner search rectangle**. This eye just squeezes in.*

The same rectangles are used for the right eye, except that the green search rectangle is shifted to the right.

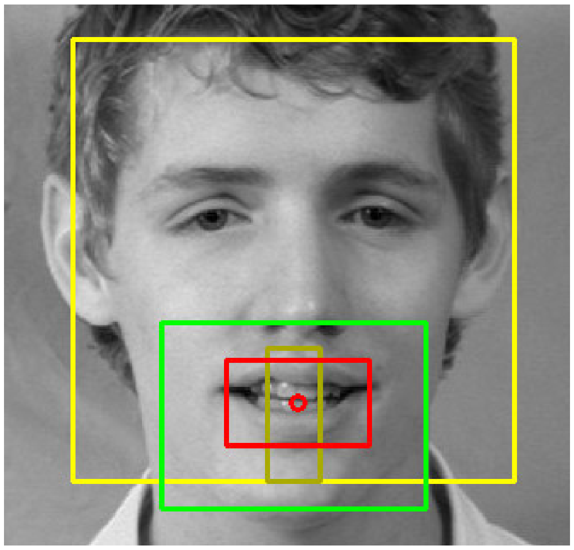


Figure 4.3: *Detecting the mouth.*

*The big rectangle is the **face detector rectangle**. Also shown is the **detected mouth**.*

*We search for the mouth in the **mouth search area**, but consider it valid only if its center is in the **inner search rectangle**.*

4.3 Rotating the face upright

The descriptors used by our software assume that the face is upright, and therefore we must rotate the image so the search face is upright. (Classical one-dimensional descriptors are rotated to align with the edge of the shape, but 2D gradient and HAT descriptors aren't rotated for alignment, thus uprightness is important.) Forcing the face upright also made for a better start in old versions of Stasm, before we used the eyes and mouth to position the start shape (Figure 4.4).

After detecting the face, our software therefore also detects the eyes (as described in the previous section). It then rotates the image so the detected eyes are horizontal. For efficiency we extract the region of the image around the face rectangle, and work with only that region, not the whole image. To minimize preprocessing, the image is rotated only if the eye-angle is greater than 5 degrees. If either eye isn't detected the image is left as is.

Even with the scaffolding described in the previous section, the eye and mouth detectors will still sometimes make false positives (e.g. detect an eyebrow instead of an eye) or false negatives (fail to detect an eye, possibly because the eye isn't in the search rectangle). In these situations the face won't be rotated upright correctly, making the ASM's job more difficult and mispositioned landmarks more likely.

The training faces aren't rotated, because in the MUCT data used for training the faces are already approximately upright. In this data there is some random minor tilting of the subject's faces that (we hope) assists the model learn how to handle minor rotational variance.

This section is concerned mostly with models for more-or-less upright frontal faces without large rotations. Chapter 9 takes these ideas further for faces with more varied poses.

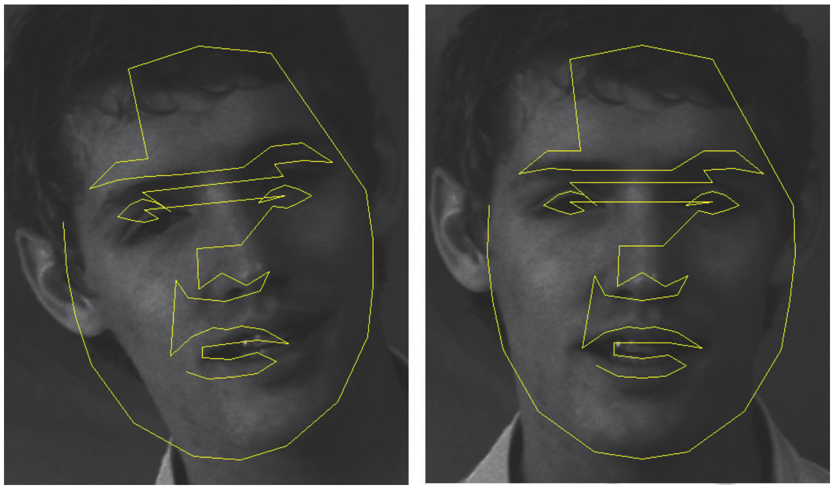


Figure 4.4: *Start shapes In these two examples the start shape is aligned to the face detector box.*

Left *Start shape on the original face.*

Right *Start shape after first rotating the image so the face is upright.*

4.4 Positioning the start shape

Alignment of the start shape is important for good automatic landmarking, at least for "difficult" faces. An ASM search often won't recover from a poorly positioned start shape.

4.4.1 Aligning to the face detector

The classical ASM generates a start shape by aligning the model mean shape to the face detector rectangle. During training, the model learns the optimal mapping of the mean face to the face detector rectangle. After the face has been detected the start shape is generated by mapping the mean face onto the detector rectangle (Figure 4.5).

In practice landmarking accuracy is marginally improved if the start shape is reduced slightly in size (5%) before the ASM search begins, probably because this reduction makes it less likely that the background outside the face will be mistaken for part of the face.

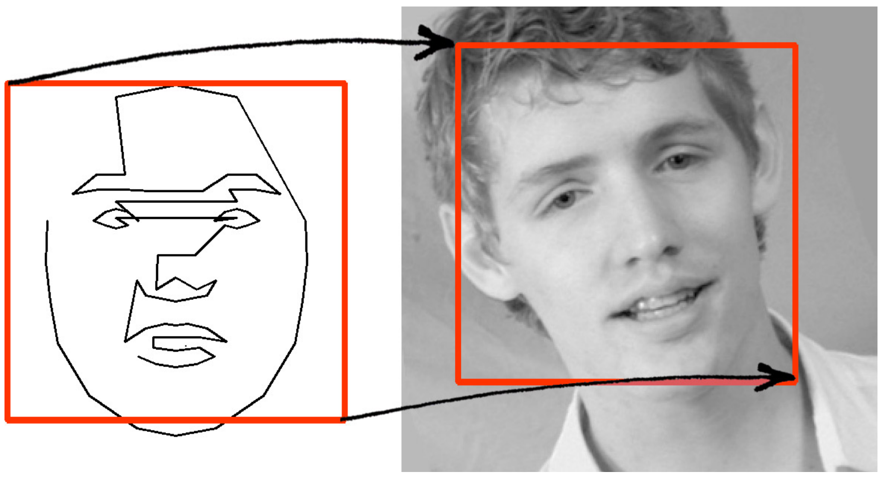


Figure 4.5: *Generating the start shape with the face detector box.*

On the left is the mean face with its optimum position in the face box determined during training.

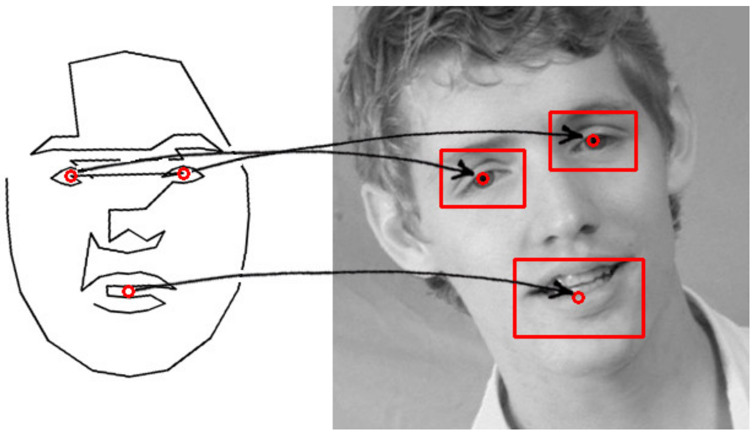


Figure 4.6: *Generating the start shape with the detected eye and mouth positions.*

The mean shape is on the left. Its eyes and mouth are aligned to the detected eye and mouth positions.

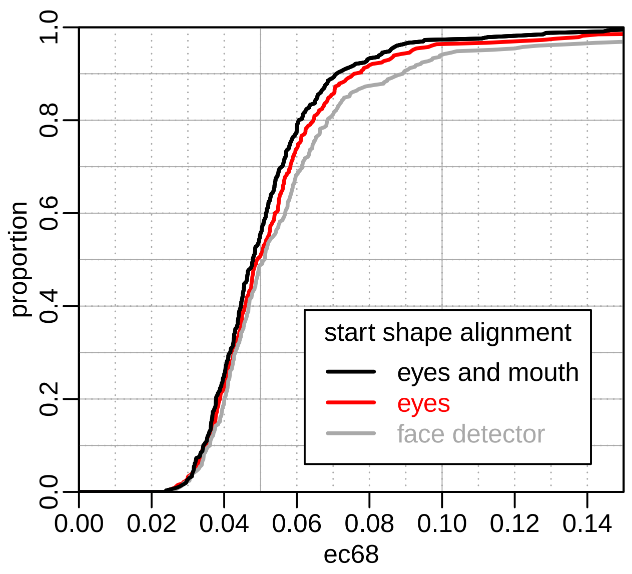


Figure 4.7: *Different methods for positioning the start shape*

Note especially the offset of the gray curve from the top axis on the top right of the graph. This indicates the percentage of catastrophic failures when only the face detector is used for positioning the start shape.

The figure shows fits by the multiview model (Chapter 9) using the ***ec68*** *measure on the Helen test set [61]. Please see Figure 9.12 on page 130 for details of the test set up.*

4.4.2 Aligning to the eyes and mouth

Automatic landmarking accuracy is better if the start shape is aligned to the detected eyes and mouth (instead of just aligning it to the face detector box as described above). We do this by aligning the triangle formed by the eyes and mouth in the mean shape to the eye-mouth triangle of the face in the image (Figure 4.6). If the mouth isn't detected we estimate its position from the eyes and face detector rectangle. If an eye isn't detected we fall back to using the face detector rectangle as described in the previous section. Other approaches may also make sense.

Jumping the gun a little by using the multiview model that will be described in Chapter 9, Figure 4.7 shows how different start shape alignment techniques affect fits on the Helen test set [61]. Notice how the alignment technique especially affects the "difficult" faces in the upper percentiles of the error distribution.

4.4.3 User initialization

In some applications, as an alternative to using face and eye detectors, the user manually pins several key points on the face. For example, we may require the user to initially pin the eye and mouth corners (possibly as a fallback method if the face detector or automatic landmarker fails). We generate the start shape by using these four key points to align the mean shape to the face in the image. If necessary the user can then further interact with the image to adjust points, as described in Section 3.12.

The face's pose can also be estimated if an additional key point (say the tip of the nose) is also manually pinned to give depth information. From these key points the face's yaw can be estimated using say a MARS model (Section 8.8) built by regressing the ground-truth yaw on the key points in the training shapes.

4.4.4 Stacking models

One way of better positioning the start shape is to run two ASM searches in series, using the results of the first search as the start shape for the second search [82].

Although this technique was found to give better results in older versions of Stasm, for the HAT models introduced in this thesis (Chapters 8 and 9), stacking doesn't improve overall landmark accuracies. This probably because HAT descriptors perform better than 2D gradient descriptors and thus can recover better from a poor start shape, and because the methods described above for aligning the start shape work better than in older versions of Stasm.

4.5 The eye-mouth distance

Stasm defines the *eye-mouth distance* as the distance from the centroid of the pupils (the mean position of the two pupils, which is near the top of the nose) to the bottom of the bottom lip.

The eye-mouth distance of the search face is estimated from the start shape; the eye-mouth distances during training are taken from the manual landmarks. If the pupils or bottom-of-the-bottom-lip landmarks aren't available, we use nearby points as surrogates. In the XM2VTS set ([18, 72]) of neutral frontal faces, the ratio of the median eye-mouth to interpupil distance is 1.3.

Other definitions of the eye-mouth distance are also suitable. In retrospect it may have been better to use the outer eye-corners instead of the pupils, since the eye corners are generally more visible and stable with respect to the rest of the face.

4.6 Scaling the face to a fixed eye-mouth distance

Before the search begins, we scale the image so the face has a fixed eye-mouth distance. The same scaling is used when training the model. The principle here is that the same region of the face should be covered by the descriptors during searching as when training, up to variations in pose and (often substantial) variations in face shape.

Stasm normalizes to an eye-mouth distance (Section 4.5) of 110 pixels. This particular value is mostly for historical reasons and may not be optimal. This distance is fundamental in that if it is changed, many other model parameters must also be redetermined (by selecting the best parameter value from a range of values on a parameter-selection set). For example, the number of pixels covered by the descriptors must be re-optimized.

Historically the inter-pupil distance was used for scaling, but with yawed faces the inter-pupil distance is no longer suitable (because inter-pupil distances decrease as the subject's head is yawed).

Chapter 5

Descriptor models

Recall that the job of the descriptor models is to take an approximate face shape and produce a better shape by descriptor matching at the tentative positions of landmarks. Before that process begins we first initialize the ASM search by positioning the mean face on the image using a global face detector (Figure 5.1). We then alternate the descriptor and shape models as described earlier (and summarized in Figure 2.4 on page 19).

A model descriptor is a template that captures some distinguishing quality of the image feature in question. It's a short signature that is unique to that feature. The template for say the outer corner of the left eye should match only that feature and no other, irrespective of the face in question, its pose, expression, makeup, or lighting. In reality no model descriptor is completely reliable.

This chapter first describes the 1D gradient descriptors used in the classical ASM [23]. (One-dimensional descriptors are usually called *profiles*, but we use the term "descriptor" in this thesis to situate them in the context of other kinds of descriptor.) This chapter then describes 2D gradient descriptors. Chapter 8 will introduce "HAT" descriptors, after a diversion for a few chapters to discuss model evaluation and face databases.

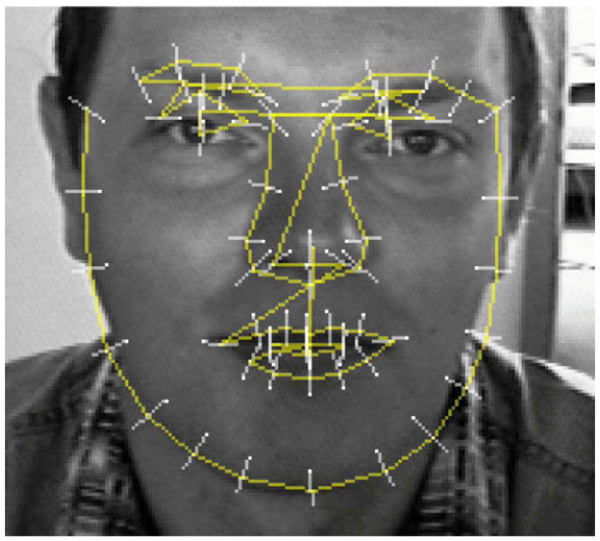


Figure 5.1: *The mean face (yellow) positioned over a face at the start of a search.*

In this example it so happens that the nose and mouth are already positioned quite accurately in the start shape. The eyes are badly positioned. (This is a 68-point start shape positioned on a BioID face [2] with a face detector.)

The image intensity under the white vectors is sampled to form 1D descriptors. The descriptors are orthogonal to the current shape, and are shown here centered on the current position of the landmark.

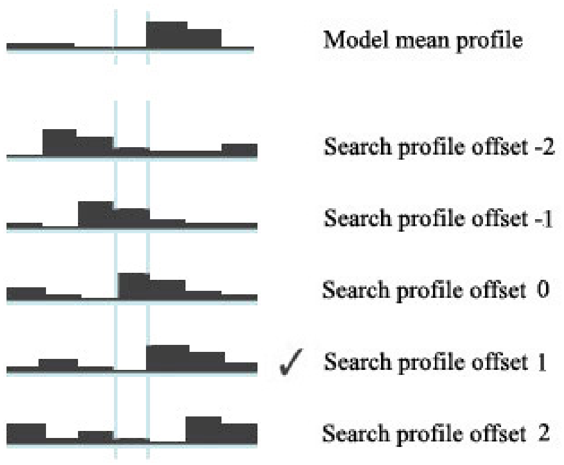


Figure 5.2: *Searching for a descriptor match.*

The top descriptor is the model mean descriptor $\bar{\boldsymbol{g}}$. The remaining descriptors are descriptors generated during the search by sampling the image at various offsets along the whisker from the current position of the landmark. The best match in this example happens to be at offset 1.

In practice, the Mahalanobis distance of the intensity gradient is used rather than the simple skyline match shown here.

5.1 1D descriptors

Looking at the jaw in Figure 5.1, a straightforward approach for nudging a point to its correct position on the jaw line would be to move the point to the strongest edge on the whisker. That is, move the point to the maximum intensity gradient on the whisker. This was the approach used in the "pre-classical" ASM [24]. (A *whisker* is a line at the landmark orthogonal to the shape boundary. By image intensity we mean the gray level, a value from 0 to 255. We don't use color.)

But looking for the maximum gradient is a bit simplistic, especially for more the complicated image structure internal to the face, where many of the landmarks aren't positioned on simple edges. We therefore adopt a pattern matching approach.

Instead of simply looking for the maximum gradient, we first build a template of the gradient along the whisker—what the image "looks like" around the landmark in question. This template forms the model *profile* or *1D gradient descriptor* for the landmark. This template is built during ASM training by averaging the gradient along the whisker across all training images. A separate template is required for each landmark at each pyramid level.

Then during the search for the best landmark position, we generate descriptors at various positive and negative offsets along the whisker, and select the offset that best matches the template (Figure 5.2). We typically use offsets up to ± 3 pixels along the whisker. This process is repeated for each landmark (as shown in step 4 in Figure 5.3) before handing control back to the shape model.

5.2 Generating a 1D descriptor

In detail, the descriptor vector $\boldsymbol{g}$ at a point is generated as follows (Figures 5.4 and 5.5):

1. Place the descriptor vector along the whisker, centered on the point, as illustrated in Figure 5.1.

```
input image of a face

1. Locate the face in the image
2. Position the start shape on the face
3. repeat
4a.     for each shape point
4b.         for each offset along the whisker
4c.             Generate a descriptor at the offset
4d.             Measure the fit of the descriptor against the model
4e.         Move the point to the offset that gives the best fit
5.      Adjust the suggested shape to conform to the Shape Model
6. until convergence

output shape giving the x,y coordinates of the face landmarks
```

Figure 5.3: *The ASM search algorithm, with expanded descriptor matching step. This figure is identical to Figure 2.4 on page 19, except that here we see more detail in step 4, the descriptor matching step.*

2. Set each element of the descriptor vector to the image intensity 0. . . 255 of the pixel under it.

3. Set each element to the intensity gradient, by replacing the element at each position i with the difference between it and the element at $i - 1$.
   (To avoid indexing off the vector when calculating the gradient of the first element, we temporarily extend the vector to include an extra element before the first element.)

4. Normalize the vector by dividing each element by the sum of the absolute values of the elements. (If the sum is zero, skip this step.)

The classical ASM uses the 1D descriptors described above. A number of variations of these 1D descriptors have been tried, but the above method works as well as any. Gradients are used because they have been found to give better results than gray levels or rates of change of gradient. Gradients are invariant to the overall light level. Normalization helps make the descriptor invariant to changes of overall contrast at the landmark.

5.3 Descriptor models

The descriptor model for a landmark consists of the mean descriptor $\bar{\boldsymbol{g}}$ and its covariance matrix $\boldsymbol{S}_g$. These are calculated from the training descriptors (one for each image) at the manual landmark.

If the length of the descriptor is 9 (as in Figures 5.4 and 5.5), $\bar{\boldsymbol{g}}$ will have 9 elements, and $\boldsymbol{S}_g$ will be a 9×9 matrix. If there are 77 landmarks and 4 pyramid levels, there will

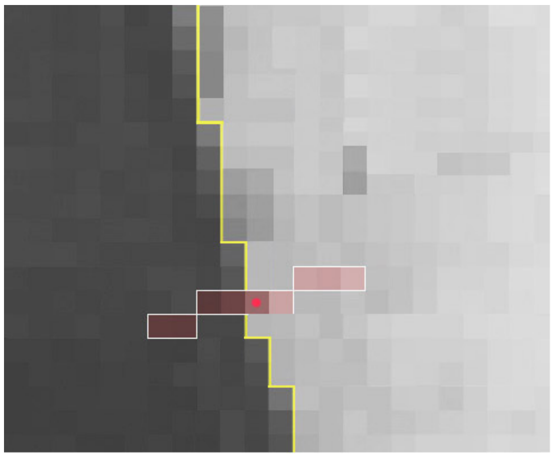


Figure 5.4: *A 1D descriptor.*

The current shape boundary is yellow.

The reddish area is the descriptor vector, along the whisker orthogonal to the shape.

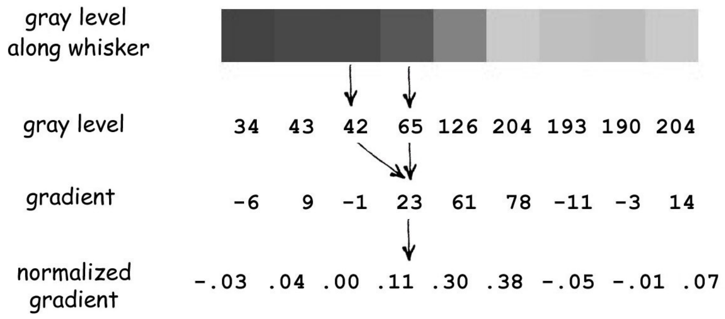


Figure 5.5:

How a 1D descriptor is generated from image intensities.

be 77 × 4 separate $\bar{\boldsymbol{g}}$'s and $\boldsymbol{S}_g$'s. (To reduce notational clutter, in this document we don't include suffixes on $\bar{\boldsymbol{g}}$ and $\boldsymbol{S}_g$ to indicate the pyramid level and landmark number.)

We need a way of calculating the quality of match between a search descriptor $\boldsymbol{g}$ and the model mean descriptor $\bar{\boldsymbol{g}}$. One way of doing that is to use Euclidean distances:

$$\text{Euclidean distance} \;=\; \sqrt{(\boldsymbol{g}-\bar{\boldsymbol{g}})^T\,(\boldsymbol{g}-\bar{\boldsymbol{g}})}\;. \tag{5.1}$$

However we get better results with Mahalanobis distances (readers unfamiliar with Mahalanobis distances may want to visit the overview in Appendix B):

$$\text{Mahalanobis distance} \;=\; \sqrt{(\boldsymbol{g}-\bar{\boldsymbol{g}})^T\,\boldsymbol{S}_g^{-1}\,(\boldsymbol{g}-\bar{\boldsymbol{g}})}\;. \tag{5.2}$$

Mahalanobis distances give more weight to elements of the descriptor that carry more information (i.e., elements that vary less across the training set and are therefore be more likely to be a reliable representation of the image gradient at the coordinate in question). Mahalanobis distances also account for linear relationships between elements (if two elements tend to vary together or in opposite directions, that covariance will be taken into consideration).

Note that at each landmark the ASM looks only for the single best match in the search region, and so we need be concerned only with the *relative* quality of match. A disadvantage of this simple approach is that it doesn't incorporate probabilities into the model (which could be used for instance to weight the points during the search, giving more reliable points more weight).

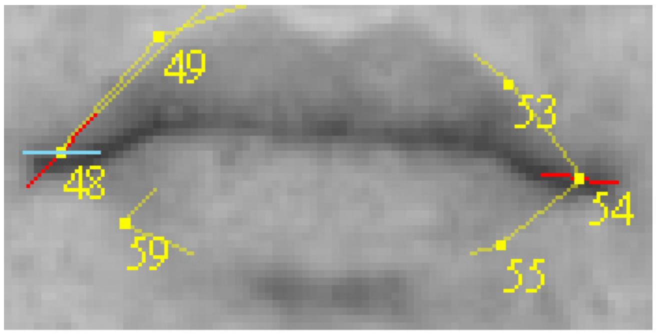


Figure 5.6: *Whisker directions.*

The direction of the two red whiskers is for the classical ASM.

The pale blue whisker uses explicitly specified "previous" and "next" landmarks.

(These landmarks are from the FGNET markup of the XM2VTS dataset [18, 72].)

5.4 Whisker directions

In the classical ASM, the direction of a whisker is determined by the order in which the landmarks are numbered, because a whisker's direction at a landmark is determined by the position of the previous and next landmarks. This direction can be rather arbitrary.

For example, consider landmark 48 in Figure 5.6. The previous landmark is 47, which is the right nostril (in this 68 point set). The next landmark is 49 which is on the upper lip. Compare the odd direction of the resulting whisker (shown in red) to that of the whisker on the symmetrical landmark 54.

Instead of using the ordering of previous and next landmarks to determine whisker direction, we can explicitly specify the landmarks to be used, and hence determine the direction along which the landmark can be moved during the search. In Figure 5.6, we specify landmarks 59 and 49, and the resulting whisker is shown in pale blue.

Explicitly specifying whisker directions like this gives slightly better search results (Section 5.4.8 in [75]).

5.5 Missing points in the training data

Points may be missing in some faces in the training data (Section 3.13). When generating the covariance matrix $\boldsymbol{S}_g$ for a landmark during training, we skip these missing points. The covariance matrix is thus an *approximate* covariance matrix, because the number of observations isn't the same for all variables [48]. Therefore it may no longer be positive-definite (a property necessary for meaningful Mahalanobis distances, Section B.3), but we force it so using the method in Section B.4.

We mention that when data is missing an alternative approach is to use iterative methods [6] for generating a maximum-likelihood covariance matrix that is positive-definite, but the approach in Section B.4 is simpler and suffices for our purposes.

See also Section 3.13.1, which discusses how missing points affect the shape model (as opposed to the descriptor models we are focusing on here).

5.6 2D gradient descriptors

The classical ASM uses one-dimensional descriptors. Now we make the logical extension to two-dimensional descriptors [82]. Instead of sampling a one-dimensional line of pixels along the whisker, we sample a square region around the landmark (Figure 5.7). Intuitively, a 2D area captures more information and this information, if used wisely, should give better results.

Note that 2D gradients aren't used in Stasm version 4.0 and higher (they have been superseded by HAT descriptors), so Sections 5.6 and 5.7 in this document apply only to older versions of Stasm.

In detail, a 2D descriptor matrix at a point is generated as follows:

1. After centering the descriptor matrix on the point, set each element of the matrix to the image intensity (0 . . . 255) of the pixel under it.

2. Replace each element by the intensity gradient using the convolution mask

$$\begin{array}{ccc} 0 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 0 \end{array} \tag{5.3}$$

   In words: double the pixel's intensity, then subtract the pixel above and the pixel to the left.[1] We temporarily extend the matrix to avoid indexing off its edge when calculating the gradients at its left and top boundaries.

3. Normalize the matrix by dividing each element by the sum of the absolute values of all elements. (If the sum is zero, skip this step.)

4. Apply sigmoid equalization by transforming each element x of the matrix as follows:

$$x' = \frac{x}{abs(x) + c} . \tag{5.4}$$

   The constant c determines the shape of the sigmoid. Figure 5.8 on the next page shows some sigmoids with different values of c. Sigmoids compress big positive or negative values but are approximately linear for input values near 0. This pulls in outliers, which are likely to be noise, such as the bright spots on the nose in Figure 2.5 on page 20.

[1]Although the term "gradient" is used as a shorthand for this convolution mask, this operation is actually an addition of horizontal and vertical derivatives, not a gradient in the mathematical sense.

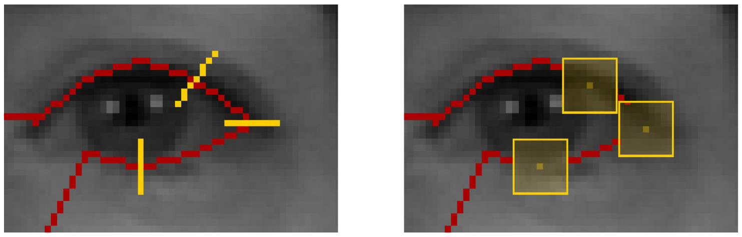


Figure 5.7:

left *1D descriptors.*

right *2D descriptors at the same points.*

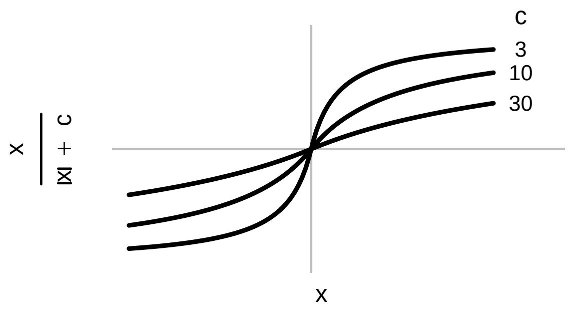


Figure 5.8:

Sigmoids for different values of the shape constant c.

When searching for the best position for the landmark, we displace the descriptor in both horizontal and vertical directions around the current position of the landmark, and move the landmark to the position that gives the best match to the model descriptor.

Just as for 1D descriptors, the descriptor model for a landmark consists of the mean descriptor $\bar{\boldsymbol{g}}$ and its covariance matrix $\boldsymbol{S}_g$, and we use Mahalanobis distances to calculate distances (Section 5.3). For the purposes of the equations in that section and internally in the software we treat the 2D matrix as one long vector by appending the rows end to end.

Unlike 1D descriptors, we don't rotate 2D descriptors to align with the shape boundaries. We must thus rely on the face being approximately upright like the training faces. For this reason, we rotate the image so the face is upright before the ASM search begins (Section 4.3).

Many variations of 2D descriptors have been evaluated, but the above descriptors were shown to work well for landmarking (in a limited study) in Chapter 5 of my master's thesis [75]. It isn't clear why the addition of horizontal and vertical derivatives (and normalization by the absolute sum) outperformed other techniques such as using gradient magnitudes or normalization by Euclidean length, but this question is now perhaps only of historical interest.

An example of the variations mentioned above is *weight masking*, which gives more importance to pixels closer to the landmark (Figure 5.9). If there is enough information in the training set such masking shouldn't be needed, because Mahalanobis distances automatically down-weight unimportant pixels (pixels with gradients that are highly variable across the training data). This was found to be true at least for the face data used in experiments in my master's thesis.

It turns out empirically that we get slightly better fits if we use classical 1D gradient descriptors on some landmarks, and 2D descriptors on others. More precisely, we use

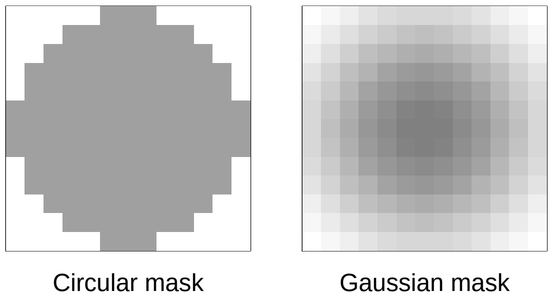


Figure 5.9:

Weight masks.

Elements of the descriptor matrix that are closer to the center get more weight.

1D gradient descriptors

(i) on the jaw and forehead landmarks at all pyramid levels,

(ii) on all landmarks at the coarsest pyramid level (level three, one-eighth full width),

and 2D descriptors everywhere else. See Section 8.5 for further discussion of this topic.

For efficiency, before the search begins at each pyramid level we precalculate the gradients for the region of interest on and around the face. This method is faster than calculating the gradients on an as-needed basis each time a 2D descriptor must be generated.

5.7 Trimming the descriptor covariance matrix

The covariance between two pixels in a descriptor tends to be much higher for pixels that are closer together. This means that for 1D descriptors, as we move away from the diagonal of the covariance matrix, the magnitudes of the elements get smaller (Figure 5.10). Intuitively then, we can *trim* elements that are too far from the diagonal, i.e., clear them to 0. Empirically, good results are obtained by trimming matrix entries that are more than 3 pixels apart.

Using trimmed covariance matrices makes landmark searches much faster. For 2D descriptors, ASM search time is dominated by the calculation of Mahalanobis distances. Representing the trimmed matrix as a sparse matrix, because most entries are zero, means that we can calculate Mahalanobis distances quickly—far fewer additions and multiplications are needed. In our software, a sparse matrix is represented by a $n \times 3$ array, where n is the number of non-zero elements in the matrix. The three elements in a row of the array specify the row, column, and value of an element of the sparse matrix.

In practice, we trim only 2D descriptor covariance matrices, because trimming 1D descriptor matrices results in a negligible overall speed increase. The size difference between the original and trimmed covariance matrices isn't big enough. (For illustrative purposes, Figure 5.10 shows the covariance of a 1D descriptor, where pixels that are close on the image are on the matrix diagonal. For 2D descriptors in vector form, pixels that are close on the image are in a grid structure in the covariance matrix, rather than on the diagonal. Trimming involves clearing elements of the covariance matrix that

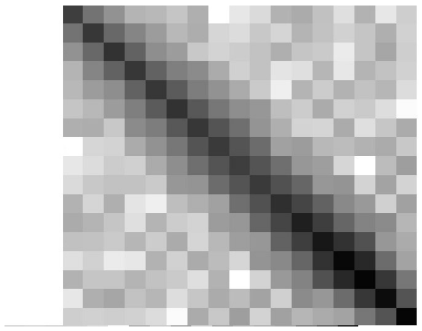


Figure 5.10:

Covariance matrix for a 1D descriptor. The landmark at the tip of the chin is illustrated.

The figure shows absolute values. Bigger absolute values are darker.

```
input a (possibly approximate) covariance matrix $\boldsymbol{S}$

force $\boldsymbol{S}$ positive-definite if necessary (Section B.4 on page 153)
invert $\boldsymbol{S}$
repeat
    force $\boldsymbol{S}^{-1}$ positive-definite if necessary (Section B.4)
    trim $\boldsymbol{S}^{-1}$ by clearing entries for pixels that are far apart
until the trimmed matrix $\boldsymbol{S}^{-1}$ is positive-definite

output the trimmed inverted covariance matrix $\boldsymbol{S}^{-1}$
```

Figure 5.11: *Inverting and trimming a covariance matrix.*

aren't in this grid, rather than simply away from the diagonal, but the principle is the same. See the function `TrimCovar` in the Stasm version 3.1 source code for details.)

Trimming results in a matrix that may no longer be positive-definite, a property necessary for meaningful Mahalanobis distances (Section B.3). Thus we force the trimmed covariance matrix positive-definite using the algorithm in Figure 5.11. Note that this algorithm is needed only when training the descriptor models, not when searching for landmarks.

5.8 Parallelization

During an ASM search, descriptor matching at each landmark is done independently for each descriptor (Step 4 in Figure 5.3). Thus the algorithm lends itself to parallelization across processor cores. In our software this is achieved with OpenMP [88]. In our experience, OpenMP can sometimes offer substantial speed advantages, although these may depend on the compiler in non-obvious ways—probably because modern compilers can automatically generate code that does loop parallelization across cores, without requiring explicit OpenMP or similar directives in the source code.

5.9 The number of landmarks

A straightforward way to improve the mean fit is to increase the number of landmarks in the model (Fig. 5.12). Fitting a landmark tends to help the fits on nearby landmarks, so results can be improved by fitting more landmarks than are actually needed. ASM search times increase approximately linearly with the number of landmarks.

Details on Figure 5.12: The figure was constructed as follows from the XM2VTS [72] set of manually landmarked faces. For a given number (from 3 to 68) of landmarks, that number of landmarks was chosen randomly from the 68 in the XM2VTS test. With the chosen landmarks, a model was built and tested to give one gray dot. This was

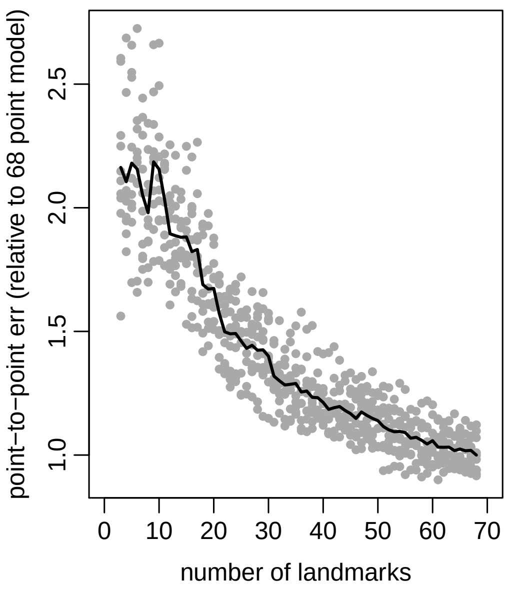


Figure 5.12:

Mean error versus number of landmarks.

Overall fits are better when the model has more landmarks.

repeated ten times for each number of landmarks. The black line shows the mean error for each number of landmarks.

5.10 The error surface

The left side of Figure 5.13 shows the region around the outer eye-corner of the left eye in an example image. The middle figure shows the gradient used for the 2D descriptors in this region.[2] The right side shows the mean 2D descriptor—the mean gradient around the landmark in the training images.

When searching for the best position of the a landmark, we generate 2D descriptors in an area around the current position of the landmark. The Mahalanobis distance of these descriptors from the model descriptor forms an *error surface* or *fitness surface* above the image.

Figure 5.14 illustrates an example error surface. It shows how the distance varies for descriptors sampled around the eye landmark in Figure 5.13. Ideally we would like to see a bowl shape here with a single clearly defined minimum (like the gray bowl in Figure B.2 on page 152). In reality the error surface is quite jagged.

Figure 5.15 shows the error surfaces for some landmarks on the image. If we averaged these surfaces over many images we would start seeing the ideal bowl shape.

An ASM will typically yield slightly different landmark positions if we use the same source image but at different resolutions. Similarly, if we apply an ASM to a mirrored version of a face, the resulting landmarks typically won't be an exact reflection of the original landmarks (Yang and Patras [122] investigate this phenomenon). This lack of invariance to scaling and mirroring is partially due to the roughness of these error surfaces and the consequent multiple near-minima.

[2]As remarked in the footnote on page 66, these aren't gradients in the mathematical sense; they are the scalar result of a convolution mask.

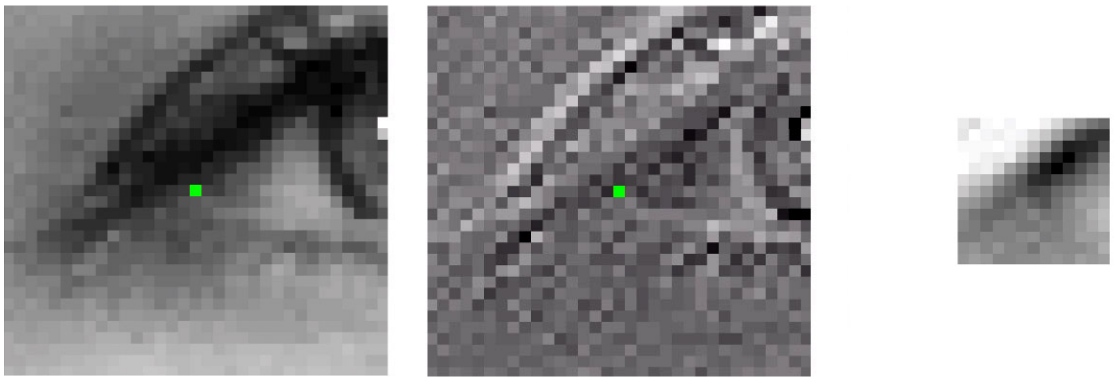


Figure 5.13: ***Left*** *A region of an image. The manual landmark is shown in green.*
Middle *The "gradient" used for the 2D descriptors in this region.*
Right *The model mean descriptor for the landmark.*

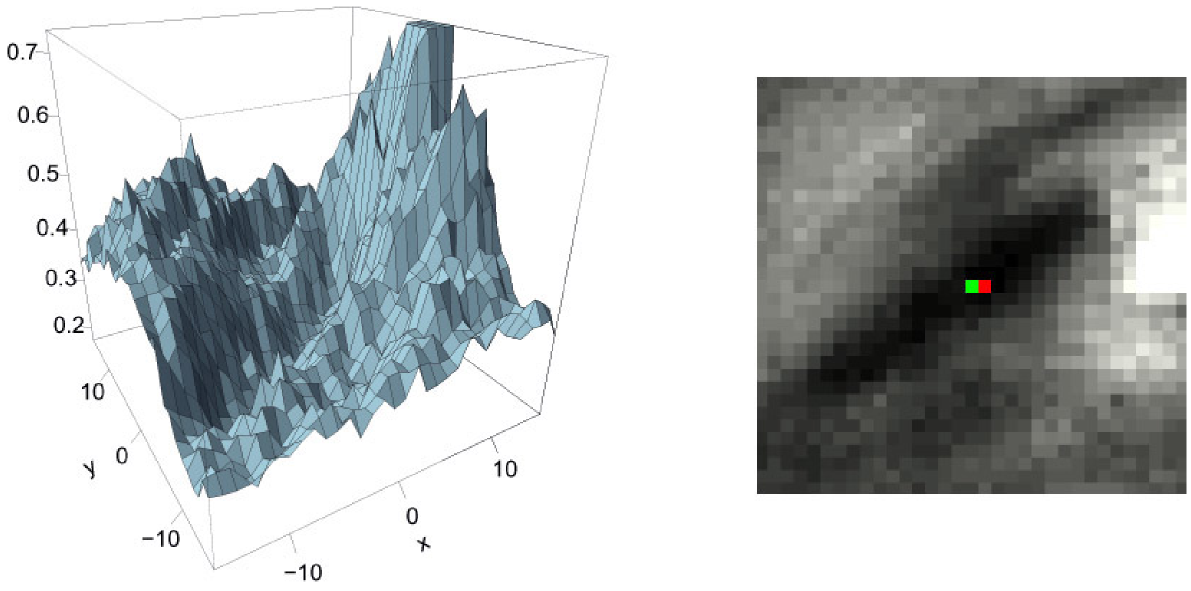


Figure 5.14: *The error surface across the region of the image in Figure 5.13.*

Both plots show the same surface, but represented in different ways. The plots show the Mahalanobis distance of the model mean descriptor from image descriptors at various offsets from the manual landmark.

The right plot is a plan view. Lower distances are darker. The green pixel is at the manual landmark and the red pixel is at the minimum distance.

For illustrative purposes the figures show a large search area, but in practice during the ASM search we actually search a smaller area.

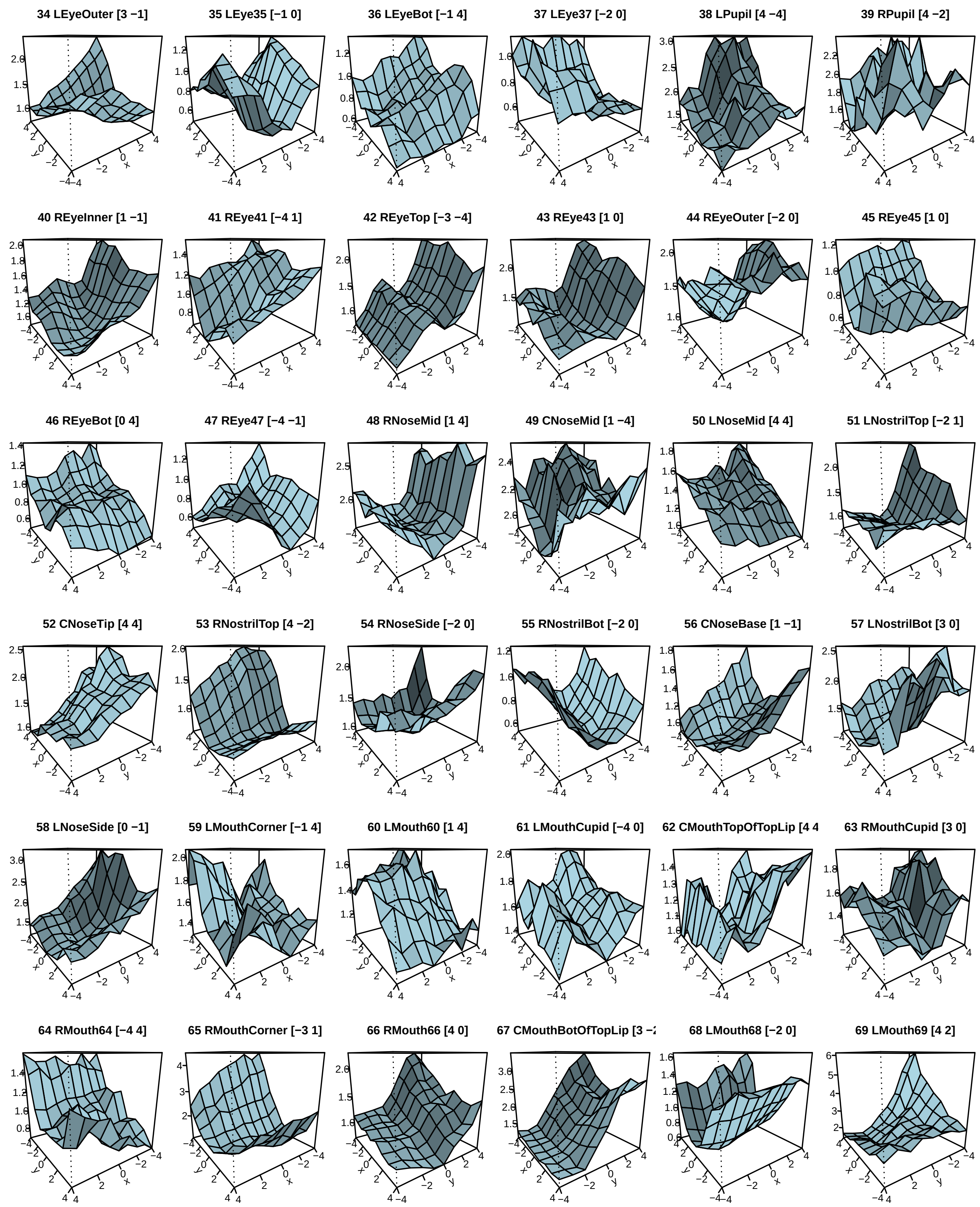


Figure 5.15: *Error surfaces for some 2D gradient descriptors on a single image.*

The numbers in [square brackets] in the title of each plot shows the x,y offset of the minimum.

The plots were generated for 2D gradient descriptors at pyramid level 1 (eye-mouth distance 55 pixels) using the MUCT 77 landmarks.

Chapter 6

Model evaluation

The previous chapter described descriptor models and we would like now to get to HAT descriptors, the form of SIFT descriptors that are novel to this thesis. However, before we can do that we need to take a diversion for a few chapters to discuss model evaluation and face databases.

This chapter in particular discusses how we evaluate models by plotting fitting errors. It also discusses the various definitions for these errors. Readers uninterested in these engineering details can proceed directly to Chapter 8 "HAT descriptors". Readers unfamiliar with the need for independent training, tuning, and test sets may want to first visit Appendix C, which is a more general overview of the datasets necessary when building and evaluating models.

6.1 Graphs for comparing models

Assume we have a set of landmark fitting errors for a number of faces. For example, we may have the mean error for each of 300 faces. In this section we look at different ways of plotting such error values. (The next section will look at ways of measuring fitting error.)

Figure 6.1 shows histograms of the 300 values. Histograms have an intuitive appeal, but with different numbers of bins the plots vary substantially. There isn't a way to automatically choose the number of bins that is unambiguously "correct" (e.g. Venables and Ripley [113] page 112). An additional problem is that when used to compare models, the histogram plots become cluttered. On the left side of Figure 6.2 it's quite

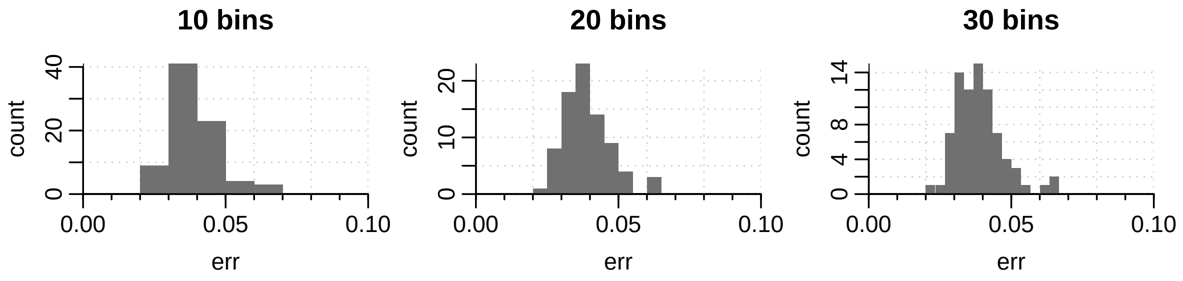


Figure 6.1: *Three different histograms of the same data. The shape of the histogram depends on the number of bins.*

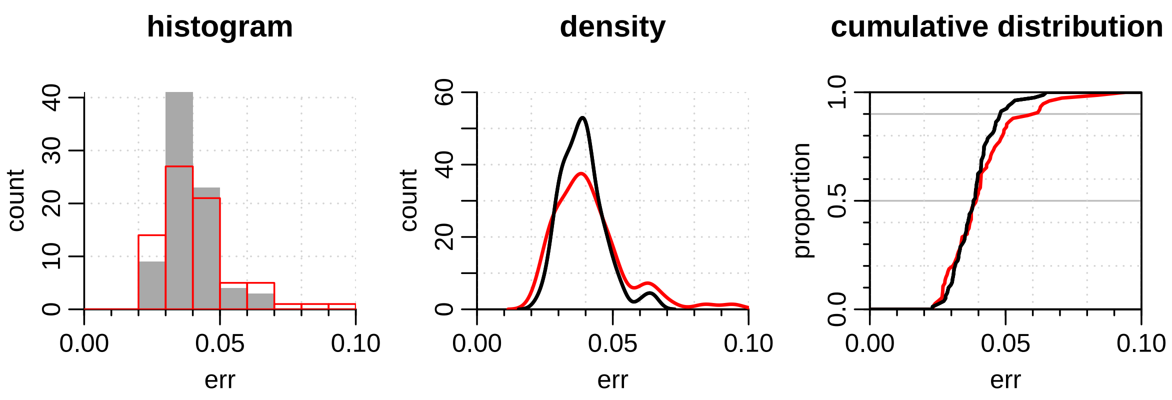


Figure 6.2: *Three different ways of comparing two models.*

difficult to tell which model is better, the gray shaded model or the red model.

Density plots are an alternative to histograms. The density plot in the middle of Figure 6.2 is less cluttered than the histograms. But it's still fairly difficult to determine which of the two models has better performance—and would be even more so if several models were plotted. The plotted curve will differ with different density estimation techniques and bandwidths, and there isn't a way to decide which of these is correct for the given data (e.g. [113] page 126ff.).

Cumulative distribution plots don't suffer from these disadvantages. The right side of Figure 6.2 shows an example. The quicker the S curve starts and the faster it reaches the top, the better the model.

In the example cumulative distribution plot we see that the black model performs better on the whole than the red model. The red model does slightly outperform the black model in the left part of the curve, but the black model does better on the more difficult faces above the median.

Given a vector of errors `errs` with `n` elements, the pseudocode to draw such a curve is

```
x = sort(errs)
y = (1:n) / n
plot(x, y)
```

This is a plot of the Empirical Cumulative Distribution Function (ECDF), which is a step function: for each step $1/n$ in the vertical direction, in the horizontal direction we move to the next biggest error. The approximate bell shape in the density plot translates to an approximate S shape in the cumulative distribution plot.

There are a few minor implementation decisions about how the curve is plotted concerning interpolation between adjacent points, how ties are handled, etc. However, these decisions have a negligible effect on the curve if there are enough values (say 30), without many ties.

It can be helpful to draw horizontal lines at the 90th percentile and 50th percentile (the median). Note that in the example the median for both models is almost identical. So in this example merely quoting the median fits would be an inadequate way of comparing the models.

6.1.1 Compare models using the same test data

When comparing models, the same test data should be used for all models. This seems obvious, but there are a few practical difficulties that tempt people to compare curves in the same plot on different subsets of the data, typically when comparing their model to curves published by other authors.

Sometimes all the faces in the original dataset are no longer available. For example for copyright reasons the image files of the LFPW [7] dataset aren't directly downloadable from the LFPW site—just the web addresses of the files are provided. Since many of these addresses are now stale, many of the images are no longer available, so people tend to publish curves on subsets of the data. This becomes a problem if the same figure compares curves on the full set or a different subset.

Faces that aren't detected by the face detector shouldn't be omitted (unless the same faces are left out in all curves). The error value for faces that aren't detected should be taken as infinite. (Actually any value greater than the maximum on the horizontal axis will give the same curve.) If non-detected faces are omitted, the worse the face detector the better the model appears. This somewhat counterintuitive result is because "difficult" faces tend to be filtered out because they aren't detected.

Figure 6.3 shows an example that compares the same model on different subsets of the BioID data. The black curve is for the full set of data. The other curves make the

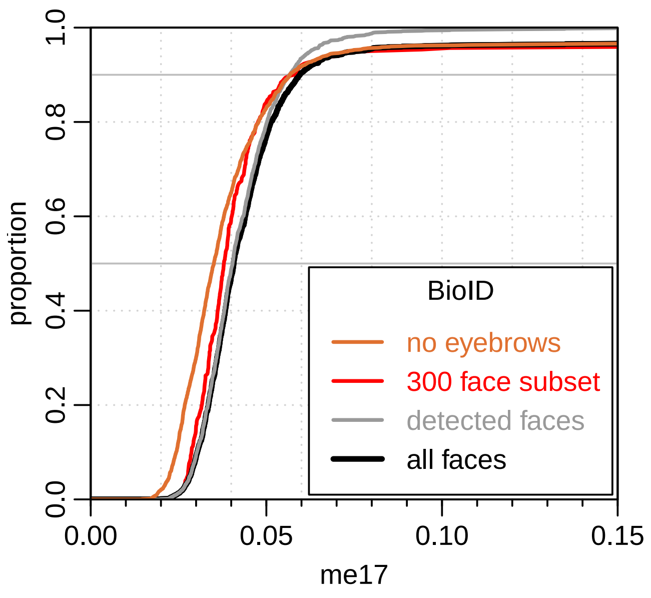


Figure 6.3: *All the faces in the test set should be plotted. Leaving out faces or points makes the curves incommensurate.*
(This plot uses the ***me17*** *measure described in Section 6.2.1.)*
The ***orange*** *curve omits the eyebrow points.*
The ***red*** *curve is for only the first 300 faces.*
The ***gray*** *curve omits faces that weren't detected by the face detector.*
The ***black*** *curve is the me17 for the full set of data. (It's the same as the Stasm3 curve in the top left plot of Figure 8.18.)*

model appear better than the black curve.

The same set of reference landmarks should be used as a basis for comparison in all curves—for some well-known datasets there is now more than one set of reference points (Section 9.6.3).

The face detector used by the model can have a substantial impact on landmark fitting accuracies. For this reason, ideally all models being compared should use the same face detector. However often this isn't possible because different organizations use different (often proprietary) face detectors.

Cross-validation is sometimes employed for measuring landmarking performance, but its use is often ill-advised (Section C.3).

6.2 Performance measures

This section describes some measures used to measure the performance of automatic face landmarkers. These compare the position of the automatic landmarks to the manual landmarks. All the measures here are in use by various researchers, although sometimes with different names.

The **point-to-point error** is the Euclidean distance in pixels between the point discovered automatically by the ASM search and the corresponding manually-landmarked reference point.

The **mean normalized point-to-point error** is traditionally defined as

$$me \quad = \quad \frac{1}{d_{eye}} \; \frac{1}{n} \; \sum_{i=1}^{n} d_i \; , \tag{6.1}$$

where d_{eye} is the pupil-to-pupil distance in pixels on the manually landmarked reference shape, n is the number of landmarks in the face, and d_i are the point-to-point errors in pixels.

The word “traditionally” appears above because distances other than the inter-pupil distance are also used to normalize the data, especially in recent years. For example in Chapter 9, to compare results with other researchers that use a different standard, we will normalize with the distance between the eye *corners*, using the `ec51` and `ec68` measures.

6.2.1 The me17 measure

Introduced by Cristinacce ([28] and [27] Section 6.1.4), the `me17` is the mean point-to-point error divided by the inter-pupil distance (Definition 6.1) over 17 standard points. These 17 points are a subset of the BioID points [2]. Figure 6.4 illustrates.

Note that the me17 points are internal to the face, thus the measure doesn't account for landmarking accuracy on external points such as those on the jaw.

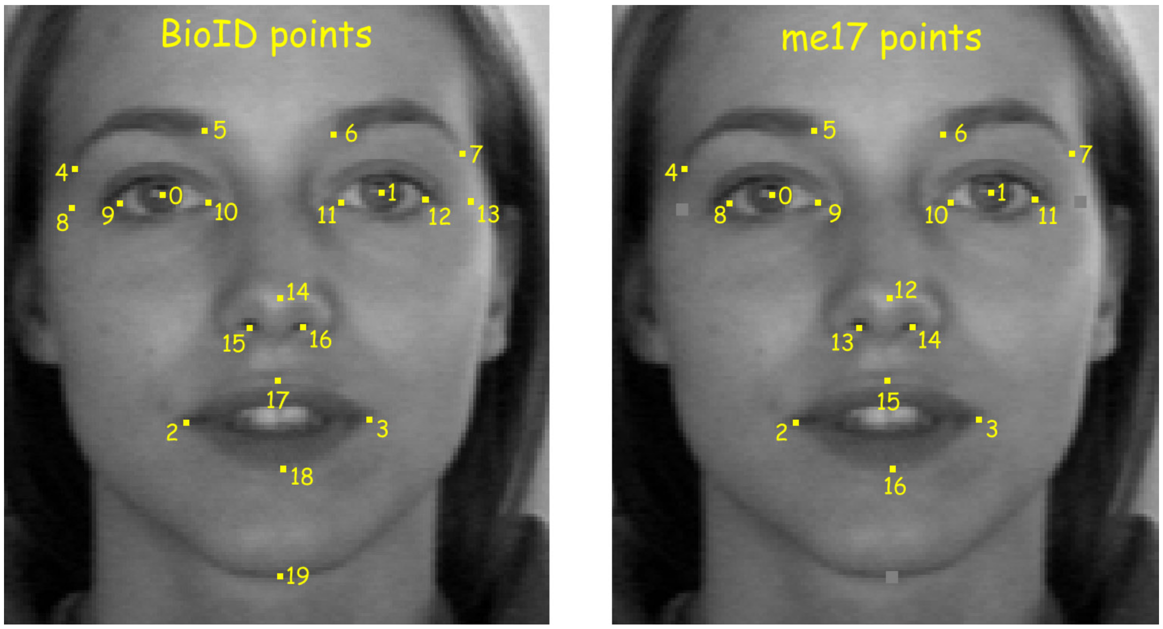


Figure 6.4:
Left *The BioID points [2]. The BioID faces were manually landmarked by David Cristinacce and Kola Babalola [18].*
Right *The* `me17` *points are a subset of the BioID points. All points are interior to the face.*

6.2.2 The pe17 measure and confusion in the literature

There is some confusion in the literature, as pointed out in e.g. Lindner et al. [67] Section 4.2.4. To help clear this confusion, and so we can study the details, we also give names here to some additional measures.

The **normalized point-to-point error** of a point is

$$pe = \frac{d}{d_{eye}} \, , \tag{6.2}$$

where d is the point-to-point error in pixels, and d_{eye} is the pupil-to-pupil distance in pixels on the reference shape

The **pe17** is a set of 17 normalized point-to-point errors. The same points are used as the me17 measure. We emphasize that the pe17 is a *set* of 17 points—the points aren't averaged over the face. When plotting pe17 errors, the distribution of the individual point errors is plotted, regardless of the face to which they belong. (Each point on the graph represents one landmark, whereas in an me17 plot each point represents one face.) Figure 6.5 is an example that plots the pe17s and me17s for the same model on the same test data: the curves are incommensurate.

The confusion in the literature arises because people make the easy mistake of confounding the pe17 (individual errors) with the me17 (errors averaged over the face). For example, Figure 8 in Belhumeur et al. [7] mixes up pe17 and me17 measures on the same graph. Figure 6.6 is a marked up version of their Figure 8. Notice that the figure uses the pe17 for the two left curves but the me17 for the other curves.

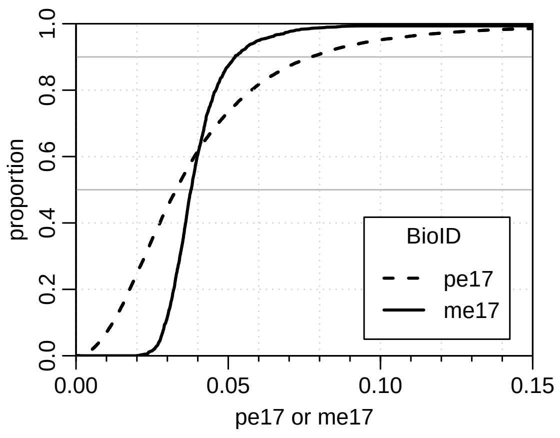


Figure 6.5: *Two different measure of fit with the same model on the same data. The curves aren't comparable.*

*These results are for Stasm4 (Chapter 8) on the BioID data. (The **pe17** curve is identical to the black curve in Figure 8.20. The **me17** curve is identical to the black curve in the top left plot of Figure 8.18.)*

It's fairly easy to spot when a curve is for the pe17 and not the me17. The leftmost point of a pe17 curve typically starts at a fit of very close to zero, whereas an me17 curve starts at about 0.02 or greater, because of the inherent uncertainty in manual landmarks (Section 7.2). An me17 of zero would require that the automatic landmarks match the manual landmarking noise in all 17 of the manual landmarks—which would require severe overfitting to the manual landmarks. Both curves have somewhat of an S shape (because the errors have an approximate bell shaped density distribution), although the S tends to be more pronounced for me17 curves.

I personally favor the me17 over the pe17, for its historical lineage and because it allows overall fits on faces to be easily compared.

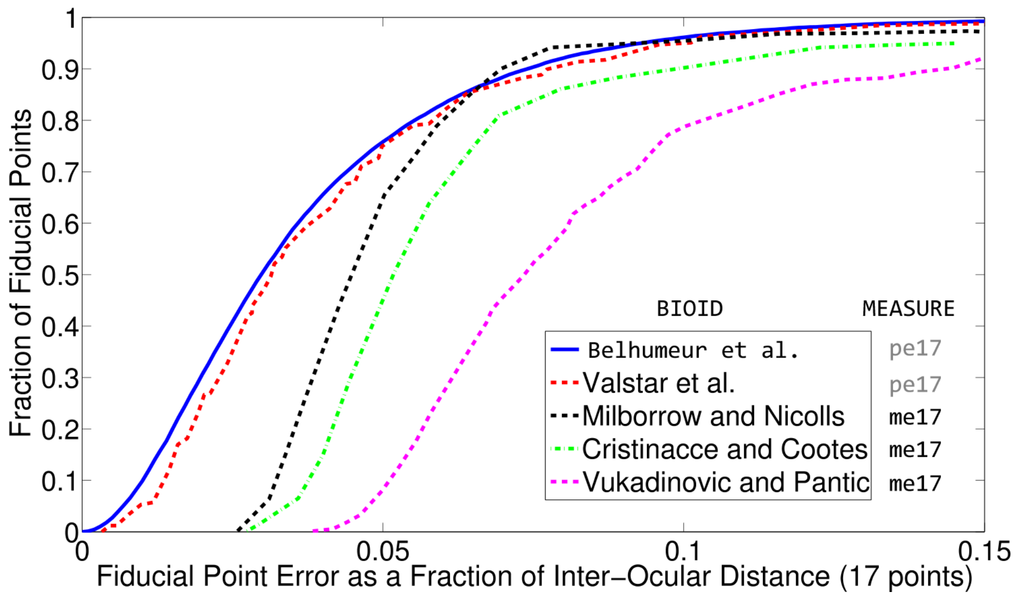


Figure 6.6: *Copy of Figure 8 in Belhumeur et al. [7]. The legend has been updated to include an indication of which measure was used for each curve.*
*The **pe17** and **me17** measures are incommensurate, and shouldn't actually be shown on the same graph.*
*The papers referred to in the legend are Belhumeur et al. [7], Valstar et al. [112], Milborrow and Nicolls [82], Cristinacce and Cootes [28], and Vukadinovic and Pantic [117]. See Figure 8.20 on page 115 for a comparison of **pe17**s for Stasm and the Belhumeur et al. model.*

6.2.3 Rounding

By historical precedent the estimated landmark positions aren't rounded to integers before calculating point errors. If they were rounded, the leftmost point on the pe17 distribution curve would typically hit the vertical axis above zero, since a portion of the landmarks would have a zero point-to-point error.

Rounding typically has a very small or negligible effect on me17 curves.

6.2.4 The mem17 measure

Normalizing by the inter-pupil distance helps make the measure independent of the overall size of the face. However, for datasets that include three-quarter views, normalizing with the inter-pupil (or similar transverse) distance is inappropriate—because as a face turns to the side, the inter-pupil distance decreases.

For this reason, we define the **mem17** measure. This uses the same points as the me17 measure, but normalizes by dividing by the eye-mouth distance (Figure 6.7). Stasm defines the eye-mouth distance as the distance from the centroid of the pupils to the bottom of the bottom lip (Section 4.5). Normalizing with the eye-mouth distance is more universal than with the inter-pupil distance, because it's appropriate for both frontal and yawed faces. It's not appropriate for pitched faces, although these are much less common in real applications.

Another way of normalizing is to divide by the mean of the height and width of the face's bounding box [132], however this too will vary as the face's pose changes. Also it should be borne in mind that the size of this bounding box is subject to the substantial unreliability of the reference manual eyebrow and jaw landmarks.

At the current time (2016) inter-pupil or similar transverse normalizations are still very prevalent, even though they are inappropriate for the multipose datasets that are now in common use.

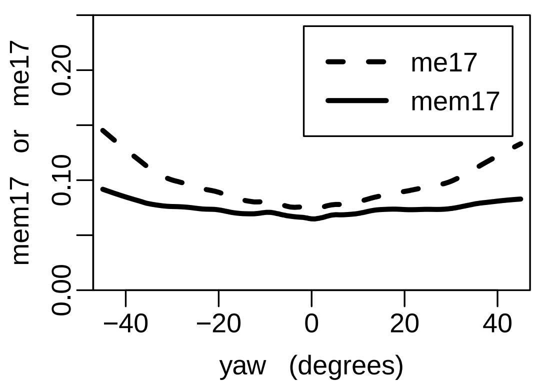


Figure 6.7: *Two different measure of fit with the same model on the same data. The curves show the mean landmarking error on faces at different yaws.*

The `me17` *isn't a good measure for datasets with yawed faces. This is because as faces turn the inter-pupil distance decreases and the* `me17` *artificially increases.*

These results are for the multiview model (Chapter 9) on 3500 AFLW faces. The `mem17` *curve here is identical to the black curve in Figure 9.7.*

6.2.5 Other measures

The me17 and similar measures ignore points on the perimeter of the face, and give equal weight to all included points. But points on the perimeter of the face do in fact matter, and accurate location of some points is usually more important than others. For example it's usually more important to locate the corners of the eyes accurately than the tip of the nose. Thus weighted measures become attractive.

For model parameter selection we have used weighted measures internally that include points on the perimeter of the face. The weighting is anisotropic, because even if locating the edge of the jaw (for example) is important, the precise position *along* the edge doesn't matter. There is a good deal of arbitrariness about the definition of such measures, but they give a better idea of overall fitness than measures like the me17 or measures that use all the points but give them equal weight.

6.3 Independent tests on Stasm version 3.1

In 2013 Çeliktutan et al. [15] made independent tests using all automatic landmarkers available for download at the time. The version of Stasm at that time was version 3.1, which uses 2D gradient descriptors (Section 5.6). As can be seen in Figure 6.8, by their measures Stasm 3.1 performed well on these sets of frontal faces.

Given the discrepancy between these results and those claimed in some papers, a certain amount of skepticism is warranted when reading comparative results in papers—and even more so when developers don't make their software available for independent verification. It's not unusual to find mistakes in papers comparing fitness curves, such as plots of model fitness measured on test data that is contaminated with data used to build the model (Appendix C).

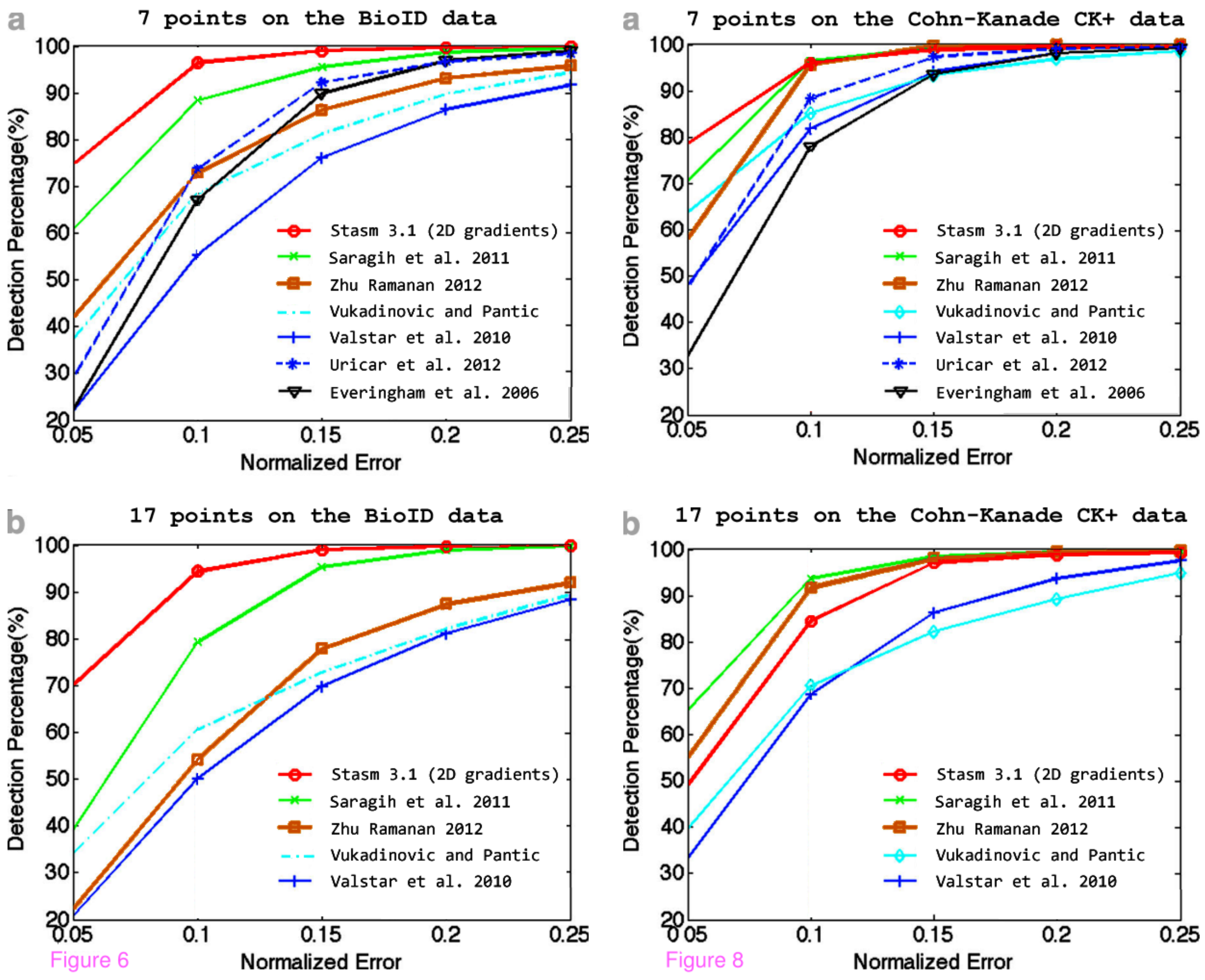


Figure 6.8: *Relative performance of various automatic landmarkers.*

Reproduced with permission from Figures 6 and 8 in Çeliktutan et al. [15]. Please see that paper for details of the fitness measures used. For clarity in the current context, the legends were updated and the color of the Zhu Ramanan curve was adjusted.

The curves are for Stasm version 3.1, Saragih et al. [102], Zhu and Ramanan [132], Vukadinovic and Pantic [117], Valstar et al. [112] (same model as Figure 6.6), Uricar et al. [111], and Everingham et al. [36]. The databases are the BioID data [2] and Cohn-Kandade CK+ data [70].

Chapter 7

Landmarked face data

This chapter discusses some aspects of landmarked face data.

7.1 The MUCT database

There are a variety of downloadable manually landmarked databases. This section describes the MUCT data [81] that was used for training the models in this thesis.

The MUCT data consists of 3755 images of 276 subjects of which 51% are female. The mean inter-pupil distance is 88 pixels. All subjects were 18 or more years of age. Subjects who wore makeup, glasses, or headdresses retained those for the photographs. Subjects weren't asked to evoke any particular facial expression. In practice this meant that most were photographed with a neutral expression or a smile. The database was created to provide more diversity of lighting, age, and ethnicity than similar landmarked databases at the time (2009).

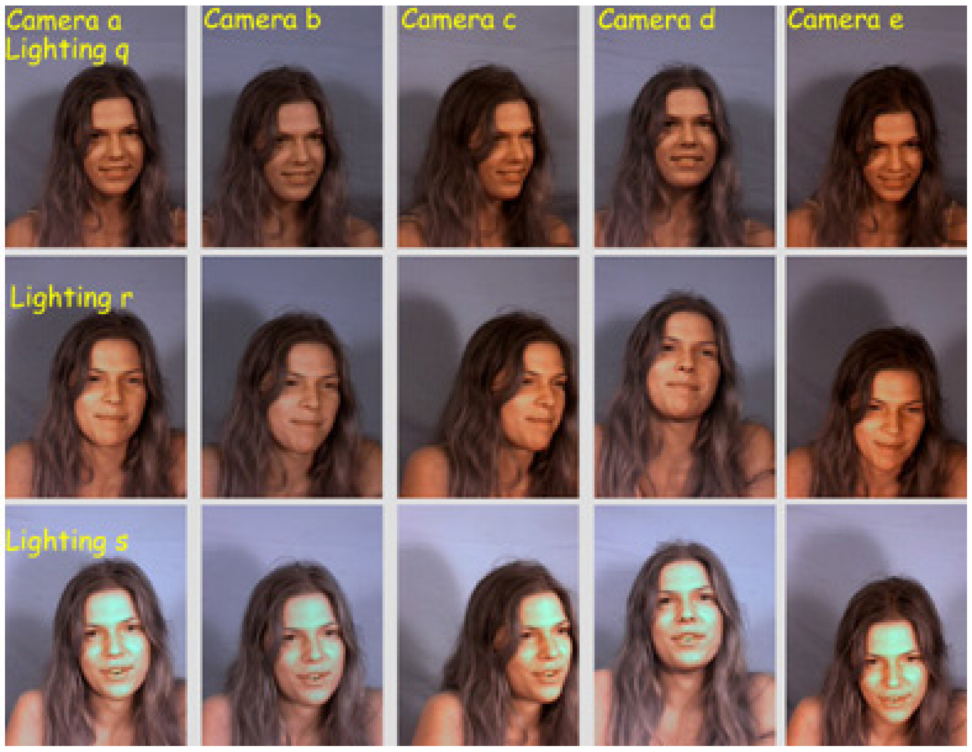


Figure 7.1: *Images of the first subject in the MUCT data, showing different poses and lighting.*

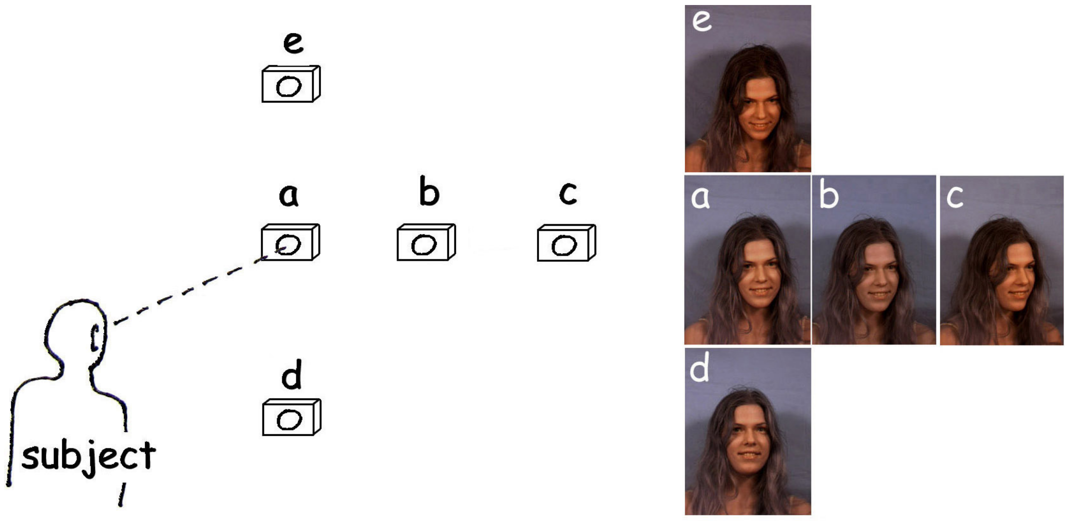


Figure 7.2: *The five MUCT cameras and their relationship to the subject's face. Camera* `a` *is directly in front of the face, up to variations in height of seated subjects.*

Ten different lighting setups were used, and each subject was photographed with two or three of these lighting sets. Not every subject was shot with every lighting setup, to achieve diversity without too many images.

The subjects were photographed against a blank background (Figure 7.1). Five cameras were used (Figure 7.2) to get five poses for each subject (labeled `a` to `e`). No cameras were located to the left of the subject, since those views can be approximated for training purposes by mirroring the images from the cameras on the right. Each subject was seated and faced camera `a`. Differences between the seated height of subjects and natural variations in pose introduced some variation in their orientation relative to the cameras. (Small variations like this probably help models trained on the data to handle a wider variety of faces when searching for landmarks.)

7.1.1 Using the MUCT data for training

For training the models for the applications focused on in this thesis, we need faces that are of predominantly medium or high resolution, and of good quality (as described in Section 1.2 paragraph iii). Surprisingly, of the many downloadable multipose landmarked databases currently available (2016), the MUCT database remains one of the few that comes close to meeting these criteria. Most downloadable multipose databases have too many low-quality faces (see the discussion in Section 7.3). Therefore, although not quite ideal, the MUCT data was used for training the models in this thesis. (The MUCT data is not quite ideal because the faces weren't photographed in the wild and the resolution ideally should be a bit higher.)

Details on how the data was partitioned for training the models are given in Figure 9.6 on page 123. In summary, the HAT frontal model (presented in Chapter 8) was trained on the frontal or near-frontal `abde` poses. The three-quarter model (presented in Chapter 9) was trained on the three-quarter or near-three-quarter `bc` poses.

7.1.2 The MUCT 77 landmarks

The new models in this thesis were trained on the MUCT data with 77 landmarks. This subsection gives a bit of background on the MUCT landmarking schemes.

The MUCT data was originally landmarked and published with 76 manual landmarks. Up to small manual landmarking biases these are a superset of the 68 XM2VTS landmarks. (Eight extra landmarks were added to the eyes.) Later we converted these to 77 landmarks. This isn't a superset of the original 76 landmarks, but most of the landmarks are shared. (There are several figures in this thesis that illustrate these landmarks, e.g. Figure 1.1 on page 10.) The 77 point set was used for training the models in this thesis. This 77 point scheme also allowed the MUCT data to be incorporated into a 77 point internal (unpublished) database that is closer to the ideal data mooted in Section 7.3. Software routines to convert the 77 landmarks to other mainstream landmark formats are provided with the Stasm software.

For the 77 point set some landmarks felt to be redundant in the original set were removed (at the top of the nose, in the center of the mouth, and some landmarks along the jaw). Forehead landmarks were added, but these turned out not to be very useful, because the area where the forehead meets the hair varies too much across subjects.

7.2 Manual landmarking uncertainty

There is ambiguity in the position of most landmarks, even with the most careful manual landmarking. Figure 7.3 shows some examples of this ambiguity.

This inherent noise, purely due to manual landmarking uncertainty, sets a lower limit

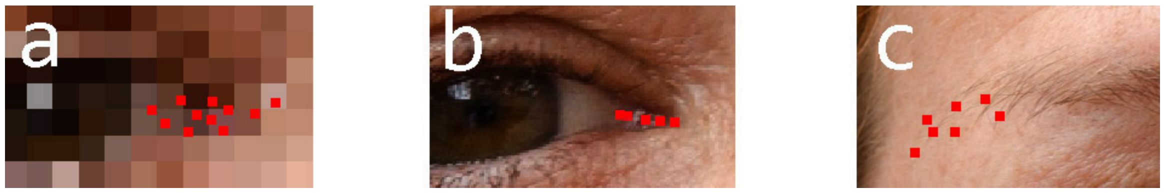


Figure 7.3: *Inherent uncertainty in manual landmarking.*

(a) Where is the corner of the eye in this low resolution image?

(b) The corner can be better estimated in a higher resolution version of the same face. Ideally, manual landmarking should be done on high resolution images (whose resolution can then be reduced if necessary). Even so, different people will make different estimates of the position the corner of the eye.

(c) Where is the outer edge of the eyebrow? It's impossible to say.

The inter-pupil distance is 34 pixels for the face in the left image (the same distance as the smallest BioID faces), and 380 pixels in the other two images.

for me17 values and similar measures. Figure 8.19 on page 114 shows an example face where the automatic landmarks are as good as the manual landmarks, if not better. The me17 of 0.195 on this example face is due to manual landmark uncertainty as much as it's due to automatic landmarking.

In the next few sections we perform some tests to investigate manual landmarking uncertainty. Also of interest is the paper by Smith and Zhang [106], which discusses transferring and combining manual landmarks from different datasets.

7.2.1 Re-marked BioID and MUCT points

To approximately quantize the manual landmarking uncertainty, we manually re-marked four landmarks on the first 600 faces in the BioID and MUCT datasets. The objective was to see how the new landmarks differed from the original manual landmarks [18,81]. (This study is similar to a study in the MUCT paper [81] which used a smaller sample.)

The four re-marked landmarks are the left eye pupil, the outer corner of the left eye,

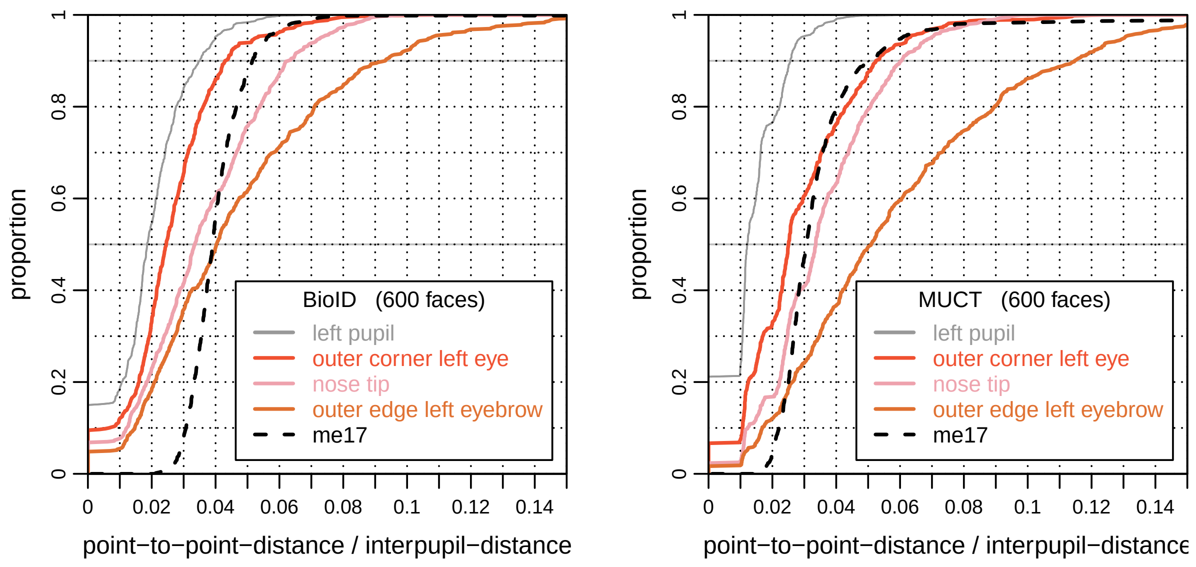


Figure 7.4: *Manual landmarking uncertainty on the BioID and MUCT data.*

The solid curves show distances between the original manual landmarks and manually re-marked points.

The dashed curve is for reference. It shows me17s on the same 600 faces with landmarks automatically located with the Stasm HAT model described in Chapter 8 .

For the 600 BioID faces, the mean inter-pupil distance is 56.4 pixels. Thus a distance of $1/56.4 =$ ***0.018*** *corresponds to one pixel.*

For the 600 MUCT faces, the mean inter-pupil distance is 88.8 pixels. Thus a distance of $1/88.8 =$ ***0.011*** *corresponds to one pixel.*

the nose tip, and the outer edge of the left eyebrow. These four points were chosen because the inherent uncertainty of manual landmarking for these points ranges from small (the pupil) to big (edge of the eyebrow). (See Figure 9.17 on page 134, which shows the relative error of automatic landmarks.)

We measured the distance between each of these re-marked landmarks and the original manual landmarks, and divided by the inter-pupil distance. Figure 7.4 shows the distributions of these normalized distances. The little horizontal line at the start of the curves is due to pixelation—the smallest non-zero distance between points is 1 pixel (which gets divided by the inter-pupil distance).

The mean inter-pupil distance is 56.4 pixels for these 600 BioID images, thus a distance of $1/56.4 = 0.018$ corresponds to one pixel. The distributions indicate that for faces like those in the BioID set, one can expect median manual landmarking uncertainties of perhaps 3% of the inter-pupil distance, or about one and half pixels. The lowest possible value for an me17 on this data seems to be about 0.02.

7.2.2 Re-marked XM2VTS points and the iBUG data

To further investigate landmarking uncertainty, we also looked at the reference landmarks generated by the iBUG group. Section 9.6.3 on page 127 gives details on these landmarks; here we just illustrate the difference between the iBUG landmarks [100] and the original XM2VTS landmarks [18, 72], using the outer corner of the left eye as an example. We mention that the MUCT eye-corner definitions are similar to the XM2VTS definitions.

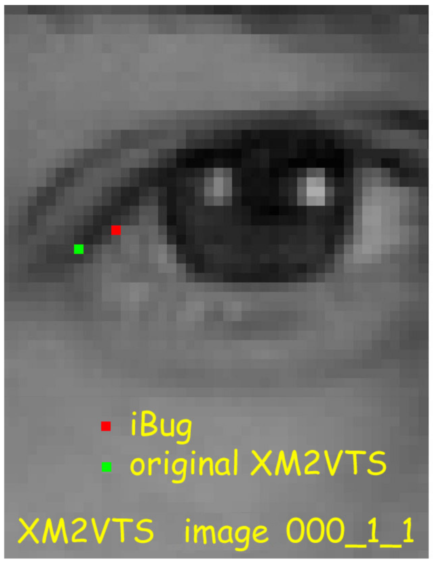
Figure 7.5 shows this difference. Also to be noted for this landmark is that the mean horizontal distance between the iBUG and XM2VTS landmark is 2.8% and the mean vertical distance is 1.4% (of the distance between the outer eye-corners). This discrepancy in the mean positions implies that the iBUG landmark positions are biased with respect to the original XM2VTS landmarks—the iBUG eye-corner landmark tends to be above and to the right of the XM2VTS landmark.

As can be seen from the figure, the differences between the reference landmark positions can be quite considerable. This doesn't mean per se that one set is better than the other. It's just that different landmarkers have different ideas of what constitutes the "corner" of the eye (which introduces bias), and there is also the inherent uncertainty in manual landmarking (which introduces noise). An automatic landmarker trained on one set of landmarks will suffer an apparent but artificial performance hit if measured against a set landmarked with different biases.

7.2.3 Discussion on the landmarking uncertainty experiments

The above study isn't comprehensive, but is enough to broadly characterize the nature of manual landmarking uncertainty. The graphs could be extended in interesting ways. For example, we could also compare re-marked landmarks on the *sides* of the face, where inherent uncertainty is even greater. We could also compare re-marked landmarks on some of the popular in-the-wild databases (like the AFW [132], and LFPW [7] data). The low quality and resolution of many faces in these databases means the manual landmarking uncertainty would be greater than in the data graphed in Figures 7.4 and 7.5.

Summarizing, the figures in this section show that there is more uncertainty in manual landmarking than seems to be widely assumed. There is both *bias* (caused by different definitions of landmarks in different datasets, and by biases introduced by different human landmarkers) and *noise* (caused by the difficulty of manually pinpointing landmarks). Manual landmarks aren't the golden standard we'd like them to be.



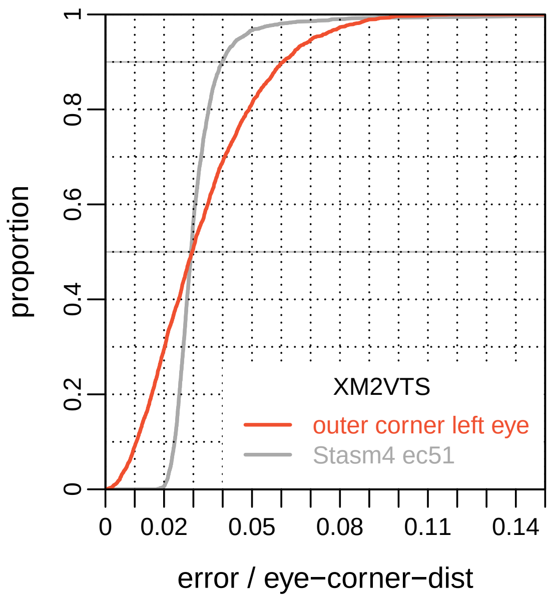


Figure 7.5: *Reference landmarking uncertainty: differences between the iBUG reference landmarks and the original XM2VTS reference landmarks.*

For the outer corner of the left eye, the red curve shows the distance between the iBUG reference landmark and the original XM2VTS reference landmark, divided by the distance between the outer eye-corners.

The mean inter-pupil distance is 101 pixels for the 2360 XM2VTS faces, thus a distance of $1/101 =$ ***0.01*** *corresponds to one pixel.*

Similar curves can be drawn for the other landmarks. iBUG reference landmarks aren't available for the MUCT data (used for training the Stasm models), but it's likely that the curves would be similar.

The gray curve is for reference. It shows ec51 fits w.r.t. the iBUG landmarks of our multiview model on the XM2VTS faces. It's the same as the gray curve on the left side of Figure 9.12. The intersection of the curves at the median is coincidental.

7.3 The “ideal” face database

The ideal face database depends on the applications for which the automatic landmarker is intended. Training and test sets should contain representative data.

The question of what is “representative” is tricky, but as mentioned in Section 1.2, this thesis focuses on the medium- to high-resolution faces typically processed by applications like automatic or interactive retouchers. Faces processed by these applications are typically of good quality, or at least not of bad quality, and blurred or low-resolution faces are unusual.

For our purposes the ideal data for building and testing the models should match the requirements of these applications, with face dimensions and expressions roughly matching those seen by the applications. Tuning a model for optimal results only on difficult faces may adversely affect its performance on the bulk of the faces. To use an analogy, in general one shouldn’t evaluate a high school by evaluating the performance of only the most difficult students. Therefore the evaluation data shouldn’t have a large proportion of blurred or low resolution faces (eye-mouth distances should be at least 80 pixels, and preferably much higher). The proportion of faces with extreme expressions should be small, although smiles are important.

Also making blurred or low-resolution faces undesirable is the fact that the position of landmarks is ambiguous in such faces (Figure 7.6). Testing against uncertain manual landmarks introduces unnecessary randomness into the test results. The cumulative distribution curves by which we compare models become less trustworthy.

The tuning and test sets should contain a sufficient number of faces to counteract overfitting (optimizing for the test set) and to bound sampling variation, especially in the worst 5 or 10% of fits. We would recommend at least 600 faces, preferably more, so there are at least 30 faces in the worst 5%.

As a practical matter, it’s best if there is only one face per image. If there is more than one face in the original image, for each face the image should be reproduced with a unique file name, with all faces blurred or masked except the face in question (Figure 7.7). This alleviates issues with matching the multiple faces detected in an image with reference faces. Testing is easier when each manually landmarked reference face is uniquely tagged with the file name. This is especially necessary because different face detectors return multiple faces in different sequences. (It helps if the detector returns multiple faces in a predictable order, such as left-to-right and from top-to-bottom within that.)

Although we have seen proprietary commercial datasets that do meet these requirements, no suitable commonly accepted public standard test set has yet emerged. The BioID data [2] is well known, but its faces are frontal and were photographed in an uniform office setting. For variety, faces should be collected from photographs taken “in the wild”. However, for our needs most of the in-the-wild datasets that have become popular in papers in the last few years have too many low-quality and low-resolution faces. For example, the median eye-mouth distances in the AFLW [60], AFW [132], iBUG [100], and LFPW [7] test sets are only 74, 90, 71, and 70 pixels respectively, with

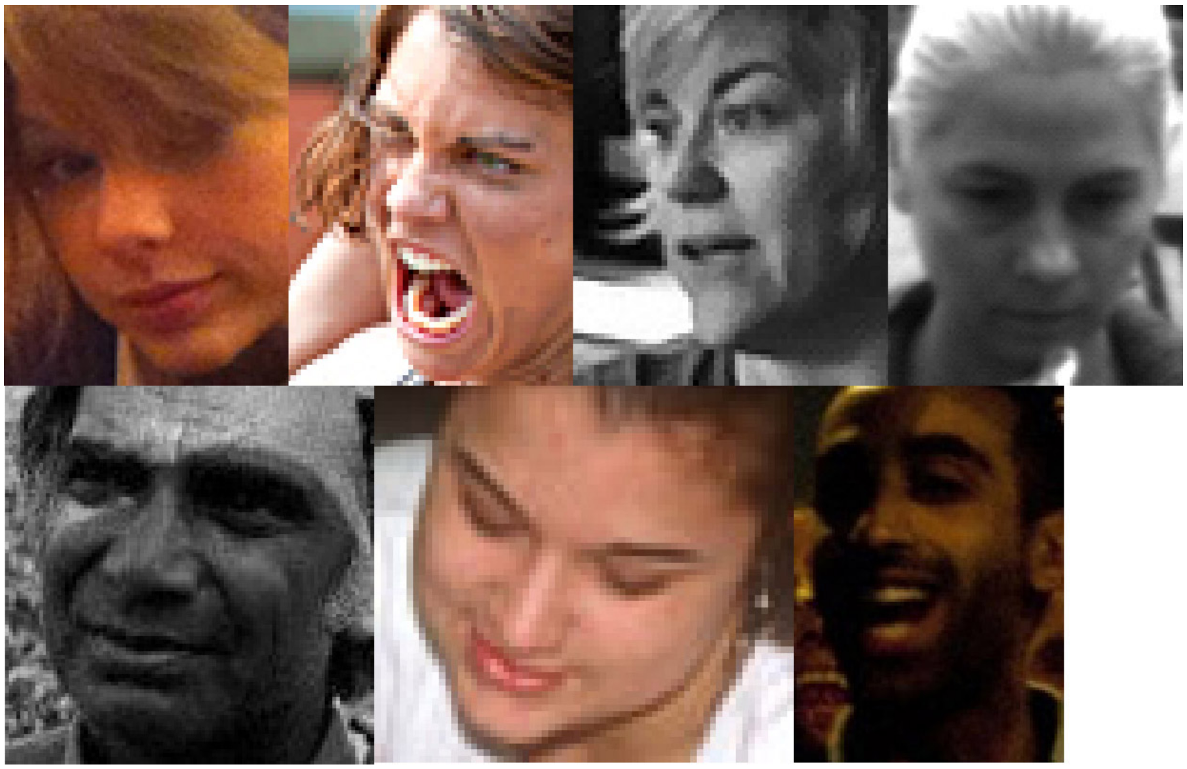


Figure 7.6: *Low-quality images of faces.*
Where is the position of the left eye corner (for example) in these faces? The manual landmarking uncertainty is high relative to the face size.
Your PDF viewer or printer may smooth out imaging artifacts such as pixelation in these images. These artifacts may be easier to see if the image is expanded in the PDF viewer.
The images are from the 2013 300-W Challenge test set (Section 9.6.3). Figure 9.10 on page 126 shows landmarks fitted by the multiview Stasm model on these faces.

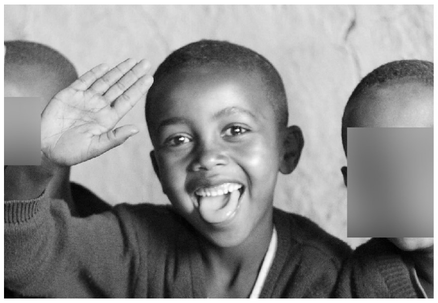


Figure 7.7: *Extra faces in the image should be masked so there is one face per image file name. The face detector returns the single face.*
(This is image 3253427659_1 from the Helen test set [61], with the left and right faces masked.)

many of the faces in all these sets significantly smaller than those numbers.

Although the MUCT data is used for training models in this thesis, it too falls somewhat short of being ideal, because the faces weren’t photographed in the wild and the median eye-mouth distance of 111 pixels is too low for optimal manual landmarking. The faces of the Helen data [61] are perhaps closest to what we would consider ideal (they have a median eye-mouth distance of 222 pixels), although the designated test subset is a bit small (330 faces) and one could quibble over the definitions and quality of the manual landmarks.

Chapter 8

HAT descriptors

To summarize our presentation of descriptors so far, a descriptor for template matching captures some distinguishing quality of the image feature in question. For example, a descriptor might capture the nature of the outer corner of the left eye. Chapter 5 discussed 1D and 2D gradient descriptors, which take the form of a one- or two-dimensional array with the same dimensions as the image region they cover; each element of the array is a measure of the gradient at a pixel.

The previous chapters described existing ASM techniques (although those chapters also presented a few novel ideas here and there). Now in this chapter we extend the state of the art with Histogram Array Transform (HAT) descriptors. These descriptors are better than gradient descriptors at matching only the desired feature (they have better sensitivity) while avoiding the area around the feature (they have better specificity).

HAT descriptors are essentially the local image descriptors used as part of the SIFT algorithm (Lowe [68]). Actually they are also similar to other orientation histograms such as R-HOGs (Dalal and Triggs [29]). But since SIFT descriptors are the most well known, we present HAT descriptors as a variant of SIFT descriptors. SIFT descriptors in turn are derived from earlier techniques using histograms of image features, such as those in Belongie et al. [8], Freeman and Roth [39], and Freeman [38].

8.1 Related Work

Numerous proposals have been made to replace the 1D and 2D gradient descriptors described in Chapter 5. We mention just a few SIFT based schemes, emphasizing earlier papers. We used the open-source SIFT code of Rob Hess as a reference [47].

D. Zhou et al. [129] use SIFT descriptors with Mahalanobis distances. They report improved eye and mouth positions on the FGRCv2.0 database. Z. Li et al. [63] use SIFT descriptors with GentleBoost. Kanaujia et al. [54] use SIFT descriptors with multiple shape models clustered on different face poses. Belhumeur et al. [7] use SIFT descriptors with SVMs and RANSAC. Y. Shi et al. [104] use SIFT descriptors in hierarchical ASM models for medical images. Xiong and de la Torre [120] use SIFT descriptors in their influential Supervised Descent Method paper. SIFT descriptors for face data are also used in e.g. Querini et al. [94], Rattani et al. [95], J. Zhang et al. [126], and L. Zhang et al. [127].

8.2 HAT descriptors

This section describes HAT descriptors. Later sections will do some analysis and comparisons.

8.2.1 Gradient magnitude and orientation

The *gradient magnitude* at pixel coordinates x,y is a nonnegative scalar defined as

$$m_{x,y} \;=\; \sqrt{(\boldsymbol{i}_{x,y} \;-\; \boldsymbol{i}_{x-1,y})^2 \;+\; (\boldsymbol{i}_{x,y} \;-\; \boldsymbol{i}_{x,y-1})^2}, \tag{8.1}$$

where $\boldsymbol{i}$ is a two-dimensional image of gray pixels $0 \ldots 255$. This measures the gradient at the pixel relative to the pixels above and to the left of it. Other ways of defining the gradient magnitude are also common [42]. Lowe's SIFT paper [68] for example uses a more symmetrical formula

$$m_{x,y} \;=\; \sqrt{(\boldsymbol{i}_{x+1,y} \;-\; \boldsymbol{i}_{x-1,y})^2 \;+\; (\boldsymbol{i}_{x,y+1} \;-\; \boldsymbol{i}_{x,y-1})^2}, \tag{8.2}$$

but empirically in the context of ASMs, landmark accuracies with Definition 8.1 are just as good. It has the minor advantages that it actually uses the pixel at the coordinate in question and it "wastes" one less row and column of pixels at the image boundary.

The *gradient orientation* at x,y is

$$\theta_{x,y} \;=\; \text{atan2}(\; \boldsymbol{i}_{x,y} \;-\; \boldsymbol{i}_{x,y-1},\; \boldsymbol{i}_{x,y} \;-\; \boldsymbol{i}_{x-1,y}). \tag{8.3}$$

The function atan2 is used instead of the usual mathematical function arctangent so the signs of the arguments determine the quadrant of the result in the 0° to 360° circle [118]. With IEEE arithmetic, atan2 is defined to return sensible values if the first and/or the second argument is zero.

We use degrees here, but radians can be used instead—it doesn't matter as long the units are used consistently. In our software 0° is due east, degrees are always in the range 0 (inclusive) to 360 (exclusive), and degrees increase anticlockwise. Figure 8.1 shows some examples of gradients. (The anticlockwise rotation arises naturally using the standard C++ functions because y increases as we go *down* the image.)

As is commonly done, we slightly abuse terminology and talk about a "pixel's gradient", but pedantically a single pixel can't have a gradient: the gradient is a property of the area at and around the pixel.

Just as for 2D gradient descriptors, for efficiency we precalculate the gradients and their magnitudes for the region of interest around the face. This must done at each pyramid level, and is faster than calculating the gradients each time a HAT descriptor must be generated.

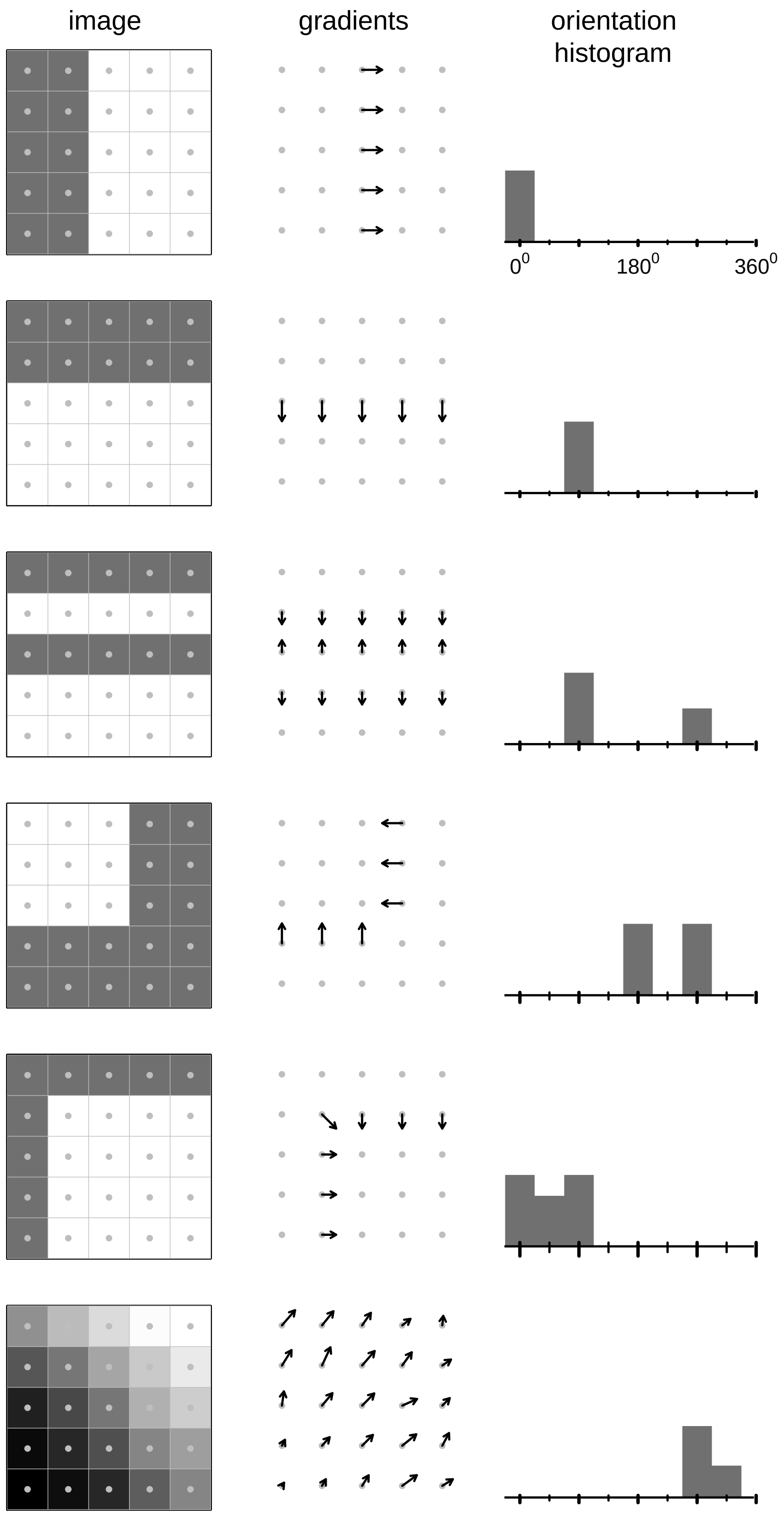


Figure 8.1: *Example orientation histograms.*

8.2.2 Orientation histograms

An orientation histogram is generated from an image, or, in the ASM context from a rectangular subregion of the image. The histogram consists of a number of bins covering orientations from 0° to 360°. Each bin is a nonnegative scalar that measures the "prevalence" of the corresponding orientation in the image region of interest. If the number of bins is say 8, then each bin covers 360/8 = 45°. Bin 0 is for orientations of 0° (inclusive) to 45° (exclusive), bin 1 is for $45 \ldots 90^{\circ}$, and so on up to bin 7 for $315 \ldots 360^{\circ}$.

To generate the orientation histogram for an image, we first clear all bins to zero. Then for each pixel in the image we select the bin corresponding to the pixel's gradient orientation, and increment the bin's value by the pixel's gradient magnitude. Figure 8.1 shows some example histograms. Note that in the usual sense of the word, a histogram merely keeps a count of the number of objects that fall into a bin, but here we also include the object's magnitude.

8.2.3 Histogram smoothing

If the image changes slightly its descriptor should also change just slightly. This property helps make descriptor matching robust—small changes to the target image shouldn't cause large changes to its descriptors.

However, in the algorithm for histograms just described, a small rotation to the image may cause a large change to the descriptor, as the orientations near bin boundaries jump abruptly from one bin to another.

We thus modify the algorithm slightly. Instead of assigning each pixel to a single bin, we use linear interpolation to apportion its gradient magnitude across two bins (wrapping around at 360°). If a pixel's orientation falls on the boundary between bins, its magnitude gets shared equally between both. Only when the pixel's orientation falls exactly in the middle of a bin does it get assigned to a single bin.

8.2.4 Arrays of histograms

The orientation histograms just described could be considered to be descriptors in their own right. However we get better results if we divide the image region of interest into a coarse grid, and generate a histogram for each cell in the grid. Each cell covers a portion of the rectangular image region (Figure 8.2). It is this array of histograms that forms the Histogram Array Transform (HAT) descriptor.

The coarse grid provides a degree of spatial invariance, because the exact position of an image feature in the area covered by the cell doesn't matter—all pixels in that area map to the same cell.

Remember that we rotate the image so the face is upright before the ASM search begins (Section 4.3), and scale the face to a fixed eye-mouth distance (Section 4.6). This makes the area of the face covered by the descriptor the same as in the training faces, up to variations in pose and face shape.

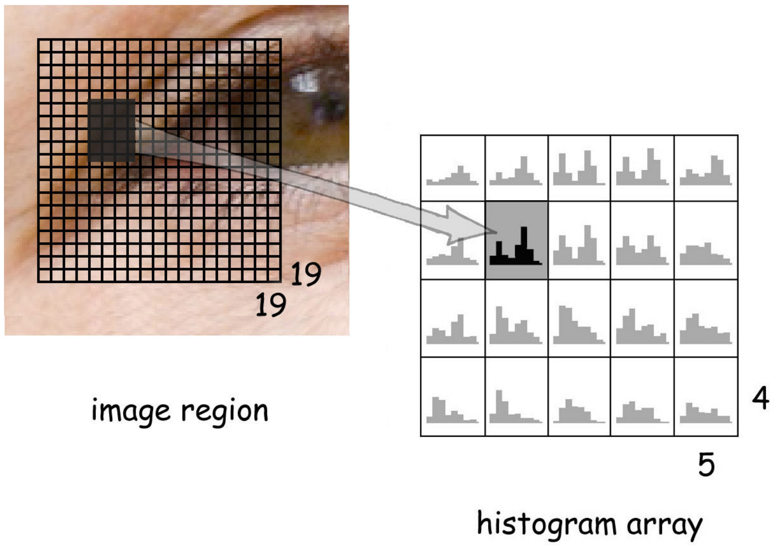


Figure 8.2: *A HAT descriptor is an array of histograms.*

An area in the image maps to a cell in the array: the histogram is for that area.

In this example, a 19×19 array of pixels is covered by a 4×5 histogram array. (These are the dimensions used by Stasm.)

8.2.5 Gaussian weighting

In order to give pixels at the center of the image region more importance than pixels further away, we apply two-dimensional Gaussian smoothing to the gradient magnitudes (Figure 8.3). This is done before forming the histograms. Specifically, a pixel's gradient magnitude is scaled by

$$\exp\Big(-\,const \times dist^2\Big) \tag{8.4}$$

$$= \exp\Big(-\,const \times \frac{x^2+y^2}{(\frac{w}{2})^2+(\frac{h}{2})^2}\Big)\,, \tag{8.5}$$

where *const* is a constant determined empirically during model tuning and *dist* is the normalized Euclidean distance $0\ldots1$ of the pixel from the center of the grid. The second equation shows how *dist* is calculated: x and y are the pixel's horizontal and vertical distance in pixels from the center, and w and h are the image region width and height in pixels. The denominator is just a normalizing constant so the weight at the edge of the region is independent of the size of the region.

Empirically, the precise value of *const* isn't important. Stasm uses a value of 2, which weights magnitudes at the edge of the region by $\exp(-2) = 0.135$.

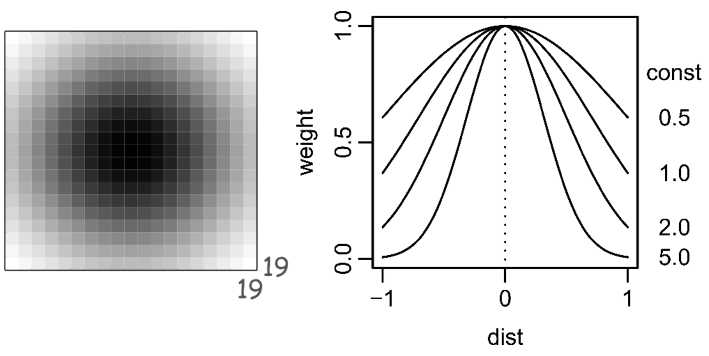


Figure 8.3:

left *A Gaussian weighting mask.*

right *Weighting for different values of the constant in Equation 8.4.*

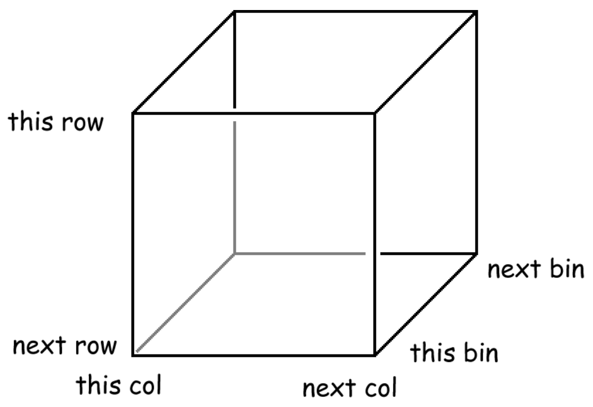


Figure 8.4: *Trilinear interpolation. Each corner of the cube represents a histogram bin.*

A point in the cube is apportioned among the eight corners of the cube.

8.2.6 Trilinear interpolation

In the setup described above, a small change in the position of the image region may cause a large change to the descriptor, as the mapping for pixels near cell boundaries jumps from one cell to another. Using similar reasoning to the histogram smoothing described previously, we therefore linearly interpolate, or "smear out", a pixel's contribution across adjacent cells.

Together with the histogram smoothing, we now have a total of three interpolations (horizontally and vertically across cells, and across bins within a histogram), or *trilinear* interpolation (Figure 8.4). Although we (and Lowe) use the term "interpolation" here, strictly speaking this is *inverse* interpolation because we are spreading out a point among several points, not combining several points into one point. Calculation details may be found in the function `TrilinearAccumulate` in the Stasm C++ code. Trilinear interpolation is computationally demanding (i.e., it's slow), but tests verify that all three interpolations are necessary for optimum fits in ASMs (Section 8.6).

8.2.7 Normalizing the descriptor

Finally, we take the square root of each element of the descriptor vector and normalize the resulting vector to unit length. We do this by dividing each element by the vector's Euclidean length. If all elements are zero, this step is skipped.

Taking the square root reduces the effect of extreme values such as the bright spots on the nose tip in Figure 2.5 on page 20. SIFT descriptors in their original form use a slightly more complicated scheme for reducing the effect of gradient extremes (based on a threshold), but for ASMs we have found on our data that taking square roots is effective, and slightly quicker. We used sigmoids for the same purpose with 2D gradient descriptors (Section 5.6), but here a simple power transform like the square root can be used because there aren't any negative elements.

Normalizing to unit length increases invariance to changes in image contrast. Invariance to the absolute illumination level is already achieved because we are using gradients, not gray levels. Notice that we don't normalize each histogram independently—thus relative differences in gradients across cells are preserved.

8.3 HAT descriptors contrasted to SIFT descriptors

HAT descriptors are essentially the local image descriptors used in the SIFT algorithm. This section describes how they differ.

A crucial consequence of these differences is that HAT descriptors can be generated much more quickly than the descriptors in the standard SIFT framework. Speed is important because to locate the landmarks in a single face the ASM needs to evaluate something like 20 thousand descriptors. (Using typical parameters: 77 landmarks $\times$ 4 pyramid levels $\times$ 4 model iterations $\times$ 4 $\times$ 4 search grid = 20k.)

8.3.1 No automatic selection of keypoints

The framework or context in which HAT descriptors are used (i.e., refining the position of landmarks) differs considerably from that in which SIFT descriptors are used (i.e., discovering and matching keypoints [68]). Readers familiar with SIFT will be aware that the overview of descriptors in Section 8.2 doesn't mention this overall SIFT framework.

This framework first does a time consuming scale-space analysis of the image to automatically discover which parts of the image are keypoints. For example, the corner of a roof could be automatically determined to be a keypoint, making it possible to locate the same corner in a different image of the same building, even if photographed with a different resolution and from a different viewpoint.

In contrast, in ASMs the keypoints are predetermined—they are the facial landmarks. We don't search the face to discover which parts of the face are of interest. We already know which parts of the face we are searching for.

8.3.2 No automatic determination of scale

This SIFT framework mentioned above also determines the intrinsic scale of each of the keypoints. This is achieved with a sub-octave Difference-Of-Gaussians pyramid so an image feature can be analyzed simultaneously at multiple scales.

In contrast, in ASMs the face is prescaled to a constant size before the ASM search begins (Section 4.6), thus we don't need SIFT's automatic determination of scale. The ASM descriptor patches are a fixed size, and we use a simple 4 octave image pyramid with the search proceeding one pyramid level at a time, from coarse to fine resolution.

8.3.3 No patch orientation

In the SIFT framework once the intrinsic scale of a descriptor has been determined there is an additional preprocessing step (omitted in Section 8.2): the local structure of the image around the point of interest is analyzed to determine the predominant gradient orientation, and the descriptor is formed from an image patch aligned to this direction. The orientation is computed by looking for peaks in an orientation histogram (independently of the histograms in the descriptor array).

However, in our software before the search begins we rotate the image so the face is upright (Section 4.3). Thus the automatic orientation described in the previous paragraph is unnecessary, and in our experiments actually reduced automatic landmarking accuracy. We also found it unnecessary to orientate the descriptors to the shape boundary, as is done for 1D gradient descriptors and for the descriptors in some other automatic landmarking techniques (e.g. Kanaujia and Metaxas [54]).

Some rotational variation will still remain—because not every face is the same, and the eye detectors sometime give false positives on the eyebrows or fail to find eyes, causing incorrect orientation of the start shape on the image face (Section 4.3)—and so we must also rely on the intrinsic invariance properties of the descriptors. The trilinear interpolation helps here.

8.3.4 Precalculation of the mapping from patch to histogram array

When generating a SIFT or HAT descriptor, each pixel in the patch must be mapped to a position in the array of histograms. Calculating this mapping takes time, especially for scaled and rotated patches, and must be done each time we generate a descriptor.

But HATs don't use such scaling or rotation, and the mapping therefore depends only on the histogram array size (4×5) and the patch size (19×19 pixels), which remain constant through the ASM search. Thus we can precalculate the mapping indices just once for all descriptors (or once per pyramid level if we use different patch sizes at different pyramid levels). In our software this precalculation reduced total ASM search times by 40%.

8.3.5 One-to-many versus many-to-one matching, and nearest-neighbors

In a typical application envisaged in Lowe's SIFT paper, keypoints from a new image are matched against a library of keypoints that were discovered in a previously-seen image or images. As we discover a keypoint in the new image, we find its nearest neighbor in the library—this is one-to-many matching. Use of a nearest-neighbor approach is natural in this setting. After repeating this process for each keypoint in the new image, we can then use the keypoints to recognize objects or stitch the images together, for example.

The ASM search context is different. The ASM extracts descriptors around the tentative position of the landmark. It matches these against a model descriptor, looking for the best fit—this is many-to-one matching.

Use of a nearest-neighbors approach for matching descriptors in the ASM setting may be possible in theory, but would probably be too slow and require too much memory. A straightforward implementation would store example descriptors from all the training images. If we stored only the positive examples for say 6000 training images (Section 8.9) we would need to store well over a million descriptors, requiring over

400 megabytes. (Using typical parameters: 6000 training-images $\times$ 77 landmarks $\times$ 3 pyramid-levels $\times$ 160 elements-per-descriptor $\times$ say 2 bytes-per-element = 443M.) For optimum results using a nearest-neighbors approach, we would likely also have store negative examples, and this would require several times the above amount of memory. It may be possible to somewhat reduce these requirements by using nearest-neighbor condensation techniques.

8.4 Descriptor caching

As we iterate the search around a point during an ASM search, we often revisit the same image coordinates. Additionally, the search areas of nearby landmarks overlap, especially at coarse pyramid levels, and this also causes the same coordinates to be revisited. It thus makes sense to save the HAT descriptor associated with a coordinate for reuse, since generating a descriptor is fairly time consuming.

In our software a hash table [26] is used to store descriptors (Figure 8.5). The table is cleared before the search at each pyramid level. The hash key is a 32-bit word consisting of the packed x and y point coordinates. As is the nature of hash tables, the time to access a descriptor is constant, irrespective of the number of saved descriptors, except in rare instances, if ever, where there is a hash key collision or container overflow. A trial alternative implementation that stored the descriptors in an array directly indexed on the pixel coordinates was slower, possibly because the memory required exceeds processor hardware cache limits.

In practice we get a cache hit rate of over 65% (meaning that over 65% of the requests for a descriptor can be satisfied by using a saved descriptor). Caching reduces search time by a further 70%.

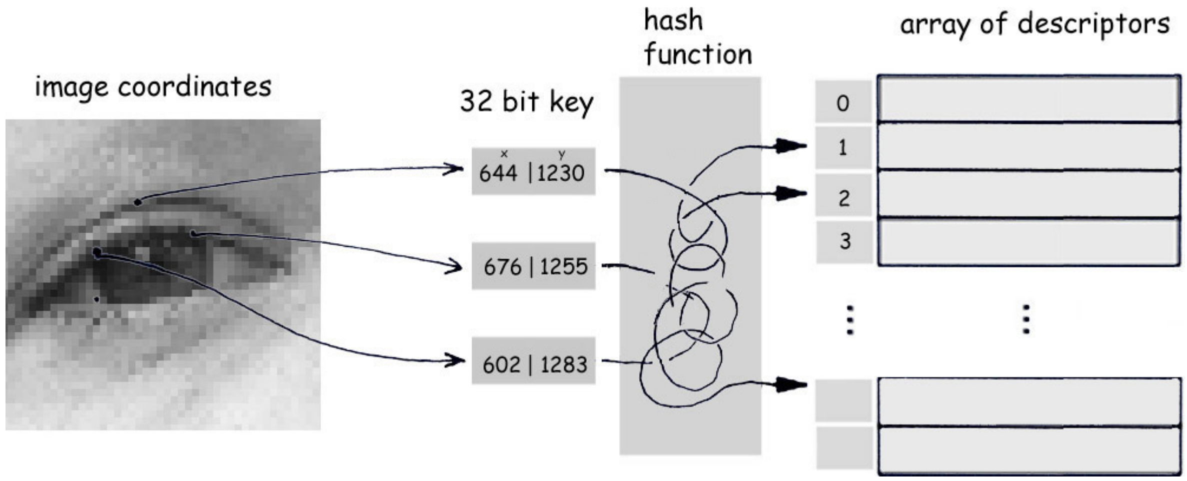


Figure 8.5: *A hash table is used to save HAT descriptors for reuse. The hash key consists of the image x,y coordinates packed into a 32 bit word.*

This diagram is based on a Wikipedia diagram by Jorge Stolfi: `https://commons.wikimedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg`.

8.5 Partial use of 1D descriptors

Just as for 2D gradient descriptors (Section 5.6), it turns out that we get slightly better fits if HAT descriptors aren't used for all landmarks. More precisely, we use 1D gradient descriptors

(i) on the jaw and forehead landmarks at all pyramid levels,

(ii) on all landmarks at the coarsest pyramid level (level three, one-eighth full width),

and HAT descriptors everywhere else.

This partitioning of landmarks between 1D and HAT descriptors was arrived at after some basic experiments. Similarly to the analysis in Table 5.15 in [75], we evaluated a couple of dozen different partitions based on pyramid levels and sensible subsets of facial features (such as the sides of the face, the forehead, the eyes, and so on). A more exhaustive analysis to determine the optimal partition wasn't done, but would likely result in only small improvements over the above partition, if any.

It's interesting that humble 1D gradient descriptors give better overall results than HAT descriptors for the exterior landmarks. This merits more investigation. The landmarks on the edge of the face include more random background texture than the interior landmarks. They are on the whole more difficult to place than interior landmarks, both for human and automatic landmarkers (Figure 9.17). Other researchers such as E. Zhou et al. [130] have also found it best to use quite different models for the exterior and interior landmarks. Perhaps HAT descriptors with dimensions specifically tuned for exterior points (and/or rotated to align with the face edge) could be coaxed to outperform 1D gradients for the exterior points.

8.6 HAT parameter tuning

What are the best values for the various descriptor parameters when HATs are used in ASMs? We have found that good results can be obtained with 4×5 histogram grids and 8 orientation bins. The entire descriptor thus has $4 \times 5 \times 8 = 160$ bins. We use an 19×19 image region (Figure 8.6) and a value of 2 for the Gaussian weighting constant (Equation 8.4).

With 19×19 image regions, for certain landmarks at coarse pyramid levels the edge of the face may be reached (i.e., part of the descriptor will be off the face). This is a potential problem if we train on images with uniform backgrounds (like the MUCT data), because the model will learn to expect a uniform background, which won't be the case on the more varied images used when searching for landmarks. We haven't done a detailed analysis, but in practice this doesn't seem to be an issue, possibly because of the Gaussian downweighting at the edge of the patch.

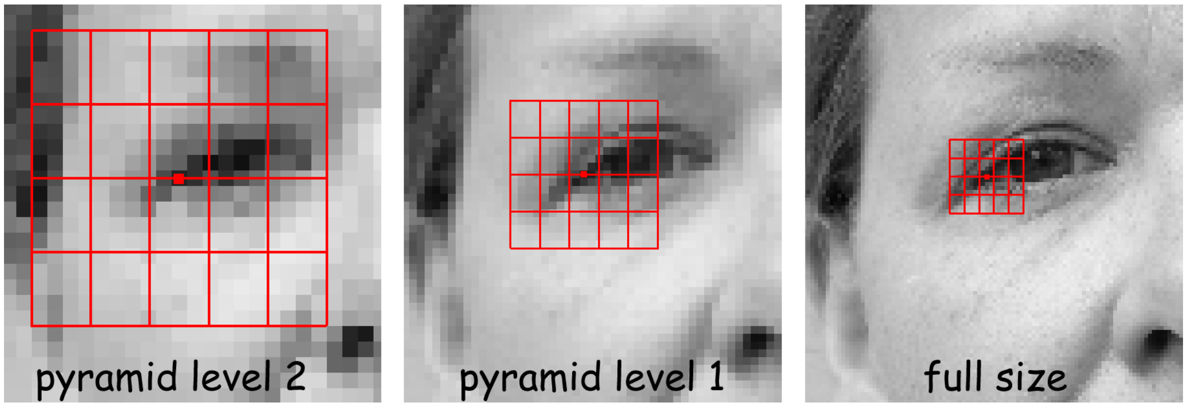


Figure 8.6: *The 19×19 image region covered by the HAT descriptors at different pyramid levels, after scaling the face to an eye-mouth distance of 110 pixels.*

8.6.1 HAT parameters

The parameters above were determined empirically by testing against a parameter-selection set (Section C.2). This internal set consisted of medium and high resolution faces gathered from the internet. The parameters were determined using the complete ASM at all four pyramid levels.

This section now examines the parameters for HAT descriptors in isolation, without the shape model. For reproducibility, results are shown on the XM2VTS set (but we didn't actually use the XM2VTS results for parameter selection—the results are similar but not identical to the results on the internal parameter-selection set we actually used). See also Dalal and Triggs [29] for studies of the effect of different parameter values on histogram grids.

Figure 8.7 shows the effect of changing the histogram grid width and number of cells. A 4×4 grid with 10 bins gave the best results with this setup. These parameters will vary

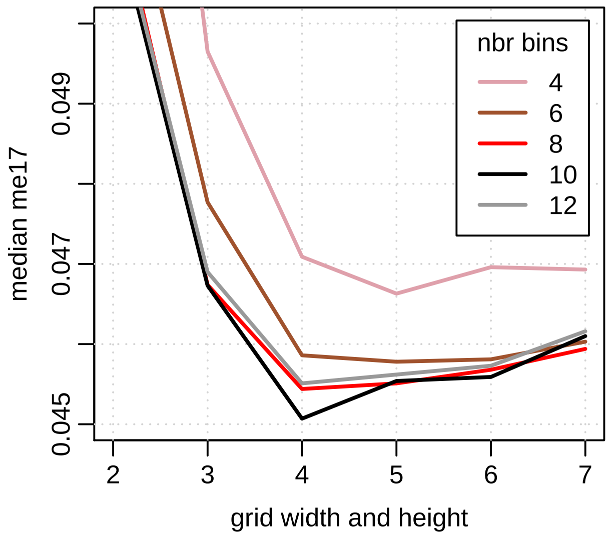


Figure 8.7:

Effect of histogram cell grid size and number of bins.

Measured at pyramid level 0 with a 19×19 image patch. The plot shows the median `me17` *on the XM2VTS data before correction by the shape model.*

somewhat with different data. In the original SIFT paper, Lowe found that a 4×4 grid with 8 bins was optimal. The effect of different parameters is less than a first glance at Figure 8.7 might indicate—note the fairly small range of the vertical axis.

Details on the test setup: For each point in the graph, an ASM was trained on the MUCT `abde` faces. (This is the same data that was used for training the standard Stasm frontal model, Section 8.9.) We then ran this ASM on the XM2VTS data (all 2360 images) on the final pyramid level (full scale), but to isolate the effect of the HAT parameters, excluded correction by the shape model. We used MARS models (Section 8.7) to match descriptors. These were trained anew for each point in the graph. We plot the median rather than the mean me17 (Section 6.2.1) because the distribution of me17s is highly skewed.

Figure 8.8 shows the effect of changing the size of the image patch used for the descriptor. The optimum value varies across pyramid levels (lower resolutions prefer smaller patches), but we found that using a 19 × 19 grid at all pyramid levels gave best overall results. We tried using different patch sizes at different pyramid levels, but found

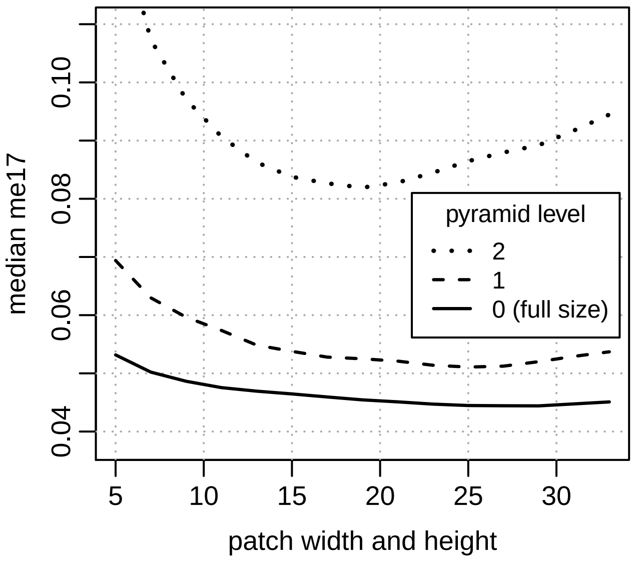


Figure 8.8: *Effect of patch size.*

Measured on a 4 × 4 HAT histogram grid with 8 bins. The plot shows the median `me17` *on the XM2VTS data before correction by the shape model.*

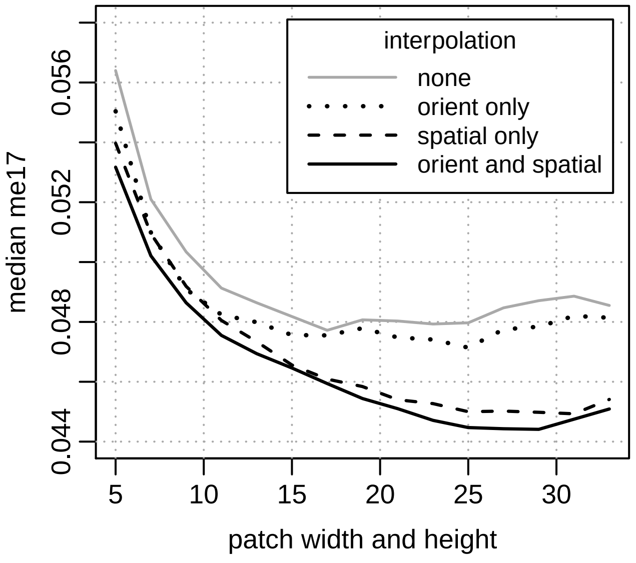


Figure 8.9: *Effect of trilinear interpolation (page 95).*

Measured at pyramid level 0 with the same HAT parameters as Figure 8.8.

The lowest curve is the same as the lowest curve in Figure 8.8, although the scale of the vertical axis is different.

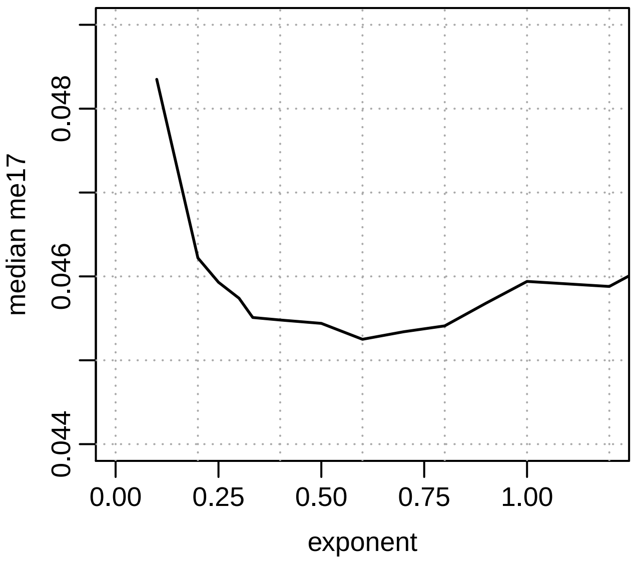


Figure 8.10: *Effect of the exponent used for the power transform before normalizing the HAT descriptor (page 95).*

A value of 0.5 (square root) was chosen for the final model.

Measured at pyramid level 0 with the same HAT parameters as Figure 8.8.

that unnecessary for optimum fits, once the shape model has been applied. We didn't evaluate rectangular (non-square) patches.

Figure 8.9 shows that trilinear interpolation (i.e. orientation and spatial interpolation) is necessary for optimum fits.

As discussed on page 95, before the HAT descriptor is normalized to unit length we take its square root. Figure 8.10 show fit results over a range of exponents other than square roots (with 0.5 being the square root chosen for the final model).

We don't hope for definitive values for these parameters—just values that work well enough on realistic data. Different face data will give different results, and no set of face data can be called definitive. For example, we have discovered empirically that the HAT model would give better results on the BioID data (but not on more varied faces) if it were to use 10 instead of 8 histogram bins. Furthermore, the use of the median as a test criterion could be questioned. Perhaps it's better to use the 90th or 95th percentile, with the argument that reducing bad fits is more important than excellent median fits, especially if the median fits are already close to the manual landmarking noise floor. However in practice the parameters selected using these alternative criteria would be very close, if not identical.

8.7 Descriptor matching methods

Once we have the descriptor at an image location, we need a measure of how well the descriptor matches the facial feature of interest. This section reviews some techniques for descriptor matching, leading to the MARS models used in our software.

As in the classical ASM, at each landmark we look only for the single best match in the search region, and so need be concerned only with the relative quality of matches.

8.7.1 Euclidean distances

We want to measure how well the descriptor sampled from the image matches the facial feature of interest. A basic technique is to take the Euclidean (root-mean-square) distance between the descriptor sampled from the image and the mean training descriptor for the landmark. This approach gives equal weight to every histogram bin—but that is inappropriate because not all areas of the region covered by the descriptor are equally important. Furthermore, a high variance bin is likely to add more noise than information. Looking at it from a distribution perspective, optimality of Euclidean distances for this purpose requires that the covariance matrix of the bin values is diagonal (no interaction between bins) with all elements of the diagonal equal (all bins have the same variance in the training set).

8.7.2 Mahalanobis distances

Mahalanobis distances are used for comparing descriptors in the classical descriptor model. When used for HAT descriptor matching, they give more important bins more weight—a bin is considered important for matching if it has low variance in the training set. Mahalanobis distances also account for (linear pairwise) interactions between bins.

The use of Mahalanobis distances assumes that the descriptors have an approximately elliptical distribution, and thus their distributions can be defined by their means and covariance matrices, with distance from the center measured as a quadratic form (Appendix B). The Mahalanobis distance is optimal to the extent that the distribution of the bin values is Gaussian, but this can be at most only approximately true for data that is bounded on one end (Figure 8.11).

Euclidean and Mahalanobis distances assume that there is a single unique center of the distribution around which the points cluster in multidimensional space. That is, they assume that for each landmark there is a single mean descriptor that characterizes the landmark. This assumption isn't always valid—surely it must be invalid for landmarks on the inner side of the lips, because the area around the landmark will be different if the mouth is open or closed, implying at least two different clusters of descriptors (multimodality).

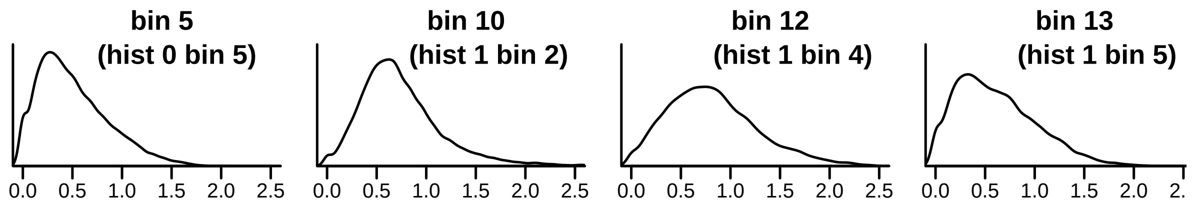


Figure 8.11: *Non-normality is evident in the distribution of values across the training set in example histogram bins.*
Without the square root used when normalizing the descriptors (page 95) these would be even more non-Gaussian (the square root pulls in the long right tail).
These examples are for the bottom left eyelid in the full scale image.

Calculating the Mahalanobis distance is slow. It's proportional to the square of the number of elements in the descriptor—in our case $160 \times 160 = 26\text{k}$. Trimming as described in Section 5.7 might reduce this number, but we're still talking about a lot operations.

To reduce dimensionality and noise we could use the Mahalanobis distance on a reduced set of principal components of the descriptors. There are other related measures of distance between histograms, such as the Hellinger distance (e.g. [3, 90]). We didn't explore these alternatives, but instead moved directly to the regression-based approach described next.

8.7.3 Regression for descriptor matching

Another tack is regression. This approach creates a regression model for each landmark that estimates goodness-of-fit for the landmark from any descriptor sampled from the image.

To generate the model for a landmark, we first collect a training set of descriptors on and around the "true" (manually landmarked) position of the landmark. These descriptors are collected from the manually landmarked training images. The descriptors are labeled with their distance from the true position (or alternatively simply labeled as 1 if on the true position or 0 if not). We then regress the labels on the descriptors, to create a model that predicts goodness-of-fit as a function of the elements of the descriptor.

There is a slight conceptual shift here because the descriptor model is no longer matching against an explicit template descriptor. However this doesn't change the basic mode of operation inherited from the classical ASM—we still search in the area around the point for the descriptor that best fits the model—it's just that we use regression instead of Mahalanobis distances for matching. This differs from approaches, also using regression, where the position of the landmark relative to the current descriptor position is predicted (e.g. Lindner et al. [67]).

8.7.4 Decision trees

What form of regression is best for our purposes? One approach is to use decision trees [11]. We found that trees give mediocre results in this application, but we present a brief example anyway since they are a fundamental statistical learning technique. Figure 8.12 shows a tree for matching the HAT descriptor at the landmark at the top eyelid in the full scale image.

Decision trees are highly interpretable. In the root node at the top of the example tree, if bin 58 is less than 1.5 we take the left branch; otherwise we take the right branch. We progress down the tree in this manner until we reach a terminal node, which is labeled with the predicted quality-of-match.

The bins and thresholds at each decision point are automatically selected by the model training algorithm [109]. The training process first selects a variable (in our case a histogram bucket, i.e., a range of gradients in a cell) and a value of that variable (i.e.

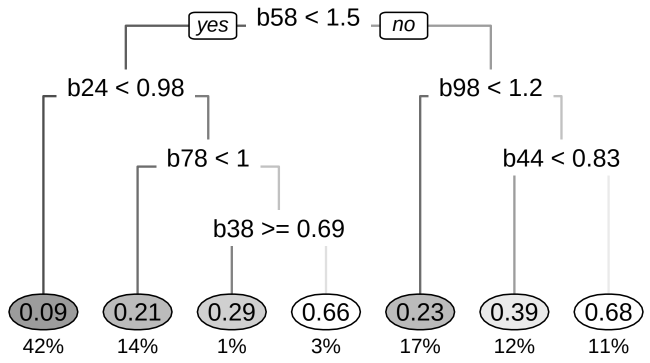


Figure 8.12: *Regression tree [77] for HAT descriptor matching.*
The shaded terminal nodes along the bottom show the predicted quality of match $0 \ldots 1$*. Also shown are the percentage of observations falling into the nodes.*

the prevalence of that range of gradients in the area covered by the cell). The variable and value are chosen to split the training data into the two sets that best partition the response values (i.e., how well the descriptor matches the image feature of interest). This first split forms the node at the top of the tree (in our example, bin 58 was selected with a value of 1.5). To build the rest of the tree, this process is continued recursively on each of the subsets formed by the first split.

8.7.5 Various forms of regression and MARS

What other forms of regression should be considered? Linear regression would certainly be a first option. Linear regression does allow multimodality to some extent, because a linear model can for example predict high fitness if say either bin 3 or bin 7 is full. But linear regression assumes that the effect of a bin on the match is proportional to its value. Also a straightforward application of linear regression makes the assumption that it's unnecessary to account for interactions between bins. These assumptions may be simplistic given that we are looking for a pattern of pixels.

To test that there is enough data in reasonably sized training sets to make more complicated regression assumptions, we trained Support Vector Machines (SVMs, e.g. [45,73]) for predicting fitness of HAT descriptors. SVMs did indeed outperform the other methods mentioned above. However, SVMs are too slow in this setting. The SVM at each landmark typically has over a thousand support vectors—the sheer number of support vectors for SVMs in this setting makes evaluation slow, because to evaluate the descriptor at any point in the image we have to take the dot product of the 160-element descriptor vector with each of these vectors ($160 \times$ say $1000 = 160$k operations).

We thus turned to Multivariate Adaptive Regression Spline (MARS) models [40, 84]. MARS models give ASM results that are almost as good as SVMs, and are much faster (Figure 8.13). In our software, MARS decreased total automatic landmarking time by over 80%. The essential difference between MARS and linear models for our purposes is that MARS automatically models non-linear relations. An overview of MARS will be given in the next section.

We mention also that in our tests linear SVMs gave slightly worse fits than the RBF SVMs we used above, with comparable speeds. Random Forests [10, 65] gave not quite as good fits and took about the same overall time (but we didn't do extensive tuning of Random Forests).

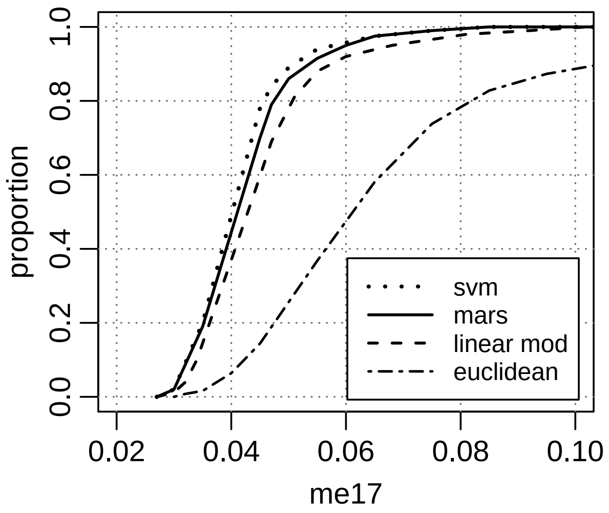


Figure 8.13: *Different forms of regression for descriptor matching.*

On this data SVMs give the best fits. MARS models give almost as good fits and are much faster

Distances were measured as me17s on an internal test set on full scale images before applying the shape model.

8.8 An overview of MARS

Multivariate Adaptive Regression Spline (MARS) models were introduced by Jerome Friedman in a 1990 paper [40]. After the release of Breiman et al.'s influential CART book [11] in 1984, MARS was a response to the often poor performance of decision trees on data that is continuous (as opposed to categorical). MARS has been shown to perform well in diverse applications (e.g. [62,116]), although it's apparently not well known in the image processing community.

A clear description of MARS can be found on Wikipedia [119]. See also Hastie et al. [45]. and Friedman's original paper [40]. Here we just give a brief overview of MARS by way of an example.

The MARS formula to estimate the descriptor match at the bottom left eyelid in the full scale image is

$$\begin{aligned} match = {} & 0.026 && 1 \\ & + 0.095\ max(0,\ 1.514 - b_5) && 2 \\ & + 0.111\ max(0,\ 2.092 - b_{10}) && 3 \\ & + 0.258\ max(0,\ b_{12} \quad - 1.255) && 4 \\ & - 0.108\ max(0,\ 1.574 - b_{13}) && 5 \\ & \ldots \end{aligned} \tag{8.6}$$

where the b_i are HAT descriptor bins $0 \ldots 159$. There happen to be 17 terms in the formula; we have shown just the first few. The MARS model building algorithm generated the formula from the training data. There is a similar formula for each landmark at each pyramid level.

What distinguishes this formula from that of a linear model is that the variables (bins in this case) enter the formula via the *max* functions instead of directly as they would in a linear model. The *max* functions are characteristic of MARS and are called hinge functions in this setting. Hinge functions allow nonlinear dependence on the value of a bin, and contain the effect of a bin to a range of its values. The effect of two of the hinge

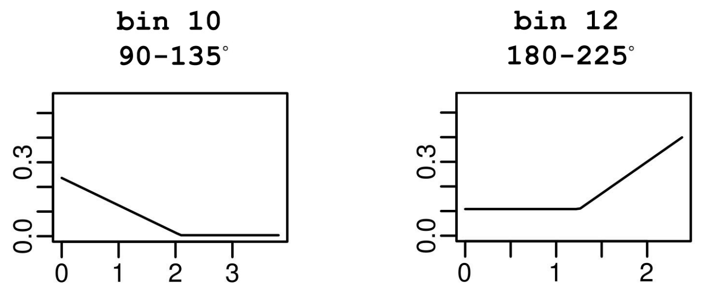


Figure 8.14: *Hinge functions showing the effect of bins 10 and 12 in lines 3 and 4 in the MARS formula on the previous page. From the plots we see that the estimated quality of match is increased by low values in bin 10 and/or high values in bin 12. From Figure 8.11, high values in bin 12 are uncommon. Low values are common but uninformative.*

functions is shown in Figure 8.14. The same bin can appear in different hinge functions in the same formula, allowing complex non-monotonic regression surfaces (but always piecewise linear).

MARS has included only certain histogram bins in the formula (there are 160 bins but only 17 terms in our example). HAT descriptors lend themselves to this kind of variable selection. The innate structure of the image feature makes some bins more important than others. Furthermore, the trilinear interpolation induces collinearities between nearby bins. When matching, it may therefore make little difference if we use a bin or its neighbor. The MARS model building algorithm will arbitrarily pick one or the other, depending on training sample noise. Once the bin is in the formula, the other bin can be ignored as it brings little new information. Usually the final formula is so short that evaluating it is quicker than it would be to say take the Euclidean distance between two descriptors. Exploiting structure in the data like this it saves unnecessary calculation and reduces noise from uninformative variables.

We compiled the MARS formulas as C++ code directly into our application. We performed MARS model selection (i.e., optimization of the number of terms) not by using the standard MARS Generalized Cross Validation [119] but by optimizing descriptor match performance on a tuning set of faces, a procedure that more closely matches the requirements of the model in our setting. With smaller training sets (less than a thousand faces) our experience has been that linear models can perform as well as MARS. MARS also allows interactions between variables, but interactions don't appear to give better fits in this setting unless large amounts of training data are available (more than ten thousand faces).

8.9 Training the HAT ASM

The frontal model was trained on the 6008 frontal and near frontal images of the MUCT `abde` subset. Mirroring was used to bring the number of faces from 3004 to 6008. (Figure 9.6 illustrates how this subset partitions the data. Section 7.1.1 explains why the MUCT data was chosen for training.)

It takes under three minutes to train this model on a 2.9 GHz processor. Parameter selection (such as selection of the HAT parameters in Section 8.6) was performed on an independent set of faces gathered from the internet.

We generated regression training data for the MARS descriptor models as follows. (This regimen was reached after some experimentation but we don't claim it's optimal.) At the landmark of interest in each training face, after scaling the face to the standard eye-mouth distance of 110 pixels, we generated

(i) Three negative training descriptors at x and y positions randomly displaced by 1 to 6 pixels from the "true" (manually landmarked) position of the landmark.

(ii) A positive training descriptor at the true position of the landmark. To bypass issues with imbalanced data, we duplicated this positive descriptor thrice so there were equal numbers of positive and negative training descriptors.

We regressed on this data with the positive training descriptors labeled 1 and the negative training descriptors all labeled 0, irrespective of their distance from the true position of the landmark.

8.10 Why do SIFT-based descriptors perform well?

Surprisingly, in the literature there seem to be only a few scattered comments on *why* SIFT descriptors perform well, without any extensive discussion. This section attempts to summarize the reasons for their good performance. I hope a certain amount of arm waving will be excused.

8.10.1 SIFT descriptors are based on gradients

SIFT descriptors are based on image gradients and inherit their good properties for template matching. We briefly summarize these properties below.

Gradients are invariant to the overall level of lighting, since by definition they use the difference of intensities, not absolute intensity.

After normalization of the descriptor (Sections 5.6 and 8.2.7), gradients are invariant to linear changes in overall contrast.

Many studies have compared different convolution masks for use in descriptors. An example is the comparison of 2D descriptors in Chapter 5 of my master's thesis [75]. That study found that descriptors incorporating signed gradient information performed best for ASMs.

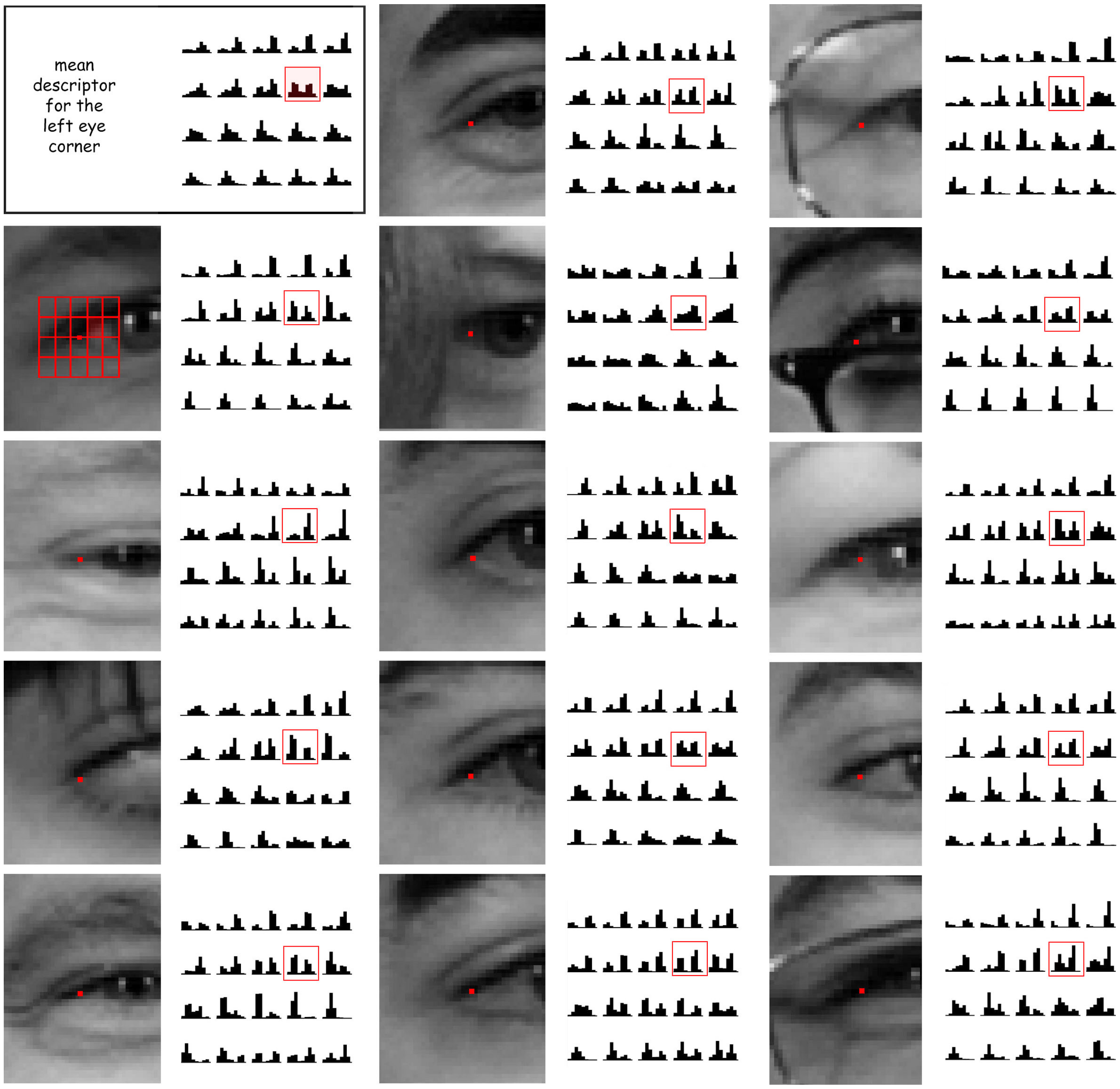


Figure 8.15: *HAT descriptors for the landmark at the left eye-corner.*

top left rectangle *Mean descriptor over the frontal training set.*

remaining images *The descriptor of some training images.*

The descriptors were extracted at the manual landmark after rotating the faces upright and scaling them to an eye-mouth distance of 110 pixels.

Despite the variety of eyes some similarities can be seen between all the descriptors. For example nearly all the histograms boxed in red have two distinct peaks in approximately the same bins.

8.10.2 SIFT descriptors allow some variations in features

We rotate the face upright and scale it to a fixed eye-mouth distance before the ASM search begins, so the image region covered by a descriptor during the search is as similar as possible to the descriptor regions in the training shapes. But even so, face images differ substantially, and on different images there will be a wide variety in the relative position and shape of image features in the immediate vicinity of the landmark.

However, the coarse grid of cells used in SIFT descriptors provides a degree of insensitivity to this kind of spatial variation. The exact position of an image feature in the cell doesn't matter; all pixels in the cell area map to the same cell, up to interpolation. Thus SIFT descriptors are insensitive to small changes in the face around the landmark. However, they still allow us to locate the landmark precisely using information interpolated and combined across the cells.

Figure 8.15 shows some SIFT descriptors for the landmark at the left eye-corner. Despite the variety of face shapes around the eye-corner, some similarities can be seen across all the descriptors.

The use of histograms is key, because it allows SIFT descriptors to aggregate gradient information over the cell area. It might be an interesting exercise to measure the performance of new types of descriptor that combine histograms with 2D gradient descriptors (or other convolution descriptors, Section 5.6). Such a descriptor would be made up of a coarse grid of histograms (as in HAT descriptors) measuring the prevalence of the value given by a convolution mask (as in 2D gradient descriptors). However it seems unlikely that such descriptors would outperform HAT descriptors.

In the SIFT paper Lowe mentions that the spatial invariance of SIFT descriptors is similar to that of certain neurons in biological vision. These neurons respond to image gradients at a particular orientation [34]. For these biological receptors, lab experiments have shown that the gradient at the orientation of interest can be anywhere in a small receptive region on the retina; the gradient doesn't have to be at a precise location.

8.10.3 HAT descriptors allow use of more context around the landmark

Without conscious effort, as humans we intelligently use the context around a point to precisely position the landmark. Computer descriptors also use this context, although not as intelligently—but HAT descriptors can make use of a larger context around the point than 2D gradient descriptors. They are better able to make use of the extra information in the larger area, without being confused by the extra noise and variability that comes along with that information.

For example, in the right side of Figure 8.16 although we are focused on the corner of the eye, the HAT descriptor includes quite a large surrounding area. The 19×19 region is bigger than the 13×13 (or so) region found to be optimal in this application for 2D gradient descriptors ([75] Section 5.5.4).

Simply increasing the area for 2D gradient descriptors doesn't help (and in fact degrades performance) because for these descriptors the large variation in the image areas far

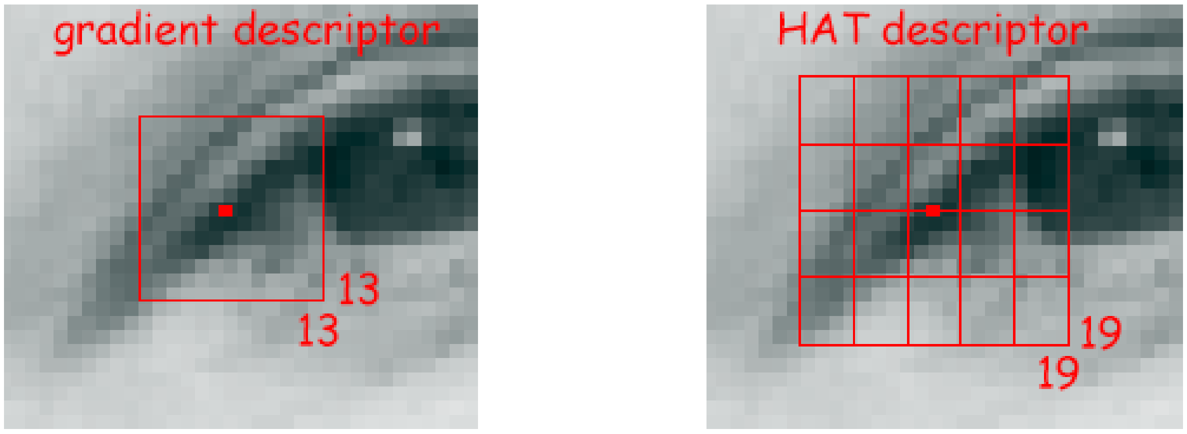


Figure 8.16: *Portion of the face covered by descriptors.*
The descriptor patch sizes shown above were determined to be optimal when tuning the models (with the faces scaled to an eye-mouth distance of 110 pixels).
Left *2D gradient descriptor.*
Right *HAT descriptor.*

from the landmark tends to smother the usable information in those areas. On the other hand, SIFT/HAT descriptors are relatively insensitive to this variation, as discussed in the above subsection.

It's also interesting to note from Figure 8.9 on page 101 that the optimum HAT patch width is larger with spatial interpolation than without. (In the figure the two curves with spatial interpolation bottom out after the other two curves.) That is, spatial interpolation allows the descriptor to usefully incorporate more context around the landmark—smoothing tends to increase the signal to noise ratio in the high variability areas far from the landmark.

8.10.4 Error surfaces for SIFT descriptors are relatively smooth

The error surfaces for 2D gradient descriptors are quite rough and noisy (Section 5.10). The trilinear interpolation used by SIFT descriptors makes for smoother error surfaces. This interpolation prevents sudden changes as the descriptor moves over the image surface during the search.

In general, smoothing of noisy error surfaces makes it more likely that the minimum is closer to the true position (Figure 8.17). We hypothesize that this effect occurs during landmark searches with SIFT descriptors.

It might be worthwhile measuring the effect of adding an extra step to explicitly smooth the 2D error surface before finding the minimum. This might give small improvements in landmarking accuracy—although its effect would probably be more noticeable on 2D gradient descriptors than on HAT descriptors (which already have built-in interpolation/smoothing)—and so such an experiment would be mostly of historical interest.

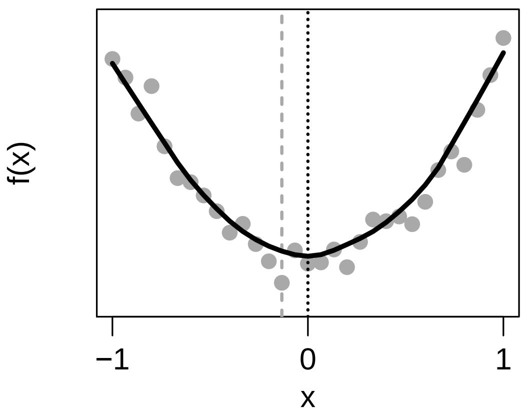


Figure 8.17: *Smoothing makes minimum-finding less susceptible to noise. This is an artificial one-dimensional example; for landmark finding we would work in two dimensions across the surface of the image.*
The gray points are sampled from $f(x) = x^2$ plus noise. The black curve is a smooth of the points [16].
The minimum of the smooth (dotted vertical line) is closer to the correct value of 0 than the minimum gray point (dashed vertical line).

8.11 Results with the HAT model

The HAT descriptors presented in this chapter are incorporated into Stasm version 4. This section compares Stasm version 4.2 (called *Stasm4* in the figures) to version 3.1 (called *Stasm3*), which used 2D gradient descriptors. Recall from Section 6.3 that Stasm version 3.1 itself performed well against other automatic landmarkers in independent tests made by Çeliktutan et al. [15].

Figure 8.18 shows results using the me17 measure (Section 6.2.1) on four different datasets disjoint from the training data. In all cases HAT descriptors outperform 2D gradient descriptors. Tests using measures other than the me17 gave results similar to those shown here.

The top left plot also compares Stasm to the Random Forest Regression Voting method presented in Cootes et al. [20] and Lindner et al. [67]. The results in the latter paper are to my knowledge currently the most accurate me17 results published on the BioID data. Lindner et al. outperform Stasm, although above the median the distributions are close. Their model takes 82 ms to landmark 17 points on a 2.9 GHz Xeon 5670 processor, whereas Stasm takes 46 ms on 77 points on a similar(?) processor (Figure 8.21). However Lindner et al. also present a reduced model that gives very slightly worse accuracies on 17 points at only 38 ms per image.

The fits on the left of the BioID curve are in the manual landmarking noise floor. That is, the landmarking errors in the faces on the left part of the curve are due to manual landmarking uncertainty as much as they are to automatic landmarking errors. Figure 8.19 illustrates this. See also Section 7.2 which discusses manual landmarking uncertainty.

Both versions of Stasm are limited to near-frontal views by their reliance on the OpenCV frontal face detector, so are inappropriate for the Helen set [61] with its wider range of poses, but HAT descriptors do substantially better (bottom right plot). In the next chapter we will look at a multiview model with much better performance on this Helen dataset.

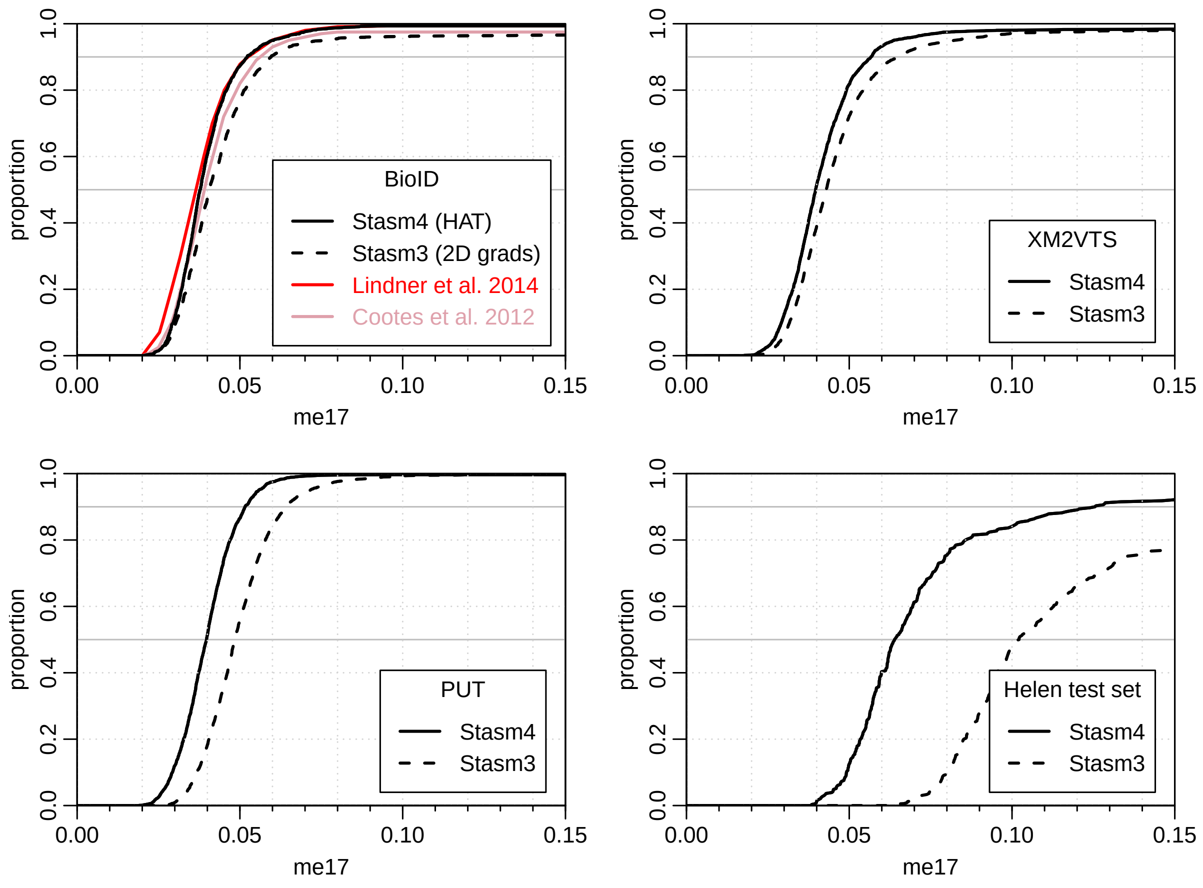


Figure 8.18: *Stasm4 (HAT descriptors) versus Stasm3 (2D gradient descriptors). HATs outperform 2D gradients.*
These are `me17` *distributions on datasets disjoint from the training set. The datasets are the BioID [2], XM2VTS [72], and PUT [55] sets, and the designated test set of the Helen set [61].*
Each plot includes the complete dataset (faces for which the face detector failed are included with an me17 of infinity).

More extensive tests of our methods against other automatic landmarking techniques (and not just against 2D gradient descriptors) are deferred until the next chapter, since the frontal model in this chapter is just a stepping stone to the multiview model in the next chapter. To avoid duplication, we also defer showing photographs of example landmarked faces until the next chapter (Section 9.6.2).

8.11.1 Results with the pe17 measure

Figure 8.20 compares Stasm to Belhumeur et al. [7] using the pe17 measure (Section 6.2.2). On this frontal set, the Stasm4 results are practically identical to those of Belhumeur et al., but Stasm is much faster (their landmarker takes on the order of a second per landmark, whereas Stasm processes the entire face in less than 50 ms, Figure 8.21). As mentioned for the previous graph, the fits on the left of the curve are in the manual landmarking noise floor.

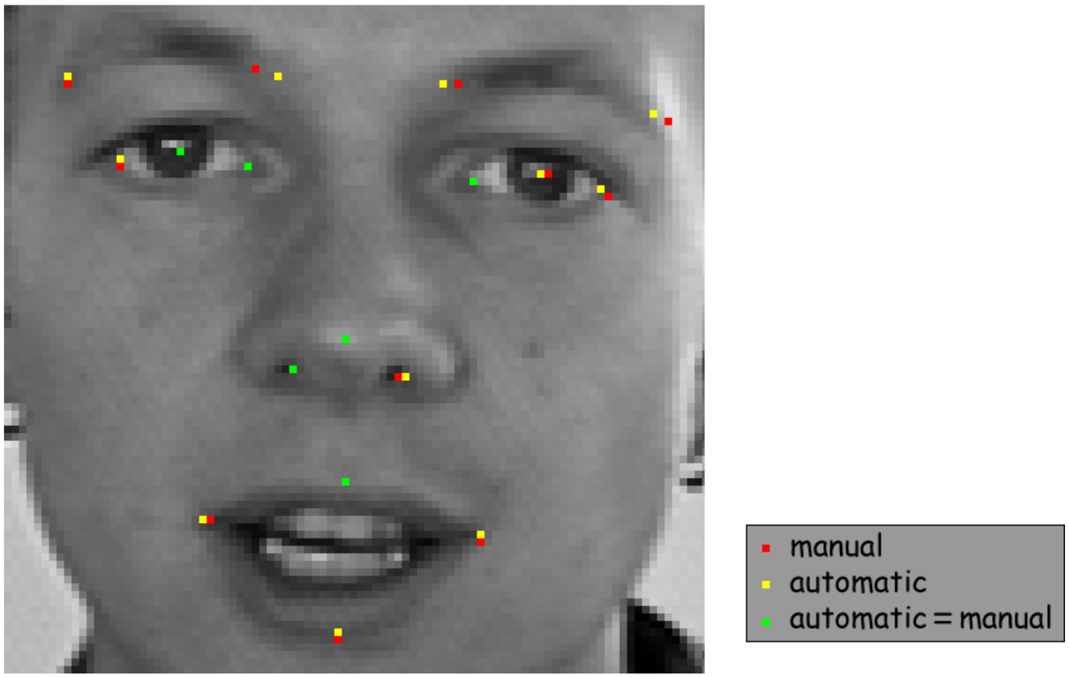


Figure 8.19: *Landmarks on the BioID face with the best fit by Stasm4. (This fit is the leftmost point on the black curve in the top left plot in Figure 8.18.)*

The `me17` *is 0.0195. This is a mean error of about one pixel* ($0.0195 \times 49.3 = 0.96$ *pixels, where 49.3 is the inter-pupil distance of this face).*

In this image, most (if not all) discrepancies between the manual and automatic landmarks are within manual landmarking uncertainty.

For example, the automatic landmark (yellow) on the right pupil is closer to the correct position than the manual landmark (red). Most people would say this is also true for the landmarks on the outer eye-corners.

To my knowledge, although several techniques outperform Belhumeur et al. in this test (e.g. Cao et al. [13]), they do so by no more than a hair, and all are far more complicated than Stasm. It should be remembered however that the BioID faces are frontal and thus well suited to the Stasm frontal model.

8.11.2 Search times

Figure 8.21 compares search times. Stasm4 is faster than Stasm3. The times increase as we go rightwards because the images and faces are larger: face detection takes longer on larger images, and it also takes longer to scale larger faces to the standard eye-mouth distance before the ASM search begins.

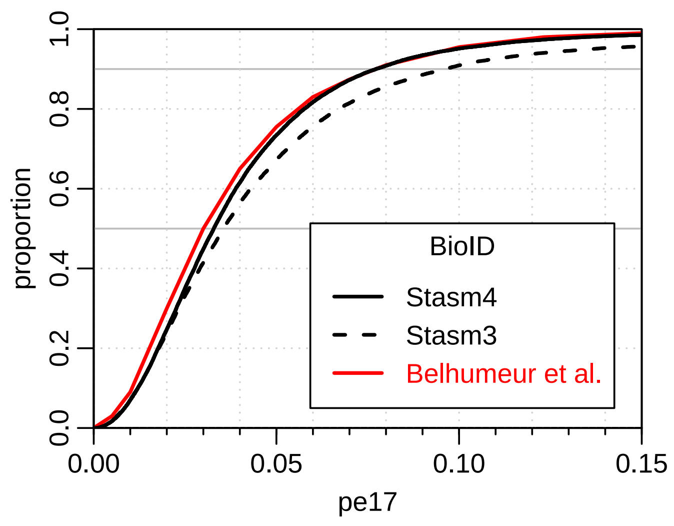


Figure 8.20: *Stasm versus Belhumeur et al. [7] on the BioID data. See also Figure 6.6.*

Note that this plots the `pe17` *measure (not the* `me17` *used in the previous graphs). See Sections 6.2.1 and 6.2.2 for definitions of these measures.*

Faces for which the OpenCV face detector failed are included with pe17s of infinity.

8.11.3 Details on the test setup

The Stasm curves in Figures 8.18 and 8.20 include all faces in the test sets. Faces not found by the face detectors are included with an me17 of infinity (Section 6.3). The times in Figure 8.21 don't include faces for which the face detectors failed.

For the PUT set several extra landmarks needed for calculating the me17 were manually added to the faces before testing began.

For the Helen set we approximated the pupil and nose points from neighboring points, introducing noise into the results (but equally for both implementations). The pupil position in the Helen faces was taken to be the mean of the eye-corners. The Belhumeur et al., Cootes et al., and Lindner et al. curves were transcribed from figures in their papers. All other curves in this chapter were created by running software on a local machine.

Stasm versions 3.1 (2D gradient descriptors) and 4.2 (HAT descriptors) were used for

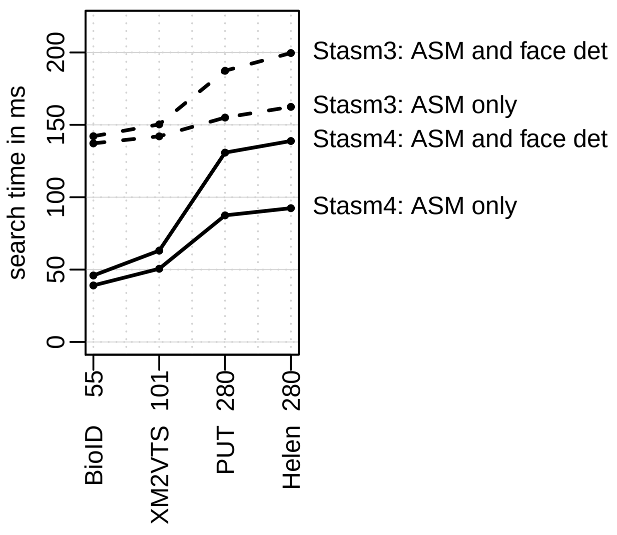


Figure 8.21: *Times for face detection and ASM search.*

The databases are arranged in order of mean inter-pupil distance (shown after the database name on the horizontal axis).

Times were measured on a 2.9 GHz i7 4910MQ processor with the same datasets as Figure 8.18.

On the BioID set, Stasm 4 takes 46 ms to detect the face and find the landmarks.

these tests. Both versions were trained on the MUCT data, and both use the OpenCV frontal face and feature detectors. Stasm version 4.2 is a 64-bit executable file (as opposed to a 32-bit executable), and uses OpenMP parallel processing (Section 5.8). For a more direct comparison of HATs to 2D gradient descriptors without these additional implementation differences, please see Milborrow and Nicolls [83]. The conclusions in that paper are the same as those here, but the HAT model here is better tuned.

Stasm is open-source and includes tools to reproduce these results.

8.12 Discussion

This chapter shows that HAT descriptors together with MARS work well with ASMs. HAT descriptors outperform gradient descriptors. HAT descriptors with MARS bring significant computational advantages over SIFT descriptors with Mahalanobis distances or SVMs.

An obvious next step would be to investigate other modern descriptors such as GLOH [74], SURF [5], or HOG [29] descriptors. However, evidence from other domains indicate that such alternatives would probably give little improvement in fits over HATs.

Fits might be better if the descriptor model regressed on a subset of the principal components of the HAT descriptor (rather than directly on the descriptor itself).

A disadvantage of ASMs is the lack of wider context around a point during template matching, especially at fine resolutions. The relatively large area of the image covered by HAT descriptors helps (Figure 8.16), but perhaps more could be done. The descriptors of Belhumeuer et al. [7] may be of use in this regard, and would fit easily into our scheme. They form their descriptors by concatenating two SIFT descriptors (we would use HATs) at different image resolutions—the lower resolution descriptor provides context for the higher resolution descriptor. Along the same lines, Q. Zhang et al. [128] take the tack of incorporating all four pyramid levels into a single descriptor for each landmark. Of approaches that also make use of the context around the landmark (but without impractical search times), another avenue could be regression-based models like the Random Forest model of Cootes et al. [20].

The advantages of HATs are diminished by the principal-component ASM shape model, in the sense that the improvement of HATS over 2D gradient descriptors is substantially larger before the shape constraints are applied. Alternatives to the classical shape model thus become attractive.

In recent years researchers have paid considerable attention to improving the way the descriptor and shape models work together. Instead of the separation between descriptor matching and the shape model of the classical ASM, one can build a combined model that jointly optimizes the descriptor matchers and shape constraints. An early example is the Constrained Local Model of Cristinacce and Cootes [28]. An informative taxonomy up to 2010 is given in Saragih et al. [101]. As mentioned on page 111, error surfaces over the search area are smoother for HATs than for 2D gradient descriptors, and this smoothness should ease the difficult optimization task.

Chapter 9

Multiview models

Given the solid performance of the HAT-based ASM in the previous chapter, it's natural to ask if the model could be extended to also handle non-frontal faces. In this chapter we make those extensions.

We build a multiview model using a straightforward strategy of three submodels. These submodels are ASMs optimized for frontal, left three-quarter, and right three-quarter views respectively (Figure 9.1). Using the position of the face supplied by a global face detector, we first estimate the face's yaw and then search for landmarks with the submodel that best matches that yaw.

After a review of related work, this chapter describes how the face's pose is estimated. It then describes the ASMs used as submodels. Finally, it gives results and discusses future directions.

9.1 Related Work

The idea of using different submodels for different poses isn't new. Cootes et al. [25] present an early example. They demonstrate that a small number of models can represent a wide range of poses. Their model, AAM based [19], can also synthesize views of the face from new angles.

The approach we take in this chapter rigidly separates pose detection and landmarking. A limitation of this approach is that if the pose is misestimated the wrong ASM is applied with no chance for correction. In contrast it seems that most examples in the literature unify pose and landmark detection. A fairly recent example is Zhu and Ramanan [132]. As in the method presented in chapter, Belhumeur et al. [7] use SIFT

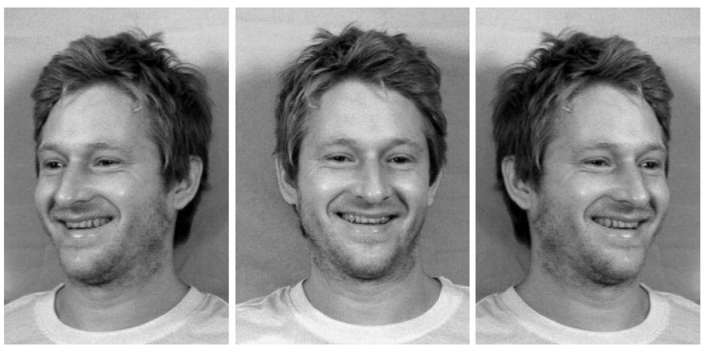


Figure 9.1:

The three canonical yaws used in the multiview model:

(i) *left three-quarter*
(ii) *frontal*
(iii) *right three-quarter.*

descriptors and multiple models, but they use a large number of global models, jointly optimizing pose and landmark detection using a probabilistic model with SVMs and RANSAC. Kanaujia and Metaxas [54] also use SIFT descriptors and multiple models. They determine pose clusters automatically, unlike our method which simply assumes that three models suffice for the poses of interest. The approach described in this chapter is faster and more basic than any of the methods just mentioned.

9.2 Why submodels?

Why do we even bother with submodels? In other words, why not simply take the frontal ASM presented in the previous chapter and instead of training it only on frontal faces, train it on a wide variety of poses.

Unfortunately an "über ASM" trained like that doesn't perform well. In attempting to be a jack of all trades it becomes master of none.

With large pose variations, the distribution of face shapes isn't at all Gaussian, and the ASM shape model based on principal components doesn't work well. To accommodate the large variation in shapes, the shape model has to be very flexible, and thus it doesn't adequately constrain the shapes suggested by the descriptor models. Furthermore, as the pose changes the features change to an extent that prevents them from being characterized by single descriptor models—for example, the tip of the nose appears quite different in left and right three-quarter views.

9.3 The multiview model

This section describes the multiview model. Figure 9.2 summarizes how the multiview model works.

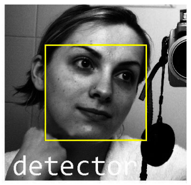
9.3.1 Face detection

As always, we first locate the face with a global face detector. Since the faces have a wide variety of poses, the OpenCV frontal detector no longer suffices as it did for the frontal-only model. Section 9.5 will discuss the face detector. For now we just mention that the face detectors we currently use don't return pose information, therefore we have to independently estimate the face's pose.

9.3.2 Pose estimation

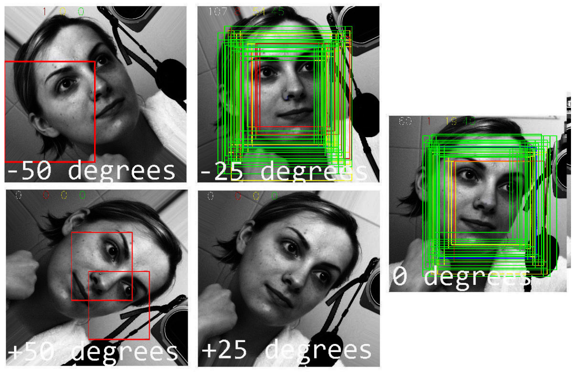
Before the search begins we must estimate the face's pose so we can rotate the face upright and choose the appropriate ASM submodel. This subsection describes how we estimate the pose.

Figure 9.2: **The multiview model**



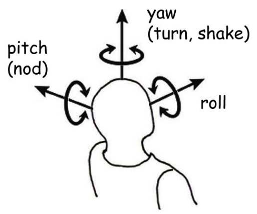
(a) **Detect the face** The figure shows the original image with the face detector rectangle.

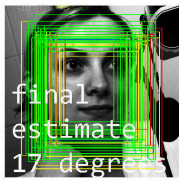
(This is image 313914487_1 from the Helen test set [61].)



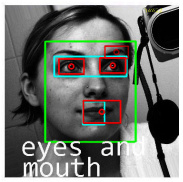
(b) **Estimate rotation** The image is rotated by -50, -25, 0, 25, and 50 degrees. The left three-quarter, frontal, and right three-quarter face subdetectors are applied to each rotated image in the area around the face detector rectangle.

In this example, the -25 degrees image has the largest number of detections. The estimated rotation (17 degrees) is a thus a weighted sum of -50, -25, and 0 degrees. The weights are the number of detections in the -50, -25, and 0 degree images. The false detections on the highly rotated faces are essentially ignored.

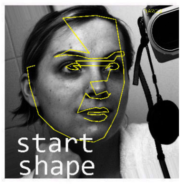




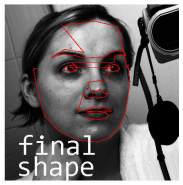
(c) **Estimate yaw** After estimating the in-plane rotation of the face, the image is rotated so the face is upright. We work with this rotated image until the ASM search is complete.
The three face subdetectors are re-applied and a MARS model estimates the face's yaw using the relative number of detections by each subdetector on the upright face. In this example the estimated yaw is 20 degrees.



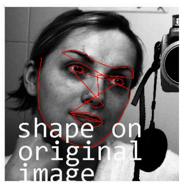
(d) **Estimate eye and mouth positions** Eye and mouth detectors are applied. (The eye and mouth positions will be used to position the start shape, Chapter 4.) The false detection on the right eyebrow is ignored because its centroid is outside the cyan legal zone (Section 4.2).



(e) **Place the start shape** The start shape for the right three-quarter ASM is aligned to the face, using the face detector rectangle and the eye and mouth positions if available. We use the right three-quarter model here because the yaw was estimated to be 20 degrees in step (c) above.



(f) **Search with appropriate sub ASM** The face is scaled to an eye-mouth distance of 110 pixels and the right three-quarter HAT ASM is applied. The figure to the left shows the final shape after the ASM search.



(g) **Map the shape back to image frame** The final shape is derotated and scaled back to the original image.

The subdetectors

We use the position of the face determined by the face detector, but to estimate its pose apply three further face detectors. The subdetectors are trained for left three-quarter, frontal, and right three-quarter faces respectively. The next few paragraphs describe these subdetectors; we will shortly give details on how they are deployed.

Each of the subdetectors consists of a truncated cascade of just 14 weak Viola-Jones classifier stages, rather than the typical 20 or more stages used in standard face detectors. This increases false detections (undesirably), but more importantly greatly increases the number of detections per face (desirable in this application because a larger number of detections increases the granularity of the rotation and yaw estimators).

The subdetectors are Viola-Jones detectors [66, 114] with Local Binary Patterns [64] built with the OpenCV tools on the AFLW dataset [60]. Local Binary Patterns were used for speed. The AFLW dataset was used for training because of its wide variety of faces and also because we don't need it for testing. The ground-truth pose of each face was estimated (thanks to Tom E. Bishop) using the POSIT code [30]. This pose information makes it possible to select faces with the range of yaws appropriate for training each subdetector. The AFLW manual landmarks are unneeded for this application of the dataset.

The number 14 of stages above was chosen after some experimentation. Full details of the detector parameters may be found in the comments embedded in the XML files in the multiview Stasm code.

Estimating rotation

To estimate the face's (in-plane) rotation, we first generate images rotated by -50, -25, 0, 25, and 50 degrees, as illustrated in Figure 9.2(b). On these rotated images we apply the three face subdetectors independently in the area on and around the detected face. Face rectangles that are too far from the median rectangle in each image are discarded as false positives. One of the rotated images will have the largest number of detections. The estimated rotation of the face is taken to be the weighted sum of this image's rotation angle and the rotations to each side of it, with the weights being the number of detections in the three rotated images. (We also tried a MARS model here, but a weighted sum gives results that are just as good, and is simpler.)

Estimating yaw

Once in-plane rotation has been estimated, the image is rotated so the face is upright. For efficiency we actually use just the region around the face rather than the entire image. We work with this rotated region of interest until the ASM search is complete.

The three subdetectors are re-applied to the upright face, as illustrated in Figure 9.2(c). The face's yaw is estimated from the number of detections by each of these subdetectors.

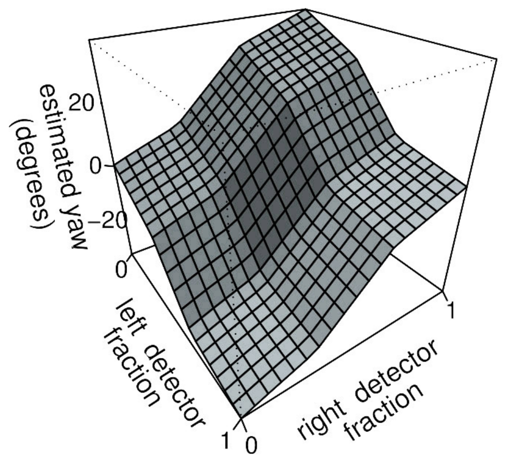


Figure 9.3:

MARS model to estimate the face's yaw.

This estimates yaw from the number of detections by the left three-quarter and right three-quarter face detectors (as a fraction of the total number of detections by all three detectors).

This estimation is made with the MARS model illustrated in Figure 9.3. It estimates the yaw from the fraction of left and right three-quarter detections (of the total number of detections by all three subdetectors). Intuitively, MARS is appropriate here because the yaw can be estimated as a weighted sum of the counts, but with some adjustment for non-linearity.

Note that the number of frontal detections enters the model implicitly. For example, on the far left of the figure, the left and right fractions are zero, implying that all detections were by the frontal detector, which is strong evidence for a frontal face.

The MARS model was trained on 2000 AFLW images, for each face regressing the ground-truth yaw on the relative counts of the three detectors on the upright face. The MARS model has a regression R^2 (coefficient of determination) of 0.79 on this training data, slightly better than a linear model which has an R^2 of 0.76.

How well does this basic yaw detector work? Figure 9.4 shows the estimated yaw versus the ground-truth yaw (which is itself an estimate made with the POSIT code). Figure 9.5 shows how fitness varies with the absolute error in estimating yaw. There is only a weak relationship between fit and yaw error, i.e., improving the yaw detector would give only a very small overall improvement in fit on this data. We verified that to be true by running the model again on the same data, but using the ground-truth instead of the estimated yaw to select the submodel for each face. The resulting error distribution curves were almost identical (not shown). Similar results applied for the rotation detector (not shown).

Alternative pose estimators

As an alternative approach, we trained an SVM to estimate the yaw from the histogram-equalized region in the face detector rectangle. After some experimentation with different subregions in the rectangle, this approach was abandoned for now because it didn't give as good estimates of yaw.

However it would worthwhile pursuing this alternative further, as it is more efficient than our current run-all-detectors technique. Viola and Jones [115] for example describe a highly efficient pose detector that uses a decision tree on standard Viola-Jones image

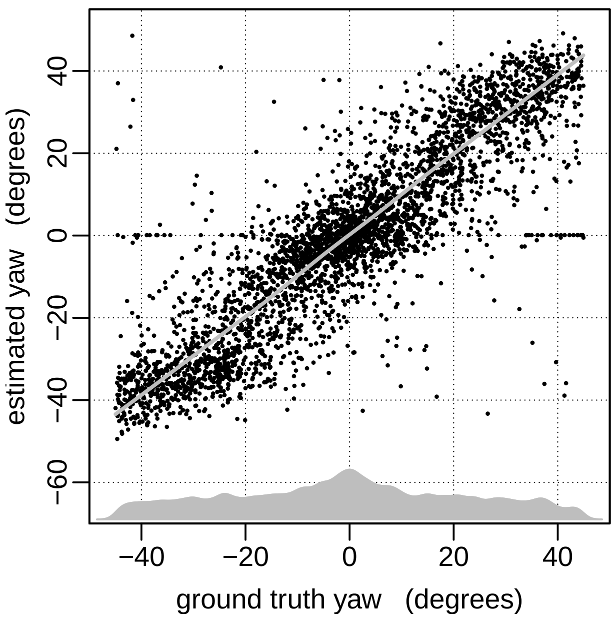


Figure 9.4:

Estimated yaw versus ground-truth yaw.

The shaded curve on the bottom shows the density of ground-truth yaws in the test set.

Measured on the same AFLW data as Figure 9.7.

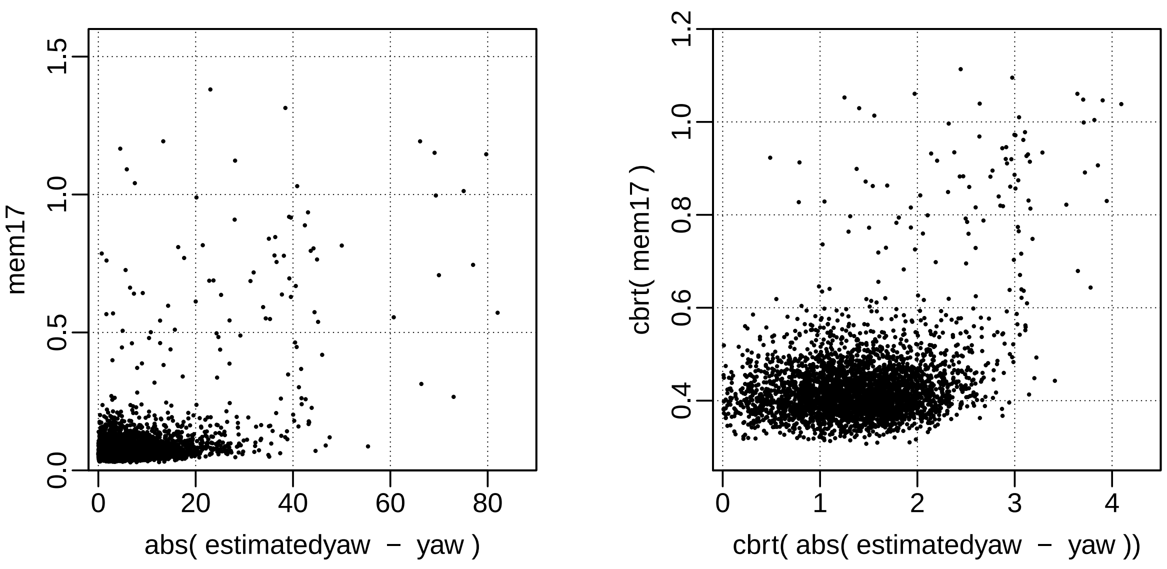


Figure 9.5: *Landmark fits versus error in estimating yaw.*
Left `mem17` *versus absolute difference between estimated and ground-truth yaw.*
Right *The same, after taking cube roots on both the horizontal and vertical axes to decompress the crowded section of the plot.*
Measured using the multiview Stasm on the same AFLW data as Figure 9.7.

features. It would also be worthwhile looking into the possibility of an open-source face detector that also returns pose information (e.g. see the survey by Zafeiriou et al. [124]).

9.3.3 Applying the appropriate ASM

The estimated yaw determines which ASM to use: if the yaw is between -14 and 14 degrees, the frontal ASM is used; otherwise the appropriate left or right three-quarter ASM is used. The parameter 14 was determined by empirical testing on an internal parameter-selection set. Its exact value isn't critical.

As usual the start shape is positioned using the detected eyes and mouths, before the appropriate ASM is applied. The eyes and mouths are detected as described in Section 4.2, although for three-quarter faces we must use different search areas within the face detector rectangle.

Once the ASM search for landmarks is over, the shape is derotated and scaled back to the original image as illustrated in Figure 9.2(g).

9.4 Training the multiview ASMs

Per-face yaw labels are needed to train the pose-specific submodel ASMs. For this project the models were trained on the MUCT data, which has faces categorized into frontal and three-quarter views. Figure 9.6 illustrates how the MUCT data was partitioned for the submodels. (Section 7.1.1 explains why the MUCT data was chosen for training.)

The frontal ASM for the multiview model is the same as that presented in the previous chapter. As described in Section 8.9, this model was trained on the 6008 frontal and near frontal images of the MUCT `abde` subset (using mirroring to bring the number of faces from 3004 to 6008).

A single three-quarter ASM was trained. This is the right-three-quarter model, which is flipped if necessary during the search to act as a left-three-quarter model. Actually we flip the image, not the model, but the net effect is the same.

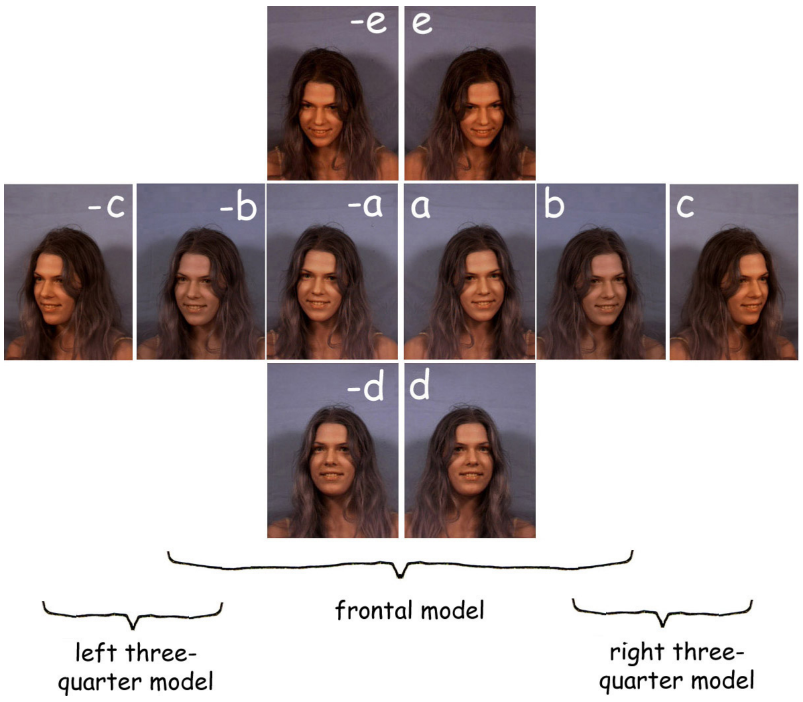


Figure 9.6: *How the MUCT data is partitioned to train the ASM submodels.*

The labels **a**. . . **e** *refer to the camera position, i.e., the pose. Mirrored images have negative labels.*

In reality the left three-quarter model wasn't actually generated. Instead the right three-quarter model is flipped if necessary during the search.

The three-quarter model was trained on the much smaller set of 1502 yawed faces of the MUCT `bc` subset. The `c` faces are three-quarter faces. The `b` faces aren't three-quarter (more like five-eighths) but are used to augment the training set as described in Section 3.14.3. Mirroring can't be used because the faces aren't frontal. The size of this set was artificially increased to 3500 by applying x- and y-stretching, small rotations, and changing the intensity gradient across the face. Note that in the three-quarter views many points will be obscured. Such missing points are handled as described in Section 3.14 "Imputing missing points for training three-quarter views".

Training the right three-quarter model takes a little less time than the frontal model (which takes less than three minutes) because the training set is smaller. The total training time for the entire multiview model is thus under six minutes.

Model parameters for all submodels were determined by testing against an internal parameter-selection set of faces gathered from the internet (Section C.2).

9.5 The multiview face detector

The Stasm code uses the OpenCV [87] libraries, which currently don't provide a multiview face detector.[1] Therefore for generating reproducible test results in this chapter we use the face detector of the 2013 iBUG 300-W Challenge (Section 9.6.3). To be precise, we take our existing multiview model trained on the MUCT data but use the 300-W face detector instead of the commercial detector for which our model is optimized.

The iBUG group don't make the actual detector available. However, they do provide face detector rectangles for faces in some popular datasets. Using these rectangles we can run tests on these datasets as if we had the detector.

Unfortunately, coordinates aren't available for this detector on the BioID and PUT data, so we can't display fitting results on that data—but they would be the same as those for the frontal-only HAT model in the previous chapter (Figure 8.18), up to differences caused by the different face detector.

9.6 Results with the multiview model

This section presents results for the multiview model.

9.6.1 Fit versus yaw

Figure 9.7 shows the fit for faces at different yaws with different submodels. The combined model (black curve) achieves the best fit across all yaws, by automatically selecting the ASM appropriate for the yaw as described previously. However, misestimates of yaw sometimes occur, as was seen in Figure 9.4 and can be seen here from

[1]OpenCV provides frontal and side detectors. Neither of these work well on three-quarter faces. This was checked in the version of OpenCV at the current time, OpenCV 3.0.0.

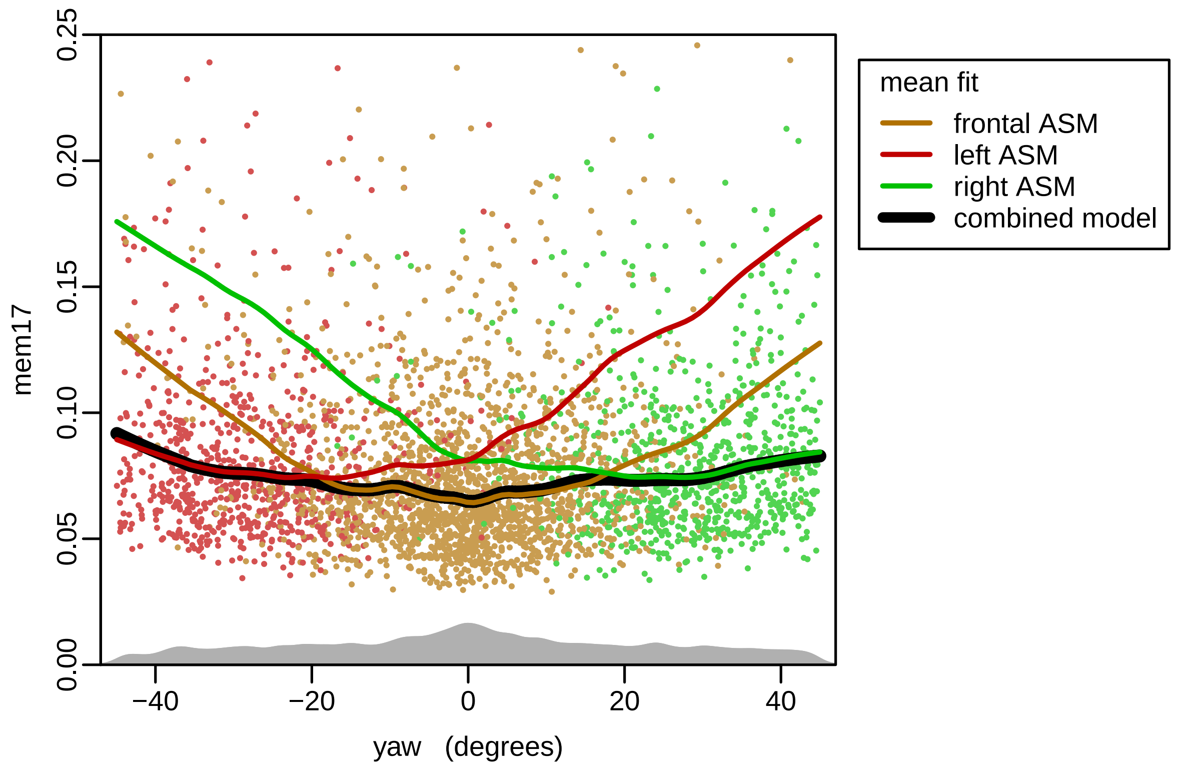


Figure 9.7: *Mean fit versus yaw using submodels applied individually and combined.*
The horizontal axis is the ground-truth yaw.
The dots show the fit on each face using the combined model, with their color indicating the ASM submodel that was selected using the estimated yaw.
The shaded curve on the bottom shows the density of ground-truth yaws in the test set.
Fits were measured on 3500 faces from the AFLW set. The text below has details.

misplaced colored dots. The effect of these misestimates is mitigated by the partial robustness of the models to poses out of their yaw range.

From the slight concave shape of the black curve we see that fits are generally slightly worse on non-frontal faces, even with submodels. The smaller training set used for the three-quarter models may be playing a part here (Section 9.4).

Details on Figure 9.7: Fits were measured using the `mem17` measure (the mean error of 17 face points, normalized by the eye-mouth distance, Section 6.2.4). Fits were measured on 3500 faces randomly drawn from the AFLW set [60] with ground-truth yaw and pitch between -45 and 45 degrees and rotation between -30 and 30 degrees. We don't have a face detector capable of detecting these faces, nor do we have the AFLW face detector rectangles for the 300-W detector (Section 9.5). Therefore the position and size of the face detector boxes was simulated by adding random noise to the face extent specified by the manual landmarks. The level of noise was tuned so our software gave similar mem17s on the Helen training set regardless of whether the simulated or actual 300-W face detector was used (Section 9.5). A real face detector would give slightly different results.

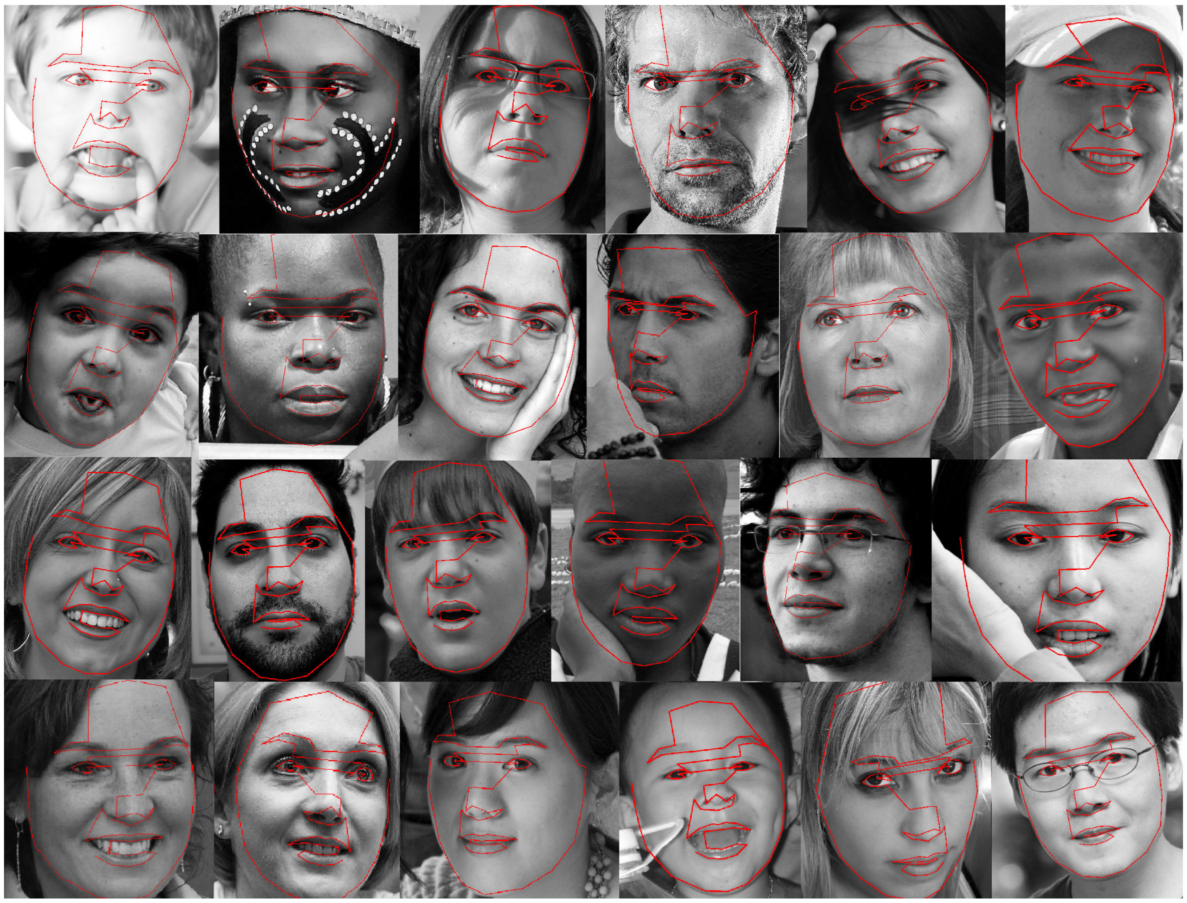


Figure 9.8: *Faces automatically landmarked by the multiview model. These faces are from the datasets used in Figure 9.12.*

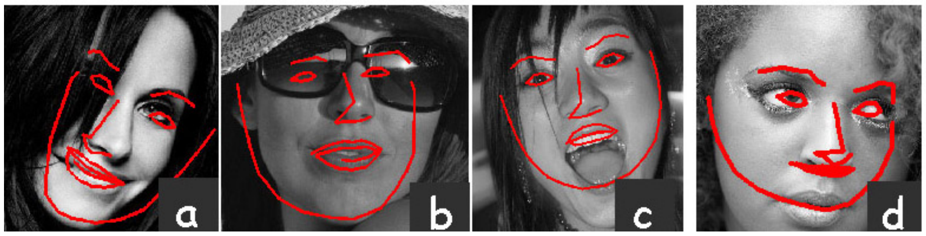


Figure 9.9: *Bad fits by the multiview model:*
(a) yaw was incorrectly estimated as being leftwards, so the wrong submodel was used
(b) eye locations confused by sunglasses
(c) mouth open, confused bottom lip for chin
(d) confused base of nose for mouth, and mouth for chin.
These faces are from the datasets used in Figure 9.12.

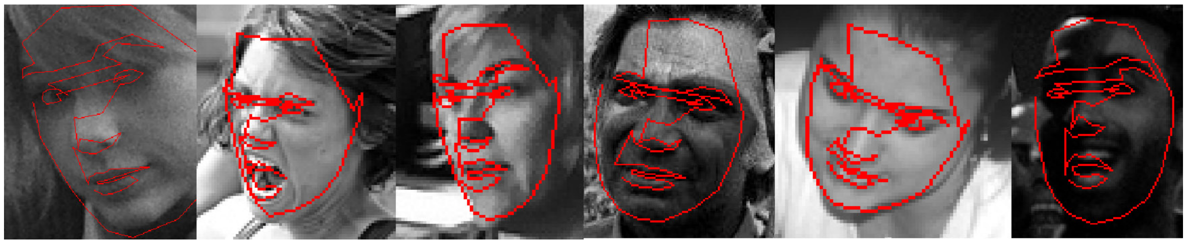


Figure 9.10: *Fits by the multiview model on the low-quality images from Figure 7.6.*

9.6.2 Example fits, good and bad

Figure 9.8 shows some faces automatically landmarked by the multiview model. The figure illustrates the model successfully handling faces in various poses and with partial occlusions. (Such montages of faces, although visually appealing and common in papers, are too small a sample to be indicative of overall performance. The cumulative distribution graphs displayed later are more informative.)

Figure 9.9 shows examples of bad results with the multiview model. The yaw was misestimated in Figure 9.9(a) because the hair across the side of the face was mistaken as background by the face detectors. Shadows across the face can similarly mislead the face detectors. Sunglasses as in Figure 9.9(b) are nearly always a problem. Perhaps a sunglass detector could be used to indicate that the local texture in the region of the eyes isn't to be trusted, and that the position of the eyes should be determined by the overall shape model or simply marked as not available. Likewise an open-mouth detector could resolve cases like Figure 9.9(c). However in general we prefer to avoid such accretions to the algorithm.

Figure 9.10 shows fits on some low-quality images referred to in Section 7.3.

9.6.3 The iBUG 300-W challenge and face data

This section describes the 2013 300-W challenge and the associated iBUG data. The following sections will then present the results of applying the multiview model to this data.

The 2013 300-W challenge

The 2013 iBUG 300-W Challenge [100] tested the ability of participants' software to automatically locate 68 facial landmarks in unseen images of faces. A hidden test set, not available to the participants, was used to evaluate the software. This set consists of 300 indoor and 300 outdoor faces with reference landmarks created semi-automatically. Participants submitted their software to the challenge organizers who ran the software on the hidden set and reported the results.

To prevent variation in landmark accuracies caused by use of different face detectors, all face positions were estimated with the standard iBUG 300-W face detector. As mentioned on page 124, the actual face detector isn't made available, but the detector rectangles are supplied for the example faces. These rectangles were used as a surrogate face detector for the graphs in this chapter.

The organizers measure fitness as the mean distance between the automatically located points and the reference landmarks, divided by the distance between the reference outer eye-corners. This fitness measure is reported for all 68 landmarks (called the `ec68` measure in this document) and also for the 51 landmarks interior to the face (the `ec51` measure).

The iBUG landmarks

The organizers of the challenge supplied reference landmarks created semi-automatically for several thousand example faces for images from popular face datasets, such as the XM2VTS set. Note that these iBUG landmarks don't exactly match the landmarks provided by the original developers of these datasets. Note also that only 224 of the original 300 LFPW test faces are available, and only this subset is shown in the plots.

The iBUG test set

The organizers of the challenge also provided a small test set of 135 faces intended to be representative of the faces in the hidden set. We call this set the iBUG `test` set (not to be confused with the iBUG `hidden` set).

Many of the faces in the test set are of low resolution or quality. For examples, see Figure 7.6 on page 89. As discussed in Section 7.3 "The ideal face database", in this thesis we are more interested in applications where such faces are unusual. Thus this test set and the hidden set aren't representative of the data that our multiview model is designed for, but nevertheless present an interesting academic challenge.

Converting Stasm to iBUG landmarks causes performance degradation

Our model was trained on the 77 MUCT landmarks. When the iBUG manual landmarks are used as the ground truth for measuring fitness, our landmarks have to be converted after the search to the 68 iBUG landmarks.

These two sets of landmarks define the landmark positions differently. For example, the iBUG outer left eye-corner landmark is above and to the right of the MUCT landmark (Section 7.2.2).

To perform the conversion to iBUG points, our software estimates their position either by simply copying points (e.g. eye-corners) or by interpolating between nearby points (e.g. jaw landmarks). (Ad hoc interpolation constants were estimated from the first 10 faces of the Helen training data. Because this data was used for this purpose during training, the Helen training data isn't displayed in the test results in this chapter.)

Because of landmark conversion discrepancies, when the iBUG manual landmarks are used as the ground truth the performance of our model is artificially degraded compared to models trained directly on iBUG landmarks.

9.6.4 Multiview versus frontal-only model

Figure 9.11 shows the cumulative distribution of fits on various datasets using the multiview model (solid lines) and the frontal-only model (dashed lines). Of these sets, the Helen test set with its wide variety of at least medium-quality faces is most representative of the faces our software is designed for (Section 7.3).

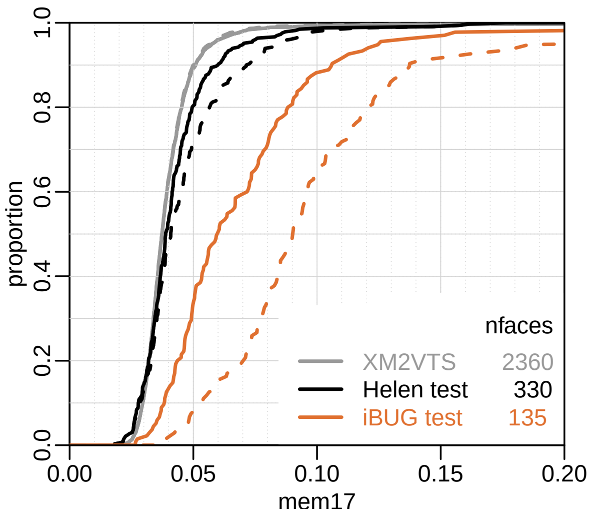


Figure 9.11: *Multiview model (**solid** lines) versus frontal-only model (**dashed** lines).*

The multiview model gives better fits on datasets with non-frontal faces.

The `mem17` measure of fitness is used (the mean error of 17 face points, normalized by the eye-mouth distance, Section 6.2.4). Note that the reference landmarks in this figure are the iBUG re-marked landmarks. For details on the data please see Figure 9.12 and the accompanying text.

For the XM2VTS set, with nearly all frontal faces, the frontal-only model suffices; for sets with three-quarter views the multiview model gives better fits. The slight degradation of fits of the multiview model on the XM2VTS set (barely perceptible in the graph) is caused by misestimates of yaw (e.g. using a three-quarter model on a frontal face).

Details on Figure 9.11: The frontal-only model in the figure is identical to the multiview model but with the pose detector forced to always return frontal. Or to put it another way, it's the HAT model presented in Chapter 8 but with the 300-W face detector (Section 9.5) rather than the OpenCV face detector. (This in itself is sufficient to improve landmark accuracies.) The data sets are the XM2VTS [72], Helen test set [61], and the iBUG test set supplied for the 2013 300-W Challenge. The pupil landmarks aren't provided, therefore the pupil positions for this plot (necessary for the mem17 measure) are taken to be the mean of the eye-corners.

9.6.5 Results on the iBUG landmarks

Figure 9.12 shows the results of running the multiview model on the datasets provided on the 300-W web page. As expected, we do best on the clean frontal faces of the XM2VTS set.

9.6.6 Results on the iBUG hidden set

Figure 9.13 shows the results of running the multiview model on the 300-W hidden set. These curves were generated by the 300-W organizers.

On this set of faces, landmarking accuracy isn't as good with our model as the models of E. Zhou et al. [130] and J. Yan et al. [121]. Note however that converting Stasm to iBUG landmarks causes our model to take an artificial performance hit, and that the test data isn't representative of the faces Stasm was designed for (as discussed on page 128). The next section has results on more representative data.

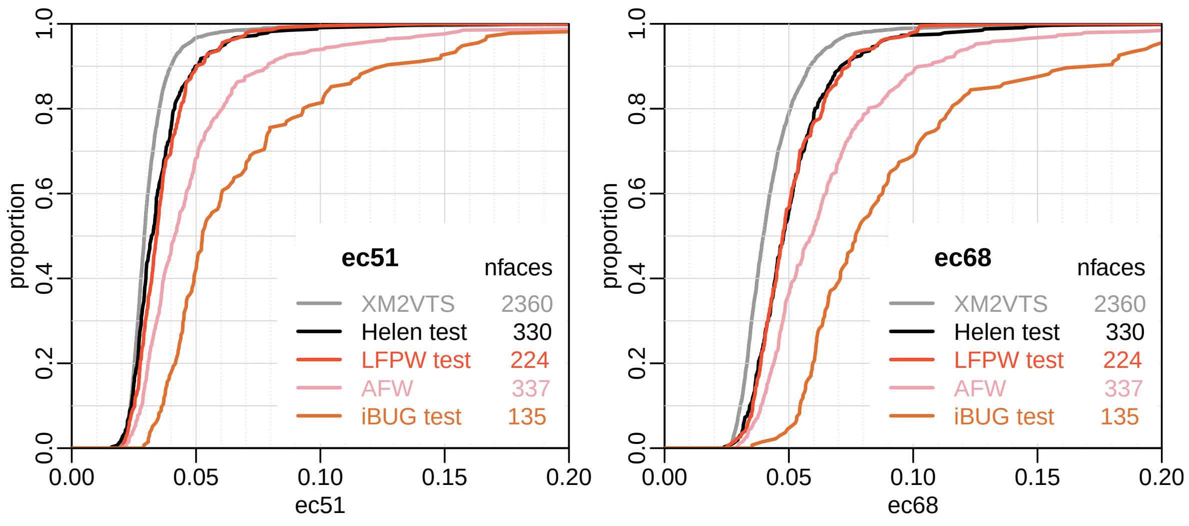


Figure 9.12: *Fits for our multiview model on the datasets provided on the 300-W web page [100], after converting our model's landmarks to iBUG landmarks*
***Left**: **ec51** The mean distance between the 51 automatically located points internal to the face and the reference points, divided by the distance between the reference eye outer corners.*
***Right**: **ec68** The same measure for all 68 landmarks.*
Note that the reference landmarks for all figures in this chapter (except those showing AFLW data) are the iBUG re-marked landmarks [100], not the landmarks from the original databases (XM2VTS [18], Helen [61], LFPW [7], and AFW [132]).

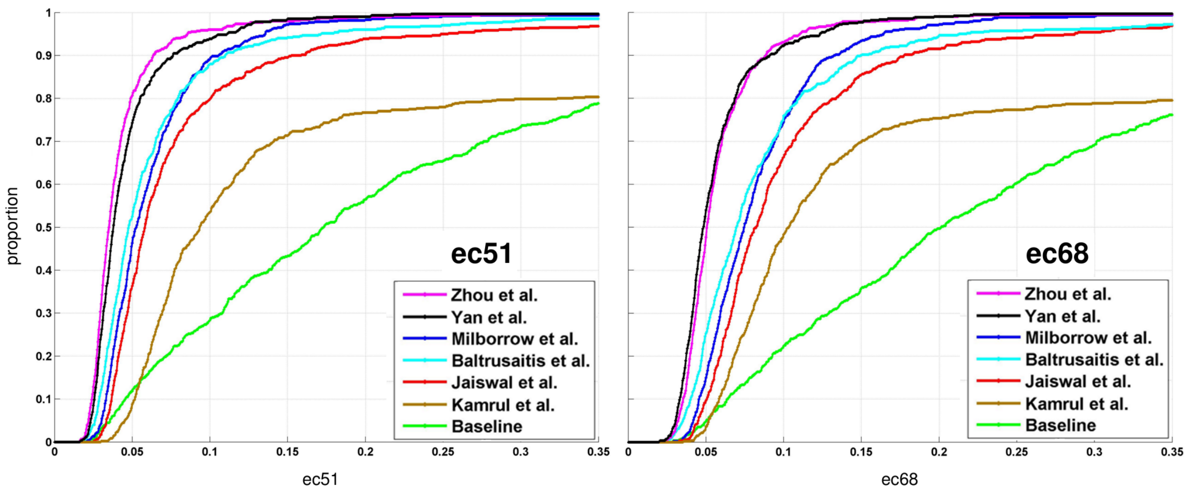


Figure 9.13: *Fits for various models on the hidden set, as reported by the organizers of the 300-W challenge.*
The "Milborrow et al." curve is an early version of the multiview model described in this chapter. The graphs shows the fits after converting our model's landmarks to iBUG landmarks. Note that this artificially difficult set of faces isn't representative of the data that our multiview model is designed for (page 128).
From a figure at `http://ibug.doc.ic.ac.uk/resources/300-W` *(accessed Feb 2016) showing results on the combined indoor and outdoor hidden sets (600 faces in total). For details see [100]. The models in the graph are described in E. Zhou et al. [130], J. Yan et al. [121], Milborrow et al. [80], Baltrusaitis et al. [4], Jaiswal et al. [50], and Hasan et al. [44].*

The landmark conversion issue explains only part of the gap between the curves for the multiview model and Zhou et al. and Yan et al. models. It's likely that also contributing to the better accuracy of their models is the way they work in a sophisticated way from coarse-to-fine resolution. Our model, on the other hand, simply estimates the pose and uses the standard ASM image pyramid, with no consideration of alternative search paths. This is fast but doesn't allow recovery from poor initialization. The search times for the various models on the hidden data weren't made available.

9.6.7 Results on the Helen test data

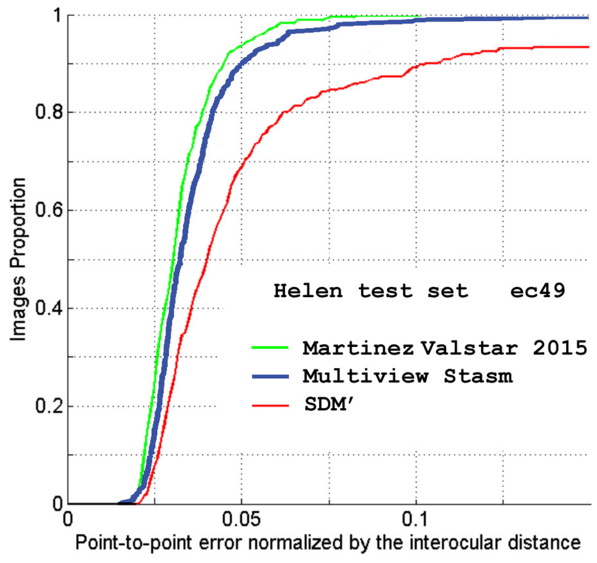
Of the landmarked data provided for the 300-W Challenge, the Helen dataset is the most representative of the applications we have in mind in this thesis (Section 7.3). We thus have a special interest in this set. Figures 9.14 and 9.15 compare our multiview model on this data to very recent models presented by Martinez and Valstar [71] and Tzimiropoulos [110].

Both these models start with the cascaded regression of Dollár et al. [31] and the Supervised Descent Method (SDM) of Xiong and De la Torre [120]. Martinez and Valstar improve convergence of the SDM by using a combined $L_{2,1}$ distance measure instead of the usual L_2 distances, and by using multiple initializations over small spatial displacements and poses (which are then aggregated). The $L_{2,1}$ measure makes comparison between shapes of facial descriptors more robust to partial occlusion (by the subject's hair, for example). A disadvantage as always with L_1 versus L_2 measures is that optimization is more difficult.

The PO-CR method of Tzimiropoulos learns a sequence of average Jacobian and Hessian matrices from the image. These matrices give descent directions for optimization in a subspace orthogonal to local variation in image appearance—variation is removed by projecting it out so only useful information remains. The paper reports excellent fits with this model and robustness on difficult faces.

Figures 9.14 and 9.15 show that our multiview model (after converting the MUCT landmarks to the iBUG landmarks as described on page 128) doesn't quite reach the accuracies of these models on the Helen test data. Our model is however markedly faster than the current implementations of these models.

Details on Figures 9.14 and 9.15: The curves show results on the designated test subset of the Helen data (330 faces) using the re-marked iBUG reference landmarks. This data wasn't used when training any of the models. The curves for the multiview model were generated on a local machine; the other curves are reproduced directly from figures in the two cited papers. We don't know how the performances of the face detectors for the two models compare to the one we use; this probably has some effect on the results.

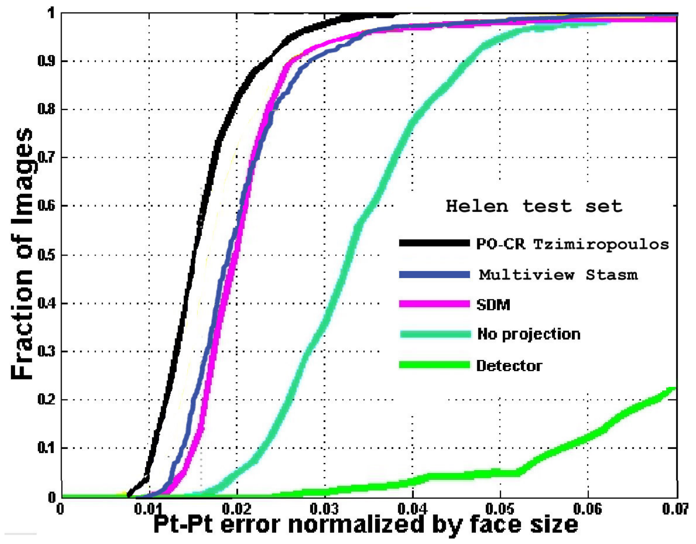


The green curve shows the Martinez and Valstar $L_{2,1}$ *accumulation model [71].*

The blue curve shows the multiview model presented in this chapter.

The red curve shows the Martinez and Valstar implementation of the SDM of Xiong and De la Torre [120]

Figure 9.14: *Comparison of our multiview model to the 2015 model of Martinez and Valstar [71] on the Helen test data [61], after converting our model's landmarks to iBUG landmarks*
This figure is based on Figure 3(b) in [71]; please see that paper for details.
*The curves show the **ec49** measure: the mean distance between 49 automatically located points internal to the face and the reference iBUG points, divided by the distance between the reference eye outer corners. This is the same as the ec51 measure but without two points on the interior of the lips. (In practice the ec51 curve is virtually identical; see the black curve in Figure 9.12.)*



*The **black** curve shows the Tzimiropoulos PO-CR model [110].*

The dark blue curve shows the multiview model presented in this chapter.

The magenta curve shows the SDM of Xiong and De la Torre [120].

Figure 9.15: *Comparison of our multiview model to the 2015 model of Tzimiropoulos [110] on the Helen test data [61], after converting our model's landmarks to iBUG landmarks.*
This figure is based on the top right of Figure 2 in [110]; please see that paper for details. The yellow Chehra curve in the original figure is removed here to reduce clutter and because the Chehra model used the Helen test data in training.
The curves show the mean distance between 49 automatically located points internal to the face and the reference iBUG points, divided by the face size, where the face size is defined as the mean of the face's width and height [132]. Note that all 68 points are used to compute the face size although only 49 points are used to compute fits.

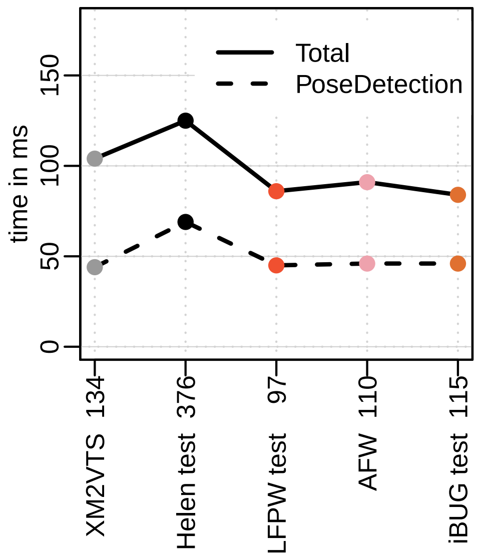


Figure 9.16: *Search times for the multiview model.*

***Total** is for all steps except reading the image from disk and once-off model initialization. Face detection time by the 300-W face detector isn't included.*

***PoseDetection** is for steps b and c in Figure 9.2.*

Times were measured on a 2.9 GHz i7 4910MQ processor with the same datasets as Figure 9.12.

9.6.8 Search times

Figure 9.16 shows search times for the multiview model. In the current implementation, detecting the pose takes about half the total time. Pose detection time could be reduced by relatively easy optimizations to the code. The mean eye-mouth distance in pixels is shown after the dataset name on the horizontal axis. Pose detection takes longer on bigger faces, because the face detectors used for pose detection take longer.

9.6.9 Error per landmark

Figure 9.17 shows the mean fit error per landmark of the multiview model. The figure reflects the level of inherent ambiguity in the landmark positions. It shows that positioning of points interior to the face is better than for exterior points—the worst errors occur on the jaw. Of the interior points, the eyebrows fare the worst, especially the edges of the eyebrows (landmarks 18, 22, 23, and 27). The eye landmarks have the smallest errors.

9.6.10 Effect of pose diversity on fits

A measure of the diversity of poses in a set of faces is the standard deviation of the "aspect ratio" of the faces. The aspect ratio is defined here as the ratio of the eye-mouth distance to the eye-corner distance. We measure the eye-mouth distance here from the centroid of the eye outer corners to the bottom of the bottom lip. (This definition differs from the eye-mouth distance used elsewhere in this thesis because the pupil manual landmarks aren't available.)

This aspect ratio will vary as the pose varies and as the mouth opens. The mean aspect ratio for neutral frontal adult faces is coincidentally close to 1 (its mean value over the XM2VTS faces for example is 0.98).

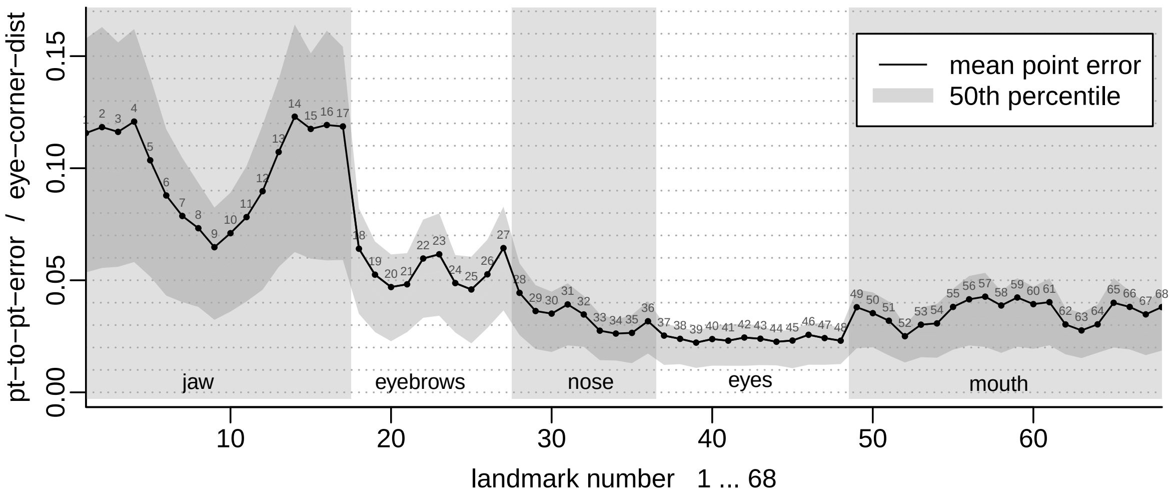


Figure 9.17: *Per landmark error of the multiview model.*
Error here is the distance between the automatic and reference landmarks, divided by the reference eye-corner distance.
The plots shows fits by the model against the iBUG reference landmarks on the 330 faces of the Helen test set [61] and mirrored versions of the same (660 faces in total).

Figure 9.18 shows the linear relationship between the median fit and diversity of the faces. It plots for each dataset the median ec68 versus the standard deviation of the aspect ratio. Not surprisingly, diversity tends to cause worse overall fits.

9.7 Discussion

Using three submodels as described in this chapter is sufficient for many practical applications. Increasing the number of basic poses above three is an obvious future direction. Certainly models for side views are necessary to handle the full gamut of

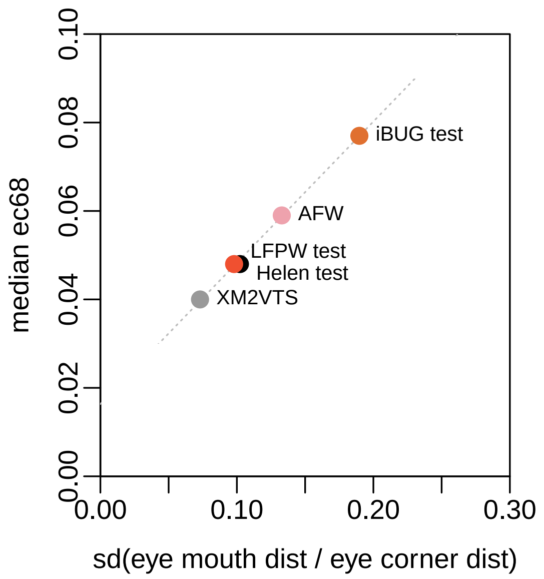


Figure 9.18:

Median fit versus standard deviation of the face "aspect ratio".

This standard deviation is a measure of diversity.

Diversity causes worse median fits.

The colored dots correspond to the points on the median line (proportion 0.5) in the right figure of Figure 9.12.

faces. Models for pitched views may also help, but one quickly reaches the point of diminishing or even negative returns—a problem compounded by misestimates by our pose detector and the combinatorial explosion of yaws, pitches, and rotations.

Ad hoc additions to the model to handle some of the worst cases in Figure 9.9 may be of some benefit. For example, cases like Figure 9.9(c) would benefit if we detected that the mouth was open. However, such additions would affect only a small proportion of faces in realistic test sets and may have unwanted side effects in the bulk of the faces.

Performance would probably be improved with a better training set: our model was trained on the MUCT faces, which weren't photographed in the wild, and have a limited number of three-quarter views.

For efficiency the integral images required by the Viola-Jones face detectors should be shared rather than redundantly regenerated afresh for each of the three detectors. It may be more efficient to use face detectors specialized for rotated views instead of rotating the image. The current implementation applies the face detectors used for pose detection sequentially, but parallelization on multicore processors could appreciably reduce the pose detection times seen in Figure 9.16.

With the multiview HAT model we have pushed the old-fashioned ASM quite far. But there is only so much mileage one can get out of the ASM. It's intrinsically limited by the rigid separation between shape and descriptor models, the simple principal-component shape model, and its selection of a single best point during descriptor matching (instead of making use of a fitness or probability surface). For accuracies very competitive with the current state of the art, we would have to consider replacing this approach with modern techniques that jointly optimize pose, the shape constraints, and descriptor matching. Longer search times may then become a concern.

Chapter 10

Conclusion

This thesis investigated new methods for automatically finding landmarks in images of faces. As stated in the introduction, the overall intention was to introduce methods that perform well enough for use in real-world applications like face remodeling or retouching. The methods investigated are extensions to the Active Shape Model (ASM) of Cootes and Taylor [23].

The first extension replaces the gradient descriptors used in the classical ASM with a form of SIFT descriptors. These descriptors, called Histogram Array Transform (HAT) descriptors, are like SIFT descriptors but can be generated and tested against the image much more quickly than standard SIFT descriptors. Multivariate Adaptive Regression Spline (MARS) models are used to match image descriptors against the model, instead of the Mahalanobis distances used in the classical ASM. The MARS models bring both speed and better predictions in this setting.

This HAT-based ASM is shown to perform well on frontal faces, with close to state-of-the-art accuracies on the BioID data. The last section of Chapter 8 discusses possible improvements to the model and compares the methods used to some other recent approaches.

The model is then extended to handle faces with different poses. This is done by first estimating the target face's pose, rotating the face upright, then applying one of three submodels optimized for frontal, left, or right three-quarter views.

Although this multiview model may not be the most accurate landmarker ever devised, on the data we are interested in it's among the most accurate, fastest, and simplest. On the iBUG 300-W set of unrealistic "difficult" faces, the model's landmarking accuracy wasn't as good as two of the six models compared against. On the more representative Helen data the multiview model approaches the accuracies of the most current models. The last section of Chapter 9 discusses limitations of the model and offers some potential improvements.

Appendix A

Principal components

Principal Component Analysis (PCA) is a widely used method for analyzing and processing multidimensional data. This appendix gives an informal introduction, skipping over much of the mathematics. The presentation is general in nature, rather than focusing just on ASMs. More mathematical treatments may be found in e.g. James et al. [51] or in most multivariate statistics textbooks e.g. Johnson and Wichern [53].

A.1 Principal axes

Before we can discuss principal components we first need to discuss principal axes.

Consider a set of data points in three dimensions (Figure A.1). Of all the different axes we can draw running through the center of the data, one axis will point in the direction of maximum variation of the points. (More precisely, the coordinates or orthogonal projections of the points on this axis have the maximum variance.) This axis is the first *principal axis*. So if the points are scattered in the shape of a rugby ball, the first principal axis will be along the long axis of the rugby ball.

Now we seek the second principal axis. This by definition must be orthogonal to the first principal axis, so must be somewhere on the plane orthogonal to that axis (Figure A.2). Of all the axes we can draw on the plane running through its center, one axis will point in the direction of maximum variation of the points projected onto the plane. This axis is the second principal axis.

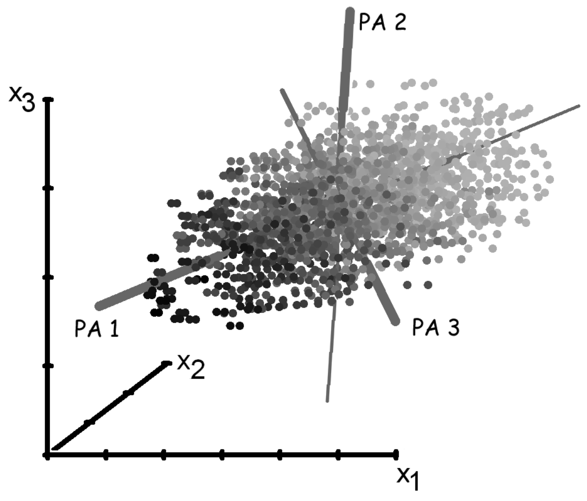


Figure A.1:

Some three-dimensional data with its principal axes.

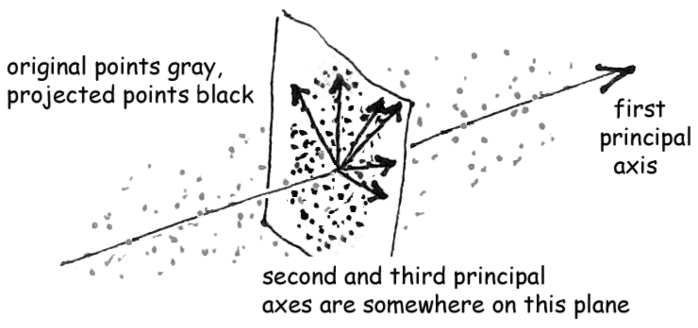


Figure A.2: *With 3D data, the second principal axis lies somewhere on a 2D plane orthogonal to the first principal axis.*

The direction of the second principal axis is the direction of maximum variation of the points projected onto the plane.

The third principal axis must be orthogonal to the first two. Due to the constraints of three-dimensional space, there is only one such direction remaining, and we use it for the third principal axis.

A.2 Principal axes in two dimensions

Figure A.3 shows another example, this time for two-dimensional data. Of all the axes we can draw through the center of the data, one will be in the direction of maximum variance. This is the first principal axis. In this example it's at 157 degrees.

An alternative interpretation is that the first principal axis is as close to the points as possible. More precisely, of all possible axes running through the center of the data, the sum of the squared distances from the points to the principal axis is the lowest. It turns out this alternative interpretation is equivalent to the maximum variance approach—it chooses the same axis.

In two-dimensional space, there is space for only one more orthogonal axis, which becomes the second principal axis.

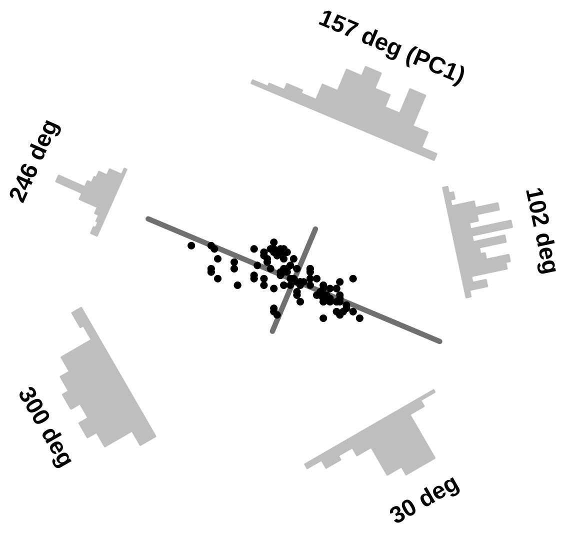


Figure A.3:

Histograms produced by projecting some two-dimensional data at various angles [76].

The projection with the maximum variance gives the direction of the first principal axis. For this data that direction is 157 degrees.

After Flury and Riedwyl [37] Figure 7.1.

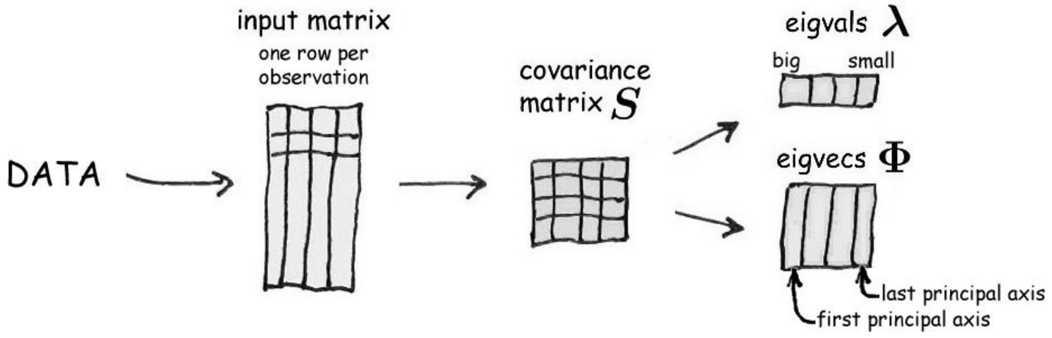


Figure A.4: *Generating the principal axes:*

*(i) Arrange the data into an **input matrix**, one data point (observation) per row. (Some authors use the transpose: one observation per column.)*
In this example the matrix is illustrated with four columns (the data lives in a four-dimensional space, four variables per observation).

*(ii) Generate the **covariance matrix** from the data in the input matrix.*

*(iii) Use a standard software routine to extract the **eigenvectors** and **eigenvalues** from the covariance matrix.*

Because the eigenvectors are extracted from the covariance matrix, they are the principal axes of the data. The eigenvalues give the variance of the data along each principal axis.

A.3 Multiple dimensions

We can sketch only two or three dimensions on a printed page, but if the data lives in a space with more than three dimensions, we can continue seeking further principal axes. Each principal axis is orthogonal to the previous principal axes and points in the direction of maximum remaining variation of the data points.

For shape models, we treat shapes as vectors in multidimensional space (Section 3.1). This space has $2n$ dimensions, corresponding to the two coordinates of n shape points.

A.4 Principal axes are eigenvectors of the covariance matrix

It turns out that the above process of seeking principal axes of the data can be performed by generating the eigenvectors of the covariance matrix of the data. Eigenvectors are mathematical objects which are the principal axes of the data if they are generated from its covariance matrix.

Figure A.4 illustrates how we generate principal axes. The two arrows on the right of the covariance matrix in the figure can be thought of as a function, a MATLAB or C++ routine if you will, that takes as input the covariance matrix and generates two objects:

(i) The square eigenmatrix $\boldsymbol{\Phi}$. Each column of $\boldsymbol{\Phi}$ is an *eigenvector* or principal axis. Each principal axis is of unit length and is orthogonal to the other principal axes.

(ii) The vector of scalar *eigenvalues* $\boldsymbol{\lambda}$. Each eigenvalue is a measure of importance of the corresponding principal axis. More precisely, the value of an eigenvalue is the variance of the data in the direction of the principal axis. The first principal axis has the largest eigenvalue, and the last principal axis has the smallest eigenvalue.

Internally the software doesn't generate the eigenvectors using the variance maximizing process described in the previous sections—it uses more efficient and numerically stable methods—but the net effect is the same.

A.5 An alternative coordinate system

A point in multidimensional space can be specified by its coordinates on the conventional axes of that space. Alternatively we can specify the point by its coordinates on the principal axes. The principal axes form an alternative coordinate system.

We can see this idea being used in the shape model formula (Equation 3.5), and reproduced here:

$$\hat{\boldsymbol{x}} \;=\; \bar{\boldsymbol{x}} \;+\; \boldsymbol{\Phi b}\,. \tag{A.1}$$

Figure A.5 illustrates this equation in two dimensions. In 2D, the equation expands to

$$\hat{\boldsymbol{x}} \;=\; \bar{\boldsymbol{x}} \;+\; b_1\boldsymbol{\Phi}_1 \;+\; b_2\boldsymbol{\Phi}_2\,. \tag{A.2}$$

The left side of the equation is a point $\hat{\boldsymbol{x}} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ in the original coordinate system. The right side $\bar{\boldsymbol{x}} + \boldsymbol{\Phi b}$ specifies the same point in terms of the principal axes: the right side is the offset $\bar{\boldsymbol{x}}$ to the center of the cloud of data, plus the position of the point specified by its coordinates $\boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ on the principal axes $\boldsymbol{\Phi}$ of the data.

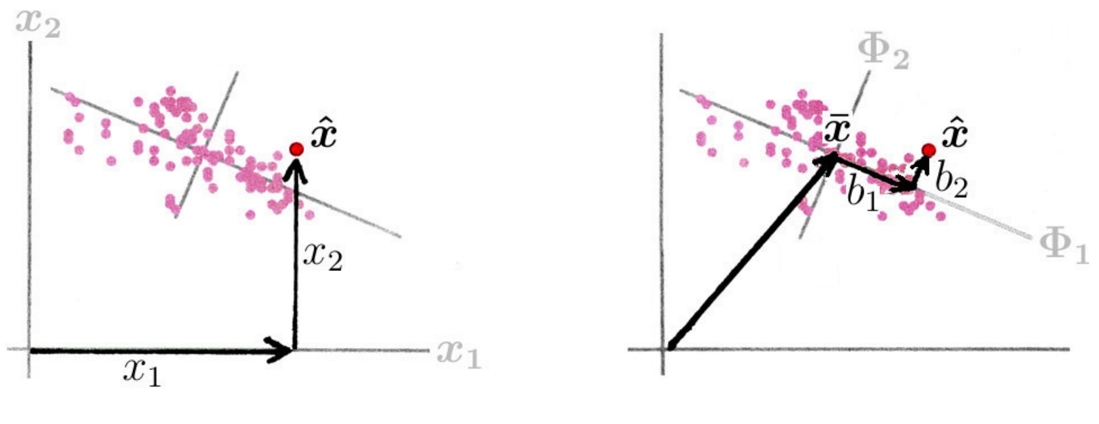


Figure A.5: *The same point $\hat{\boldsymbol{x}}$ represented in two different coordinate systems.*
Left *Original conventional coordinates.*
Right *Principal component coordinates* $\bar{\boldsymbol{x}} + b_1\boldsymbol{\Phi_1} + b_2\boldsymbol{\Phi_2}$.

A.6 Principal components

The *principal components* (PCs) of the data points are their coordinates on the principal axes. These coordinates are also known as the *scores*.

To elaborate, the coordinate of a point on the first principal axis is the first principal component (PC) of that point. Likewise, the coordinate of the point on the second principal axis is its second principal component, and so on for further axes. In the 2D example in Figure A.5 we call these principal components b_1 and b_2.

However, it should be mentioned that people often refer to the principal axes themselves as the principal components.

A.7 Principal Component Analysis

In traditional *Principal Component Analysis* (PCA), researchers (often in the life sciences) seek to understand the data by examining the relationship between the data variables and the principal axes.

For example, if the variables are test results for several subjects by school students, the first principal component might represent overall academic ability, the second a contrast between science and art subjects, and with the remaining axes having no simple interpretation.

We don't need to do this kind of analysis for shape models, although it can be intriguing to see how the face shape varies with different principal components (Figure 3.4 on page 32).

Having said all that, it should also be mentioned that people often use the term "Principal Component Analysis" in a very general sense to mean any manipulation of the data that involves the principal axes.

A.8 Dimensionality reduction and compressing data

What is the advantage of using the principal axes over the original axes? One advantage is that we can use the principal axes for optimally reducing dimensionality. For example, say we were allowed only one axis to specify points in the 3D space of Figure A.1. We would capture as much variation as possible if we used the first principal axis and specified the points by their coordinate on that axis. We project the data down to the one dimension in the direction of the axis, but by choosing that direction carefully we lose as little information as possible.

We can optimally project the data down to two dimensions by using the first two principal axes. This compresses the data down to the plane spanned by the first two principal axes. We capture more information than when we compressed down to only one axis.

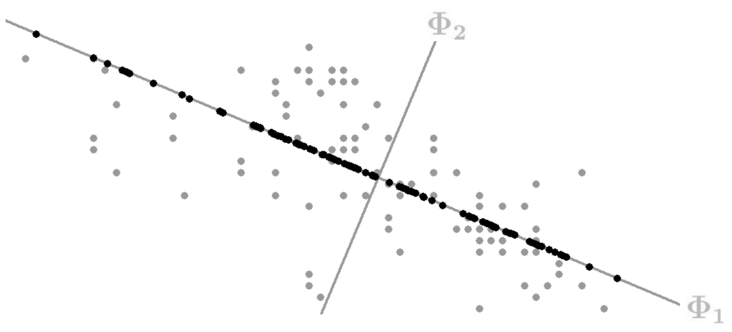


Figure A.6:

Some 2D data (gray dots) and its reconstruction (black dots) along the first principal axis.

And so on for further axes. If we use all the principal axes to specify the data, no compression would take place, and in that case we may as well use the original axes.

Figure A.6 shows a 2D example. The gray points are the original data. The black points are the first principal component. The black points are a 1D approximation of the gray points.

To transmit the positions of the gray points, we need to send two coordinates per point. We can greatly reduce the amount of data to be sent by transmitting only the position of the black points instead. We would do this by transmitting the offset and direction of the first principal axis, followed by the single coordinate of each point on the axis. This roughly halves the bandwidth required, but the receiver can reproduce the gray points only approximately. It's lossy compression.

On the other hand, if the points *should* be distributed along the line, and any offset from the line is due to noise, then by reducing the data down to 1D we *denoise* the data. The data is intrinsically one-dimensional although it exists in a two-dimensional space. Admittedly with only a few dimensions these examples are a bit contrived—the ideas are more useful in higher dimensional spaces.

Figure A.7 shows some data that is intrinsically two-dimensional—the points are on a plane—although they exist in a three-dimensional space. In an ASM shape space (which for a 77 point model has $77 \times 2 = 154$ dimensions), the bulk of the variation in the training faces is accounted for by the first thirty or so principal axes. The remaining principal axes mostly represent noise or quirks of the training set. We can represent the true distribution of shapes underlying the training sample more accurately by keeping only the important axes—the thirty or so axes that represent the intrinsic dimensionality of face shapes. Section 3.9 gives an extended example.

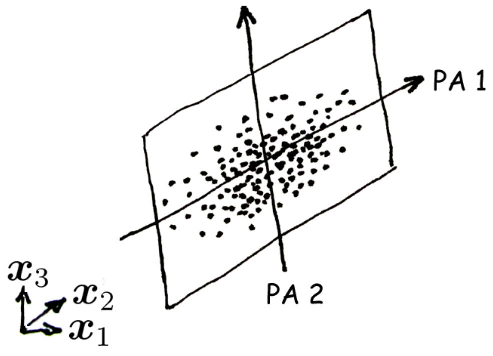


Figure A.7: *2D data in a 3D space.*

Although the points are in a 3D space, they are on a 2D plane in that space.

There is no variance in the direction of the third principal axis. The third eigenvalue is zero.

A.9 Characteristics of principal components

The principal axes are derived from the covariance matrix—in Figure A.4, the eigenvector matrix “sees” only the covariance matrix, not the data itself. Properties of the data not captured in the covariance matrix aren't represented by the principal axes.

This section discusses some characteristics of principal components, making use of examples of data in two dimensions. Most of these characteristics follow from the above-mentioned covariance-matrix view of the world.

In Figure A.8(a), the variables x_1 and x_2 are highly correlated and there is variation only along a line. The data can be fully described by the first PC. Although the data is in a two-dimensional space it's intrinsically one-dimensional. This data is perfectly suited to PCA.

Figure A.8(b) shows the other extreme in terms of correlation. The variables x_1 and x_2 are uncorrelated and the variance of the data is the same in all directions. The data cloud doesn't have a “direction”, and thus there is no axis of most variation. If we were forced to use just one dimension to characterize this data, we could choose either the horizontal or vertical axis, or indeed an axis at any angle through the center of the data. None of these choices is better than any other. We really need two dimensions to describe uncorrelated data like this with reasonable accuracy. In practice there will be at least some correlation in almost all datasets, especially in high dimensions.

Another type of data is shown in Figure A.8(c). There is obviously a relationship between x_1 and x_2, but it's nonlinear. This nonlinearity won't be reflected in the covariance matrix. Dimensionality reduction using PCs is often ineffective for data with strong nonlinear relationships, because projecting the data onto the principal axes washes out the nonlinearity and thus discards an important characteristic of the data.

There are techniques related to principal components for reducing the dimensionality of nonlinear data (e.g. Hastie and Stuetzle [46], Roweis and Saul [99]). Alternatively, it's sometimes possible to first transform the data so that it's more linear in the transformed

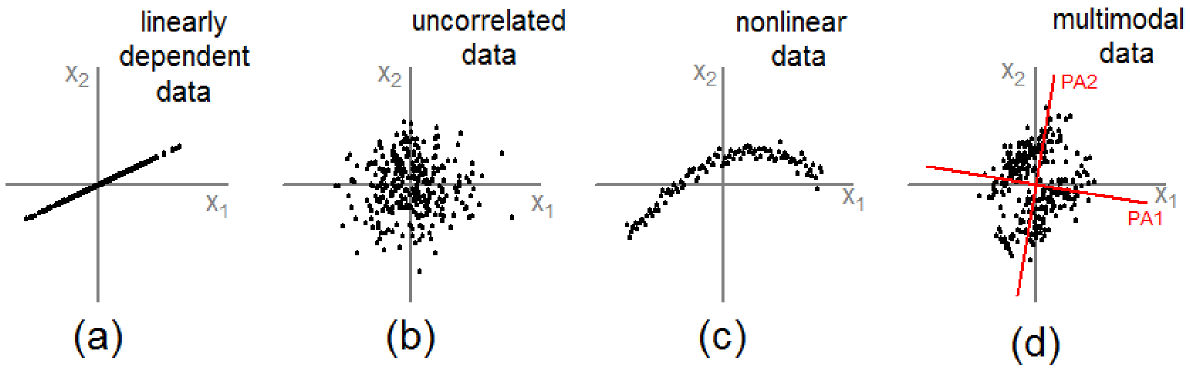


Figure A.8: *Clouds of data with different shapes.*

space, then do the PCA in the transformed space. It may be sufficient to take the log, square, or square root of some of the variables, or perhaps convert them to polar coordinates.

Finally, Figure A.8(d) shows data in two clusters. The principal axes are shown in red. They don't reveal much about the data in this example. Crucial information will be lost if we project the data down to one principal axis. Although this data can be represented using Equation A.1, that would be a misleading parametrization of the data. Constraining the principal components (Section 3.10.2) will constrain the points to the area around the origin, where the density of the data is in fact low.

For this kind of data, where there are two groups, what is usually more important is the axis that optimally separates the groups—the *linear discriminant* axis—but that is another topic.

Shadows

Another way of building intuition about what kind of data is best represented by principal components is to consider the shapes of clouds of data in three dimensions, and the shadows of those clouds. A shadow is a two-dimensional projection of a three-dimensional object.

Ellipsoidal data (such as Gaussian data) lends itself to PCA because in any direction the shadow of an ellipsoid is an ellipse, with natural major and minor axes (Figure A.9). The shadow of a rugby ball is an ellipse. The overall shape of the data doesn't change as we reduce dimensionality.

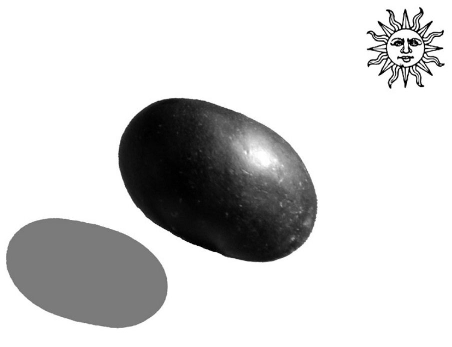


Figure A.9:

The shadow of an ellipsoid is always an ellipse or a degenerate ellipse.

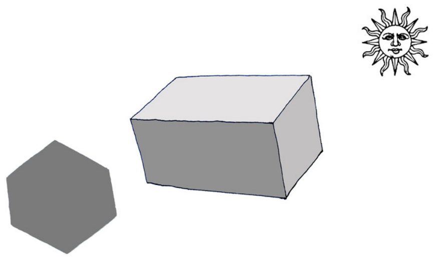


Figure A.10:

The shadow of a box is in general not a rectangle.

(These figures doesn't show the principal components; the shadows are in arbitrary directions.)

On the other hand, data uniformly distributed in a box shape is often inappropriate for PCA, because the shadow of a box is in general not rectangular (Figure A.10). If the points are uniformly distributed in the box, the first principal axis will be roughly along the longest diagonal within the box, from corner to corner. This would probably not be a helpful way to parameterize this (somewhat unusual) data, unless the box is very long or thin.

And data with points grouped into two or more separate clouds is usually inappropriate for PCA, because the groups will often merge into one unsightly blob in the shadow. That said, a preprocessing step using PCA to project high-dimensional multi-centered data down to its intrinsic dimensionality may still be useful before further analysis.

Higher dimensions

We can extrapolate our experience from these 3D examples to higher dimensions. However, it must be said that although useful for building intuition this kind of mental extrapolation can break down when the number of dimensions is very high. One of the manifestations of the curse of dimensionality is that feasibly sized samples of data are always sparse when the number of dimensions is very high. The word "cloud" becomes misleading because no point is near another, and the points can't be said to have an overall characteristic shape (e.g. Hastie et al. [45] Section 2.5, Bishop [9] Section 1.4). PCA is still useful in high dimensions (in fact often more so), it's just that our geometric intuition about the dispersion of points in a 2D or 3D cloud may not generalize when the number of dimensions is high.

For shape models we partially finesse the curse of dimensionality by making distributional assumptions of approximate normality. In other words, we make the reasonable assumption that the data is approximately Gaussian even though we can't formally infer that—the dimensionality of the space is too high to make a formal inference from a realistically sized set of training shapes (Section 3.16).

Importance versus variance

Principal Component Analysis equates "importance" with "high variance". That may not be valid. We have already seen the multimodal example in the right of Figure A.8 where the highest variance axes don't reveal what is important about the data.

The letters **O** and **Q** are written differently by different people. A shape model with only the high-variance principal axes will account only for the overall circular shape of the two letters and variations on that shape—but what matters isn't necessarily the overall shape but the tiny glyph that distinguishes an **O** from a **Q**. This glyph may not be represented in the high-variance principal components.

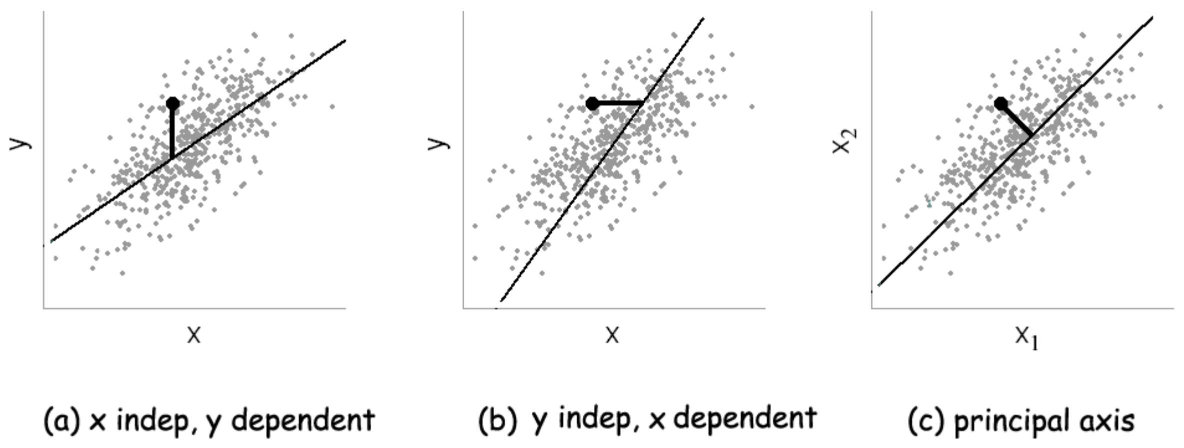


Figure A.11: *Different kinds of linear regression, and principal components. The short black line shows the residual for one point. The definition of the residual depends on the form of regression.*

A.10 PCA versus linear regression

In a linear regression model, the response or dependent variable is modeled as a constant offset plus a weighted sum of one or more independent variables. The left plot of Figure A.11 shows an example for a single independent variable x. The gray dots are the observed values of x and the response y. The regression line is the model's estimate of the value of y for any x. The *residuals* are the vertical distances between the observed values and the line.[1] The regression line is positioned so the residuals are as small as possible (more precisely, the sum of the squared residuals is minimized).

In regression there is an asymmetry between the dependent and independent variables. If the roles of x and y are swapped (so y is the independent and x the dependent variable), the residuals will be calculated differently and the regression line will be different (middle plot of Figure A.11).

How do these regression lines compare to the first principal axis? In PCA, the variables enter the analysis on an equal footing—there isn't a distinction between independent and dependent variables. The residuals are the shortest distances from the points to the regression line, which means that the residuals are orthogonal to the principal axis (right plot of Figure A.11).

We remark also that the line passes through the mean of the two variables in all three graphs, and the principal axis lies between the two regression lines.

[1]Some details: The assumption here is that we know the true value of x, whereas the true value of y is obscured by noise. The way the residuals are calculated is determined by the way noise is deemed to enter the regression model (which may be determined by the nature of the data under analysis or may be a choice that the person building the model is free to make). Here we estimate a noisy y as a linear function of x. It would be incorrect to use this regression line to estimate x given y; for that we should use the middle plot in Figure A.11.

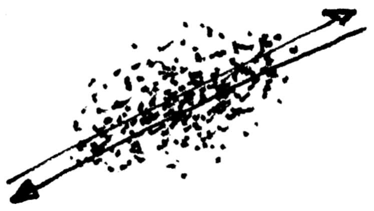


Figure A.12: *The principal axis can point in either direction. The shape of the cloud of points doesn't determine which direction is better.*

A.11 Implementation notes

The eigenvectors of the covariance matrix are unique after normalization to unit length, but only up to a sign change. There is a sign ambiguity because an axis can point in either direction and mathematically there is nothing in the covariance matrix to establish which direction is better (Figure A.12). To remove the ambiguity, after generating the eigenvectors our software negates an eigenvector if necessary so its largest element is positive. (There will still be ambiguity if the two largest elements are identical in magnitude but opposite in sign. This is very rare in practice.)

Even though a covariance matrix is at least positive-semidefinite and thus has non-negative eigenvalues, in practice numeric error may introduce very slightly negative eigenvalues. Furthermore, because of numeric issues different implementations and machines can give slightly different values for the smallest eigenvalues. Therefore negative or very small eigenvalues are set to zero in our software for training shape models. Eigenvalues less than 10^{-6} times the largest eigenvalue get zeroed; the 10^{-6} figure is fairly arbitrary.

These techniques give more consistency across compilers and machines, which is especially necessary when running test scripts that check the generated eigenvalues.

A.12 History of PCA

PCA has been reinvented many times in different contexts. It's thus sometimes known by different names, such as the *Karhunen–Loève transform* or the *Hotelling transform*. The principal axes are sometimes called the *characteristic vectors*, the *modes of variation*, the *principal directions*, or the *empirical orthogonal functions*.

The two papers commonly cited as pivotal are those by Pearson [89] in 1901 and Hotelling [49] in 1933. Hotelling coined the term "principal component". The ideas in these papers became widespread only after computers to do the calculations become easily available.

Appendix B

Mahalanobis distances

This appendix describes Mahalanobis distances. It first gives an overview. It then discusses various properties of the covariance matrices used when calculating Mahalanobis distances.

B.1 Overview

The Mahalanobis distance is a measure of the distance of a point from the center of the cloud of points, incorporating the probability distribution of the data.

B.1.1 Definition of the Mahalanobis distance

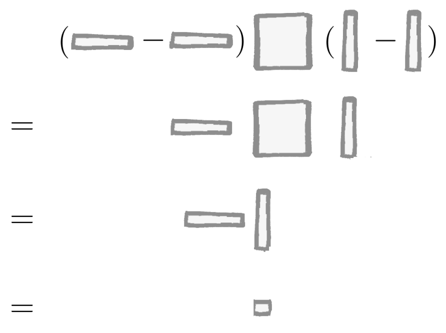
More precisely, the *Mahalanobis distance* of a point $\boldsymbol{x}$ from the center $\bar{\boldsymbol{x}}$ of the distribution of those points is defined as:

$$\text{Mahalanobis distance} \;=\; \sqrt{(\boldsymbol{x}-\bar{\boldsymbol{x}})^T \; \boldsymbol{S}^{-1} \; (\boldsymbol{x}-\bar{\boldsymbol{x}})}\;, \tag{B.1}$$

where $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ are vectors and $\boldsymbol{S}$ is the covariance matrix of the points.[1]

Although the expression on the right of the equation is a matrix expression, it evaluates to a scalar. Pictorially:

$$(\boldsymbol{x}-\bar{\boldsymbol{x}})^T \quad \boldsymbol{S}^{-1} \; (\boldsymbol{x}-\bar{\boldsymbol{x}})$$



(B.2)

[1]This equation is the same as the equation for descriptor distances (Equation 5.2 on page 64), except that here we use the more general symbol $\boldsymbol{x}$ for the points rather than the $\boldsymbol{g}$ we used for descriptor vectors.

Note from the equation that if the covariance matrix $\boldsymbol{S}$ is an identity matrix, then the Mahalanobis distance is identical to the Euclidean distance from the center of the data:

$$\text{Euclidean distance} \;=\; \sqrt{(\boldsymbol{x}-\bar{\boldsymbol{x}})^T\;(\boldsymbol{x}-\bar{\boldsymbol{x}})}\;. \tag{B.3}$$

For one-dimensional (scalar) data, Equation B.1 simplifies to

$$\text{Mahalanobis distance in 1D} \;=\; \frac{x-\bar{x}}{s} \tag{B.4}$$

where $\bar{x}$ is the mean of the data and s is its standard deviation. This equation gives the distance of the point x from the center of the data in units of standard deviation. Mahalanobis distances are a multi-dimensional generalization of this measure.

B.1.2 Mahalanobis distances in 2D

We now illustrate Mahalanobis distances using a two-dimensional example. Figure B.1 shows some points drawn from a two-dimensional Gaussian distribution. The setup is something like an astigmatic dart player throwing darts at the center of a dart board. (The points in both sides of the figure are identical.)

The density of the points is highest towards the center. Because the points are correlated the ellipses are tilted. The ellipses are equi-probability contours, meaning that the chance of a point falling at any place on the ellipse is the same. (To avoid issues raised by infinitesimal probabilities, assume that the contours have a small non-zero width.) The shape and orientation of the ellipses is determined by the covariance matrix of the data. All points on an ellipse have the same Mahalanobis distance from the center.

In the left figure, P and Q are the same Euclidean distance from the center. But the density of points at P is lower than at Q. Point P is less likely than point Q. The Mahalanobis distance of P is greater than that of Q.

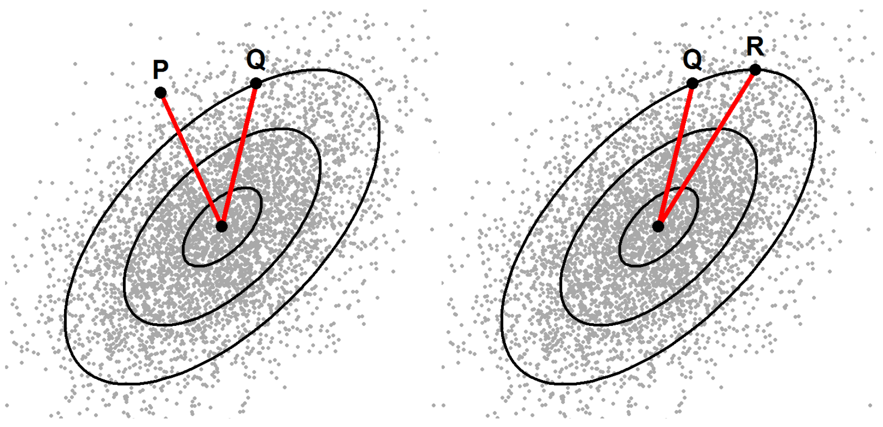


Figure B.1: *Mahalanobis distances on correlated 2D data.*
The figures show some points drawn from a two-dimensional Gaussian distribution. The ellipses are equi-probability contours.
Left *Points P and Q are the same Euclidean distance from the center.*
Right *Points Q and R are the same Mahalanobis distance from the center.*

In the right figure, points Q and R are the same Mahalanobis distance from the center. They are equally likely to occur. Points Q and R are equally far from the center in this sense, even though their Euclidean distances are different.

If the data had an identity covariance matrix, the ellipses would be circles, and Mahalanobis and Euclidean distances would be identical.

B.1.3 Multidimensional data

Mahalanobis distances incorporate the probability structure of the data; they incorporate the shape of the cloud. For three-dimensional Gaussian data, the ellipses become ellipsoids like rugby balls (e.g. Figure A.1 on page 137). The nested ellipses in Figure B.1 become nested ellipsoids. All points on the surface of an ellipsoid are at the same Mahalnobis distance from the center of the data. We can't draw data in higher dimensions, but the same idea applies. Points with equal Mahalanobis distances from the center are equally likely.

In the context of ASM descriptor models (Section 5.3 on page 63), if our 1D descriptors have 9 elements we work in a 9 dimensional space. Each descriptor is a point in that space. The training descriptors for a landmark are a swarm of points in a 9 dimensional cloud. At the center of the cloud is the mean descriptor. A descriptor that has a low Mahalanobis distance is close to the center, close to the mean descriptor.

Mahalanobis distances assume that the data has an ellipsoidal distribution. By far the most-used ellipsoidal distribution is the Gaussian distribution. In practice the data may not be exactly ellipsoidal, but Mahalanobis distances may be a good enough approximation, or at least better than Euclidean distances. The covariance matrix of the data determines the shape of the ellipsoids—precisely how that works requires an understanding of quadratic forms, which we won't go into here.

It should be mentioned that the term "Mahalanobis distance" is also used for a similar measure of distance between *groups* of data (rather than between a point and the center of the data, which is the sense in which we have been using the term). It should also be mentioned that P. C. Mahalanobis was an Indian statistician who studied distances between ethnic groups.

That concludes our overview of Mahalanobis distances. For more a more mathematical treatment see a multivariate statistics textbook such as Johnson and Wichern [53]. The rest of this appendix focuses on the properties of the covariance matrices necessary to calculate Mahalanobis distances.

B.2 Invertibility of covariance matrices

The covariance matrix must be invertible because we use its inverse when calculating Mahalanobis distances (Equation B.1). A covariance matrix is invertible if the variables used to calculate it are linearly independent. In the ASM setting, this means that there must be more images than elements in the descriptor vector, or even more images than that if some descriptors happen to be duplicated (the gray texture at a landmark of one training image matches that of another).

In the context of ASMs, this isn't a concern with realistic numbers of training images. However, the Stasm test scripts build a model with a small number of images for speed: the resulting covariance matrices must be forced positive-definite and thus invertible. We do the forcing using the technique described in Section B.4.

Generally speaking, computation of the inverse of a matrix is to be avoided, because inversion is sensitive to numerical issues (e.g. [17]). However, in practice these issues make no material difference for descriptor models. We just remark that it's possible to compute Mahalanobis distances without explicitly inverting the covariance matrix, by first performing a Cholesky decomposition and then solving by conventional substitution. In MATLAB code (and using notation based on Equation B.1):

```
L = chol(S);
z = L \ (x - x_bar);
mahalanobis_distance = sqrt(z' * z);
```

B.3 Positive-definiteness of covariance matrices

This section discusses positive-definiteness of covariance matrices. We will get to the definition of positive-definiteness shortly, but first it should be explained why positive-definiteness is a topic of interest. For meaningful Mahalanobis distances (Equation B.1 on page 148), the inverted matrix $\boldsymbol{S}$ must be positive-definite. If it isn't positive-definite, the Mahalanobis distance can become negative. The usual meaning of distance breaks down if distances can be negative—for example, the shortest distance between two points may not be a straight line. Distances should always be nonnegative, and zero only if there is a perfect match. (See for example Duda et al. [33] Section 4.6 which discusses distance as a *metric*.)

A covariance matrix is always at least positive-semidefinite. It's strictly positive-definite if the variables are linearly independent, or equivalently in our ASM context, if we have a sufficient number of training images. For realistic descriptor models this will always be the case, so why is it necessary to even raise the issue of positive-definiteness? Here is a brief answer, which is elaborated on in other sections in the body of this thesis. During testing and debugging it's useful to build ASM models with a small number of images, and the covariance matrices will thus be only positive-semidefinite (Section B.2). Furthermore, if there are missing points (Section 5.5) or if we "trim" the matrices (Section 5.7), they no longer will be true covariance matrices, and may not be positive-definite. Therefore we force them positive-definite using the technique described in Section B.4.

B.3.1 Positive-definite matrices

A square matrix $\boldsymbol{S}$ is *positive-definite* if $\boldsymbol{x}^T\boldsymbol{S}\boldsymbol{x} > 0$ is true for all non-zero vectors $\boldsymbol{x}$. (We assume that the dimensions of $\boldsymbol{S}$ and $\boldsymbol{x}$ are conformable.)

Similarly, a square matrix $\boldsymbol{S}$ if *positive-semidefinite* is $\boldsymbol{x}^T\boldsymbol{S}\boldsymbol{x} \geq 0$ is true for all non-zero vectors $\boldsymbol{x}$. Notice that the only difference from the above definition is that here we use $\geq$ instead of $>$.

Readers unfamiliar with positive-definiteness don't really need to come to grips with the above definitions for our purposes. You just need to know that positive-definiteness is a property of some matrices and is important for Mahalanobis distances, as was discussed above.

Very loosely speaking, positive-definite matrices are like positive numbers, and positive-semidefinite matrices are like nonnegative numbers. Note that a *positive* matrix is one which all the elements are positive, and is in general not the same animal as a positive-definite matrix. A positive-matrix may or may not be positive-definite, and vice versa.

Figure B.2 illustrates how the quadratic form $\boldsymbol{x}^T\boldsymbol{S}\boldsymbol{x}$ varies as $\boldsymbol{x}$ varies for a two-dimensional vector $\boldsymbol{x}$. The overall shape of the surface is dependent on the positive definiteness of the matrix $\boldsymbol{S}$. Only when $\boldsymbol{S}$ is positive-definite do we get a bowl-shaped surface that is always greater than zero except when $\boldsymbol{x}$ is zero. This is a property that we expect of distances. The equi-probability ellipses in Figure B.1 on page 149 are slices through the bowl.

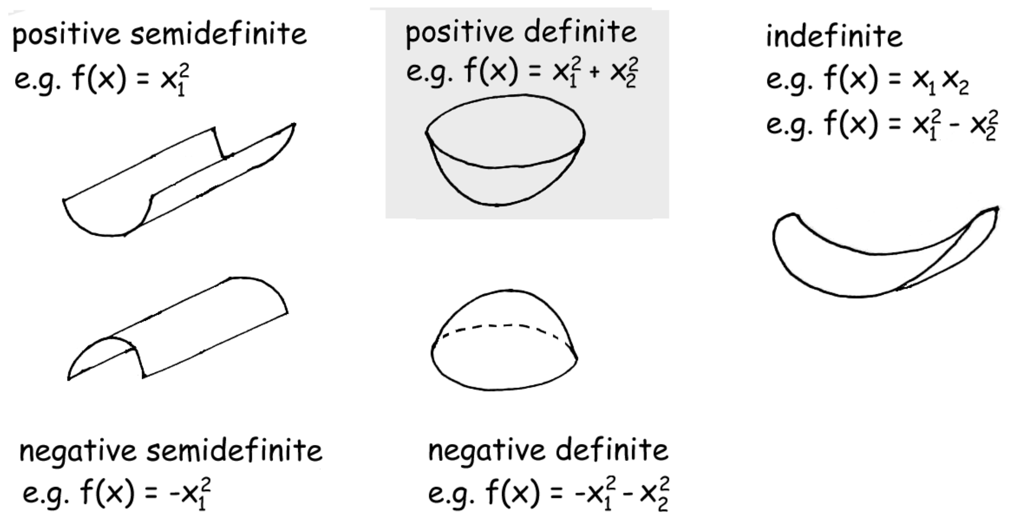


Figure B.2: *Quadratic forms in two dimensions.*

The curves plot the value of the quadratic form $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{S}\boldsymbol{x}$ *as the vector* $\boldsymbol{x} = [x_1, x_2]$ *varies.*

The positive-definiteness of $\boldsymbol{S}$ *determines the overall shape of the curve.*

B.3.2 Properties of positive-definite matrices

Some properties of positive-definite matrices that are important in the current context:

- A covariance matrix is always at least positive-semidefinite. It's strictly positive-definite if the variables are independent.
- Positive-definite matrices are invertible, but positive-semidefinite matrices aren't invertible. If a matrix is positive-definite then so is its inverse.
- The eigenvalues of a positive-definite matrix are positive, and the eigenvalues of a positive-semidefinite matrix are nonnegative.

B.3.3 Testing for positive-definiteness

One way of testing for positive-definiteness of a matrix is to calculate its eigenvalues—if all the eigenvalues are positive the matrix is positive-definite. However calculating eigenvalues is quite slow (although in the context of ASMs this test is needed only when training the model, not when using it).

Testing for positive-definiteness can be done more efficiently by attempting a Cholesky decomposition of the matrix. If a diagonal entry of the attempted Cholesky decomposition isn't positive then the matrix isn't positive-definite. Taking into account numerical limitations, if a diagonal entry is less than say 10^{-16}, we can say that the matrix isn't *numerically* positive-definite. Such matrices can be forced positive-definite as described in the next section.

B.4 Forcing positive-definiteness

A square symmetric matrix $\boldsymbol{S}$ can be forced to a "nearby" positive-definite matrix using the algorithm in Figure B.3. (The algorithm yields a positive-definite matrix that is near the original matrix in the sense that its eigenvalues are close and it will give similar Mahalanobis distances, up to discrepancies caused by forcing any nonpositive eigenvalues positive.)

The algorithm in the figure easily suffices for the covariance matrices in our descriptor models, where the absolute sum of nonpositive eigenvalues is only a small fraction of the sum of all eigenvalues (much less than one percent). Thus for our models the covariance matrices are "nearly positive-definite". That won't be the case if large proportions of the observations are missing. More sophisticated methods are available in that case. More analysis and methods can be found in Higham [48] and a little more accessibly in Lucas [69].

To my knowledge the algorithm in the figure was first presented explicitly in my master's thesis [75], although given its simplicity it must have been known to mathematicians for a long time. Another technique for forcing a matrix positive-definite is to add a

input a symmetric square matrix $\boldsymbol{S}$ that may not be positive-definite

1. Perform a spectral decomposition $\boldsymbol{S} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues.
2. Change the diagonal matrix $\boldsymbol{\Lambda}$ to $\boldsymbol{\Lambda}_{new}$ by replacing elements on its diagonal that are smaller than λ_{min} with λ_{min}.
   The constant λ_{min} is a small positive value, say λ_1 / 10^8, where λ_1 is the largest eigenvalue.
3. Reconstruct the matrix: $\boldsymbol{S}_{new} = \boldsymbol{Q}\ \boldsymbol{\Lambda}_{new}\ \boldsymbol{Q}^T$. The matrix $\boldsymbol{S}_{new}$ is positive-definite, because all diagonal elements of $\boldsymbol{\Lambda}_{new}$ are positive.

output a symmetric square positive-definite matrix $\boldsymbol{S}_{new}$ that is near the original matrix $\boldsymbol{S}$

Figure B.3: *Forcing a matrix positive-definite.*

sufficiently large positive scalar to the diagonal elements, but the approach described in the figure seems simpler (it doesn't require determination of the magnitude of a "sufficiently" large scalar).

B.5 Efficiency when calculating Mahalanobis distances

Some basic techniques can improve runtime efficiency when calculating Mahalanobis distances:

- We can store the inverted covariance matrix with the model, so the inversion operation isn't required when calculating distances when using the model.
- We can exploit the symmetry of the (inverted) covariance matrix to roughly halve computation time of distances.
- To avoid square roots, we can compare squared distances, rather than the distances themselves.

Appendix C

Model Building and Evaluation

This appendix discusses some basic principles of model building and evaluation. It's not uncommon to see experimental results in published papers that are invalid because these fundamental principles are disregarded. The discussion below is general in nature and not limited to models for landmarking faces. Hastie et al. [45] Chapter 7 is recommended for a more complete presentation of these topics.

C.1 Overfitting

An overfit model fits the training data well but won't give good predictions on new data. The idea is that the training data captures the underlying structure in the system being modeled, plus noise. We want to model the underlying structure and ignore the noise. An overfit model models the peculiarities and specific realization of noise in the training data and is thus too specific to that training data.

An example is a model that beautifully models stock market prices in historical data but fails miserably at predicting tomorrow's stock prices. Another example is a landmarking model trained only on the BioID data. The model may work well at locating landmarks on BioID faces, but if it's overfit to the BioID data it won't work well on other similar faces (perhaps because it expects an office background).

Some overfitting in flexible models is almost always inevitable. Thus to be credible, published test results for the final model must show performance on data that is *independent of any data used when building the model.* To prevent monkey business, the test data should be "kept in a vault" until released for final testing of the completed model. In practice this may be difficult to do.

C.2 Partitioning the data: training, tuning, and testing

This section discusses the datasets needed when building and testing models. Typically we want to measure our model's *generalization* performance, and so want to measure fitting error on new data, i.e., not on data that was used to train the model. This should be immediately obvious from the above discussion of overfitting.

We typically want to measure performance in two scenarios:

1. For **parameter selection**, i.e., to choose certain key model parameters during the model building process. For example, for HAT descriptors we need to select the optimal grid size so we must try a range of grid sizes (as illustrated in Figure 8.7). The new data is used as *parameter-selection* data, also commonly called *tuning*, *model-selection*, or *validation* data.

2. For **model assessment**, i.e., to measure the performance of the final model. Here the new data is used as final *test data*.

We thus require three independent datasets:

(i) the **training** data,
(ii) the **parameter-selection** data for (1) above,
(iii) the final **test** data for (2) above.

It may be excusable under certain conditions to conflate the training and parameter-selection data (although doing so increases the risk of overfitting), but test results for the final model should always be measured on independent test data. Any data that was used for training or parameter selection must be excluded from the test data. No decisions about the way the model is built should be made from the test data.

A note on linear models. Many of us started learning about statistical modeling by studying linear models. It's reasonable to ask why we don't bother with all the above datasets with linear models. The answer is that linear models are relatively inflexible, and with these simple models the difference between the performance[1] measured on the training data and on independent test data is inconsequential, provided certain assumptions are met. So we don't need a separate test set. Additionally, when building a linear model there is no separate parameter-selection step, so we don't need a parameter-selection set. This will no longer be true if we adopt a more flexible modeling procedure. For example, if we are doing variable selection for a linear model (or if we have a large number of possibly irrelevant variables), we should measure R^2 on data that is independent of the training data, or adjust R^2 as if we had such data. With more flexible models overfitting becomes more likely, since a flexible model can adapt to the peculiarities of the training data.

C.3 Cross validation and its hazards

Without access to the three sets of data described above, people resort to other techniques. One such technique is *cross-validation*. In this section, we assume that the reader already knows the basics of cross-validation—that is, partition the data into say 10 subsets, repeatedly train a model on all but one of those subsets (the *in-fold* data, nine-tenths of the data), and measure performance on the left-out data (the *out-of-fold* data, one-tenth of the data). For more explanation see e.g. Hastie et al. [45] Section 7.10 or Duda et al. [33] Section 9.6.

[1]For linear models performance is usually measured as R^2, i.e., the residual sum-of-squares relative to an intercept-only model.

Cross-validation, although popular, is prone to subtle errors. Some of these are listed below. Given how easy it is to make these mistakes, considerable skepticism is warranted when papers present final model results based on cross-validation.

C.3.1 Independence of observations

The out-of-fold data must play the role of new data. It's thus important that the out-of-fold isn't "contaminated" by the in-fold training data. This implies that the observations must be independent.

Lack of independence means that the in-fold data used for training is partially duplicated in some sense in the out-of-fold data used for testing, and cross-validation will tend to give optimistic fits. At a minimum, we should avoid "twinned" observations—in the context of landmarking models, for example, if the same face appears twice in a database in similar conditions then use of that database for cross-validation is questionable.

C.3.2 Pre-tuning

Don't use a set of data for selecting model parameters, then use the same data for subsequent cross-validation of the model. Cross-validation must be applied to the entire model building process. Any parameter that is tuned to the training data must not be tuned before cross-validation begins.

A word of explanation on the above paragraph. Let's say we do in fact optimize a parameter to the full dataset before cross-validation begins. By doing so, we are also to some extent optimizing to the out-of-fold data used during cross-validation (because the out-of-fold data is, after all, drawn from the full data). The out-of-fold data is thus contaminated and can no longer legitimately play the role of independent test data, and performance measured on the out-of-fold data will appear better than it should.

There are however contexts where it's acceptable to make decisions before cross validation. Decisions that are made independently of the training data are acceptable. Decisions about which variables to include that are made independently of the response are acceptable. See the comments near the end of Section 7.10.2 in Hastie et al. [45].

C.3.3 Confusing parameter-selection fitness with final test fitness

If fits obtained by cross-validation are used to select a model's parameters, then the final test fit quoted for the selected model must be recalculated for that model using a *new* set of independent data. The cross-validation fit used when selecting the model cannot be quoted as the fitness for that model—that would be optimizing for the test set by conflating the parameter-selection and test data.

Bibliography

[1] E.H. Adelson, C.H. Anderson, J.R. Bergen, P.J. Burt, and J.M. Ogden. *Pyramid Methods in Image Processing.* RCA Engineer 29-6, 1984. `http://web.mit.edu/persci/people/adelson/pub_pdfs/RCA84.pdf`. Cited on page 21.

[2] BioID AG. *The BioID Face Database.* `http://www.bioid.com/support/downloads/software/bioid-face-database.html`, 2001. Cited on pages 26, 55, 61, 76, 77, 81, 88, and 113.

[3] Relja Arandjelović and Andrew Zisserman. *Three things everyone should know to improve object retrieval.* CVPR, 2012. Cited on page 104.

[4] T. Baltrusaitis, L.-P. Morency, and P. Robinson. *Constrained Local Neural Fields for Robust Facial Landmark Detection in the Wild.* ICCV, 2013. Cited on page 130.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. *SURF: Speeded Up Robust Features.* ECCV, 2006. Cited on page 116.

[6] E.M.L. Beale and R.J.A. Little. *Missing Values in Multivariate Analysis.* Journal of the Royal Statistical Society. Series B (Methodological), 1975. Cited on page 65.

[7] P.N. Belhumeur, D.W. Jacobs, D.J. Kriegman, and N. Kumar. *Localizing Parts of Faces Using a Consensus of Exemplars.* CVPR, 2011. Cited on pages 26, 75, 77, 78, 87, 88, 90, 113, 115, 116, 117, and 130.

[8] S. Belongie, J. Malik, and J. Puzicha. *Shape Matching and Object Recognition Using Shape Contexts.* PAMI, 2002. Cited on page 90.

[9] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2007. `http://research.microsoft.com/en-us/um/people/cmbishop/PRML/index.htm`. Cited on page 145.

[10] Leo Breiman. *Random Forests.* Machine Learning, 2001. Cited on page 105.

[11] Leo Breiman, Jerome H. Friedman, R.A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Wadsworth, 1984. Cited on pages 104 and 106.

[12] Christopher J.C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition.* Data Mining and Knowledge Discovery 2:121–167, 1998. Cited on page 24.

[13] Xudong Cao, Yichen Wei, Fang Wen, and Jian Sun. *Face Alignment by Explicit Shape Regression.* IJCV, 2012. Cited on pages 26 and 114.

[14] M. Castrillón Santana, O. Déniz Suárez, M. Hernández Tejera, and C. Guerra Artal. *ENCARA2: Real-time Detection of Multiple Faces at Different Resolutions in Video Streams.* Journal of Visual Communication and Image Representation, 2007. Cited on page 55.

[15] O. Çeliktutan, S. Ulukaya, and B. Sankur. *A Comparative Study of Face Landmarking Techniques.* EURASIP Journal on Image and Video Processing, 2013. `http://jivp.eurasipjournals.com/content/2013/1/13/abstract`. Cited on pages 13, 25, 80, 81, and 112.

[16] William S. Cleveland. *LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression.* The American Statistician, 1981. Cited on page 112.

[17] John D. Cook. *Don't invert that matrix.* `http://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix`, 2010. Cited on page 151.

[18] T.F. Cootes and Coordinator. *FGNET Manual Annotation of Face Datasets*, 2002. `http://www-prima.inrialpes.fr/FGnet/html/benchmarks.html`. Cited on pages 60, 65, 77, 85, 86, and 130.

[19] T.F. Cootes, G.J. Edwards, and C.J. Taylor. *Active Appearance Models.* ECCV, 1998. Cited on pages 25 and 117.

[20] T.F. Cootes, M. Ionita, C. Lindner, and P. Sauer. *Robust and Accurate Shape Model Fitting using Random Forest Regression Voting.* ECCV, 2012. Cited on pages 112 and 116.

[21] T.F. Cootes and C.J. Taylor. *Active Shape Models - 'Smart Snakes'.* BMVC, 1992. `http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/tfc_publications.html`. Cited on pages 1 and 24.

[22] T.F. Cootes and C.J. Taylor. *A Mixture Model for Representing Shape Variation.* Image and Vision Computing, 17(8):567–574, 1999. Cited on page 24.

[23] T.F. Cootes and C.J. Taylor. *Statistical Models of Appearance for Computer Vision.* Technical Report. University of Manchester (School of Medicine), 2004. `http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/Models/app_models.pdf`. Cited on pages 10, 11, 16, 19, 24, 28, 30, 40, 61, and 136.

[24] T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham. *Active Shape Models — their Training and Application.* Comput.Vis.Image Underst., 1995. Cited on page 62.

[25] T.F. Cootes, G.V. Wheeler, K.N. Walker, and C. J. Taylor. *View-Based Active Appearance Models.* Image and Vision Computing, 2002. Cited on page 117.

[26] cplusplus.com. *unordered_map.* cplusplus.com, 2015. `http://www.cplusplus.com/reference/unordered_map/unordered_map`. Cited on page 98.

[27] D. Cristinacce. *Automatic Detection of Facial Features in Grey Scale Images.* Doctoral Thesis. University of Manchester (Faculty of Medicine, Dentistry, Nursing and Pharmacy), 2004. Cited on page 76.

[28] D. Cristinacce and T.F. Cootes. *Feature Detection and Tracking with Constrained Local Models.* BMVC, 2006. `http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/Models/clm.html`. Cited on pages 25, 76, 78, and 116.

[29] N. Dalal and B. Triggs. *Histograms of Oriented Gradients for Human Detection.* CVPR, 2005. Cited on pages 26, 90, 100, and 116.

[30] Daniel F. DeMenthon and Larry S. Davis. *Model-based object pose in 25 lines of code.* IJCV, 1995. Cited on page 120.

[31] P. Dollár, P. Welinder, and P. Perona. *Cascaded Pose Regression.* CVPR, 2010. Cited on pages 25 and 131.

[32] I.L. Dryden and Kanti V. Mardia. *Statistical Shape Analysis.* Wiley, 1998. `http://www.maths.nott.ac.uk/personal/ild/book/index.html`. Cited on page 23.

[33] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* Wiley-Interscience Publication, 2000. `http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471056693.html`. Cited on pages 151 and 156.

[34] Shimon Edelman, Nathan Intrator, and Tomaso Poggio. *Complex Cells and Object Recognition.* Neural Information Processing Systems, 1997. Cited on page 110.

[35] G. Edwards, C.J. Taylor, and T.F. Cootes. *Interpreting Face Images Using Active Appearance Models.* International Conference on Automatic Face and Gesture Recognition, 1998. Cited on page 25.

[36] M. Everginham, J. Sivic, and A. Zisserman. *Hello! My name is...Buffy — Automatic Naming of Characters in TV Video.* BMVC, 2006. Cited on page 81.

[37] B. Flury and H. Riedwyl. *Multivariate Statistics: A Practical Approach.* Chapman and Hall/CRC, 1988. Cited on page 138.

[38] W.T. Freeman. *Computer Vision for Computer Games.* AFGR, 1996. Cited on page 90.

[39] W.T. Freeman and M. Roth. *Orientation Histograms for Hand Gesture Recognition.* AFGR, 1995. Cited on page 90.

[40] Jerome H. Friedman. *Multivariate Adaptive Regression Splines (with discussion).* Annals of Statistics 19/1, 1991. `https://statistics.stanford.edu/research/multivariate-adaptive-regression-splines`. Cited on pages 1, 11, 105, and 106.

[41] B. van Ginneken, A.F. Frangi, J.J. Stall, and B. ter Haar Romeny. *Active Shape Model Segmentation with Optimal Features.* IEEE-TMI, 21:924–933, 2002. Cited on page 25.

[42] R.C. Gonzalez and R.E. Woods. *Digital Image Processing (2nd Edition).* Prentice Hall, 2002. `http://www.imageprocessingplace.com`. Cited on pages 21, 24, and 91.

[43] Colin Goodall. *Procrustes Methods in the Statistical Analysis of Shape*. Journal of the Royal Statistical Society, 1991. Cited on page 30.

[44] M. K. Hasan, S. Moalem, and C. Pal. *Localizing facial keypoints with global descriptor search, neighbour alignment and locally linear models*. ICCV, 2013. Cited on page 130.

[45] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd Edition)*. Springer, 2009. Downloadable from `http://web.stanford.edu/~hastie/ElemStatLearn`. Cited on pages 47, 105, 106, 145, 155, 156, and 157.

[46] Trevor Hastie and Werner Stuetzle. *Principal Curves*. Journal of the American Statistical Association, 1989. Cited on page 143.

[47] Rob Hess. *An Open-Source SIFT Library*. ACM MM, 2010. Cited on page 90.

[48] Nicholas J. Higham. *Computing the Nearest Correlation Matrix – A Problem from Finance*. IMA J. Numer. Anal. 22, 329-343, 2002. `https://nickhigham.wordpress.com/2013/02/13/the-nearest-correlation-matrix`. Cited on pages 43, 65, and 153.

[49] H. Hotelling. *Analysis of a Complex of Statistical Variables with Principal Components*. Journal of Educational Psychology, 1933. Cited on page 147.

[50] S. Jaiswal, T. Almaev, and M. Valstar. *Guided unsupervised learning of mode specific models for facial point detection in the wild*. ICCV, 2013. Cited on page 130.

[51] Gareth James, , Daniela Witten, Trevor Hastie, and Rob Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, 2013. Cited on page 137.

[52] O. Jesorsky, K. Kirchberg, and R. Frischholz. *Robust Face Detection using the Hausdorff Distance*. AVBPA, 2001. `http://www.bioid.com/downloads/facedb`. Cited on page 55.

[53] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis (6th Edition)*. Prentice Hall, 2007. Cited on pages 137 and 150.

[54] A. Kanaujia and D.N. Metaxas. *Large Scale Learning of Active Shape Models*. ICIP, 2007. Cited on pages 90, 97, and 118.

[55] A. Kasinski, A. Florek, and A. Schmidt. *The PUT Face Database*. IPC, 2008. Cited on page 113.

[56] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. *Snakes: Active Contour Models*. International Journal of Computer Vision, 1:321–331, 1987. Cited on page 23.

[57] David G. Kendall. *A Survey of the Statistical Theory of Shape*. Statistical Science, 1989. Cited on page 23.

[58] Matthias Kirschner and Stefan Wesarg. *Active Shape Models Unleashed.* SPIE Medical Imaging, 2011. Cited on page 54.

[59] J.J. Koenderink and A.J. van Doorn. *The Structure of Locally Orderless Images.* Int.J. Comput. Vis., 1999. Cited on page 25.

[60] Martin Koestinger, Paul Wohlhart, Peter M. Roth, and Horst Bischof. *Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization.* First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies, 2011. Cited on pages 88, 120, and 125.

[61] V. Le, J. Brandt, Z. Lin, L. Boudev, and T.S. Huang. *Interactive Facial Feature Localization.* ECCV, 2012. Cited on pages 59, 89, 112, 113, 119, 129, 130, 132, and 134.

[62] J.R. Leathwick, D. Rowe, J. Richardson, J. Elith, and T. Hastie. *Using Multivariate Adaptive Regression Splines to Predict the Distributions of New Zealand's Freshwater Diadromous Fish.* Freshwater Biology, 2005. `http://www.botany.unimelb.edu.au/envisci/about/staff/elith.html`. Cited on page 106.

[63] Z. Li, J-i Imai, and M. Kaneko. *Facial Feature Localization using Statistical Models and SIFT Descriptors.* Robot and Human Interactive Communication, 2009. Cited on page 90.

[64] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z. Li. *Learning Multi-scale Block Local Binary Patterns for Face Recognition.* International Conference on Biometrics, 2007. Cited on page 120.

[65] Andy Liaw, Mathew Weiner; Fortran original by Leo Breiman, and Adele Cutler. *randomForest: Breiman and Cutler's random forests for regression and classification*, 2014. R package, `https://CRAN.R-project.org/package=randomForest`. Cited on page 105.

[66] Rainer Lienhart and Jochen Maydt. *An Extended Set of Haar-like Features for Rapid Object Detection.* Submitted to ICIP2002, 2002. Cited on pages 55 and 120.

[67] C. Lindner, P.A. Bromiley, M.C. Ionita, and T.F. Cootes. *Robust and Accurate Shape Model Fitting using Random Forest Regression-Voting.* PAMI, 2014. Cited on pages 77, 104, and 112.

[68] D.G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints.* IJCV, 2004. Cited on pages 1, 11, 90, 91, and 96.

[69] Craig Lucas. *Computing Nearest Covariance and Correlation Matrices.* Master's Thesis. University of Manchester (School of Mathematics), 2001. Cited on page 153.

[70] P. Lucey, J.F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews. *The extended Cohn-Kanade dataset (CK+): a complete facial expression dataset for action unit and emotion-specified expression.* Proc. of Workshop on CVPR for Human Communicative Behavior Analysis, 2010. Cited on page 81.

[71] Brais Martinez and Michel F. Valstar. *$L_{2,1}$-based regression and prediction accumulation across views for robust facial landmark detection.* IVC preprint, 2015. Cited on pages 26, 131, and 132.

[72] K. Messer, J. Matas, J. Kittler, J. Luettin, and G. Maitre. *XM2VTS: The Extended M2VTS Database.* Conference on Audio and Video-base Biometric Personal Verification, 1999. `http://www.ee.surrey.ac.uk/Research/VSSP/xm2vtsdb`. Cited on pages 60, 65, 69, 86, 113, and 129.

[73] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2015. R package, `https://CRAN.R-project.org/package=e1071`. Cited on page 105.

[74] K. Mikolajczyk and C. Schmid. *A Performance Evaluation of Local Descriptors.* PAMI, 2005. Cited on page 116.

[75] S. Milborrow. *Locating Facial Features with Active Shape Models.* Master's Thesis. University of Cape Town (Department of Image Processing), 2007. `http://www.milbo.users.sonic.net/stasm`. Cited on pages 14, 19, 25, 38, 39, 65, 67, 99, 108, 110, and 153.

[76] S. Milborrow. *plotpc: Plot principal component histograms around a bivariate scatter plot*, 2009. R package, `https://CRAN.R-project.org/web/packages/plotpc`. Cited on page 138.

[77] S. Milborrow. *rpart.plot: Plot rpart models. An enhanced version of plot.rpart*, 2011. R package, `http://www.milbo.org/rpart-plot`. Cited on page 105.

[78] S. Milborrow. *Stasm User Manual.* `http://www.milbo.users.sonic.net/stasm`, 2013. Cited on page 55.

[79] S. Milborrow. *Building Stasm 4 Models.* `http://www.milbo.users.sonic.net/stasm`, 2015. Cited on page 12.

[80] S. Milborrow, Tom E. Bishop, and F. Nicolls. *Multiview Active Shape Models with SIFT Descriptors for the 300-W Face Landmark Challenge.* ICCV, 2013. Cited on pages 13 and 130.

[81] S. Milborrow, J. Morkel, and F. Nicolls. *The MUCT Landmarked Face Database.* Pattern Recognition Association of South Africa, 2010. `http://www.milbo.org/muct`. Cited on pages 10, 17, 18, 82, and 85.

[82] S. Milborrow and F. Nicolls. *Locating Facial Features with an Extended Active Shape Model.* ECCV, 2008. Cited on pages 13, 25, 60, 66, and 78.

[83] S. Milborrow and F. Nicolls. *Active Shape Models with SIFT Descriptors and MARS.* VISAPP, 2014. `http://www.milbo.users.sonic.net/stasm`. Cited on pages 11, 13, and 116.

[84] S. Milborrow. Derived from mda:mars by T. Hastie and R. Tibshirani. *earth: Multivariate Adaptive Regression Splines*, 2011. R package, `http://www.milbo.users.sonic.net/earth`. Cited on page 105.

[85] B. Moghaddam and A. Pentland. *Probabilistic Visual Learning for Object Representation*. PAMI, 1997. Cited on page 54.

[86] M. Nixon and A.S. Aguado. *Feature Extraction and Image Processing (2nd Edition)*. Academic Press, 2008. Cited on page 24.

[87] OpenCV. *Open Source Computer Vision Library*. `http://opencv.org/`, 2012. Cited on pages 55 and 124.

[88] OpenMP. *OpenMP Application Program Interface*. `http://www.openmp.org`, 2008. Cited on page 69.

[89] K. Pearson. *On Lines and Planes of Closest Fit to Systems of Points in Space*. Philosophical Magazine 2 (6) 559-572, 1901. Cited on page 147.

[90] Ofir Pele and Michael Werman. *The Quadratic-Chi Histogram Distance Family*. ECCV, 2010. Cited on page 104.

[91] A.P. Pentland and S. Sclaroff. *Closed-form Solutions for Physically Based Modelling and Recognition*. PAMI, 1991. Cited on page 23.

[92] E. Persoon and K.S. Fu. *Shape Discrimination using Fourier Descriptors*. IEEE Trans. Syst., Man, Cybern., 1977. Cited on page 23.

[93] Simon J.D. Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, 2012. Downloadable from `http://www.computervisionmodels.com`. Cited on page 24.

[94] M. Querini and G. F. Italiano. *Facial Biometrics for 2D Barcodes*. Computer Science and Information Systems, 2012. Cited on page 90.

[95] A. Rattani, D.R. Kisku, A. Lagorio, and M. Tistarelli. *Facial Template Synthesis based on SIFT Features*. Automatic Identification Advanced Technologies, 2007. Cited on page 90.

[96] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. *Face Alignment at 3000 FPS via Regressing Local Binary Features*. CVPR, 2014. Cited on page 26.

[97] M. Rogers and J. Graham. *Robust Active Shape Model Search*. ECCV, 2002. Cited on page 25.

[98] S. Romdhani, S. Gong, and A. Psarrou. *A Multi-view Non-linear Active Shape Model using Kernel PCA*. BMVC, 1999. Cited on page 24.

[99] Sam T. Roweis and Lawrence K. Saul. *Nonlinear Dimensionality Reduction by Locally Linear Embedding*. Science (AAAS), 2000. Cited on page 143.

[100] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. *300 Faces in-the-Wild Challenge: The First Facial Landmark Localization Challenge*. ICCV, 2013. `http://ibug.doc.ic.ac.uk/resources`. Cited on pages 86, 88, 127, and 130.

[101] J.M. Saragih, S. Lucey, and J.F. Cohn. *Deformable Model Fitting by Regularized Landmark Mean-Shifts*. IHCV, 2010. Cited on pages 25 and 116.

[102] J.M. Saragih, S. Lucey, and J.F. Cohn. *Deformable Model Fitting by Regularized Landmark Mean-shift*. Int.J. Comput. Vis, 2011. Cited on page 81.

[103] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. *Nonlinear Component Analysis as a Kernel Eigenvalue Problem*. Neural Computation Vol. 10, No. 5, 1998. Cited on page 24.

[104] Y. Shi and D. Shen. *Hierarchical Shape Statistical Model for Segmentation of Lung Fields in Chest Radiographs*. MICCAI, 2008. Cited on page 90.

[105] Christopher G. Small. *The Statistical Theory of Shape*. Springer, 1996. Cited on page 23.

[106] Brandon M. Smith and Li Zhang. *Collaborative Facial Landmark Localization for Transferring Annotations Across Datasets*. ECCV, 2014. Cited on page 85.

[107] L.H. Staib and J.S. Duncan. *Boundary Finding with Parametrically Deformable Models*. PAMI, 1992. Cited on page 23.

[108] M.B. Stegmann. *A Brief Introduction to Statistical Shape Analysis*. Available online, 2002. `http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/403/pdf/imm403.pdf`. Cited on page 30.

[109] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2014. R package, `https://CRAN.R-project.org/package=rpart`. Cited on page 104.

[110] Georgios Tzimiropoulos. *Project-Out Cascaded Regression with an application to Face Alignment*. CVPR, 2015. Cited on pages 26, 131, and 132.

[111] M. Uricar, V. Franc, and V. Hlavac. *Detector of Facial Landmarks Learned by the Structured Output SVM*. Int. Conf. on Computer Vision Theory and Applications,, 2012. Cited on page 81.

[112] M. Valstar, B. Martinez, X. Binefa, and M. Pantic. *Facial Point Detection using Boosted Regression and Graph Models*. CVPR, 2010. Cited on pages 78 and 81.

[113] W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S (4th Edition)*. Springer, 2002. `http://www.stats.ox.ac.uk/pub/MASS4`. Cited on pages 73 and 74.

[114] P. Viola and M. Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. CVPR, 2001. Cited on pages 11 and 120.

[115] P. Viola and M. Jones. *Fast Multi-view Face Detection*. Mitsubishi Electrical Research Laboratories, 2003. Cited on page 121.

[116] C. Vogel, R. de Sousa Abreu, D. Ko, S. Le, B.A. Shapiro, S.C. Burns, D. Sandhu, D.R. Boutz, E.M. Marcotte, and L.O. Penalva. *Sequence signatures and mRNA concentration can explain two-thirds of protein abundance variation in a human cell line*. Molecular Systems Biology, 2010. Cited on page 106.

[117] D. Vukadinovic and M. Pantic. *Fully Automatic Facial Feature Point Detection using Gabor Feature Based Boosted Classifiers*. IEEE Int. Conf. on Systems, Man and Cybernetics, 2005. Cited on pages 78 and 81.

[118] Wikipedia. *atan2*. `https://en.wikipedia.org/wiki/Atan2`, Accessed Jan 2016. Cited on page 91.

[119] Wikipedia. *Multivariate Adaptive Regression Splines*. `http://en.wikipedia.org/wiki/Multivariate_adaptive_regression_splines`, Accessed Jan 2016. Cited on pages 106 and 107.

[120] Xuehan Xiong and Fernando De la Torre. *Supervised Descent Method and its Applications to Face Alignment*. CVPR, 2013. Cited on pages 26, 90, 131, and 132.

[121] J. Yan, Z. Lei, D. Yi, and S.Z. Li. *Learn to combine multiple hypotheses for face alignment*. ICCV, 2013. Cited on pages 129 and 130.

[122] Heng Yang and Ioannis Patras. *Mirror, mirror on the wall, tell me, is the error small?* CVPR, 2015. Cited on page 70.

[123] A.L. Yuille, P.W. Hallinan, and D.S. Cohen. *Feature extraction from faces using deformable templates*. Int.J. Comput. Vision, 1992. `http://www.stat.ucla.edu/~yuille`. Cited on page 23.

[124] Stefanos Zafeiriou, Cha Zhang, and Zhengyou Zhang. *A Survey on Face Detection in the Wild: Past, Present and Future*. CVIU, 2014. Cited on page 122.

[125] C. Zahn and R. Roskies. *Fourier Descriptors for Plane Closed Curves*. IEEE Transactions on Computers C–21 269–281, 1972. Cited on page 23.

[126] J. Zhang and S. Y. Chen. *Combination of Local Invariants with an Active Shape Model*. BMEI, 2008. Cited on page 90.

[127] L. Zhang, D. Tjondronegoro, and V. Chandran. *Geometry vs. Appearance for Discriminating between Posed and Spontaneous Emotions*. NIP, 2011. Cited on page 90.

[128] Qiang Zhang, A. Bhalerao, E. Helm, and C. Hutchinson. *Active Shape Model Unleashed with Multi-scale Local Appearance*. ICIP, 2015. Cited on page 116.

[129] D. Zhou, D. Petrovska-Delacrétaz, and B. Dorizzi. *Automatic Landmark Location with a Combined Active Shape Model*. BTAS, 2009. Cited on page 90.

[130] Erjin Zhou, Haoqiang Fan, Zhimin Cao, Yuning Jiang, and Qi Yin. *Extensive Facial Landmark Localization with Coarse-to-fine Convolutional Network Cascade*. ICCV, 2013. Cited on pages 99, 129, and 130.

[131] Y. Zhou, L. Gu, and H.J. Zhang. *Bayesian Tangent Shape Model: Estimating Shape and Pose Parameters via Bayesian Inference*. CVPR, 2003. Cited on page 25.

[132] X. Zhu and D. Ramanan. *Face Detection, Pose Estimation and Landmark Localization in the Wild*. CVPR, 2012. Cited on pages 79, 81, 87, 88, 117, 130, and 132.