Applying Statistical and Syntactic Pattern Recognition Techniques to the Detection of Fish in Digital Images

Evelyn June Hill

This thesis is presented for the degree of
Master of Engineering Science
of The University of Western Australia
Faculty of Engineering, Computing and Mathematics.
September, 2004

Acknowledgements

My two supervisors for this project, Dr Mike Alder (School of Mathematics and Statistics, University of Western Australia) and Dr Chris deSilva (Australian Research Centre for Medical Engineering, Murdoch University) were exceptionally helpful. They were generous with their time and always approachable; they provided me with many interesting ideas to test and supplied constructive criticism when needed.

All the images used in this thesis were generously supplied by Dr Euan Harvey at the School of Plant Biology, University of Western Australia.

I would like to acknowledge Fiona Evans and Warick Brown for discussing aspects of my research work with me and for making helpful suggestions for solving problems.

Finally, thank you to the computer support staff and the administrative staff in the School of Electrical, Electronic and Computer Engineering and the School of Mathematics and Statistics for their friendly and willing assistance in various matters.

Abstract

This study is an attempt to simulate aspects of human visual perception by automating the detection of specific types of objects in digital images. The success of the methods attempted here was measured by how well results of experiments corresponded to what a typical human's assessment of the data might be. The subject of the study was images of live fish taken underwater by digital video or digital still cameras. It is desirable to be able to automate the processing of such data for efficient stock assessment for fisheries management. In this study some well known statistical pattern classification techniques were tested and new syntactical/structural pattern recognition techniques were developed.

For testing of statistical pattern classification, the pixels belonging to fish were separated from the background pixels and the EM algorithm for Gaussian mixture models was used to locate clusters of pixels. The means and the covariance matrices for the components of the model were used to indicate the location, size and shape of the clusters. Because the number of components in the mixture is unknown, the EM algorithm has to be run a number of times with different numbers of components and then the best model chosen using a model selection criterion. The AIC (Akaike Information Criterion) and the MDL (Minimum Description Length) were tested. The MDL was found to estimate the numbers of clusters of pixels more accurately than the AIC, which tended to overestimate cluster numbers.

In order to reduce problems caused by initialisation of the EM algorithm (i.e. starting positions of mixtures and number of mixtures), the *Dynamic Cluster Finding algorithm* (DCF) was developed (based on the *Dog-Rabbit strategy*). This algorithm can produce an estimate of the locations and numbers of clusters of pixels. The *Dog-Rabbit strategy* is based on early studies of learning behaviour in neurons. The main difference between Dog-Rabbit and DCF is that DCF is based on a toroidal topology which removes the tendency of cluster locators to migrate to the centre of mass of the data set and miss clusters near the edges of the image.

In the second approach to the problem, data was extracted from the image using an edge detector. The edges from a reference object were compared with the edges from a new image to determine if the object occurred in the new image. In order to compare edges, the edge pixels were first assembled into curves using an *UpWrite* procedure; then the curves were smoothed by fitting parametric cubic polynomials. Finally the curves were converted to arrays of numbers which represented the signed curvature of the curves at regular intervals.

Sets of curves from different images can be compared by comparing the arrays of signed curvature values, as well as the relative orientations and locations of the curves. Discrepancy values were calculated to indicate how well curves and sets of curves matched the reference object. The total length of all matched curves was used to indicate what fraction of the reference object was found in the new image. The curve matching procedure gave results which corresponded well with what a human being being might observe.

Contents

List of Tables

List of Figures

Introduction

This thesis is concerned with examining several methods for training a computer program to be able to locate certain types of objects in an image, with the aim of then being able to count the number of these objects in a single image. The objects that have been chosen to be the subject of this study are fish. The fish are recorded on digital images taken with an underwater camera.

To appreciate the difficulties of this problem, the reader may wish to examine Figure 1.1. One should be able to count 32 fish (or part fish) in this image. Notice how (1) the outlines of the fish are poorly defined; (2) in very few of the cases can the whole fish be observed, a lot of the fish are obscured by other fish or are cut off by the edge of the image; (3) one may use different features to identify different fish (e.g. a bright eye with a dark centre, curves of light reflected from the body, a dark forked tail). One fish in particular near the bottom and centre of the image is defined only by a dark curve with a slightly lighter area above it (fish number 27 in Figure 1.2) but we still identify it as a possible fish because the lines are similar to line patterns on other more clearly defined fish. There is a bright spot near where we would expect its eye to be and a dark area near where we would expect its tail to be, this helps to confirm our identification of the line as being part of a fish.

Human beings do not need to see a whole fish to identify it, only part of a fish which has the right spatial relationships to other parts of a fish is required. In this case, all the fish are all approximately the same size and swimming in the same direction, and that makes pattern detection of the fish easier.

1.1 Why Fish?

The images have been obtained as part of ongoing research projects on video monitoring of marine fish in their natural habitat (Harvey, Fletcher & Shortis 2002, Harvey, Shortis, Stadler & Cappo 2002, Harvey, Cappo, Shortis, Robson, Buchanan & Speare 2003). These research projects have produced vast quantities of video data which require processing. Automation or partial-automation of this process would greatly reduce the time and expense involved in extracting useful data from the images. The underwater images are supplied by Dr Euan Harvey of the School of Plant Biology at the University of Western Australia.

1.2 Previous Work on this Topic

Some examples of current or recent research on recognition and measurement of fish are listed below. All the published work on this topic is based on monitoring fish in a controlled environment (i.e. either dead fish on a conveyor belt or fish in tanks or cages). Trying to identify fish in a wild environment is much more complicated; for example, there may be other objects in the image; there may be more than

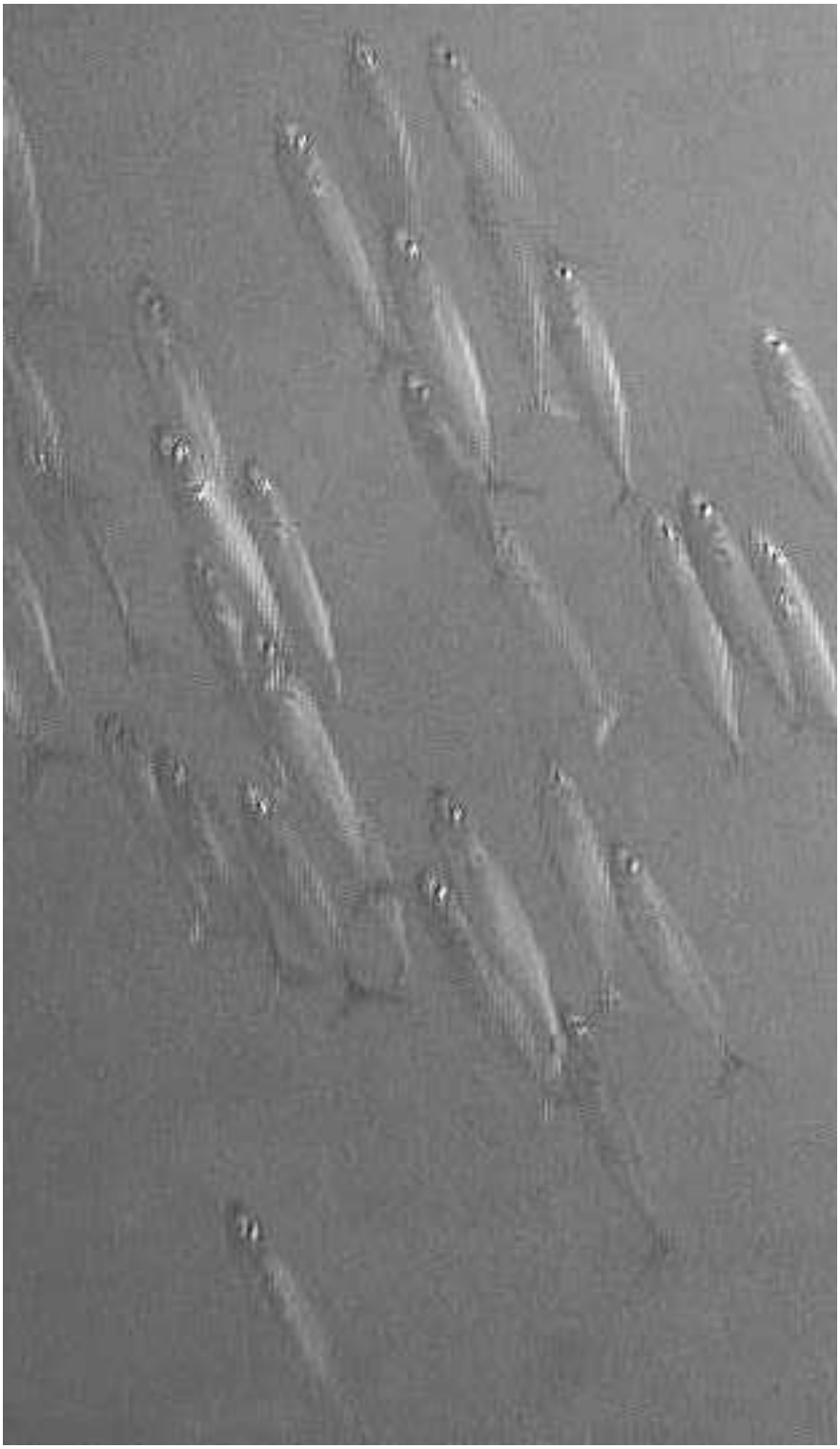


Figure 1.1: Image of a school of pilchards taken using an underwater camera.

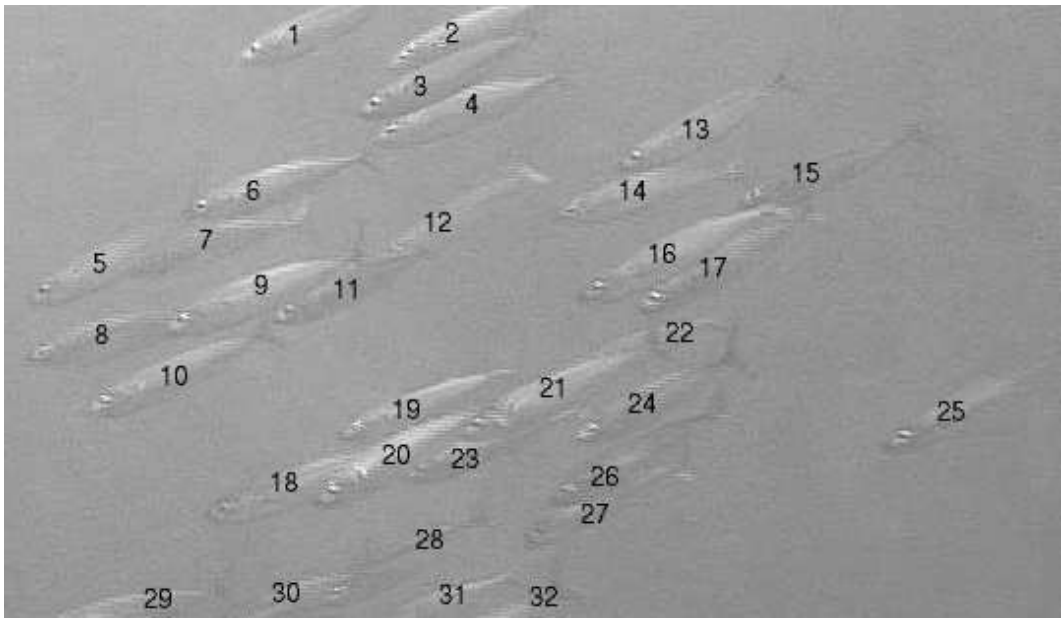


Figure 1.2: The number of fish in Figure 1.1.

one species of fish; lighting conditions and water clarity may be less than ideal; the orientation of fish will be more variable.

Recent and current work on fish recognition and measurement includes:

- Strachan (1993) recorded the colour and shape of silhouettes of dead fish; then used discriminant analysis of colour and shape descriptors to separate the various species of fish.
- Zion et al. (1999) captured the shape of dead fish using a video camera. They used the method of moment invariants to classify the fish shapes into 3 classes corresponding to 3 species.

Both Strachan and Zion *et al.* were only required to be able to distinguish between a limited number of species of fish. Using dead fish meant that they would carefully control the environment in which the video images were recorded by providing specific lighting and a single colour background.

- Tillet et al. (2000) used stereo underwater video to record salmon in tanks and cages. They estimated the location of fish using motion detection and applied a deformable template (3D point distribution model) to estimate fish dimensions. They used a carefully controlled environment and only a single species of fish.
- Shortis and Seager (*in* Harvey, Shortis, Seager & Robson 2003) are using motion detection to establish the location of fish in images. They use a background modeling approach that absorbs slow changes (which get updated into

the background model), whilst detecting motion as fast changes against the background. They are working on video images of a single species (tuna) in a controlled environment (aquaculture cages), but which may have poor lighting and water clarity.

1.3 Research Techniques

Much of the research work in this thesis follows on from various pattern recognition techniques developed by my supervisors, Dr Mike Alder and Dr Chris deSilva (e.g. Alder 2001, Alder et al. 1990), and their current and past postgraduate students (e.g. Evans 2003, Evans et al. 2003, McLaughlin 2000, Hew 1999, Williams 1999, Lim 1997, McKenzie & Alder 1994).

I have experimented with two different approaches to this problem. The first was to extract the whole fish from the background using thresholding and attempt to locate whole fish; the second approach was to run an edge detector on the image and then attempt to locate the fish from the shapes of the edges. The first method would only be expected to work well on fish in a controlled environment where the fish are in contrast to the background, there are no other similar objects in the frame and there is not a great degree of overlap of fish. The second method would be a more suitable approach for fish in a less controlled environment; however, it is a much more difficult problem.

A wide variety of techniques were experimented with to discover which might be the most suitable approach to this problem. These included:

- statistical pattern recognition techniques
- cluster detection based on a simple neuronal function model
- template matching
- edge detection
- the UpWrite (a procedure based on syntactical pattern recognition)
- curve matching.

The application of each of these techniques to the problem will be discussed in the following chapters, and their usefulness will be assessed.

1.4 Outline of Chapters

In Chapters 2 and 3 statistical techniques are applied to data extracted from images in order to locate and count the number of elliptical or semi-elliptical objects in the image. The pixels representing objects in an image form approximately elliptical clusters of data which are modelled as a Gaussian mixture. In Chapter

2 the optimal locations, sizes and approximate shapes of clusters are determined using the EM algorithm. Once clusters are located, confirmation is required that these clusters represent the desired type of object, Appendix C illustrates a simple method for locating a fish eye within the area of the image detected as a potential fish.

In Chapter 3 the number of clusters is estimated using two different model selection criteria, the AIC and the simplified MDL. These criteria use the maximum likelihood estimate of the model given the data, which is a measure of how well the data fits the model. This estimate is then penalised for the complexity of the model, i.e. the more clusters, the the higher the penalty and the less likely the model.

An alternative method for finding clusters and estimating the numbers of clusters present in the data is discussed in Chapter 4. A cluster finding algorithm, based on the learning behaviour of neurons (in particular, lateral inhibition and habituation), is tested and modified to account for edge effects. In many cases this cluster finding strategy works better than the widely used k-means clustering algorithm; a series of comparison tests is presented in Appendix D.

In the next two chapters, a completely different approach to fish recognition is tested. This approach involves firstly finding edge pixels in a reference image using an edge detector, and secondly, building up the fish by grouping edge pixels into line segments, then grouping the line segments into curved lines, and finally, grouping all the curved lines which represent one fish. In order to detect a fish in a new image, the curved line detection is repeated for the new image and then the new curved lines are compared to the reference set to determine whether any groups of the curves may belong to a single fish. Chapter 5 describes detection of curves and a method for describing their curvature. Chapter 6 describes how new curves can be compared with a reference set of curves and illustrates the method with examples.

Further appendices included are Appendix A which lists symbols used in this thesis; and in Appendix B the volume under a 2D Gaussian surface is calculated for use in Chapter 2. Appendix E contains the computer code used for testing the algorithms and a description of the algorithms used in Chapters 5 and 6.

The EM Algorithm for Gaussian Mixture Models

2.1 Introduction

The aim of *clustering* or *cluster analysis* is to divide the data into a number of useful subsets based on similarity of data points. The method used here is a type of *unsupervised learning*, which means that, before the experiment begins, it is not known how many subsets (i.e. clusters) there are or how they are distinguished from each other.

The EM algorithm for Gaussian mixture models is a well known method for cluster analysis (e.g. McLachlan & Basford 1988, Titterington et al. 1985, Fraley & Raftery 2002). A useful outcome of this method is that it produces a likelihood value for the clustering model and the likelihood values can be used to select the best model from a number of different models providing that they have the same number of parameters (i.e. same number of clusters). If there are different numbers of clusters in the different models then they must be compared using some other criteria (see Chapter 3). Another useful outcome of the EM algorithm for Gaussian mixture models is that it calculates a covariance matrix for each component which can be decomposed to provide information on the size, shape and orientation of the data cluster (e.g. Banfield & Raftery 1993, Bensmail & Celeux 1996).

A significant problem with the EM algorithm is that it converges to a local maximum of the likelihood function and hence the quality of the result depends on the initialization. This problem is illustrated in this chapter using examples, and a method for improving initialization is discussed in Chapter 4.

Another problem with this method is that the number of mixture components, m, is fixed at the start. The traditional method of finding the optimal number of components in the data is to re-run the EM algorithm for all reasonable values of m and then use an information criterion for selecting the best model (i.e. the model that requires the least information to describe the data). Some of the popular information criteria methods have been tested on the fish images, see Chapter 3.

2.2 Gaussian Mixture Models

The data are treated as a mixture model, where the total likelihood of model θ with m components given the observed data points, $X = x_1, \ldots, x_n$ ($X \subset \mathbb{R}^n$) is:

$$L(\theta \mid X) = \prod_{i=1}^{n} \sum_{j=1}^{m} w_j f_j(x_i|\theta_j) \tag{2.2.1}$$

where f_j is the probability density function and θ_j is the vector of parameters for the jth component of the mixture. Each component contributes a proportion, w_j,

of the total population, such that:

$$\sum_{j=1}^{m} w_j = 1. \tag{2.2.2}$$

Log likelihood may be used instead of likelihood. The probability density function used here for all the components is the Gaussian distribution. The Gaussian probability for data point x_i in mixture j is given by:

$$g_j(x_i \mid \mu_j, \Sigma_j) = \frac{\exp(-0.5(x_i - \mu_j)^T \Sigma_j^{-1}(x_i - \mu_j))}{\sqrt{(2\pi)^n \det \Sigma_j}}. \tag{2.2.3}$$

where μ_i and Σ_j are the mean and covariance matrix of component j, respectively.

The objects in the image (i.e. the fish) should have quite sharp boundaries and so the distribution of pixels would not truly fit a Gaussian distribution. However, if a rectangular distribution were used to model a component of the mixture instead of a Gaussian distribution (Figure 2.2.1) the component would not be influenced by any data points which did not lie directly beneath the rectangle. Therefore, if the location of a rectangular distribution is initialized so that it does not overlap a cluster of data, then it will not see the cluster at all, i.e. the likelihood of these points belonging to the distribution will be zero and no data cluster will be found. A Gaussian distribution, on the other hand, has tails which overlap the entire data region, so a Gaussian distribution that is initialized with its centre well away from a cluster of data will still be influenced by the data points in the cluster and hence, by use of the EM algorithm (below), the Gaussian probability density function can be modified to fit the data.

Another reason for favoring the Gaussian distribution is that the extraction of pixels from the image is never a perfect process and the outlines will never be as sharp as required. There may be holes in the data and noise scattered around the object, especially close to the boundaries. The Gaussian model allows for blurring of the outlines. The ideal model would be like a Gaussian distribution but with steeper sides and flatter top. However, this shape would require a more complex function which would slow processing time. Despite the fact that the clusters are not strictly Gaussian in distribution, searching for clusters made from pixels of objects using the Gaussian mixture model works well and so appears to be a good compromise.

2.3 The EM Algorithm

The process of optimizing the model to achieve the maximum likelihood for the data can be simplified by employing the EM algorithm, which is an iterative two step procedure (Dempster et al. 1977). If the initial estimate for θ is θ^0 then the E step is given by the expected value of the total likelihood (or log likelihood) given the observed data X and θ^0:

$$\mathcal{Q}(\theta, \theta^0) = \mathbb{E}[L(\theta) \mid X; \theta^0] \tag{2.3.1}$$

Figure 2.2.1: Comparison of the shapes of a rectangular and a Gaussian probability density function. The likelihood of points (black circles) belonging to a rectangular probability density function (left) is zero unless the rectangle overlaps the points. For a Gaussian distribution (right), all points will have a likelihood > 0 of belonging, irrespective of what the parameters of the distribution are.

Subsequently, at iteration t, this formula becomes:

$$\mathcal{Q}(\theta, \theta^{t-1}) = \mathbb{E}[L(\theta) \mid X; \theta^{t-1}] \tag{2.3.2}$$

The M step calculates the values for θ which maximize $\mathcal{Q}(\theta, \theta^{t-1})$. This is achieved by first recalculating the weights using the total probability that each of the data points x_i belongs to component g_j:

$$w_j = \frac{\sum_{i=1}^{n} R_j(x_i)}{|X|} \tag{2.3.3}$$

where $|X|$ is the cardinality of the data set and $R_j(x_i)$ is given by:

$$R_j(x_i) = \frac{L_j(\theta_j \mid x_i)}{\sum_{j=1}^{m} L_j(\theta_j \mid x_i)}. \tag{2.3.4}$$

Then the value for the mean and the covariance of the component can be recalculated:

$$\mu_j = \frac{\sum_{i=1}^{n} x_i R_j(x_i)}{\sum_{i=1}^{n} R_j(x_i)} \tag{2.3.5}$$

$$\Sigma_j = \frac{\sum_{i=1}^{n} R_j(x_i)[(x_i - \mu_j)(x_i - \mu_j)^T]}{\sum_{i=1}^{n} R_j(x_i)}. \tag{2.3.6}$$

At each step of the EM algorithm likelihood cannot decrease, i.e. $L(\theta^{t+1}) \geq L(\theta^t)$ (Dempster et al. 1977). Therefore, if the sequence is bounded above, $L(\theta^t)$ will converge to a local maximum. The problem remains to find the largest local maximum and this is usually approximated by finding several maxima and comparing the log likelihood values (e.g. Redner & Walker 1984).

2.4 Singularities

The EM algorithm may converge to a position where the covariance matrix becomes singular (the cluster degenerates to a straight line or a point in $\mathbb{R}^2$). I deal

with this problem by testing for these cases at each iteration of the algorithm as follows.

If $\det \Sigma_j = 0$, then make $L(\theta_j \mid x_i) = 0$.

This means that $\sum_{i=1}^{n} R_j(x_i) = 0$.

The latter equality is tested for in each cluster and if true then the weight, mean and covariance matrix of any cluster that satisfies the equality are reset to their initial values and the weights are renormalized.

2.5 Coding the Algorithm

The statistical programming language R has been used for coding the EM algorithm and all other coding in this thesis except where stated otherwise. R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License at http://www.r-project.org/. While R coding is not suited to computer intensive tasks as it is slow, it is a very useful research tool. R has a a graphical environment and can be used interactively as well being able to source existing files of pre-written code. Although packages do exist for running the EM algorithm in R (Fraley & Raftery 2000a), the EM algorithm was rewritten here in order to allow easier manipulation of the program, such as the method of dealing with singularities described above. The coded algorithms are included in Appendix E. Various packages were used to extract the fish from the background of the images including GGobi (www.ggobi.org) and Gimp (www.gimp.org). Tiling and thresholding algorithms were also written in Java to assist in extraction, these are included in Appendix E.

2.6 Fitting Ellipses to the Clusters

The bodies of the fish in the images are approximately elliptical in shape. When the EM algorithm has converged, the final covariance matrix calculated for each cluster can be used to draw an ellipse which approximates the size, location and orientation of each fish. The ellipse is the set of points which satisfy the function:

$$\sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} = 2 \tag{2.6.1}$$

The centre of the ellipse will be at the mean (μ) of the distribution, and the points on the ellipse will lie at a Mahalanobis distance of 2 from the mean, i.e. the distance of a point on the ellipse from the mean depends on the square root of the covariance. In future I will refer to this ellipse as the M2 ellipse. An M2 ellipse encompasses approx. 86% of the points in a two dimensional Gaussian distribution (see Appendix B).

Using its eigenvalues and eigenvectors, the covariance matrix can be decomposed into:

$$\Sigma = Q\Lambda Q^{-1} \tag{2.6.2}$$

where Q is an normalized, orthogonal matrix which contains the eigenvectors of Σ; and Λ is a diagonal matrix containing the eigenvalues of Σ. Λ determines the size and shape of the ellipse and Q determines the orientation of the ellipse (Alder 2001, Bensmail & Celeux 1996).

An M2 ellipse can be drawn by transforming a unit circle as follows:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \mu + 2Q\sqrt{\Lambda}\begin{pmatrix} x \\ y \end{pmatrix} \tag{2.6.3}$$

where (x, y) are points on the circle given by

$$\left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2 : \exists t \in [0, 2\pi), \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} cos(t) \\ sin(t) \end{pmatrix} \right\} \tag{2.6.4}$$

and where the square root of the diagonal matrix (i.e. $\sqrt{\Lambda}$) is the diagonal matrix of positive square roots.

Examples where ellipses have been fitted to clusters of data points, where each cluster represents one fish, are shown in Figures 2.6.1 and 2.6.2.

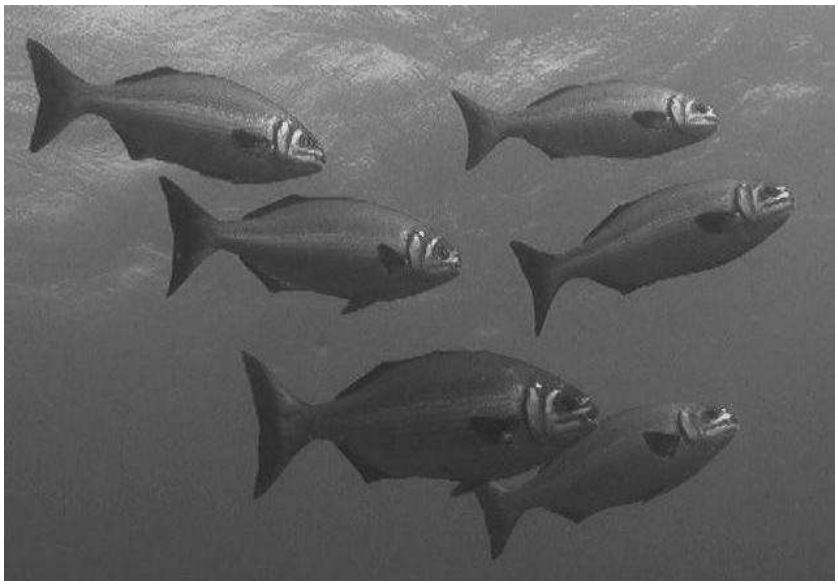
2.7 Discussion: Elimination of Undesirable Objects

Once the ellipses have been calculated elimination of undesirable objects can proceed. For example, ellipses that are too small or defined by too few points, or ellipses that have unrealistic 'non-fish' shapes. A more sophisticated model of a fish can be applied to the data clusters to test for fishy features, such as the presence of an eye or tail and fins. Relative locations of features can be tested as well, for example the eye should be near one end of the data cluster and should lie on the opposite end of an elongate ellipse to the tail. Appendix C shows an example where template matching is used to find the eye of a fish in the region of the data cluster.

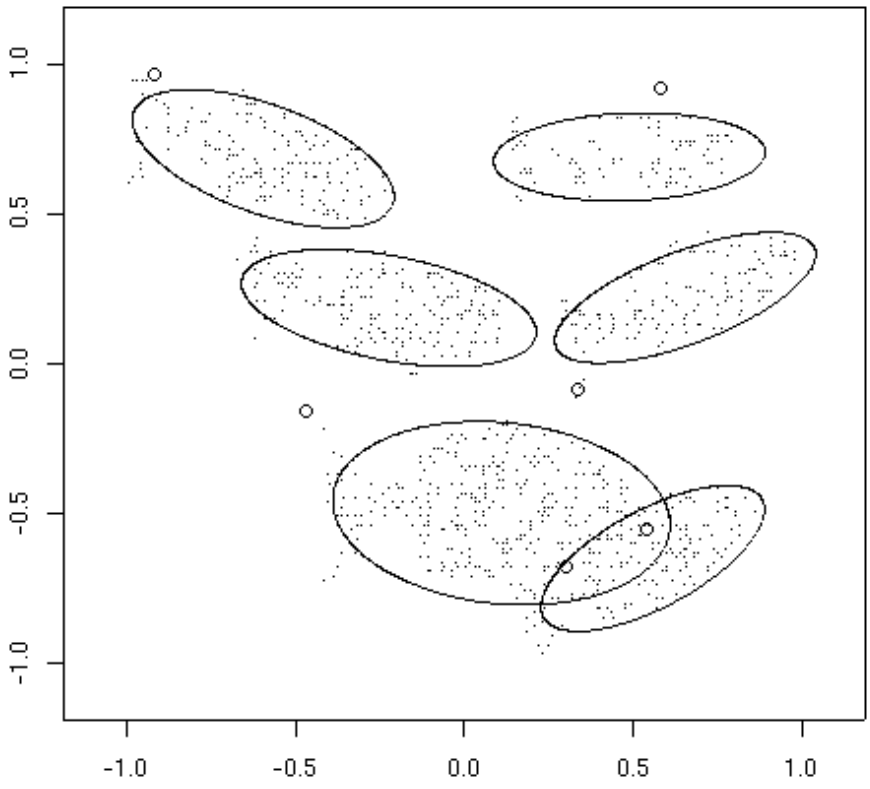
2.8 Discussion: Starting values for the EM algorithm

The convergence rate and success of clustering using the EM algorithm can be degraded by a poor choice of starting values for the means, covariances and weights of the components (McLachlan & Basford 1988). In the experiments presented here, uniform random values were chosen for the initial means (μ_j); the initial value of the covariance matrices was given by $\Sigma_j = \lambda \mathrm{I}$ (i.e. an initially spherical distribution of data, scaled by λ depending on the scale of the data); and the initial values of the weights were $w_j = 1/m$, where m is the estimated number of mixture components.

The results of repeated experiments indicated that using random starting values for initial estimates of the means frequently gave poor results (see Figure 2.6.2, for

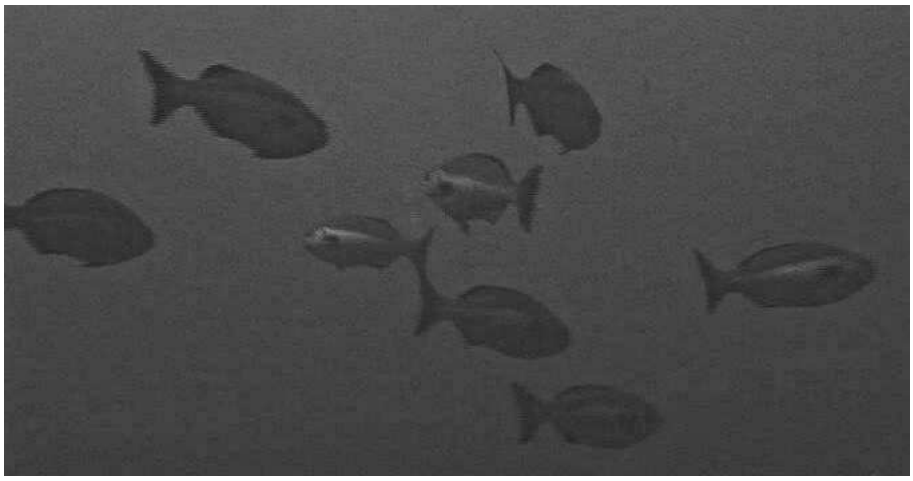


(a)

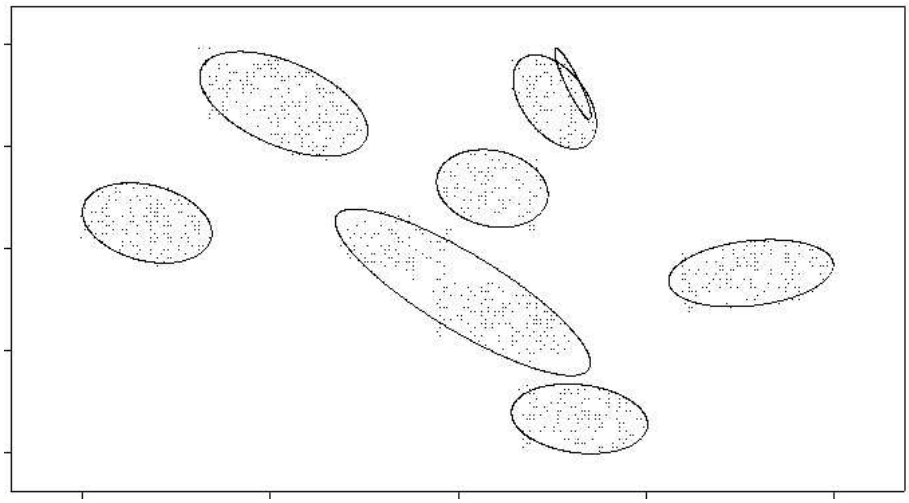


(b)

Figure 2.6.1: (a) Image of 6 fish from which data was extracted, first by thresholding to remove background, and then by random selection of data points from 'fish' pixels. (b) M2 ellipses have been drawn for all data clusters detected using the EM algorithm. Small open circles represent initial positions of cluster centres.

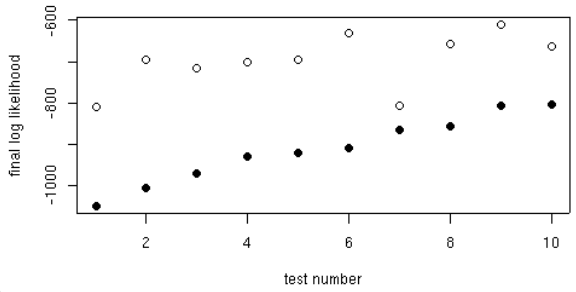


(a)

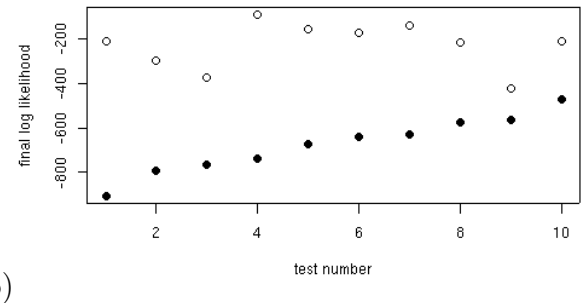


(b)

Figure 2.6.2: (a) Image of 8 fish from which data was extracted. (b) M2 ellipses have been drawn for each data cluster. (b) would be considered a poor result as two fish have been counted as one and one fish has been counted as two.



(a)



(b)

Figure 2.8.1: Plots of log likelihood values after one iteration of the EM algorithm (filled circles) and after convergence of the EM algorithm (open circles) for a set of random initial mean values; (a) for the 6 fish shown in Figure 2.6.1(a); (b) for the 8 fish shown in Figure 2.6.2(a).

example). The highly variable nature of the results of the tests is reflected in the very inconsistent values for the final log likelihood (see Figure 2.8.1). In order to get good results from using random starting values a large number of runs of the EM algorithm would be required, which is a very computer intensive process. To improve the outcome of the EM algorithm on Gaussian mixture models it is necessary to find a better method of estimating initial mean values for the components. This is dealt with in Chapter 4. In the next chapter the problem of estimating the correct number of components (i.e. clusters) in the data will be addressed.

Model Selection

3.1 Introduction

Part of the model selection process is to determine the appropriate number of clusters with which to model the data (also known as cluster validation). While the EM algorithm is very useful for detecting clusters in the data, it cannot determine how many clusters are present. An estimate of the number of clusters (m) must be made before running the algorithm.

Cluster validation criteria provide rules which allow the best model to be selected from a set of models in which the number of clusters varies. The maximum likelihood cannot be used as this criterion because it is a non-decreasing function of m (Figueiredo & Jain 2002), i.e. the maximum likelihood value will always increase (or remain the same) as the number of clusters in the model increases. However, the criteria incorporate maximum likelihood in their rules and the EM method is usually used to calculate this likelihood value.

In this chapter I discuss two popular cluster validity criteria and show results from experimenting with these criteria, namely, the AIC (Akaike's information criterion) and the simplified MDL (minimum description length), which uses the same formula as the BIC (Bayesian information criterion). These criteria for determining the number of clusters in a mixture rely on a two part formula. The first part uses the estimate of the log likelihood of the model given the observed data. The second part penalizes the model for its degree of complexity. The overall formula attempts to find the best balance between goodness-of-fit and complexity, to avoid over fitting of the data and loss of generality.

There are three alternative approaches used for comparing and selecting models using cluster validation criteria:

- The maximum likelihood is calculated using EM for all models over a reasonable range of values for m and the best value calculated is selected for the chosen criterion (e.g. Fraley & Raftery 1998).

- In the agglomerative method (e.g. Figueiredo & Jain 2002) the EM algorithm is initiated with a very large number of components (equivalent to the maximum anticipated number of components). Each time the EM algorithm converges, the components are assessed and those components that do not pass certain tests are absorbed into the nearest components. These steps are repeated until all components pass the required tests. This method has an advantage in that it is less sensitive to initialization.

- In the splitting method (Evans et al. 2003), the EM algorithm is initialized using a single component. When the algorithm converges the component is split in two, then rerun through the EM algorithm. One of the cluster validity criteria is used to compare the two models. If the unsplit model is preferred

then the algorithm stops. If the split model is preferred then the algorithm continues splitting the components and testing for the preferred model.

In the examples in this chapter the first approach has been used because the aim here is to examine how the criteria behave with the data, although the more sophisticated second and third methods may be more appropriate in practice. Before describing the cluster validation criteria, the question of *what is the correct answer?* must be addressed.

3.2 The Correct Number of Clusters in the Data

The correct number of clusters in the data depends on the reason for wanting to know what the number of clusters is. In this case, the reason is that we are trying to simulate the cognitive process when a person attempts to count the number of objects in an image. So a correct answer is one which confirms what the human eye detects.

In some cases the correct answer is clear to a human being, but in others it may be ambiguous unless the person has further information about the characteristics of a cluster (such as shape, size and possible sub-structure). For example, in figure 3.2.1, there are clearly 5 separate clusters of data; given the extra information that the clusters we are looking for are all approximately the same size and shape and may be slightly overlapping, then 6 clusters appears to be the preferable answer.

The cluster detection method which gives the answers which correspond most closely to the answers that a human would give for the same set of data is the method which is to be preferred in this case. This does not mean that this is the best cluster detection method for universal problem solving.

3.3 The Akaike Information Criteria (AIC)

The Akaike Information Criteria or AIC (Akaike 1973) uses the Kullback-Leibler distance, which is a measure of the directed distance between two probability distributions, usually the distance between the estimated distribution and the true distribution. It is given by:

$$I(g,f) = \int g(x \mid \theta) \log \left(\frac{g(x \mid \theta)}{f(x \mid \hat{\theta})} \right) dx \tag{3.3.1}$$

where $f(x|\hat{\theta})$ is the estimated distribution and $g(x|\theta)$ is the true distribution. $I(g,f)$ can be considered as the information lost when model f is used to approximate model g.

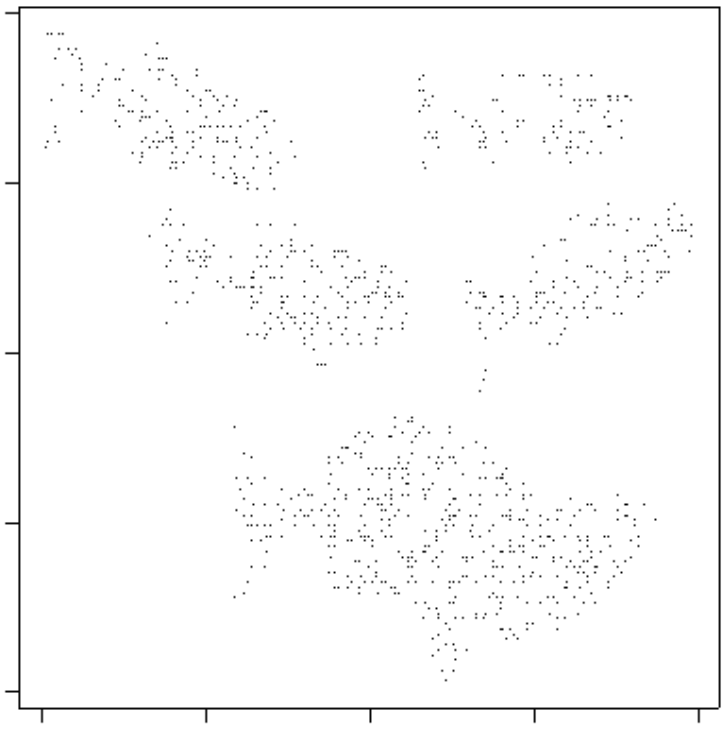


Figure 3.2.1: Data used to test model selection criteria (see Figure 2.6.1 for original image). In the first test in section 3.5 only the top 4 fish were used, in the second test all 6 fish were used.

Akaike showed that $I(g, f)$ can be expressed as the difference between two expectations, as follows:

$$I(g,f) = \int g(x \mid \theta) \log \left(\frac{g(x \mid \theta)}{f(x \mid \hat{\theta})} \right) dx \tag{3.3.2}$$

$$= -\int g(x \mid \theta) \log f(x \mid \hat{\theta}) dx + \int g(x \mid \theta) \log g(x \mid \theta) dx \tag{3.3.3}$$

$$= -\mathbb{E}_{g,\theta}[\log f(x \mid \hat{\theta})] + \mathbb{E}_{g,\theta}[\log g(x \mid \theta)] \tag{3.3.4}$$

so

$$I(g,f) - \mathbb{E}[\log g(x \mid \theta)] = -\mathbb{E}[\log f(x \mid \hat{\theta})] \tag{3.3.5}$$

$\mathbb{E}[\log g(x|\theta)]$ is an unknown constant, and so $-\mathbb{E}[\log f(x|\hat{\theta})]$ measures a relative distance between models, therefore the model which minimizes $-\mathbb{E}[\log f(x|\hat{\theta})]$ should give the closest model to the true model. This means that when we are dealing with a set of data X with estimated parameters $\hat{\theta}$, we need to minimize the maximum log likelihood for the data, i.e. $\log L(\hat{\theta}|X)$. However, Akaike found that the maximum log likelihood is biased upward as a model selection criteria when it is used to compare several alternative models with different forms (i.e. different numbers of parameters) and so he introduced the AIC to correct for this bias:

$$\text{AIC} = -2 \log L(\hat{\theta} \mid X) + 2k, \tag{3.3.6}$$

where k is the number of parameters. Akaike (1974) derives this criterion from the following expression, with the requirement that N (the number of independent observations) be sufficiently large:

$$\mathbb{E}_\infty 2NI(\theta_0; \hat{\theta}) = N\|\theta - \theta_0\|_{J^2} + k \tag{3.3.7}$$

where E_∞ is the mean of the approximate distribution; k is the number of parameters independently adjusted for the maximization of the likelihood; J refers to the Fisher information matrix and the following expression is used as an estimate of $N\|\theta - \theta_0\|_{J^2}$:

$$2 \left(\sum_{i=1}^{N} \log f(x_i \mid \theta_0) - \sum_{i=1}^{N} \log f(x_i \mid \hat{\theta}) \right) \tag{3.3.8}$$

The term $\sum \log f(x_i \mid \theta_0)$ is discarded from the above equation as it is common to all models and hence the AIC gives only a relative value for comparison of models for a particular set of observations.

Some authors (e.g. Sakamoto et al. 1986) note that the AIC may perform poorly if there are too many parameters in relation to the size of the sample, and corrections may be made in these circumstances. For example, Hurvich et al. (1998) calculated a modified AIC for estimating smooth regression functions. Only the original version of the AIC was tested here.

3.4 Minimum Description Length

Rissanen's (1987) method of estimating Minimum Description Length (MDL) is based on Shannon's Information Theory (Shannon & Weaver 1963) and is of the form:

$$\mathcal{C}(X;\hat{\theta}_k) = \mathcal{L}(X|\hat{\theta}_k) + \mathcal{L}(\hat{\theta}_k) \tag{3.4.1}$$

where $\mathcal{L}(X|\hat{\theta}_k)$ and $\mathcal{L}(\hat{\theta}_k)$ are the minimum code lengths that can be generated to describe the data and the parameters of the model, respectively. The closer the fit of the data to the model, the shorter will be the length $\mathcal{L}(X|\hat{\theta}_k)$. However, if the model is a very complex one (i.e. with many parameters) it will have a longer length for $\mathcal{L}(\hat{\theta}_k)$. This method is very similar in structure to that of Wallace & Dowe (1999) who developed the Minimum Message Length (MML) method. MML is a Bayesian method of model comparison which also has two parts, the first measures the length of the model and the second the length of the data given the model, where the *length* is the shortest code length.

The MDL is actually the minimum number of digits required to encode the data as a binary string. The data do not have to be encoded to calculate the code length as the minimum code length can be approximated using the entropy of the distribution. The MDL can be used to select a preferred model from a family of models, where a family of models $\mathcal{M}$ is defined as a collection of model classes M_k:

$$\mathcal{M} = \{M_k \mid k \in \mathcal{K}\}, \tag{3.4.2}$$

where each model in the same class has the same number of parameters k.

Rissanen (1989) defines the MDL as:

$$\text{MDL} = \min_k\{\text{stochastic complexity} + \text{model class code length}\} \tag{3.4.3}$$

which can be written as:

$$L(X \mid \mathcal{M}) = \min_k\{I(X \mid M_k) + L(M_k \mid \mathcal{M})\} \tag{3.4.4}$$

where L is the likelihood. The calculation of stochastic complexity and the code length needed to specify the model class (relative to the model family) are discussed below.

Stochastic Complexity. The shortest coded sequences can be achieved if the most frequently occurring elements in the data have the shortest code words. Because the coded elements will be of different lengths, a prefix code is used to allow the code to be uniquely decodable without the addition of extra elements to mark the beginning and end of words (Huffman 1951). In a prefix code no code can be a prefix for another code, as in the example given in the box below.

For a binary code, the length of a code word x is given by:

$$\mathcal{L}(x) = -\log_2 P(x) \tag{3.4.5}$$

where $P(x)$ is the probability of x and the probabilities are integer powers of $1/2$. If the latter requirement is not true then:

$$\mathcal{L}(x) = \lceil -\log_2 P(x) \rceil \tag{3.4.6}$$

Prefix code, an example:
Create a code for the letters $x \in \{a, b, c, d\}$, with the probabilities given below.

x	$P(x)$	$-\log_2 P(x)$	code
a	0.5	1	0
b	0.25	2	10
c	0.125	3	110
d	0.125	3	111

Example: **1101000111** is uniquely decodable as: **c b a a d**

The expected code length of a sequence with alphabet A is given by:

$$\sum_{x \in A} P(x)\mathcal{L}(x) \tag{3.4.7}$$

This length is bounded below by the entropy of P:

$$\sum_{x \in A} P(x)\mathcal{L}(x) \quad \geq \quad -\sum_{x \in A} P(x) \log_2 P(x) \tag{3.4.8}$$

In other words, the entropy of the model is greater than or equal to the entropy of the true distribution. The closer the model is to the true distribution, the shorter the code length will be. The shortest code length for the data relative to the model is called the *stochastic complexity* of the model.

Coding the Model Class. Coding the model class requires coding of the parameters, θ. To encode θ to a fixed length code word it must have a specified precision. Precision does not need to be considered when encoding the data in the first step of encoding as the value will be the same for all models. However, if the number of parameters varies between models in the family then the precision will affect the relative code lengths.

Rissanen (1989) showed that the optimal precision to choose for θ is $1/\sqrt{n}$ and he simplified the MDL for large values of n, where n is the number of data points, to:

$$\text{MDL}(k) = -\log P(X, \hat{\theta}) + \frac{k}{2} \log n. \tag{3.4.9}$$

This expression is the same as that used for the BIC, a model selection criterion presented by Schwarz (1978) using a Bayesian approach. Like the AIC, the MDL also requires large sample sizes for the approximations used in the calculation of the criterion to be valid; presumably this means large sample sizes compared to the number of parameters.

3.5 Using the Criteria for Counting Fish

The AIC and MDL criteria (equations 3.3.6 and 3.4.9, respectively) were used to count the number of fish in an image. The fish are represented by clusters of pixels (Figure 3.2.1). The log likelihood was calculated for Gaussian mixtures of 1 to 10 components, then the AIC and MDL were calculated for each of these. The results are plotted in Figures 3.5.1 and 3.5.2.

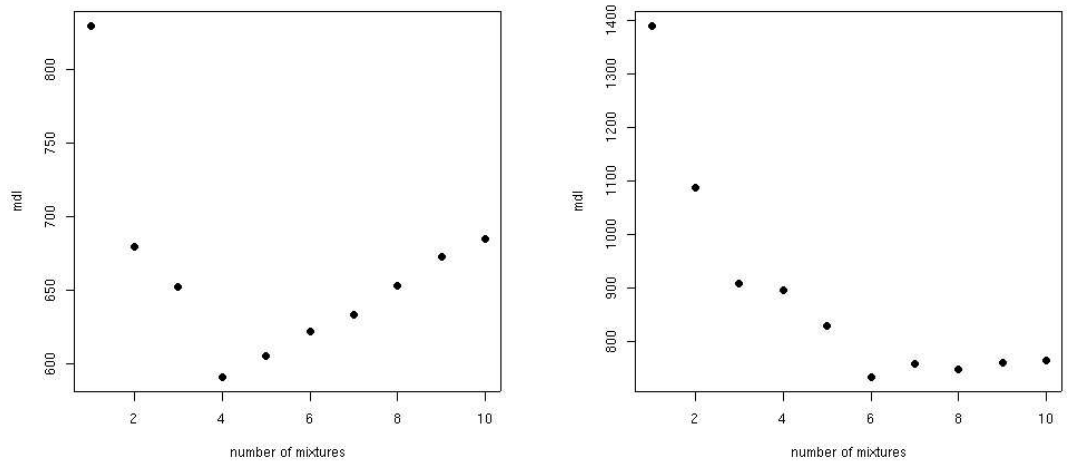
Using the MDL criterion on images with 4 and 6 fish gave the same answer a human being would get, although MDL values for mixtures of 6 and more (for the 6 fish image) are all quite low. The result for AIC is less accurate. For the 4 fish image, the lowest AIC value was for 7 components; however, all AIC values for 4 components and above were very similar. For the 6 fish image, the lowest AIC value was for the largest number of components tested, i.e. 10. The compensation for the upward bias of maximum log likelihood in the AIC is obviously insufficient and as a result the AIC overestimates the number of components.

A plot of the negative log likelihood values for the sets of data, Figure 3.5.3, shows how these values decrease as the number of components increases. However, the rate of decrease flattens out considerably once the true number of components is reached. It is this point of abrupt change in rate of decrease that indicates the true number of components. Comparison of Figure 3.5.3 with Figure 3.5.1 shows how the compensation for number of parameters converts the point of change into a minimum value. Perhaps a better criterion for selection would be one which finds the point which represents the maximum change of gradient in the curve defined by the maximum log likelihood values.

3.6 Conclusions

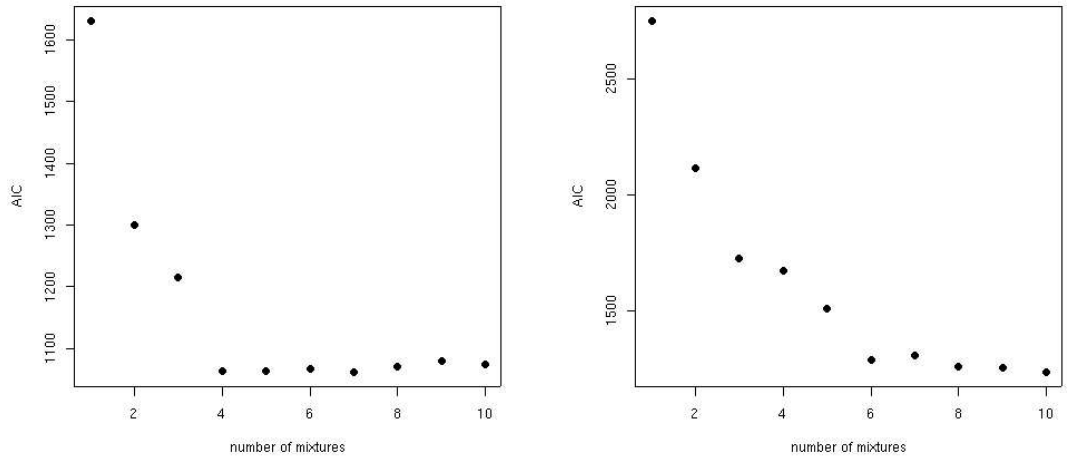
Preliminary tests indicate that the EM algorithm and the MDL selection criterion are suitable for detecting and counting the number of fish or any other objects which are close to elliptical in shape (i.e. the shape of a 2D Gaussian distribution) in an image, providing that the objects are easy to separate from the background and that the objects have minimal overlap. The AIC is not practicable as a selection criterion for this problem as it overestimates the number of objects.

It is interesting to note that although, from a human point of view, a count of 5 clusters would be considered a fair answer to the question of how many objects there are in Figure 3.2.1; both the MDL and the AIC show a much lower level of preference for the value 5 than for values greater than 6. So it is likely that neither of these methods is a particularly good simulation of human visual perception.



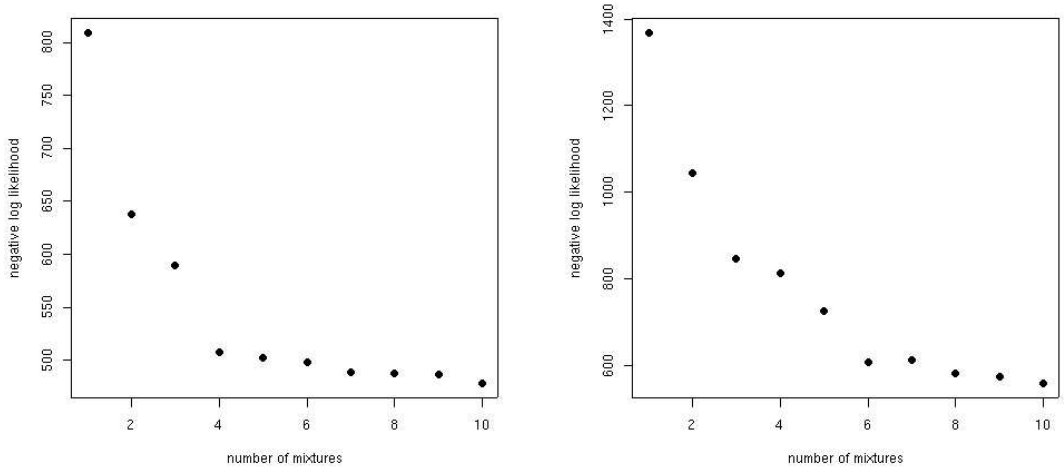
(a) (b)

Figure 3.5.1: MDL values calculated for 1 to 10 mixture components for: (a) 4 fish and (b) 6 fish. The MDL accurately predicted the number of fish in the image.



(a) (b)

Figure 3.5.2: AIC values calculated for 1 to 10 mixture components for: (a) 4 fish and (b) 6 fish. The AIC tends to over-estimate the number of components.



(a) (b)

Figure 3.5.3: Negative log likelihood values for 1 to 10 mixture components for: (a) 4 fish and (b) 6 fish. The rate of decrease of negative log likelihood values is sharply reduced when a critical number of mixtures is reached.

An Alternative Method for Finding Clusters

4.1 Introduction

The best starting position for the EM algorithm, in regard to the estimates of the means, would be to have one estimated mean per cluster, and for each cluster to have a unique estimated mean which is closer to the true mean of that cluster than any other cluster.

In order to achieve this aim I have explored a cluster finding strategy called the Dog-Rabbit strategy presented by McKenzie & Alder (1994) as a reliable method for initializing the EM algorithm for Gaussian mixture models. The Dog-Rabbit strategy has been shown to perform better than the widely used k-means algorithm as a cluster finding method (McKenzie & Alder 1994, see also Appendix D, this thesis). I have modified the Dog-Rabbit method to allow for edge effects and renamed it the Dynamic Cluster Finding algorithm (DCF). R code for the DCF algorithm is included in Appendix E.

4.2 Learning Behavior in Neurons

The Dog-Rabbit strategy (DR) is based partly on studies of the learning behavior of neurons in the visual cortex of cats and monkeys by Hubel & Wiesel (1979) and Blakemore (1974). Hubel & Wiesel (1962) discovered that specific neurons recognize characteristic simple patterns; for example, different neurons in the visual cortex respond to differently oriented edges in the field of view. They called this process feature extraction or feature detection. Furthermore, Blakemore (1974) showed that feature detection by neurons in a cat's visual cortex is a learned behavior. The aim of DR is to mimic some aspects of this learning behavior in order to group objects which have similar features. Two particular aspects of neural learning behavior, habituation and lateral inhibition, have been adopted by the DR strategy, as discussed below.

Neurons transmit information to themselves and to other neurons via circuits. These circuits may exhibit lateral inhibition (see Figure 4.2.1a). Edge detection by neurons is enhanced by the process of lateral inhibition. Lateral inhibition results in a lowering of the overall excitation level of the neurons and the differences at edges become exaggerated (p. 44, Ewert 1980). Negative feedback loops (see Figure 4.2.1b) cause a neuron to become progressively less excited by a stimulus when the stimulus is constantly repeated; this behavior is called habituation (e.g. Kandel 1979).

4.3 The Dog-Rabbit Strategy

In the DR strategy the *dogs* represent the neurons, or cluster locators, and the *rabbits* represent the features. Features can be plotted as points in $\mathbb{R}^n$, and similar

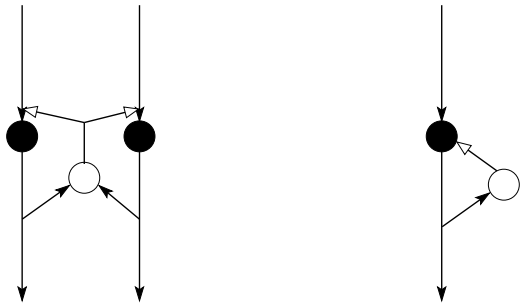


Figure 4.2.1: Simple neural circuits: (a) circuit for lateral inhibition; (b) circuit for negative feedback. The filled circles are ordinary neurons, the open circles are inhibitory interneurons (adapted from Ewert 1980).

features will plot close to each other in clusters. The cluster locators attempt to learn the location of the clusters by finding their centres. They are initialised to random positions. At each iteration of the algorithm a single data point is selected randomly from the set; this will be termed the *active* data point. Cluster locators learn the active data points by modifying their state (i.e. position) to be more similar (i.e. closer) to that of the active data point. Each time a cluster locator c_j learns about a data point x_i its location is adjusted according to the formula:

$$c_j' = \alpha\beta(x_i - c_j) + c_j \qquad x_i, c_j \in \mathbb{R}^n \tag{4.3.1}$$

where α is the lateral inhibition factor given by:

$$\alpha = \begin{cases} 1 & \text{if} \quad d_{ij} = \min\limits_{c_j}\{d\} \\ \dfrac{d_{ij}}{\lambda + d_{ij}} & \text{otherwise,} \end{cases} \tag{4.3.2}$$

and β is the habituation factor given by:

$$\beta = \frac{2d_{ij}}{(1 + d_{ij})^{h_j}}, \qquad d_{ij} = \|x_i - c_j\|.$$

λ is the inhibition parameter and h_j is the habituation parameter. The effect of the two factors, lateral inhibition and habituation, on the cluster-locating strategy is discussed below.

The desired outcome is to have each cluster of data points adopted by a different cluster locator. Lateral inhibition is used to make the active data point more attractive to the cluster locator nearest it; all other cluster locators show a much reduced reaction to the data point (hence the condition on equation 4.3.2). The result is that each locator will tend to be attracted to different clusters of data. The degree of lateral inhibition is controlled by the inhibition parameter λ.

Once a cluster locator has entered an area of dense data points, it is desirable that it should remain in that region and not be unduly influenced by distant data

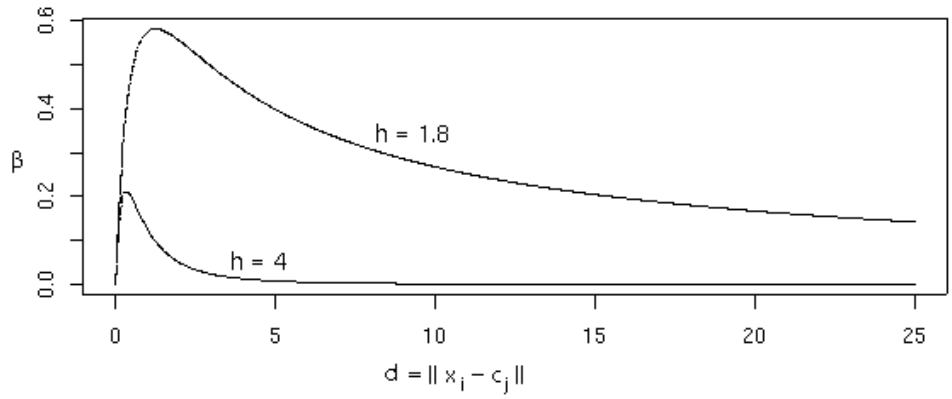


Figure 4.3.1: Shape of the function of the habituation factor (β) for habituation values of $h = 1.8$ (very weakly habituated) and $h = 4$ (strongly habituated).

points (i.e. it becomes habituated to the nearby data points). The DR strategy progressively increases the value of the habituation parameter of the locator when it arrives within a certain small distance from the active data point and so significantly reduces the distance that the locator will move in future.

Habituation is calculated at each iteration of the algorithm and is only applied to the locator which is closest to the active data point, i.e. $c_{closest} = \arg_{c_j} \min\{d_{ij}\}$. If the minimum distance $d_{ij} < distH$, where $distH$ is some predetermined small distance, then the adjusted value for the habituation of locator c_j is $h'_j = h_j + addH$, where $addH$ is the increment by which to increase habituation.

Figure 4.3.1 shows how the habituation factor (β) affects the distance moved by the cluster locator compared to the distance that the locator lies from the active data point. When a habituated cluster locator (e.g. $h = 4$) is very distant from the active data point ($d > 5$), the locator will be weakly influenced. However if the habituated locator is in the vicinity of the active data point it will be strongly influenced (near $d = 1$). And if the locator is already very close to the active data point ($d \ll 1$) it is already in a good position so again it is weakly influenced. The distance, d, required to achieve the maximum value of the habituation factor is critical to finding clusters of the desired size, this can be achieved by scaling d (see below).

4.4 Parameters and Scaling

While there are a large number of parameter values associated with inhibition and habituation to set before the DR algorithm can be run, the algorithm is fairly robust with respect to changes in these values. The best method for determining the appropriate values is to experiment with the data. If the data is always scaled

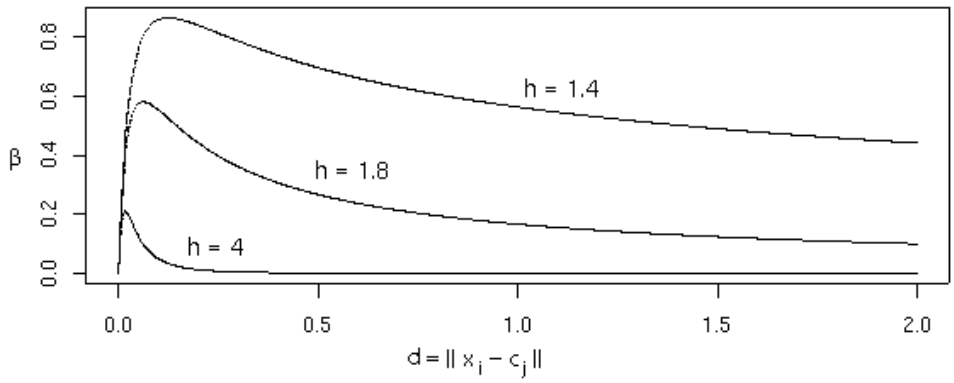


Figure 4.4.1: Shape of the function of the habituation factor (β) for habituation values of $h = 1.4$, $h = 1.8$ and $h = 4$ after scaling d by 20.

to the same range, this reduces the need to change the values of the parameters for different data sets which may have different ranges of values.

As discussed in the previous section, the equation for the habituation factor β must be modified to fit the scale of the data values, as follows,

$$\beta = \frac{scaleH \times 2d_{ij}}{(1 + scaleH \times d_{ij})^{h_j}}. \tag{4.4.1}$$

In the examples presented here, the components of the data vectors were scaled to the range $(x, y) \in [-1, 1] \times [-1, 1]$ and d was scaled by a factor of 20 (i.e. $scaleH = 20$). This scale factor modifies the function as shown in Figure 4.4.1 and is more suitable for finding clusters for data values in the chosen range.

If very small values of initial habituation ($initH$) are used then cluster locators will be strongly attracted to many different clusters and will take a long time to converge to a single cluster (see curve for $h = 1.4$ on Figure 4.4.1). A similar effect will occur if the increments of habituation are too small. If large values of initial habituation are used or increments of habituation are large then the cluster locators will become very sluggish. They will be slow to converge and may not reach a cluster before habituation becomes so high their subsequent movements are insignificant (see curve for $h = 4$ on Figure 4.4.1). Experimentation with the data indicated that reasonable values for $initH$ and $addH$ are 1.8 and 0.1, respectively.

A third parameter that must be set for for habituation, $distH$, is the maximum distance between the locator and the active data point at which to start adding habituation to the closest locator. This value is dependent on the scale of the data and size of the clusters. For these data a value of 0.1 worked well, larger values resulted in slow convergence.

4.5 Convergence of the Algorithm

The distance that each cluster locator moves at each iteration of the algorithm is highly variable because it depends on whether it is the closest locator to the active data point or not. In other words, the distance moved by each locator at each iteration is not monotonically decreasing, so distance moved cannot be used as a measure of convergence of the algorithm. As an example, Figure 4.5.1 shows the distance moved by 3 different locators during a test. The larger values of distance represent the distance moved by the locator when it was the closest to the active data point, the smaller distances would represent iterations when it was not the closest.

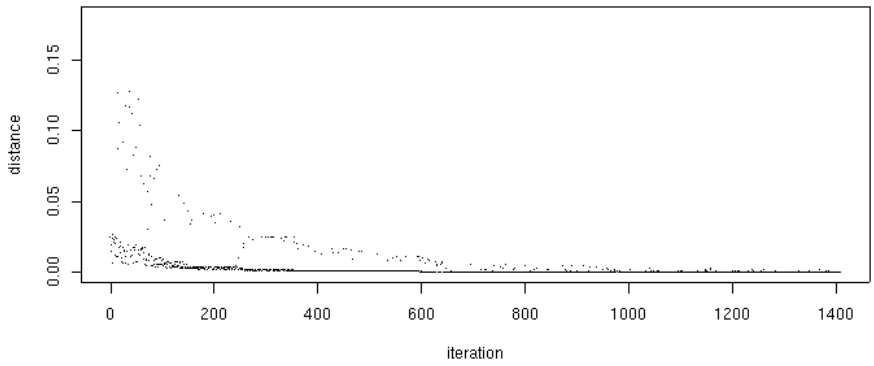
Habituation controls the distance that a locator moves and its value never decreases. It is always unchanging or increasing which suggests that it may be a good candidate for a stopping criterion. Experimentation has shown that at a habituation value of 4 the amount of movement of a locator is insignificant. The maximum distance any locator will move at a habituation of 4 can be calculated by finding the value for β at which $\frac{\mathrm{d}\beta}{\mathrm{d}d} = 0$, this evaluates to approximately 0.017. At this value the maximum distance the locator can move is given by $0.017\beta = 0.0035$, which is small enough to consider that the locator has converged. Therefore the algorithm is halted when all locators reach a habituation value of 4.

Occasionally, a locator will get stranded in a space between clusters. This can occur when all clusters already have locators attached to them and the stranded locator is then inhibited from approaching any of the clusters. This locator will not become habituated because it never approaches within $distH$ of an active data point. There must be a back-up algorithm termination procedure for these cases. In this study, the algorithm was always terminated after 2000 iterations. Any non-habituated locators can then be culled as they are considered excess to requirements, representing neurons which have learned nothing.

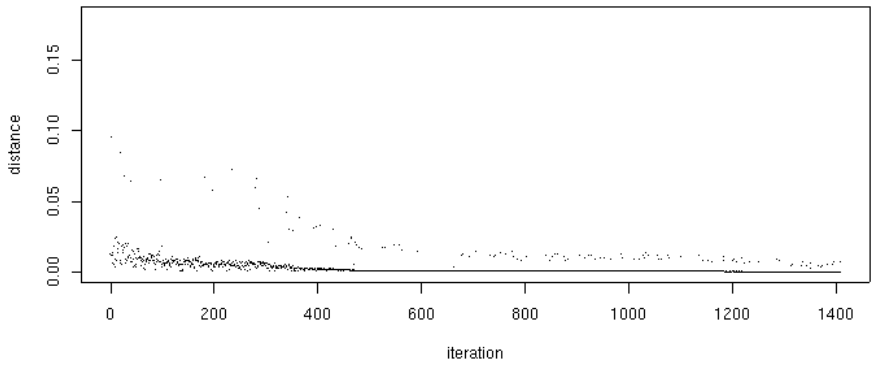
4.6 Edge Effects

When the DR algorithm was tested on a variety of cluster arrangements one problem became obvious, namely that the cluster locators tended to be attracted towards the centre of mass of the whole data set. The result of this phenomenon is that clusters near the edges of the data region may be missed, especially if they are small. The data region is the area selected by the user in which data points to be considered lie; in this case it is the area defined by: $(x, y) \in [-1, 1] \times [-1, 1]$. Cluster locators will tend to congregate on clusters nearest to the centre of the data mass or, if there is no cluster near the centre, they will accumulate in empty space, see Figure 4.6.1.

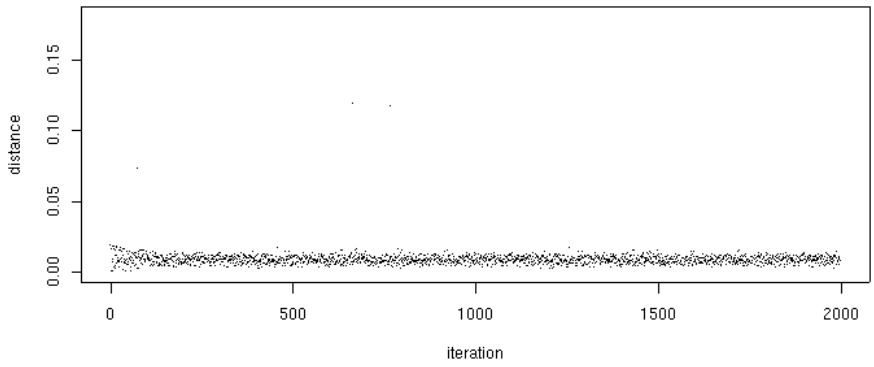
Some extreme examples are used to demonstrate this problem. Eight clusters were generated with a Gaussian probability density function and placed near the margins of the region with open space in the centre, as in Figure 4.6.1(b). The Gaussian probability density functions were calculated using the *mvrnorm* function



(a)

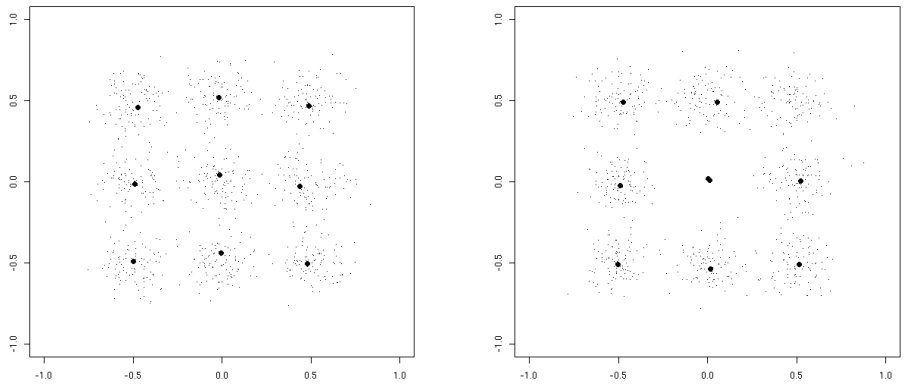


(b)



(c)

Figure 4.5.1: Distance moved by two cluster locators at each iteration in one test: (a) the first locator to reach a habituation of 4, it did so at 477 iterations; (b) the last locator (of nine locators) to reach a habituation of 4, it did so at 1409 iterations. (c) In a separate test this locator did not converge after 2000 iterations of the DR algorithm.



(a) (b)

Figure 4.6.1: Plot (a) shows the results of the DR algorithm when tested on 9 clusters (equal sized, equi-dimensional Gaussian distributions) with 9 cluster locators; this is considered a highly desirable result. Plot (b) is the same as (a) but without the centre cluster. This is considered an undesirable result: two cluster locators did not converge and are stranded in the free space in the centre of mass of the data set, as a result the top right cluster has not been detected. One of the unconverged locators was used to produce Figure 4.5.1(c).

in R, each with 100 points and a covariance matrix of $0.01 \times \mathrm{I}$:

$$\begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}$$

The DR algorithm was run 50 times with 8 different uniform random starting points for the locators each time. In 40% of the runs at least one of the cluster locators ended up stranded in the open space. If the clusters are placed closer to the boundaries of the region and the open space in the middle is consequently increased, then the percentage of runs in which one or more locators are stranded in this space is increased to 98%. Problems in spatial data which are caused by nearness to the boundaries of the data region are generally termed *edge effects*.

In the images we are studying, all objects in the image (i.e. clusters of data) are of equal importance. We do not want to show any preference for finding objects which lie near the center of the image. Because the image represents a sample of a much larger space, it is assumed that the arrangement and density of objects in the image is representative of this larger space. If there were no boundaries on the image a cluster locator could be attracted toward an object outside the image and so move toward the edges of the image away from the center of mass of the data.

This effect can be simulated by duplicating the image at each edge and each corner of the original image (resulting in 8 duplicate images, Figure 4.6.2). This has the topological effect of mapping the image onto a torus (Figure 4.6.3). Similar methods have been used for removing edge effects when calculating nearest-neighbor

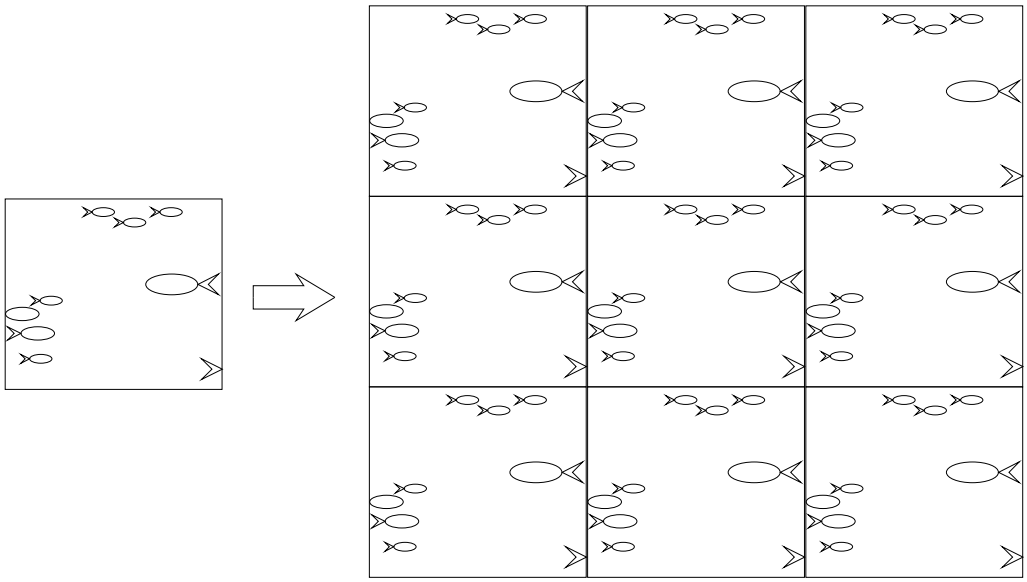


Figure 4.6.2: The left diagram shows an image, on the right the image is duplicated at the four edges and four corners of the original image.

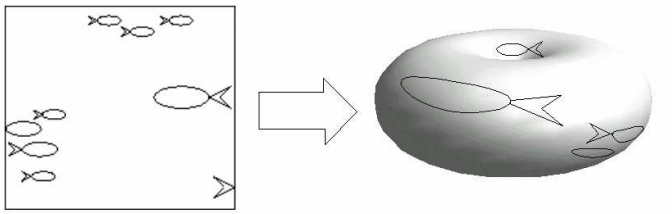


Figure 4.6.3: Duplicating the image at the edges and corners has the topological effect of wrapping the image onto a torus.

distances in spatial point statistics (e.g. Ripley 1979, Thönnes & van Lieshout 1999). This version of the DR strategy will now be called the Dynamic Cluster Finding (DCF) algorithm.

When calculating distance from each cluster locator to the active data point using DCF, nine distances must be calculated: the distance to the data point in the image and the distances to the eight duplicates of the data point. The direction to the nearest data point is identified and the locator moves in this direction, Figure 4.6.4. If the locator moves off the original image onto a duplicate then it must be shifted to the corresponding position on the original image before the next iteration of the algorithm proceeds.

By applying the DCF algorithm to the problem above we can decrease the percentage of runs in which a locator was stranded in the open space from 98% to 20%. While this appears to be a remarkable improvement, some of the gains are offset by an increased tendency of a single cluster to attract more than one locator, resulting

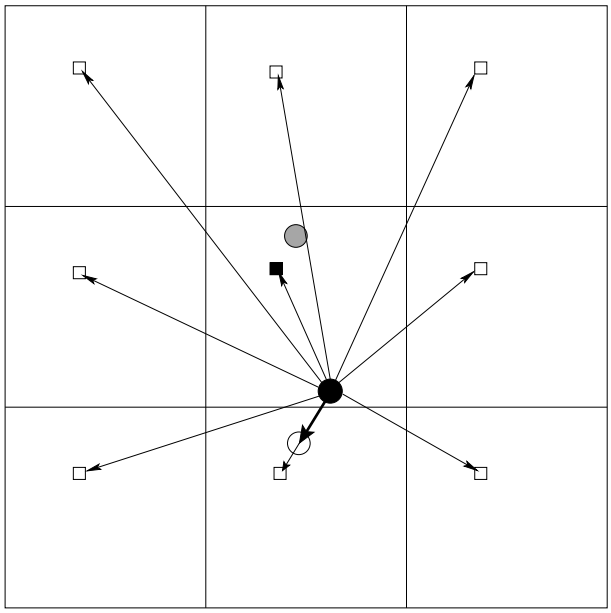


Figure 4.6.4: The distance from the cluster locator (black circle) to the active data point (black square) and its eight duplicates (open squares) is calculated; the shortest distance is selected and the locator is moved towards this point (bold arrow) to its new position (open circle). If the movement of the locator takes it beyond the original image, then the locator is moved to the corresponding position on the original image (gray circle).

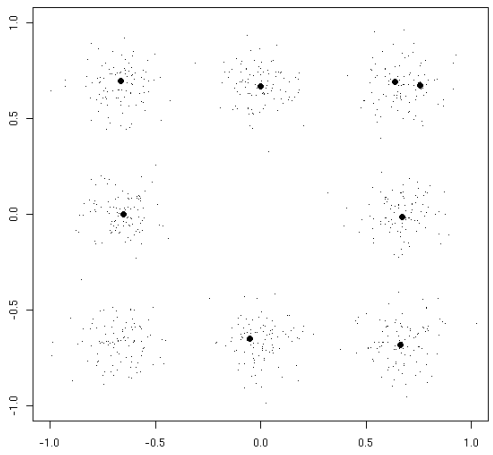


Figure 4.6.5: Final positions of cluster locators (black circles) after running DCF on 8 clusters of data points placed near the edge of the data region. Two locators have chosen the same cluster.

in some clusters being unlocated (e.g. Figure 4.6.5).

4.7 Spheroidal DR Strategy

There are other methods that one could use to remove the boundaries of the image, for example the region could be mapped onto a sphere. We compared a mapping onto a sphere with the mapping onto a torus. A simple method of mapping onto a sphere is shown in Figure 4.7.1. This is equivalent to folding the top edge of the image to the right edge and the bottom edge of the image to the left edge (it is also possible to fold the top edge of the image to the left edge and the bottom edge of the image to the right edge). The advantage of this method over the toroidal method is the shorter computation time (because there are half as many duplicate images).

Running the same set of 50 tests as in the previous section resulted in 62% of runs having a least one locator stranded in open space, which is an improvement on the DR algorithm, but not as substantial as that for the toroidal DCF. And, like the toroidal strategy, there was a tendency for a single cluster to attract more than one locator.

4.8 How many Cluster Locators?

The DCF algorithm is very successful at finding all the clusters if you already know how many clusters you have (see also Appendix D). This section looks at how to find all the clusters when you don't know how many clusters to look for. If we are to imitate the human brain with its extraordinarily large number of neurons then we might wish to start with a very large number of locators, far more than we actually think we need and then ignore the ones that don't learn anything (i.e. the ones that do not become habituated).

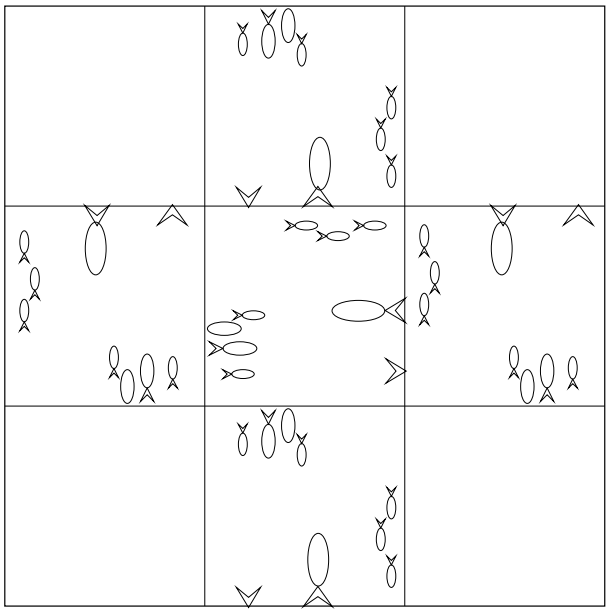


Figure 4.7.1: The image is duplicated at the four edges to simulate a spheroidal topology.

When a very large number of locators are used the interaction between locators at each iteration of the algorithm may be different than when dealing with small numbers of locators, so I have experimented with varying values of lateral inhibition to see how this parameter affects the behaviour of the locators and so choose a good value for inhibition for future runs. For this purpose artificial data was created.

A test on a simple configuration of data points, arranged in two identical and well separated clusters, was performed. Each cluster contained 100 points selected using R's *mvrnorm* function for Gaussian probability density functions (this is the method used for creating all the clusters described in this chapter). The covariance matrix for the probability density functions is $0.01 \times \mathrm{I}$. The results for inhibition values of 1, 10 and 100 are illustrated in Figure 4.8.1. The first plot clearly shows that in the first two iterations (starting at the open circles) the locators were attracted to the right-hand cluster, in the next two iterations they were attracted to the left-hand cluster and so on, as shown by the trail of black dots. As the inhibition increases (second and third plots) the steps taken at each iteration decrease in size.

The plots also show that all the cluster locators are strongly attracted to the region in which the clusters lie. However, the imposed lateral inhibition effect means that once a cluster locator has reached a cluster of data it prevents the other locators from approaching as closely with the result that the excess locators tend to congregate in the spaces between the clusters.

Interestingly, the higher inhibition values result in more locators on each cluster. The reason for this phenomenon appears to be because the short steps taken by the cluster locators at each iteration mean that the locators which start near a cluster do not move far away from the cluster when data points from other clusters try to

Table 4.8.1: Numbers of clusters located in Figure 4.8.2a, b and c using varying values of λ. These values can be compared to a human being's interpretation of the actual number of clusters present in the data set, shown on the last line.

λ	Fig.4.8.2(a)	Fig.4.8.2(b)	Fig.4.8.2(c)
0.1	0	0	2
1	3	2	5
5	7	5	8
10	9	8	9
40	9	8	12
60	10	10	9
100	11	11	10
1000	12	11	8
actual no. of clusters:	3 large or 9 small	2 large and 1 small or 9 small	5

attract them. The result is that they tend to remain with the original cluster rather than being attracted into the space between clusters.

Further tests were then carried out on more complex arrangements of data, namely clusters which occur in groups. In the first two examples (Figure 4.8.2a, b) the clusters are arranged in groups; in the third example (Figure 4.8.2c) there is a single group of clusters surrounded by a small amount of noise (5%).

The tests described below used 30 cluster locators. The algorithm was run for 800 iterations and then terminated. Cluster locators which had a habituation value of less than 4 were eliminated, the remaining locators were checked visually, and in all cases these locators had terminated on clusters of data. High numbers of locators are desirable to simulate the large numbers of neurons in a brain, however, increasing the number has a detrimental effect on the speed of the algorithm; the value of 30 locators was chosen as a compromise. The value of 800 iterations was chosen after some experimentation showed that the number of locators to reach terminal habituation value generally did not increase by much after 800 iterations.

The number of clusters located for a variety of values of lateral inhibition are given in Table 4.8.1. The correct number of clusters can be determined by viewing the clusters in Figure 4.8.2 and counting them. The correct number of clusters may vary depending on whether the viewer counts only the well separated clusters or whether the viewer also counts the sub-clusters within the well separated clusters. Table 4.8.1 shows that larger values of λ resulted in a higher estimate of the number of clusters and small values resulted in an lower estimate. However, the number of clusters found tends to plateau out at high values of λ. In the noisy example (c), the cluster locators become distracted by the noise and, because the distance moved at each step is much smaller at higher inhibition, fewer of the locators reach the region in which the clusters lie as inhibition increases.

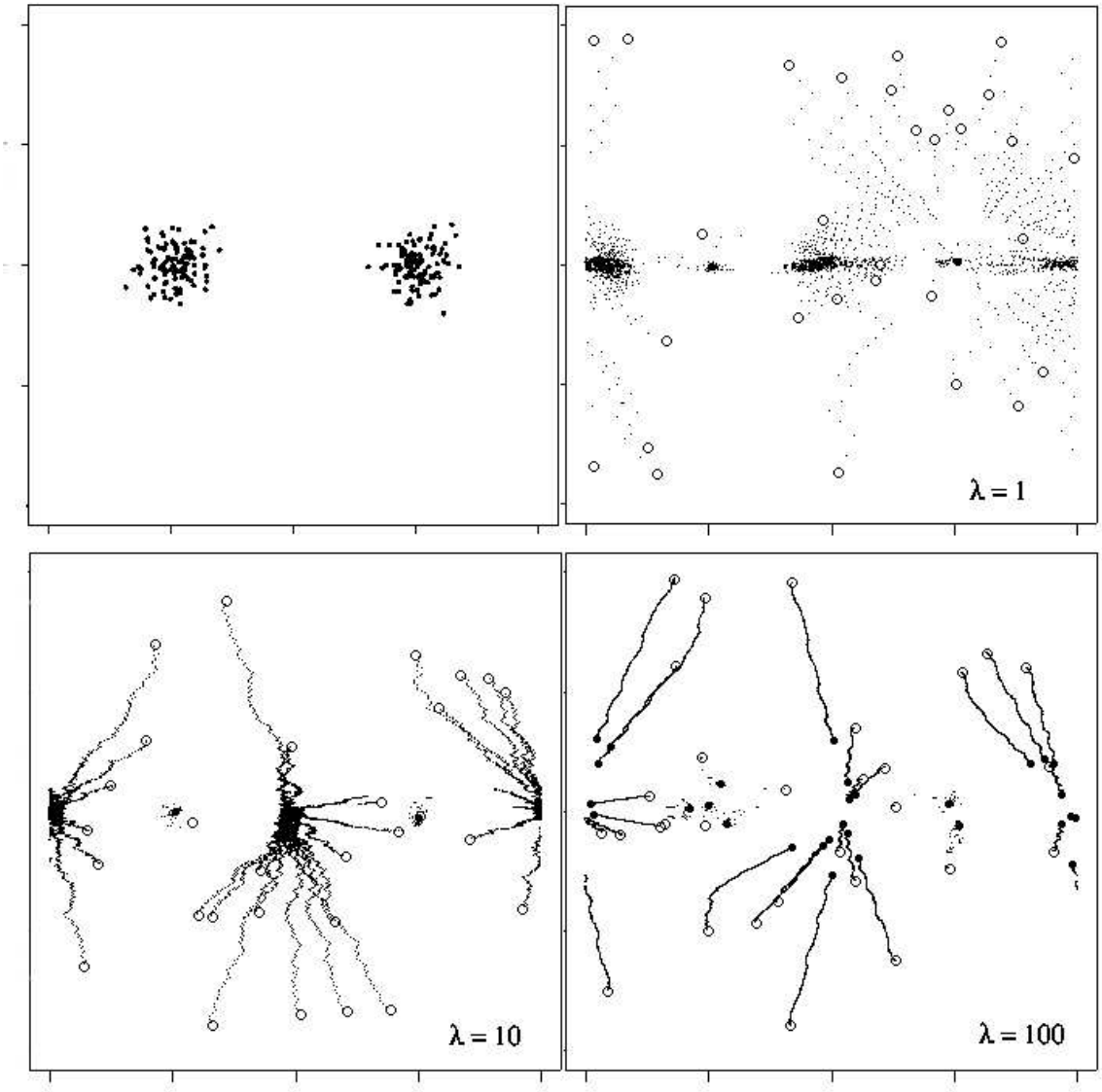
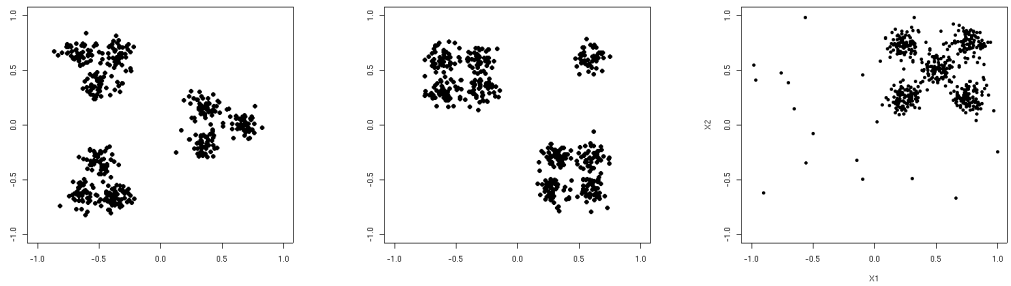


Figure 4.8.1: Behaviour of cluster locators near two Gaussian clusters when varying inhibition values λ are used. Top left figure shows the distribution of data in the two clusters. The other figures show the location of the locators at each iteration of the DCF algorithm. Open circles represent random starting positions for the locators, closed circles are the final positions, and small black dots show the path taken by the locators. At $\lambda = 1$ and $\lambda = 10$ each cluster was found by one locator; at $\lambda = 100$ the clusters were found by 4 and 2 locators, respectively.



(a) (b) (c)

Figure 4.8.2: (a) Nine clusters arranged in 3 groups of 3; (b) nine clusters arranged in 2 groups of 4 plus a single cluster; (c) five clusters arranged in a tight group, plus noise.

4.9 Discussion: How Many Clusters?

The results of the experiments in the previous section suggest that we can control the number of clusters we wish to find by varying the inhibition value. Lower inhibition values ($\lambda \simeq 1$) will find only the well-separated clusters of data, and higher inhibition values ($\lambda \geq 10$) are more likely to detect sub-structure in the larger clusters.

The answer to the question *how many clusters are really there?* depends on what the user is looking for. In Figure 4.8.2(a) the value 3 may be the desirable answer in some circumstances, and 9 in others. The best answer from a pattern recognition point of view would be *there are 3 large clusters with a further 3 sub-clusters in each*, but simple cluster finding algorithms such as k-means and DCF cannot be expected to deliver this level of sophistication. In pattern recognition problems simply finding the clusters of data is only the first step in the process. The significance of the detected clusters can be determined by further processing of the information available and including any prior knowledge we may have about the structure of objects we expect to find.

For example, in the problem of counting fish from digital images, a single cluster of pixels may represent a tightly bunched school of many fish, in which case the cluster needs to be broken down into sub-clusters. There needs to be some method of checking that each sub-cluster corresponds to one fish, such as checking that the shape of the cluster fits the shape of a fish, or looking for a single eye near one end of each elongate cluster (as in Appendix C).

Alternatively, a cluster of pixels may represent only part of a fish, such as a distinctive, coloured patch on the body or a line of reflected light on the upper surface of the body. If this is the case then we need a method of assembling the various parts, considering their shape and spatial location in the image, to show that they represent parts of a single fish.

The next two chapters will demonstrate how fish-like objects can be built up

from simple components.

Fitting Curves to Edges

5.1 Introduction

An alternative approach to the problem of finding objects in an image is to use edge detectors and to fit smooth curves to the edges. Recognising the fish will require matching of the shapes of the curves found in an image with those known to be characteristic of fish. The more matching curves that are found the greater the confidence of recognising a fish. Where there is more than one curve matched, the spatial relationships of the various curves are important. For example a tail curve must be at the tail end of a head curve to be the same fish; or an upper body curve must lie above a lower body curve and face in the same direction to be the same fish. There are two major parts to this process: the first is to create smooth curves for the edges (this chapter) and the second is to match curves (Chapter 6).

The methods used in this chapter rely on the aggregation of relatively small objects into larger, more complex objects. This is a form of syntactic pattern recognition similar to the well known pattern recognition problem of recognising types of chromosomes from the curved segments of which the chromosome is composed (e.g. Fu 1982). In this study there are several stages of aggregation; first pixels are aggregated into small neighbourhoods (in order to represent small straight lines segments), next the neighbourhoods are aggregated into gentle curves and smoothed, finally the curves are aggregated into a fish.

The original jpeg images referred to in this chapter and in the next chapter are available on the CD included with Appendix E, in directory: code/R-code/curve/data/.

5.2 Edge Detectors

There are many edge detectors available. Most involve convolution of the image with a filter where the filter simultaneously smooths the image (to remove high frequency noise) and applies an edge detector which approximates the first derivative of the image's pixel values. Local maxima in the first derivative represent edges in the image.

The Canny edge detector (Canny 1986) was used because it gave more useful results than other well known methods such as the Sobel or Laplacian of Gaussian edge detectors. Figure 5.2.1 shows the results from these 3 edge detectors; it is clear from these figures that the Canny edge detector produces clearer, more continuous lines which are potentially more useful for detecting continuous curves which define the features of the fish. This edge detector uses the first derivative of a Gaussian filter to simultaneously smooth and detect edges. These edges are then refined by finding lines defined by connected edge pixels whose values exceed hysteresis thresholds. Hysteresis thresholding means that, to start a new line, the pixel values must exceed a certain threshold, but connected edge pixels need only to exceed a

lower threshold. This method of thresholding gives better continuity to lines because it includes parts of lines which have weak edges but does not include lines which only have weak edges.

The method used for edge detection is independent of the methods used for finding curves described in the next section. It does not matter what edge detector is used as long as satisfactory edges are found. However, it is important that edges within the fish as well as edges which represent the boundary of the fish are detected. This is for two reasons: (1) the boundaries of the fish may be unclear when the background is a similar colour to that of the fish but the patterns on the body of the fish may provide sufficient extra information for a fish to be recognisable (see Figure 1.1); and (2) patterns on the body of the fish are essential for distinguishing between species of fish which have similar shapes. For these reasons closed curve edge detectors such as 'snakes' or active contour methods are not used. Additionally, 'snakes' and active contour methods were not used because they describe curves in relation to subsets of points (knots) and therefore the description depends on how the knot points are selected. It is essential that the curve description used here allows any curve or subsection of the curve to be compared to any other curve or subsection of it and to be found similar if the shape of the curves are similar, independent of the method used to construct the description.

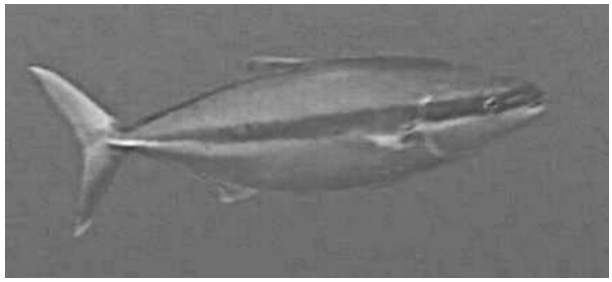
5.3 Finding Curves

Once edges have been found these need to be transformed into smooth curves which match the smooth curves of the fish. The lines found by the edge detector are not smooth and contain numerous small breaks between sections. In order to deal with noise in the edge detection process the curves can be modelled using a partially stochastic process, such as the *Quadratic UpWrite* of McLaughlin & Alder (1998).

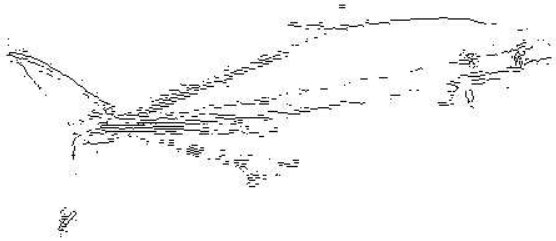
In McLaughlin & Alder (1998) the authors first segmented the black and white image into small neighbourhoods by repeatedly selecting a black pixel and assigning all other black pixels within a radius of that pixel to its neighbourhood. Data within a neighbourhood was tagged so that it would not be re-used in another neighbourhood. Secondly the data within each neighbourhood was modeled with a straight line. The line of best fit was the one which passed through the mean of the neighbourhood points, with an orientation parallel to the principal eigenvector of the covariance matrix of the data, this is the *local model*. Finally the local models were grouped if they lay along the same geometric feature, such as a curve. A predictive element was incorporated in the *UpWrite* process by assuming locally constant curvature of the curve.

I have modified the original *Quadratic UpWrite* in the following ways:

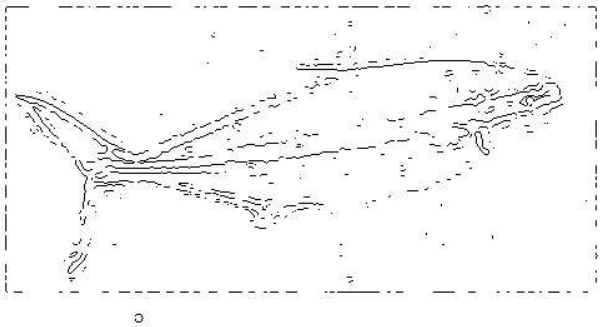
- the algorithm picks the first pixel in the data file (usually the lower left pixel of the image) and finds the neighbourhood, pixels in the neighbourhood are tagged. While tagged pixels cannot be used to start a search for a new curve



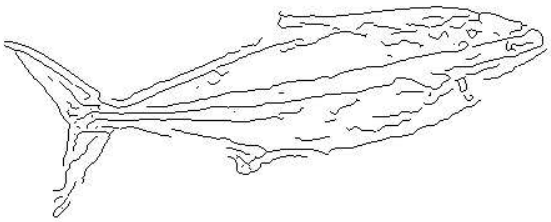
(a) digital image of a kingfish



(b) Sobel



(c) Laplacian of Gaussian



(d) Canny edge detector with thresholds at 0.05 and 0.08, and standard deviation of the Gaussian filter set at 1.5.

Figure 5.2.1: Comparison of results of edge detectors on an image of a kingfish. The Canny edge detector has the advantage of removing weak edges while retaining continuity of lines. Figures (b)-(d) were created using the MATLAB Image Processing Toolbox.

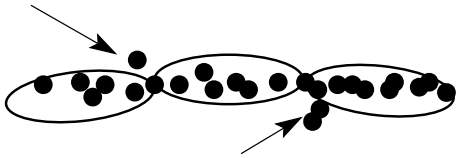


Figure 5.3.1: Two untagged points (arrowed) not included in an initial neighbourhood search (neighbourhoods defined by ellipses) may be used to find the next curve, the new curve may reuse any of the tagged points in the previous neighbourhoods to define a new curve.

they can be re-used in other neighbourhoods, unlike the original *UpWrite* tagging. The reason for this modification is discussed below.

- The next neighbourhood is found by predicting its location using the mean and covariance of the current neighbourhood (described below).
- After the second neighbourhood has been found, the third neighbourhood and all subsequent neighbourhoods are predicted using the previous 2 neighbourhoods (described below).
- The curve search stops when a new neighbourhood contains less than a minimum number of pixels (head of curve), the search for neighbourhoods is then repeated from the first neighbourhood but in the opposite direction (tail of curve) until a new neighbourhood contains less than a minimum number of pixels.

The modification of the original *UpWrite* to allow tagged pixels to be re-used for new curves searches means that curves which cross each other may share pixels in the region of the crossing. Another result of this method is that several fits may be found for a single curve. For example, in Figure 5.3.1, if the two arrowed points were not included in the neighbourhoods of the curve defined by the elliptical neighbourhoods, then a new curve search might start at either of these two points and redefine the same curve in a slightly different shape. How this variation in shape affects the curvature will be discussed in the following chapter.

The algorithm is designed to find geometric features which have gentle curvature and to terminate when the adjacent pixels are no longer consistent with a gently curving feature. The algorithm works in this way as we require a set of fairly simple curves which will be easy to compare with other curves. The algorithms for finding curves (this chapter) and matching curves (chapter 6) were all written in R, they are included in Appendix E.

Predicting the location of the second neighbourhood. Using a predictive process has the advantage of potentially finding longer lines where they are cryptic; however, it has the drawback of possibly continuing a line into regions of data where the data no longer defines a strictly linear feature of the type predicted, see Figure 5.3.2 for examples of these situations.

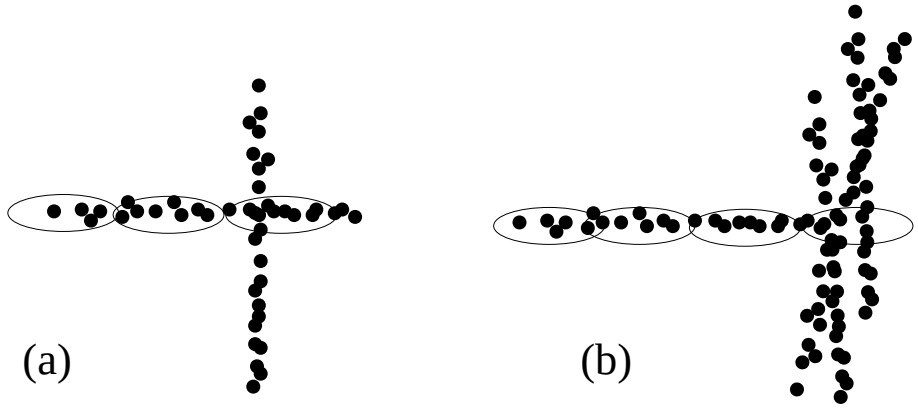


Figure 5.3.2: Using a predictive process for finding the next neighbourhood on a geometric feature in the *UpWrite* can be useful for situations such as (a) but not for (b). Using prediction to find the next neighbourhood in (a) allows the horizontal curve to be continued past the intersection with another curve. However, (b) shows how prediction may cause the curve to be extended beyond the end of the geometric feature if data from other geometric features coincides with the predicted neighbourhood.

The location of the second neighbourhood is predicted to lie in the direction of the dominant eigenvector of the covariance matrix of the data in the first neighbourhood. The second neighbourhood contains the pixels which lie within an area bounded by the following limits:

1. $minD < d < maxD$

2. $cos(a) \geq cos(maxA)$

These limits and the shape of the new neighbourhood defined by these limits is shown in Figure 5.3.3. d is the euclidean distance from the mean of the first neighbourhood to the pixel under consideration; $minD$ and $maxD$ are distances which are proportional to the length of the principal eigenvector; a is the angle between the principal eigenvector and the line from the mean of the first neighbourhood to the pixel. Values for these parameters were determined experimentally.

The ellipse shown in the Figure 5.3.3 (and all other ellipses shown in figures in this chapter) represents the mean and covariance of the neighbourhood's pixel distribution. The long axis of each ellipse is parallel to and twice the length of the principal eigenvector of the neighbourhood (i.e. it is an M2 ellipse[1]). This neighbourhood shape is designed to capture pixels which lie along a geometric feature which may continue to be parallel to the feature modeled by the current neighbourhood, or may curve gently away from the current neighbourhood in either direction.

[1]see Chapter 2, section 2.6 for an explanation of the M2 ellipse

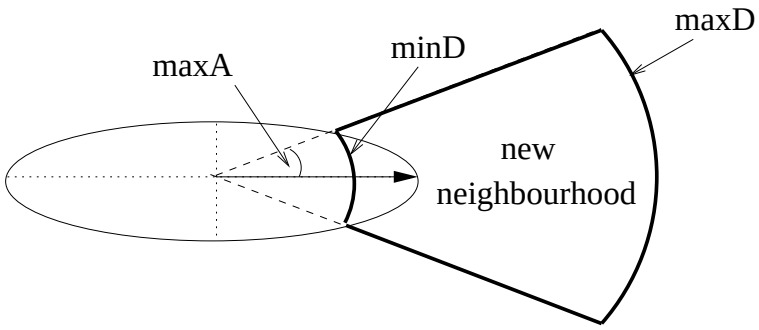


Figure 5.3.3: Method for finding the second neighbourhood. The ellipse represents the mean and covariance (i.e. an M2 ellipse) of the first neighbourhood. The second neighbourhood lies within a distance of $minD$ and $maxD$ from the mean, and within an angle of $maxA$ from the principal eigenvector (shown as an arrow).

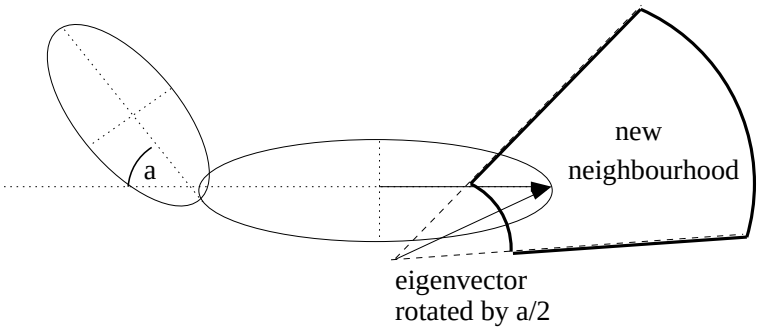


Figure 5.3.4: Method for finding the third and subsequent neighbourhoods in a sequence of neighbourhoods where there is an angle, $a > 0$, between the line segments of the two previous neighbourhoods.

Predicting the location of subsequent neighbourhoods. After the second neighbourhood has been found, the direction of curvature is predicted as follows: if the principal eigenvectors of the last two neighbourhoods are not parallel then the subsequent neighbourhood is rotated by an angle proportional to the angle between the previous two neighbourhoods. The angle for the new neighbourhood needs to take into consideration that constant curvature should not be assumed, on a natural object like a fish the curve may continue to bend or it may straighten out. Experimentation revealed that an angle of half the angle between the previous eigenvectors worked well. The rotation of the new neighbourhood is shown in Figure 5.3.4. The neighbourhood is rotated at the point where the M2 ellipse crosses the principal eigenvector.

Figure 5.3.5 shows how a chain of M2 ellipses can be used to represent a curve found using the *UpWrite* procedure.

Figure 5.3.6 shows the results of the new *UpWrite* applied on the kingfish image in Figure 5.2.1(a). For the example shown in Figure 5.3.6, the parameters $minD$ and $maxD$ had the values 0.7×length principal eigenvector and 4.5×length principal

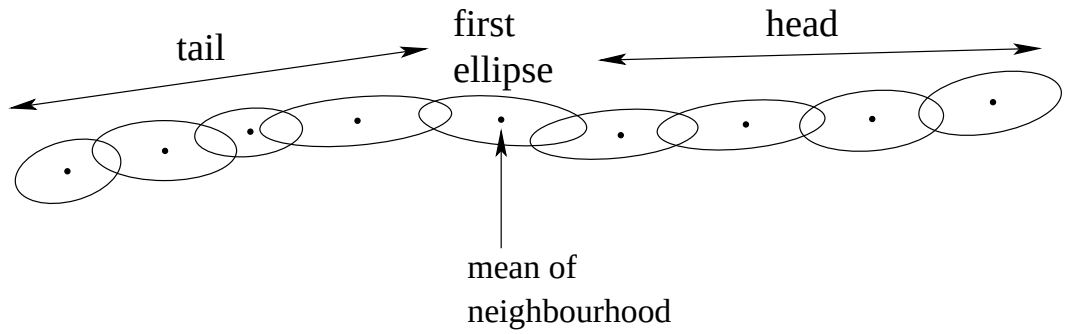


Figure 5.3.5: Chain representation of line segments in a curve. Each link in the chain is an M2 ellipse. The point in the centre of each ellipse represents the mean of the neighbourhood.

eigenvector, respectively; $maxA$ had the value $\pi/8$. The radius of the neighbourhood was 0.03. The best values to use for the parameters were determined by experimentation.

The final step in creating smooth curves is to fit cubic polynomials to the set of means in the chain of neighbourhoods. The best cubic polynomial fit is chosen using the least squares method. The final result is a set of smooth curves which approximate the edges in the image (Figure 5.3.7).

Although the results illustrated in Figure 5.3.7 suggest that the method of fitting cubic polynomials is satisfactory, the method would have failed if any of the curves contained vertical sections. The cubic polynomials are represented in the form:

$$[a, b, c, d]$$

where a,b,c and d are the coefficients of the polynomial:

$$y = ax^3 + bx^2 + cx + d.$$

However, if a curve passes through the vertical then it can not be represented by a single function. A parametric representation of the curve will overcome this problem:

The parametric polynomial equations are:

$x = \mathrm{a}_1 t^3 + \mathrm{b}_1 t^2 + \mathrm{c}_1 t + \mathrm{d}_1$ and

$y = \mathrm{a}_2 t^3 + \mathrm{b}_2 t^2 + \mathrm{c}_2 t + \mathrm{d}_2$,

5.4 Describing Curves Using Arc Length and Signed Curvature

Once smooth curves have been fitted to the edges of the object, the next step is to find a format for describing or defining the curve which will allow curves to be compared. Curves that have been found in an image need to be matched against a reference set of curves which represent typical shapes of the object we are seeking. Because a cubic polynomial is potentially infinite in length, we also need to ensure

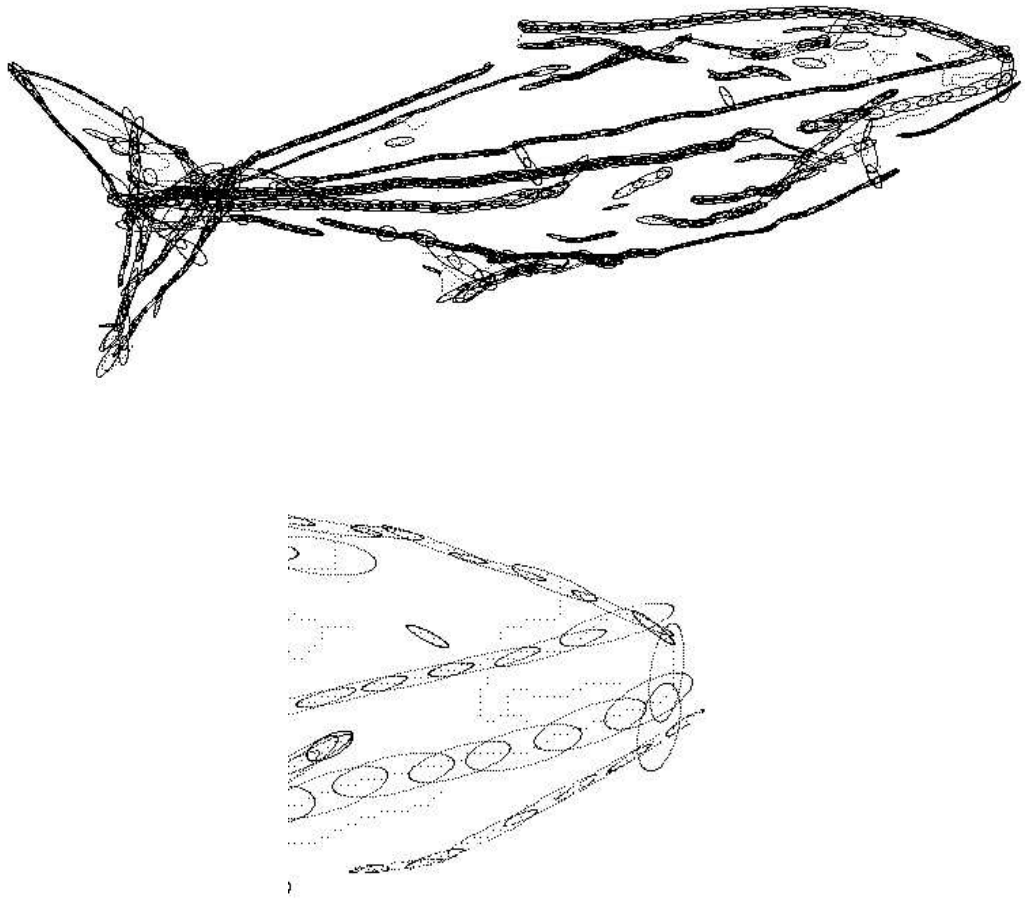


Figure 5.3.6: (a) Results of the search for line segments which lie on curves for the kingfish in Figure 5.2.1; small dots indicate the positions of the edge pixels. (b) Enlarged view of the head region of the fish.

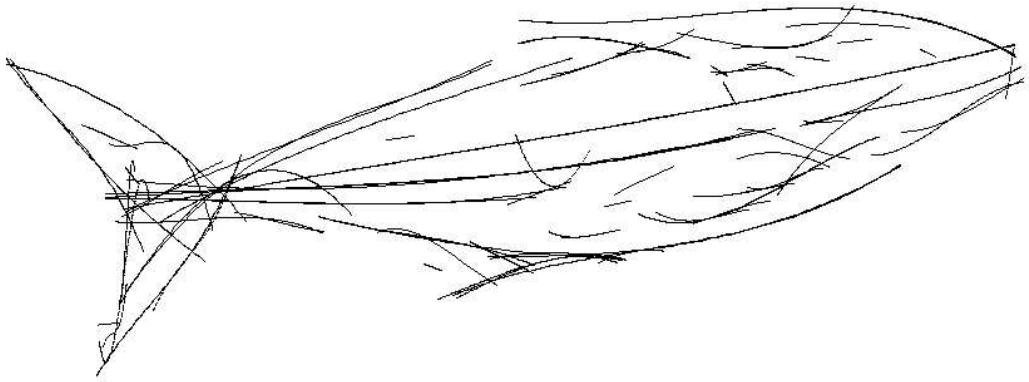


Figure 5.3.7: Cubic polynomials fitted to means of ellipse chains using least squares method.

that the particular interval on which the polynomial matches the curve is included in the description of the curve.

Comparing coefficients of intervals of polynomial curves is not a good method for comparing curves as they are neither rotation nor translation invariant and they are not stable under minor perturbations. Therefore the polynomial description of each curve has been transformed into a signed curvature description, which is invariant to translations and rotations.

In this study I do not address scale invariance and mirror imaging of curves. Although these are important issues they are beyond the scope of this study and should be addressed in any future work. It is not required that the curvature descriptions be invariant under all affine transformations as fish do not undergo significant shearing or stretching transformations[2].

Signed curvature may be defined as (see, for example, Rutter (2000)):

$$\begin{aligned}
\kappa_S &= S\|\mathrm{T}'(s)\| && (5.4.1)\\
&= S\left\|\frac{\mathrm{d}}{\mathrm{d}s}(\mathrm{T}(s))\right\| && (5.4.2)\\
&= S\frac{1}{\|\mathbf{r}'(t)\|}\left\|\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{\|\mathbf{r}'(t)\|}\mathbf{r}'(t)\right)\right\| && (5.4.3)\\
&= S\frac{1}{\|\mathbf{r}'(t)\|}\left\|\frac{\mathrm{d}}{\mathrm{d}t}(\mathrm{T}(t))\right\| && (5.4.4)
\end{aligned}$$

Where S is the sign of the curvature, T is the tangent to the curve and s is the arc length parameter. t is the original parameter (in this case, the index number of the point in an array of points representing the curve), $\mathbf{r}$ is the function of the curve and $\mathbf{r}'$ is the first derivative of the function of the curve.

For two parametric cubic polynomials x and y the equation for unsigned curvature is:

$$\kappa = \frac{1}{\|\mathbf{r}'(t)\|}\left\|\frac{\mathrm{d}}{\mathrm{d}t}(\mathrm{T}(t))\right\| \qquad (5.4.5)$$

and is computed using:

$$\begin{aligned}
\|\mathbf{r}'(t)\| &= \left\|\begin{pmatrix} x' \\ y' \end{pmatrix}\right\|, \text{ where} && (5.4.6)\\
x' &= 3\mathrm{a}_1t^2 + 2\mathrm{b}_1t + \mathrm{c}_1 && (5.4.7)\\
y' &= 3\mathrm{a}_2t^2 + 2\mathrm{b}_2t + \mathrm{c}_2 && (5.4.8)
\end{aligned}$$

[2]Affine invariant edge detection for image segmentation is described by Sapiro (2001). However, affine invariance provides too much freedom to change the shape of the fish. Significant shearing or stretching of the shape of a fish (or part of a fish) may cause one species to be mistaken for another.

and

$$\left\|\frac{\mathrm{d}}{\mathrm{d}t}(\mathrm{T}(t))\right\| = \left\|\begin{pmatrix} p \\ q \end{pmatrix}\right\|, \text{ where} \tag{5.4.9}$$

$$p = \frac{x''}{\|\mathbf{r}'(t)\|} - \frac{x'(\frac{\mathrm{d}(x')^2}{\mathrm{d}t} + \frac{\mathrm{d}(y')^2}{\mathrm{d}t})}{\sqrt{((x')^2 + (y')^2)^3}} \tag{5.4.10}$$

$$q = \frac{y''}{\|\mathbf{r}'(t)\|} - \frac{y'(\frac{\mathrm{d}(x')^2}{\mathrm{d}t} + \frac{\mathrm{d}(y')^2}{\mathrm{d}t})}{\sqrt{((x')^2 + (y')^2)^3}}, \text{ and} \tag{5.4.11}$$

$$x'' = 6\mathrm{a}_1 t + 2\mathrm{b}_1 \tag{5.4.12}$$

$$y'' = 6\mathrm{a}_2 t + 2\mathrm{b}_2 \tag{5.4.13}$$

The equations for p and q are calculated as follows:

$$\left\|\begin{pmatrix} p \\ q \end{pmatrix}\right\| = \left\|\frac{\mathrm{d}}{\mathrm{d}t}(\mathrm{T}(t))\right\| \tag{5.4.14}$$

$$= \left\|\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathbf{r}'(t)}{\|\mathbf{r}'(t)\|}\right)\right| \tag{5.4.15}$$

$$\mathrm{p} = \frac{\mathrm{d}}{\mathrm{d}(t)}\left(\frac{x'}{\sqrt{(x')^2 + (y')^2}}\right) \tag{5.4.16}$$

$$= \frac{\frac{\mathrm{d}(x')}{\mathrm{d}(t)} \cdot (\sqrt{(x')^2 + (y')^2}) - 0.5x' \frac{\mathrm{d}}{\mathrm{d}(t)}(\sqrt{(x')^2 + (y')^2})}{(x')^2 + (y')^2} \tag{5.4.17}$$

$$= \frac{\frac{\mathrm{d}(x')}{\mathrm{d}(t)}}{\sqrt{(x')^2 + (y')^2}} - \frac{0.5x'\left(\frac{\mathrm{d}(x')^2}{\mathrm{d}(t)} + \frac{\mathrm{d}(y')^2}{\mathrm{d}(t)}\right)}{\sqrt{((x')^2 + (y')^2)^3}} \tag{5.4.18}$$

$$\mathrm{q} = \frac{\mathrm{d}}{\mathrm{d}(t)}\left(\frac{y'}{\sqrt{(x')^2 + (y')^2}}\right) \tag{5.4.19}$$

$$= \frac{\frac{\mathrm{d}(y')}{\mathrm{d}(t)} \cdot (\sqrt{(x')^2 + (y')^2}) - 0.5y' \frac{\mathrm{d}}{\mathrm{d}(t)}(\sqrt{(x')^2 + (y')^2})}{(x')^2 + (y')^2} \tag{5.4.20}$$

$$= \frac{\frac{\mathrm{d}(y')}{\mathrm{d}(t)}}{\sqrt{(x')^2 + (y')^2}} - \frac{0.5y'\left(\frac{\mathrm{d}(x')^2}{\mathrm{d}(t)} + \frac{\mathrm{d}(y')^2}{\mathrm{d}(t)}\right)}{\sqrt{((x')^2 + (y')^2)^3}} \tag{5.4.21}$$

In this study the sign S is defined as the rotation direction from the tangent to the curve to the normal of the curve (Figure 5.4.1). If this direction is anti-clockwise

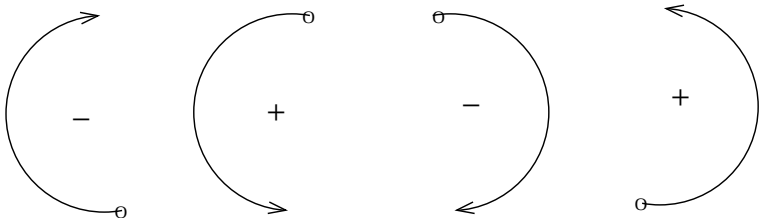


Figure 5.4.1: Examples of the sign of curvature.

then $S = 1$ and if clockwise then $S = -1$. It is calculated using:

$$S = sign\left(det\begin{pmatrix} x' & x'' \\ y' & y'' \end{pmatrix}\right) \tag{5.4.22}$$

The curve shape can now be described as a series of curvature measurements taken at points which are separated by equal distances measured in arc length along the curve (this is the arc length parameterisation) from the start point to the end point of the curve. Examples of curvatures calculated for curve segments which represent parts of fish are given in the next chapter.

Comparing Curves

6.1 Introduction

In this chapter the methods outlined in the previous chapter are used to describe the shape of segments of curves as a series of signed curvature measurements. First, signed curvature is calculated for the values of the original parameter t, then these values are linearly interpolated to find the curvature for the arc length parameter s. In the examples described below the arc length separating each curvature measurement is 0.01 (the best interval to choose depends on the scale of the data and the desired precision of curve matching).

Once the signed curvature has been calculated at fixed intervals along a curve, a method for comparing a new set of curves to a reference set of curves needs to be established. The reference set of curves represents typical curves for one object of the type we wish to recognize. The method devised here comprises three steps:

1. *Curvature:* compare the curvature of each new curve with each reference curve and list the curves (or sections of curves) which match.

2. *Orientation:* from the list of matching curves determine which sets of curves have the correct orientation relative to each other to match the relative orientation of the equivalent curves in the reference set; make a list of these sets.

3. *Position:* from the list of sets determine which subsets of curves have the correct position relative to each other to match the relative position of the equivalent curves in the reference set; make a list of these subsets.

The final subsets of curves represent objects which match the reference object or, more likely, part of the reference object.

To make the problem of finding and defining curves more manageable initially, I have simplified the fish images to silhouettes. However, as fish patterns, especially prominent stripes, are very useful in fish identification, I will revert to testing the method on complete gray-scale fish images in section 6.9.

6.2 Understanding the Curvature

In order to help the reader to relate the signed curvature of the curves to their geometry the curvature has been calculated for some simple parametric cubic polynomial functions shown in Figure 6.2.1; their curvature values are shown in Figure 6.2.2. Where the curves are mirror images of each other, curvature values remain the same but the sign is reversed. Figure 6.2.2 (b) and (c) demonstrates that when the start point and end points are reversed, the series of curvature values are reversed

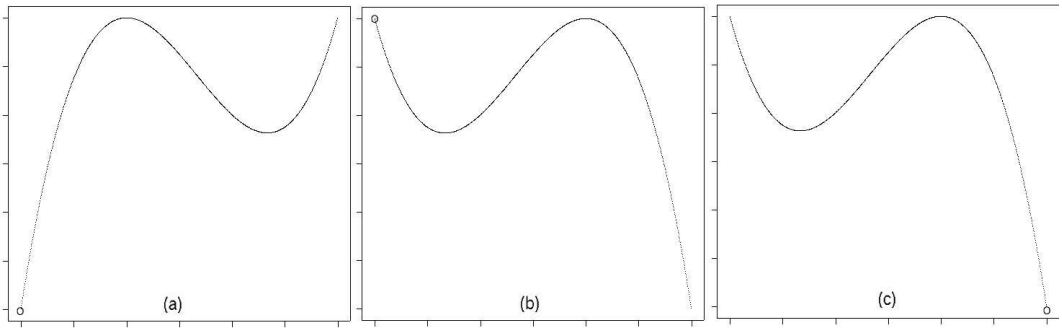


Figure 6.2.1: Three parametric cubic polynomials (a) $x = t$, $y = t^3 - 2t^2$, (b) $x = t$, $y = -t^3 - 2t^2$, (c) $x = -t$, $y = t^3 - 2t^2$. (b) is the mirror image of (a) reflected in a vertical line. (c) is the same as (b) but is traversed in the opposite direction (the open circle indicates the start of the curve).

and the sign is reversed. Figure 6.2.2 (a) is a mirror image of (b) but the direction is reversed so curvature values are simply reversed without change of sign.

A set of curves found for the silhouette of a kingfish are shown in Figure 6.2.3; their signed curvature values are plotted in Figure 6.2.4. The latter figure illustrates the variation in signed curvature values for a set of curves for one fish. It demonstrates the symmetry of the curve at the end of the tail (curve a), and the mirror imaging of the curves on either side of the tail (curves b and g). The closeness of the mirror image can be seen in Figure 6.2.5(a), where curve g is plotted with reverse sign.

As discussed in section 5.3, in the previous chapter, the curve finding algorithm may find several possible fits to each curve because pixels used to define one curve can be re-used to define another curve. Figure 6.2.3(c) showed only one example for each curve that was manually chosen for the reference set. As an example, the variation in the signed curvature values for the end of the tail (curve a) for the nine different curves which were found by the algorithm are shown in Figure 6.2.5(b). These nine curves are not easily distinguishable by eye at the resolution of the computer screen. The aim of this figure is to illustrate the amount of variation in curvature that might be expected for apparently identical curves.

6.3 Comparing Signed Curvature for a Pair of Curves

To compare a new curve to a reference curve, each signed curvature value in the new curve is compared to the corresponding signed curvature value in the reference curve in order (Figure 6.3.1). Only part of the new curve may match the reference curve or vice versa. For example, when part of an object is occluded we may only be able to detect part of the curve; or a curve may represent two separate but collinear objects which appeared to the algorithm to be one curve. To allow for these possibilities curvature comparison is applied to all possible overlaps of parts of the two curves as shown in Figure 6.3.2. In addition, the direction in which the

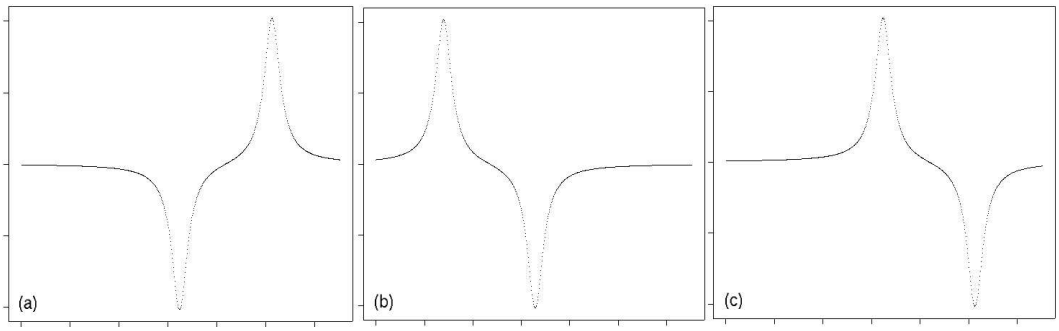


Figure 6.2.2: Curvature of the 3 parametric cubic polynomials in the previous figure. The curvature values for (b) are the reverse of the values for (a). The curvature values for (b) are the reverse of the curvature values for (c) and the sign is also reversed.

curves are traversed may affect the list order of the curvature values and their sign, so a new curve needs to be compared to both the original reference curve and the reversed reference curve.

Initially, two measures were used for comparison of signed curvature values:

1. the sum of the squared differences between corresponding values on the two curves; the average of the squared values (*assqd*) is used so that the length of the curve does not affect the final value;

2. the maximum difference between any corresponding pair of values (*md*).

The smallest values for *assqd* and *md* indicate the best curve matches. However, these comparison values did not correspond well with what a person might consider to be the best match for the curves, and since we are attempting to imitate a person's response to the images it is desirable to modify the algorithm appropriately. Two modifications were made to the comparison measures: (1) to increase the length of matching curves found, and (2) to allow larger variations near curve ends, these are described below.

It is desirable to have the largest possible overlap of matching curves. In order to bias the algorithm in favour of a larger overlap, the comparison value was divided by the square of the length of overlap, so that larger overlaps gave smaller comparison values.

When considering the shape of a curve viewed from a human perspective, differences in the curvature near the middle of the curve affect its overall appearance to a much greater degree than differences in curvature near the end of a curve, see for example Figure 6.3.3. Therefore the comparison algorithm has been modified to reduce the effects of differences which are near the ends of curves. The equation for the new modified *assqd*, called *mssqd*, is:

$$mssqd = \frac{\sum_{i=1}^{N_c} ((C1_i - C2_i)^2 \times \alpha)}{N_c^3} \times \beta, \tag{6.3.1}$$

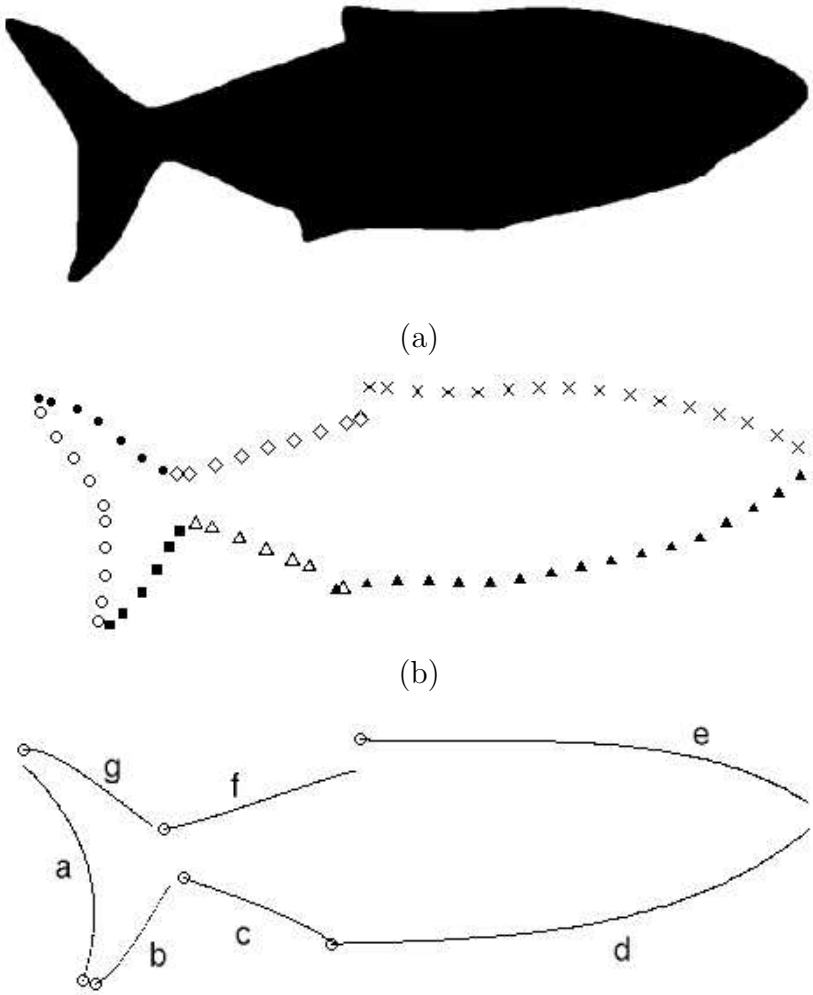


Figure 6.2.3: (a) Silhouette of a kingfish. (b) The symbols show the location of the means of seven chains of ellipses calculated for the edges of the silhouette. (c) Seven parametric cubic polynomials calculated for the points shown in (b), open circles indicate the start of each curve.

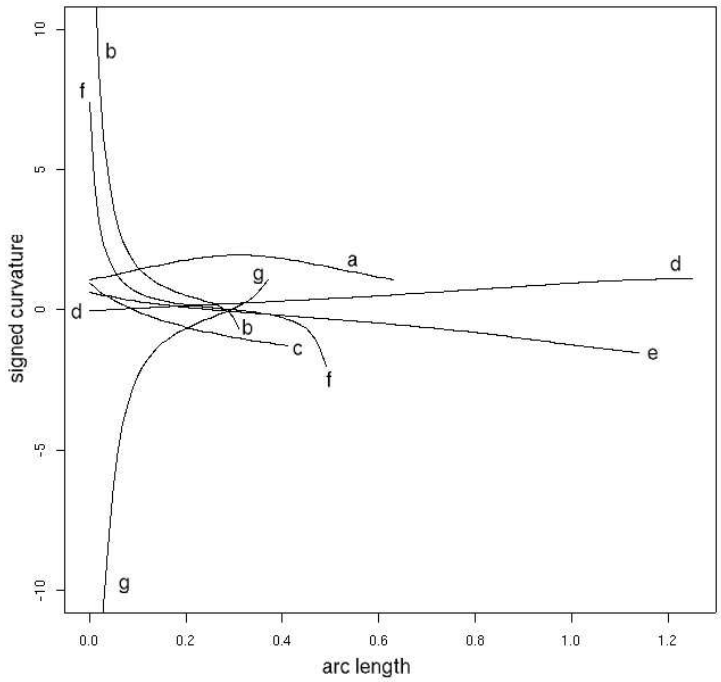
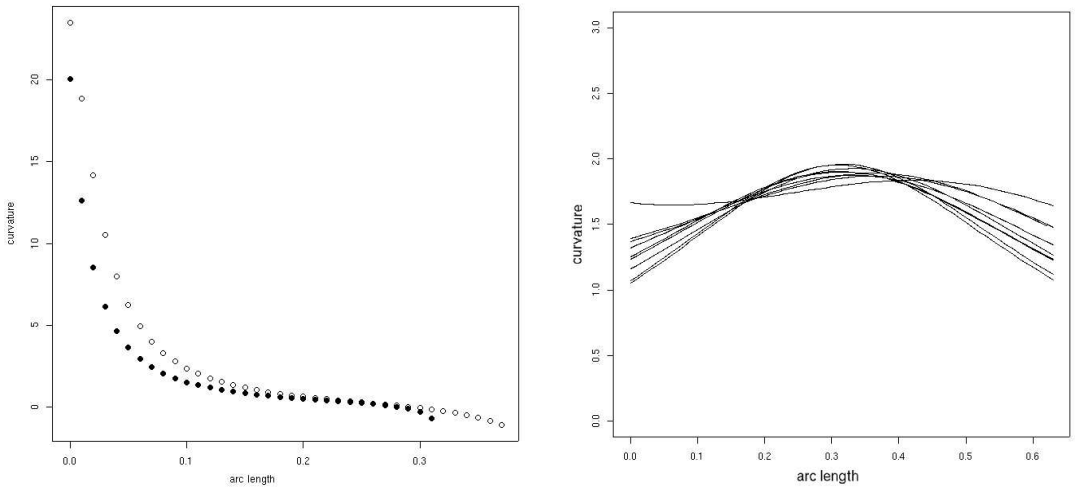


Figure 6.2.4: Signed curvature signature of the 7 parametric cubic polynomials defining the kingfish silhouette shown in Figure 6.2.3.



(a) (b)

Figure 6.2.5: (a) Curvature values for the two sides of the tail of the fish silhouette are plotted for comparison. Curve b (filled circle) is plotted as actual values and curve g (open circle) is plotted with sign reversed to show the similarity of the absolute curvature values.(b) Signed curvature calculated for 9 similar curves fitted to the tail of the fish silhouette (curve a) by the curve finding algorithm.

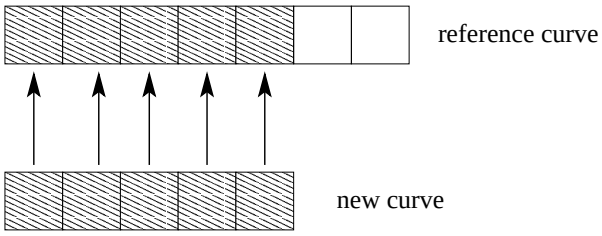


Figure 6.3.1: Each square box represents one signed curvature measurement for a curve, the measurements for the pair of curves are compared in order over the length of the shortest curve.

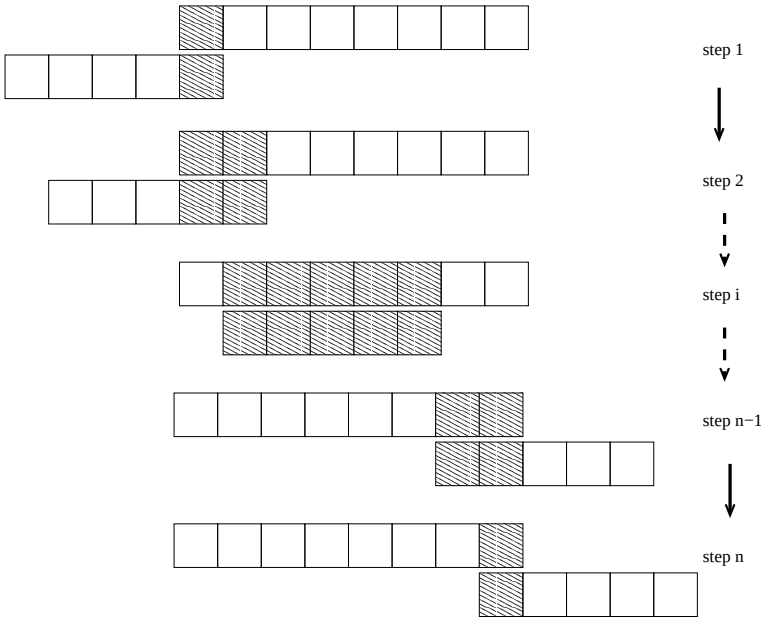


Figure 6.3.2: The curve matching algorithm is applied over all possible overlaps of the two curves as indicated on the diagram. The series of signed curvature values of each curve is represented by a series of boxes. At each step, only the shaded boxes are compared. The total number of steps, $n = length(curve1) + length(curve2) - 1$.

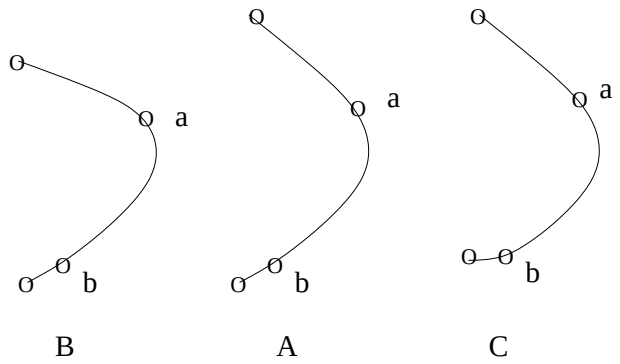


Figure 6.3.3: The visual effects of modifying curvature values depends on where the modification occurs. The curvature of A has been modified a small amount at point a (near the middle of the curve) to produce B, this curve looks quite different from A. By comparison, C has undergone a large modification of curvature at point b (near the end of the curve) but results in a curve which looks less different.

where $C1_i$ and $C2_i$ are the curvature values at i for the two curves under comparison; N_c is the total number of curvature values in the series of curvature values being compared; α is the modification factor; and β is a constant designed to increase the size of the resultant value of *mssqd* simply to make the values more readable, here $\beta = 1000$.

The modification factor is given by:

$$\alpha = \sqrt{1 - \left(\frac{i-m}{m}\right)^2}$$

where $m = (N_c + 1)/2$. The modification factor is of the form shown in Figure 6.3.4. The value 1 is added to N_c in the equation for m, i.e. $(N_c + 1)/2$, so that the modification does not cause the comparison value to become zero (i.e. indicating a perfect match). *mssqd* is divided by N_c three times: once to get the average and twice to strongly bias the algorithm in favour of larger overlaps.

The other comparison measure, maximum difference, is also modified by α and multiplied by β to produce a new measure, *mmd*, modified maximum difference:

$$mmd = \max \left| \frac{(C1_i - C2_i) \times \alpha}{N_c^2} \right| \times \beta. \tag{6.3.2}$$

A comparison between the results from *assqd* and *mssqd* is shown if Figure 6.3.5.

6.4 Relative Orientation and Relative Position

Once a list of matching curves has been extracted from the curves of the new image, it is then necessary to find sets of these curves which have the same orientation and position relative to each other as the equivalent curves in the reference set.

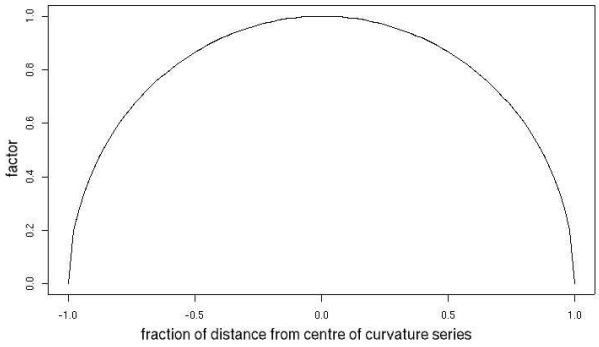
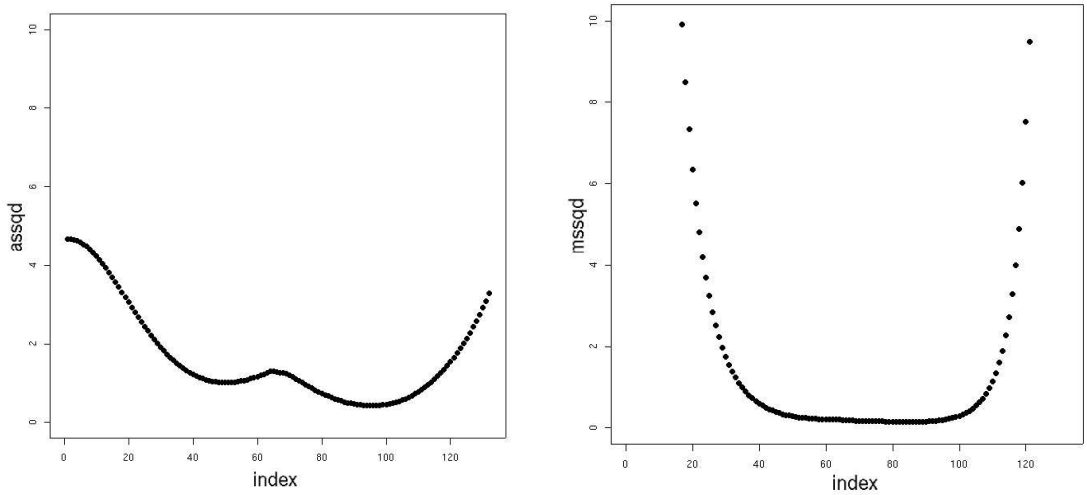


Figure 6.3.4: Plot of modification factor for reducing comparison differences near ends of curves.



(a) (b)

Figure 6.3.5: Comparison between (a) *assqd* and (b) *mssqd*. The effect of the modified version, *mssqd*, is (1) to bias the comparison against matches for small curvature overlaps by greatly increasing the comparison value and (2) to reduce the effects caused by increased curvature near the ends of curves which removes the local maximum in the centre of curve (a). The value of the index on the x-axis refers to the step number illustrated in Figure 6.3.2.

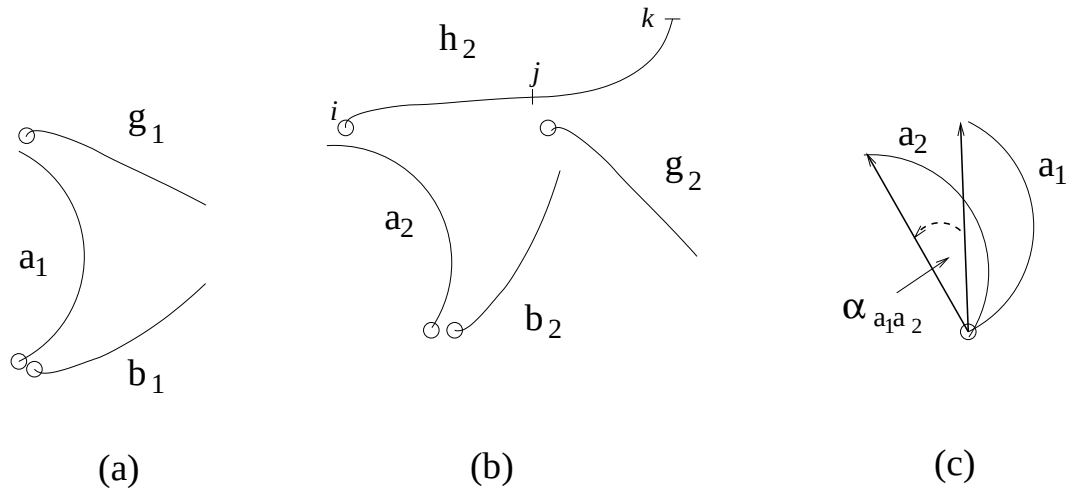


Figure 6.4.1: Figure shows the method of checking the relative orientation of matched curves on an idealised fish tail: (a) reference set of curves for tail object; (b) new set of 4 curves to compare; (c) angle α_{a1a2} between two curves is the angle between the two lines which join the start and end points of the curves a_1 and a_2.

A hypothetical example, shown in Figure 6.4.1, is used to illustrate how the relative orientation of the curves is compared. Orientation of a curve is taken as the vector which goes from the start point of the curve to the end point of the curve. To check if any of the parts of the new curves in Figure 6.4.1 may belong to the same type of object as the reference object:

- first find all matching curves: in the example the following curves appear to have matching curvature:
  1. $a_1 \approx a_2$
  2. $b_1 \approx b_2$
  3. $g_1 \approx h_{2[i-j]}$ (partial curve match)
  4. $g_1 \approx g_2$
- then check that the relative orientation of pairs of curves (or parts of pairs of curves) is consistent, for the example:
  1. $\alpha_{a1a2} \approx \alpha_{b1b2}$
  2. $\alpha_{a1a2} \approx \alpha_{g1h2[i-j]}$
  3. $\alpha_{a1a2} \not\approx \alpha_{g1g2}$.

The results indicate that a_2, b_2 and $h_{2[i-j]}$ may belong to the same type of object as a_1, b_1 and g_1, but g_2 does not.

A slightly modified example, shown in Figure 6.4.2, is used to illustrate how relative position is checked. P is the vector from the start of one curve to the start of another curve from the same image.

- First the curves which appear to match in curvature and orientation are identified. In the example, these are:
  1. $a_1 \approx a_2$
  2. $b_1 \approx b_2$
  3. $g_1 \approx g_2$
- Next, the relative position of curves (or parts of curves) in the reference object is compared to the relative position of matching curves (or parts of curves) from the new image:
  1. $P_{a1b1} \approx P_{a2b2}$
  2. $P_{a1g1} \not\approx P_{a2g2}$

In the example, it is clear that curves a_2 and b_2 belong to the same type of object as the reference object. However, although the curvature of g_2 is a good match for g_1 and is in the correct relative orientation to other curves, it is not in the correct position relative to curve a_2; therefore curves a_2 and g_2 do not belong to a single object of the same type as the reference object.

The relative position between two pairs of curves is calculated as follows. Given that there are a pair of curves in the reference set, a_r and b_r, which match in curvature and orientation a pair of curves in the new set, a_n and b_n. The two vectors which join the starting points in each of the pairs of curves (or the beginning of the matching portion of the pairs of curves) are calculated to give $P_{a_rb_r}$ and $P_{a_nb_n}$. Because the object in the new image may be rotated relative to the reference object, each vector, $P_{a_rb_r}$ and $P_{a_nb_n}$, is rotated the amount required to bring the vector from the start point to the end point of the first curve in the pair (a_r and a_n, respectively) parallel to the x-axis, giving $rP_{a_rb_r}$ and $rP_{a_nb_n}$. Then the distance between the two vectors is calculated: $d_{ab} = |rP_{a_rb_r} - rP_{a_nb_n}|$. To be accepted, the distance must lie within a predefined limit. This limit has been made to depend on the length of $P_{a_rb_r}$ to allow more leeway for error on curves that are further apart. The limit is of the form:

$$de = a + \frac{|P_{a_rb_r}|}{b} \tag{6.4.1}$$

where a is the maximum error allowed when $|P_{a_rb_r}| = 0$, and $1/b$ is the fraction of the length of $P_{a_rb_r}$ allowed as error.

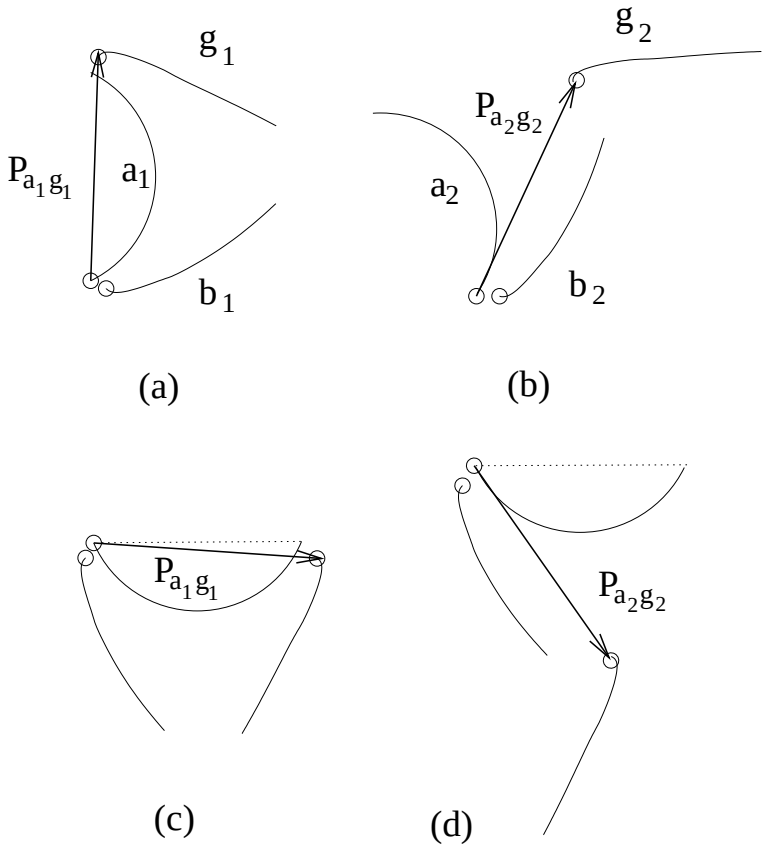


Figure 6.4.2: Figure shows the method of checking the relative position of matched curves on an idealised fish tail: (a) reference set of curves for tail object; (b) new set of 3 curves to compare. P_{a1g1} is the vector from the start of curve a_1 to the start of curve g_1. (c) and (d) show rotated versions of (a) and (b), respectively. Each set of curves is rotated until the vector joining the start and end points of the first curve (a_1) lies parallel to the x-axis (shown as a dotted line). Then the vectors which join the start points of curves in the reference set can be directly compared to equivalent vectors in the new set. In this example, P_{a1g1} should be similar in length and orientation to P_{a2g2} if the two curves a_2 and g_2 belong to the same type of object as the reference curves $a1$ and $g1$; this is clearly not the case.

6.5 Comparison of Curves from Fish Silhouettes

Four more fish silhouettes have been processed to extract edges and fit curves to the edges (i.e. as parametric cubic polynomials). The silhouettes and the longer curves are shown in Figure 6.5.1. Very short curves which may form on the fins and nose of the fish have been omitted. Signed curvature has been calculated for the new curves and the curves are compared with the reference set. The reference fish silhouette and its curves are those depicted in Figure 6.2.3.

Cut-off values for similar curves were determined experimentally by choosing *mssqd* or *mmd* values for curves which looked similar in shape. It was found that the two curve matching criteria *mssqd* and *mmd* gave very similar results for matching of curves and so just the *mssqd* criteria was selected for use in future. A fairly lenient threshold value of $mssqd = 0.4$ was chosen as curve shape may vary considerably between fish.

For any pair of curves there may be many variations of overlap between the curves for which *mssqd* is below the threshold, in this case the section of the curve which gives the maximum length of match but is still below the threshold is chosen. If the threshold used is quite generous, this may not result in the best match being chosen, as the best match is a balance between similarity of curvature and length of match but how the best compromise might be chosen is unclear. Ideally all matches which are below the threshold should be considered, but this would greatly increase the size of the computation.

Initially the thresholds for angle and position variation were chosen by examining the images and deciding what a human being might allow as reasonable variation. These thresholds were experimented with and were found to work reasonably well. Small changes to these initial estimates did not appear to improve the performance of the curve-matching algorithm. The values chosen by other users of the algorithm may vary depending on the degree of rigidity of objects being studied. The smallest thresholds that give good matches are best as then the algorithm is less likely to find good matches for objects which appear to the human observer to be quite different.

The upper limit chosen here for allowed difference in relative angle between curves is 0.14 radians (approx. 8°). The limit used for allowed difference in position was: $de_{ij} = 0.08 + d_{ij}/10$ where 0.08 is the maximum variation allowed for two curves which start at the same point (i.e. where the distance d_{ij} between the start points of two reference curves is zero) and 1/10 gives the fraction of the curve allowed as an additional difference in distance between the two start points.

The curves of the reference fish silhouette were compared with fish curves from 4 other images in order to test the curvature comparison algorithm, these fish silhouettes are shown in Figure 6.5.1. The results of the comparison with fish (iv) in Figure 6.5.1 will be discussed in detail below. Results of comparisons with the other three fish will be given in a summarised form in the section following a discussion on measuring curve match discrepancy.

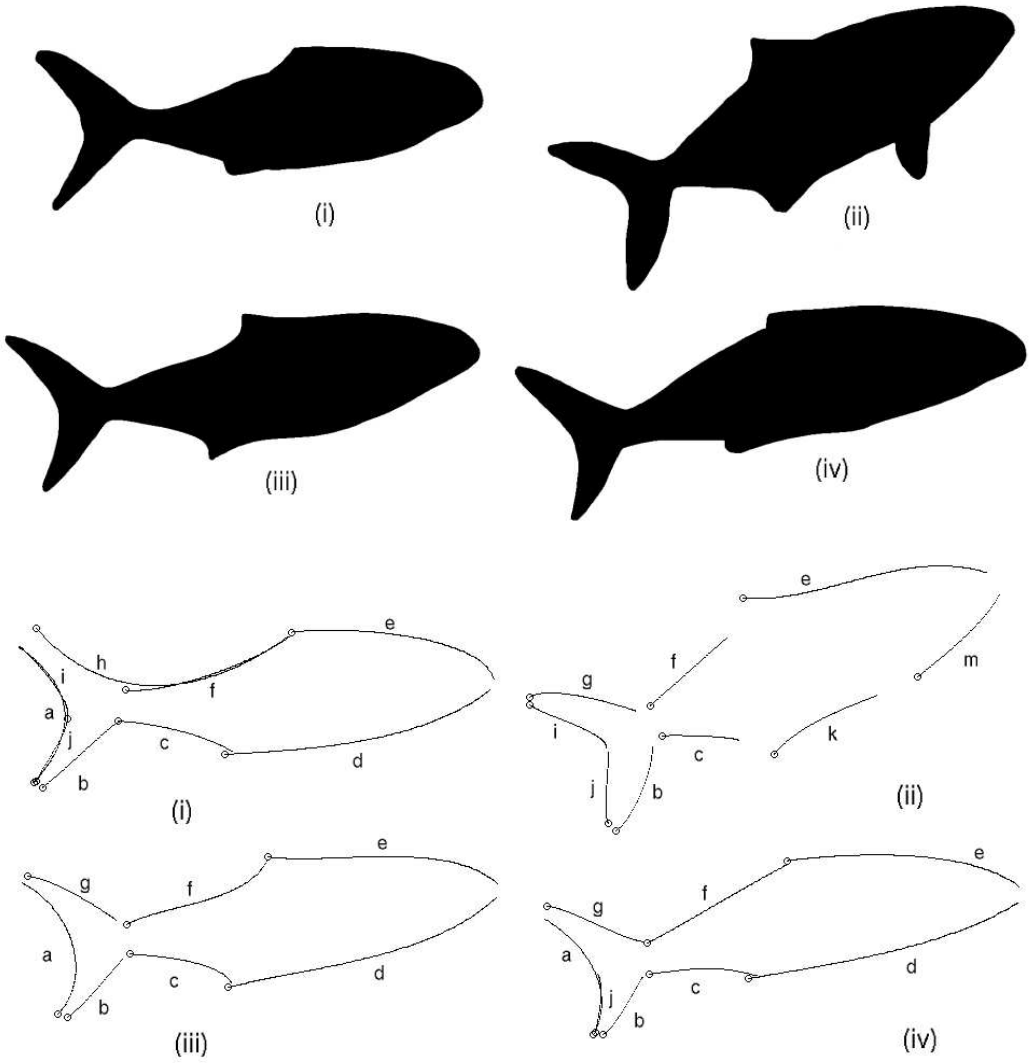


Figure 6.5.1: Parametric cubic polynomial curves have been calculated for 4 kingfish silhouettes. The curves have been labelled in the same order as Figure 6.2.3. In addition, curve a may be split into curves i and j, curve d may be split into curves k and m, and curve h may overlap curves f and g.

Comparison with Silhouette of Fish (iv). Table 6.5.1 shows the results of step 1 of the algorithm, i.e. the list of pairs of curves which matched. The first column gives the *mssqd* value for the longest match found for each pair of curves whose *mssqd* value did not exceed the threshold value. The columns headed *ref. curve* and *new curve* indicate which two curves were matched from the reference fish silhouette and the new image, respectively (see Figure 6.2.3(c) and fish (iv) in Figure 6.5.1(c) for location of labelled curves). For example, curve g on the reference fish silhouette (the upper side of the tail) matched the shape of curves d and g of fish (iv) (the ventral curve and the upper side of the tail, respectively). The table indicates that only a small fraction of d matched reference curve g. A fraction of 0.19 of the length of curve d matched a fraction of 0.61 of the length of curve g. The table also indicates what the start and end curvature indices[1] were of the section which matched (i.e. columns headed: *ref. start* and *ref. end* for the reference curve, and *new start* and *new end* for the new curve).

The reference curves must be compared in both directions, so a particular pair of curves may show two matches if they matched in both directions. In Table 6.5.1 a double line separates matching curve pairs for different reference curves and a single line separates matching curve pairs for the same reference curve matched in different directions. The direction of the match can be determined from the start and end indices, if the start index is larger than the end index of the reference curve then the curve matched in reverse. For example, reference curve b matched new curve f in both directions. In its normal orientation the reference curve matched f from indices 1 to 22, and in its reverse direction it matched a different section of f, from indices 50 to 69. The *fraction of curve* column indicates that in both cases about 2/3 of the length of reference curve b matched about 1/3 of new curve f.

The results of the second step of the comparison, i.e. the relative orientation, are given in Table 6.5.2. This table shows which sets of pairs of matching curves had similar relative orientations in the new set compared to the reference set. For example, in set 1, the angle between reference curve f and new curve d is similar to the angle between reference curve e and new curve e. This set would be expected to be eliminated in the next step of the algorithm as the relative positions of reference curves f and e are very different to the relative positions of new curves d and e. If more than one set of curves is found, then any of these sets which are subsets of the other sets are removed before the next step, any repetitions of curve pairs seen in the table would be due to matches which occurred both for the normal and for the reverse directions of the reference curve.

In the final step, i.e. the relative position, subsets of the sets in the previous step are found in which the relative positions of new curves are similar to the relative positions of corresponding reference curves. Any two subsets which share pairs of matching curves are joined by taking the union of the two subsets.

[1]The curvature values are stored as a list, the curvature indices sequentially number each curvature measurement starting from 1. The arc length can be calculated by multiplying the index by the parameter interval length, i.e. 0.01

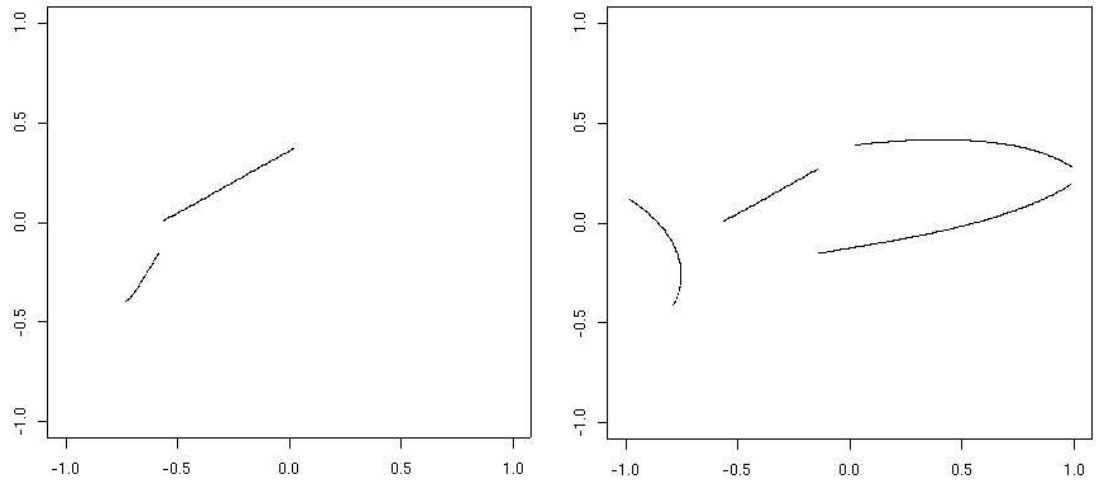


Figure 6.5.2: Plots of curve sections from fish silhouette (iv) which matched curves from the reference set. Two subsets of matching curves were found by the algorithm, shown in (a) and (b). The second subset is a much more promising candidate for a fish silhouette than the first subset.

In the example given here, only two subsets of curves were within the threshold limit, see Table 6.5.3. The matching sections of the new curves are plotted in Figure 6.5.2. Of these two subsets, subset 2 included a much larger fraction of the total length of the reference set than the other, with fractions of 0.71 and 0.21, respectively. In subset 2, 4 out of 7 curves matched along most of their length, a strong indication that the new set of curves does represent a fish silhouette similar to the reference fish. Subset 1, with only 0.21 of the reference curves matched, is highly unlikely to represent a fish silhouette, it can be rejected based on the small fraction of matching curves.

6.6 Discrepancy of Matched Curves

The current algorithm uses threshold values to decide whether a curve should be accepted as similar to a reference curve. Threshold values exist for:

- maximum allowed value for *mssqd*;
- minimum amount of matching overlap between reference and new curve measured as number of curvature measurements (n), n = path length/s, where s is the distance (measured as arc length) between curvature measurements;
- the maximum deviation angle from α allowed within the set where α is the average angle of rotation between a matching set of reference curves and a set of new curves;
- the maximum allowed variation in relative position of two curves in the new image compared to a reference pair of curves.

Table 6.5.1: Results of comparison of signed curvature values between curves of the reference fish silhouette and curves of fish (iv). Each row of the table lists the results for one pair of curves whose *mssqd* did not exceed the threshold value. See text for explanation.

	Reference Curve:				New Curve:			
mssqd	curve label	fraction of curve	start index	end index	curve label	fraction of curve	start index	end index
0.399	b	0.41	20	32	c	0.29	1	13
0.359	b	0.69	11	32	f	0.32	1	22
0.375	b	0.62	32	13	f	0.29	50	69
0.135	a	0.97	3	64	a	1	1	62
0.258	a	1	1	64	d	0.54	56	119
0.236	a	0.7	64	20	c	1	1	45
0.165	a	1	64	1	e	0.64	37	100
0.002	d	0.94	1	119	d	1	1	119
0.026	d	0.55	1	69	f	1	1	69
0.149	d	0.79	1	100	e	1	1	100
0.041	d	0.94	119	1	d	1	1	119
0.169	d	0.36	126	82	c	1	1	45
0.027	d	0.55	69	1	f	1	1	69
0.044	d	0.79	126	27	e	1	1	100
0.398	c	0.5	1	21	b	0.68	11	31
0.315	c	1	1	42	d	0.35	1	42
0.111	c	1	1	42	c	0.93	1	42
0.18	c	1	1	42	f	0.61	28	69
0.074	c	1	1	42	e	0.42	37	78
0.374	c	0.64	42	16	b	0.87	5	31
0.185	c	1	42	1	d	0.35	51	92
0.399	c	0.55	23	1	c	0.51	1	23
0.164	c	1	42	1	f	0.61	1	42
0.092	f	0.58	1	29	b	0.94	3	31
0.35	f	1	1	50	d	0.42	24	73
0.395	f	0.64	19	50	c	0.71	1	32
0.266	f	1	1	50	f	0.72	1	50
0.279	f	0.98	2	50	e	0.49	1	49
0.251	f	0.52	50	25	b	0.84	6	31
0.292	f	0.02	1	1	j	0.04	1	1
0.391	f	1	50	1	d	0.42	1	50
0.368	f	0.9	47	3	c	1	1	45
0.288	f	1	50	1	f	0.72	20	69
0.265	f	1	50	1	e	0.5	30	79
0.382	g	0.61	16	38	d	0.19	1	23
0.226	g	0.87	38	6	g	0.77	11	43
0.309	e	0.22	1	25	b	0.81	7	31
0.067	e	1	1	115	d	0.97	1	115
0.064	e	0.39	69	113	c	1	1	45
0.005	e	0.6	1	69	f	1	1	69
0.005	e	0.87	16	115	e	1	1	100
0.395	e	0.38	115	72	a	0.71	19	62
0.032	e	1	115	1	d	0.97	1	115
0.399	e	0.39	45	1	c	1	1	45
0.005	e	0.6	69	1	f	1	1	69
0.096	e	0.87	100	1	e	1	1	100

Table 6.5.2: Results of comparison of orientation of curves for fish silhouette (iv). Each block of the table consists of a set of matched pairs of curves. The difference between the relative angle (in radians) between each pair of curves in the set is within the threshold limit. The average angle between the pairs of curves for each set is given in column 2.

set	average angle	ref. curve	new curve
1	-0.02	f	d
		e	e
2	-2.42	c	d
		f	b
3	0.925	c	f
		g	d
4	-3.105	d	d
		e	c
5	3.07	f	d
		e	e
6	0.643	c	d
		f	b
		e	f
7	-0.277	d	e
		f	e
		f	c
8	-2.97	d	d
		f	f
		g	g
9	0.287	c	e
		f	f
		e	c
10	0.112	a	a
		d	d
		f	f
		e	e

set	average angle	ref. curve	new curve
11	1.083	a	c
		a	e
		c	f
		e	b
12	-2.648	d	f
		c	c
		e	f
		e	d
13	2.706	b	f
		d	c
		d	e
		f	e
		f	c
14	0.394	d	f
		c	c
		c	e
		e	d
		e	c
15	0.51	d	f
		c	d
		c	c
		e	f
		e	d

Table 6.5.3: Results of comparison of position of curves for fish silhouette (iv). The following two subsets of curves matched and had similar relative orientation and relative position. Subsets 1 and 2 were formed from sets 6 and 10 of the previous table. *curve section* refers to the range of curvature indices for which the match occurred, and *fraction* refers to the fraction of the length of the total reference set of curves which matched the new set of curves. These are the two subsets of curves illustrated in Figure 6.5.2.

subset number	ref. curve	curve section	new curve	curve section	fraction
1	f	[1-29]	b	[3-31]	
	e	[1-69]	f	[1-69]	0.21
2	a	[3-64]	a	[1-62]	
	d	[1-119]	d	[1-119]	
	f	[1-50]	f	[1-50]	
	e	[16-115]	e	[1-100]	0.71

Thresholds give a binary answer, either the curve does match or it does not. It would be preferable to have a method for measuring how much the final subset of curves differed from the reference set. The use of thresholds would still be desirable otherwise the number of potential curve matches to consider would be enormous as it would include all possible combinations of reference curves and new curves, and all possible sections of each curve. A method of measuring discrepancy is proposed below.

Discrepancy, D, between a subset of curves from the reference set and a matching set of curves from the new image is equal to the square root of the weighted sum of squares of the discrepancies for the curve match, the relative orientation and the relative position, and is given by:

$$D^2 = (w_{mssq} mssq)^2 + (w_\alpha D_\alpha)^2 + (w_p D_p)^2 \tag{6.6.1}$$

The discrepancy for relative orientation is calculated for a set as a whole and is given by:

$$D_\alpha = \frac{1}{n} \sum (\alpha - \bar{\alpha})^2$$

where α is the angle between the reference curve and the new curve, $\bar{\alpha}$ is the mean α of the set; n is the number of curves in the set. The discrepancy for relative position is also calculated for a subset as a whole and is given by:

$$D_p = \frac{1}{n-1} \sum_{i=2}^{n} d_{a_1 a_i}$$

where d_{ab} is the distance between two vectors: $P_{a_r b_r}$ and $P_{a_n b_n}$, where $P_{a_r b_r}$ is the vector which joins the start point of two curves in the reference set and $P_{a_n b_n}$ is the

vector which joins the start point of two curves in the new image; n is the number of curves in the subset. The first curve in the subset (a_1) is not counted as it is the one against which the position of all others are measured.

The discrepancy for any set of curves with a perfect match will be zero. The weights adjust the discrepancies so that any set of curves which is close to the threshold at any step will have a discrepancy, D_i, close to 1. The weights w_i are given by:

$$w_{mssq} = \frac{1}{thr_{mssq}}$$

$$w_\alpha = \frac{1}{(thr_\alpha)^2}$$

$$w_p = \frac{1}{thr_p}$$

where thr_{mssq} is the threshold for $mssq$, thr_α is the threshold for relative angle, and thr_p is the threshold for relative position.

The discrepancy values calculated for the two subsets of matching curves found for fish (iv) are 0.685 and 0.426, for subsets 1 and 2, respectively, indicating that the second, larger subset of curves is the better match for the reference set.

6.7 Recognition of Three More Fish Silhouettes

The other three fish silhouettes from Figure 6.5.1 have been compared to the reference fish silhouette (i.e. Figure 6.2.3) and the results are summarised below.

Four subsets of matching curves were found for fish (i), see Table 6.7.1 and Figure 6.7.1. A high fraction of matching curves and a relatively low discrepancy value for the first subset make this a strong candidate for a fish silhouette similar to the reference fish. The other subsets of curves had rather low fractions of matching curves. The discrepancy value for set 3 was good and Figure 6.7.1(c) shows that there definitely appears to be a strong similarity between the two curves in this subset and curves f and e of the reference set. But with only 25% of the reference fish silhouette found this subset would not be considered a good candidate for a fish silhouette.

Only one subset of curves was found to match fish (ii), see Table 6.7.2 and Figure 6.7.2(a). These two curves accounted for only 15% of the reference fish silhouette and the discrepancy value is quite high indicating that this fish is a very poor match for the reference fish silhouette. In the Discussion section below, the problem of poor matches for fish in unfavourable poses, such as fish (ii), will be addressed.

One subset of curves was found to match fish (iii), see Table 6.7.3 and Figure 6.7.2(b). A high fraction of the fish matched the reference fish silhouette (0.77) and the discrepancy value of 0.273 is low. These results indicate that fish (iii) is a good match for the reference fish silhouette.

Table 6.7.1: The following four subsets of curves from fish (i) matched and had similar relative orientation and relative position to the reference curves. A high fraction of matching curves and relatively low average discrepancy value for the first subset make this a strong candidate for a fish silhouette similar to the reference fish.

subset no.	ref.	section	new	section	fraction	discrepancy
1	a	[1-64]	a	[7-70]		
	d	[3-118]	d	[1-116]		
	c	[1-42]	c	[4-45]		
	f	[1-50]	f	[2-51]		
	e	[24-115]	e	[1-92]	0.78	0.407
2	a	[1-64]	h	[14-77]		
	c	[42-1]	b	[2-43]	0.23	2.035
3	f	[7-50]	b	[1-44]		
	e	[1-74]	f	[1-74]	0.25	0.635
4	d	[119-1]	h	[1-119]		
	e	[50-1]	c	[1-50]	0.36	2.966

Table 6.7.2: The subset of curves below is from fish (ii). The curves matched the reference curves and had similar relative orientation and relative position to the reference curves.The fraction of matching curves was very low and the discrepancy value was high making this fish a poor match for the reference fish silhouette.

subset no.	ref.	section	new	section	fraction	discrepancy
1	c	[12-42]	c	[1-31]		
	g	[2-38]	g	[1-37]	0.15	0.958

The decision on whether a subset of curves does represent a fish silhouette depends largely on what fraction of the curves matched. Comparing the results with the matched curves plotted in Figures 6.5.2, 6.7.1 and 6.7.2 indicates that a match of greater than 0.7 of the reference curves should be acceptable as being a fish silhouette. The lowermost boundary for acceptability would have to be determined by testing the method against a larger data set which comprised both fish and non-fish objects. The discrepancy value may be useful for making a decision in borderline cases. The preliminary experiments conducted here suggest that a good match would have a discrepancy value below 0.5.

In conclusion, three of the four fish in Figure 6.5.1 (i.e. (i), (ii) and (iv)) can be identified as matching the reference fish.

6.8 Dealing with Variations in Fish Pose

Fish silhouette (ii) in Figure 6.5.1 gave a very poor match to the reference fish silhouette despite the fact that it is the same species of fish. The difference is in

Table 6.7.3: The subset of curves below are from fish (iii), they matched the reference curves and had similar relative orientation and relative position to the reference curves. The fraction of matching curves was high and discrepancy value is low.

subset no.	ref.	section	new	section	fraction	discrepancy
1	a	[1-64]	a	[6-69]		
	d	[1-121]	d	[1-121]		
	c	[3-42]	c	[1-40]		
	g	[3-38]	g	[1-36]		
	e	[18-115]	e	[1-98]	0.77	0.471

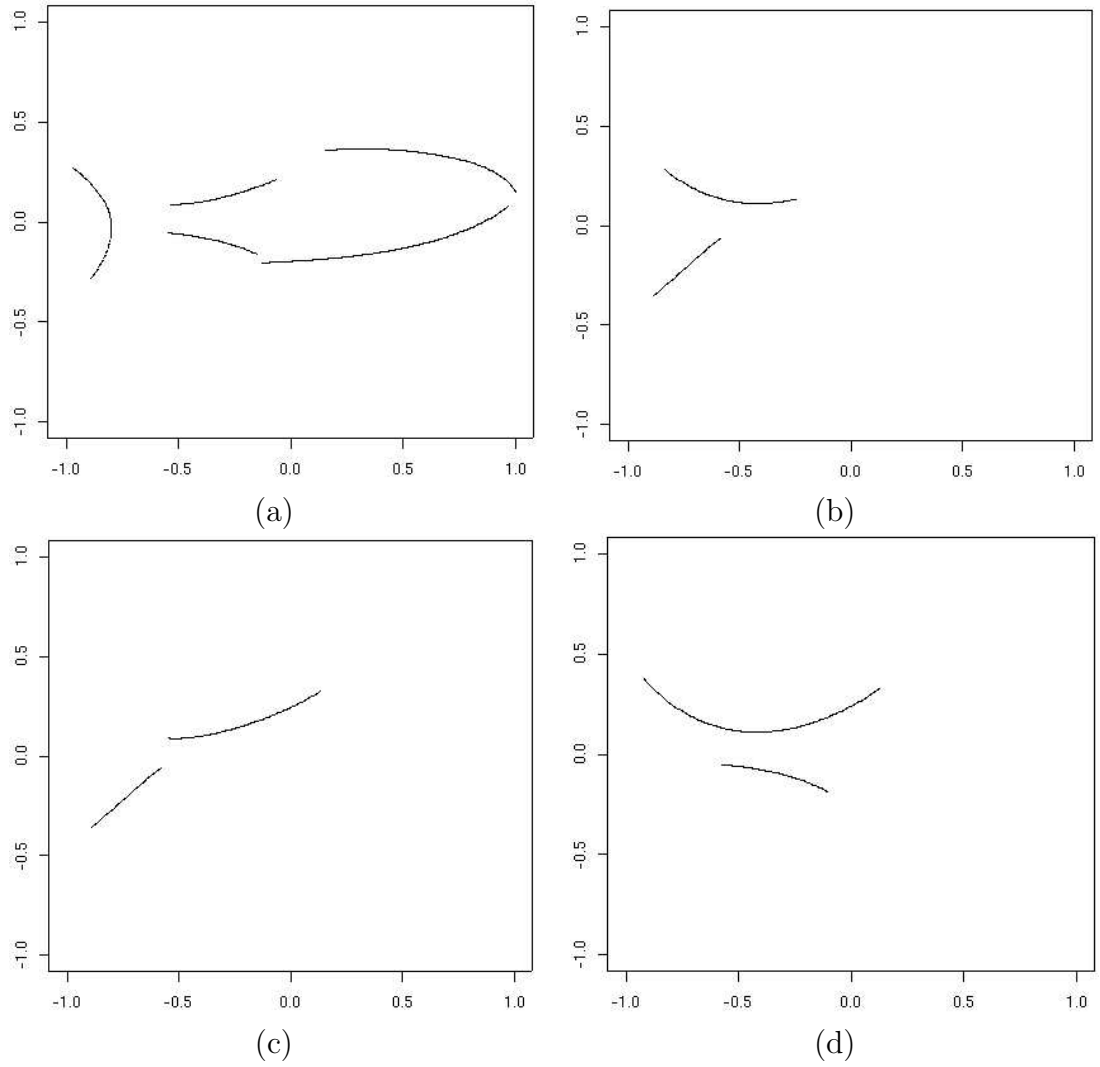
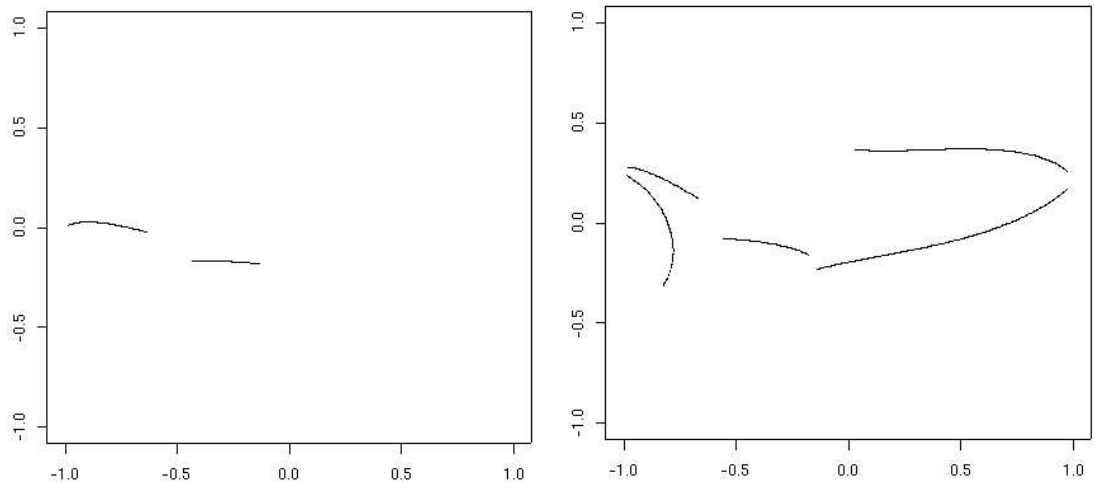


Figure 6.7.1: Four subsets of curves from fish silhouette (i) which matched curves from the reference set.



(a) (b)

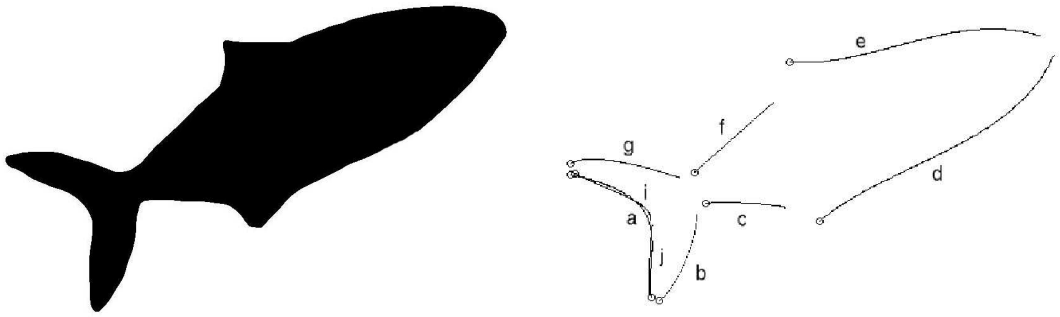
Figure 6.7.2: (a) and (b) show subsets of curves from fish silhouettes (ii) and (iii), respectively, which matched curves from the reference set.

(a) (b)

Figure 6.8.1: Gray-scale images of (a) the reference fish and (b) fish (ii).

the pose of the fish, see Figure 6.8.1 which shows the gray-scale image from which the silhouette of the fish was constructed. Because fish are not rigid objects a single fish can change its appearance to a limited degree. The main difference between the reference fish and fish (ii) is that fish (ii) has its fins extended outwards from its body and it is turning slightly towards the camera. All the other fish that have been illustrated here have their fins in a streamlined position for swimming straight forward. There are a number of methods that might be used to reduce this problem, some possible solutions are discussed below.

Increasing Thresholds. One simple way of increasing the number of matching curves when the fish shows more variation from the reference fish is to increase the thresholds. For example, when the maximum deviation angle from α was increased to 0.35 (approx. 20°) and the maximum allowed variation in relative position was increased to $0.08 + d_{ij}/5$, the fraction of curves which matched fish (ii) increased to 0.5



(a) (b)

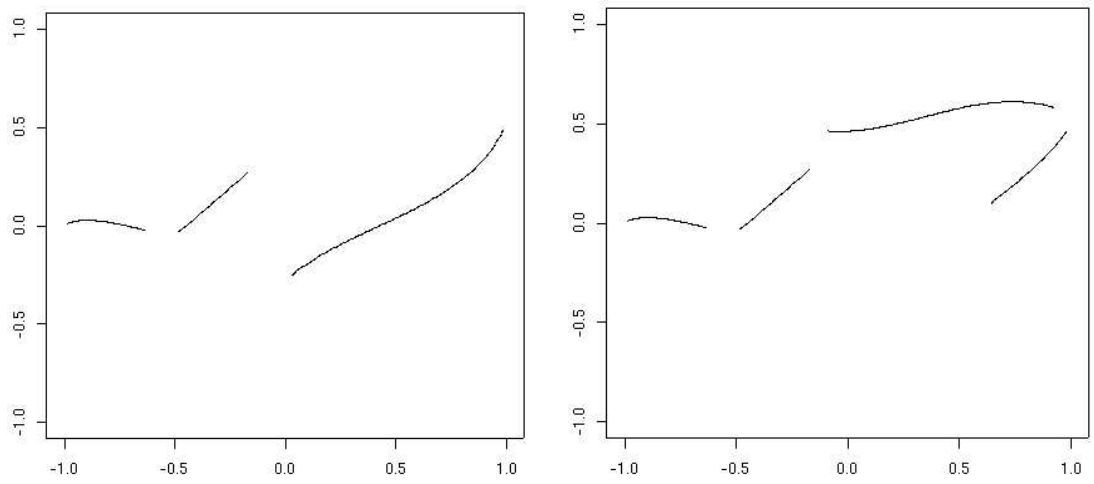
Figure 6.8.2: (a) Silhouette of fish (ii) with its pelvic fin erased. (b) Subset of curves found for modified fish (ii).

There is a problem with the idea that simply increasing the thresholds will allow all variations of fish to be found. Thresholds which are too high will also mean that non-fish objects may be found to match a reference fish, this is a highly undesirable outcome. Experimentation with a wide variety of fish and non-fish objects would be required to determine the optimal thresholds to use.

Expanding the Reference Set. Another way to solve this problem is to have more than one reference set of curves. The reference collection could include curve sets from a broad range of poses that the fish may commonly be photographed in. This may mean including reference sets from images of fish swimming towards or away from the camera. The problem with having a reference set for a range of possible attitudes is that this set might get very large and slow the computation time considerably.

Creating Sub-objects. An extra step could be added into the *UpWrite* procedure so that the curve matching algorithm first searches for sub-objects and then assembles the sub-objects into a complete object. A complex object like a fish could be divided into a number of sub-objects, such as tail, fins (dorsal, anal, pelvic, pectoral), body, head. A range of possible shapes for each sub-object, which fit the various poses of the fish, can be included in the reference set. A fish would then be defined as being composed of the correct number and type of sub-objects in the appropriate relative orientations and positions. This approach would reduce the size of the reference set as there would be no need to store all the possible combinations of shapes of sub-objects.

An experiment showed the effect of simply removing the unusual pelvic fin from fish (ii). The fin was manually erased to allow the ventral (lower body) curve of the fish to be recognised, see Figures 6.8.2 and 6.8.3. In this modified version of fish (ii) the fraction of matching curves has increased significantly and the discrepancy value has been halved, see Table 6.8.1.



(a) (b)

Figure 6.8.3: (a) Plot of curves from fish silhouette (ii) which matched curves from the reference set when the pelvic fin was erased. (b) Plot of curves from fish silhouette (ii) which matched curves from the reference set when the thresholds were increased significantly.

Table 6.8.1: The subset of curves below is from the modified version of fish silhouette (ii), in which the pelvic fin was erased (see Figure 6.8.2). These are the curves which matched the reference curves and had similar relative orientation and relative position to the reference curves. The fraction of matching curves has increased significantly and the discrepancy value has improved.

subset no.	ref.	section	new	section	fraction	discrepancy
1	d	[1-123]	d	[1-123]		
	f	[6-49]	f	[1-44]		
	g	[2-38]	g	[1-37]	0.44	0.596

Object Tracking An alternative approach is to have several examples of the new object rather than several examples of the reference object. If a video sequence, rather than still images, were captured of the new object(s) then object tracking could be applied to all moving objects in the image (e.g. Branson et al. 2003). If a single object is tracked through multiple frames then the best pose could be used for determining a match with the reference object.

6.9 Comparing Curves from Fish Images

Once the method had been thoroughly tested on the relatively simple fish silhouettes, it was tested on the original (gray-scale) images of the fish. The reference fish (Figure 6.8.1(a)) was compared to fish (iii), shown in Figure 6.9.1. The Canny edge detector was applied to the gray-scale images (as in Figure 5.2.1(d)) and curves were fitted to the edge pixels. A set of curves was extracted from the result by removing

Table 6.8.2: The subset of curves below is from fish silhouette (ii). They are the curves which matched the reference curves when higher thresholds for relative orientation angle and relative position were used. The fraction of matching curves has increased significantly and the discrepancy value has decreased (compare table 6.7.2).

subset no.	ref.	section	new	section	fraction	discrepancy
thresholds increased 1	d	[49-98]	m	[1-50]		
	f	[6-49]	f	[1-44]		
	g	[2-38]	g	[1-37]		
	e	[1-104]	e	[1-104]	0.5	0.035

very short curves or those that were nearly identical to curves already in the set. The sets of curves found for each fish are shown in Figure 6.9.2 and the matched curves have been plotted in Figure 6.9.3(a).

The largest subset of matching curves found by the algorithm is shown in Table 6.9.1. The fraction of matching curves found by the algorithm was 1.51. There is obviously a problem here as the fraction cannot be greater than one. Inspection of Table 6.9.1 shows that some of the reference curves matched with more than one of the new curves. This can occur where one of the reference curves lies within the threshold limits of more than one of the new curves. For example, curve 1 (on the lower tail of Figure 6.9.2(a)) matches with both curves 16 and 17 on the tail of fish (ii).

The opposite situation may also occur where several of the reference curves are sufficiently similar in shape, orientation and location that they all lie within the threshold limits of one reference curve. For example, this has occurred with curve 21 (the curve which defines the ventral side of fish (iii), see Figure 6.9.2(b)). According to Table 6.9.1 curve 21 matches parts of reference curves 2, 4 and 6.

To calculate the true fraction of potentially matched curves, sections of reference curves which matched more than one new curve were only counted once. This resulted in a fraction of 0.75 matched curves. The discrepancy value for the matching of all the curves in this subset was 0.503.

The degree to which curves match many other curves can be reduced by tightening up the thresholds. However this also has the effect of reducing the number of curves which match. For example, if de_{ij} is reduced to from $0.08 + d_{ij}/10$ to $0.04 + d_{ij}/14$ then the true fraction of curves which matched was only 0.5 (see Table 6.9.2). However, looking at Figure 6.9.3(b) this fraction may still be considered sufficient to identify a fish.

An alternative method for deciding which of several matches to choose from would be to choose the pair with the lowest discrepancy value. The last column in Table 6.9.2 gives the discrepancy values for each pair of curves. For example, curve 5 matches with curves 20, 22 and 25, but the lowest discrepancy value is for curve

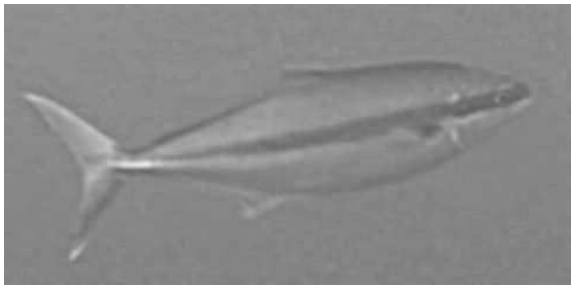


Figure 6.9.1: Gray-scale image of fish (iii).

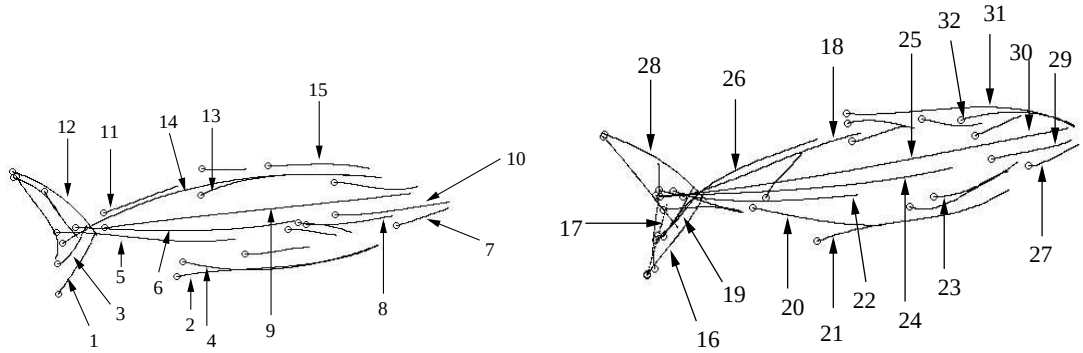


Figure 6.9.2: Sets of curves fitted to (a) the reference fish and (b) fish (iii). Numbered labels for curves refer to matched curves in Table 6.9.1

22. Looking at Figure 6.9.2, the match of curve 5 with curve 22 also appears to the human eye to be the best match of these three. For curves 6 and 11 which also have multiple matches, the lowest discrepancy values also appear to correlate with the best match by human eye (i.e. with curves 24 and 26, respectively). These results suggest that minimum discrepancy values would be a good method of choosing the best matched curves.

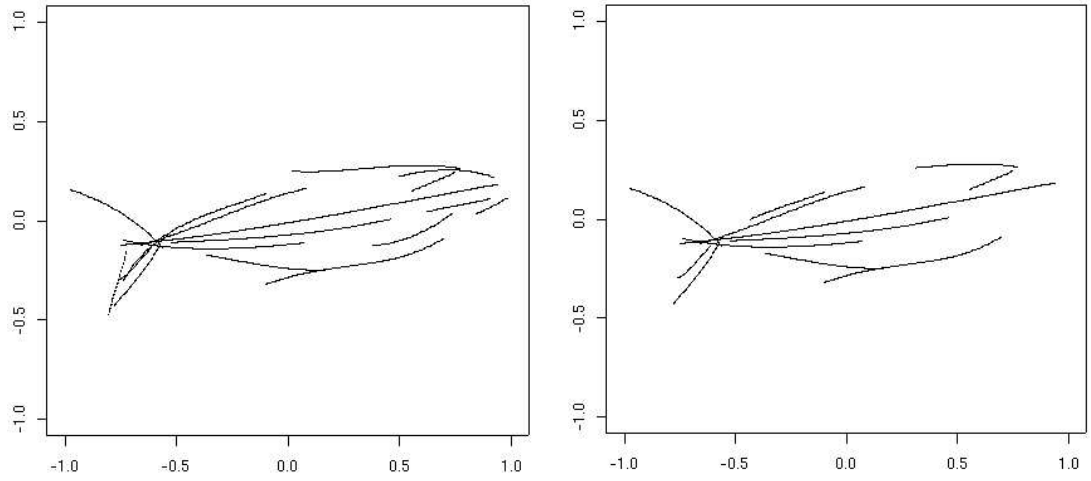
6.10 Comparing Different Species of Fish

The fish used for all the tests in this chapter so far is a kingfish. In order to test whether the curve matching method can distinguish a different species of fish, the kingfish has been compared with a pike. The pike is also a pelagic fish with streamlined body shape (Figure 6.10.1(a)), but it has quite a different tail shape and a prominent eye; it also lacks the distinctive band down the middle of the body that marks the kingfish. The curves found for the pike are shown in Figure 6.10.1(b).

The largest subset of matching curves found by the curve matching algorithm was examined. The discrepancy values of the curve pairs were used to remove multiple matches of reference curves and overlapping matches of pike curves were also removed. The final fraction of matching curves calculated for the subset was 0.32. The matching curves from the pike are plotted in Figure 6.10.1(c).

Table 6.9.1: Subset of curves from image of fish (iii) which matched the reference fish. Note that some of the reference curves matched more than one of the new curves. Curve numbers match the labels in Figure 6.9.2.

ref.	section	new	section
1	[1-36]	16	[6-41]
1	[1-34]	17	[1-34]
2	[5-88]	21	[1-84]
2	[1-93]	24	[33-125]
3	[3-25]	19	[1-23]
4	[8-91]	21	[1-84]
4	[1-91]	24	[35-125]
5	[25-79]	20	[1-55]
5	[1-82]	22	[3-84]
5	[1-82]	25	[1-82]
6	[17-100]	21	[1-84]
6	[1-100]	24	[26-125]
6	[1-100]	25	[1-100]
6	[1-80]	26	[1-80]
7	[1-22]	30	[1-22]
8	[1-40]	23	[1-40]
8	[22-43]	30	[1-22]
8	[1-44]	31	[1-44]
9	[1-140]	25	[34-173]
9	[1-80]	26	[1-80]
9	[119-140]	30	[1-22]
10	[1-23]	23	[18-40]
10	[1-17]	27	[8-24]
10	[16-37]	30	[1-22]
11	[1-37]	18	[44-80]
11	[1-37]	26	[20-56]
12	[1-52]	28	[2-53]
13	[1-81]	25	[93-173]
13	[52-81]	29	[1-30]
14	[1-80]	18	[1-80]
15	[1-47]	25	[127-173]
15	[1-22]	30	[1-22]
15	[1-47]	31	[30-76]
15	[4-47]	32	[1-44]

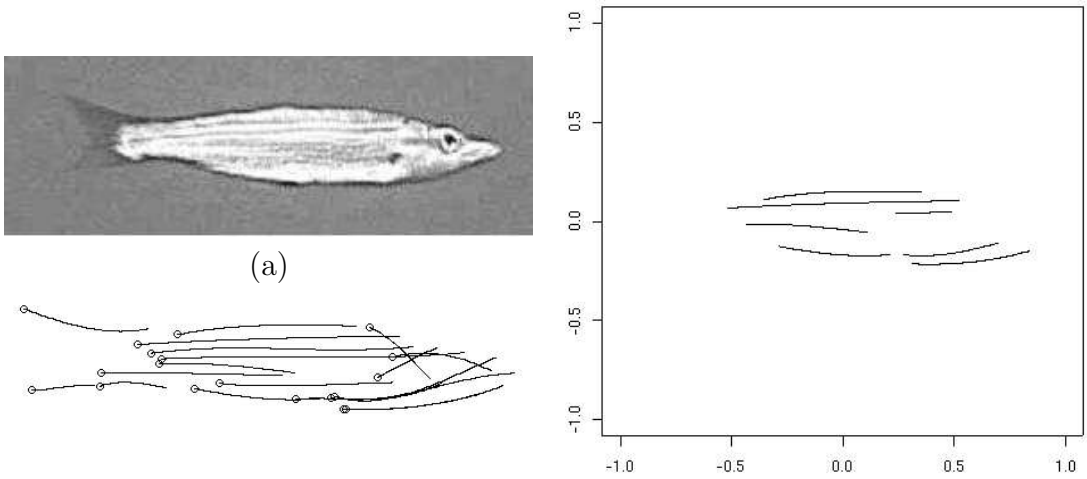


(a) (b)

Figure 6.9.3: (a) The largest subset of curves from fish (iii) which matched the reference curves. The plot clearly shows that there were sufficient curves found to consider that fish (iii) is the same type of object as the reference fish. (b) Shows the matches for the largest subset after the threshold for relative position was decreased.

Table 6.9.2: Subset of curves from image of fish (iii) which matched the reference fish when the threshold for relative distance is reduced. Discrepancy values are given for each pair in the subset.

ref.	section	new	section	discrepancy
1	[1-36]	16	[6-41]	0.3
2	[5-88]	21	[1-84]	0.27
3	[3-25]	19	[1-23]	0.38
5	[25-79]	20	[1-55]	0.38
5	[1-82]	22	[3-84]	0.29
5	[1-82]	25	[1-82]	0.47
6	[1-100]	24	[26-125]	0.4
6	[1-80]	26	[1-80]	0.74
8	[22-43]	30	[1-22]	0.34
9	[1-140]	25	[34-173]	0.27
10	[16-37]	30	[1-22]	0.28
11	[1-37]	18	[44-80]	0.43
11	[1-37]	26	[20-56]	0.38
12	[1-52]	28	[2-53]	0.4
15	[1-47]	31	[30-76]	0.36



(a) (b) (c)

Figure 6.10.1: (a) Gray-scale image of a pike. (b) Set of curves fitted to the pike. (c) Subset of matching curves found for the pike when it was compared to the reference kingfish.

The low fraction of matching curves could be taken to indicate that while there are some similarities between the two objects, they are not the same object. Inspection of Figure 6.10.1(c) indicates that the body of the pike matched reasonably well with the body of the kingfish, but there were no matches of curves in the head region or the tail. These results correspond well with what a person might see when observing the two images of the fish, i.e. that the two bodies are broadly similar in shape but the tails and heads are quite different.

6.11 Choosing a Set of Curves from an Image.

When the curve finding algorithm is applied to an image it will generally find several possible fits for each curve in the image. For example, the number of curves found for each individual fish silhouette tested here varied from 46 to 147. Before the curve comparison algorithm was run, this large set of curves was reduced manually to between 7 and 9 distinct curves per fish silhouette by eliminating small curves (such as those which outline the fins) and removing curves which were similar to other curves (or similar to subsections of other curves). For example, only one curve (preferably the longest) needs to be selected from all the curves in Figure 6.2.5(b). It is desirable to automate this process.

An algorithm to automate the curve selection process was created that worked as follows:

1. Find all possible curves for the image using the curve-finding algorithm and calculate the signed curvature of all the curves; remove curves whose length is below a threshold. The remaining curves form the original set.

2. Choose the longest curve in the original set, remove it from the original set and place it in a new set, called the *unique* set. Find all other curves whose curvature matches (within threshold limits) any part of the longest curve. Check that the locations of the start and end positions of the matching sections lie within a threshold limit. Eliminate each matching curve or section of matching curve from the original set.

3. Repeat the previous step with the longest curve reversed.

4. Repeat steps 2 and 3 until no curves remain in the original set.

The algorithm was tested on the reference fish silhouette. The algorithm reduced the number of curves in the set from 53 to 13. This is a significant reduction, however, manual reduction of the set reduced it to 7 curves. The curves which are still duplicated are those which curve strongly near one end resulting in very high curvature values at the end or beginning of the list. It does not require much apparent difference in curvature (i.e. to the human eye) of the curve to produce large changes in these values when they are high already.

The algorithm was then run on the four fish silhouettes from Figure 6.5.1. The number of unique curves selected automatically for each of the four fish was: 9, 12, 7 and 7, respectively. By contrast, manual selection had reduced the number of unique curves for each fish to 9, 9, 7 and 8 curves, respectively. The unique curves selected by the algorithm are shown in Figure 6.11.1, compare these to the manually selected curves in Figure 6.5.1.

It is essential to remove any duplicate curves for the reference set. However, for comparison purposes, it is not essential to remove duplicate curves found in the new images. Duplicate curves should not affect the result, as duplicate matches for a reference curve are removed by the algorithm. Duplicate curves from the new image will slow down the computation, but may be an advantage as they will provide more than one possible fit to match with the reference curve, increasing the chance of a good match.

To improve the automatic reduction of the duplicate curves in the reference image it would be necessary to change the method of measuring the difference between signed curvatures so as to reduce the effects of changes when the curvature measurements are very large, such as using the log of the curvature.

The automatically selected sets of curves were tested with the curve comparison algorithm. The 6 extra curves in the reference set were removed leaving 7 of the automatically selected curves. All the automatically selected curves in fish silhouettes (i) - (iv) were used. The largest subset of matching curves found for each silhouette is shown in Figure 6.11.2. The fraction of the total length of the reference curves found to match each of the illustrated silhouettes (i) - (iv) was: 0.64, 0.21,0.74 and 0.71, respectively. These values compare favourably, although slightly lower in most cases, to those found using a manual selection process, which were: 0.78, 0.15, 0.77 and 0.71, respectively. The slightly lower values indicate that there is still some

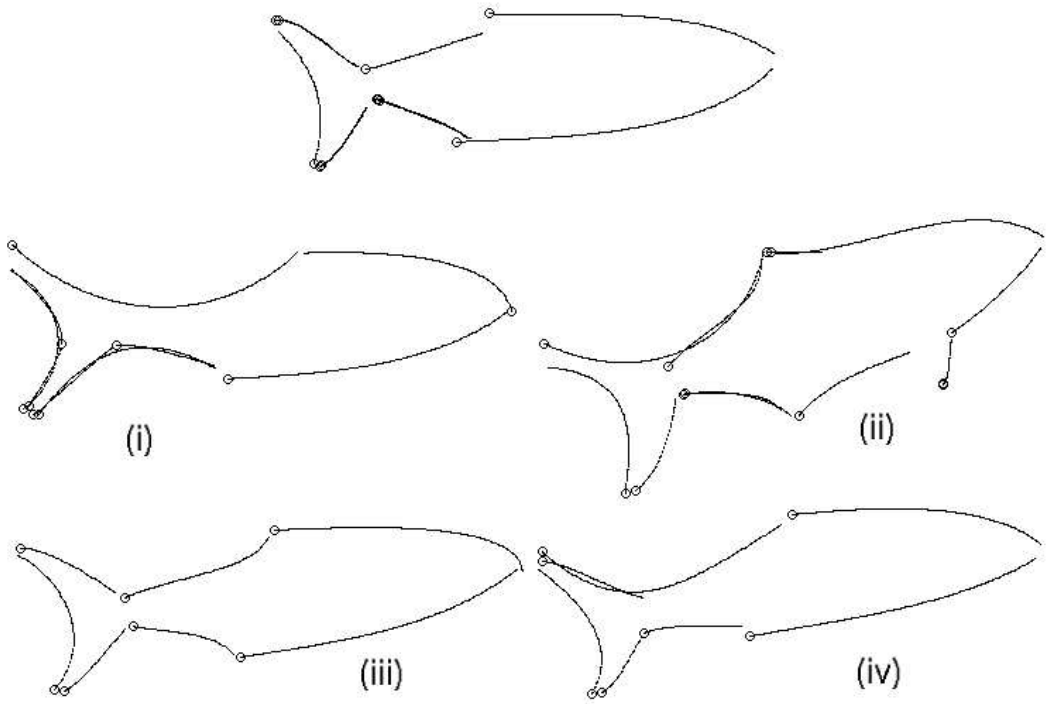


Figure 6.11.1: Unique curves automatically selected from the fish silhouettes: reference fish (top) and the four extra fish from Figure 6.5.1.

room for improvement in the curve matching algorithm.

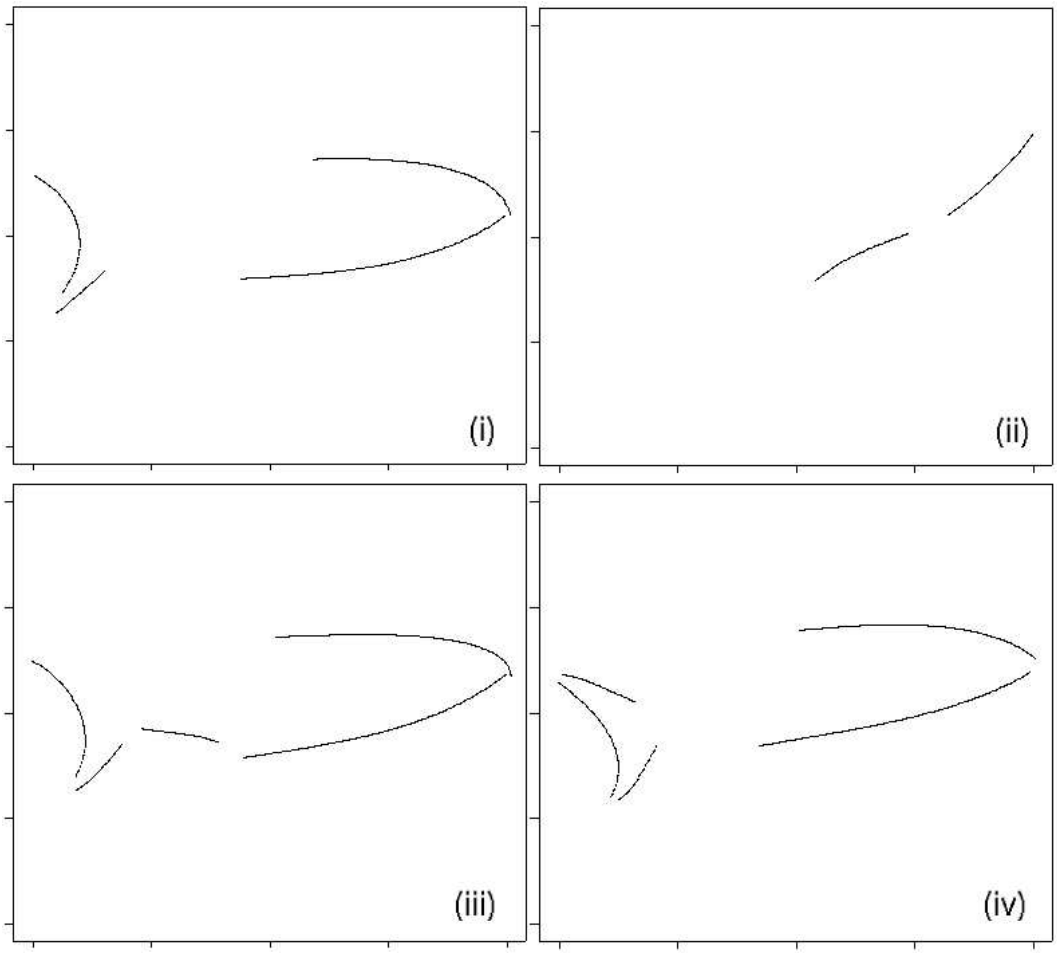


Figure 6.11.2: Plots of the best subset of matching curves for each of the four fish silhouettes in the previous figure.

Conclusions

7.1 Statistical Pattern Classification with the EM algorithm

There are several drawbacks to using the EM algorithm for Gaussian mixture models to locate objects in an image. The first problem is that the pixels which may represent fish (or other objects of interest) need to be extracted cleanly from the background of the image. This is often not possible in practical situations. Another problem is that the 2D Gaussian distribution is a highly simplified model of the arrangement of pixels in a digital image of a fish. Either a more sophisticated probability model is required, or each cluster (once it is located) needs to be tested for other attributes to confirm that it is a fish. Furthermore, this method cannot detect partly occluded objects unless the occlusion is sufficiently minimal that separate objects are still reasonably distinct.

The results of tests on locating fish using the EM algorithm indicated that the initialisation of the algorithm was critical to the success of the method, both in setting the number of components in the mixture and in setting the initial location of the components. Maximum likelihood can be used to choose between results when initialising the EM algorithm with different starting positions for the components; and model selection criteria (such as MDL and AIC) can be used for choosing between results when initialising the EM algorithm with different numbers of components. But these criteria can only be applied after the EM algorithm has been run many times with all possible numbers of objects and a variety of starting positions in each case.

Initialisation of the EM algorithm can be improved by first estimating the number and location of clusters. The k-means algorithm is a popular method for locating clusters, however it cannot be used to estimate the number of clusters. The DCF (Dynamic Cluster Finding algorithm), developed here, can be used to approximate both the location and the number of clusters. Experiments revealed that the DCF compared favourably with the k-means algorithm in locating clusters of data.

Counting the number of objects in an image by the number of clusters found in the pixels is problematic. For example, a single object may be best modeled by more than one cluster, e.g. a fish may have one cluster for the body and one for the tail. In the DCF algorithm manipulation of one of the parameters, the habituation value, can be used to control the degree to which sub-clusters may be found within larger clusters and this may be useful, for example, for counting individual fish within several schools of fish.

However, the problem of what a cluster of pixels really represents; one object, part of one object, or several adjacent objects, is one that cannot be adequately solved using these statistical approaches.

7.2 Pattern Recognition using Edge Detection and Curve Matching

For the limited number of examples tested here with the curve matching algorithm, the results indicated that the method has great potential for recognising particular types of objects for which reference examples exist. The method has the ability to find objects (or parts of objects) which are similar to a reference object (or part of a reference object) and the ability to reject objects (or parts of objects) which are different.

In the curve matching method, recognition of objects is independent of rotation or translation of the object. However, recognition is not independent of scale and mirror imaging. These two areas need further work. Recognition of mirror images is a fairly trivial exercise. It can be achieved by (1) reversing the sign of the curvature values for each curve and (2) reflecting the locations of the curves in the line $x = 0$ (for a reflection in the vertical line). Reflections in other lines do not have to be included as they can be considered as reflections in the vertical line plus rotations.

Future work on this topic should also include:

- Experimentation on effects of varying threshold values. Ideally these should be determined by statistical analysis of large data sets.
- In the first step of the algorithm, where several different sections of the reference curve may match the new curve (i.e. within the threshold limit), deciding how to choose the best option by balancing curvature discrepancy with length of match.
- In the final step, choosing what curves to include in the set when several reference curve sections match with a new curve section and
- choosing what curves to include in the set when one reference curve section matches with several new curve sections.
- Testing how well objects can be recognised in the presence of background noise.

The method used here can match objects independent of their orientation and location; however a major drawback is that the method is not independent of scale. Scale is important to consider because, even if the fish are of similar size, they may be located at different distances from the camera. The issue of scaling the reference curves is something that needs to be considered in any future work if the method is to be of practical use. It may be necessary to select a discrete set of scales for the reference object which would allow objects at any scale to be recognised within the tolerance levels of the algorithm.

Another consideration in the application of this method is that for more complicated objects than the fish, especially those with a wider range of movement. The idea introduced in chapter 6 of using sub-objects may become necessary. For example, in order to recognise a horse it might be necessary to be able to divide the horse

into semi-rigid parts which are joined by flexible joints. The method will need to be able to separately recognise the head, neck, body, foreleg above knee, foreleg below knee, hind leg above the knee, hind leg below the knee and tail. Then the method must be able to assemble various body parts in a manner which is consistent with the natural poses of a horse.

This method could potentially be used for detecting partly occluded objects. For example, if 2 heads and 3 forelegs but only one body were found in appropriate positions, and the positions overlap, then we might be able to conclude that there are at least 2 horses in the image, but one is partly occluded by the other.

A final point to mention is that the fish used in this study were composed of long gentle curves. If the objects under study were composed of sharper curves then the current method would not work so well as the smoothing of the curves requires that they are fitted to a cubic polynomial. If less simple curves were to be fitted to the edges of the object then either a higher degree function would be required, or the curve would need to be sectioned into several smaller, simpler curves.

Acronyms and Symbols

Acronyms

EM	Expectation-maximisation algorithm
GMM	Gaussian mixture model
AIC	Akaike Information Criterion
MDL	Minimum Description Length
DR	Dog-Rabbit strategy
DCF	Dynamic Cluster Finding algorithm

Symbols for chapters 2 and 3

$\mathbb{E}$	expected value
θ	vector of parameters
x	data point, $x_i \in R^n$
X	set of data points $X = x_1, \ldots x_n$
w	weight
f	probability density function
g	Gaussian probability density function
μ	mean
Σ	covariance matrix
L	likelihood
l	log likelihood
I	identity matrix
λ	scale factor
Q	normalised orthogonal matrix
Λ	diagonal matrix
$\hat{\theta}$	vector of estimated parameters
P	probability
f, g	probability density functions
$\mathcal{C}$	criterion function
$\mathcal{M}$	model family
M	model class
$\mathcal{Q}$	expected value of the total likelihood
k	number of parameters, $k \in \mathcal{K}$
I	stochastic complexity
N	number of independent observations
J	Fisher information matrix
$\mathcal{L}$	minimum code length

Symbols for chapter 4

c_j	(x,y) location of a cluster locator
x_i	(x,y) location of a data point
d_{ij}	distance between c_j and x_i
α	lateral inhibition factor
λ	lateral inhibition parameter
β	habituation factor
h	habituation parameter

Symbols for chapter 5

κ_s	signed curvature
S	sign of curvature
T	tangent to curve
$\mathbf{r}$	parametric function of a curve

Symbols for chapter 6

α	modification factor for curve matching algorithm
β	scale factor for curve matching algorithm
P_{ab}	vector joining the starting points of two curves, a and b
D_i	discrepancy value for curve matching

Volume under the 2D Gaussian surface

The fraction of data points which may lie within 2 standard deviations of the mean in a two dimensional Gaussian probability density function (Gpdf) can be calculated from the volume under the surface.

For the Gpdf with covariance matrix:

$$\mathrm{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

centred on the origin $\mathbf{0}$

$$f(x, y) = N(\mathbf{0}, I) = \frac{1}{2\pi} e^{-\frac{1}{2}(x^2+y^2)}$$

the total volume under the surface is given by:

$$A = \int_{x=-\infty}^{\infty} \int_{y=-\infty}^{\infty} f(x, y)\, \mathrm{d}x\mathrm{d}y = 1$$

or expressed in polar coordinates:

$$A = \int_{\theta=0}^{2\pi} \int_{r=0}^{\infty} f(r, \theta) r\, \mathrm{d}r\, \mathrm{d}\theta = 1$$

The volume of the area from the mean (i.e. the origin for a normal Gpdf) to 2 standard deviations (2σ) is:

$$\begin{aligned} A_{2\sigma} &= \frac{1}{2\pi} \int_0^{2\pi} \int_0^2 e^{-\frac{1}{2}(r^2)}\, r\, \mathrm{d}r\, \mathrm{d}\theta \\ &= \frac{2\pi}{2\pi} \int_0^2 e^{-\frac{1}{2}(r^2)}\, r\, \mathrm{d}r \\ &= \int_0^2 e^{-u}\, \mathrm{d}u \text{ (where } u = 1/2\, r^2,\ \mathrm{d}u = r\, \mathrm{d}r) \\ &= \left[-e^{-u}\right]_0^2 = -e^{-2} + e^0 \\ &\approx 0.86 \end{aligned}$$

Hence, approximately 86% of points will be included in an ellipse drawn at 2σ from the mean of a 2D Gpdf.

Template Matching: Finding an Eye

C.1 Introduction

Template matching is a popular tool in the literature for pattern recognition for simple objects such as characters, but it has a number of limitations: the patterns to be recognised must be of the same scale and orientation as the template and must not be distorted. In the case of underwater images there can also be a great degree of variation in colours between the object to be recognised and the reference object due to the available light and water clarity.

C.2 Making a Template for an Eye

Template matching is a suitable method for finding eyes in fish as the eyes tend to be very small in the images, generally only one or two pixels in width, so scale is less of a problem. The shape of the eye is also very simple and close to symmetrical so changes in orientation have a less significant effect.

The language Java was used to code the template matching algorithm. Java was preferred over R in this case because of its ability to easily interact with digital image data. The code for the template matching algorithm is given in Appendix E.

An example image is used to create the initial template. The template design used in this example is shown in Figure C.2.1. The eye is essentially a small dark region surrounded by a lighter region, because of its small size there is generally a one pixel wide region around the eye whose brightness lies somewhere between that of the eye and that of the surrounding region. A template has been constructed based on a study of size and colouration of several fish eyes in an image. In this example the template is 6 pixels wide and 5 pixels high and includes the eye region which is 2 by 1 pixels (Figure C.2.1).

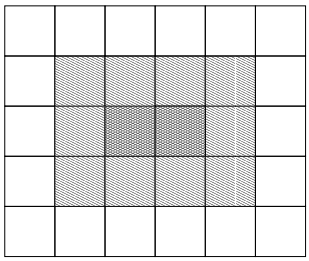


Figure C.2.1: The template design for locating a fish eye. The centre region is black (shown cross-hatched) and represents the actual eye. The outer region (white) represents the average colour of a fish body. The middle region (diagonal lines) is the average of the centre and outer regions.

The centre of the template is black (although in reality the centre of the eye of a fish in these images is usually recorded as very dark greenish-blue). The surrounding

Figure C.2.2: Four eyes located on fish by template matching are indicated by white squares. This is the original image from which the template colours and size were defined.

colour is taken from an average of the colour of all the fish found in the image (found using thresholding and the EM algorithm). And the middle colour of the template is an average of the surrounding colour and the centre colour. The average surrounding colour which has been calculated is compared to the original fish colours and the maximum variability is used to define tolerance levels for template matching.

Once the template and tolerance levels have been defined, a rectangular search region is defined around the fish using the extent of the 2σ ellipse (calculated by the EM algorithm). The intensity of each pixel in each 6 by 5 pixel area inside the search region is compared with the template and the total difference is calculated. If this difference is within tolerance levels then a second test is performed to check that the difference in brightness between the centre and the surround reaches a predefined level. Any area passing these two tests is a candidate for an eye. Once the whole region is searched the best candidate is chosen based on the best match with the template.

Figure C.2.2 shows the template matching algorithm run on the image from which the original eye colours were defined. All four visible eyes have been located. Figure C.2.3 shows the same template applied to a new image with similar lighting conditions and same species of fish (Australian herring). In this case only two of the four visible eyes have been located. The other two eyes appear to be slightly more elongate than the originals suggesting that a slightly larger template needs to be applied in addition to the original one.

Figure C.2.3: Two eyes located on fish by template matching are indicated by white squares. Poor success rate is attributed to larger eye size in this image compared to the original (previous figure).

Comparing the k-means and DCF algorithms

D.1 Introduction

This section documents the results of comparison tests between the performance of the well known and widely used k-means clustering algorithm and the Dynamic Cluster Finding (DCF) algorithm (Chapter 4). The R code used for running the tests and the data sets on which the tests were conducted are included in Appendix E.

DCF and k-means were subjected to 5 different types of tests to determine if the number and arrangement of clusters affects the success rate of the two clustering algorithms. The data in each cluster are selected from a Gaussian probability density function using R's *mvrnorm* function. In addition, varying amounts of random uniform background noise was added to check what effect noise has on the results. In these tests the number of clusters being searched for is presumed to be already known.

Every test was repeated 50 times and then this value was doubled to obtain the percentages of successful tests recorded in the tables below. A successful test is one in which all the clusters were located; examples of successful and failed tests are given in the figures below.

D.2 Results of Tests

Test 1: Equal-eight. This test consists of eight equal clusters (i.e. same number of points, same probability distribution) evenly spread throughout a square region (Figure D.2.1(a)). The amount of random uniform background noise added is shown in Table D.2.1 and is calculated as a percentage of the total number of points in the eight clusters.

The results of Test 1 are documented in Table D.2.1. They show that DCF has a much higher success rate than k-means for this type of data. Interestingly, adding noise improved the success rate of both algorithms. Examples of successful and failed tests are shown in Figure D.2.1.

Table D.2.1: Test 1, percentage of successful tests performed on k-means and DCF algorithms, with varying amounts of noise.

noise	0	5%	10%	20%	40%	60%	100%
k-means	34	32	26	38	38	46	64
DCF	58	68	72	80	74	82	90

Test 2: Tetrahedron. In the second test four equal clusters (i.e. same number of points, same probability distribution) form a pattern of a flattened tetrahedron

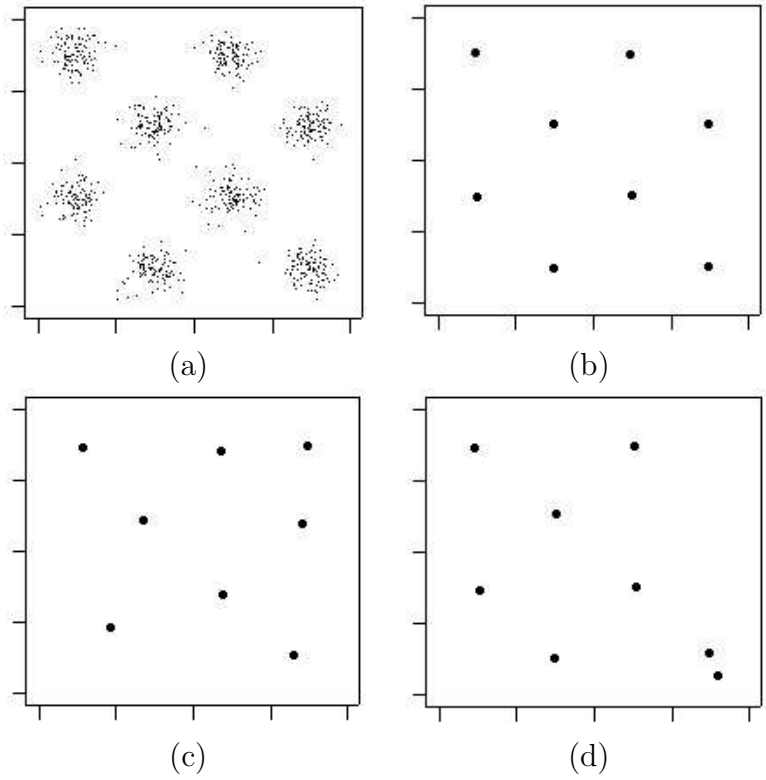


Figure D.2.1: (a) Arrangement of clusters used in Test 1; (b) successful test; (c) typical failed test for k-means; (d) typical failed test for DCF.

(Figure D.2.2(a)). The effects of adding noise were also tested.

Both algorithms performed well, although DCF showed a marginally lower success rate than k-means. Unlike the first test with eight clusters, in this test adding noise decreased the performance of both algorithms (Table D.2.2).

The different effects of noise may be due to the amount of open space between clusters. In the test where there is little open space the noise is not scattered far from clusters. However, where there are large amounts of open space, the noisy data points may lie much further away from the clusters and hence are more likely to attract the locators away from the centre of the clusters.

Table D.2.2: Test 2, percentage of successful tests performed on k-means and DCF algorithms, with varying amounts of noise.

noise	0	5%	10%	20%	40%	60%	100%
k-means	96	96	88	74	60	68	44
DCF	92	92	82	76	56	60	38

Test 3: Reduced Tetrahedron. This test is designed to determine whether poor cropping of the data region effects the results of the two cluster finding algo-

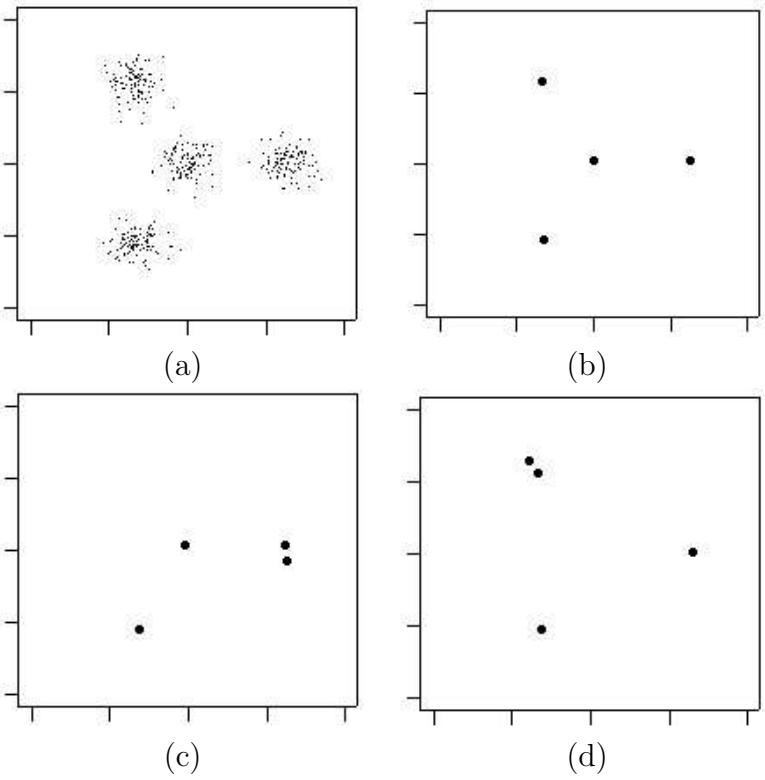


Figure D.2.2: (a) Arrangement of clusters used in Test 2; (b) successful test; (c) typical failed test for k-means; (d) typical failed test for DCF.

rithms. The same cluster arrangement as Test 2 was used. Two versions of the tests were run:

1. the clusters are reduced to lie in the top left-hand quarter of the data region but the noise and initial positions of cluster locators were distributed throughout the region;

2. the clusters, noise and initial positions of the cluster locators were reduce to the top left-hand quarter of the region.

In the first version of the test (see results in Table D.2.3(a)) k-means performed extremely well when there is no noise; however, the presence of minor amounts of noise quickly rendered the algorithm useless, i.e. the algorithm failed to locate all of the clusters in all of the tests. While the DCF algorithm performed significantly worse than k-means in the absence of noise, it was much less negatively affected by the presence of noise.

In the second version of the test, k-means performance was essentially the same as in the non-reduced version (compare Table D.2.3(b) with Table D.2.2). However, the performance of the DCF algorithm has been affected by the scaling of the data and the locators to the top left quarter. This scaling would eliminate the effects of the toroidal topology because locators would always be closer to the data in the flat

image and therefore would never cross the boundaries of the image to take advantage of the toroidal topology. For low or no noise DCF performed better in the unscaled test, but for higher noise levels DCF performed better in the scaled version (compare tables D.2.2 and D.2.3(b)).

The second version of tests suggest that the performance of the DCF algorithm in a poorly cropped data region with low noise will be significantly reduced because of the reduction in the effects of the toroidal topology.

Table D.2.3: Results of test 3: (a) clusters only have been reduced to the top left-hand quarter of the data region; (b) clusters, noise and initial positions of cluster locators have been reduced to the top left-hand quarter of the data region.

(a)

noise	0	5%	10%	20%
k-means	100	46	0	0
DCF	68	66	48	44

(b)

noise	0	5%	10%	20%
k-means	100	98	92	74
DCF	74	86	88	86

Test 4: Sub-clusters. This example consists of two clusters which comprise 2 and 3 smaller sub-clusters, respectively (Figure D.2.3). Each sub-cluster has an equal number of data points. It is designed to test the ability of the clustering algorithms to find the correct number of sub-clusters.

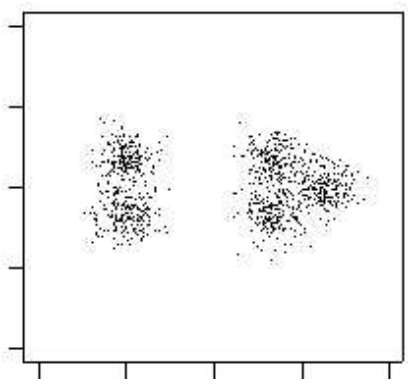


Figure D.2.3: Arrangement of clusters used in Test 4.

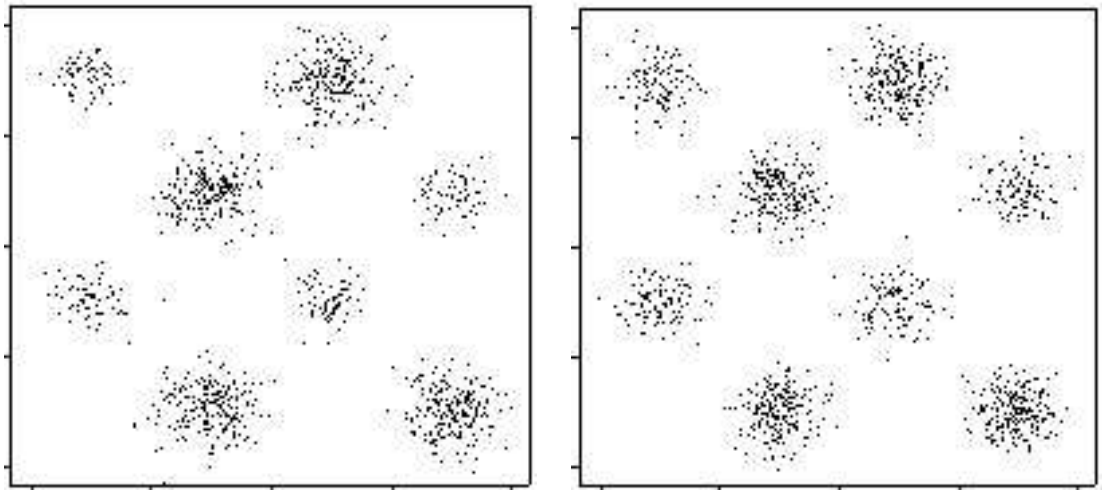
Table D.2.4: Test 4, percentage of successful tests performed on k-means and DCF algorithms, with minor variation of noise.

noise	0	5%	10%	20%
k-means	34	28	52	30
DCF	44	58	52	44

DCF performed marginally better than k-means. The addition of noise has no consistent effect.Overall performance of both algorithms is poor (Table D.2.4).

Test 5: Size/density Variation. Eight clusters were used in this test in the same arrangement as Test 1. However, 4 of the clusters have half as many points in them as the other 4 clusters. Two variations were tested: (1) the smaller clusters have the same size covariance matrix as the larger clusters, (2) the smaller clusters have a correspondingly smaller covariance matrix making it more dense than the small clusters in the first example. These are shown in Figure D.2.4.

Performance of both algorithms is not encouraging; however DCF is more successful at detecting clusters where the size varies.



(a) (b)

Figure D.2.4: Types of clusters used in Test 5: in (a) and (b) four of the clusters (e.g. the top left-hand cluster in both figures) have half as many data points as the other clusters. In (a) a smaller covariance matrix was used for the smaller clusters; in (b) covariance matrices were the same for both sizes of clusters.

Table D.2.5: Test 5, percentage of successful tests performed on k-means and DCF algorithms. Two different tests performed by varying the size of the covariance matrices (cov.) on the Gaussian distributions for the half-size clusters.

test	smaller cov.	same cov.
k-means	24	36
DCF	32	50

D.3 Conclusions

The results of these tests indicate that, except when the data region is poorly selected and there is no noise (Test 3), the DCF method either out-performed the k-means algorithm or gave only marginally poorer performance. That the DCF

performed badly for a poorly selected data region when there is no noise is not considered a serious drawback as it would be a trivial matter to crop the data region to fit the data more compactly when there is no noise, as it is the noise which makes cropping of the data region problematic.

Algorithms and Computer Code

The algorithms used in generating the curves as described in Chapter 5 and for matching curves as described in Chapter 6 are described here. Only a shortened description is provided for those algorithms already described in detail in the chapters. The computer code from all algorithms tested in the thesis is included on the enclosed CD as part of this appendix.

E.1 Computer Code on CD

Computer code included in this appendix has been written in:

- R, a the statistical programming language (http://www.r-project.org/)
- Java (http://java.sun.com/)
- MATLAB (http://www.mathworks.com/)

In addition GGobi (http://www.ggobi.org/) and the Gimp (http://www.gimp.org/) were used for visualisation and manipulation of images.

Code on the CD is organised in directories as follows:

- R-code/
  - em/ [code for EM algorithm]
  - dcf/ [code for DCF algorithm]
  - curve/ [code for curve-finding and curve-matching algorithms]
- java-code/
  - tileImage/ [code for tiling an image]
  - threshold/ [code for thresholding images]
  - template/ [code for finding a fish eye using a template]
- matlab-code/
  - code lines for using the Canny edge detector in MATLAB
  - includes an R file for importing MATLAB output to R

Each directory contains a README file which explains what the code does and how it is used.

E.2 Algorithms from Chapter 5

FILE: *run-curv.r*

SUMMARY:

- The file *run-curv.r* contains lines of code which call functions to find smooth and gentle curves in the edge pixels of an image.

INPUT: The user is required to edit the file *run-curv.r* to provide:

- the name of the data file containing (x,y) coordinates of data points (i.e. edge pixels), x values in column 1 and y values in column 2;
- the radius of the neighbourhood (0.05 was used for silhouettes and 0.03 for gray-scale images), the file *run-neighb* is provided for the user to experiment with the potential results of using different neighbourhood sizes.

OUTPUT: The following R objects are produced by this code:

- *allkv*: a list of arrays, each array contains the signed curvature values for one curve;
- *allxy*: a list of arrays, each array contains the (x,y) coordinates of the signed curvature values for one curve;
- *long.kv*: as for *allkv*, but with short curves removed;
- *long.xy*: as for *allxy*, but with short curves removed.

Code is also provided for saving the objects and plotting the curves.

ALGORITHM:

1. Import data and plot location of edge pixels.
2. Find gently curving lines in the edge pixels using the function *all.curves*; store as a list of arrays, where each array contains the locations of the means of all the neighbourhoods belonging to one curve.
3. Fit a parametric cubic polynomial to each curve using the function *pc.coef*; store as an array with the 4 coefficients for each cubic polynomial.
4. For each curve which consists of more than 2 neighbourhoods, use the polynomial coefficients to calculate: (a) a list of signed curvature values and (b) a list of locations at which the curvatures were calculated, using the function *curvature*.

5. Short curves can be optionally discarded by setting a minimum length of curve (minlength = 0.2 was used).

6. The data for a full set of curves or a set of long curves only can be saved as objects; the name must be edited to give unique names to objects (sets of curves) before saving so that many sets can be loaded into one session without them overwriting each other.

7. Two functions are provided for plotting the curves in an existing window:

   - *plot.curve* plots the curves from points calculated from the parametric cubic polynomial coefficients;
   - *plot.lines* plots the curves using lines joining the locations of each of the curvature measurements.

   Each of these functions has one required argument, the index number of the curve in the list of curves, and one optional argument, a number for the colour.

8. The set of curves to use for the reference fish must be manually selected from the list of curves as this part of the algorithm has not yet been fully automated, the plotting functions are designed to assist in this process. However, most of the duplicate curves can be removed using the code in the file *run-select.r*.

FUNCTION: *all.curves*

SUMMARY:

- Finds gently curving lines in a set of data points (edge pixels).

INPUT:

- An array of (x,y) coordinates of the data points (x values in column 1 and y values in column 2);
- radius of the first neighbourhood.

OUTPUT:

- A plot showing chains of ellipses where each ellipse represents the mean and covariance matrix of the neighbourhood; ellipses are plotted at a Mahalanobis distance of 2 from the mean. Each separate curve, represented by a chain of ellipses, is plotted in a separate colour. This plot is created dynamically during the running of the algorithm.
- A plot of the lines created by joining the means of each curve.

- A list of arrays, each array stores the (x,y) coordinates of the means of the neighbourhoods for one curve.

ALGORITHM:

1. Select the first point in the data set to be the centre of the first neighbourhood in the curve.
2. If the point is tagged, reject the point and go on to the next point in the data set, else calculate the mean and covariance of the neighbourhood using the function *neighbourhood*.
3. Tag and reject the data point if there are insufficient data points to form a new neighbourhood.
4. Tag and reject the data point if the ratio of major to minor eigenvectors is below a threshold.
5. Find all the adjacent neighbourhoods which define a curve using the function *find.curve*.
6. Tag all data points which are in the neighbourhood.

FILE: *run-neighb*

SUMMARY:

- Finds and plots neighbourhoods for edge pixels, neighbourhoods are located randomly. Designed to allow the user to test for the best neighbourhood size.

INPUT:

- The location and name of the file containing the edge pixel locations (data will be automatically scaled);
- the radius of the neighbourhood.

OUTPUT:

- Plots an ellipse at a Mahalanobis distance of 2 from the mean of each neighbourhood.
- Plots a histogram showing the frequency of various ratios of the maximum eigenvalue to the minimum eigenvalue (a high frequency of high ratios is desirable).

FUNCTION: *neighbourhood*

SUMMARY:

- Finds the data points which lie in the neighbourhood of a selected data point, and calculates the mean and covariance of the neighbourhood data points.

INPUT:

- (x,y) coordinates of the selected data point;
- an array of (x,y) coordinates of all the data points (x values in column 1 and y values in column 2);
- the radius of the neighbourhood.

OUTPUT:

- A list with:
  1. the mean,
  2. the covariance matrix,
  3. an array of tags for each data point (1 if the point is in the neighbourhood, 0 if not).

ALGORITHM:

1. For each data point in the set, if the distance between it and the selected data point is less than or equal to the radius then tag the data point.
2. If the total number of tagged data points is less than 5 reject the neighbourhood, else calculate the mean and covariance of the tagged data points.

FUNCTION: *find.curve*

SUMMARY:

- Finds a series of neighbourhoods which can define a curved line from the edge pixels.

INPUT:

- An array of (x,y) coordinates of all the data points (x values in column 1 and y values in column 2);

- the mean of first neighbourhood;
- the covariance matrix of first neighbourhood (2 by 2 array);
- a number to indicate colour for plotting a chain of ellipses.

OUTPUT:

- Plots the chain of ellipses in the specified colour in an existing plot window;
- a list with:
    1. an array of tags for each data point (1 if the point is in any of the neighbourhoods in the chain, 0 if not);
    2. an array of (x,y) coordinates of means of each neighbourhood.

ALGORITHM:

1. Search in one direction from the first neighbourhood, find the next neighbourhood using function *next.ellipse*, this is the head of the curve.
2. Plot the ellipse for the new neighbourhood.
3. Tag the data points in the new neighbourhood.
4. Keep searching for the next neighbourhood, plotting and tagging; stop when either (a) the function *next.ellipse* returns the value NULL or (b) the angle between the eigenvector of the next neighbourhood and the current neighbourhood exceeds a threshold (i.e. successive neighbourhoods no longer define a gentle curve).
5. Repeat search, starting at the first neighbourhood again but in the opposite direction, this is the tail of the curve.

FUNCTION: *next.ellipse*

SUMMARY:

- Finds the next neighbourhood on a curve using the angles between previous neighbourhoods to predict the location.

INPUT:

- An array of (x,y) coordinates of all the data points (x values in column 1 and y values in column 2);

- a list which describes the current neighbourhood, it contains the same structure and data as the OUTPUT of this function, if the current neighbourhood is the first neighbourhood then element (4) of the list is the default principal eigenvector and element (5) of the list is 0;
- a value of 1 if the search is in the head of the curve and 0 if the search is in the tail of the curve;
- the covariance matrix of the first neighbourhood in the curve.

OUTPUT:

- A list with:
  1. an array of tags for each data point (1 if the point is in the new neighbourhood, 0 if not);
  2. the mean of the new neighbourhood;
  3. the covariance matrix of the new neighbourhood;
  4. the principal eigenvector of the new neighbourhood aligned in the direction of search;
  5. the angle between the principle eigenvector of the new neighbourhood and that of the previous neighbourhood.

ALGORITHM:

1. The mean and principal eigenvector of the previous neighbourhood are rotated by the angle between the principal eigenvectors of the new neighbourhood and the previous neighbourhood; the point of rotation is at a Mahalanobis distance of 2 from the mean in the direction of the search.
2. Tag data points which lie (1) within the distance limits from the rotated mean of the previous neighbourhood and (2) within the angle limits from the rotated principle eigenvector of the previous neighbourhood. Search in the opposite direction to the principal eigenvector direction if searching in the tail instead of the head of the curve.
3. If the number of points tagged for the new neighbourhood is less than the threshold, then return NULL for all OUTPUT values, else:
   - calculate the mean and covariance matrix of the new neighbourhood,
   - determine the direction of the next search,
   - and calculate the angle between the previous and current neighbourhood's principal eigenvectors.

Values of Thresholds Set in this Function:

- 4, minimum number of points to form a new neighbourhood.
- $4.5\sqrt{max\{eigenvalue\}}$, maximum distance from mean to search area.
- $0.7\sqrt{max\{eigenvalue\}}$, minimum distance from mean to search area.
- $\frac{\pi}{18}$, maximum angle either side of major axis to search.
- $1.5 \times min\{eigenvalue\}$, lowest minimum eigenvalue value allowed to avoid singular (or nearly singular) covariance matrices. If the minimum eigenvalue is below this amount then the eigenvalue is reset to this value and the covariance matrix recalculated using this eigenvalue.

All the threshold values were determined experimentally.

FUNCTION: *pc.coef*

SUMMARY:

- For each chain of ellipses found using *all.curves* calculate the coefficients of the parametric cubic polynomials.

INPUT:

- An array of (x,y) coordinates of points.

OUTPUT:

- An array of 8 coefficients: $[a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2]$, where:
  1. $x = a_1x^3 + b_1x^2 + c_1x + d_1$,
  2. $y = a_2x^3 + b_2x^2 + c_2x + d_2$.

ALGORITHM:

1. Call the function *para.cubic* to get the coefficients of fitted parametric cubic polynomial and the values of residuals.
2. Plot a series of points representing the polynomial on an existing plot in black, if a given residual threshold (0.03 used) is exceeded then plot the polynomial in red and print a statement.

FUNCTION: *para.cubic*

SUMMARY:

- Fits a parametric cubic polynomial to an array of points using the least squares method.

INPUT:

- Array of points as (x,y) coordinates.

OUTPUT: A list with:

- an array of 8 coefficients: $[a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2]$, where:
  1. $x = a_1x^3 + b_1x^2 + c_1x + d_1$,
  2. $y = a_2x^3 + b_2x^2 + c_2x + d_2$;
- residuals for x produced by *lm()*;
- residuals for y produced by *lm()*.

ALGORITHM:

1. Format data.
2. Call R's *lm()* function for fitting linear models.

FUNCTION: *curvature*

SUMMARY:

- Calculates the signed curvature values at fixed length intervals on a section of a parametric cubic polynomial.

INPUT:

- The coefficients of the parametric cubic polynomial;
- the first parameter value of the section of the polynomial;
- the last parameter value of the section of the polynomial;
- the number of points at which to calculate curvature values initially;
- the length of interval at which to calculate the final curvature values.

OUTPUT: A list with:

- an array of locations of points at which curvature was calculated;
- an array of tangents at points at which curvature was calculated;
- an array of signed curvature values.

E.3 Algorithms from Chapter 6

FILE: *run-comp.r*

SUMMARY:

- *run-comp.r* contains lines of code which call functions to compare two sets of curves.

INPUT: The user is required to edit the file *run-comp.r* to provide two set of curves to compare (i.e. the reference set and the new set) in the following format:

- EITHER choose one of the existing sets by assigning set numbers to variables *new.set* and *ref.set* and running function *choose-set*
- OR create the following arrays:
  1. Reference set:
     - *ref.kv*: name of list of arrays of signed curvature values (i.e. *all.kv* or *long.kv* from *run-crv*);
     - *ref.xy*: name of list of arrays of locations of curvature values (i.e. *all.xy* or *long.xy* from *run-crv*);
     - *ref.ind*: array of indices of curves to use from list, if whole set is to be used then *ref.ind* = 1:*length(ref.kv)*.
  2. New set:
     - *new.kv*: name of list of arrays of signed curvature values (i.e. *all.kv* or *long.kv* from *run-crv*);
     - *new.xy*: name of list of arrays of locations of curvature values (i.e. *all.xy* or *long.xy* from *run-crv*);
     - *new.ind*: array of indices of curves to use from list, if whole set is to be used then *ref.ind* = 1:*length(ref.kv)*.

OUTPUT: Objects produced by this code include:

- *all.mssq*: a list of data frame objects, one data frame for each reference curve. Each data frame contains the following columns:
  1. the index number of the curve from the list of reference curves;
  2. the fraction of the reference curve which matched the new curve;
  3. the first index in the section of the reference curve which matched;
  4. the last index in the section of the reference curve which matched;
  5. the index number of the curve from the list of new curves;

6. the fraction of the new curve which matched the new curve;
7. the first index in the section of the new curve which matched;
8. the last index in the section of the new curve which matched;
9. the angle between the two lines which join the start and end points of the reference curve and new curve, respectively;
10. the discrepancy value for the curve match.

Each row represents one curve pair, where a curve pair is a section of a reference curve and a corresponding section of a new curve which had the best match for curvature values within the threshold.

- *ang.OK*: A list of arrays, one array for each set of curves with similar relative orientation. Each row of the array contains the following fields:

  1. the data frame number and
  2. the row number of the data frame, to indicate which curve pair it refers to;
  3. the average discrepancy value in the orientation for the set;
  4. the average angle between all the curve pairs in the set.

- *pos.OK*: A list of arrays, one array for each subset of curve pairs with similar relative position. Each row of the array contains the following fields:

  1. the data frame number and
  2. the row number of the data frame, to indicate which curve pair it refers to;
  3. the average discrepancy value in the orientation for the subset;
  4. the average angle between all the curve pairs in the subset;
  5. the average discrepancy value in relative position for each subset.

  Arrays with only one row and arrays which are subsets of other arrays are removed.

- *combined*: A list of arrays, same as *pos.OK*, but where 2 subsets in *pos.OK* contain the same curve pair these subsets have been combined by taking the union of the 2 subsets.

- *D.total*: An array of values calculated for each subset in *combined*: the square root of the weighted sum of square discrepancies from *mssq*, the relative orientation and the relative position.

ALGORITHM:

1. For each reference curve in the set:

- use function *longest.match* to find *mssqd* of the longest matching section with each curve from the new set (forward search);
- repeat for reversed reference curves, append to forward search results;
- calculate the weighted discrepancy values for each curve pair;
- for each curve pair, calculate the angle from the reference curve to the new curve using the function *curv.angle*;
- save all results as a data frame and add it to the list of all data frames (*all.mssq*).

2. Make an array *ovr.OK* of all curve matches whose length exceeds a threshold, using the function *overlap*.

3. Make a list of arrays *ang.OK*, where each array stores a set of curve pairs which have similar relative orientation, using the function *orientation*. Calculate the weighted discrepancies for each set, remove any sets which are subsets of other sets.

4. Make a list of arrays *pos.OK*, where each array stores a subset of curve pairs which have similar relative position using the function *position*. Calculate weighted discrepancies for each subset; remove any subsets which are subsets of other sets; remove any subsets with only one pair.

5. Make a list of arrays *combined* by combining sets which have pairs in common using the function *combine*.

6. Calculate the square root of the sum of all squared weighted discrepancies (*D.total*).

7. Plot the new curve sections which matched the reference curves, each subset in a separate plot; print results.

FUNCTION: *longest.match*

SUMMARY:

- Calculates *mssqd* values of the longest matching section within the threshold for each new curve.

INPUT:

- Array of curvature values for reference curve;
- list of arrays of curvature values for all new curves;
- threshold for *mssqd*;

- reverse=TRUE if matching reversed reference curve.

OUTPUT:

- An array with *mssqd* value for each match, or NA if it exceeds threshold;
- a 2 column array with start and end indices of the matched section for the reference curve;
- a 2 column array with start and end indices of the matched section for the new curve;
- an array with the fractions of the matched section for the reference curve;
- an array with the fractions of the matched section for the new curve.

If mssqd is NA then indices and fractions are 0.

ALGORITHM:

1. For each new curve, calculate *mssqd* and *mmd* for all allowable section matches with the reference curve, using the function *crv.cmp*.
2. Choose the longest match which lies within threshold using the function *best.length*.
3. Create an array of *mssqd* values for the best match for each new curve, if no matches were within threshold assign NA to that curve for the *mssqd* value.
4. Find the range of indices for the section which matched best using the function *find.range*, store start and end indices for the reference curve and the new curve in arrays.
5. Using the range, calculate the fraction of each curve that matched.
6. Invert the order of the indices if the match was with a reversed reference curve.

FUNCTION: *crv.cmp*

SUMMARY:

- Calculates *mssqd* and *mmd* for all section matches of the new curve with the reference curve.

INPUT:

- Array of signed curvature values for the reference curve;

- array of signed curvature values for the new curve.

OUTPUT:

- An array, column 1 = *mssqd*, column 2 = *mmd*.

ALGORITHM:

1. Find which curve is longest;
2. use functions: *mssq* and *mmd* to calculate *mssqd* and *mmd* for:
    - all overlaps which include the start of the longer curve and the end of the shorter curve;
    - all overlaps which include part of the longer curve and all of the shorter curve;
    - all overlaps which include the end of the longer curve and the start of the shorter curve.

FUNCTION: *best.length*

SUMMARY:

- Chooses the matching curve pair which has the greatest length but with *mssqd* within the threshold. If two have same size overlap then the overlap with the lowest *mssqd* is chosen.

INPUT:

- Array of *mssqd* values for all overlapping sections for one curve pair;
- the length of reference curve;
- the length of new curve;
- threshold for *mssqd*;

OUTPUT:

- The index of the array of *mssqd* values at which the best overlap occured.

ALGORITHM:

1. Make an new array containing the indices of the input array which are within the threshold.

2. Find the longest section which matches.
3. If more than one section is same length, then choose the one with the lowest *mssqd*.
4. Calculate the index of the *mssqd* array which refers to this curve section.

FUNCTION: *find.range*

SUMMARY:

- Finds the range of indices for the sections of matching curve using the index number and the lengths of the two curves.

INPUT:

- The index of the array of *mssqd* values that gave the best matching section for a pair of curves;
- the length of longer curve;
- the length of shorter curve.

OUTPUT:

- An array with indices for: the start of the long curve, the end of the long curve, the start of the short curve, the end of the short curve.

ALGORITHM:

1. Determine whether a section of the longer curve matched the whole short curve or the beginning section or the end section of the shorter curve.
2. Assign index values to the start and the end of the long curve, using index of best *mssqd* value for that curve pair.
3. Assign index values to the start and the end of the short curve, as above.

FUNCTION: *mssq*

SUMMARY:

- Computes the modified sum of square errors (*mssqd*) for a section of a curve.

INPUT:

- Array of signed curvature values for a section of the reference curve;
- array of signed curvature values for a section of the new curve.

OUTPUT:

- *mssqd* value.

FUNCTION: *mmd*

SUMMARY:

- Computes the modified maximum error (*mmd*) for a section of a curve.

INPUT:

- Array of signed curvature values for a section of the reference curve;
- array of signed curvature values for a section of the new curve.

OUTPUT:

- *mmd* value.

FUNCTION: *curv.angle*

SUMMARY:

- Finds the line joining the start and end points of the matching section of the reference curve and then of the new curve. Finds the angle of rotation from the reference line to the new line.

INPUT:

- Data for all the matched curve pairs for one reference curve;
- coordinates of the reference curvature values;
- coordinates of the new curvature values.

OUTPUT:

- Array of angles.

FUNCTION: *overlap*

SUMMARY:

- Gets data frame indices for all curve pairs which have at least a minimum length of section match.

INPUT:

- List of data frames;
- threshold.

OUTPUT:

- An array:
  1. col 1 holds data frame number;
  2. col 2 holds row number in data frame.

ALGORITHM:

1. For each pair of curves calculate the length of the matching section.
2. If the length is greater than or equal to the threshold, add the indices of the pair to an array.

FUNCTION: *orientation*

SUMMARY:

- Gets sets of data frame indices for all curve pairs which have at similar orientation.

INPUT:

- List of data frames;
- indices for all curve pairs to test (output of *overlap*);
- threshold.

OUTPUT:

- A list of arrays, where each array represents one set, in each array:

1. col 1 holds the data frame number;
2. col 2 holds the row number in the data frame;
3. col 3 holds the discrepancy value for the set (all same value);
4. col 4 holds the average angle for the set (all same value).

ALGORITHM:

1. Group all curve pairs whose orientation is within the threshold.
2. Calculate the average angle between pairs in each set.
3. Calculate the discrepancy in orientation for each set.

FUNCTION: *position*

SUMMARY:

- Makes subsets of data frame indices for all curve pairs which have similar position.

INPUT:

- List of data frames;
- lists of indices for all curve pairs to test (output from *orientation*);
- threshold.

OUTPUT:

- An array:
  1. col 1 holds the data frame number;
  2. col 2 holds the row number in data frame;
  3. col 3 holds the angle discrepancy for the set (all same value);
  4. col 4 holds the average angle for the set (all same value);
  5. col 5 holds the position discrepancy for the set (all same value).

ALGORITHM:

1. For each set, make subsets when pairs within the set have relative orientation within the threshold.

2. Find the average discrepancy for each subset.

FUNCTION: *combine*

SUMMARY:

- Finds the union of all list items (items are arrays) using the first 2 elements in each row, i.e. if the first 2 elements of a row are identical in two different list items, then those two list items are combined by removing the duplicate rows.

INPUT:

- List of arrays (output from *position*).

OUTPUT:

- List of arrays

ALGORITHM:

1. For each item in the list, check all other list items for matching rows (i.e. matching in the first two fields only).
2. If a matching item is found then combine the two items, remove matching rows.
3. Make a new list with the combined item and all other items from the old list.
4. Repeat above steps with the new list until the newest list has no matching items.

FILE: *run-select.r*

SUMMARY:

- The file *run-select.r* contains lines of code which reduce the size of a set of curves by removing curves which duplicate other curves in the set.

INPUT: The user is required to edit the file *run-select.r* to provide:

- A list of arrays with curvature values and a list of arrays of curvature locations for a set of curves (i.e. output of *run-curv*);

- values for thresholds: *thr.mssq* (maximum allowed values for *mssqd*); *thr.d* (maximum allowed variation in distance between end points of two similar curves); *thr.l* (maximum variation in length of curve allowed).

OUTPUT: The following R objects are produced by this code:

- *unique.kv*: a list of arrays with curvature values for the reduced set of curves;
- *unique.xy*:a list of arrays of curvature locations for the reduced set of curves.

ALGORITHM:

1. Create new lists to hold the unique curves from the original list of curves.
2. Repeatedly call the function *reduce.set* until the original list has been depleted of curves either by moving each curve to the *unique* list or by removal of each curve because it duplicates curves in the *unique* list.
3. Print the number of curves in the *unique* set.
4. Plot the curves in the *unique* set.

FUNCTION: *reduce.set*

SUMMARY:

- Finds the longest curve in the original list, transfers it to the *unique* curve list. Searches the original list for any duplicate sections of curve and removes them.

INPUT: A list containing 4 sub-lists:

- the old list of signed curvature values (*old.kv*);
- the old list of locations of curvature values (*old.xy*);
- the new list of signed curvature values (*unique.kv*);
- the new list of locations (*unique.xy*);
- the loop number (i.e. number of times this function has been called).

OUTPUT:

- Updated version of INPUT list of 4 sub-lists.

ALGORITHM:

1. Find the longest curve in the old list;
2. Add the longest curve to the *unique* list, remove from the old list.
3. Find sections of curves in the old list which have *mssqd* within the threshold of the longest curve.
4. Check that the location of sections of curves is within the threshold of the matching section of the longest curve.
5. Remove the matching sections of curves from the old list.
6. Reverse the longest curve and repeat steps 3-5.

Bibliography

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle, *in* B. N. Petrov & F. Csaki (eds), *Proceedings of 2nd International Symposium on Information Theory*, Akadémiai Kiadó, Budapest, pp. 267–281.

Akaike, H. (1974). A new look at the statistical model identification, *IEEE Transactions on Automatic Control* **AC-19**(6): 716–723.

Alder, M. (2001). *An Introduction to Pattern Recognition*, HeavenForBooks.com.

Alder, M., deSilva, C. & Attikiouzel, Y. (1990). Automatic knowledge acquisition, *Technical Report TR90-14*, The Centre for Intelligent Information Processing Systems, Department of Electrical and Electronic Engineering, The University of Western Australia.

Banfield, J. D. & Raftery, A. E. (1993). Model-based Gaussian and non-Gaussian clustering, *Biometrics* **49**(3): 803–821.

Bensmail, H. & Celeux, G. (1996). Regularized Gaussian discriminant analysis through eigenvalue decomposition, *Journal of the American Statistical Association* **91**(436): 1743–1748.

Bezdek, J. C., Reichherzer, T. R., Lim, G. S. & Attikiouzel, Y. (1998). Multiple-prototype classifier design, *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews* **28**(1): 67–79.

Blakemore, C. (1974). Developmental factors in the formation of feature extracting neurons, *in* F. Schmitt & F. Worden (eds), *The neurosciences: Third study program*, Cambridge, MA: MIT Press, pp. 105–113.

Blakemore, C. & Campbell, F. W. (1969). On the existence of neurones in the human visual system selectively sensitive to the orientation and size of retinal images, *Journal of Physiology* **203**: 237–260.

Boyle, R. A. (1983). On the convergence of the EM algorithm, *Journal of the Royal Statistical Society B* **45**(1): 47–55.

Bradley, P. S. & Fayyad, U. (1998). Refining initial points for k-means clustering, *Proceedings of the 15th International Conference on Machine Learning*, Morgan Kaufmann, pp. 91–99.

Branson, K., Rabaud, V. & Belongie, S. (2003). Three brown mice, see how they run, *Proceedings of the Joint IEEE International Workshop on VS-PETS, Nice, France*, pp. 78–85.

Canny, J. (1986). A computational approach to edge detection, *IEEE Transactions on pattern analysis and machine intelligence* **8**(6): 679–698.

Dempster, A. P., Laird, N. M. & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society* **B39**: 1–38.

Dubes, R. C. (1987). How many clusters are best? – an experiment, *Pattern Recognition* **20**(6): 645–663.

Dubes, R. C. (1993). Cluster analysis and related issues, *in* C. Chen, L. Pau & P. Wang (eds), *Handbook of Pattern Recognition and Computer Vision*, World Scientific Publishing Company, pp. 3–32.

Duda, R. O. & Hart, P. E. (1973). *Pattern Classification and Scene Analysis*, John Wiley & Sons.

Evans, F. H. (2003). Detecting fish in underwater video using the EM algorithm, *Proceedings of the IEEE International Conference on Image Processing, Barcelona*, pp. 1029–1032.

Evans, F. H., Alder, M. D. & deSilva, C. J. S. (2003). Determining the number of clusters in a mixture by iterative model space refinement - with application to free-swimming fish detection, *Proceedings of Digital Image Computing - Techniques and Applications, Sydney*, pp. 79–88.

Ewert, J.-P. (1980). *Neuroethology*, Springer-Verlag, Berlin, Heidelberg, New York.

Figueiredo, M. A. T. & Jain, A. K. (2002). Unsupervised learning of finite mixture models, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(3): 381–396.

Figueiredo, M., Leitao, J. & Jain, A. K. (1999). On fitting mixture models, *in* E. Hancock & M. Pellillo (eds), *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Springer-Verlag, pp. 54–69.

Fraley, C. & Raftery, A. (2000a). MCLUST: Software for model-based clustering, density estimation and discriminant analysis, *Technical Report 415*, University of Washington, Department of Statistics.

Fraley, C. & Raftery, A. (2000b). Model-based clustering, discriminant analysis, and density estimation, *Technical Report 380*, University of Washington, Department of Statistics.

Fraley, C. & Raftery, A. E. (1998). How many clusters? Which clustering method? Answers via model-based cluster analysis, *The Computer Journal* **41**(8): 578–588.

Fraley, C. & Raftery, A. E. (2002). Model-based clustering, discriminant analysis, and density estimation, *Journal of the American Statistical Association* **97**(458): 611–632.

Fu, K. (1982). *Syntactic pattern recognition and applications*, Prentice-Hall, Englewood Cliffs, N J.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition*, second edn, Academic Press, New York.

Fukunaga, K. (1993). Statistical pattern recognition, *in* C. Chen, L. Pau & P. Wang (eds), *Handbook of Pattern Recognition and Computer Vision*, World Scientific Publishing Company, pp. 33–60.

Hansen, M. H. & Yu, B. (2001). Model selection and the principle of minimum description length, *Journal of the American Statistical Association* **96**(454): 746–774.

Haralick, R. M. & Shapiro, L. G. (1992). *Computer and Robot Vision*, Vol. 1, Addison Wesley.

Hartigan, J. A. & Wong, M. A. (1979). Algorithm AS136: A k-means clustering algorithm, *Applied Statistics* **28**(1): 100–108.

Harvey, E. S., Cappo, M., Shortis, M. R., Robson, S., Buchanan, J. & Speare, P. (2003). The accuracy and precision of underwater measurements of length and maximum body depth of southern bluefin tuna (thunnus maccoyii) with a stereo-video camera system, *Fisheries Research* **63**(3): 315–326.

Harvey, E. S., Fletcher, D. & Shortis, M. (2001a). A comparison of the precision and accuracy of estimates of reef-fish lengths made by divers and a stereo-video system, *Fishery Bulletin* **99**(1): 63–71.

Harvey, E. S., Fletcher, D. & Shortis, M. (2001b). Improving the statistical power of visual length estimates of reef fish: A comparison of estimates determined visually by divers with estimates produced by a stereo-video system, *Fishery Bulletin* **99**(1): 72–80.

Harvey, E. S., Fletcher, D. & Shortis, M. (2002). Estimation of reef fish length by divers and by stereo-video. a first comparison of the accuracy and precision in the field on living fish under operational conditions, *Fisheries Research* **57**(3): 257–267.

Harvey, E. S., Shortis, M. R., Seager, J. & Robson, S. (2003). The implementation and validation of a stereo-video system for measuring the length of southern blue fin tuna during transfers, *Technical Report 48 pp*, Final report AFMA Grant R01_1299, Australian Fisheries Management Authority.

Harvey, E. S., Shortis, M. R., Stadler, M. & Cappo, M. (2002). A comparison of the accuracy and precision of digital and analogue stereo-video systems, *Marine Technology Society Journal* **36**(2): 38–49.

Hew, P. C. (1999). *Pixels to strokes to digits*, PhD thesis, University of Western Australia.

Hubel, D. H. & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *Journal of Physiology* **160**: 106–154.

Hubel, D. H. & Wiesel, T. N. (1979). Brain mechanisms and vision, *Scientific American* **241**(3): 150–162.

Huffman, D. A. (1951). A method for construction of minimum redundancy codes, *Proceedings IRE* **40**: 1098–1101.

Hurvich, C. M., Simonoff, J. S. & Tsai, C.-L. (1998). Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion, *Journal of the Royal Statistical Society, Series B* **60**(2): 271–293.

Jain, A. K. (1987). Advances in statistical pattern recognition, *in* P. Devijver & J. Kittler (eds), *Pattern Recognition Theory and Applications*, Vol. F30 of *NATO ASI Series*, Springer-Verlag, pp. 1–19.

Jain, A. K., Duin, R. P. W. & Mao, J. (2000). Statistical pattern recognition: A review, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(1): 4–37.
*citeseer.nj.nec.com/jain99statistical.html

Kandel, E. R. (1979). Small systems of neurons, *Scientific American* **241**(3): 66–76.

Lim, S. G. (1997). *Visual object shape recognition using hierarchial syntax extraction*, PhD thesis, University of Western Australia.

McKenzie, P. & Alder, M. (1994). The EM algorithm used for Gaussian mixture modelling and its initiallization, *in* E. Gelsema & L. Kanal (eds), *Pattern Recognition in Practice IV*, Elsevier Science, The Netherlands, pp. 91–105.

McLachlan, G. J. (1992). *Discriminant Analysis and Statistical Pattern Recognition*, John Wiley & Sons, New York.

McLachlan, G. J. & Basford, K. E. (1988). *Mixture Models: Inference and Applications to Clustering*, Marcel Dekker Inc., New York.

McLaughlin, R. A. (2000). *Intelligent algorithms for finding curves and surfaces in real world data*, PhD thesis, University of Western Australia.

McLaughlin, R. A. & Alder, M. D. (1998). The Hough transform versus the UpWrite, *IEEE Transactions on Pattern Recognition and Machine Intelligence* **20**(4): 396–400.

Moon, T. (1996). The expectation maximization algorithm, *Signal Processing Magazine* **13**(6): 47–60.

Naiberg, A., Petrell, R. J., Savage, C. R. & Neufeld, T. (1993). Non-invasive fish size assessment method for tanks and sea cages using stereo-video, *in* J. Wang (ed.), *Techniques for Modern Aquaculture*, American Society of Agricultural Engineers, pp. 372–381.

Petrell, R. J., Shi, X., Ward, R. K., Naiberg, A. & Savage, C. R. (1997). Determining fish size and swimming speed in cages and tanks using simple video techniques, *Aquacultural Engineering* **16**(1,2): 63–84.

Redner, R. A. & Walker, H. F. (1984). Mixture densities, maximum likelihood and the EM algorithm, *SIAM Review* **26**(2): 195–239.

Ripley, B. D. (1979). Tests of randomness for spatial point patterns, *Journal of the Royal Statistical Society B* **41**(3): 368–374.

Rissanen, J. (1987). Stochastic complexity, *Journal of the Royal Statistical Society* **49**(3): 223–239.

Rissanen, J. J. (1989). *Stochastic Complexity in Statistical Inquiry*, World Scientific, Singapore.

Rutter, J. W. (2000). *Geometry of Curves*, Chapman & Hall/CRC New York, chapter 7.

Sakamoto, Y., Ishiguro, M. & Kitagawa, G. (1986). *Akaike Information Criterion Statistics*, KTK Scientific Publishers, Tokyo.

Sapiro, G. (2001). *Geometric partial differential equations and image analysis*, Cambridge University Press.

Savage, C. R., Petrell, R. J. & Neufeld, T. P. (1994). Underwater fish-video images: Image quality and edge detection techniques, *Canadian Agricultural Engineering* **36**(3): 175–183.

Schwarz, G. (1978). Estimating the dimension of a model, *The Annals of Statistics* **6**(2): 461–464.

Shannon, C. E. & Weaver, W. (1963). *Mathematical Theory of Communication*, University of Illinois Press.

Shieh, A. C. R. & Petrell, R. J. (1998). Measurement of fish size in atlantic salmon cages using stereographic video techniques, *Aquacultural Engineering* **17**(1): 29–43.

Storbeck, F. & Daan, B. (2001). Fish species recognition using computer vision and a neural network, *Fisheries Research* **51**(1): 11–15.

Strachan, N. J. C. (1993). Recognition of fish species by colour and shape, *Image and Vision Computing* **11**(1): 2–10.

Theodoridis, S. & Koutroumbas, K. (1999). *Pattern Recognition*, Academic Press.

Thönnes, E. & van Lieshout, M. (1999). A comparative study on the power of van Lieshout and Baddeley's J-function, *Biometrical Journal* **41**: 721– 734.

Tillet, R., McFarlane, N. & Lines, J. (2000). Estimating dimensions of free-swimming fish using 3D point distribution models, *Computer Vision and Image Understanding* **79**: 123–141.

Titterington, D. M., Smith, A. F. M. & Makov, U. E. (1985). *Statistical Analysis of Finite Mixture Distributions*, John Wiley & Sons, San Diego.

Wallace, C. & Dowe, D. (1999). Minimum message length and kolmogorov complexity, *Computer Journal* **42**(4): 270–283.

Williams, P. S. (1999). *The automatic hierarchical decomposition of images into sub-images for use in image recognition and classification*, PhD thesis, University of Western Australia.

Wu, C. (1983). On the convergence properties of the EM algorithm, *Annals of Statistics* **11**(1): 95–103.

Zhang, J. & Modestino, J. W. (1990). A model fitting approach to cluster validation with application to stochastic model-based image segmentation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(10): 1009–1017.

Zheng, L., Chan, A., McCord, G., Wu, S. & Liu, J. (1999). Detection of carcinoma masses for screening mammography using DWT based multiresolution Markov random field, *16th Symposium for Computer Applications in Radiology (SCAR 99)*.

Zion, B., Shklyar, A. & Karplus, I. (1999). Sorting fish by computer vision, *Computers and Electronics in Agriculture* **23**(3): 175–187.