# Specific Object Recognition Using Iso-Luminal Contours

A thesis presented in fulfilment of the requirements for the degree of

**Master in Engineering**

in

**Mechatronics**

at Massey University,

Palmerston North, New Zealand.

**John William Howarth**

**2013**

## 1.1. Abstract

" Specific Object Recognition Using Iso-Luminal Contours "

Object recognition is a broad topic in the study of computer vision. In this case the task of distinguishing between specific instances of various objects is addressed. The ability to perform this task would allow robots to operate in unstructured environments, allowing greater and more efficient automation of many tasks. Techniques currently proposed tend to have low accuracy rates, high processing time, or both. This research seeks to establish a method that can quickly and accurately find instances of objects within a scene.

Iso-luminal contours were used to gather the initial data, from which higher level features were extracted. Basic geometric features were used as the intermediate data, consisting of lines, arcs, and lobes (a custom type suited to describe corners). The high level data was a custom type, called blocks; each block contains a few features and describes the spatial relationships between them. The features and blocks are designed to be spatially invariant, so the blocks are directly compared to determine which objects are in a scene.

The objectives of this research were not met. The results show the geometric features were not robust to changes in image sets, although they did work well with the image set they were developed with. Unfortunately this means the performance of the subsequent 'block' related steps cannot be established. Most of the work was focussed on this aspect. Future work would entail increasing the robustness of the features part of the algorithm, and then gauging if the block based research is of practical use.

It is thought that the research results were poor because feature extraction was poor. It is further thought that the high level analysis has merit.

## *1.2. Acknowledgements*

I would like to thank those who made significant contributions to this work:

My supervisors Rory Flemmer and Huub Bakker.

Funding contributions from the Todd Foundation and Sir Alan Stewart Postgraduate scholarships.

## 1.3. List of Figures

## 1.4. List of Tables

## 1.5. *Table of Contents*

## 2. Introduction

Artificial vision refers to the capturing of image data using cameras and analysing it using computer algorithms in order to derive a higher level of data than that offered by the raw pixels. The field has, over the last fifty years, been successful at providing detailed answers to precise and well-defined questions. Examples are metrology of well-defined outlines and determining attributes of structures in the images – such as face recognition. This success seems to be tied to analysing a well-defined situation. For the general case where an unstructured image is analysed, results have been less pleasing. This latter case has been of interest to academic researchers as they have sought to advance the field.

The development of embodied artificial intelligence makes more stringent demands upon the field because biomimetic robots need vision. They need to operate in an unstructured environment and they need to be able to derive information from the images which they capture in this environment. This presents the requirement that a computer algorithm should be able to process an image and report the objects in the image and their orientation – even if they are partly obscured.

This problem was initially approached (R O Duda & P E Hart, 1972) (Shantz, 1981) with the idea that determination of the edges would allow us to produce cartoons delineating objects which could then be recognized from geometric relationships. Capabilities have improved over the years but current edge-followers do not yet permit this process to be successful. Many other approaches have been attempted including scale invariant feature transforms (SIFTs) but a mature methodology to recognise and orient objects in images is not yet available.

In 2005, Flemmer and Bakker proposed a technique which did not set out to find edges nor to attempt to cartoon objects. They determined a large number of isoluminal contours in images and then extracted, from this large number, those contours which had significance. The criterion for significance was that enough contours at different light levels had to be coincident. The loci delineated by these contours are similar, but not identical, to what is traditionally viewed as an edge. The difference is that isoluminal contours will trace out, sequentially, those points of constant brightness that lie on a significant luminal gradient. Further, the process leads to an ordering of the points so that the edges tend to be contained in these contours even though a sharpest gradient follower would not draw them out. It was pointed out (Flemmer & Bakker, 2005) that when these contours were plotted as the local second derivative versus arc length, the resultant curves were rotation invariant. It was suggested that, if the data in these curves could be coded in terms of the features present, it might be possible to recognise objects purely on the basis of this coding.

This work considers the question, "How shall we use such curves, derived by any means, to recognise objects?" Here we do not care by what means the curves were developed. We will plot them as second derivative versus arc length and attempt to use this data to recognise objects. Flemmer and Bakker considered a specific problem that is of importance in developing a biomimetic robot and this work considers that same problem. The robot needs to be able to find any object in its world and then to be able, subsequently, to recognise that identical object. When it encounters the object, it will not recognise it the first time but will spend time rotating it until it has learned it from all angles. Flemmer and Bakker proposed that twenty equally spaced views of the object would provide enough information and therefore every rigid object could be described by the isoluminal contours derived for each of these twenty views; interpolation was possible between views.

This analysis does not assume the ability to deal with universals. It is assumed that all mugs are different and the robot only knows that particular mug which it has examined. This is a very restricted version of the general problem of recognising objects in images.

This work considers the specific problem that we have a database of known views of definite objects and that we have applied a protocol which yields a set of isoluminal contours from each view of each object. We imagine that our database might contain 300,000 such views. Our problem is that, presented with a new image, we run the protocol, acquire the isoluminal contours of this image and then ask whether any of the database objects is present. This problem has been considered in a very simple form where template matches or a few SIFT's are applied but we are conscious that any solution must be capable of managing the data problem so that the judgement is made within a few seconds.

In this work, we will consider improving the protocols of Flemmer and Bakker in terms of speed and also quality of derived isolumes. We will then explore ways in which these isolumes can be used to recognize specific views of specific objects. We will not use one of the usual databases because our need is not to seek a class of objects such as cars but rather to seek a specific orientation of a specific object. The most appropriate visual database would be what the robot’s eyes recorded over a period of time as it went about its business. Failing that, we work from the database of Flemmer and Bakker which offers 100 images, some of which contain views, either frank or partially occluded, of two “target” objects.

# 3. Literature Review

The literature review begins with a brief history of the field of object recognition, an overview of the current state-of-the-art systems, and an overview of the human vision system.

The review focuses on literature and systems similar to the proposed approach, which builds on that of Flemmer and Bakker 2005. The proposed method propagates information from grey level contours, through to the more complex relationships between the contour features. Various approaches and comparisons at each step are reviewed, as well as a general review of the most successful or notable systems in the field.

A "conclusions" section summarises the findings of the reviewed literature and provides insights relevant to the thesis topic.

## *3.1. Research History*

Object-recognition using computer vision has been subject to much research over the years. Articles on the subject have been published prior to the 1980s (Uhr, 1972). Today there are many examples of working object-recognition systems used in many different industries. In the 1980s the available computing power severely limited any significant work; many complex algorithms are implemented today due to the 100,000-fold (Ross, 2008) increase in computing power since then. And yet, despite this huge increase there are still no realistic implementations of a generalised, object-recognition system. The use of an object's curves for comparison with those in a database has always been considered, with edge detectors conceived around 1968 (Sobel & Feldman, 1968), level sets becoming common place in the early 90's (Mark, Peter, & Stanley, 1994), and isolumes in 2005 (Flemmer & Bakker, 2005).

## *3.2. Overview of the Current State of the Art*

Most existing object-recognition algorithms cannot work with large object databases (Pinto, Cox, & DiCarlo, 2008) (Tistarelli, 1995); either the computation time increases, or the recognition rate decreases. Today the majority of object-recognition systems are designed for specialised applications, where a small range of similar scenes are involved with a correspondingly small object database. Many different algorithms have been used with varying degrees of success; neural networks, object shape features, colour, texture, shape, and edge curves. Currently there does not appear to be any computer object-recognition system that achieves performance comparable to a human.

Development of video-based object-recognition has been made possible by advances in computer processing power; currently techniques take around 40-100ms per frame (Laika & Stechele, 2007). Providing cues to assist automobile drivers is one area where there has been significant progress (ibid.). This, of course has the advantage of a less generic scope (ibid.).

Systems have been designed based on an object's edge curves (see Edge Curve Generation Methods section). Some of these achieve high accuracy rates, but none seems to be able to scale up to large object libraries. Object edge curves can be generated in many different ways. The system used seems to depend on the application at hand, so it is not thought that one system is 'better' than the rest.

## *3.3. Analysis of the Human Vision System*

Human vision is a complex system able to identify objects from within a scene quickly and efficiently. Although it does not use object edge curves exclusively we can still gain insight from an analysis of the algorithms used in the human vision system.

### 3.3.1. Image Acquisition

The human eye operates in much the same way as a standard CMOS or CCD camera—a lens forms an image onto a mosaic of light sensitive receptors (Gregory, 1978, pg. 26)—with the difference being that the sensors are non-uniformly distributed. The photoreceptor cells are considered to be in two groups, rods and cones. The cones are receptive to the red, green, and blue values in the image, while the rods are mainly used for vision in low light situations (Bowmaker & Dartnall, 1980).

### 3.3.2. Information Propagation

In order to gain better insight into the brain's vision algorithm we will evaluate the system from an information handling perspective. Although the information propagates through different areas of the retina, cortex, and brain (Morito, Tanabe, Kochiyama, & Sadato, 2009), the interactions between these areas can still be thought of in computer algorithm terms.

This vision pathway is categorised into five groups, the V1, V2, V3, V4, and V5 cells (Nolte, 2002, pg. 551), also known as visual cortical areas. V1 cells respond to visual input from only a tiny part of the retina, and each cell is tuned for a specific input patterns, such as a line at a specific angle (Gregory, 1978, pg. 26) (Over, 1972, pg. 7).

The eye performs saccades many times per second, which shifts the field of view to different parts of the viewed scene. These saccades completely alter what each V1 cell is 'seeing', and therefore each cell's output. So while cells may be tuned to very specific shapes; a half circle at a certain radius, or a certain type of texture, these saccades enable each part of the viewed scene to be thoroughly examined (Vallines & Greenlee, 2006).

The information is processed through the five cell groups in a hierarchical manner (R. J. Watt & Phillips, 2000). The exact nature of the network between the five cell groups is not yet known, but the following valuable insights have been gained. There are both feed-forward and feed-back connections, present within the hierarchy (Livingstone & Hubel, 1988). The cells can pass information across multiple visual cortical area levels (Hawkins & Blakeslee, 2004, pg. 113). The higher levels contain cells sensitive to higher order interactions such as complex shapes and motion (Born & Bradley, 2005). Once the 'image' has reached the high order cells it is no longer affected by the eye's saccades (Hawkins & Blakeslee, 2004, pg. 120).

The conditions by which the cells are triggered vary from person to person, suggesting that the network is learned and is able to change, more effectively to process the person's environment (Damaraju, Huang, Barrett, & Pessoa, 2009).

In summary, there are a number of features of the human vision system that can be applied to a computer vision system algorithm.

- A number of simple features are used, these can be localised anywhere in the viewed image.
- A hierarchical structure with feed-back and feed-forward connections is used to represent complex arrangements of the simple features.
- The hierarchical structure is learned, and is dictated, by the objects seen and known.

## *3.4. Object Recognition Systems Overview*

### 3.4.1. Scale Invariant Feature Transforms

The Scale-Invariant-Feature-Transform method was developed by Lowe (Lowe, 1999) who claimed it can achieve robust object-recognition within cluttered and partially occluded images, all with a computation time of under two seconds. The features used are scale- and rotation-invariant, and partially invariant to change in illumination and camera viewpoint (Lowe, 2004). A large collection of local feature vectors, around 1000 per image, are used to provide the high level of discrimination. The vector locations are obtained by selecting the maxima and minima of a difference-of-Gaussian function, taken at different scales over the image (Lowe, 1999). The vector data is calculated from the magnitude and direction of the image's local gradient. It is combined with the location to make a scale-invariant-feature-transform-key, or a *SIFT key* (ibid). A nearest-neighbour approach is applied when comparing the *SIFT keys* with an object database. Database objects that contain high numbers of matched keys—that agree on the objects location and pose—are considered to be a match (ibid).

Each *SIFT key* must be compared to all those within the library (Lowe, 2004), requiring very large numbers of comparisons for large object databases, causing the system to run slower, and the accuracy to decrease (ibid). Both of these parameters would become unacceptable in this research and so we do not use *SIFT keys* here.

### 3.4.2. Content-based Image Retrieval

Content-based image retrieval (CBIR) is the term given to a system capable of retrieving or organising digital picture archives by their content (Ritendra, Dhiraj, Jia, & James, 2008). CBIR utilises techniques from several disciplines, such as; computer vision, machine learning, database systems, etc.

The architecture for image retrieval contains three steps, feature transformation, feature representation, and a similarity function (Vasconcelos & Lippman, 2000).

This type of system generally deals with far larger image sets than the other algorithms listed in this review. Due to the associated computing constraints, simple and quick functions are used to generate the feature data, such as dominant colour, homogenous texture, and edge histogram descriptors (Till, Ullrich, nich, Lars, & Manjunath, 2004). The simplicity of the features used, makes this type of system unsuitable to identify which specific objects are present within an image.

### 3.4.3. Context-Based Object-Class Recognition and Retrieval by Generalised Correlograms

Jaume, Nicu et al (Jaume, Nicu, & Petia, 2007) propose a method to retrieve images with matching object classes by using Generalised Correlograms (GCs) in the matching process. A GC is associated with one local part of the image, but also contains the spatial relationships between it and other parts. This method uses local parts to describe the image contents as it provides greater robustness against clutter and partial occlusions than global representations (ibid). Spatial relationships between parts are added to give a significant increase in distinction.

A search object is generated from multiple images where the object is highlighted by the user. This learning step allows the system to decide what GCs are most effective to use when searching other images.

The system has been tested against the COIL-100 database (Nene, Nayar, & Murase, 1996) and the Caltech database (Fei-Fei, Fergus, & Perona, 2004), as well as other images to provide more insight. Results against the Caltech database have accuracy rates of 84-98% for different classes. The COIL-Database was used to examine the effects of scaling and rotation—both slightly decrease the accuracy—with a scale factor of 0.5 reducing it from 100 to 93.2%, and a rotation change of 20 degrees causing a reduction from 100 to 85.3%. Additional images were added to some categories in the Caltech database, to ensure the background was not being learnt instead of the object itself. Background dependence was proven when the ROC equal error rate for the planes category jumped from 62.2% to 87.6% when images with a sky background that didn't contain a plane.

The design, and accuracy, of this method are suited to class matching; which is to put the objects into broad categories like planes, rather than specific instances such as a Boeing 747. Parts of the algorithm may also be suitable for identifying and locating specific objects, but this does not appear to have been tested.

### 3.4.4. Weakly Supervised and Unsupervised Category Learning

Many object-recognition systems make use of sets of learning images, containing pictures of the object, or object category, to be identified. Using multiple images of the same object or class allows the algorithm to decide which parts of the image are most distinctive to that object or class.

Fully supervised learning requires time-consuming labelling of every image; weakly supervised and unsupervised learning requires no categorising of the original images (Fergus, 2003). However, weakly supervised learning does require identifying only those images which contain the object. Unsupervised learning can be done by creating clusters to represent different objects or categories, and refining these clusters based on their fit (Lee & Grauman, 2009). While this is computationally expensive, it is performed while learning the objects so it only needs to be performed once.

Lee and Grauman apply an unsupervised learning method to the Caltech-101 and Microsoft Research Cambridge v1 (MSRC-v1) datasets. Semi local descriptors are extracted from the image, compared with those from other images and given a weighting. Transformations are applied and iterative clustering performed to obtain image clusters containing objects of the same class. Results were 79% pure (the portion of a cluster containing the dominant class) for 7 classes and 66% for 20 classes using the Caltech 101 database. For the Caltech-4 database the results were up to 91%.

Some of the theory behind weakly supervised, class learning could probably be applied to our system, to ensure the best object representation is used in the database.

## 3.5. Edge detection

An object's curves are often used in differentiating between different objects or classes. Common types of curves are constant grey level contours, or boundary/edge curves. The data gathered in each case is of one of two similar types; discrete X and Y co-ordinates, or a function which fits to such points, in each case the data delineates part of the object.

### 3.5.1. General Edge Detectors

Edge detection algorithms are used for many computer-vision applications such as, object detection (Vicente, Munoz, Molina, & Galilea, 2009), medical imaging (Suri, Singh, & Reden, 2002), and pattern recognition (Rosin, 2009), to name a few.

A common approach to edge detection is to take a greyscale image, apply a smoothing filter to it and then take the spatial derivative with respect to both axes. The maximum gradients obtained are associated with edges (Pinho & Almeida, 1997). Accurate localisation can be achieved by applying a second order differentiation and locating the zero crossing associated with significant edges (ibid.), although there are obviously issues with noise.

The most widely used smoothing filter for edge detection is the Gaussian filter (Basu, 2002), which is an orientation-independent filter that is easy to scale and apply to discrete values. A discrete, differentiation operator is often used to find the gradient at each point; a number of these are available, such as the *Sobel*, *Robert's Cross*, and *Prewitt's*. The *Canny* edge detector is similar but uses four filters to generate the gradient at specific points as well as hysteresis thresholding and non-maximum suppression (Tony Lindeberg, 1998). Hysteresis thresholding reduces the gradient threshold when an edge is found nearby, allowing complete edges to be found even in noisy images (ibid.). Non-maximum suppression filters the edge points to include only those with local maximum gradients; the filtering is performed normal to the edge to ensure a single-pixel, continuous line is found (ibid.).

Edge detection algorithms often process the image at multiple scales to ensure the edge points found are true edge points and not caused by other image factors (Basu, 2002). The *Marr-Hildreth* operator is one such and is also orientation independent (ibid.).

A common characteristic of all these edge detectors is that they produce fractured shapes and curves. Some list only the individual edge pixels, while others group the edge pixels into straight lines, while ignoring curves entirely. With the intent of our research being to deal with the shapes of complete contours, much extra processing would be required to extract useful information from the output of these edge detectors. An algorithm such as the Hough transform (pg 24) would be suitable to arrange the points into simple lines and arcs, however extracting full shape curves would be much more difficult.

### 3.5.1. Level Sets

A level set is defined as a set of points that share the same function value (Osher & Fedkiw, 2003, pg. 5). For a function of two variables the members of the set will lie as curves (or contours) on a level plane (ibid., pg. 87). One of the features of a level set is that they can describe quite complex two-dimensional curves, including disjoint curves, using simple equations.

The level set algorithm can be applied to many different situations, for instance articles have been published in the fields of; computer vision, mathematics, chemistry, and biology. Some of these include, Karantzalos and Argialas' (2009) use of level sets in analysing aerial images, Brandmans (2008) computation of eigenvalues of elliptic operators on compact hyper-surfaces, Alagona, Ghio et al's (1999) detection of synthetic, dipeptide orientations, and Can, Chen et al's (2006) simulation of molecular interactions.

Image processing is concerned with two-variable functions of pixel position, x and y, that yield level curves or contours of constant brightness. An example of such an algorithm (Sethian, 1996, pg. 8) examines the image for multiple intensity levels; at each level the image is separated into intensities greater and less than the specified intensity. The points on the boundary curve are defined as an 'explicit interface'. This initial stage gives a set of regions bounded by these explicit interface curves. This can be used for image segmentation to give the outline of the object contained within an image. There are many implementations of this method, namely; (Cremers, Rousson, & Deriche, 2007), (Sethian, 1996, pg. 219), (Wang, Song, Tan, & Wang, 2009), and (Osher & Fedkiw, 2003, pg. 119). Methods such as *fast marching* and *active contours* use a multiple-iteration approach to obtain the best-fit boundary for the object outline.

The segmentation of an image using level sets is a powerful tool in many applications, however segmentation is not the goal in this application, as the object's internal curves should be recorded and considered when matching objects. The explicit interface curves will contain the internal curve information, along with superfluous curves where there is an insignificant gradient. Extracting useful curves from the explicit interface data would require an additional algorithm, but would not require a segmentation step, making level sets of limited use.

### 3.5.2. Phase Congruency

Phase congruency edge detection finds the image edges by examining the frequency domain. Local frequency information for the image points can be calculated using wavelets (Peter Kovesi, 1999). The Fourier components obtained are maximally in phase where an edge occurs (P. Kovesi, 2003). Because the phases are examined it is a magnitude independent process (ibid). The wavelet calculation only provides frequency information for one direction/dimension, so it must be calculated and evaluated at multiple orientations. It is also beneficial to use multiple wavelet scales to calculate the Fourier components (Peter Kovesi, 1999).

The phase congruency method can be used to give reliable identification of edge points within an image. However the many calculations required for each point makes this algorithm unattractive for our purpose.

### 3.5.3. Iso-luminal Contours

An *Iso-luminal Contour,* or *isolume,* is a curve formed by a collection of points that follow the same brightness within a greyscale image. The iso-luminal contour algorithm is a recent development and has not yet been implemented in many different applications (Flemmer & Bakker, 2005).

The image is viewed as a cartographic map, with the intensity being represented by altitude (Flemmer & Bakker, 2009). The iso-luminal contours are obtained by following a specified grey level on the surface until it completes a closed curve, or the gradient fades away (Figure 1) (Figure 2). This method generates open- and closed-loop contours. Interpolation between the pixels is performed to produce contours with sub-pixel resolution. The grey levels to be examined are chosen to provide significant redundancy on the object edges. The algorithm combines contours that are overlapped by larger ones, and records the extent of redundancy for those remaining. Contour winnowing can thus easily be performed and those without sufficient redundancy removed. The processing of the contours reduces the size of the data and ensures that the majority of significant edges are found, with relatively few false edges (Flemmer & Bakker, 2005).

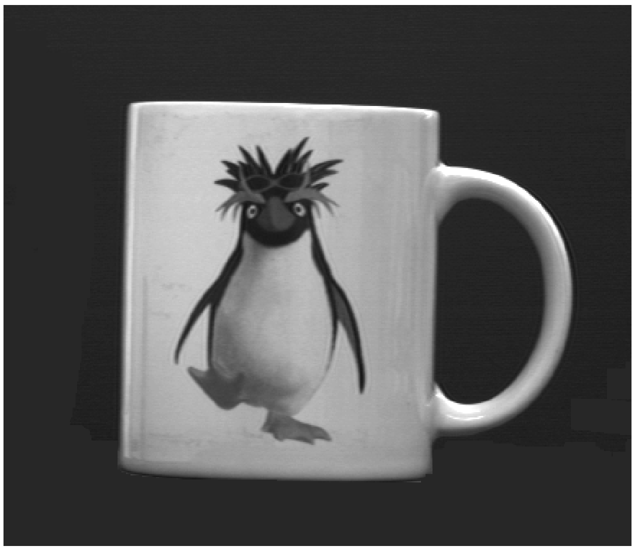


**Figure 1. Greyscale image of the mug**

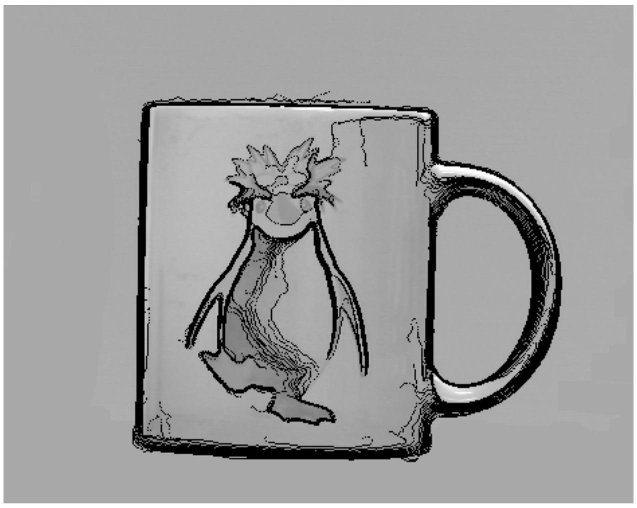


**Figure 2. Isolumes on the mug**

The isolumes are represented by a set of points, giving the rate of curvature (second spatial derivative of the contour). The ensuing graph (Figure 3) is known as the isolume's fingerprint. The fingerprint is represented by the inverse of the local radius at any point on the isolumes. The radius is calculated by fitting an arc to the selected point and its neighbours. The shape of the fingerprint can be considered scale- and rotation-independent (Bakker & Flemmer, 2009), although with larger contours there will be more points. The fingerprint technique could be adapted and applied to other algorithms with two dimensional curve data, such as level sets

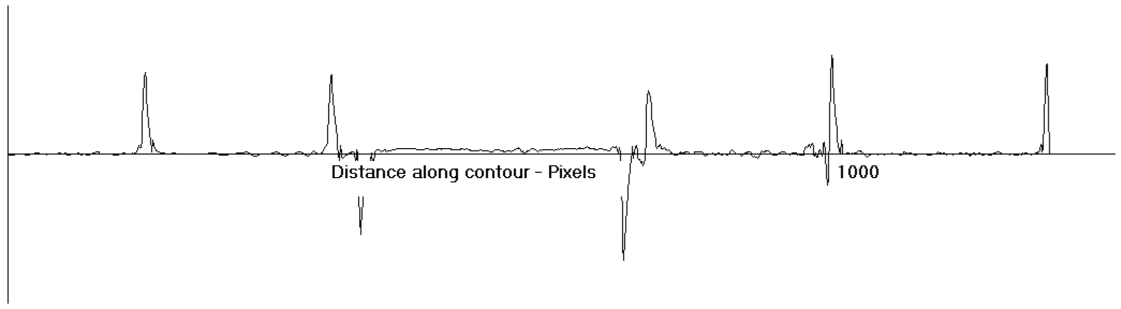


**Figure 3. Fingerprint plot for the mug's outline**

The algorithm will provide iso-luminal contours in the form of a set of ordered points with their corresponding fingerprint. These isolumes are most often found on the major grey level gradients within an image, thus representing the significant object edges.

Isolumes are also found on an object's interior features, such as a colour change, or surface orientation change. Using the isolume's fingerprints to extract the required features will be simple and efficient. We have chosen this method to provide the required curves for the research.

## *3.6. Object Curve Comparison Techniques*

An object's edge curves go by many different names in the literature; contours, isolumes, boundary curves, and snakes, but they all represent a set of x and y points that lie on an object's edge.

### 3.6.1. Simple Contour Description Methods

There are numerous ways to describe and compare the various attributes of a contour; those outlined below provide a simple measure of similarity.

- The segment length ratio method splits the contour at all the points of high curvature; these new lengths are split again at the zero curvature crossings. These small segments can be easily characterised with respect to their parent length in order to describe the contour, using parameters such as the length, or the angle of each segment (R. Watt, 1991, pg. 112).

- The medial axis process combines a number of contours, and provides a contour that lies between the originals (Shantz, 1981; R. Watt, 1991, pg. 112). This method is useful, in applications such as text recognition, for taking interior and exterior outlines and converting them to a single edge.

- Contour centroids are calculated by taking an average of the all the x and y points on the contour. When this is done for multiple contours the distribution of these centroids can be used to describe and compare an image in a very compact form (Cheng & Yan, 1998; R. Watt, 1991, pg. 112).

- A measure of a contour's symmetry can provide a simple description for easy comparison with those in a database. It describes how symmetrical a contour is—for both rotational and mirror symmetries—and provides an axis or centre of symmetry where appropriate (Bigun, 1988; Marola, 1989; Zabrodsky, Peleg, & Avnir, 1992).

### 3.6.2. Contour Distance Transform

The contour distance transform calculates an array the same size and dimension as the image, each value represents the distance to the nearest contour. When using this, it is preferable initially to divide the image into probable object sections (Kassahun & Sommer, 2004), which enhances the effectiveness of the algorithm. There are a number of techniques available for image segmentation including; stereopsis, colour matching and level set-based segmentation.

Contours are then extracted from the sub-images, and the minimum distance transform is generated for each sub-image. Each pixel position is given a corresponding number representing the minimum distance to a contour (Huang & Mitchell, 1994; Ragnemalm, 1992). This method provides a direct comparison of two contours that can be found in different images, used in situations where the two contours are likely to be very similar. It also provides accurate contour localisation (Kozinska, Tretiak, Nissanov, & Ozturk, 1997).

It is expected that using this approach for object identification would not be robust with respect to occlusions, partially because of the reduced contour information, and partially because the scene-division algorithms lose effectiveness when occlusions occur.

### 3.6.3. Contour Segment Networks

Contour segment networks essentially break up and recombine the original contours.

One of the challenges with using grey level contours is that the edges are not reliably extracted from complex images that contain a lot of clutter. A contour may pass through multiple objects if the grey level is shared by them (Choi, Lam, & Siu, 2001). Breaking the contours into segments gives a much better representation of the scene and the underlying contours (Wong, Yuen, & Tong, 1998).

It is broken into straight lines wherever possible, simplifying the rest of the algorithm. From the multitude of lines, networks are constructed by adding lines that have approximately co-incident end points, or are tangentially co-incident over a discontinuity. This essentially reforms the contours in a crude representation of the originals (Ferrari, Tuytelaars, & Gool, 2006). This deconstruction and reconstruction of the contours provides a more robust representation of a scene. Although the original contours might not fully encompass an object, at least one contour network will (ibid). A comparison is then performed on the relative line lengths and orientations with those in the database. The system achieves an 82% recognition rate with a seven object database (ibid).

The process generates a large amount of data that must be compared with a large database, therefore it is not expected that the computation time will scale well.

### 3.6.4. Cross-Correlation

Cross-correlation is widely used as a measure of similarity in matching tasks (Zhao, Huang, & Gao, 2006). A contour can be represented as a binary image and correlated in the same way as a normalised image.

Cross-correlation is performed by comparing the pixel values when the images are overlaid on one-another; different positions are tried and a measure of accuracy generated for each. Normalised cross-correlation techniques are very fast and accurate for images without large orientation changes, such as adding computer-generated special effects in movies (Lewis, 1995), or facial recognition programs (Brunelli & Poggio, 1993). Once a scale, rotation, or viewpoint change is introduced, correlating the system becomes more complex and inefficient (Paul, Pascal, & Andrew, 1992).

For object-recognition the changing rotation, scale and viewpoint must be taken into account. The standard, normalised, cross-correlation method can overcome the problem of rotation and scale (Zhao et al., 2006), though it is usually done by transforming the image and re-correlating, which is a time consuming process. There is a modified, normalised, cross-correlation method that is resilient to changes of rotation, scale and viewpoint, written by Zhao, Huang & Gao. Each point is assigned a vector as well as co-ordinates; the vector is determined by some parameter of the point itself. By averaging the vectors a dominant direction and scale can be obtained. The points of one image can then be translated to 'calibrate' it to the other image. With both images now in the same orientation and scale, normalised cross-correlation can be performed.

Because the correlation must be performed with each pair of objects, it would be computationally expensive to apply this method to the full object database. This makes the approach unsuitable for general object-recognition.

### 3.6.5. Curvature Scale Space

Curvature scale space (CSS) is a form of representing a two-dimensional shape or outline in a form that is rotation and scale invariant (Mokhtarian & Mackworth, 1986). It is currently one of the most popular multi-scale curvature representations for 2D curves (Costa & Cesar, 2009, pg 480).

The representation is a function showing curvature of the outline along its length. It is generated by convolving the x and y functions with a one-dimensional Gaussian kernel. A number of kernels are used with differing widths to provide information at multiple detail levels, all of which are superimposed on a single CSS plot (Mokhtarian & Mackworth, 1986). When presented with a set of discrete data points, such as edge points found within an image, a discrete, Gaussian kernel can be calculated and used, which is preferable to a sampled kernel (T. Lindeberg, 1990).

Various methods to compare the generated CSS plots have been implemented. The commonly used contour maxima method compares the points at which there is a local curvature maximum. The positions and levels of curvature are used to match the images (Yue, 2007).

An Eigen-CSS method recently developed by Drew et al represents the CSS plot in eigen space. This is done by first summing curvature data at x values along the plot, as a set of vectors. Phase shift (the problem of different starting points) is handled by passing the vectors into the frequency domain and back again (Drew, Lee, & Rova, 2009). This method is more computationally efficient than the maxima method, and produces better results (ibid). Using the MPEG-7 contour database, the classification success rate in a ‘leave-one-out’ test was 94.1% over 1400 objects in 70 classes.

A CSS plot may be of use when locating features on an object curve. However comparing the object curves directly is likely to be vulnerable to occlusions.

## *3.7. Extracted Features*

This section focuses on the different feature types extracted from the object curves, rather than the techniques used to compare them. The type of representation used in describing an object's shape will have a significant impact on the effectiveness of a general, object-recognition system (Sebastian & Kimia, 2001). Object features are generally chosen to maintain a unique representation of an image with the minimum amount of data. Feature types can range from points, to simple geometric features, to template images.

### 3.7.1. Object Contours Using Arc Length and Tangent Orientation

A contour can be represented by an arc length variance vs. tangent orientation variance plot, which is  used to provide a better comparison of the data (Xu & Xu, 2004). The ‘arc length’ describes how far along the contour a point is, and ‘tangent orientation’ is the tangent angle of the contour at that point.

The contour is replaced with a Bezier-Spline curve to increase the efficiency; it gives a good approximation of the contour, but with far fewer control points. The arc length vs. tangent orientation variance plot can be calculated from the curve. The resulting plot can be considered rotation- and partially scale-invariant (ibid). The system is only partially scale invariant due to the fixed distance sampling to calculate the variance. The arc length vs tangent angle variance plot is similar to the curvature scale space method used in other systems (Mokhtarian, 1997), and similar to the fingerprint used in the isolume system (Flemmer & Bakker, 2005).

The plot is then broken into smaller patches representing significant features on the contour. Various attributes of these patches are recorded and can be compared with an object database for identification (ibid), the division into patches aid in its ability to process occluded contours. Results show around a 90% accuracy rate for jet plane shapes.

The replacement with a B-Spline will remove some resolution, and plotting the arc length vs tangent angle seems more suited for comparing contours directly.

### 3.7.2. Hough Transform

The Hough transform detects and records geometric primitives within an image, such as lines, curves, and even ellipses (Aguado & Nixon, 1996). It achieves this by determining parameters that characterise these patterns (Illingworth & Kittler, 1988).

Candidate points for the transform can be found by applying an edge detector to the image. For the linear version of the Hough transform, lines at various angles are fitted to each of the edge points. The parameters are determined by transforming the lines from real space into Hough-parameter-space—where they appear as sine waves—using the angle and normal distance from the origin. The intersection points of the sine waves in Hough-space correspond to the line parameters in real space (Richard O Duda & Peter E Hart, 1972), where they are represented as infinite-length lines. An accumulator array is used to record how many sine waves pass through each set of parameters, this array has to be very large to get accurate results (Fernandes & Oliveira, 2008). Adaptations of this system have been developed for other more complex shapes (Aguado & Nixon, 1996). Arcs fitted to the real points appear as lines in Hough-parameter space, and again the intersection points give the main curves within an image.

It is worth noting that once the features have been extracted, for some image processing applications they must be bound to finite length lines and curves instead of infinite length, which adds additional computation requirements. The Hough-transform technique can generate false positives, the number of which increases with the complexity of the image (Grimson & Huttenlocher, 1988).

The lack of definite end points and the possibility of false positives make this algorithm unsuitable for our purposes.

### 3.7.3. Best Fit Geometric Features

Many systems use an algorithm to fit a set of geometric features to the data, this data may be the raw image pixels (Zhao et al., 2006), contour data (Flemmer & Bakker, 2005), edge points (R O Duda & P E Hart, 1972), or something different. The geometric features used are varied; blobs (Duygulu, Barnard, de-Freitas, & Forsyth, 2002, pg. 99), lines

(Bakker & Flemmer, 2009), partial ellipses (Howarth, Bakker, & Flemmer, 2009), skeletal members (Sebastian & Kimia, 2001), and more. The Hough transform is a popular method to obtain the best fit, but many applications use dedicated algorithms.

Creating specialised algorithms and using features that suit the proposed algorithm can offer many benefits such as; custom features, specialised feature representation, faster runtime, and enhanced accuracy.

## *3.8. Feature Comparison Algorithms*

The generalised requirements for the feature comparison part of the system are to make many comparisons in a short amount of time, while still maintaining good accuracy and discrimination. The systems reviewed here mainly come from the algorithms reviewed in above sections; however there is a vast number of methods used for comparing two sets of data and algorithms from other fields may be applicable.

### 3.8.1. Object Classes/Clusters

Grouping objects into similar classes can reduce the number of comparisons needed, making it possible to process very large databases (Jain, Murty, & Flynn, 1999).

Hierarchical clustering is an approach that greatly reduces the number of comparisons needed for large object set. The objects are grouped by some measure of similarity. The search is performed by first finding the viewed object's group, then searching within the group for an exact match (Cole, Austin, & Cole, 2004). The library objects are grouped together by the similarity of their features, and are represented by an average of these features (ibid). Implemented using a dataset of Lego bricks and a template matching algorithm, recognition rates of 90% and times of 6.7 sec per image have been achieved.

Decision trees can be used to select the correct class from a large database very quickly (Quinlan, 1986). Using a decision tree, rather than the usual cascaded class detector checks, provides far better scalability for large object class databases (Zehnder, Koller-Meier, & Gool, 2006). Object-recognition is well suited to the implementation of a decision tree due to the large amount of similar features shared between object classes (ibid). Decision trees are a cornerstone of many of database engines such as MySQL, the index defined is used to create m-trees (Teixeira, 2011) to facilitate much faster searches .

In the method of Zehnder, Koller-Meier (ibid) each step of the tree contains a ternary decision point, rather than a binary one. There are situations where the answer is unclear or Dependent on unknown factors; the decision can be reviewed at a later stage when more information is present. Only the feature being checked at each decision node must be calculated, making the algorithm run faster for easy-to-recognise objects (ibid). The decision tree ends with class detector nodes, to ensure low false positives.

The comparisons can be done with a simple method such as normalised cross correlation. The effectiveness of the system depends greatly on the features used, and the algorithm to compare them. Both hierarchical clustering and decision trees show merit, and either may be added to the algorithm at a later time if applicable.

### 3.8.2. Maximum Entropy Framework

Maximum entropy is a statistical method used to predict the outcome of a system based on its inputs, done by self learning with large amounts of data (Adam, Vincent, & Stephen, 1996). When applied to object recognition, various features and their parameters can be viewed as the inputs, and the identified object is the outcome of the system.

Lazebnik, Schmid et al have implemented an object-recognition system based on the maximum entropy principle. The features are 'binned' according to various parameters, and standard 'word' values are generated. The system can then operate similarly to the text translation programs using the same principle (Ocg & Ney, 2002). Additional feature types could be added, requiring very little changes to the system.

Clustering algorithms are used to write the dictionary and the bins for each word, by grouping sets of similar features together, ensuring there are no duplicate entries in the dictionary (Lazebnik, Schmid, & Ponce, 2005). The dictionary not only contains specific instances of the features, but also the geometric relations between them (ibid). The maximum entropy model is used with a set of training images to generate a probability distribution for the dictionary words in the object. Object searches are performed by calculating the number of words present in the image that correspond to the database object.

The time scalability of the maximum entropy framework method is expected to be quite good, as the generation of the dictionary words is done only once per image. However the word data is fairly low level and will result in a very large dictionary. The results for this method are very good, with recognition rates of around 90%. It is not known if the recognition rate would decrease when this system is run with a large database.

### 3.8.3. Semi-Local Affine Parts

A semi-local affine part is described as a collection of features that are present in multiple views of the same object (Lazebnik, Schmid, & Ponce, 2004), other methods have also taken a similarly view-independent approach (Ferrari, Tuytelaars, & Gool, 2004).

These are found by applying geometric transforms to various features in an object view to get them to match with those in another view; if the match is good, and the transform is the same for a group of features, it is considered a semi-local affine part (Lazebnik et al., 2004). Each object in the database contains a list of these semi-local affine parts, which are matched to features within an image. If enough matches are found it can be said the object is contained with the image.

The results for this method look promising with recognition rates of around 90%; however the computation to match each part is rather intensive, so a full database search is expected to take a long time, making this type of system unsuitable.

## 3.9. Conclusions on the Literature Survey

The literature review has provided insight into many facets of the object-recognition problem; the human vision system, different edge-curve generation methods, and existing algorithms based on edge curves.

A number of edge-curve generation methods are available, each has strengths and weaknesses, Flemmer and Bakker's *Iso-luminal contours* seems best suited to this application. The processing time is comparable, and the data gathered is in an easily useable format. The data also shows good accuracy and repeatability.

A possible contributing factor to the effectiveness of the human vision system is its hierarchical structure, which converts the image to higher level data through a number of stages. Emulating this type of system may lead to better performance.

In the field of computer vision, there are very few systems being designed for large database object-recognition, and none that seem feasible in the near future. The majority of systems are designed for specialised tasks of small scope. Many systems fail to scale up to large object databases as either the processing time increases, or the recognition rates decrease. To design a suitable system we must learn from the shortcomings of these other systems.

# 4. Recognition Method

The following section describes the algorithms used in this work to recognise specific objects within a scene. The first subsections are the isolume extraction and feature recognition methods. A test method and results for the feature extraction part of the algorithm is described, and shows the performance of the system's initial stages.

Two methods are outlined for constructing and comparing high level data constructed from the features found; bit descriptors, and pair and triple blocks. In the case of the pair and triple blocks the comparison method is multi-stage, consisting of direct block comparison, clustering, and finally spatial matching. Both of the methods make use of a similar learning algorithm to assign weights to the more important bit descriptors or blocks.

## 4.1. Isolumes

The proposed system uses an object's characteristic curves to identify higher level features and relationships. These curves are found on the object edges and anywhere else where there is a significant tonal change. A general term used in literature for this sort of curve is a *contour*. Contours contain a list of the points used to make up an object's curve. This system builds upon the work of Flemmer and Bakker, and uses isoluminal contours to identify object features.

Isoluminal contours—hereafter known as *isolumes*—are contours that follow a specified brightness. An image is made up of discrete points 'pixels' arranged in rows and columns, this system uses images with 640 columns and 480 rows. The colour image is converted to grey scale so each pixel is represented by a single brightness value; instead of red, green and blue values. An algorithm finds the isolumes by searching for curves located on a constant grey level intensity within the image. An isolume is defined as a set of points located on the same brightness, or luminance; giving the name, iso-lume.

In addition to the constant grey level across the points, they must also be located on a brightness 'cross-gradient' of sufficient magnitude. The cross-gradient is a measure of how much the brightness is changing, which occurs on object edges and internal features. It is taken perpendicular to the direction of the isolume, and is calculated using the grey-level difference between two neighbouring orthogonal points.

To illustrate the role of isolumes, the image can be thought of as a topographic map; with the red, green, and blue values of each point combined to give a grey level 'height'. An isolume follows the hillsides at a specific height until they flatten off or come full circle. Topographic maps place lines along a specified altitude level, in a similar way to an isolume being fitted to a specific grey level. Figure 4 shows a topographic map of an object corner, the red line is an isolume which finishes due to lack of cross-gradient as the 'slope' gets gentler. Steeper gradients are more strongly associated with the object/image features we are looking for, so an isolume will not be fitted when there is insufficient cross-gradient.

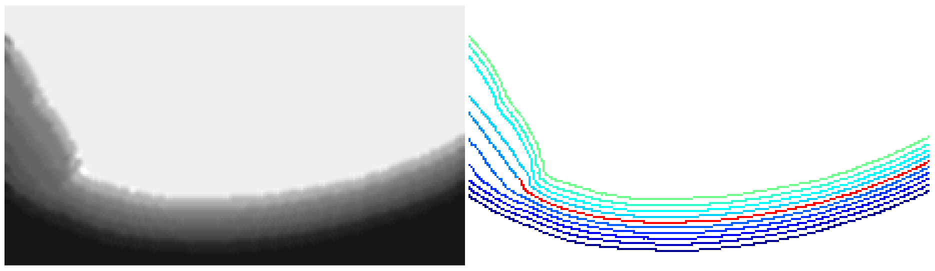


**Figure 4. Topographic map of an object's corner generated from a greyscale image**

*Shown are the image and its contours interpolated from the brightness values, i.e. isolumes. Dark blue contours have the lowest grey level, light green the highest.*

### 4.1.1. Extraction

The isolume extraction algorithm was based on one designed by Flemmer and Bakker (Flemmer & Bakker, 2005) which is also described in the literature review section (Iso-luminal Contours, pg 15). Some modifications have been made to the algorithm to enhance processing speed and to tailor the results to the feature-extraction algorithm's requirements.

#### 4.1.1.1. Specific Implementation

The implementation of the isolume extraction algorithm was optimised for speed without comprising its accuracy or robustness. The algorithm was efficiently implemented using threading and unmanaged memory operations where practical. Threading makes use of multiple processor cores, allowing the program to process multiple subroutines in parallel. Unmanaged memory operations use memory address pointers to copy or work with large blocks of data, ignoring the time consuming data type checks etc. The basic algorithm can be described with a flowchart (Figure 5), and is explained in further detail below. C# is used as the programming language, as it supports object oriented programming, and is easy and efficient to use.

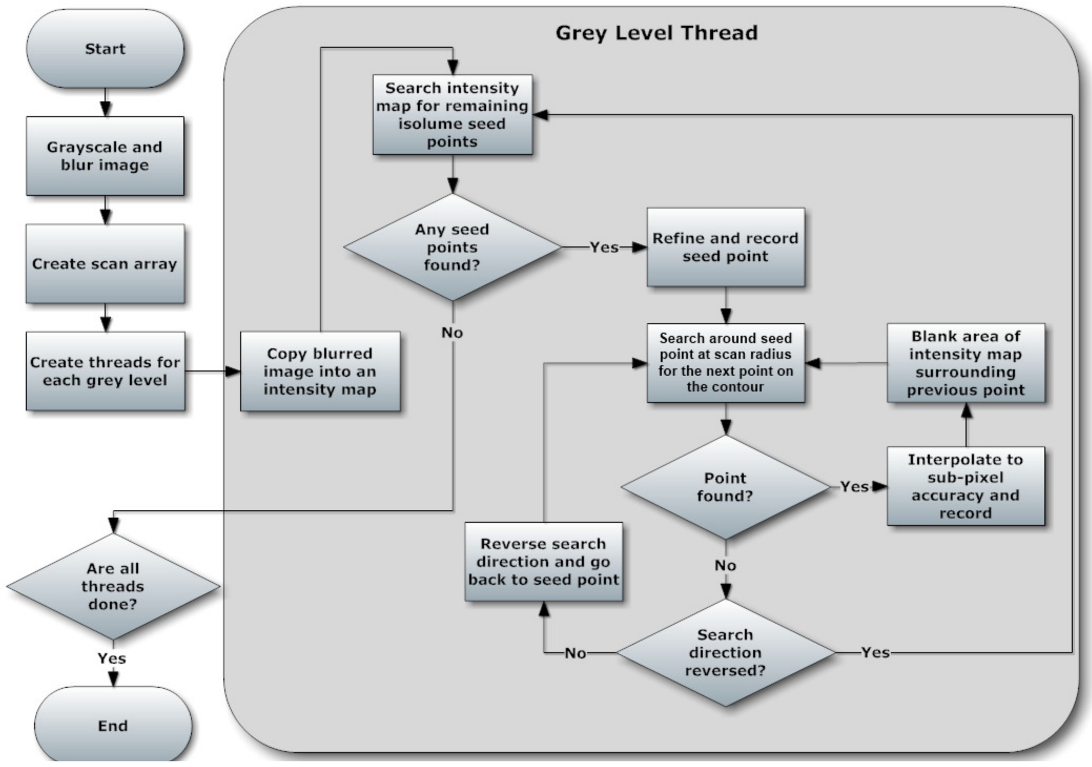

Start
Grayscale and blur image
Create scan array
Create threads for each grey level
Grey Level Thread
Copy blurred image into an intensity map
Search intensity map for remaining isolume seed points
Any seed points found?
Yes
No
Refine and record seed point
Search around seed point at scan radius for the next point on the contour
Blank area of intensity map surrounding previous point
Point found?
Yes
Interpolate to sub-pixel accuracy and record
No
Reverse search direction and go back to seed point
Search direction reversed?
No
Yes
Are all threads done?
Yes
End

**Figure 5. Isolume extraction flowchart**

## Algorithm Summary

Firstly the image is grey-scaled to convert each pixel to a single intensity. The image is then blurred to smooth it. Next the contour start points—or seed points—are sought. For a particular grey level a seed point is selected, and the contour is traced along the same grey level. Starting at the seed point the contour is traced out until the grey level cross gradient diminishes, or it comes full circle, or it meets the borders of the image. During tracing, a 'scan array' is used to find the next contour point by 'sweeping' ahead. Any potential seed points in the neighbourhood of each contour point are erased as it is traced. These operations will be discussed separately—and in more detail—in the following sections.

## Greyscale

A greyscale conversion is used to obtain a single intensity value for each pixel, from the RGB values. A commonly used calculation (Equation 1) is used to convert the image to greyscale, pixel by pixel. Individual weights for the different colours are used to correct for the colour perception of a human eye, making it easier for a human to code and debug.

Unmanaged memory operations within C# are used for the calculation, which greatly reduce the time required to retrieve and store the RGB and grey level information.

$$G_l = \frac{11 * R + 16 * G + 5 * B}{32}$$

**Equation 1. RGB to grey level conversion**

*$G_l$ represents the calculated grey level intensity, with R G and B the input values obtained from the colour of each pixel in the image.*

## Blur

The greyscale image is blurred to reduce the effects of image noise, artefacts, and aliased edges; it is expected that this will create smoother contours. Image noise and artefacts can be introduced by the camera. A high quality camera correctly set up should reduce these. Aliased edges occur because the image edge must be put in discrete pixels, and the edge colour information will be put into the nearest pixel; it is similar to an access ramp following the same path as some stairs. The blur is performed by taking a weighted average of the pixel intensities surrounding each point in the image.

A Gaussian distribution was chosen because it is a circular blur, and puts more ‘weight’ on the pixels closest to the centre. Placing greater weight on central pixels is expected to remove noise without reducing image information like the grey level cross-gradient as much as other blur types.

The Gaussian blur is often performed by convolving the two-dimensional kernel (an array used as an operator in the convolution) with the image intensities, however, this two dimensional convolution requires many calculations. The Gaussian kernel has a linear-separable property because its transposed kernel is the same as the original. This allows the convolution to be split into two single-dimensional convolutions, and greatly reduces the processing time. The single-dimensional kernel is convolved with the image intensities to calculate the effects from the neighbouring horizontal pixels. These calculated intensities are then convolved with a transposed single-dimensional-kernel to add the vertical effects. For a blur radius of 5 pixels, this reduces the number of calculations from 81 to 18 for each pixel. Unmanaged memory addressing was also used to speed up access and storage times for the intensities.

The Gaussian function is a bell curve of infinite width and unit area, with the steepness defined by the standard deviation used to calculate the function. Although a theoretical Gaussian blur will have an infinite radius, adjusting the standard deviation—and the bell curve's steepness—allowed practical adjustment of the blur radius. The finite radius kernel was calculated by adjusting the Gaussian distribution's standard deviation (in pixels) until the sum of all values within the specified radius was greater than 0.99; all kernel values were then scaled to bring the sum up to 1.

For a 640x480 pixel image the greyscale and blur functions took around 35ms on a 2009 desktop machine with a dual-core processor and threading of the functions.

**Scan Array**

The next step in the algorithm is to initialise the scan array and create threads for each grey level. The scan array (Figure 6) is used by the contour following part of algorithm to scan ahead easily, and find the pixel intensity of points on a given heading and distance. The scan array could be viewed as 24 vectors of length approximately 4 pixels. The index of each of these vectors corresponds to its heading. The vectors provide offset values, which are applied to the current point's indices to find the next potential contour point along a given heading.

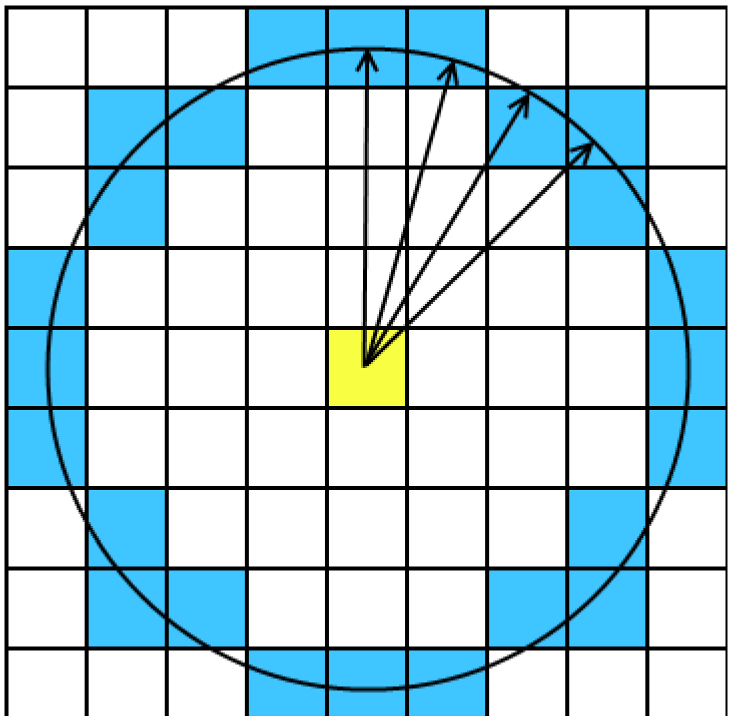


**Figure 6. Scan Array**

*The scan array contains the vector x and y values for 24 different headings around the current contour point (yellow). The vectors are all 4 pixels long, with their x and y values rounded to the nearest integer (blue).*

For example, if the contour is travelling upwards, the offsets obtained from the scan array are the scan-radius for x, and 0 for y; if it is coming from the point 85,33 the next point will be 85 + the scan-radius, 33. The scan array contains the x and y offsets for all points along the scan circumference, indexed clockwise from the top. Without the scan array two trig calculations would need to be performed, and rounded to the next pixels, multiple times for each new contour point.

The number of discrete headings, and the size of the scan array, is driven by the scan-radius used. Through experimentation we found that a scan-radius of 4 provided good contours across many different types of images.

### Thread Initialisation

The contour extraction algorithm uses a single thread for each grey level, with 50 grey levels being searched. The large number of grey levels ensures significant isolume overlap, to identify and represent the object features better. The blurred greyscale image is copied into a 'pixel intensity array' at the start of each thread, allowing the thread subroutine to remove/change the intensity values of points recorded in its isolumes.

For each grey level there may be a number of isolumes present. To ensure all the data is gathered the seed point acquisition and isolume following methods are repeated to find and extract them all (Figure 5).

### Seed Point Acquisition

The pixel intensity array is searched to find a contour start—or 'seed'—point. A seed point is any point with its intensity close to the grey level being searched. To find the seed points the image is scanned left to right, going top to bottom along the way. This point is not actually used in the isolume, but provides the starting point for a local fine-grained search to find the optimal point to begin tracing an isolume. The fine grained search is done in the same way as the contour following part of the algorithm.

### Isolume Following

To follow the isolume four basic operations are performed. These are done for each 'step' as the system 'walks' its way along an isolume. These four steps are now summarised, and explained in further detail below. Firstly the next point is found by scanning ahead. This point must be on the desired grey level, the scanning direction is adjusted until it is found. The intensity cross gradient is checked at this point, and also used to refine the point. Finally the seed points close to any point on the isolume are removed. These four operations are repeated until the intensity cross gradient diminishes, or the isolume comes full circle, or the image border is reached.

The isolume has a current point and a heading at each step. These are used to identify the two points ahead that straddle the grey level. The contour points are always a fixed distance of approximately four pixels (the scan radius) apart. The grey level of the point closest to the current heading is checked first, then those further around. The scan array (Figure 7) is used to obtain the index of the points ahead, for a given heading.

To find the two points straddling the desired grey level, the isolume heading is adjusted. The algorithm determines whether to search clockwise or anticlockwise from the current heading based on the brightness level found in accordance with the ‘bright on the right’ rule. This rule means that the point with the higher grey level must always be on the right in the direction of travel along the isolume. This makes it more likely to follow the same image feature, and standardises the algorithm. The vectors of the scan array are used to identify the next point to evaluated, with the array index incremented or decremented to find the next point along the search radius. If both points are above the desired grey level the system will continue to search anticlockwise, if they are below, the direction moves clockwise.

The isolume only continues if there is a sufficient cross-gradient. The point on the right must have an intensity of 7 or greater than point on the left.

The point is recorded in the isolume with sub-pixel accuracy according to the interpolation formula in Equation 2. Linear interpolation is used to place the isolume point between the two straddling pixels, where the intensity gradient crosses the desired grey level.

$$X_i = X_l + (X_r - X_l)\frac{G - G_l}{G_r - G_l}$$

**Equation 2. Isolume point interpolation formula**

*Where $X_i$ is the interpolated x value, $X_r$ is the X value on the right of the grey level, $X_l$ is on the left, G is the grey level being followed, $G_l$ is the grey level of the point on the left, $G_r$ is the grey level on the right. An identical equation is used to calculate the interpolated Y value.*

To erase the seed points for subsequent isolumes, all points within the scan radius in the pixel intensity array are set negative. A rubout array (Figure 7) is used to perform this operation quickly, the array contains the X and Y offset indices of all points inside the scan array relative to the current contour point. It is a simple matter to iterate through and set these points in the pixel intensity array to -1 to ensure they will not be used in other isolumes.

A discrete point is required to set as the current isolume point to continue the following steps; the interpolated point is simply rounded to suit.

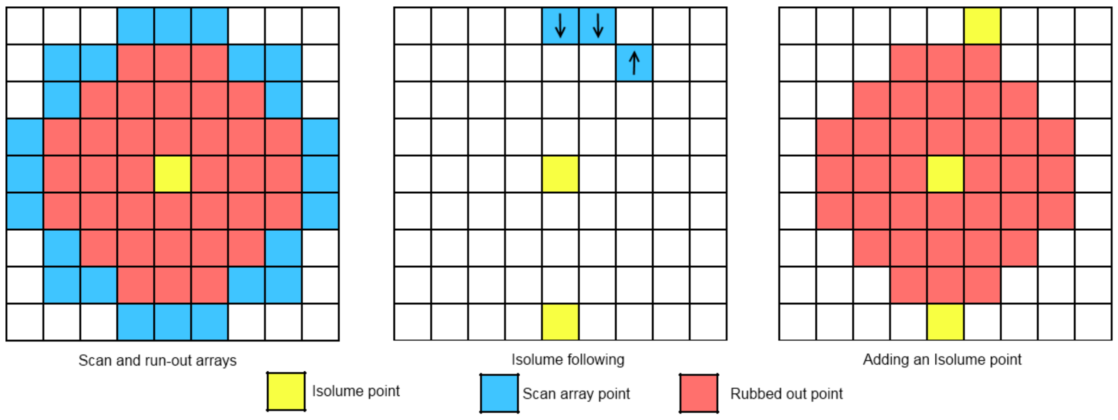


**Figure 7. Pixel operations for isolume extraction**

*Figure 7 illustrates the process at the pixel scale. The first grid shows the size of the scan (blue) and rub-out (pink) arrays, centred about the current isolume point (yellow). The second shows the process used to find the next isolume point. The last grid shows the interpolated isolume pixel rounded to the nearest integer, and the red squares show the other pixels which will not be used again for this grey level.*

## Contour Finishing

The isolume following algorithm continues until the grey level cross-gradient falls below the threshold, or the isolume comes full circle, or reaches the edge of the image. If the isolume ends without coming full circle then only one end has been found. To find its other end, the current point is set to the start point of the isolume, and the contour heading and search direction are reversed (to bright on the left). The contour-following loop then proceeds in the same fashion to find the other end.

With both ends of a single isolume found, and the points removed from the pixel intensity map, the contour-following process selects another seed point and repeats to find and add another isolume.

When the seed points are exhausted, the thread concludes by adding the isolumes to those from other grey levels. Once all the specified grey levels have been processed and their isolumes added to the image’s group, the system moves on to the next stage; the extraction of the arc, line, and lobe features.

### 4.1.2. Comparison with Flemmer and Bakker’s Algorithm

The isolume extraction algorithm is based on that created by Flemmer and Bakker; however we have added a few differences to customise it for the rest of this system.

The image blur was performed with a Gaussian blur. Flemmer and Bakker used a simple average taken from the surrounding grey levels. The blur is done to reduce image noise, ensuring that the isolume-extraction algorithm finds curves with minimal effects from aliasing, artefacts, and image noise. The Gaussian blur assigns less weight to pixels further away, and is expected to remove the noise while maintaining a better representation of the image than an averaged square.

Flemmer and Bakker use a straight-line interpolation step to add one or more points in between the extracted isolume points. This is done to make it easier to calculate the curvature at each point. However, because the additional points do not add any information to the isolume they were left out of this algorithm, and other approaches to reliable curvature calculation were explored.

Flemmer and Bakker’s isolumes allow varied span distance between the points; designed to get shorter at areas of high curvature and longer on straight parts. This is done to reduce the aliasing effect seen when interpolating pixel points along a line. There was no observed reduction in calculation time by varying the step size. The adjustable step following algorithm must perform at least two calculations per point, which counteracts the lower number of overall points from a processing time perspective. The variable step approach was compared with a fixed width approach, and found to make little difference in feature identification due to subsequent processing steps (described in later sections). Variable step was abandoned in favour of uniform point separation to simplify the calculations required in subsequent methods.

## 4.2. Features

This section discusses the types of features chosen to represent the objects. These features are lines, lobes, arcs and partial ellipses and are all simple geometric shapes, apart from the lobe. The 'lobe' is a new data shape used to represent the corners of an object. These features must reliably represent a variety of shapes, and be mathematically fast to process. The features are extracted from the analysed contours (pg 32) using the contour's curvature plot, described in detail later as a 'fingerprint'. This section outlines how the feature extraction algorithm examines the contours to obtain these features. Emphasis is placed on finding a representation that is robust when presented with small feature visual from viewing the same feature in different images (i.e. parametrically identical arcs for the a portion of an object, measured in two different environments).

**Algorithm Summary**

The system takes a contour, made up of a large number of x and y points. These points trace out a curve and are found on the object's edge as well as internal parts where a tonal change occurs. To identify the boundaries between the different feature types a rate of curvature plot is used to describe how 'sharp' the corners are, or how 'flat' the straights are. This plot is also know as a contour's 'fingerprint' and is described in more detail in the following paragraph. The associated contour points between these boundaries are used to calculate the parameters of each feature. Co-incident and adjoining features from different contours are combined and the level of coincidence recorded as a measure of feature significance. The level of significance is used to rank features of each type, ensuring the best are passed to the later processing stages. This is done because these stages can only process a relatively small number (30) of features of each type.

### 4.2.1. Fingerprint Calculation

A contour's fingerprint is a plot of how sharp the corners are, or how flat the straights are. This is plotted from the start of the contour to its end. If the contour were a road, the fingerprint would show how much the steering wheel must be turned along the way. The fingerprint is used to identify ranges of points that are likely to belong to a single feature (i.e. line, arc, lobe, ellipse). Fingerprints have previously been used to identify feature divisions in contours successfully (Flemmer & Bakker, 2005).

The fingerprint is a graph of the contour's curvature with respect to the distance from its start. This is the contour's second spatial derivative. The rate of curvature is calculated by taking the inverse of the local radius, *r*, (Figure 8) at each point. The local radius is found by fitting an arc to three points; the current point, and the points two ahead and two behind it. This arrangement produced a more stable curvature plot than using the point one ahead and behind while still providing sufficient information on the sharper corners.

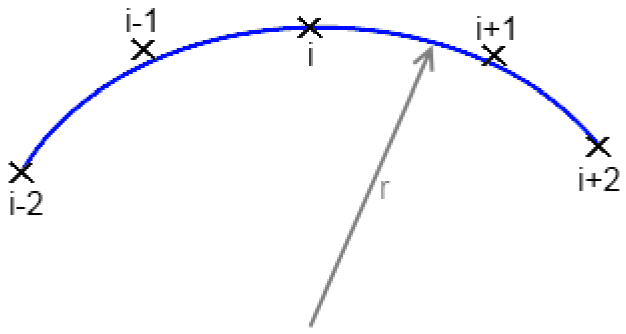


**Figure 8. Contour arc fitting**

The distance between contour points is five pixels, giving a fitted arc with a span of approximately twenty pixels. Although this five pixel scan distance may make smaller features difficult to identify, small features are more susceptible to image noise and are unlikely to provide significant information anyway. A five pixel scan distance gives repeatable fingerprints with stable curvature for many differently sized image features. Figure 9 shows the fingerprint generated for the outline of a mug.

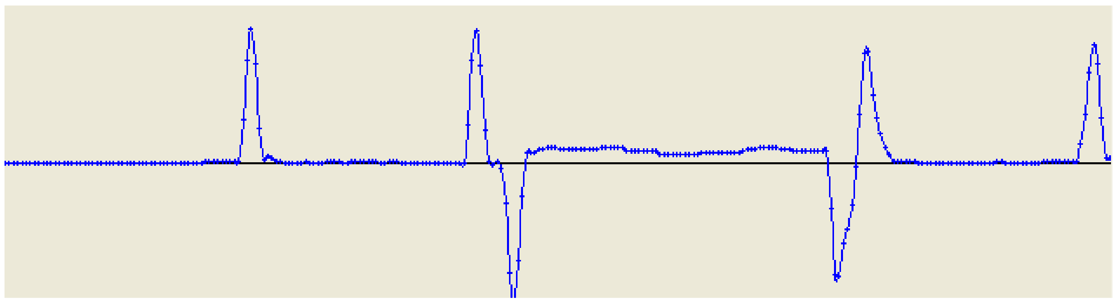


**Figure 9. Fingerprint generated by the system**

*The fingerprint is taken from the outline of a common mug (Figure 10), with 3 flat sides, one curved handle, and six corners (including those at the top and bottom of the handle). The curvature spikes shown are located on the major corners, the long areas of no curvature represent the flat sides, and the flat area of some curvature represents the handle.*

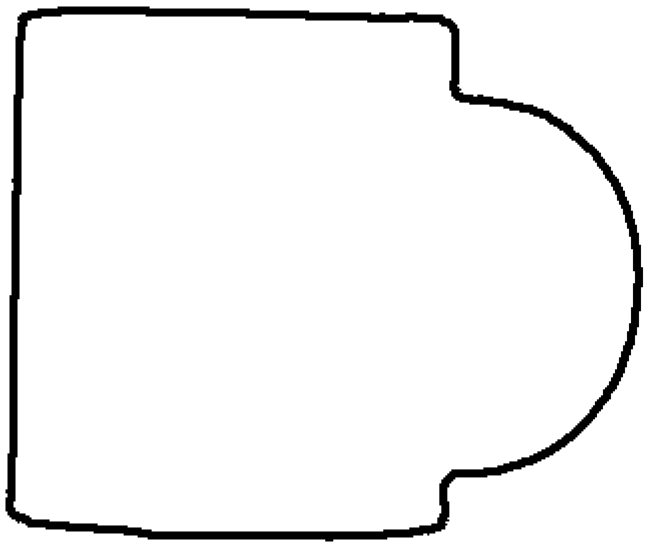


**Figure 10. Mug outline**

## Fingerprint Smoothing

The curvature calculation forms the basis of the fingerprint. However there are also a few processing and smoothing steps to ensure the feature boundaries are found correctly.

In following a contour, it is unreasonable to expect accuracy better than plus or minus one tenth of a pixel. Given that we are concerned with the slope of the contour—from which we derive the curvature—this level of error over the distance of a pixel leads to large errors, as may be seen in Figure 11. These errors were observed as sinusoidal ‘curvature harmonics’, with relatively low amplitudes. They are thought to be present due to the aliasing effects caused by the discretisation of the image. Aliased edges occur because the image edge must be put in discrete pixels, so the edge grey level will be put into the nearest pixel. This effect is similar to an access ramp following the same path as some stairs. Although the blur step and contour extraction were designed to remove these curvature harmonics, it was not completely effective. Despite the curvature harmonic’s low amplitude it was often large enough to cause a single feature’s group of points to be split.

The fingerprint smoothing method parses the fingerprint and examines the four points on either side of the current one to remove these harmonics. The period of the harmonics was observed to be three to ten pixels in most cases (Figure 11). If there is no significant change in the surrounding curvature values—which would definitely show a change in feature type—the fingerprint value is replaced with a nine point average.

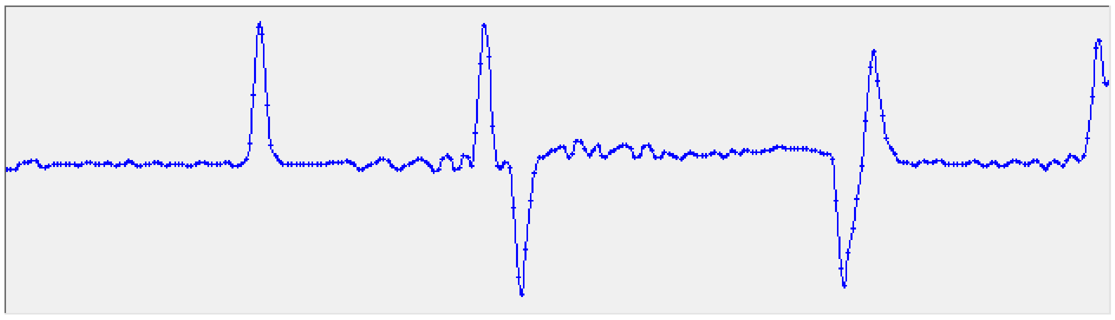


**Figure 11. Fingerprint showing curvature harmonics**

The selective nine point average removes the curvature harmonics, but preserves both the overall curvature trend and the lobe start/end points (Figure 12).

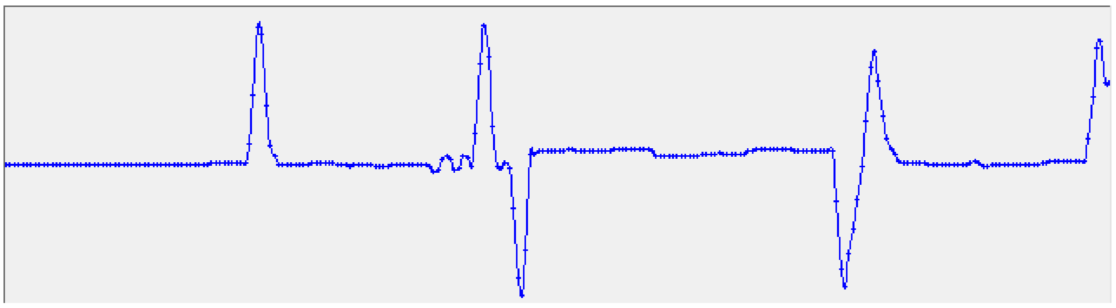


**Figure 12. Fingerprint with curvature harmonics removed**

## Feature Boundary Identification

The next step is to identify arc, line, and ellipse portions of the fingerprint. These are characterised by long flat groups of curvature values. ‘Spikes’ will be ignored initially. After the flat parts are processed the spikes will be used to find the lobe features.

The large segments of the fingerprint that are relatively flat are averaged to a single value. This makes it easier to identify the end points of lines, ellipses, and arcs, and provides the average curvature of the feature. A running average is used to evaluate the next point along the contour. A curvature threshold value (0.01) is used to determine if the next point belongs to this feature. Although the search begins from what is expected to be the beginning of a flat portion, sometimes the initial average is not representative of the whole portion. To address this issue the start is adjusted once the end has been found using the same process and threshold value, but with a more accurate average curvature value. The fingerprint boundary divisions are recorded when both ends of the flattened portion exceed the curvature threshold, or the fingerprint ends are reached. In the case of a closed loop contour, the system continues the flattening process across the ends, to find features spanning the contour’s start end join.

The fingerprint can then be divided (Figure 13) into sections based on the curvature behaviour. Each section represents a single feature. Divisions are placed at the ends of the flat sections. A set of points all located close to the x-axis are indicative of a line, and the start and end points of this line section are recorded for later processing. The same is done for ellipses/arcs which are seen as flat portions of the fingerprint, but a distance away from the x-axis. The remaining sections are processed for zero crossings, which indicate a lobe-lobe division as the contour follows a corner, turning the opposite direction. With the zero crossings added, any divisions of sufficient size (>=3 points) that remain are processed as lobes.

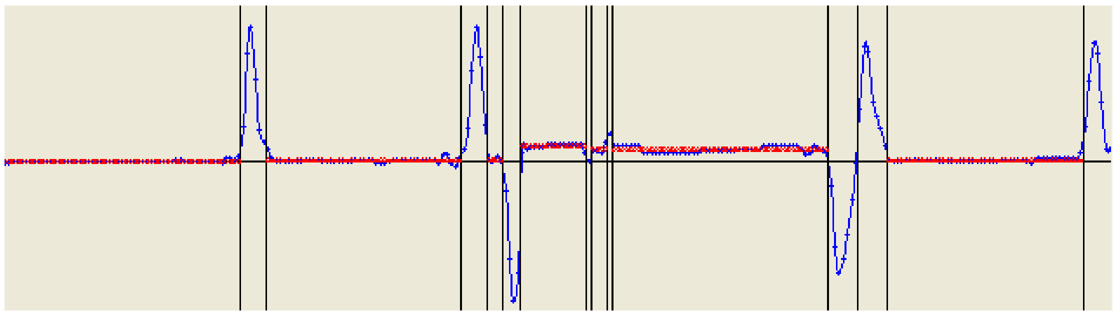


**Figure 13. Fingerprint with divisions shown**

Both the fingerprint points, and corresponding contour points of the individual sections are passed to the line-, lobe-, arc-, and ellipse-fitting methods described in detail below.

**Feature Coincidence**

Features found in different contours but on the same part of the image are combined for conciseness. A coincidence value is assigned to each feature to represent how many other features have been combined to form it. This is used in the feature combing method and in later stages of the analysis. It indicates the level of duplication in the represented feature. The coincidence value is used as a measure of importance, on the assumption that coincident features found on many contours are more important than features found only once. As each feature is found its coincidence value is set to one, and it is increased as co-incident features are combined.

### 4.2.2. Lines

Lines are the simplest of the features used, requiring only four parameters to represent. The parameters are; direction (θ), length (L), and x and y co-ordinates of the centre point (x, y) (Figure 14). The line feature is used to represent the long straight portions of a contour; such portions must have at least 9 points, giving a minimum line length of approximately 36 pixels.

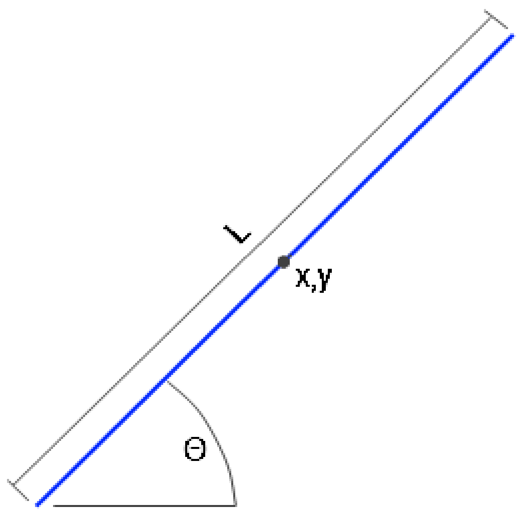


**Figure 14. Line feature type**

The line appears as a set of points along the x-axis of the contour's fingerprint. The image points corresponding to the set of fingerprint points are used to calculate the line's parameters.

The line is fitted between the two end-points of the identified contour section, so it is a simple matter to calculate the parameters. An end-point fit was chosen because it will better maintain the relationship between the line and other features following on the same contour. These 'join' relationships are seen to be important later in the analysis for creating the feature groups called 'bit descriptors' and 'blocks' (explained in details in a later section, pg 63). The line direction is normalised so it always points from left to right, or bottom to top in the case of a vertical line. This makes the feature direction independent of the contour direction. This is desirable as the contour will reverse direction if the background is lighter than the object, due to the bright on the right contour following approach.

Once the lines from all the isolumes are extracted they are examined for coincidence. Coincident lines are combined to streamline the subsequent processing steps. The criteria tested for coincidence are; line direction and a calculated perpendicular distance from the larger line's centre point to the smaller line. If two lines are found to be co-incident, the parameters are combined using a weighted average with the relative lengths and levels of coincidence giving the weight. The measure of coincidence for the new line is given from the sum of the old line's coincidence values. The coincidence value is the only non-averaged parameter, and used later to identify the most significant lines.

### 4.2.3. Lobes

Lobes are the a geometric feature, and require eleven parameters. The lobe is used to represent corners and features of widely varying curvature. The lobe is defined by its middle point ($x_m$, $y_m$), two end points ($x_s$, $y_s$, $x_e$, $y_e$), length (L), tangent angle at the centre (θ), and the angle the contour changes from start to finish (α) (Figure 15). 'Skewness' (S) and 'Kurtosis' (K) are used to describe the shape of the lobe's fingerprint (Figure 16). Although the contour representation cannot be accurately recreated from this definition, the end points and tangents are defined accurately, and the general shape is characterised.

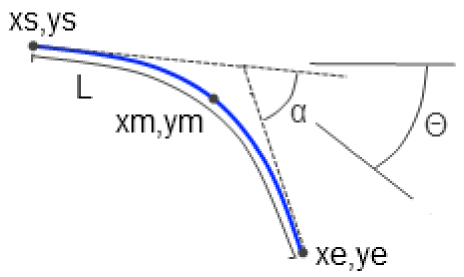


**Figure 15. Lobe feature type**

The lobe is fitted to the spikes on the fingerprint plot using the x and y points in the image associated with the lobe partition. The length parameter is calculated as the sum of Pythagorean distances between sequential points. The points used to represent the start, middle, and end, are taken with respect to the feature boundaries as discussed above. The angle is the difference between the contour's tangent at the start and end (α, Figure 16). The tangent is the angle bisecting the lobe start and end tangents (θ, Figure 16). Kurtosis and Skewness provide a numerical representation of the shape of the lobe's curvature plot. Kurtosis is a measure of how evenly distributed the curvature is—how 'pointy' the curvature plot is. Skewness refers to whether the centre of curvature is in the middle of the points, or to one edge—how much of a 'lean' the curvature plot shows. The lobe direction is made independent of the contour direction; which is desirable as the contour will reverse direction if the background is lighter than the object, due to the bright on the right contour following approach. To do this the start and end of the lobe are switched to ensure it is always presented in a clockwise direction; this is required roughly half the time.

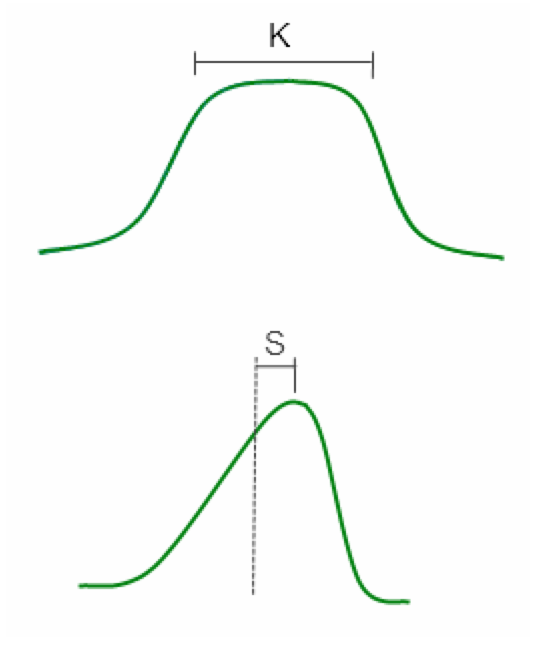


**Figure 16. Lobe skewness and kurtosis**

$$\frac{\left(\frac{\sum_{i=0}^{l} f(i) * i}{\sum_{i=0}^{l} f(i)}\right)}{l}$$

**Equation 3. Skewness calculation**

Equation 3 shows how the skewness value is calculated for a lobe. The values range from 0 to 1, with 0.5 having neutral skewness. *i* represents the index of any point along the lobe, which ranges from 0 to the length l; *f(i)* gives the curvature values at point *i*.

$$\frac{\sum_{i=0}^{l} f(i) * (i-c)^2}{\sum_{i=0}^{l} f(c) * (i-c)^2}$$

**Equation 4. Kurtosis calculation**

Equation 4 shows how the kurtosis value is calculated for a lobe. The values range from 0 to 1, with a value of 1 describing a completely flat fingerprint. *i* represents the index of the point along the lobe, which ranges from 0 to the contour segment length l. *c* is the lobe centre, calculated from the skewness multiplied by the lobe length. *f(i)* or *f(c)* gives the curvature values at the given point.

Coincident lobes are combined if the angle, tangent, and length, are similar, as well as having very close points. To calculate the parameters for the combined lobe, a weight is generated from each lobe's level of coincidence (1 if the lobe used has not been combined already); the new lobe is formed by a weighted average of the parameters. The new lobe's measure of coincidence is the sum of those in the combined lobes. The coincidence value is the only non-averaged parameter, and used to identify the most significant lobes.

### 4.2.4. Arcs

Arcs represent portions of the contour that have constant curvature. The parameters are; radius (r), centre point x and y (x, y), angle encompassed (α), tangent at centre (Θ), and length (L) (Figure 17). Although the length can be calculated from the other parameters (radius and angle), it is still included because it is used often and time is saved by not having to recalculate it. The image points corresponding to an arc are identified by the flat portion of the fingerprint, with a significant curvature value; as can be seen in the middle of Figure 13.

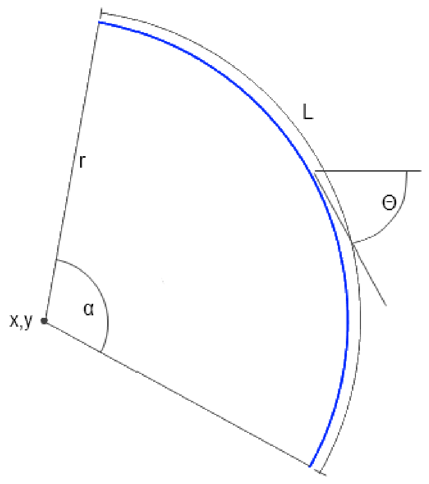


**Figure 17. Arc feature type**

The arc parameters are calculated by first constructing a triangle between three points; the two ends and one in the middle (Figure 18). The angle between the two lines connecting the endpoints to the midpoint is half the angle subtended by the arc (α). The radius is calculated by using the relationship between subtended angle, and chord length (cL). The tangent angle (Θ) is the direction from the start to the end point, and normalised so that arcs are always going clockwise. The centre point is calculated with trigonometry using the midpoint, tangent angle and radius. The length is calculated using the radius and angle (in radians).

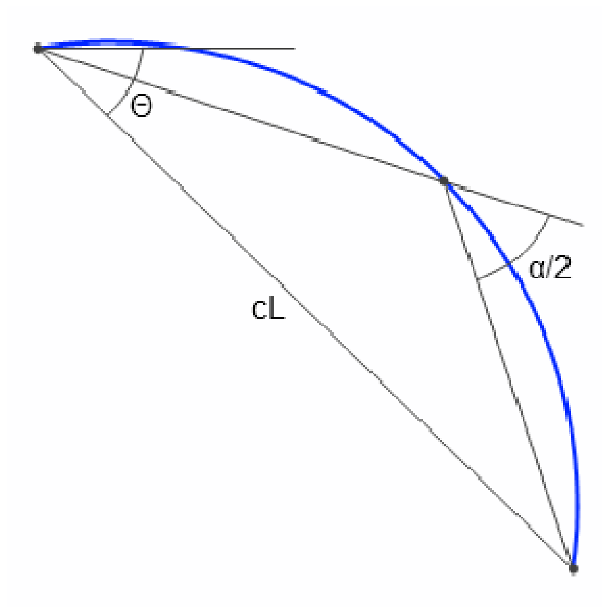


**Figure 18. Arc parameter calculation**

Arcs are combined when they are similar. To be considered similar, the arcs must be close to co-radial (sharing a similar radius and centre-point), as well as being at least partially coincident. There are two passes in this process, the first checks for duplicate arcs, while the second checks for co-radial arcs with different end points.

For the first pass, the arc centre location error is checked by calculating the Pythagorean distance between the arc centres; this must be smaller than a set fraction (0.1) of their average radii. The other parameters' differences are compared directly.

The parameters are all combined using a weighted average, the weighting given by each arc's level of coincidence. The level of coincidence for the new arc is the sum of the coincidence levels in the combined arcs.

The second pass initially compares the centre points and radii to check if they are co-radial (Figure 19). If the features are close to co-radial, the start and end angles for each are found. If the longer arc fully encompasses the smaller, the larger arc is returned (with an increased level of coincidence) and the smaller discarded. If neither encompasses the other, but there is some coincidence, the end points on the outside of the combined arc are found and a new arc constructed using the average centre-point and radius.

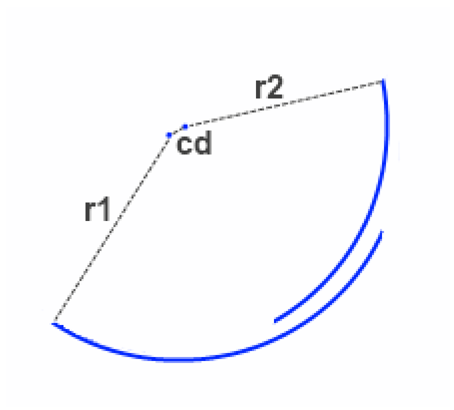


**Figure 19. Arc combining**

*The two arcs must have similar radii (r1 and r2), and the distance between the centres of rotation (cd) must be less than a set fraction of the average radius. The arcs must also show at least ten degrees of overlap.*

### 4.2.5. Ellipses

The ellipse feature type is designed to represent stable curves which vary slightly in radius. These are often found when a circular part of an object is viewed from an angle. Arcs cannot accurately represent these parts of the contour, so an ellipse feature was developed as a replacement. The ellipse is found on portions of the fingerprint with very little curvature variation, the same parts used to create arcs. An ellipse contains the following parameters; centre point x and y co-ordinates (x, y), major and minor radii (r1, r2), tangent angle (ϴ), ellipse angle (α), ellipse length (L), and the offset of the minor axis with respect to the ellipse tangent (β) (Figure 20).

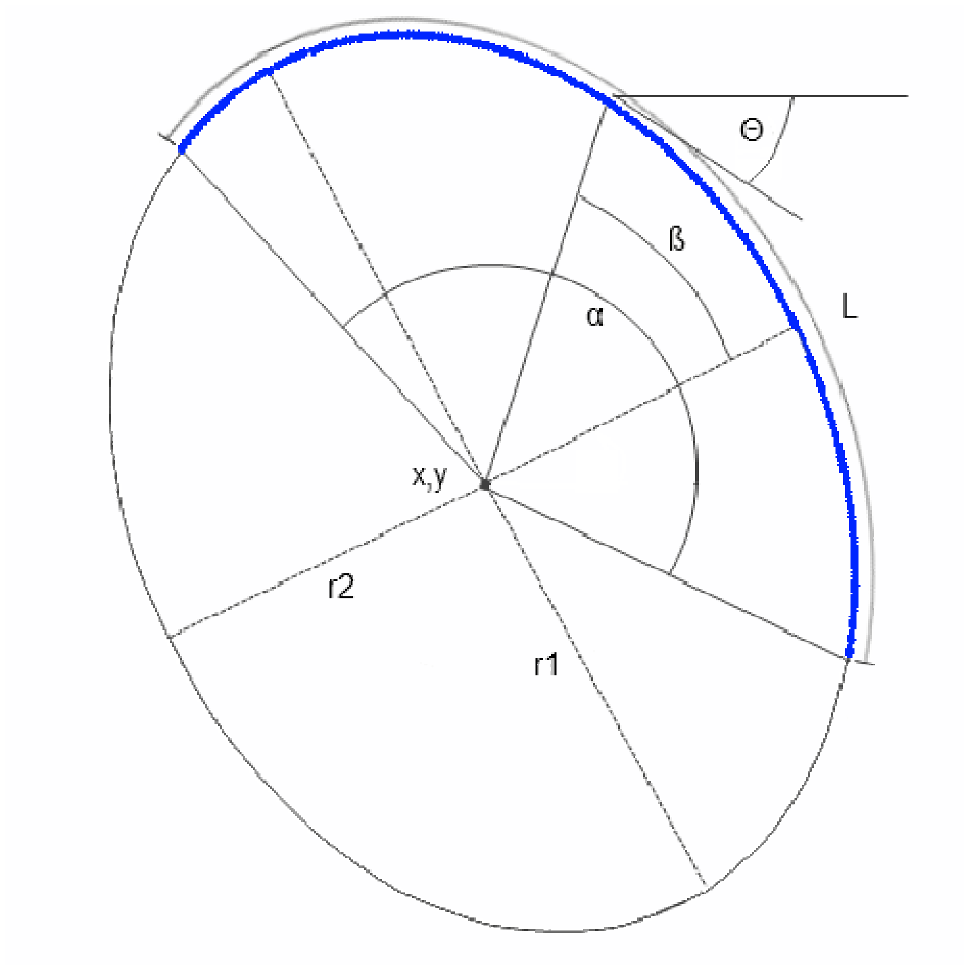


**Figure 20. Ellipse feature type**

The ellipse calculation is a modified form of Wen and Yuan’s separate parameter estimation technique (Wen & Yuan, 1994). To summarise Wen and Yuan’s method; the separate parameter estimation technique makes use of ellipses' skewed-symmetry property. The theory shows that an ellipse has countless skewed symmetry axes, which cross at its centre. For an ellipse, the mid points of any set of parallel lines running from one side to the other form a skewed symmetry axis (Wen & Yuan, 1994). The method takes a subset of points and fits parallel lines between the ellipse points to give an axis of symmetry. A reliable estimation of the ellipse's centre point is given by finding multiple axes of symmetry from different subsets, then identifying their intersection point. The remaining parameters (major and minor radii, and orientation) can be calculated from any three points using equations given in the article (Wen & Yuan, 1994). The average of multiple 3-point sets is used to best estimate the parameters.

The calculated parameters (orientation, centre-point, major and minor radii) and the ellipse points are used to identify the length, contained angle, and offset angle parameters to give a fully defined partial ellipse. However this technique only works reliably if there are sufficient points and a significant level of skewness, otherwise it is difficult to identify the major ellipse angle. When there is insufficient skewness an arc is fitted instead.

Ellipses are combined in a similar way to the arcs, with searches for duplicity, and coincidence. The first search compares all the parameters directly. If they are similar enough a new ellipse is created using a weighted average of the parameters.

In the second search the centre points and angles are reviewed similarly to the arcs (Figure 19); if there is a complete or partial overlap the ellipses are combined. Combining two ellipses is more complex than for other types, as the ellipse's skewed angle makes a large difference to the parameters. If the centre points and radii are similar, a number of points are created from each ellipse at known angles from the averaged centre point. Any angles which have points from both ellipses are averaged to a single point (Figure 21). An ellipse is then fitted to the generated points. A measure of error is given by calculating the distance between the new ellipse, and points from the original ellipses. If this is low enough the new ellipse is returned as the combined ellipse. The significance weightings from the original ellipses are combined to give the weight for the new ellipse.

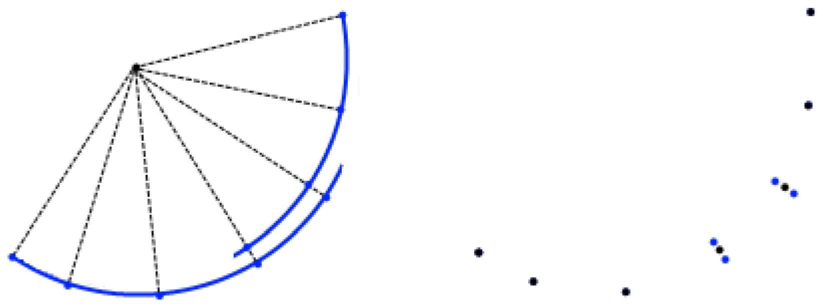


**Figure 21. Ellipse combining**

*Shown are two partial ellipses on the left, and their generated points on the right. The points are generated for each ellipse at known angles from their averaged centre-point. The black points (on the right) are used to create the new ellipse, for the overlapped part of the ellipses these points are placed between the two ellipse points (blue) for the given angles.*

## 4.3. Feature Extraction Test

A test method was developed to identify and investigate any limitations in the process of contour and feature extraction.

The test uses images of two objects in a variety of poses (Figure 22), to assess how the feature extraction system deals with rotation, scale, position and small lighting changes. The changes used by this test were selected to represent common object poses encountered by humans on a daily basis.

The objects used for this test were a mug for the line and lobe feature types, and a pair of scissors for the arc and ellipse types. These two different objects were used to provide a greater variety of feature shapes. Features from at least four different parts of the object are used for each feature type. Each image set contained twenty four images, varying in scale, rotation, position. To give a representative average, two pictures were taken for each position, rotation, or scale change. For comparison, the parameters from the features in the two target pictures are averaged. Each physical change of object position, rotation, or scale also introduced an unmeasured but observable lighting change on the object. Two physical position changes were used, moving the object by approximately 25% and 50% of the viewed area. The rotation changes used were 20, 45, 90, 180 and 270 degrees. The object scale was increased and decreased by approximately 25% and 50%. A thumbnail sample from the two image sets shows the twelve different object poses (Figure 22).

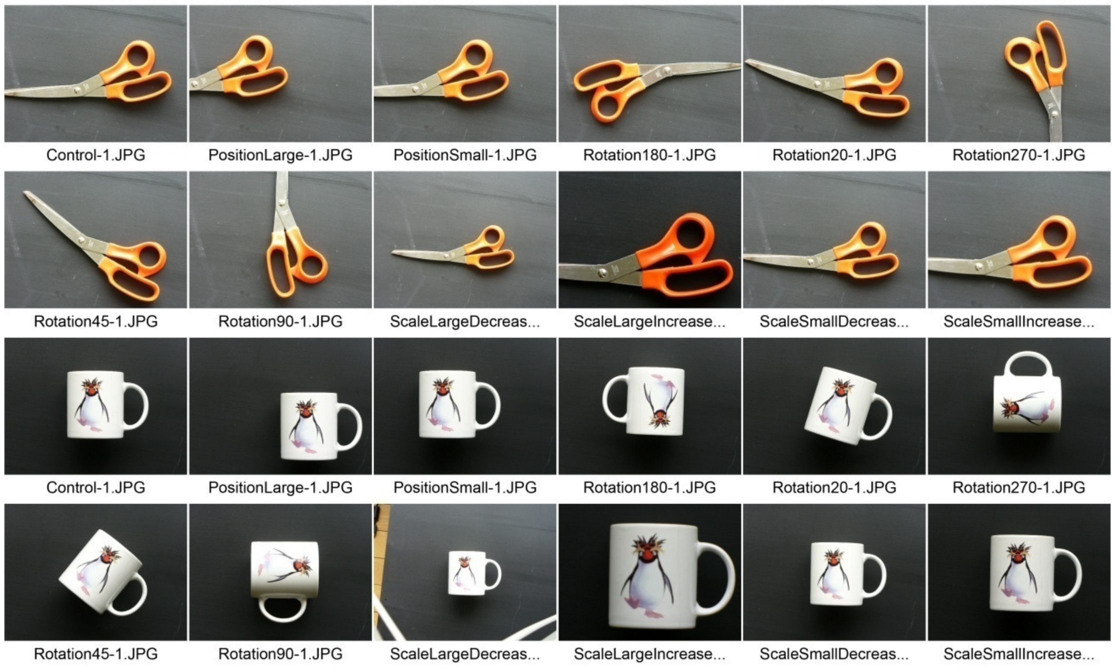


**Figure 22. Sample from feature test image set**

Each object has a 'target' control image that is considered to be the best case representation; the picture is taken of the front of the object, with lighting, scale, and rotation suitable for human object identification. A number of object parts corresponding to the different feature types were selected from the target images. These parts were corners, straight edges, or stable curves, and were found on the object edges. The two target images were processed and the line/lobe/arc/ellipse features identified in these object parts were recorded.

Each image in the test set was processed by the system, and all the features found were shown. The required feature for each part of the object was then manually identified and recorded for later comparison.

The test checked how reliably the system's features represent the same object part under different conditions. This was achieved by comparing the feature parameters that have scale-, rotation-, and position-independent representations. Some examples of these are; the angle contained by the arc/ellipse/lobe types, the skewness and kurtosis of the lobes, and the ellipse eccentricity. The test also compared some parameters that are scale- and rotation-dependent by using image-transform calculations to map the target object feature to the test image. Some examples of these are; tangent angle, feature length, and radius.

An image transform was used to identify the exact rotation and scale difference between the target image and the viewed image. The transform only uses scale and rotation; position parameters were not used as they had larger differences and were partially dependent on the scale and rotation. The image transform was calculated for each image using the chosen features, as well as some reference ones if required. In the case of the scissors which test for arcs, some lines were identified to give the image rotation. In the case of the mug, an arc was identified to give scale information.

The difference between each of the feature parameters in the target and the tested image was calculated. In the case of transform-dependent parameters the image transform is first applied to the target feature's parameters to remove the transform change from the results. The differences are then averaged across all the features of that type for each test image. This gives one error value for each parameter and image combination.

While testing the software it was observed that the position-dependent parameters (centre-point of circle, midpoint of line, end of lobe, etc) were too susceptible to camera effects such as fisheye, focal length, etc to give meaningful results when the simple image transform was used. These results were not recorded.

For the chosen parameters the average and standard deviation of the difference across the test image set were calculated (Table 1).

A number of features were not identified within the images. The percentage of features missed was calculated for each type (Table 2). The cause was observed to be most often due to the change in lighting conditions, with some feature endpoints incorrectly identified.

## 4.4. Feature Extraction Results

Results presented in Table 1 show how the repeatability of various parameters varies across the different types of feature. Features representing the same part of an object were recorded for several different instances of the object. The difference between the features in the target image, and the test images was calculated. This difference was used to calculate an average difference and standard deviation for the parameters in the different feature types.

Most feature types have parameters that can be compared independently of the image-transform, as well as some that are transform dependant. A transform is used so parameters can be compared using a small adjustment to normalise for the image viewpoint change. For example, if the object is rotated by 90 degrees the line directions should be changed by 90 degrees; subtracting the image transform's rotation allows the error difference to be calculated.

Parameters are categorised into "Transform Independent" or "Transform Dependent" because the calculated transform can introduce errors into the latter category. Angle deviation is given in radians; while distance based parameter errors are given in percentage of the 'target' parameter.

**Table 1. Feature comparison results**

| **Lines** | | Average Abs Difference | Standard Deviation | Units |
|---|---|---|---|---|
| Transform Dependent | Tangent | 0.023 | 0.019 | radians |
| | Length | 6% | 7% | % of original value |
| **Arcs** | | | | |
| Transform Independent | Angle | 0.348 | 0.547 | radians |
| Transform Dependent | Radius | 15% | 19% | % of original value |
| | Tangent | 0.212 | 0.302 | radians |
| **Lobes** | | | | |
| Transform Independent | Angle | 0.053 | 0.043 | radians |
| | Skewness | 8% | 7% | % of original value |
| | Kurtosis | 15% | 13% | % of original value |
| Transform Dependent | Tangent | 0.024 | 0.022 | radians |
| | Length | 11% | 12% | % of original value |
| **Ellipses** | | | | |
| Transform Independent | Angle | 0.595 | 0.932 | radians |
| | Eccentricity | 40% | 94% | % of original value |
| | Tangent Offset | 0.339 | 0.362 | radians |
| Transform Dependent | Tangent | 0.602 | 0.939 | radians |
| | R1 | 33% | 63% | % of original value |
| | R2 | 130% | 393% | % of original value |

In most cases the average levels of error and standard deviation are acceptable. However, for the ellipse feature type the errors are very large.

**Table 2. Features found, by type**

| | Found | Total Present in Image | Percentage |
|---|---|---|---|
| Lines | 90 | 92 | 98% |
| Arcs | 88 | 92 | 96% |
| Lobes | 126 | 138 | 91% |
| Ellipses | 86 | 92 | 93% |

Table 2 summarises the proportion of features found in the test images. None of the lobe features for an object part were present in every image, however most of the lines, and half the arcs and ellipses were. Because the system is only looking for features at a specific location—which are identified manually—extra features found are not recorded.

**Table 3. Additional arc test**

| **Arcs** | | Average Differenc | Standard Deviation | Units |
|---|---|---|---|---|
| Transform Independent | Angle | 0.033 | 0.037 | radians |
| Transform Dependent | Radius | 1% | 2% | % of original value |
| | Tangent | 0.036 | 0.024 | radians |

Table 3 shows a further test for arcs. It was conducted using a different object with end points that are easier to locate.

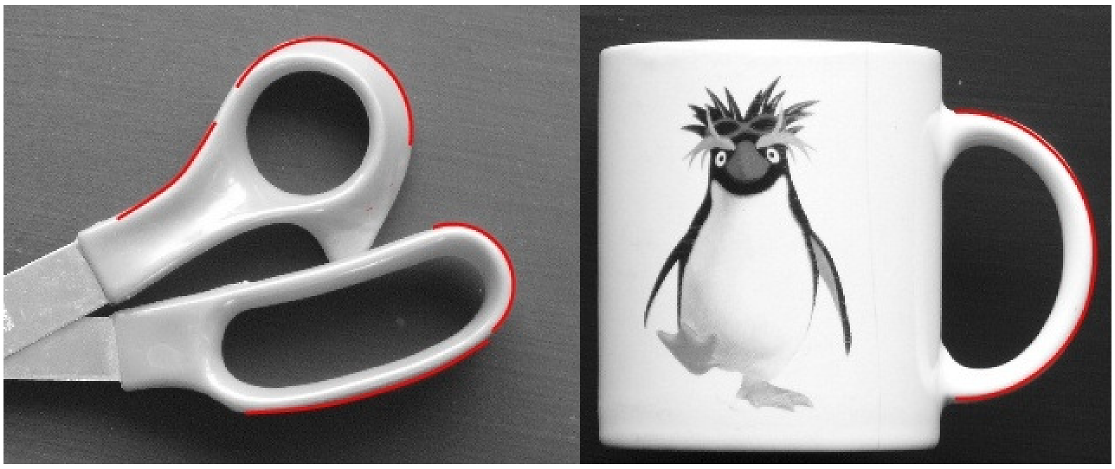


**Figure 23. Additional arc test feature comparison**

Figure 23 shows the arcs used in the first test, and the arc used in this test. The arcs on the pair of scissors do not have well defined ends like those on the mug. The object was the mug used for the line and lobe features, and only a single arc from the outside of the handle was used. The results were gathered and presented in the same way as for Table 1. When compared with the results in Table 1 this test shows how the accuracy of the identified arc features depends on the correct end point identification.

## Limitations of Analysis

The analysis encompasses the contour extraction as well as the feature extraction, so contour flaws or inconsistencies could propagate through and affect the result of one or more feature types.

Although a wide range of feature instances were used, not all shapes of arc or lobes are present, and the system could have less repeatability in some specific instances like very sharp corners with a narrow angle, or very high eccentricity ellipses, or many others.

## 4.5. Feature Extraction Conclusions

The test analysed the repeatability for a number of different features on two objects.

The proportion of features not identified in the images was low, with lobes having the most missing at 9%. On closer inspection it was found that the bulk of missed features were due to insufficient contours along an object part. The contours were missing due to shadows reducing or moving the intensity gradient too much. This effect could be reduced by adding more grey levels to process (which would incur a processing cost), or ensuring better lighting, either of which can be simply implemented later if required. Alternative edge detection methods may also reduce this affect; however, they are likely to be less suitable in other areas.

The consistency of the parameters is of concern for ellipses and arcs, while the lines and lobes show more acceptable variation. Because ellipses and arcs are assigned to the same sections of the contour only one should be kept. Because ellipses have far greater errors, only arcs will be used. The errors in arc parameters mainly come from the incorrect identification of the end points. The end points are difficult for the system to identify correctly on the scissors due to the many arc-arc transitions present. A further test using the handle of the mug confirmed that when the end points are more easily identified, the features are more repeatable (Table 3).

## *4.6. Comparison Techniques*

This section describes how the feature data is used to compare items in viewed scene to objects in the database. The features used are lines, lobes, and arcs (pg 32 for the contours, pg 42 for the features). The comparison must allow accurate object identification, whilst being quick to process. A direct comparison of the features satisfies neither of these constraints, so relationships between the features will be created and compared along with the feature's shape.

Two different methods were attempted; one utilising manually identified relationships, and the other using relationships identified by the system. The following sections describe the user-identified, 'bit descriptor' approach, and the computer-identified, 'pair and triple block' approach.

### 4.6.1. Bit Descriptors

This method sought to identify and compare the spatial relationships between features. The 'descriptors' are a set of predefined geometric relationships able to be identified between features (see later in this section). Each descriptor's presence in an image is indicated by a binary value; hence the term 'bit descriptor'. The descriptors are stored as an array of bits which allows the use of fast bitwise comparisons. The comparison step generates a match score for the object and viewed image.

The bit descriptor method is designed to be the first-pass in an object recognition system, so it need only generate a shortlist of potential object matches. A slower but more accurate system can then be used to identify a positive match from the shortlisted objects. The first-pass attribute means an emphasis is placed on a quick processing time, and a very small number of objects not identified in an image when they should be (false negatives).

#### 4.6.1.1. Descriptors

The purpose of the Bit Descriptors method is to provide a way to identify a number of significant geometric relationships associated with each object. These relationships are user defined, and may or may not be present in an image. There is no partial match for an individual descriptor. In the same way that a human can provide a fair and accurate description of an object using only words; the bit descriptor method aims to describe the object using only the bit descriptors.

A descriptor specifies a geometric relationship between features of a given type. The descriptors used were chosen to give a wide range of relationships, while being specific enough to provide a distinctive description of the object. A small example set of descriptor names is given in Table 4, some of which are subsequently explained in more detail.

**Table 4. Descriptor name examples**

| |
|---|
| Parallel Lines |
| Circles |
| Line Square |
| Triangles |
| Parallelograms |
| Symmetrical Lines |
| Axis of symmetry common to 3 feature pairs |
| Squares with Lobe Corners |
| Rotationally symmetric Lines |
| Right angle Line-Lobe-Line tangent groups |

**Arc Symmetry Descriptor**

The arc symmetry descriptor represents two very similar arcs which can be mapped to one-another using a line of symmetry. A line of symmetry is calculated for two arcs provided a number of constraints are met. The arcs must have similar angle (α), and length (L); having similar angle and length also ensures a similar radius. The distance between the arc features is also checked as distant arcs are less likely to be from the same object. Provided the arcs are similar enough, two intersecting lines are constructed from each arc's end points (Figure 24). The two constructed lines are then bisected by the axis of symmetry. Finally, perpendicular lines are plotted from the arcs' mid points (the mid-point is defined as the average of the two arc end points), to the symmetry line. If the distance (Pd) between the intersection points along the symmetry line is small enough the arcs are considered symmetric, and the line of symmetry is stored.

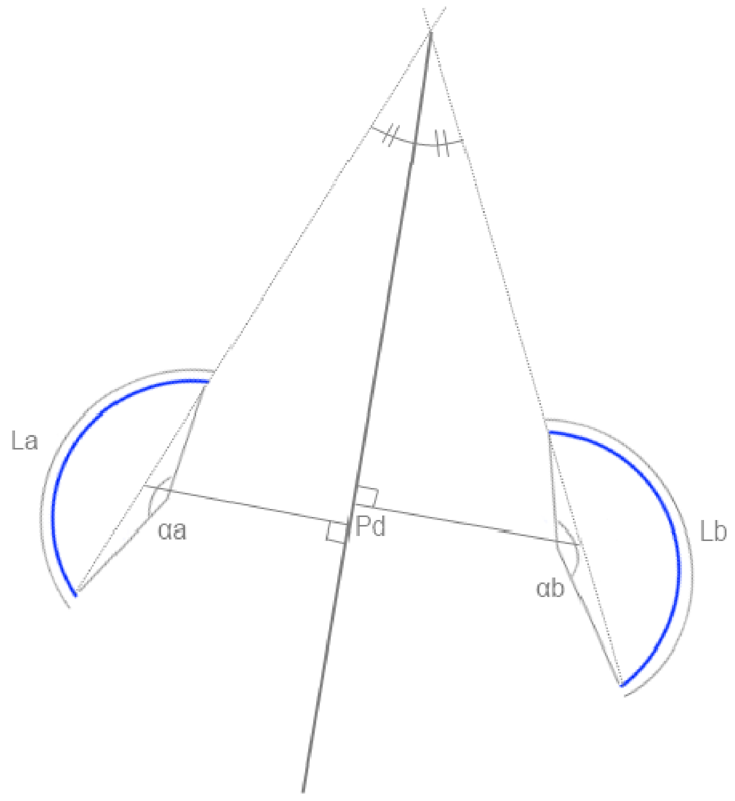


**Figure 24. Arc symmetry descriptor calculation**

Provided all the constraints are met, the associated bit is set to one. The line of symmetry is also recorded as it is used in another descriptor to identify features sharing a common line of symmetry.

**Parallel Line Descriptor**

The parallel line descriptor represents pairs of lines that are parallel, as well as being partially symmetric and in close proximity. The main constraint of parallel lines is that they must have a minimal deviation angle (α) between them. Perpendicular lines are constructed at each endpoint, and at least one of these must meet the other line to ensure partial symmetry. The distance between centre points (Cd) must be less than the average line length (La/2+Lb/2). The two latter checks are to make it more likely that the parallel lines come from the same object.

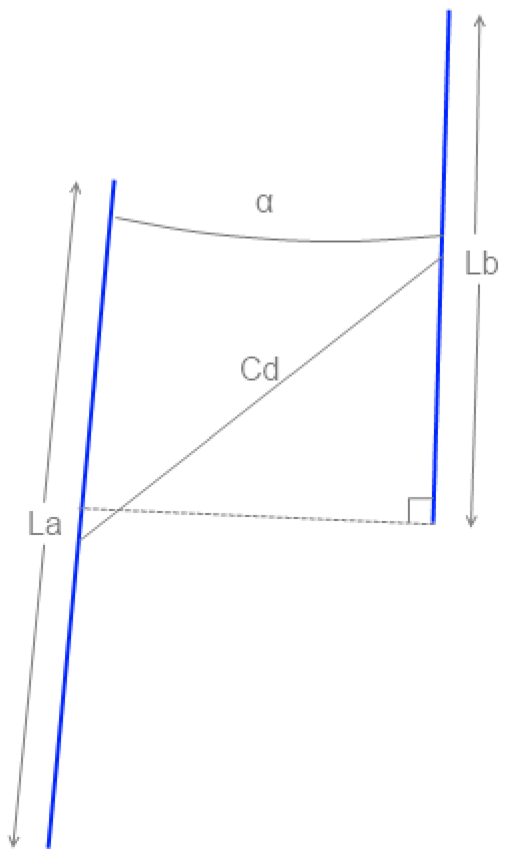


**Figure 25. Parallel line descriptor calculation**

Provided that all the constraints are met, the associated bit is set to one. The parallel line pair is also recorded as it is used in other descriptors to identify squares and rectangles.

**Line Square Descriptor**

Descriptors for line squares and parallelograms are some of the more complex descriptors present; representing a parallelogram formed by lines, and a square if the line lengths are equal. This descriptor uses the parallel line pairs stored from a previous descriptor's calculations; and also uses a shared method to find line-line 'chains'. A chain of lines is created when the line endpoints are within close proximity. The criterion for proximity factors in the line length. For example, Gap(DA) must be smaller than half of the minimum line length of either line A or B. For any closed four sided shape, there will be a looped line chain of A-B-C-D-A, for a parallelogram, the pairs A-C and B-D will also be stored in the parallel lines.

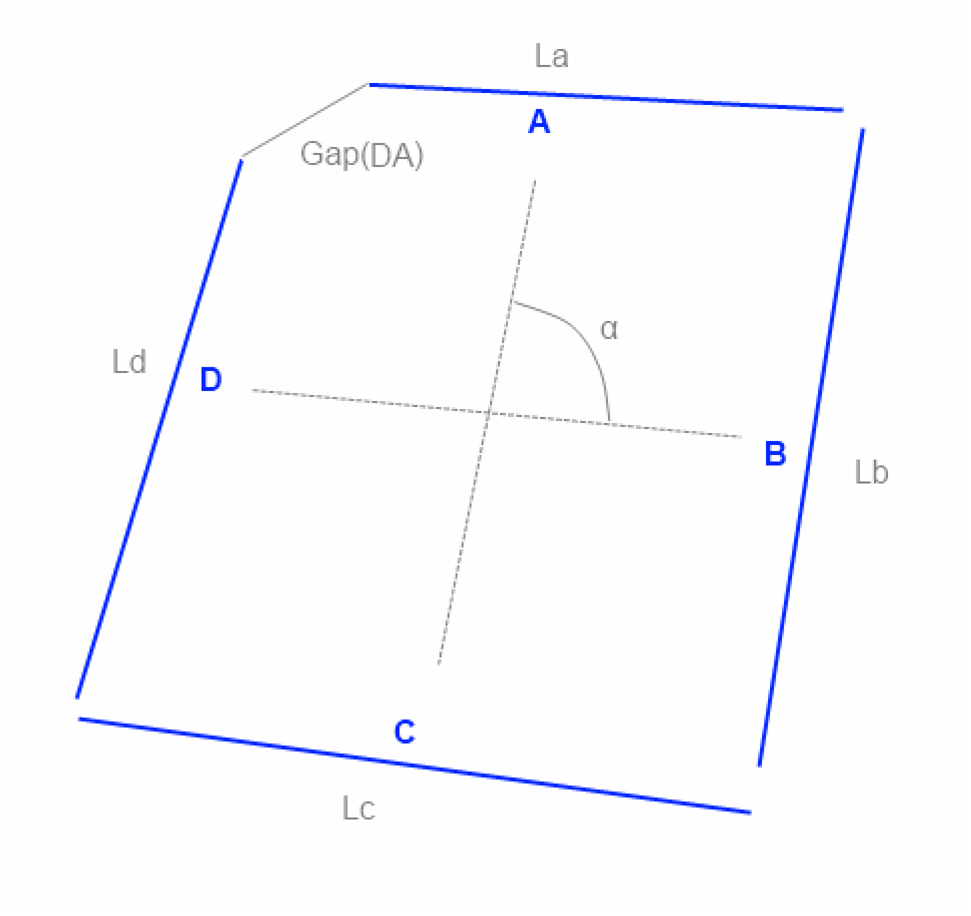


**Figure 26. Line square descriptor calculation**

For the specific case of a square, the average angles of the line pairs are compared, and must have a difference of close to 90 degrees. The lines also must have a similar length to ensure it is a square and not a rectangle.

**Other Descriptors**

The Bit Descriptor list contains 152 spatial relationships which we suggest might be used to describe the significant attributes of objects within an image. See Appendix A – Bit Descriptor List for a full listing of the descriptors used.

**Descriptor Storage and Detector Calculation**

Once all the bit descriptors are processed, they are stored as a one-dimensional array of bits. Each bit in the array relates to a specific descriptor. If a descriptor is found in the image, its associated bit will be set to 1. Storage and comparison of the descriptors is therefore very compact and quick.

The descriptors present in an image are calculated independently of the object it is compared to. Because the descriptor identification is only performed once when processing an image, it is possible to calculate and use complex relationships.

The descriptors often require a custom 'detector' method to iterate through features of the right types looking for a specific relationship. However, some detectors are assigned to multiple relationships; i.e. three sets of parallel lines in an object is just the parallel line identifier finding three separate parallel line instances. While there are some sub-methods that can calculate minor parts and be used by multiple detectors, the bulk of the calculation cannot; like the line chain used for the parallelogram/square, it is also used for a triangle detector.

The class of descriptor detectors forms the largest part of the bit descriptor system. In two-dimensional space it is a relatively simple task to write each detector using the various feature parameters, and shared sub-methods. Although there are 152 bit descriptors, the number of detectors is less than 50 due to the shared methods and some very similar descriptors.

#### 4.6.1.2. Comparison

The comparison algorithm is a very important part of any object recognition system. In this case the emphasis is placed on fast processing time, and identifying any objects likely to be in the image. Also important is identifying which objects are not likely to be within the image, but a small number of errors in this category is acceptable because it is only a first-pass system.

Both the database object and the viewed image are represented by a string of 152 bits, showing which descriptors are present for each. The comparison between the viewed image and a target object from the database is performed by comparing which bit descriptors are present in both, using a bitwise AND operation. Occlusions or multiple objects can cause the scene description to have too few or too many bits present to match the string directly so the bitwise comparison is used. An AND is used rather than an XOR so other objects and their descriptors present in the viewed image do not detrimentally affect the comparison.

Using bitwise operations makes for intrinsically fast computation, but using a managed and sorted object database could further improve this. Database software such as MySQL could be used to index the database for even faster search times.

To provide greater discrimination with very little additional processing cost, an object specific ‘weight’ is generated for each descriptor. The more often the descriptor is found with the object, and less often it is found in other images, the greater the weight. This is calculated using two probabilities; that the descriptor is present with the object, and that it is present without the object (Equation 5). The probabilities are found by processing a ‘teaching’ image set. A set of 100 images was used, with more than a quarter containing the object at various levels of occlusion. The weights were then scaled so that the total weight across all the descriptors for the object is 1.

$$P(Descriptor\ present \mid Object\ present) \times (1 - P(Descriptor\ present \mid \overline{Object\ present}))$$

**Equation 5. Descriptor weight calculation**

If a significant number of bit descriptors found in an image are also present in a database object, the weights of those found are then added to give a match score. Because the total 'weight' for each object is scaled to one, the match scores can be compared directly. Objects with a high enough match score from the viewed image are gathered in a short-list. The short-list is ranked using the object's match score, which is the likelihood that the object is in the image.

Creating a short-list with only the most likely objects narrows down the field to a handful of object images, from a potential object database of approximately 300,000 (Flemmer & Bakker, 2005). This handful of images should be far more manageable for a more intensive comparison to confirm which objects are present. The more intensive search could match the individual features, compare spatial distances, or a completely different matching algorithm such as template matching could be applied.

**Summary**

The bit descriptor method outlined in this section was designed to allow a computer to identify similar object features and relationships to those a human would use. The object relationships must be chosen and written manually. The method is not standalone and will require a final pass algorithm. With only a shortlist of potential objects remaining the final pass algorithm can afford to use more processor time to achieve greater discrimination.

### 4.6.2. Pair and Triple Blocks

The pair and triple block method sought to allow the computer to generate its own spatial relationships between the features. It does this using groups of two or three features, which can in turn be combined to form a representation of the relationship between them; hence the term 'pair and triple blocks'. The technique is briefly summarised below, with a more detailed explanation following directly afterwards.

The block data type was designed to provide a scale and rotation invariant description for different feature groups. Each block specifies the spatial relationships between its features, as well as the shape of the individual features themselves. The feature types used are lines, lobes, and arcs. Blocks contain either a pair or triple of the features, resulting in 16 block types which are described in more detail in this section (page 73 onwards). Blocks discard the 'bright-on-the-right' method of recording angle signs used by the contour extraction method; they are instead determined by the layout of the features. This was done to ensure different coloured objects would have the same blocks, and a brighter background would result in the same blocks as a dark background for the same object.

Each block contains two descriptions; referred to as internal and external definitions. The first describes the configuration of the features and relationships between them internal to the block, i.e. the shape of the block. The second is a set of four 'external' parameters, to provide scale, rotation, and position data so it can be accurately superimposed onto the viewed image, i.e. the location of the block. The descriptions are flexible enough to represent any combination of feature types in most positions. Groups of 2 or 3 features from 3 types generate 16 individual block types, each with different description parameters (pages 73 and 80). The parameters were chosen to minimise their sensitivity to small changes in feature arrangement. However this was not completely effective, resulting in a small number of specific configurations where one or more parameters are very sensitive.

The system generates tens of thousands of blocks for each object in the database. The number of blocks is greatly reduced by removing any blocks which do not offer a beneficial level of discrimination. A 'teaching' image set is used (page 90), where the objects in each image have been recorded. Blocks that are found often in images not containing the object are removed, along with those very rarely found with the object.

The comparison of the image blocks with those in that database is three-stage (see block comparison method from page 89). The first is a direct match between the blocks, to obtain a match score. Secondly a cluster is created from the matched block pair's position-dependent image to database mappings. The cluster size—both magnitude and proportion of the matched block pairs—is used to remove images which don't contain the object. Finally, the block pairs in the cluster are 'spatially' matched to ensure they are in the same relationship as for the object. The spatial cluster size is used to identify which object is in the image. Spatial matching is performed by combining two or more blocks to create 'derived blocks' (pg 87), which represent the configuration and shape of all the features contained in all the blocks used.

#### 4.6.2.1. Block Types

The block types are broken up into two main categories, those with two features (pairs) and those with three features (triples). Each block type has slightly different methods of parameter generation and comparison. There are six types of pair blocks, and ten types of triple block. The following sections describe the representation, creation, and comparison of each block type.

#### 4.6.2.2. Pair Blocks

The six types of pair blocks represent two features either 'joined' together, or in close proximity. The pair description parameters and generation methods were designed to provide a distinctive, robust, and flexible representation of two features' shape and position, while being very fast to generate.

The three feature types used by the system give six different pair types (Figure 27), with the number of parameters for each ranging from 4 to 7.

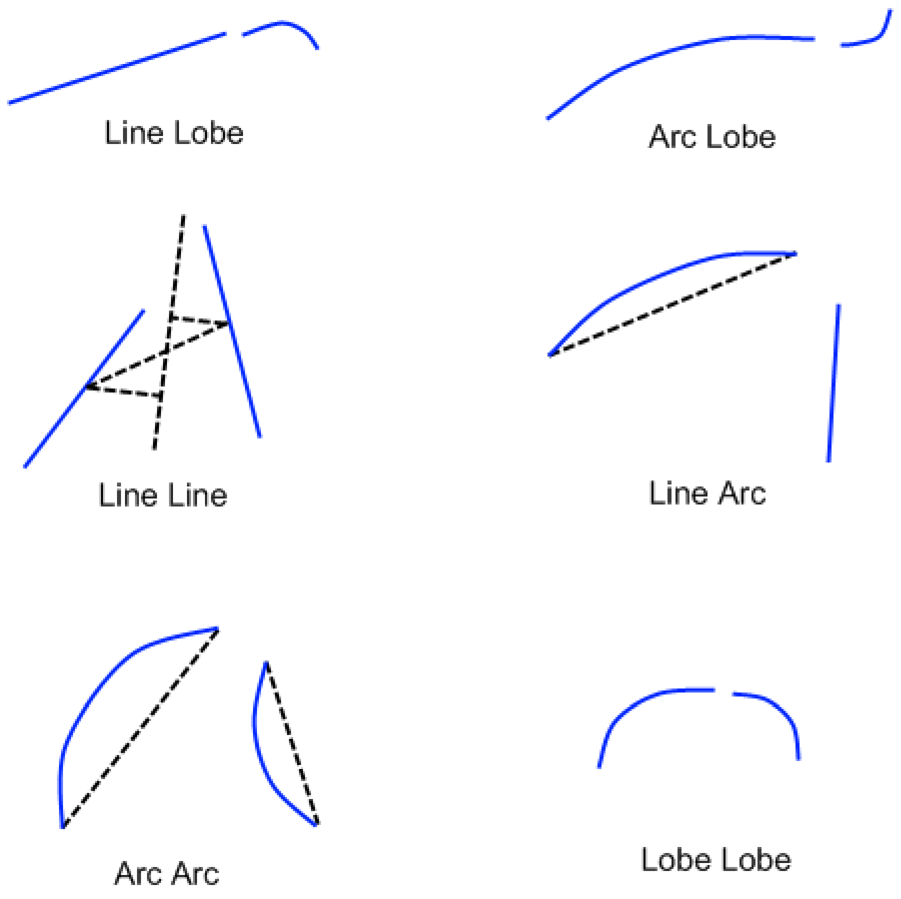


**Figure 27. Different type of feature pairs**

*The features are shown in blue, and the constructed geometry shown in some cases with dotted lines.*

Figure 27 illustrates the 6 different types of pair blocks. In most cases there are constraints to satisfy for the pair to be recorded; these affect the shape of the possible pair blocks. The constraints were added to increase the chances of both features belonging to the same object. Blocks involving a lobe must have an end-end ‘join’, satisfied by proximity and similar tangent angle. The remaining three must have their centre points within a distance which depends on the feature lengths.

The pair-block generation methods can be separated into two groups based on common parameter generation methods.

## Line-Line, Arc-Arc, and Arc Line blocks

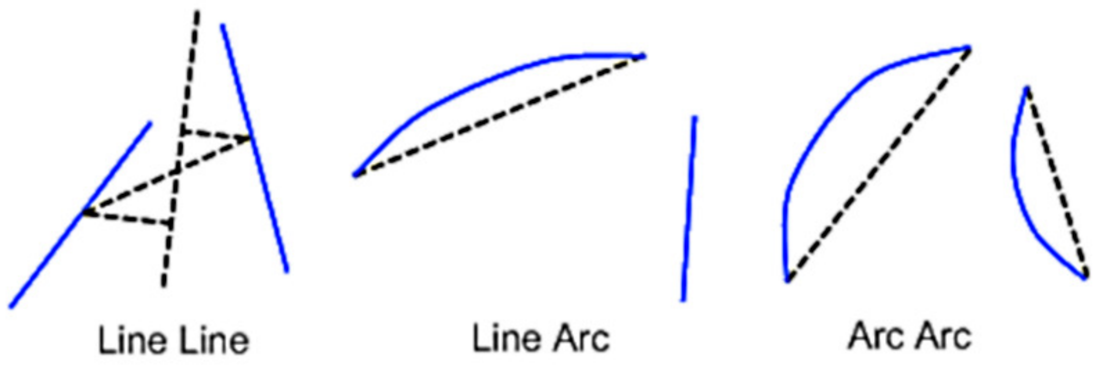


**Figure 28. Line Line, Arc Line, and Arc Arc block examples**

The line-line, arc-arc, and arc-line pair blocks all make use of the same method for the first four parameters. To make the same method process all three, those that involved an arc had an added construction line (Figure 27) using the two arc end points. This common method finds the first four parameters of each pair block. Table 5 shows all the parameters for the first three pair block types.

**Table 5. Parameters for Line Line, Arc Line, and Arc Arc blocks**

| Line Line | Line Arc | Arc Arc |
|---|---|---|
| Angle Between Lines | Angle Between Lines | Angle Between Lines |
| Midpoint Offset | Midpoint Offset | Midpoint Offset |
| Line Length A | Line Length | Arc Endpoint Distance A |
| Line Length B | Arc Endpoint Distance | Arc Endpoint Distance B |
| | Arc Position | Arc Angle A |
| | Arc Angle | Arc Angle B |
| | Arc Side of Constructed Line | Arc Side A |
| | | Arc Side B |

The common method shared by these three block types requires some simple calculation to find the angle between the lines, and the midpoint offset. The construction lines used in the parameter calculations are shown only on the line-line type for simplicity (Figure 28), but are used for the line-arc and arc-arc as well. First a line is plotted between the centre points of the two feature lines (or construction lines in the case of an arc). The angle difference must be found in the right 'quadrant' according to the line directions. If instead of finding the correct quadrant a simple average, or smallest angle calculation were used, it would show a small difference in the case of a very obtuse angle between the feature lines. To account for this the feature lines' intersection point is obtained, and the direction required to bisect the two feature lines is calculated. The angle-between-lines is found by calculating the angle contained by the quadrant the bisector line passes through. The bisector direction is also used to plot a constructed centre line between the feature lines. Lines perpendicular to the constructed centre line and beginning at the feature lines' midpoints are used to calculate the offset (along the bisector line) between the features.

The four parameters calculated by this shared method are; the angle between the lines, a midpoint offset, and the two lengths associated with the original features. Scale independence is provided by dividing the length parameters by the block's scale (from its external parameters, see page 78).

Parameters for the line-arc and arc-arc blocks are added that give its included angle, and describe which side of the construction line the arc lies on; inside or outside given by a 0 or 1 respectively. For the line-arc block an extra parameter also shows whether the LHS (left hand side) or RHS (right hand side) feature is the arc. Any blocks which have an insufficient feature length to middle point distance ratio are considered too far apart and discarded. This is to help ensure the features used for a pair come from the same object.

### Line-Lobe, Arc-Lobe, and Lobe-Lobe blocks

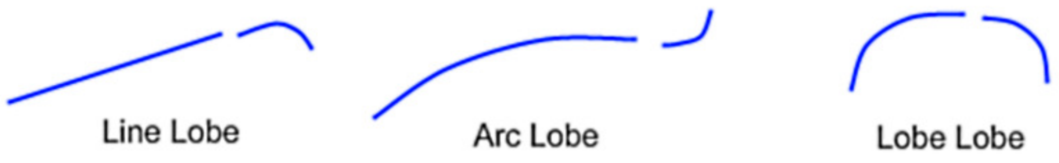


**Figure 29. Line Lobe, Arc Lobe, and Lobe Lobe block examples**

Line-lobe, arc-lobe, and lobe-lobe pairs are based around an apparent join between them. Blocks can only be created from feature combinations that have end points in close proximity (given by a proportion of feature length) and with similar end tangent angles.

The parameters used (Table 6) include the angles and lengths of features where applicable. There are no parameters for the shape of the layout, as these pair block must always be joined tangentially from end to end.

**Table 6. Parameter for Line Lobe, Arc Lobe, and Lobe Lobe blocks**

| **Line Lobe** | **Arc Lobe** | **Lobe Lobe** |
|---|---|---|
| Lobe Angle | Lobe Angle | Smaller Lobe Angle |
| Lobe Length | Arc Angle | Larger Lobe Angle |
| Lobe Skewness | Lobe Length | Smaller Lobe Length |
| Lobe Kurtosis | Lobe Skewness | Smaller Lobe Skewness |
| | Lobe Kurtosis | Smaller Lobe Kurtosis |
| | | Larger Lobe Skewness |
| | | Larger Lobe Kurtosis |

Few calculations are needed for this block type. Potential tangent joins are checked by comparing Pythagorean distance and tangents for the feature end points. The parameters are obtained directly from the features. Scale independence is provided by dividing the length parameters by the block's scale (from the external parameters). The lobe (or smallest lobe in the case of lobe-lobe pair) is considered to be the second feature, as the order is important when assigning the angles. The angles are stored as positive for clockwise rotation and negative for anticlockwise with respect to the direction from the first to second feature. The lobe skewness and kurtosis are also given.

**Pair Block Location Data**

The external parameters; scale, rotation, and position are calculated for each pair-block. This data is used for the cluster and spatial matching steps later in the algorithm (pg 93 and 97). The parts used from each pair-block to calculate these values are shown in Figure 30. The vector gives direction, a solid black line for scale, and the point provides the X and Y position values.

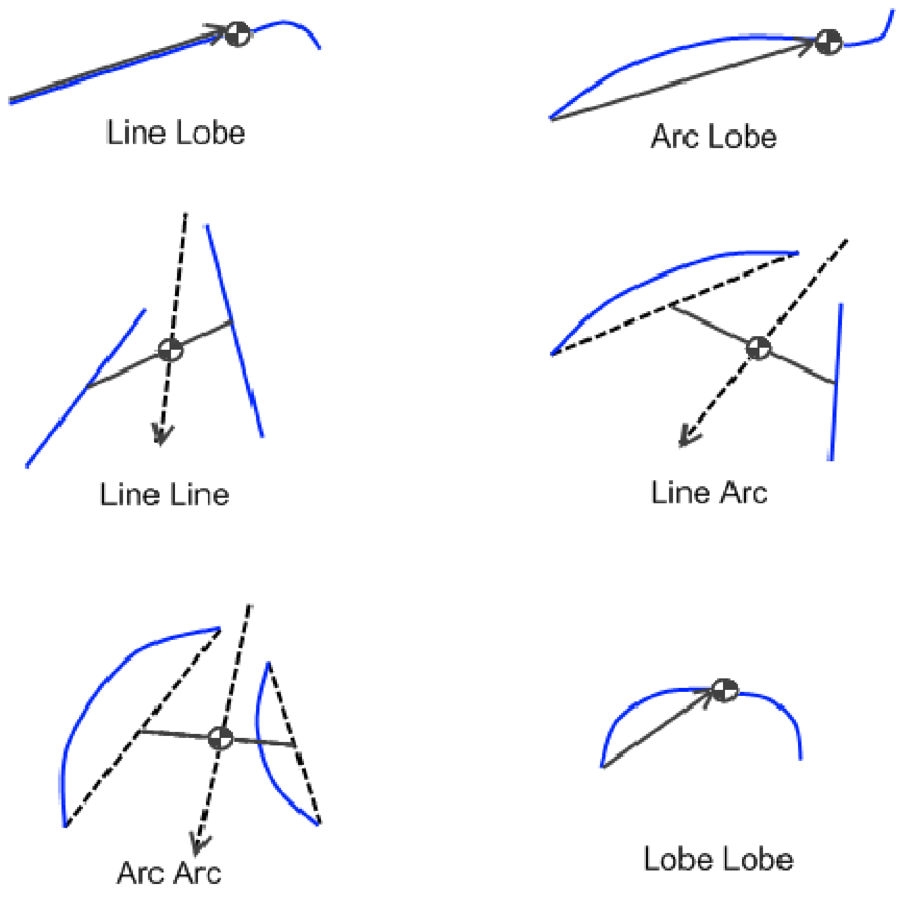


**Figure 30. Scale, direction, and position data for pair blocks**

For the three types containing lobes, the scale and direction are given by a vector travelling from the start point of the first feature to its end point. The position is the average of the two features' 'joined' end points. For the remaining three types, the position is the midpoint of the line constructed between the feature lines. The scale is given by the length of this constructed line, and the direction given by the constructed centre line, always going away from the end where the two feature lines intersect.

The direction given by the bisector can be very susceptible to small angle changes in the original features if they are close to parallel, a small change in feature angle can flip it 180 degrees. To account for this; the system produces two blocks with reversed direction vectors when the feature angle is close to parallel.

#### 4.6.2.3. Triple Blocks

A feature-triple block is designed to allow the system to represent a group of three features in any position. Using three features mean their positions can be used to form a triangle with a discriminating aspect ratio. Its description parameters and generation methods were designed to provide a distinct, robust, and flexible representation of any three features.

The lines and arcs are processed as infinite length lines and circles for triples. The larger number of features used allowed some end point dependent parameters to be discarded while maintaining a distinctive block representation. The lack of end point parameters speeds up processing, and—provided the majority of the feature is correctly located—reduces errors associated with incorrect end point identification. A triangle is generated from three feature centre points or three constructed points, using the angles of the triangle gives a scale independent representation of their spatial relationship.

The ten different types of blocks (arising from different feature combinations) are stored in separate arrays, in both the database and the data extracted from the image. Triples have the same first two parameters across all types; these parameters describe the aspect ratio of a triangle formed by three feature points. Using the same two first parameters across all triple types will simplify and speed up the comparison stage of the algorithm.

The algorithms used to generate triple-blocks are more standardised than for pair-blocks; in this case circles and lobes are processed similarly as points. This separates the block generation algorithm for triples into four main types (Figure 31). The circles and lobes are distinguished by a few parameters added on to the end of the description, making ten different types of triple-blocks.

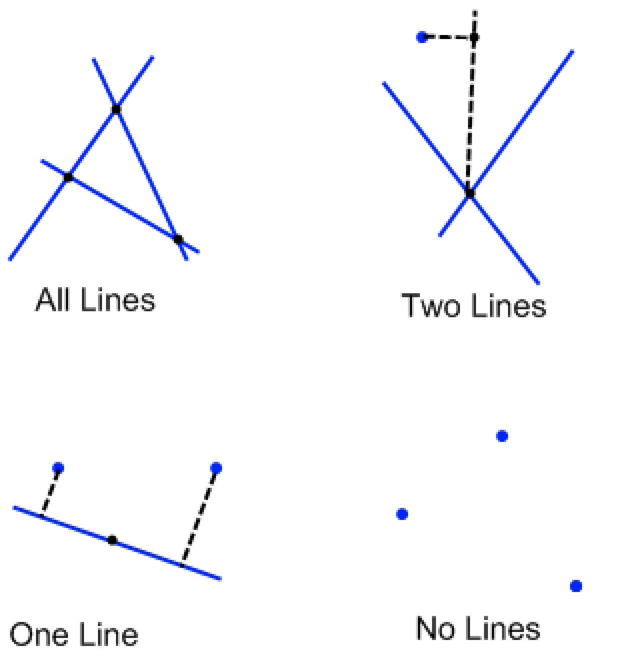


**Figure 31. Four main triple types**

The four main triple types each have a separate block generation method. The method used by the system depends on the number of lines present in each triple, which varies from zero to three. The circles and lobes are treated as points, with the centre point used for the circle, and the midpoint used for the lobe. Some sub methods are used by all four methods, such as the triangle parameter calculations.

The following tables (Table 7, Table 8, Table 9, and Table 10) show the parameters used for each block type; they are separated by the method used to generate each block type.

**Table 7. All Line block parameters**

| |
|---|
| Tip Triangle Angle |
| RHS (right hand side) Triangle Angle |

**Table 8. Two Line block parameters**

| **Line Line Lobe** | **Line Line Circle** |
|---|---|
| Tip Triangle Angle | Tip Triangle Angle |
| RHS (right hand side) Triangle Angle | RHS (right hand side) Triangle Angle |
| Point Index (0=tip, 1=RHS, 2=LHS) | Point Index (0=tip, 1=RHS, 2=LHS) |
| Angle Between Lines | Angle Between Lines |
| Lobe Length | Circle Radius |
| Lobe Angle | |
| Lobe Tangent | |
| Lobe Skewness | |
| Lobe Kurtosis | |

**Table 9. Two Point Block Parameters**

| **Circle Circle Line** | **Lobe Circle Line** | **Lobe Lobe Line** |
|---|---|---|
| Tip Triangle Angle | Tip Triangle Angle | Tip Triangle Angle |
| RHS Triangle Angle | RHS Triangle Angle | RHS Triangle Angle |
| Line Direction | Line Direction | Line Direction |
| Line Index | Circle Index | Line Position |
| Radius of ‘Last’ Circle | Lobe Index | ‘Last’ Lobe Length |
| Radius of Remaining Circle | Circle Radius | ‘Last’ Lobe Angle |
|  | Lobe Length | ‘Last’ Lobe Tangent |
|  | Lobe Angle | ‘Last’ Lobe Skewness |
|  | Lobe Tangent | ‘Last’ Lobe Kurtosis |
|  | Lobe Skewness | Remaining Lobe Length |
|  | Lobe Kurtosis | Remaining Lobe Angle |
|  |  | Remaining Lobe Tangent |
|  |  | Remaining Lobe Skewness |
|  |  | Remaining Lobe Kurtosis |

**Table 10. All Points Block Parameters**

| **Lobe Circle Circle** | **Lobe Lobe Circle** | **Lobe Lobe Lobe** | **Circle Circle Circle** |
|---|---|---|---|
| Tip Triangle Angle | Tip Triangle Angle | Tip Triangle Angle | Tip Triangle Angle |
| RHS Triangle Angle | RHS Triangle Angle | RHS Triangle Angle | RHS Triangle Angle |
| Lobe Index | Circle Index | Tip Lobe Length | Tip Circle Radius |
| Lobe Length | Circle Radius | Tip Lobe Angle | RHS Circle Radius |
| Lobe Angle | ‘Last’ Lobe Length | Tip Lobe Tangent | LHS Circle Radius |
| Lobe Tangent | ‘Last’ Lobe Angle | Tip Lobe Skewness | |
| Lobe Skewness | ‘Last’ Lobe Tangent | Tip Lobe Kurtosis | |
| Lobe Kurtosis | ‘Last’ Lobe Skewness | RHS Lobe Length | |
| ‘Last’ Circle Radius | ‘Last’ Lobe Kurtosis | RHS Lobe Angle | |
| Remaining Circle Radius | Remaining Lobe Length | RHS Lobe Tangent | |
| | Remaining Lobe Angle | RHS Lobe Skewness | |
| | Remaining Lobe Tangent | RHS Lobe Kurtosis | |
| | Remaining Lobe Skewness | LHS Lobe Length | |
| | Remaining Lobe Kurtosis | LHS Lobe Angle | |
| | | LHS Lobe Tangent | |
| | | LHS Lobe Skewness | |
| | | LHS Lobe Kurtosis | |

A number of parameters for the triple blocks are common across the different block types, making processing and notation simpler. The triangle formed by the three features has a 'Tip' where the angle is smallest, a right hand side (RHS) which is the corner clockwise from the tip, and the remaining corner is the left hand side (LHS). This notation is used to describe the angle of the triangle corner directly, e.g. 'tip angle', or a parameter of the feature located at that point, e.g. 'LHS Lobe Kurtosis'. 'Last and 'remaining' are used in the same way; where there are two features of the same type the last is the furthest clockwise from the tip, and the remaining is the one whose parameters are not yet recorded. Some of the features parameters give an 'index' to show which position they are in, where 0 is the tip, 1 is RHS and 2 for the LHS.

### Triangle Parameter Calculation

Figure 31 shows the how the triangle corner points are assigned for each of the main types; black for constructed geometry, blue for feature geometry. The triangle points for no lines or all lines blocks are simply a case of the feature points and line intersection points respectively. For a two line triple the intersection point and the feature point are used, as well as a point located on the line bisecting angle (out through the same quadrant as the point) and a line normal to the feature point. For a single line group, two feature points are used, and lines are constructed from these points perpendicular to the line feature, the middle point between the two intersections is used as the triangle's third point.

The three points give a triangle with a specific aspect ratio, characterised by three angles. The most acute is retained and its point assigned as the triangle tip, the right hand point's angle is retained as well, and the third angle is not used as it is easily derived from the other two.

### Additional Parameter Calculation

Other parameters indicate where different feature types are located on the triangle; tip, left hand or right hand side. For the two-line case the angle between the lines is given as an extra parameter. When a circle is involved, a radius is given and is normalised with respect to the triangle's scale. If a lobe is present the angle, skewness, and kurtosis are given, along with a normalised tangent; given with respect to the block's direction.

The standardised nature of the triple blocks means that the four main block generation methods are very similar. They generate the same types of parameters using the same simple calculations, so the four methods require no individual explanation.

### External Block Data

Each triple also contains the external data parameters. Scale, rotation, and position are used to locate the block within the image. This is done once the triangle is already defined, and is the same for all feature-triple block types. First a line is plotted from the centre of the back side to its tip (shown in Figure 32). The centre of this line gives the position data, the length provides the scale, and the angle (towards the tip) is used as the block direction. This information is useful for normalising scale dependent parameters, creating derived blocks, and for the later clustering and spatial mapping.

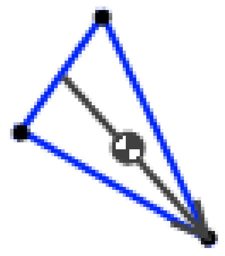


**Figure 32. Triple scale, rotation and position assignment**

Triples have a few parameters sensitive to small amounts of change in feature position. The risk is that for certain shapes of triangle like the equilateral, a small amount of feature position change can cause features to be assigned to different points. However, only a small proportion of combinations will be susceptible, and those that do not offer good discrimination will not make it into the database due to the learning algorithm.

#### 4.6.2.4. Derived Blocks

Pair or triple blocks can be combined with one-another to form derived blocks; two or more derived blocks can also be combined into a single derived block. This is used to create and compare blocks representing a much larger number of features than pair or triple blocks can. This ‘spatial matching’ comparison process is described in more detail in the following sections (page 97).

Any type of block can be grouped with another, to form a derived block. The derived block contains the two original blocks, and a scale-/rotation-/position-invariant description of the relationship between them (stored in the contained blocks external definition), and an external definition as before.

The derived block’s external definition (scale, rotation, x and y position) is calculated by a simple average of those contained within. This can create an issue if the directions are opposite, as the average can be out by 180 degrees for nearly identical input blocks. This is overcome in the comparison method by checking for a match with a reversed block direction as well. The external parameters are used to locate the block within the image, which is used to create more derived blocks and to debug the process.

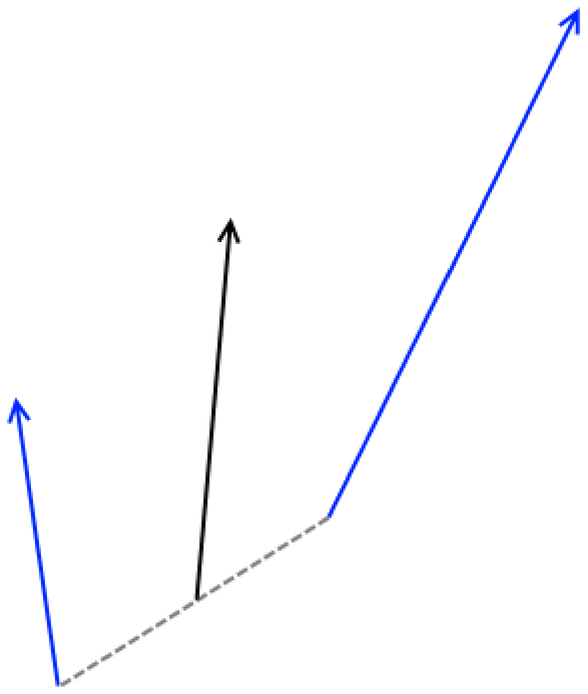


**Figure 33. Derived block external data calculation**

*The two blue lines represent the scale, heading, and position (base of vector) of the two blocks, while the black represents the same data from the derived block.*

For a later comparison stage, the external definition of the contained blocks must be made scale-/rotation-/position-invariant so that blocks within the derived blocks can be compared directly. The normalising calculations are performed using the created derived block's external parameters. The data is translated into the same 'space' as the created derived block. Scale is normalised to a proportion of the created block's scale given by dividing the contained block's scale by the derived block's scale. The direction is the difference to the contained block's direction from the derived block's direction. The position is normalised to the derived block's direction, and scale; so the calculated offset values are given as a proportion of the derived block's scale, and the y axis is coincident with the derived block's direction.

**Table 11. Example derived block value calculation**

| | Initial Block A | Initial Block B | Derived Block | Block A within derived block | Block B within derived block |
|---|---|---|---|---|---|
| Scale | 30 | 50 | 40 | 0.75 | 1.25 |
| Direction | -0.3 | 0.7 | 0.2 | -0.5 | 0.5 |
| X position | 80 | 105 | 92.5 | -0.3 | 0.3 |
| Y position | 130 | 150 | 140 | -0.26 | 0.26 |

Table 11shows an example numerical solution to the derived block parameter calculation of the blocks shown in Figure 33.

**Summary**

The feature-pair and feature-triple blocks provide the system with a method to quantify the important aspects of spatial relationships between small groups of features. The implementation places emphasis on an accurate reconstruction, scale/rotation/position invariant parameters, which are also robust to small changes. However, there were a few specific cases where robust parameters were not achieved.

The ability to derive blocks to create more and more complex relationships will be used in the final, spatial matching, stage of the system.

#### 4.6.2.5. Block Comparison

The method used to compare the blocks from a viewed image to those in the object database will have a large effect on the speed and accuracy of the system.

A learning algorithm (page 90) is used to identify and separate only the target object blocks offering good discrimination. This enhances the results and creates a smaller set of blocks to process.

The comparison is three-stage; first the blocks in the viewed image are compared directly to the target object blocks (page 91). If there are enough matches to a database object, the matches are passed to a clustering algorithm (page 93). If the clustering algorithm yields good results the matches are passed to the final part of the system, the spatial matching algorithm (page 97).

The following sections describe in detail the four different parts of the block comparison and matching algorithm.

### 4.6.2.6. Learning Algorithm

The purpose of the learning algorithm is to identify, retain, and assign a 'weight' to the blocks that offer significant discrimination for the object. The learning algorithm requires an object image, and an annotated learning set containing some images with the object and some without.

It is expected that some blocks will be very common across different objects and scenes, or be too specific to the target image of the object stored in the database; in either case providing poor discrimination. Retaining only those blocks known to offer good discrimination is expected to enhance both the processing speed and accuracy of the system.

The blocks with a high degree of discrimination are identified through the use of the teaching image set, with a rating applied to each block to show how discriminatory it is. Firstly, all blocks found in the target object image are generated. The teaching image set blocks are then generated and matched with the object blocks. This allows the calculation of two conditional probabilities for each object block. One for the chance the block is present when the object is in the image, and one for when there is no object present. Blocks that are often present with the object, but rarely found without, will provide very good discrimination.

$$P(Block\ present \mid Object\ present) \times (1 - P(Block\ present \mid \overline{Object\ present}))^{20}$$

**Equation 6. Block weight calculation**

Equation 6 shows the formula used to calculate an effectiveness rating for each block. A block's weight can vary from 0 to 1, which ensures a consistent and simple range for all ratings. Blocks with a high enough rating are stored in the database, the rest are discarded. Only those with a rating above 0.1 are stored; this value was chosen to allow an order of magnitude difference between the highest and lowest rated blocks used. The exponent of 20 was chosen by experimentation to ensure that only blocks very unlikely to appear in other images will be stored, reducing processing time and increasing accuracy.

The use of a learning set and block rating allows the number of comparisons required for directly matching the blocks to be vastly reduced. This will speed up the process significantly, and will probably increase the accuracy as well.

## 4.6.2.7. Block Comparison Pass

The first stage of the comparison algorithm compares all blocks found in the viewed image with the target objects stored in the database. The number of matches is recorded for each target object, and those with enough matches are further processed by the system.

The database is arranged with different tables for each block type, and each type is ordered by the first parameter of the block's internal definition (described in detail from pg 71). Extra data is included in each table (Table 12) to provide the object's name, the block's index, and the block's weight. This is done because blocks from different objects are all stored in the same table to reduce the block search time. The table is organised by the first parameter in ascending order.

**Table 12. Circle-circle-circle block database table columns**

| Triangle Tip Angle (search key) | Triangle RHS angle | Tip Circle Radius | RHS Circle Radius | LHS Circle Radius | Object Name | Block Index | Block Weight |
|---|---|---|---|---|---|---|---|

The flowchart (Figure 34) shows the steps taken in the algorithm to calculate a match score for each image.

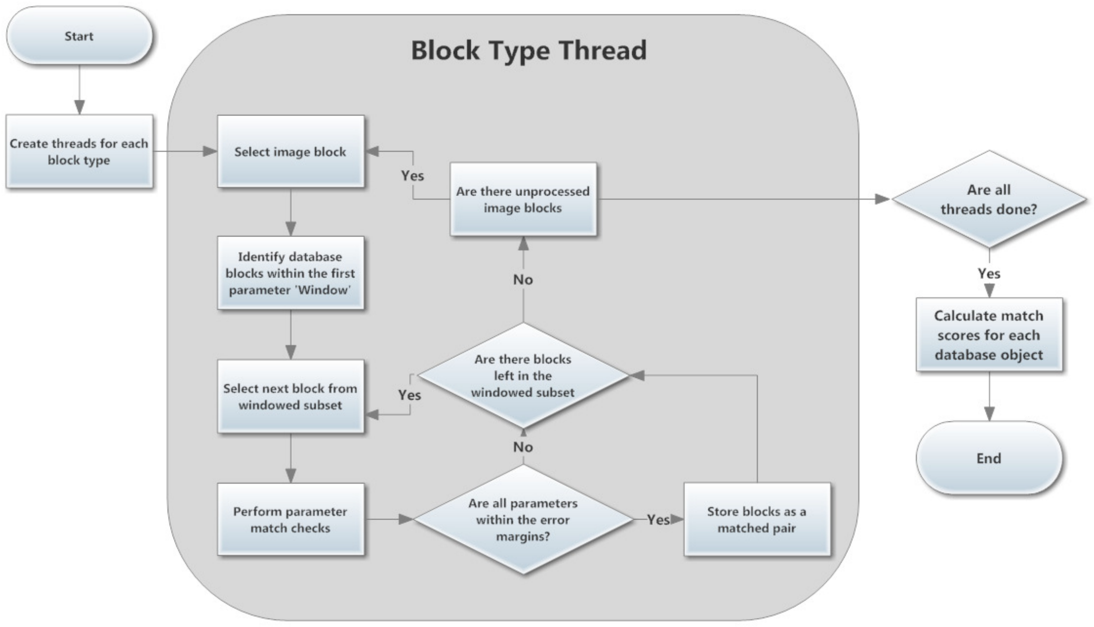

Start
Create threads for each block type
Block Type Thread
Select image block
Yes
Are there unprocessed image blocks
Identify database blocks within the first parameter 'Window'
No
Select next block from windowed subset
Yes
Are there blocks left in the windowed subset
No
Perform parameter match checks
Are all parameters within the error margins?
Yes
Store blocks as a matched pair
Are all threads done?
Yes
Calculate match scores for each database object
End

**Figure 34. Block comparison flowchart**

Each block type is processed by a single thread; this is simple because the different types are stored in different tables.

The algorithm iterates through each block of the given type from the viewed image, comparing them with the database blocks from all objects. Firstly, two binary searches of the ordered first parameter give a two index ‘window’ to vastly reduce the blocks left to search.

Blocks within this window are processed parameter by parameter. The parameters are compared directly and blocks with any single parameter outside the error range are discarded. A cumulative error approach was also considered, but would incur a higher processing cost. The acceptable error values are stored in arrays for each block type (Table 13), and allow the parameter’s error to be tuned for each block type.

**Table 13. Arc-lobe example error values**

| Lobe Angle | Arc Angle | Length Proportion | Lobe Skewness | Lobe Kurtosis |
|---|---|---|---|---|
| 0.2*π | 0.75*π | 0.15 | 0.4 | 0.4 |

The tuning of these values was done by a best guess to start with; then refined by evaluating the blocks found to match, or the parameters causing non-matches. Blocks found to match are stored in an array of matches with their associated object name, index, and rating.

Once all the blocks in the viewed image have been processed, the matched blocks are used to generate a match score for each target object. The match score is given by adding the ratings of found blocks for each object, and dividing by the sum of the ratings in that particular object; meaning the score can range from zero to one.

If the match score is high enough it is probable the object is in the viewed image. The match score threshold used is 0.06, found through experimentation in the 'Bit Descriptor vs Blocks' test on page 118. Objects selected to go through the clustering matching algorithm have their associated block matches retrieved and stored in an array for each object; the rest of the results are discarded.

## 4.6.2.8. Clustering

The second stage of the matching algorithm is clustering the matched blocks. The cluster uses the block's external parameters of scale, rotation, x, and y data. A mapping is calculated for each matched pair from the previous block matching step. This mapping contains the adjustment required to translate the block from its position, scale, and rotation in the viewed image to the database image. If a number of matches come from the same object, they will have very similar mappings resulting in a large cluster.

**Mapping Calculation**

Each mapping contains four transform values, scale, rotation, x offset, and y offset. The scale describes the size of the viewed image block relative to the target object block; its transform is calculated by dividing the viewed block scale by the target block scale. The rotation describes the difference in direction from the target block to the viewed image block; its transform is calculated by subtracting the target block direction from the viewed image block direction. The x and y offsets describe the position of the viewed image block relative to the target block within the images; and are normalised for the scale and rotation changes.

To calculate the position mapping the scale and rotation effects must first be removed. The target block's x and y values are transformed by the scale and rotation mapping changes to remove these effects on the position change part of the mapping. If the position changes were directly compared, a simple rotation change would result in large position differences at the outside parts of an object, despite it still being in the centre of the image. Applying the mapping to the target blocks allows the rotation and scale components to be removed; so in the case of the simple rotation no position change would be found. The dependence is removed by plotting a vector from the centre of the image to the block's position. The vector's length is scaled by the scale transform value, and the rotation transform is added to its direction. The difference between the new vector endpoint and the viewed block position are used as the x and y transform values. Simply using the centre of the image will mean the target blocks are moved across the image, and introduces a position change that does not reflect the objects actual position in the image. However, this added change is applied uniformly, and will not affect the clustering process. With all four parts of the mapping calculated for the blocks, the clustering of these values can begin.

**Generic Quality Threshold Clustering Algorithm**

The Quality Threshold clustering algorithm requires the specification of the threshold distance within the cluster and the minimum number of elements in each cluster (Naik, 1999). From each data point all the points within this threshold distance are found and stored with the cluster. The largest cluster is regarded as the true cluster, and its points are removed from any others. The process is repeated until no more clusters of sufficient size can be formed.

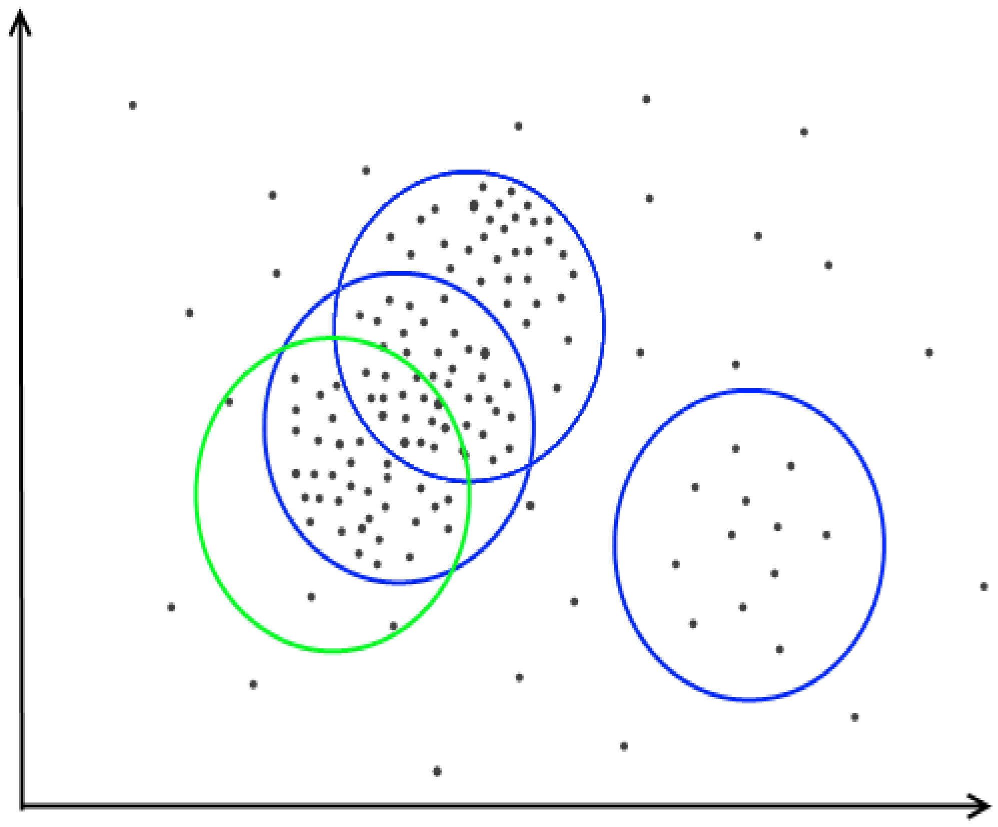


**Figure 35. Two dimensional QT clustering example**

*The blue circles show the first pass, they are all the same size to reflect the specific threshold distance. The green circle shows how a cluster (left most blue circle) is 'moved' due to the removal of points on the subsequent passes.*

Naik gives some advantages and disadvantage of this approach.

**Advantages**

- Quality Guaranteed - Only clusters that pass a user-defined quality threshold will be returned.
- Number of clusters does not require specification.
- All possible clusters are considered - a cluster is generated with respect to every data point and tested in order of size against quality criteria.

**Disadvantages**

- Computationally intensive and time consuming - Increasing the minimum cluster size or increasing the number of data points can greatly increase the computational time.
- Threshold distance and minimum number of element in the cluster must be specified.

**Implementation**

The matched block pairs have been stored in two arrays, and each matched pair is passed directly to the clustering algorithm. The first step calculates all the mappings, scale and rotation are calculated first so they can be used to calculate scale and rotation independent position mappings.

The QT clustering algorithm was implemented as above, with one significant difference to save time. The maximum number of clusters was set. Because there must be a large number of matches contained in a cluster, it was found that there would often be over 20 clusters converging on the same set of matches. Setting a maximum number instead of using every point reduced the number of convergent clusters to only a few, and vastly reduced the processing time.

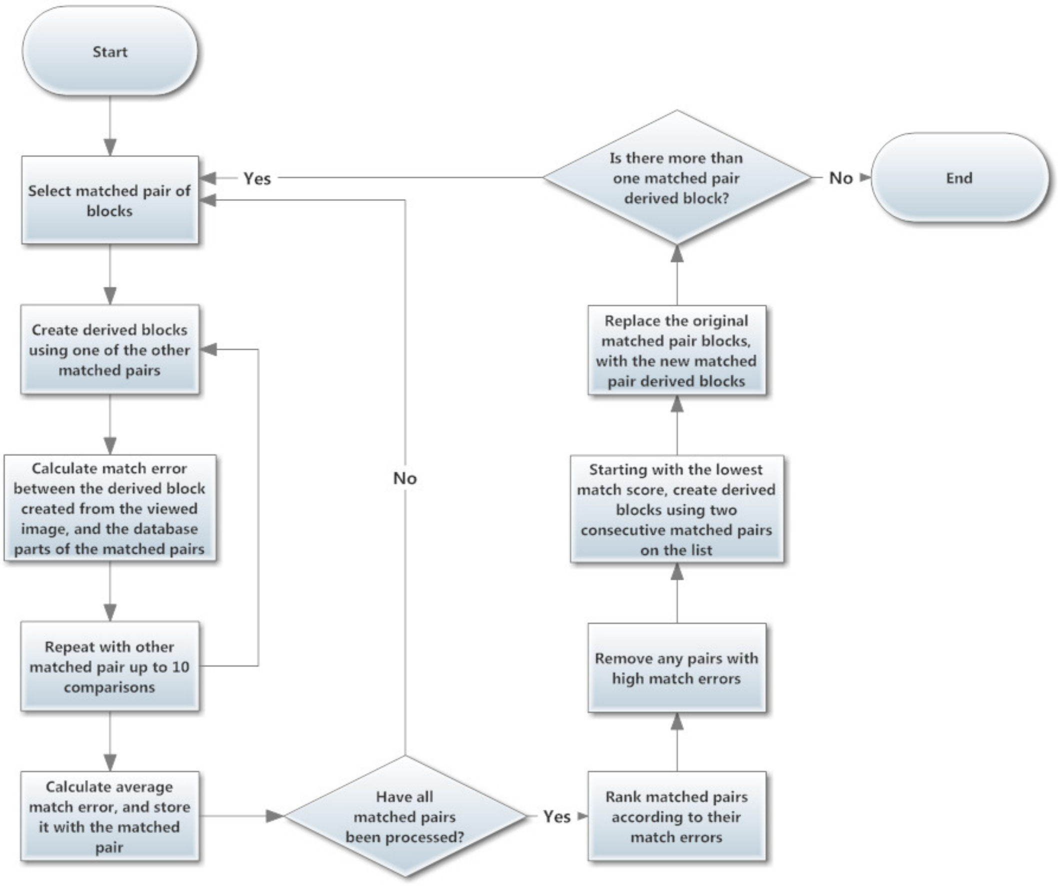
The constraints to pass this stage of the matching algorithm are a minimum cluster size, and a minimum proportion of the previously matched blocks. If these are met the blocks contained in the largest cluster were passed on to the final stage for spatial matching.

Currently the system is limited to only identifying one of each object in an image. However, it is expected to be a simple task to enable the clustering algorithm to return multiple block groups if multiple objects are likely.

The maximum error was required to be large, to ensure blocks that should be in the cluster made it through. The main reason for this was large deviation in the block's external parameters. The calculated position mapping is affected by errors in scale and rotation.

### 4.6.2.9. Spatial Matching

The final stage in the matching algorithm is spatial matching, to ensure those blocks in the same cluster are from the same object. The spatial match algorithm checks that each block within the cluster is in the same relative position in the viewed image as the target object image in the database. In theory those in the same group with the same mapping will by definition map back to the same relative positions. However, as outlined previously (pg 93) the clustering system is too sensitive to small errors, so this third match stage is required.



Start
Select matched pair of blocks
Create derived blocks using one of the other matched pairs
Calculate match error between the derived block created from the viewed image, and the database parts of the matched pairs
Repeat with other matched pair up to 10 comparisons
Calculate average match error, and store it with the matched pair
Have all matched pairs been processed?
No
Yes
Rank matched pairs according to their match errors
Remove any pairs with high match errors
Starting with the lowest match score, create derived blocks using two consecutive matched pairs on the list
Replace the original matched pair blocks, with the new matched pair derived blocks
Is there more than one matched pair derived block?
Yes
No
End

**Figure 36. Spatial matching flowchart**

This stage is more thorough than clustering, performed by utilising the blocks in a recursive fashion. Each block can be grouped with another, to form a derived block. This block contains the data in the two blocks, a scale/rotation/position invariant description of the relationship between them (stored as the contained blocks external definition), and an external definition as before. To compare two derived blocks directly some simple calculations are performed to ensure the blocks contained in each have object scale/rotation/position independent parameters representing the relationship between them.

Two blocks from both the viewed image cluster group, and the two corresponding database blocks, are used to create a pair of derived blocks. If the viewed image blocks contain features on the same parts of the objects, the scale/rotation/position invariant definition of the derived blocks will match.

To match a derived block the scale/rotation/position invariant descriptions are compared directly. The contained blocks do not need to be compared because the derived block is created from already matched pairs of blocks from the viewed image and object database. The spatially invariant relationship between the contained block is compared. A match error score is easily generated because there will always be the same four error values. A Pythagorean distance is calculated from the error values and used as the match error.

To minimise the impact and identify any spurious blocks still present, a match performance number is generated for each derived block. It is obtained by using one block pair to create ten derived blocks using ten other random blocks. The derived block is then matched with the nine others, and the match errors added. Large total match errors will signify that the block doesn't fit with many others, and is unlikely to be from the object. Those blocks with large errors are discarded by the system.

It is now much less likely to contain spurious data, but still possible. The algorithm continues recursion revising the blocks at each stage, and creating derived blocks from other derived blocks. The set of derived blocks is ordered by their total match error values, and used to form a set of derived blocks (from derived blocks) by combining any adjacent two. In this way the best fitting derived block will be combined with the second best, to form a 'higher level' derived block representing many features. The new derived blocks then replace the original blocks and the algorithm repeats. The process stops when there is only one block remaining, this may only take one or two iterations and contain very few original blocks, or take more than eight, containing hundreds of original blocks, and over a thousand features.

This single derived block contains a record of how many individual blocks were used to create it, and if this is greater than a set number (ten), and comprises more than a set proportion of the cluster (25%), the object is said to be within the image.

An added benefit of creating a single cluster is that the location of the object in the viewed image is already recorded.

**Comparison Summary**

Using a multistage comparison system is expected to give accurate results with a fast computation time. Through the stages, the algorithm becomes more specific, and more discriminatory. The first steps are primarily to reduce the number of comparisons required by the final stage, spatial matching. However, the reduction of blocks also increases the likelihood that those which make it to the final stage are contained by the object, thus increasing the overall accuracy.

# 5. Results

This section evaluates the performance of the system. The tests performed are described in detail and their results presented. In some cases the results are used to compare different methods or data types, and select those which give the best performance. The results provide insight into the limitations present in the system. Below is a brief outline of the tests performed; more detail and the actual results are provided later in this section.

A large number of tests were required to obtain thorough and meaningful results due to the complexity of the system. First the two main comparison techniques were evaluated against each other. The lesser performing technique was then examined to identify its flaws. The better technique was further tested to optimise some data types and variables; after which its capabilities for accuracy, speed, and learning were evaluated.

The better performing technique was tested, with separate tests for each of its three stages, and a further test to evaluate its learning ability. The first test evaluates the effectiveness of the different blocks types, and removes those which do not provide sufficient benefit. The clustering and spatial matching performances are also evaluated. The learning ability is evaluated, and in the same tests a minimum learning set size is established. A minimum learning set size is required because the learning sets must have a number of images containing the object and a number which do not contain it.

The system is run against multiple image sets to evaluate its performance over a diverse range of objects.

The system is finally compared against some of the better performing systems developed by others, to identify any relative strengths or weaknesses.

The majority of tests are performed on a custom image set. This set contains two database objects, and 100 test images (Figure 37, pg 104).

## *5.1. Test Method*

This section explains the methods used to test the performance of the system and its components. The tests have been chosen to isolate and evaluate both the overall results, and the performance of its component parts.

**Image Sets Used**

The test method used 'target' images and 'scene' images. The target images were of the objects, with optimal lighting and a bland background (according to human perception). The scene images sometimes contained the target object and sometimes not. They were 'real world' images and contained background and foreground clutter accordingly. A look-up table was manually developed to record to what extent a target object was present. This showed a level of occlusion from 0% (clearly visible) to 100% (not present). Also recorded were images in which the object was observed from different viewpoints, and those with a lot of background clutter.

### 5.1.1. Bit Descriptors vs Blocks

Two different methods were tested and compared to select the better one. The purpose for this test is to benchmark the performance of both methods using the same test method and image sets.

Both the block-comparison and bit-descriptor methods use spatial relationships between the line, lobe, and arc features. The main difference between them is that the bit descriptors only identify pre-programmed relationships; while the blocks are much more flexible and can represent the majority of two or three feature combinations. Exactly the same features are used for both the bit descriptor and block tests.

Bit descriptors (pg 64) are represented as a single bit indicating whether the spatial relationship is present or not. The system uses 152 bit descriptors, representing relationships such as parallel lines, squares, symmetrical features, or many others.

The block data type (pg 71) was designed to provide a scale- and rotation-invariant description for different feature groups. Each block represents the spatial relationships between its features, as well as the shape of the individual features themselves. The parameters contain sufficient information to redraw the extracted features.

Each descriptor or block found in a target image is assigned a 'weight' to show how strongly it is associated with the object. Both methods make use of a learning algorithm to assign these weights to the descriptors or blocks. The weights are calculated in the same way and are calculated for each object; which means that the same descriptor or block will have different weights when used with different objects.

A target object descriptor/block's weight is calculated (Equation 6) from two conditional probabilities; the probability that the descriptor/block is present given that the object is in the image, and the probability that it is present given that the object is not. This means that a learning subset requires a number of images both containing, and not containing, the object.

$$P(Block\ present \mid Object\ present) \times (1 - P(Block\ present \mid \overline{Object\ present}))^{20}$$

**Equation 7. Block weight calculation**

The weight calculation's exponent was obtained by manual adjustment to give the best results. The full image library was used as the learning set, to give the most accurate probabilities. It was found that lower exponents gave too much weight to common blocks and descriptors, which resulted in an increase of wrongly identified objects. Higher exponents resulted in only a few descriptors/blocks having significant weights—which were not enough for a definitive match—so objects were not identified. An exponent of 20 provided a good balance. The bit-descriptors-versus-block-tests were performed with a number of different weight thresholds, to find the optimum threshold. The calculation can return values between zero and one, and those used for matching must achieve a minimum weight.

Using our basic dataset of 100 images (Figure 37) both techniques were run and the identification rates were compared across different weight thresholds. Each test gave two results for each object; a 'true positive rate' which showed the proportion of objects identified in images that contained the object, and a 'false positive rate' which showed the proportion of objects wrongly identified in images that did not contain the object.

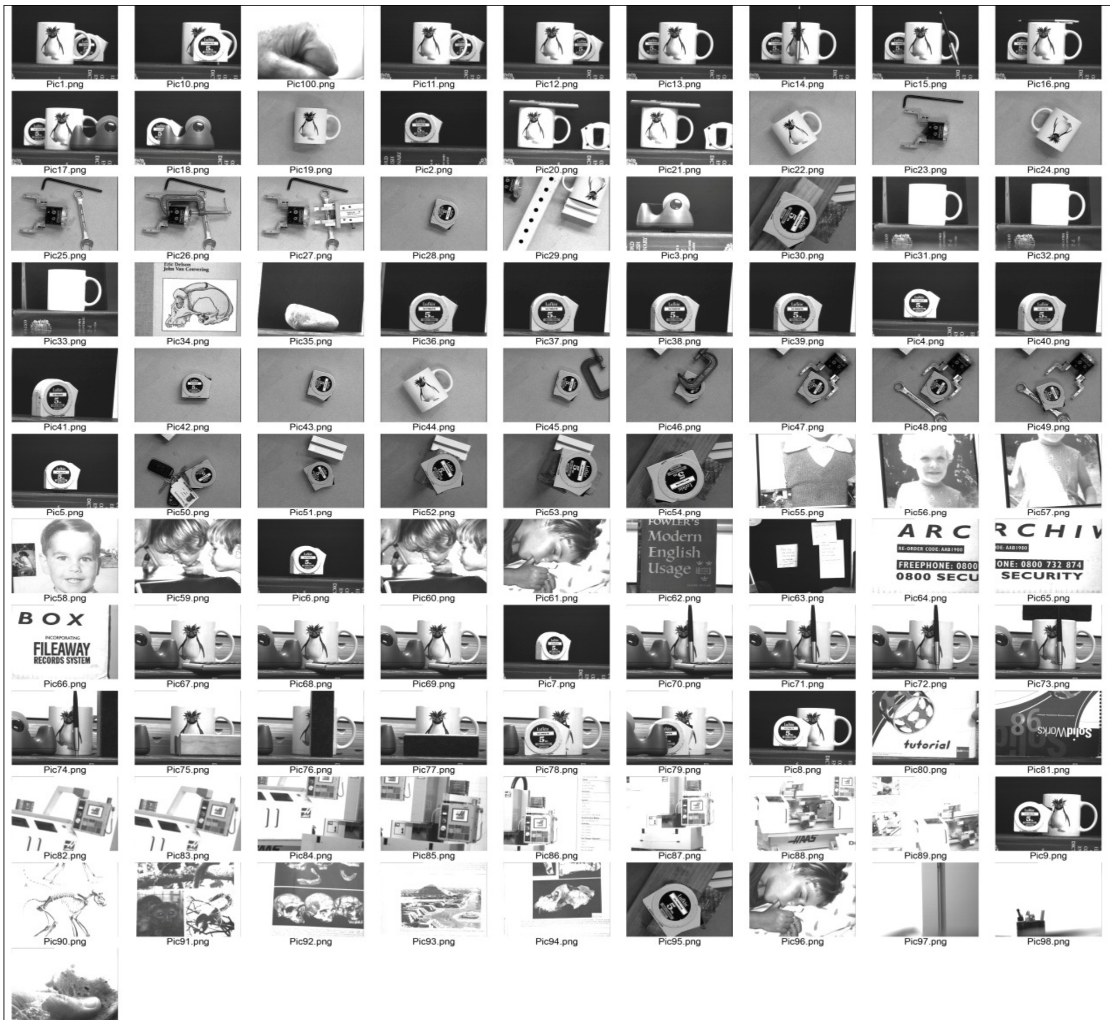


**Figure 37. Custom 100 pictures dataset**

Two different versions of the block system were tested. In one case the blocks were generated from feature combinations across the image and in the second, only the features from the same contours were used to create blocks. It was thought using blocks created from only the same contours would ensure the features used came from the same object.

The identification rates were plotted on two graphs for each method tested to show true-positive rates and false-positive rates (Bit Descriptor vs Blocks, pg 118). Three different categories were used; bit-descriptors, blocks created from all features available, and blocks created from features on the same contour. The x-axes on both show the block weight threshold used.

### 5.1.2. Bit Descriptor Flaws

Initial results showed poor performance from the bit descriptors. A test was run to see why the bit descriptors performed poorly. Knowing what caused the poor performance can help avoid similar flaws in the method used.

Bit descriptors differ from blocks because they can only identify the relationships thought significant enough to write detectors for. A match score is given by adding the weights from descriptors found in both the scene and target images then dividing by the total weight of descriptors found in the target image, to give a weighted proportion. A score of zero indicates that no descriptors were found in both images; a score of one indicates that all target-image descriptors were found in both.

If the descriptors do not have a strong association with the objects, it would account for the lower performance. The weighting is a measure of how strongly the descriptor is associated with the object, as it is calculated from the two conditional probabilities; how often it is found when the object is present and how often it is found without the object. Descriptors found only occasionally with the object will show little association, while those found often in images without the object show association but little selectivity.

To evaluate how effective the various descriptors were, a histogram of the descriptor weights for the tested objects was plotted (Bit Descriptor Flaws, pg 122).

### 5.1.3. Pairs and Triples Type Evaluation

A block's ‘type’ refers to the combination of feature types it contains. Different combinations are likely to have different worth to the method. Identifying and removing block types that do not offer significant benefits will increase the system's speed. This test's purpose is to enhance the accuracy of the system by removing superfluous data types. The test also evaluates the effectiveness of the block matching step by itself.

The method requires that the target object blocks be learned and stored. To obtain the most accurate block weights the full image set was used as the ‘learning subset’ to calculate the block probabilities. The optimal block weight threshold was identified from the results of a previous test (Pairs and Triples Type Evaluation, pg 123); blocks with a weight below 0.1 were culled.

The system compares the blocks generated for each scene image with those stored in the database for each target object. A score is assigned for each match of the database objects to each scene image. This score is calculated by adding the weights of the target object blocks found in each scene image, and dividing this by the sum of all the block weights for that target object; which gives a weighted proportion. This is then summed over all the matches.

The block generation algorithm was modified to generate only a single type, so they could be evaluated separately. Tests were performed for each of the block types, and the scores obtained were presented in charts. To allow a threshold score to be determined a plot was made of the number of true and false positives against threshold score for each block type (Figure 47).

High true-positive rates and low false-positive rates were desired across the threshold range. Also desirable was a wide, flat spot on the true-positive rates, to show that the threshold was not too sensitive. A numerical estimate of a block type's effectiveness was calculated from the area of the plots; the area under the true-positive lines divided by the area under the false-positive lines.

The tests were repeated with a lower block weight threshold, to ensure that the results were consistent if more blocks remained in the system, which may be required by later matching stages. The same tests were repeated with the ensemble of block types.

A small number of block types were removed from the system, and results gathered with this remaining set (Pairs and Triples Type Evaluation, pg 123). The performance of the now-reduced set of block types was compared with the full set, to ensure there had been no significant loss of accuracy.

### 5.1.4. Clustering

The clustering test evaluates the performance of the clustering and cluster matching algorithms.

The second stage in the matching algorithm is to find any groups of matched blocks with similar ‘transforms’. A transform is defined as the rotation, scale, and position change mapping a target block onto a scene block.

The initial block-matching step records matched pairs of blocks that are in both the scene image and a target object. Each block in a matched pair contains scale, rotation, and position information that is used to calculate the transform. Although the clustering algorithm does not use the weights learned from the teaching set directly, only matched blocks with a high enough weight can make it to this stage.

When calculating the transform, the scale and rotation parameters are calculated first. The scale proportion is calculated by dividing the scene block scale by the target block scale. The difference in angle is calculated by subtracting the target block direction from the scene image block direction. A direct comparison of the blocks position would contain changes introduced by the scale and rotation, these effect must first be removed. An example of this effect is a pure 180 degree rotation, a block located on the handle would show a position change equal to the cup width. To remove this dependency the inverse of the scale and rotation mappings are applied to the scene blocks, using the image centre as a scale and rotation centre. The direct comparison can now be performed.

Ideally, the transforms for all the matched blocks for a target object in a number of scene images would be exactly the same. However due to image inconsistencies and algorithm imperfections, there will always be a small error.

The clustering algorithm seeks to find those points that form a ‘cluster’ or closely spaced group of points. The cluster is in terms of the four parameters of the transform. The clustering algorithm seeks to identify groups that lie within four-dimensional spheres in four-dimensional transform space. The dimensions are normalised to give different error margins for each parameter.

A ‘quality-threshold’ clustering algorithm was used to find the clusters, it was chosen as it requires the maximum error to be specified for each parameter (Clustering, pg 93). The transform data was found to contain very few significant clusters (often one or none), with some scattered transforms not belonging to the object.

The clustering algorithm selects a number of random points as ‘seed points’, to grow the clusters from. The centre value of the cluster or 'centroid' is set to the seed point. The closest point to the centroid is identified and added to the cluster, the centroid is then updated to reflect the average of all points with the cluster. For each cluster, points are added until no more are left within the specified error radius.

The largest cluster is selected as the one most likely to represent a target object in the scene image, the remainder are discarded. The case of an image containing two target objects would yield different large clusters of significance; however the image sets used only contain a single instance of any object so this is not a concern at this time.

If the cluster contains more than 15 transforms, the proportion of transforms in the cluster to total transforms is recorded as the cluster score. Fewer than 15 matched blocks is not considered enough for a definitive object match and the score is set to zero.

The tests were performed without removing scene images with low scores in the initial matching step. This ensures that the performance of the clustering step will be less skewed by the results of the previous step. Normally only scene images that show a significant number of block matches with an object would be passed through to the clustering stage.

As for Pairs and Triples Type Evaluation (pages 105 and 123), plots were made of the number of true positives and true negatives versus image score (Clustering, pg 107). Again, high true-positive rates, low true-negative rates and a wide, flat true-positive curve were considered desirable.

### 5.1.5. Spatial Matching

The spatial matching test evaluates the performance of the spatial matching and spatial match comparison algorithms.

The final matching stage is spatial matching, which compares the layout of all the matched block pairs. This ensures that the arrangement of blocks in the scene matches that of the target.

The spatial matching step uses the blocks in the largest cluster. Although having a similar transform should mean the blocks will be in a very similar arrangement, the position transform's dependence on the calculated scale and rotation transform required large error margins. Matched block pairs that are not from the object can make it into the final cluster because of the larger error margins; spatial matching provides a way to remove these.

The flexible design of blocks allows the creation of ‘derived blocks’ from two or more ‘normal blocks’. The derived blocks provide a scale-/rotation-/position-independent representation of the relationship between the contained blocks. Blocks from the cluster come in pairs, one from the target, and one from the scene, and are thought to represent the same part of the object. Two pairs are used to create a derived block in both target, and scene image space. The derived blocks will only match if the contained blocks are in a similar arrangement in the target and scene images.

To make it easy to identify which block pairs do not belong to the object, each pair of matched blocks is used to create ten derived blocks with other random pairs (Figure 38). If a matched pair is not a correct match, few of the ten derived blocks will match, and if it is a true match the majority of the derived blocks will match. Pairs of matched blocks that do not perform well over the ten matches are discarded.

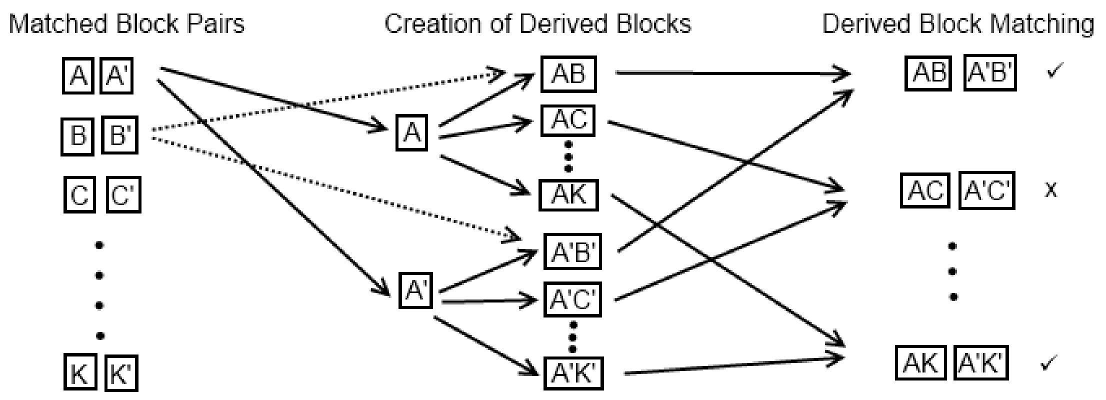


**Figure 38. Derived blocks creation process**

*A is a target block, and A' is a scene block that may match A.*

However there is still a chance that a derived block matches but its contained blocks are not from the object (false positive). A further step is taken to reduce this possibility, and a recursive algorithm is used to create derived blocks from previously derived blocks.

The derived blocks are ranked by their match scores. A match-score for each block-pair is calculated by adding the match errors across all ten derived blocks. Those with an error too large are discarded. This reduced set of blocks is then used to form a number of second-order, derived-blocks from the matched-pairs. The best performing pair is matched with the second best; then third to fourth, fifth to sixth and so on until all blocks are contained in a derived block. In the example figure (Figure 39) the performance of the block-pairs are ranked as follows; A, E, C, D, down to the bottom two, F, and K.

These second-order, derived blocks are then checked across ten random matches as before. Those that match well enough are used to create more (third-order) derived blocks, and so on, until one (nth-order) derived block contains all the normal blocks (through its layers of derived blocks) that have not been discarded.

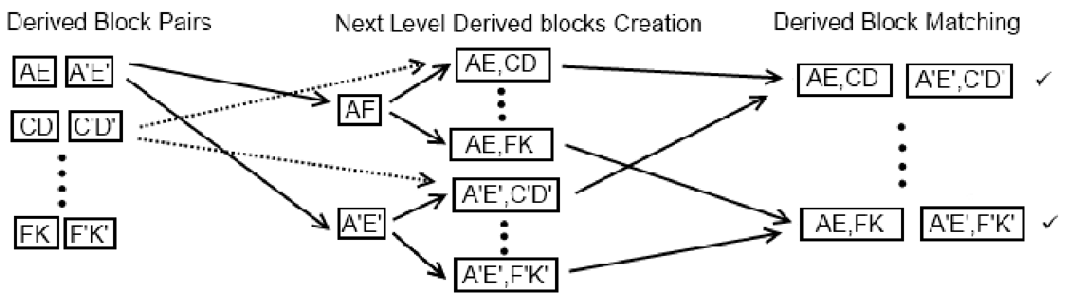


**Figure 39. Next level derived blocks creation process**

*AE,CD is a second-order, derived block containing the two, first-order, derived blocks, AE and CD.*

The spatial match score is calculated by the number of normal blocks still present in the final derived block. The score is the number of normal blocks in the final derived block as a percentage of those originally clustered. If there are fewer than ten blocks it is not considered enough to provide a definitive object match, and the score is set to zero. A threshold of ten blocks was found, through experimentation, to give the best results.

The tests were performed without a threshold on the block-matching step or the clustering step, so the performance of the spatial-matching step would not be skewed as much by the results of the previous steps. When performing a match instead of testing the system, only images that pass the selection criteria of the previous two stages would be passed through to the spatial-matching stage.

As for Pairs and Triples Type Evaluation (pages 105 and 123), and Clustering (pages 107 and 128), plots were made of the number of true positives and true negatives versus image score (Spatial Matching, pg 129). Again, high true-positive rates, low true-negative rates and a wide, flat true-positive curve were considered desirable.

## 5.1.6. Supervised Learning to Establish a Subset Size

The best size for the learning set was sought. The method learns by using a 'teach subset' of images to identify blocks that correlate with the object's presence. This test evaluates the performance of the learning component, and recommends a suitable subset size for future use. This test also establishes the results of the full system on our image set.

The method fundamentally responds to recognition of blocks. However, because some blocks are rarer than others and tend to occur only in the images of certain objects, these are given greater 'weight' than those that are common.

It is therefore possible to develop a strategy to make use of the varying significance of each block. The number and type of images used in the teach subset will affect the results. Before testing different strategies to choose the images for the subset, we establish a minimum subset size.

A target block's weight is calculated (Equation 8) from two conditional probabilities; the probability that the block is present, given that the object is in the image, and the probability that it is present, given that the object is not. This means that the subset requires a number of images both containing, and not containing, the object.

$$P(Block\ present \mid Object\ present) \times (1 - P(Block\ present \mid \overline{Object\ present}))^{20}$$

**Equation 8. Block weight calculation**

The exponent of the second term (20th power) was obtained by trial and error to give the best results. The full image library was used as the learning set, to give the most accurate probabilities. The calculation can return values between zero and one, and blocks used for matching must achieve a minimum weight of 0.1. A threshold value of 0.1 was set to allow some blocks to be worth ten times as much as others, which allows some more common blocks to be of small use to the system. It was observed that lower exponents would allow too many common blocks, which increased wrongly identified objects. Higher exponents did not allow enough blocks to be present for a definitive match, and objects were not identified. An exponent of 20 provided good results in both areas.

The method attempts to match blocks found in the target image with blocks found in the scene images. The learning stage uses these matched blocks to calculate the two probabilities with reference to the object's presence from the lookup table. More accurate probabilities (and therefore block weight) provide better results, so it is important to select the learning subset to reflect the rest of the image set. Object blocks that do not meet the minimum weight are discarded.

The reduced set of object blocks is compared with the scene image blocks, and a score generated for each object and each image. This score is the sum of the weights of any target object blocks found in a scene image, divided by the sum of all weights for that object. For this test, an object with a score above the threshold value of 0.06 was considered present. The clustering and spatial matching steps were left out because they are not as sensitive to changes in block weight as the block-matching step.

Several learning strategies were tried, including *supervised* where human judgement guided the evaluation of images and *automatic* where humans were not involved (Table 16). In some cases, the full library was used for teaching the method; in others a subset.

Each learning set gave two results for each object; a 'true-positive rate', which showed the proportion of objects identified in images that contained the object, and a 'false-positive rate', which showed the proportion of objects wrongly identified in images that did not contain the object.

The true- and false-positive rates were plotted for different learning set sizes (Figure 52, Figure 53). Also plotted were horizontal lines indicating the true-positive rates from the full library learnt test, to show the best case scenario in which the probabilities calculated reflected the whole image set. The two plots created show how the identification rates change when either the number of object containing images was increased, or the number of images without the object was increased. The learning subset images were chosen to either show the object with little occlusion or background clutter, or to provide a diverse range of images that do not contain the object.

A table was created to show how the different strategies performed (Table 16). The performance of each object is shown. The number of images in each set was determined by the previous plots (Supervised Learning to Establish a Subset Size, pg 131).

Also evaluated and shown in the table was the performance when multiple objects were learnt using the same subset of images. It will simplify and speed up the learning process if there is no performance loss associated with multiple object learning.

### 5.1.7. Alternate Image Sets

To ensure the system works on different objects in different environments, it was tested on two other image sets.

A second image set with five objects across 180 images was compiled for the purpose. The objects were set in a variety of poses, scales, and occlusion levels. The objects were exactly the same, and photographed from a very similar viewpoint.

The images were categorised by the object's presence and level of occlusion. The levels used were; object not present, background clutter, not occluded, less than 25% occluded, and more than 25% occluded.

A subset was chosen for the system to learn the objects with, according to the criteria specified in a previous test (Supervised Learning to Establish a Subset Size, pg 112 and 131). The system was also allowed to learn with the complete set of images.

Both tests were run and the results set out in a table with the proportion of objects found for each category.

The third set tested was a portion of the Caltech-101 image set. This image set was developed by the California Institute of Technology to evaluate the effectiveness of algorithms designed to differentiate between classes of objects. Although this is better suited for use with class detector systems than our specific object detector, the majority of leading systems have been benchmarked against it. Only a sample of the image set was used due to the long learning and processing time of the system. The images in each category are more often not of the same exact object, which this system is not designed to handle. This is why only a single test was performed using the same images for both the learning set and the test set.

The Caltech-101 images were categorised by whether the object is present or not, so the results are not separated by different levels of occlusion.

### 5.1.8. System Flaws for Alternate Image Sets

Due to the poor performance on the alternate image sets, a more in-depth analysis of the second image set was carried out. The results are used to identify the system flaws, and recommend potential improvements.

A program was developed to graphically analyse the blocks which did not match, and determine the reasons why.

To provide an informative sample, blocks from the objects were selected at random. Images containing the object but missing the block were examined; i.e. false negatives. Ten instances of blocks without matches were selected for each block type, for each object.

These block match errors were categorised into one of five reasons; block parameters that do not match, incorrect feature type, feature not found, feature not significant, and image occlusion. A block is recorded to have an unmatched parameter if the features making up the block are present and represent the appropriate points on the object in the scene image, but the block does not match. Incorrect feature types occur when there is a feature representing the appropriate part of the object in the scene image, but it is of the wrong type; i.e. an arc corner instead of a lobe corner. Features may be missing from the test image if they were not found at all, or they were removed as being not significant enough. An image occlusion is a valid reason for an object block to be missing from a scene image.

The program provided random blocks for a user to classify into one of the five reasons. For the block error and insignificant feature reasons a block was created from user chosen features for further examination. In the case of insignificant features the block was matched against the object block, to better estimate how many matches were being lost by discounting insignificant features. In the case of blocks that do not match, the parameter or parameters causing this were recorded.

The results are collated and show the most common type of errors. An additional breakdown shows the distribution of match-failure-causing parameters. This is to highlight any cases in which the parameter threshold is too low. Blocks formed with scene image features found to be insignificant are matched against the object block, and rates calculated to reflect the likelihood of these providing successful matches.

## 5.2. Results and Discussion

This section shows and evaluates the results from the tests outlined in the previous section. The meaning of the results are discussed and interpreted to evaluate the strengths and weaknesses of the system.

### 5.2.1. Bit Descriptor vs Blocks

The graphs for each method show the relative performance of the different feature comparison methods. Each of the three methods has two lines; showing the results of the cup and mug objects. As the weight threshold increases, fewer blocks/descriptors are used by the system, but those left are of higher value. The 'true positive rate' indicates how often an object is found, in images where it is present. The 'false-positive rate' indicates how often an object is found, in images where it is not present.

#### 5.2.1.1. Bit Descriptor Performance

Bit descriptors are user defined detectors for specific spatial relationships.

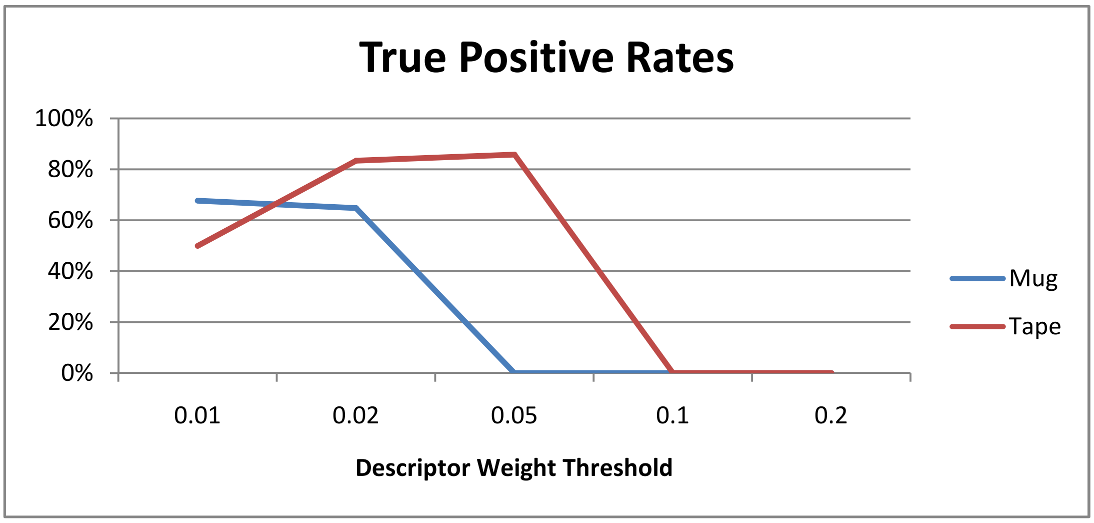


**Figure 40. True positive rates for bit descriptor method**

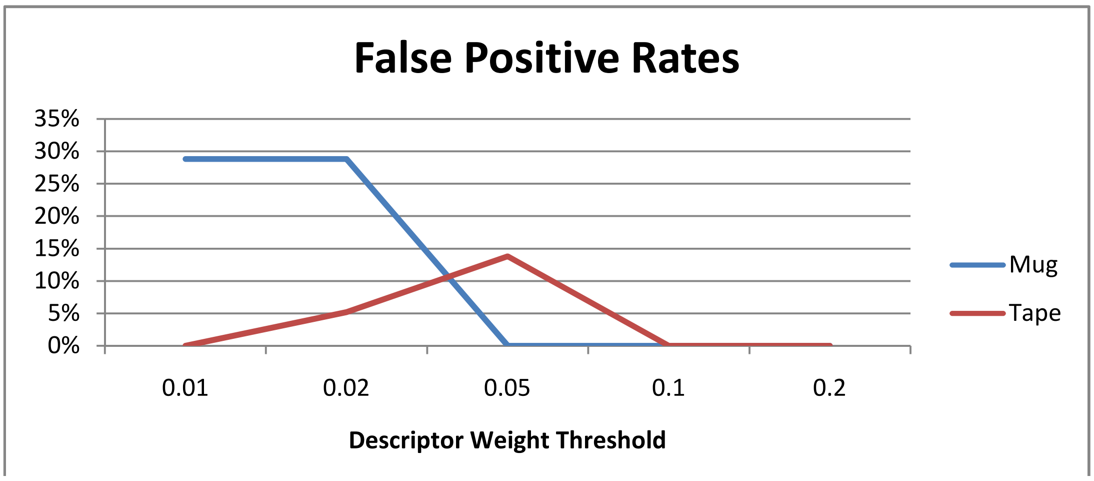


**Figure 41. False positive rates for bit-descriptor method**

The bit descriptors method shows a poor ability to detect objects at high descriptor weight thresholds. It also does not perform very well at lower thresholds. The two different objects have different false positive characteristics; the mug's start high and drop off while the tape's start low and peak then fall off again. In both cases the false positives show a similar trend to the true positive rates.

### 5.2.1.2. Blocks from All-Features Performance

Blocks are machine generated relationships used to compare any feature combination of two or three features. The all-features version compares blocks created from any features found within the image.

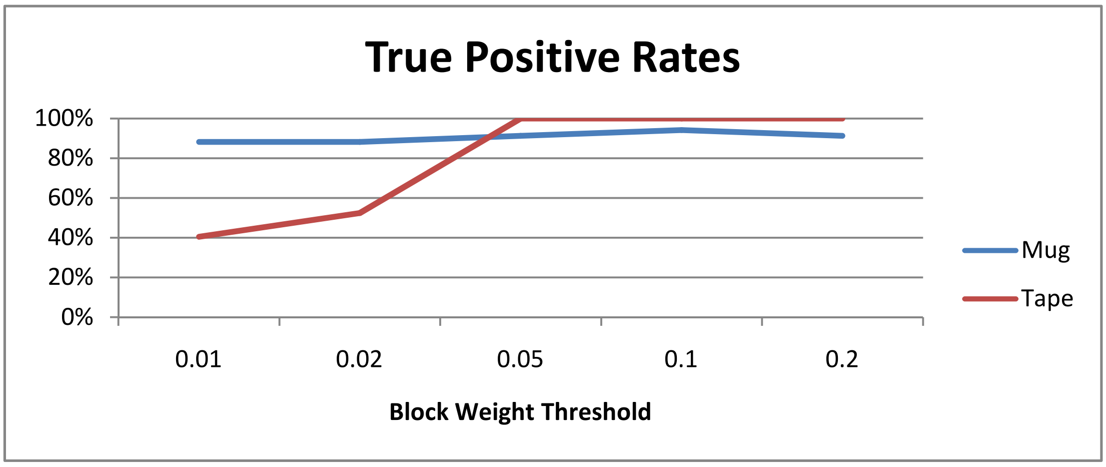


**Figure 42. True positive rates for all-features block method**

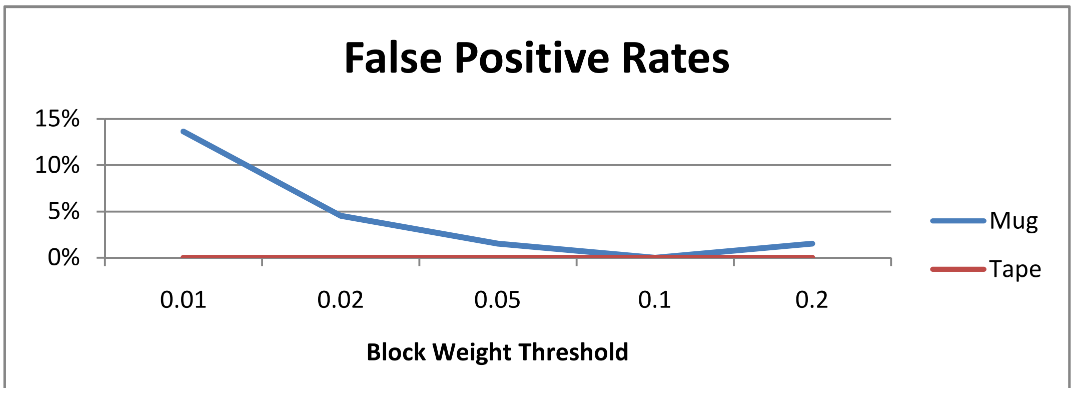


**Figure 43. False positive rates for all-features block method**

The blocks generated from all features method shows excellent performance for high block weight thresholds. The true positive rates start lower and increase as the block threshold is increased, then start to fall off slightly on the highest threshold. The false positives show a similar performance trend, with the best performance (lowest rate on both objects) seen at the 0.1 threshold, and worst performance seen at the 0.01 threshold.

#### 5.2.1.3. Blocks from a Single-Contour Performance

Blocks are machine generated relationships used to compare any feature combination of two or three features. The single contour version compares blocks created from features found on the same contour.

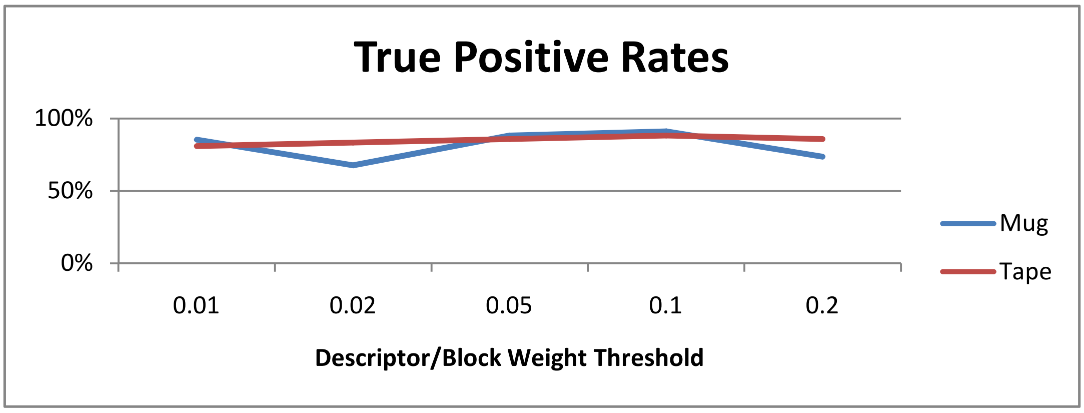


**Figure 44. True positive rates for single-contour blocks method**

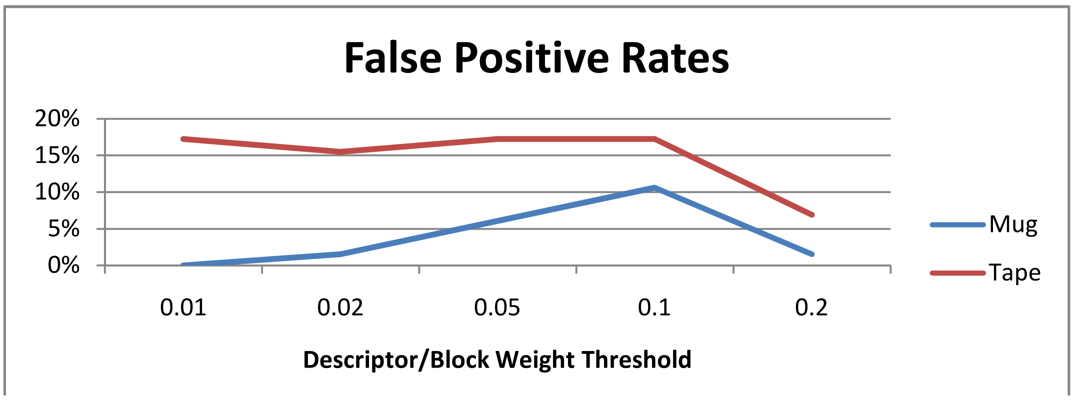


**Figure 45. False positive rates for single-contour blocks method**

The blocks generated from features on the same contour method shows good performance for high block weight thresholds. The true positive rates start slightly lower and increase slightly as the block threshold is increased, then start to fall off on the highest threshold. The false positives show poor performance, with the best performance (lowest rate on both objects) seen at the 0.2 threshold where the true positives are starting to decline. The false positives show the worst performance at the 0.1 threshold, when the true positives are performing best.

#### 5.2.1.4. Comparison Between Methods

The two graphs from each of the methods were used together to select the method with the highest true-positive rates and lowest false-positive rates, and to select the weight threshold to give the best results.

The pairs-and-triples, true-positive rates for both objects show the highest results at the higher threshold levels, with the single-contour, pair-and-triples rates trailing slightly behind. The bit-descriptor true-positive rates show a large decline as the block-weight threshold is decreased, showing this method does not offer such a high level of discrimination.

The pairs-and-triples results show the best performance in the false-positive tests as well, declining to zero at one of the block-weight thresholds before a slight rise for the mug object. The pair and triples from the same contour show much higher false positives across the range of block-weight thresholds. The bit descriptors show higher false-positive rates initially, declining to none at higher block weight thresholds.

The pairs-and-triples results are significantly better than the other methods across the range of thresholds and objects. These results show promise with over 90% true positives, and no false positives. The optimum threshold to maximise true-positives and minimise false positives for this method can be seen at 0.1.

**Limitations of results**

The tests were performed on one dataset, containing two objects in an office environment, so it does not provide a large spectrum of real world objects and scenes. However there is no indication that any one of the systems should perform better than the others in different environments.

### 5.2.2. Bit Descriptor Flaws

The histogram shows the distribution of bit-descriptor weights for each object. The algorithm assigns a zero weight to any bit descriptor found more often in images that do not contain the object than those that do. The majority of the 0 - 0.01 weight range had a weight of zero, so have a negative, or no, association with the object.

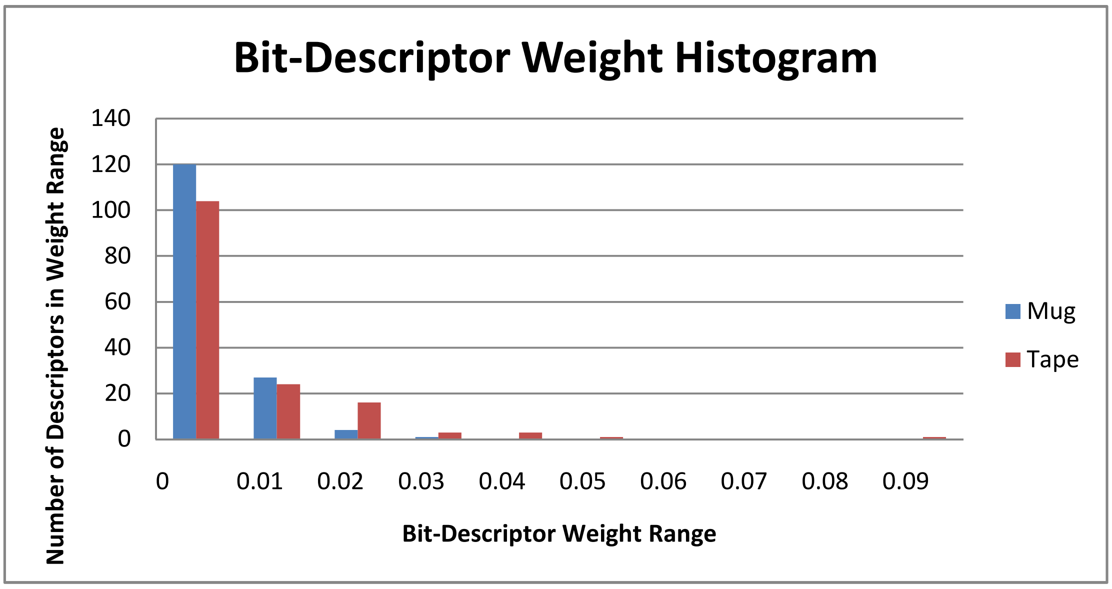


**Figure 46. Bit-Descriptor weight histogram**

The majority of descriptors fall within the 0 - 0.01 weight range. Only a few have a positive association with the objects and in most cases this is still very weak. This means that, individually, the bit-descriptors are a very poor way to discriminate between different objects. The poor results for the bit-descriptors collectively in the previous test are caused by the poor individual performance.

**Limitations of results**

A histogram of the bit-descriptor weights does not give insight into the performance of specific bit descriptors. However, it does show that overall there are very few descriptors with a high weight, suggesting the fatal flaw is not with a few specific descriptors, but with the method itself.

### 5.2.3. Pairs and Triples Type Evaluation

The graph shows how the true-positive, and false-positive, rates change with the match-score threshold. True-positive rates (TP) are the percentage of images that contained the object, and in which the object was found. False positives (FP) are the percentage of images that do not contain the object, but in which the object was found. Each image is given a 'block score' according to how many blocks match the database object.

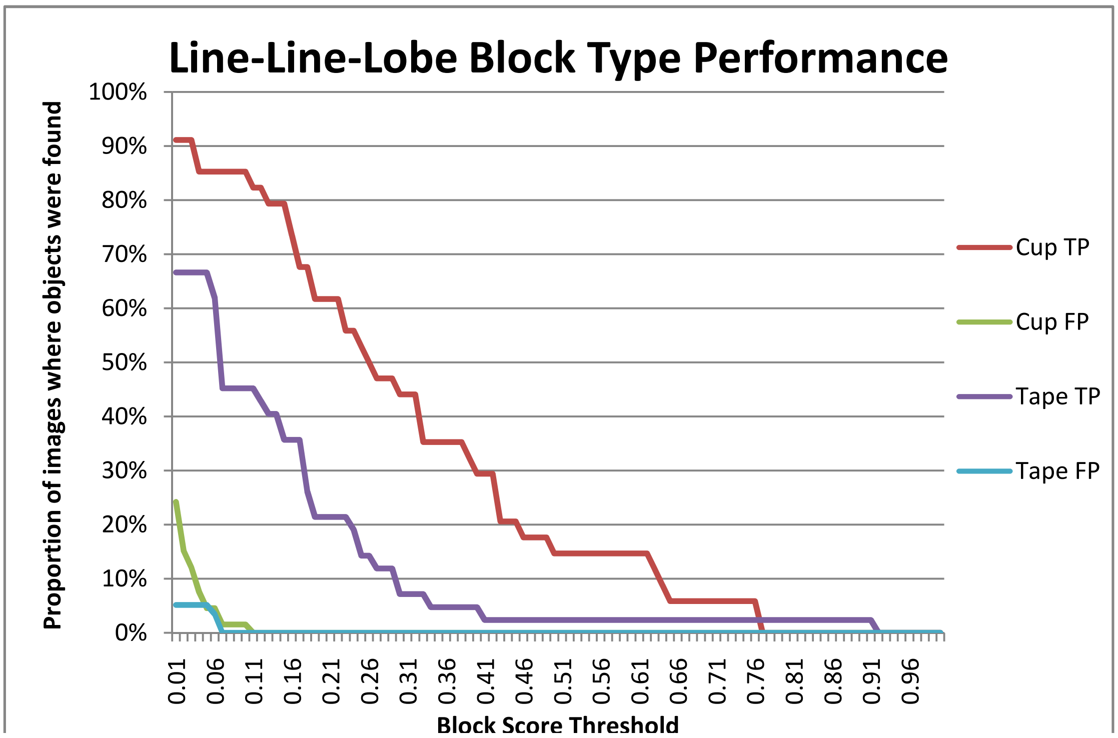


**Figure 47. Example discrimination graph**

True positive rates close to 100% and false-positive rates around 0% are desired. The area under each line is used to assess the performance across the range of block-score thresholds.

The graph shows both true- and false-positive rates declining for the objects. By the time the false positives have declined to 0, the true positives are around 45% for the Tape, and 90% for the Mug. The true-positive rates do not show any wide flat portions suitable for a threshold selection. The above graph is just for one type of block, the results from all the block types are shown later in this section (Figure 48).

The tables below summarises the graphs for the different block types. The desirable 'flat portions' can be approximated numerically by calculating the areas under the two true-positive lines (one for each test object). The false positive effects are quantified in the same way. The ratio of the two areas is used as a measure of how useful each block type is to the system.

Two tables were created for different block-rating thresholds; this refers to the minimum 'usefulness' of the individual blocks stored in the object database.

**Table 14. Block type effectiveness comparison (rating >0.1)**

| **Block Rating Threshold 0.1** | | | |
|---|---|---|---|
| *Type* | *TP Area* | *FP Area* | *Ratio* |
| ArcLobe | 1271 | 0 | MAX |
| LineLine | 1044 | 0 | MAX |
| LineArc | 695 | 0 | MAX |
| ArcArc | 832 | 0 | MAX |
| LineCircleLobe | 374 | 1 | 374 |
| LobeLobeCircle | 763 | 4 | 190.75 |
| LobeLobeLobe | 497 | 3 | 165.6667 |
| LineLobeLobe | 308 | 3 | 102.6667 |
| LobeCircleCircle | 517 | 6 | 86.16667 |
| CircleCircleCircle | 442 | 13 | 34 |
| LineCircleCircle | 352 | 15 | 23.46667 |
| LineLineLobe | 1534 | 66 | 23.24242 |
| LineLobe | 0 | 0 | None |
| LobeLobe | 0 | 0 | None |
| LineLineLine | 0 | 0 | None |
| LineLineCircle | 0 | 0 | None |

**Table 15. Block type effectiveness comparison (rating > 0.05)**

| **Block Rating Threshold 0.05** | | | |
|---|---|---|---|
| *Type* | *TP Area* | *FP Area* | *Ratio* |
| LineArc | 300 | 0 | MAX |
| LobeLobe | 200 | 0 | MAX |
| LineCircleCircle | 159 | 0 | MAX |
| LineCircleLobe | 185 | 0 | MAX |
| LineLobeLobe | 168 | 0 | MAX |
| LobeLobeCircle | 462 | 1 | 462 |
| ArcArc | 558 | 2 | 279 |
| LobeLobeLobe | 356 | 2 | 178 |
| LobeCircleCircle | 370 | 3 | 123.3333 |
| LineLine | 854 | 18 | 47.44444 |
| CircleCircleCircle | 247 | 7 | 35.28571 |
| ArcLobe | 925 | 57 | 16.22807 |
| LineLineLobe | 1571 | 177 | 8.875706 |
| LineLobe | 1800 | 400 | 4.5 |
| LineLineCircle | 1570 | 387 | 4.056848 |
| LineLineLine | 0 | 0 | None |

Although a lot of types do not have large true-positive area, a lack of false-positive area means they still contribute to higher scores when combined with the other types. The results from the tables show most block types contribute positively to the system, and only a few should be removed. Only those with no 'TP Area' were removed from the system.

The two graphs below show the performance of the system before and after removing some of the block types. Again, the graphs show how the true-positive, and false-positive, rates change with the match-score threshold.

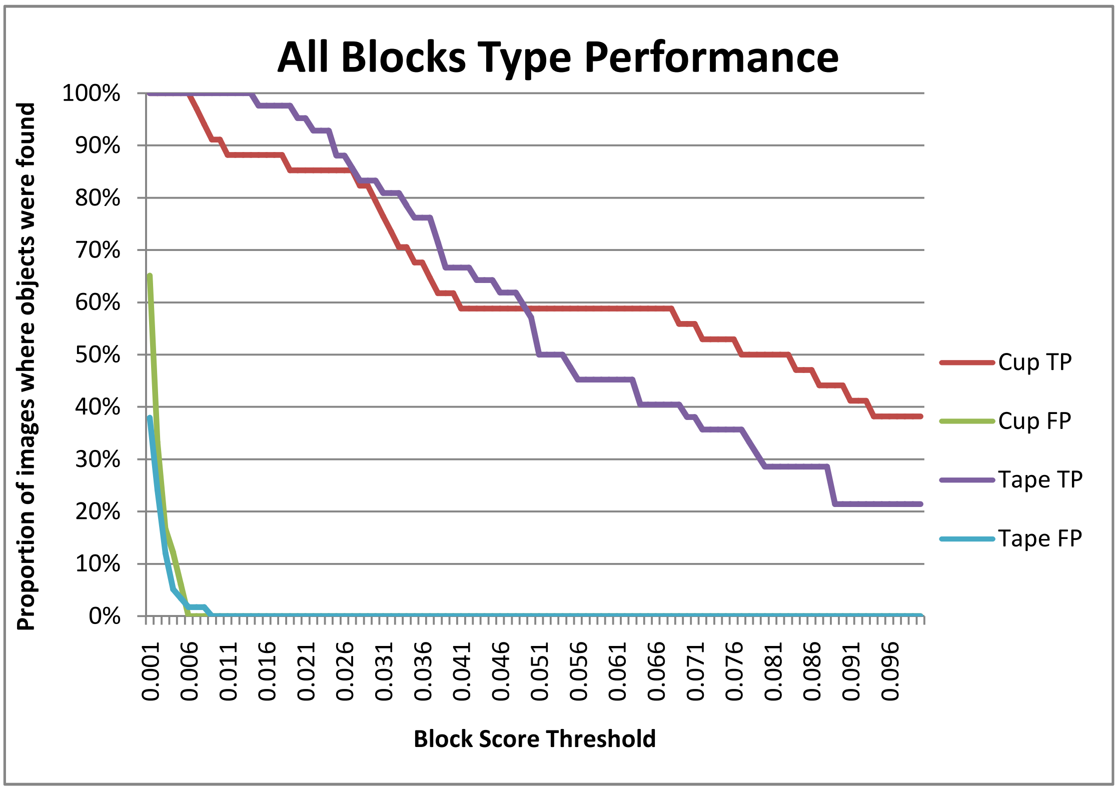


**Figure 48. Discrimination graph for all types of block**

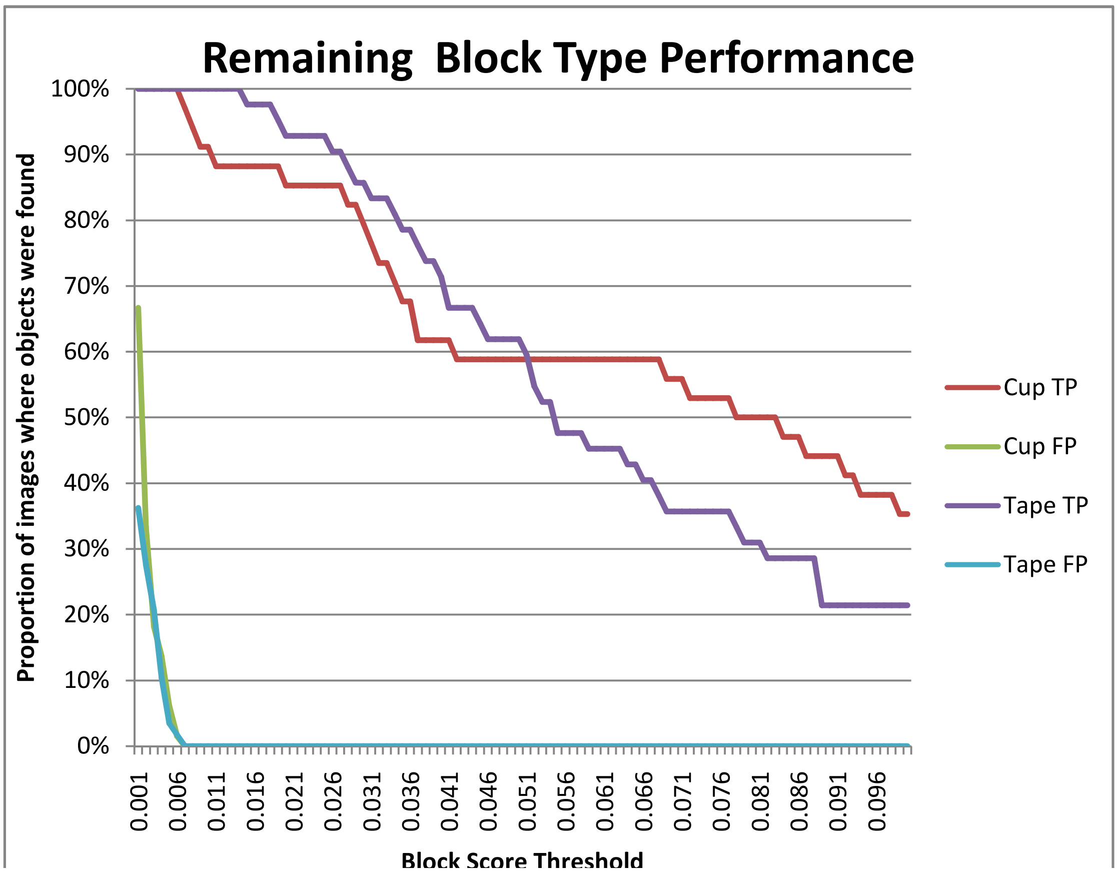


**Figure 49. Discrimination graph for all block types left in the system**

The graphs are almost identical; their similarities mean the types removed did not contribute either positively or negatively to the results. However, their removal will speed the process up. The results themselves are promising; with over 95% true positives once the false positives have fallen to zero. There is no wide flat portion on the true positive lines, suggesting this stage is sensitive to block-score threshold selection.

The previous graph was also used to determine the optimal block score threshold, where the false positives have fallen to zero. This occurs at a block-score of 0.006

**Limitations of results**

The block types were evaluated on sets with the same learning images as the test images. This means the learning ability of the different types is not tested across different subset groups of images, or whether learning from a subset gives similar results to learning from the complete set. It is not expected that different types of blocks will have significantly different learning abilities so these two factors are tested later on (Supervised Learning to Establish a Subset Size, pg 131).

## 5.2.4. Clustering

The graph shows how the true- and false-positive rates for the clustering part of the system vary according to the match score threshold. The cluster score is calculated according to how many matched pairs of blocks are grouped by the same rotation, scale, and translation information.

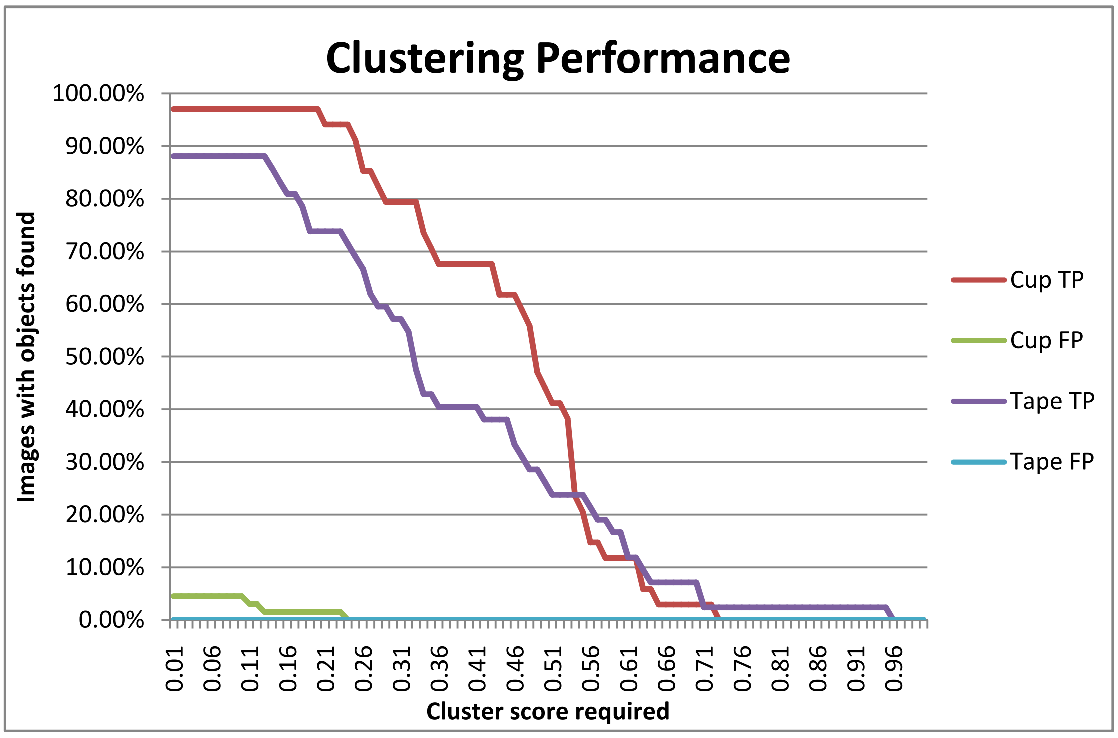


**Figure 50. Cluster performance curves**

The graph shows a flat portion on the true-positive lines for both objects at low cluster scores, where the false positive rates are in decline. By the time the false positives have declined to 0, the true positives are around 70% for the Tape, and 90% for the Mug. Compared with the block performance charts (Figure 49) there are fewer false positives, and a wider flat portion of true positives for both objects. However, due to the minimum cluster size, even at the lowest cluster-score thresholds neither of the objects is found 100% of the time. These results are good, but there is still room for improvement with another matching step.

The threshold for the cluster score was chosen from the previous graph, at 0.15 the false positives have almost dropped to zero, while the true positives are still up around 90%. A minimum cluster size of 15 was also selected by observation.

**Limitations of results**

The performance of the clustering system cannot be completely independently evaluated, as it relies on the previous block matching process to find a significant number of blocks, containing few false positives. All images are passed through to the clustering stage even if they do not meet the block-score threshold, which may introduce a few more false positives but ensures any true positives rejected are due to the criteria of this matching stage. The results of those tests (Figure 49) suggest it does this adequately, however better results there would likely follow through to this part as well.

### 5.2.5. Spatial Matching

As for previous charts, the graph shows how the true-positive and false-positive rates for the spatial matching part of the system vary according to the match score threshold. The spatial match scores are calculated according to how many blocks are used in the final derived block, thus matching in relative position, scale, and direction between the viewed image and the object database.

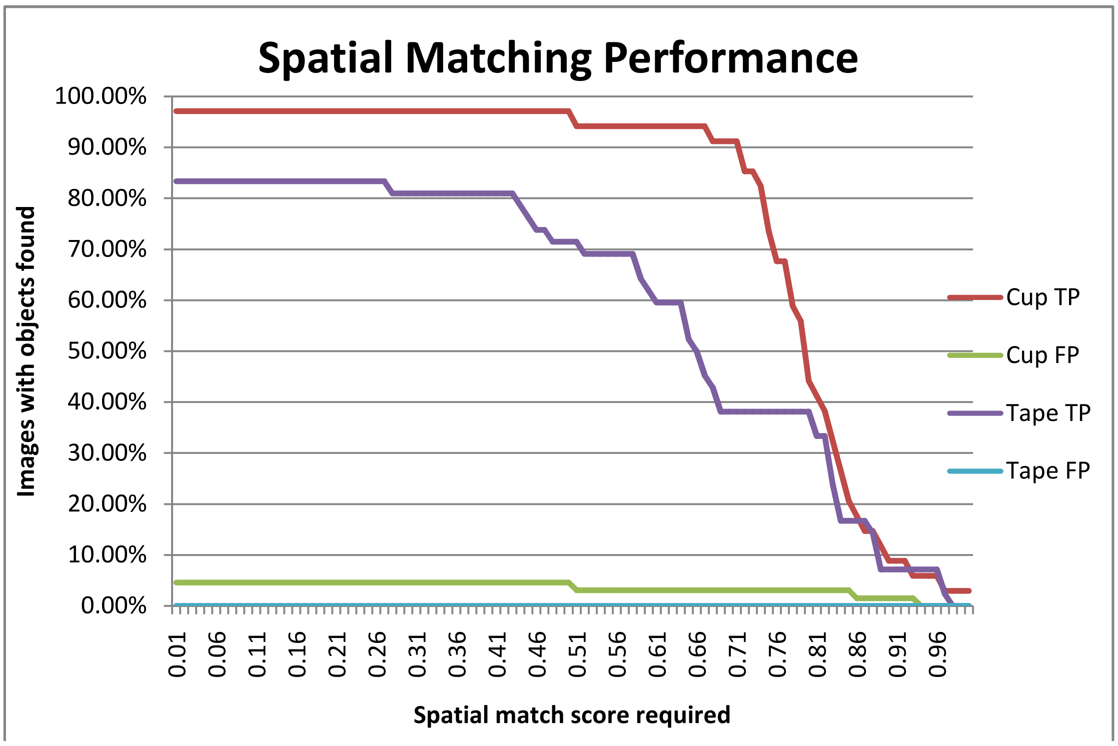


**Figure 51. Spatial Match performance curves**

The true-positive rates show much wider flat portions across a large range of block thresholds, followed by steeper declines than any of the previous graphs. This indicates the spatial-matching step is not as sensitive to threshold selection, provided a value corresponding to the flat portion of the true-positive curves is selected. There are a few false positives persistent across most of the range; however it is expected these will be removed when thresholds are applied to the previous clustering and block matching stages. Greater than 80% for the Tape object, and over 95% for the Cup are very good results.

The spatial-match threshold score was selected as 0.25, which is where the true positives start to decline. Also chosen was a minimum spatial-match group size of 10, found by observation.

## Limitations of results

The performance of the spatial matching system relies on the previous block matching process and clustering algorithm to provide a suitable cluster to begin with. All images are passed through to the spatial matching stage even if they do not meet the block-score or cluster-score thresholds, which may introduce a few more false positives but ensures any true positives rejected are due to the criteria of this matching stage. Again, the results of those tests (Figure 49, Figure 50) suggest they do this adequately, however better results in those tests would be likely to contribute to better results in this test as well.

### 5.2.6. Supervised Learning to Establish a Subset Size

The two graphs below illustrate how the number of objects in the learning set affects the proportions of true and false positives found. True-positive rates (TP) are the percentage of images that contained the object, and in which the object was found. False positive rates (FP) are the percentage of images that do not contain the object, but in which the object was found.

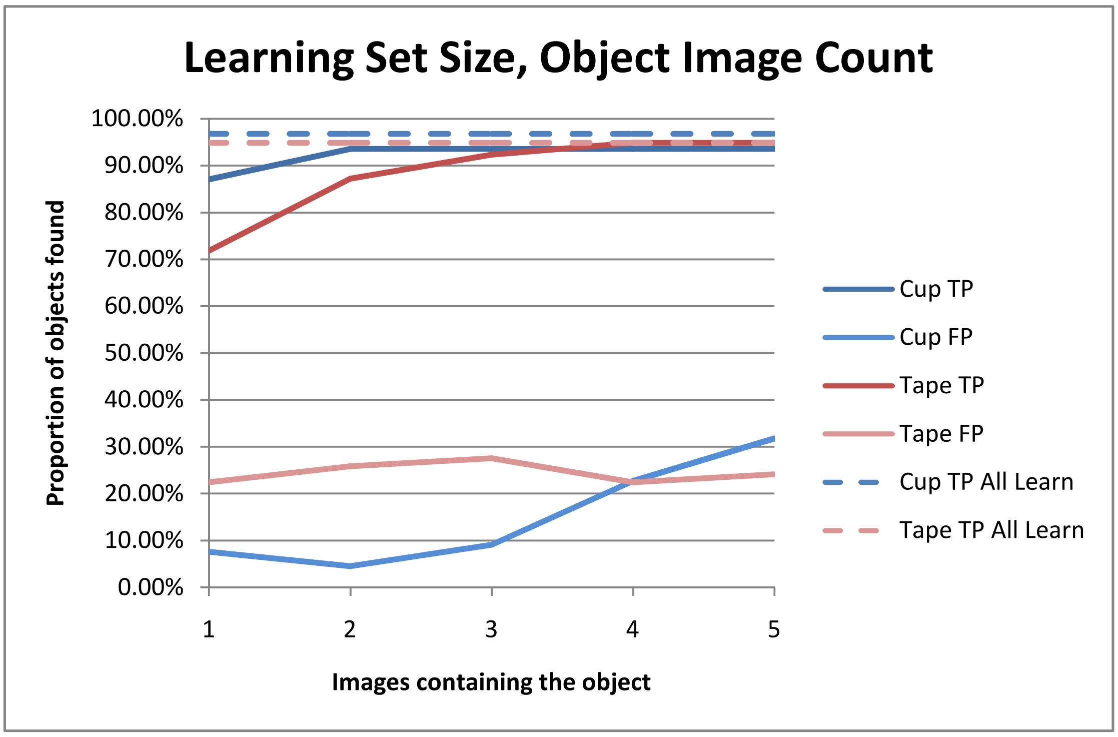


**Figure 52. Learning set size, object image count**

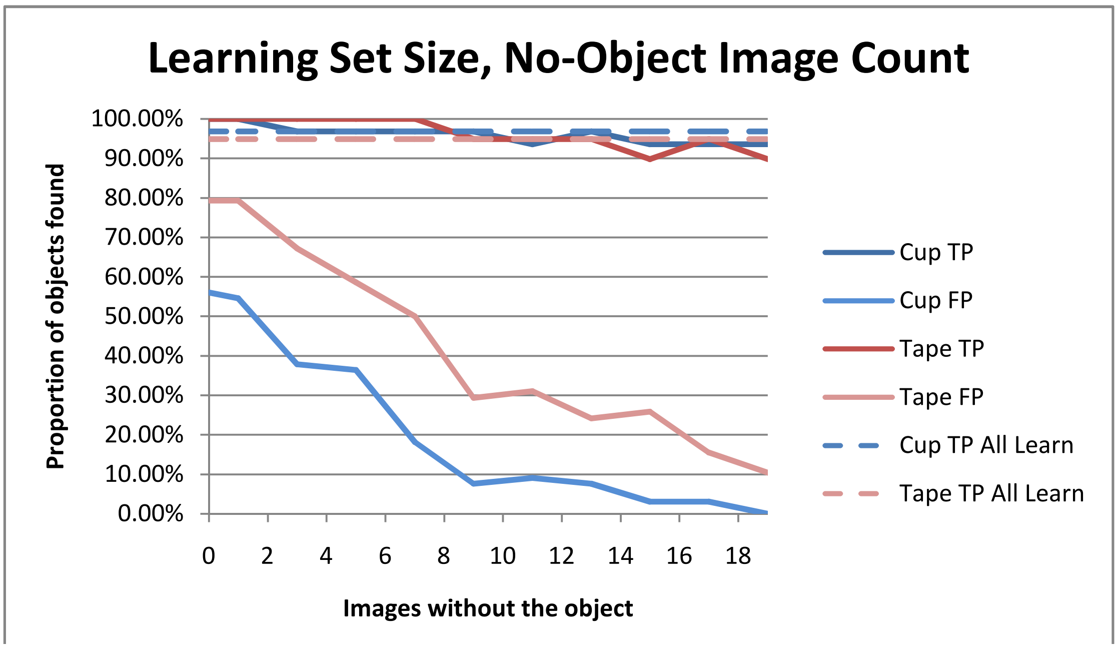


**Figure 53. Learning set size, no-object image count**

As more images are added to the learning set, the true- and false-positive rates increase and decrease respectively. Although it appears the number of false positives increase with the true positives as well; this is due to the order in which the images were selected. The best looking images were added to the group first, and the 5th was not as good a representation as the first. When the 5th and 2nd images were swapped, this trend disappeared.

The true-positive rates approached (to within 5%) the rates found when the whole set was used to learn with after just four object containing images. The false-positive rates declined sharply until nine images were added then slowly declined after that. Four images containing the object and nine images without it were chosen as the minimum size for the learning subsets. This test shows the system has a good learning ability, with both the true and false positives converging on the desired values after only a small number of images are added.

The two tables following show how the results vary for different methods of choosing the image sets. Each table has rows for the different objects, and columns for the differing levels of occlusion. The rates at which the objects are found show the relative behaviours of the different learning strategies.

**Table 16. Learning set selection methods**

| | Not Present | Background Clutter | Not Occluded | <25% Occlusion | <50% Occlusion | >50% Occlusion | >18deg rotation | Total There (not inc >18deg) |
|---|---|---|---|---|---|---|---|---|
| **User Chosen** | | | | | | | | |
| Cup % | 0% | 100% | 100% | 100% | 80% | 50% | 0% | 93% |
| Tape % | 0% | 40% | 68% | 67% | 75% | n/a | 33% | 66% |
| **Machine Chosen Worst Result** | | | | | | | | |
| Cup % | 0% | 100% | 100% | 100% | 75% | 100% | 33% | 96% |
| Tape % | 0% | 75% | 62% | 0% | 0% | n/a | 0% | 46% |
| **Machine Chosen Worst Unoccluded Result** | | | | | | | | |
| Cup % | 0% | n/a | 100% | 92% | 60% | 50% | 33% | 85% |
| Tape % | 0% | 75% | 60% | 0% | 0% | n/a | 33% | 43% |

*Table 17. Combined learning results*

| | Not Present | Background Clutter | Not Occluded | <25% Occlusion | <50% Occlusion | >50% Occlusion | >18deg rotation | Total There (not inc >18deg) |
|---|---|---|---|---|---|---|---|---|
| **Separate Tests** | | | | | | | | |
| Cup % | 0% | 100% | 100% | 100% | 80% | 50% | 0% | 93% |
| Tape % | 0% | 40% | 68% | 67% | 75% | 0% | 33% | 66% |
| **Combined Test** | | | | | | | | |
| Cup % | 0% | 100% | 100% | 100% | 80% | 50% | 0% | 93% |
| Tape % | 4% | 60% | 63% | 67% | 88% | 0% | 33% | 69% |

Although there is a breakdown by occlusion level, the 'total there' column provides the best indication of accuracy. The user chosen set shows the best performance for learning set selection. Combining the two sets into one actually slightly improved the results over having separate learning sets for each object. It is convenient that the user chosen set showed the best results, as it will be the quickest and easiest method to use. Also useful is the combined learning attribute, as this means the system can use other database object images to learn new objects put into the database.

**Limitations of results**

The choice of images for the learning sets will affect the results. The images chosen for the learning size tests started with those that appeared to have the best characteristics, and got slightly worse due to elimination. This is especially evident in Figure 52 where the false-positive rate for the cup appears to increase when more object images are added.

### 5.2.7. Alternate Image Sets

The following three tables show the proportion of times the object is correctly or incorrectly identified; categorised by different occlusion levels for the second set we developed, or by whether an object of the class is present for the Caltech-101 sample.

**Table 18. Second Set Full Learning Set Performance**

| Object Type | Not Present | Background Clutter | Not Occluded | <25% Occlusion | >25% Occlusion | Total There |
|---|---|---|---|---|---|---|
| Micrometer | 0% | 0% | 56% | 50% | 25% | 33% |
| Mini Vise | 0% | 0% | 56% | 83% | 0% | 39% |
| Motorcycle Sculpture | 0% | 25% | 50% | 67% | 50% | 44% |
| Vise Grips | 1% | 17% | 63% | 0% | 0% | 33% |
| X360 Controller | 0% | 40% | 89% | 100% | 100% | 78% |

The results for the second image set using the whole set to learn from show between 33 and 78 percent true positive rates, with only one object showing a false positive. The true-positive rates are lower than is desirable, and a lot lower than the previous tests, suggesting some part of the algorithm does not deal with the newer objects or images very well.

**Table 19. Second Set Subset Learning Performance**

| Object Type | Not Present | Background Clutter | Not Occluded | <25% Occlusion | >25% Occlusion | Total There |
|---|---|---|---|---|---|---|
| Micrometer | 0% | 0% | 50% | 75% | 0% | 31% |
| Mini Vise | 0% | 8% | 44% | 83% | 0% | 36% |
| Motorcycle Sculpture | 0% | 33% | 75% | 50% | 50% | 56% |
| Vise Grips | 0% | 0% | 50% | 0% | 0% | 22% |
| X360 Controller | 0% | 20% | 78% | 100% | 50% | 64% |

The results for the second image set using only a subset to learn from show between 22 and 64 percent true-positive rates, with no false positives. The true-positive rates are lower than is desirable, and unlikely to be of practical use. Again, suggesting some part of the algorithm does not deal with the newer objects or images very well.

**Table 20. Caltech-101 Sample Object Class Performance**

| Object Class Examined | False Positive Rate | True Positive Rate |
|---|---|---|
| Accordion | 0% | 0% |
| Anchor | 0% | 5% |
| Bass (fish) | 0% | 2% |
| Binocular | 0% | 3% |
| Brain | 0% | 2% |
| Buddha | 0% | 1% |
| Camera | 0% | 4% |

Five percent or lower recognition rates are completely unusable. Even though the Caltech set uses different object of the same type (rather than the same object and very similar viewpoint this system is designed for), better results were still expected.

## Limitations of results

The Caltech set is much more suited towards class detection systems than our specific object recognition approach, which is reflected in the results.

### 5.2.8. System Flaws for Alternate Image Sets

The table shows the reasons individual blocks did not match where they should have. A random selection of blocks was manually analysed to generate the results. Ideally, the only type of error in the system would be from image occlusions. ‘Block does not match’ means the block parameters are outside the acceptable error ranges. ‘Incorrect feature type’ means there is a feature present in the right place but it is of the wrong type; i.e. instead of an arc, it might be a lobe. ‘Feature not significant’ means the feature was discarded due to not lying on enough contours.

**Table 21. Categorised Error Results**

| | Block Does Not Match | Image Occlusion | Incorrect Feature Type | Feature Not Found | Feature Not Significant |
|---|---|---|---|---|---|
| Error Totals | 64 | 67 | 34 | 257 | 58 |
| Error Percentages | 13% | 14% | 7% | 54% | 12% |

The most common error was ‘feature not found’; with the others being much less frequent. Features not being found shows that there is a problem with the feature extraction algorithm, one of the initial stages of the system.

To illustrate the potential improvement if the insignificant features were retained (at great computational cost), the table below shows match rates for user specified blocks containing features previously deemed not significant enough.

**Table 22. Match Rates of Assembled Blocks using Insignificant Features**

| | Match Rate |
|---|---|
| Arc Lobe | 50% |
| Line Line | 0% |
| Line Line Lobe | 0% |
| Line Circle Circle | 0% |
| Line Circle Lobe | 50% |
| Line Lobe Lobe | 0% |
| Lobe Circle Circle | 0% |
| Lobe Lobe Circle | 20% |
| Lobe Lobe Lobe | 19% |
| Circle Circle Circle | 33% |
| Match Rate Over All Types | 17% |

In most cases these features do not match anyway; so their inclusion is unlikely to be of noticeable benefit to the object matching performance.

The following table shows how well-distributed the parameter errors are. If a parameter has a disproportionate number of errors, its error threshold for block matching might be too sensitive.

**Table 23. Parameter Error Proportions**

| Block Type | Parameters | Errors | Maximum Errors for a Single Parameter | Percentage of Total Errors for a Single Parameter |
|---|---|---|---|---|
| Arc Lobe | 5 | 8 | 3 | 38% |
| Line Line | 4 | 15 | 9 | 60% |
| Line Arc | 7 | 30 | 6 | 20% |
| Arc Arc | 8 | 4 | 2 | 50% |
| Line Line Lobe | 11 | 33 | 6 | 18% |
| Line Circle Circle | 7 | 26 | 6 | 23% |
| Line Circle Lobe | 12 | 16 | 4 | 25% |
| Line Lobe Lobe | 14 | 62 | 12 | 19% |
| Lobe Circle Circle | 11 | 18 | 5 | 28% |
| Lobe Lobe Circle | 15 | 21 | 6 | 29% |
| Lobe Lobe Lobe | 18 | 73 | 9 | 12% |
| Circle Circle Circle | 6 | 16 | 4 | 25% |

It can be seen from the table that higher percentages for a single parameter generally correspond to the block having fewer parameters, which is to be expected. Sensitive parameter block-match thresholds are unlikely to be causing the poor results.

### Limitations of results

The categorisation of errors and construction of additional blocks was done by a human's best match, and some human errors may have been introduced.

# 6. Conclusions

## *6.1. Bit Descriptor vs Blocks*

The blocks method of comparing the object features performed significantly better than the bit descriptors method.

The bit descriptor's true- and false-positive rates were worse than the blocks system for nearly every test. The true positives from the bit descriptors declined rapidly as the weight threshold was increased, while the block systems showed an increase.

The block results were from two slightly different strategies, one which formed blocks from features across the image, while the other used only features from the same contour. The set using features from the whole image performed consistently better, so this method was used.

An optimal block weight threshold can also be deduced from these results. The optimal weight for both true and false positives on the chosen block system is at 0.1. Both the highest true-positive rates and lowest false-positive rates occurred at this threshold.

## 6.2. Bit Descriptor Flaws

The results were presented by a histogram of the bit weights. Out of 152 descriptors, many were found more often in images without the object, and were of no use for that object. The majority of those left had only a low block weight, thus had little correlation with the object's presence. The main flaw of the bit descriptor system is that the descriptors are not specific enough to each object. Perhaps with a much larger number of descriptors their combined effect would be promising; however writing the detectors would be very time-consuming.

## 6.3. Pair and Triples Type Evaluation

The performance of the different block types is evaluated using the ratio of true positives to false positives across a range of block score thresholds. The different block types are characterised by the type and number of features contained within. The test was performed twice at different block weight thresholds as well, to ensure the results were consistent. Four block types were found that offered very little or no benefit to the system for both block weight thresholds. The types were; Line-Lobe, Lobe-Lobe, Line-Line-Line, and Line-Line-Circle. These were removed to save calculation time.

The same test was used to record the performance with all blocks present, and after removing the four block types. These graphs were very nearly identical; and it was concluded the removed block types did not affect the match rates of the system.

The graph after the four block types were removed was used to determine the optimal block score threshold. This occurs when there are a high rate of true positives, and only a few false positives; which can be seen at 0.006. This value was used as the weighted proportion of object blocks required to be in the viewed image for the system to move onto the next match stage. The gradual slopes of the true-positive rates suggest there is a large range of block scores where it is difficult to say with any certainty whether the object is present from the block score alone.

## 6.4. Clustering

The clustering performance was shown with true- and false-positive rates for each object, against the cluster score threshold; similarly to the previous tests. The clustering performance shows true-positive rates with steeper slopes than the block-score test. However the true-positive rates still degrade significantly as the cluster-score threshold is increased to remove the false positives, requiring another match step to get good results. A value of 0.15 was chosen as the cluster match threshold, along with a minimum cluster size of 15. These values ensure the majority of images containing objects are passed to the spatial matching system, and include few images without the object.

## 6.5. Spatial Matching

The spatial matching performance was shown with true- and false-positive rates for each object, against the spatial match score threshold; similar to the previous tests. The spatial matching results show the widest flat portion and steepest true positive rate curve yet. This suggests more match certainty, and less sensitivity to threshold selection provided the threshold is selected from the flat portion. The threshold score selected was 0.25, with a minimum number of 10 block transforms required for an object to be considered present.

## *6.6. Learning Ability*

The two graphs (Figure 52, Figure 53) show how the system results are affected by the number of images either containing or not containing the object.

More images containing the object increase the rate of true positives. Although it appears the number of false positives increases as well, this is due to the order in which the images were selected. The best looking images were added to the group first, and the 5th was not as good a representation as the first. When the 5th and 2nd images were swapped, the false positives rose sharply then declined.

As the number of images which do not contain the object is increased, the number of false positives declines, and the true positives also show a slight decline. This decline is due to some of the object blocks being found in the added images, thus reducing their weight.

From the graphs the minimum learning set size was determined to be 4 images containing the object, and 9 images without it.

Also tested were different strategies to select the images, consisting of two machine chosen methods, and one user chosen. The machine chosen methods selected the image with the most error, and added it to the learning set. One of the machine chosen methods only used object images that were un-occluded. The user chosen images were of diverse objects and background when the object was not present, and uncluttered, un-occluded images of the object.

The user chosen method shows slightly better results, and is far quicker than the machine chosen methods, and will be used hereafter.

When the learning sets for multiple objects were combined the results show little difference. In these cases there were four images from each object, and nine images containing neither. This ability speeds up the learning process for learning sets with multiple objects, but more importantly allows an object to be quickly added to an existing database using the other database objects as images not containing the added object

## *6.7. Alternate Image Sets*

The results for the two other image sets run through the system are poor, and there are obviously some flaws present. Although the Caltech-101 set is better suited for class detectors, the few percent true-positive rates are still unacceptably low. The second set we compiled is far more suited to this system, but still shows poor results, with the majority of objects found in less than half the object containing images.

## *6.8. System Flaws for Alternate Image Sets*

The system flaws test allocates the results into one of five main categories. Over half fell into the feature not found category, which is far worse than the previous Feature Extraction Test on page 57. This suggests a problem with the feature extraction algorithm, unfortunately this is also the first process, and problems there may contribute to errors attributed to other categories. The feature extraction problem is likely to be caused by the system not adapting to the colour/size/content of the images in the alternate sets.

The table shows the potential benefits of allowing the 'insignificant' feature to remain in the system. Few of the blocks created from manually identified insignificant features match, suggesting simply increasing the number of features used to create blocks will do little more than slow the system down.

The final results were designed to highlight any parameter with error thresholds that were too low, however the distributions were not uneven, so error thresholds are unlikely to be a significant problem.

## 6.9. Overall Conclusion

As the system stands, it is not a satisfactory solution to the problem. However, specific object recognition is a very difficult problem that has yet to be solved. The system contains many parts which utilise high level data types to perform the comparison, some of which may be eventually used to solve the problem.

Three similar approaches were attempted in this work, bit descriptors, and two sorts of pair and triple blocks. All three approaches use geometric features extracted from isoluminal contours. Through the methodology and results, this work provides insight into the strengths and weaknesses of these methods.

The results across the set it was developed with are very good, showing it has potential to solve the problem. If the system can be developed further to work as well on other sets as it does on the development set, it would be an attractive solution to the problem of specific object recognition.

# 7. Recommendations

The feature extraction part of this algorithm is not up to a standard to make it of any practical use. It is recommended that this is improved if possible, or replaced with a more reliable method.

The block system could even be adapted to work with completely different feature types, as long as they still have a direction, position, and size within the image. While the block system's performance cannot be reliably established due to the substandard feature extraction in some image sets; it did work well when there was good feature data.

# 8. Appendix A – Bit Descriptor List

| | |
|---|---|
| 1-5 Parallel Line Pairs | 1-3 RS Centers coincident with 2 others |
| 6-10 Parallel Line Pairs | >3 RS Centers coincident with 2 others |
| >10 Parallel Line Pairs | 1-3 RS Centers coincident with >2 others |
| 1-3 Sets of 3 Line Parallel Groups | >3 RS Centers coincident with >2 others |
| >3 Sets of 3 Line Parallel Groups | 1-3 Symmetrical Lobes |
| 1-3 Sets of 4 Line Parallel Groups | 4-8 Symmetrical Lobes |
| >3 Sets of 4 Line Parallel Groups | >8 Symmetrical Lobes |
| 1-3 Sets of >4 Line Parallel Groups | 1-2 Lobe Cornered Squares |
| >3 Sets of >4 Line Parallel Groups | >2 Lobe Cornered Squares |
| 1-3 Sets of 3 Line Parallel Series | 1-2 Lobe Cornered Triangles |
| >3 Sets of 3 Line Parallel Series | >2 Lobe Cornered Triangles |
| 1-3 Sets of 4 Line Parallel Series | 1-2 Lobe Cornered Parallelograms |
| >3 Sets of 4 Line Parallel Series | >2 Lobe Cornered Parallelograms |
| 1-3 Sets of >4 Line Parallel Series | 1-3 Right Angle Lobe Symmetry Pairs |
| >3 Sets of>4 Line Parallel Series | >3 Right Angle Lobe Symmetry Pairs |
| 1-2 Circles | 1-3 Right Angle Line-Lobe-Line Tangents Groups |
| 3-6 Circles | >3 Right Angle Line-Lobe-Line Tangents Groups |
| >6 Circles | 1-3 Line-Lobe-Line Tangents |
| 1-2 Squares | 4-8 Line-Lobe-Line Tangents |
| 3-6 Squares | >8 Line-Lobe-Line Tangents |
| >6 Squares | 1-3 Parallel Lobe Symmetry Pairs |
| 1-2 Triangles | >3 Parallel Lobe Symmetry Pairs |
| 3-6 Triangles | 1-3 Parallel Arc Symmetry Pairs |
| >6 Triangles | >3 Parallel Arc Symmetry Pairs |
| 1-2 Parallelograms | 1-2 Equal Sided Parallelograms |
| 3-6 Parallelograms | >2 Equal Sided Parallelograms |
| >6 Parallelograms | 1-2 Equal Sided Squares |
| 1-3 Symmetrical Lines | >2 Equal Sided Squares |
| 4-8 Symmetrical Lines | 1-2 Equal Sided Triangles |
| >8 Symmetrical Lines | >2 Equal Sided Triangles |
| 1-3 Symmetrical Arcs | 1-2 Circle Segments (Ds) |
| 4-8 Symmetrical Arcs | >2 Circle Segments (Ds) |
| >8 Symmetrical Arcs | 1-10 LobeLobe Tangent Pairs |
| 1-3 Axis of Symmetry with 2 Pairs of features | >10 LobeLobe Tangent Pairs |
| >3 Axis of Symmetry with 2 Pairs of features | 1-3 LobeLobes with equal start/end tangents |
| 1-3 Axis of Symmetry with 3 Pairs of features | >3 LobeLobes with equal start/end tangents |
| >3 Axis of Symmetry with 3 Pairs of features | 1-3 Lobelobes with 90 degree, and equal SE tangents |
| 1-3 Axis of Symmetry with >3 Pairs of features | >3 Lobelobes with 90 degree, and equal SE tangents |
| >3 Axis of Symmetry with >3 Pairs of features | 1-2 Arc Line Tangents Mid Arc |
| 1-3 Sets of 2 Concentric Arcs | >2 Arc Line Tangents Mid Arc |
| >3 Sets of 2 Concentric Arcs | 1-3 Line-Arc-Line Tangents |
| 1-3 Sets of 3 Concentric Arcs | >3 Line-Arc-Line Tangents |
| >3 Sets of 3 Concentric Arcs | 1-3 Lobe Arc Tangents |
| 1-3 Sets of >3 Concentric Arcs | >3 Lobe Arc Tangents |
| >3 Sets of >3 Concentric Arcs | 1-3 Arc Arc Tangents |
| 1-2 Arc Cornered Squares | >3 Arc Arc Tangents |
| >2 Arc Cornered Squares | 1-3 Arc-Line-Arcs |
| 1-2 Arc Cornered Triangles | >3 Arc-Line-Arcs |

| | |
|---|---|
| >2 Arc Cornered Triangles | 1-3 180 degree Arc-Line-Arcs |
| 1-2 Arc Cornered Parallelograms | >3 180 degree Arc-Line-Arcs |
| >2 Arc Cornered Parallelograms | 1-3 Lobe-Line-Lobes |
| 1-3 Right Angle Arc Symmetry Pairs | >3 Lobe-Line-Lobes |
| >3 Right Angle Arc Symmetry Pairs | 1-3 180 degree Lobe-Line-Lobes |
| 1-3 Right Angle Line Symmetry Pairs | >3 180 degree Lobe-Line-Lobes |
| >3 Right Angle Line Symmetry Pairs | 1-3 Arc-Lobe-Arcs |
| 1-3 Parallel Line Symmetry Pairs | >3 Arc-Lobe-Arcs |
| >3 Parallel Line Symmetry Pairs | 1-3 Lobe-Arc-Lobes |
| 1-3 Right Angle Line-Arc-Line Tangents Groups | >3 Lobe-Arc-Lobes |
| >3 Right Angle Line-Arc-Line Tangents Groups | Line Line Included angle 10-20 degrees |
| 1-3 Line-Arc Tangents | Line Line Included angle 20-30 degrees |
| 4-8 Line-Arc Tangents | Line Line Included angle 30-40 degrees |
| >8 Line-Arc Tangents | Line Line Included angle 40-50 degrees |
| 1-3 Rotationally Symmetric Line Pairs | Line Line Included angle 50-60 degrees |
| >3 Rotationally Symmetric Line Pairs | Line Line Included angle 60-70 degrees |
| 1-3 Rotationally Symmetric Arc Pairs | Line Line Included angle 70-80 degrees |
| >3 Rotationally Symmetric Arc Pairs | Line Line Included angle 80-90 degrees |
| 1-3 Rotationally Symmetric Lobe Pairs | Line Line Included angle 90-100 degrees |
| >3 Rotationally Symmetric Lobe Pairs | Line Line Included angle 100-110 degrees |
| 1-3 Rotational Symmetry Groups of 2 Pairs | Line Line Included angle 110-120 degrees |
| >3 Rotational Symmetry Groups of 2 Pairs | Line Line Included angle 120-130 degrees |
| 1-3 Rotational Symmetry Groups of 3 Pairs | Line Line Included angle 130-140 degrees |
| >3 Rotational Symmetry Groups of 3 Pairs | Line Line Included angle 140-150 degrees |
| 1-3 Rotational Symmetry Groups of >3 Pairs | Line Line Included angle 150-160 degrees |
| >3 Rotational Symmetry Groups of >3 Pairs | Line Line Included angle 160-170 degrees |
| 1-3 RS Centers coincident with 1 other | 1-3 Arc centers on angle bisectors |
| >3 RS Centers coincident with 1 other | >3 Arc centers on angle bisectors |

# 9. References

**Adam, L. B., Vincent, J. D. P., & Stephen, A. D. P. (1996).** "A maximum entropy approach to natural language processing." *Comput. Linguist., 22*(1), 39-71.

**Aguado, A. S., & Nixon, M. S. (1996).** "A New Hough Transform Mapping for Ellipse Detection." *Image, Speech and Intelligent Systems Research Journal.*

**Alagona, G., Ghio, C., & Villani, V. (1999).** "Basis set, level, and continuum solvation effects on the stability of a synthetic dipeptide: PIDOTIMOD." *Journal of Physical Chemistry A, 103*(29), 5823-5832.

**Bakker, H. H. C., & Flemmer, R. C. (2009).** "Data Mining for Generalised Object Recognition." *Paper presented at the ICARA (International Conference on Autonamous Robots and Agents.*

**Basu, M. (2002).** "Gaussian-based edge-detection methods - A survey." *Ieee Transactions on Systems Man and Cybernetics Part C-Applications and Reviews, 32*(3), 252-260.

**Bigun, J. (1988).** "Recognition of local symmetries in gray value images by harmonic functions." *Paper presented at the Pattern Recognition, 1988., 9th International Conference on.*

**Born, R. T., & Bradley, D. C. (2005).** "Structure and Function of Visual Area MT." *Annual Review of Neuroscience, 28.*

**Bowmaker, J. K., & Dartnall, H. J. (1980).** "Visual Pigments of Rods and Cones in a Human Retina." *Journal of Physiology, 298.*

**Brandman, J. (2008).** "A Level-Set Method for Computing the Eigenvalues of Elliptic Operators Defined on Compact Hypersurfaces." *Journal of Scientific Computing, 37*(3), 282-315.

**Brunelli, R., & Poggio, T. (1993).** "Face recognition: features versus templates." *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 15*(10), 1042-1052.

**Can, T., Chen, C. I., & Wang, Y. F. (2006).** "Efficient molecular surface generation using level-set methods." *Journal of Molecular Graphics & Modelling, 25*(4), 442-454.

**Cheng, D., & Yan, H. (1998).** "Recognition of handwritten digits based on contour information." *Pattern Recognition, 31*(3), 235-255.

**Choi, W.-P., Lam, K.-M., & Siu, W.-C. (2001).** "An adaptive active contour model for highly irregular boundaries." *Pattern Recognition, 34*(2), 323-331.

**Cole, L., Austin, D., & Cole, L. (2004).** "Visual Object Recognition using Template Matching." *Paper presented at the Australian Conference on Robotics and Automation.*

**Costa, L. d. F., & Cesar, R. M. (2009).** "Shape Classification and Analysis: Theory and Practice" *(Second ed. ed.). Boca Raton, FL: CRC Press, Inc.*

**Cremers, D., Rousson, M., & Deriche, R. (2007).** "A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape." *International Journal of Computer Vision, 72*(2), 195-215.

**Damaraju, E., Huang, Y. M., Barrett, L. F., & Pessoa, L. (2009).** "Affective learning enhances activity and functional connectivity in early visual cortex." *Neuropsychologia, 47*(12), 2480-2487.

**Drew, M. S., Lee, T. K., & Rova, A. (2009).** "Shape retrieval with eigen-CSS search." *Image and Vision Computing, 27*(6), 748-755.

**Duda, R. O., & Hart, P. E. (1972).** "Use of the Hough Transformation to Detect Lines and Curves in Pictures." *Communications of the ACM.*

**Duda, R. O., & Hart, P. E. (1972).** "Use of the Hough transformation to detect lines and curves in pictures." *Commun. ACM, 15*(1), 11-15.

**Duygulu, P., Barnard, K., de-Freitas, J. F. G., & Forsyth, D. A. (2002).** "Object Recognition as Machine Translation: Learning a Lexicon for a Fixed Image Vocabulary. *In Lecture Notes in Computer Science: Springer Berlin / Heidelberg.*

**Fei-Fei, L., Fergus, R., & Perona, P. (2004).** "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories." *Workshop on Generative-Model Based Vision.*

**Fergus, R. (2003).** "Object Class Recognition by Unsupervised Scale-Invariant Learning.

**Fernandes, L. A. F., & Oliveira, M. M. (2008).** "Real-time line detection through an improved Hough transform voting scheme." *Pattern Recognition, 41*(1), 299-314.

**Ferrari, V., Tuytelaars, T., & Gool, L. (2004).** "Integrating Multiple Model Views for Object Recognition." *IEEE Computer Vision and Pattern Recognition.*

**Ferrari, V., Tuytelaars, T., & Gool, L. (2006).** "Object Detection by Contour Segment Networks." *Paper presented at the European Conference on Computer Vision.*

**Flemmer, R. C., & Bakker, H. H. C. (2005).** "Sensing Objects for Artificial Intelligence." *Paper presented at the ICST (International Conference on Sensing Technology).*

**Flemmer, R. C., & Bakker, H. H. C. (2009).** "Generalised Object Recognition." *Paper presented at the ICARA (International Conference on Autonomous Robots and Agents).*

**Gregory, R. L. (1978).** "Eye and Brain the Psychology of Seeing". *New York: McGraw-Hill Book Company.*

**Grimson, W. E. L., & Huttenlocher, D. P. (1988).** "On the sensitivity of the Hough transform for object recognition "*: Massachusetts Institute of Technology Artificial Intelligence Laboratory.*

**Hawkins, J., & Blakeslee, S. (2004).** "On Intelligence". *New York: Owl Books.*

**Howarth, J. W., Bakker, H. H. C., & Flemmer, R. C. (2009).** "Feature-Based Obect Recognition." *Paper presented at the ICARA (International Conference on Autonomous Robots and Agents).*

**Huang, C. T., & Mitchell, O. R. (1994).** "A Euclidean Distance Transform Using Grayscale Morphology Decomposition." *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16*(4).

**Illingworth, J., & Kittler, J. (1988).** "A survey of the hough transform." *Computer Vision, Graphics, and Image Processing, 44*(1), 87-116.

**Jain, A. K., Murty, M. N., & Flynn, P. J. (1999).** "Data clustering: a review." *ACM Comput. Surv., 31*(3), 264-323.

**Jaume, A., Nicu, S., & Petia, R. (2007).** "Context-Based Object-Class Recognition and Retrieval by Generalized Correlograms." *IEEE Trans. Pattern Anal. Mach. Intell., 29*(10), 1818-1833.

**Karantzalos, K., & Argialas, D. (2009).** "A Region-based Level Set Segmentation for Automatic Detection of Man-made Objects from Aerial and Satellite Images." *Photogrammetric Engineering and Remote Sensing, 75*(6), 667-677.

**Kassahun, Y., & Sommer, G. (2004).** "Model Based Evolutionary Object Recognition System." *Paper presented at the 8th Conference on Intelligent Autonomous Systems.*

**Kovesi, P. (1999).** "Image Features From Phase Congruency." *Videre: A Journal of Computer Vision Research, 1*(3).

**Kovesi, P. (2003).** "Phase Congruency Detects Corners and Edges." *Paper presented at the The Australian Pattern Recognition Society Conference: DICTA 2003.*

**Kozinska, D., Tretiak, O. J., Nissanov, J., & Ozturk, C. (1997).** "Multidimensional Alignment Using the Euclidean Distance Transform." *Graphical Models and Image Processing, 59*(6), 373-387.

**Laika, A., & Stechele, W. (2007).** "A review of different object recognition methods for the application in driver assistance systems." *Paper presented at the Workshop on Image Analysis for Multimedia Interactive Services.*

**Lazebnik, S., Schmid, C., & Ponce, J. (2004).** "Semi-Local Affine Parts for Object Recognition." *Paper presented at the British Machine Vision Conference.*

**Lazebnik, S., Schmid, C., & Ponce, J. (2005).** "A Maximum Entropy Framework for Part-Based Texture and Object Recognition." *Paper presented at the IEEE International Conference on Computer Vision.*

**Lee, Y., & Grauman, K. (2009).** "Foreground Focus: Unsupervised Learning from Partially Matching Images." *International Journal of Computer Vision, 85*(2), 143-166.

**Lewis, J. P. (1995).** "Fast Template Matching ". *Quebec City, Canada: Canadian Image Processing*

*and Pattern Recognition Society.*

**Lindeberg, T. (1990).** "Scale-Space for Discrete Signals." *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*, 234-254.

**Lindeberg, T. (1998).** "Edge Detection and Ridge Detection with Automatic Scale Selection." *International Journal of Computer Vision, 30*(2).

**Livingstone, M., & Hubel, D. (1988).** "Segregation of Form, Color, Movement, and Depth: Anatomy, Physiology, and Perception." *Science, 240.*

**Lowe, D. G. (1999).** "Object recognition from local scale-invariant features." *Paper presented at the International Conference on Computer Vision.*

**Lowe, D. G. (2004).** "Distinctive image features from scale-invariant keypoints." *International Journal of Computer Vision, 60*(2).

**Mark, S., Peter, S., & Stanley, O. (1994).** "A level set approach for computing solutions to incompressible two-phase flow." *J. Comput. Phys., 114*(1), 146-159.

**Marola, G. (1989).** "On the detection of the axes of symmetry of symmetric and almost symmetric planar images." *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 11*(1), 104-108.

**Mokhtarian, F. (1997).** "Silhouette-based occluded object recognition through curvature scale space." *Machine Vision and Applications, 10*(3).

**Mokhtarian, F., & Mackworth, A. (1986).** "Scale-Based Description and Recognition of Planar Curves and Two-Dimensional Shapes." *Pattern Analysis and Machine Intelligence, IEEE Transactions on, PAMI-8*(1), 34-43.

**Morito, Y., Tanabe, H. C., Kochiyama, T., & Sadato, N. (2009).** "Neural representation of animacy in the early visual areas: A functional MRI study." *Brain Research Bulletin, 79*(5), 271-280.

**Naik, A.** "Quality Threshold (QT) clustering algorithm." *Retrieved 10/8/2012, 2012, from https://sites.google.com/site/dataclusteringalgorithms/quality-threshold-clustering-algorithm-1*

**Nene, S. A., Nayar, S. K., & Murase, H. (1996).** "Columbia Object Image Library (COIL-100) ".

**Nolte, J. (2002).** "The Human Brain: An Introduction to its Functional Anatomy" *(6 ed.). Philadelphia: Mosby.*

**Ocg, F. J., & Ney, H. (2002).** "Discriminative Training and Maximum Entropy Models for Statistical

Machine Translation." *Paper presented at the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia.*

**Osher, S., & Fedkiw, R. (2003).** "Level Set Methods and Dynamic Implicit Surfaces" *(Vol. Applied Mathmatical Sciences). New York: Springer Science.*

**Over, R. (1972).** "Feature Analysis in the Human Visual System". *Queenslan, Australia: The Courier-Mail Printing Service.*

**Paul, S., Pascal, F., & Andrew, J. H. (1992).** "Computational strategies for object recognition." *ACM Comput. Surv., 24*(1), 5-62.

**Pinho, A. J., & Almeida, L. B. (1997).** "A review on edge detection based on filtering and differentiation." *Electrónica e Telecomunicações - Revista do DETUA, 2*(1).

**Pinto, N., Cox, D. D., & DiCarlo, J. J. (2008).** "Why is Real-World Visual Object Recognition Hard?" *PLoS Comput Biol, 4*(1), e27.

**Quinlan, J. R. (1986).** "Induction of Decision Trees." *Machine Learning, 1*(1).

**Ragnemalm, I. (1992).** "The Euclidean distance transform in arbitrary dimensions." *Paper presented at the Image Processing and its Applications, 1992., International Conference on.*

**Ritendra, D., Dhiraj, J., Jia, L., & James, Z. W. (2008).** "Image retrieval: Ideas, influences, and trends of the new age." *ACM Comput. Surv., 40*(2), 1-60.

**Rosin, P. L. (2009).** "A simple method for detecting salient regions." *Pattern Recognition, 42*(11), 2363-2371.

**Ross, P. E. (2008,** 2/9/2009). "Why CPU Frequency Stalled." *from http://www.spectrum.ieee.org/computing/hardware/why-cpu-frequency-stalled*

**Sebastian, T. B., & Kimia, B. B. (2001).** "Curves vs skeletons in object recognition." *Paper presented at the International conference of image processing.*

**Sethian, J. A. (1996).** "Level Set Methods and Fast Marching Methods". *New York: Cambridge University Press.*

**Shantz, M. (1981).** "Surface definition for branching, contour-defined objects." *SIGGRAPH Comput. Graph., 15*(2), 242-270.

**Sobel, I., & Feldman, G. (1968).** "A 3x3 Isotropic Gradient Operator for Image Processing."

**Suri, J. S., Singh, S., & Reden, L. (2002).** "Computer vision and pattern recognition techniques for 2-D and 3-D MR cerebral cortical segmentation (Part I): A state-of-the-art review." *Pattern Analysis and Applications, 5*(1), 46-76.

**Teixeira, M. (2011).** "Efficient indexing structures for fast media search and browsing." *Paper presented at the Content-Based Multimedia Indexing*

**Till, Q., Ullrich, M., nich, Lars, T., & Manjunath, B. S. (2004).** "Cortina: a system for large-scale, content-based web image retrieval." *Paper presented at the Proceedings of the 12th annual ACM international conference on Multimedia.*

**Tistarelli, M. (1995).** "ACTIVE SPACE-VARIANT OBJECT RECOGNITION." *Image and Vision Computing, 13*(3), 215-226.

**Uhr, L. (1972).** "Layered Recognition Cone Networks that Pre-Process Classify and Describe." *IEEE Transactions on Computers.*

**Vallines, I., & Greenlee, M. W. (2006).** "Saccadic Suppression of Retinotopically Localized Blood Oxegen Level-Dependent responses in Human Promary Visual Area V1." *The Journal of Neuroscience*(26).

**Vasconcelos, N., & Lippman, A. (2000).** "A Unifying View of Image Similarity." *Paper presented at the International Conference on Pattern Recognition, Barcelona, Spain.*

**Vicente, A. G., Munoz, I. B., Molina, P. J., & Galilea, J. L. L. (2009).** "Embedded Vision Modules for Tracking and Counting People." *Ieee Transactions on Instrumentation and Measurement, 58*(9), 3004-3011.

**Wang, Y., Song, S. H., Tan, Z. J., & Wang, D. S. (2009).** "Adaptive variational curve smoothing based on level set method." *Journal of Computational Physics, 228*(17), 6333-6348.

**Watt, R. (1991).** "Understanding Vision". *San Diego: Academic Press Limited.*

**Watt, R. J., & Phillips, W. A. (2000).** "The function of dynamic grouping in vision." *Trends in Cognitive Sciences, 4*(12), 447-454.

**Wen, W., & Yuan, B. (1994).** "Detection of Partial Ellipses Using Separate Parameters Estimation Techniques." *Institute of Information Science, Northern Jiatong University.*

**Wong, Y. Y., Yuen, P. C., & Tong, C. S. (1998).** "Segmented snake for contour detection." *Pattern Recognition, 31*(11), 1669-1679.

**Xu, D., & Xu, W. (2004).** "Description and Recognition of Object Contours using Arc Length and Tangent Orientation." *Pattern Recognition Letters, 26*(7).

**Yue, W. (2007).** "2D Affine-Invariant Contour Matching Using B-Spline Model." *IEEE Transactions on Pattern Analysis and Machine Intelligence, 29*, 1853-1858.

**Zabrodsky, H., Peleg, S., & Avnir, D. (1992).** "A measure of symmetry based on shape similarity." *Paper presented at the Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on.*

**Zehnder, P., Koller-Meier, E., & Gool, L. (2006).** "Efficient, Simultaneous Detection of Multiple Object Classes." *Paper presented at the 18th International Conference on Pattern Recognition.*

**Zhao, F., Huang, Q., & Gao, W. (2006).** "Image Matching by Normalized Cross-Correlation." *Paper presented at the International Conference on Acoustics, Speech, and Signal Processing.*