Detecting Existence and Pose of Objects within a Single Captured Image

Submitted in partial fulfilment
of the requirements of the degree

of Rhodes University

Michael Horne

November 6, 2005

Abstract

This paper documents a complete image recognition system that searches an image for the existence of known objects. Key features are extracted from images in the form of edges and lines are derived from them. This information is built up into a graph representation containing nodes of object corners in the image and linked by edges. This image data is then searched for correspondences by matching of vertices based on the random sample consensus paradigm of model fitting. This is a model-based class of system, where a predefined set of known models are used for recognition. A method of estimation of the pose and location of the object models is designed and discussed. The goodness of solution is defined and its use in removing inaccurate solutions for the random matches is discussed. Additionally, a strategy for matching multiple objects is within the same image is proposed and results of implementation are discussed. The success of the system is also discussed.

Acknowledgements

I would like to thank my supervisor Philip Sterne for his guidance and feedback given through the various stages of this project. I would also like to thank the Computer Science Department of Rhodes University for facilitating this research and for use of the hardware supplied to me this year.

I would also like to extend a special thanks to Telkom, Business Connexion, Comverse, Verso Technologies, and Thrip for their sponsorship of this year.

Contents

List of Figures

Chapter 1

Introduction

Currently computers do not deal with captured images intelligently. Images are often regarded as just a flat array of data, containing a vast amount of information. The ability to use the information within captured images would create the possibility of decision making with regard to what was contained in the image. Specifically where real world interaction could be possible. Systems could be introduced where vision could collect information about the surroundings autonomously or even allow for computer interactions in real situations.

The field of computer vision is vast, multidimensional and divided. Some research has gone into vision systems modeled on the stereoscopic vision of humans, where depth can be determined by the separation of cameras. Others simply use information from range sensors to directly determine depth information. But these devices are costly and uncommon. A vision system using only what is common to many desktop set ups would be much more useful to normal users.

1.1 Model-Based systems

There are two main approaches when identifying three dimensional objects. There is the view-based where objects are seen within images and the object's structures are learned. This often requires much more information perhaps gained by analysis of two (or more) slightly different views of the same scene and depth information can be inferred. Range data from range sensors has also been used for depth extraction.

Model based systems take a different approach. The objects in the system are known before the system does any analysis. As the objects are known, the information needed from image data is decreased as objects need not be reconstructed, but only matched to the image data. These systems search for links between geometrical models of objects and features present in the image, and this is the approach that will be examined in this thesis.

1.2 Problem statement

A model-based approach will be taken, and thus there will be a set of objects known for use in recognition. An internal representation for the object models needs to be created. Features need to be extracted from the images and need to be defined and identified from within images in a way that bears a likeness to the representation of objects known to the system. Given a set of objects, the extracted image features will be searched for the occurrence of one or more of these objects.

To search for objects, the image and model data needs to be linked in a way that will allow for the robust detection of objects and for the accurate estimation of their pose and location in the image. The image data will then be used to estimate the pose and location of the objects present in the image.

Multiple objects must also be able to be detected, and the algorithm for estimating pose should be capable of handling estimation for multiple objects.

1.3 Chapter summary

In the following chapter, methods of image preparation and feature extraction are discussed. The effects of noise and a means to filter the noise present in captured images is covered. The features needed for later stages are also defined. Edge detectors are compared and an edge detector for the system is chosen. A suggested internal representation for the image data is defined as well as the means of extracting this information from raw image edges.

In chapter 3, random sample consensus, is introduced as a robust method for model fitting. The result of using a random method and the effectiveness of method are investigated. This chapter defines how the image data will be linked to the object models, in preparation for the next stage (pose estimation.)

Methods of pose estimation are discussed in chapter 4, and an optimised solution based on one algorithm is implemented. The pose estimation will rely on the matching stage before, as a base for calculation. A means of judging the estimated pose is created and the effectiveness of using this statistic to distinguish between good and bad/erroneous solutions evaluated.

In chapter 5, strategies for matching of multiple objects are discussed and the success of matching is documented for various situations. And in the final chapter, conclusions are made about the success of the vision system as a whole.

Chapter 2

Image Preparation and Feature Extraction

This part of the system involves taking a captured image, and using various techniques, to extract the data that will be useful in the matching stage to follow. Images essentially have too much data to analyse directly. For this system a limited subset of only the essential data will be extracted for use. Ultimately, it is the most vital process in the system as the results will all be directly based on the information inferred in this stage. The image will no longer be needed after this stage.

This chapter discusses a means of coping with image noise, which is present in all captured images. Once the image has undergone noise filtering, edge information is extracted from the image. These arbitrary shaped edges will be used to converted into straight lines. Vertices will be created by analysing the intersections of these lines and will represent corners of objects the images.

A representation for the image data that resembles the actual geometry of an object will be formed by creation a graph like representation of the image data where vertices are nodes connected by edges.

2.1 Image Noise

Real images are often degraded by random noise, which will in most cases degrade the performance of image analysis in the later stages of this work. Image capture can suffer from various types of noise. Two such common sources of noise are white noise and impulsive noise.

Idealised or white noise, is a common way of representing noise in an image. A special case of white noise is Gaussian noise. This type of random noise has a constant amplitude and is often a good approximation to the noise that usually occurs in an image[Sonka et al., 1999]. This approximate model assumes the noise is additive in nature evenly distributed over the image with with mean zero and some positive deviation(σ_n.) The amount of noise in the image can be quantified by calculation of the SNR (signal to noise ratio) [Sonka et al., 1999, Trucco and Verri], where $\sigma_i$is the standard deviation of the image data. The SNR is thus generally stated as $\frac{\sigma_i}{\sigma_n}$, thus the larger the ratio the clearer the image.

The second type of noise, impulsive noise [Trucco and Verri] is type of noise more closely related to a corruption of the image. Pixels within the image are substituted with new ones with random values. This type of noise usually results due to a transmission error from the capture device.

With any type of noise, complications will result in the edge detection stage. Edges could be detected erroneously due to prominence of noise in the image. Both types of noise mentioned will contain high frequency components. Since the edge detection stage will rely on this high frequency data for edge detection, false edges may be detected originating purely out of noise, or edges may grow tails resulting from noise at the ends of the actual edge being mistaken as being extensions of the true edge.

Any false detections will result in later stages being more time consuming as all the data extracted will have to be analysed, and thus methods need to be introduced to reduce the amount of noise in the image. Each of the following filters have their strengths at reducing the above mentioned types of noise. However, filtering may have the effect of removing the large gradient components of the edges that would allow them to be detected. Real edges are at risk of being lost, thus the parameters of the filters have to be adjusted in order to minimise noise and minimise the loss of edges.

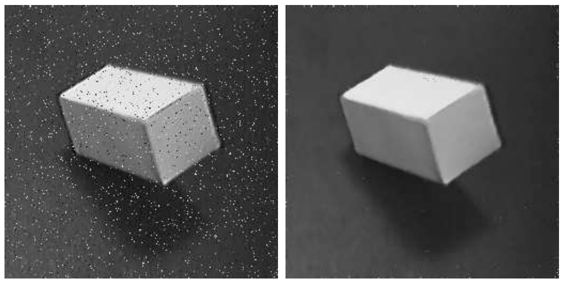
In certain cases, edges will still be lost, these cases will be discussed and necessitate the robust algorithms as described in the following chapters.

Gaussian

The Gaussian filter[Trucco and Verri], as the name suggests is based on the Gaussian normal distribution. Again this filter can be seen as being applied one pixel at a time for each pixel in the image. Thus we can consider the one pixel case for the explanation.

When applying the filter the current pixel and the surrounding ones are used to calculate the new value. To define what proportion of theses pixels is used in creation of the new pixel, we consider the the Gaussian kernel.

$$G(x,y) = \frac{1}{\sqrt{2\Pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



(a) Image damaged by impulsive noise (b) After median filter has been applied

Figure 2.1: The effect of a median filter

The Gaussian distribution (given above) is used to generate a kernel used in the filtering process. This conversion of data into discrete proportions is approximate and care must be taken to ensure all the columns and rows of the kernel add up to one. Incorrect kernel construction could result in the colours drifting brighter or dimmer depending on whether the sum of the elements of the kernel do add up to more or less than one.

Median Filter

Seul et al. [2000] suggests this filter which is aimed at removing impulsive noise (also known as salt and pepper noise.) Given that the noise is due to errors in the capture of the image, the information is not just inaccurate, but not related to the original data in any way. An alternate to averaging type filters is thus needed.

The filter is designed in a way that will take advantage of the fact that erroneous pixels due to this type of noise differ from the surrounding pixels greatly. For each pixel in the image, that pixel and it's surrounding pixels are recorded, and sorted. The median of these pixel is used to replace the original pixel. Noisy pixels are less likely to occur as the median than undistorted pixels which will usually have similar values to each other. An example of the sort of output that can expected is illustrated in 2.1.

2.2 Edge Detection

The sharp variations in intensity in images, often belonging to lines, curves and contours, are the basic elements for recognition. These features are detected as as chains of edges points. Edges are definitive characteristics of objects Trucco and Verri. Knowing the location of edges in an image would allow for mapping of image features to the features of the objects. These mappings of features will provide a basis for the estimation of the pose and location of the objects in the image at a later stage (in chapter 3.)

The Canny, Revised Canny and DSC are common edge detectors and the merits of each are discussed here.

2.2.1 Canny Edge Detection

The Canny Edge detector [Canny, November 1986] operates on images to find areas of interest. Particularly areas in images of interest are characterised by quick changes between pixels and their neighbours. The canny edge detector is designed to take advantage of this, by analysis of the gradients in the image. These are the features in the image that will be of the most use in later stages as the edge data will be extracted from them.

For each pixel in the image the gradients are calculated by doing a discrete calculation of the gradients in both the vertical and horizontal directions to provide and estimation of the gradient at each point, thus a local edge normal (perpendicular to the edge) can be calculated. Two new images are formed from this information. A strength image, with each pixel representing the value of the gradient at each pixel and an orientation image with representations of the edge direction at each pixel.

This output will in most cases have wide regions with large gradients where it is possible that an edge may exist. For this information to be useful edges would ideally be only one pixel wide and exist along the most likely path of an edge.

A technique call nonmaxima suppression is used to detect areas that are not local maxima, and remove them from consideration as an edge. The possible gradient slope directions are simplified to one of four directions. Taking the component's direction with the maximum component of the original direction, that pixel is compared to its two neighbours in the approximated direction (normal to the edge). If it is less than at least one of those neighbours, then the pixel information is zeroed. Thus the edge data is trimmed to form edges of only one pixel wide.

Unfortunately this output often includes detected edges which are often not part of any interest[Trucco and Verri]. This can be solved by hysteresis thresholding. Two thresholds are defined

T_1 and T_2, where $T_1 > T_2$. Edge date will only start being tracked once the gradient value is greater than T_1 and will be tracked in both directions along the edge while the gradient is still larger that T_2. This prevents the breaking up of edges with larger variations in gradients. Also edge points of little significance are removed and since these edges often originate from noise, the noise is also reduced.

2.2.2 Revised Canny

Implementations of the Canny detector are often only approximations of the original specification defined by [Canny, November 1986]. While the common implementation is preferred due to its simplicity, often obvious edges are missed. An improvement is suggested by [Ding and Goshtasby, 2001].

The problem, it is suggested, lies in the implementation of the non-maxima suppression stage of the edge detection. This stage, while ensuring that the edges are kept to thinned single pixel traces of the edges. However, this prevents the possibility of being able to detect edges that branch as in 2.2.

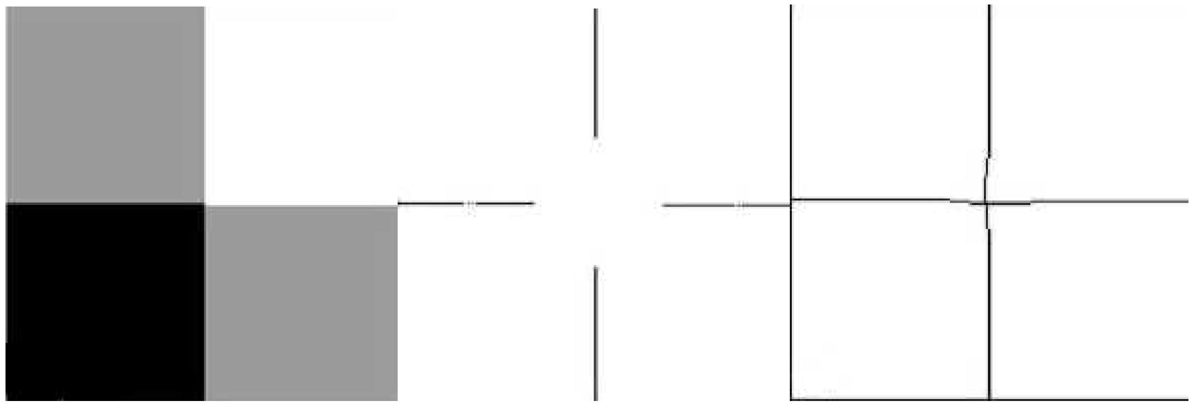
This detector has the advantage over the canny edge detector in that it can detect two types of edges, minor and major edges. Where the canny only detects what are called major edges by Ding and Goshtasby [2001]. Further edge detection is done excluding the already detected edges, thus finding what they call minor edges. If these edges have endpoints that connect with major edges they are included in the result. This technique has an advantage over the Canny in the way that it lacks the discontinuities that the canny results in when the two lines intersect. However, the extra computational effort and complexity of implementing this revised Canny detector will be unnecessary due to the algorithm used later to find image vertices (or corners) as explained in section 2.3.4.

2.2.3 DSC[1] Edge Detector

DSC Edge Detector (DSCED) and it's sister detector DSC anti-noise edge detector (DSCANED) have been been implemented by Hou and Wei [2002] as a hybrid approach to edge detection. DSCED a fine grained detector for detection of fine edges and DSCANED, a coarse grained edge detector for use in noisy images.

The convolution used in the detection process is based on an approximate version of Shannon's delta kernel (differentiating kernel.) By varying the parameters of the delta kernel, the

[1]Discrete Singular Convolution



(a) Test Image (b) Regular Canny (c) Revised Canny

Figure 2.2: Results of the revised Canny[Ding and Goshtasby, 2001]

frequency response of the convolution is altered, thus the response can be limited to frequencies lower than that of image noise.

Comparisons to other edge detectors such as the Canny, Pewitt and Sobel indicated that DSC detectors performed better in certain circumstances with higher image noise [Hou and Wei, 2002]. However, in general, the edge detector proved to perform as well as that of the Canny, but since a choice of which edge detector (fine grained or coarse grained) had to be made based on noise, the expense of calculation of noise in an image might suggest the canny would be a better choice in the general case instead.

2.3 Design

Objects in our system will be represented as wire-frame data. Ideally, information inferred from images would bear a strong resemblance to this data, which would allow for simpler more direct matching to the object data, as detailed in chapter 3.

2.3.1 Image Preparation

The image needs to be suitably prepared before any analysis takes place. As discussed in section 2.1, noise in the image will greatly affect the performance of the detection process and at least one of the above noise filters mentioned has to be used to counteract the effects of noise. A median filter, effective at removing impulsive noise would only be necessary if the images captured actually were affected by this type of noise, otherwise it's use could be discarded for reduction of the computational effort of a median filter. The median filter proved to be unnecessary as the capture device used hardly suffered from this type of noise.

White noise will always occur in a captured image, especially when using low quality devices such as web cams as for capturing. White noise occurs at all frequencies, as mentioned in 2.1. However, only some of these frequency ranges needed to be filtered out. In images, edges on objects seem obvious to the observer as they are areas of quick change in colour or intensity which translates to high frequency change. This in turn means that noise of a high frequency nature is noise that will affect the edge detection in later stages the most. Low frequency white noise has little to no effect on edge detection as edges are in the high frequency domain.

A Gaussian filter[Trucco and Verri] is thus chosen. A Gaussian filter is an effective low pass filter allowing high frequencies in the image to be suppressed. The selection of exactly what parameters are used while applying a Gaussian filter is essential. While high frequency noise will be suppressed, edges which are also high frequency features in the image, will be affected. The maximum amount of filtering will have to be applied while still keeping a the loss of feature data to an absolute minimum.

There is a clear reason for the choice of the Gaussian filter over another filter such as an averaging filter for example. The frequency responses of the two kernels have been analysed in figure 2.3[2]. The spatial frequency is given on the horizontal axis in periods per pixel, and hence any frequencies higher that 0.5 are not meaningful. Where the averaging filter does filter out higher frequencies to an extent, it leaves some high frequency artifacts. These proves troublesome still for later analysis as edges might still be detected due to these artifacts. Hence the choice of the slightly more computationally expensive Gaussian filter is easily justifiable.

Edges will inevitably still be lost in some cases not just because of filtering but also due to low contrast between faces and the system will have to take this into account in later stages. Further analysis of the data needs to be done in a way that will keep it robust enough to handle the loses of one or more edges, so that matching and pose solution can still function effectively despite this missing data.

2.3.2 Feature extraction

The ultimate goal of feature extraction is to reduce the large amount of data in the image to a small amount of useful information. This conversion needs a clear definition of what type of

[2]Image originally from http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

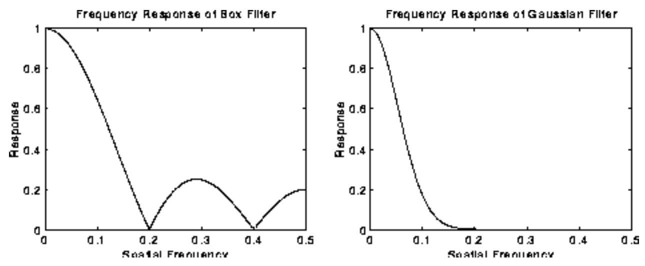


Figure 2.3: Analysis of the frequency response of Gaussian and averaging filter kernels

information is useful. Edges are characteristic of the objects in an image, thus are ideal for use a a base for further processes. Unfortunately, the result is still that much image data has been discarded, such as color and shading of objects. The effects of this extraction will be investigated later (section 2.4.)

The edge detectors discussed earlier (in section 2.2) each had their various merits. The revised canny edge detector seemed to be a good choice for detection of edges, as the branching would allow for edges to intersect and meet up at the corners of the objects. Corners will later be a important part of recognition, however the methods chosen for vertex detection (section 2.3.4) will circumvent the need for revised detector as the minor edges will be irrelevant. Also, the DSCED and DSCANED detectors showed little consistent improvement over the original Canny detector, and given that it is still widely used by recognition systems, is a tribute to it's effectiveness. In a comparison done by Heath et al. [1996] it was suggested that the Canny would provide the best results over a large range of images. The Canny edge detector was chosen for the the purposes of this system.

Canny takes a gray scale image as it's input, traces the edges in the image and outputs edges in the form of lists of pixels. Each pixel list represents and edge detected in the image.

2.3.3 Extracting lines from arbitrary shaped edges

The edge detector is designed to find edges in the image of any shape. Since the object model we use is one based on a wire frame, the edges will have to be in the form of straight lines to be useful. Simple techniques could involve fitting a line by connecting the endpoints of the edge, though this is ineffective if edges have artifacts such as small tails on the end of real edges. Instead a least squares regression method could be used to find the best fit [Seul et al., 2000].

This is an incomplete solution however, because there is no way that one can ensure the output from the edge detector contains one edge only. The conversion thus must be able to determine how many lines are contained in one edge, as well as be able to split them up into optimum sections.

Seul et al. [2000] suggests a progressive technique for determining the lines along a given edge. It starts connecting a line to an adjacent point on the edge further along. Each time the point furthest from the endpoint of the edge is moved along, keeping in mind the area between the line and and the actual edge. If this area becomes larger than a user defined threshold, the process is stopped and the line is estimated using regression. Then a new line begins from this point until all of the edge is used and all of the edges have been examined. This method will be used to resolve the edge data into lines.

In the implementation, it might be advantageous to ignore short lines. Due to the nature of the edge detection, short lines are often detected from noise and could, while adding extra vertices for search might add no useful information to the extracted image data.

2.3.4 Converting lines into vertices

In the matching stage, covered in chapter 3, vertices are needed to do the initial matching. Now that the data has been converted into a list of straight lines, the points of intersection of the lines could be used to define the corners of objects.

However, due to the nature of the edge detection, the edges often do not have endpoints that meet or in some drastic cases, endpoints might not even come close to matching up (see example image 2.2 (b).) To remedy this the lines are to be extended until they do intersect. The distance the lines are extended would have to have to be determined, otherwise every line would add an intersection for every other line in the image.)

Ideally, the distance the line would be extended would depend on the distance of the edge from the camera. This would mean that in real world coordinates, the lines would be extended relative to their actual length, but since this would only be able to be determined after a solution was found, it would have to, instead, be based of the length of the line in the image.

For a intersection to be valid, it must either be on the line, or less than a certain distance from the ends of each line. As an optimisation, instead of checking for each of those cases, the distance can simply be measured from the midpoint. Thus for an intersection to occur the following formula would have to hold for the for both lines.

$$\frac{1}{2}(\alpha+1)L_i > \sqrt{(x-x_{imid})^2+(y-y_{imid})^2}$$

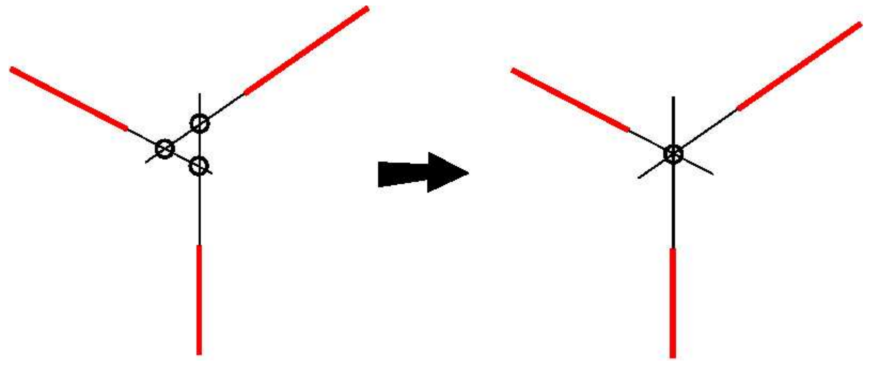


Figure 2.4: Vertex Clustering

Where L_i is the line length and x_{imid} and y_{imid} are the midpoint coordinates for that line, and x and y are the calculated intercepts. α is the tolerance level given as a ratio of the total line length. This would mean that long lines in the image, given that the longer a line is the more likely it is to being an edge, the more tolerance it is given with respect to missing portions on the ends.

Through experimentation, a value for α around 25% provided enough extension to allow most valid extensions to occur. Over extension is still a problem however, and is discussed in section 2.4.5.

2.3.5 Vertex Clustering

Edge detected regions around corners will often have more than two edges intersecting to form the corner. This can result in the creation of n^2 vertices in close proximity (where n is the number of lines.) This is problematic as the performance of the sampling technique used later will be heavily dependent on the total number of vertices detected. The vertices are thus clustered together (illustrated in figure 2.4.) Each vertex is chosen in turn, and vertices within a prescribed distance are removed and a vertex with the average position is added.

2.3.6 Representation of information

The information retrieved from the images needs to be represented in a way that will keep as much of the information available for later processing. Just knowing the locations of intersections is not enough. Edge data (in section 3) will need to be compared to the image edges as well. For

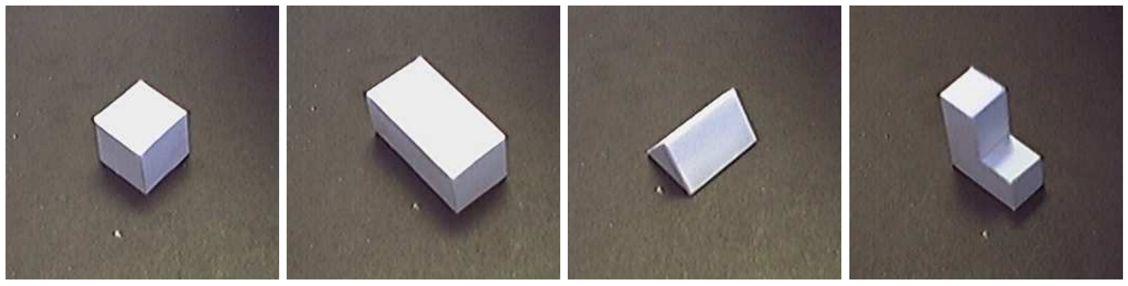

(a) Cube (b) Rectangle (c) Prism (d) "L" shape

Figure 2.5: Objects used in test scenarios

this reason, a graph like structure is built up from the image data, where the vertices are the graph nodes and the edges are links. The image data now has a geometry more like that of the object models (discussed in section 4.4.1.)

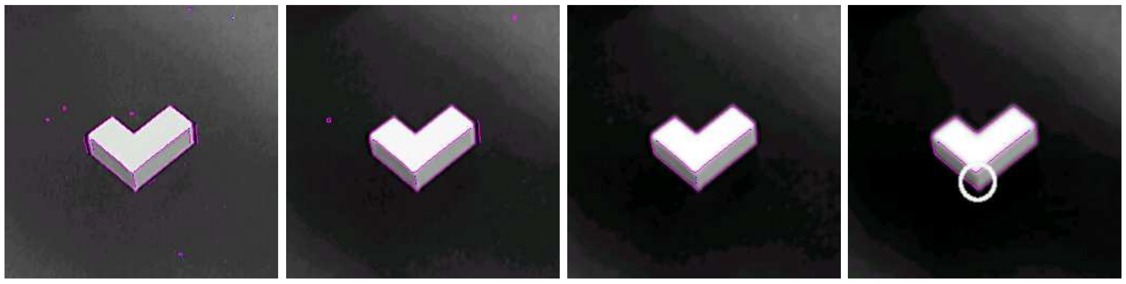
2.4 Results

Some of the results found were accumulated during the implementation of the recognition system, however for testing of the final system, a larger set of test data was accumulated for grading the general performance of the various stages of the recognition process. The test data-set and results from the image preparation and feature extraction stages are discussed in the following sections.

2.4.1 Test Data

Images of four different objects (as seen in figure 2.5) were placed in various poses and attempts were made to recognise each of them. For each object, 10 images of varying levels of difficulty of pose were chosen for the test images to test the robustness of the algorithms. Degenerative object poses (most difficult cases possible) as where the object was viewed from a position perpendicular to one of the faces were chosen for some of the test cases as well as views more like those in figure 2.5 were setup, being easier cases.

Each of the test objects used have varying amounts of symmetry. For example the cube has the most, and a many different correct poses could estimated for any one cube in a single image, whereas the “L” shape would only have one possible pose. Because a large amount of symmetry



(a) No noise reduction (b) Gaussian kernel size 3 (c) Gaussian kernel size 5 (d) Gaussian kernel size 7 (edge lost)

Figure 2.6: Effect of different Gaussian kernel sizes on image noise

of an object would cause more poses to be estimated, the many different poses would be found and far more frequently and result in a quicker solution of pose.

Due to the variance in the quality of images due to the capture device, the brightness level was controlled so that it would be as constant as possible for the capture of the test images. This should not usually be necessary and the use of a higher quality image capture device would allow for control to be left up to hardware (see section 2.4.4.)

2.4.2 Noise

Illustrated in figure 2.6 is one image with varying levels of a Gaussian filter applied to them. In the first image, even the slightest deviation due to noise and in this case tiny particles of dirt are detected as edges. This is highly undesirable as surfaces of a rough nature would exhibit the same artifacts. Also the capture device seemed to exaggerate large changes in intensity creating double lines as the intensity value normalised (in figure 2.6 a.) Even with a small amount of filtering the noise can be greatly reduced, although a Gaussian kernel of size 5 appeared to be the optimum for noise removal.

2.4.3 Shadows

Shadows are image features that end up being extracted just as object edges do. In some cases however the detection of shadows can prove useful to the system. Edges detected due to shadows will meet up with where object corners exist. Consider figure 2.7. The vertical edges were lost, but the vertices were still recovered due to the presence of shadow edges. This case is not

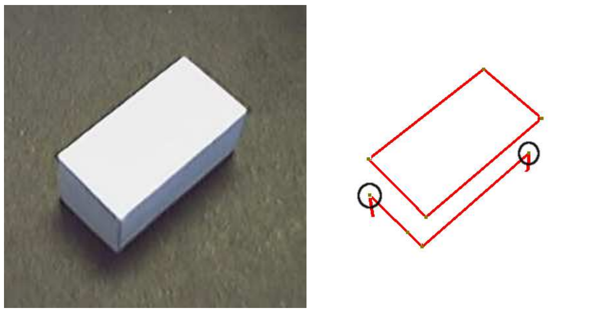

(a) Image of rectangle (b) Shadow lines intersect at points of real vertices

Figure 2.7: Cases where shadows create correct vertices

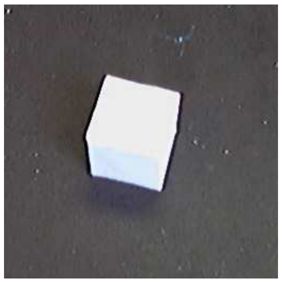


Figure 2.8: Example of an image where poor contrast between object faces occurred due to incorrect camera adjustment occurred

insignificant as is was successfully detected as a rectangle in at a later stage.

2.4.4 Effects of poor contrast

Images with poor contrast seemed to be the biggest issue in detection. Due to the low quality of the camera, and its adjustments to light conditions, most of the time faces on objects that are obviously separate to the naked eye can lose their contrast due to the camera's poor adjustments to lighting conditions (as in figure 2.8.) For this reason edges are often lost. Sometimes this is unavoidable and would instead be a good test of the system's ability to operate with missing data. The pose estimation will need to be able to function with sufficient robustness to make the effects of small amounts of missing data negligible.

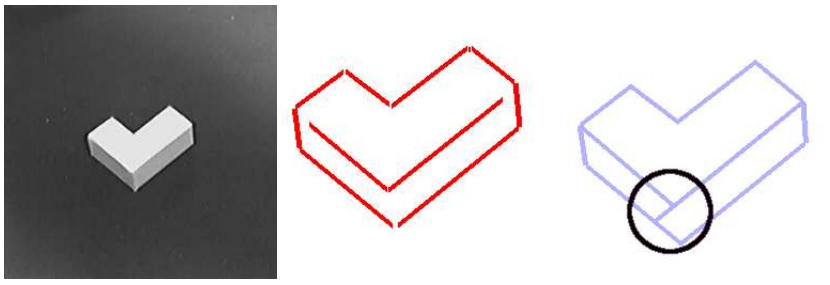

(a) Original Image (b) Edge Information (c) Line incorrectly extended

Figure 2.9: Example of edge over extension

2.4.5 Over Extension of Edges

For longer lines there are cases where the extensions made for intersections to be possible will result in extra vertices that are made in error. As seen in figure 2.9, the one edge was extended some distance past it's actual length. This results in the line making another intersection, which has the effect of creating a false short line in the detected image data. This should not adversely affect later stages as the existence of the false line would be identical to a case where a line is detected from noise.

In the test data over extension errors occurred very often (for 27% of the images) occurring especially for more degenerate poses of the object. However, there was no significant correlation between their occurrence and failures in detection of objects, and thus we assume the effects to be negligible in general. Attempting to reduce this amount would result in some of the lines being under extended, and since the over extensions seemed to have no adverse effect on the pose estimation, the amount lines were extended was left at this level.

2.4.6 Other Issues

Often when slightly degenerate poses are encountered the vertex clustering causes the edges of barely visible face to be squashed together. The vertices on the ends of these valid edges are close enough together to be clustered (in error.) This can be fixed by the adjustment of the tolerance for the clustering of vertices. When sufficiently small, the possibility of clustering is significantly reduced.

In some instances of feature extraction, loss of multiple vertices occurred from the failure of the detection of one edge. Since intersections of edges are used to find vertices, cases will occur where a edges will be left stranded and no intersection will be possible.

2.5 Conclusion

This section discussed the various steps necessary to extract straight lines and vertices from edges in the images. A Gaussian filter proved to be a effective method of reducing noise in images and allowed the minimisation of false edges detected in the image. A method for converting the edge information to lines was defined and method of finding vertices (representative of corners of objects) was covered. A representation of the two dimensional the image data was created that would be easily compared and matched (in the next chapter) to the object models that exist for recognition.

Chapter 3

Matching

3.1 Introduction

In model based recognition systems, matching where the data extracted from the image is used to calculate the solution. For pose calculation to occur, some link between the data in the image (extracted by methods detailed the previous chapter) and the geometry of the object must be found.

The correspondences may take on various forms, depending on the strategies involved in the solution. Points, lines or surface normals (or combinations of which) could be used to estimate the pose of the object. The matching technique has to be robust and not allow small sets of missing data or noise to adversely affect the outcome.

Matching in our system will be between the vertices detected in image data (representing object corners) and the vertices in the object models. The matches will be used at a later in calculation of the pose of the objects.

The a method of matching of vertices (covered in the previous chapter) to object models, based on the Random Sample Consensus style of fitting method and various possible optimisations to its design will be covered in this chapter. Results of the success of this method of matching will also be investigated.

3.1.1 Random Sample Consensus

Fischler and Bolles [1981] introduce a new line of thought with regard to general model fitting. The applications of their research involve use of RANSAC[1] Cantzler in solution of the location

[1]Random Sample Consensus

determination problem.

Statistically, model-fitting is usually done with respect to all the data points being fitted. This holistic approach, while using all the data available for fitting models, does have certain disadvantages. Outliers could be included in the model fitting procedure that would make fitting inaccurate for small sets data sets.

Later, cases are discussed where there are multiple models that need to be fitted to the data. In this case we need a method of differentiating between data belonging to different models. This task could be complex, but with the use of random sampling methods we could circumvent this problem.

RANSAC uses a completely different paradigm for model fitting. For each model being fitted, the least number of points needed for a solution are selected at random. Then the model is then extended to fitting all the remaining from the the initial guess. The solution is judged on the number of other data points that can now be added to the initial guess. This process of making an initial guess is then repeated many times to allow many different possibilities to be considered.

This method is more robust with respect to fitting multiple models as outliers will be discarded as they would form models with minimal coherence of other data points. Also in the event of multiple matches, there is likely to be a number of good solutions that apply to different model with different parameters. Thus, both models could be considered as solutions as long as the data points are separated amongst the two solutions.

3.2 Design

For our purposes, the RANSAC method is ideal, as in our applications, noise is common, and false positives in feature detection could result in many outliers, and if possible as many outliers need to be filtered as possible. Multiple models will definitely be present and as many object models need to be fitted to out input data as possible to allow for detection of multiple object types.

3.2.1 Classification

A problem of classification exists. For the input data, which models need to be fitted and which data points do they need to be fitted to. Fischler and Bolles [1981] suggest that the problem of classification needs not be solved in this initial model fitting stage. Instead, a trial and error approach need be run instead. Pose estimation (chapter 4) will be used as the classification method. Models will be fitted at random, and then later tested for correctness.

3.2.2 Matches required for solution

Essentially it is useful to know the number of matches required to ensure a successful match. However, since the process is random, a correct guess can essentially never be guaranteed. Thus we need to calculate the number of matches instead that would allow for a certain level of certainty that a guess would be correct.

The first step would be to calculate the probability of any single guess being accurate. Say there are n data points in the image and we need to select m points from those. We will also assume that all the object's vertices have been correctly detected and are present in the image data. The object has m of it's vertices selected at random initially. The random selection of vertices must now coincide with the object vertices and thus the probability is given by

$$P(correct\,guess) = \frac{m!}{n!}$$

for each match.

The number of trials required for a match to occur with certainty α is:

$$N = \frac{\log(1-\alpha)}{\log(1-P)}$$

For example, given a situation with $n = 12$ and $m = 4$ one would need around 40000 guesses to allow for a 95% certainty of a correct guess. Thus the order of the number of guesses required is large. Each of these has to be tested for correctness individually.

3.2.3 Reducing the Search Space

Currently the search space of randomly selected matches is large. To remedy the problem it is proposed that these random matches are improved for slightly more intelligent matching. To make use of simple rules and exclude obviously incorrect matches would allow for a quicker overall solution as the next phase is more time consuming than the random selection phase. However these strategies will only affect the matching process, while the solution phase will still rely on matched vertexes alone.

Similar Overall Geometry

Rather than just selecting vertices, one can consider that no vertex can exist on its own, without having a corresponding edge connected to it. Thus points will still be chosen at random, but each subsequent vertices chosen, must be linked to and of the previous vertices by at least one edge.

This would also have an effect on the proximity of the vertices of the matches made, forcing the points to be closer to each other as they must be connected by an edge. Points selected from an image will now have a greater chance of all being selected from the same object. Reasons for not implementing this are discussed later.

Similar Properties of Vertices

Another approach would be not to insist that edges were necessary. The first match is made at random as before. As each new vertex is added it would be checked against the already selected vertices to have the same connectivity as the corresponding vertices in the object had. This excludes less of the matches that the previous method, but is much faster to check at run time. In theory, this optimisation would have no effect on the robustness of the matching process as it would only remove matches that are flawed.

Implementation

During initial implementation of the first method to reduce the state space, there were some drawbacks found that caused the method to be excluded from the system. Unfortunately, too many checks had to be made to ensure that after the initial correspondence was set, it did still allow for all the rest of the correspondences to be possible. To check for this the size of the local image graph and the object model had to be checked to contain at least the number of vertices required for a match. This is a complicated and computationally heavy step. Alternatively, if a random selection of the other correspondences fails too many times the initial matching process could be restarted. This, unfortunately again defeats the goal of the search space reduction.

The second method also appeared to be far more computationally intensive in checking the connectivity of the vertices. The pose estimation stage was less computationally expensive than both of these methods of reducing the search space and thus, checking for logical error in the initial match would take longer to compute than the pose estimation and validate it (as discussed in the next chapter.) While passing fewer matches through, it would take longer to generate these matches, defeating the goal of quick recognition.

3.3 Results

Now that the objects have been matched the accuracy of the matching process can be quantified. The table below shows the results of the matching process. The percentage for each of the objects is the proportion of correct matched to total matches. A larger number of vertices would mean a smaller chance of finding a solution.

The object test images were captured in controlled environment and thus contained very little noise. The percentages this table are thus close to direct indications of the number of matches required on average to guess a correct match. The varying numbers are due to the symmetry as discussed earlier and the number of vertices in each object. The order of the table is from most symmetric to least, where the "L" shaped object model will only have one correct pose for occurrence in an image because of it's lack of symmetry.

Object Model	Correct correspondence guess percentage	Symmetric positions	Vertices
Cube	0.91%	24	8
Rectangle	0.11%	8	8
Prism	0.55%	6	6
"L" Shape	0.004%	1	12

3.4 Summary

The use of RANSAC is easily justifiable in the case of a small vision system. RANSAC has been proven as a robust method for the fitting of models Fischler and Bolles [1981] while still remaining a efficient technique for making matches. Some optimisations for matching were proposed, but during implementation proved too computationally expensive.

Matching has been done at random, and essentially what has been created is a large set of different matches containing different correspondences. Most of the matches are incorrect. Classification of these matches will be done in the next chapter on pose solution.

Chapter 4

Estimation of Pose

The problem of pose solution lies in the loss of data that occurs in projection of an object into an image. Depth information is lost. Also there is the matter that objects captured also have undergone a perspective projection that makes a solution even more complex.

The image information must be used to constrain the possible positions that the object could lie in [Fischler and Bolles, 1981]. The constraints are defined as covered in the previous chapter on matching. Estimation of pose uses these matches to for the estimation.

The pose estimation needs to be robust enough to deal with missing image data, as quite commonly, edges and vertices are lost in the feature extraction stage (chapter 2.)

Because of the RANSAC style matching process, covered in the previous chapter, ideally the pose estimation technique would be fast enough to test thousands of matches and be able to judge quickly whether or not the match made was in error.

Many methods of pose solution in model based scenarios are available. All used correspondences of some sort between the image and object data. The Newton Raphson, Least Squares and POSIT methods are covered in this paper. Since the matching method covered in the previous chapter are made at random, methods of validating the matches are covered here. Methods judging the overall success of a match by use of a goodness ratio is discussed, as well as results indicating the success of the pose estimation overall.

4.1 Newton Raphson Method

This method was used in the SCERPO[Lowe, 1987] vision system. The system made use of matching wire-frame models and image features. To increase robustness image lines were matched instead of image vertices. The match is based finding correspondences between lines and trying to fit lines so that they pass through one another rather that matching the entire lines (matching endpoint to endpoint.) This adds a certain robustness since often the feature extraction process fails to extract the entire line length. The idea being that the direction and position of edge lines is often accurate, but the endpoints are often not. In doing so the importance of the endpoints is minimal.

An initial guess of the pose and location is made by either estimating from known constraints on the orientation (when an object can only lie in certain ranges of positions) or just at random (no constraints). When the orientation of the model was initially within 60 degrees of the solution it showed excellent robustness in obtaining a correct solution.

Due to the mathematical methods used (the Newton-Raphson method), it is by nature an iterative process of solution. Implementation is in such a way that the initial guess need not be very accurate, and may contain matches that were not in the final solution. Adaption in the algorithm allows for addition of new associations between model data and image data to increase the accuracy [Lowe, 1987] the resulting match. With each iteration more data from the initial feature extraction can be associated, based on certain parameters. The model is projected onto the image data and from this, selections for addition can be made based on factors like parallelism, co-linearity and proximity. These more advanced technique reduce the need for backtracking by making more accurate guesses as to suitability of the matches, and gives an overall performance gain due since backtracking is time consuming.

This method however, still requires an initial guess which when guessed incorrectly would cause the solution never to converge despite the match made being accurate. Even if a few random guesses were tried for each match, this would create a doubly iterative algorithm which might be very time consuming. Considering that process would be repeated many thousand times, it would seem that another method would be more suitable for use in this system.

4.2 Statistical least squares iterative method

A least squares style algorithm for solution of pose is proposed by Or et al. [1998]. Their system breaks pose estimation into two linear processes. Depth recovery and pose calculation are separated. The function e^2 is a measurement of the current fit and when a global minimum is found R and T would represent and estimated pose and location . Essentially is represents the sum of squares total of the distance between each projected object vertex and their corresponding image vertices.

$$e^2\left(R,T,\{d_i\}\right) = \min\left\{\sum \|d_i\hat{v}_i - (RP_i + T)\|^2\right\}$$

R and T are the rotation and translations represented as matrices and $P_i$is a vector representing the location of the i'th model point. Additionally $d_i$corresponds to the depth of that point along the projection ray v_i that is relative to the location of the point in the image.

Since the original 3D coordinates are transformed by the perspective projection, the objective function is nonlinear and complex. So instead of finding a minimum directly an iterative process is used to find a solution. The parameters R and T are calculated, assuming $\{d_i\}$are correct. Then $\{d_i\}$ are calculated using R and T. These steps are repeated until the parameters converge on a solution.

One problem that is introduced by using an iterative method is that now an initial guess is needed. Depending on the quality of the initial guess, there is a possibility of minimising the function to a local minimum instead of the actual minimum.

This method, is still time consuming, and required many floating point operations to achieve a stable solution. Also, in each iteration, the calculation of R and T require the SVD needs to be constructed, which is also done in an iterative process. Thus perhaps a less computationally expensive method could be found in favour of this method of pose estimation.

4.3 POSIT

This method proposed by Dementhon and Davis [1995] takes a different approach from the above methods. They define the two algorithms that they use to compute the pose of an object. The POS algorithm (Pose from Orthography and Scaling) and POSIT (POS with Iterations.)

Use of the SOP[1] projection model

The scaled orthographic projection is an approximation to the "true" perspective projection. However, it is important that the situations that this approximation is a good one are identified. The approximation involves scaling of the object's vertex vectors uniformly instead of using the depth values for to determine the scaling factor. Thus to ensure that the approximation is a good one, the depth (z values) of the object must be similar enough to be treated as the same. The

[1]Scaled Orthographic Projection

approximation is good in cases where the object is small and reasonable distance from the camera, causing distances between the z value to be insignificant in comparison to the distance from the camera. Thus the images taken for the test data (section 2.4.1) would have to be captured considering these criteria.

POS

The algorithm uses the SOP model in calculations. Instead of having to deal with the complex solution process involved when a solution is attempted with the true projection model, the SOP model allows for a solution to be found directly.

Unfortunately, this algorithm still has a time consuming calculation involving finding the pseudo inverse of matrices, but to its advantage, will only have to calculate this inverse once per solution. Also, there is no need for a initial guess to be made which was to the detriment of the other methods mentioned. This makes POSIT more robust with respect to estimating solutions, allowing for a certainty that if the estimation is bad, it was due to the matching process having failed.

However, does have one major drawback. The matches made in the previous section cannot be co-planar. If this is the case, estimation cannot take place as the pseudo inverse is impossible to calculate.

A minimum of four correspondences are needed to calculate pose. This is more than needed for a solution in other algorithms. The pose will always be calculated if the points in the object model are not coplanar. Where the matches are incorrect, the estimation resulting from POS will be nonsensical, and will deform the object model to force a fit. This will be used later when judging the validity of the match (in section 4.5)

Iterative POS

While POS has resulted in an estimation of the pose of the object model, these estimations are based on the scaled orthographic projection alone. To increase the accuracy of the match the results from the initial pose estimation are used to adjust the image point to be closer to the projection rays associated with that point. With each iteration, moving the points onto their lines of sight would cause the solution to tend to be that of a solution based on the actual perspective projection. This ensures the initial approximation made by using the Scaled Orthographic projection quickly becomes negligible. This algorithm has the advantage of only needing to calculate the SVD during calculation once for all iterations. Dementhon and Davis [1995] also compare their method to other pose estimation methods and conclude that is uses an order of magnitude less floating point operations and for good matches required on average only four or five iterations to converge on a solution.

4.4 Approach Taken

4.4.1 Object Model

The proposed technique for matching requires that the objects structure is known before any solution can be found. A representation for the storing of the structure of the objects known to the system is needed. Since the matching process will require vertices to match against and the edge detection process will facilitate the checking against edges later, wire frame models are ideal for storing objects.

Objects are defined as being made up of polygons, where the shape of each polygon is defined by a list of vertices. For use as described in section 4.5.2 later, the order of these vertices is crucial. If a face is viewed from a direction that will be visible (outside) it must be distinguishable from one that is invisible. Thus vertices are specified in an anti-clockwise direction. This convention is needed as the ordering is used later to calculate which side of a polygon is facing outward.

The size of the model needs to be relative to the size of the object in the real world. The pose estimation will include a scaling factor that will be used to scale the object to the appropriate size to overlay upon the image. If all the objects have sizes that are relative to actual measurements, then the scale will be representative of the distance of the object to the camera.

4.4.2 Design

Our design will be based on the POSIT style solution method. Its use is easily justified by its simple implementation and robustness. The other methods mentioned require an initial guess of the pose parameters, limiting the robustness as if this initial guess is not close enough (as with SCERPO it needed to be within 60 degrees) a solution will not converge. POSIT, to it's advantage requires no initial guess for estimation pose and translation. Also, the speed to estimate a solution will prove a great advantage, as our system will require many thousand estimations before any good solution is guaranteed.

In keeping with the suggested design of RANSAC algorithms[Fischler and Bolles, 1981], the minimum sample size required for an estimation of pose is used. For POSIT a minimum of four vertices is needed for a solution. Interestingly this allows for number of optimisations in the algorithm in the next section.

The pose solution will need to result in parameters that will be able to definitively place the object in space. A rotation matrix and a translation matrix will be calculated that are relative to the camera's position.

Point of Reference

For later use, we define an object center as a point of reference. This center need not be the actual centre of the object and could be any arbitrary point. We define the object centre as the first vertex in the set of matches. All rotations and translations that occur during and after the pose estimation will now use this point of reference.

Firstly the matches must be organised for use in computation. Matches are put into their respective matrices, the model matrix and the image matrices. The corresponding rows in these matrices are defined by the correspondences made in the previous section.

$$M = \begin{bmatrix} \vec{0}_1^t & 1 \\ \vec{m}_2 - \vec{m}_1{}^t & 1 \\ \vec{m}_3 - \vec{m}_1{}^t & 1 \\ \vec{m}_4 - \vec{m}_1{}^t & 1 \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

The original POS algorithm involved the calculation of the SVD [2] of the model matrix in calculation of the pseudo-inverse. Calculations of SVD's is implemented as an iterative progress which is computationally expensive. If the four coordinates of the model matrix are specified in homogeneous coordinates, it keeps the model matrix a square matrix, thus allowing the inverse of the model matrix to be directly calculated at minimal computational expense.

Rotation Matrix

The rotation matrix is of form:

$$R = \begin{bmatrix} r_1^t \\ r_2^t \\ r_3^t \end{bmatrix}$$

The rotation (r_i) vectors are calculated as with the POS method mentioned above. Firstly

[2]Singular Value Decomposition

intermediates matrices I and J are calculated (containing the x and y coordinates of the image points respectively) and normalised to find $r_1$and r_2. The factors required to normalise the two vectors are averaged to find the scale of the object.

$$I = M^{-1}x \qquad J = M^{-1}y$$

If the match is good, and the first two vectors of the rotation matrix $r_1$and $r_2$will be close to orthogonal. The third row vector is simply obtained by the cross product of the other vectors in this order:

$$r_3 = r_1 \times r_2$$

The resulting rotation matrix can be applied to the object to result in the final orientation. The next process is to find the location of the object. The location of the object can be represented by a single translation vector.

Estimating the Translation vector

The estimation of the translation vector is a relatively simple task from this point. The matrices are initially prepared in such a way that there is a point of reference defined about which rotations and translations are applied and this point is defined as one of the matches made between the object and image. Thus the translation vector needs to be one that would put the projection of the object in line with this match.

The translation vector is thus given by

$$T = \begin{bmatrix} u_1 \\ v_1 \\ focal\, length \end{bmatrix} / scale$$

this formula forces the first correspondence found by matching to line up after the projection has occurred. The focal length of the camera is measured and specified before the estimation of pose and serves as a relative measurement for distance. When the scale is bigger or smaller than one, the object is closer or further than the focal length of the camera respectively.

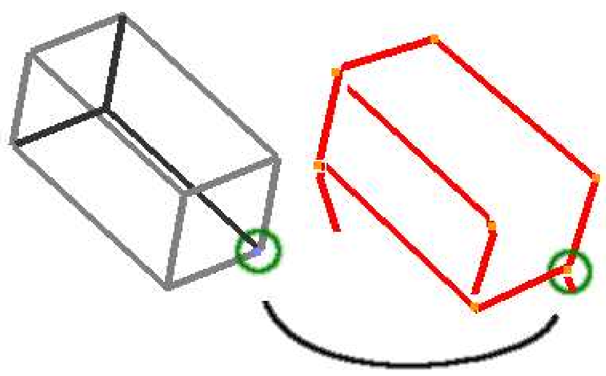


Figure 4.1: The point of reference is used to calculate the translation vector directly

4.4.2.1 The iterations of POSIT

The results documented by Dementhon and Davis [1995] suggest that if fewer are points used in the POSIT algorithm then fewer iterations are necessary. Since we are using the absolute minimum number of points, the solution derived here would not need the iteration steps at all. When only four points are involved, the initial calculation is reasonably accurate and for the purposes of our solution will provide a solid enough base for pose calculation. Thus only one iteration will be used to estimate the pose.

4.5 Judging the goodness of fit

Goodness of fit is an important concept in the pose estimation stage. Up to this point there is still no guarantee that any of the estimations are correct. Since there could be as many as a thousand incorrect matches to every correct one, many of the incorrect estimations need to be discarded, and ideally as quickly as possible, filtering out all but correct solutions. Thus there will be many stages to the judging process, moving from less time consuming checks initially to filter out the majority of incorrect matches, to the more computationally expensive checks later which would be more to compare correct solutions.

Checking the validity of the solution is done in multiple stages to eliminate as many incorrect matches as possible initially. Firstly the validity of the rotation matrix is checked. This is followed by the more computationally expensive stages that check that similarity of image and object geometries.

4.5.1 Validity of the Rotation Matrix

Since the estimation will assume that the matches are correct, calculating a pose with incorrect matches will result in a rotation matrix that is malformed and cause the object model to be sheared[3] in some way after rotation to force the erroneous correspondences. For this reason we can test that the rotation matrix is well formed.

The rows of a rotation matrix should be orthogonal[PlanetMath]. Since the third row of the matrix will be orthogonal to the other two, as it was derived from them, the first two will be used for this check. With a perfect solution, the dot product of the two orthogonal row vectors will be zero. For increasingly worse solution the values will tend to one.

When the dot product between r_1 and r_2 of the rotation matrix exceeded 0.05, that solution was discarded. In our test situations, this consistently discarded around 96% of the matches. This large percentage would allow only a few matches to pass through to the following more computationally expensive stages.

4.5.2 Forward facing edges

Knowing which vertices are forward facing (visible to the camera) is vital for being able to judge the whether the match made is a good one. All vertices in the images are by definition visible, while the not all the vertices in the object will be. A great deal of the matches made can immediately be discarded based on whether if one or more of the vertices matched, are not visible. This checking can be done now that the pose has been determined. Thus if any one image vertex has been matched with a vertex that is invisible on the object after is has been positioned, we can immediately assume the the solution is erroneous and discard it.

To determine which vertices on the object are visible, first each side of the object in model space must be separated in to two groups, visible and invisible faces. This is done by first calculating the normal vector of the object face. These normal vectors are calculated when the object is loaded to optimise computation. Three points on each face will be needed for the following formula:

$$\vec{n} = \frac{e_1 \times e_2}{|e_1|\,|e_2|}$$

[3]Shearing causes the object to be scaled in a non-uniform way, thus scaled unequally between the three axis's.

where $e_1$and e_2 are vectors representing any two edges on the face. Since all faces are represented by coplanar points, the face normal will be the same for any pair of edges on the object face. For the face to be forward facing, the following must be true:

$$(R\vec{n}) \bullet F < 0$$

Thus after the rotation has been applied to the normal vector, the dot product between the vector will be greater than zero, if the normal vector of the face is pointing away from the forward vector. The case that is dealt with here is where the forward vector $F$is simply facing away from the camera ($F = \begin{bmatrix} 0 & 0 & -1 & 1 \end{bmatrix}^T$) This formula above thus calculates the faces that will face forward after the object is rotated and projected onto the image.

Each of the faces is tested. Every vertex that is a member of a forward facing face is set to visible. Matches made as in chapter 3 are then checked, if any of them are matches to invisible faces, no further checking is needed and that match is discarded. Typically of the poses checked, approximately 85% were discarded. This result was consistent between the different types of object tested.

4.5.2.1 Extending the matches

Now that the solution is more likely to be correct, more computationally intense methods can be used for verification. Up to this point only 4 correspondences have been made between the vertices of the object and image. The more vertices that correspond the better the match, or more specifically the more visible vertices matched. Object vertices need to be projected onto the image so that the distances can be measured.

A new matrix is defined for these purposes, the projection matrix P here is based on the approximation documented earlier (scaled orthographic projection.) $O_i$represents the location of an arbitrary object vertex.

$$P = \begin{bmatrix} scale & 0 & 0 & 0 \\ 0 & scale & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad O_i = \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

Each of the visible object vertices that are not already matched are then transformed through the formula:

$$O_i^{'} = PTR(O_i - O_c)$$

$O_c$is the point of reference that was defined earlier in the pose solution stage. $O_i^{'}$is the point's location in image coordinates.

Each of the available image points is not compared to the location of the projected object vertices. If their distance is less that a given threshold (for our experiments 10 pixels) a new correspondence is made. With this, an important statistic now arises, we can now calculate the ratio of the number of correspondences to the total number of visible vertices on an object. This will be the first measure of the goodness of the fit.

Any object that fails to extend the number of matches beyond the original four made in the matching process is unlikely to be a correct match.

$$Visible\,Vertex\,Ratio = \frac{Matched\,Vertices}{Visible\,Vertices}$$

4.5.2.2 Edge Correspondences

Another method that will be used to judge the goodness of fit will be to compare the edge data. Each vertex will be compared to every other vertex to check whether or not they are connected. This will serve as a check that the image and object have the same geometry. The following statistic will be used as a judge of the similarity of geometry

$$Edge\,Ratio = \frac{Corresponding\,Edges}{Total\,Edges}$$

This however doesn't include edges that exist in the image and not in the object model. However, since image data might contain edges from shadows and other erroneous edges due to noise, the omission of detection of these inconsistencies increases the robustness of the detection. For example, over extensions as mentioned in Section 2.4.5 will be ignored.

4.5.2.3 Overall quality of pose estimation

To be able to compare estimation, it would be useful to define a statistic that would represent the overall quality of the pose estimation. Based on this statistic, solutions could be classified as good or bad fits. We will define it as the $EdgeRatio * VertexRatio$. This will allow for a single method of determining whether the pose solution was successful.

Also, it is possible that the different sets of vertices matched in the first stage would result in solutions that are similar. Close solutions like this will have the same vertex matches that could be identical after the extension of matches, which thus, would result in a identical goodness ratio. In these cases a secondary comparison of the total distance, between the projected vertices and the image vertices is used to judge the best fit.

4.5.3 Classification of objects

Usually, before any solution or matching can occur the object model that will be fitted to the data has to be selected. This decision is known as the classification problem. With the RANSAC approach the problem of choosing which objects to match against a given set of data is solved indirectly after calculation of the pose[Fischler and Bolles, 1981]. The POS style solution method will be able to match any data, regardless of whether it is the result of an incorrect match or not.

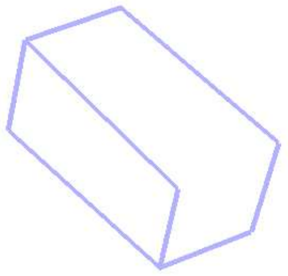
Attempts are made to match all of the object models available to the image data and each are treated with equal likelihood of occurring. For an object that don't exist in the image data, very poor results will result from the goodness of fit testing stage. Thus objects that don't exist will be filtered out. The most plausible (with the highest goodness ratio) is chosen and the system would both classify which object exists and which match of that object is the best fit in this manner.

4.6 Results

4.6.1 Goodness level

Choosing a level for discarding of solutions is vitally important. Setting the threshold too high will in result matches never being found due to the occurrences of errors in the image data. Lower values allow matches to occur for more corrupted image data, however, low values also allow for the wrong object model to be matched to the image data. For example in figure 4.2 the incorrect model is matched to the image data despite that a rectangle was one of the objects attempted. Essentially the vertices are matched correctly, and reasonably accurately, however a low threshold in this case has allowed an entire section on the object to be mismatched. The root of the problem lies in the fact that of all the vertices matched, the edges corresponded between the image data and the object data and this had the effect of elevating the goodness ratio. Even though the ratio of possible vertex matches is low, it wasn't low enough to reject the match.

Also, the goodness level needs to be high enough to handle the case where no objects are present in the image. In this case the goodness level cutoff needs to be high enough to exclude


(a) Geometry of image data for a rectangle

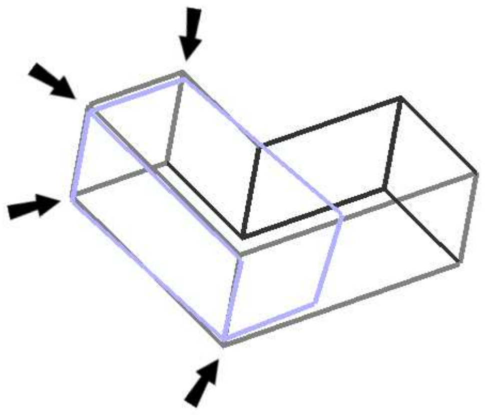

(b) Only four vertices matched

Figure 4.2: Failed solution due to low goodness level

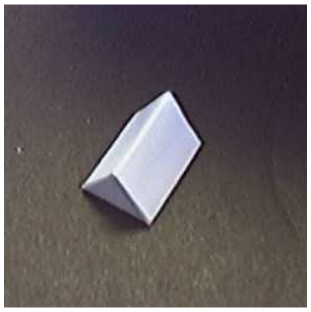
all the estimations made.

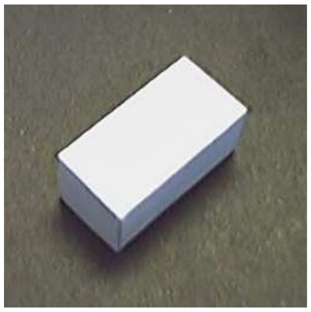
4.6.2 Effects of missing edges and missing vertices

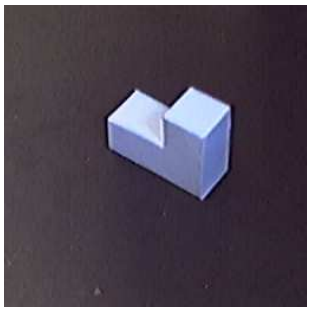
Often due to losses in the feature extraction stage, some vertices and edges are not extracted from the image all. However, for the objects tested as many as two or three edges could be lost before the solutions were lost. The reason for not finding any solutions was due the setting of a threshold for the goodness values. This is perhaps a flaw in the filtering process as solution are still likely to be found, but now just considered insignificant due to the cutoff. A balance had to be found between finding objects from incomplete data, and finding objects from incorrect data.

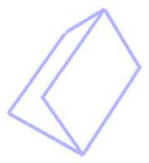
Due to the nature of the matching process, there could be many possible false estimations with average goodness of fit values. Rather than risk incorrect pose estimations passing through, we leave the threshold at a higher value to make objects, while slightly less commonly detected, more accurately detected. A good level from tests seems to be around 0.65.

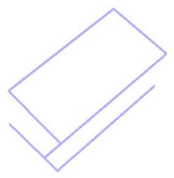
The effects of having a missing edge is minimal with reasonably complete image data due to the way the goodness level is used to judge the estimations. On average, for the test images used, the goodness values were around 85% for correct matches. Coupled with the cutoff being around 0.65, allowed for objects with quite some error due to missing data to still have their poses considered significant.

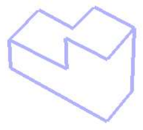

(a) Prism

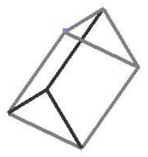

(b) Rectangle

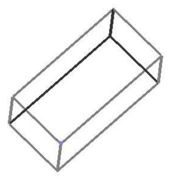

(c) L Shape


(d) Geometry


(e) Geometry


(f) Geometry


(g) Solution with goodness of 83


(h) Solution with goodness of 67

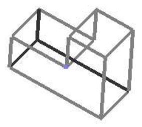

(i) Solution with goodness of 0.76

Figure 4.3: Images with missing data and their goodness values

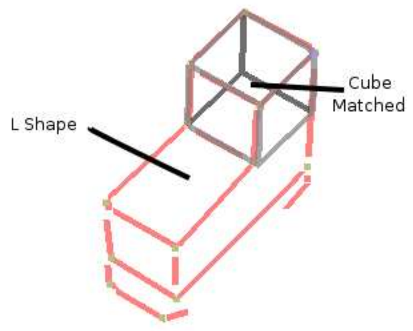


Figure 4.4: Ambiguous Situation

4.6.3 Similar Geometries

Sometimes, there exist situations that result in false matches which seem perfect solutions. The geometry of a larger object may contain subsections that are similar to the geometry of smaller objects. For example, in figure 4.4, the cube object was matched to part of the l-shape. In theory the cube should have never have been a better fit that the L shape. Here the goodness value of fit was well in excess of the threshold, thus was not only fitted, but judged as the best fit, over the l-shape. The reason for the judging of the cube over the correct object was due to a partially incorrect extraction of the image geometry from the image. This kind of error is unavoidable and in this case resulted in a incorrect detection.

4.6.4 Typical Pose results

The results from the tests described in 2.4.1 showed positive results overall. From the test suite of images it showed a success rate of 85% over the 40 images tested with both classifying the correct object and making an estimation of the pose of the object.

Of the results that failed, 2 were due to degenerate case extreme enough to make the POS solution impossible (only coplanar points were present as in figure 4.5a) The other failures were where incorrect models had been matched to the image data. One such case was where the prism was correctly detected in the image, but the rectangle was matched to the image data instead. A better goodness value was calculated for the incorrect model, which resulted in the that model being detected as in figure 4.5b.

The pose estimation attempts to place the vertices of the objects as close to the image vertices

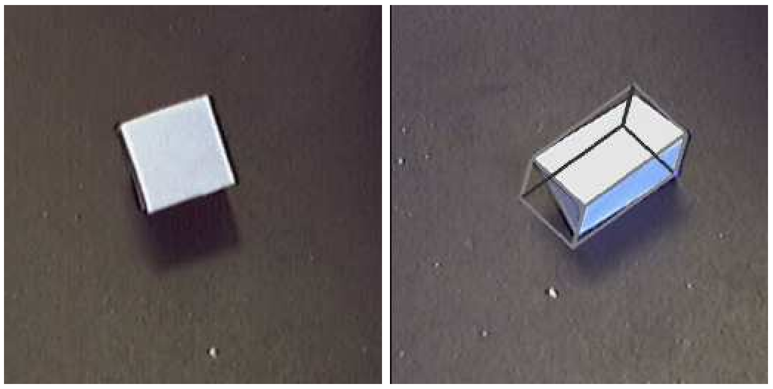

(a) Degenerate pose where only one face visible (b) Incorrect Object Matched

Figure 4.5: Failures in pose estimation

as possible. The data in the following table suggests that it was successful in general.

Object Model	Distance between image vertex matched object vertex
Cube	1.7 pixels
Rectangle	2.2 pixels
Prism	2.3 pixels
“L” Shape	1.3 pixels

These values seem perfectly respectable considering the scale of the objects in the image is around 4 pixels to every 1 millimeter and thus on average estimated vertices where less that a millimeter off. Also considering that the smallest edge on any of the objects is 10 millimeters long it is less than a 8% displacement relative to the shortest edge.

Of the test results, for all of the matches that were successfully made, the goodness values averaged around 84%. This would indicate that the threshold for excluding bad fits was easily low enough to allow the majority of good matches to pass as good solutions.

4.7 Conclusion

A successful means of calculating the pose and location of an object has been covered in this chapter. Methods of judging the correctness of the matches passed into the pose solution phase have been created. The judging of the goodness of the fit has been found successful for both

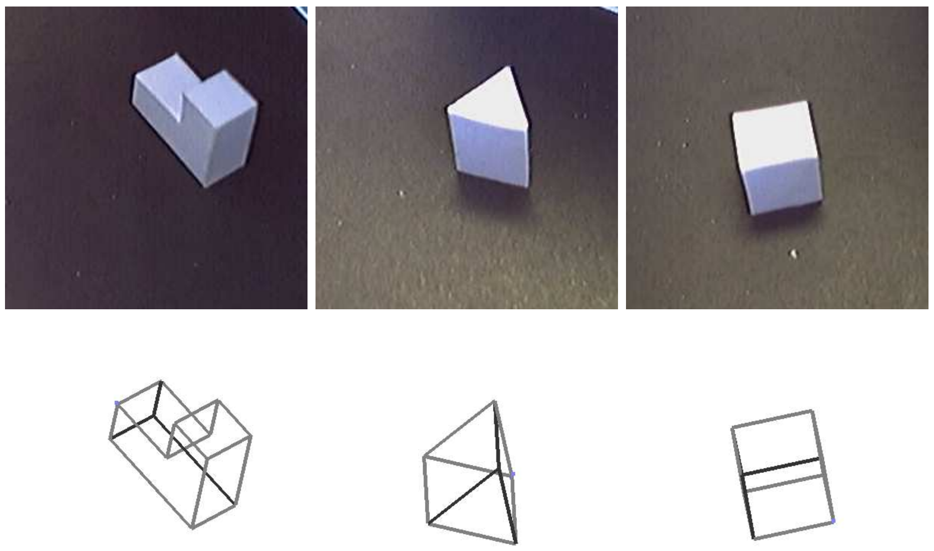


Figure 4.6: Examples of correct pose estimation

classifying which objects are present in the image and for judging the best fit made from the matches for each object. The results from this chapter will now be extended to allow for the detection of multiple objects in the next chapter.

Chapter 5

Multiple Objects

Now that a system has been developed that can estimate the pose of a single object in an image, it could be extended to function for multiple objects in a single image. Only some situations with multiple objects are expected to be handled correctly. It is expected that images with multiple spatially separated objects would function as well as for the single object case, while where objects are obscured, the pose estimation might not be as effective.

An algorithm for matching of multiple objects is covered and the motivation of using this method is discussed.

5.1 Algorithm

The previous pose estimation stage attempted to match all the object models to the image data and would result in one object being found. For every type of object in the image, there is a possible estimated of pose for that object. There is, unfortunately, at this stage no guarantee that these solutions are independent. Some overlap of the estimations of objects may occur where vertices are shared (multiple objects over the same image data.) This is obviously undesirable as no two objects should in theory be able to share a vertex. In practice this may differ as spatially the vertices may be separate, but in the image they may appear to be the same. Generally it would still be desirable to keep the detection independent. For this reason only the best scoring solution from the pose estimation section will be taken and the rest discarded. This way independence of the solutions is ensured.

A side effect of the increasing amount of image data is that for each object, more random matches will need do be made (as mentioned in section 3.2.2.) This will ensure that the same level of robustness is maintained while more vertices are present in the image data.

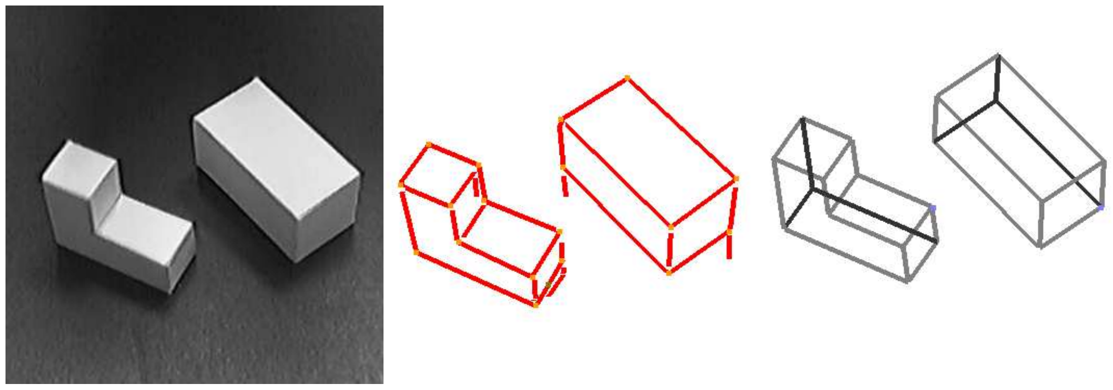


Figure 5.1: Multiple objects separated allowing good performance of the recognition system

The vertices that were matched to the detected object are now marked as matched and are excluded from further use. This process is necessary as each iteration of the detection will search for all the object models once again, thus without exclusion of vertices would continue to find the same object over and over again. Also by removing the vertices from consideration in the image data, the search space for subsequent objects is greatly reduced with each object found.

This process is repeated while objects are still being successfully detected. When there are no objects with solutions that pass the goodness test mentioned in4.5, the process is stopped and it is assumed that all the objects have been found. Since after each iteration the vertices used in a solution are being marked as matched, when fewer than four vertices are available (the number required to estimate a solution) the process is also halted.

5.2 Results

Results for multiple objects were in most cases as expected. Where the objects were separated by a small amount, the recognition happened essentially as for the one object because there was no interference in these cases. Each of the objects were recognised one at a time (figure 5.1.)

However, for more complex cases, the recognition process failed regularly, in fact, the vast majority of the time. Thus there was an immediate indication that the failure was in the design of the system, and it would not be able to recognise obscured objects with any level of success. No further tests were carried out for these cases. One of the best solutions seen is illustrated in figure 5.2.

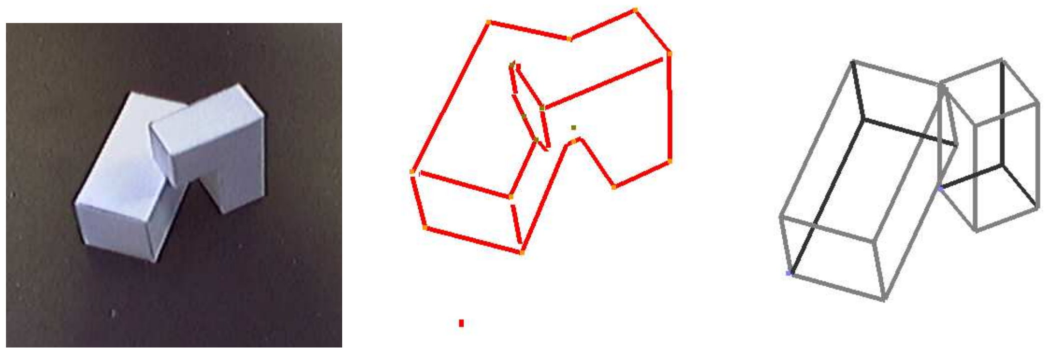

(a) Obscured objects (b) Image features (c) Incorrect Detection

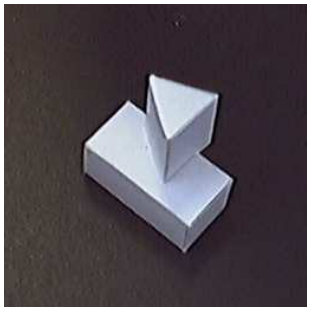
Figure 5.2: Example of incorrect detection of multiple obscured objects

5.2.1 Proximity of objects

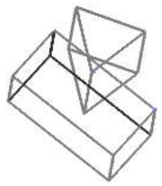
When objects are positioned too close to each other in images, the image data might be extracted in a way that the objects appear to be attached. This might occur particularly when corners are in close proximity and vertex clustering causes the vertices to be combined. This in fact was not the case as once the vertex clustering tolerance was set lower (to it's current value), errors resulted from the edge detection stage before the vertices could be clustered, and as a result the objects were not successfully identified due to the missing data, and not due to clustering of vertices.

5.2.2 Obscured objects

Objects that were obscured could only work in a few choice cases. The image data is extracted in such a way that obscured objects tend to result in the image data being interpreted as if there is just one more complex object (combination of two objects.) This is because there is no real grouping or way of distinguishing between the features extracted. Thus when data is matched, the first object matched will often claim some data that would be needed for matching of both objects. Detecting the second object would often yield unexpected results because certain parts of the object would be missing as other objects used the vertices. If the image data was retested for the objects on their own, they would be far more likely to be correctly detected properly.



(a) Object placed as to cause the loss of only one line



(b) Correct pose estimation

Figure 5.3: Multiple objects obscured but situation engineered to result correctly (uncommon)

5.3 Conclusion

The performance of recognition with multiple spatially separate objects was much the same as for the single image case. More complicated situations involving occlusion however proved in general, impossible for the system correctly recognise. Thus the issues with recognition of multiple occluded objects resulted from the feature extraction stage covered in chapter 2. The system could reliably identify visually separate objects, but would fail for more complex cases where the objects appeared obscured in images.

Chapter 6

Conclusion

In this thesis a system for the identification of known objects from within an image was presented. The system was done in various stages. First the image was prepared for analysis using a Gaussian filter to remove excess noise, followed by the use of the Canny edge detector to extract edge information. The edges were then converted into a set of straight line segments fitted via regression to the edge information. These lines were extended to find vertices, that would represent corners of objects. A matching process based on the RANSAC model fitting paradigm was used, randomly associating image vertices with that of the predefined objects. Later after the pose was estimated using the a method based on the POSIT technique, and the associations from the matching stage were judged for validity and the best fitting object was selected. Finally the extension of this method for use with multiple objects in the image was considered.

6.1 Implications of this work

cost

6.2 Summary

The systems developed was able to detect single objects within images with 85% success. Given that missing data was present most of the time (68% in our test data) it proved that the system was also quite robust with regard to missing data. Thus we can conclude that a successful model-based vision system has been implemented for the single object case.

However, certain cases are still present where the object are guaranteed to be undetectable. An inherent flaw in the POSIT solution causes detection to fail in degenerate cases when the object is perfectly detected but all of the points are coplanar. Cases in which only one face is visible for example. Fortunately this is rare. From the current results, is can be seen that the approximations that were made by using the weak projection model (SOP model) were also justified as the errors generated from pose estimation with this model are minimal and thus acceptable.

For multiple objects, the detection of visually separate objects in a scene occurred just as successfully as for the one object case. And thus for these cases the system was as robust and the overall recognition was successful. However in more complicated cases, such as visual occlusion, the system failed, mainly due to there not being a method of distinguishing between the image data of different objects.

The vision system as a whole was successful and for spatially separate objects was robust and could detect a range of objects composed of flat surfaces.

6.3 Extensions

The system presented in this paper is a simple and robust one, that could be extended in many ways. A few likely extensions are covered here.

6.3.1 Background Subtraction

Often in a large complex scene there are small regions of interest. To be able to reduce the amount of image data to be analysed to just there regions would be of a great advantage as all the image features outside of these regions could be omitted from detection and reduce the amount of time required to find a match. One such method is using background subtraction to identify these regions.

By using a image of a scene taken before the objects are present and subtracting it from a scene where the objects are present, one could determine the areas that have had significant change. Thus only these regions could be considered.

6.3.2 Real World Coordinates

The estimations here are all with respect to the position of the camera. The information could possibly be more useful if it were transformed in such a way that the data could be located not with respect to the camera, but perhaps relative to some object in view of the camera such as a plane on a desk, thus giving the pose and location a real world significance.

6.3.3 Use of full POSIT

Currently our pose solution method only uses an optimised version of POSIT only capable of handling four correspondences. A step could be added which uses the full POSIT method to further optimise the solution using the extra matches found in in section 4.5.2.1. This would increase the accuracy of the estimation.

6.3.4 Curved surfaces

Currently, only objects with planar surfaces can be detected. Unfortunately the system has been designed is a way that would need a large portion of it to be redesigned to be able to robustly detect objects with curved surfaces. A new type of system would have to be invented to detect curved surfaces and the detection of curved surfaces would be out of the scope an extension to this project.

Bibliography

J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–698, November 1986.

H. Cantzler. Random sample consensus. *On-Line Compendium of Computer Vision*.

D. Dementhon and L. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

Lijun Ding and Ardeshir Goshtasby. On the canny edge detector. *The Journal of the Pattern Recognition Society*, 34:712–725, 2001.

Matrin A Fischler and Robert C Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communication of the ACM*, 24:123–141, 1981.

Mike Heath, Sudeep Sarkar, Thomas Sanocki, and Kevin Bowyer. Comparison of edge detectors: A methodology and initial study. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 143+, 1996.

Z.J. Hou and G.W. Wei. A new approach to edge detection. *The Journal of the Pattern Recognition Society*, 35:1559–1570, 2002.

David G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31(3):355–395, 1987.

Siu-Hang Or, W. S. Luk, K. H. Wong, and Irwin King. An efficient iterative pose estimation algorithm. In *ACCV (2)*, pages 559–566, 1998. URL `citeseer.ist.psu.edu/article/or97efficient.html`.

PlanetMath. Rotation matrix. URL `http://planetmath.org/encyclopedia/RotationMatrix.html`.

Michael Seul, Lawrence O'Gorman, and Michael J. Sammon. *Practical Algorithms for Image Analysis*. Cambridge University Press, 2000.

Minal Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis and Machine Vision (Second Edition)*. Thomson Learning Vocational, 1999.

Emanluele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall; 1st edition (March 6, 1998).