An Investigation of a Remote Visual Navigation System for a Building Inspection Robot

Alastair Mark Paterson
BSc. MSc.

This thesis is submitted for the Degree of Doctor of Philosophy

City University

Department of Computer Science
School of Informatics

January 1996

Contents

Appendices

References

Acknowledgements

The Lord GOD Almighty		Help and inspiration. For giving me the strength and ability to pursue this project.
Tony Bonomini	City University	General support.
Denis Chamberlain	City University	Organising the research work, rescuing me from unemployment, supervising the work, proof reading.
Julian Chua	City University	Help with Windows programming.
Tim Clark	City University	Help with image processing.
Mike Cooper	City University	Information about photogrammetry.
Geoff Dowling	City University	Supervising my work, finding good conferences, proof reading, suggestions (the polite ones!).
Tim Ellis	City University	Lectures on image processing, Canny algorithms
EPSRC (formerly SERC)		Funding and grant.
Guy Hammersley	Laing Technology Group	Contact with commercial world, inspection information, financial support via CASE award.
LTG Library Staff	Laing Technology Group	Help in locating building inspection information.
Alex Murray	Programmer	Introduction to Windows programming.
John Reilly	City University	Keeping the PC hardware working and listening to all my development stories!
Clive Thomas	St. Marys Hospital	Introduction to C programming.
Mr A. Vinton	Northwick Park Hospital	Allowing a visit to inspect their buildings.
Ian Zejma	Laing Technology Group	Information about building surveys.

and

My Parents for keeping me fed and watered!

Declaration

"I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement".

Abstract

The work presented here shows the development of a machine vision algorithm for finding the position of a building inspection robot on the outside of a large building. The reasons for external building inspection are introduced along with the types of tests used. Existing methods are examined giving their limitations in terms of practicality and safety and an alternative using remote access is proposed. The work concentrates on the navigational aspects and shows how one possible solution using machine vision could be implemented and this is compared to similar work carried out elsewhere.

The major part of the thesis covers the development of the robot location algorithm starting with the fundamentals of image processing and finishing with the actual robot's position. Different methods of edge detection are investigated and a pixel linking routine is used to group together data in an image that form features and principal lines. The algorithm investigates the use of the lines for detecting vanishing points and tries to identify the features highlighted in the image. The most significant part of the work concentrates on the development of a method of identifying specific features such as a target on the robot and different windows along with a way of matching the features to a computer model of the building thus enabling the position of the robot to be calculated.

Results are given showing how the algorithm performed on a model building and robot in the laboratory with various tests using different camera positions, image enhancement and spurious features. The results presented show that the algorithm was capable of finding the position of a model robot to sufficient accuracy (typically 3% of the size of the robot target) and that the errors measured were predictable. Additional results show how the algorithm performed on a real building and indicate the problems associated with real images with the conclusion that the algorithm will work under a certain range of conditions providing that certain elements of it can be improved.

Glossary

Bit	Short for 'Binary Digit'. Has the value 0 or 1.
Byte	Unit of computer memory - equal to one character, has 8 bits.
CAD	Computer Aided Design.
CCD	Charge Coupled Device - essentially electronic film.
Centroid	The point about which the object would spin if all its pixels had the same mass. The object's centre of gravity.
CURIO	City University Robot for Inspection Operations.
DXF	Data eXchange Format file often used transferring CAD drawings.
Feature	A collection of objects grouped in some way, for example, by a surrounding object.
FSM	Finite State Machine.
GPS	Global Positioning System.
LTG	Laing Technology Group Ltd.
MMI	Man-Machine Interface.
Normal	A line normal to a plane has an angle between it and the plane of 90^{o}.
Object	A closed group of pixels e.g. an ellipse or rectangle.
Orthogonal	At right angles.
PC	Personal Computer - IBM and IBM compatible.
Pixel	Short for 'Picture Element', an image is made up of pixels.
Scan Line	An image line with a constant y value.
VP	Vanishing Point.
Windows 3.1	A common and popular operating system for PCs.
X (direction)	The horizontal position in an image.
Y (direction)	The vertical position in an image.

1

Introduction

A team at the Construction Robotics Unit at City University, London is developing an inspection robot called CURIO (Bleakley and Chamberlain, 1994; Chamberlain and Bleakley, 1994) to be used externally on tall structures such as tower blocks or storage tanks. This incorporates a number of different disciplines, one being robot navigation. The position of the robot is vital to the recording of defects and for searching for particular inspection sites. There are a number of ways the navigation could be performed and it was decided to see whether a machine vision system would be suitable for this task. Having a real commercial application, machine vision offers a potential solution using cheap, readily available components and a high 'ease of use' factor requiring a minimum amount of effort in setting up. This thesis presents the first attempt at using a vision system and shows that this method of navigation has potential.

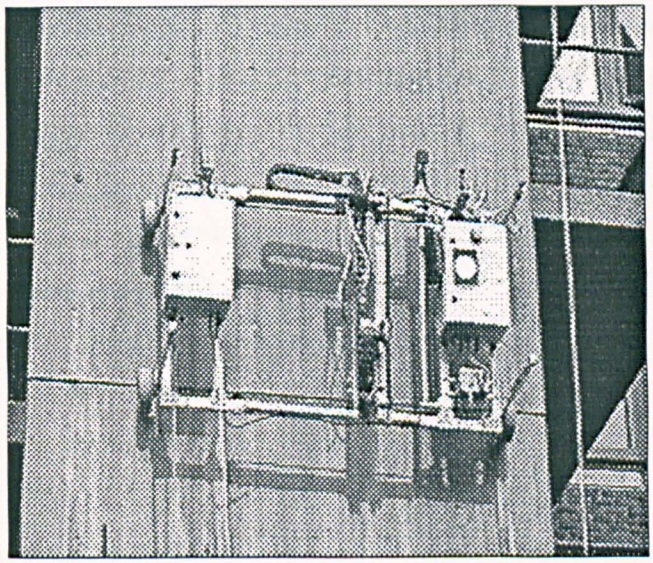


Figure 1.1 *CURIO (prototype) being tested on a wall at City University*

For this specific application, there has been no previous work found in the literature, and as it is only three years old at the time of writing, everything presented here represents the first investigations. The only source of related work must come from similar projects done by others and this can be broadly split into two main categories. Firstly there is the field of robot navigation, with an emphasis on machine vision and inspection; secondly there are general image processing techniques that are already in existence.

Automatic Inspection Robots

Virtually all references to other inspection robots concentrate either on the ability to perform a specific inspection, or the mobility of the robot. The positioning of the robot appears to be a subject always left for the future and remains largely unaddressed. Perhaps the most similar application to CURIO is presented by Cusack and Thomas, (1992) which aims "to travel quickly to the points requiring inspection...It is anticipated that in the future this system will be interfaced to a CAD system for autonomous guidance." This is the only work found to date where navigation for a building inspection robot is explicitly mentioned although no details of 'how' are given. Their results show the position of a crack relative to the robot. Similarly, Rösch and Schaab, (1995) look for incorrectly placed dowels in carriageway slabs using a covermeter and a frame type robot very similar to CURIO. The dowels are measured only relative to the robot frame, presumably someone has previously calculated and recorded the position of the robot. In another building application, Ashikawa et al, (1992) developed a robot for removing coatings from exterior walls. They were only interested in travel speed, position was not mentioned but appears to be done manually by the operator. Tillotson, Snaith and Tachsti, (1993) have modified a Landrover to carry inspection equipment that looks for defects in roads. Again, nothing is mentioned about actual defect position but in this case, the vehicle has a driver and approximate position could be found using the trip meter. Similarly, other inspection methods may concentrate on the mechanics of the test as demonstrated by Sarr, (1992) who produced a hand held machine for accurately measuring scratches in aircraft skins using computer vision in conjunction with a laser. Results given show how well the scratches were found and measured, but does not mention how the operator records the location of the defect. A somewhat simpler approach is taken by Mangold, Friedmann and Rammelkamp, (1995) who use infrared thermography to locate "pins" (cladding fixing dowels ?) in tower blocks. Images are taken and then processed in the laboratory and location work appears to be done by eye, however, this deviates from the application developed in this thesis where images are to be processed on site. Many other robots are primarily designed to research mobility, for example, Lee et al, (1994) look at the ability to move over an uneven terrain but do not mention how the robot will know where it is. Other robots which are designed to work on buildings such as the ones developed by Portech, (1995) concentrate again on mobility rather than position although a window cleaning robot developed by OCS Group Ltd., (1995) can be positioned to "within a few millimetres" using guide rails on the building. One area of robotics that is developing the use of computer vision is that of road following vehicles although this tends to be mainly for guidance and object avoidance rather than absolute

positioning. Haifeng and Mäkelä, (1991), for example, use the detection of edges in an image to look for kerbs whereas Campbell and Thomas, (1992) look for trapezium shapes in the image, since a road viewed from a vehicle becomes narrower with distance. Distance can also be obtained by using different cameras as demonstrated by Hock, Behringer and Thomanek, (1994) who use two of them set at different focal lengths for short and long range measurements.

Building Interior Robots

Further and perhaps more relevant work can be found from robots that work in the interior of buildings. These have to move around a three dimensional world rather than the two dimensional face of a building, but they still have to find their position. Probably the closest project to CURIO as far as visual navigation is concerned, is a robot for testing air conditioning vents developed by Fukuda et al, (1993) and Abe et al, (1994). This uses a camera to take an image of the ceiling and looks for the rectangular and circular features that make up the vents. The vents are quite distinctive and knowing their positions and using trigonometry, the position of the robot can be found. A similar project by Dulimatra and Jain, (1994) uses ceiling lights and the numbers on doors to navigate by. These robots do however, require that the features are specific and clear - almost as if they were targets. If a building has no clear distinguishing features, then it will be necessary to find an alternative navigation method. A commonly used approach makes use of depth information; that is, by looking around and finding the distance to objects such as walls, a three dimensional picture is built up giving the robot its local position. This is an area of research carried out by the Oxford University Robotics Group which is specialising in a high performance binocular head/eye platform. In particular, Beardsley et al, (1994) are using an uncalibrated active stereo vision system to navigate an autonomous vehicle and they use detected and matched corners to provide 3D point information. Also, Ferarri et al, (1991) use stereo vision, as in nature, to measure distance. This is extended further by Weckesser, Gastinel and Dillmann, (1995) who use trinocular vision with calibrated cameras They are also able to detect image edges in real time and use parallel image processing to create a powerful navigation system. The robot works from a floor plan of the building and has a detailed knowledge of certain landmarks such as pillars. Multicamera vision systems tend to be rather complex and a simpler method of obtaining distance is to employ laser range finding and triangulation, as demonstrated by Tanaka et al, (1995) who use it to find the position of a robot relative to pillars. If movement without position information is required, then a mobile robot developed by Yagi, Kawato and Tsuji, (1994) will be of interest as it has a novel way of looking at the

scene with a vision system that captures images using a conical mirror. Object avoidance is performed by observing features that move towards the centre of the resultant image. Finally, Salagnac and Vinot, (1991) review specifications for construction site robots and they propose that vision-based positioning systems use building components such as floor tiles or vents which may be easily recognised.

Photogrammetry

So far, all the navigation methods described above use a positioning system located on the robot/vehicle. What if the navigation system was remote from the robot such that the robot was told its location? A class of surveying known as photogrammetry uses imaging techniques to measure objects and locate points with unknown positions. However, it tends to concentrate on producing an accurate 3D representation of a given object which could range from an industrial component to the Earth itself, rather than giving general positions of an object. The precision is obtained by the use of at least two images in the case of stereo vision and the cameras have to be accurately positioned and calibrated so that the true world co-ordinates of a point found in the images can be calculated. Csáki, (1990), for example, shows a typical application where stereo vision and surveying techniques are used to draw and measure building data, in this case archaeological sites. Quite often targets, which are easy to see in images, are placed over the object to be measured, but in the context of buildings, many of the features themselves can be used instead. Benning and Schwermann, (1995) investigate the use of straight lines rather than points to obtain orientation information. This is particularly useful on buildings where lines are easy to find, especially around windows, but their example uses three images in which the corresponding lines have to be found and matched. A novel approach taken by Streilein, (1995) was to use a single, moveable S-VHS camcorder to record and subsequently capture a large number of images. These images were processed later to reproduce the building, but the camera must be calibrated. Good 3D reconstruction of the scene is shown but it requires extensive processing.

Image Processing - Edges

It has been shown above that there are a number of methods that have been used for navigation and locating robots. Most of these robots and vehicles use various forms of image processing to extract information from an image. A number of commonly used techniques which will be incorporated in one form or an other into the inspection robot location software are now reviewed. There are many books that cover the basics of image processing at a general level such as Gonzalez and Woods, (1993) or Sonka, Hlavec and

Boyle, (1994). These show methods which are likely to be used for most image processing applications, but once a specific task is required then specialised techniques have to be used which may be very application specific and cannot be covered by textbooks. What also tends to be apparent in the image processing literature is that there is a considerable amount of work done on specific techniques, but very few applications are mentioned at all. A case in point is the use of methods of finding edges in images. This is usually a vital step in image processing and, apart from the commonly used algorithms given in text books, much research is conducted into this area. Park, Nam and Park, (1994) compare a number of different edge detection methods and propose their own one, which is as good as the Canny edge detector (discussed later) and performs better on diagonal edges. Sarkar and Boyer, (1991) have also proposed their own detector which is claimed to work better than Canny. Their method works well in noisy images and can be "readily adapted to real-time hardware implementation". One of the problems encountered with other edge detectors is that they may produce good results but the implementation is very complex, thus making them less attractive at an early development stage. Higgins and Hsu, (1994) also compare different detectors but propose one that has many parameters. The difficulty then becomes how to chose what value the parameters should take or how to change them for different images. A couple of other edge detectors developed by Tewfik and Deriche, (1993) and Qian and Titterington, (1993) are also very mathematical in nature although the latter performs well on textured surfaces, which may have an application in looking for decorative features on a building. Alternative approaches to edge detection which attempt to learn the nature of edges, just as humans would, are also being developed by Bhandarkar, Zhang and Potter, (1994) who use genetic algorithms to cope with different amounts of noise in an image. Also by attempting to mimic the behaviour of the brain, Kendall and Hall, (1992) look at applying neural networks to image processing functions, such as edge detection and texture classification and reckon that their Quantised Neural Networks have "great potential".

Image Processing - Features

Assuming that one method or another has successfully picked out the edges of interest, it then becomes necessary to try to identify the shapes and features that correspond to the edges. This is another area of active research similar to that in edge detection, where work is concentrated on a specific function rather than an application; again, some approaches are highly complex and would take a considerable amount of time to implement if adopted. A fairly frequently found topic is the identification of basic shapes such as lines,

corners, etc. Xin, Lim and Hong, (1994) use a description of an edge boundary to identify features which are made up of a number of corners, line ends, arcs and lines; however their example only uses a simple object: a pair of pliers. Similarly, Cooper, (1993) looks at hypothetical objects and examines the relationships between the arcs and curves that make up the objects. This could have an application in recognising more complex features on a building. Others just concentrate on searching for lines as these are fairly prominent in man-made structures. Biao, (1994) looks for lines in eight different directions to try to identify buildings from aerial images. Kahn, Kitchen and Riseman, (1990) on the other hand look at speeding up line detection in order to guide a robot down a path and suggest that their method could be implemented in hardware which would greatly speed up processing. Corners are significant in perception and there are several examples of corner detectors such as Cooper, Venkatesh and Kitchen, (1993) who obtain good results by first looking at the direction of the edges and then identify real corners based on the noise levels in the image. Rosenthaler et al, (1992) use orientation energy from directional filters to look for line intersections and actually give an example showing the corners of windows and panels in an image of a building. Giraudon and Deriche, (1991) present another example of a method that uses many parameters which all have to be set for a certain image, but they can find corners in noisy images. An alternative to corner detection is to look for 'common junction types' where lines join such as 'L, T, K and +' junctions, as shown by Lee, Pong, Slagle and Esterline, (1994). By trying to mimic human perception, Fischler and Wolf, (1994) have developed an algorithm that looks for key points on a curve, the points being very similar to those chosen by humans and could therefore be used to pick out the vertices of features. At a higher level, attempts are made to try and fit specific shapes to objects in an image. This has an advantage if objects are partially obscured, as is often the case, in that it may then be possible to predict where the whole of the feature lies. However, it is important to look at general solutions unless there is a specific application in mind. Kumar, Raganathan and Goldgof, (1994) look for circles in images using parallel methods, but in reality circles are seldom seen as orientation always makes them appear as an ellipse. A more realistic approach is taken by Wong and Kittler, (1993), who attempt to recognise objects such as blocks, pyramids and 'roofs' in a single image. It may be possible to use this if the majority of a building is visible so that its outline can be found. If features are not clear, then it may be possible to use an 'active contour'. An active contour can be used to find the border of a complex shape when there is no well defined edge. 'Snakes' can take two basic forms such as those which surround a feature and move in towards it, as

used by Leymarie and Levine, (1993) to find the outsides of a single living cell in an image. The other type is a 'balloon' which starts inside a feature and moves outwards to find its inner boundary, as demonstrated by Cohen and Cohen, (1993) to find the space occupied by the left ventricle of the heart. Both of these methods may have applications in finding either the outer boundary of a window or the individual window panes, but are mathematically and computationally complex.

Finally, after examining all the related work, the question can be asked: Is there a development system in existence which contains some of the algorithms mentioned above? A number of different image processing development systems are available for different platforms and a well known one, 'Khoros', is used at City University. A good review of it is given by Konstantinides and Rasure, (1994) where they discuss its advantages for image processing development, but is it needed for the development of the CURIO navigation algorithms? Since the proposed work is intended to form part of a much larger and real project it is necessary to develop the routines independently, thus enabling them to be integrated into the final application. A development platform is useful for trying out a few ideas but even if they are found to be useful, the routines would still have to be rewritten.

1.1 Aims and Intentions

The overall aim of this project is to obtain the position of the robot on the building. This requires a number of smaller goals to be achieved, as follows:

1. To investigate the use of cheap, readily available components to produce a simple system for the operator.

2. To investigate the use of a single gray level image from a single camera (no stereo vision) so that data processing and image sizes are kept to a minimum.

3. To see if it is possible to eliminate the need for the operator to measure the position of equipment when setting up.

4. To investigate the use of an uncalibrated camera, that is, there is no prior knowledge known about the properties of the camera's optical system.

5. To see if it is possible to eliminate the use of navigation targets placed on the building. Only the robot may have a target placed on it in a permanent, known location.

6. To be able to uniquely identify the robot target and to locate some reference point on it.

7. To be able to recognise significant building features such as windows from a CAD (Computer Aided Design) description.

8. To identify the specific position of the features and hence define the pose of the building, making use of vanishing points for oblique camera angles.

9. To obtain the position of the robot to the required degree of accuracy - this being to say that it is at a given floor and near a particular feature.

The intention is to develop an algorithm capable of capturing images from a video camera, processing the image and displaying the real co-ordinates of a point on the robot target relative to the bottom left corner of the face of a building being inspected. Initial work is to be carried out using a model building and robot to develop the methods and ideas to see if this form of navigation is feasible. Once the model robot can be identified and its position calculated, it is intended to test the system on a real building assuming that the real robot is available. If not, the system will try to identify the building from its features. The accuracy of the algorithm will be found by fixing the model robot on the model building and physically measuring the position of the centre of the target. A number of images will be taken with the camera in different positions to see how the calculated position of the robot compares to the measured position. Also the robustness of the algorithm will be tested by using these images and by changing the visible features.

1.2 Thesis Overview

The work presented in this thesis begins by looking at how buildings are currently inspected and demonstrates that present practices require improvement. A method of automation using a robot is proposed and chapter 3 discusses how it may be possible to find the position of the robot on a building. The actual work researched and developed is covered in chapter 4 which begins by looking at how features may be recognised and then gives the overall algorithm for robot location. An introduction to image processing is

given and the remainder of the chapter deals with the processing of image data to produce features and how they are matched to a diagram of the building. Chapter 4 concludes by showing how the position of the robot may be found using a model building and robot. Chapter 5 details the experiments and results obtained to see how well the algorithm functioned on the model when different camera angles and image processing techniques were used. This gives an indication of the limits which an operator would have to observe when setting up. Further improvements and suggestions for future development are given in chapter 6 should this project be continued. The conclusions are given in chapter 7 and show that the aims have been achieved but further research is required to produce a more reliable algorithm.

2

Current Inspection Methods and the Need for Improvement

There are a number of well known types of defect that commonly occur in buildings, all of which require detection and monitoring. The defects may occur as a result of a number of different factors such as poor design and construction, substandard materials and external influences such as acid rain, earth movement and even bomb damage. There are several inspection methods which are documented in many publications, for example CEB, (1989) and Mallett, (1994); they can be divided into three broad categories: surface, subsurface and visual inspection. Surface inspection requires equipment to be placed in contact with the building and is used, for example, to measure the condition of reinforcement bars. Subsurface testing often requires the removal of material for subsequent testing such as finding the salt content of concrete. Visual inspection is usually the simplest and performed first of all. Cracks, rust stains etc. can be easily seen and usually indicate that there is some underlying problem where further inspection is required. Figures 2.1 and 2.2 show a rust stain defect with spalling and the possible deterioration that results if the problem remains unchecked. In common with all these inspections is the need for access.

Figure 2.1 *Rust Stain and the Start of Spalling at City University*

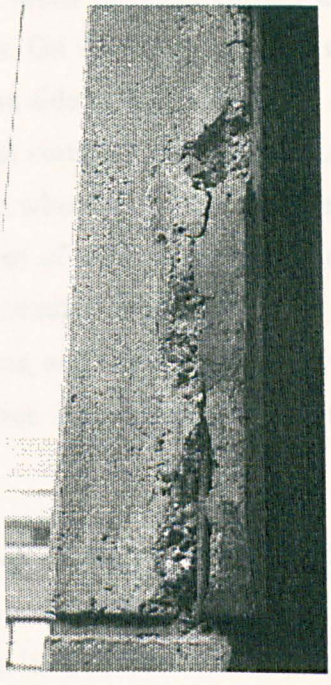


Figure 2.2 *Serious Concrete Failure at City University*

Figure 2.3 *Concrete Cover Inspection by Abseiling. Photo: Laing Technology Group*

Current methods require one or more inspectors to be present on the building face. Sometimes it is possible to use existing access equipment such as window cleaning platforms, but often the inspector has to abseil to the desired location, as shown in Figure 2.3 where a typical inspection using a cover meter is being carried out. This introduces a number of safety and accuracy problems. On the safety side, there are the physical dangers of being located some distance up the side of the building. A failure of the rope, arresting equipment or harness would almost certainly guarantee death. Sadly, a greater risk can come from the residents themselves, whom the inspectors are trying to protect. There are several reports of people leaning out of windows trying to cut ropes, pouring fat out of the windows and the dropping of needles and razor blades on to the inspectors! Safety is becoming a bigger and bigger issue, and with the increase in liability and the soaring cost of legal action, fewer companies are prepared to use their own personnel in these situations.

The other major problem encountered is the accuracy of the results, particularly with visual inspection. When a defect such as a crack is located, a sketch is made of that part of the building and the defect is drawn on it. This is later marked on a drawing of the building and possibly copied into a report. Numerous errors can occur here. Firstly, when one is very close to the building, it is difficult to see where one is and so it is easy, for example, to lose count of the floor or to make a reference against the wrong window.

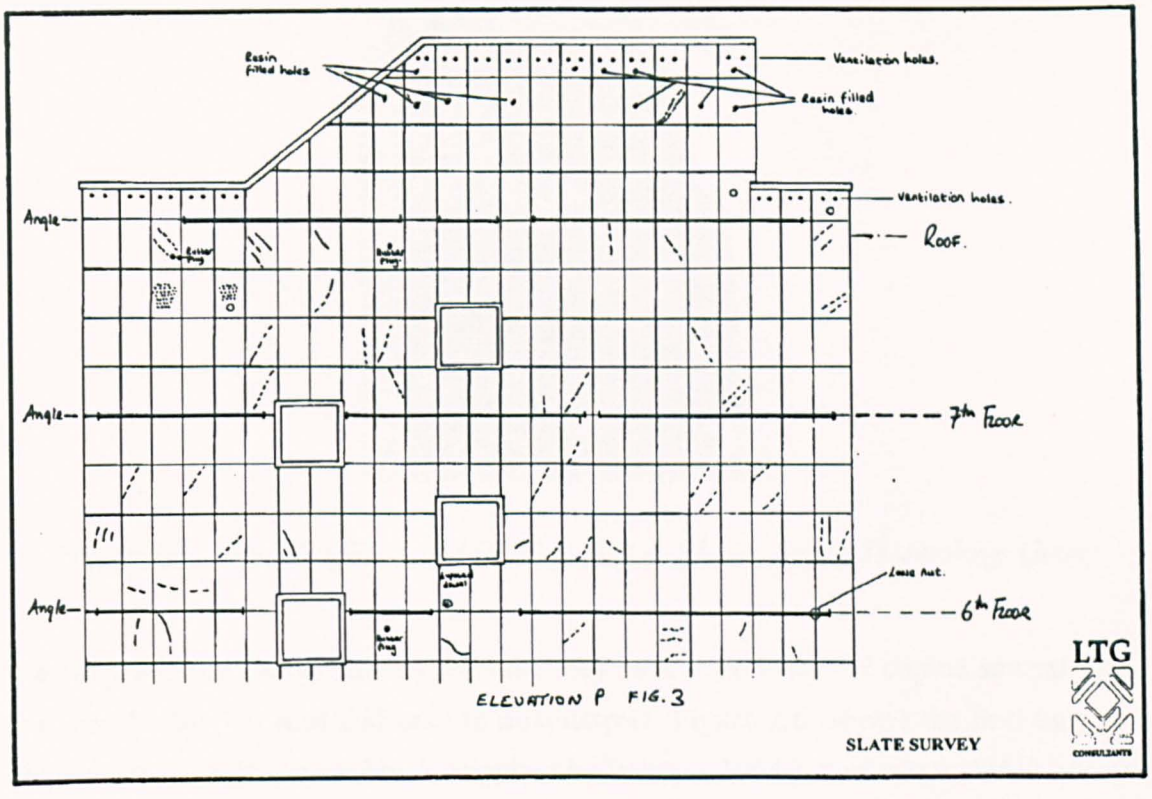


Figure 2.4 *Example of Building Survey Report. Diagram: Laing Technology Group*

Secondly, a sketch is made of the defect risking further loss in accuracy. Finally, the sketch is copied on to a proper drawing resulting in a further loss in accuracy, along with the possibility of drawing it in the wrong place. Coupled with the ease of miscounting identical features, the result the customer sees may be rather different from what is actually present, making re-examination more difficult. This also has the added risk that a serious fault could be overlooked. An example of a real inspection report is given in Figure 2.4 which represents the part of a building shown in Figure 2.5. In this instance, the side of the building was made up of a large number of slates which needed to be inspected. No drawings were available so a sketch was made and drawn up later using pen and paper. The defects were also added later from notes and photographs taken at the time of the inspection. Here, the accuracy of the defects themselves was not too important but what is very clear, is that there is a large potential for miscounting the slates and marking a defect in the wrong place because the slates, and each of the floors, were very repetitive in appearance.

What is needed is a way of producing results more accurately and safely. A significant improvement is to use computer aided design (CAD) diagrams as these can be printed new each time they are required, or can be called up on a computer display. Many existing

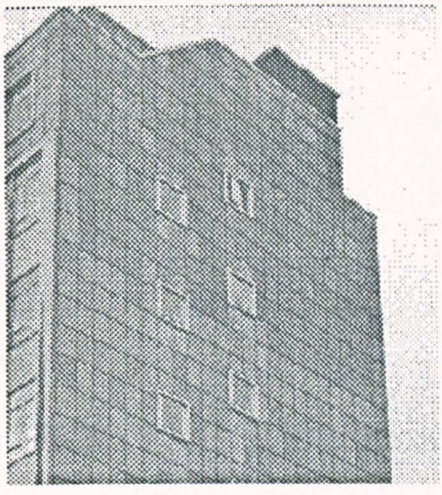


Figure 2.5 *Building Surveyed for Figure 2.4. Photo: Laing Technology Group*

drawings are hand drawn and by the time they have been used and copied several times they can be hard to read and easy to misinterpret. Figure 2.6 shows the best part of a diagram of a London tower block, supplied by Wong, (1994). Fortunately CAD systems are readily available and most, if not all, new buildings are designed on computer. Also old hand drawings may be digitised to produce as good as new diagrams, as demonstrated by Sivaloganathan, Jebb and Wynn, (1991).

With computer diagrams available the major improvement in inspection is to replace the inspector by a machine or robot. The implications of this may sound bad, particularly at a time of high unemployment, but the aim is to change the role of the inspector to inspection robot operator. An inspection robot would effectively perform remote sensing and leave the inspector in a safe place. With the safety problems mentioned earlier, it is far better for the robot to become damaged than for a human to be injured or killed. The main safety risk in using a robot is it falling off the building, but simple procedures can greatly minimise the risk of injury to third parties. A robot also has the advantage that it can work 24 hours a day without having to rest. True, an operator needs to be present but not all the time. However, the main advantages of using a robot from a practical point of view is that it can minimise errors and produce more accurate results. Cameras and computers can keep track of the robot's position so that it knows precisely where it is, thus minimising confusion in defect location. Results can be taken to a better accuracy, particularly those arising from visual inspection. Calibrated equipment can be used to take consistent readings and the use of computers can speed up the data processing. These data can then be collated automatically and placed directly into reports without the need of intermediate manual work.

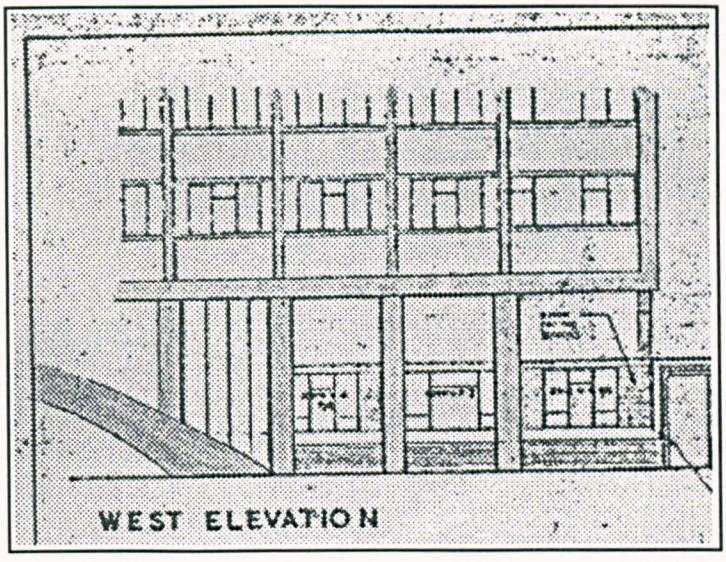


Figure 2.6 *An Example of the Most Legible Part of a Building Drawing*

Summarising, it is seen that there is a considerable need for building inspection which would greatly benefit from automation; an inspection robot would be the ideal solution for this task.

2.1 Proposed Semiautomatic Visual Inspection

As mentioned earlier, one of the first inspections to be carried out is the visual inspection. This is also one of the simplest mechanically and would make an ideal task for the robot. The first mode of operation is to scan the whole building surface looking for specific defects. A local camera on the robot is used to spot the faults with either an operator viewing the camera output and deciding whether a defect is present, or on board image processing used to automatically find a defect. Crack detection, for example, is a subject researched by several groups using different techniques such as Doihara et al, (1992), Bryson et al, (1994), Miura et al, (1991) and Song, Petrou and Kittler, (1992) and would be used to produce data directly. With an operator, a simpler short term approach would be for the operator to trace out the defect with a mouse or light pen. The results in either case are stored in a data base and then can be drawn directly on to the CAD diagram.

The advantage of having this computerised data means that the second mode of operation of the robot can be employed, that is, to return to the defect area or site of interest. This allows repeat measurements to be taken, say at yearly intervals, and these can be compared to previous readings. The greater accuracy leads to better comparisons which give an indication as to the state of the defect, for example, is it getting worse?

Both these modes of operation, the searching for and then returning to a specific place require one thing to be known in order to work - the location of the robot. Without knowing where the robot is, it is not possible to record the position of the defect and it would not be possible to return to it later. This then, gives the basis of the research presented here; how is the location of the robot determined?

3

Robot Location and Navigation

In order for an inspection robot to be of any use, it is necessary to know its location so that defects can be accurately recorded. Two methods of operation are used, the first one being that the robot moves over the building looking for a specific defect and the second one being that the robot manoeuvres itself to a specific place where a test is to be performed. In either case, there are several ways of finding its position, all with advantages and disadvantages. Since this is an industrial application, priorities must be placed on cost and safety for the market place to be interested in the product. The capital cost of equipment can be reduced by using existing technologies available 'off the shelf' and time costs are reduced by using equipment that is quick and easy to set up. With this in mind, there are three main possibilities.

Firstly, sensors can be placed on the robot winch mechanisms to measure how far it has travelled from a given point. Although this may be easy to develop, it requires considerable setting up since the robot is designed to work on many buildings using a variety of access equipment. Each one would have to be individually set up and calibrated, taking time. However, there are devices that contain a wire, similar in principle to a tape measure, which can be connected to some fixed location such that as the robot moves, the wire is drawn from the device which then gives a measure of distance travelled. Whereas this may be suitable for vertical measurement, problems would occur with horizontal movement as the wire would sag under its own weight and would possibly drag over other objects.

Secondly, is the use of the Global Positioning System (GPS). A number of Earth orbiting satellites transmit signals that are picked up by a receiver either on the ground or on the robot. Knowing the positions of the satellites gives the location of the receiver and hence the robot. Edmundson and Novak, (1992) have developed a highway data recording system mounted in a van that integrates GPS, an inertial system and stereo vision to keep track of the van with the aim that if the GPS is unable to obtain a position, the other two systems can. Varying degrees of accuracy are achieved from a few tens of meters to a few centimetres. The latter would be required for the robot but with precision comes a very high price tag. Although this is dropping all the time, GPS equipment available from outdoor activity shops sell at around £500 and some can be interfaced to computers for a further £100. These sets are at the cheap end of the range and although no resolution is

given in the brochures, the accuracy would be within the tens of meters range - suitable for walking or sailing activities. The higher accuracy systems as used by the military would probably cost many times this. Also, there may be physical problems in receiving satellite signals very close to a building as the building will obscure a significant part of the sky, thus blocking out the satellites. A different form of GPS, developed by Leica, uses a laser and a 'wand' that can detect position from a distance of up to 100m. There may still be problems with objects obscuring the path of the laser however.

The final solution is to literally 'look' to see where the robot is. A vision system is used to view the building and find the robot giving its location. This has the advantage of using affordable, readily available equipment and is very easy to set up. It is this last solution that is researched here to find out what would be required, and how realistic an approach it is.

3.1 Location Theory

The location of the robot is carried out in two stages. The first one, and the one which this work concentrates on, is the coarse positioning, that is, identifying at which floor the robot is, or between which two windows it is located. The second stage is to refine the position given that it is near some feature identified in the first stage. A number of different approaches have been taken to finding position using vision, with a popular one being stereo vision. This might seem an obvious choice since nature has developed this extremely well in the form of predatory animals, including humans, who employ stereo vision to judge distance. If the distance to objects in a scene is known, then a considerable amount of unwanted data can be discarded, allowing processing to concentrate on regions of interest. An overview of the tasks required, showing the complexity involved is given by Nishihara and Poggio, (1984) and above all, the 'fusion' of the left and right images is a complex task requiring much processing. This is an active area of machine vision where there is a considerable research effort, for example, Yau and Wong, (1994). Hellwich and Faig, (1994) attempt to match stereo images using the prominent edges in the two images to form a graph-based map of neighbouring edges. This is mathematically intensive but does produce a result, as demonstrated by locating the welded seams on oil tanks. Another approach to finding distance is taken by Ens and Lawrence, (1993) who use just a single camera to obtain depth information from the focus settings of the lens. This may, however, have problems if the camera is located some distance away (as with a building) where objects at different distances will all be in focus. A simpler alternative

might be to use a single camera and a laser. By shining a laser on to the building and if the camera and laser orientations and positions are known, then the distance to the laser spot can be found giving the same range information as stereo vision. However, a practical problem is encountered. In order for the camera to detect a spot of laser light some distance away on the building, the laser would have to be quite powerful and problems of safety would need to be addressed; a powerful laser shining through someone's window could be a serious health risk! Photogrammetry makes use of the range information to provide very precise co-ordinates of features allowing the details of, say, a building or the ground, to be accurately recorded. For example, Brown, (1994) gives an example where 215 targets on the ground are used in conjunction with two sets of 26 photographs to obtain centimetre lateral accuracy or 1 part in 500 000 of the width of the photographic field. Specialist cameras are also employed which naturally increase the cost. This level of detail is not, however, required here and the precise positioning of cameras required would add considerably to the setting up time. Also, the nature of the problem here virtually eliminates the need for stereo vision and/or photogrammetric techniques as two dimensional planes only are involved, that is, the side of the building, where simpler methods can be used.

3.1.1 Coarse Positioning

The method developed in this application for the coarse positioning of the robot, is based on how humans see objects in a picture. If a black and white photograph showing the building with the robot on it is given and a diagram of the face of the building, then without much difficulty, the position of the robot on the diagram can be marked fairly accurately. In keeping with the ease of use strategy, this method requires a minimal amount of setting up. A single video camera is placed in a position where it can view the building and is connected to a computer; a typical set-up is shown in Figure 3.1. No calibration of the camera is required, no accurate placement of the camera is necessary, no measurements are needed and no complex stereo vision or photogrammetry need to be performed. In the proposed method, where the tasks are to be as simple as possible, the robot has a target permanently fixed on it making it easily recognisable. A description of the building (CAD diagram) is loaded into the computer by the operator who merely has to align the camera and sets the software running. It could be possible for the operator to 'click' a mouse, or some other pointing device, on easily identifiable points in the image to help the processing, such as the visible corners of the building but this detracts from the aim of the research to produce an automatic system.

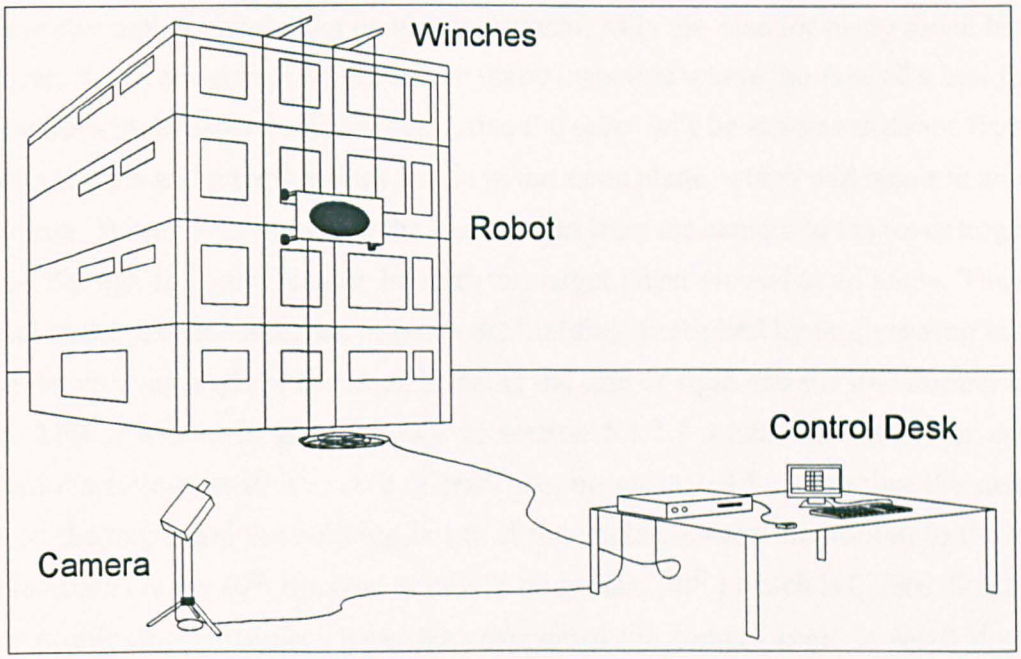


Figure 3.1 *Typical Building Inspection Robot Set-Up*

The result of computer architecture is that it can calculate a more precise position than a human can and it is necessary to find a mapping between the image and building model or CAD diagram. To map the image to the diagram requires the 'Projective Transformation' (Thompson, 1966: 805-807) and is given below.

$$X = \frac{b_{11}x + b_{12}y + b_{13}}{b_{31}x + b_{32}y + 1} \qquad Y = \frac{b_{21}x + b_{22}y + b_{23}}{b_{31}x + b_{32}y + 1} \qquad [\,3.1\,]$$

Interestingly, Georgopoulus and Tournas, (1994) use the same equations for 'digital rectification' and they also suggest that lines and circles can be used as well as the points here. If the eight mapping parameters $\mathbf{b}_{ij}$ are known then any point $(\mathbf{x},\mathbf{y})$ in the image can be mapped onto a point $(\mathbf{X},\mathbf{Y})$ on the CAD diagram. If this point happens to be some point on the robot, then its true location is immediately known. In order to evaluate the eight parameters, four corresponding points from the image and CAD diagram are required and it is this process that forms a bulk of the work. The overall process to find the robot's true position is:

1. Find four points in the image and their corresponding points in the CAD diagram.
2. Calculate the eight mapping parameters.
3. Identify some point on the robot in the image.
4. Using equation 3.1, calculate the real position of the robot.

This transformation only works on planar surfaces, as is the case for many tower blocks. However, this is not always so. There are many instances where the face of a building is not smooth and deviates from the ideal. Also the robot will be at some distance from the building surface and therefore does not lie in the same plane, which will result in an error in position. This error is caused by the line of sight from the camera to the robot target not passing through the point directly beneath the target when viewed at an angle. The error is equal to the distance from the target to the building multiplied by tan(viewing angle), where the viewing angle is the angle between the line of sight and the line normal to the target. This is shown in greater detail in section 5.1.1.1 where the effects of camera position are investigated. The size of error can be estimated by assuming the distance between the target and the building is 1m. If the angle between the normal to the target and the camera is say 30^{o}, then the error will be 1 x tan(30^{o}) which is 0.58m. Since only coarse positioning is required here, the error when the camera angle is small does not matter although it can be calculated for larger angles by using a suitable target. As long it is known at which feature or floor the robot itself is located, confusion is avoided and a correct position can be found from fine positioning.

3.1.2 Fine Positioning

This section has been added for completeness only, as the fine positioning of the robot is beyond the scope of this work. Assuming the robot has stopped at some defect, its coarse position will have been calculated as above. An onboard camera, which is also used for visual inspection, will be able to view a limited part of the building surface including part of a recognised feature. Since this camera is part of the robot and does not require setting up by the operator, it can be used as a pre-calibrated measuring device and therefore its orientation to a specific feature can be found. With this knowledge and the knowledge of the feature, the robot's or the defect's position can be found accurately relative to the feature. It is not necessary to know the precise position from some far off point, for example, the bottom left corner of the face of the building, especially as buildings are not constructed with a high level of precision. Coarse tolerance, construction methods and actual movement of the building could change measurements by several mm per floor for example, which would then accumulate over the building. By only finding the position accurately relative to a known feature, errors arising through dimensional tolerances can be avoided. It should be noted that the onboard camera cannot be used for the coarse positioning as it is too close to the building to see all the relevant features. A realistic robot camera would be about half to one meter away from the building.

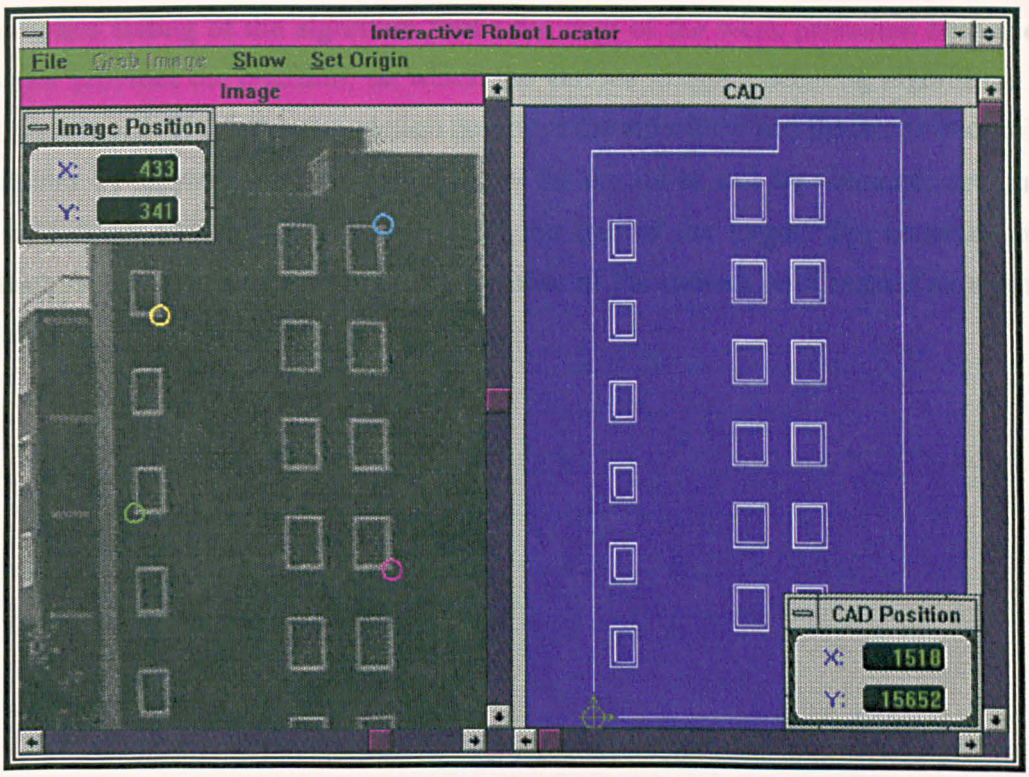


Figure 3.2 *Display of the ROBOLOC Program*

3.2 Manual Location Program

As this work is part of a much larger project, a program, ROBOLOC, has been developed to locate the robot 'manually'. This is a quick development solution that allows the operator to select the four sets of points and to pick out the robot. The software then calculates the image to CAD mapping and gives the robot's true location as described above. A sample screen showing the program is given in Figure 3.2 where a captured image is displayed along with the CAD model. Although this might provide a suitable development program, it is still prone to human error, such as selecting the wrong point or not accurately placing a point, which the automatic version is designed to avoid.

3.3 Automatic Robot Location

The algorithm being researched is coded in the program, ROBONAV, which is designed to automatically find the robot on the building without human interaction during the image processing, although it could be used in conjunction with the manual program to select the building corners if ambiguities arise. It essentially performs the same task as the manual program but lets the computer select the four most suitable control points. The

writing and testing of the algorithm forms the core of the work presented here and the program provides the platform, on which the various image processing algorithms have been developed. The following section describes the algorithm and functions in detail and concludes by showing that a good position for the robot can be obtained, accurate to within a small percentage, around 3%, of the size of the target. The accuracy of the position does depend, however, on the position of the camera with respect to angle and distance.

4

The Automatic Robot Location Algorithm

Now that the location principles have been defined, it is necessary to break down the task into a number of specific steps and look at just how the grabbed camera image is going to be matched to the CAD model along with identifying the robot. For the robot, there is control over how it appears and a simple target placed on it is used to help find the robot. The buildings on the other hand, pose a difficult problem since their many different appearances and locations are not designed for recognition by image processing. This section presents the method used to identify the building and explains the different steps and algorithms employed.

4.1 The Building Recognition Principle

The principle used to recognise a building is based on how we as humans might perceive a scene. If a black and white photograph of a building is shown along with a diagram of that building, then it is fairly easy to identify and match the various features in it, such as the windows. This can be done without the use of any measurements or any knowledge of the camera optics (lens focal length for example). Willson, (1994) demonstrates that a zoom lens, which is the most beneficial in a vision system, can be calibrated for any lens setting, but this requires many measurements from the lens before it can be used. Clearly it is an advantage if this can be avoided. It is then a reasonable assumption to make, that if a human can perform this simple match, then it ought to be possible for a machine to accomplish the same task. The problem comes in trying to identify just what is it that the human is doing when attempting to match the picture to the diagram.

As a scene is viewed, people do not keep their eyes fixed in one place and the entire scene is not perceived at once even though the complete image is present. In fact the eyes are constantly wandering about, latching onto small details here and there, which are then combined to give a perception of the scene. Applying this to tower blocks, especially when seen at close range, the eyes tend to scan round the different windows or other prominent features and if the building was being compared to a diagram, these features would be scrutinised even more closely. Before it is possible to find which window is which, it is necessary to identify the individual types of window (or feature). Again,

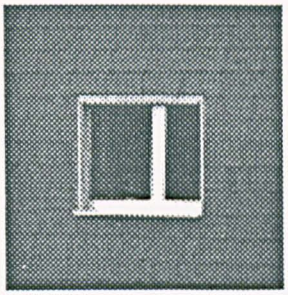
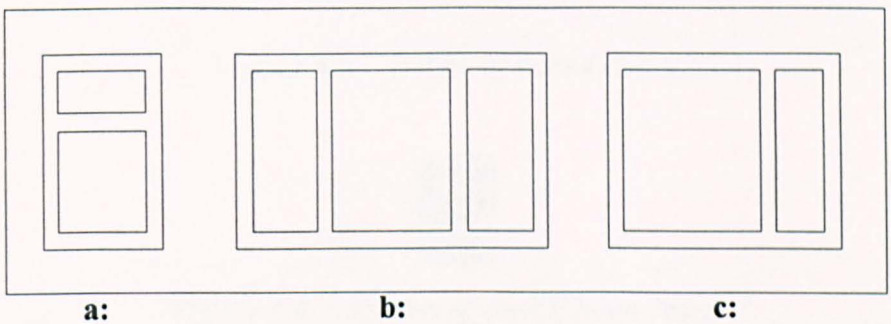


Figure 4.1 *Image of a Window*



Figures 4.2a - c *Diagrams of Windows*

studying how the human manages this can suggest ways in which a computer could perform the same task. By referring to Figures 4.2a through 4.2c, try to find the diagram which represents the actual window shown in Figure 4.1. Without much effort it is clearly Figure 4.2c. However, this problem was very easy to solve, as it could be done automatically without thinking. Further insight can be gained when the features are not quite so obvious, for example, when all the features are similar. If the image is now changed so that the aim is to take the small group of baked beans in Figure 4.4 and find them in the general picture of Figure 4.3, then it is not automatic and the human mind tends to employ a search algorithm. Assuming that individual beans have been identified, it is noticed that the target group is made up of three beans. However, in the large picture there are many combinations of three beans so numbers alone cannot identify the group. It is the sizes, orientations and relative positions that identify the group and allows them to be found. The particular group here is characterised by two beans being vertically orientated (with reference to their major axes) and next to each other with the third being horizontal and above them.

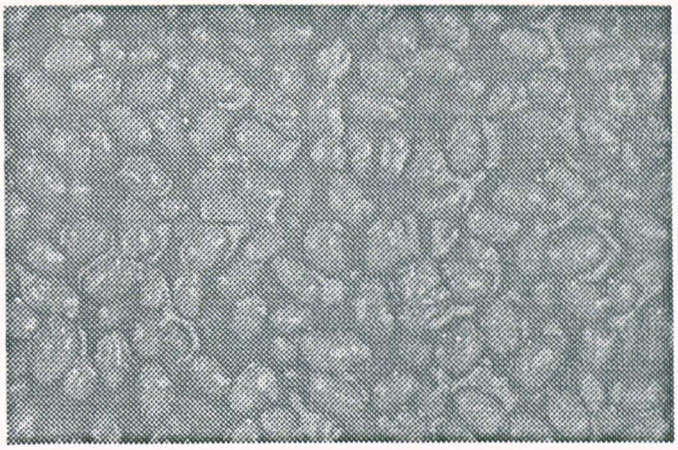


Figure 4.3 *Image of Baked Beans*

Figure 4.4 *Section of Baked Bean Image*

It is this topographical description that is used to identify a building; that is, using descriptions such as 'left of', 'above' or 'inside' for example, rather than providing precise mathematical measurements. Using the bean analogy above, a window (the small group of beans) is made up of a number of panes (the individual beans). As well as position information (left of), relative area in the sense of pane 1 is twice as large as pane 2 is used along with relative distances. A more complete description of two panes might be: Pane 1 is twice as large as- and three pane 1 widths to the left of pane 2. Similar work done by Biederman et al, (1993) develop an idea called Geon Theory and they also suggest that the mind works in this manner. Bergevin and Levine, (1993) also attempt to recognise an object from 2D images by using the geon approach to build up relationships between lines and arcs which make up different possible cross sections and then use these to build up parts of features. A significant advantage of this topographical approach is that it is fairly invariant when the image is distorted by perspective. Two identical windows must be identified as such even if one is much further away than the other. By using only the neighbouring panes of a window, the effects of distortion are greatly reduced. Although identical panes may be of a considerably different size in the image, by moving from one to the other via neighbouring panes, the local variations can be eliminated and the overall image distortion will have no effect, allowing the windows to be correctly identified. Similarly the effects of camera lens distortion can be virtually eliminated. An advantage

of being qualitative is that where maths requires precise equations and numbers giving exact answers, a descriptive approach means that small deviations and errors do not change the final result. This method has a hierarchical nature so where the discussion has so far concentrated on identifying windows from their panes, the same method can be applied to identifying a building from its windows (or other features). The CAD diagram of the building face being inspected is used to generate a map of features again with the descriptions of neighbouring feature positions, for example: Window 1 has a neighbour window 2 above it, a neighbour window 5 to the right of it and no neighbours to the left or below it. A separate list is generated of the individual features describing how they are constructed from shape primitives such as rectangles and circles etc. and is used to compare with shapes found in the image.

4.1.1 Outline of Overall Algorithm

The previous section has introduced the theory of the building recognition and chapter 3 has described how, given a number of known points, an image can be mapped to the CAD diagram. This section looks at the steps necessary to perform this mapping and they can be summarised as follows:

```
For each new building face, do:
    Load CAD model and generate mapping lists.
    For each new robot or camera position, do:
        Grab and pre-process the image.
        Reduce the image to shape boundaries and
        classify them.
        Identify the robot target and remove it
        from the image.
        Identify feature types in the image.
        Map the features to their specific CAD
        counterparts.
        Choose a set of control points and
        calculate the image to CAD mapping.
        Apply mapping function to robot target to
        obtain its true location.
```

These steps are shown in greater detail in Figure 4.5 which represents the main functions of the software that have been developed. The remainder of this chapter looks at the individual functions and how they were implemented.

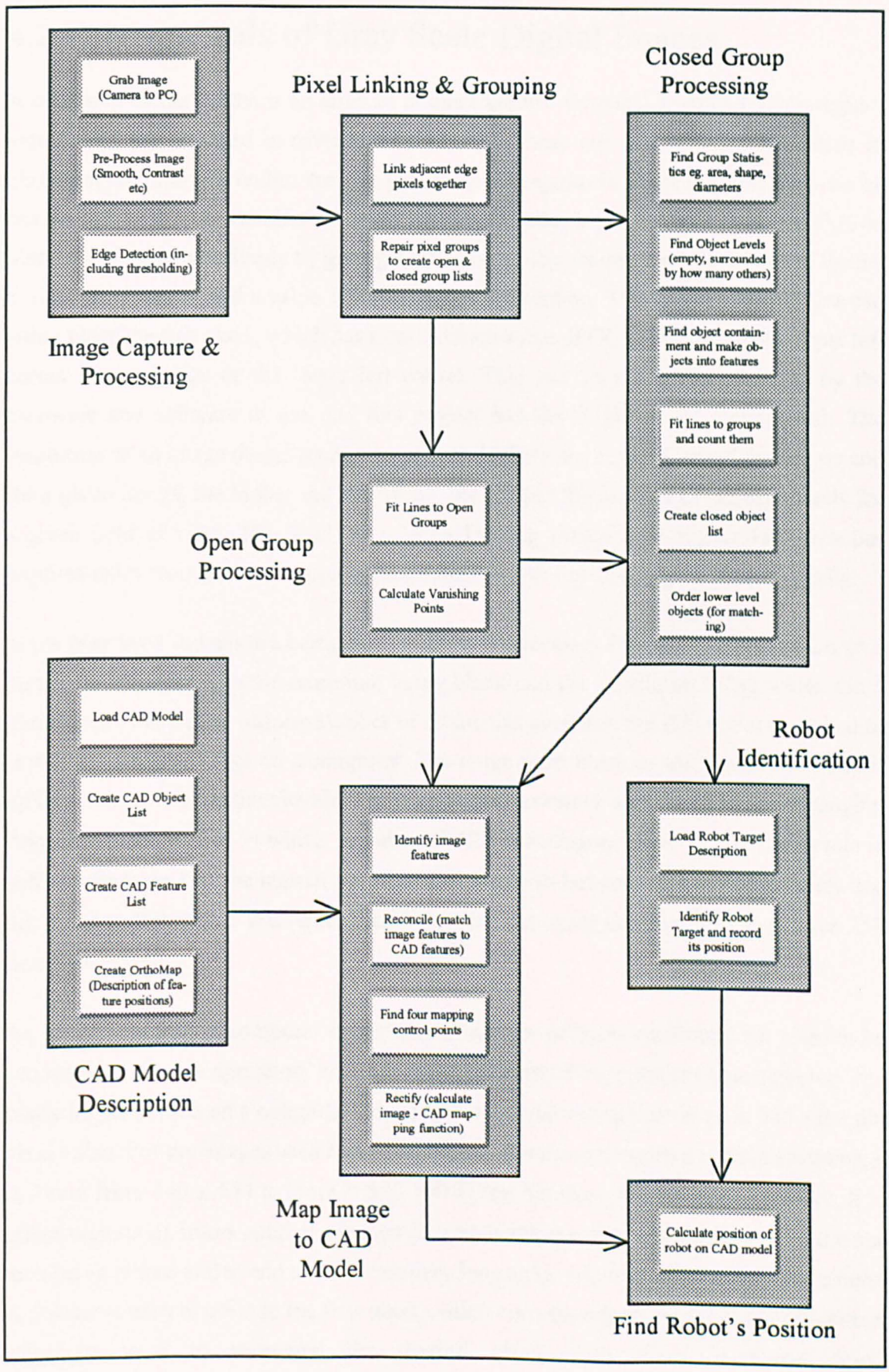


Figure 4.5 *General Algorithm Flow Diagram*

4.2 Fundamentals of Gray Scale Digital Images

A computer image contains an array of pixels (picture elements) which form a regular pattern and are arranged in rows and columns; (These are analogous to the grains in photographic film). In reality they are generally rectangular in shape but here they can be considered as a large number of small squares. Unlike a photograph, it is possible to identify a single pixel simply by giving it two co-ordinate values, namely a value X for the horizontal direction and a value Y for the vertical direction. Two conventions in use can either place the first pixel, which has a co-ordinate value of (X,Y) = (0,0), in the upper left corner of the image or the lower left corner. This can be sometimes dictated by the hardware and software in use and this project has the origin in the bottom left. The resolution of an image describes how many pixels there are in the X and Y directions and for a given image, the higher the resolution, the greater the number of smaller pixels for a given field of view. This leads to a better looking image with higher definition but requires extra storage. The resolution used here is 640 horizontally by 544 vertically.

Since gray level images are being dealt with, it is necessary for each pixel to be set to a particular intensity with the minimum being black and the maximum being white. On a photograph, there is an infinite number of intensities available but it is not economical to store this complete range on a computer. The range from black to white needs to be split up into a number of distinct levels. The pixels used here have a total of 256 levels ranging from 0 = black to 255 = white. A value of 127 is mid-gray. This number of levels is sufficiently large that the human eye cannot distinguish between two adjacent levels and the number 256 is also convenient for computer use since one byte can also have 256 possible values.

An image is stored in computer memory as a number of bytes enabling each pixel to be processed. A simple operation, which is the equivalent of the photographic negative, can easily be performed on a computer image by setting the new pixel value to 255 - the old pixel value. For the images used here, the amount of memory required to hold each image is found from 640 x 544 x 1byte = 348 160 bytes. To store the image in memory, it is effectively cut up into a number of strips (a strip being one horizontal line) and the strips then being placed end to end to form one very long strip, which is then placed in memory. A pointer is used to point to the first pixel, which corresponds to the pixel at (0,0) and an offset is used to reference the desired pixel. The offset is found from: (Y x image width) + X.

The image shown in Figure 4.6, which will be used for most of the examples throughout this work, demonstrates the above points.

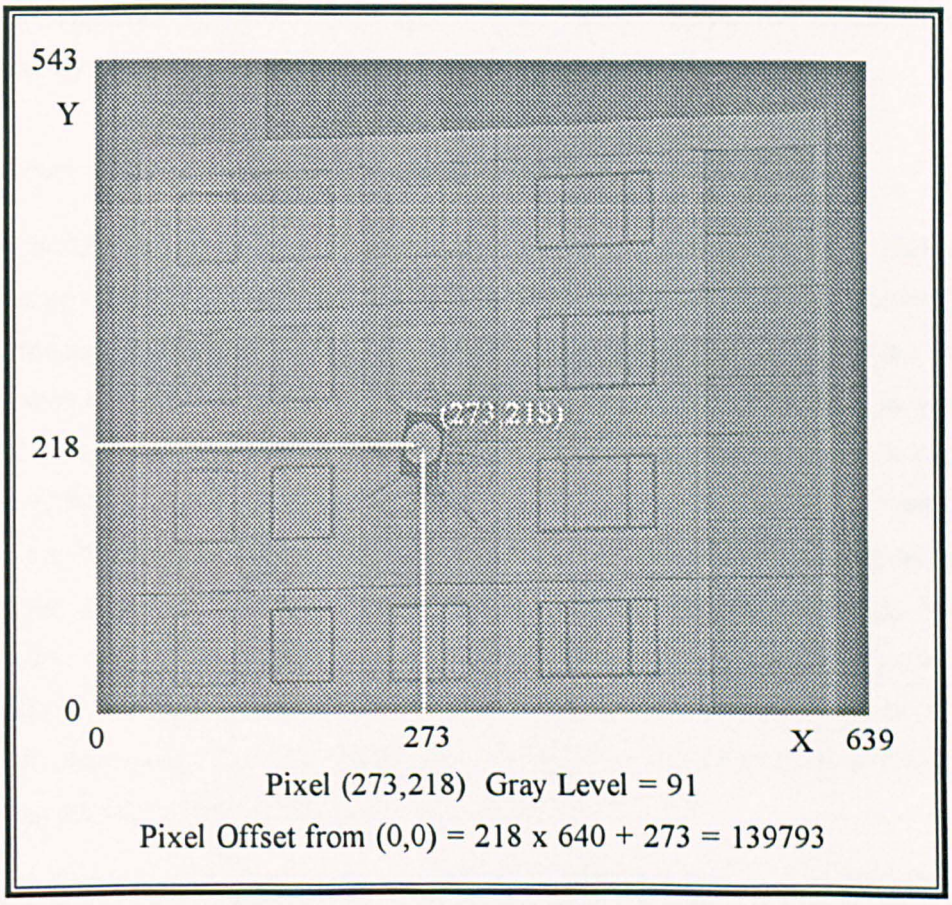


Figure 4.6 *Image Parameters*

4.3 Global Pre-Processing

Global pre-processing covers uniform functions applied to an image before any attempt is made to extract features. In a photographic context, going from a negative to a positive is a global, uniform function. Some of these functions can be performed by the hardware at the image capture stage which is always preferable since hardware operations can work in real-time and are considerably faster than software operations.

4.3.1 Image Capture - Camera/Frame Grabber

An image is captured by the light from the scene being projected through the camera lens onto a Charge Coupled Device (CCD). The signal generated by the CCD passes through some electronics and emerges from the camera either as separate luminance (Y) and chrominance (C) signals forming S-VHS, which are then combined to form an additional composite video output known as VHS. If a gray level image is to be captured (as is the case here), then only the intensity (or luminance) part of the signal is used. A full description of these are given in JVC, (1992). An important test of image quality is the step response of the system, that is, how well the camera responds to a rapid change in intensity such as black to white. Work carried out (Paterson, Dowling and Chamberlain, 1994a) has shown that different cameras have different step responses which can alter subsequent processing. The horizontal step response for the Panasonic S-VHS camera used to capture the images in this work are shown in Figure 4.7.

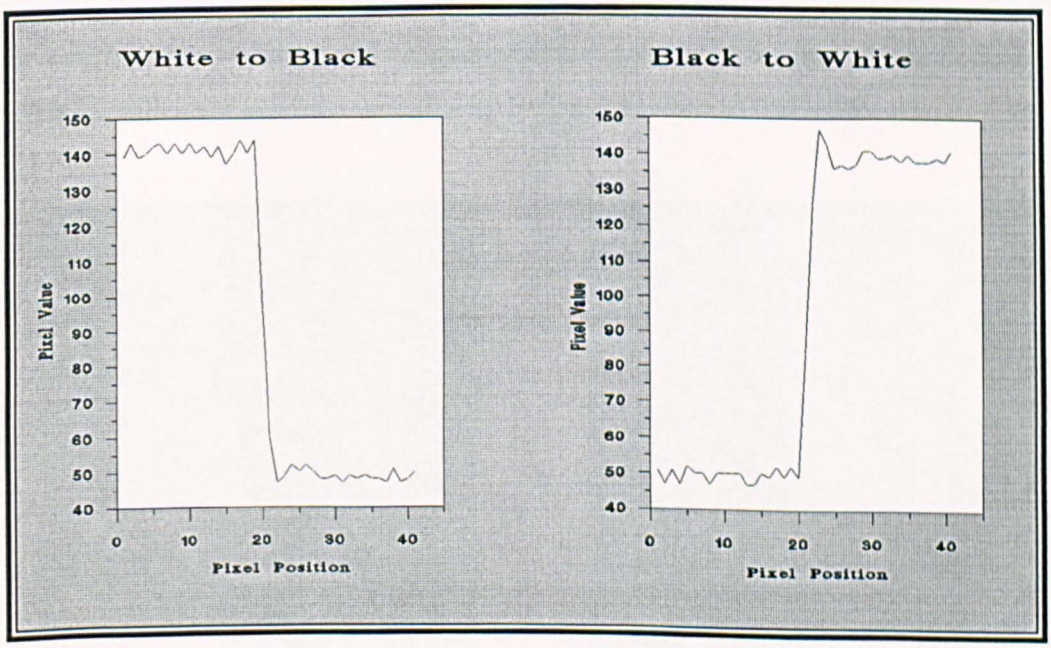


Figure 4.7 *Camera Step Response*

A frame grabber is used to convert the time varying analogue video signal (from the camera), into a digital signal which is sampled to produce an array of numbers representing the image in computer memory. Some grabbers have hardware controls enabling certain pre-processing functions to be performed. The one used here (see Appendix 4) has settings that can change the contrast and brightness of the image. This particular frame grabber has two controls, one called BLACK and one called WHITE and the best overall image was obtained when they were set to 12 and 5 respectively (total range is 0 to 15) with an S-VHS input signal. However, these controls may not be available on other systems and it may be necessary to implement contrast functions in software with them.

4.3.2 Contrast Enhancement

A typical grabbed image will not usually have gray levels ranging from 0 to 255 (black to white) but will tend to occupy some band in between. This may give a visually dull image which can be significantly improved by stretching the contrast over the whole range. The maximum possible contrast is produced by locating the maximum and minimum pixel values and finding the difference between them. This is divided into 255 to produce a scaling factor with an offset calculated from the lowest value. Every pixel value is then updated to create a new image that is visually improved. This function is linear but often an image may contain a concentration of gray levels at a particular range. Histogram equalisation, (Marion, 1991: 215-222) gives an even spread of gray levels and helps to bring out hidden detail as shown in Figure 4.8. Whereas these two contrast operations produce visually improved images, it was found that the effect on the image was often to amplify any noise (spurious pixels) present in the image which then had a detrimental effect on several edge detection processes described later.

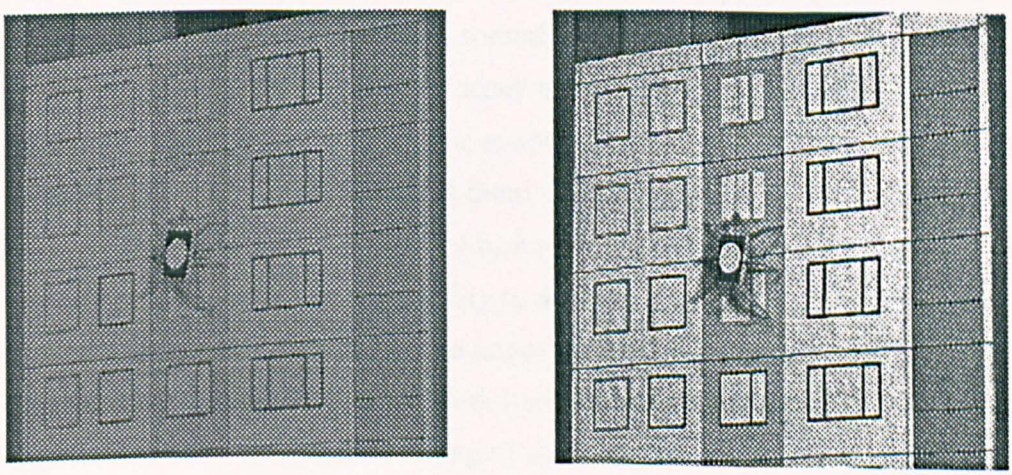


Figure 4.8 *Histogram Equalisation Applied to a Dark Image (Original on the left)*

4.3.3 Smoothing

Since the imaging and grabbing processes are not perfect, pixels will be assigned values different from their true values. This can be caused, for example, by the CCD not having a uniform response to a given light level. Dirt and marks on the object being viewed can also be classed as noise. Smoothing aims at minimising the effects of noise by blurring adjacent pixels. The simplest smoothing technique tried was to average the pixels in a 3 x 3 window. A given pixel will have 8 neighbours and the new pixel value is set to the mean of all 9 pixels. Although this method can remove small changes, a large noise spike such as a white pixel on a black background will spread its influence over the neighbouring pixels and the noise is not removed. Also it was found that this type of smoothing had a detrimental effect on significant edges. A good sharp transition, which is visually significant, became blurred and could, therefore, be missed in subsequent processing. Note that blurring can simply be performed by defocusing, if necessary.

An alternative is to use median smoothing (Gonzalez and Woods, 1993: 191-195). This takes into account the surrounding pixels to remove any noise spikes. As with mean smoothing, a 3 x 3 pixel window is used and the pixel values are sorted in order of intensity. The middle value is then taken as the new pixel value. Noisy pixels will tend to be placed at the beginning or the end of the list and will, therefore, not be included. This technique was found to be good at reducing noise while maintaining sharp image edges and it was a useful step to perform before certain edge detectors were used.

4.3.4 Edge Detection

Edge detection is probably the single most important step in image interpretation. When an object is looked at and perceived, it is not the object itself that is important but its edge. It is the edge which distinguishes it from something else and it is what the eye tends to focus on. An edge can be considered to occur when one texture changes into another. Much has been written about texture. For example, Caelli, (1993) classifies various texture processing methods and compares them with human vision. Examples are given of good segmentation between different texture types. Also Picard and Gorkani, (1994) compare an algorithm with the human ability to determine texture orientation and, in this application, may be useful for finding the edges of such objects as decorative concrete panels. However, as a starting point the work here takes a simple texture as being a fairly uniform gray surface. A building consisting of concrete panels and windows will have a texture change between the panel and the window frame usually in the form of different

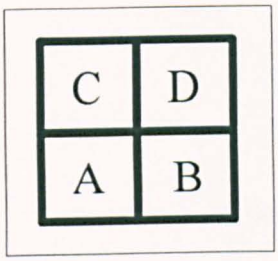


Figure 4.9 *Roberts Cross Pixel Window*

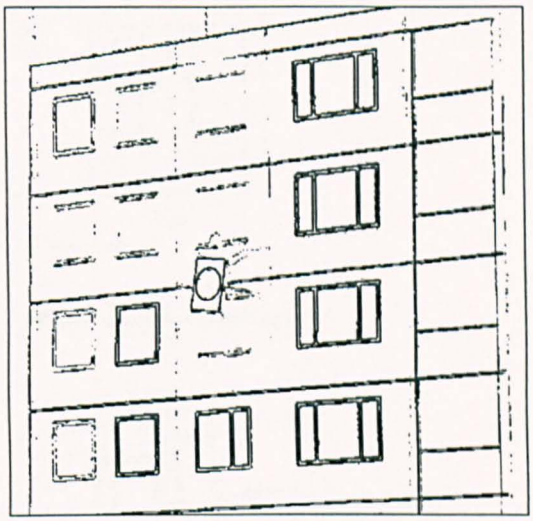


Figure 4.10 *Roberts Cross Edge Detection with Thresholding*

gray levels. Edge detection looks for these changes and there are a number of different techniques each with advantages and disadvantages; for example, Ramesh and Haralick, (1992) examine the performance of different edge detectors. Unfortunately it was found that the best results were obtained from detectors that took the longest time to execute.

The desired result of an edge detector is to produce single pixel width curves where a significant edge exists. Two basic types of detector were looked at for suitability, the first type being gradient edge detectors including Roberts Cross, Sobel and Canny. These require thresholding to produce an image which shows either edge or no edge. This is presented later in section 4.3.4.5. The second type looks at the behaviour of the pixels and decides whether an edge is present.

4.3.4.1 The Roberts Cross Edge Detector

This detector (Roberts, 1965; Gonzalez and Woods, 1993: 199-200) is probably the simplest of the gradient edge detectors as it only involves the use of a window of four pixels per image pixel. Figure 4.9 shows the 2 x 2 pixel window where pixel **a** is the current pixel of interest. The new value of **a** is found from $\mathbf{a}' = |\mathbf{a} - \mathbf{d}| + |\mathbf{b} - \mathbf{c}|$ and a thresholded output is given in Figure 4.10. The threshold was set to show the robot target as clearly as possible and it can be seen that large gaps occur in some features and some edges merged together to produce thicker edges. Further processing would be required to produce single pixel width edges but the routine is fast with a relative execution time defined here as 1.0.

-1	0	1
-2	0	2
-1	0	1

Gx

1	2	1
0	0	0
-1	-2	-1

Gy

Figure 4.11 *Sobel Pixel Windows*

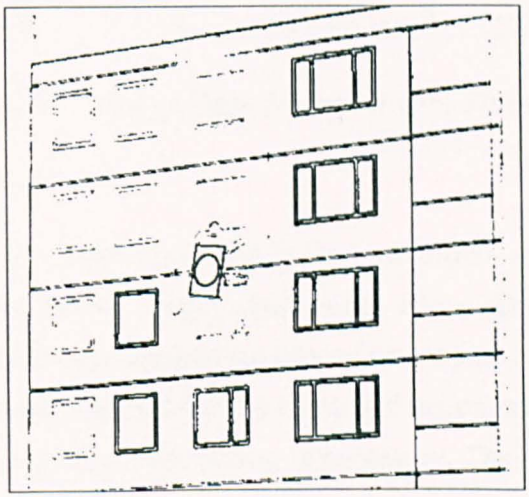


Figure 4.12 *Sobel Edge Detection with Thresholding*

4.3.4.2 The Sobel Edge Detector

One of the more commonly used edge detectors is the Sobel edge detector (Gonzalez and Woods, 1993: 418-420). Here, two 3 x 3 windows are used to calculate the gradient in the horizontal X direction and the gradient in the vertical Y direction. From these two values, the overall edge strength can be found as well as the direction of the edge, which may be useful in some applications. The two windows shown in Figure 4.11 are centred on the chosen pixel and the values **Gx** and **Gy** are calculated from the respective windows by multiplying the pixel by the number in the window and summing the result. Here, **Gx** and **Gy** correspond to the gradients in the horizontal and vertical directions respectively and the resultant gradient is given by $\mathbf{G} = |\mathbf{Gx}| + |\mathbf{Gy}|$. Figure 4.12 shows the thresholded (see section 4.3.5) result and is similar to the Roberts Cross detector described earlier especially in that the edges are more than one pixel wide. The relative execution time compared to the Roberts Cross method is 3.3.

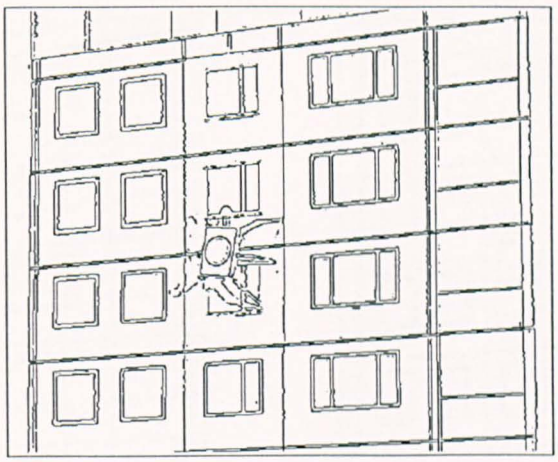


Figure 4.13 *Canny Edge Detection with Thresholding*

4.3.4.3 The Canny Edge Detector

This edge detector (Canny, 1986) is probably the best known and most commonly used since it is capable of producing single pixel width edges. Broadhurst, Pridmore and Taylor, (1994) chose it for their work in automated sewer pipe inspection, who state that it has become the "de facto standard for the computer vision community." It has a built in Gaussian filter, so no previous smoothing is necessary. The effects of smoothing are discussed in section 5.1.4.2. v.d.Merwe and Rüther, (1994) use this filter in four different directions (horizontal, vertical and the two diagonals) to enhance their version of the detector. The algorithm filters in the X and Y direction to produce partial images. These are then used to produce temporary gradient images to which an algorithm is applied to determine the most likely point for an edge. An algorithm originally written for Sun computers (Ellis, 1993) has been rewritten to work under Microsoft Windows 3.1. Although producing superior results, as shown in Figure 4.13, a considerable amount of memory is required for the storage of temporary images in floating point format and the relative execution time of 63.8 includes some disk activity which further slows down the execution. Since time is not important at this stage of development, the Canny edge detector was adopted on the basis of edge quality, although research into speeding it up is being conducted, for example, Mirmhedi and Ellis, (1993) have investigated implementing Canny in hardware using up to 16 transputers giving execution speeds of around 1.6 seconds compared with 30 seconds for a SUN Sparc1+ workstation using a 256 x 256 pixel image. As with the other edge detectors so far mentioned, the edges need to be thresholded (see section 4.3.5). It should also be noted that this detector searches for edges, so that weak edges where adjacent gray levels are similar are found as well as strong edges. The result of this is that more of the window frames have been highlighted particularly in the top left of the Figure 4.13.

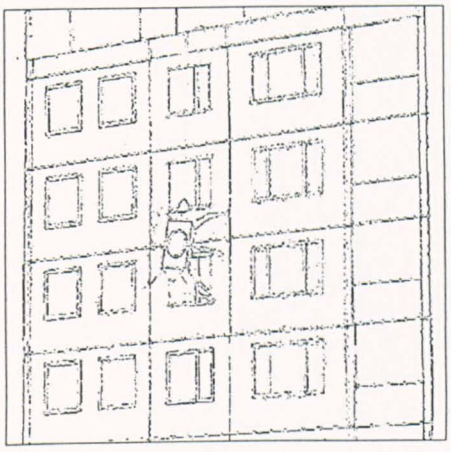


Figure 4.14 *Finite State Machine Output*

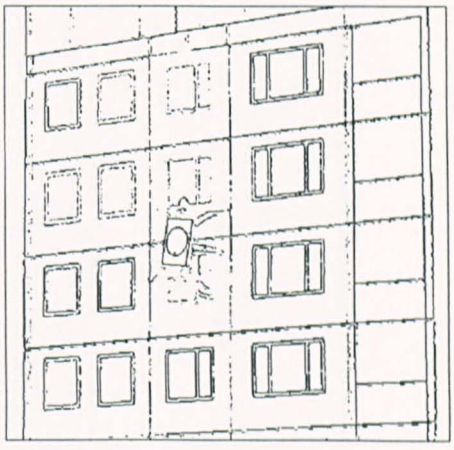


Figure 4.15 *Finite State Machine Output after Median Smoothing*

Although the edges shown here represent edge strengths that occur above a single threshold, Canny goes on in his paper to describe a method of extracting true edges from background noise by the use of 'hysteresis thresholding'. By taking a single threshold, there is the possibility that an edge, although valid, will have a strength at some point that falls below the threshold and will thus result in a gap appearing which should not be there. An improvement can be obtained by selecting two threshold levels. Any edge pixels whose strength is greater than the upper threshold are automatically classed as edges. An edge is then tracked with following pixels being accepted as edges until the strength falls below the lower threshold and the edge is then terminated. Canny uses a ratio of high to low threshold of two or three to one with only the top 20% of values being classed as definite edges. This additional processing has not been implemented here as it was the performance of the actual edge detector that was of interest and separate thresholding is described and implemented in section 4.3.5 with a separate gap 'repair' algorithm given in section 4.4.1.

4.3.4.4 The Finite State Machine (FSM) Edge Detector

A totally different approach to edge detection is presented by Ginige, (1992). Here, the image is scanned in the horizontal and vertical directions to produce a list of pixel intensity values. A value g is calculated from the difference in the current pixel value and the next pixel value. The pixel is then classified into one of seven groups depending on value of g. An edge is assumed to exist where combinations of these groups have occurred in a particular order. The FSM is a method for searching the combinations by creating a pattern of states. Movement from one state to the next is determined by the pixel

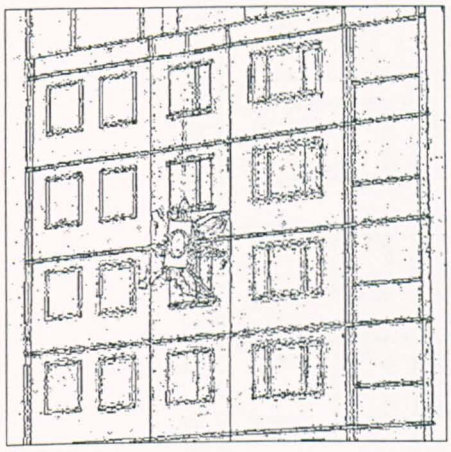


Figure 4.16 *Window Edge Detector Output*

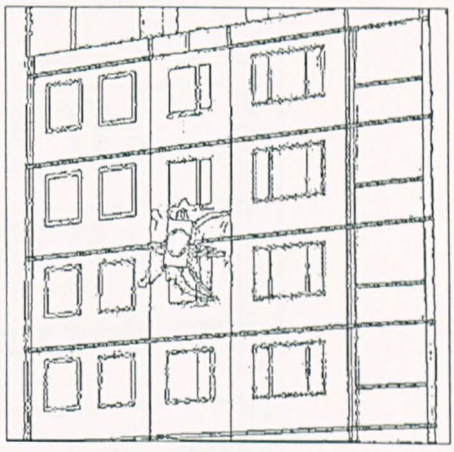


Figure 4.17 *Window Edge Detector Output after Median Smoothing*

difference group and many different paths will be taken as the image is scanned. Only if a particular state is reached, is an edge said to exist at that particular pixel. This method has a major advantage over gradient edge detectors in that weak, but significant, edges give the same result as strong edges and, furthermore, the edges are a single pixel wide. The output of this detector is shown in Figure 4.14. It is fairly quick, with a relative execution speed of 1.6. Although it can be clearly seen that all the significant features have been highlighted evenly, the quality of the edges is poor. By median smoothing (see section 4.3.3) the image first, a considerable improvement is achieved as shown in Figure 4.15, however the edge quality does not match that of the Canny edge detector.

4.3.4.5 Window Edge Detector

This detector was developed by the author to try and spot perceived edges rather than just strong edges. The assumption was made that if one could see a change in an image, then an algorithm should also be able to detect these changes. A scan line was made across an image and any perceived edges were marked and similarities were noted. It was found that if a scan window five pixels wide is used and that the difference in the pixel values at the ends of the scan window was more than ten, then there was a good chance of a perceived edge. If more than one adjacent pixel satisfied this condition then the central pixel was chosen to be the edge. This process is performed in the X and Y directions with the results being combined to give the image in Figure 4.16. As with the FSM method, all the significant features have been highlighted but there is a considerable amount of noise. Median smoothing of the image first, again produces a much better output as shown in Figure 4.17 although here, the weaker edges have produced a better result. This method

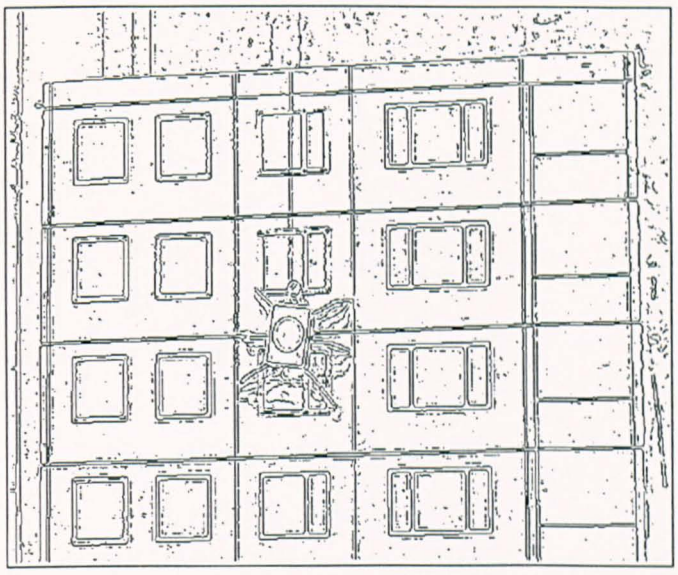


Figure 4.18 *Threshold level = 10*

was a useful experiment and runs quickly with a relative time of 1.8, but can give a quite different result if the brightness and contrast values are changed. Further work would be necessary to see if it is worth the extra processing to cope with these changes and clean up the edges, particularly when the Canny gives a far better, albeit slower, result.

4.3.5 Thresholding

Thresholding is used to convert an image from its normal 256 gray levels to a 'binary' image, that is, an image that only contains two colours, usually black and white. If a pixel has a value greater than that of the threshold level, it is turned white (255), otherwise it is turned black (0). It is used here as a post edge detection process, when gradient edge detectors are used, to obtain white pixel edges on a black background. As mentioned earlier (see 4.3.4.2), some edge detectors produce an output proportional to edge strength and this has become a necessary step since the Canny edge detector was adopted. The problem now becomes to decide at what level the threshold is to be set. If it is set too high, then many of the significant edges will be lost. If it is set too low, then very weak edges will be included along with background noise and they will tend to obscure the desired edges. Tseng and Huang, (1993) take an interesting look at thresholding gray level images based on human perception which required "neither an iterative operation nor edge detection", but they do not apply their results to edge detector output. Figures 4.18 through 4.20 show the effect of different thresholds after Canny edge detection with thresholds of 10, 22 and 60 respectively. The low threshold of 10 shows clearly how noise

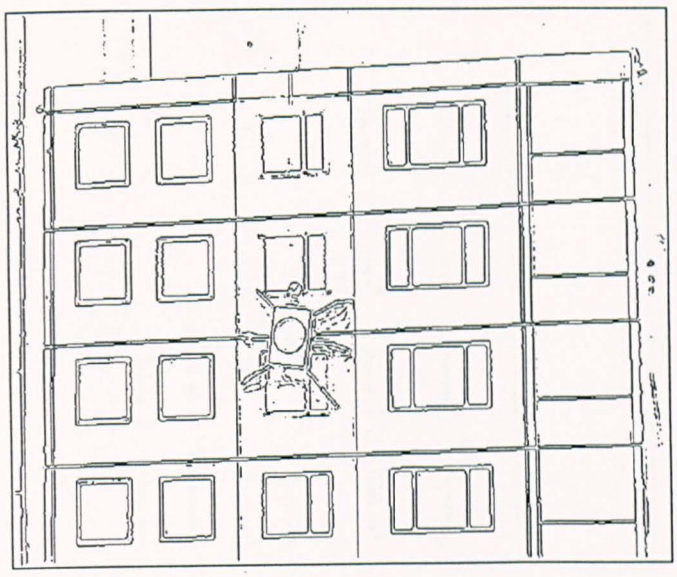


Figure 4.19 *Threshold level = 22*

is being included as well as spurious edges. A close inspection of the robot target shows that a 'ghost' circle is appearing within the real circle. The high threshold of 60 removes any unwanted edges, but it is also removing desired, but weak, edges. Only the strongest changes, which were originally black against white, have been highlighted. The level of 22 was chosen statistically and has the best combination of a small amount of noise while keeping the weaker edges. This level was found by calculating the mean and standard deviation of the pixel values of the edge output. It was demonstrated that setting the threshold level T equal to the mean plus one standard deviation gave a good result. However, this may change for different images; the way to find the best level for a given image is to try several different thresholds and see which one yields the greatest number of identifiable features. A slightly simpler approach is to look at the number of open and

Object Count Ratios	Stat. Thr.	T - 8	T - 7	T - 6	T - 5	T - 4	T - 3	T - 2	T - 1	Statistical Threshold	T + 1
Figure 4.6	22	0,405	0,505	0,578	0,581	0,615	0,584	0,558	0,519	0,594	0,646
Figure 5.1d	22	0,259	0,263	0,248	0,251	0,257	0,261	0,275	0,277	0,289	0,315
Figure 5.16	10	0,111	0,159	0,198	0,214	0,214	0,251	0,309	0,308	0,367	0,371
Figure 5.24	9	-	-	-	0,229	0,240	0,247	0,286	0,279	0,338	0,373

T + 2	T + 3	T + 4	T + 5	T + 6	T + 7	T + 8	T + 9	T + 10	T + 11	T + 12	T + 13	T + 14	T + 15	T + 16
0,627	0,724	0,865	0,902	0,770	0,682	0,547	0,583	0,416	0,432	0,409	0,402	0,406	0,418	0,393
0,287	0,291	0,308	0,325	0,319	0,327	0,333	0,309	0,295	0,301	0,322	0,313	0,326	0,285	0,328
0,464	0,477	0,467	0,456	0,439	0,481	0,510	0,474	0,483	0,532	0,506	0,500	0,538	0,513	0,440
0,410	0,454	0,490	0,495	0,489	0,489	0,382	0,371	0,337	0,327	0,289	0,301	0,277	0,258	0,226

Table 4.1 *Open and Closed Object Count Ratios at Different Thresholds*

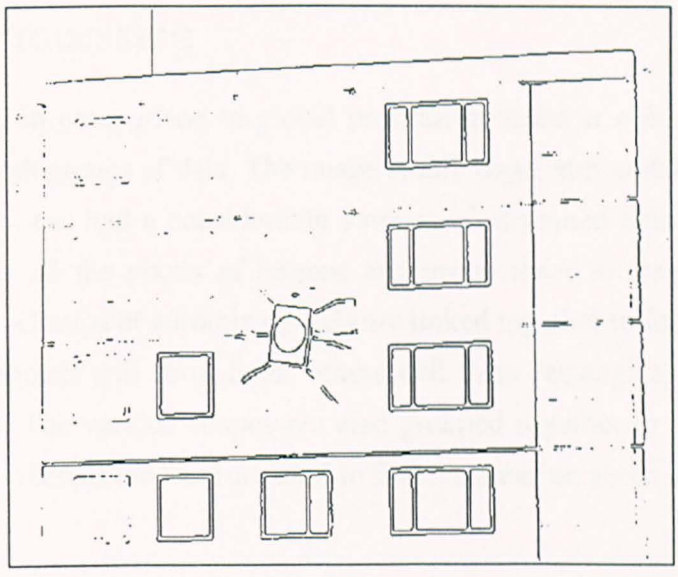


Figure 4.20 *Threshold level = 60*

closed pixel groups (see section 4.4.1) and determine how they vary at different thresholds. Table 4.1 shows the ratios of closed groups to open groups for four different images at a range of thresholds, the first image being the one used in the accompanying figures. It was found that when the ratio was at a maximum over the selected range, then it was more likely that the features of interest would be identified. The drawback of doing this is that two of the processing steps need to be repeated many times, increasing the duration of the processing, although in a parallel system, this would not be a problem.

4.4 Local Processing

Local processing in comparison to global processing looks at either small areas of an image or deals with groups of data. The image at this stage, although containing the same number of pixels, has had a considerable amount of unwanted data removed. The task now is to collect all the pixels of interest and group them in ways that will become meaningful later. Groups of adjoining pixels are linked together to form different objects. Some of these objects will form lines, others will form rectangles, simple shapes, and random patterns. The various shapes are also grouped together to try and form recognisable features, whereas the lines are used to find information about the orientation of the image.

4.4.1 Pixel Linking

At this stage there are single pixel width edges, but there is no indication as to how the pixels relate to each other. Pixel linking, or connectivity, attempts to group together those pixels that are neighbours, as these groups are the precursors to higher level features such as lines and rectangles. This stage is also the one where the transition between processing the entire image and processing specific data structures occurs. The pixel linking function can be split into the two main tasks of initial grouping and group repair followed by the calculation of group statistics for subsequent processing. The aim of the initial grouping is to run through the edge pixel image and create two lists, an open list and a closed list, of all pixels that are neighbours. The essential parts of an algorithm (Rosin and West, 1989) are as follows:

1. Convert neighbours to 8-connected form
2. Remove single pixels
3. Link pixels to form the open group list
4. Remove single pixels

8-connected neighbours

An individual pixel can have up to eight neighbours but, since single pixel width edges are being dealt with, there should be a maximum of two. However, the result of the edge detection process can give more than two neighbours as shown in Figure 4.21. These pixels are connected in what is known as 4-neighbours (Sonka, Hlavec and Boyle, 1994: 31) such that it is only possible to move from one pixel to the next in the up, down, left and right directions. It can be seen that the corner pixels are effectively redundant if

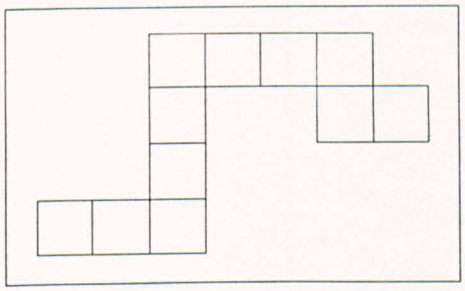


Figure 4.21 *4-Connected Pixels*

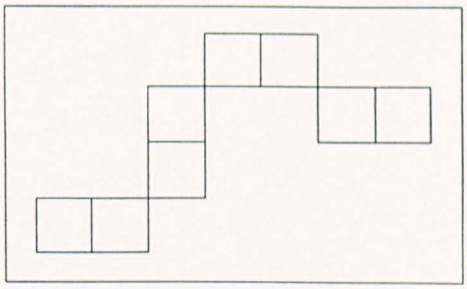


Figure 4.22 *8-Connected Pixels*

being allowed to move in the diagonal directions as well. These corner pixels are therefore removed (made the same colour as the background) to leave the 8-neighbour form shown in Figure 4.22, a necessary step for the linking process. In addition, further information about the pixel group can be gained when the list is 8-connected.

Removal of single pixels

Single pixels are generally the result of noise and should, therefore, be removed. In addition, a group of one pixel is meaningless and can give no possible indication as to which lines might pass through it.

Open group linking

The first set of groups to be found are open groups. These are groups of pixels which have two ends and may be part of lines for example. The algorithm searches the image looking for a pixel with only one neighbour and then follows the pixels until the other end is found. Once the group data and pixel co-ordinates have been stored, the group is deleted from the image and the process continues until no more open groups are found. It may be possible for the algorithm to leave single pixels so once again single pixels are removed. Only groups that have greater than 5 pixels are accepted.

Closed group linking

The only remaining pixels in the image are those that belong to closed groups, that is, groups that do not have ends and could form shapes such as circles and rectangles. The algorithm simple scans the image looking for a pixel and then follows the pixels, adding them to the closed list, until it returns to the starting point. As with the open groups, the closed group is deleted from the image and the process continues until nothing is left.

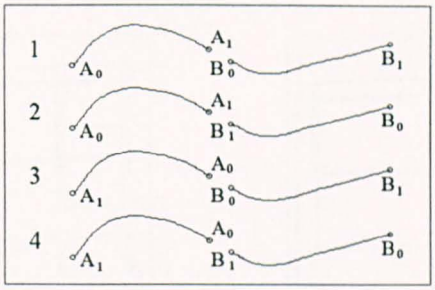


Figure 4.23 *Possible Open Group Combinations*

Group repair

Although the linking process is ideal for creating groups, the pixels from the edge detection process are not perfect. Just as noise can add extra pixels, it can also remove desired pixels. As this form of processing is pixel based, it only takes one pixel to be missing from a closed group to make it open and is, therefore, added to the wrong group. The group could form part of a significant feature and, if missing, the feature would not be recognised, all because of one pixel. Similarly, there may be several breaks in one closed group meaning that the ends cannot be joined. This is shown later in Figure 4.24. The aim of the group repair function developed by the author is to see if any of the open groups can be joined together. Here, if there is a gap of up to 5 pixels between group ends, then we assume that this is a genuine gap and not two separate groups. The repair function works as follows:

```
Repeat:
    For each open group k, work through the remainder
    of the open group list ( k + 1 to k = n ) and see if
    any of these groups can be joined to group k. If it
    can, then a new joined group is created with the
    gap filled in  by adding extra pixels and the
    original two groups are deleted.
Until it is not possible to join any more groups.
```

The repaired closed list is checked to determine if both ends of a given group lay within a certain distance of each other. This distance is the same as that used for joining the open groups above. If the ends are within this distance, then extra pixels are added to close the group, the group is moved from the open group list to the closed group list.

Joining groups is not quite as simple as it may seem. It can be seen quite clearly where to join two groups in an image but, since these groups are stored as lists in computer memory, there are four possibilities to consider to ensure that the correct ends are joined.

Figure 4.24 *Raw Open Groups*

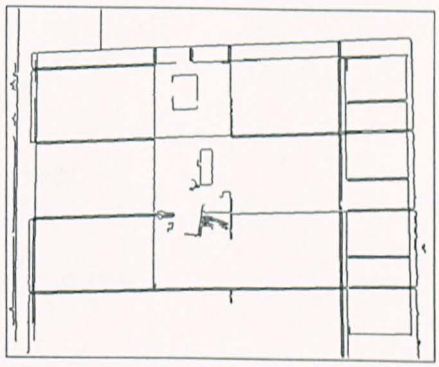


Figure 4.25 *Open Groups with Gaps*

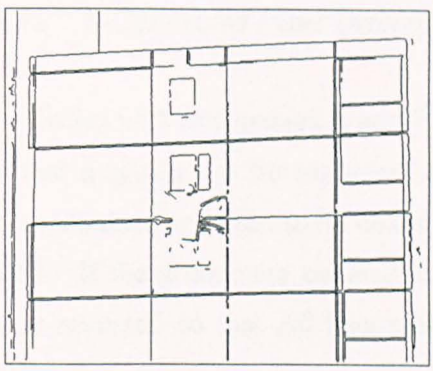


Figure 4.26 *Repaired Open Groups*

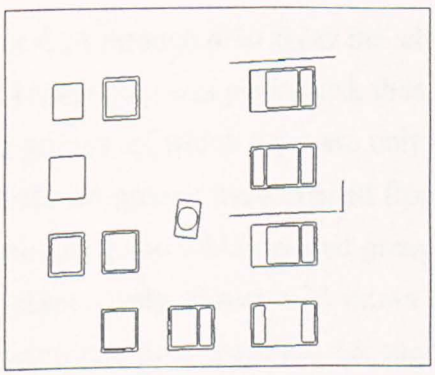


Figure 4.27 *Raw Closed Groups*

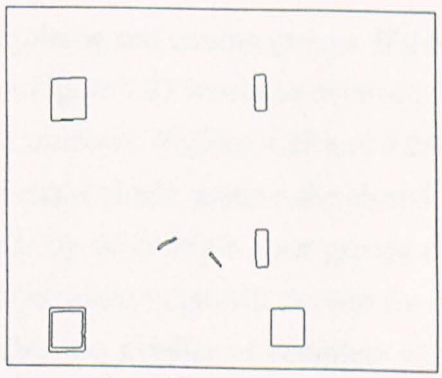


Figure 4.28 *Closed Groups with one Gap*

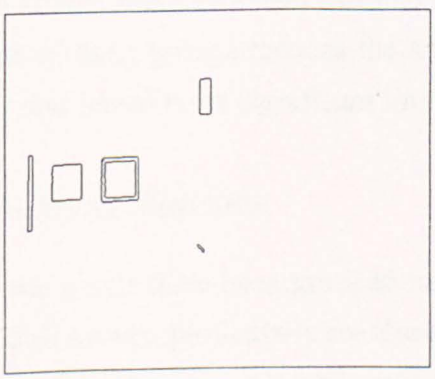


Figure 4.29 *Closed Groups with many Gaps*

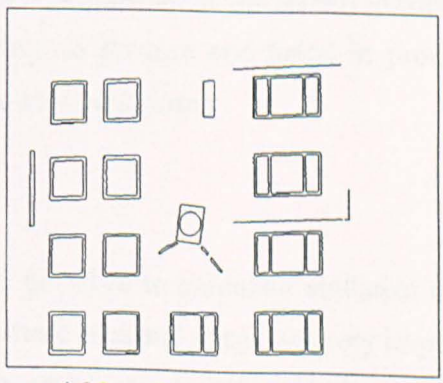


Figure 4.30 *Repaired Closed Groups*

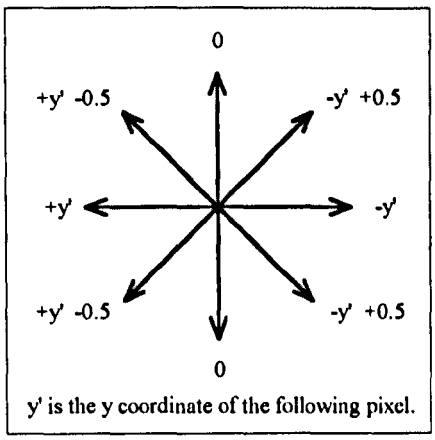


Figure 4.31 *8-Connected Pixel Direction Sums*

Figure 4.23 shows these possibilities with two groups A and B. The end '0' is the first one in the list and so it is seen that a group can be represented in two ways. In the first example, the ordering of the pixels does not need to be changed and the gap between A1 and B0 just needs to be filled in. If the groups are ordered as in the third example, then the pixels of group A must be reversed so that A0 becomes A1 and the group can be joined to B0.

Figures 4.24 through 4.30 show the advantages of joining and closing groups. If the pixel linking stage only was performed, then the result in Figure 4.27 would be obtained for the closed groups, of which there are only 6 complete windows. Figures 4.28 and 4.29 show which closed groups were created from either joining a single group (the closed group had one gap) and which closed groups were made up of multiple open groups (many gaps) respectively. Figure 4.30 shows the final closed groups that will be used for further processing and now there are 13, more than double the number of complete windows which is a considerable improvement. Similarly for the open groups, Figure 4.25 shows which groups could be joined. Although this is not so important as the closed groups, this joining of open groups reduces the amount of data to process and helps in producing longer and hence more significant lines in subsequent processing.

4.4.2 Object Statistics

Once the pixels have been grouped together it is possible to calculate statistics for the individual groups, particularly the closed groups where area and shape are very important. Some of these groups can be related to each other and extra statistics can then be found for the accumulated groups. These statistics will be used later in the feature identification processes where group statistics are compared to equivalent groups from the CAD model.

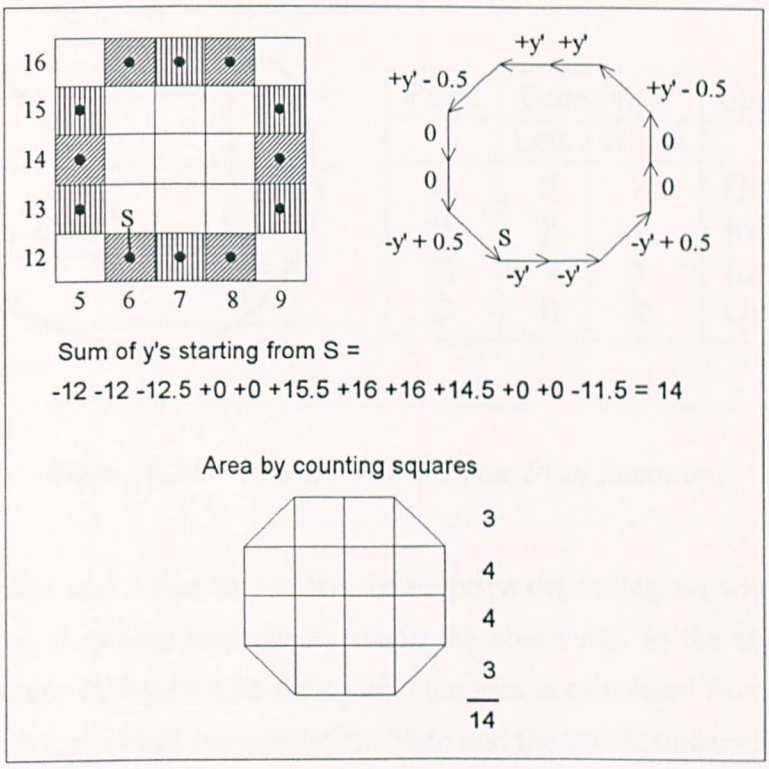


Figure 4.32 *8-Connected Area Calculation*

4.4.2.1 Single Object Statistics

Area

The area of an object can be determined if its group of boundary pixels, found previously, are connected as 8-neighbours. There is no need to draw the object and count internal pixels as only the boundary pixel co-ordinates are required. The area is found by tracking along the pixels and seeing in which direction the following neighbour lies. A vertical single pixel width slice through the object has an area equal to the difference in the y co-ordinates across the object so as we go round the pixels, the y value is either added or subtracted depending on the x direction. Half pixels are added or subtracted if we travel in diagonal directions as indicated in Figure 4.31.

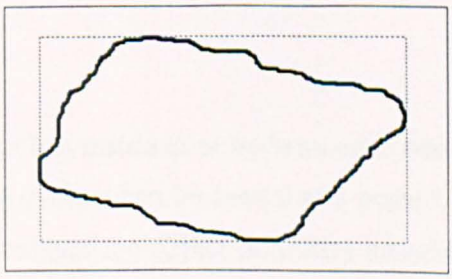
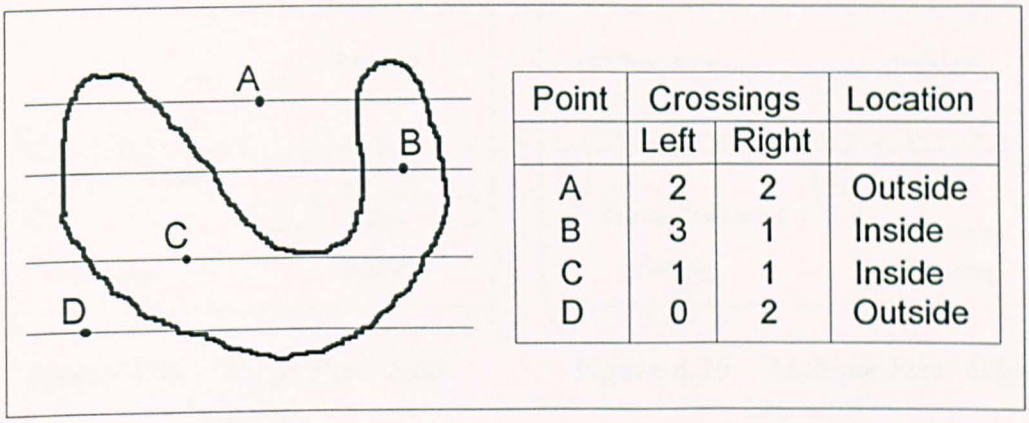


Figure 4.33 *Orthogonal Bounding Rectangle*



Point	Crossings		Location
	Left	Right	
A	2	2	Outside
B	3	1	Inside
C	1	1	Inside
D	0	2	Outside

Figure 4.34 *Interior and Exterior Pixel Locations*

The resultant area will either be positive or negative depending on whether travelling anticlockwise or clockwise respectively round the object and so the absolute value is taken. The example in Figure 4.32 shows how the area is calculated from the pixels and compares it with a graphical representation. Note that the true boundary is considered to be at the centre of the pixels.

Bounding rectangle

The bounding rectangle is a rectangle that encloses the object. It can have a number of different sizes depending on its orientation but here one is taken that has horizontal and vertical sides as this is useful for setting limits when working on the object. The rectangle is found by taking the minimum and maximum **x** co-ordinates and the minimum and maximum **y** co-ordinates in the pixel group as shown in Figure 4.33.

Object centre (Centroid)

The object centre is taken to be the centre of mass of the boundary pixels. Since each pixel can be considered as having a weight of 1, the centre is found by finding the mean **x** co-ordinate and the mean **y** co-ordinate. The object centre is very useful for comparing shape topologies by seeing if the centre lies inside or outside the object. (The letter U has its centre outside).

Object centre position

Determining whether a point lies inside or outside an enclosed shape is in fact a nontrivial problem. Looking at Figure 4.34, it can be seen that a point lies inside an object if a line passing through that point crosses the object boundary an odd number of times on either side of the point. This applies to any line direction but for convenience, a scan line is taken

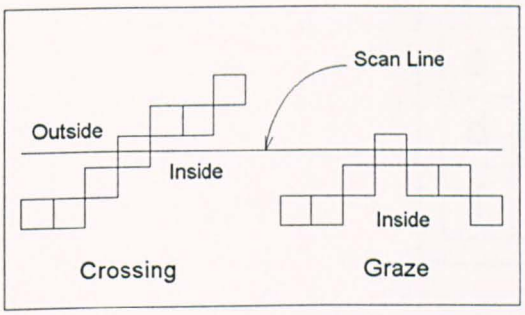


Figure 4.35 *Single Pixel Edge Crossing*

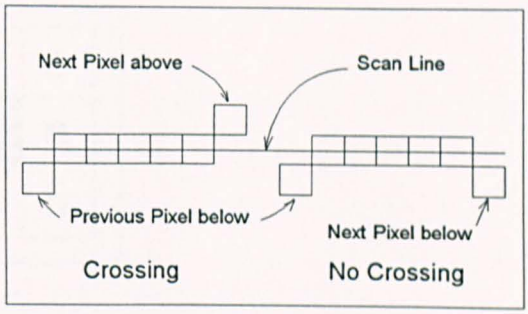


Figure 4.36 *Multiple Pixel Edge Crossing*

running in the x-direction. There is now another problem. What is meant by 'crossing the boundary'? This is a transition from outside the object to the inside or vice versa but it is necessary to know whether the point is inside or outside, which is what is being looked for! By using the bounding rectangle above, the initial starting point can be set to one pixel outside the bounding rectangle so it is known that the start is outside the object. Since the object boundary is made of pixels of finite size, there is the problem of deciding whether the boundary is actually crossed. Because of quantisation, the boundary may be several pixels wide along the scan line. A resulting problem is that the object can be 'grazed' and appear to be inside when in fact we have remained outside as shown in Figure 4.35. When a pixel is encountered the lines above and below the scan line must be inspected to see in which direction the border was travelling. Figure 4.36 shows the difference between a true crossing when pixels at the end of a border coincident with the scan line are on opposite sides of the scan line, and a graze when end pixels are on the same side. With 8-connected pixels, the algorithm developed by the author for a crossing is as follows:

1. Track along the scan line until a pixel is hit.
2. Using the window in Figure 4.37, set the letters **a** to **h** to 1 if their corresponding position contains a pixel, otherwise set them to 0.
3. Set **s** = **a** + **b** + **c** and **t** = **f** + **g** + **h**
4. If **d** + **e** = 0, we have a single pixel width edge AND if **s** = **t** = 1, mark the border as crossed, STOP.
   else
   do steps 5 to 9.
5. If **d** + **e** > 0, we have a multiple pixel edge.
6. If **s** = 0 AND **t** = 1 set **u** = 1 else if **s** = 1 AND **t** = 0 set **u** = 0.

Figure 4.37 *Edge Crossing Window*

7. Continue along scan line until last pixel is reached then do steps 2 and 3.
8. If **s** = 0 AND **t** = 1 set **v** = 1 else if **s** = 1 AND **t** = 0 set **v** = 0.
9. If **u** is not equal to **v** then mark the border as crossed.

To determine whether a point is inside or outside the object, the scan line **y** value is set to the point's **y** co-ordinate and start at the left edge of the bounding rectangle, keeping count of crossings until the selected point is reached. If the number of crossings is odd, then the point is inside the object. A check can be made by continuing from the point and seeing, if again, the number of crossings is also odd.

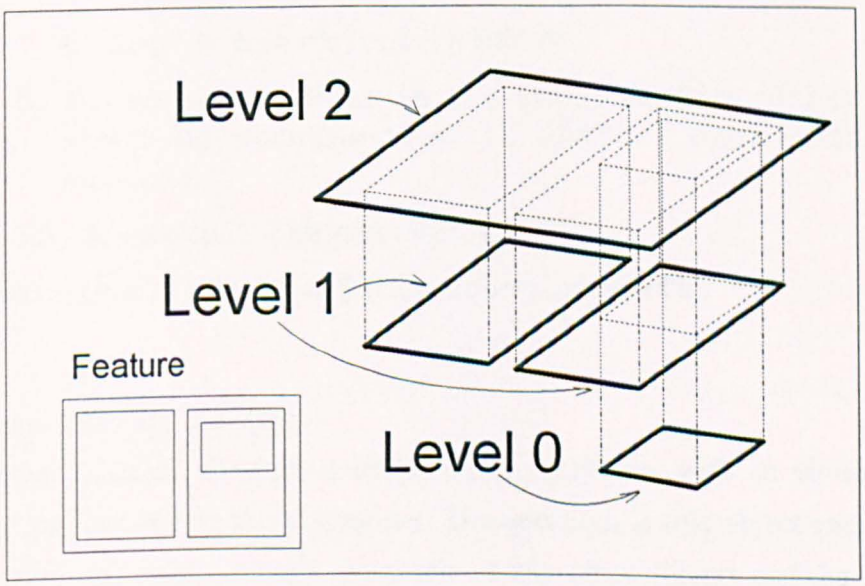


Figure 4.38 *Object Levels*

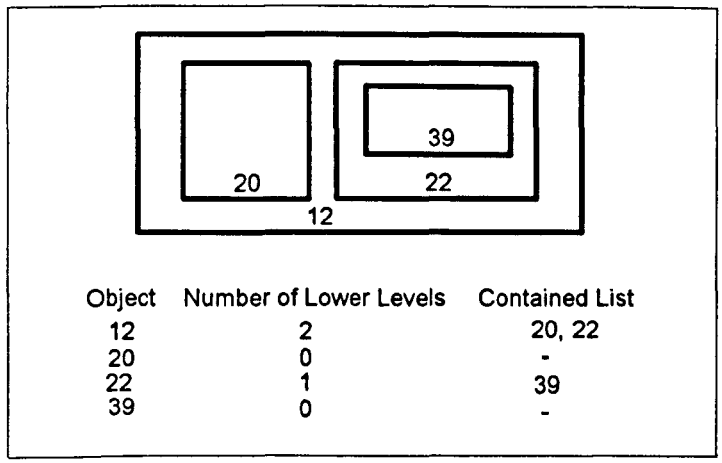
4.4.2.2 Calculation of object levels

Most of the features to be identified do not consist of a single closed group of pixels but are more likely to consist of a number of closed groups within other closed groups. This section aims at finding out at which level a pixel group, or object, is located. An object at a given level will contain at least one other object and an empty object is defined as being at level zero. A feature consisting of objects at different levels can be compared with a CAD feature to see if it has the same number of levels. What is meant by 'level' can be visualised in Figure 4.38 where objects at the same level are drawn on the same plane. The algorithm for finding the levels is as follows, where specific colours are given for clarity:

```
1. Set the current level to 0 ( empty ).
2. Set the level of each object in the closed group
   list to NO_LEVEL.
3. Draw all the closed objects in WHITE on a BLACK
   background.
Repeat
   5. For each object in the list, do 6 - 8.
      6. If the object's level is set to NO_LEVEL then
         draw the object in RED.
      7. If the object is empty ( see text ) then set
         the object's level to the current level.
      8. Redraw the object in WHITE.
   9. Erase all objects in the image at the current
      level by drawing them in BLACK ( background
      colour ).
   10. Increment the current level.
Until there are no objects drawn in WHITE.
```

Determining if an object is empty

This is another instance of where a trivial human problem, with an obvious answer becomes a complicated task for a computer. The question: Is this object empty? can be rephrased: Does this object contain the pixels of any other object? and the problem is essentially reduced to same one as determining whether the object's centroid is inside the object as given in section 4.4.2.1. Using the bounding rectangle (see section 4.4.2.1) of the object, a scan is made at each **y** value in the bounding rectangle. With the object drawn in the test colour RED on a BLACK background the algorithm moves along the scan line



Object	Number of Lower Levels	Contained List
12	2	20, 22
20	0	-
22	1	39
39	0	-

Figure 4.39 *Object Containment*

until it finds a WHITE pixel, which must belong to another object. This pixel is tested to see if it lies inside or outside the test object. If the entire bounding rectangle is scanned without a pixel being found inside the object, then the object is empty.

4.4.2.3 Object Containment

Once the object levels have been found, it is possible to find which objects are contained or enclosed by another. This enables various objects, whose order in the closed list is fairly random, to be grouped together to potentially form a feature. The ordering of containment, namely which object is contained by the current object, forms a very strong basis for feature recognition since this ordering cannot be changed by image distortion. The letter **A**, for example, will always have a hole in the upper half no matter how it is distorted. The levels found in section 4.4.2.2 are used to see whether an object is contained as it is only possible to contain an object at the next lower level. An object at an even lower level will be contained only by the object at the level immediately above it. Figure 4.39 demonstrates this in that object 39 is only contained in object 22 and not object 12. The algorithm for finding containment is given as follows where again colours are given for easier understanding:

1. Set the image space to the background colour - BLACK.
2. For each object in the closed object list, do:
   3. Set the lower level count to 0 (empty).
   4. If the current object is not empty (level not equal to 0) do steps 5 to 10.

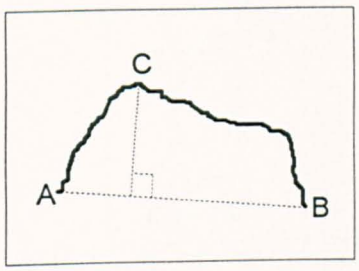


Figure 4.40 *Line Finding - 1st Attempt*

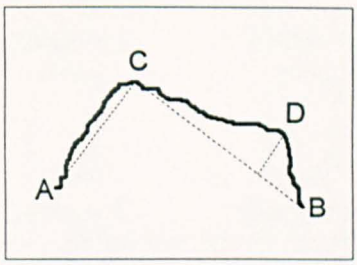


Figure 4.41 *Line Finding - 2nd Attempt*

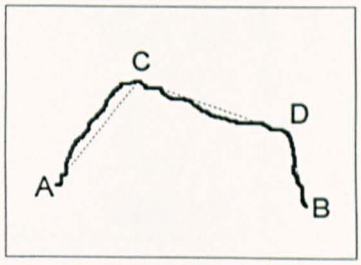


Figure 4.42 *Line Finding - 3rd Attempt*

5. Draw the current object in the test colour RED.

6. For each object in the closed object list, not including the current object, do steps 7 to 9.

   7. If the object's level is one lower than the current object, see if the object's centre lies inside the current object using the method described in section 4.4.2.1 and if it is, perform steps 8 and 9.

      8. Add the object to the current object's list of contained lower level objects.

      9. Increment the lower level count of contained objects.

10. Delete the current object from image space by drawing it in BLACK.

4.4.2.4 Line Finding and Counting

Since the images being looked at are specifically of buildings, it is reasonable to assume that many of the features in the image will be made up of straight lines. Most windows in tower blocks appear as quadrilaterals, for example. The processing so far has not resulted

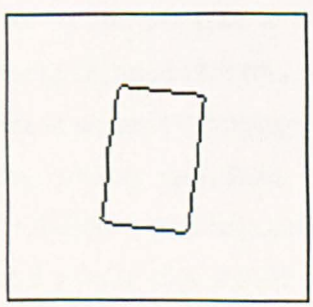


Figure 4.43 *4-Sided Shape from an Image with rounded Corners*

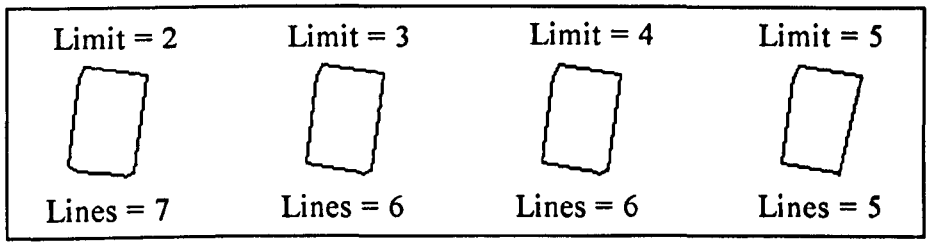


Figure 4.44 *Lines Fitted with Different Limits*

in a description of the actual shapes. It should also be noted that circular (elliptical) shapes can be crudely detected since they will consist of many short edges whereas a quadrilateral shape should consist of four lines.

One method of line finding developed by Nelson, (1994) is to grow line segments from a strong point on an edge. This is fairly complex and a simpler method, which produces just as good, if not better results developed by Rosin and West, (1989) to break down edges into arcs and lines. They use a method described by Lowe, (1987) to find lines which recursively works on a group of 8-connected pixels (see section 4.4.1) fitting lines until no more are possible. It is emphasised here that the lines being fitted are not the best fit through the pixel group since the shape can be quite complex. Consider the group of pixels given in Figure 4.40 (an open group is used for clarity). A line is initially drawn between the two end points A and B. Each pixel in the group is taken in turn and its (normal) distance from the line is found. The pixel with the greatest distance from the line, point C, is considered an end point of two new lines, which form a better approximation to the pixel group. Figure 4.41 shows these new lines and the process is repeated with segments AC and CB. If the maximum distance from a pixel to a line is less than a pre-defined limit, then the process is halted for that line segment. The segment AC is a good enough fit but the segment CB yields a further point D. The final result is shown in Figure 4.42 where the pixel group has been completely approximated by straight lines.

The setting of the minimum pixel to line distance is dependent on the type of features being searched for. At first it may be thought that this value should be zero so that there is perfect line fitting but since there is noise in the image and pixel positions are quantised, there would be a very large number of very short lines. At this part of the process, closed objects are being examined and a fairly coarse line fitting is required. A typical 4-sided shape is shown in Figure 4.43 and ideally this should have only four lines fitted by the algorithm. But in reality there are problems caused by the corners because one needs to

ask: What is a corner? What may be clear to a human as a corner is not necessarily obvious at a pixel level where a corner may appear as a quarter circle, for example. Coarse line fitting, however, results in poorly aligned lines but they are sufficient to say that a pixel group is 4-sided and not circular, which is the aim of this section. The results of using different distant limits on the shape in Figure 4.43 are shown in 4.44. The best value to chose was one that best distinguished between a rectangle and a circle (the robot target) without a significant deterioration of the shape. The value was set to 5 pixels.

Another application of this work was in the detection of building blocks for a wall building robot in Stuttgart University, Germany with details of the robot given in Pritschow et al, (1995). It was necessary to detect the outside edge of the block but it was known before hand what orientation the block had. In this case open pixel groups were used and it was found that coarse line fitting produced too many line segments at unusual angles. By reducing the minimum pixel to line distance, more line segments were created and they had a more accurate alignment. Only certain segments were then used to find a best line to approximate to the block boundary. In that application, the pixel to line distance limit was set to 3 pixels for best results.

A consequence of pixel linking is that the first pixel in a closed group could be anywhere. If it happens to be in the centre of one of the edges of a 4-sided shape, then the line fitting algorithm will fit an extra line such that one edge will be made up of two smaller line segments. Additional processing is required to detect this condition such that the two segments can be joined to form one line if they are sufficiently similar.

4.4.2.5 Orthogonal Diameters

The orthogonal diameters of an object are the diameters taken through the centroid of the object in the **x** and **y** directions as shown in Figure 4.45. These diameters will be used later to help locate neighbouring objects. Although the object's true diameters in the vertical and horizontal directions are distorted by perspective, and will not necessarily appear as the orthogonal diameters in the image, they do give a reasonable approximation if the distortion is not too severe. This is considerably easier than trying to calculate the true diameters, which would involve the use of vanishing points. The overall algorithm uses the orthogonal diameter data given at this stage for a first attempt at finding the directions of neighbours, as this gives a satisfactory approximation. If there is too much distortion then the true diameters would have to be used. The orthogonal diameters are found by taking the object's centroid and looking at each of the pixel co-ordinates. The pixels with

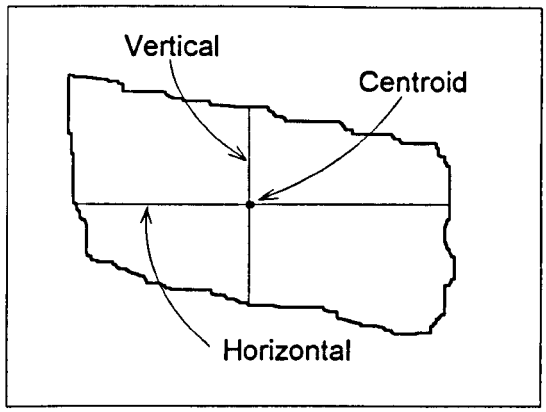


Figure 4.45 *Orthogonal Diameters*

the same **y** co-ordinate will give the horizontal diameter by finding the difference between the maximum **x** value and the minimum **x** value. Similarly the vertical diameter is found from pixels having the same **x** co-ordinates as the centroid.

4.4.2.6 Lower Level Object Ordering

The grouping together of objects by their relative levels above, gives the first indication of whether a feature is a likely match. If a feature consists of a single object containing another single object then it is easy to identify a specific object to compare with the CAD diagram. If, however, an object contains more than one lower level objects, then they must be sorted in such a way that they are compared with the correct potential CAD objects during matching. The calculation of object levels in section 4.4.2.2 creates a list of objects at a given level and the order in which these objects appear is essentially random. Several methods of ordering were considered but each was found to fail in one or two critical circumstances. The first was to order by linear position. With reference to Figure 4.46 there is an owner object containing five lower level objects where the number represents the object's position in the closed pixel group list. The points mark the centroid of each object and the diagram could typically be a window viewed from some angle. The ordering by linear position simply places the objects in order of their centroid's **x** value. In the example here, that would create an order of 13, 20, 47, 17, 16. For simple features this works but when, as in the example, two or more objects have similar **x** values then the ordering can very easily be changed if the feature is viewed from a different angle. This is shown in Figure 4.47 where the ordering now becomes 20, 13, 47, 16, 17. Clearly this is unsatisfactory as a small change in perspective dramatically changes information about the feature. If objects 20 and 13 were identical there would be no problem since they

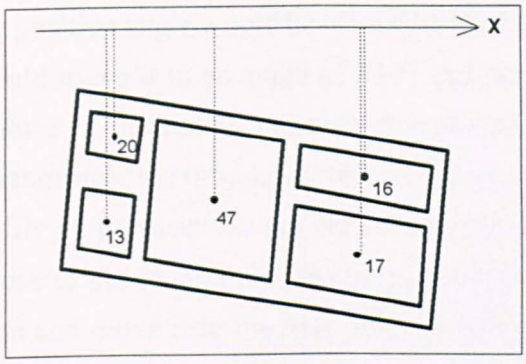


Figure 4.46 *Linear Position Ordering*

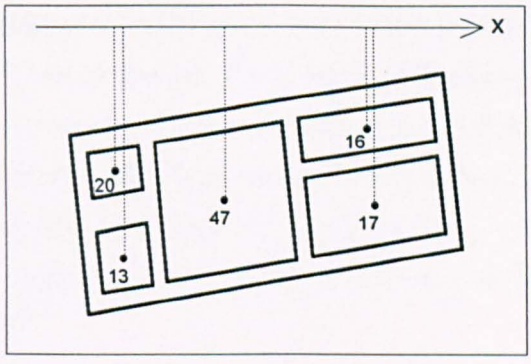


Figure 4.47 *...alternative Orientation*

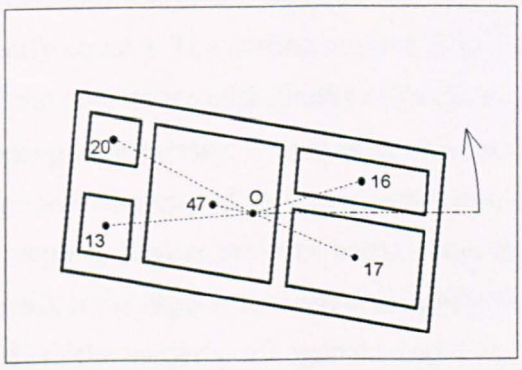


Figure 4.48 *Angular Position Ordering*

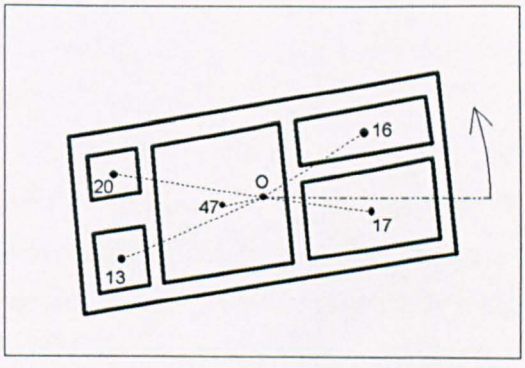


Figure 4.49 *...alternative Orientation*

would have the same parameters but, in reality, this can never be assumed and the feature identification processes could compare the wrong objects. This problem also occurs if the features are sorted in the **y** direction.

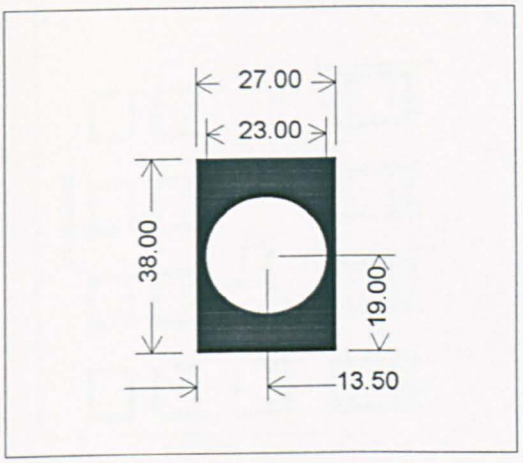
The next alternative looked at was to sort by angular position and distance. Figures 4.48 and 4.49 show the same features used in figures 4.46 and 4.47 respectively, but using angular ordering with the objects being sorted in increasing order from 0^{o}. The point of origin is taken as the centroid of the owner object marked with 'O'. This is clearly an improvement as the change in perspective has not changed the order of the objects which is 16, 20, 47, 13, 17. Unfortunately however, a similar problem occurs to the first method if two objects happen to have a very similar angle. The quantisation effects of the pixels (positions are not precise) plus the effects of noise can mean that if there is a change in position of only a couple of pixels of the object centroids, then the ordering of the objects may be reversed. It is preferable to have a system that is not so sensitive to minor changes. An added problem is that if the object is directly to the right of the owner centroid, then

its position angle would be 0^{o} and be the first object in the list. A change of only one pixel could move it to an angle of 359^{o} and place it last in the list. Also there is the physical nature of the radial approach that means that angles vary greatly with a small linear change near the origin, just like being near the North Pole. Although, in this case, the two different perspectives did not change the result, Figure 4.49 shows that point 17 is quite close to the 0^{o} line and, therefore, a small change in perspective could push it over this line and move it to the first position in the list.

The solution to these problems uses a combination of sorting methods and a 'fuzzy' approach to deciding positions. Instead of saying whether two things are equal or not equal, they are said to be not equal or nearly equal (where equal is just a special case of nearly equal). The sorting method first sorts objects by area placing the smallest first. If all the objects are of distinctly different sizes, then the ordering is complete and no further sorting is necessary. If as is often the case the areas are similar, in this case within 80%, then the subgroup of objects is sorted by centroid **x** co-ordinate with the lowest value first. Accepting similar areas as being equal avoids the area changes caused by perspective. Again, if the objects are well separated processing is complete but if the objects are within 10% of the owner's orthogonal width (see section 4.4.2.5) then they are treated as if they have the same **x** value. These sub- subgroups are then sorted in the **y** direction with the lowest first. Using Figure 4.46 as an illustration, sorting by area gives: 20, 13, 16, 17, 47. There is clearly no problem with object 47 as it is much larger than any of the others but objects 13 and 20 have similar areas so these then need to be sorted by **x** position. Again they have similar values so finally they need to be sorted by **y** position. Here there is a definite difference and the final sort order becomes 13, 20, 16, 17, 47. Applying the same algorithm to Figure 4.47 yields the same result as desired.

4.4.2.7 Identification of the Robot

Whereas the buildings under consideration can be of any shape and form, the robot design is under the control of the Department and its appearance can, therefore, be determined.. It would be too difficult to try to identify the mechanics of the robot using computer vision but a target placed on it can greatly simplify the task. The target can also act as a protective cover. When considering the target design, it should be made as simple as possible and should be unique so that it can not be confused with some other features on the building. Targets are found in numerous applications and come in a number of different forms. Clarke, (1994) gives a good analysis of different target types used in photogrammetry. However, these are small and used for high precision as demonstrated



Target	Object 0	Object 1
Centre X	27 / 2	As Object 0
Centre Y	38 / 2	As Object 0
Area	27 x 38	PI x 23 x 23 / 4
Level	1	0
Lower levels	1	0
List	1	-
Centre	Inside	Inside
Edges	4	6

Figure 4.50 *Model Robot Target*

Table 4.2 *Model Robot Target Description*

in Clarke and Wang, (1995) where sub-pixel accuracy is achieved, but this is not required in this application. Aerial work described by Brown, (1994) required the use of ground based targets which were located at a considerable distance and problems occurred with illumination when retroreflectors were used. Instead, circular targets were used which were painted with bright red fluorescent paint which made them stand out well when viewed through a red filter. Åmdal and Metronour, (1992) have created a measuring tool created with a number of pre-calibrated light emitting diodes (LEDs) to provide 3D measurement. But as one of the first stages of the location process requires the detection of edges in a black and white image, it was decided that the target itself should contain black and white features to produce maximum strength edges. This approach is adopted by Huang and Harley, (1990) who use a black circle containing a white circle for use in camera calibration. The target is identified by two closed objects having the same centroid. A slightly more complex version developed by v.d.Heuval, Kroon and Le Poole, (1992) also have a circular black and white target that has in addition, an outer ring made up of a 10-bit code allowing up to 1024 uniquely identifiable targets. However, for the application under consideration, the target should be as large as possible so that features are not lost due to the image resolution caused by the target lying at some distance. As it is intended to identify building features such as windows, it is reasonable to design a target of a similar size and the current robot dimensions allow for a target of 2400mm x 1200mm to be used. The design of the target should be such that it is as easy as possible to locate using computer vision. The initial work done here was with a model robot and it was decided to use a rectangle containing a circle as shown in Figure 4.50. This is a compromise between simplicity and uniqueness in that the rectangle with straight lines is easy to locate and only two shapes reduces the complexity. However, to

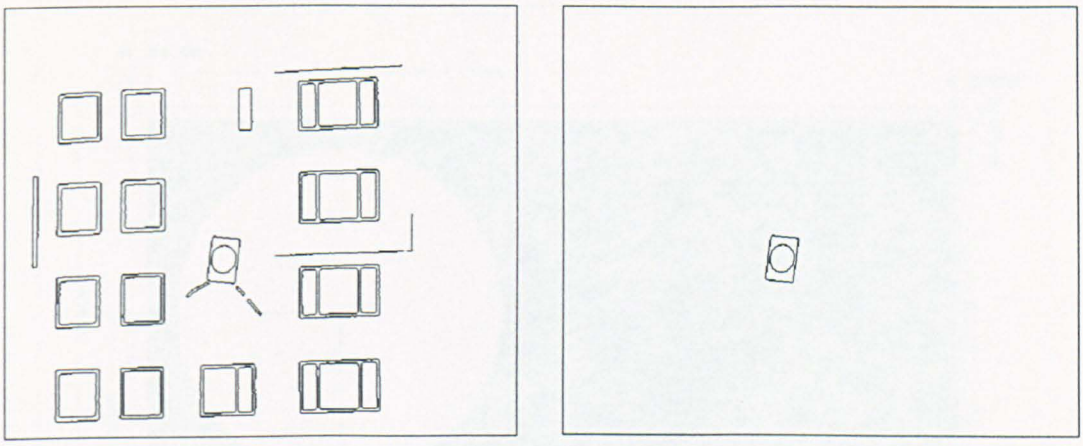


Figure 4.51 *Identification of Model Robot Target*

try to avoid possible confusion with a basic window frame, a circle is used as the inner shape. This type of shape combination has not been observed in windows and the only possible confusion with the circle may be with a satellite dish!

Using the data given in Figure 4.50, the information about the robot target can be entered into the software and this is used to match against all the objects in the closed object list. The model robot target is made up of two objects, a rectangle containing a circle with the same centre position, which can be specified as given in Table 4.2.

Although Object 1 is a circle, it has been given an edge count of 6, since to a very coarse approximation, it is a hexagon. To improve reliability, a shape detector should be used to look for four sided shapes likely to be rectangles and to look for ellipses. This would take extra processing time and the aim here is to see if the robot target can be found in as simple a way as possible. The following algorithm is used to see if an image feature matches the robot target:

1. For each object in the closed object list, see if the level value is equal to the Object 0 level.
2. If levels are equal (1 in this case) set Object 0 score to 10.
3. If the centres of the list object and Object 0 are both inside, then add 10 to the Object 0 score.
4. Add 10 - (| list edge count - Object 0 edge count |) to Object 0 score.

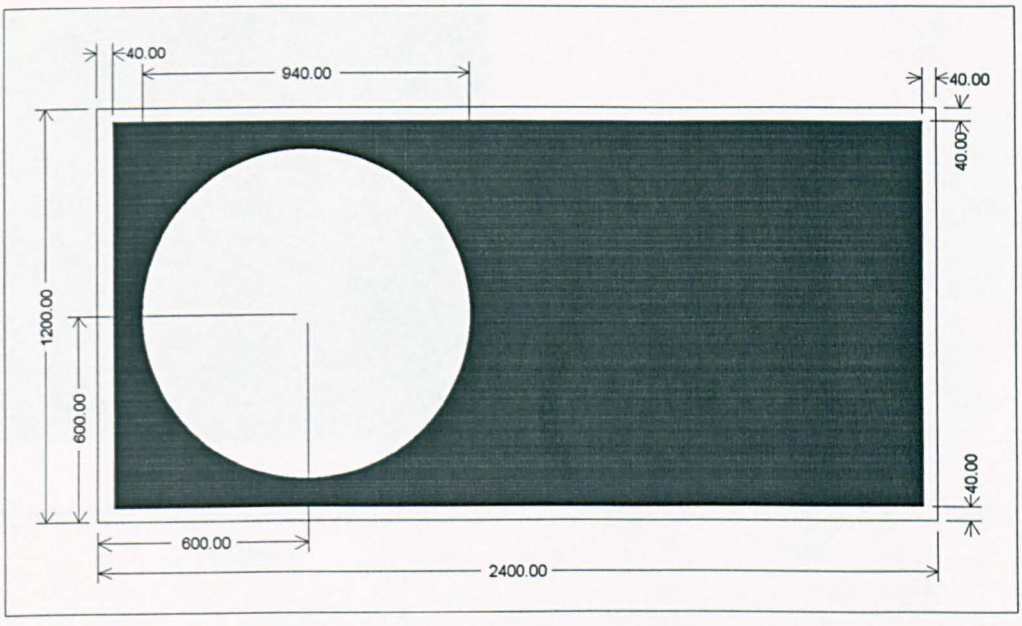


Figure 4.52 *Inspection Robot Target*

5. If the Object 0 score is greater than or equal to 25 check the lower level object.
6. Do steps 2, 3 and 4 with the lower list object and Object 1 (instead of object 0) in the table.
7. Calculate the list and robot area ratios by dividing the lower object area into the upper object area.
8. Divide the list ratio by the robot target ratio and if greater than one take the reciprocal to ensure the ratio is less than or equal 1.0.
9. Add 10 * area ratio to the Object 2 score.
10. Calculate the total score from Object 0 score + Object 1 score and if it is greater than the pervious highest score, store it along with the reference to the object in the closed object list.

The maximum possible score in the above algorithm is 70, 30 from Object 0 and 40 from Object 1. A limit is set to 60 below which a feature is said not to be the robot. The scoring is fairly arbitrary but takes into account the strength of the visual impact of the object. For example, if the centre of an object was outside rather than inside, its visual appearance would be significantly different and is unlikely to be part of the feature that is being

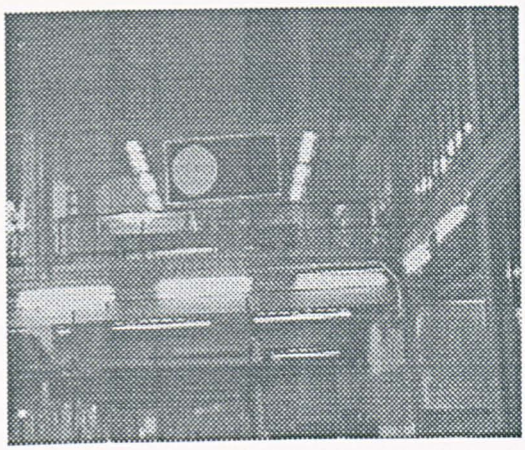


Figure 4.53 *Life size Mock-up Robot Target*

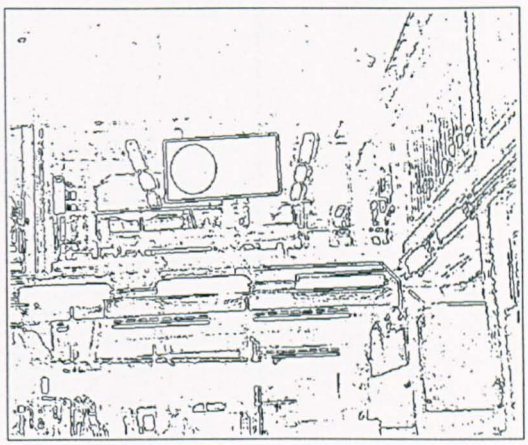


Figure 4.54 *Image after Edge Detection*

looked for and hence the score of 10 is used for significant characteristics. On the other hand, the ratio of areas between objects can vary considerably before it can be said that two objects are not the same and therefore 10 times the ratio of area ratios is used. If the areas are the same, the ratios would be 1 giving a maximum score of 10 for this characteristic. Ratios, rather than the absolute areas are used by v.d.Merwe and Rüther, (1994) who look for "size as well as type and number of corners" to take into account the distortions caused by perspective.

Figure 4.51 shows the robot target being identified from the complete set of closed groups. The top level object representing the robot target is tagged to indicate that it is the robot, so that it is ignored during future processing when building features are searched for.

There is a problem with this type of target in that if the target is sitting on a building that has a similar gray level (black in this case) then it is not possible to pick up the outside edge of the target. Similarly, as was often found during experiments, if a light was used that cast a shadow, it was again not possible to find the correct edge of the target as the edge detector would follow the target and the shadow. Consequently, a new target has been designed for the real robot as shown in Figure 4.52. This can be considered as a target within a target. The original target, which is similar to the model target, is contained within an additional white border. The initial search tries to locate a match with a feature containing three levels instead of two, namely a rectangle containing another rectangle containing a circle. If for reasons just mentioned, the outer object is confused in the image and can not be found, then the inner two level object is searched for. By placing the third

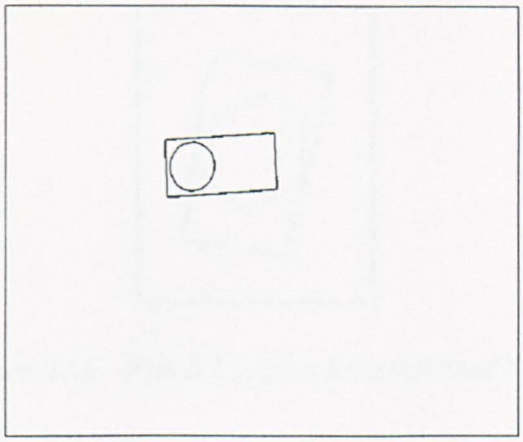


Figure 4.55 *Identified Target*

object round the other two the likelihood is increased that the inner target will be located, as it is unaffected by the background. Also placing the circle to one side will assist any future work on orientation.

A full size target has been constructed that can be used separately from the robot, since it is only necessary to see the target and not the robot. Should the robot be unavailable then the target can still be used for experimentation. From a construction point of view, and bearing in mind that the target should cause no visual problems, it must be painted with a matt (satin) finish to minimise the possibilities of reflections from the target. Due to the amount of time and effort required to satisfy safety issues and to have the wooden target risk assessed, it was decided ultimately not to hang the target on the exterior of the University building. Instead, Figure 4.53 shows the full size mock-up target being suspended in the heavy structures laboratory where lighting is poor and a considerable amount of visible clutter present. Figure 4.54 shows the result of edge detection with a statistical threshold of 16 and Figure 4.55 finally shows the successfully identified target. This shows that the overall target design worked, especially with the use of an outer border. Close inspection of Figure 4.54 shows that the outer edge is incomplete due to background noise leaving the inner part of the target without any defects at all. In this image, the target height is about 80 pixels giving a single pixel resolution of approximately 15mm. Closer inspection of Figure 4.54 reveals that the outer part of the target is separated by only one or two pixels indicating that it is at the limit of resolution even though the target is amply large enough to be seen. A future target would benefit from having a smaller circle and a thicker outer border as it is the distances between these edges which determine whether the target will be detected.

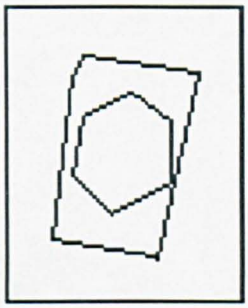


Figure 4.56 *Fitted Lines to Model Robot Target*

4.4.2.8 Shape Detection

Shape detection is a very important part of object recognition as it forms the main description of an object. Two objects may have very similar parameters such as area or that the centroids are inside their boundary. However, when it can be seen that one object is square and the other is a circle, then the distinction is made clearer. But the problem is what is the actual difference between a square and a circle or, more generally, a rectangle and an ellipse? A human can see the difference but the problem here is again to get the computer to 'see'. Perhaps the simplest difference is that the rectangle contains four straight lines whereas as an ellipse has no straight lines. Other shapes, such as triangles, will consist of only three lines, but discussion will be limited to rectangles and ellipses as these are the most likely shapes to be searched for in this application. Whilst having observed that an ellipse contains no straight lines, it actually consists of a large number of very short lines, namely the lines joining adjacent pixels. This, however, is too fine a distinction to use since the number of lines generated will equal the number of pixels forming the object boundary and, again, there is no way of identifying the shape.

To a first approximation, line fitting (see section 4.4.2.4) can be used to generate a polygon representing an ellipse. For example, changing from a hexagon to an octagon and so on gets ever closer to a circle. The result is that the simplest way of distinguishing between a rectangle and an ellipse is to count the number of fitted lines. Figure 4.56 shows the straight line approximation to the objects that make up the model robot target. The ellipse is made up of six, more or less equal line segments and the rectangle is in fact made up of five segments, a short one being in the upper left corner. This is caused by the original edge not being perfectly straight. Although line count is used at present, it can be seen from this example that there is only a difference of one line separating the rectangle from the ellipse and, therefore, a small error could produce a mismatch. More robust methods need to be found.

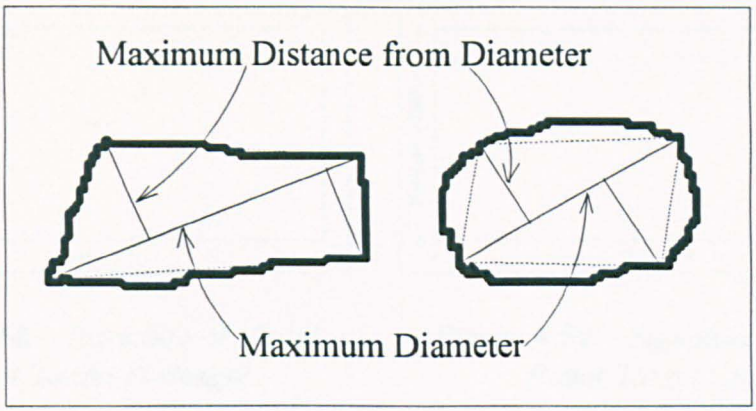


Figure 4.57 *A Method of Finding Four Sided Shapes*

When a specific shape is being searched for, particular methods can be used to see if an object conforms to that shape. A typical situation here is to search for four sided shapes, these being rectangles distorted by perspective. Instead of using just the lines characteristic given above, the four corners can be looked for and then these corners joined by lines. If all the object pixels lie on or near the lines, the object can be said to be four sided. First, however, it is necessary to decide what is meant by a corner. This is a question of scale, since at high magnification, a corner, such as a piece of paper, will closely resemble a quarter of a circle. The same is true of the groups of pixels since a magnified image is effectively being worked with. For a quadrilateral, however, two of the four corners are at points furthest from the centroid. If the largest diameter is found it is assumed that the two points corresponding to the diameter are two of the corners. The other two corners are found using the same method as the line fitting algorithm (see section 4.4.2.4). Figure 4.57 shows the results for a quadrilateral and elliptical shape. As can be seen there is a close correspondence between the line joining the located corners and the pixels whereas, for the ellipse there is hardly any correspondence. There is, however, a significant problem with this method in that the diagram shows that the two points searched for, after the diameter is found, lie on opposite sides of the diameter line. If the shape happened to be a trapezium, a valid four sided shape, then the maximum diameter would lie along the longest edge. It then becomes a difficult problem to decide where the other two corners are located, resulting in a considerable increase in computational effort. The most reliable solution is to look at the object signatures. A method given in Sonka, Hlavec and Boyle, (1994: 203-205) suggests that the distance of a point normal to the current section of the object is found. This can be very susceptible to noise. An alternative is to plot the position vector of every pixel in the group with respect to the centroid, that is, plot a graph of distance (radius) and angle. Two plots are shown in Figure 4.58 and Figure 4.59

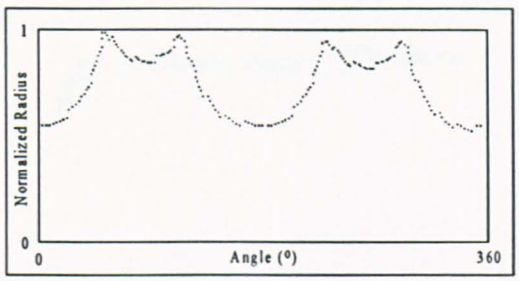


Figure 4.58 *Signature of Model Robot Target Rectangle*

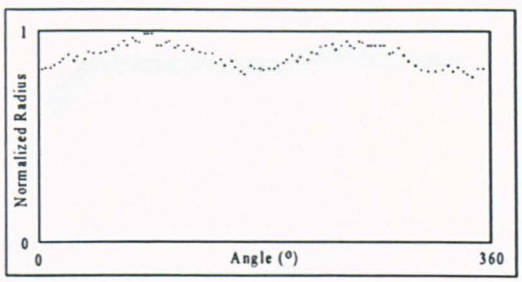


Figure 4.59 *Signature of Model Robot Target Circle*

representing again the model robot target. There is a clear difference between the two shapes with the four sided shape being characterised by four peaks and the ellipse is characterised by not having peaks. The peaks correspond to the four corners with the 'U' shaped curves corresponding to the lines. If the mathematical functions for the curves can be found, then the peaks correspond to discontinuities where one function stops and the other one starts. Although this seems, at first sight, a better way to detect shapes, it does require considerable extra processing to determine whether or not peaks are actually present. An ellipse of large eccentricity will have a similar signature to a narrow rectangle with slightly rounded vertices. Noise is also introduced due to pixel position quantisation and closer inspection of Figure 4.58 shows that small steps occur along the curve since image lines are actually made up of a combination of horizontal and vertical lines segments. A processing approach to take this into account, that increases reliability while keeping times to a minimum, would be to use the 'line approximation and count method' (see section 4.4.2.4) to pick up significant differences. If there is a possibility of confusion, then the shape signature process is used to refine the result.

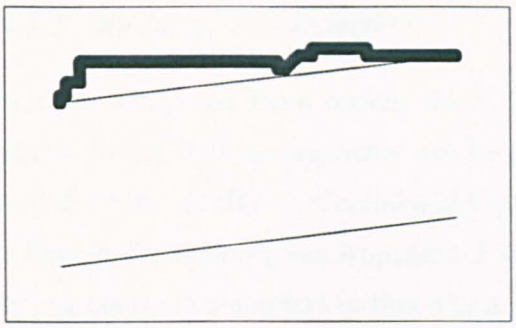


Figure 4.60 *Coarse Line Fitting*

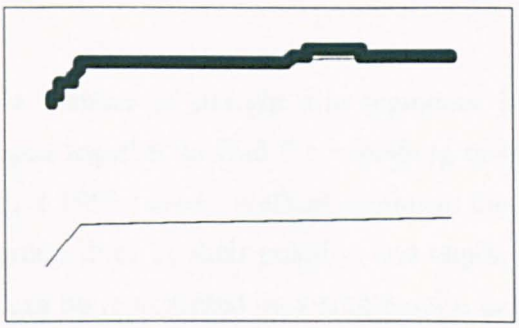


Figure 4.61 *Fine Line Fitting*

4.4.3 Open Group Processing

The pixel linking process described in section 4.4.1 first creates a list of open groups (groups with two ends) before the closed group list can be generated and is, therefore, a by-product of closed group processing. The question is, since this information exists, can it be used in any way? Recalling Figure 4.26, it is clear that open groups tend to form lines at the edges of the building and at the gaps between the floors and vertical structures, particularly if the building is constructed from panels. These lines also tend to lie in the horizontal and vertical directions.

4.4.3.1 Line Fitting

Section 4.4.3.2 deals with vanishing points, but before they can be found, the open groups of pixels need to have lines fitted to them. This is essentially identical to the closed group line fitting described in section 4.4.2.4 but the fitting accuracy needs to be different. With the closed groups, the line fitting is fairly coarse but for the open groups, the lines need to be more accurate as their orientation angles are more critical. A small deviation in the line can have a large effect on the position of a vanishing point, especially if the lines are nearly parallel. Work carried out on block identification at Stuttgart University (Paterson, 1995) has shown that small changes in the line fitting parameters are significant. For the closed group processing, the maximum deviation of a pixel from the line is 5 pixels and this has been changed for 2 for the open groups. The example in Figure 4.60 shows how coarse line fitting has ignored a small deviation in the pixels and has resulted in a line with a different orientation to most of the pixels. Figure 4.61 shows that by using fine line fitting, a better approximation is achieved at the expense of creating more, shorter line segments.

4.4.3.2 Vanishing Point Detection

Once the image has been broken down into a number of straight line segments, it is possible to see if these segments can be grouped together to find the vanishing points. Work done by Straforini, Coelho and Campani, (1990) uses a method similar to that of the Hough Transform (see Appendix 1) to group lines by their position and angle. By plotting the line parameters in this way a line can be represented by a single point in the plot space. The plot space is divided up into different regions and the number of lines falling in these regions determine the type of vanishing points that are present as there may be different numbers of vanishing points. If there is a strong likelihood of a vanishing point, then a least squares method is used to find the position of the point that lies closest to all the selected line segments. However, closer inspection of the work done by Straforini et al indicates that they have concentrated on internal scenes which are likely to have a vanishing point somewhere near the centre of the image. Our scenes on the other hand deal with external views of buildings, usually of just one side, which yield different families of vanishing points. Tai et al, (1993) also tend to concentrate on vanishing points near the centre of a scene and their algorithm looks at the positions of line intersections and ignores all those that lie outside a region twice the linear size of the original image. Most of our building images have vanishing points outside this boundary. Having tried Straforini's method it was found that the parameter space was not suitably divided, the equations require a slight alteration and that nearly parallel lines, which often occur in building images, require special attention. In a later publication, Straforini, Coelho and Campani, (1993) extend their work to outdoor scenes with apparent good results, but they imply that this method is not complete by stating that a different partitioning of the parameter space (see later) is required if the camera is not parallel to the floor. They do not however give the partitioning information and most of the scenes presented have vanishing points that are again near the centre of the image.

Consider one of the line segments in the image. Its position and orientation can be specified by the following equation:

$$\mathbf{px + qy + r = 0} \qquad [4.1]$$

It is not possible to represent this line simply as the three parameters require a three dimensional parameter space so by dividing through by **q**, we obtain the more familiar equation of the line with only two parameters:

$$\mathbf{px/q + y + r/q = 0} \qquad [4.2]$$

which is equivalent to:

$$y = mx + c \qquad [\,4.3\,]$$

The two parameters **m** and **c** could now be plotted such that the line is represented by a point but problems occur for near vertical lines when **m** and **c** approach infinity thus requiring an infinite, and unavailable plot space. If the angle of the line is used instead of plotting the gradient **m**, then all the lines fall in the range 0^o to 360^o. The angle is found from:

$$\theta = \tan^{-1}(\,m\,) \text{ or } \theta = \tan^{-1}(\,-p / q\,) \qquad [\,4.4\,]$$

If the values of the parameters in [4.1] are modified such that:

$$p^2 + q^2 = 1 \qquad [\,4.5\,]$$

then the modified line offset parameter **r** lies in the range **-C** to **+C** where:

$$C = \sqrt{(\,\text{image width}^2 + \text{image height}^2\,)} \qquad [\,4.6\,]$$

In order to find how the parameters are modified, rewrite [4.3] in the form of [4.1] to get:

$$mx - y + c = 0 \qquad [\,4.7\,]$$

If this is divided by $\sqrt{(\,m^2 + 1\,)}$, then condition [4.5] will be satisfied and the line equation now becomes:

$$\frac{mx}{\sqrt{(m^2+1)}} - \frac{y}{\sqrt{(m^2+1)}} + \frac{c}{\sqrt{(m^2+1)}} = 0 \qquad [\,4.8\,]$$

with the **r** value calculated from:

$$r = \frac{y}{\sqrt{(m^2+1)}} - \frac{mx}{\sqrt{(m^2+1)}} \qquad [\,4.9\,]$$

One of the three parameters has been conveniently eliminated, with the other two lying within usable limits thus allowing all lines to be represented by a point in a 2D parameter space. Since a line is unchanged after a 180^o rotation, the limits for the horizontal range are reduced to 0^o to 180^o. The **x,y** values are referenced to the centre of the image. Figure 4.62 shows an example of the lines used and figure 4.63 shows their corresponding points in plot space.

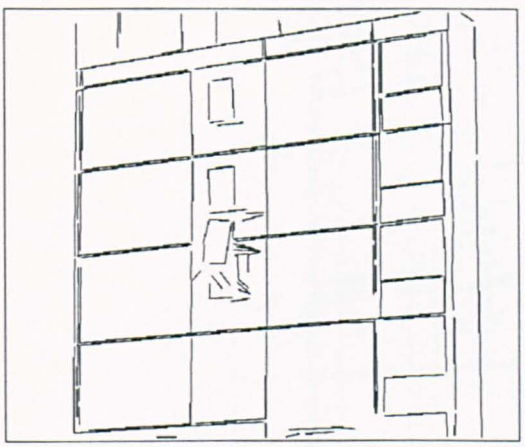


Figure 4.62 *Lines used for Vanishing Point Detection*

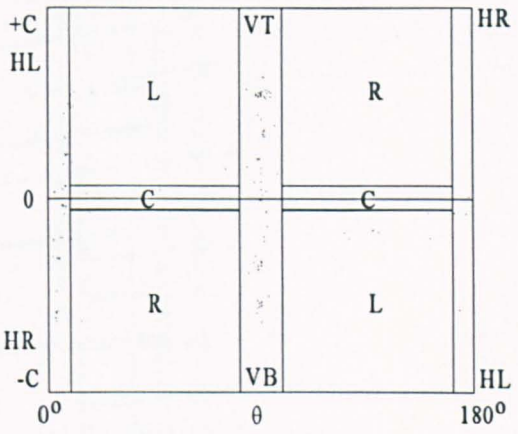


Figure 4.63 *Line Paremeter Space*

The plot space has been divided up into a number of regions slightly different from Straforini's partitioning as follows:

Hx	Horizontal lines	Width = 180^{o} / 10
Vx	Vertical lines	Width = 180^{o} / 10
L	Lines converging to the left	
R	Lines converging to the right	
C	Lines that pass through the centre of the image	Height = **C** / 8

In Addition, the horizontal and vertical regions are further subdivided into **HL** and **HR** representing horizontal lines that converge left and right respectively and **VT** and **VB** representing vertical lines that converge to the top and bottom respectively. To determine if there is a possibility of a vanishing point, a total of the number of points falling within a region is found and compared with the total number of lines present. The values **nh**, **nv**, **nl**, **nr** and **nc** are used for the regions and after some experimentation, it was found that a possible vanishing point exists if **nx** / (**nh** + **nv** + **nl** + **nr** + **nc**) is greater than 0.150. Since a line could converge to more than one vanishing point, certain assumptions have to be made. Straforini et al state that the camera is nearly parallel to the floor but in our case it may well have some pitch, so the limitation imposed here is that the camera has a limited amount of roll - no more than ± 9^{o}. (Roll affects the position of the **H** and **V** groups and additional processing could be done to re-establish them).

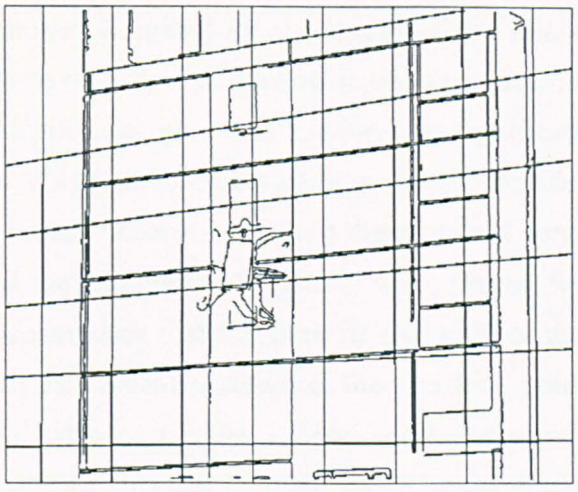


Figure 4.64 *Vanishing Point Grid*

If a vanishing point falls within the **L**, **R** or **C** groups, then there is a fair amount of convergence in the lines and a least squares method is used to locate the point nearest to all the lines within the group. However, the nature of our images tends to lead to lines mainly falling in the horizontal and vertical groups. Assume for the moment that all the lines are actually parallel. The slight inaccuracies caused by the processing will mean that the actual lines will have slight deviations with some converging say, to the left for the horizontal group and some converging to the right, albeit at some distance. If the least squares method is used to try and locate the near infinite vanishing point, a totally false result will occur as the left and right convergence tend to cancel each other out. For this reason the **HL**, **HR**, **VT** and **VB** groups were introduced. Depending on the position of say the horizontal vanishing point, a given line in the horizontal group could either converge to the left or the right and this ambiguity can only be resolved by looking at the other lines in the group. Again, by experimentation, it was found that if a least squares convergence is found for the **HL**, **HR** and both groups together, then a perceived convergence exists when all the signs of the three **x** co-ordinates are the same and the vanishing point is taken from the combined subgroups. Similarly for the vertical group if all the signs of the **y** values are the same then the least squares is used on the combined **VT** and **VB** groups. If the signs are different, then the lines are considered to be parallel and have an infinite vanishing point. The angle of the lines is calculated by finding the mean of the theta values within the group. A pseudo vanishing point is then created by saying that it is at a distance of 30000 (near maximum value for an integer in the C language) from the centre of the image and the **x** and **y** values found accordingly. Since the longer lines tend to be the most significant, the line parameters for the least squares

and mean theta are actually weighted by the length of the line with small lines contributing less. Lines of less than 15 pixels are not used. It was decided to only use the lines from the open groups of pixels as these tend to arise from significant parts of the building such as its outer edge. Using the closed groups was also considered, however, the resulting lines tend to be much shorter and would therefore not contribute significantly to the vanishing points as the lengths are weighted. Also, shorter line segments are more prone to quantisation errors such that the error in the angle of the fitted line would be greater and consequently reduces the accuracy of the vanishing point. In a vanishing point detector by Brillault-O'Mahony, (1991) which uses the Canny edge detector then straight line extraction, it is stated that in a 256 x 256 image, line segments less than 15 pixels long "are very numerous and do not provide meaningful information". This would translate into roughly 30 pixel lines in our images but it was felt that in building images, these lines are still valid. As we are only interested in one plane of the building, the two strongest vanishing points are taken as representing that plane. Figure 4.64 shows a grid of lines converging on the two main vanishing points laid over the open group image giving an acceptable level of agreement.

Since the vanishing points are being used to determine the likely direction in which to search for a neighbouring feature, it can be argued whether it is necessary to find the global vanishing points of the whole image. If a given feature is rectangular, then the approximate orthogonal directions (or those of the local vanishing points) can be inferred from the two pairs of parallel lines that make up the rectangle. This naturally only works for rectangles, however, and other shapes would cause a problem, although most features on buildings are indeed made up of rectangles. Using local vanishing points would also overcome problems induced by severe lens distortion if, for example, a very wide angle lens was used. All the straight lines in the image, especially the long ones, would in fact be curves with no well defined vanishing point. The distortion of an individual feature would, however, be considerable less thus still giving a good approximation to the local orthogonal directions, but this situation is unlikely to be encountered.

It is interesting to note that a precise vanishing point is not necessary for perception. Figure 4.65 taken from Ernst, (1985: 44) has no single vanishing point in the centre of the image due to the way it was drawn and yet there is no confusion in the perception of the scene. It is still possible to say which directions are forward, backward, left and right even though the lines drawn over the picture are far from converging.

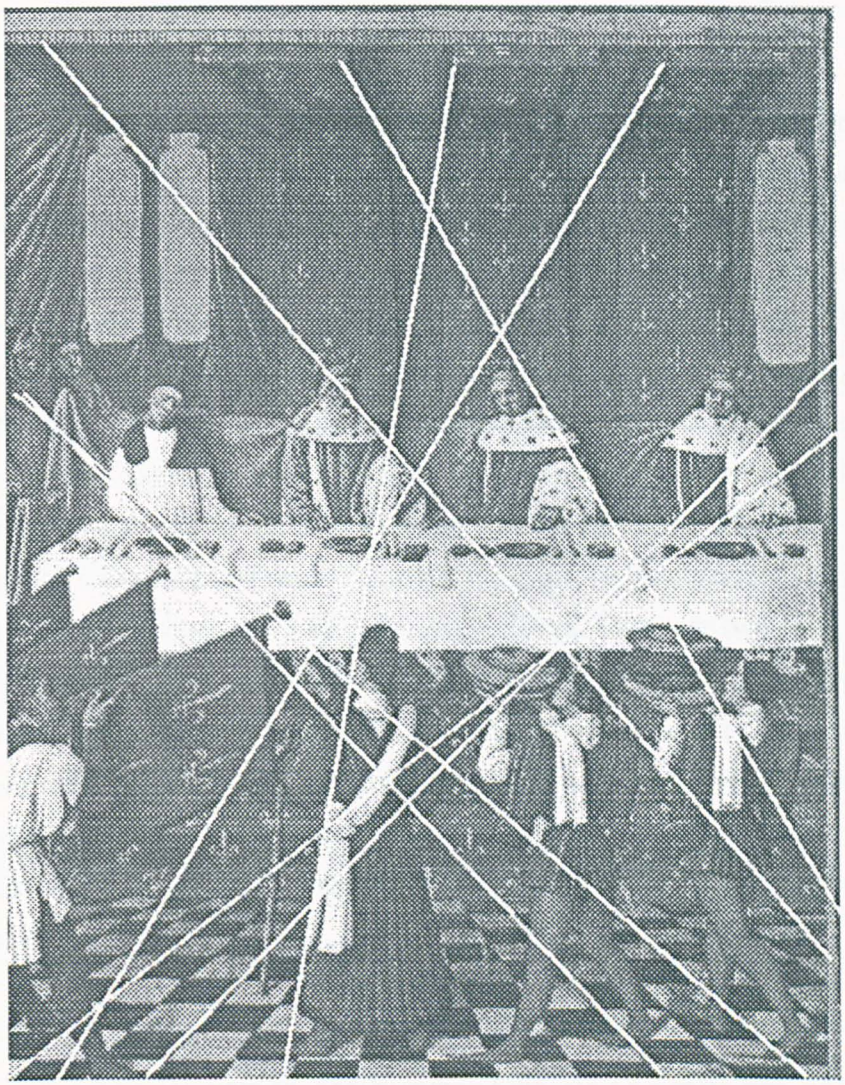


Figure 4.65 *Jean Fouquet, The Royal Banquet*

4.5 CAD Representation of Data

CAD diagrams or models of buildings are used in this application to find the actual dimensions and positions of features. Although it was stated at the beginning that no calibration or measurements are required, it would be more accurate so say that they are not required at the time of operation. The measurements are necessary but have normally been done at some time during the design or after some survey. This does assume, however, that diagrams exist for the buildings but many of those needing inspection will be a few decades old and were certainly designed before computer tools were available. It is then necessary to redraw the building as a computer diagram and of course, this will provide an additional cost which must be taken into consideration. However, once drawn, the information will always be readily available and can be incorporated into reports very easily so overall there should be very little cost involved. There are two ways in which the data can be entered into a drawing package. Firstly, most of the buildings in question will have drawings and these can be digitised. Secondly, if drawings are not available, then a survey using existing photogrammetric techniques can be carried out. This is the more complex approach but there are many companies and departments that can do just this. Once the data have been entered, they can be saved in a suitable file format which can be read by the robot location software.

The aim here is to read in the appropriate information about the face of the building being inspected and create an internal representation of the face that can be used by the software. There are many CAD packages available on the market ranging from under £100 such as TurboCAD (used for many of the drawings here) to ones like AutoCAD (commonly used in industry) costing several thousands of pounds. The application software needs to be able to read the various files produced by these packages and fortunately most if not all of these packages can create files in the Data eXchange Format or .DXF. This format, a summary of which is given in Appendix 3, is the one used here. A process very similar to that used for the image, builds up various lists of data starting from the drawing primitives such as lines. The lines are grouped together to form objects, the objects are grouped to form features as shown in figures 4.66 and 4.67 and the features are used to form a map called the 'Orthomap'. The main problem to overcome is to distinguish between wanted and unwanted data, for example, a dimension marked on the drawing will also consist of lines that are not part of the building diagram. A feature of .DXF files and just about all drawing applications in general, is the facility to either use different 'pen' colours or different layers to draw the entities. This allows, say pipes to be

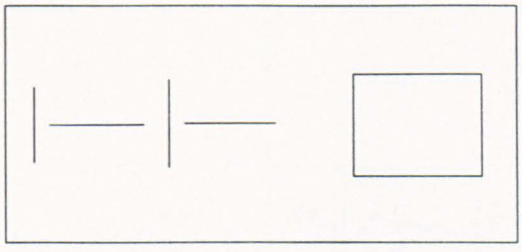


Figure 4.66 *CAD Diagram Primitives to Closed Objects*

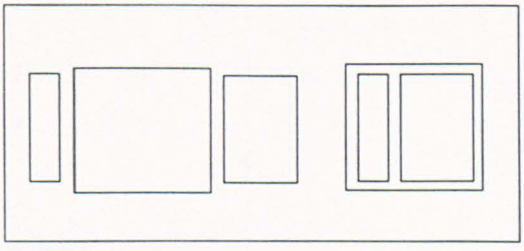


Figure 4.67 *Building Features from Objects*

drawn on one layer and electrical cables to be drawn on another making the various parts of the drawing clearer and easier to maintain. As it is not known what information will be supplied on a drawing, there will have to be a specification that states that all the parts of the diagram that are required for the robot location software be drawn on a specific layer. Also, if different file formats are used, then extra 'filters' would need to be written and linked into the software to convert the supplied file to the internal data format.

It should be noted, that when .DXF files are used, the origin of the co-ordinate system is the bottom left corner of the paper so when the CAD diagram has been loaded, it must be displayed and the operator asked to select the (0,0) point. Also the units of measurement actually stored in the file are always in inches.

4.5.1 The CAD Object List

This list contains simple closed objects such as rectangles and circles and is created from the list of primitives in the CAD file. As the file is loaded, a search is made for lines or other basic shapes which are then stored in memory. The object list created is essentially the same as that created for the closed groups of the image in section 4.4.2. However, the

Feature	Data			
ID	8			
Centre	x = 5623	y = 11500		
Width	1500			
Height	800			
Neighbours	E = No N'bor	N = 4	S = 12	W = 7
Relative Distance	-	2.5	2.5	1.23

Table 4.3 *Typical Entry in the CAD Orthomap*

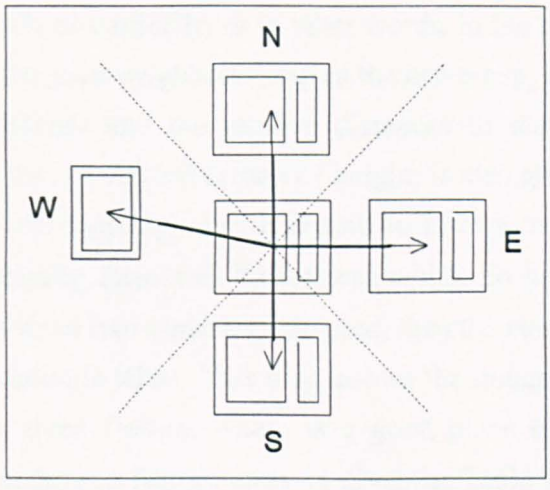


Figure 4.68 *A Feature with Four Neighbours*

process is somewhat easier since we already have line information. The lines are searched to see if any of the ends join and whether they can be formed into a closed object. The same statistics are calculated as those of the image objects although the methods used will be different. Geometry rather than pixel positions can be used far more effectively to calculate centroid, diameter, line count and some shape information which will already be supplied by the CAD file.

4.5.2 The CAD Feature List

The CAD feature list contains groups of related objects. Just as the image objects contain other objects, a CAD object belongs to another if it is contained by the other one, so a feature is complete when the outermost object cannot be surrounded by any other object. Once a feature has been created, its internal objects are sorted as described in section 4.4.2.6 and then it is compared with the current features to see if it is the same or not. If it is different, it is given a new feature identification, the first one having an ID of 1 and so on. If the feature is the same as another one, it is just given the same ID.

4.5.3 The CAD Orthomap

Once the feature list has been created, the features need to be ordered in some way. The method adopted here is take a feature and look for its nearest neighbour in each of the four directions: east, north, south and west as shown in Figure 4.68. As tower blocks are usually constructed with near identical floors, the major features such as windows will

either line up horizontally or vertically, or in other words, in the orthogonal directions. A map of features and orthogonal neighbours, called the orthomap, is created such that each feature has four neighbours and the relative distances to these neighbours, that is, horizontal distance / width or vertical distance / height, is also given. Even if a feature is located at the edge of the building, then it is said to have a 'no-neighbour' in the associated direction. Naturally there will be features which do not lie exactly in the orthogonal directions but since four quadrants are used, then the nearest feature lying within $\pm 45^{o}$ of the main direction is taken. This map allows the image features to be identified and to say for a given feature, where is a good place to search for a possible neighbour. A typical orthomap feature entry is given in Table 4.3 and an orthomap is shown in Figure 4.69.

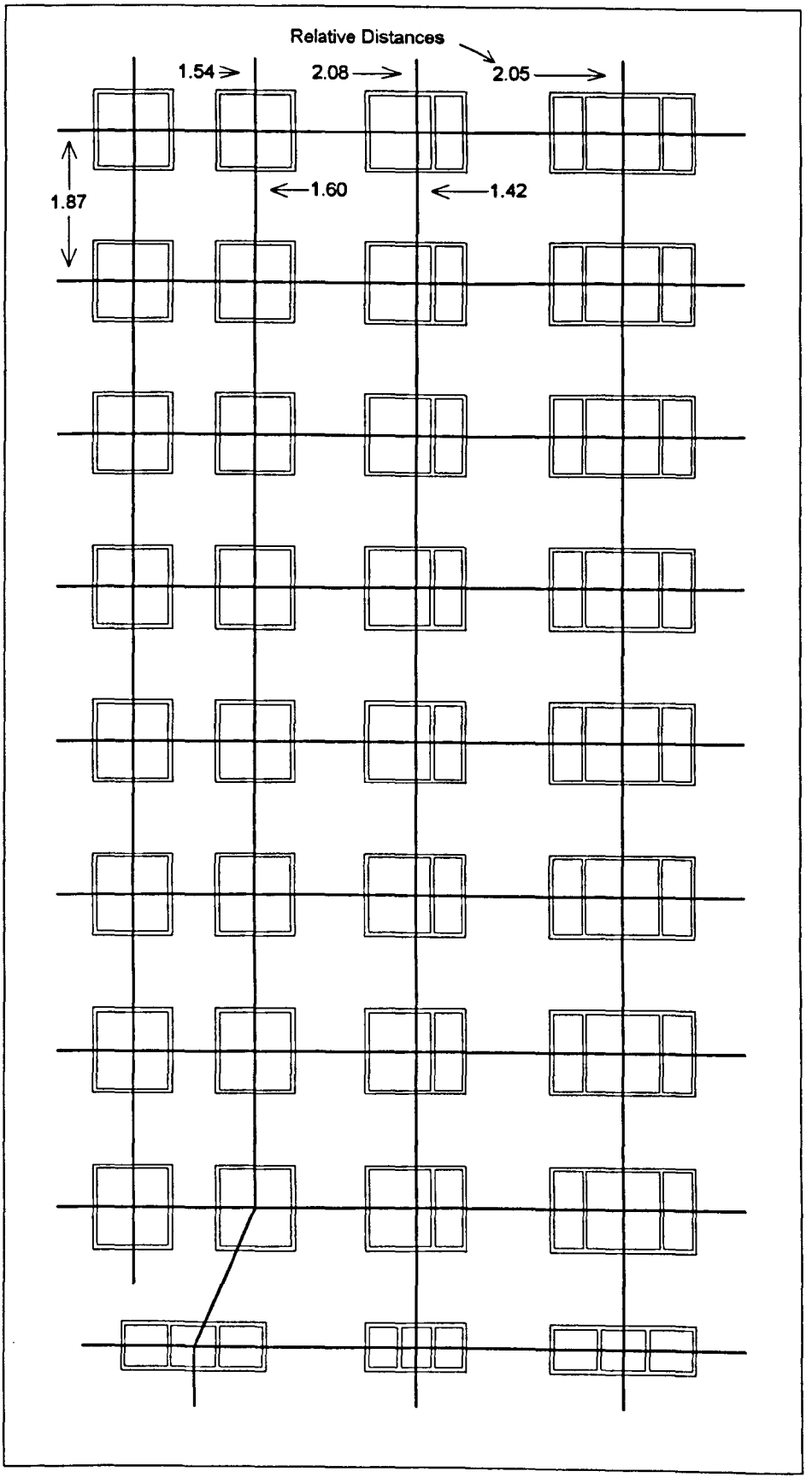


Figure 4.69 *Typical CAD Orthomap*

4.6 The Matching and Location Process

Up to this point, a large amount of information has been gathered about various objects in the image and CAD diagram. This information is now grouped and compared to try to identify image features from the CAD feature descriptions and then identify the specific position of an image feature in the CAD diagram. Once the features have been identified as having a particular ID, they are then joined together as a group of nearest neighbours. This image feature map is then compared to the feature map created from the CAD diagram so that an image feature can be matched against a specific CAD feature thus enabling the image co-ordinates and the real world co-ordinates of the features to be combined. Four image features are then found to give four image/CAD co-ordinate pairs from which the image to CAD model mapping can be calculated. Once the mapping is known, a point from the robot target in the image can be mapped directly onto the CAD model giving the robot's actual position.

4.6.1 Identification of Image Feature Types

Before any mapping can be carried out, individual features in the image need to be identified. It is not necessary to find their real world positions, but merely to specify that a given feature is a particular window frame or panel, etc. The process is essentially the same as that used for identifying the robot target in section 4.4.2.7, but this time the feature description has been generated from the CAD information in section 4.5.2 rather than being pre-defined. Also the complexity of some of the features requires further processing. Recalling that all the objects in the CAD diagram are in the CAD object list and the image objects are in the image object list, then the algorithm for identifying a feature is as follows:

```
1. For each object in the image object list, do:
   2. If the object is a top level object ( surrounds
      other objects and must be a feature ) AND is not
      the robot target
      3. If the object is not empty, find the ID of the image
         feature then do subroutine FIND_FEATURE_ID.
         ( Assume that single objects are not features ).
```

FIND_FEATURE_ID

```
1. For each object in the CAD object list do:
   2. Clear CAD object's top level score.
```

3. If the number of lower, image level- and CAD level objects is the same then:

    4. Do subroutine *SCOREOBJECT* for the top level image object and store the score with the current top level CAD object.

    5. For each of the top level CAD objects with a score do:

        6. For each of the CAD object's lower level objects:

            7. If the CAD- and image lower level count is the same:

                8. Set Total_score = 0 and for each corresponding lower level object do:

                    9. Set Total_score = Total_score + result of *SCOREOBJECT* for current lower level object.

        10. Add Total_score to the top level score for the current CAD object.

11. Find the CAD object with the highest score.

12. If Total_score is within 90% of the maximum possible score, mark the image object with the CAD feature's ID.

SCOREOBJECT

1. Set score = 0.

2. Score = score + 10 - abs(Number of image Object edges - Number of CAD object edges).

3. If the position of the centroid flags agree, then Score = Score + 10.

4. If both objects are empty with no lower levels, then Score = Score + 10 else Score = 10 * min(Number of image object lower levels : Number of CAD object lower levels). Where the min function represents the minimum ratio of the two values and is less than or equal to 1.

5. If the objects are both top level objects with no owner object, then Score = Score + 10
else
Score = 10 * min(min(Image object area : Image object owner area) : min(CAD object area : CAD object owner area)).

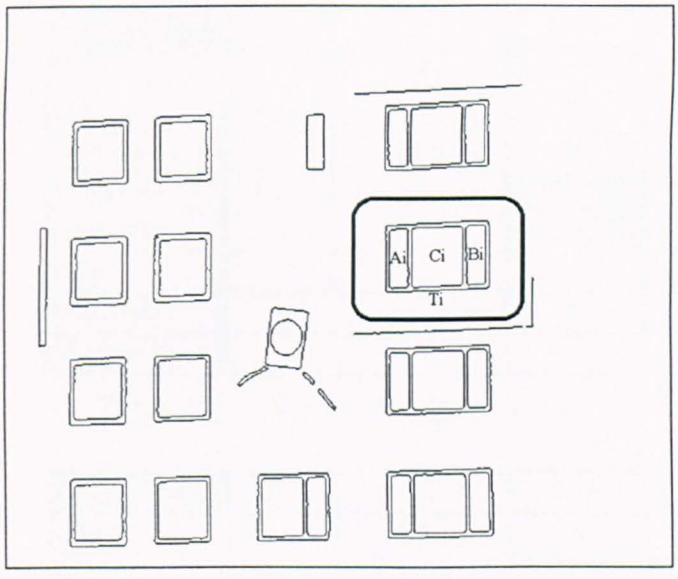


Figure 4.70 *Selected Image Feature*

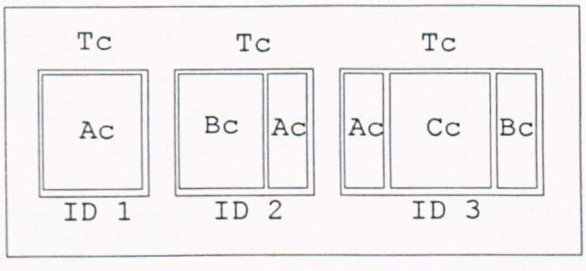


Figure 4.71 *CAD Features*

```
6. If Score is greater than 50% of the maximum score,
   then return Score
   else
   return 0.
```

An example is explained in the following: Referring to the highlighted image feature in Figure 4.70 it will be attempted to try to find which of the CAD features best match in Figure 4.71. Each of the features has had their objects marked in the sorting order found in section 4.4.2.6 with T representing the top level object and A,B,C etc. representing the sorted lower level objects. The image object taken from the list is Ti and since it is not the robot and is a top level object, subroutine *FIND_FEATURE_ID* is now executed. The number of lower levels for Ti is 3 (objects Ai, Bi and Ci) so step 4 will only be executed when the feature ID3 is reached as this has the same number of lower levels. The other two features have a lower level count of one and two and will therefore be ignored. The parameters for Ti and Tc can now be tabulated and a top level object score created

T	Tc	Ti	Score
Edges	4	4	10
Centroid	Inside	Inside	10
Lower Levels	3	3	10
Object Area	3225	6317	
Owner Area	-	-	
Ratio	-	-	10
Total Score			40
Maximum Score			40

Table 4.3 *Top Level Object Scores*

A	Ac	Ai	Score
Edges	4	4	10
Centroid	Inside	Inside	10
Lower Levels	0	0	10
Object Area	585	1008	
Owner Area	3225	6317	
Ratio	0.181	0.16	8.8
Total Score			38.8
Maximum Score			40

Table 4.4 *Object A Scores*

B	Bc	Bi	Score
Edges	4	4	10
Centroid	Inside	Inside	10
Lower Levels	0	0	10
Object Area	585	1032	
Owner Area	3225	6317	
Ratio	0.181	0.163	9
Total Score			39
Maximum Score			40

Table 4.5 *Object B Scores*

C	Cc	Ci	Score
Edges	4	4	10
Centroid	Inside	Inside	10
Lower Levels	0	0	10
Object Area	1443	2680	
Owner Area	3225	6317	
Ratio	0.447	0.424	9.5
Total Score			39.5
Maximum Score			40

Table 4.6 *Object C Scores*

(see Table 4.3) from subroutine *SCOREOBJECT*, noting that the maximum possible score is 40. Then proceed directly with finding the scores for the lower level objects, which are shown in Tables 4.4 to 4.6. Since there is only one possible CAD object, step 12 of *FIND_FEATURE_ID* checks to see if the image feature is a valid one. The total score is 40.0 + 38.8 +39.0 + 39.5 = 157.3 which when compared to the maximum score of 160 gives a 98.3% match implying that the feature is indeed valid and can be marked as having CAD ID 3.

4.6.2 Location of Individual Features

So far, the various image features have been identified based on the different feature descriptions derived from the CAD diagram but it is not known which image feature corresponds to which specific CAD feature. Typically there will be several identical features and it is necessary to find the correct correspondence in order to obtain the image to CAD mapping. The method used to locate the features is to produce a map of image features and their neighbours, and then to 'convolve' this map over the CAD orthomap created in section 4.5.3.

4.6.2.1 The Image Orthomap

The generation of the 'image orthomap' is somewhat more complicated than the CAD orthomap since there will be features that are missing and the perspective distortion and camera roll will mean that neighbours are not necessarily in the orthogonal directions. As with the CAD orthomap, neighbours are assumed to be either east, north, south or west of the current feature, however, the definitions of the four directions must be changed for each feature.

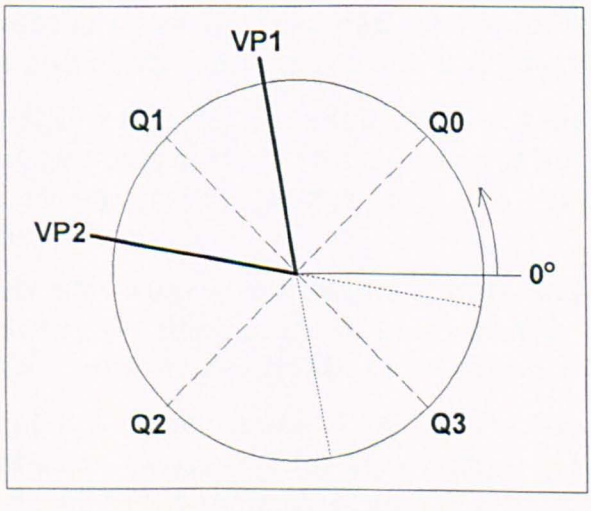


Figure 4.72 *Location of Quadrants define by Vanishing Points*

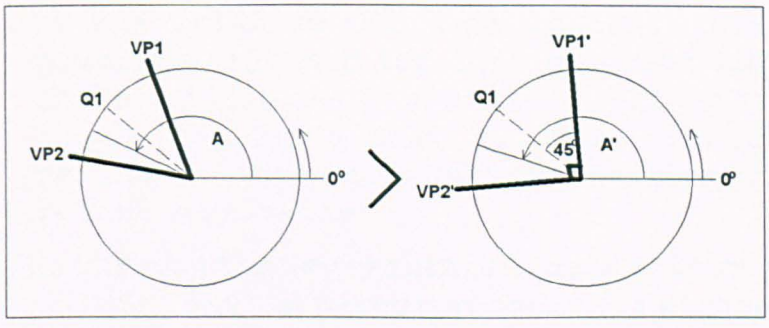


Figure 4.73 *Orthogonal Angle Adjustment*

To commence, for the given feature, the angles from the centre of the feature to the two main vanishing points are calculated. Since these angles are assumed to represent the true horizontal and vertical directions, they are used to create the four quadrant boundaries as shown in Figure 4.72. The quadrant boundaries are simply found from the midpoint between vanishing point angles. The first quadrant boundary, **Q0**, is defined as being less than 90^{O} ensuring that the north direction is equivalent to up.

Now that the 'real' orthogonal directions have been found. the image feature list is scanned to find the nearest neighbour in each of the quadrants. This can of course include a 'no-neighbour' for the instances where a feature lies near one of the edges of the building. The algorithm proceeds as follows:

1. For each image feature that is not the robot AND not the current feature do steps 2 - 7
   2. Calculate the angle and distance from the current feature centre to the neighbour feature centre and determine quadrant.
   3. Adjust the angle based on the vanishing points as well as the actual vanishing point direction for that quadrant (see following text).
   4. Calculate the difference in angle between the adjusted neighbour angle and the adjusted vanishing point angle of the appropriate quadrant.
   5. Fetch the feature diameter depending on the quadrant - vertical diameter for north or south, horizontal diameter for east or west.
   6. Calculate the relative distance to the neighbour by dividing the actual distance by the diameter found in step 5.

7. If the relative distance is less than the previous distance for this quadrant AND the angles and distances are within certain limits (see following text), then update the neighbour with the current neighbour and relative distance.
8. Go through the newly created image orthomap and delete any non-reciprocating neighbours, that is, keep only the neighbours that form a neighbour pair e.g. AB and BA.

Step 3 above is used to find the most likely real angle to a neighbour if there is perspective distortion. Since the camera is likely to be at some angle to the building face, the angle between the lines to the vanishing points will not be 90^{o}. A direction falling between the two vanishing points is adjusted as if the vanishing points were orthogonal as shown in Figure 4.73. If θ_1 is the angle to one vanishing point (or the opposite direction) and θ_2 is the angle to the other with $\theta_2 > \theta_1$, then the adjusted angle is found from:

$$\theta_{New} = \frac{(\theta_2 - \theta_1)}{2} + \frac{\theta_{Old}}{(\theta_2 - \theta_1)} * 90 - 45 \qquad [4.10]$$

Note that the calculation needs to be altered slightly by adding 360^{o} to any angle less than 180^{o} if the angles are in the east quadrant (**Q3** to **Q0**) to allow for sign changes. The result should then have 360^{o} subtracted if it is greater than 360^{o}. The new angle is an approximation to the angle the neighbour would have if it was viewed with the camera being normal to the face of the building.

To determine if a potential neighbour is a valid neighbour then limits have to be placed on the distances and angles to the neighbour. Since a quadrant could contain several neighbours it was decided to accept only those which lie within a set angle range of the horizontal and vertical directions, as the nearest neighbours in most buildings lie in these directions. Additionally, limits are also defined from the image itself. Therefore, a feature is considered a neighbour if:

The adjusted angle difference (step 4) is less than 20 degrees

AND

The relative distance is less than 3 diameters OR the actual distance is less than 200 pixels.

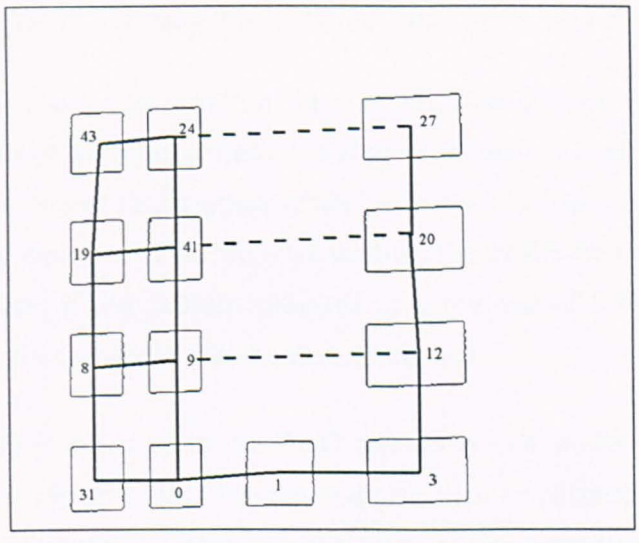


Figure 4.74 *Image Orthomap*

Feature	Relative Neighbour Distances			
	East	North	West	South
0	2,02	1,82	1,51	-
1	2,03	-	1,55	-
3	-	1,85	1,39	-
8	1,50	1,81	-	1,84
9	*4,56	1,83	1,50	1,84
12	-	1,84	2,46	1,88
19	1,47	1,80	-	1,81
20	-	1,84	2,48	1,84
24	*4,51	-	1,46	1,75
27	-	-	2,45	1,87
31	1,48	1,82	-	-
41	*4,46	1,78	1,44	1,83
43	1,49	-	-	1,83

* Indicates a Far-Neighbour

Table 4.7 *Image Orthomap Relative Distances*

To allow for possible missing features, it was also decided that if the above conditions could not be met but there was a feature further away, it could be considered a 'far neighbour' if the angle difference was less than 10^{o}. This finer limit restricts the amount potential spurious neighbours.

A typical image orthomap is shown in Figure 4.74 where, although some of the features are missing, it is possible to reach every recognised feature from any other. Table 4.7 gives the relative distance to each of the neighbours and it can be seen that there is close agreement between the neighbours in a given direction for similar feature types, especially in the north and south directions regardless of their position in the image. Also, it is interesting to compare these distances with the equivalent ones of the orthomap in Figure 4.69 of section 4.5.3.

4.6.2.2 Image- and CAD Orthomap Convolution (Reconciliation)

Matching by convolution can be best thought of by considering two separate patterns with one pattern being part of the other pattern. If the smaller one is moved, or convolved, over the larger then it will appear to disappear when it exactly lines up with the corresponding section of the larger pattern and so the total visible area of the two patterns will be at a minimum. Conversely, if the pattern consisted of a number of features, then the total number of features visible would also be at a maximum.

The image orthomap is matched to the CAD orthomap in a similar fashion where this time, the pattern is made up of the features and also their relationship with one another. The process is complicated by the fact that the image orthomap is not a uniform (affine) transformation of the CAD orthomap and so cannot be convolved on a pixel by pixel basis. Instead, the convolution is performed on a feature by feature basis by seeing how many features have matching neighbours. The process starts by making an assumption that a given CAD feature corresponds to a given image feature. It then looks to see how many of the neighbours are the same and assigns a score equal to the number of correct neighbours (maximum of 4). The process recursively passes around the matching neighbours and performs the same scoring on each new feature. A total score is built up for the alignment and stored. The process is repeated but this time the initial match is assumed to be a different pair of features and a total score is built up again. Finally, the alignment with the highest score is taken to be the correct alignment and each of the image features is tagged with its corresponding CAD feature. This can be described formally as follows: Suppose there are **Q** different types of CAD feature in a list $\mathbf{F_f}$, **f** = 1...**Q**. The CAD orthomap consists of **R** features labelled $\mathbf{C_{f,j}}$, **j** = 1..**R** and the image orthomap contains **P** identified features labelled $\mathbf{I_{f,i}}$, **i** = 1..**P**. The matching algorithm is then:

1. For each of the CAD features C_f do steps 2 to 3.
   2. Find the feature's first occurrence in the image list ($I_{f,1}$).
   3. Convolve the two orthomaps and find a best matching score.
4. Use the alignment with the highest matching score to mark all the image features.

The convolution stage in step 3 can be described as:

3.1 Align $I_{f,1}$ with each of CAD features of the same type $C_{f,j}$.

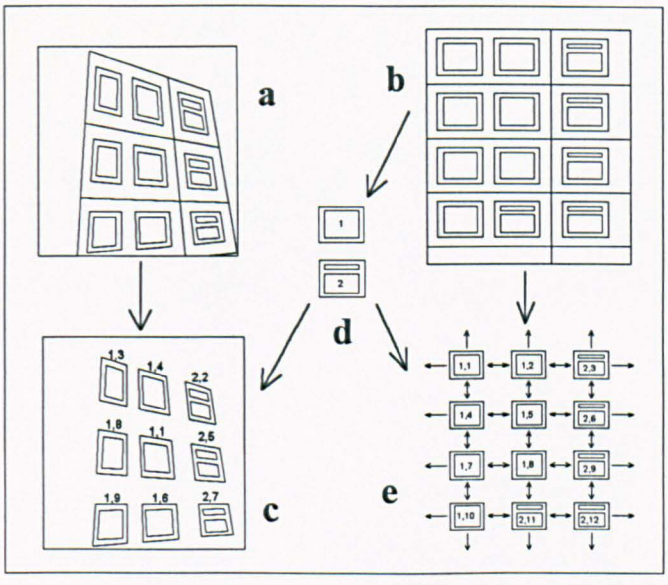


Figure 4.75 *The Relationships between Image, Diagram and Orthomaps*

```
3.2 Calculate a matching score for this alignment.
3.3 Save this score and alignment details if the
    score is higher than any earlier scores.
```

Step 3.1 above increments the matching score for each successful matching neighbour. Two neighbours are considered the same if they occur in the same quadrant, have the same IDs, and the ratio of the two relative distances to the neighbours is greater than 0.8, the ratio being the minimum of A/B or B/A. A match is also considered to exist if there are no neighbours in the given quadrant. In the case when the CAD feature has a neighbour and the image feature does not, it is possible that the image has a missing feature. The matching process then tries to look for a 'far neighbour' in the given quadrant to see if a further match is possible. As with the ordinary neighbours, the distance ratio must be greater than 0.8.

Figure 4.75 taken from Paterson, Dowling and Chamberlain, (1995) shows how the two orthomaps, c and e are derived from the CAD diagram, b, and the image, a. The numbering of the features in the feature list, d, is used to create the labelling in the algorithm above. The algorithm would start by assuming that the image feature 1,1 corresponds with CAD feature 1,1, builds up a score and then continue with CAD feature 1,2.

In this example it is shown that there is only one possible alignment and that is with $\mathbf{I_{1,1}}$ and $\mathbf{C_{1,5}}$. However, in reality there may be several alignments yielding the same score. It is then assumed that the uppermost alignment is the correct one as one of the set-up

criteria specifies that the top of the building must appear somewhere in the image, thus avoiding confusion. Figure 4.76 shows each of the alignment scores for the image in Figure 4.74 when the image feature **0** is convolved over the corresponding CAD features. That is, only a score of 1 is obtained if image feature **0** is aligned with CAD feature $\mathbf{C_{1,1}}$ whereas a maximum score of 48 results from alignment with CAD feature $\mathbf{C_{1,14}}$. The same high score is obtained, as expected, if the algorithm is executed using features 2 and 3 instead of 1.

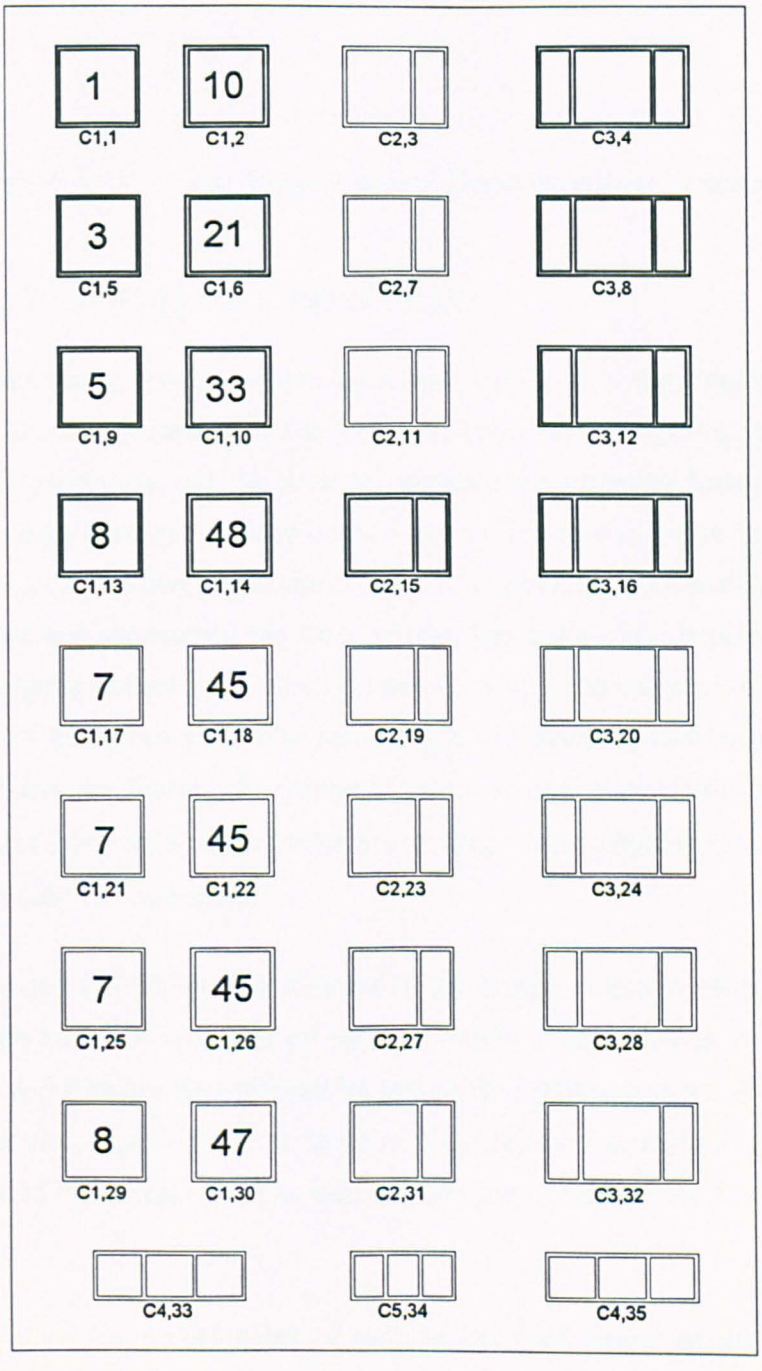


Figure 4.76 *Convolution Scores for Feature Type 1*

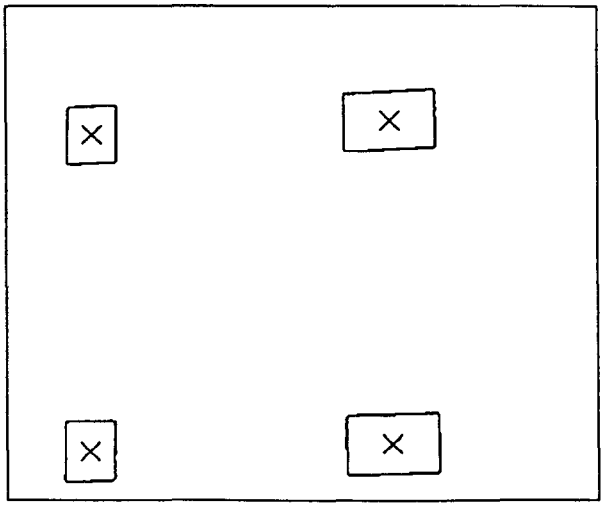


Figure 4.77 *Four Image Control Points and their Features*

4.6.3 Finding Four Mapping Control Points

Now that the individual image features have been matched to their counterparts in the CAD diagram, it will now be possible to locate the robot on the building. What is needed are four control points that will be used to calculate the mapping function in the next section. Since a high level of accuracy is not required, it was decided to use the centroids (see section 4.4.2) of the features as the corresponding points. The centroids, particularly in the image, are not necessarily the true centres, but are a close approximation. Considering a rectangular feature then, when viewed normally, the centroid will be the centre but since most of the views will be at some angle, the centroid will be offset. The true centre can be found by finding the intersection of the two lines joining diagonally opposite corners but this would require extra processing, the centroid is located already and not all features may be rectangular.

It must now be decided which four features in the image to use to generate the control points. Although accuracy is not of prime importance, it is desirable to obtain the best possible result and this can be achieved by taking four points that are as far as possible from each other and, ensuring that no three of them fall on a straight line. These criteria are easily fulfilled by taking points as near as possible to each of the four corners of the image.

The algorithm finds the co-ordinates of each of the four image corners, and for each corner, goes through the image feature list looking for the feature nearest to that corner. Once four features have been picked, a check is made to see if any of the features have

been selected by two or more corners and if so, the algorithm fails. The algorithm finally examines groups of three points from the image and the CAD model to see if any of them fall in a straight line. In this case, a straight line is considered if the angles between the lines joining the points are less than 5^{o}. If all conditions are satisfied, then the four image points and the four CAD points are passed on to the mapping function. Figure 4.77 shows a typical set of four image points.

4.6.4 Calculation of the Image to CAD Model Mapping Parameters

Recalling the perspective transformation equations given in section 3.1.1, we see that there are 8 parameters to calculate requiring the four co-ordinate pairs from the previous section. Multiplying by the denominator and rewriting gives:

$$x_i b_{11} + y_i b_{12} + b_{13} - X_i x_i b_{31} - X_i y_i b_{32} = X_i \qquad [4.11]$$
$$x_i b_{21} + y_i b_{22} + b_{23} - Y_i x_i b_{31} - Y_i y_i b_{32} = Y_i$$

Where x_i, y_i are the image points, X_i, Y_i are the CAD model points, b_{mn} are the unknown parameters and **i** represents the four points 0,1,2 and 3. The **i** equation pairs can be written more conveniently in matrix form as follows:

$$\begin{pmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -X_0x_0 & -X_0y_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -X_1x_1 & -X_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -X_2x_2 & -X_2y_2 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -Y_0x_0 & -Y_0y_0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -Y_1x_1 & -Y_1y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -Y_2x_2 & -Y_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -X_3x_3 & -X_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -Y_3x_3 & -Y_3y_3 \end{pmatrix} \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{21} \\ b_{22} \\ b_{23} \\ b_{31} \\ b_{23} \end{pmatrix} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ Y_0 \\ Y_1 \\ Y_2 \\ X_3 \\ Y_3 \end{pmatrix} \qquad [4.12]$$

which is of the form

$$\underline{A}\,\underline{b} = \underline{C} \qquad [4.13]$$

The unknowns, b_{mn}, are theoretically found from

$$\underline{b} = \underline{A}^{-1}\,\underline{C} \qquad [4.14]$$

but since this is a large matrix, it is not practical to find the inverse of $\underline{A}$. Instead Gaussian elimination (Lennox and Chadwick, 1979: 172-173) is used and this is the reason why the matrix in [4.12] was written in the way it is as this ensures that no zeros appear on the leading diagonal as it will ultimately form an 'upper triangular matrix'. Gaussian

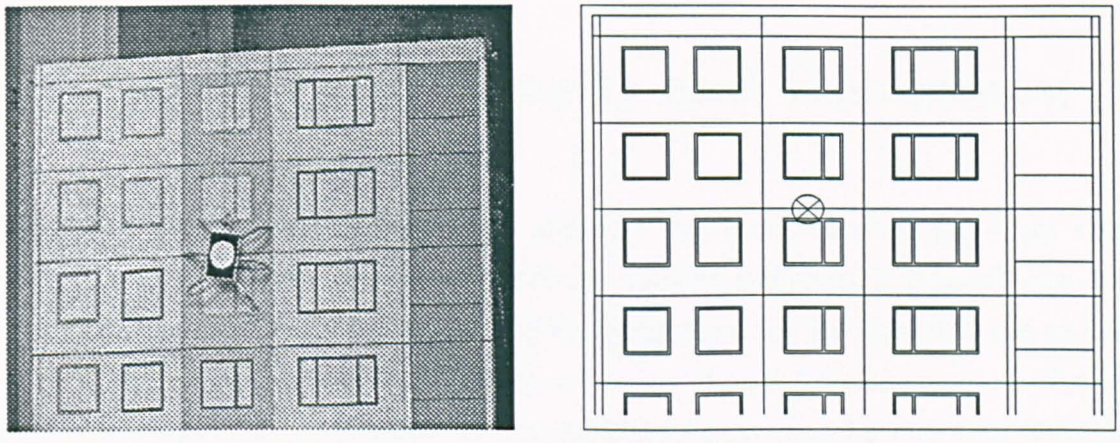


Figure 4.78 *Located Robot Target*

elimination proceeds by adjusting each of the rows such that the bottom left triangle of the matrix has only 0 elements resulting in only a single value in the row corresponding to $\mathbf{b}_{23}$. $\mathbf{b}_{23}$ is simply evaluated from $\mathbf{b}_{23} = \mathbf{Y}_3 / \mathbf{A}_{8,8}$ and the result is substituted back into the row above allowing $\mathbf{b}_{31}$ to be found. The process continues until all the $\mathbf{b}_{mn}$ are evaluated, which allows any point from the image to be mapped to a point in the CAD diagram.

4.6.5 The Location of the Robot on the CAD Diagram

The final step is to calculate the position of the robot in real world co-ordinates from the CAD model. Using the 8 mapping parameters from the previous section, the image co-ordinates, (x,y) of the robot target found in section 4.4.2.7 and feeding them into the perspective transformation equation 3.1.1 yields the (X,Y) position of the robot target. Figure 4.78 shows a comparison of the original image with a section of the CAD diagram with the transformed robot target position superimposed on it. It is clear that the method is reliable, that is, showing which floor and between which windows is the robot situated. The main source of error is the fact that the target is not in the same plane as the building so the intersection of the line of sight through the target and the building will be different from the line normal to the target and the building.

5

Experimental Results and Discussion

Up to this point the development of the algorithm has been described and it has been demonstrated that, for a given image of a model building and robot, it is possible to find the robot's position. To gain some idea of the performance of the algorithm and to find out what improvements may be necessary, a number of tests have been devised for the model building and a real building. The model building assumes the near ideal situation allowing other parameters, such as camera angle, to be checked without being concerned about the quality or complexity of the building, and the real building is used to find out what the performance is like in the real world - the ultimate goal of this work.

5.1 The Model Building

As stated earlier, a model building was used to develop the essential stages of the algorithm without the added complications involved with real buildings. This model is now used to test various parameters of the algorithm and to find out when and how the algorithm fails. By using a near ideal model, results can be obtained to test the latter stages of the algorithm in a controlled way. The features on the model have been deliberately enhanced to make them stand out clearly thus minimising the effects of lighting conditions and poor contrast. Some of the tests are aimed at trying to find an operating envelope for the camera with regards to position and roll. The larger this envelope is, the more freedom the operator has in setting up. The other tests try to find out how robust the algorithm is to spurious data such as missing or incorrectly identified features assuming the camera is operating within the correct envelope. Finally image quality is examined to see if additional global processing before edge detection has any advantages.

5.1.1 Robot Placement Accuracy

The most important measurement that can be made is to find out how accurate the algorithm is at placing the robot, as this gives an indication of the effectiveness of the algorithm. There are three main factors which influence the position of the robot, namely the position of the camera, the features chosen to provide the mapping points and the part of the feature chosen for the mapping points. The following tests show how these have influenced the robot position.

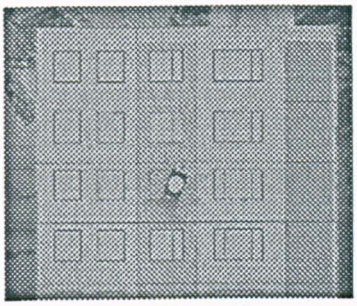
5.1.1.1 Effects of Camera Angle of Incidence

When the building is viewed face on, features are easily recognisable but when moving around it, the features become harder to recognise. The same is true with the robot location algorithm and these tests were designed to see at what point the algorithm fails for different camera angles. Firstly, the camera was placed normal to the building and a position for the robot was found, then the building was rotated at 15^{o} intervals to 90^{o} to see what would happen. Rotating the building had the same effect as moving the camera but was simpler to do. Next the process was repeated but this time the building was tilted back to simulate looking up at it. Finally the building was tilted and rotated to give a change in two planes simultaneously. Figure 5.1 shows all the images used and the terms yaw and pitch are used to indicate the amount of rotation in the horizontal and vertical planes respectively. Table 5.1 shows the results indicating the amount of position error or the type of failure that occurred if it was not possible to obtain a position. The actual robot placements are shown graphically in figure 5.2.

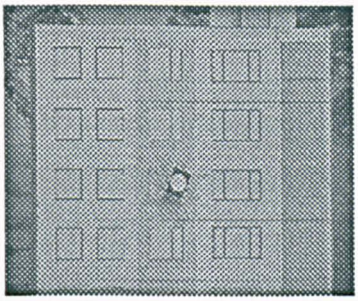
The model robot was fixed on the building and the position was measured at x = 237mm and y = 539mm. The first result with the camera normal to the building gave a zero error although the accuracy of measurement is approximately 1mm in any direction. As the yaw of the camera increased, so did the amount of error in the x direction. This is expected

Robot Position	Camera Angle (degrees)		Robot Position (mm)		Position Error (mm)	
	Yaw	Pitch	x	y	dx	dy
Normal	0	0	207	539	0	0
Horizontal	15	0	212	538	5	-1
	30	0	218	539	11	0
	45	0	225	538	18	-1
	60	0	Robot not found			
	75	0	No Features found			
	90	0	Nothing visible			
Vertical	0	15	207	543	0	4
	0	30	207	550	0	11
	0	45	208	557	1	18
	0	60	Robot not found			
Diagonal	15	15	212	544	5	5
	30	30	220	549	13	10
	45	45	Vanishing Point failure			

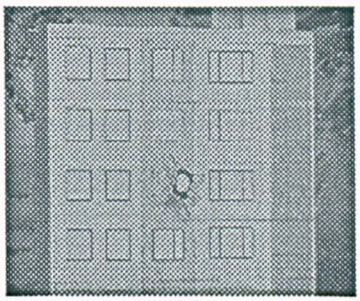
Table 5.1 *Changes in Robot Position with Camera Angle*



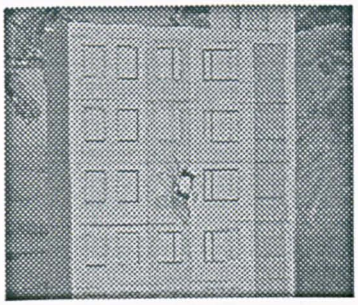
a: Normal



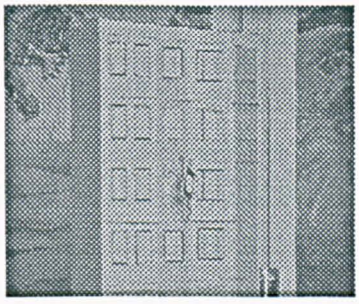
b: Yaw = 15, Pitch = 0



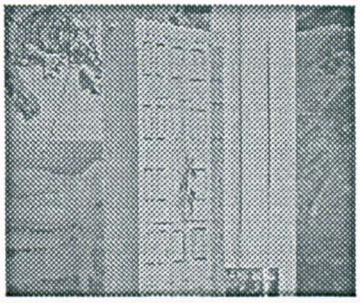
c: Yaw = 30, Pitch = 0



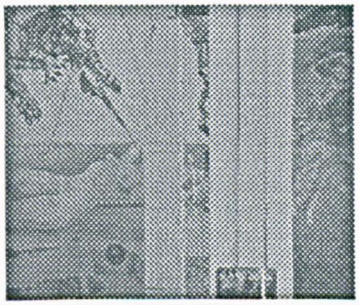
d: Yaw = 45, Pitch = 0



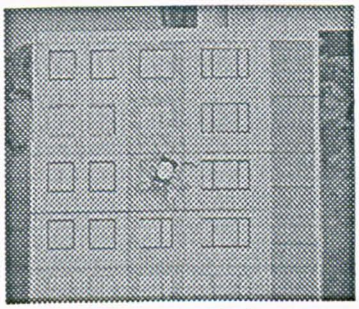
e: Yaw = 60, Pitch = 0



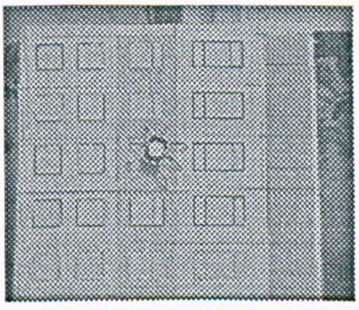
f: Yaw = 75, Pitch = 0



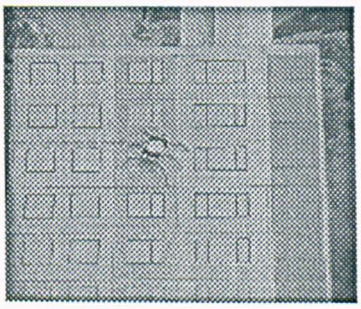
g: Yaw = 90, Pitch = 0



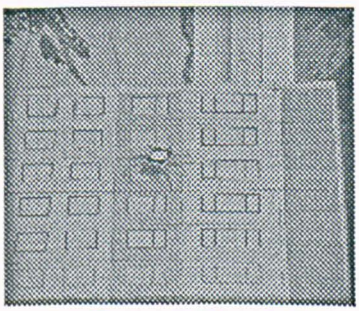
h: Yaw = 0, Pitch = 15



i: Yaw = 0, Pitch = 30

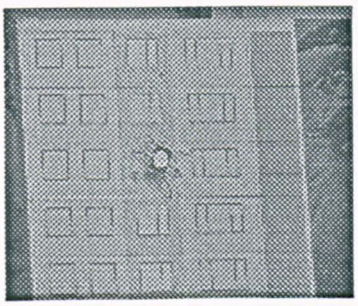


j: Yaw = 0, Pitch = 45

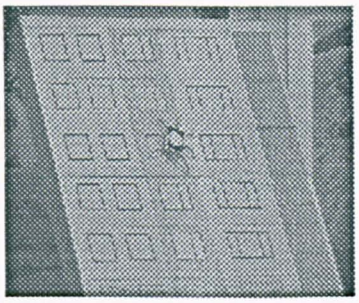


k: Yaw = 0, Pitch = 60

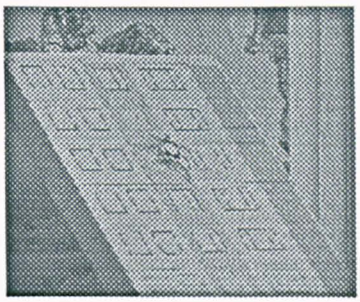
Figure 5.1 *Views of the Model Building from Different Camera Angles*



l: Yaw = 15, Pitch = 15



m: Yaw = 30, Pitch = 30



n: Yaw = 45, Pitch = 45

Figure 5.1 *Continued*

since the robot target is not in the same plane as the building and therefore the line of sight from the camera to the robot target passes through the target and intersects the plane of the building at a different point. The model target was approximately 20mm from the building and the error when the angle was 45^{o} was 18mm. At 45^{o}, the error should be equal to the distance of the target from the building and the result here verifies this. More specifically, if the camera is at an angle θ to the normal to the building face, then the error is given by:

$$\text{Error} = h \tan(\theta)$$

Where h is the distance of the selected point on the robot target from the building.

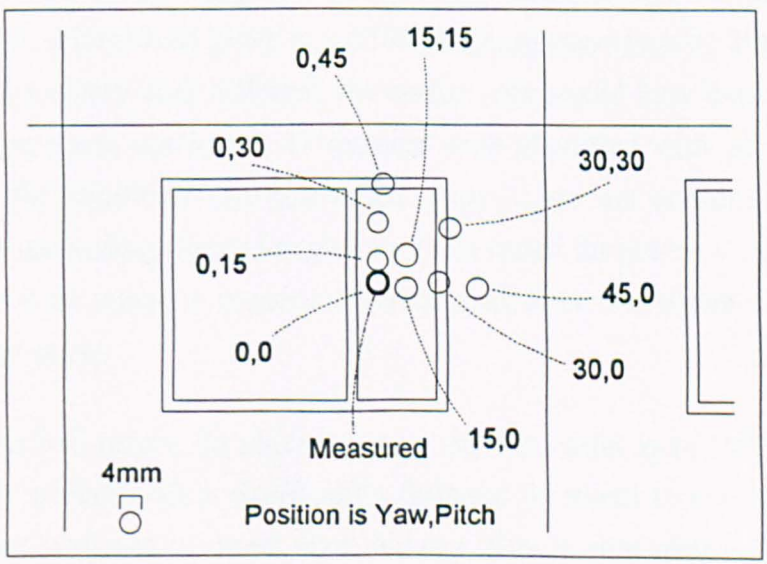


Figure 5.2 *Changes in Calculated Robot Position with Camera Angle*

As expected, similar results were obtained when the camera was moved in the vertical plane with errors being a near perfect match for the equivalent angles. The results when the camera was moved in two planes are slightly different but it must be considered that the resultant angle will be greater than the two angles indicated.

These results show that the algorithm has worked with angles up to 45^{o}, but at 60^{o} there are problems. For the case of movement in the horizontal plane, the algorithm failed at this value because it was unable to identify the robot target. By a chance alignment, part of the robot target, the robot leg and the edge of a window coincided in such a way that the edge detection process created a closed group that combined the robot and window into a single feature and was hence unrecognised. If this had not occurred, the algorithm may have worked since six features were identified correctly and could have yielded four suitable control points. Changing the contrast and threshold levels did not affect the result. When moved to 75^{o}, not a single feature was identified. At this point the width of the window frames in the vertical direction are becoming sufficiently narrow that where two edges should occur, only one appeared, in other words, the resolution limit of the system had been reached. The 90^{o} case was really included for completeness for when the camera was in the plane of building face, none of the features could be seen and it was effectively beneath the robot target. Very similar failures were found when the camera was moved in the vertical plane. At 60^{o}, the robot was again not detected but for a different reason. At this angle, more of the side of the robot was visible creating more edges near the target. A small gap existed after edge detection but instead of the repair process closing it, it joined the group to a different group very close by. Had the ordering of the groups in memory been different, the correct join would have been made. Again, although the target was not found, 17 features were identified with only one missing indicating that the algorithm only just failed. Also it was not possible to correct the problem by pre-processing. Greater angles were not tested for practical reasons and also because there was no reason to expect a different result from that obtained when moving in the horizontal plane.

When moving in both planes, the algorithm was only successful up to 30^{o}. Even then, the 15^{o} image only worked with a significantly different threshold to remove a false edge which had again produced incorrect pixel linking. This in turn reduced the number of identified features but sufficient were found to obtain a valid answer. The algorithm did not work at 45^{o} due to a vanishing point failure. The target had been identified along with 11 features (7 missing) so sufficient data was present to produce a result. However, all

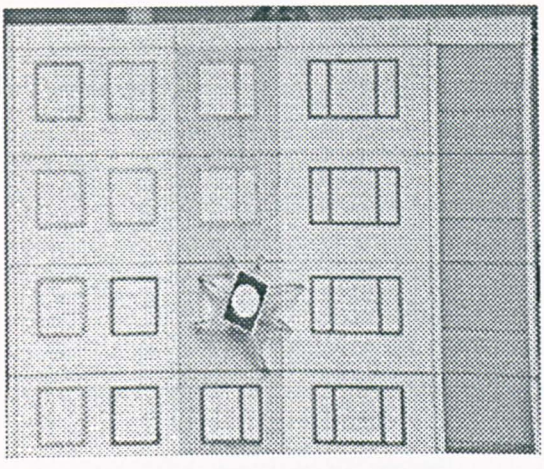


Figure 5.3 *Original image with maximum contrast processing.*

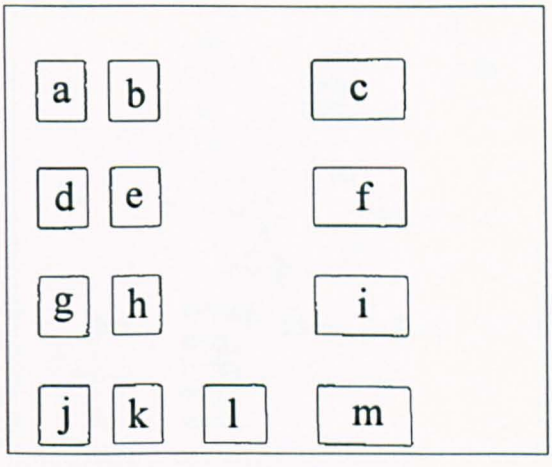


Figure 5.4 *Identified features.*

the real vertical lines in this image were tilted over quite considerably to the point that the line parameters moved into a different group in the vanishing point algorithm, which then assumed that a different vanishing point should be used. This false vanishing point meant that wrong directions were searched for, resulting in incorrect feature neighbours and therefore no matches and control points could be found.

5.1.1.2 Effects of Different Control Points

The effects of using different features to provide the four mapping points are presented here. Figure 5.3 shows the image used where the camera was placed such that the line of sight was normal to the building surface in order to minimise the influence of other effects. The image was processed to use maximum contrast and a threshold level of 9 was used after the edge detection to end up with the 13 identified features shown in Figure 5.4. The aim here was to see if the calculated robot position is changed by using different groups of four features.

The results of the different groups are shown in Table 5.2 and the first group, acmj, represents the group chosen automatically by the software. Other groups have been chosen to try to find the ones producing the greatest error and as expected, these come from groups made up of four neighbouring features. The positions are shown graphically in Figure 5.5 where the dark circle represents the actual measured position of the robot with an accuracy of ±1mm. Similarly the calculated circles represent a precision of ±1 digit or ±1 pixel. The largest errors are from groups containing feature g and h forming small groups to one side of the robot. This could result from lens distortion coupled with

Robot Position		Position (mm)		Error (mm)	
		x	y	Dx	Dy
Automatic	ACMJ	205	539	-2	0
Corner Groups	abed	206	540	-1	1
	bcfe	205	539	-2	0
	himk	206	539	-1	0
	ghkj	203	537	-4	-2
Centre Groups	dehg	209	537	2	-2
	efih	205	539	-2	0
Other Groups	bcih	205	539	-2	0
	efmk	205	539	-2	0
	acig	205	540	-2	1
	dfjm	205	540	-2	1
	dfgi	204	540	-3	1
	gijm	206	540	-1	1

Table 5.2 *Robot positions using different features.*

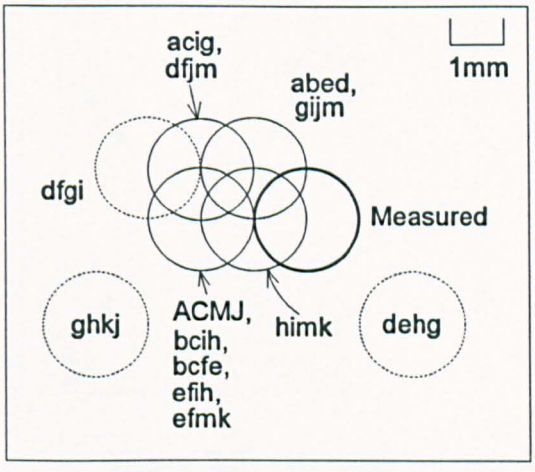


Figure 5.5 *Positions of the robot calculated using different features.*

small errors from the position of the centroids. Groups chosen, including the automatically selected group, such that the features surround the robot, produce results that are all in close agreement with a deviation within 1 mm, however, they are all slightly to one side of the measured position. This could be caused by a number of reasons such as the model building not being exactly the same size as that given in the CAD diagram (inaccuracies in construction, as with real buildings, means that distances are not exact, but this is not a problem as the final measurements will be relative to a known feature) and more likely, the camera line of sight not being exactly normal to the centre of the robot target as is the case here. Remembering that the target is not in the same plane as the building surface, the apparent position when viewed at an angle will not be the same as the actual position.

In conclusion, the choice of the four features does make a difference as expected but the error produced is well within the range of the desired accuracy as it is still known which features are closest to the robot. The automatic selection of the four furthest points does produce the most reliable position.

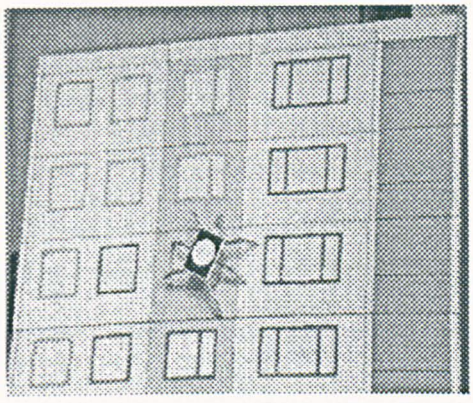


Figure 5.6 *Image with perspective distortion*

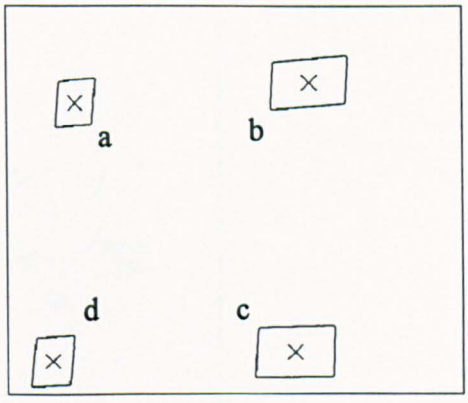


Figure 5.7 *The four automatically chosen features and points*

5.1.1.3 Effects of Using Vertices

The aim of this test is to determine if using the centroid of an image feature for a mapping point is a suitable approximation to the centre. Figure 5.6 was used for this experiment but had to be enhanced by sharpening since it was out of focus and the processing was unable to identify the robot target. Figure 5.7 shows the features chosen automatically by the algorithm and the locations of the centroids used. An image with perspective distortion was used as the true centres of the rectangular features are not in the same place as the centroids. It was decided that the best indication of accuracy was to use the vertices of the rectangles rather than the true centres as the vertices are easier to locate and the result is just as valid. The vertices were located manually using an image editing package which allowed lines to be drawn over the feature edges. The position of the intersection of these lines then gave the co-ordinates of the vertices and was more accurate than picking one of the pixels by sight as a certain amount of corner rounding occurs.

Robot Position		Vertices				Position (mm)		Error (mm)	
		a	b	c	d				
Groups	Centroid	-	-	-	-	199	546	-	-
	Outer	TL	TR	BR	BL	198	546	-1	0
	Inner	BR	BL	TL	TR	201	546	2	0
	Top Left	TL	TL	TL	TL	199	546	0	0
	Top Right	TR	TR	TR	TR	201	546	2	0
	Bottom Left	BL	BL	BL	BL	200	546	1	0
	Bottom Right	BR	BR	BR	BR	201	547	2	1

Table 5.3 *Calculated robot positions*

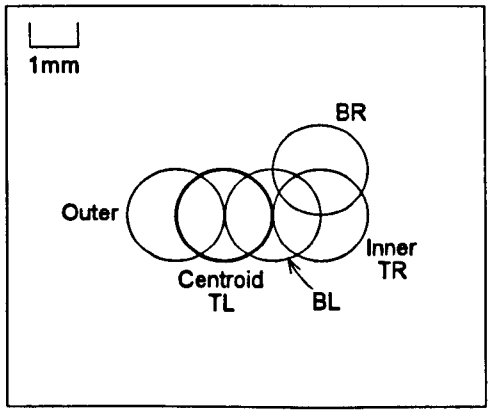


Figure 5.8 *Relative positions of the robot*

The results for the different combinations of the four mapping points are given in Table 5.3 and are shown graphically in Figure 5.8 where the circles represent the one pixel quantisation error in position. The groups of four were chosen such that each of the vertices of the features were used and these have been labelled TL for top left, BR for bottom right etc. The outer group has the greatest spread of points, the inner has the minimum spread and the others use the same corner on each of the features. There is very little difference in the results and considering that the size of the model robot target is 27mm x 38mm, the errors are very small indeed and the deviation in position using the centroids is similar to that of using the different vertices. This shows then, that using the centroids of the features as an approximation for the mapping points is quite acceptable.

5.1.2 Feature Identification

The results so far have concentrated on finding the position of the robot and in order for that to be possible, the features themselves must first be identified. This test looks at how camera angle affects feature recognition. Using the same images as those in Figure 5.1 it has already been shown that moving the camera to an angle of 45^{o} was sufficient to identify enough features whereas at larger angles, failures started to occur. Figures 5.9 and 5.10 show the closed groups for angles of 60^{o} and 70^{o} respectively using the statistical threshold. At 60^{o}, features were beginning to disappear as close groups such as adjacent window panes could not be separated. Although several features look intact, several of them could not be recognised as false edges were beginning to appear. By 75^{o}, not a single feature was recognised and close inspection of the closed groups reveals that the remaining features are in fact made up of smaller but false closed groups. The results here give an upper limit on performance as the model building surface is totally flat. In reality, windows and other features will be proud of the surface so that as the angle of incidence increases, these parts of the features will obscure other parts causing failure at lower angles.

5.1.2.1 Effects of Camera Roll

The effects of the camera angle on the features were covered indirectly in the tests on the robot position in section 5.1.1 where some features were still detectable at quite large angles of incidence. This test primarily looks at what happens if the camera is kept in the same position but is rolled about its optical axis (or the line of sight). This effectively simulated what would happen if the camera was not set up evenly on a real site and gave some indication as to how much leeway there is.

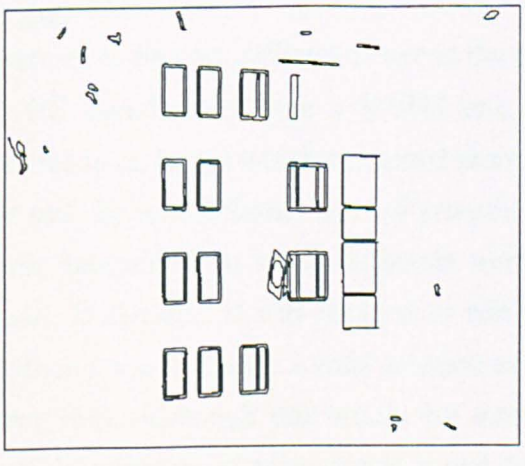


Figure 5.9 *Closed Groups at 60^{o}*

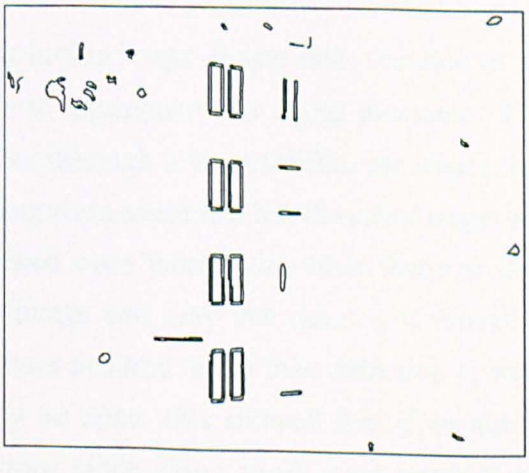


Figure 5.10 *Closed Groups at 75^{o}*

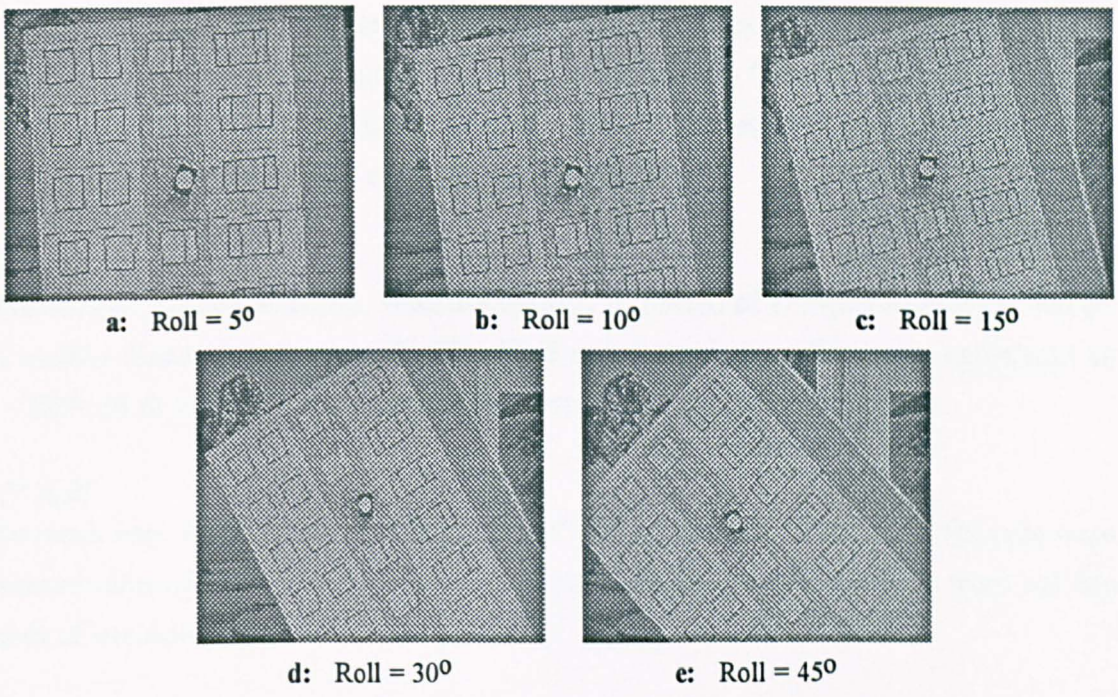


Figure 5.11 *Views of the Model Building with different Camera Roll Angles*

For practical reasons, the test again used a fixed camera and rotated the building the desired amount. As humans have a good perception of horizontal and vertical, that is, are able to detect small deviations from these directions, it was decided to concentrate more on smaller angles than the larger ones. It is highly unrealistic that someone would set up the camera with a roll of more than 20^{o} and is probable that no more than a 5^{o} roll would normally be present. Figure 5.11 shows the different images used and even at 5^{o}, there is an obvious tilt that the operator would probably correct. However, it was interesting to see how the algorithm performed and although the main aim here was to see if the features could remain detectable, the robot position was also calculated when possible.

5^{o} Roll

It proved to be very difficult to detect the robot in this image. It was only possible to use a VHS signal rather than a S-VHS one, due to equipment not being available. This resulted in an image which contained more noise although it was useful to see what effect this had. Several different types of pre-processing were attempted but the robot target was never detected since incorrect pixels were linked even though the other features were found. In the end, it was decided to edit the image and only one pixel was moved to produce a result. This is a valid solution as feature position rather than detection is being tested here. Although this would not normally be done, this showed that if an almost negligible amount of noise occurs in just the wrong place, then a result is not possible and the editing was merely done to check the final matching process. Without the editing, all

the features visible were correctly identified when the statistical threshold level of 21 was used. The calculated robot position was x = 208mm and y = 539mm which corresponded to only a 1mm error in the x direction so assuming that another image was taken where the target could be identified, a 5^{o} roll had no effect.

10^{o} Roll
This imaged worked perfectly. With a statistical threshold of 17, the robot target and all 14 visible features were correctly identified and the robot position was calculated as x = 207mm & y = 539mm which had zero error.

15^{o} Roll
The result here was identical to that of the 10^{o} roll except that 13 out of 15 features were detected although the failures were caused by incomplete objects which were not the result of the camera roll.

30^{o} Roll
At this angle many of the features were still being identified with only two missing, again because of incorrect features, but the algorithm started failing as a result of the vanishing point failures described in section 5.1.1.1 with large angles of pitch and yaw.

45^{o} Roll
The results from this image were essentially the same as for a roll of 30^{o}. This time only one feature was missing and the overall algorithm failed through a vanishing point failure. Angles larger than this are rather academic as this would not happen in reality. Also beyond 45^{o} the x and y axes become effectively swapped which would mean that the ordering of smaller objects within larger ones would change greatly, reducing the likelihood of a successful match. Some features such as squares would, however, remain unaffected since they have a 90^{o} rotational symmetry and appear unchanged at angles greater than 45^{o}.

In summary, the roll results showed that angles up to 15^{o} had no effect on the ability to identify features and correctly locate the robot. Although this is quite a small angle numerically, the way humans perceive a scene makes this appear as a large angle visually meaning that an operator would be very unlikely to set up the camera with an angle as large as this. (Humans can easily see if a camera view is deviating only slightly from the horizontal). At angles of 30^{o} and greater many of the features were still identified but the overall algorithm failed at the matching stage although this is not relevant here as these angles would not be expected.

5.1.3 Feature Location - Convolution

The position of the robot can only be calculated if the features in the image can be correctly identified and located. The following set of tests look at how well the algorithm performs when there are errors in some of the features and if features are missing. Ideally, changes in the data should not make any difference to the final robot position for good reliability, but naturally as the number of errors increases, the correct positions of features become ambiguous, for example, one set of features is mistaken for another. The tests are designed to identify at what point features cannot be recognised when the camera is moved, then see what effect missing, or incorrectly identified features have.

5.1.3.1 Effects of Camera Angle and Roll

The effects of camera angle and roll on convolution have been covered in sections 5.1.1 and 5.1.2 respectively, where robot location failures have been described. The main cause of failure at angles of greater then 45^{o} was a failure in the vanishing point algorithm. As this routine does not give an output at these angles, it is not possible to search in the correct direction for neighbours. Indeed, limits were set for the search angles and without vanishing point modification, the neighbours fell outside the accepted range leaving no features to convolve. This implies that an alternative vanishing point algorithm is needed.

5.1.3.2 Effects of Missing Features

It is almost a certainty that there will be features missing from the image and it is necessary to see how well the algorithm copes with partial data. This could result from the earlier stages of processing not filling gaps in pixel data for example, but in an ideal situation, the robot itself may well obscure one or more windows. The test image used, is the same as the one in Figure 5.3 since this minimises the effects of perspective distortion. The features located from this image are shown in Figure 5.12 and it can be immediately seen that there are already three missing features. One window is hidden by the robot and the other two were not detected since the background gray level happened to be very similar to the window frame rendering it indistinguishable in black and white (although in colour there was a large difference).

This test removes a number of different features by editing the data in the software so that a feature is ignored in the subsequent processing and has the same effect as hiding a feature in an image. The results of the feature removal are shown in Table 5.4 and are split up into a number of different sections. The first shows what happens when just a single

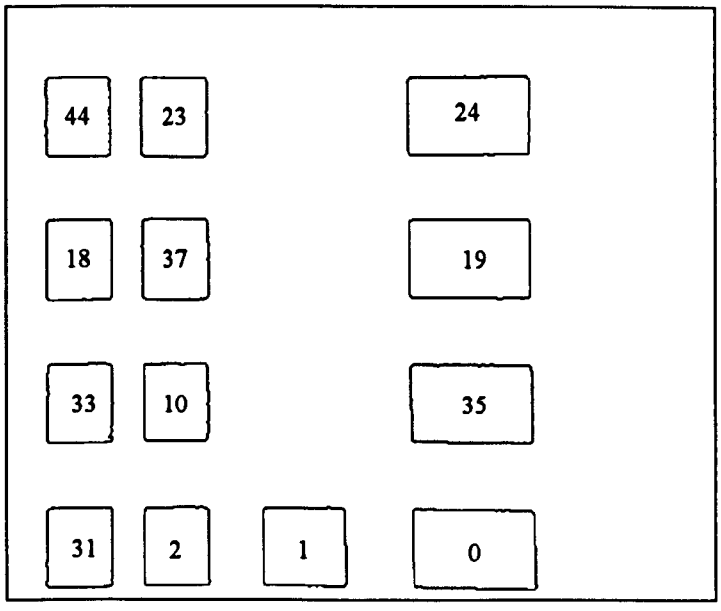


Figure 5.12 *Identified Image Features*

feature is removed and in all cases the calculated robot position is no more than 2mm different from the automatically calculated position which is caused by picking a different set of control points (see the test in section 5.1.1.2). If the set of control points chosen is the same as for the automatic set (features 44, 24, 0, and 31), then the table entry is marked as 'Normal' otherwise the chosen features are shown. Two types of position error have occurred here marked 'A' and 'B'. Error type A is caused by using a different set of control points and all the remaining features were used in the convolution. Error type B is slightly different in that the feature removed has meant that a number of features have been left out of the orthomap. Take the removal of feature 0 for example. Here, it is not possible to class features 1 and 35 as neighbours using the orthomap and as a result the group of features 24, 19 and 35 have not been included even though they were identified. Consequently none of them can be used for the selection of a control point.

The next group in the table looks at removing two features and the ones chosen were the features chosen for control points by the algorithm. A new type of error has occurred here and is marked as 'C' in the table. This was unexpected and needs to be investigated. However, it is likely to have been caused by feature 23 ending up with only one neighbour below it and has caused an error in the convolution. A type C error occurs when the remaining features produce ambiguous data for example, when the row of features 44, 23 and 24 are removed, the windows below are then assumed to be at the top of the building and the robot position has moved up one floor. There is no way of knowing that the

Robot Position		Features Removed	Robot Position		Error		Chosen Control Points	Error Type
			x	y	dx	dy		
Feature Groups Removed	Single	0	207	538	2	-1	44,23,1,31	B
		1	207	538	2	-1	44,23,2,31	B
		2	203	541	-2	2	44,23,10,31	B
		10	205	539	0	0	Normal	-
		18	205	539	0	0	Normal	-
		19	206	539	1	0	44,35,0,31	B
		23	205	539	0	0	Normal	-
		24	205	540	0	1	44,19,0,31	A
		31	205	539	0	0	44,24,0,2	A
		33	205	539	0	0	Normal	-
		35	206	538	1	-1	44,23,0,31	B
		37	205	539	0	0	Normal	-
		44	205	540	0	1	23,24,0,31	A
	Dual, Chosen Features	44,24	205	220	0	-319	23,19,0,31	C
		24,0	207	538	2	-1	44,23,1,31	B
		0,31	207	538	2	-1	44,23,1,2	B
		31,44	205	539	0	0	23,24,0,2	A
	Rows	44,23,24	205	620	0	81	18,19,0,31	C
		18,37,19	206	540	1	1	33,35,0,31	C
		33,10,35	-	-	-	-	-	D
		31,2,1,0	208	537	3	-2	44,23,10,33	B
	Columns	44,18,33,31	205	539	0	0	23,24,0,2	A
		23,37,10,2	-	-	-	-	-	D
		24,19,35,0	207	538	2	-1	44,23,1,31	B
	Misc.	18,23,2,33	-	-	-	-	-	D

Table 5.4 *Robot Positions with Feature Removal*

windows are not at the top and an assumption has to be made. Although the type C errors are large, considering that the distance between floors on the model building is 80mm, it can be seen that the error size is a multiple of floor height and the actual position error is again only a couple of millimetres. The final type of error marked as 'D' has occurred in the last three groups. Here, three or more features were removed and that resulted in only a small group of connected features remaining. If the column of features 23, 37, 10 and 2 is removed, then as before, an incomplete image orthomap is produced and only features 44, 18, 33 and 31 are used for the convolution. If these are correctly identified, then the control points end up in a straight line and it is not possible to produce a mapping.

Summarising, the algorithm is robust when up to 25% of the features are missing with the robot's position well within the desired accuracy range. As the percentage increases, more significant errors occur due to wrong assumptions being made during the convolution of the orthomaps and when most of the features are missing, it is not possible to find four suitable control points. However, certain features such as the ones near the edges of the

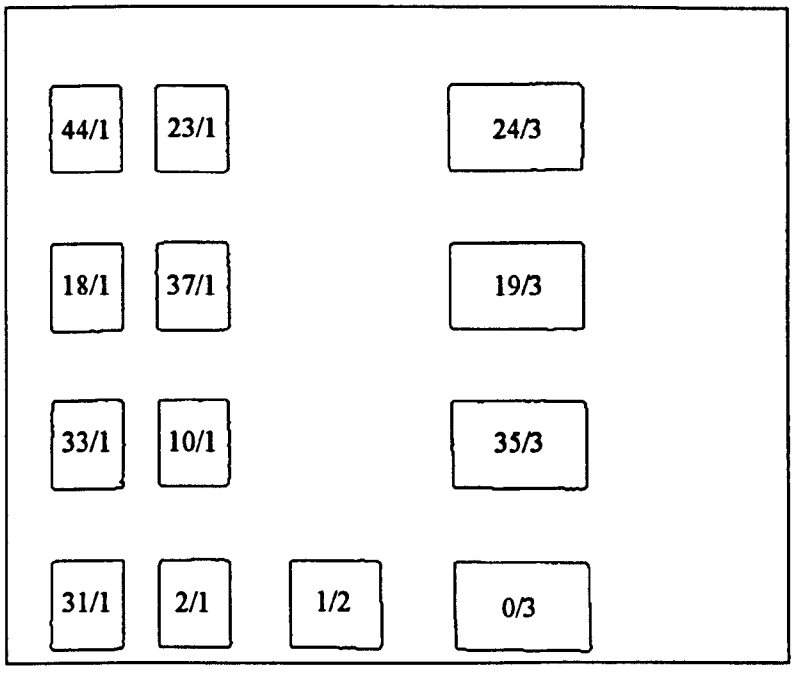


Figure 5.13 *Recognised Features and their Identities*

building have a greater significance and are therefore more likely to cause an error/failure if they are missing. These results show that for relatively simple situations, the orthomap convolution works fine but in more complex situations, a different method of determining neighbours is needed and possibly information about the edge of the building could be used to remove the floor ambiguity.

5.1.3.3 Effects of Incorrectly Identified Features

Some features in an image may be very similar to other features or may be partially obscured so that they appear to look like another one after the edge detection stage. This can lead to one feature being mistaken for another and this test is aimed at finding out how an incorrectly identified feature affects the position of the robot. The image used is the same as the one in Figure 5.3 and the resultant features used for this test are shown in Figure 5.13 along with their index in the data list followed by their type identification.

The image feature list was edited in the software to simulate incorrect identification. All the features of type 1 were altered type 2, those of type 2 were changed to type 3 and type 3 features were changed to type 1. Only one feature was changed at a time and the results giving the robot's position and the four control points chosen are shown in Table 5.5. With one feature incorrect, there was no significant change in the robot's position, with a maximum of only 2mm in either the x or y directions. Closer inspection of the results

Incorrect Features		New ID	Robot Position (mm)		Error (mm)		Chosen Control Points
			x	y	dx	dy	
Feature	44	2	205	540	0	1	23,24,0,31
	23	2	205	539	0	0	Normal
	18	2	205	539	0	0	Normal
	37	2	205	539	0	0	Normal
	33	2	205	539	0	0	Normal
	10	2	205	539	0	0	Normal
	31	2	205	539	0	0	44,24,0,2
	2	2	203	541	-2	2	44,23,10,3
	1	3	207	538	2	-1	44,23,2,31
	24	1	205	540	0	1	44,19,0,31
	19	1	206	539	1	0	44,35,0,31
	35	1	206	538	1	-1	44,23,0,31
	0	1	207	538	2	-1	44,23,1,31

Table 5.5 *Robot Position found from Incorrect Features*

reveals that they are the same as those for single missing features in section 5.1.3.2 and therefore the same conclusions can be drawn. The algorithm is deleting features whose identifications do not match after the reconciliation process has found the best alignment between the image and CAD diagram.

5.1.3.4 Effects of Extra Features

Extra features are unlikely to occur unless they are small and simple. It is not possible for a window containing a few panes to just appear on the building surface for example. If for some reason extra features do appear, for example when something is hanging out of a window, then the chances are that they will not be recognised and will therefore not appear in the image orthomap. The problem can be treated the same as for missing features, since an incorrect match will be ignored leaving the position of the robot un-changed.

5.1.4 Image Pre-processing

The following two tests are aimed at seeing what effects pre-processing the image has on identifying features and ultimately the robot's position. Every image has a certain level of contrast and a degree of blurring, both of which change the nature of edges between dark and light objects. Since edge detection is a crucial stage in the algorithm, it was decided to see how these two operations change the type of edges detected and compare the number of closed objects found.

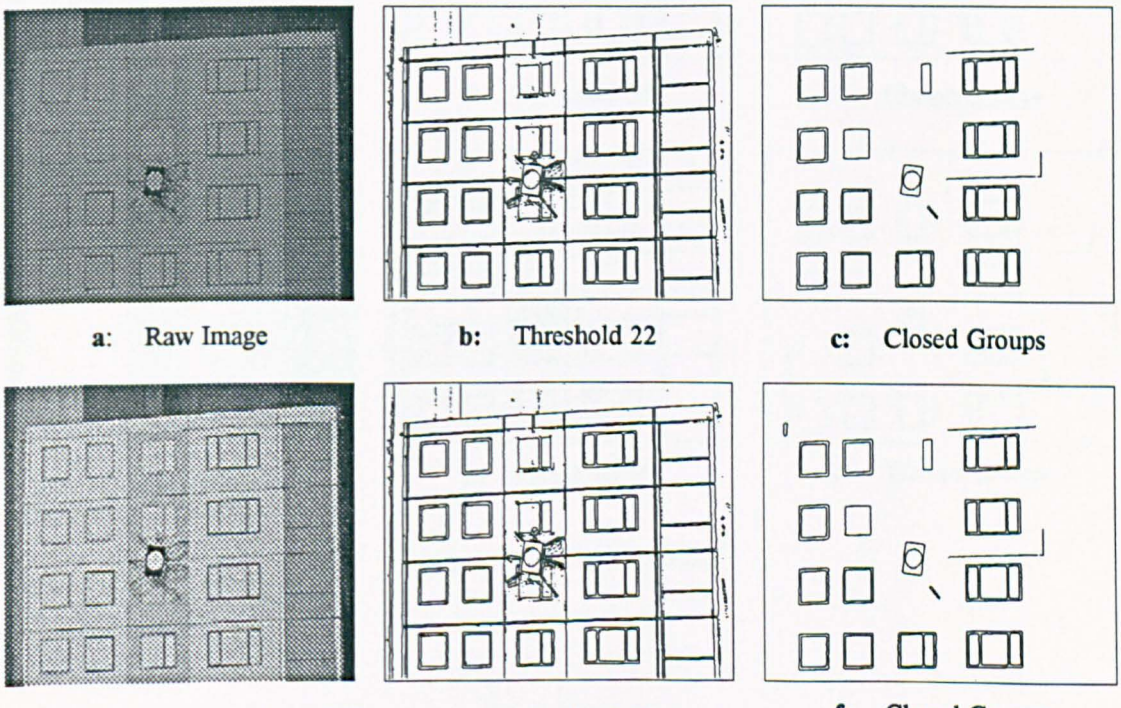
5.1.4.1 Effects of Contrast

Lighting conditions, surface finishes, camera hardware and frame grabber settings can all effect the range of gray levels of an image. The range for the images used here is from 0 (black) to 255 (white) and ideally an image wants to have intensities spread over most of this range. However, this is not always the case and a small range of gray levels can lead to edges not being detected.

The experiment carried out here was to see what happened to the edges and closed groups when the contrast was stretched. The raw image used to generate all the others is shown in Figure 5.14a and has a gray level range from 39 to 179 which is just over half the maximum possible. The first experiment stretches the gray level range of the entire image by setting the image minimum of 39 to 0, the image maximum of 179 to 255 and scaling all intermediate values with the resultant image shown in Figure 5.14d. The remaining images have their contrasts increased in a similar manner except that a smaller range in the image is chosen. One reason for doing this is that an image may contain a single dark or light noisy pixel which is not representative of the image and may unduly affect the contrast stretching.

The next experiment sets the minimum and maximum levels of the image to 5% of the raw image giving a range from 46 to 172. This is then scaled as before to fit all 256 levels with any pixels falling outside the full range being clipped to 0 and 255. Figures 5.14g, j and m have percentage levels of 5%, 10% and 20% respectively. Each of the gray level images was processed in the same way to produce the edge images including the use of a statistical threshold to create the binary edges. The threshold levels increased which indicates that the overall edge strength increased with contrast as expected. The last image in each group shows the closed groups from pixel linking and is used to see what effects increasing the contrast has. Although the raw image had a very low contrast and was quite

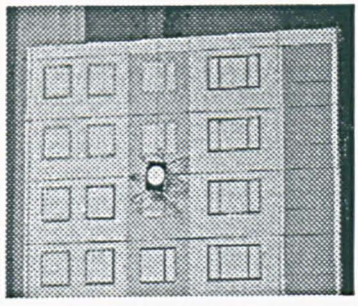
dark, most of the features were visible as can be seen by looking at the closed groups. Increasing the contrast has really only improved the visual appearance of the image with Figure 5.14m being much clearer than Figure 5.14a. Increasing the contrast increases the strength of edges which can sometimes produce extra closed groups as can be seen in Figures 5.14l and 5.14m since the edge strengths were brought above the threshold level. However, noise is also amplified and this has caused some of the features to be incomplete in the final set of images. On the whole, contrast has not made much difference to the output but if the contrast is exceptionally low, then it may be useful to increase it a little, say to the 5% range. A single missing feature is not that critical as shown in the previous tests but if it happened to be the robot target, then the system would fail and a change in contrast could make the difference.



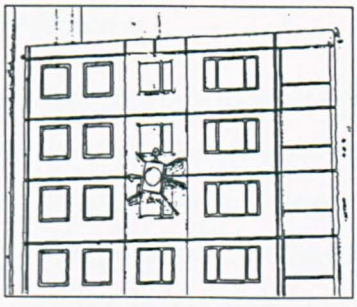
a: Raw Image **b:** Threshold 22 **c:** Closed Groups

d: Maximum Contrast 0% - 100% **e:** Threshold 22 **f:** Closed Groups

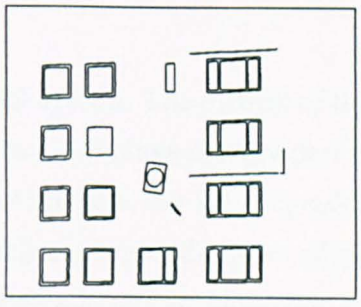
Figure 5.14 *The Effects of Changing Contrast...*



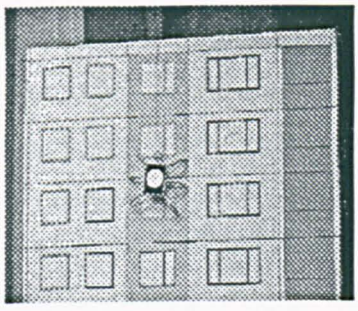
g: Contrast 5% - 95%



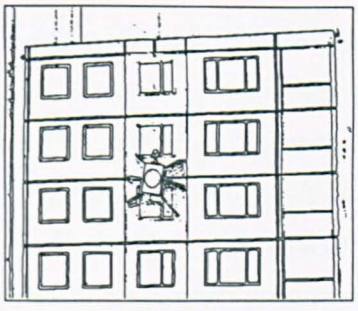
h: Threshold 24



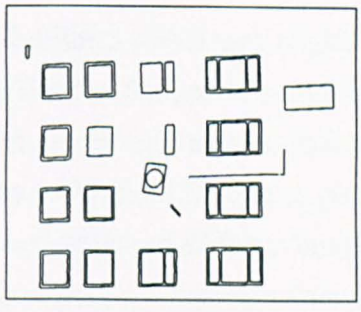
i: Closed Groups



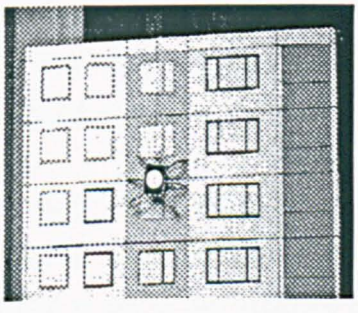
j: Contrast 10% - 90%



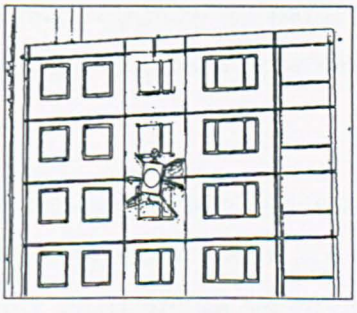
k: Threshold 25



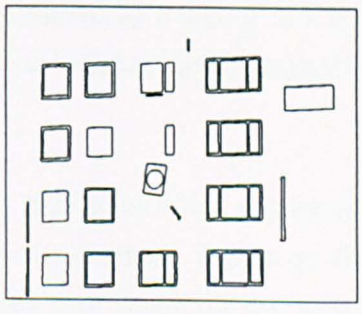
l: Closed Groups



m: Contrast 20% - 80%



n: Threshold 29



o: Closed Groups

Figure 5.14 *Continued*

5.1.4.2 Effects of Smoothing/De-focusing

Each image captured has passed through some form of optical system. The quality of the captured image is controlled by a number of factors but the one that plays a major part is focus. Even if the camera optics are ideal, an otherwise perfect image is severely degraded if the focus is incorrectly set. In this application, this has the effect of smoothing out edges with the possibility that they may not be detected. Smoothing however, can have an advantage in that noise, maybe in the form of a textured surface, can be reduced.

The test carried out here used a typical image of the model building which was slightly blurred due to imprecise focusing. The image was then artificially sharpened using an image editing facility to produce Figure 5.15a. This was then processed using statistical thresholding to produce the edge image in Figure 5.15b followed by pixel linking to give the closed groups in Figure 5.15c. Smoothing was used to simulate an out of focus image and provides a good approximation. The method used was to take an n x n pixel window with n being odd and set the central pixel value to the mean value of all the pixels in the window. Figures 5.15d, 5.15g and 5.15j show the image smoothed by 3 x 3, 5 x 5 and 7 x 7 windows respectively with Figure 5.15d being very similar to the original image that was sharpened.

It can be seen by looking at the closed groups in Figure 5.15 that quite a few groups are missing from the sharp image. Here, noisy pixels and other unwanted marks on the building have the greatest effect. A significant improvement was made by the minor blurring using the 3 x 3 window with most of the features being complete. As smoothing increased, fewer and fewer closed objects were found. This was particularly significant with this image as the robot target was incomplete and a position could not be found. The outer part of the target was blended with some of its surroundings, which gave false edges. The final image exaggerated the problems and, where features remained intact, they were somewhat distorted. Clearly, if smoothing was increased, then one window pane would merge with the next and it would not be recognised.

These results show that a small amount of smoothing is advantageous, which supports the use of the Canny edge detector in section 4.3.4.3. This can be done using software, where a 3 x 3 mean window is best but takes time. Alternatively, if the camera lens is defocused slightly, the smoothing is achieved instantly.

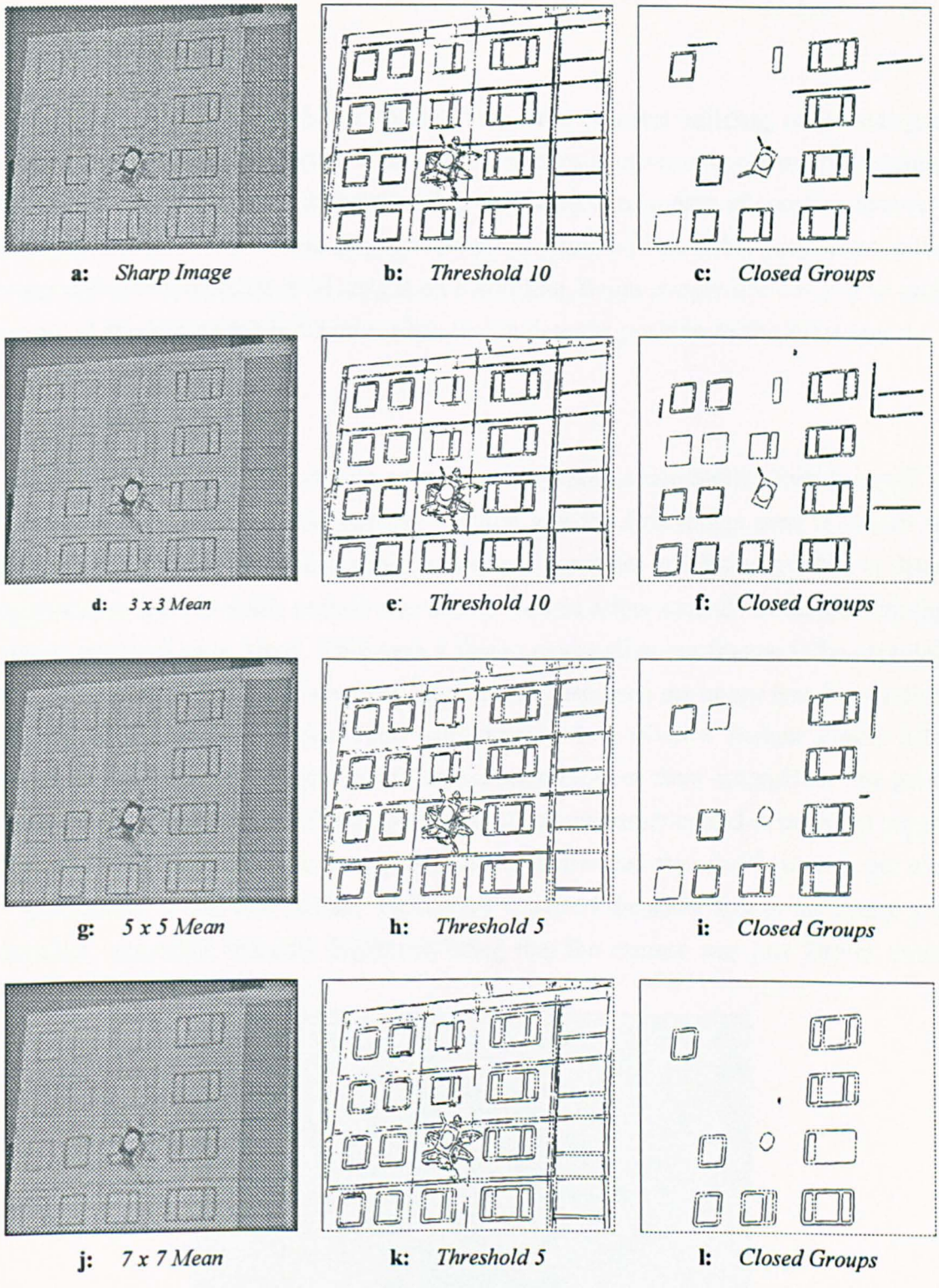


Figure 5.15 *The Effects of Applying Smoothing to an Image*

5.2 Real Building

The ultimate aim of this work is to locate a real robot on a real building, rather than just using a model. The performance of the algorithm is tested here on two types of building, one with a simple construction and the other consisting of a number of complex features. Since no robot is available, and considerable effort would be required to produce a replica target and obtain permission to hang it on a building, it was simply decided just to pick a point at random on the building surface to simulate the position of the robot target.

5.2.1 Simple Building

The building chosen representing a simple structure was a residential block for staff at Northwick Park Hospital near Harrow, London and the first image used is shown in Figure 5.16. Since it was not possible to set up a computer and frame grabber at these locations, an S-VHS video camera was used to record a few seconds of image with the camera mounted on a tripod. This gave a steady image allowing frames to be captured with the minimum amount of degradation. It can be seen from the image that the building consists of a dark brick construction with simple white window frames. Figure 5.17 shows the identified closed groups and as expected most of them are visible. The point marked by the cross is the randomly chosen point that was used instead of the robot target. This is a valid substitution, as it has been shown earlier that the model robot target was easy to detect. A full size version would have appeared the same size in the image and therefore detectable, the only difference being that the camera was just further away.

Figure 5.16 *Real Tower Block with Simple Features*

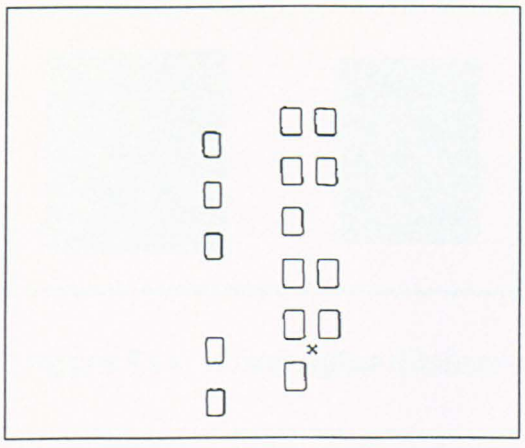


Figure 5.17 *Identified Features plus Simulated Robot Target*

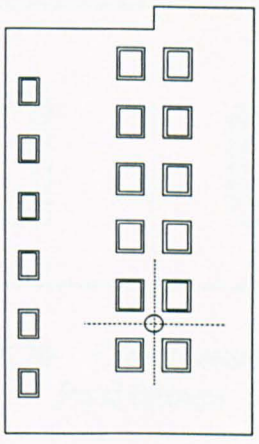


Figure 5.18 *CAD Diagram of Tower Block plus calculated Robot Position*

Figure 5.18 shows the CAD diagram for the face of the building and the calculated robot position. Clearly, there is a very close correspondence, showing that the algorithm worked correctly and, had a robot been present, its location would have been found successfully.

Although this image is simple, the windows at the left presented potential problems by not being aligned orthogonally in the horizontal direction. These windows were part of the stair well and consequently did not line up with the floors. Fortunately, the flexibility designed into the orthomap has meant that the features were properly identified showing that even though there were considerable misalignments, the algorithm coped successfully. Larger misalignments than this though would cause problems using the current orthomap implementation and further work is required to extend the orthomap principle to cope with features that do not line up in the two principle directions.

Three of the building windows were not detected and the reason that this did not effect the final robot's position was explained earlier in section 5.1.3.2. This raises the question as to why such clearly visible features were missed? After pixel linking, the groups making up a couple of these features were open groups rather than closed groups and were hence missed. Figure 5.19 shows two of the features in close up and Figure 5.20 shows the corresponding linked pixel groups. Looking more closely at the windows, it is seen that the vertical edges of the window frame are not straight and contain an oscillation. This originated from the image coming from a video tape with a slight difference between frames producing the effect as alternate frames are grabbed using the interlace. This has not affected the edge detection significantly since the Canny edge detector applies a degree of filtering. Close inspection of the linked pixels reveals the true reason why these

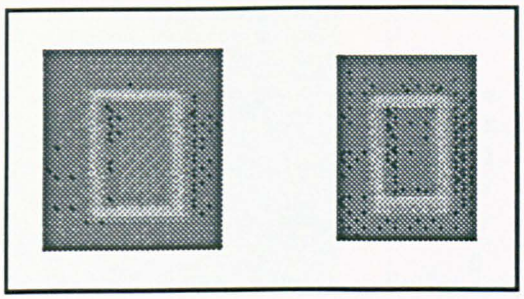


Figure 5.19 *Unidentified Windows*

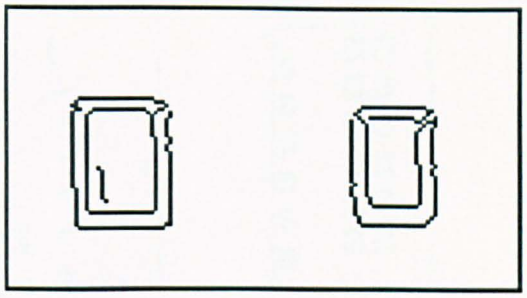


Figure 5.20 *Corresponding Linked Pixel Groups*

features were not detected. In both the examples an edge has been detected in the upper right hand corner and the linking process has tracked from the outside of the window frame and crossed to the inside therefore splitting the two closed objects that should exist. It is very hard to understand with the original image why this occurred, but there is a barely visible shadow caused by the upper part of the window frame. Since this is slightly darker, a false edge has been found in the corner. This shows that the current edge detection process is extremely sensitive to small errors, as in this case only a small percentage of wrong pixels has caused a failure in the identification of a feature, but the robustness of the algorithm allowed a correct map to be established.

It has now been shown that small errors occurred in the image that could cause large failures in the feature detection. However, if only a few were unrecognised, the robot's position is still calculated correctly. So how well did the algorithm perform on other parts of this building which have a more complex structure? The next image used was taken on the opposite side of the building and was in direct sunlight. Figure 5.21 shows the image after 5% contrast stretching with Figures 5.22 and 5.23 showing the thresholded edges (the level selected by hand and used was 16, slightly lower than the statistical threshold

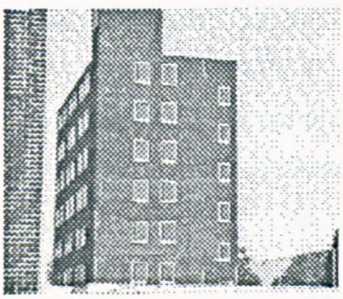


Figure 5.21 *Image*

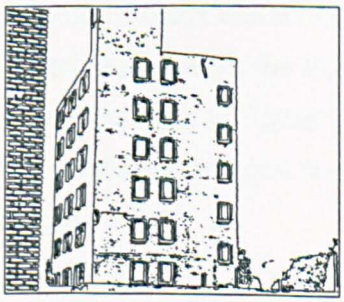


Figure 5.22 *Edges*

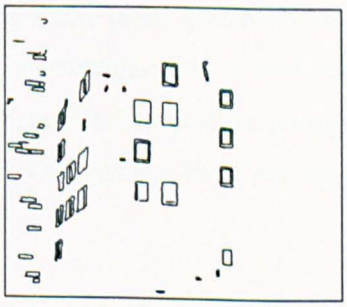


Figure 5.23 *Closed Groups*

Figure 5.24 *Image*

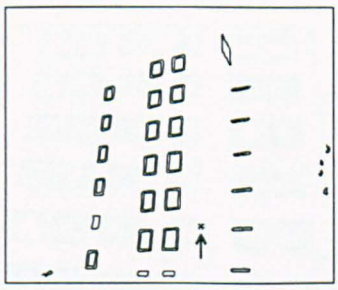


Figure 5.25 *Closed Groups plus Robot Position*

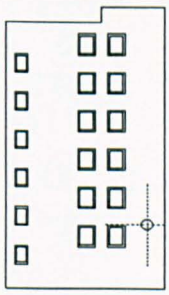


Figure 5.26 *CAD Diagram plus Robot Position*

of 19) and the closed groups respectively. The threshold chosen was the one that produced the most complete features. The first observation is how direct lighting has brought out more of the blemishes on the building with the brickwork beginning to become visible (rather than appearing as a uniform surface). This could cause problems as bricks are unwanted features, but since they have not made a significant impression in this case, it is unlikely that they would need to be treated separately. The most obvious problem is the lack of closed groups making up features. Only 5 out of 17 visible features could be identified. Close inspection of the original image shows dark vertical bands appearing just to the right of the vertical parts of the white window frames. This could be an artefact produced by using video tape which often occurs when there are strong light to dark transitions and caused extra edges to be detected, which confused the pixel linking process.

Returning to the original side of the building, Figure 5.24 is taken from a position much closer to the building requiring a wide angle lens to view the whole building. This image was taken directly into the sunlight which required over exposing to see the building details with a threshold level of 11 chosen (statistical = 9) the maximum number of complete features were produced, as shown in Figure 5.25. This produced a very good result with only one feature missing leaving plenty to perform a mapping. As with the first image, a point was chosen to represent the robot and it had the same pixel co-ordinates as Figure 5.17. (It appears in a different position on the building because the co-ordinates correspond to a different place on the building in figure 5.24 due to the camera being in a different position). The mapping produced the point shown in Figure 5.26 which again is in close agreement with the image.

Taking a different view of the building, Figure 5.27 shows a more typical situation with different types of window, some of which are open and many containing other visible features. This view was taken again in fairly strong sunlight at close range using a wide

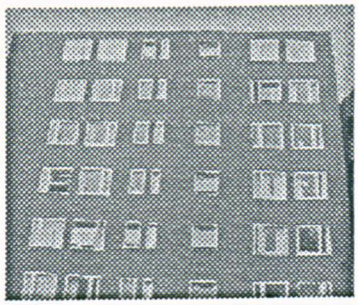


Figure 5.27 *Image*

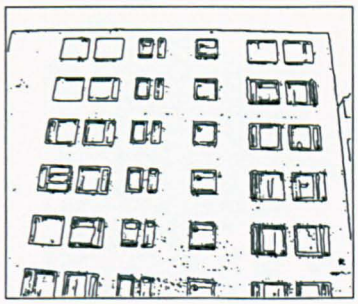


Figure 5.28 *Edges*

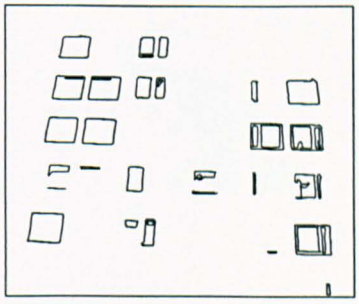


Figure 5.29 *Closed Groups*

field of view which gives the strong perspective. A number of different threshold levels were tried to produce the greatest number of complete features after pixel linking. Figure 5.28 shows the edges with a threshold level of 19 (statistical = 16) and it is immediately apparent that very few complete features are present. Only three were found, with one partially complete and thirteen where only the outer boundary was detected. The failure to pick up the features was caused by a variety of reasons. Particularly noticeable in the upper left and upper right of the image are windows reflecting a large amount of light from the sky. The gray level of this reflection, unfortunately, matches that of the window frames meaning that only the outer boundary was detected. There was also a high level of visual clutter caused by objects behind the windows such as curtains, lights, stickers and things placed on window sills. This led to a number of true but unwanted edges being found and, in one case, in particular at the right of the closed group image, the artefacts have joined together to produce a highly irregular but closed object, which clearly does not exist. Curtains pose a particular problem due to their size being the same as the windows and, being adjacent to them, are illuminated by light from outside making them bright. Added to this, curtains are often pale in colour, especially net curtains, or have a pale backing which does not produce a contrast between the curtains and the windows thus making the window edges disappear. Also, since the illumination in a room is much

Figure 5.30 *Image*

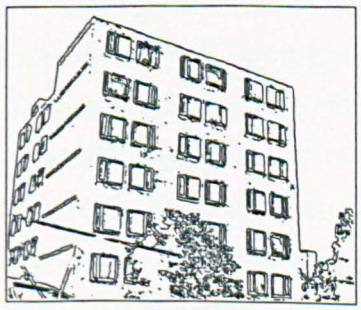


Figure 5.31 *Edges*

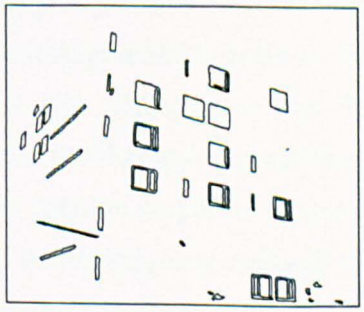


Figure 5.32 *Closed Groups*

Figure 5.33 *Complex Building Image*

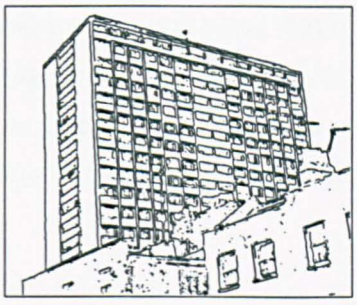


Figure 5.34 *Edges*

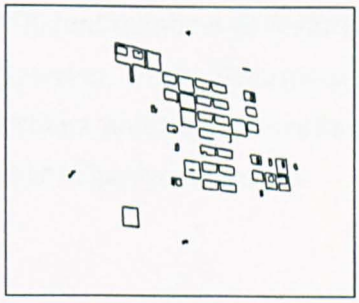


Figure 5.35 *Closed Groups*

less than that outside, the area behind the window appears dark and so a pale curtain stands out particularly well and a very strong edge is seen which then confuses the processing software. By looking at the window area in greater detail and extracting the straight lines as well as looking for rectangular type features, may improve on this aspect. Open windows, as seen on the mid-left, also altered the appearance and caused them not to be detected. The three features that were detected had no curtains present and there was little light reflected from the glass. These images show that there needs to be an improved method of picking out features - edge detection alone has no knowledge of the image and therefore often picks the incorrect edges.

Figures 5.30, 5.31 and 5.32 show a similar scene from a slightly different building. Again the building was in strong sunlight which highlighted some of the brickwork. In addition this image contains a tree plus a cable obscuring part of the building. However, if the visible features had been detected, then there would have been sufficient to find a proper match. Here, the statistical threshold produced a value of 14 for the edges and the one used to produce the most features was 12. With this value, only 5 features were complete with 4 partial and 7 with just a single closed group. As with the previous image, the failures were due to reflections, open windows and visual clutter within the window panes. It is interesting to note that although there is effectively a low signal to noise ratio with these two images, the edge pixels do form 'clumps' in the approximate areas of the feature. It may be an area for further research to see if it is only necessary to find the outside of these clumps, using a 'snake' type of active contour that behaved much like a rubber band stretched around the pixels. These clumps could then be inspected to see if they form a pattern that matches feature distribution in the orthomap, especially if the shapes of the clumps are similar to the expected features. If most of the features are found in this way, it would probably not be necessary to identify them specifically, merely their distribution could be matched to the orthomap.

Summarising the simple building results, it is clear that if sufficient numbers of features are detectable, then a good mapping result is achieved. However, many features are missed, particularly in direct sunlight which enhances reflections and objects inside a room, both of which cause a significant increase in the number of unwanted edges.

5.2.2 Complex Building

The image shown in Figure 5.33, which is of a large tower block near City University has a complex visual structure. Along with the usual windows, there are recessed balconies leaving holes in the building surface and there are large concrete columns and decorative panels. The detected edges are shown in Figure 5.34 and it is immediately apparent that there are a very large number of edges making it hard to differentiate features and, after pixel linking, only a few closed groups were created as shown in Figure 5.35. In this case, there would be no possibility of producing a mapping. Apart from the difficulties covered in the previous section, the main reason that features have not been found is that the resolution is too low, causing features, or rather their boundaries to merge with adjacent features. For example, the recessed balconies in the centre of the building have produced a fairly dark region with a concrete pillar in the middle. The edge detection process has joined the two regions into one with the edges running along the shaded parts of the pillar. Also the window frames have not been found as they are really too narrow. Increasing the resolution would mean much larger images from the increased number of pixels and would require the use of expensive sensors. Alternatively the same camera with a zoom lens may be able to pick up individual features and this is discussed further in chapter 6.

5.3 Summary

Summarising, the general algorithm worked well when used with the model building and robot which consisted of idealised features. This proved the principle of the algorithm with the robot being found to within a millimetre of its measured position when viewed normal to it. The parallax error caused by camera position was predictable and can be calculated thus eliminating it. Difficulties were encountered however, when real buildings were used with many features not being identified due to poor edge detection, although when sufficient features were identified, the algorithm worked successfully.

6

Further Work and Exploitation Considerations

This chapter discusses how the research might be continued towards improvements and greater reliability. Since this work has ultimately a real application, a few thoughts on exploitation are also presented.

6.1 Further Work

The major problem experienced is the extraction of recognisable features. So far, only complete features have been used, which have been shown to work well for an idealised model building. However, the difficulty with real buildings is the amount of noise present in forms of unwanted additional data such as reflections in glass. The edge detection phase has proven to be the most critical phase and a few alternatives, each one requiring considerable effort, are presented.

6.1.1 Close Up Feature Recognition

The existing method of taking a single image of the whole building and processing it globally to find smaller features has been shown to work only when the features are very well defined i.e. pale window frames on a dark building. An alternative, which is also based on how humans might recognise things, is to do the reverse and concentrate on a local area of the scene.

Suppose the camera previously used has zoomed into part of the build and only a single feature occupies the field of view. The resolution will be much higher, yielding greater detail; window frames will be more than a couple of pixels wide. If the same processes are used, there may well be the same number of edge pixels but they will tend to form fewer- and larger straight lines and corners. This makes it somewhat easier to fill in missing data. As an example, Figure 6.1 shows part of a window frame on a brick building. The results of edge detection shown in Figure 6.2 have managed to separate the individual window panes, which certainly would not have been possible at a lower resolution because the frame here is very thin. By fitting lines to the open groups and employing the existing filtering in the vanishing point algorithm, Figure 6.3 shows that the longest lines do in fact form part of the window frame. The large number of horizontal

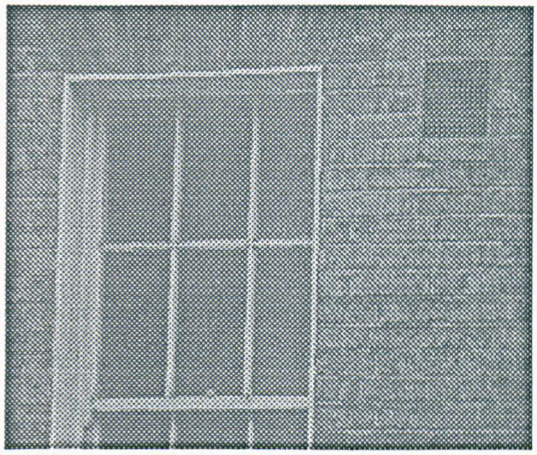


Figure 6.1 *Close up of a window*

lines are caused by the brickwork. Note also that the air vent to the upper right of the window, although not really visible, does have two vertical lines in Figure 6.3. By looking for line intersections of sets of parallel lines to make rectangles, it could be said that there is a potential feature here. For the first feature, extra computational effort can be used to identify it, for as soon as it is recognised, there is a key point from which to search for other features. The orthomap and CAD data is used to indicate what other features could be present and their positions relative to the feature. The identified feature would automatically yield scaling and directional information, the camera is then moved so as to have a good approximation of what may be present. (A computer controlled moveable camera mount that only has to be calibrated once for angles, can easily be used to search the building.). Also, if a complete feature is not visible, then corner information can indicate in which directions to move to bring a feature into view. Instead of using a

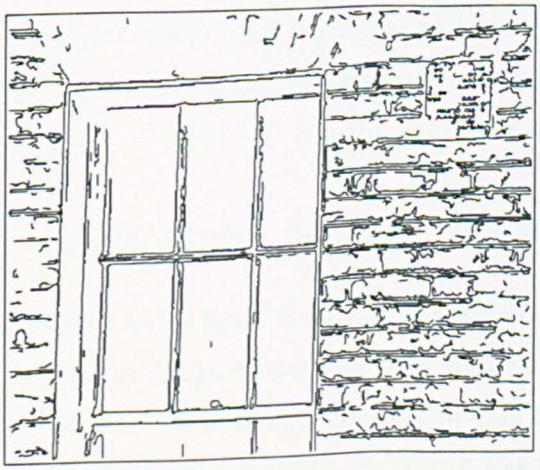


Figure 6.2 *Edges*

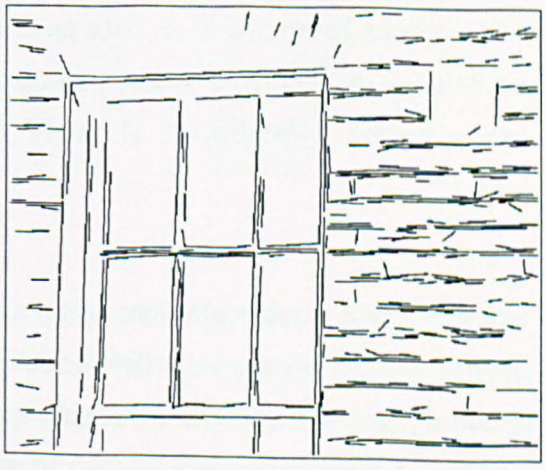


Figure 6.3 *Fitted and Filtered Lines*

moveable camera, one with a very high resolution CCD could be employed to take a single image. This would, however, add significantly to the cost. This approach would make interesting further work, with use of the various routines developed here.

6.1.2 Neural Networks / Fuzzy Logic

A single gray level image contains enough information to successfully identify all the visible features; after all, humans can manage that without too much difficulty, so why are there problems using edge detection alone? Clearly a feature consists of more than just its edges and the difficulty comes in trying to find what makes a window frame, a window frame. A human can be shown an unknown building and straight away pick out a window, because of having learned some set of characteristics about a window that makes it instantly recognisable as one. Neural networks are designed around the structure of the brain with the aim of mimicking it. A useful area of research would be to design a network and train it on a number of images of different windows. Spirkovska and Reid, (1994) have a network that can work with images which are rotated, scaled and translated and can be trained with only one view of an object. They use the outlines of aircraft and it may be possible to adapt this to windows frames. However, it would have to be more generalised and there is no coverage of distortion due to perspective. The images should only contain one window and then, if small areas of a building image are presented to the network, it would indicate that a window is present. Only then are the edge detection and other processes used to refine the position and type of window. This, if it worked, would have the advantage that no image processing needs to be done at the start and the network can work on the raw image. However, neural networks and the understanding of them is still in its infancy and this may be a rather ambitious task, although they are already being used in areas such as vehicle recognition. Alternatively, networks along with fuzzy logic could be used to search for particular patterns in an image such as an improved edge detector. By training with images marked with only the desired edges, a network may be able to perform a kind of filtered edge detection. There is considerable potential here.

6.1.3 Hardware / Parallel Processing

Hardware and parallel processing are worth pursuing once algorithms have been thoroughly investigated and shown to work reliably. The benefits are processing speed, which will certainly be the main requirement in any exploitation. Costs could be high, however. Specialist hardware needs designing and the use of parallel processing means increasing the amount of equipment used. Ideally, since most of the stages work in a serial manner,

Execution Times Routine	Time (mS)
Canny Edge (includes disk access)	208 166
Statistical Threshold	3 130
Link Pixels	8 513
Mark Object Levels	2 142
Find Containment	769
Closed Lines	2 197
Open Lines	5 328
Orthogonal Diameters	769
Arrange Lower Objects	55
Identify Robot	495
Vanishing Points	1 428
Identify Features	604
Reconcile	714
Find Four Points	549
Rectify	0
Locate Robot	0
Negative	714
Stretch Contrast	4 120
Histogram Equalisation	5 273
Mean 3 x 3	12 578
Canny Edge 256 x 256 (includes disk access)	49 871

Table 6.1 *Execution Times*

hardware would be designed for each task so that new images can be captured while previous ones are still being processed. In the current algorithm there are only a few processes which would benefit from parallel processing. The open and closed groups can be treated separately and some of the closed group processing can be done in any order with results being combined at the end. Table 6.1 shows the execution times of various functions when applied to Figure 4.6 which is 640 x 544 pixels in size. Clearly the Canny is by far the slowest routine and is the one that would need to be tackled first. The times given depend on the image and some extra functions are given for comparison.

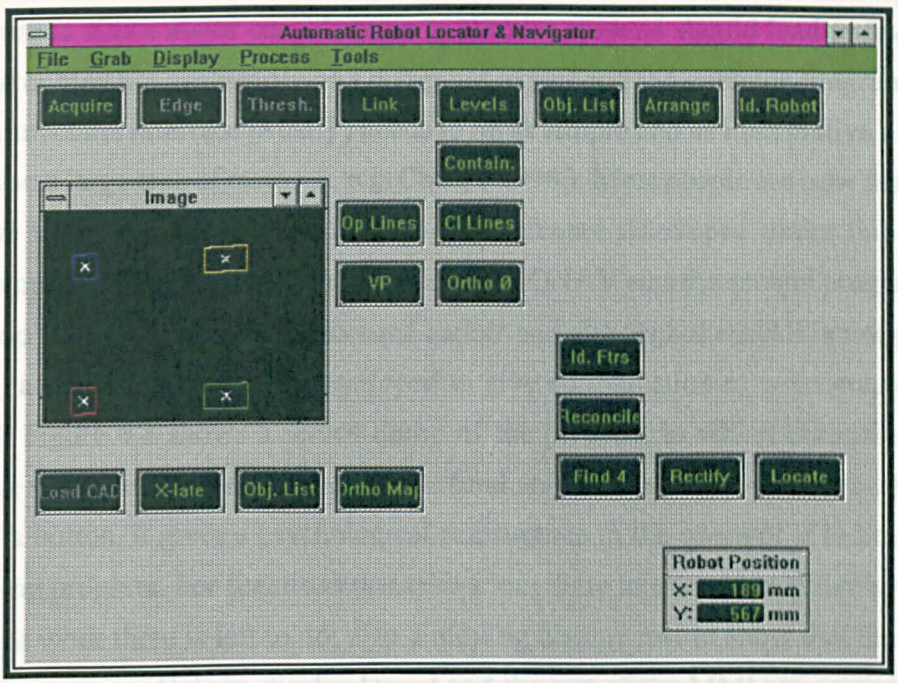


Figure 6.4 *Example of the ROBONAV Screen Display*

6.2 Exploitation and Practical Considerations

The ultimate objective of this research is to contribute to simpler and more cost effective ways of carrying out external building inspection. Since the end result is aimed at industry, exploitation considerations need to be taken into account as well as the research findings. For robot navigation to be fully automatic, many more man-years are required to develop and prove the algorithms and build hardware. In the meantime, what is important is to have a system working, particularly since navigation is part of a larger project. The best practical solution is to continue developing the ROBOLOC program mentioned in section 3.2. This makes the compromise of marrying the best of human perception ability to the precision of computer processing. A fully working system is close to completion and will give a result in its most basic form. As real inspections are carried out, modifications to the program can be used to enhance and simplify its performance by increasing the amount work done by the computer. This would allow development to continue while maintaining a working system. The enhancements may be in the form of local image processing. When the operator has selected a particular part of the image, the processor could effectively 'snap' (as in CAD packages) to the correct position, say a corner of a window, thus improving accuracy.

Ease of use is also a major consideration as simpler systems should lead to fewer errors. An operator may not necessarily be a computer expert and the operator-program interface (or man-machine interface [MMI]) needs to be as simple and as informative as possible. This is also a large area of research, e.g. Chua, (1996). Most people are now familiar with Windows based applications on PCs, so any software developed under this operating system would be an advantage. The program ROBONAV was the second program written to develop the ideas and algorithms used earlier and although the MMI is not part of the research, a partial interface was developed to give an indication of what may be used in the field. Figure 6.4 shows a screen dump of the program in which the main aspect to notice are the status buttons. Although the final software is meant to give a result at the touch of a button, it gives a psychological reassurance to the operator if indicators come on showing progress. For functions that take a long time to execute, a slider bar indicates how much work there is left to do; also a moving display just to show that the computer is doing something. Colour, display layout and the rest of the MMI are covered by work done by others.

7

Conclusions

The main aim of the research is to investigate the application of using image processing techniques to the problem of locating a building inspection robot. The work and the results presented here show that the principles of the algorithm worked and further investigations are required to improve the reliability on real buildings.

7.1 Main Conclusions

Referring back to the original aims stated in section 1.1 it is now possible to see how successfully each one was achieved. The first, second and third aims were accomplished by using a readily available video camera with single gray level images being captured on a PC. The results show conclusively that only a single camera is necessary and no particular setting up of the camera is required. Merely placing it at some convenient location is appropriate. The experiments were conducted with the camera at different distances from buildings (model and real) with the lens adjusted to provide the best picture. This satisfies the fourth aim by showing that no previous knowledge of the camera optics was required. The only target used in obtaining the robot's position was placed on the robot, indicating that it is unnecessary to place targets on the building, thus fulfilling aim five. The sixth aim was achieved since the target was successfully identified, and it was possible to locate a point on the target to the nearest pixel, inferring that the overall design of the target was satisfactory. The algorithm was able to distinguish between different windows on the model building and to classify them according to a simple CAD description of the windows. This, in turn, meant that the positions of the features and hence the pose of the building could be found satisfying aims seven and eight. The main innovation of the project was, therefore, demonstrated, namely the orthomap idea of locating features and that it has potential, to overcome the problems of perspective and distortion. It could be used in a variety of situations, with some improvements, without being restricted to one particular design of building. Other applications which may benefit from it, include the inspection of large structures such as storage tanks where the welds between plates may act as the features to be searched for. The final aim was realised by obtaining positions for the robot within the resolution of measurement (± 1 pixel) when the camera was normal to robot. Predictable errors (parallax) occurred which could be use to obtain a more accurate position for large camera angles.

Although, the results when the system functioned in its entirety were excellent (the position of the robot was found to within the measurement accuracy of ±1mm), there were often image processing failures which showed the weak points of the algorithm; the most significant being the detection of individual features. It was shown on several occasions that as few as 2 false pixels (out of 348 160) could be sufficient to cause failure if they occurred at a significant place. The method of edge detection followed by pixel linking worked satisfactorily for very simple features but was easily confused on real buildings by shadows, open windows, reflections and uneven surfaces which may partially obscure a feature when observed at an angle. The other main failing occurred when there were two vanishing points close to the image centre, as would occur if the camera was placed on the ground near the building at some angle to it and looking upwards. This had the effect of preventing the location of neighbouring features and hence the algorithm failed. The results showed that the algorithm, could, however cope with a single strong vanishing point.

Alternative methods, discussed in section 6.1.1, for feature matching suggest that a major difference from the present algorithm would be that a single feature could be searched for and positively identified. Extra time could be spent locating the first feature, but once found, it would provide the scale and orientation necessary to look for other features with simpler and quicker algorithms. The results also showed that features which occur near the edge of a building and form its outline are far more critical to success than features in the middle. Indirectly this has proven that the bounding edges of an object are far more significant in perception than the interior of an object and so an improved search algorithm could concentrate on finding these more significant features. Additionally, if the building edges can be detected (see the segmentation in Appendix 2), then it may be of benefit to give an extra weight to those features found near the building edge thus representing their significance. The orthomap method of looking for neighbours that occur in nearly orthogonal directions proved satisfactory and works well, except when features are placed in odd locations such as windows in stair wells. A map producing a mesh type grouping of nearest neighbours would overcome this problem. The vanishing points require further work to ensure that they are always correctly found, as the existing method started to fail once the angle between the visual horizontal/vertical lines and the true horizontal/vertical became large. This may occur if there are limitations in placing the camera, for example, a narrow street where camera angles would be similar to those in the results when the system failed. An alternative would be to use the feature shapes themselves to produce local vanishing points. Rectangles are very good indicators for searching for neighbours.

The research carried out here has laid the foundations for a machine vision based robot location system and has shown that correct results can be achieved. The system should be investigated further with improvements made to the various stages, particularly looking at alternatives to individual pixel based processing for detecting the boundaries of features.

7.2 Potential Improvements

During the development of this research, ideas arose which could not be investigated in the time scale. A number of enhancements are suggested here, concentrating on the image processing and analysis stages.

7.2.1 Edge Detection and Thresholding

This has turned out to be the most crucial area for determining the success of the operation. Chosen edges are found from thresholding the whole image and there is no precise way of determining what that threshold is. One way is to simulate the manual method of trying a level and seeing how well it turns out. The existing method would be used as a starting point and the number of identified features stored. Thresholds higher and lower than the starting threshold are then used to find new numbers of identified features. The process is repeated until a maximum number of features are identified. A simpler version of this was tried in section 4.3.5 which looked at the numbers of closed and open objects but did not check to see if the objects formed part of an identifiable feature. This adds significantly to the amount of processing required, but if implemented on a parallel system, different threshold levels could be checked simultaneously.

An alternative to thresholding, which covers the entire image uniformly, is to select the strongest edges and track along them. Although edges may have quite different strengths, some weak edges may be just as perceptually significant as the strong ones. A strong edge may even turn into a weak edge, the latter being deleted by the thresholding process with the possibility that a feature may not be detected.

7.2.2 Shape Detection

A more robust method of shape detection is needed. The current method of counting straight lines is suitable as a very coarse approximation, and a small perturbation can mean the difference in recognising or missing the robot target, for example. The signatures covered in section 4.4.2.8 are worth looking at for complete shapes but would add

significantly to the processing, as the problem is shifted to looking at the shape of a graph. Partial shapes may be completed by using gradient results from edge detection. So far, only the amplitude of the edge is used but the direction normal to the edge, that is, from light to dark, can be used in possible places of confusion where one shape occludes another. The Hough Transform mentioned in Appendix 1 can also be used for shapes as well as lines, for example, ellipses as in Tsuji and Matsumoto, (1978). An alternative method of finding arcs and ellipses is given in Rosin and West, (1989). Since the data at this point are only from a small part of the image, many of the problems associated with Hough would be eliminated.

7.2.3 Open Group Processing

The open groups have only been used so far for vanishing point detection. It may be possible to use the lines from these groups to detect the visible boundary of the building. Certainly, if the building has the sky as a background, then strong boundary edges will exist. If the outside edge of the building is known, then any ambiguities in the location of the features may be resolved. If there are two or more equally possible feature alignments, then the alignments can be compared to the probable building outline as a window cannot be in the sky, and a more reliable result achieved.

7.2.4 Control Points

The algorithm requires that only four control points are required. The locations of these four points was shown to make a difference in the final position calculated for the robot. It may be possible to obtain a more accurate result by using more control points since lens distortion cannot be mapped in this way. However, considering that a high degree of accuracy is not necessary, this needs only to be investigated for interest.

7.2.5 Image Aspect Ratio

Most images are presented to the human eye in 'landscape' form, that is, the picture is wider than it is tall. For image processing, there is no reason why a camera cannot be used on its side to give a 'portrait' view with the software simply exchanging the rows and columns to obtain the correct orientation. This would make better use of the pixels available since most buildings under inspection will probably be taller than they are wide. More detail can therefore be included in the image by being able to zoom in further.

7.2.6 Alternative Robot Detection

Although the purpose of this work was to investigate the possibility of finding the robot by 'seeing', other methods still using vision may be easier and less 'glamorous' to implement. The more distinctive the robot target is, the more easier it is to find. If colour were to be used (requiring a more expensive camera) then two bright colours next to each other which would not normally be found together could be used. Simple software colour filters would soon identify the robot.

Another approach is to use a changing target. A regularly flashing light is easy to detect by capturing different frames from the camera and subtracting them from each other. As long as the frame capture rate was similar to the frequency of the flashing light, the light would be the only feature to show up as a difference between two images. Several pairs of images can be used to eliminate a moving object. Image capture and difference calculations are fast operations and would be relatively easy to implement in hardware.

7.2.7 Image Capture

Specialist cameras are available giving a much higher resolution, for example, 1024 x 1024 pixels. For a single image, assuming good optics, this allows more of the small features to be visible. Also when looking at a narrow object, say part of the window frame, then the two sides will be distinguishable rather than appearing as a noisy line, which gives a false result at the edge detection stage. Also, since the edge detection and group selection stages are very susceptible to pixel noise, a technique commonly employed in astronomy could be investigated. Not all the CCD elements respond in the same way so with the camera lens completely covered, a black image can be captured. Ideally all the pixels should register the same amount (may not be 0 due to offsets and biases), but in reality there will be a difference. These pixels contribute to permanent noise in the image and the effects can be reduced by subtracting the black noisy image.

Appendix 1

Line Finding using the Hough Transform

This appendix covers the considerable amount of work done with the Hough Transform, which proved not to be central to the main research. Although a promising technique, for reasons given here, it was decided not to implement it. The technique, called the Hough Transform (HT), was popularised by Duda and Hart, (1972) and is widely used today. However, it will be seen that it tends to fail in certain situations, particularly, in images of tower blocks.

The HT works by searching the edge image for pixels and then fitting all possible lines through the pixels. Each line can be described by two parameters which form the two axes of 'parameter space'. Every time a given line passes through a certain pixel, the corresponding cell in the parameter space is incremented. If several pixels lie in a straight line, the cell for that line will have a higher number of counts or 'votes'.

Usually the equation of a line is given as $y = mx + c$, so the two parameters describing it are **m** and **c**. A problem arises for near vertical lines when the values for **m** and **c** approach infinity. This would require an infinite parameter space which is not possible. An alternative is to use angle and distance parameters since the angle of a line can only range between 0^o and 180^o, and the distance of the line from the centre of the image is no more than the distance to a corner of the image. The equation of the line becomes:

$$\rho = x * \cos(\theta) + y * \sin(\theta) \qquad [\text{A1.1}]$$

where ρ is the distance from the origin and θ is the angle. This is shown diagrammatically in Figure A1.1.

Taking the simple case of the rectangle in Figure A1.2, consisting of a pair of horizontal- and vertical lines, then the resulting parameter space is shown in negative form in Figure A1.3. The two dark peaks in the centre of the parameter space correspond to the two horizontal lines and the peaks right at the edge (which tend to wrap round the parameter space) correspond to the two vertical lines. In this instance, the peaks are well defined and the found lines will have parameters given by the locations of the peaks.

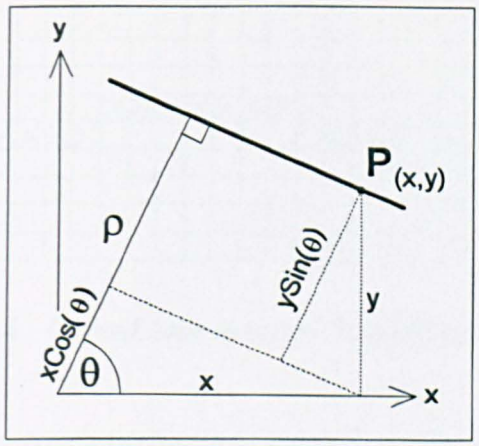


Figure A1.1 *Definition of Line Parameters*

A1.1 Practical Problems Encountered with the Hough Transform

The first and simplest problem is that the lines found in the parameter space do not have any length information. The main interest is in line segments and further processing is required to trace along a given line to pick out the segments. One of the main attractions of the HT is its ability to fill in missing data. Lines with pixels missing can be found; however, it then becomes a question of whether a gap is valid or not and this leads to the first problem encountered with building images. A typical building may have a whole line of features such as windows all at the same height with a fairly small gap between them. The HT would interpret the edge output as having a strong (many votes) line with a few gaps. The question is should these gaps be filled in or do they occur at the corners of the feature?

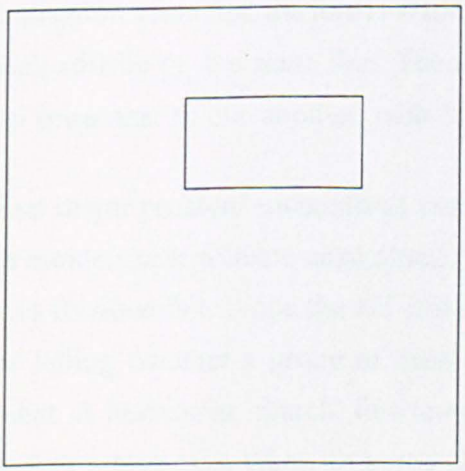


Figure A1.2 *Image of Rectangle*

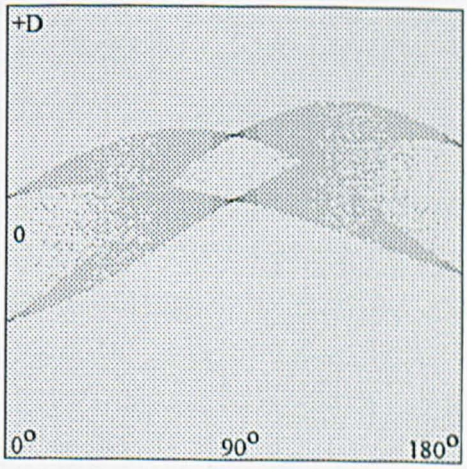


Figure A1.3 *Hough Transform of Rectangle*

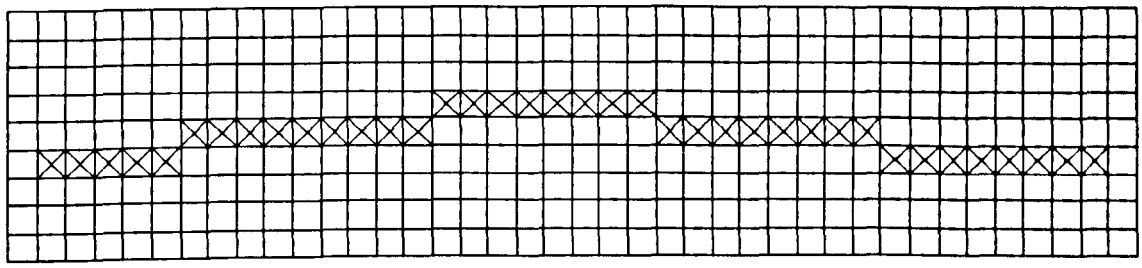


Figure A1.4 *Curved Line made of Straight Line Segments*

A similar problem encountered with building features is that of close, parallel or nearly parallel lines as are particularly likely to occur at window frames. Examples of the HT in literature such as Leavers, (1992: 68) often show some simple object with well defined and separated lines. In these cases the voting peaks are easy to locate. However, two parallel lines near each other translate to two adjacent peaks in parameter space with the consequence that they tend to merge together. Coupled with noise, particularly in the form of quantisation, it becomes difficult to distinguish the two peaks with the result that possibly only one line is found. Also if the resultant peak is in a slightly different position, the equivalent line may have a slightly different angle which means that away from the centre of the line, it may be some distance away from the pixels that contributed to it.

Images taken using conventional cameras with flat sensing devices suffer from a degree of distortion, most noticeable when wide angle lenses are used. This results in real straight lines appearing as curved lines in the image, particularly away from the centre. Although this may not be perceptible, a deviation of a few pixels may nevertheless exist over the length of the line. Figure A1.4 shows an exaggerated curved horizontal line. The effects of quantisation mean that the line is made up of a number of straight line segments, some of which still lie on the same line. The result is that the HT sees these as a number of parallel lines near to one another, with the problems mentioned above.

The final major problem encountered was caused by having a complex image containing a large number of legitimate edge pixels. The advantage of the HT in this situation now becomes its downfall. Since the HT has no actual knowledge of connectivity, it has no way of telling whether a group of pixels actually form a line or are merely a chance alignment. A horizontal 'search' line may cross a large number of vertical line segments resulting in a high vote when no perceivable line exists. For complex images, this was particularly the case where false lines appeared along the diagonals of the image. This occurs because if the pixels were spread at random, then more would lie on the diagonal

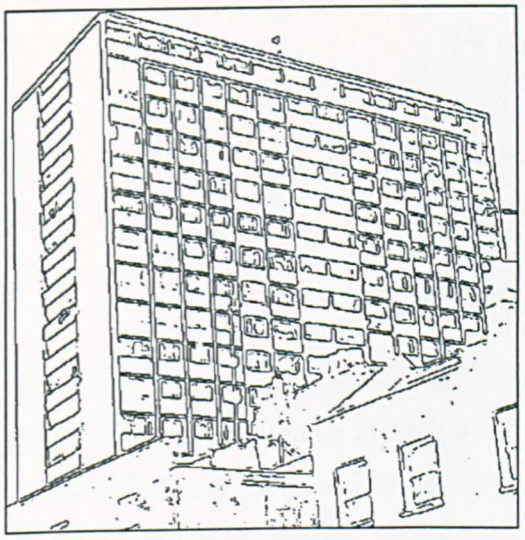


Figure A1.5 *Complex Edge Image*

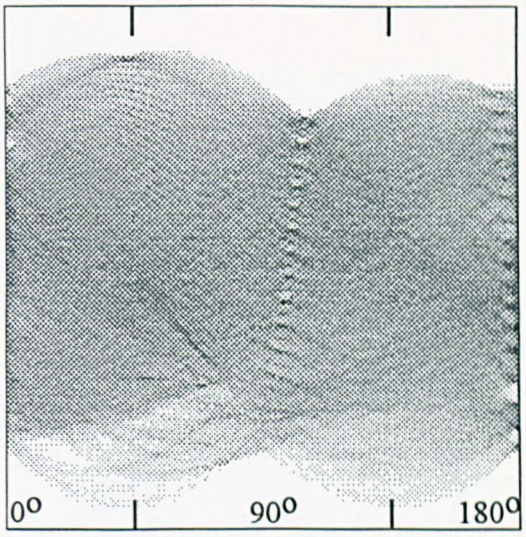


Figure A1.6 *Hough Transform with False Peaks*

lines than any other direction. Figure A1.5 shows a complex edge image with Figure A1.6 showing the corresponding HT in negative form. It can be seen that there are several false groupings corresponding to angles of 45^{o} and 135^{o} - the angles of the diagonals of a square image.

A1.2 Hough Parameter Space Enhancements

In an ideal situation, a desired line would be found in parameter space by a single peak. The way the HT works however tends to form ridges hence the appearance of the peaks in Figure A1.3. If a peak is zoomed into, it is seen that it is more than a single isolated point. Table A1.1 shows individual parameter space cells for an image containing a single horizontal line of length 512 pixels with a **y** value of 300. It is clear that the peak lies on a ridge and contains two 'bumps' giving three maxima in total. The two bumps with values of 116 must not be detected as lines. Filters can be used to enhance the true peaks and one developed by Leavers, (1992: 70) known as the Butterfly filter is shown in

Hough Space		Theta															
		82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97
Rho	296	0	0	0	0	0	0	0	0	0	114	56	38	30	24	20	16
	297	0	0	0	0	0	0	0	0	0	114	58	38	28	22	18	16
	298	8	2	0	0	0	0	0	0	0	116	56	38	28	24	20	16
	299	14	18	16	14	12	8	6	4	0	112	54	32	20	10	4	0
	300	14	16	20	24	28	40	58	114	512	0	0	0	0	0	0	0
	301	14	16	18	22	28	38	56	114	0	0	0	0	0	0	0	0
	302	16	16	20	22	30	38	58	116	0	0	0	0	0	0	0	0
	303	14	16	20	24	28	38	58	114	0	0	0	0	0	0	0	0

Table A1.1 *Close up of a Single Line Peak in Hough Space*

0	-2	0
1	2	1
0	-2	0

Figure A1.7 *Butterfly Filter*

Figure A1.7. The actual values can be altered to eliminate certain lines and the results of using the filter are shown in Figures A1.8 and A1.9 where Figure A1.8 shows the lines detected before any filtering is applied to the parameter space. Figure A1.9 shows that more lines have been detected and the number of false lines bunching together has been reduced.

The problem of false lines appearing is partly caused by aliasing and is discussed in Kiryati, (1989) although a simple improvement suggested by Leavers, (1993) is to use a vote of 1 as usual for the given cell but to also add a vote of 0.5 to the adjacent cells in the ρ direction of parameter space.

There are no doubt other enhancements that can allow lines to be more easily detected but this would form a research project in its own right. The fact that no solution has yet been found suggests that the HT should only be used in certain circumstances and other line finding algorithms used for complex images.

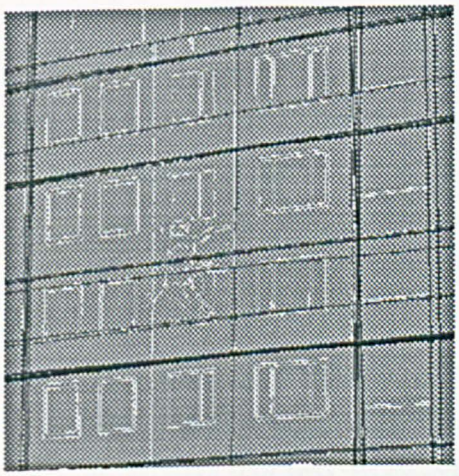


Figure A1.8 *Hough Lines*

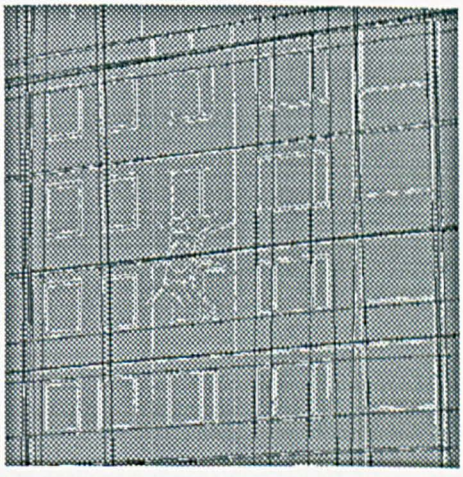


Figure A1.9 *Hough Lines with Butterfly Filter and Anti-aliasing*

Appendix 2

Image Pixel Intensity Segmentation

One of the problems encountered in edge detection is the break up real edges and the inclusion of extra edges resulting from noise. If the number of gray levels in an image are reduced, it then takes on a somewhat 'chunky' appearance but with well defined edges. Very simple edge detection can be used to merely search for a difference in adjacent pixels resulting in continuous single pixel edges. Two methods were investigated, namely global segmentation and local segmentation.

A2.1 Global Segmentation.

Global segmentation uses a method essentially the same as flood filling used in computer graphics. A pixel is chosen and tagged as visited and its value is stored. All neighbouring pixels that have a value within a certain range of the original pixel are stored. The first pixel in the list is chosen and all its neighbours, less previously tagged ones, are checked. This process continues until there are no untagged neighbours left. Figure A2.2 shows the results when the intensity range is set to ±10 gray levels and applied to figure A2.1. The background sky has been successfully segmented, which may well be good for finding the edges of the building; but on the building itself, nothing else has. A problem occurs if the intensity of a region changes beyond the limits set, perhaps by a shadow or the region is textured. Another method tried was to use local segmentation.

A2.2 Local Segmentation

This method works in the same way as global segmentation but instead of storing the original pixel value, the current pixel value is used and neighbours are only considered if they are within a certain range of this pixel. This solves intensity variations as only local changes are taken into account. The results of this method are shown in Figure A2.3 with a neighbour intensity range of ±4 gray levels. There is an improvement in that the local variations have been eliminated but a new problem occurs from 'leaking'. If for some reason noise is present at a perceived boundary or an intensity edge is not sharp, then the segmentation process will leak from one region to another resulting in the large patches seen, possible erasing desired features. Further research is required into the many different types of region segmentation to see if they are of any potential use.

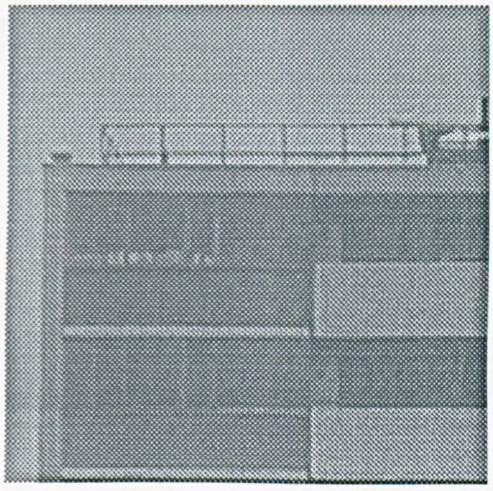


Figure A2.1 *Raw Image of Part of a Tower Block*

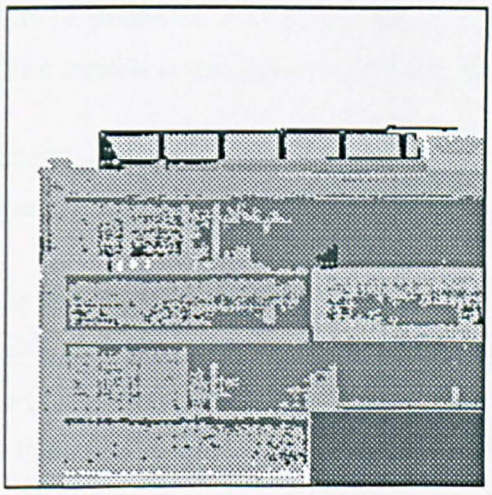


Figure A2.2 *Globally Segmented Image*

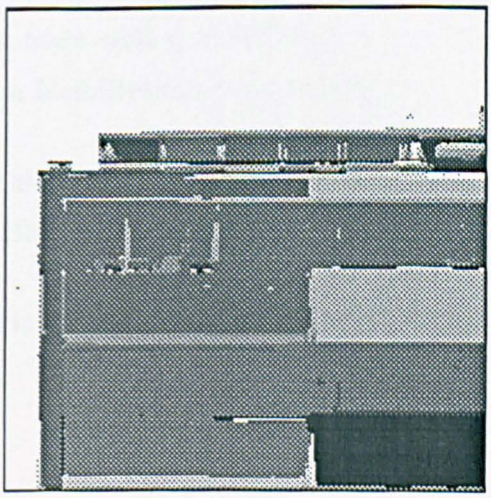


Figure A2.3 *Locally Segmented Image*

Appendix 3

DXF File Specification

The following sections describing the specification of the DXF file are taken from Autodesk, (1990: 527-557) and cover the necessary steps for extracting line information:

"To assist in interchanging drawings between AutoCAD and other programs, a Drawing Interface file format (DXF™) has been defined". "DXF files are standard ASCII text files. They can easily be translated to the formats of other CAD systems or submitted to other programs for specialised analysis."

General File Structure

A Drawing Interchange File is simply an ASCII text file with a file type of *.dxf* and specially formatted text. The overall organisation of a DXF file is as follows:

1. HEADER section - General information about the drawing is found in this section of the DXF file. Each parameter has a variable name and an associated value.

2. TABLES section. - This section contains definitions of named items.
    * Linetype table (LTYPE)
    * Layer table (LAYER)
    * Text Style table (STYLE)
    * View table (VIEW)
    * User Co-ordinate System table (UCS)
    * Viewport configuration table (VPORT)
    * Dimension Style table (DIMSTYLE)
    * Application Identification table (APPID)

3. BLOCKS section - This section contains Block Definition entities describing the entities comprising each Block in the drawing.

4. ENTITIES section - This section contains the drawing entities, including any BLOCK references.

5. END OF FILE

A DXF file is composed of a multiplicity of *groups*, each of which occupies two lines in the DXF file. The first line of a group is a group *code*, which is a positive nonzero integer output in FORTRAN I3 format (that is, right-justified and blank filled in a three-character field). The second line of the group is the *group value*, in a format which depends on the type of the group as specified by the group code.

The specific assignment of the group code depends upon the item being described in the file. However, the type of the value this group supplies is derived from the group code in the following way:

Group Code range	Following value	Group Code range	Following value
0 - 9	String	999	Comment (string)
10 - 59	Floating-point	1000 - 1009	String
60 - 79	Integer	1010 - 1059	Floating-point
210 - 239	Floating-point	1060 - 1079	Integer

Table A3.1 *Group Code Ranges*

Thus a program can easily read the value following a group code without knowing the particular use of this group in an item in the file. The appearance of values in the DXF file is not always affected by the setting of the Units command: co-ordinates are always represented as decimal (or possibly scientific notation if very large) numbers, and angles are always represented in decimal degrees with zero degrees to the east of the origin. Co-ordinate values always appear to be in inches.

Variables, table entries, and entities are described by a group that introduces the item, giving its type and/or name, followed by multiple groups that supply the values associated with the item. In addition, special groups are used for file separators such as markers for the beginning and end sections, tables, and the file itself.

Entities, table entries, and file separators are always introduced with a 0 group code that is followed by a name describing the item.

Group Codes

Group codes are used both to indicate the type of the value of the group, as explained earlier, and to indicate the general use of the group. The specific function of the group code depends on the actual variable, table item, or entity description. This section indicates the general use of groups, noting those as '(fixed)' and which always have the same function.

Group code	Value type
0	Identifies the start of an entity, table entry, or file separator. The text value that follows indixcates what it is
1	The primary text value for an entity.
2	A name; Attribute tag, Blockname, and so on.
3 - 4	Other textual or name values.
5	Entity handle expressed as a hexadecimal string.
6	Line type name (fixed).
7	Text style name (fixed).
8	Layer name (fixed).
9	Variable name identifier (used only in HEADER section of the DXF file).
10	Primary X coordinate (start point of a Line or Text entity, center of a circle, etc.).
11 - 18	Other X coordinates.
20	Primary Y coordinate. 2n values always correspond to 1n values and immediately follow them in a file.
21 - 28	Other Y coordinates.
30	Primary Z coordinate. 3n values always correspond to 1n values and 2n values and immediately follow them in a file.
31 - 37	Other Z coordinates.
38	This entity's elevation if nonzero (fixed). Exists only in output from versions prior to R11.
39	This entity's thickness if nonzero (fixed).
40 - 48	Floating-point values (text height, scale factors, etc.).
49	Repeated value - multiple 49 groups may appear in one entity for variable length tables (such as the dash lengths in the LTYPE table). A 7x group always appears before the first 49 group to specify the table length.
50 - 58	Angles.
62	Color number (fixed).
66	"Entities follow" flag (fixed).
70 - 78	Integer values, such as repeat counts, flag bits, or modes.
210, 220, 230	X, Y, and Z components of extrusion direction.
999	Comments.

Table A3.2 *Entity Group Codes...*

Group code	Value type
1000	ASCII string up to 255 bytes long.
1001	Registered application name (ASCII string up to 31 bytes long).
1002	XDATA Control string.
1003	XDATA Layer name.
1004	Chunk of bytes (up to 127 bytes long).
1005	XDATA Database handle.
1010, 1020, 1030	XDATA X, Y, and Z coordinates.
1011, 1021, 1031	XDATA X, Y, and Z coordinates of 3D World space position.
1012, 1022, 1032	XDATA X, Y, and Z components of 3D World space displacement.
1013, 1023, 1033	XDATA X, Y, and Z components of 3D World space direction.
1040	XDATA Floating-point value.
1041	XDATA Distance value.
1042	XDATA Scale factor.
1070	XDATA 16-bit integer.
1071	XDATA 32-bit signed long.

Table A3.2 *Continued*

Comments

The 999 group code indicates that the following line is a comment string. Thus the 999 group can be used to include comments in a DXF file which has been edited. For example:

```
999
This is a comment.
999
This is another comment.
```

HEADER Section

The HEADER section of the DXF file contains settings of variables associated with the drawing. These variables are set with various commands and are the type of information displayed by the Status command. Each variable is specified in the header section by a 9 group giving its name, followed by groups that supply its value. The following list shows the header variables and their meanings.

Variable	Type	Description
$ACADVER	1	The AutoCAD drawing database version number; AC1006 = R10, AC1009 = R11.
$ANGBASE	50	Angle 0 direction.
$ANGDIR	70	1 = clockwise angles, 0 = counterclockwise.
$ATTDIA	70	Attribute entry dialogues, 1 = on, 0 = off.
$ATTMODE	70	Attribute visibility: 0 = none, 1 = normal, 2 = all.
$ATTREQ	70	Attribute prompting during Insert, 1 = on, 0 = off.
$AUNITS	70	Units format for angles.
$AUPREC	70	Unites precision for angles.
$AXISMODE	70	Axis on if nonzero.
$AXISUNIT	10,20	Axis X and Y tick spacing.
$BLIPMODE	70	Blip mode on if nonzero.
$CECOLOR	62	Entity colour number; 0 = BYBLOCK, 256 = BYLAYER.
$CELTYPE	6	Entity linetype name, or BYBLOCK or BYLAYER.
$CHAMFERA	40	First chamfer distance.
$CHAMFERB	40	Second chamfer distance.
$CLAYER	8	Current layer name.
$COORDS	70	0 = static coordinate display, 1 = continuous update, 2 = "d<a" format.
$DIMALT	70	Alternate unit dimensioning performed if nonzero.
$DIMALTD	70	Alternate unit decimal places.
$DIMALTF	40	Alternate unit scale factor.
$DIMAPOST	1	Alternate dimensioning suffix.
$DIMASO	70	1 = create associative dimensioning, 0 = draw individual entities.
$DIMASZ	40	Dimensioning arrow size.
$DIMBLK	2	Arrow block name.
$DIMBLK1	1	First arrow block name.
$DIMBLK2	1	Second arrow block name.
$DIMCEN	40	Size of centre mark/lines.
$DIMCLRD	70	Dimension line colour, range is 0 = BYBLOCK 256 = BYLAYER.
$DIMCLRE	70	Dimension extension line colour, range is 0 = BYBLOCK, 256 = BYLAYER.
$DIMCLRT	70	Dimension text colour, range is 0 = BYBLOCK, 256 = BYLAYER.
$DIMDLE	40	Dimension line extension.
$DIMDLI	40	Dimension line increment.
$DIMEXE	40	Extension line extension.
$DIMEXO	40	Extension line offset.
$DIMGAP	40	Dimension line gap.
$DIMLFAC	40	Linear measurements scale factor.
$DIMLIM	70	Dimension limits generated if nonzero.
$DIMPOST	1	General dimensioning suffix.
$DIMRND	40	Rounding value for dimensioning distances.
$DIMSAH	70	Use separate arrow blocks if nonzero.
$DIMSCALE	40	Overall dimensioning scale factor.
$DIMSE1	70	First extension line suppressed if nonzero.
$DIMSE2	70	Second extension line suppressed if nonzero.
$DIMSHO	70	1 = Recompute dimensions while dragging. 0 = drag original image.

$DIMSOXD	70	Suppress outside-extensions dimension lines if nonzero.
$DIMSTYLE	2	Dimension style name.
$DIMTAD	70	Text above dimension line if nonzero.
$DIMTFAC	40	Dimension tolerance display scale factor.
$DIMTIH	70	Text inside horizontal if nonzero.
$DIMTIX	70	Force text inside extensions if nonzero.
$DIMTIM	40	Minus tolerance.
$DIMTOFL	70	If text outside extensions, force line extensions between extensions if nonzero.
$DIMTOH	70	Text outside horizontal if nonzero.
$DIMTOL	70	Dimension tolerances generated if nonzero.
$DIMTP	40	Plus tolerance.
$DIMTSZ	40	Dimensioning tick size: 0 = no ticks.
$DIMTVP	40	Text vertical position.
$DIMTXT	40	Dimensioning text height.
$DIMZIN	70	Zero suppression for "feet & inch" dimensions.
$DRAGMODE	70	0 = off, 1 = 0n, 2 = auto.
$ELEVATION	40	Current elevation set by Elev command.
$EXTMAX	10,20, 30	X, Y, and Z drawing extents upper right corner (in WCS).
$EXTMIN	10,20, 30	X, Y, and Z drawing extents lower-left corner (in WCS).
$FILLETRAD	40	Fillet radius.
$FILLMODE	70	Fill mode on if nonzero.
$HANDLING	70	Handles enabled if nonzero.
$HANDSEED	5	Next available handle.
$INSBASE	10,20, 30	Insertion base set by Base command (in WCS).
$LIMCHECK	70	Nonzero if limits is on.
$LIMMAX	10, 20	XY drawing limits upper-right corner (in WCS).
$LIMMIN	10,20	XY drawing limits lower-left corner (in WCS).
$LTSCALE	40	Global linetype scale.
$LUNITS	70	Units format for co-ordinates and distances.
$LUPREC	70	Units precision for co-ordinates and distances.
$MAXACTVP	70	Sets maximum number of viewports to be regenerated.
$MENU	1	Name of menu file.
$MIRRTXT	70	Mirror text if nonzero.
$ORTHOMODE	70	Ortho mode on if nonzero.
$OSMODE	70	Running object snap modes.
$PDMODE	70	Point display mode.
$PDSIZE	40	Point display size.
$PELEVATION		
$PEXTMAX	40	Maximum X, Y, and Z extents for paper space.
$PEXTMIN	40	Minimum X, Y, and extents for paper space.
$PLIMCHECK	70	Limits checking in paper space when nonzero.
$PLIMMAX	40	Maximum X and Y limits in paper space.
$PLIMMIN	40	Minimum X and Y limits in paper space.
$PLINEWID	40	Default polyline width.
$PUCSNAME	2	Current paper space UCS name.
$PUCSORG	10, 20,30	Current paper space UCS origin.
$PUCSXDIR	10,20,30	Current paper space UCS X axis.
$PUCSYDIR	10,20,30	Current paper space UCS Y axis.
$QTFXTMODE	70	Quick text mode on if nonzero.

$REGENMODE	70	Regenauto mode on if nonzero.
$SHADEDGE	70	0 = faces shaded, edges not highlighted. 1 = faces shaded, edges highlighted in black. 2 = faces not filled, edges in entity colour. 3 = faces in entity colour, edges in black.
$SHADEDIF	70	Percent ambient/diffuse light, range 1 - 100, default 70.
$SKETCHINC	40	Sketch record increment.
$SKPOLY	70	0 - sketch lines, 1 - sketch polylines.
$SPLFRAME	70	Spline control polygon display, 1 = on, 0 = off.
$SPLINESEGS	70	Number of line segments per spline patch.
$SPLINETYPE	70	Spline curve type for Pedit Spline.
$SURFTAB1	70	Number of mesh tabulations in first direction.
$SURFTAB2	70	Number of mesh tabulations in second direction.
$SURFTYPE	70	Surface type for Pedit Smooth.
$SURFU	70	Surface density (for Pedit smooth) in M direction.
$SURFV	70	Surface density (for Pedit smooth) in N direction.
$TDCREATE	40	Date/time of drawing creation.
$TDINDWG	40	Cumulative editing time for this drawing.
$TDUPDATE	40	Date/Time of last drawing update.
$TDUSRTIMER	40	User elapsed timer.
$TEXTSIZE	40	Default text height.
$TEXTSTYLE	7	Current text style name.
$THICKNESS	40	Current thickness set by Elev command.
$TILEMODE	70	1 for previous release compatibility mode, 0 otherwise.
$TRACEWID	40	Default Trace width.
$UCSNAME	1	Name of current UCS.
$UCSORG	10,20,30	Origin of current UCS (in WCS).
$UCSXDIR	10,20,30	Direction of current UCS's X axis (in WCS).
$UCSYDIR	10,20,30	Direction of current UCS's Y axis (in WCS).
$UNITMODE	70	Low bit set = display fractions, feet-and-inches, and surveyor's angles in input format.
$USERI1-5	70	Five integer variables intended for third-party use.
$USERR1-5	40	Five real variables intended for third party use.
$USRTIMER	70	0 = timer off, 1 = timer on.
$WORLDVIEW	70	1 = set UCS to WCS during Dview/Vpoint, 0 = don't change UCS.

Example

Most of the above parameters look rather confusing and are not likely to be used in this application, where basically a line drawing is all that is required to represent one face of a building. In reading a DXF file, if only the lines are of interest, for example, it becomes merely a case of identifying the appropriate groups in order to read them and jump past them to only pick out the line entities. The following file shows the significant part of a DXF file representing a 100mm x 50mm rectangle drawn on layer 0 at position 75mm,75mm relative to the bottom-left corner of a sheet of A4 paper orientated in portrait format. The text 'Rectangle' is written on layer 1 and is placed near the top of the page. Comments added later are in [square brackets]. Further details can be found from the AutoCAD manual.

```
0                       [ HEADER section ]
SECTION
2
HEADER
9                       [ Global line scale = 0,5 ]
$LTSCALE
40
0.5
9                       [ Lower-left drawing limits ]
$LIMMIN
10                      [ X = 0,0 ]
0.0
20                      [ Y = 0,0 ]
0.0
9                       [ Upper-right drawing limits ]
$LIMMAX
10                      [ X =  210,82mm or 8,3" ]
8.3
20                      [ Y =  297,18mm or 11,7" ]
11.7
9                       [ Lower-left drawing extent in WCS ]
$EXTMIN
10
0.0
20
0.0
9                       [ Upper-right drawing extent in WCS ]
$EXTMAX
10
8.3
20
11.7
9                       [  Current layer name is 1 ]
$CLAYER
8
1
9                       [ Entity colour number  is 7 ]
$CECOLOR
62
7
9                       [ Entity linetype name is CONTINUOUS ]
$CELTYPE
6
CONTINUOUS
0
ENDSEC
0                       [ TABLES section ]
SECTION
2
TABLES
0                       [ Table LTYPE  has 10 items ]
TABLE
2
```

```
LTYPE
70
10
0                       [ LTYPE CONTINUOUS ]
LTYPE
2
CONTINUOUS
70
64
3                       [ Line description = 'Solid line' ]
Solid line
72                      [ Alignment code = 65 ]
65
73                      [ Number of dash lengths = 0 ]
0
40                      [ Total pattern length = 0 ]
0.0
.
.
.
0
ENDTAB
0
TABLE                   [ Table LAYER ]
2
LAYER
70                      [ Layer is on and thawed ]
256
0                       [ LAYER 1 ]
LAYER
2
1
70                      [ on and thawed ]
0
6                       [ Linetype CONTINUOUS is colour 1 ]
CONTINUOUS
62
1
.
.
.
0
ENDTAB
0
ENDSEC
0                       [ ENTITIES section - the drawing proper ]
SECTION
2
ENTITIES
0                       [ Line entity ]
LINE
8                       [Layer name = 1 ]
1
```

```
6                                   [ Line type is CONTINUOUS ]
CONTINUOUS
62                                  [ Colour = 1 ]
1
10                                  [ From X = 75mm ]
2.95275590549863E+0000
20                                  [ From Y = 75mm ]
2.95275590549863E+0000
11                                  [ To X = 75mm ]
2.95275590549863E+0000
21                                  [ To Y = 125mm ]
4.92125984250015E+0000
0                                   [ Line from 75,125mm to 175,125mm ]
LINE
8
1
6
CONTINUOUS
62
1
10
2.95275590549863E+0000
20
4.92125984250015E+0000
11
6.88976377950166E+0000
21
4.92125984250015E+0000
0                                   [ Line from 175,125mm to 175,75mm ]
LINE
8
1
6
CONTINUOUS
62
1
10
6.88976377950166E+0000
20
4.92125984250015E+0000
11
6.88976377950166E+0000
21
2.95275590549863E+0000
0                                   [ Line from 175,75mm to 75,75mm ]
LINE
8
1
6
CONTINUOUS
62
1
```

```
10
6.88976377950166E+0000
20
2.95275590549863E+0000
11
2.95275590549863E+0000
21
2.95275590549863E+0000
0                       [ Text entity ]
TEXT
8                       [ Layer name = 1 ]
1
62                      [ Colour 1 ]
1
10                      [ X position = 59,3mm ]
2.33636707746336E+0000
20                      [ Y position = 256,0mm ]
1.00758467478054E+0001
40                      [ Height 20mm ]
7.87401574800242E-0001
1                       [ Text string is 'Rectangle' ]
Rectangle
7                       [ Text style name is 'STANDARD' ]
STANDARD
11                      [ Alignment point X =  59,3mm ]
2.33636707746336E+0000
21                      [ Alignment point Y =  265,9mm ]
1.04695475352055E+0001
50                      [ Angle = 0 degrees ]
0.00000000000000E+0000
41                      [ Relative X scale factor ]
3.67135835028439E-0001
0
ENDSEC
0                       [ End of file ]
EOF
```

Appendix 4

Development Platform

The work presented here was developed using the following hardware:

IBM compatible PC consisting of:

- 486 DX 33MHz processor
- 8MB Ram
- Windows 3.1 operating system

Vidi-PC 24 frame grabber

- 256 gray level mode
- 640 x 544 pixels (dual interlaced grab)
- S-VHS and composite video inputs
- Supplied by:

  Rombo Ltd.
  Kirkton Campus
  Livingston
  SCOTLAND
  EH54 7A2
  Tel: +44 (0)1506 41 46 31

Panasonic S-VHS video camera

And the following software:

Microsoft Visual C++ Version 1.0 including the
Windows Software Development Kit

Vidi-PC 24 Toolkit 'C' library routines from Rombo Ltd.

References

ABE Y., TANAKA K., TANAKA Y., FUKUDA T., ARAI F. and ITO S. (1994). Development of air conditioning equipment inspection robot with vision based navigation system. In: AUTOMATION AND ROBOTICS IN CONSTRUCTION XI, Brighton UK, 1994. Proceedings. Amsterdam: Elsevier Science B.V., pp. 665-674.

ÅMDAL, K. and METRONOR, A.S. (1992). Single Camera System for Close Range Industrial Photogrammetry. In: ISPRS INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY AND REMOTE SENSING, Vol. XXIX, Part. B5, Commission V, 1992, Proceedings. Committee of the XVII International Congress for Photogrammetry and Remote Sensing, pp. 6-10.

ASHIKAWA, M., ADACHI, T., KATANO, H., MIYASHITA, T. and MATSUI, S. (1992). Development of Wall Coating Removal Robot. In: THE 9TH INTERNATIONAL SYMPOSIOUM ON AUTOMATION AND ROBOTICS IN CONSTRUCTION, Tokyo, Japan, Vol. 2, June 1992, Proceedings. Japan Industrial Robot Association, pp. 565-572.

AUTODESK. (1990). AutoCAD Release 11 Reference Manual. (Autodesk Inc.).

BEARDSLEY, P.A., REID, I.D., ZISSERMAN, A. and MURRAY, D.W. (1994) Active visual navigation using non-metric structure. Technical Report No. OUEL 2047/94. Oxford University Robotics Group, UK.

BENNING, W. and SCHWERMANN, R. (1995). Automatic Orientation in Close Range Photogrammetry Using Straight Lines. In: OPTICAL 3-D MEASUREMENT TECHNIQUES III - APPLICATIONS IN INSPECTION, QUALITY CONTROL AND ROBOTICS, Vienna, Austria, Oct 1995. Proceedings. Wichman, pp181-193.

BERGEVIN, R. and LEVINE, M.D. (1993). Generic Object Recognition: Building and Matching Coarse Descriptions from Line Drawings. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.15, No.1, Jan 1993. IEEE, pp19-36.

BIAO, G.C. (1994). Extraction of Line using Morphological Processing and Line Linking. In: THE THIRD INTERNATIONAL CONFERENCE ON AUTOMATION, ROBOTICS AND COMPUTER VISION, Vol. 1, Singapore, Nov 1994. Proceedings. The Institution of Engineers, Singapore, pp. 158-161.

BIEDERMAN I., COOPER E.E., HUMMEL J.E. and FISER J. (1993). Geon Theory as an Account of Shape Recognition in Mind, Brain, and Machine. In: 4th BRITISH MACHINE VISION CONFERENCE, Vol 1, Guildford, UK, 1993. Proceedings. pp. 175-186.

BILLINGSLEY J. and SCHOENFISCH M. (1994). A Vision System for Agricultural Tractor Guidance. In: THE THIRD INTERNATIONAL CONFERENCE ON AUTOMATION, ROBOTICS AND COMPUTER VISION, Vol. 3, Singapore, Nov 1994. Proceedings. The Institution of Engineers, Singapore, pp. 1941-1944.

BHANDARKAR, S.M., ZHANG, Y. and POTTER, W.D. (1994). An Edge Detection Technique Using Genetic Algorithm Based Optimization. Pattern Recognition Vol.27, No.9. Elsevier Scientific Ltd, pp1159-1180.

BLEAKLEY, G.J. and CHAMBERLAIN, D.A. (1994). NDT Tasking Robot for Concrete and Steel Structures. In: MECHATRONICS AND MACHINE VISION IN PRACTICE, Toowoomba, Australia, 1994. Proceedings. IEEE, pp145-150.

BRILLAULT-O'MAHONY, B. (1991) New Method for Vanishing Point Detection. In: CVGIP: Image Understanding, Vol.54, No.2, Sep 1991. Academic Press Inc. pp289-300.

BROADHURST, S.J., PRIDMORE, T.P. and TAYLOR, N. (1994). Sensing for feature identification in sewers. In: AUTOMATION AND ROBOTICS IN CONSTRUCTION XI, Brighton UK, May 1994. Proceedings. Amsterdam: Elsevier Science B.V., pp. 675-682.

BROWN, D.C. (1994). New Developments in Photogeodesy. In: Photogrammetric Engineering and Remote Sensing, Vol.60, No.7, July 1994. American Society for Photogrammetry and Remote Sensing. pp877-894.

BRYSON, N., DIXON, R.N., HUNTER, J.J. and TAYLOR (1994). Contextual classification of cracks. In: Image and Vision Computing Vol.12, No.3, Apr 1994. Butterworth-Heinemann Ltd. pp149-154.

CAELLI, T. (1993). Texture Classification and Segmentation Algorithms in Man and Machines. In: Spatial Vision Vol.7, No.4, 1993. VSP, pp277-292.

CAMPBELL, N.W. and THOMAS, B.T. (1992). The Trapezium as an Image Processing Primitive. In: 4TH INTERNATIONAL CONFERENCE ON IMAGE PROCESSING AND ITS APPLICATIONS, Maastricht, The Netherlands, 1992. Proceedings. IEE. pp151-154.

CANNY J. (1986). A Computational Approach to Edge Detection. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 8, No 6, 1986. IEEE, pp. 679-698.

CEB COMITE EURO-INTERNATIONAL DU BETON. (1989). Diagnosis and Assessment of Concrete Structures State of the Art Report, Bulletin d'Information No. 192, (Secretariat permanent: Case Postale 88, CH-1015 Lausanne).

CHAMBERLAIN, D.A. and BLEAKLEY, G.J. (1994). CURIO NDT Robot for the Construction Industry. Industrial Robot, Vol. 21, No. 4. MCB University Press: 29-31.

CHUA J. (1996). Man-Machine Interface and Non-Destructive Testing for a Building Inspection Robot. PhD Thesis: City University, London, Department of Civil Engineering.

CLARKE, T.A. (1994). An analysis of the properties of targets used in digital close range photogrammetric measurement. In: VIDEOMETRICS III, Boston, USA, Nov. 1994. SPIE - The International Society for Optical Engineering, pp251-262.

CLARKE, T.A. and WANG, X. (1995). An analysis of subpixel target location using Fourier Transform based models. In: VIDEOMETRICS IV, Philadelphia, USA, Oct. 1995. SPIE - The International Society for Optical Engineering, pp77-78.

COHEN, L.D., and COHEN, I. (1993). Finite Element Methods of Active Contour Models and Balloons for 2-D and 3-D Images. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.15, No.11, Nov 1993. IEEE, pp1131-1147.

COOPER, J., VENKATESH, S. and KITCHEN, L. (1993). Early Jump-Out Corner Detectors. In: IEEE Transactions on Pattern Analysis and Machine Intelligence - Correspondence, Vol.15, No.8, Aug 1993. IEEE, pp823-828.

COOPER, M.C. (1993). Interpretation of line drawings of complex objects. In: Image and Vision Computing Vol.11, No.2, Mar 1993. Butterworth-Heinemann Ltd. pp82-90.

CSÁKI, G. (1990). Application of Computer-assisted Photogrammetry in the Documentation of Excavations and Care of Monuments in Egypt. In: CLOSE-RANGE PHOTOGRAMMETRY MEETS MACHINE VISION, Zürich, Switzerland, SPIE Vol. 1395, Sept. 1990. Proceedings. The International Society for Optical Engineering. pp. 948-955.

CUSACK, M.M. and THOMAS, J.G. (1992). Wall Climbing Robot for the Inspection and Maintenance of Buildings. In: THE 9TH INTERNATIONAL SYMPOSIOUM ON AUTOMATION AND ROBOTICS IN CONSTRUCTION, Tokyo, Japan, June 1992, Proceedings. Japan Industrial Robot Association, pp. 555-564.

DOIHARA T., HIRONO K., KAZUO and ODA. (1992). Crack Measuring System Based on Hierarchical Image Processing Technique. In: ISPRS INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY AND REMOTE SENSING, Vol. XXIX, Pt. B5, Commission V, 1992. Proceedings. Committee of the XVII International Congress for Photogrammetry and Remote Sensing, pp. 155-159.

DUDA R.O. and HART P.E. (1972). Use of the Hough Transform to Detect Lines and Curves in Pictures. In: COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, GRAPHICS AND IMAGE PROCESSING, Vol. 15, No. 1, Jan 1972. Proceedings. pp. 11-15.

DULIMATRA, H.S. and JAIN, A.K. (1994). Mobile Robot Localization in Indoor Environment. In: THE THIRD INTERNATIONAL CONFERENCE ON AUTOMATION, ROBOTICS AND COMPUTER VISION, Vol. 3, Singapore, Nov 1994. Proceedings. The Institution of Engineers, Singapore, pp. 2204-2208.

EDMUNDSON, K. and NOVAK, K. (1992). On-Line Triangulation for Autonomous Vehicle Navigation. In: ISPRS INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY AND REMOTE SENSING Vol. XXIX. Pt. B5. Commission V, 1992. Proceedings. Committee of the XVII International Congress for Photogrammetry and Remote Sensing, pp. 916-922.

ELLIS, T. (1993). City University, London. Personal Communication.

ENS, J. and LAWRENCE, P. (1993). An Investigation of Methods for Determining Depth from Focus. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.15, No.2, Feb 1993, IEEE, pp97-108.

ERNST, B. (1985). The Magic Mirror of M.C.Escher. (Diss, England: Tarquin Publications).

FERRARI, F., FOSSA, M., GROSSO, E., MAGRASSI, M. and SANDINI, G. (1991). A Practical Implementation of a Multilevel Architecture for Vision-Based Navigation. In: FIFTH INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS, Vol. 2, Pisa, Italy, 1991. Proceedings., IEEE: New York, pp. 1092-1098.

FISCHLER, M.A. and WOLF, H.C. (1994). Locating Perceptually Salient Points in Planar Curves. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 16, No 2, Feb 1994. IEEE. pp. 113-129.

FUKUDA, T., ITO, S., OOTA, N., ARAI, F., ABE, Y., TANAKA K., and TANAKA Y. (1993). Navigation System based on Ceiling Landmark Recognition for Autonomous Mobile Robot. In: IECON,93 ROBOTICS, VISION, AND SENSORS; AND SIGNAL PROCESSING AND CONTROL 1993. Proceedings. IEEE, pp. 1466-1471.

GEORGOPOULUS, A. and TOURNAS, E. (1994). Digital Rectification Using a PC. In: COMMISSION V SYMPOSIUM - CLOSE RANGE TECHNIQUES AND MACHINE VISION, Melbourne, Australia, March, 1994. Proceedings. Australian Photogrammetric and Remote Sensing Society. pp. 102-108.

GINIGE, A. (1992). A Unified Approach to Image Feature Detection Using Finite State Machines. In: 4th INTERNATIONAL CONFERENCE ON IMAGE PROCESSING AND ITS APPLICATIONS, Maastricht, The Netherlands, 1992. Proceedings. IEE, London, pp. 486-489.

GIRAUDON, G.and DERICHE, R. (1991). On Corner and Vertext Detection, In: IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, Maui, Hawaii, June 1991. Proceedings. IEEE Computer Society Press, pp. 650-655.

GONZALEZ R.C. and WOODS R.E. (1993). Digital Image Processing, (Addison-Wesley Publishing Co.).

HAIFENG, Y. and MÄKELÄ, H. (1991). A New Edge Extracting Method for Vehicle Navigation. In: FIFTH INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS, Vol. 2, Pisa, Italy, 1991. Proceedings. IEEE: New York, pp. 1039-1044.

HELLWICH, O. and FAIG, W. (1994). Graph-Based Feature Matching Using Descriptive and Realtional Parameters. In: Photogrammetric Engineering and Remote Sensing, Vol.60, No.4, July 1994. American Society for Photogrammetry and Remote Sensing, pp. 443-450.

v.d. HEUVAL, F.A., KROON, R.J.G.A and LE POOLE, R.S. (1992). Digital Close-Range Photogrammetry using Artificial Targets. In: ISPRS INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY AND REMOTE SENSING Vol. XXIX, Pt. B5, Commission V, 1992. Proceedings. Committee of the XVII International Congress for Photogrammetry and Remote Sensing, pp. 900-915.

HIGGINS, W.E., and HSU, C. (1994). Edge Detection using Two-Dimensional Local Structure Information. Pattern Recognition Vol.27, No.2. Elsevier Scientific Ltd, pp. 277-294.

HOCK, C., BEHRINGER, R. and THOMANEK, F. (1994). Intelligent Navigation for a Seeing Road Vehicle using Landmark Recognition. In: PROCEEDINGS OF THE COMMISSION V SYMPOSIUM - CLOSE RANGE TECHNIQUES AND MACHINE VISION, Melbourne, Australia, March, 1994. Australian Photogrammetric and Remote Sensing Society, pp83-190.

HOCK, C and DICKMANNS, E.D. (1992). Intelligent Navigation for Autonomous Robots using Dynamic Vision. In: ISPRS INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY AND REMOTE SENSING Vol. XXIX, Pt. B5, Commission V, 1992. Proceedings. Committee of the XVII International Congress for Photogrammetry and Remote Sensing, pp. 900-915.

HUANG, Y.D. and HARLEY, I. (1990). CCD Camera Calibration Without a Control Field In: CLOSE-RANGE PHOTOGRAMMETRY MEETS MACHINE VISION, SPIE Vol. 1395, Zürich, Switzerland, Sept. 1990. Proceedings. The International Society for Optical Engineering, pp1028-1034.

JVC (1992). Explanation of S-VHS and the enhanced Professional S, (JVC Professional Products (UK) Ltd.).

KAHN P., KITCHEN, L., and RISEMAN, E.M. (1990). A Fast Line Finder for Vision-Guided Robot Navigation. In: IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE Vol.12, Nov. 1990. pp. 1098-1102.

KENDALL, G.D. and HALL, T.J. (1992). Performing Fundamental Image Processing Operations using Quantised Neural Networks. In: 4TH INTERNATIONAL CONFERENCE ON IMAGE PROCESSING AND ITS APPLICATIONS, Maastricht, The Netherlands, 1992. Proceedings. IEE. pp. 226-229.

KIRYATI, N. and BRUCKSTEIN, A.M. (1989). Antialiasing the Hough Transform. In: THE 6TH SCANDINAVIAN CONFERENCE ON IMAGE ANALYSIS, Oulu, Finland, June 1989. Proceedings. pp. 621-628.

KONSTANTINIDES, K. and RASURE, J.R. (1994). The Khoros Software Development Environment for Image and Signal Processing. In: IEEE Transactions on Image Processing Vol.3, No.3, May 1994. IEEE, pp43-252.

KUMAR, S., RANGANATHAN, N. and GOLDGOF, D. (1994). Parallel Algorithms for Circle Detection in Images. Pattern Recognition Vol.27, No.8, Elsevier Scientific Ltd, pp. 1019-1028.

LEAVERS V.F. (1992). Shape Detection in Computer Vision Using the Hough Transform, (London: Springer-Verlag).

LEAVERS V.F. (1993). King's College, London. Personal Communication.

LEE, C.M., PONG, T.C., SLAGLE, J.R. and ESTERLINE, A. (1994). An Experimental Study of an Object Recognition System that Learns. Pattern Recgnition Vol. 27, No.1, Elsevier Scientific Ltd., pp. 65-89 .

LEE, K.Y., WITTEFELDT, P., McHUGH, P. and DISSAYANAKE M.W.M.G. (1994). Development of a Legged-Wheeled Mobile Robot. In: MECHATRONICS AND MACHINE VISION IN PRACTICE, Toowoomba, Australia, 1994, Proceedings, IEEE, pp. 166-169.

LENNOX S.C. and CHADWICK M. (1979). Mathematics for Engineers & Applied Scientists, 2nd Ed., (Heinemann Educational Books Ltd.).

LEYMARKE, F. and LEVINE, M.D. (1993). Tracking Deformable Objects in the Plane using an Active Contour Model. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.15, No.6, June 1993, IEEE, pp. 617-634.

LOWE D.G. (1987). Three-dimensional object recognition from single two-dimensional images. In: Artificial Intelligence, Vol.31, 1987, Elsevier Science Publications B.V., pp. 355-395

MÄKELÄ, H. and KOSKINEN, K. (1991). Navigation of Outdoor Mobile Robots Using Dead Reckoning and Visually Detected Landmarks. In: FIFTH INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS, Vol. 2, Pisa, Italy, 1991. Proceedings. New York: IEEE, pp. 1051-1056.

MALLETT, G.P. (1994). State-of-the-Art Review - Repair of Concrete Bridges. Transport Research Laboratory

MANGOLD, M., FRIEDMANN, M. and RAMMELKAMP B. (1995). Infrared Thermography to Localise Anchors in Three-Layer Outside Walls. In: INTERNATIONAL SYMPOSIUM NON-DESTRUCTIVE TESTING IN CIVIL ENGINEERING (NDT-CE), Vol. 1, Berlin, Germany, Sept 1995, Proceedings. Deutsche Gesellschaft für Zerstörungfreie Prüfung e.V., pp53-360.

MARION A. (1991). An Introduction to Image Processing, English Ed., (London: Chapman & Hall).

v.d. MERWE, N. and RÜTHER, H. (1994). Image Matching Through a Combination of Feature- and Area Based Matching. In: PROCEEDINGS OF THE COMMISSION V SYMPOSIUM - CLOSE RANGE TECHNIQUES AND MACHINE VISION, Melbourne, Australia, March, 1994. Australian Photogrammetric and Remote Sensing Society, pp. 407-413.

MIRMEHDI, M. and ELLIS, T.J. (1993). Parallel approach to tracking edge segments in dynamic scenes. In: Image and Vision Computing Vol.11, No.1, Jan 1993. Butterworth-Heinemann Ltd., pp. 35-48.

MIURA M., MIYAJIMA T., MIURA S., OGASSAWARA K. (1991). An Automated Measuring System of Cracks in Concrete which uses Image Processing Techniques and Artificial Intelligence Theory. In: 8TH INTERNATIONAL SYMPOSIUM ON AUTOMATION AND ROBOTICS IN CONSTRUCTION, Vol. 2, Stuttgart, Germany, June 1991. Proceedings. Copydruck GmbH, pp. 631-638.

NELSON, R.C. (1994). Finding Line Segments by Stick Growing. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 16, No 5, May 1994. IEEE, pp. 519-523.

NISHIHARA, K.H. and POGGIO, T. (1984). Stereo Vision for Robotics. Robotics Research ISBN 0-262-02207, pp. 489-505.

OCS GROUP LTD. (1995). Autonomous Robotic Cleaning of Windows - ARCOW. Company Literature. 79 Limpsfield Road, Sanderstead, Surrey, CR2 9LB.

PARK, J.P., NAM, K.M. and PARK, R.H. (1994). Edge Detection in Noisy Images based on the Co-Occurence Matrix. Pattern Recognition Vol.27, No.6, Elsevier Scientific Ltd., pp. 765-777.

PATERSON, A.M. (1995). Personal notes from a study visit to the Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen, Universität Stuttgart.

PATERSON, A.M., DOWLING, G.R. and CHAMBERLAIN D.A. (1994a). Identifying Key Features in a Building Using a Single Uncalibrated Camera. In: AUTOMATION AND ROBOTICS IN CONSTRUCTION XI, Brighton UK, 1994. Proceedings. Amsterdam: Elsevier Science B.V., pp. 657-664.

PATERSON, A.M., DOWLING, G.R. and CHAMBERLAIN D.A. (1994b). Locating a Building Inspection Robot on a Tower Block using Visual Methods. In: THE THIRD INTERNATIONAL CONFERENCE ON AUTOMATION, ROBOTICS AND COMPUTER VISION, Singapore, Vol.3, Nov 1994, Proceedings. Photoplates Pte Ltd., pp. 1412-1413.

PATERSON, A.M., DOWLING, G.R. and CHAMBERLAIN D.A. (1995). The Science and Engineering of Robot Location. In: 5th INTERNATIONAL CONFERENCE ON IMAGE PROCESSING AND ITS APPLICATIONS, Edinburgh UK, 1995. Proceedings. IEE: 410, pp. 692-696.

PICARD, R.W., and GORKANI, M. (1994). Finding perceptually dominant orientations in natural textures. In: Spatial Vision Vol.8, No.2, 1994. VSP, pp. 221-253.

PORTECH - Portsmouth Technology Consultants Ltd. (1995). Company Literature. 70 Solent Road, Havant, PO9 1JH.

PRITSCHOW, G., DALACKER, M., KURZ, J. and GÄNSSLE, M. (1995). Technological Aspects in the Development of a Mobile Bricklaying Robot. In: 12TH INTERNATIONAL SYMPOSIUM ON AUTOMATION AND ROBOTICS IN CONSTRUCTION, Warsaw, Poland, May 1995. Proceedings. IMBiGS, pp. 281-290.

QIAN, W. and TITTERINGTON, D.M. (1993). Bayesian Image Restoration: An Application to Edge-Preserving Surface Recovery. In: IEEE Transactions on Pattern Analysis and Machine Intelligence - Correspondance, Vol.15, No.7, July 1993. IEEE, pp. 748-752.

RAMESH, V. and HARALICK, R.M. (1992). Performance Characterization of Edge Detectors. In: APPLICATIONS OF ARTIFICIAL INTELLIGENCE X: MACHINE VISION AND ROBOTICS, Vol. 1708, Orlando, USA, April 1992. Proceedings. SPIE - The International Society for Optical Engineering, pp. 252-266.

ROBERTS, L.G. (1965). Machine perception of three-dimensional solids. In: Optical and Electro-Optical Information Processing. MIT Press, pp. 159-197.

RÖSCH, A. and SCHAAB, A. (1995). An in-situ NDT-method to Detect Incorrectly Positioned Dowel Bars in Carriageway Slabs of Concrete Highways. In: INTERNATIONAL SYMPOSIUM NON-DESTRUCTIVE TESTING IN CIVIL ENGINEERING (NDT-CE), Vol. 1, Berlin, Germany, Sept 1995, Proceedings. Deutsche Gesellschaft für Zerstörungfreie Prüfung e.V., pp. 269-276.

ROSENTHALER, L., HEITGER, F., KÜBLER, O. and VON DER HEYDT, R. (1992). Detection of General Edges and Keypoints. In: COMPUTER VISION - ECCV '92 SECOND EUROPEAN CONFERENCE ON COMPUTER VISION, Vol 588, Santa Margherita Ligure, Italy, May 1992. Proceedings. Springer-Verlag, pp. 78-86.

ROSIN, P.L. and WEST, G.A. (1989). Segmentation of Edges into Lines and Arcs. In: IMAGE AND VISION COMPUTING, Vol.7, 1989. Proceedings. pp. 109-114.

SALAGNAC, J-L. and VINOT, B. (1991). Mobile Robots on Building Construction Sites: Specifications Based on Tasks Requirements and Geometrical Analysis. In: 8TH INTERNATIONAL SYMPOSIUM ON AUTOMATION AND ROBOTICS IN CONSTRUCTION, Vol. 2, Stuttgart, Germany, June 1991. Proceedings. Copydruck GmbH, pp021-1027.

SARKAR, S. and BOYER, K.L. (1991) Optimal Infinite Impulse Response Zero Crossing Based Edge Detectors. In: CVGIP: Image Understanding, Vol.54, No.2, Sep 1991. Academic Press Inc., pp. 224-243.

SARR, D.P. (1992). Scratch Measurement System Using Machine Vision: Part II. In: APPLICATIONS OF ARTIFICIAL INTELLIGENCE X: MACHINE VISION AND ROBOTICS, Vol. 1708, Orlando, USA, April 1992. Proceedings. SPIE - The International Society for Optical Engineering, pp. 811-818.

SIVALOGANATHAN, S., JEBB, A. and WYNN, H.P. (1991). Processing Isometric Free-Hand Sketches in Computer-Aided Design In: 6TH INTERNATIONAL CONFERENCE ON CAD/CAM, ROBOTICS AND FACTORIES OF THE FUTURE, London, UK, Aug 1991. Proceedings. Southbank Press, pp. 46-53.

SONG K.Y., PETROU M. and KITTLER J. (1992). Wigner Based Crack Detection in Textured Images. In: 4th INTERNATIONAL CONFERENCE ON IMAGE PROCESSING AND ITS APPLICATIONS. 1992. Proceedings. IEE, pp. 315-318.

SONKA, M., HLAVEC, V. and BOYLE, R. (1994). Image Processing, Analysis and Machine Vision, (London: Chapman & Hall).

SPIRKOVSKA, L. and REID, M.B. (1994). Higher-Order Neural Networks Applied to 2D and 3D Object Recognition. In: Machine Learning, Vol.15, 1994. Kluwer Academic Publishers, pp. 169-199.

STRAFORINI, M., COELHO, C. and CAMPANI, M. (1990). A fast and precise method to extract vanishing points. In: CLOSE-RANGE PHOTOGRAMMETRY MEETS MACHINE VISION, SPIE Vol.1395, Zürich, Switzerland, 1990. Proceedings. The International Society for Optical Engineering, pp. 266-274.

STRAFORINI, M., COELHO, C. and CAMPANI, M. (1993). Extraction of vanishing points from images of indoor and outdoor scenes. In: Image and Vision Computing Vol.11, No.2, Mar 1993. Butterworth-Heinemann Ltd., pp. 91-99.

STREILEIN, A. (1995). Videogrammetry and CAAD for Architectural Restitution of the Otto-Wagner-Pavillion in Vienna. In: OPTICAL 3-D MEASUREMENT TECHNIQUES III - APPLICATIONS IN INSPECTION, QUALITY CONTROL AND ROBOTICS, Vienna, Austria, Oct 1995. Proceedings. Wichman, pp. 305-314.

TAI, A., KITTLER, J., PETROU, M. and WINDEATT, T. (1993). Vanishing Point Detection. In: Image and Vision Computing Vol.11, No.4, May 1993. Butterworth-Heinemann Ltd., pp. 240-245.

TANAKA, K., KAJITANI, M., KANAMORI, C., ITOH, H., ABE, Y. and TANAKA, Y. (1995). Development of Marking Robot Working at Building Sites. In: 12TH INTERNATIONAL SYMPOSIUM ON AUTOMATION AND ROBOTICS IN CONSTRUCTION, Warsaw, Poland, May 1995, Proceedings. IMBiGS, pp. 235-242.

TEWFIK, A.H. and DERICHE, M. (1993). An Eigenstructure Approach to Edge Detection. In: IEEE Transactions on Image Processing Vol.2, No.3, July 1993. IEEE, pp. 353-368.

THOMPSON, M.M. (1966). Manual of Photogrammetry, 3rd Ed., Vol. 2, (American Society of Photogrammetry).

TILLOTSON, H.T., SNAITH, M.S. and TACHTSI, V. (1993). The Integrity of Road Condition Data, Unpublished paper, School of Civil Engineering, The University of Birmingham.

TRIBE, R. (1992). City University Computer Science Seminar - Advance technologies for automotive collision avoidance. Nov. 1992. Lucas Automotive Ltd.

TSENG, D.C. and HUANG, M.Y. (1993). Automatic thresholding based on human visual perception. In: Image and Vision Computing Vol.11, No.9, Nov 1993. Butterworth-Heinemann Ltd., pp. 539-548.

TSUJI, S. and MATSUMOTO F. (1978). Detection of Ellipses using Modified Hough Transformation. In: IEEE T-COMP, Vol.27

WECKESSER, P., GASTINEL, F. and DILLMANN, R. (1995). Advances in Mobile Robot Navigation Using Digital Cameras and Photogrammetric Algorithms. In: OPTICAL 3-D MEASUREMENT TECHNIQUES III - APPLICATIONS IN INSPECTION, QUALITY CONTROL AND ROBOTICS, Vienna, Austria, Oct 1995, Proceedings. Wichman, pp. 69-77.

WILLSON, R.G. (1994). Modelling and Calibration of Automated Zoom Lenses. In: VIDEOMETRICS III, Boston, USA, Nov. 1994. Proceedings. SPIE - The International Society for Optical Engineering, pp. 170-186.

WONG, K.C. and KITTLER, J. (1993). Recognizing polyhedral objects from a single perspective view. In: Image and Vision Computing Vol.11, No.4, May 1993. Butterworth-Heinemann Ltd., pp. 211-220.

WONG, T. (1994), Lewisham Housing, London. Personal Communication.

XIN, K., LIM, K.B. and HONG, G.S. (1994). Image Feature Extraction Through Scale-Space Filtering. In: THE THIRD INTERNATIONAL CONFERENCE ON AUTOMATION, ROBOTICS AND COMPUTER VISION, Vol. 1, Singapore, 1994. Proceedings. The Institution of Engineers, Singapore, pp. 11-14.

YAGI, Y., KAWATO, S. and TSUJI, S. (1994). Real-Time Omni Directional Image Sensor (COPIS) for Vision-Guided Navigation. In: IEEE Transactions on Robotics and Autmation, Vol.10, No.1, Feb 1994. IEEE, pp. 11-22.

YAU, C.T. and WONG, K.H. (1994). Stereo Correspondance Using Horizontal Line Segments from Intensity Images. In: THE THIRD INTERNATIONAL CONFERENCE ON AUTOMATION, ROBOTICS AND COMPUTER VISION, Vol.3, Singapore, Nov 1994. Proceedings. Photoplates Pte Ltd., pp. 424-428.