TRINOCULAR STEREO 3D PARTS LOCATION

Craig A. Morrison

Department of Artificial Intelligence

University of Edinburgh

May 26, 1989

ABSTRACT

Recent work has shown that stereopsis with three cameras improves image correspondence and is more successful at locating objects in 3D world space accurately than the equivalent two camera system. [Ayache & Lustman 1986] This report documents a project to investigate this claim by constructing a trinocular stereo system to locate parts in 3D.

The motivation behind the work is presented and the work is placed in the context of similar work that has been carried out. A description of the theory and implementational details is then presented before the conclusions of the project are discussed and work that could follow from it suggested.

Examination of the results produced by the system confirms Ayache & Lustman's claim that the technique of trinocular stereo does indeed improve image correspondence and is successful at locating objects in 3D.

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENTS

Thanks are due to the following;

Dr. Bob Fisher for invaluable support and advice throughout the duration of the project. His knack for turning up when he was most needed was uncanny and much appreciated.

Dr. John Hallam was a rich source of information on the more obscure matters that arose and the benefit of an ever-open door can never be underestimated.

Li Dong Cai for helping me extend my limited grasp of geometry.

The workshop boys led by Tom Alexander and David Wyse were always approachable no matter how strange the request and proved most helpful throughout the project.

The Department of Artificial Intelligence for providing the resources and backup which enabled this project to be carried out.

CHAPTER 1

INTRODUCTION

The following quote from David Marr's classic book "Vision" sums up neatly the purpose of the project described in the rest of this report;

"Vision is the process of discovering from images what is present in the world, and where it is." [D. Marr 1982]

The human visual system long ago mastered this process with the development of the retina, the optic nerve and the visual cortex. These tissues make up the visual pathway which allows us to *see*. It has long been the goal of many workers in the field of artificial intelligence to model this visual process using computers.

If this could be achieved, the resultant machines could be used for such diverse application areas as, industrial assembly and inspection, automated medical x-ray screening, monitoring of earth resources by remote sensors and control of autonomous vehicles. They could assist in many tasks that are either routine, tedious or dangerous for humans to perform such as bomb disposal or nuclear power plant work. It is not difficult to see the attraction of developing such systems. [Nevatia 1982]

However, the development of such systems is a huge undertaking and there are major problems to be solved. One such problem is that of obtaining the depth of objects in a scene. One of the most popularly offered solutions to this problem and one which is biologically motivated is that of binocular stereopsis.

Ballard and Brown (1982) present the following *precis* of the technique;

1) Take two images separated by a baseline.
2) Identify identical points in the two images.
3) Use the principal of triangulation to derive the depth of the point.

They also identify stage 2 as the hardest part of the technique. This problem is termed the correspondence problem and many approaches have been suggested to solve it. This report documents the construction of a system designed to alleviate this correspondence problem by adding information from a third image. The work is motivated by a claim made by Ayache and Lustman (1987) that stereopsis using three cameras is more reliable and produces more accurate results because of the use of the third camera to overcome the correspondence problem.

An impression of how a system would appear can be obtained with reference to figure 1.1.

The scene is set-up on a table top using a metal frame to provide support for the cameras and is lit from the roof by a normal fluorescent tube light.

The only constraint that is enforced is that the object under test must have some polyhedral edges. This means that unlike many other stereo systems no special lighting is required and the cameras may be placed in any arrangement.

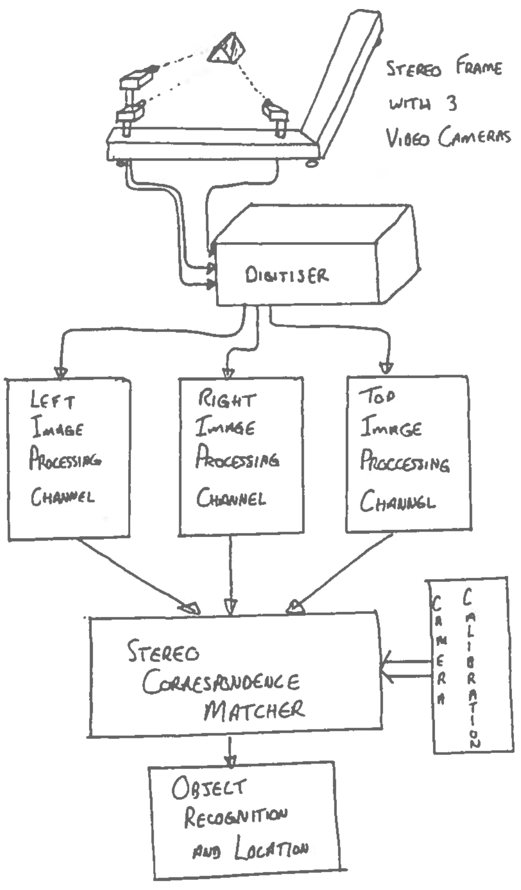

STEREO FRAME WITH 3 VIDEO CAMERAS
DIGITISER
LEFT IMAGE PROCESSING CHANNEL
RIGHT IMAGE PROCESSING CHANNEL
TOP IMAGE PROCESSING CHANNEL
STEREO CORRESPONDENCE MATCHER
CAMERA CALIBRATION
OBJECT RECOGNITION AND LOCATION

Figure 1.1 - System Overview

The images are obtained from the cameras before undergoing a series of image processing operations. This stage is called the monocular image processing and carries out such operations as noise reduction, edge detection and corner location which result in the production of a set of line segments from each image describing the 2D location of the edges of any object in the image.

These sets of line segments are then passed to the stereo matching phase of the operation where an attempt is made to form matches between line segments from different images which correspond to the same object edge before being combined using the process of triangulation to produce the 3D co-ordinates describing the object edge. Therefore, the result of the stereo matching is a set of 3D line segments in the world space describing the object in the original images.

The conversion of image co-ordinates to world co-ordinates is only realistic if certain parameters describing the cameras are available. The camera calibration routines determine these parameters which are utilised by the stereo matcher.

The set of 3D line segments describing the object in the real world is the input to the object recognition and location phase of the system. The recogniser compares the description of the object supplied to it with its database of models and picks the most successful match. Using this model and the 3D image edges, the locator then calculates the most plausible position of the object.

The advantages of using three cameras instead of the more normal binocular set-up become apparent in the stereo matching phase introduced above.

The major advantage of trinocular stereo over its binocular counterpart is its solution to the correspondence problem. It can be seen in figure 1.2 that the search for corresponding points between images is restricted to a line, while in figure 1.3 the search is reduced to a mere verification at a point in the third image. This addition of a

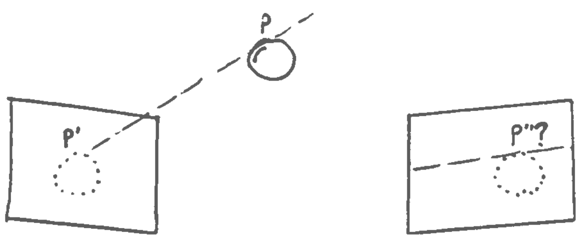


Figure 1.2 - Corresponding points in two images

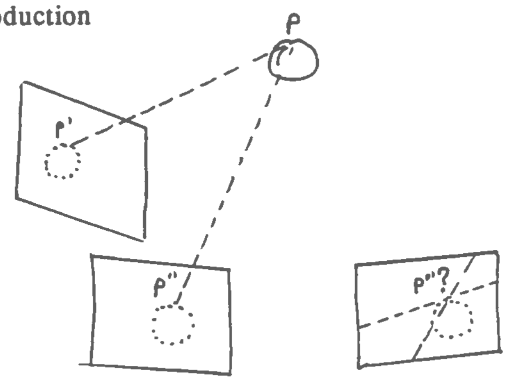


Figure 1.3 - Corresponding points in three images

third camera then, promises to turn a non-deterministic process into a deterministic one. There are other problems which exist in a binocular system which are surmountable using the three camera system.

In a two camera system where the length of the baseline between the two cameras is significant, it is possible for the edge of an object to be visible in one image whilst at the same time being hidden from view in the other. This problem of occlusion means that no match and consequently no 3D data can be obtained for such segments. The three camera system offers a solution to this problem which means that it is possible for a match to be found even when the corresponding segment is missing from one of the images. (see 8.4 for details)

Another problem arises when segments in either of the images of a binocular system are aligned with the baseline between the cameras which makes matching these segments and any subsequent calculations using them very inaccurate indeed. However in the trinocular set-up there will always be at least two cameras whose baseline is not aligned with and particular segment and matching can continue unhindered. (see 4.3.4 for details)

1.1. DESCRIPTION OF CONTENTS

This section describes the main contents of each chapter of the report.

A survey of literature describing similar work that has been carried out is presented in chapter 2. The relation between it and the work described in this report is also presented.

In chapter 3 a detailed description of the monocular image processing operation is presented. The operation is broken down into its constituent parts and each one is described in terms of the theory behind it and the way that the theory has been implemented in this system. The chapter concludes with an example run of the monocular image processing operation showing intermediate results from each part.

Chapter 4 describes how the third camera is used in the way suggested by Ayache and Lustman. Again issues involved with implementing the matching algorithm are discussed before the implementation of each operation is presented. Results from a sample run of the matcher are given at the end of the chapter and illustrate an example of the matcher trying to process a single segment in one of the images.

The model matcher used in the final stages of recognition and location of the object is described in chapter 5. A description of the theory behind it precedes a brief section on how to use the model matcher to carry out the operations we need to use it for.

The process of camera calibration is introduced in chapter 6 where an account of the motivation behind the mathematics is given before a guide on how to use the suite of camera calibration programs is presented.

Chapter 7 presents the conclusions that have been drawn during the development of the system and from the results produced.

The final chapter highlights several ways to extend the system by either improving its performance or its capabilities.

There follows a bibliography of all the material referred to in the report and a few appendices describing some of the theoretical derivations used as well as a brief user's guide.

1.2. EXAMPLE OF SYSTEM RUNNING

The pages at the end of the chapter show an example of the system in operation. The first three figures 1.4-1.6 show the images obtained from each camera. The first is from a camera which was directly above the third, which itself was directly to the left of the second camera.

These three images are passed through the monocular image processing operation and result in figure 1.7 where the located edges in each image are represented by line segments. These three images are passed to the trinocular matcher which produces the output of figure 1.8. The first column describes the match that has been found in terms of the segment number in each image, the letter and the number after it are debugging information and should be ignored, the next two columns are the 3D co-ordinates of the end points of the segment, each one has an error value associated with it. The results show that after disambiguation the matcher has been able to make correspondences between ten edges and the co-ordinates of their end-points are shown.

Figure 1.9 shows these ten edges highlighted in purple on top of the segmented images of figure 1.7.

1.3. PROGRAMS USED

The following section describes the origin of much of the software which is used in this project.

Some of the code corresponding to the image processing phase of the project has been adapted from the *Human Information Processing Software (HIPS)* library of image processing programs. These were developed in the Department of Psychology at New York University by Yoav Cohen and modified to operate in the Edinburgh A.I. department by David Croft. Other parts of the code have been developed in the department and implemented using the same calling conventions. The specific programs used are; *acanny.c*, *track.c* and *corner.c*. Although the code existed already, extensive modifications, including debugging and modularisation, had to be carried out in order to use the programs in the project. *Capture.c* and *merge.c*, were both written from scratch.

The work carried out by Mark Orr for his M.Sc. (1984) on camera calibration and the subsequent suite of programs developed to aid in the calibration process were relied on heavily and parts of the suite have been adapted and are used extensively in the stereo matching phase of the system.

The recogniser/locator written as part of Watson's M.Sc. thesis (1985) was used in the latter stages of the project to enable the geometric matching and location of the whole object in world. This program was not altered.

A program developed by Dr. J. Hallam to display graphics on the SUN monitor using windows was butchered to ease the display of segmented images and results on the VDU.

Various utilities developed by postgraduate students in the department Nang Chan and Li Dong Cai were used for the production of hard copy of results obtained in order to put them in this report.

1.4. EQUIPMENT USED

The equipment required for this project is split between the M.Sc. vision lab on the first floor and the Ph.D. lab on the ground floor of the A.I. building at Forrest Hill.

In the M.Sc. lab there is;

- A specially built stereo frame to hold the camera mounts at known positions.
- Three specially manufactured camera mounts to control the location, tilt, roll and pan of the cameras.
- Three video cameras - a Hitachi vk-m98e, a Panasonic WV-1500/b and a Minitron CCV.
- A CRS framestore and display module linked to a colour video monitor.
- A SUN 3/140 running under UNIX 4.2.

In the Ph.D lab there is;

- A MaxVideo^{TM} Digimax digitizer and display module linked to a monochrome Hitachi video monitor.
- A SUN 3/260 with floating point accelerator and colour display running under UNIX 4.2

Figure 1.4 - Original image from camera 1

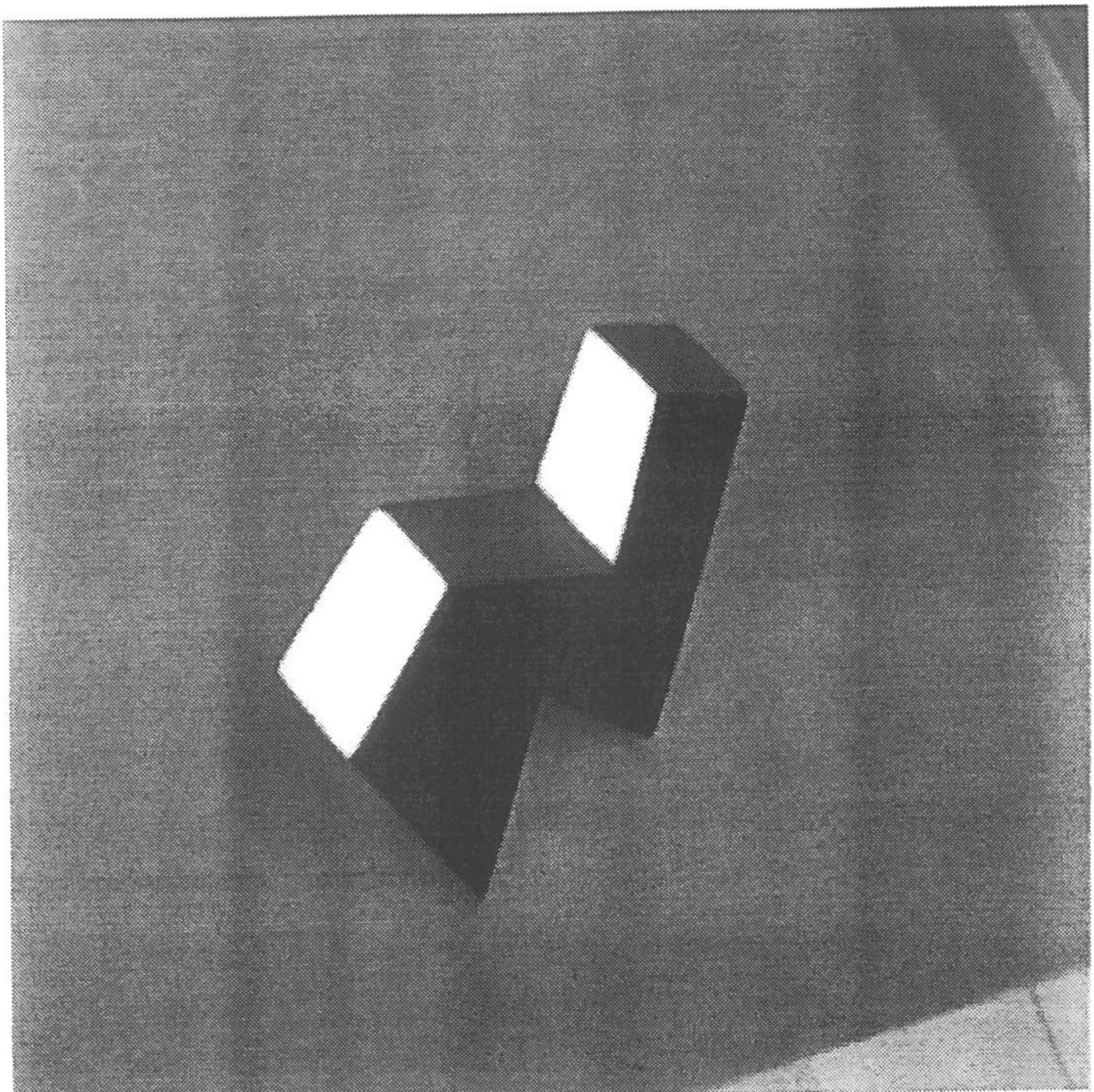


Figure 1.5 - Original image from camera 2

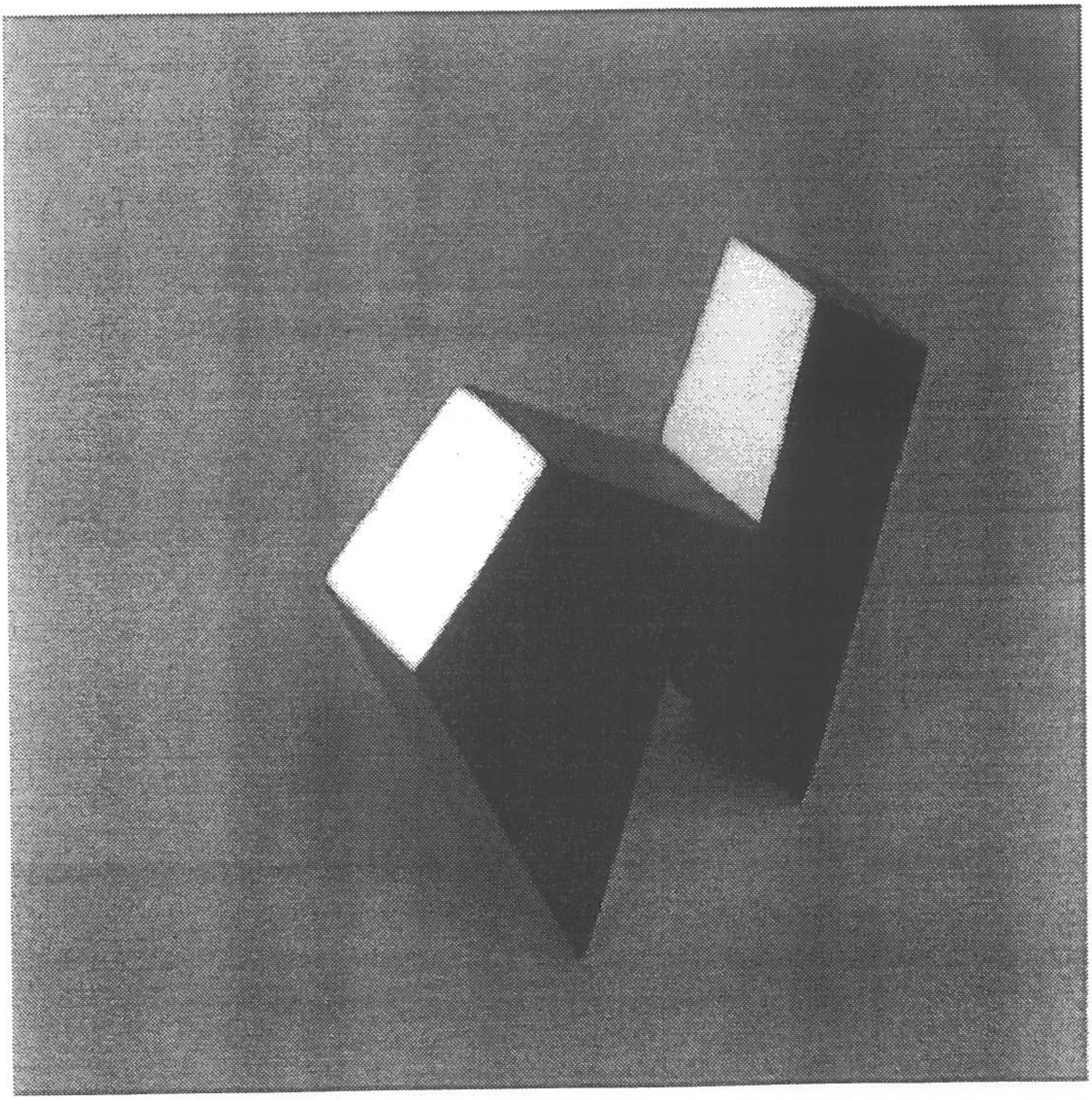
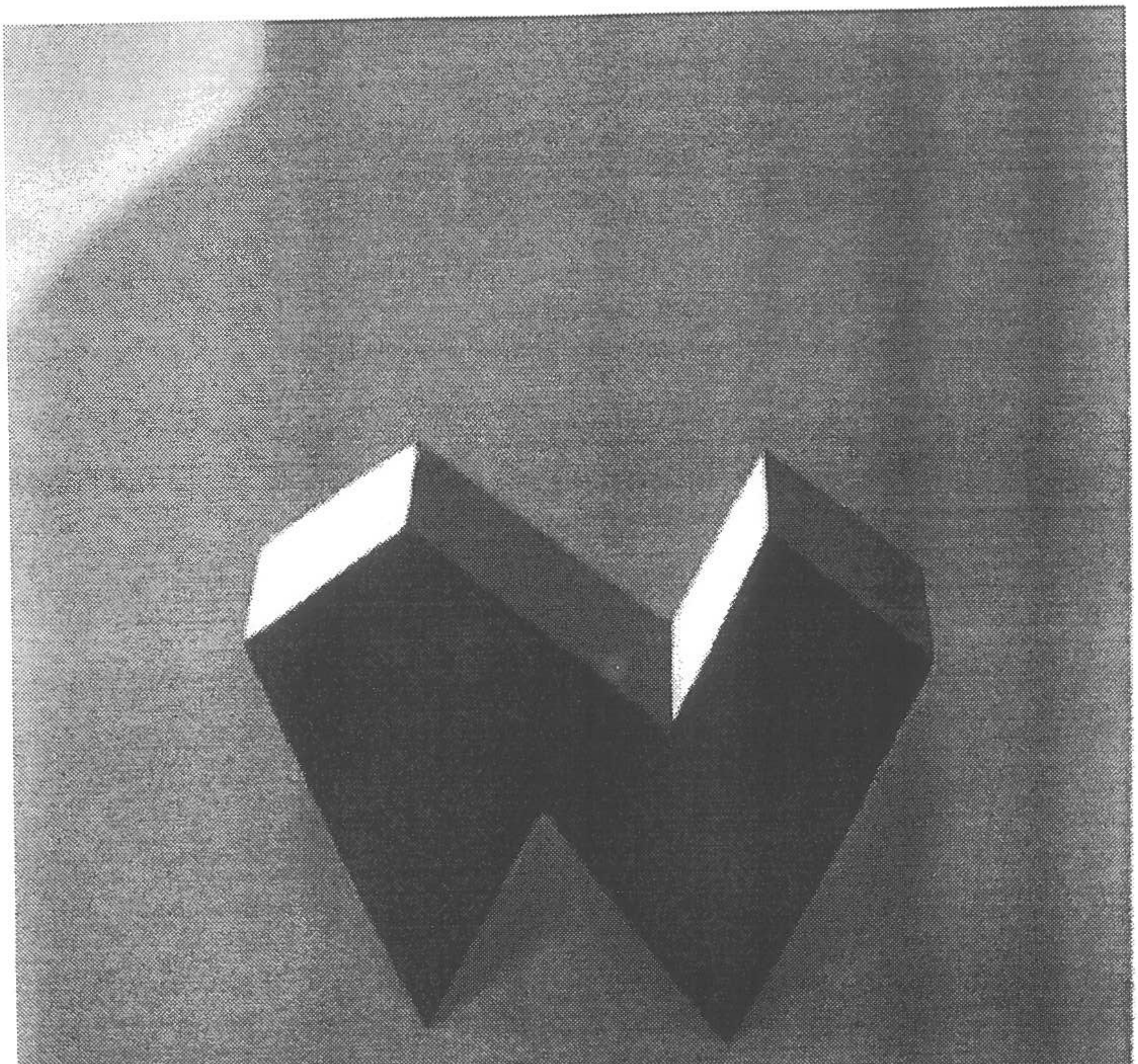


Figure 1.6 - Original image from camera 3



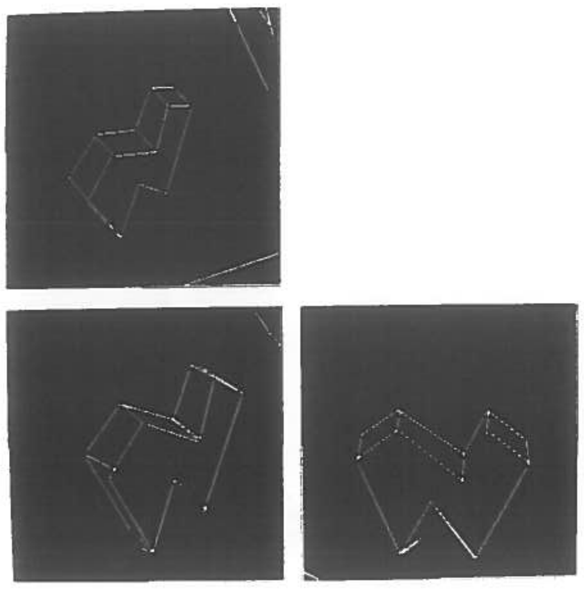


Figure 1.7 - The three segmented images

```
PRODUCE MATCHES:
Which segment are you interested in : -1
SEGMENT : 0
SEGMENT : 1
SEGMENT : 2
   "     "
   "     "
SEGMENT : 24
SEGMENT : 25
There are 41 potential matches.

PRODUCE OVERLAPS:
.........................................

There are 16 matches to disambiguate:-
        10:14:12 (a)/9.14       (-23.73,5.24,48.49)-0.0482      (-25.99,-0.02,47.00)-0.0629
        12:16:21 (r)/4.95       (-28.91,3.08,45.53)-0.0515      (-30.33,0.04,44.66)-0.0952
        14:20:19 (a)/17.92      (-33.57,4.72,43.74)-0.1821      (-31.99,1.99,44.22)-0.1195
        14:20:20 (a)/12.26      (-33.37,4.97,43.47)-0.0196      (-30.94,0.69,44.23)-0.0931
        15:9:9 (r)/9.58         (-27.07,6.51,49.37)-0.0418      (-28.06,4.59,48.82)-0.0697
        16:10:8 (a)/4.58        (-28.04,4.48,48.37)-0.0614      (-27.22,4.44,46.74)-0.0646
        16:10:10 (a)/2.02       (-28.05,4.46,48.39)-0.0663      (-27.26,4.35,46.86)-0.0693
        18:9:6 (a)/3.10         (-26.42,7.18,47.43)-0.2794      (-26.44,7.38,45.37)-0.4754
        20:4:13 (t)/18.88       (-28.92,4.98,48.13)-0.0409      (-31.84,6.87,46.37)-0.1015
        21:5:13 (a)/8.16        (-31.63,6.78,46.03)-0.0348      (-30.91,7.19,44.58)-0.2502
        21:5:14 (a)/5.66        (-31.72,6.66,46.20)-0.0808      (-31.28,6.68,45.23)-0.1062
        22:6:17 (a)/10.04       (-31.43,6.35,45.09)-0.0258      (-33.32,4.94,43.95)-0.0454
        23:7:15 (a)/1.50        (-32.37,6.33,46.91)-0.0305      (-34.10,5.01,46.31)-0.0422
        24:18:8 (t)/16.58       (-27.52,4.78,46.22)-0.0348      (-30.09,7.56,43.62)-0.1602
        24:18:10 (t)/4.32       (-27.49,4.56,46.47)-0.0254      (-30.08,7.47,43.71)-0.1367
        24:18:18 (t)/2.49       (-27.46,4.26,46.77)-0.0333      (-30.90,6.43,45.14)-0.0278

These are the segments after disambiguation :-
        10:14:12 (a)/9.14       (-23.73,5.24,48.49)-0.0482      (-25.99,-0.02,47.00)-0.0629
        12:16:21 (r)/4.95       (-28.91,3.08,45.53)-0.0515      (-30.33,0.04,44.66)-0.0952
        14:20:20 (a)/12.26      (-33.37,4.97,43.47)-0.0196      (-30.94,0.69,44.23)-0.0931
        15:9:9 (r)/9.58         (-27.07,6.51,49.37)-0.0418      (-28.06,4.59,48.82)-0.0697
        16:10:10 (a)/2.02       (-28.05,4.46,48.39)-0.0663      (-27.26,4.35,46.86)-0.0693
        20:4:13 (t)/18.88       (-28.92,4.98,48.13)-0.0409      (-31.84,6.87,46.37)-0.1015
        21:5:14 (a)/5.66        (-31.72,6.66,46.20)-0.0808      (-31.28,6.68,45.23)-0.1062
        22:6:17 (a)/10.04       (-31.43,6.35,45.09)-0.0258      (-33.32,4.94,43.95)-0.0454
        23:7:15 (a)/1.50        (-32.37,6.33,46.91)-0.0305      (-34.10,5.01,46.31)-0.0422
        24:18:18 (t)/2.49       (-27.46,4.26,46.77)-0.0333      (-30.90,6.43,45.14)-0.0278
neon.aicam      >
```

Figure 1.8 - Sample output

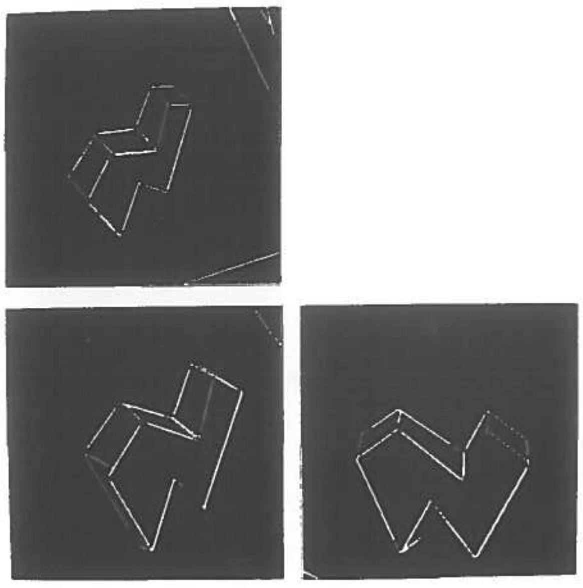


Figure 1.9 - Results highlighted on original images

CHAPTER 2

RELATED WORK

The field of stereopsis has been studied comprehensively as a means to enable computers to see. However, the vast majority of this work has, centered on the technique of binocular stereo. Binocular stereo only has a bearing on this project because one of the central problems encountered when using this technique motivates the work carried out in the project. Because of this, we are not directly concerned with documenting any similarities between work carried out using this technique and the work described herein. Papers by Baker and Binford (1981) and Bolles et. al. (1983) may be consulted to obtain an impression of how a binocular system may operate. In this chapter the aim is to relate the work carried out in this project to the work carried out by other workers in the field of *trinocular* stereopsis. Any major differences between them will be discussed and any similarities will be pointed out.

The idea of trinocular stereo grew out of work in the binocular domain. Yachida (1986) introduced the principle of trinocular stereo and went on to describe a procedure to determine the 3D position of each point by trinocular stereo. Since then, several studies have been concerned with the use of the third camera, Ito & Ishii (1986), Pietikainen & Harwood (1986), Gerhard et. al. (1986), Gurewitz et. al. (1986) and Ohta et. al. (1986). However, few investigations have been conducted and this list is more or less a survey of the literature. Of the papers that could be obtained, Ohta et. al. is the only work of any fundamental difference to the work described in this report.

Ohta, Watanabe and Ikeda present a considerably different approach to trinocular stereo than any of the other workers and this merits a closer look. The three cameras are set-up so that the base-lines between them forms a right-angled triangle. The correspondence search is performed on the two image pairs on either side of the right angle and produces two depth maps in the camera at the right angle which are perpendicular to each other. These maps which are produced using the technique of dynamic programming should be identical but may have been altered by erroneous correspondences, ambiguous edges or occlusion. The trinocular part of the system is introduced by combining the two maps to produce a better depth map. This is achieved using a relaxation technique based on a minimisation function.

The scheme they offer has the intervals between edges on epipolar lines being used as the units of matching which is similar to the approach taken here. The scheme differs in the fact that the cameras are restricted in their relative orientations. This is a major drawback when compared with the flexibility offered by the system in this report. No explicit triangulation is required, however, as a measure of the depth comes as a side effect of the matching operation.

Another scheme that restricts the orientation of the cameras is that of Pietikainen and Harwood. They restrict the placement of the cameras to be in a plane as well as making the assumption that the camera's were linear. This greatly simplifies the matching procedure but the disadvantage shows up in the empirical results that they present. For synthesised images, those not coming from real cameras, the results are good, but the results from an object in the real world are not as good with errors measured in centimetres. This is a results of the non-linearities associated with real cameras. However, their procedure makes good use of constraints derived from the local properties of the edge tracks which they use as primitive features. The orientation and intensity properties are used during the proposal of matches and edge connectivity is used to eliminate false matches in a post-processing phase. Their's is the only scheme that uses an element of automatic thresholding to set the thresholds on orientation and intensity. A histogram technique as proposed in chapter 8 is utilised.

The other works do not differ significantly from the work conducted here apart from in minor aspects such as the local properties used by Ayache & Lustman and the status of the epipolars in Yachida. The major point that is emphasised in all the studies including Ohta's and Pietikainen's is the advantage that trinocular stereo has over its binocular alternative in areas such as occlusion, dealing with lines of arbitrary orientation and verification of feature correspondences. The reasons for these advantages are made apparent throughout the rest of the report.

CHAPTER 3

MONOCULAR IMAGE PROCESSING

As described in the introduction before any trinocular processing can take place between the images, each image must be processed to obtain a description of the edges in the scene. This processing consists of a sequence of operations which gradually reduce the video image to a set of line segments which describe the edges of the objects in the original image. (Table 3.1)

Operation	Description
capture	Obtains the image from the camera
canny	Carries out the edge detection process on the raw data
track	Collects edgelets together into continuous tracks
corner	Splits these tracks at their corners to form line segments
merge	Takes the raw segments and merges any which constitute the same edge into a single line segment

Table 3.1 - Image processing operations

Because of the need to apply this sequence of operations to each input image it was decided to parameterise them and implement them as subroutines using intermediate data structures to pass the data from one operation to the next. (Figure 3.1)

This means that for example to carry out the sequence of operations for a single image the following set of calls would be required;

```
capture(FILE_NAME,Raw);
acanny(Raw,Can);
track(Can,Trk,&trkctr,Pts);
for (i= 0; i< trkctr; i+ + )
    corner(Pts,Trk[i].first,Trk[i].last,Rsg,&segno);
merge(Rsg,Fsg,&segno);
```

capture.c

↓

raw - 2D array of grey levels from the digitiser

↓

acanny.c

↓

can - 2D array of edge strength values derived from raw.

↓

track.c

↓

trk,pts - An array (trk) indexing into an array
of tracked edgelets (pts)

↓

corner.c

↓

rsg - An array of segment descriptors describing
the edges in the scene

↓

merge.c

↓

fsg - An array of parameterised line segment descriptors which
are the result of combining smaller segments from rsg.

Figure 3.1 - Intermediate data structures

This itself can be parameterised and made into a subroutine which when called will carry out all the image processing on a given input image. Using this approach, all the image processing required for the left (L), right (R) and top (T) image can be achieved by the following calls;

```
lowlevel(lfile,Lraw,Lcan,Lpts,Ltrk,Lrsg,Lfsg,&LNumsegs);
lowlevel(rfile,Rraw,Rcan,Rpts,Rtrk,Rrsg,Rfsg,&RNumsegs);
lowlevel(tfile,Traw,Tcan,Tpts,Ttrk,Trsg,Tfsg,&TNumsegs);
```

This results in the 3 sets of line segments, Lfsg,Rfsg and Tfsg which can then be passed on to the correspondence matcher part of the system.

In the sections below, the theory behind each operation is presented before going on to describe the implementation of the operation as a subroutine including the algorithms and data structures used.

3.1. IMAGE CAPTURE

As the project stands, image data is obtained from files which have been captured some time before and stored there. All the *capture* subroutine does is to load the data from the specified file into the array.

It is envisaged that in an operational system this image capture would be a two-way process between the program and the digitiser which enabled the program to grab consistent image data from the correct frame store within the digitiser and place it in the array before returning to begin processing it.

In an operational system it would be desired that the image capture for all the cameras should be carried out in as short a period of time as possible. If the system processed one image before capturing the next (as it does at present), it would run the risk that the scene would have changed in the elapsed time. This could happen in a conveyor belt system or an environment with moving objects such as a robot. The solution would be to capture all the images and then proceed to process them.

3.2. CANNY'S EDGE DETECTOR

The first of the processes to be carried out on each image is that of edge detection. The image is represented by an array of grey levels and the object of the edge detection is to filter the image so that the areas of the image with a change in grey level (i.e. an edgelet) become most prominent. There have been many operators proposed to carry out this process, but of these an operator developed by Canny performs particularly well. [Canny '83]

3.2.1. Theory

The operator is derived by setting down three performance criteria and finding the optimum mask to convolve with the image data to maximise these criteria. [Beattie'85, pp59-62] The criteria are;

- There should be a low probability of marking non-edges and of failing to mark edges.
- The edge points should be placed as accurately as possible.
- There should only be one response to a single edge.

When these three criteria are mathematically formulated and optimised the solution consists of a combination of four complex exponentials which can be approximated by the derivative of a Gaussian curve. The size of the curve controls both the error rate and the accuracy of the localisation. The wider the curve the better the signal to noise ratio but the worse the localisation. A good compromise is a width of about one standard deviation.

Canny based his mathematical formulations of the criteria on the assumption that all edges could be modelled by the step function. However, an edge can occur in many

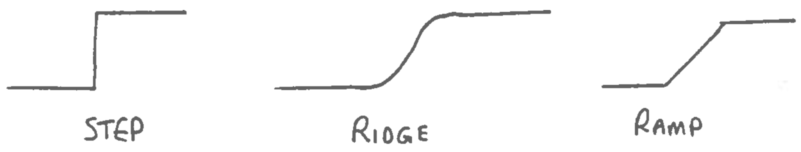


Figure 3.2 - Various types of edge

forms such as a ridge or a ramp. (Figure 3.2) This fact effects the performance of the operator. The higher the proportion of the edges in the image are step edges, the better the operator will perform.

The third criteria is not covered by the effects of the operator but is achieved by a combination of smoothing the image data prior to edge detection and carrying out an operation called non-maxima suppression.

So, the whole edge detection operation can be split into 4 major phases, each of which can be treated as a filter.

Smoothing

There are many ways of smoothing data to reduce the amount of noise present but an effective, and in the case of Canny edge detection, appropriate method is that of Gaussian smoothing. This is simply achieved by convolving a mask formed from the values of a Gaussian curve with the image data. (Figure 3.3) If the mask is centered on (0,0) then the mask can be calculated using the following formula;

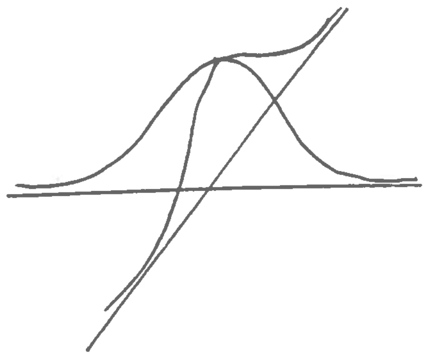


Figure 3.3 - Gaussian mask, used for smoothing

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.1}$$

where σ is the width of the mask in standard deviations. x and y both run from $-w$ to w where w is defined as the width of the mask in pixels and is the largest positive integer for which $G(0,y)$ or $G(x,0)$ is significant e.g. $G(0,y) > 0.005$.

For example when $\sigma = 1$;

$$e^{-\frac{w^2}{2\sigma^2}} > 0.005$$

take the $\log_e$ of both sides,

$$-\frac{w^2}{2\sigma^2} > -5.3$$

which means that,

$$\frac{w^2}{2} > 5.3, \text{ and } w^2 > 10.6,$$

therefore $w=4$.

The convolution proceeds as follows;

$$S(x,y) = \sum_{i=-w}^{w} \sum_{j=-w}^{w} I(x+i,y+j) \times G(i,j) \tag{3.2}$$

with x running across the image and y running down the image.

Gradient calculation

The gradients present across pixels are calculated using the following formulae;

$$\delta x(x,y) = S(x+1,y) - S(x,y) \tag{3.3}$$

$$\delta y(x,y) = S(x,y+1) - S(x,y) \tag{3.4}$$

$$GM(x,y) = |\delta x(x,y)| + |\delta y(x,y)| \tag{3.5}$$

Non-maxima suppression

Using the measures of the gradient along each axis (Equations 3.3 & 3.4) and the total gradient magnitude involved (Equation 3.5) the peak of the gradient can be localised with respect to the change in the image data by interpolating the expected gradient magnitude at sub-pixel positions. This is done by using the gradient magnitudes of surrounding points and the direction of the gradient at the current pixel.

In the example of figure 3.4 there is an edge running across the grid from top left to bottom right. [Fisher '88] $G=GM(x,y)$, $G_1=GM(x+1,y+1)$, $G_2=GM(x+1,y)$, $G_3=GM(x-1,y)$, $G_4=GM(x-1,y-1)$. It can be seen from the diagram that a measure of the direction of the edge, A can be calculated thus;

$$\frac{1}{A}=\frac{|\delta x(x,y)|}{|\delta y(x,y)|}$$

We can interpolate G_α and G_β using A;

$$G_\alpha = A \times G_1 + (1-A) \times G_2$$

$$G_\beta = A \times G_4 + (1-A) \times G_3$$

If $G_\alpha > G$ or $G_\beta > G$ then G is not the local peak across the pixel and should not be marked as an edge point.

There are three other cases corresponding to the three other quadrants that the gradient can be pointing towards. In each case a different set of points $G_1 \cdots G_4$ is used for the calculations.

Hysteresis tracking

Edges are prone to break up if edge strength varies both above and below a certain threshold along its length. This phenomena has been termed the *streaking problem* and is a good example of just one of the many problems

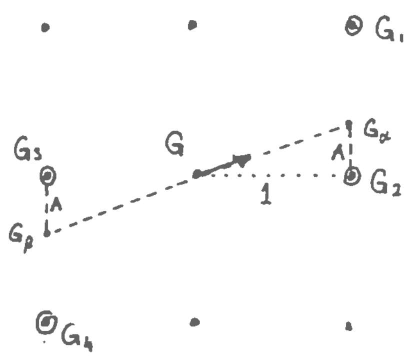


Figure 3.4 - Non-maxima suppression for a single pixel

encountered when using vision techniques in the real world. Canny developed the hysteresis tracker to minimise this problem by using both a high and low threshold. An edge point with an intensity above the high threshold is output along with the entire contour containing it, if no part of the contour is below the low threshold.[1] With this scheme, an edge will only be started if it is above the high threshold and will only be broken if it drops below the low threshold.

3.2.2. Implementation

The program *acanny.c* which implements Canny's operator is constructed in four sections which correspond exactly to the phases described above. It takes the grey level image data from the image capture operation and returns a similar array with values corresponding to edge strengths where there are edges and zeroes everywhere else.

There is an initial section which calculates and stores both the Gaussian mask and the derivative of the Gaussian in arrays. These look-up tables save a considerable amount of calculation when it comes to the smoothing and gradient calculation phases.

The smoothing and gradient calculation operation can be described with reference to figure 3.5. I is the original grey level image. S_1 and S_2 are the two sums of equation 3.2 that combine to give the smoothed image. H_x and H_y correspond to δ_x and δ_y of equations 3.3 and 3.4 respectively and GM is the magnitude of the gradient as described in equation 3.5.

This scheme is necessary to increase the efficiency of the edge detection by utilising both the associativity of the convolution operator and the seperability of the Gaussian mask.

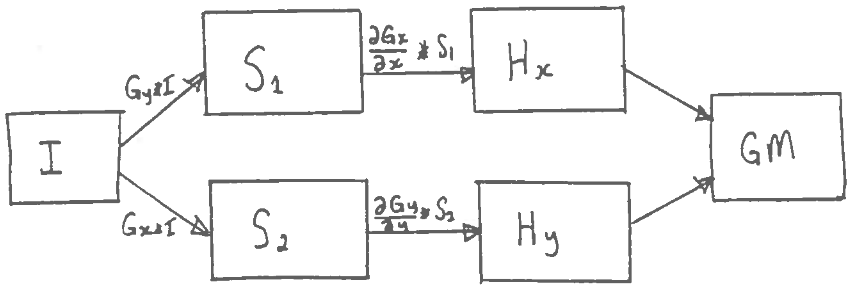

$G_y * I$
$G_x * I$
S_1
S_2
$\frac{\partial G_x}{\partial x} * S_1$
$\frac{\partial G_y}{\partial y} * S_2$
I
H_x
H_y
GM

Figure 3.5 - Arrays used to implement the smoothing and gradient operations

[1]This is conditional on all the edge points having similar orientations.

Ultimately we wish to calculate $\nabla(G*I)$ which can be split into its two components;

$$H_x = \frac{\partial}{\partial x}(G*I)$$

and

$$H_y = \frac{\partial}{\partial y}(G*I)$$

If we separate G into its two components G_x and G_y which satisfy $G = G_x * G_y$ i.e.

$$G_x(x,y) = e^{-\frac{x^2}{2\sigma^2}}$$

$$G_y(x,y) = e^{-\frac{y^2}{2\sigma^2}}$$

then $(G*I)$ changes to $(G_x * G_y * I)$ and the complexity of the convolution changes from $O(w^2N^2)$ to $O(2wN^2)$ where w is the width of the mask and N is the linear size of the image.

So, we now have;

$$H_x = \frac{\partial}{\partial x}(G_x * G_y * I)$$

and

$$H_y = \frac{\partial}{\partial y}(G_x * G_y * I)$$

Using the associativity of convolutions we have;

$$H_x = \frac{\partial G_x}{\partial x} * (G_y * I)$$

and

$$H_y = \frac{\partial G_y}{\partial y} * (G_x * I)$$

which corresponds to the operations of figure 3.5.

In the formulae, $\frac{\delta G(i)}{\delta i}$ is the derivative of the Gaussian curve of equation 3.1. (Figure 3.6) It provides a weighting to multiply each gradient calculation in the sum by.

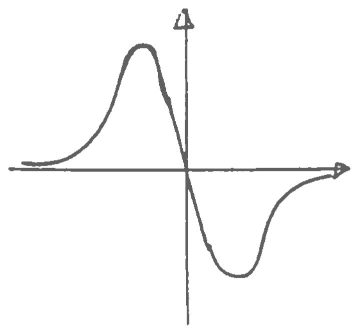


Figure 3.6 - The derivative of a Gaussian curve

The weighting reduces the further away from (x,y) the calculation is taking place. It also preserves the directionality of the gradient across the pixel. I.e. if the data is decreasing from left to right across the pixel, the value will be positive and vice versa.

The convolution of the Gaussian mask is simply achieved using a doubly nested loop which indexes through the rows and columns. At each iteration the mask is applied to the pixels aligned with the current pixel and the results are stored in two temporary arrays. One holds the results of the convolution in the x-direction the other the results of the convolution in the y-direction. This means that each summation in equation 3.2 is carried out separately.

The gradient calculation is carried out by using a doubly nested loop to convolve the appropriate component of the derivative of the Gaussian with the results of the smoothing operation just described.

Again a doubly nested loop is used to carry out the non-maxima suppression for each pixel[2]. The four cases corresponding to the four quadrants are distinguished using an *if-else-if...* construct and if the pixel concerned is not to be suppressed its edge strength and orientation are stored in separate arrays, otherwise the corresponding element in each array is reset.

The hysteresis tracking is carried out in a simple minded fashion. The rows and columns are indexed in the usual way and if a pixel has an edge strength value which is greater than a *HIGH* threshold a subroutine *follow* is called. If the pixel has not been tracked before the subroutine stores its edge strength value in the output array before going on to examine the neighbouring 8 pixels. For each one if its value is greater than

[2]These doubly nested loops can not be combined into one because the results from one phase are required for the next phase. This prevents the phases from being carried out in parallel.

a *LOW* threshold and it has a similar orientation to the central pixel then *follow* is called recursively one it.

3.3. BEATTIE'S EDGE TRACKER

Now that the edgelets have been localised it is desirable to reduce the representation of the image to just these edgelets. This focuses the attention of further processing to the only relevant points in the image, the points on an edge. This problem was analysed by Beattie whose resultant edge tracking algorithm is used in this project to collect together all the edgelets. [Beattie'85, pp119-120] Note that the hysteresis tracker described in the last section could have been altered to carry out this task, but as Beattie's edge tracker had already been implemented within the department it was decided to utilise the existing routine.

3.3.1. Theory

Beattie's tracking algorithm can be summarised thus;

(1) Scan the image data in raster fashion until an edge point is discovered.

(2) If the edge point has just one neighbouring pixel or more than 2 neighbouring pixels then place it and its neighbouring pixel(s) onto the stack.

(3) The pixel at the top of the stack is now taken and used as a starting point to track an edge. As tracking proceeds, used points are marked to ensure that an edge point is only tracked once and that no infinite loops are entered. Tracking is guided by the eight neighbouring pixels using the following rules;

- Case 1 When an edge pixel has only one neighbour which is also an edge pixel it has to be the one just tracked so it has come to the end of an edge. The current track therefore ends, and a new starting point is taken off the stack.
- Case 2 In the case of the current edge pixel having two neighbours, the track continues as it must be in the middle of an edge.
- Case 3 Any more than two neighbours indicates the presence of a junction, so the current track ends and the neighbours are placed on the stack.

(4) Tracking continues using new starting pixels from the top of the stack. (Goto 3)

When the stack is exhausted the raster scan continues through the image to find a new starting edge pixel. (Goto 1)

When the raster scan finally finishes all the edges which have been found are represented by lists of co-ordinates.

(5) However, any simple closed loops will not have been found because they consist entirely of pixels with exactly two neighbours. So, an extra pass is made over the image data looking for unused edge points with exactly two neighbours. When one is found, tracking proceeds from one of the neighbours and terminates when the other is reached.

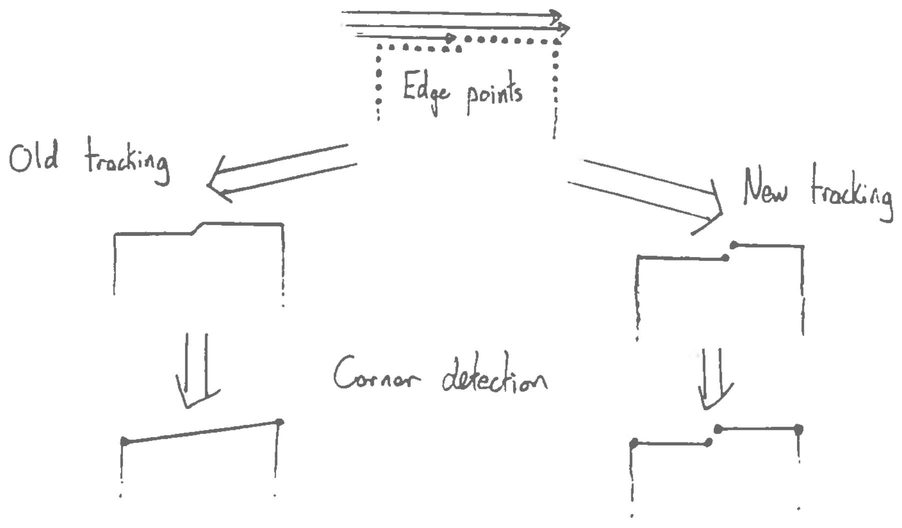


Figure 3.7 - Comparison of approaches to dealing with edge points with two neighbours

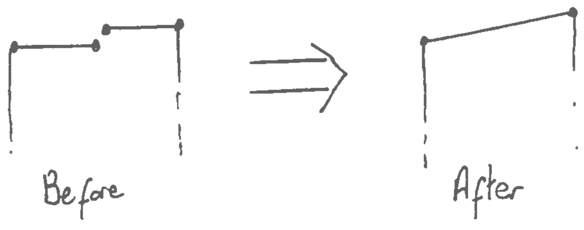


Figure 3.8 - Effects of merging on an inappropriately segmented edge

Step (5) is very wasteful as simple closed loops hardly ever occur. It was decided to change step (2) to include pixels with two neighbours and to treat them in the same way as those with more than 2 neighbours. The implication of this was that closed loops could be segmented into several different tracks. (Figure 3.7)

However, this is not a problem because the merging operator (§ 3.5), implemented to overcome some of the effects of noise, merges segments which are similarly oriented and close together and consequently will merge these segments back together. (Figure 3.8)

3.3.2. Implementation

The program *track.c* which implements the tracker is constructed from a few main subroutines corresponding to steps (1)-(4) above and several supporting subroutines which facilitate access to stacks and perform general housekeeping tasks. There is also one, *condense* which is called at the end of the process and passes over the tracked edge forming a record of the start and end indices of the tracks as well as their length. So the final structure to emerge from the program looks something like figure 3.9. The

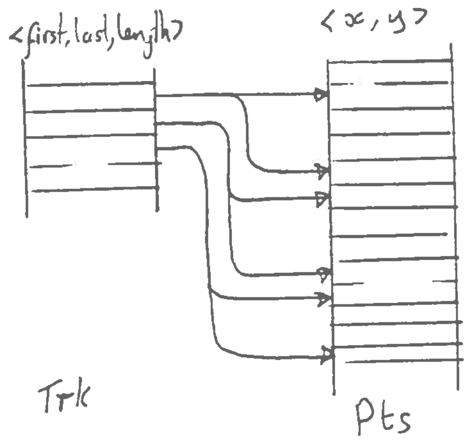


Figure 3.9 - The two arrays that represent tracked edges

first array, *trk*, holds the records created by *condense* of all the tracks in the image.

< first, last, length>

It does so by indexing into the second array of tracked points, *pts*, which can either be start points, end points or middle points as defined by their code.

< x:co-ord, y:co-ord, code>

By using these arrays access to any track can be attained without having to search through the *pts* array.

Steps (1) and (4) are implemented by the subroutine *tredges()*. It indexes through the whole image and for each new edge point encountered it gets all its neighbouring points. It then puts any that have not been tracked onto the stack and enters a loop to track them.

```
FOR each pixel in the image
        IF (new edge point)
        {
                find all its neighbours (findneigh)
                FOR each neighbour
                        IF new point PUSH it on the stack
                WHILE (stack not empty)
                        POP point off the stack
                        track the edge leading from it (edgetrack)
        }
```

Step (2) is implemented by the subroutine *findneigh(p)* which searches the pixels immediately around *p* (the pixel concerned) and returns any that are edge points.

Step (3) is implemented by the subroutine *edgetrack()* which tracks a designated edge through to completion adding any other potential tracks to the stack as it goes. It first of all starts a new edge and then goes into a loop to track the pixels.

It finds the points that neighbour the current point and depending on the number of points there are either terminates the track and returns, carries on along the track or adds some more potential edge points to the stack and then terminates the track and returns.

```
start an edge
WHILE (edge not terminated)
{
        find all its neighbours (findneigh)
        IF (one neighbour)
                terminate edge
        IF (two neighbours)
                carry on with edge if not a simple closed loop
        IF (more than two neighbours)
                stack any new edge points
                terminate the edge
}
```

3.4. MALCOLM'S CORNER LOCATOR

To produce the line segments that the matching algorithm requires, the tracked edges must be split at their corners.[3] The corner filter algorithm devised by Malcolm does just this using a threshold of "cornerness" to locate any corner points on a tracked edge. [Malcolm 1984]

3.4.1. Theory

The corner filter can be thought of as a digital worm which crawls round the tracked edge carrying out the calculations below as it goes, its head at the current pixel and the tail several pixels behind.[4] In the equations below, n is the index of the current point (the head of the worm), w is the size of the worm in pixels, and s is the amount of de-localisation that is maintained (usually the same as w).

Rectangular components of head/tail distance;

$$DX_n = X_n - X_{n-w} \tag{3.6a}$$

$$DY_n = Y_n - Y_{n-w} \tag{3.6b}$$

[3] This is a sufficient condition because we are only dealing with polyhedral objects.

[4] The worm should not be any longer than the shortest edge anticipated otherwise the edge will be lost.

Components of de-localised differentiation of rectangular distance. An approximation to the components of change in gradient;

$$SX_n = DX_n - DX_{n-s} \tag{3.7a}$$

$$SY_n = DY_n - DY_{n-s} \tag{3.7b}$$

This gives a measure of the total change in rectangular distance;

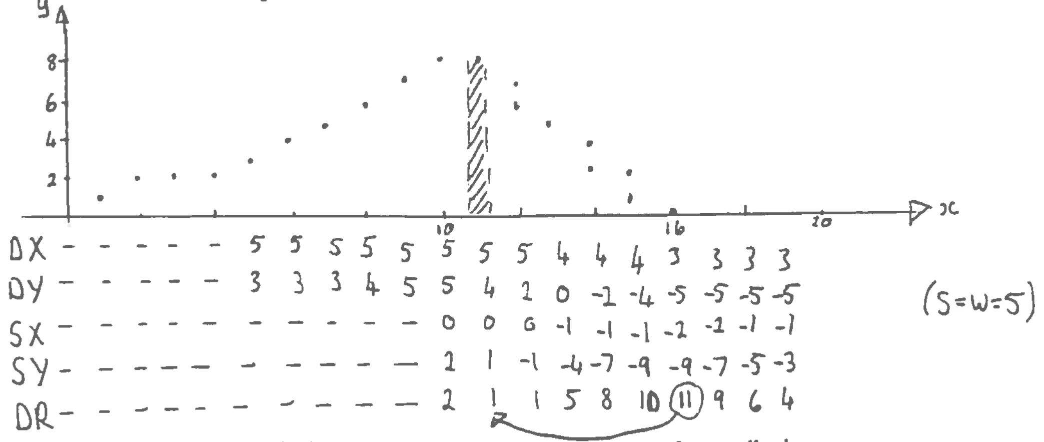
$$DR_n = abs(SX_n) + abs(SY_n) \tag{3.8}$$

As the worm moves around the edge the gradient of the line joining its head and tail will change (Equ. 3.6). De-localisation of the gradient calculations is carried out to remove any noise present (Equ. 3.7), but none the less, as the worm negotiates a corner a rapid change in this gradient will be detectable (Equ. 3.8). Figure 3.10 shows some example calculations for a small part of an edge with $w = s = 5$. Note that in the figure, the corner has to be localised after the local maximum has been found by moving it back the de-localisation distance.

Rectangular distance is used as a qualitative approximation to gradient. Its calculation is simple, not involving division, yet it tends to increase, decrease and be zero at the same time as the gradient measure. De-localisation is carried out to prevent noise affecting the calculations significantly by taking the differentiation of these gradient measures over a large distance.

3.4.2. Implementation

The program *corner.c* implements the corner locator and operates on one track at a time. It is passed the array containing all the tracked points and the start and finish indexes of the track to be processed. It adds any segments it finds to an array which



DX	-	-	-	-	-	5	5	5	5	5	5	5	5	4	4	4	3	3	3	3
DY	-	-	-	-	-	3	3	3	4	5	5	4	2	0	-2	-4	-5	-5	-5	-5
SX	-	-	-	-	-	-	-	-	-	-	0	0	0	-1	-1	-1	-2	-2	-1	-1
SY	-	-	-	-	-	-	-	-	-	-	2	1	-1	-4	-7	-9	-9	-7	-5	-3
DR	-	-	-	-	-	-	-	-	-	-	2	1	1	5	8	10	11	9	6	4

Figure 3.10 - Worked example showing worm calculations for small edge

holds all the segments found so far in the image. Each segment is described in the following way;

< startx, starty, endx, endy>

The program is based on one big loop which advances the worm round the tracked edge one pixel at a time. It calculates *DX* and *DY* and if it is far enough along the track, calculates *SX* , *SY* and *DR* as well.

After it has done the calculations it checks for a large discrepancy in the co-ordinates of the last two points. If there is one it drops a corner point there. However, in most cases there won't be one and a check is made to see if the current value of *DR* is above the threshold of cornerness. If it is and the value is at a maximum with respect to the surrounding values of *DR* then a corner point is dropped there. Note that this means corners are only dropped once for each bend in the track (at maximum bend) rather than all the way round the bend.

Once the whole track has been negotiated a loop is entered which forms the start and end co-ordinates of the segments that have been found before adding them to those that have been found on previous tracks.

3.5. SEGMENT MERGER

The corner locator produces a set of straight line segments described by the image co-ordinates of their end-points. However this set of segments can be condensed if any segments that lie on the same physical edge in the image can be merged into a single segment. The edge may have been broken up for several reasons: noise during image capture, variation of light intensity along the length of an edge, lack of contrast over an edge, starting tracking in the middle of an edge etc. Figure 3.11 shows an example of a

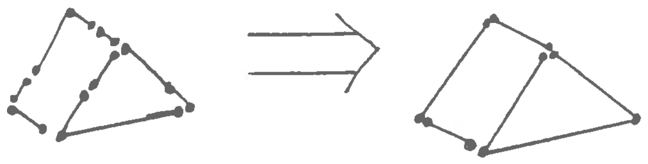


Figure 3.11 - Required action of the segment merger

typical input and desired output of the process.

3.5.1. Theory

The problem

Consider an image which has m edges, $E_1 \cdots E_m$ that are broken up into a set of segments $S = \{\alpha_{11}, \ldots, \alpha_{mk}\}$. The set of segments that make up each edge can be defined;

$$S_1 = \{\alpha_{11}, \alpha_{12}, \ldots, \alpha_{1k}\}$$

$$S_2 = \{\alpha_{21}, \alpha_{22}, \ldots, \alpha_{2k}\}$$

$$\vdots$$

$$S_m = \{\alpha_{m1}, \alpha_{m2}, \ldots, \alpha_{mk}\}$$

and the order that the segments must be in to constitute the edge can be defined by the terms;

$$E_1 = \alpha_{11}\alpha_{12}\ldots\alpha_{1k}$$

$$E_2 = \alpha_{21}\alpha_{22}\ldots\alpha_{2k}$$

$$\vdots$$

$$E_m = \alpha_{m1}\alpha_{m2}\ldots\alpha_{mk}$$

The task of the segment merger is to reduce S to $S^m = \{E_1, E_2, \ldots, E_m\}$ a set of size m by reducing the various S_i to their corresponding E_i. This means that a way of deciding whether a segment is another one's neighbour has to be used. It is the case that if two segments' orientations are within a certain threshold and their end points are within a certain distance they should be considered neighbours.[5]

Definitions

A **pass** can be defined as an operation whereby an **active segment** is compared against every other member of S to determine whether it is one of the active segment's neighbours as defined by its E_i. If this is the case then the segment found is removed from S and appended to the active segment to form a new segment and the pass

[5] Note that to determine whether two segments $a{:}b$ and $c{:}d$ are within the certain distance, all four combinations of end-to-end pairings must be tested. I.e. $\{(a,c)\ (a,d)\ (b,c)\ (b,d)\}$.

continues. A pass is complete when every element of S has been compared against the active segment.

An **intermediate state** of a set of segments S_i can be notated S_i^n. This is the state when n of S_i (the original set's segments) have been the active segment and had a pass carried out for them.

Theorem

A set of segments S_i will be reduced to its respective E_i in no more than *ik* passes.

Proof

We have $S_i^n = \{\alpha_{i1}^n, \alpha_{i2}^n, \ . \ . \ . \ , \alpha_{ik}^n\}$ and $E_i = \alpha_{i1}\alpha_{i2}\ldots\alpha_{i(n-1)}\alpha_{in}\alpha_{i(n+1)}\ldots\alpha_{ik}$. We can see that if we take an active segment $\alpha \ \varepsilon \ S_i^n$ and carry out a pass using it;

if $\alpha = \alpha_{i1}\ldots\alpha_{ix}$	then α gets replaced by $\alpha_{i1}\ldots\alpha_{i(x+1)}$ and the size of S_i^{n+1} is one less than the size of S_i^n. (For $1 \leq x < ik$)
if $\alpha = \alpha_{ix}\ldots\alpha_{ik}$	then α gets replaced by $\alpha_{i(x-1)}\ldots\alpha_{ik}$ and the size of S_i^{n+1} is one less than the size of S_i^n. (For $1 < x \leq ik$)
otherwise	$\alpha = \alpha_{ix}\ldots\alpha_{iy}\ldots\alpha_{iz}$ and α gets replaced by $\alpha_{i(x-1)}\ldots\alpha_{iy}\ldots\alpha_{i(z+1)}$ and the size of S_i^n is two less than the size of S_i^n. (For $1 < y < ik$)

So, no matter the order we select the *ik* segments of S_i, they will always reduce to E_i in no more than *ik* passes.

We now note that all the S_i are independent of each other and do not interact. I.e. any segment can only be part of one edge and consequently a member of only one of the S_i. So, if each S_i takes *ik* passes to reduce to E_i then S takes $|S|$ passes to reduce to S^m.

So, in order to carry out segment merging all that is required is to carry out a pass for each segment in the set merging segments if they are neighbours.

3.5.2. Implementation

The program *merge.c* takes the raw segments produced by the corner locator and merges them in the way described above to produce a much reduced array of parameterised line segments. It needs to be told the angle within which the segments can be considered orientationally similar as well as the distance within which they can be considered close. This enables the program to decide if two segments are neighbours or not.

The program uses a doubly nested loop to carry out a pass over the array for each segment in the array. This means that the merging is completed in time complexity $O(n^2)$.

Finally, before the segments are returned, any of them which have a length less than the closeness threshold are ignored. This is motivated by the heuristic that if merging is desired to occur at this distance, the distance must be considered insignificant. Thus, anything less can safely be ignored.

The array that is returned by this program holds the results to the monocular image processing in the form of an array of line segments with the following attributes;[6]

< edge_num, start_pnt(x,y), direction_vector(x,y), lambda,end_pnt(x,y), length>

This represents a line $l(\lambda)$ = **start_pnt+λdirection_vector** where lambda is the value of λ at the **end_pnt**. These segments go on to be used in the matching processes described in the next chapter.

3.6. EXAMPLE RUN

As described at the beginning of this chapter, a subroutine was constructed which carries out all the image processing for a given image. The figures of the next few pages show the intermediate results of the call;

```
lowlevel(lfile,Lraw,Lcan,Lpts,Ltrk,Lrsg,Lfsg,&LNumsegs)
```

on a single image.

Figure 3.12 shows *Lraw*, the image data loaded by the image capture subroutine.

Figure 3.13 shows *Lcan*, the edge detected image array after the Canny edge detection operation has been carried out. Note, the brighter the line, the stronger the edge at that point.

Figure 3.14 shows the results of Beattie's edge tracker by replotting the points from *Lpts* for each track in *Ltrk*. Note that the start and end point of each track are circled. It can be seen that where the edge was weak after edge detection the tracker produces fragmented edges.

The results of Malcolm's corner detector, *Lrsg*, are given in figure 3.15 in which only the start points of each edge are circled for clarity.

[6] Note that there is some redundancy in this representation. However if values such as length are calculated now they need not be repeatedly calculated during the matching phase.

Figure 3.16 shows the results of merging these segments, *Lfsg*. A threshold of 10 pixels was used to allow neighbouring segments closer then this to be merged. Again only the start points of each segment are circled. It can be seen that the value of *LNumsegs* in this case would have been 25.

These results are very good indeed with only one small cause for concern. The left end of the object is represented by a single long segment. This is perfectly acceptable however, as this is what appears in the original image data. It is a consequence of the angle that the object has been presented at. In order to avoid this sort of problem occurring, a heuristic could be built into the monocular processing. This heuristic would mean that in a scene where every face of an object is a single colour, there should be a corner at the end of every edge. If such a corner had been placed on the long edge at the boundary between the black and white faces, the problem would never have arisen.

Figure 3.12 - Raw image data from camera

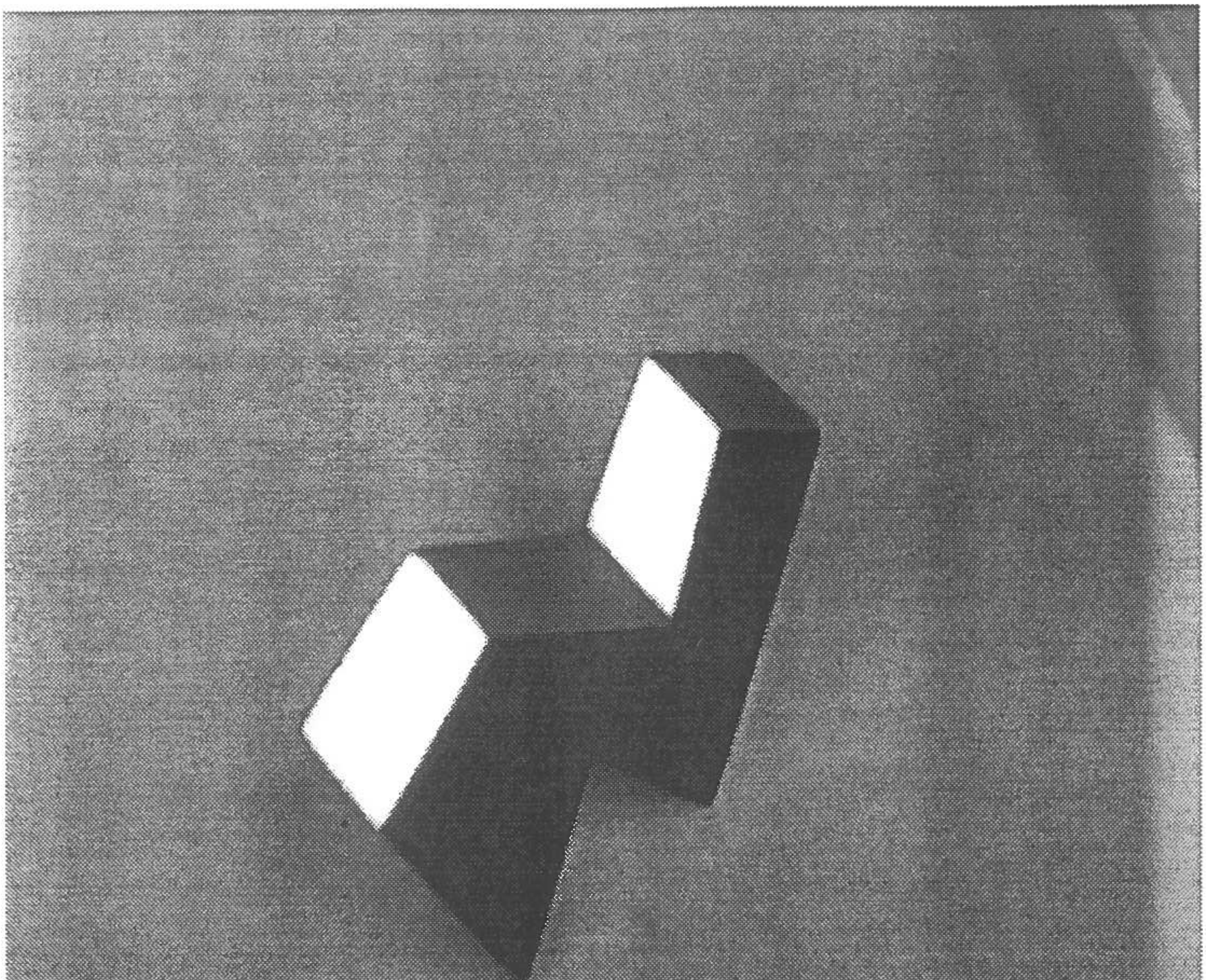


Figure 3.13 - Edge detected image data

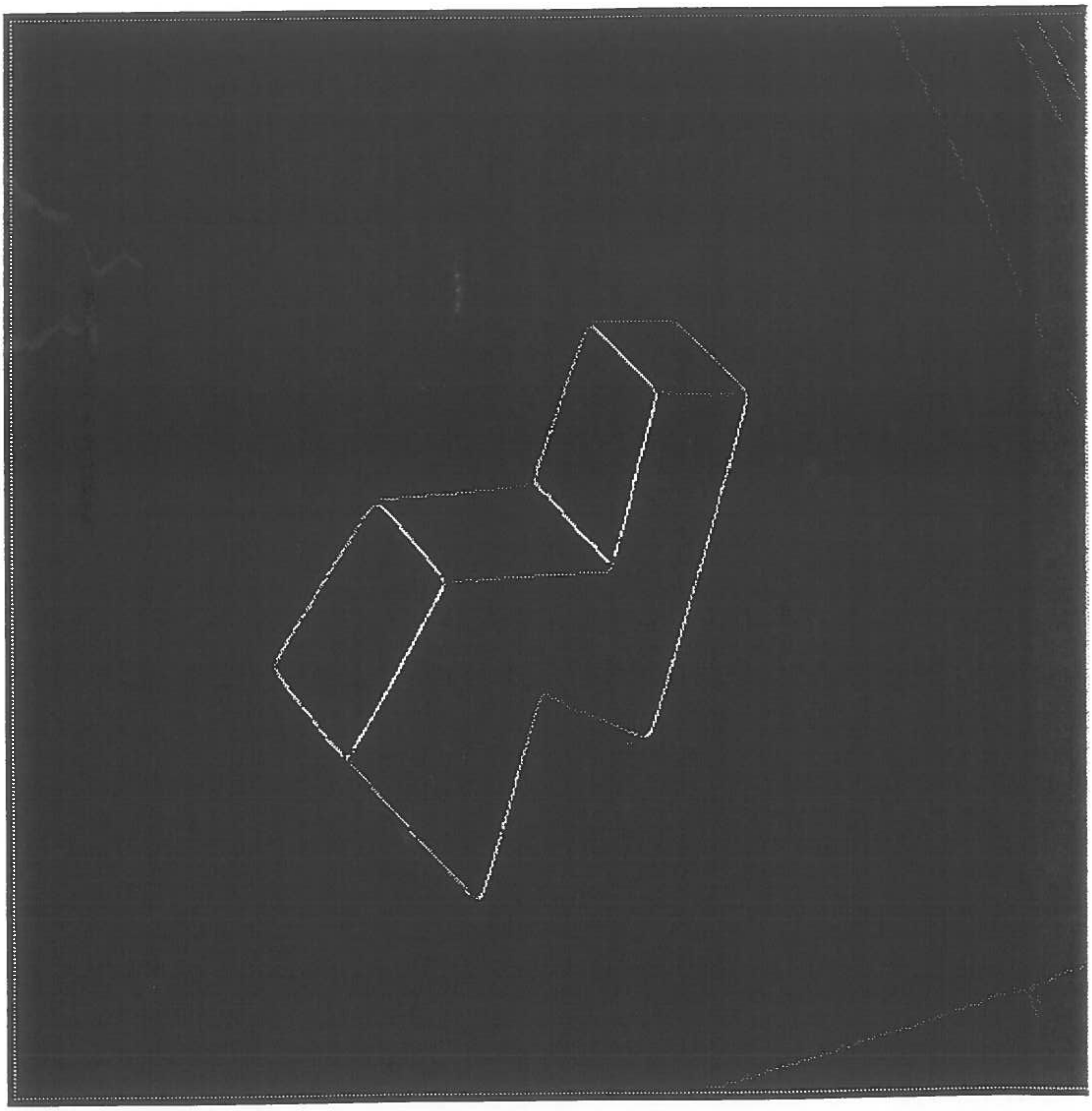


Figure 3.14 - Beattie's edge tracked data

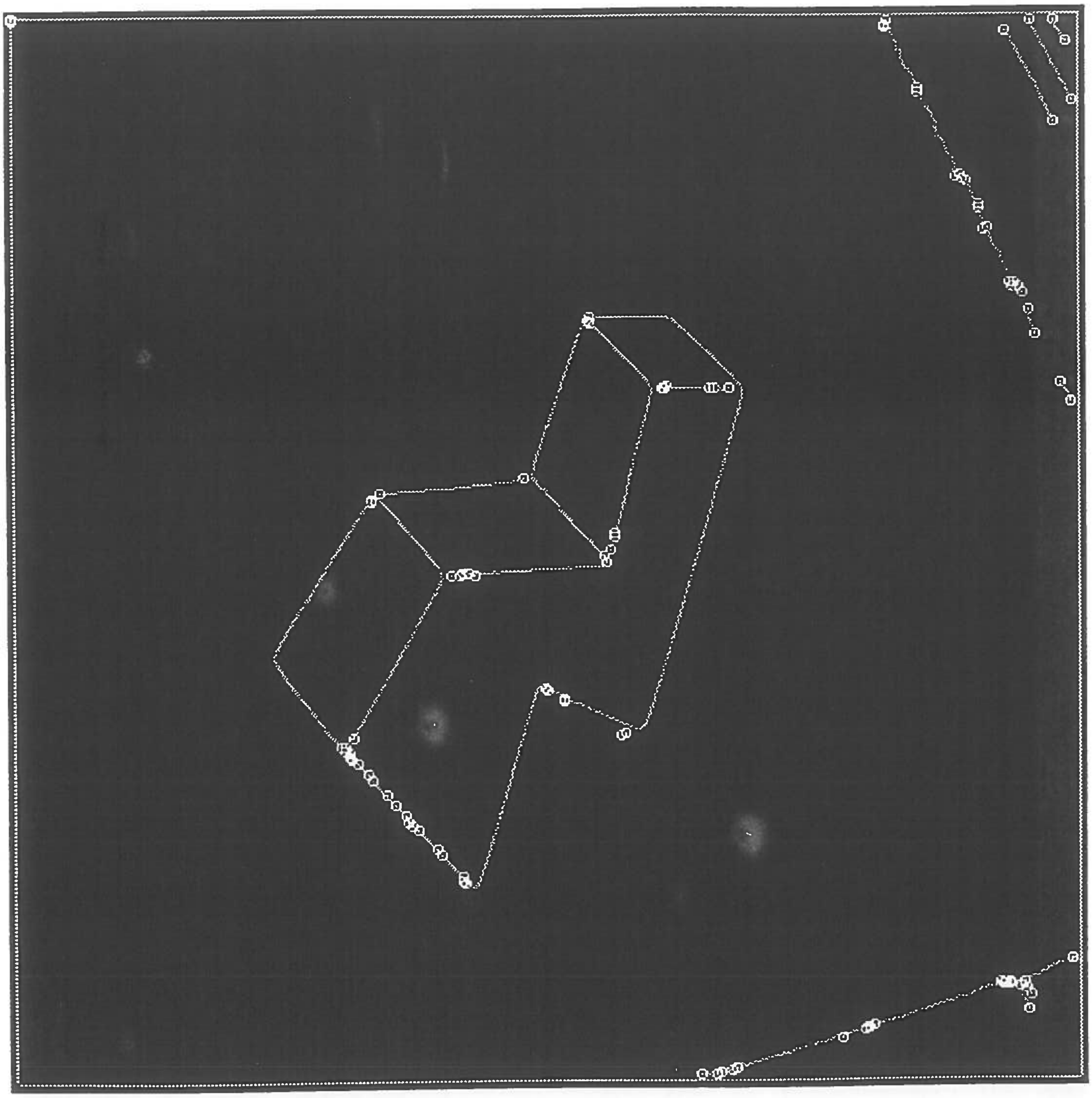


Figure 3.15 - Tracks with corners located

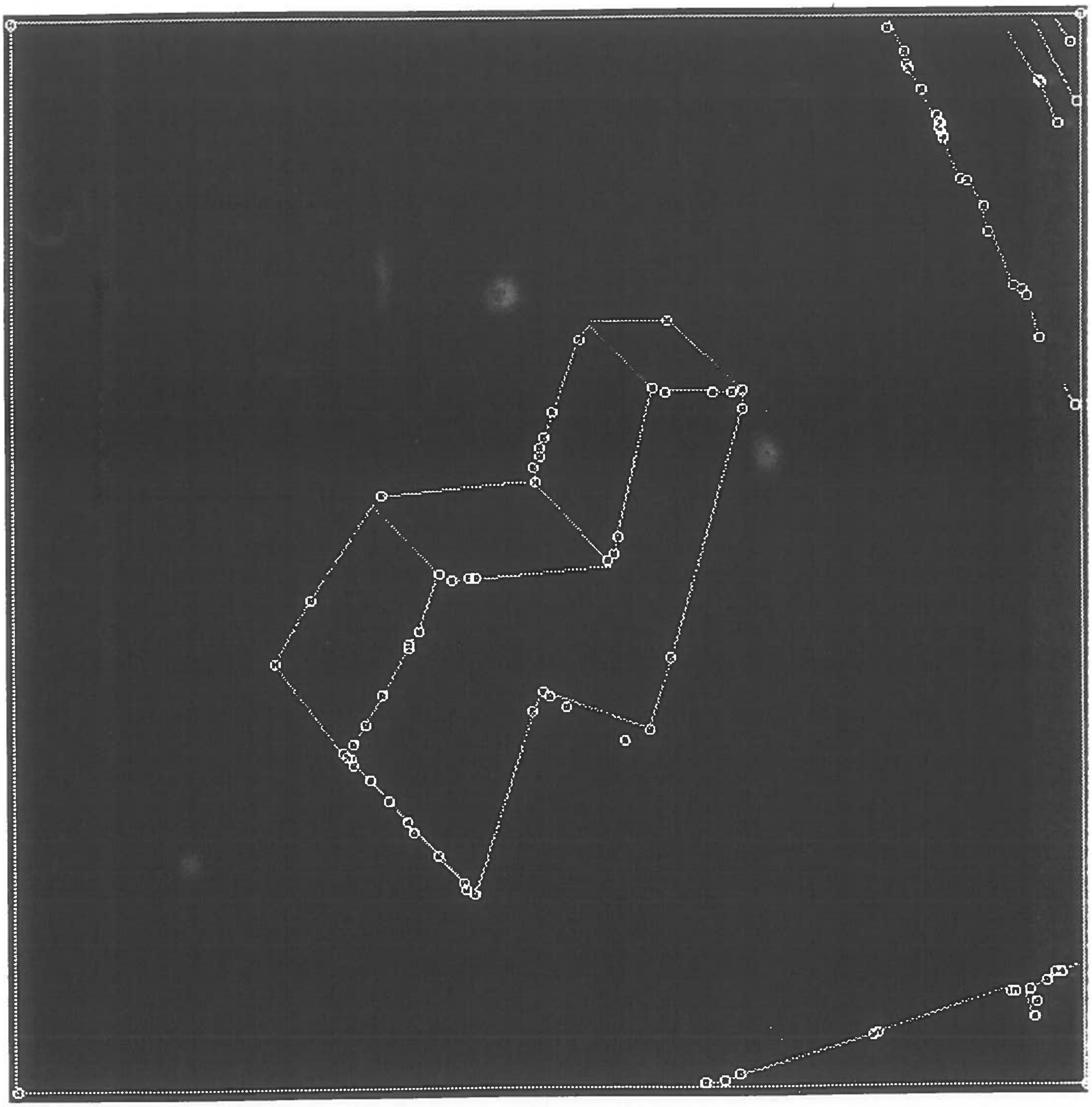
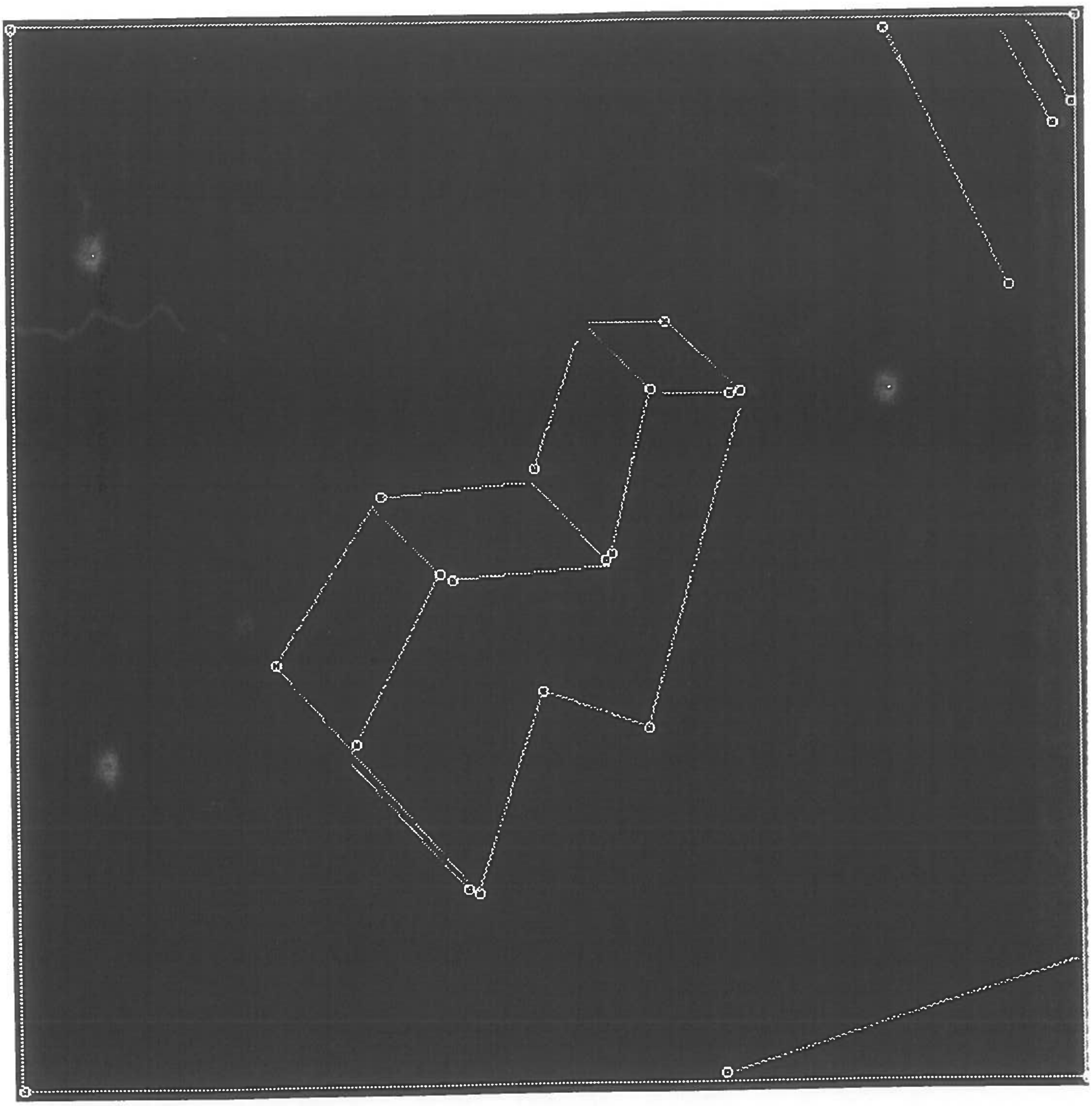


Figure 3.16 - Results of segment merging



CHAPTER 4

TRINOCULAR STEREO MATCHING

So far the monocular image processing has provided a set of 2D line segments describing the objects in each image. In order to recover the depth of these segments we must find the triples of segments which correspond to the same physical edge. Once it has these line segment triples the program can use the principle of triangulation to calculate the 3D world co-ordinates corresponding to the end points of the line segments.

The problem of finding these triples is termed the correspondence problem and is often considered to be the key problem in the field of stereopsis. E.g. with four points in each image there are $(4^4)^4$ *approx.* 4.29×10^9 potentially matching triples. In general three images each with n points will produce $(n^n)^n$ possible matches and this is clearly unacceptable.

Various constraints such as the uniqueness constraint[1] and the ordering constraint[1] vastly reduce the number of possible matches and in the case where there are the same number of points in each image and these points all correspond to the same world points, the problem becomes determinate.

However, in the real world where this case is the exception rather than the rule, we need to find other constraints to limit the combinatorial explosion of potential matches.

4.1. CONSTRAINTS UTILISED

There exists a very powerful constraint which derives from the 3D geometry of the camera set-up called the epipolar constraint. It is this constraint that serves as the basis for overcoming the correspondence problem in this project.

Figure 4.1 shows the epipolar geometry of trinocular stereo. We have three cameras, each modelled by an optical centre C_i and an image plane P_i. Given a physical

[1] The uniqueness constraint states that an image feature corresponds to a unique point in 3D space and hence each image point must match at most one image point in the other image.

[2] The ordering constraint states that matches must proceed from the left to the right of each image and

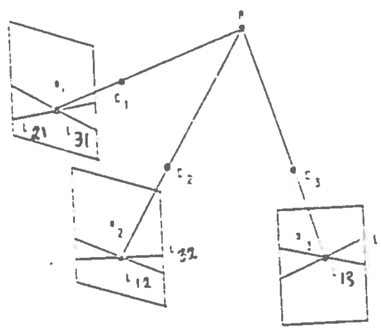


Figure 4.1 - Epipolar geometry for one point using three cameras

point A, its image in camera i is defined as the intersection of the straight line AC_i with the image plane P_i.[3] If we denote a_1, a_2, a_3 to be the image of A in each camera then they are called homogeneous image points.

Given a pair of cameras i and j and a physical point A the epipolar projection of a_i from i into j (L_{ij}) can be thought of as the intersection of the epipolar plane $P(C_i, A, C_j)$ with the image plane P_j. Ayache and Lustman state that;

"Any triple of homologous image points e.g. (a_1, a_2, a_3) is such that a_3 lies at the intersection of the epipolar projections of a_1 and a_2 into camera 3, L_{13} and L_{23} respectively."

This fact reduces the search for homologous image points between two images to a simple verification at a precise location in the third image. For instance, checking that a_1 and a_2 are a pair of homologous image points consists of verifying the presence of a_3 at the intersection of L_{13} and L_{23}. [Ayache & Lustman'87]

However, in the real world camera set-up the epipolar projections are seldom accurate and in order to constrain the search for pairs of homologous image points as well as to reduce the number of erroneous matches other ways of reducing the search must be found.

If the image points above are taken to be the midpoints of straight line segments representing the edges in each image, the number of features in each image that have to be corresponded is reduced considerably. This use of line segments also provides several other constraints.

One of these is the constraint that any two matching segments must be of similar length. Although the length of an edge in the various images will change due to

only applies if there is no transparency or self-occlusion present in the scene.

[3] This can be thought of as the ray of light entering the camera from that point in the scene.

differences in its distance from the various cameras and the differences in the focal length of the lenses, a general heuristic can be stated that the length of an edge in one image is never going to be more than a certain factor bigger or smaller than the length of its corresponding edge in another image. This would mean that a small edge would never be matched against a large edge even if it did lie on its epipolar projection.

Another effective way to check whether two segments are homogeneous, and one which is used by Cagenello is to work out the proportion of the segments that overlap. If the segments are homogeneous then the overlap will be approximately 100%. However, if the overlap falls below 50% it is highly unlikely that the two segments are homogeneous and the match can be rejected. (Figure 4.2)

As described in the introduction to the next section 4.2 the epipolars can not be relied upon to be accurate. This means that there must be some allowance made so that the feature in the third image does not have to lie exactly on the intersection of the projections from the other two images. This relaxation of the epipolar constraint means that there may be cases where some potential matches between segments satisfy these constraints while at the same time are not homogeneous. In this case the lines from the end-points of the segments can be projected back out into the world space. These lines correspond to the rays of light entering the cameras from a single point. If the rays do not converge to a point then the situation is a physical impossibility and the

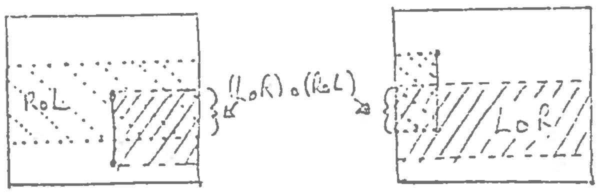


Figure 4.2 - Calculation of overlap between segments

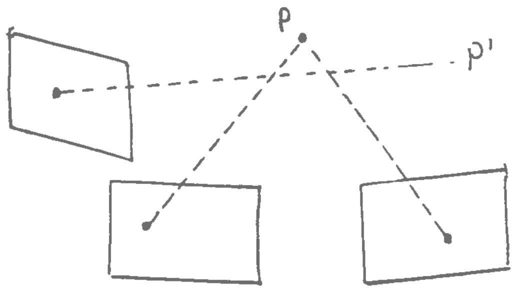


Figure 4.3 - Triangulation failure because of bad match

segments can not be homogeneous. (Figure 4.3)

Even after all the preceding constraints have been applied there still may be a few rogue matches remaining. Another constraint based on a priori knowledge of the scene set-up can be implemented by simply testing the 3D co-ordinates produced by the match to check that they are within the expected area of the scene. I.e. the location of the object can be restricted to a box in space defined by the maximum and minimum co-ordinate values expected for each axis. This "box" can be made large enough that any proposed match that results in a 3D co-ordinate outside the box can be rejected.

If the application of these constraints fails to make the set of potential matches un-ambiguous the uniqueness constraint can then be applied to finally disambiguate the matches.

4.2. THE TRIANGULATION PROCESS

The various constraints described above result in a set of potentially homologous triples of segments. For each triple the end-points can be projected back into the scene. If the points are homologous they will intersect at a point in the scene, the point that they all correspond to. So the 3D world co-ordinates of a line segment can be found by finding the 3D world co-ordinates of its end-points. (Figure 4.4) Note that if the three segments in the triple are not homologous their projections back into the scene will not intersect at a point. (Figure 4.3) This is how one of the latter constraints above is implemented.

This process suffers from one major drawback. The projections can never be relied upon to be accurate in a real system for several reasons;

- non-linearities in the cameras caused by unavoidable imperfections in the manufacturing of the lenses and other parts of the cameras,
- inaccuracies introduced during the process of camera calibration,

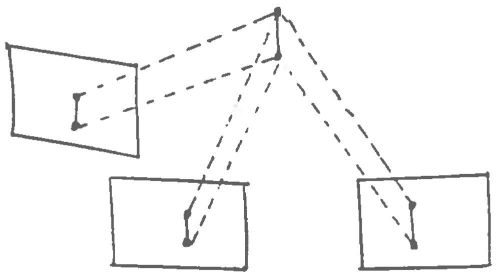


Figure 4.4 - Segment location by triangulation

- spherical aberration of the lens that bend images of straight lines.

This means that instead of the point of intersection having to be found, the point that minimises the distance to the three lines must be found instead.

4.2.1. Theory

Dr. J. Hallam of the A.I. department analysed this problem and came up with the following solution to find a point **x** closest to n nearly intersecting lines.[4]

Each line can be parameterised thus $l_i = \mathbf{p}_i + \lambda \mathbf{v}_i$, where $\mathbf{v}_i$ is a unit vector in the direction of the line and $\mathbf{p}_i$ is a point on the line.

Using the model of figure 4.5 we can see that the projection of $(\mathbf{x} - \mathbf{p}_i)$ onto l_i is given by $(\mathbf{x} - \mathbf{p}_i)\cdot\mathbf{v}_i$. Using pythagorus' theorem we have;

$$d_i^2 = |\mathbf{x} - \mathbf{p}_i|^2 - |(\mathbf{x} - \mathbf{p}_i)\cdot\mathbf{v}_i|^2$$

$$= (\mathbf{x} - \mathbf{p}_i)^t (I - \mathbf{v}_i\mathbf{v}_i^t)(\mathbf{x} - \mathbf{p}_i)$$

where t denotes the transpose of a vector.

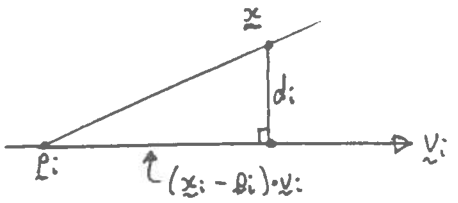


Figure 4.5 - Model used to calculate distance from point to line

[4] Note that for trinocular stereo there will be three intersecting lines, $n = 3$.

So, the total squared error is;

$$E^2 = \sum_{i=1}^{n} d_i^2 = \mathbf{x}^t L\,\mathbf{x} - 2\mathbf{m}^t\mathbf{x} + N \tag{4.1}$$

where

$$L = \sum_{i=1}^{n} \{I - \mathbf{v}_i\mathbf{v}_i^t\}$$

$$\mathbf{m} = \sum_{i=1}^{n} \{I - \mathbf{v}_i\mathbf{v}_i^t\}\,\mathbf{p}_i$$

$$N = \sum_{i=1}^{n} \mathbf{p}_i^t\,\{I - \mathbf{v}_i\mathbf{v}_i^t\}\,\mathbf{p}_i$$

Equation 4.1 is known to be minimised if;

$$\mathbf{x} = L^{-1}\mathbf{m} \tag{4.2}$$

$$and\ \ E^2 = N - (\mathbf{m}^t L^{-1}\mathbf{m}). \tag{4.3}$$

So, $\mathbf{x}$ is the point in 3D that comes closest to being an intersection of the three lines. A measure of the error in the approximation is given by E^2. [5]

4.2.2. Implementation

The subroutine *stereotriple* implements the operations described in the last section. It takes three 2-dimensional points and returns $\mathbf{x}$, a 3-dimensional point closest to the scene point corresponding to all three image points and E^2 the error value.

Its first task is to project the points back into the scene and parameterise the resultant rays to obtain the various l_i. This it does in a subroutine *get_ray* which calculates the co-ordinates of two points on the ray, one at the camera's focal point and the other some distance into the scene.[6] Using these points it then derives the parameters $\mathbf{p}_i$ and $\mathbf{v}_i$ which characterise the ray.

A loop is then entered to calculate the sums, L, $\mathbf{m}$, and N. Each time round the loop a subroutine *calc_ray* is called which returns the terms associated with each ray l_i. These are then added to the running totals.

After the loop is completed the inverse of L is calculated using the formula;

[5] If E^2 is large then it is more than likely that the three points are not homogeneous.

[6] The subroutine is based on several subroutines written by Orr which are described in § 6.2. The theory behind back projection is described in the latter part of § 6.1.

$$L^{-1} = \frac{1}{det(L)} \cdot adj(L)$$

where $det(L)$ is the determinant of L calculated using the Rule of Sarrus and $adj(L)$ is the adjugate of L. [Lennox & Chadwick'77, Brand & Sherlock'70]

Once the inverse of L is known, both x and E^2 can be calculated using equations 4.2 and 4.3.

So, given three potentially homologous image points we can calculate the 3D world co-ordinates of their corresponding scene point.

4.3. THE MATCHING ALGORITHM

At the centre of the process of trinocular stereo is the matching process. This is the part of the system that produces the potential matches between the segments in the images which are then used to produce the 3D co-ordinates of the objects in the scene. This process utilises the epipolar constraint of §4.1 to limit the search for matching segments in any image to a single line.

It is not a complex process and its derivation follows directly from the statement that for three homologous points the third point lies at the intersection of the epipolar projections of the other two.

The algorithm below, based on that of Ayache & Lustman is the standard trinocular matching algorithm presented in the literature. The following notations are used. A line L_{XY} is the projection of a_X into *Image Y*. Each point a_X is the mid-point of the corresponding segment S_X.

```
FOR each segment $S_1$ of Image 1
DO
   project $a_1$ into Image 2 to give $L_{12}$
   project $a_1$ into Image 3 to give $L_{13}$
   FOR each segment $S_2$ of Image 2
   DO
      IF ($S_2$ intersects with $L_{12}$)
      THEN
         project $a_2$ into Image 3 to give $L_{23}$
         IF ($L_{13}$ intersects $L_{23}$ at a point $a_3$)
         THEN
            IF a segment $S_3$ lies within a certain neighbourhood of $a_3$
            THEN keep $\{S_1,S_2,S_3\}$ as a potential match.
```

Note that if L_{12} only intersects with one segment we could establish the correspondence without using the third image. However, the third image still offers useful information even in this case. If the verification in the third image succeeds than the extra segment will improve the accuracy of the triangulation. If the verification fails then it is just as well we checked because we have prevented a false match.

Many issues arise out of considering the use of the algorithm above. These are discussed in the remainder of this section.

4.3.1. Representation of epipolar projections

Theory

There are several places in the algorithm where a point is required to be projected into an image to form an epipolar projection. In theory, this epipolar projection should be a straight line running from one side of the image to the other. (Figure 4.6)

However, due to the same problems of camera manufacture, calibration and spherical aberration as described in § 4.2 these projections are more often than not irregular curves. This raises the issue of how to represent the projections so that they can be used by the rest of the algorithm. -

Because the epipolar projection is the main-stay of the algorithm this issue is a major consideration in the design of the system.

The most obvious choice of representation is that of a series of points which lie on the curve. (Figure 4.7) However, a more usable representation is obtained if these

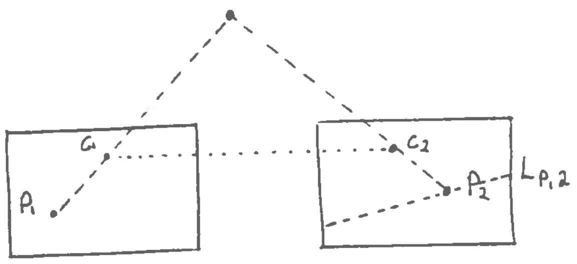


Figure 4.6 - The projection of a single epipolar

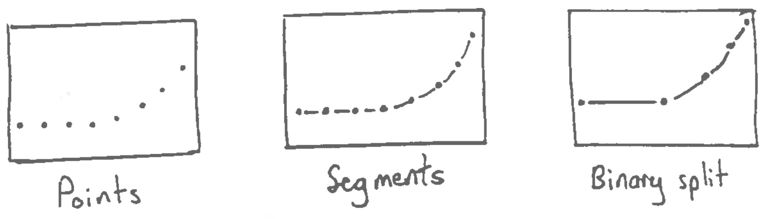


Figure 4.7 - Various representations of an epipolar projection

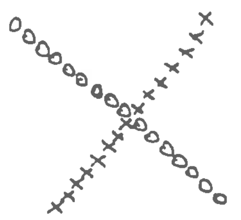


Figure 4.8 - The problems of calculating intersections using a points representation

points are joined up to form line segments. This reduces the number of points necessary to represent the projection while at the same time overcoming the inherent difficulties of calculating intersections between lines represented as points illustrated in figure 4.8. To enable intersections to be calculated using the line segment representation another loop needs to be added to the correspondence algorithm to index through their "epipolar segments". E.g.

```
FOR each segment $S_2$ of Image 2
DO
   FOR each segment $e_{12}$ of $L_{12}$
   DO
      IF ($e_{12}$ intersects $S_2$)
      THEN . . .
```

This representation is a good one but it can be improved upon if instead of arbitrarily segmenting the projection every few points it is segmented when the curvature of the projection means that considering it to be a line any longer would introduce an unacceptable error into the approximation provided by the representation. Put simply this results in less segments being required by the representation the straighter the epipolar projection is. It is desirable to keep the number of line segments needed to represent an epipolar down as much as possible as this results in an increase in speed. However, this may result in an unacceptable level of errors when the segments are used in calculations such as finding the intersection of two epipolars. A way round this is to use the segment representation to find out whether the epipolars intersect and then to return to the point representation to locate the intersection point accurately.

In comparing these representations it can be noted that the single points representation would require approximately 500 (the width of the image in pixels) points for each projection and consequently 500 calculations to be carried out to test for an intersection. The latter representation using the line segments requires at most 5 segments to represent the vast majority of its projections and consequently only has to carry out

1% of the calculations to test for an intersection. Although each calculation is slightly more involved for the latter representation, the dramatic decrease in the number of calculations performed outweighs this slight increase in complexity and this is the representation used in the project.

Implementation

The subroutine *pline* projects a point's epipolar into an image and returns an array holding the segments that make up the projection. It was adapted from one of the programs in Orr's suite [Orr 1984] to enable it to produce the epipolar in segmented form instead of by a set of points.

The program obtains the ray in the scene that corresponds to the point in the first image and steps down the ray projecting points into the second camera at regular intervals. It does this very roughly until the points start to fall within the image boundaries. At this stage the program changes so that the points fall at regular intervals in the second image as opposed to projecting points at regular intervals on the ray. This ensures that the points give a good approximation to the epipolar projection.

The task of obtaining points at regular intervals in the second image is achieved by iteratively doubling and halving the step that is taken down the ray. If the point is too far from the last point the step value is halved and the calculation is carried out again. If the point is too close to the last point the step value is doubled and the calculation us carried out again. This iteration continues until the projected point falls within a certain percentage of the required distance from the last point projected.

The projection process finishes when the points being projected run off the other side of the image. By this stage the epipolar is represented by a set of points in the second image. A recursive function *split* is now called to carry out the segmentation using a binary split on the points.

The subroutine is given a linked list of points and if the angle between the line segments formed from the first to middle and the middle to last points is greater than a certain threshold then the first point's pointer is set to link it to the last point and the routine returns. This means that the program has considered that the line segment first:last is a good approximation to the set of points it was passed. However if the bend in the points was noticeable the subroutine splits the list of points in the middle and calls itself recursively on both halves.

The base case is reached if a list is ever less than three points. If this is the case the list is left as it is and the routine returns.

Finally, to enable *pline* to return the segmented epipolar, it makes a pass over the updated linked list of points recording the $\mathbf{p}_l$, $\mathbf{v}_l$ and λ of each segment in the array.

4.3.2. Verification in the third image

As was explained in § 4.1 if a match is proposed between segments in the first two images, the match can be simply checked by projecting the two segments' midpoints' epipolars into the third image and checking whether there is a segment where the two epipolar projections intersect. This provides a convenient means of verifying a hypothesised match between two segments and corresponds to the following portion of the algorithm.

```
IF ($L_{13}$ intersects $L_{23}$ at a point $a_3$)
THEN
    IF  a segment $S_3$ lies within a certain neighbourhood of $a_3$
    THEN keep $\{S_1,S_2,S_3\}$ as a potential match.
```

However, because of the representation of epipolars as sets of line segments (§ 4.3.1), this intersection process has to try to intersect every segment of L_{13} with each of those in L_{23} to find the intersection of the two epipolars. This means that another two levels of *FOR* loops are added to the algorithm at this point. This is not as bad as it may seem however because as soon as the intersection is found, the *FOR* loops may be exited in the knowledge that the epipolars will only intersect in one point. On average this will halve the number of intersection tests that need to be carried out. The required changes are;

```
FOR each segment $e_{13}$ of $L_{13}$
    FOR each segment $e_{23}$ of $L_{23}$
        IF ($e_{13}$ intersects $e_{23}$ at a point $a_3$)
        THEN
            IF  a segment $S_3$ lies within a certain neighbourhood of $a_3$
            THEN keep $\{S_1,S_2,S_3\}$ as a potential match.
            exit from the FOR loops
```

The other stage of the verification which can only be carried out once the intersection point of the epipolars has been calculated is that of determining which segments lie within a certain distance of the point. All these segments constitute possible matches with the two from the other images. The issue of defining a "neighbourhood" is dealt with by using a threshold to determine when a segment lies close enough to the point to be considered in the neighbourhood.

Theory

The distance from the point to a segment can be one of three basic cases as shown in figure 4.9. In these figures, $\vec{AB}$ is the line segment being considered and P the intersection point involved. |PP'| is the distance from the point to its perpendicular projection on the line segment.

Once | PP'| is known, | BP'| (or | AP'| in case 2) can be worked out using Pythagorus' theorem without any difficulty. A comparison of this distance with | AB| will determine whether the shortest distance between P and $\vec{AB}$ is |AP'| as in case 1, | BP'| as in case 2, or |PP'| as in case 3.

The 2D perpendicular projection of a point (x_0,y_0) can be calculated using the model of figure 4.10 and equation (4.4) where $L=Ax+By+C=0$.

$$d=\frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}} \tag{4.4}$$

For any line $L=\begin{bmatrix}x_1\\y_1\end{bmatrix}+\lambda\begin{bmatrix}x_d\\y_d\end{bmatrix}$ we know that;

$$\frac{y-y_1}{x-x_1}=\frac{y_d}{x_d}\equiv x_d y-x_d y_1=y_d x-y_d x_1$$

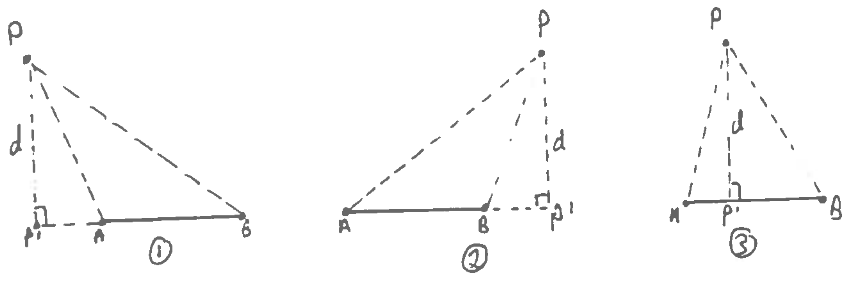


Figure 4.9 - The three cases of the relation of a point to a line

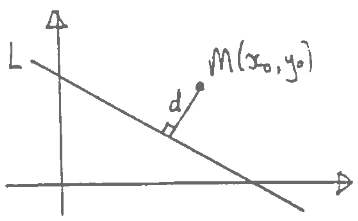


Figure 4.10 - Model used for point to line calculations

$$\equiv x_d y - y_d x - x_d y_1 + y_d x_1 = 0$$

$$\text{and } so, \quad A = -y_d,\ B = x_d,\ C = -x_d y_1 + y_d x_1 \tag{4.5}$$

So, substituting these values into equation (4.4) we get,

$$d = \frac{|-y_d x_0 + x_d y_0 - x_d y_1 + y_d x_1|}{\sqrt{y_d^2 + x_d^2}}$$

So, any segments within a certain distance of the intersection point, calculated as above, should be considered a potential matching third segment for the pair of segments that defined the intersection point.

Implementation

This mathematics is implemented in a subroutine *distance* which is passed a line segment and a point and returns the smallest distance between them. This is called at the appropriate stage in the trinocular matching algorithm to determine whether the line segment is within a certain neighbourhood of the point.

4.3.3. Calculation of intersections

At several places in the algorithm there is a requirement to calculate the intersection of two 2D line segments. This process is simply a process of solving two simultaneous equations.

Theory

Consider two lines $l_1 = \mathbf{p}_1 + \lambda_1 \mathbf{v}_1$ and $l_2 = \mathbf{p}_2 + \lambda_2 \mathbf{v}_2$. At the point of intersection, $l_1(\lambda_1) = l_2(\lambda_2)$. We must solve $\mathbf{p}_1 + \lambda_1 \mathbf{v}_1 = \mathbf{p}_2 + \lambda_2 \mathbf{v}_2$ for λ_1 and λ_2.

$$\equiv \quad \mathbf{p}_1 - \mathbf{p}_2 = \lambda_2 \mathbf{v}_2 - \lambda_1 \mathbf{v}_1$$

$$= \begin{bmatrix} \lambda_2 v_{2x} - \lambda_1 v_{1x} \\ \lambda_2 v_{2y} - \lambda_1 v_{1y} \end{bmatrix}$$

$$= \begin{bmatrix} -v_{1x} & v_{2x} \\ -v_{1y} & v_{2y} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$$

$$\equiv \begin{bmatrix}\lambda_1\\ \lambda_2\end{bmatrix} = \begin{bmatrix}-v_{1x} & v_{2x}\\ -v_{1y} & v_{2y}\end{bmatrix}^{-1} \begin{bmatrix}p_{1x}-p_{2x}\\ p_{1y}-p_{2y}\end{bmatrix}$$

$$= \frac{1}{v_{1y}v_{2x}-v_{1x}v_{2y}} \begin{bmatrix}v_{2y} & -v_{2x}\\ v_{1y} & -v_{1x}\end{bmatrix}\begin{bmatrix}p_{1x}-p_{2x}\\ p_{1y}-p_{2y}\end{bmatrix}$$

Note that if $v_{1y}v_{2x}-v_{1x}v_{2y}$ is 0 then the lines are parallel and no intersection can be calculated.

Implementation

In order to determine whether two line segments intersect all that has to be done is to calculate the point of intersection of the two lines and make a check as to whether this point falls within the bounds of both segments. This is done by comparing the values of λ_1 and λ_2 returned from the process above to the values of lambda obtained for each line segment when it was parameterised. E.g. if the range of the λ_i for two line segments are [0,6] and [0,1] and the values of λ_1 and λ_2 returned are 3 and 0.25 respectively then the lines do indeed intersect.

4.3.4. The problem of parallel lines

As was stated in the last section, if two lines are parallel no intersection can be calculated. This can be extended to state that if two lines are nearly parallel then it will be difficult to calculate their intersection accurately.

This causes problems in the algorithm as it stands because many segments turn out to be parallel to epipolars of other segments. The most obvious case of this is the standard binocular case of figure 4.11 where both cameras are horizontally aligned and the object has horizontal edges. Any epipolar projections of these edges will be horizontal and parallel to the corresponding segments in the other image. This makes the calculation of intersections for horizontal lines very inaccurate indeed and in most

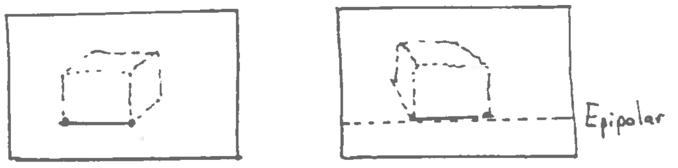


Figure 4.11 - Parallel lines in the binocular set-up

binocular stereo systems a choice is made to ignore these cases.

However, in trinocular stereo we have the third camera which offers the opportunity to avoid this problem. Unless the third camera is horizontally aligned with the other two cameras it is impossible for the epipolar projection to be horizontal in that image as well. This means that there will always be at least one image where intersections will be viable.

This necessitates a check on parallelity to be carried out before any intersections are carried out to determine which images to use. This drastically changes the original matching algorithm to take into account the various possibilities of parallelity between the images. Figure 4.12 shows the new final algorithm.

Note that this algorithm improves on the standard one offered by Ayache and Lustman because matches can be proposed for all segments respective of the relative orientations between the cameras. This means that no edges in the image are ignored as they would be in the original version if they turned out to be parallel between two of the cameras.

Implementation

The full algorithm is implemented in *produce_match* which takes the segments of all three images and returns an array with the potential matches in it. The program reflects the algorithm of figure 4.12 directly except for the addition of the extra loops required to cope with the epipolar representation as described in § 4.3.1.

The test for parallelity is made by a subroutine which calculates the largest distance between the two lines and the angle between them. If the lines are close enough together and nearly parallel then they are considered parallel for the purposes of the algorithm.

Each match has the following attributes;

< first_seg, second_seg, third_seg, code, distance>

These correspond to the identifiers of the three potentially homologous segments as well as an indication of which cameras were used to carry out the intersections which discovered the match (code). This is because these same cameras must be used when calculating the maximal overlaps of § 4.4. The distance field holds the distance that the segment was away from the intersection of the epipolars in the top image.

```
FOR each segment $S_1$ of image 1
DO
    project $a_1$ into Image 2 to give $L_{12}$
    project $a_1$ into Image 3 to give $L_{13}$
    FOR each segment $S_2$ of Image 2
    DO
        IF ($S_2$ is NOT parallel to $L_{12}$)
        THEN
            IF ($S_2$ intersects with $L_{12}$)
            THEN
                project $a_2$ into Image 3 to give $L_{23}$
                IF ($L_{13}$ is parallel to $L_{23}$)
                THEN
                    report an error - this should never happen
                ELSE
                    IF ($L_{13}$ intersects with $L_{23}$ at a point $a_3$)
                    THEN
                        IF a segment $S_3$ lies within a certain neighbourhood of $a_3$
                        THEN
                            IF ($S_3$ is NOT parallel to $L_{13}$) && ($S_3$ is NOT parallel to $L_{23}$)
                            THEN
                                keep $S_1,S_2:S_2,S_3$ as a potential match
                            ELSE
                                IF ($S_3$ is parallel to $L_{13}$)
                                THEN
                                    keep $S_1,S_2:S_2,S_3$ as a potential match
                                ELSE
                                    keep $S_1,S_2:S_1,S_3$ as a potential match
        ELSE
            project $a_2$ into Image 3 to give $L_{23}$
            IF ($L_{13}$ is parallel to $L_{23}$)
            THEN
                report an error - this should never happen
            ELSE
                IF ($L_{13}$ intersects with $L_{23}$ at a point $a_3$)
                THEN
                    IF a segment $S_3$ lies within a certain neighbourhood of $a_3$
                    THEN
                        IF ($S_3$ is NOT parallel to $L_{13}$) && ($S_3$ is NOT parallel to $L_{23}$)
                        THEN
                            keep $S_1,S_3:S_2,S_3$ as a potential match
                        ELSE
                            report an error - this should never happen
```

Figure 4.12 - The updated algorithm to take parallelity into account

4.4. FINDING THE MAXIMAL HOMOLOGOUS OVERLAP

The matching algorithm described in the previous section results in a set of potential matches between segments. Although within a match the segments may be homologous and correspond to a scene edge, each segment may cover a slightly different part of that edge. The final operation to be carried out before triangulation is to obtain the largest part of each segment that is present in all three images. This ensures that the end-points of each segment correspond to exactly the same world point.

4.4.1. Theory

This is done by a series of overlapping operations between images. Figure 4.13 shows two segments which have had the epipolars of their endpoints projected into each others image. Note that for each segment projecting its endpoints into the other image limits the overlap between the two segments to be the part of the other segment that lies between the epipolar projections. So, to obtain the maximal overlap between the two segments we must calculate the region (LoR)o(RoL), where 'o' indicates the overlap operation.

If we introduce the notation (XY) to represent the part of Y that is between the epipolar projections of the end-points of X then the part of L that falls within (LoR)o(RoL) can be written ((LR)(RL)) while the part of R within the region can be written ((RL)(LR)).

This procedural notation represents a segment in terms of how to go about calculating it. If we represent the segments declaratively in terms of the information implicit in them, that is as sequences of image identifiers with no definition of the order that the overlap operations are to be carried out in, we can utilise the projection algebra of appendix A to reduce them to their canonical forms;

$$\begin{array}{l} LRRL \rightarrow LRL \rightarrow RL \\ RLLR \rightarrow RLR \rightarrow LR \end{array}$$

Ideally we need a way to calculate an equivalent to both RL and LR with the least number of projections.

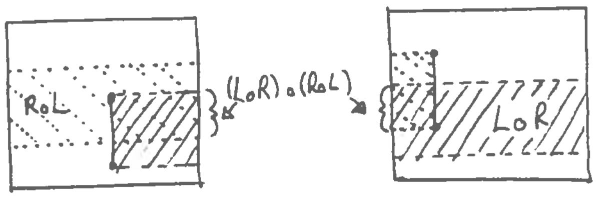


Figure 4.13 - The overlap operation between two images

All the following will do, they all require two overlaps to be carried out.

1) LR followed by (LR)L,
2) LR followed by RL,
3) RL followed by (RL)R,
4) RL followed by LR.

Things are a little more complicated for the three camera case. We require;

$$(LoR)o(ToR) \rightarrow LRTR \rightarrow LTR$$
$$(RoL)o(ToL) \rightarrow RLTL \rightarrow RTL$$
$$(LoT)o(RoT) \rightarrow LTRT \rightarrow LRT$$

However, this doesn't seem to exploit the camera set-up and appears a little naive. The scheme above requires at least seven overlap operations to be carried out; RT followed by L(RT) followed by (LRT)R followed by LT followed by R(LT) followed by (RLT)L followed by (LT)(RT), is just one solution.

Another better solution is to go round the images in turn overlapping as we go;

$$LTRLT \rightarrow LRLT \rightarrow RLT$$
$$LTRL \rightarrow TRL$$
$$LTR \rightarrow LTR$$

This scheme produces equivalent results to the naive scheme but only requires 4 overlaps; LT followed by (LT)R, followed by (LTR)L, followed by (LTRL)T.

However because of the parallelity which may be encountered between two images as described in the last section, it would be nice to be able to calculate the maximal overlap without having to use all the pairs of images. This example shows a scheme which doesn't use the pair LR;

$$LTRTL \rightarrow LRTL \rightarrow RTL$$
$$LTRT \rightarrow LRT$$
$$LTR \rightarrow LTR$$

This can be seen to produce identical results to the naive scheme and still only requires 4 overlaps to be calculated; LT followed by (LT)R, followed by (LTR)T, followed by (LTRT)L. It can be shown in a similar way that equivalent schemes exist which exclude the other camera pairs TR and LT.

These last three schemes are the ones used by the system to calculate the maximal homologous overlap between segments in the three images.

4.4.2. Implementation

The schemes above are implemented by the subroutine *produce_overlaps* which takes the segments of all three images as well as the array of potential matches and returns it with the maximal overlaps worked out for each match.

The subroutine takes each individual match and depending on which cameras were used to derive the match, selects one of the following schemes;

LRTRL
LTRTL
RLTLR

It utilises a subroutine *combine* which carries out the projection of one segment into another image to find out the overlap between the two segments in terms of the second segment. This subroutine takes the two segments concerned and the lambdas of the end-points of the maximal overlap so far on the first segment. It then projects these end-point's epipolars into the second image. The new maximal overlap is that part of the second segment that lies within the two epipolar projections and can be characterised by the lambdas of its end-points. With reference to figure 4.14 these are $\max(0,\lambda_{12})$ and $\min(\lambda_2,\lambda_{11})$.

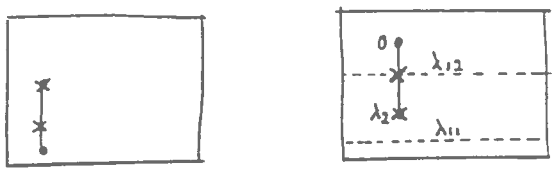


Figure 4.14 - Updating the maximal overlap

So, the first scheme above can be carried out by the following calls;

```
lam_start= 0; lam_end= 1;
combine(left_seg, right_seg, &lam_start, &lam_end);

combine(right_seg, top_seg, &lam_start, &lam_end);
/* lam_start and lam_end now characterise the maximal overlap in the top image */
/* and are now saved for use later on */

combine(top_seg, right_seg, &lam_start, &lam_end);
/* lam_start and lam_end now characterise the maximal overlap in the right image */
/* and are now saved for use later on */

combine(right_seg, left_seg, &lam_start, &lam_end);
/* lam_start and lam_end now characterise the maximal overlap in the left image */
/* and are now saved for use later on */
```

The other schemes are carried out by simply switching the order that the arguments are passed to the sub-routine.

So, after each match has been processed by *produce_overlaps* it has the old attributes;

```
< first_seg, second_seg, third_seg, code, distance,...
```

as well as the new ones;

```
fir_seg_pnt1(x,y), fir_seg_pnt2(x,y),...
sec_seg_pnt1(x,y), sec_seg_pnt2(x,y),...
thr_seg_pnt1(x,y), thr_seg_pnt2(x,y)>
```

These hold the image co-ordinates of the homologous end-points in each image and provides the information to both disambiguate matches and carry out triangulation with. These operations result in the final matches between segments with the world co-ordinates of their end points in the following form ready to pass on to the model matcher;

```
< start_point(x,y,z), error_start, end_point(x,y,z), error_end, length, match_number>
```

The errors are the E^2 terms from the triangulation process (§ 4.2) and the match_number is the index of the match in the array holding the potential matches. The other fields are those that are passed on to the model matcher; the 3D co-ordinates of both ends of the edge as well as its length.

4.5. EXAMPLE RUN

The following pages show the results produced by the trinocular matcher for a single segment. Figure 4.15 shows the three images and the segment under consideration is indicated in the top left hand image, Image1. Consulting the algorithm of figure 4.12 it can be seen that the segment is projected into the other two images to give L_{12} and L_{13}. From figure 4.17 it can be seen that these epipolars have 4 and 2 segments respectively and can be seen in figure 4.15 indicated in a similar way to the original segment.

The next stage of the algorithm steps through the segments of the second image finding those which intersect L_{12} (while at the same time not parallel to L_{12}). Of the three segments that do this, only two of them (segment 4 and 14 from figure 4.17) satisfy the similar length constraint described in 4.1. These are highlighted in blue in the bottom right image.

The algorithm now states that each of these segments should be projected into the third image to give L_{23} and the intersection point calculated. It can be seen from both figure 4.15 and 4.17 that only segment 14 produces an intersection in the third image at (411,246).

Now that the intersection point has been located the verification of 4.3.2 can take place. It can be seen from figure 4.15 that there is indeed a third segment lying close to the intersection and figure 4.17 confirms this as segment 12 at a distance of 4.58 pixels. This then is kept as a potential match (10,14,12).

Figure 4.16 shows the construction of the maximal overlap between these three segments. As was described in 4.4.2, the scheme LRTRL has been used for this task. The white lines are the projections of the endpoints of each segment and the purple highlight is the part of each segment which is exactly homogenous.

With reference to figure 4.17 it can be seen that the triangulation produces the following co-ordinates for the end-points of the homogenous segments.

(-23.73,5.24,48.49) - (-25.99,-0.02,47.00)

Comparing this to the measured object corner co-ordinates gives;

Source	End-point1			Endpoint2		
	x	y	z	x	y	z
Calculated	-23.73	5.24	48.49	-25.99	-0.02	47.00
Measured	-23.5	5.3	48.5	-26.1	0.0	46.7
Error(cm)	0.23	0.06	0.01	0.11	0.02	0.3

Table 4.1 - Results for a single segment

It can be seen that these results are very good. From figure 4.16 it can be seen that the calculated end-points of the segment do not correspond exactly with the real corners of the object and this will probably account for most of the error. However, even if this wasn't the case and the end-points should have been exactly where the corners were, the error can be blamed on the approximations introduced by the epipolar representation.

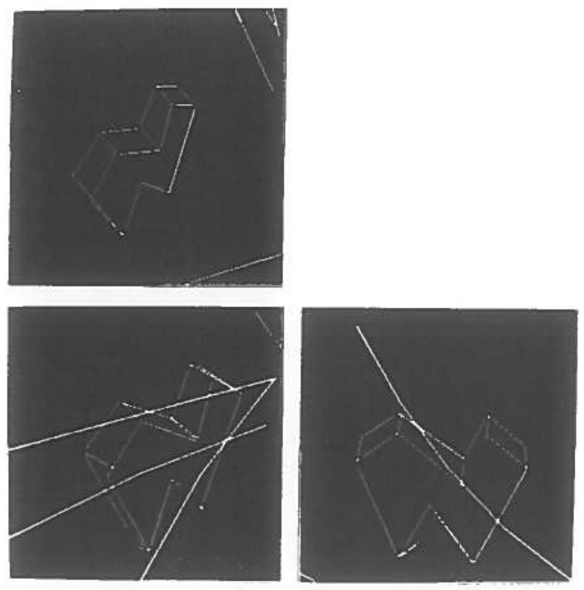


Figure 4.15 - Stages of the matching algorithm for a single segment

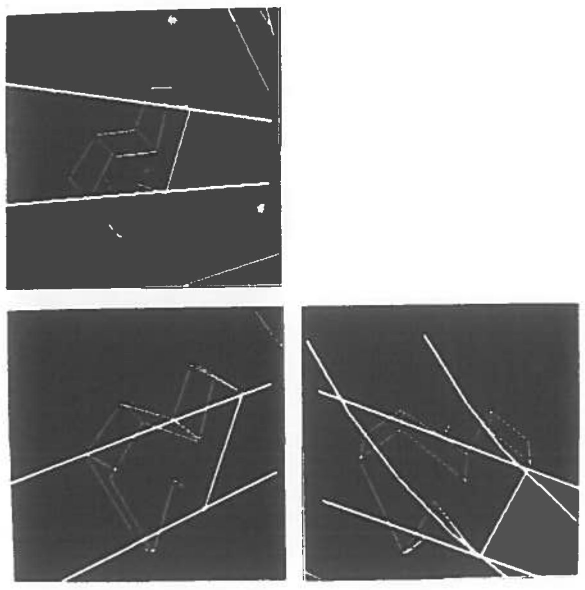


Figure 4.16 - Construction of the maximal overlap for a single segment

```
PRODUCE MATCHES:
Which segment are you interested in : 10
SEGMENT : 0
SEGMENT : 1
SEGMENT : 2
     "
     "
SEGMENT : 9
SEGMENT : 10
First segment: 10, (347,188)-> (-44,151),1-157.28
          1-> 2 has 4 segments.
          1-> 3 has 2 segments.
          Image1 segment 10 possibly matches with Image2 segment 4
                    2-> 3 has 1 segments in its epipolar
          Image1 segment 10 possibly matches with Image2 segment 10
                    Segment 10 is the wrong size.
          Image1 segment 10 possibly matches with Image2 segment 14
                    2-> 3 has 2 segments in its epipolar
                    Epipolars intersect at 411.54,245.92 in the third image.
                    1:10  2:14  3:12  (4.58)
SEGMENT : 11
SEGMENT : 12
SEGMENT : 13
     "
     "
SEGMENT : 24
SEGMENT : 25
There are 41 potential matches.

PRODUCE OVERLAPS:
......................................

There are 16 matches to disambiguate:-
          10:14:12 (a)/9.14    (-23.73,5.24,48.49)-0.0482    (-25.99,-0.02,47.00)-0.0629
          12:16:21 (r)/4.95    (-28.91,3.08,45.53)-0.0515    (-30.33,0.04,44.66)-0.0952
          14:20:19 (a)/17.92   (-33.57,4.72,43.74)-0.1821    (-31.99,1.99,44.22)-0.1195
          14:20:20 (a)/12.26   (-33.37,4.97,43.47)-0.0196    (-30.94,0.69,44.23)-0.0931
          15:9:9 (r)/9.58      (-27.07,6.51,49.37)-0.0418    (-28.06,4.59,48.82)-0.0697
          16:10:8 (a)/4.58     (-28.04,4.48,48.37)-0.0614    (-27.22,4.44,46.74)-0.0646
          16:10:10 (a)/2.02    (-28.05,4.46,48.39)-0.0663    (-27.26,4.35,46.86)-0.0693
          18:9:6 (a)/3.10      (-26.42,7.18,47.43)-0.2794    (-26.44,7.38,45.37)-0.4754
          20:4:13 (t)/18.88    (-28.92,4.98,48.13)-0.0409    (-31.84,6.87,46.37)-0.1015
          21:5:13 (a)/8.16     (-31.63,6.78,46.03)-0.0348    (-30.91,7.19,44.58)-0.2502
          21:5:14 (a)/5.66     (-31.72,6.66,46.20)-0.0808    (-31.28,6.68,45.23)-0.1062
          22:6:17 (a)/10.04    (-31.43,6.35,45.09)-0.0258    (-33.32,4.94,43.95)-0.0454
          23:7:15 (a)/1.50     (-32.37,6.33,46.91)-0.0305    (-34.10,5.01,46.31)-0.0422
          24:18:8 (t)/16.58    (-27.52,4.78,46.22)-0.0348    (-30.09,7.56,43.62)-0.1602
          24:18:10 (t)/4.32    (-27.49,4.56,46.47)-0.0254    (-30.08,7.47,43.71)-0.1367
          24:18:18 (t)/2.49    (-27.46,4.26,46.77)-0.0333    (-30.90,6.43,45.14)-0.0278

These are the segments after disambiguation :-
          10:14:12 (a)/9.14    (-23.73,5.24,48.49)-0.0482    (-25.99,-0.02,47.00)-0.0629
          12:16:21 (r)/4.95    (-28.91,3.08,45.53)-0.0515    (-30.33,0.04,44.66)-0.0952
          14:20:20 (a)/12.26   (-33.37,4.97,43.47)-0.0196    (-30.94,0.69,44.23)-0.0931
          15:9:9 (r)/9.58      (-27.07,6.51,49.37)-0.0418    (-28.06,4.59,48.82)-0.0697
          16:10:10 (a)/2.02    (-28.05,4.46,48.39)-0.0663    (-27.26,4.35,46.86)-0.0693
          20:4:13 (t)/18.88    (-28.92,4.98,48.13)-0.0409    (-31.84,6.87,46.37)-0.1015
          21:5:14 (a)/5.66     (-31.72,6.66,46.20)-0.0808    (-31.28,6.68,45.23)-0.1062
          22:6:17 (a)/10.04    (-31.43,6.35,45.09)-0.0258    (-33.32,4.94,43.95)-0.0454
          23:7:15 (a)/1.50     (-32.37,6.33,46.91)-0.0305    (-34.10,5.01,46.31)-0.0422
          24:18:18 (t)/2.49    (-27.46,4.26,46.77)-0.0333    (-30.90,6.43,45.14)-0.0278
```

Figure 4.17 - Sample output, highlighting a single segment

CHAPTER 5

THE MODEL MATCHER

The model matcher is a program written by Watson as part of his M.Sc. degree in 1985.[1] [Watson 1985] It was written in conjunction with Cagenello's binocular stereo program and uses the 3D co-ordinates of edge points to classify and locate objects.

The matcher is supplied with a set of models of objects which it might expect to have to classify. When presented with the 3D co-ordinates of the end-points of the edges found by a stereo program from a test object it will try and match them to the models to find the corresponding one. Once the matcher has found the model which it thinks corresponds to the one being seen by the stereo program, it can calculate a transformation matrix from the data and model co-ordinates and suggest the location as well as the identity of the object.

5.1. THEORY

The first thing that the program does is to carry out a pre-match test to select a subset of the model set which will contain the models most likely to succeed in the match. It does this in the following way;

```
FOR each model edge $m$
DO
   FOR each data edge $o$
      IF (length of $o$ is within 20% of length of $m$)
      THEN add 1 to the number of corresponded model edges.
```

It does this for each model, rejecting those in which the percentage of corresponded model edges falls below a certain threshold. This test is designed to eliminate any models which are the wrong size to begin with.

Attempts are then made to match each model against the object by finding representative sub-geometries from the object and the model which appear to correspond, estimating a transformation matrix from them and verifying the remainder of the model edges until the number of verifications passes some threshold.

[1] This program was later revised by Croft and Fisher to integrate it with the existing HIPS environment.

If this process succeeds, a more accurate transformation matrix can be found using all the edges that have been found to correspond and the matcher succeeds.

This process is;

```
FOR each model edge $M_j$
DO
   FOR each data edge $D_i$
   DO
      IF ( $M_j$ and $D_i$ are incompatible )
      THEN next $D_i$
      ELSE
      {  record $M_j$ and $D_i$ as first pairing
         IF (sub-algorithm succeeds for $M_j$ )
         THEN exit matcher with transformation matrix $T$
      }
refine the transformation matrix


sub-algorithm :

FOR model edge $M_k$ ($k > j$)
DO
   IF ( $M_k$ and $M_j$ are parallel or collinear)
   THEN next $M_k$
   ELSE
   {
      FOR each data edge $D_i$
      DO
         IF ($M_k$ and $D_i$ are incompatible)
         THEN next $D_i$
         ELSE
         {
            record $M_k$ and $D_i$ as a second paring
            estimate the transformation matrix
            IF (estimate fails)
            THEN next $D_i$
            IF (the edges don't verify)
            THEN next $M_k$
            ELSE return verified edges and matrix $T$
         }
   }
```

The incompatibility test in the main algorithm is the same as that carried out in the pre-test, their length must be within 20% of each other. However, the test in the sub-algorithm has two additional conditions. Firstly, the data edge can not be the same one that corresponded with the first model edge and secondly the data edge must not be parallel or collinear to the data edge that did correspond to the first model edge.

The process for estimating the transformation matrix involves estimating both the rotational and transformational components required to map the model edges in the pairings onto their corresponding data edges. The details of the mathematics are presented in Watson's report and are based on Faugeras and Hebert's work. [Faugeras and Hebert 1983]

The verification of the edges begins by checking that the second pairing verify under the transformation. If this pairing do not verify then no other match can hope to succeed because the transformation was calculated using this pairing.

The verification proceeds thus;

```
IF (second pairing does not verify)
THEN find a new transformation matrix
ELSE
{
    FOR every other model edge $M_i$
    DO
        verify the edge $M_i$
    IF (the number of verified edges exceeds a certain threshold)
    THEN match succeeds with the matrix $T$
    ELSE find a new transformation matrix
}
```

The threshold value for a match is dependent on the orientation and geometry of the object. I.e. how many edges can potentially be hidden for some given orientation. Side and plan views are particularly bad for this.

In order for a model edge to verify with a data edge they must satisfy the following criteria;

(1) they must be compatible in length
(2) they must be close together
(3) they must be nearly parallel
(4) they must have a large overlap.

If a model succeeds in being matched against the unknown object, the verified edge pairs are passed on to the refinement part of the matcher.

The final transformation matrix is found using a least squares algorithm on the set of verified model/data edge end-points and the estimated transformation matrix. However, the accuracy of the final matrix is more dependent on the accuracy of the location of the data edges than the number of verified pairs available.

5.2. IMPLEMENTATION

The model matcher is implemented as part of the *wire* program which also allows models to be manipulated.

The matcher must be provided with a set of models to select the best match for the test object from. These models are hierarchical wire frame models with the primitives; points, edges, faces and parts. Points do not need to be explicitly defined since they constitute part of the edge description. An edge is defined in terms of its two end points. A face is defined by a set of edges, which must form a complete loop. A part is an arbitrary collection of faces, which is treated as a unit by the program.

Model files must contain part definitions, followed by face definitions, which must include the faces specified in the part definitions, followed by the edge definitions which must include the edges specified in the face definitions. E.g. to describe a right angled triangle in the x-y plane, with an apex at the origin and two sides a and b with a length of 50 units, the following model description could be used;

```
PART(triangle)= FACE(face1);

FACE(face1)= LINE(a) LINE(b) LINE(hypotenuse);

LINE(a,(0,0,0),(50,0,0));
LINE(b,(0,0,0),(0,50,0));
LINE(hypotenuse,(50,0,0),(0,50,0));
```

The matcher checks these model descriptions for syntactic integrity as it reads in the model file.

The *match* command carrys out the model matching once the model file has been read in. If a data file has not already been read in and correspondences found, this has to be done before the pairing and geometrical processes can be carried out. Several of the thresholds used during matching may be set from the command line.

If a successful model match is found, a list of the matching model edges and the corresponding data edges is printed out along with the estimated and refined transformation matrices.

The successful match may be displayed overlaid on the original match under the transformation in order to check the result.

5.3. EXAMPLE RUN

Unfortunately the model matcher could not be made to produce reasonable results from the data obtained from the trinocular phase of the processing. The model of the test object defined in figure 5.1 was used by the matcher.

The matcher successfully found enough edges to estimate and refine a transformation matrix as is shown in figure 5.2. However, it can be seen with reference to figure 5.3 that the matchings that it used to obtain these were in fact not correct. This means that the transformation matrix will be incorrect and can not be used to overlay the model on the original data in order to check the results.

```
PART(camel)= FACE(front) FACE(back) FACE(left) FACE(right) FACE(top1)
            FACE(top2) FACE(top3) FACE(top4) FACE(bot1) FACE(bot2);

FACE(front)= EDGE(ft1) EDGE(ft2) EDGE(ft3) EDGE(ft4) EDGE(fr) EDGE(fb2) EDGE(fb1) EDGE(fl);

FACE(back)= EDGE(bt1) EDGE(bt2) EDGE(bt3) EDGE(bt4) EDGE(br) EDGE(bb2) EDGE(bb1) EDGE(bl);

FACE(left)= EDGE(lt1) EDGE(fl) EDGE(lb1) EDGE(bl);

FACE(right)= EDGE(br) EDGE(rb2) EDGE(fr) EDGE(rt4);

FACE(top1)= EDGE(lt1) EDGE(bt1) EDGE(t1t2) EDGE(ft1);

FACE(top2)= EDGE(t1t2) EDGE(bt2) EDGE(t2t3) EDGE(ft2);

FACE(top3)= EDGE(t2t3) EDGE(bt3) EDGE(t3t4) EDGE(ft3);

FACE(top4)= EDGE(t3t4) EDGE(bt4) EDGE(rt4) EDGE(ft4);

FACE(bot1)= EDGE(lb1) EDGE(bb1) EDGE(b1b2) EDGE(fb1);

FACE(bot2)= EDGE(b1b2) EDGE(bb2) EDGE(rb2) EDGE(fb2);

EDGE(ft1,(-2.9,5.4,0),(-0.2,6.9,0));
EDGE(ft2,(-0.2,6.9,0),(1.2,4.3,0));
EDGE(ft3,(1.2,4.3,0),(5.4,6.6,0));
EDGE(ft4,(5.4,6.6,0),(8.1,5.2,0));
EDGE(fr,(8.1,5.2,0),(5.1,0,0));
EDGE(fb2,(5.1,0,0),(3.4,3.1,0));
EDGE(fb1,(3.4,3.1,0),(0,0,0));
EDGE(fl,(0,0,0),(-2.9,5.4,0));
EDGE(bt1,(-2.9,5.4,2.1),(-0.2,6.9,2.0));
EDGE(bt2,(-0.2,6.9,2),(1.2,4.3,2.3));
EDGE(bt3,(1.2,4.3,2.3),(5.4,6.6,2.1));
EDGE(bt4,(5.4,6.6,2.1),(8.1,5.2,2.1));
EDGE(br,(8.1,5.2,2.1),(5.1,0,3.0));
EDGE(bb2,(5.1,0,3.0),(3.4,3.1,2.4));
EDGE(bb1,(3.4,3.1,2.4),(0,0,3.0));
EDGE(bl,(0,0,3.0),(-2.9,5.4,2.1));
EDGE(lt1,(-2.9,5.4,0),(-2.9,5.4,2.1));
EDGE(lb1,(0,0,0),(0,0,3.0));
EDGE(rt4,(8.1,5.2,0),(8.1,5.2,2.1));
EDGE(rb2,(5.1,0,0),(5.1,0,3.0));
EDGE(t1t2,(-0.2,6.9,0),(-0.2,6.9,2));
EDGE(t2t3,(1.2,4.3,0),(1.2,4.3,2.3));
EDGE(t3t4,(5.4,6.6,0),(5.4,6.6,2.1));
EDGE(b1b2,(3.4,3.1,0),(3.4,3.1,2.4));
STOP
```

Figure 5.1 - The model of the test object

```
***********************************************verify_all: verification_count,edges_tested_total: 5 24
the following edges are verified:
try_match: model edge(fb2), object edge R1: min_dist 0.09 dp 0.999998
try_match: model edge(ft4), object edge R7: min_dist 0.27 dp 0.993795
try_match: model edge(ft3), object edge R9: min_dist 0.64 dp 0.990197
try_match: model edge(fr), object edge R2: min_dist 0.09 dp 0.999998
try_match: model edge(bl), object edge R0: min_dist 0.56 dp 0.990790
try_match: estimated transformation matrix is:

  -0.9123 -0.0352 0.4079 -25.7863
  -0.0072 0.9975 0.0702 -0.2525
  -0.4094 0.0611 -0.9103 46.5693
  0.0000 0.0000 0.0000 1.0000

refined transformation matrix is:

  -0.8678 -0.0447 -0.1014 -25.9470
  -0.0295 1.0194 0.0911 -0.1591
  -0.4470 0.0613 0.0743 46.9397
  0.0000 0.0000 0.0000 1.0000


SOLUTION FOUND (refined results just above)

***********************************************verify_all: verification_count,edges_tested_total: 7 24
the following edges are verified:
try_match: model edge(fb2), object edge R1: min_dist 0.16 dp 0.985977
try_match: model edge(ft4), object edge R7: min_dist 0.55 dp 0.981896
try_match: model edge(ft3), object edge R5: min_dist 0.15 dp 0.997539
try_match: model edge(bt3), object edge R9: min_dist 0.63 dp 0.993946
try_match: model edge(t1t2), object edge R4: min_dist 1.27 dp 0.996294
try_match: model edge(fr), object edge R2: min_dist 0.18 dp 0.989051
try_match: model edge(bl), object edge R0: min_dist 0.07 dp 0.999345
try_match: estimated transformation matrix is:

  -0.9195 0.0371 0.3913 -26.9261
  -0.0073 0.9938 -0.1114 0.0897
  -0.3930 -0.1053 -0.9135 49.2268
  0.0000 0.0000 0.0000 1.0000

refined transformation matrix is:

  -0.8657 -0.0126 0.4502 -26.9206
  -0.0205 1.0158 -0.0626 -0.7062
  -0.4614 0.0004 -0.8733 48.0766
  0.0000 0.0000 0.0000 1.0000


SOLUTION FOUND (refined results just above)
```

Figure 5.2 - Sample results from the matcher

Model edge	Data edge
(bl)	R0
(bb2)	R1
(br)	R2
(ft2)	R3
(t2t3)	R4
(ft3)	R5
(t3t4)	R6
(bt4)	R7
(ft4)	R8
(bt3)	R9

Figure 5.3 - Correct model data edge pairings

This is not as bad as it may at first appear because the model can be compared with the calculated data and the measured data directly and this will provide an indication of how good the stereo location has been. Results of this kind can be found in §7.2.

CHAPTER 6

THE CALIBRATION PROCESS

Throughout the previous chapters an assumption has been made that the epipolar geometry of the system has been known. This has enabled us to project rays corresponding to points in images back into the scene, project epipolars of points from one image into other images and reconstruct the 3D co-ordinates of homologous points. [Faugeras 1986] However, the derivation of the system's epipolar geometry which enables these operations to be carried out requires up to 15 independent parameters to be calculated for each camera in the system. These parameters define amongst other things the position of the camera's focal point, the position of its focal plane, the location of the origin of the image plane and the orientation of the image plane.

6.1. Theory

In his M.Sc. thesis, Orr presents the work of Yakimovsky and Cunningham where they describe the model of the linear camera of figure 6.1 in detail. [Orr'84, Yakimosky & Cunningham 1978] Rather than using the parameters above to describe the model they use an equivalent set of four vectors, **C** (the position of the focal point), **A** (where the camera is pointing - a unit vector) and two vectors **H** and **V**. **H** and **V** can be described as the horizontal and vertical vector of each camera respectively and are best defined by the following equations where I and J are the image plane co-ordinates and **P** is the real world position of a point projected onto the image at (I,J);

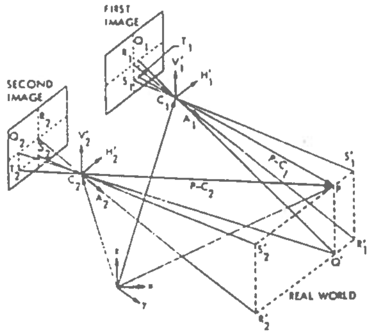


Figure 6.1 - A linear camera model set-up

$$I = \frac{(\mathbf{P}-\mathbf{C})\cdot\mathbf{H}}{(\mathbf{P}-\mathbf{C})\cdot\mathbf{A}} \tag{6.1}$$

$$J = \frac{(\mathbf{P}-\mathbf{C})\cdot\mathbf{V}}{(\mathbf{P}-\mathbf{C})\cdot\mathbf{A}} \tag{6.2}$$

In order to calibrate a camera we have to experimentally determine the values of the model parameters, in this case the four vectors **A**, **C**, **H**, **V**.

This is done by rewriting equations 6.1 and 6.2 as;

$$(P_1\times I)A_1+(P_2\times I)A_2+(P_3\times I)A_3-P_1\times H_1-P_2\times H_2-P_3\times H_3-I\times \mathrm{C}_a+\mathrm{C}_h=0 \tag{6.3}$$

$$(P_1\times J)A_1+(P_2\times J)A_2+(P_3\times J)A_3-P_1\times V_1-P_2\times V_2-P_3\times V_3-J\times \mathrm{C}_a+\mathrm{C}_v=0 \tag{6.4}$$

where $\mathrm{C}_a = \mathbf{C}\cdot\mathbf{A}$, $\mathrm{C}_h = \mathbf{C}\cdot\mathbf{H}$ and $\mathrm{C}_v = \mathbf{C}\cdot\mathbf{V}$.

Now, for a collection of reference points $\{\mathbf{P}_i\}$ and their associated image coordinates $\{(I_i,J_i)\}$ equations 6.3 and 6.4 yield two systems of homogeneous linear equations in the unknowns A_1, A_2, A_3, H_1, H_2, H_3, C_a and C_h from 6.3 and A_1, A_2, A_3, V_1, V_2, V_3, C_a and C_v from equation 6.4. Note that a subset of these are common because the direction that the camera is pointing is constant in both the horizontal and vertical planes of the linear camera model. These common parameters provide a means of cross checking the solutions of the two systems.

If a component of **A** is arbitrarily set to unity[1] the systems become inhomogeneous and each has seven unknowns.

After solving the two systems all the solutions can be rescaled so that the sum of the squares of the components of A is unity. Finally, the components of C can be calculated from C_a, C_h and C_v by solving the following linear equation system;

$$\mathrm{C}_a = C_1\times A_1+C_2\times A_2+C_3\times A_3$$

$$\mathrm{C}_h = C_1\times H_1+C_2\times H_2+C_3\times H_3$$

$$\mathrm{C}_v = C_1\times V_1+C_2\times V_2+C_3\times V_3$$

The solution of this series of linear equations completes the determination of the camera parameters. In theory the solution of the equations should be possible with just

[1] In practice some care is needed in choosing which component to set to 1 because dire consequences could result from trying to scale to unity a component which is actually zero.

seven points but in practice the more points that are available the better the solutions will be. Ayache & Faverjon suggest that up to 50 points may be required for accurate solutions to be obtained.

However, Orr discovered that small errors in the input data caused the solutions of the two linear systems to disagree on the values of the common parameters. In order to avoid this disagreement the constraint that these parameters should have the same solution in both linear equation systems was dropped. This means that equations 6.1 and 6.2 now become;

$$I = \frac{(\mathbf{P}\cdot\mathbf{H}-\mathbf{C}_h)}{(\mathbf{P}\cdot\mathbf{A}_i-\mathbf{C}_a{}^i)} \tag{6.5}$$

$$J = \frac{(\mathbf{P}\cdot\mathbf{V}-\mathbf{C}_v)}{(\mathbf{P}\cdot\mathbf{A}_j-\mathbf{C}_a{}^j)} \tag{6.6}$$

and equations 6.3 and 6.4 become;

$$(P_1\times I)A_1{}^i+(P_2\times I)A_2{}^i+(P_3\times I)A_3{}^i-P_1\times H_1-P_2\times H_2-P_3\times H_3-I\times\mathbf{C}_a{}^i+\mathbf{C}_h=0 \tag{6.7}$$

$$(P_1\times J)A_1{}^j+(P_2\times J)A_2{}^j+(P_3\times J)A_3{}^j-P_1\times V_1-P_2\times V_2-P_3\times V_3-J\times\mathbf{C}_a{}^j+\mathbf{C}_v=0 \tag{6.8}$$

The names $\mathbf{A}_i$, $\mathbf{A}_j$, $\mathbf{C}_a{}^i$ and $\mathbf{C}_a{}^j$ now distinguish between the two solutions of A and $\mathbf{C}_a$. The parameters are found in the same way as above by solving the systems of linear equations.

This new non-linear camera model provides significantly better results than those obtained by the linear model and are tolerant to measurement errors in both the various $\mathbf{P}_i$ and their associated (I_i,J_i).

In order to achieve the requirements of the previous chapters of projecting rays and epipolars as well as reconstructing 3D co-ordinates we must be able to determine the line in 3D space defined by an image point.

If we assume that the camera parameters and the image co-ordinates (I,J) are known then the equations 6.7 and 6.8 can be rewritten ;

$$P_1(I\times A_1{}^i-H_1)+P_2(I\times A_2{}^i-H_2)+P_3(I\times A_3{}^i-H_3) = I\times\mathbf{C}_a{}^i-\mathbf{C}_h \tag{6.9}$$

$$P_1(J\times A_1{}^j-V_1)+P_2(J\times A_2{}^j-V_2)+P_3(J\times A_3{}^j-V_3) = J\times\mathbf{C}_a{}^j-\mathbf{C}_v \tag{6.10}$$

and they define two planes which intersect in the required line.

This and related processes such as projecting this line into another image (§ 4.3.1) and finding the intersection point of several such lines (§ 4.2) allow all the operations for trinocular stereo to be carried out.

6.2. Implementation

Orr implemented these procedures in a suite of programs which aid in the calibration process;

collect	reads a file containing a list of 3D reference point co-ordinates physically measured in the scene and prompts the user to indicate the co-ordinates of the image of each point by moving the cursor until it is co-incident with them. It also asks the user which axis has the largest range of measurements taken from it. This is the one that will provide the best normalisation.
calib	takes these readings and calculates the camera parameters. It is at this stage that a bad choice of reference points may cause the program to be unable to calculate the parameters.
calchk	projects the reference points onto the original image. Note if the parameters are accurate these projections will be co-incident with the points selected using the cursor in *collect.*
pline	plots a line defined by a point in one of a pair of stereo images on a second image.
plotp	plots a 3D point on an image. This is used by *calchk* to plot all the original reference points.

The most important part of camera calibration is the selection of the reference points in the scene. Faugeras & Toscani (1986) suggest that for calibration to take place there must be at least 7 reference points and no set of 4 of them must be coplanar. If a subset of the points are co-planar then there is a redundancy of information and the system of equations that has to be solved may become ill-conditioned and insoluble.

This problem of ill-conditioning arises in practice because in a simple scene with just one object there are not enough non-coplanar points to reference. The solution is to add several other objects to the scene for calibration purposes and then to remove them again.

CHAPTER 7

CONCLUSIONS

Ultimately the project has achieved what it set out to do. It has fulfilled Marr's criteria of vision by "discovering from images what is present in the world, and where it is". It does this reliably and accurately as suggested by Ayache and Lustman and verified by the results obtained from the system.

7.1. DISCUSSION OF THE RESULTS

7.1.1. Geometric Accuracy

Because of the failure of the matcher to verify the calculated edge data using the model, the level of testing of results is not as good as may have been desired. However, several tests may still be carried out.

The first test that can be made is to see how close the calculated points come to the actual measured 3D co-ordinates. The following result, obtained for one of the object's corners, was presented in chapter 4 but is repeated here for completeness.

Source	End-point1			Endpoint2		
	x	y	z	x	y	z
Calculated	-23.73	5.24	48.49	-25.99	-0.02	47.00
Measured	-23.5	5.3	48.5	-26.1	0.0	46.7
Error(cm)	0.23	0.06	0.01	0.11	0.02	0.3

The furthest distance that either calculated point can be away from the measured point is 0.32cm using Endpoint2. This is a very small error considering the distances involved. Errors of this order of magnitude would probably make no difference to the performance of most systems that might use the trinocular location module. Most robot arms are not this accurate and if the robot was mobile, the accuracy would be further reduced.

Another test that can be carried out is to examine the accuracy of the orientation of the edges. The following table presents the values obtained for the calculated and the measured segments of a particular edge of the object.

	Measured	**Calculated**
Endpoint1	(-33.50, 5.00, 43.7)	(-33.37, 4.97, 43.47)
Endpoint2	(-30.4, 0.0, 44.3)	(-30.94, 0.69, 44.23)
Vector	(-3.1, 5.0, -0.6)	(-2.43, 4.28, -0.76)
Length	5.9135	4.9800

Using these values and the formula;

$$\cos\Theta = \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}| \times |\mathbf{y}|}$$

we can obtain the following result, the angle between the measured and calculated segments is 3.7 degrees. This is a good result and again would probably not effect the performance of any system using the vision module.

The final test that can be carried out is to use the same method to calculate the angle between various pairs of the calculated segments. These can then be compared to the expected angles obtained from the original drawings for the models. (Figure 7.1)

The following values are for the third and eighth located segments from the sample output of figure 1.8 which form the angle marked on the drawing.

	Segment3	**Segment8**
Endpoint1	(-33.37, 4.97, 43.47)	(-31.43, 6.35, 45.09)
Endpoint2	(-30.94, 0.69, 44.23)	(-33.32, 4.94, 43.95)
Vector	(2.43, -4.28, 0.76)	(-1.89, -1.41, -1.14)
Length	4.9800	2.619

The angle between these vectors is 87.5 degrees, compared with an angle of 90 degrees on the drawing.

This result once again bears out the accuracy of the orientation of the located segments. This time the result shows that the relative orientation of the edges is calculatable to within 3 degrees of its expected value. These 3 degrees includes any error introduced during the machining of the object.

7.1.2. Edge Identification

It has already been shown (figure 1.8 and 1.9) that the trinocular processing locates 10 edges for the scene under consideration. This may at first appear bad when compared to the possible total of 17 visible edges. However, on closer examination it can be seen that for each of the seven edges that haven't been located, there is one of the three images in which they do not appear. Some of the edges are lost during edge

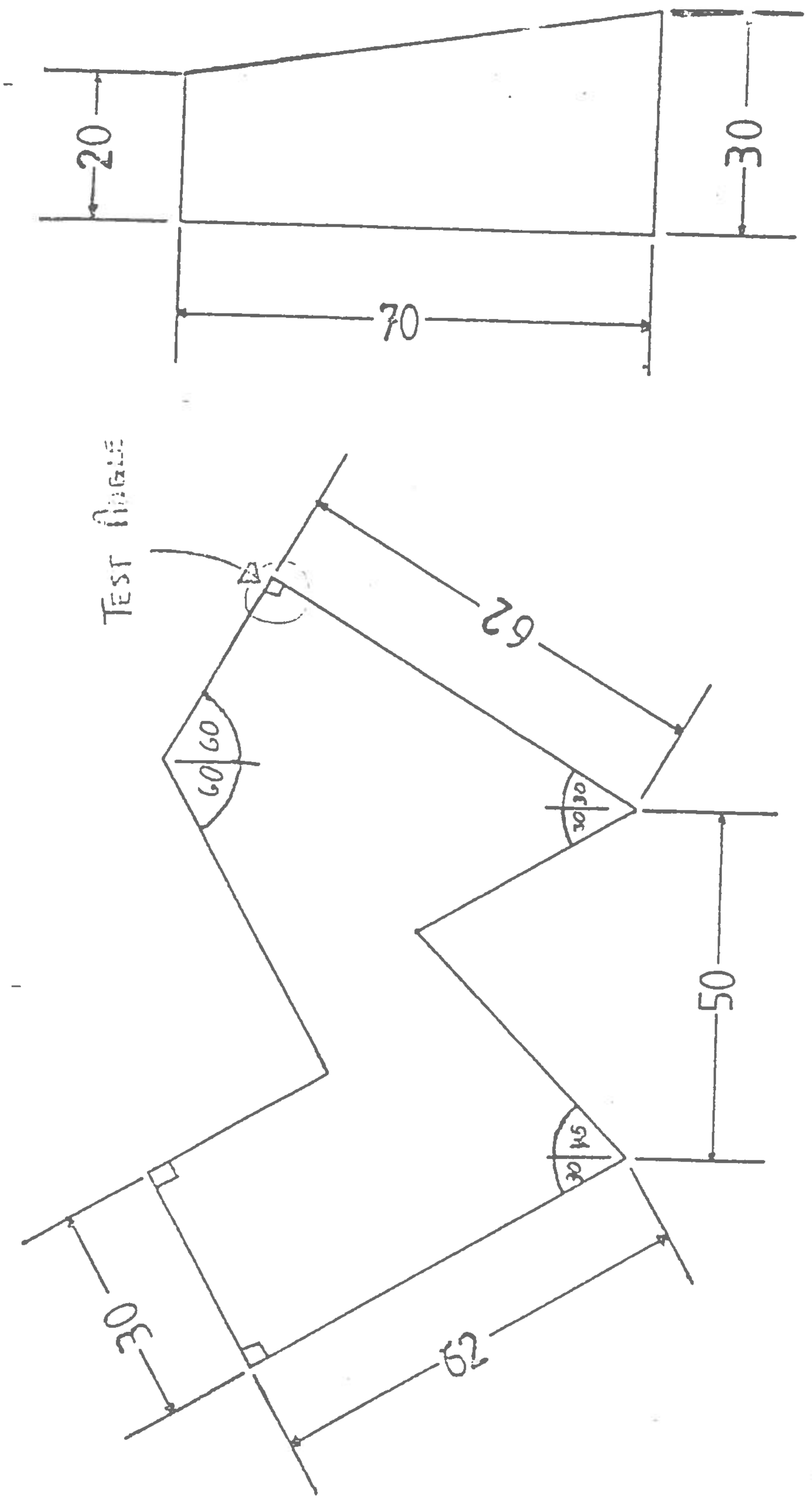


Figure 7.1 - The original drawing of the model

detection due to shadowing and the like, while others are occluded.

This problem of occlusion is one which arises in both trinocular and binocular schemes. However, in the trinocular case there is an obvious method to use to overcome it – binocular stereo. The two images in which the edge *does* appear are used to locate the edge. This task is made easier by the fact that a great deal of the matches that would have been proposed will have been eliminated by the trinocular process. This approach is explained in more detail in §8.4.

7.1.3. The Advantage of Trinocular

The advantages of the trinocular technique over the binocular one have been highlighted at various stages throughout this report but they are summarised here for completeness.

The major advantage of trinocular stereo over its binocular counterpart is its solution to the correspondence problem. This addition of a third camera turns a non-deterministic process into a deterministic one.

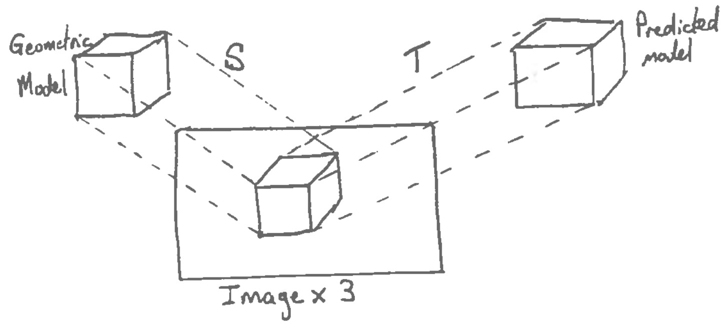
The three camera system also offers a solution to the problem of occlusion which means that it is possible for a match to be found even when the corresponding segment is missing from one of the images. (see 8.4 for details)

Another problem which is solved occurs when segments in either of the images of a binocular system are aligned with the baseline between the cameras. This makes matching these segments and any subsequent calculations using them very inaccurate indeed. However in the trinocular set-up there will always be at least two cameras whose baseline is not aligned with any particular segment and matching can continue unhindered.

The extra camera also improves the accuracy of the location of the points. The extra epipolar line which is projected back out into the scene increases the effectiveness of the least squares estimate of the location of each point.

Thus the process of stereopsis is made more efficient, more flexible and more accurate by the addition of a third camera. This is the claim made by Ayache and Lustman (1986) and it has been verified by the work reported here.

During the development of the system several other major issues arose, not specifically related to the field of stereopsis. These are discussed in the next section.



S is the synthesis transformation
T is the trinocular location module transformation
T will just be S^{-1}, but this trivialises its contribution.

— The figure 7.1 refered to on page 70. —

7.2. WORKING IN THE REAL WORLD

The problems created by working in the real world can never be under-estimated and are considerable. As soon as input is received or output is sent outwith the confines of the computer, the program becomes part of a larger, more complex and much less predictable system. These problems may range from a printer where paper can jam to a robot arm trying to pick up a block which can fall over to measurements changing over time due to expansion and contraction from heat.

It is these problems which caused the inaccuracies in the system such as the noise in original images and the curved epipolars as opposed to straight ones. In theory, an edge will always be exact and the three epipolars in the top image will always meet in a point but the fact that these objects (edges, epipolars) were derived from input from outside the bounds of the machine meant that the inaccuracies were inevitable.

One ploy of workers developing programs which purport to be realistic get round the problem by bringing the whole system back within the confines of the computer by simulating the world in which the program is to work. The images required by the system could have been synthesised from a description of the geometry of the test object but this would have trivialised the contribution of the system as can be seen with reference to figure 7.1. The use of simulation also means that the realism of the program merely becomes a function of the realism of the simulation. For example, a simulated printer is only realistic if once in a while the paper jams or the ribbon runs out or it is accidently turned off etc.

This approach to the realism of a simulation can be extended to state that no matter how complicated a simulation is, it will inevitably have to make some assumptions about the world it is modelling. If this is not done then the simulation would have to model the whole world.

Another ploy which can be used is to tackle the problems presented by working in the real world head on. This means making the program adaptive in a way that enables it to cope with the inaccuracies introduced by outside influences.

This can be achieved by incorporating certain levels of tolerance within the system by using thresholds to extend a particular value to a range of anticipated values.

These levels of tolerance can be implemented using threshold values (i.e. + / - error) to cover the range of values. These thresholds then, define the adaptability of the program to cope with the problems that the real world can produce. Thus any program that deals with the real world must have some thresholds which determine the level of performance of that program in the real world.

This is certainly the case with this project where there are between 20 and 30 explicit thresholds that need to be set. The consequence of this is that the performance of the program is a function of about 25 parameters with some parameters being independent and others having some level of dependence. All this leads to the problem of knowing what values to set the thresholds to in order to obtain the best performance from the program. This process normally comes down to an iterative refinement of the threshold values where a guess is made, the results examined and the value changed to try and improve the results. Ways to improve this are suggested in the next chapter (8.2) but this still remains the major problem with this approach.

CHAPTER 8

FURTHER WORK

During the development of the various stages of the system a number of ways to extend the basic project have presented themselves. These divide logically into two classes; those that could be used to improve the performance or ease the use of the system as it stands, these are presented in §8.1, and those that could extend the application areas of the project. These are discussed in the remainder of the chapter.

8.1. SPEED UPS

All the improvements described in this section are designed to improve the efficiency of the operation of the programs that constitute the project.

8.1.1. Storage of epipolars

As was stated in § 4.3.1 the theory of trinocular stereo relies heavily on epipolars and epipolar projections. Any improvement that can be made in the handling of these structures will benefit the performance of the system as a whole.

During any particular run of the system each segment's midpoint and end-points may be projected many times. The only way that their epipolar's and epipolar projection's can change is if the camera set-up is altered. This would necessitate a total re-calibration of each camera that was moved to determine its new parameters. Because the camera set-up can be assumed to stay the same throughout a run of the system the epipolars and their projections can be stored and re-used each time that they are used again. This saves a great deal of time as the calculation of these structures is fairly time consuming. If this scheme is adopted the number of epipolar calculations that are required becomes independent of the matching process and is instead linear in the total number of segments produced by the monocular processing.

Access to these structures is made easier by the fact that each epipolar will only ever be associated with one segment. This means that each segment could have fields in its structure which point to the various epipolars and epipolar projections associated with it. Initially these would be empty and would only be set as the epipolars were created.

This scheme would best be implemented as an abstract data type *epipolar(segment,code)*. When an epipolar was required, all that would be necessary to obtain an epipolar would be to supply the segment identifier and a code to signify whether it is information associated with the segment's mid-point or one of its end-points that is required. The program need never know whether the epipolar was pre-stored or had to be calculated from scratch. Note that this scheme also allows experimentation with the implementation techniques used for storage while not effecting the program itself.

8.1.2. Bucketing for segment storage

The other important objects that are used in trinocular stereo are the segments which describe the edges that have been found in each image. Again any optimisation that can be obtained in accessing these structures will dramatically increase the performance of the system as a whole.

Ayache and Faverjon (1987) suggest that by applying bucketing techniques, the access to segments can be considerably speeded up. The idea is to partition the image into square windows. Each window has a corresponding bucket attached to it which holds the list of segments that intersect the window.

This means that instead of having to search the set of all segments in the image, the program can limit its search to only those segments which are in the buckets associated with the windows through which a particular line passes in order to find the segments which may intersect with it.

The effectiveness of this improvement is dependent on the size of the windows. The larger the windows are the more segments will have to be searched through for each window. However, there is little point in having a large proportion of empty windows by making them too small.

This is again a question of experimenting with various different sizes until a suitable one for the application is found. However, if a large window size is decided upon then the search within each window can be reduced dramatically at the expense of an O(n log n) sorting of the segments into order by length before the bucketing is carried out. This means that a search for a segment of a known size within a bucket is reduced to a dichotomy search.

8.1.3. Parallelisation

Ayache and Lustman realised the parallelism of the basic algorithms involved and suggest that the outer loop of their matching algorithm can be split over as many processors as desired. This means that the left image segments can be distributed amongst the available processors in order to carry out the matching task. They claim to have achieved linear speed ups by using this straightforward approach on a Sequent Decaprocessor.

This parallelisation could be carried out in a similar way for the matching algorithm of § 4.3 and it is expected that a comparable speed-up would be achieved.

Apart from the matching algorithm the monocular processing is highly parallel. All the three images are totally independent and can thus easily be processed in parallel. Each image can also be split into independent regions for edge detection. However, the tracking operation can not be easily parallelised. The corner detection phase lends itself nicely to parallelisation as tracks can be dispatched to separate processors for corner detection as soon as the tracking has finished. The segment merging is a global process which once again needs all the data available in one place to be able to operate properly and consequently can't be parallelised easily.

It can be seen though that the monocular processing does offer some potential for parallelisation even if it is simply to process each image on a separate processor.

8.2. AUTOMATIC THRESHOLD SETTING

Throughout the system there are a great many thresholds that need to be set to enable effective operation of the modules. The value of these thresholds is dependent on the data that they are operating on and at the moment are set manually by trial and error. It is possible to make a rough guess of which value they should be set at by observing the input data and this rough guess is usually satisfactory for most cases. If a system is to be thought of as artificially intelligent, this is the sort of task which it must carry out for itself and it would be a nice feature if each module could carry out a simple analysis of the data provided and make the rough guess for itself.

The most common implementations of this behaviour use histograms to construct frequency functions of certain image parameters such as intensity. They then locate the various maxima and minima to avoid making a totally arbitrary choice of threshold. This was the basis of R. Ohlander's approach to choosing thresholds during the analysis of natural scenes.[Ohlander'75] During his study he encountered the inherent problem with this approach that values from regions of similar properties overlap to reduce the distinction of the maxima and minima. He proposed two ways to surmount this problem.

The first is a consequence of the observation "that quite often one or more parameters will be sensitive to data that appears uniform in other dimensions", and involves constructing up to nine histograms of different properties of the image. Thus if an overlap occurs in one of the frequency functions, information that is present in one or more of the other histograms will be able to be used to choose a value instead

The second way that he could see to overcome the problems of these overlaps is to remove regions of the image which are uniform for a given parameter once they have been located. This would eliminate the extracted points from further consideration and could result in features which were previously obscured becoming more distinct.

He also devised a set of conditions for selecting a threshold value at a peak. These are summarised in table 8.1 in order of priority.

Another method for automatic threshold selection has been devised by Beattie. [Beattie'85 pp74-106] He carried out a mathematical analysis of the utility of the output image as the threshold was varied. This utility measure is based on an information theoretic communication channel model on which the effects of noise can be examined.

He takes the difference of the information originally available, as measured by the source entropy, and the information lost in the channel, as measured by the conditional

Precedence	Conditions
1	Both minima ≤ 10% of maximum frequency value. At least one minimum separates another peak with a max/min ratio of at least 2:1.
2	Both minima ≤ 25% of maximum frequency value. At least one minimum separates another peak with a max/min ratio of at least 2:1.
3[1]	A local minimum divides two peaks both of which have a max/min ratio of at least 2:1. Maxima are within 10% of each other.

Table 8.1 - Conditions for selecting a peak from a histogram

[1] This case applies to histograms which are essentially bimodal. Both peaks are conditionally accepted.

entropy, as a measure of the information transmitted over the channel. He calls the measure the mutual information of the channel and defines it thus;

$$I(S_1,S_2) = H(S_1)-H(S_1 | S_2)$$

where S_1 is the source image, S_2 is the receiving image, $H(S_1)$ is the entropy of the source and $H(S_1 | S_2)$ is the conditional entropy of the source given the receiver.

What we need then is a threshold which maximises information transfer over a binary channel unconditionally. If this was applied to the edge detection process it would mean that we would find the threshold which, when applied to the noisy first differences, produces the edge map most similar to that which would be obtained from the noise-free first differences using the same threshold.

Beattie interprets this maximisation as follows. Beginning with a very high threshold value we have a small source entropy because few edges are generated and low conditional entropy because the high threshold produces few spurious edges. As the threshold is decreased, the source entropy increases as more edges are generated as does the conditional entropy because more spurious edges are generated. A threshold is selected when reducing it any further would increase the conditional entropy more than the source entropy. This means that the threshold obtained should produce the largest number of significant edges consistent with keeping the number of erroneous edges small.

The mutual information can be defined as a function of the threshold chosen if the noisy distribution of first differences is used as a noise-free distribution. This function also produces an output distribution which is the input distribution blurred by noise. This output can be used as another input distribution to obtain a second mutual information function and another blurred distribution. This process can be repeated as often as desired before being back extrapolated to obtain the best threshold for the original image.

Beattie presents the following algorithm for obtaining the best threshold for the edge detection process;

1) *Construct a first difference distribution for a given image.*
2) *For every threshold value in the range, compute the mutual information $I(S_1, S_2)$*
3) *Store the threshold value which gives $\max\{I(S_1, S_2)\}$*
4) *Convolve the first difference distribution with noise to generate the new first difference distribution.*
5) *Repeat steps 2)-4), n-1 more times.*
6) *Using the n values of the best threshold for each of the n convolutions, extrapolate back to get the best threshold for the original image, that image with no convolutions.*

Both Ohlander and Beattie's methods have been demonstrated to work in the systems that they developed as part of their respective Ph.D.s and there is no reason to believe that some or all of their ideas could not be utilised in the construction of an automatic threshold selection module for this project. The addition of this module would add a great deal of credibility to the system as well as making it more viable for applications in the real world where the requirement for someone to set all the thresholds would be a distinct disadvantage.

8.3. AUTOMATIC CALIBRATION

The calibration process described in chapter 6 is a laborious and time consuming one which needs to be carried out for each new set of test images. Because of the reliance on human judgement this process is extremely error-prone and requires a great deal of attention to small details. During the development of the system several sets of test images were taken and the cameras had to be calibrated each time. In order to solve the systems of equations, at least seven points had to be registered for each camera. Because the selection of these points is non-trivial, the calibration of each camera took an average of ninety minutes. The resultant parameters are sufficient for the task but their accuracy could easily have been improved by registering more points. The literature suggests at least 50 registrations are required for an accurate estimation.

This is obviously an unsatisfactory state of affairs, especially as there are several systems in existence which already automate this process.

That of Yakimovsky (1978) moves a robot arm in front of the camera taking pictures of the arm. A pattern recognition algorithm automatically finds a feature point of the arm in each picture. The arm is calibrated so that the world co-ordinates of each position are known. This gives the set of world points registered with their image points which can then be used to obtain the camera parameters. However, this is not a suitable method for the set-up we use as there is no robot arm available.

A better method is offered by Ayache and Faverjon who also automate this process in their system. A planar grid of precisely located orthogonal straight lines is placed in front of the two cameras in (at least two) different known positions. Images of these lines are extracted and their equations are accurately computed by a least-squares estimator and the intersection points of the grid are computed. As soon as enough different non-coplanar grid points have been registered between scene and image, the parameters can be determined.

Such a grid exists in the department and this method would limit the human intervention to just locating the grid at a few known positions and informing the calibration process of these positions.

If this process could be automated it would remove one of the most time consuming and error-prone parts of the system as well as improving the accuracy of the camera parameters obtained and consequently the results of the matching phase of the system.

8.4. INTEGRATION OF BINOCULAR INFORMATION

Yachida (1986) presents a major problem with trinocular stereo. The addition of the extra camera while making the correspondence problem determinate also increases the dead angles where 3D information can not be obtained. (Figure 8.1)

We can investigate the consequence of this dead angle in both binocular and trinocular set-up with reference to figure 8.2. Figure 8.2 (a) shows the set-up for the two cameras and A is the region which is the common visual field of both cameras. With three cameras (figure 8.2 (b)) we can see that the region of overlap between all three

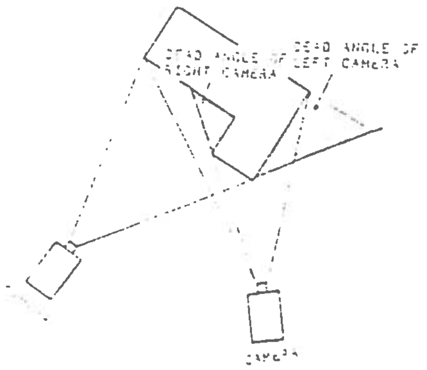


Figure 8.1 - Dead angles in the binocular set-up

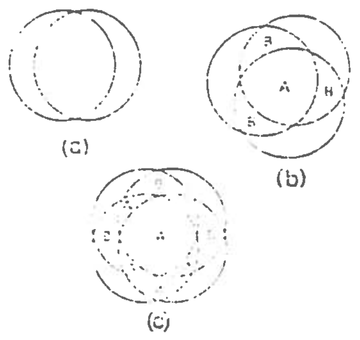


Figure 8.2 - Theoretical regions of overlap for two and three cameras

cameras, A, is less significant than it might have been hoped. But, if we could somehow utilise the regions of B, the amount of information would be a great deal better. The B regions can be thought of as a standard binocular set-up with some scene features present in only two out of the three cameras. Yachida suggests that the trinocular system can be extended by taking these binocular cases into account. He suggests that after the images have been processed by trinocular stereo and the 3D information of region A determined, that the binocular processing required for region B will be simplified because a great many of the potential correspondences will have been removed by the trinocular processing.

This proposal is intuitively pleasing and indeed there were several instances during the testing of the trinocular system where access to a binocular matcher would have provided more 3D information. Most of the code needed to implement a binocular matcher already exists within the trinocular matcher and it would not be a large task to integrate the binocular information provided by the B regions.

8.5. SURFACES AS FEATURES

In § 4.1 an assumption was made that the significant features in each image were to be edges, i.e. straight line segments. This made correspondence easier because there was always going to be less edges than edge points in an image. This theory can be extended to state that it would be better to use surfaces as features because there are always going to be less surfaces than edges. As well as reducing the correspondence problem still further the use of surfaces as primitive features in the system would overcome the restriction of only being able to recognise polyhedral objects. A surface patch can be bounded by edges of arbitrary shape, not only straight lines.

There have been various systems developed for locating regions in grey level data which could be used instead of the existing monocular processing to localise the features. The matching process which would take the regions which were found as input would obviously have to alter somewhat (i.e. there would be no concept of parallelity) but the underlying algorithm would remain the same using the third camera to help reduce the correspondence problem. Because more properties can be derived from a region than a line segment, which basically only has length and orientation, the local constraints implemented should prove more effective.

After the matching process is complete the triangulation process to obtain the coordinates of the boundaries of the regions in the world would proceed using the edges of the region in much the same way as at present. The process is described in 4.2.

This constitutes a major revision of the code already written but this avenue of development appears, at least at first sight, to be quite promising.

BIBLIOGRAPHY

N. Ayache and B. Faberjon, "Fast Stereo Matching of Edge Segments Using Prediction and Verification of Hypotheses," *Conference on CVPR*, pp. 662-664, IEEE, 1985.

N. Ayache and F. Lustman, "Fast and Reliable Passive Trinocular Stereovision," *Proceedings of ICCV*, pp. 422-427, IEEE, London, 1987.

N. Ayache and F. Lustman, "Trinocular Stereovision: Recent Results," *Proceedings of IJCAI 10*, vol. 2, p. 826, 1987.

N. Ayache and B. Faverjon, "Efficient Registration of Stereo Images by Matching Graph Descriptions of Edge Segments," *International Journal of Computer Vision*, pp. 107-131, Kluwer Academic Publishers, Boston, 1987.

Baker and Binford, "Depth from Edge and Intensity Based Stereo," *Proceedings of IJCAI 7*, pp. 631-636, 1981.

D. H. Ballard and C. M. Brown, *Computer Vision*, pp. 89,483-484, Prentice-Hall, London, 1982.

R. J. Beattie, "Edge detection for semantically based early visual processing," Ph.D Thesis, University of Edinburgh, 1984.

Bolles, Horaud, and Hannah, "3DPO : A Three-Dimensional Part Orientation System," *Proceedings of IJCAI 8*, pp. 1116-1120, 1983.

T. Brand and A. Sherlock, *Matrices - Pure & Applied*, pp. 9-17, Edward Arnold, London, 1970.

R. Cagenello, "A Binocular Stereo System Utilizing A Priori Scene Knowledge," M.Sc. Thesis, University of Edinburgh, 1985.

J. F. Canny, "Finding edges and lines in images," M.I.T. Laborotory Technical Report 720, 1983.

O. D. Faugeras and M. Hebert, "A 3D recognition and positioning algorithm using geometrical matching between primitive surfaces," *Proceedings of IJCAI 8*, vol. 2, pp. 997-999, 1983.

O. D. Faugeras and G. Toscani, "The Calibration Problem for Stereo," *Proceedings of CVPR*, pp. 15-20, IEEE, Florida, 1986.

R. Fisher, *AI3 - Vision lecture notes*, Dept. of A.I., Edinburgh, 1988.

A. Gerhard, H. Platzer, J. Steurer, and R. Lenz, "Depth Extraction by Stereo Triples and a Fast Correspondence Estimation Algorithm," *Proceedings of 8th ICPR*, pp. 512-515, Paris, 1986.

E. Gurewitz, I. Dinstein, and B. Sarusi, "More on the Benefit of a Third Eye for Machine Stereo Perception," *Proceedings of 8th ICPR*, pp. 966-968, Paris, 1986.

M. Ito and A. Ishii, "Three View Stereo Analysis," *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 8, pp. 524-531, IEEE, 1986.

S. C. Lennox and M. Chadwick, *Mathematics for Engineers & Applied Scientists*, pp. 164-165, Heinemann, London, 1977.

C. A. Malcolm, "The outline corner filter," D.A.I. Research Paper 212, University of Edinburgh, 1984.

D. Marr, *Vision*, p. 3, W. H. Freeman & Co., New York, 1982.

R. Nevatia, *Machine Perception*, pp. 1-2, Prentice-Hall, 1982.

R. Ohlander, "Analysis of natural scenes," Ph.D. Thesis, Carnegie-Mellon University, 1975.

Y. Ohta, M. Watanabe, and K. Ikeda, "Improving Depth Map by Right Angles Trinocular Stereo," *Proceedings of International Conference on Pattern Recognition (ICPR)*, pp. 519-521, IEEE, Paris, 1986.

M. J. L. Orr, "Camera Calibration for Stereopsis," M.Sc. Thesis, Heriot-Watt University, Edinburgh, 1984.

M. Pietikainen and D. Harwood, "Depth from Three Camera Stereo," *Proceedings of CVPR'86*, pp. 2-8, Miami Beach, 1986.

R. Watson, "An Edge-Based 3D Geometrical Model Matching System," M.Sc. Thesis, University of Edinburgh, 1985.

M. Yachida, "3-D Data Acquisition by Multiple Views," in *The third International Symposium on Robotics Research*, ed. O. D. Faugeras and Georges Giralt, pp. 11-18, MIT Press, Cambridge, Mass., 1986.

Y. Yakimovsky and R. Cunningham, "A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras," *Computer Graphics and Image Processing*, pp. 195-210, Academic Press, 1978.

APPENDIX A

THE PROJECTION ALGEBRA

The projection algebra is a series of re-write rules which operate on a sequence of overlap operations to reduce the sequence to its canonical form. The sequence $\alpha\beta$ notates the result in β of an overlap operation between a segment in α and a segment in β. E.g. 'LR' notates the section of the segment in the right hand image that is between the epipolar projections of the end points of the segment in the left hand image.

The rules below are given for the case of overlapping taking place between three images where α, β, γ and θ are all sequences of image identifiers.[1]

The left hand side of each rule can be matched against any sub-sequence of identifiers in the sequence being reduced and the right hand side gives the sequence that should replace it.

Single projection

$$\alpha\theta \rightarrow \theta, \ \textit{if}\ \theta = \alpha$$

$$\alpha\theta \rightarrow \alpha\theta, \ \textit{otherwise}$$

Double projection

where $\alpha\beta\theta$ is fully reduced with respect to single projections.

$$\alpha\beta\theta \rightarrow \beta\theta, \ \textit{if}\ \theta = \alpha$$

$$\alpha\beta\theta \rightarrow \alpha\beta\theta, \ \textit{otherwise}$$

Triple projection

where $\alpha\beta\gamma\theta$ is fully reduced with respect to both single and double projections.

$$\alpha\beta\gamma\theta \rightarrow \beta\gamma\theta$$

this is because;

$\theta \neq \gamma$ by single projection,
$\theta \neq \beta$ by double projection,
$\theta = \alpha$ because there are only three images.

[1] Different rule sets are available for schemes with differing numbers of images.

All sequences of overlaps for three images can be reduced to their canonical form using these rewrite rules. It can be seen that the canonical form of a sequence will never involve more than three images. This is intuitively obvious.

Equivalence of sequences

Two sequences are equivalent if

a) their canonical forms contain the same image identifiers

and

b) the last image identifier in each canonical form is the same.

E.g. The sequences 'LTRT' and 'TRLT' are equivalent because their canonical forms 'LRT' and 'RLT' are equivalent.

Identical sequences

Two sequences are identical if

a) their canonical forms contain the same image identifiers

and

b) the image identifiers appear in the same order in each canonical form.

E.g. The sequences 'LTRT' and 'TLRT' are identical because their canonical forms 'LRT' and 'LRT' are identical.

APPENDIX B

USER'S GUIDE

There are five main parts to using the system to recognise and locate objects;

1) Set up the scene.
2) Calibrate the cameras.
3) Carry out monocular processing.
4) Carry out trinocular processing.
5) Carry out geometric matching.

Set up the scene

This is the first task that needs to be carried out. It involves, placing the object in a place where good images will be obtained and determining where to place the cameras to obtain these good images. This is done by using the HIPS command *initf* to enable the view from each camera to be examined on the monitor as the scene is set up. The cameras should be oriented and the object placed so that;

a) the object appears large in each image,
b) not too many edges are occluded between the three images,
c) the definition of the edges on the object is maximised.

Paper of varying colour can be used to place the object on to improve the edge definition of its boundary. I.e. a white object on a white background will not have well defined edges.

Calibrate the cameras

Each camera must now be calibrated as described in chapter 6. This is quite an involved process and utilises M. Orr's calibration suite (~aicam/camcal). The paragraphs below indicate how to calibrate a single camera. This process must be repeated for each camera without the scene or camera positions being altered in between times.

1) the camera must be located in a steady position, all rods must be pushed firmly into their mounts and all screws must be secured to reduce the amount of camera movement.
2) a few extra objects should be added to the scene to increase the number of points available which are not co-linear. The calibration picture should now be taken using the HIPS program *rframe*.
3) next the points which are going to be used in the calibration process should be decided upon and measured as accurately as possible using the Vernier gauge from the lab to help

1) The file names in *~aicam/source/trinocular.c* should be changed so that they deal with the desired segmented data files and their corresponding parameter files.[1]

2) Any thresholds which are needed to be set should be altered in the file *~aicam/source/thresh.h.* The significance of the thresholds held in this file are described in various parts of the report.[1]

3) The program must be updated - *make trinocular*

4) The program must be run to produce the final results. *trinocular > out.res* will place the results of the program in *out.res.*

5) The program will then ask whether any particular edge is of interest. The edge may be specified or *-1* may be input to signify no particular interest. The program will then proceed to carry out the trinocular processing, reporting any matches found both before and after ambiguation with their corresponding co-ordinates.

6) *out.res.* may then be overlayed on the original segmented images using the program *~aicam/picdisp/gwin.*

Carry out geometric matching

The geometric matching should be carried out in the directory, *~aicam/matcher.*As containing a model of the object (obj.mdl) and the results from the trinocular processing should also have been placed in a file (res.data). If more detailed information on the operation of the matcher is needed, it is available on-line on neon or as a pamphlet in the A.I. dept.

1) The matcher should be invoked - *wire.*

2) The *m obj* command should be used to load up the model.

3) The *match -f res.data* command should be used to try to match the model to the data.

[1],[1] In the future these changes will be unnecessary as the parameters will be able to be altered via the user interface.