Reconfigurable Parallel Vision System Informal Specification RM/94/4

W.J. Austin and N.R. Scaife
Dept. of Computing and Electrical Engineering
Heriot-Watt University
Edinburgh EH14 4AS

May 27, 1996

```
Version history
Date       By   Comment
24-Mar-94  nrs  First version: HIPS,IMAGE,Canny,Hough,FIC,FEATURE
30-Mar-94  nrs  Created Latex version
12-Apr-94  nrs  Second version: HIPS,IMAGE,Canny,Locwinop,LHM,Hough,
                RegSeg,SCENE,FIC,FEATURE,MODEL,Match,POSE
18-Apr-94  nrs  Minor revision:LHM->EMLHM,-FIC,-FEATURE,+hwmod2
13-May-94  wja  More detail for SCENE, MODEL & MATCH
19-Jul-94  nrs  Added some algorithm details to RegSeg
```

Contents

Chapter 1

Introduction

This document is intended as the initial specification for the complete vision system envisaged in figure 1.1. The abstract data type (ADT) definitions included in this specification are intended, primarily, to aid the prototyping and instrumentation activities that will be carried out during the lifetime of the project. They could also be viewed as a means of organising the components of the parallel vision software that are implemented directly. It should be noted, however, that the ADTs in the specification and the data structures in the parallel system need not directly correspond. For instance, the 'local homogeneity measures' component in figure 1.1 need not exist as a distinct entity but could be implemented as an internal data structure within a process that converts raw images into region and surface maps. In this context, ADTs are 'virtual' structures in the final parallel system and their role in the development process will be a subject of the investigation into prototyping and instrumentation.

This document is an *informal* specification which would normally be used to construct a formal *algebraic* specification during system development. For the purposes of this project, however, it is not proposed to write such specifications but rather to use the **sml** prototypes *in lieu* of a formal specification. This is reasonable since the structure of an sml programme closely mimics that of an algebraic specification.

Abstract data types are used to represent the data input to and output from the system and to represent intermediate data between distinct phases of processing. The actual algorithms themselves, which transform one ADT into another are specified in sml structure format. This is purely a notational convenience as both data and code abstraction can be carried out with both ADTs and structures, both are simply collections of definitions. The difference is that in the **sml** language definition, abstypes are associated with a type name and a type specification to fit in with the type checking system whereas structures are the basis for separate compilation and can be given an algebraic signature to define what is visible outside the structure. In this way, abstypes represent higher order data and structures represent higher order code.

The ADT specifications are divided up into basic and non-basic generators and inquiry operators which define the interface between the ADT and its clients/users/callers[1]. Debug related functions (show operators) are also sometimes specified but the usual error definitions have not been included since the specification is in an early stage of revision. These may be added at a later date when work commences on the complete system.

Structure specifications are more informal due to the fact that the fundamental nature of the algorithms has

[1] The actual physical embodiment of the user of an ADT is difficult to define, *client* is a metaphor for ADTs used in the mode of a client-server relationship, useful for distributed systems, *user* relates more to the software engineering use of encapsulated data where the user of the ADT knows nothing about its internals and *caller* refers to a function which calls the methods in the ADT which is really another form of client-server relationship.

not been decided upon. Where possible, function definitions have been included but are likely to be rough approximations only. The structure specifications consist mostly of surveys of possible methods, indicating the advantages and disadvantages of each along with some indications of the practicalities of implementation.

1.1 System Considerations

1) Farming based on distribution of image segments incurs prohibitively high communication costs, this is better replaced by short control messages indicating regions of a pre-distributed image. Pre-distribution also fits will with the requirement that data from each level of processing is available to later processes which might require it. (This requirement allows a mixture of top-down and bottom-up processing.)

2) The depth processes described are based on the robust window fitting approach advocated by Surrey during Laird and by other vision researchers. The general approach used by Surrey should be modified to allow higher level processes in the parallel architecture to influence the use of robust operators.

3) To allow a combination of top-down and bottom-up processing in the system, the results of each processing stage should persist until the scene has been correctly classified. This allows top-down processing to re-activate low level and segmentation processes are required. This implies that large amounts of iconic data should remain in the processor network: storage of intermediate results on disk until they are needed is not a viable alternative. As well as the input image, homogeneity maps and surface maps, any intermediate stages must have memory available. Since it is not possible to dynamically create structures in occam, a region of memory should be set aside on each processing element for temporary processing requirements. Memory is also required for the model base, if a faceted representation is used, significant amounts of memory may be required.

4) The memory requirements outlined above suggest that it may be appropriate to partition the processor network into separate areas for scene manipulation and model matching. The scene processors would be responsible for low level processing and segmentation, the model processors for matching. A condensed symbolic representation of the scene features would be passed to the model processors at the end of the segmentation process. All iconic data would be retained in the scene partition for further processing as guided by the higher level processes. When matching suggests that further segmentation or surface fitting was required, the scene processors would act as servers to the high level processes.

5) The scene processing would also be an appropriate location to carry out ray trace based verification. A server based process for verification could be implemented. This server would need to consider the access requirements of the model partition: more than one model location may require verification leading to some interesting load balancing problems.

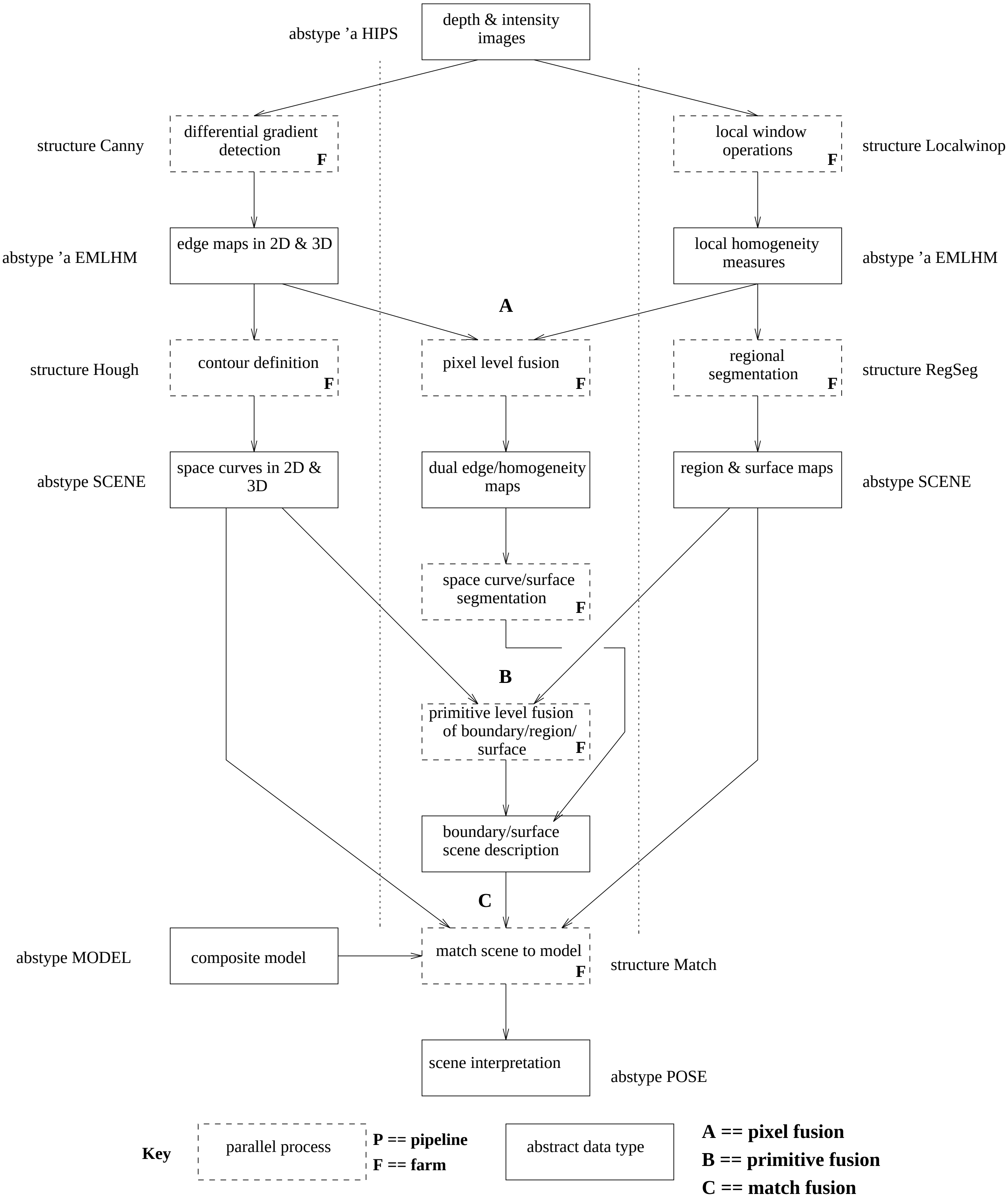

Figure 1.1: The proposed architecture of the parallel system

Chapter 2

abstype 'a HIPS

```
abstype 'a HIPS:
Informal Specification:v1.01

Version history
Date       By   Comment
12-Jan-94  nrs  First version created
19-Jan-94  nrs  created abstype 'a IMAGE to store images separately
24-Mar-94  nrs  minor modifications
30-Mar-94  nrs  converted to latex
```

2.1 Overview

The HIPS abstype is consists of a descriptive header and a sequence of 2D images (frames).

The header contains information such as the number of frames, the dimensions of the frames (which are all the same size), a sequence name and a sequence history and description.

The images are stored in an auxiliary abstype 'a IMAGE. This is to allow separate processing of the images, with the header information removed. It also allows the IMAGE abstype to be used for other methods where the header information has no meaning or is not required.

The HIPS abstype is a multi-purpose abstype with different interpretations in different contexts. A sequence consisting of a single image is used to represent a depth or intensity image, although there is no provision for distinguishing between these two types - the values in the image are purely numerical values. A binary sequence of two images is used to represent the edge strength and edge orientation of an image after processing by an edge detector such as a canny operator.

The HIPS abstype is polymorphic ('a), in keeping with the IMAGE abstype defined elsewhere. This allows intermediate results to be stored in files for visual inspection or external processing. The IMAGE abstype has no conceptual link with files, whereas the HIPS abstype does. Input images and the output from the canny detector will use int versions, convolution masks need real versions. There is also the possibility of using the HIPS abstype to store any 2D or 3D array of values, for instance hough space accumulators could be stored as a HIPS abstype.

At present only a small subset of the Hips definition is implemented, that relevant to the parallel vision system. Hips files are assumed to be eight bit byte format with no packing. A future version is planned to allow double format Hips images to be used.

2.2 Operators

2.2.1 Basic generators

```
create_hips: (fname:string) -> int HIPS
```

A HIPS abstype is required to be generated from a Hips format file. The fname parameter is the name of the file to be used in creating the abstype. The parameters for the images are read from the file header, these being image dimension and number of frames, the other entities in the file header are also stored in the HIPS header.

```
set_hips: (hdr:HIPS_HEADER) (frames:'a IMAGE list) -> 'a HIPS
```

This function constructs a new HIPS abstype from separate header and frame information and is intended to be used to reconstruct the abstype after hips_get_header and hips_get_frame have been used.

```
blank_hips:
  (name:string) (nframes:int) (nrows:int) (ncols:int) (init:'a)
  -> 'a HIPS
```

The arguments to this are the dimensions of the image and the number of frames, all the values in the image are set to the value init. The name parameter is the 'original name' field in the Hips header. The main purpose of this generator is to generate test images for debug purposes.

```
random_hips: (name:string) (nframes:int) (norows:int) (nocols:int) -> int HIPS
```

This is also a test routine returning a HIPS abstype with random values for the image data.

```
image_to_hips: (name:string) (img:'a IMAGE list) -> 'a HIPS
```

A HIPS abstype can also be created by providing a copy of the image data to the 'image_to_hips' function. The number and dimensions of the frames are deduced from the img IMAGEs.

2.2.2 Non-basic generators

```
hips_add_history:
  (hips:'a HIPS) (history:string list) (description:string list)
  -> 'a HIPS
```

This function merely adds new entries to the history and description lists.

```
hips_set_value: (hips:'a HIPS) (frameno:int) (x:int) (y:int) (new:'a)
  -> 'a HIPS
```

Returns a new HIPS abstype with frame frameno value at (x,y) set to new.

```
hips_map: (image:'a HIPS) (func:'a -> 'b) -> 'b HIPS
```

This function applies the func function to each element in an image, returning an image generated from the output from the function. This operation is not generally useful in vision algorithms but can be used to perform simple operations such as thresholding on images under debug conditions.

```
hips_map2:
  (image1:'a HIPS) (image2:'b HIPS) (func:'a -> 'b -> 'c)
  -> 'c HIPS
```

This function maps the func function over two images returning an image generated using the output from func. Again, this is not a generally useful operation but has been provided for completeness.

```
hips_map2ixy:
  (image1:'a HIPS) (image2:'b HIPS) (xstart:int) (ystart:int)
  (func:'a -> 'b -> int -> int -> 'c)
  -> 'c HIPS
```

This function performs the same operation as map2 but the function is passed the (x,y) index of the elements passed in. This function is vital in performing operations where the (x,y) values are required by the calculation, notably the transformation phase of the hough transform.

```
hips_window:
  (image:'a HIPS) (window_x:int) (window_y:int) (func:'a HIPS -> 'c)
  -> 'c HIPS
```

This function maps a window of size window_x by window_y over the image parameter and applies func to the windows in sequence, building a new HIPS image out of the function's results. This function is the basis for the convolution operation used extensively during the Canny edge detector and will also be the basis for many functions in the local window operations phase of depth processing.

The function `func` will, in most instances, have a high degree of functionality and will additionally carry substantial parameter information using partial application. For instance, for the Canny edge detector a 2D gaussian filter of dimensions (`window_x`,`window_y`) is built into the `func` function by partial application.

Note that although this is not an explicit requirement of the above functional specification the size of the image in the output abstype should be the same as the size of the image in the input abstype. How this is achieved is open to debate.

2.2.3 Inquiry operations

```
hips_get_header: (hips:'a HIPS) -> HIPS_HEADER
```

This function returns the Hips header internal representation, HIPS_HEADER is an sml record type containing the header information.

```
hips_get_frames: (hips:'a HIPS) -> 'a IMAGE list
```

This returns the list of image frames as a list of IMAGE abstypes. The main use of this function is to allow the bare image to be stripped out so that processing routines can recurse over the image without passing the header information along each time.

```
hips_get_rows: (hips:'a HIPS) -> int
```

Returns the number of rows in an image (height of the image).

```
hips_get_cols: (hips:'a HIPS) -> int
```

Returns the number of columns in an image (width of image).

```
hips_get_value: (hips:'a HIPS) (frameno:int) (x:int) (y:int) -> 'a
```

Returns the value of frame frameno at (x,y).

```
hips_foldl:
  (func:'b -> 'a -> 'b) (default:'b) (image:'a HIPS)
  -> 'b
```

This function reduces an image down into a single value. The main use of hips_foldl is in calculating statistical summaries of images and would be required if hough space were modelled using the HIPS abstype. There is also a foldr version but so far, all uses of fold on images have been associative and commutative, under which circumstances the two are equivalent.

2.2.4 Show functions

```
print_hips_header: (hdr:HIPS_HEADER) -> unit
```

Used in conjunction with hips_get_header allows the header information in a HIPS abstype to be printed on the screen. This function is also called by create_hips to verify the contents of a file when a HIPS abstype is created from a Hips file.

```
commit_hips: (hips:int HIPS) (fname:string) -> unit
```

This allows a HIPS abstype to be stored in a Hips file fname. In the current version, only **int HIPS** abstypes can be saved since only byte format Hips files are supported.

2.3 Notes

1) The **HIPS_HEADER** type is an sml record defined as follows, this closely follows the definition of the header in Hips files:-

```
type HIPS_HEADER = {
  orig_name      : string,
  seq_name       : string,
```

```
    num_frame      : int,
    orig_date      : string,
    rows           : int,
    cols           : int,
    bits_per_pixel : int,
    bit_packing    : int,
    pixel_format   : int,
    seq_history    : string list,
    seq_desc       : string list
  };
```

2) Although functions such as **map** and **window** are defined for HIPS abstypes, these are, in general, calls to the equivalent functions in the IMAGE abstype.

Chapter 3

abstype 'a IMAGE

```
abstype \'a IMAGE:
Informal Specification:v1.00

Version history
Date       By   Comment
19-Jan-94  nrs  First version created from abstype 'a HIPS
```

3.1 Overview

The abstype IMAGE is intended as a representation of a 2D array of values. These could be images as int or real values, arrays of accumulators or filters for convolution operations.

Since this type of data is quite low level and is likely to be used at several different points in the system, a wide variety of different access methods are required. The three principal types of access are **map** operations, in which the image is scanned in a particular way and transformed into another IMAGE abstype, **window** operations which are similar to map except that 2D sections of the image are scanned rather than individual values, and **fold** operations where an image is reduced to a single value, such as a statistical summary.

The internal representation for this abstype could be either a `'a list list` representation or use the NJSML defined **array2** structure. Initially, the list representation will be used since this is referentially transparent and combines well with recursive routines. If this proves to be too slow then an array implementation could be written although arrays are not referentially transparent as currently implemented.

3.2 Operators

3.2.1 Basic generators

```
create_image: (norows:int) (nocols:int) (default:'a) -> 'a IMAGE
```

This function generates a new IMAGE abstype of size nocols by norows with all the initial values set to default.

```
list_to_image: (data:'a list list) -> 'a IMAGE
```

This creates an IMAGE abstype from a list of data values.

```
random_image: (norows:int) (nocols:int) -> int IMAGE
```

This is a test routine returning an IMAGE abstype with random values for the image data.

3.2.2 Non-basic generators

```
image_set_value: (image:'a IMAGE) (col:int) (row:int) (new:'a) -> 'a IMAGE
```

Returns a new IMAGE abstype with value at (col,row) set to new.

```
image_map: (image:'a IMAGE) (func:'a -> 'b) -> 'b IMAGE
```

This function applies the func function to each element in an IMAGE abstype, returning an image generated from the output from the function.

```
image_map2:
  (image1:'a IMAGE) (image2:'b IMAGE) (func:'a -> 'b -> 'c)
  -> 'c IMAGE
```

This function maps the func function over two images returning an image generated using the output from func.

```
image_map2ixy:
  (image1:'a IMAGE) (image2:'b IMAGE) (xstart:int) (ystart:int)
  (func:'a -> 'b -> int -> int -> 'c)
  -> 'c IMAGE
```

This function performs the same operation as image_map2 but the function is passed the (x,y) index of the elements passed in.

```
image_window:
  (image:'a IMAGE) (window_x:int) (window_y:int) (func:'a IMAGE -> 'c)
  -> 'c IMAGE
```

This function maps a window of size window_x by window_y over image and applies func to the windows in sequence, building a new IMAGE image out of the function's results. Edge effects are taken into account so that the size of the resultant image is the same size as the starting image.

3.2.3 Inquiry operations

```
image_to_list: (image:'a IMAGE) -> 'a list list
```

This is the reverse of the list_to_image function and allows the data to be exported in generally recognised form.

```
image_get_rows: (image:'a IMAGE) -> int
```

Returns the number of rows in an image (height of the image).

```
image_get_cols: (image:'a IMAGE) -> int
```

Returns the number of columns in an image (width of image).

```
image_get_value: (image:'a IMAGE) (x:int) (y:int) -> 'a
```

Returns the value of the image at (x,y).

```
image_foldl: (func:'b -> 'a -> 'b) (default:'b) (image:'a IMAGE) -> 'b
```

This function reduces an image down into a single value. The main use of foldl is in calculating statistical summaries of images.

```
image_foldr: (func:'a -> 'b -> 'b) (default:'b) (image:'a IMAGE) -> 'b
```

This function reduces an image down into a single value. This is similar to foldl except the evaluation order is different.

3.2.4 Show functions

```
image_show_window:
  (image:'a IMAGE) (xmin:int) (ymin:int) (xmax:int) (ymax:int)
  -> unit
```

This function displays a limited rectangular portion of an image on the screen.

Chapter 4

structure Canny

```
structure Canny:
Informal Specification:v1.01

Version history
Date       By   Comment
24-Jan-94  nrs  First version created
24-Mar-94  nrs  Minor revision
```

4.1 Overview

The Canny edge detector is well-studied method [15] which combines several image processing functions into a single operation. The method as a whole was chosen to satisfy criteria for an ideal edge detector proposed by Canny.

The first phase of the Canny method is a smoothing operation designed to reduce the effect of noise on the edge detection phase. This is a low-pass filter in which the image is convolved with a Gaussian filter, the degree of smoothing being controlled by the σ in the Gaussian distribution.

The second phase is the edge enhancement phase. This is accomplished by convolving the image with the derivative of the Gaussian distribution, which acts as a high-pass filter and is equivalent to performing the Laplacian of the Gaussian smoothed image (the two operations can be combined as the overall effect is achieved by convolving with the ∇^2 of the Gaussian).

In practice, since this method forms a 'zero-crossing' at edge positions, orientation information can be extracted. The image is convolved with two directional filters, the angular information being extracted from the combination of the two edge directions.

The third phase is an edge thinning algorithm, based on non-maximal suppression. The gradient of the edge strength is computed in both directions normal to the edge and if the value of the gradient is such that the line position itself is not a maximum in this direction then the edge strength is set to zero. The gradients are calculated by a simple linear interpolation within a nine pixel neighbourhood.

Finally, the resulting edge strength image is thresholded using a hysteresis method. All points in the image are retained if they are above an upper threshold. In addition, all neighbouring points in a nine pixel neighbourhood are tested and if they are above a lower threshold and their orientation is within an angular threshold of the central point then they are also retained and their neighbours checked in a recursive manner.

4.2 Operators

The overall functionality of the operator is provided by the high-level call:-

```
canny:
  (image:int HIPS) (sigma:real)
  (low:real) (high:real) (angular:real)
  -> (int HIPS * int HIPS)
```

The given image is treated using the Canny method, the σ for the Gaussian filters (smoothing and enhancement) is given by `sigma`. The low, high and angular values are thresholds for the hysteresis thresholding phase. The output is in the form of two HIPS abstypes, one for edge strength, the other for edge orientation.

The function is specified here as int HIPS but future versions could use char, int or real HIPS representations. The intermediate HIPS abstypes could also be real, giving slightly greater accuracy at the expense of storage requirements. In this case, if images are required for display purposes the `real HIPS` image could be scaled and converted into an `int HIPS` using `hips_map` with a suitable conversion function.

4.3 Canny Phases

The individual phases of the Canny operator are encapsulated in the following functions:-

```
smooth: (image:int HIPS) (sigma:real) -> (int HIPS * int HIPS)
```

pre: The raw image is supplied with a σ value for the gaussians.

post: Two smoothed images are returned, one smoothed using a 1D filter in the x direction and the other smoothed similarly in the y direction.

Firstly a 1D filter is generated from the following function:-

$$G(n_i) \;=\; \frac{1}{2\pi\sigma^2}\int_{n_i-0.5}^{n_i+0.5} e^{-\frac{x^2}{2\sigma^2}}\,dx$$

where,

$$n_i \;=\; \{-n_{max}, -n_{max}+1, ..., n_{max}\}$$

and noting,

$$G(n_i) \;=\; G(n_{-i})$$

Note that Simpson's approximation is used to evaluate the integral,

$$\int_a^b f(x)dx \;\approx\; \frac{b-a}{6}\left[f(a) \;+\; 4f(\frac{a+b}{2}) \;+\; f(b)\right]$$

The image is then convolved separately in x and y directions using the 1D Gaussian filter,

$$smooth\_x(x,y) \;=\; sum_{k=-n_{max}}^{n_{max}} G(k)\; image(x+k,y)$$

and,

$$smooth\_y(x, y) \;=\; sum_{k=-n_{max}}^{n_{max}} G(k)\; image(x, y+k)$$

```
edge_detect: (smooth_x:int HIPS) (smooth_y:int HIPS) (sigma:real)
  -> (int HIPS * int HIPS)
```

pre: The image smoothed in x and y directions is supplied as a pair of HIPS abstypes along with the sigma for the Gaussian.

post: Edge enhanced images with components in the x and y directions are returned as a pair of HIPS abstypes.

Firstly, the derivative of the Gaussian distribution is formed appproximately,

$$G'(n_i) \;\approx\; \frac{1}{2\pi\sigma^2}\left[e^{-\frac{(n_i+0.5)^2}{2\sigma^2}} e^{-\frac{(n_i-0.5)^2}{2\sigma^2}}\right]$$

This can be achieved simultaneously with the calculation of the 1D Gaussian filter since the exponentials are simply the $f(a)$ and $f(b)$ terms in the Simpson's approximation equation. Note that in this case, however, $G(n_i) \;=\; -\,G(n_{-i})$.

The smoothed images are then convolved with $G'(n_i)$ to produce edge strength components in the x and y directions, note that the x edges are calculated from the image smoothed in the y direction so the net effect is smoothing using a 2D gaussian filter,

$$x\_edge(x, y) \;=\; sum_{k=-n_{max}}^{n_{max}} G'(k)\; smooth\_y(x+k, y)$$

and,

$$y\_edge(x, y) \;=\; sum_{k=-n_{max}}^{n_{max}} G'(k)\; smooth\_x(x, y+k)$$

```
non_max_suppression:
  (x_edge:int HIPS) (y_edge:int HIPS)
  -> (int HIPS * int HIPS)
```

pre: The edge strength components in the x and y directions are supplied as a pair of HIPS abstypes along with the scaling factor to be used on the output.

post: Non-maximal points will have been set to zero and the two components combined into an edge strength HIPS abstype and an edge orientation HIPS abstype.

At each point in the image, the gradient of the edge strength is calculated in both directions normal to the line. If the gradients are such that if either of the (hypothetical) neighbouring points in the directions normal to the line would have a greater strength than the central point then the point is rejected and set to zero.

The edge strength of neighbouring points is determined by linear interpolation. For example, if the line orientation at the centre of a nine pixel neighbourhood is such that the line is at approximately $150 degree$ to the horizontal (where the magnitude of the x component is greater than the magnitude of the y component) then the edge strength of the conceptual neighbouring pixel in the NNE direction can be estimated as,

$$\frac{x\_edge(x, y)}{y\_edge(x, y)} hypot(x+1, y+1) \;+\; \left(1 \;-\; \frac{x\_edge(x, y)}{y\_edge(x, y)}\right) hypot(x, y+1)$$

and in the SSW direction by,

$$\frac{x\_edge(x, y)}{y\_edge(x, y)} hypot(x-1, y-1) \;+\; \left(1 \;-\; \frac{x\_edge(x, y)}{y\_edge(x, y)}\right) hypot(x, y-1)$$

where,

$$hypot(x, y) \;=\; \sqrt{(x\_edge(x, y)^2 \;+\; y\_edge(x, y)^2)}$$

Similar equations exist in each of the eight possible directions. There are degenerate cases where:-

$$x\_edge(x, y) \;=\; 0 \quad and \quad y\_edge(x, y) \;=\; 0$$

For all edges that are retained, the edge strength is generated from the hypotenuse and the orientation in the range $-\pi$ to $+\pi$ from the arctangent:-

$$edge\_strength(x, y) \;=\; hypot(x, y)$$

$$edge\_orientation(x, y) \;=\; \tan^{-1}\frac{y\_edge(x, y)}{x\_edge(x, y)}$$

```
hys_thresh:
  (strength:int HIPS) (orientation:int HIPS)
  (low:real) (high:real) (angular:real)
  -> (int HIPS * int HIPS)
```

pre: The edge strength and edge orientation are input as two HIPS abstypes.

post: The thresholded edge strength and orientation data is output as two HIPS abstypes.

Hysteresis thresholding can be implemented in a simple recursive routine. The image is scanned and all points above the upper threshold are marked as having been examined and are retained. Then all eight neighbouring points are examined and any that are above the low threshold are retained if their orientation lies within the angular threshold of the central point's orientation. Note that parallel lines should already have been rejected by the non-maximal suppression phase.

```
for each point strength(x,y) in the input image
  if strength(x,y) >= high
    then track_line(x,y)
    else output_strength(x,y) = 0

track_line(x,y)
  if (x,y) has not been analysed
    then
      mark (x,y) as having been analysed
      output_strength(x,y) = strength(x,y)
      for each neighbour at (x + j,y + k), -1 <= j,k <= 1
        if strength(x,y) >= low and difference between orientation(x,y)
        and orientation(x + j,y + k) < angular threshold
          then
            track_line(x + j,y + k)
```

4.4 Control variables

The output from a Canny process could be qualified by the following properties:-

1) The total number of detected edges in the output. Sparsely populated images are quicker to process but do not contain as much information as more dense images. Very sparse or very dense images should be easy to detect but the quality of the information is difficult to quantify.

2) The number of detected edges which correctly correspond to a feature in the image in terms of position (the detected edge should be in the same place as the image edge). While this is not directly measurable, an approximation might be to estimate the amount of unconnected edge points during the thresholding phase, subtracting this from the total number of edge points might give a very rough estimate of the amount of 'feature' information in the image.

3) The number of detected edges that do not correspond to features in the actual image. These are effectively noise points and could be correlated (eg. digitisation effects) or uncorrelated (eg. sensor noise). Image smoothing can minimise some types of noise at the expense of reducing the sharpness of the features in the image but other types of noise cannot be treated in this way.

4) The accuracy of the angular information. This is subject to a number of errors such as digitisation effects and the method of combining the directional filters (other schemes are possible than the one specified in this document).

5) The statistical distribution of edge strengths throughout the output. This is important because later stages of processing may apply thresholds or some other selection criteria which would cause features to be rejected or accepted depending on this distribution.

As currently specified there are five control parameters. The effect of these parameters on the detected edges that are produced by the Canny detector could be:-

1) The σ parameter (**sigma**) for the Gaussian filters affects the localisation of the edges. If a low σ is used then the edges will be more localised, high σ causes greater blurring of the image at the smoothing stages. The contours of lines detected will thus become warped by neighbouring features as information from farther points becomes accumulated at the central point. Where objects in the image are isolated, this effect should be less noticeable than if there were, for instance occluded objects or highly textured backgrounds.

   Against this, however, it may be desirable to use higher values of σ to bring in hazier or less well defined edges, for instance from a poorly focussed image to provide more (although less accurate) information for later processing.

   Another effect of increasing σ is to reduce the effect of noise on the detector during the smoothing and edge enhancement phases although there is a trade-off with the accuracy of the detected edges. Noisy images may require high values of σ to provide any information at all.

2) The **low**, **high** and **angular** thresholds are mainly used to exclude redundant information from the input image. This might be, for instance, to exclude lines from specular or shadow edges from an image where they are, hopefully, less pronounced. Noise can also be classed in this way although the smoothing stages are more specifically present for this purpose.

   The effect of altering these three values independently is more difficult to predict. Altering the high threshold will 'globally' reduce or increase the amount of output data, altering the low threshold could be seen as affecting the 'connectivity' of the output data and the angular threshold affects the 'continuity' of the output data.

A simple set of measures defining the performance of the Canny process could be:-

1) The percentage of detected edges in the output (sparseness/density).
2) A histogram of the edge strength values in the output (distribution of edge strengths).
3) The number of connected and unconnected edge points (noise estimate).
4) Total time required to process the image (efficiency).

4.5 Notes

1) The points made about the control variables have all been devised by purely intuitive reasoning and as such are only vague suggestions. Before the control system itself is devised a more mathematical analysis of the relationship between the inputs to and outputs from the algorithms should be performed.

Chapter 5

structure Locwinop

```
structure Locwinop:
Informal Specification:v1.00

Version history
Date       By   Comment
06-Apr-94  nrs  First version created
```

5.1 Overview

This section defines the low-level operations applied to raw depth images although the techniques used here are very similar to methods used in the intensity route. Input is in the form of **HIPS** images and the standard methods supplied in the HIPS abstype specification can be used as local window operator higher order functions. The most useful one is the `hips_window` operator also used in the **Canny** edge detector although in this instance the operations performed on windows will be much more complex.

5.2 Local Window Operators

Local window operators are widely used in low-level processing of images. There are two main classes of local window operators:

1. *convolution operators* in which some mask is convolved with an image window to create a new set of values; and
2. *fitting operators* in which the data within an image window is fitted to some model, possibly using a least squares estimator or a robust equivalent.

Both operators can be applied to the image by the same piece of code, this requires the following inputs.

- The image to be operated on.
- The dimensions of the window.

- A window operator function.
- An empty image to be populated with the results.

The window dimensions are used to set up indices which scan the window over the image data. They also allow the border of the output window to be set to some null data value to reflect the fact that the operator cannot be applied at the edge of the image. The size of the null region is determined from the half-size of the window.

The window operator function encodes the convolution or fitting operation. For convolution operators, a generic function taking the convolution mask as input could be designed. For fitting operators, a function specifically designed for the task is required. In SML it is easy to pass the window operator function into the window application code as a parameter. In occam this is not the case and a code skeleton must be specialised to provide the required operator. Where multiple window based operations are possible, some switching function will be required.

For more complicated fitting strategies, e.g. the HaRASS program from Surrey [41], a more complex application function is required. In HaRASS three different window operations are possible, least squares fitting followed by two robust fitting stages where the residual errors are high. The initial least squares stage could be carried out by the generic application code, but the later stages requires a more complicated application strategy which takes the results of earlier processing into account.

In parallel processing, the application of window operators may be split among the available processing resources. The application process thus requires an additional input parameter to determine how the processing should be distributed. This can simply be the (x, y) dimensions indicating how much of the image should be examined by each processor. The block size can be determined at run time from the known processing resources and image size, or can be fixed according to the algorithm and architecture in use.

For processor farming, the image may be tiled, i.e. split into rectangular block of data in both the x and y directions. These tiles are then passed to the worker processes as they become available. To reduce communications overheads, the tile size should be considerably larger than the size of the window being applied. Tiling can also be used in geometric decomposition, a suitable alternative to process farming for low-level vision. In geometric decomposition, each tile is assigned to a unique processor and all tiles are examined in parallel. To avoid boundary effects, overlapping tiles should be applied. This increases the computational load on each processor, but this is generally preferable to the alternative of communicating boundary information between processors.

As an alternative to tiling, the image can be split in the y direction only, i.e. a certain number of complete rows are passed to the worker processes. This has the advantage that the data partitions are easier to work out (though the processing time gained is probably negligible when the image is pre-distributed). It is also easier to re-partition a 1D data partition, since this is just a case of moving data rows between processors. Re-partitioning can allow load balance to be maintained across a series of low-level processes on the same processor group. A discussion of re-partitioning issues in low level vision can be found in [16], though the constants described in that paper are little more than heuristic tweaks and are not based on any firm theory. Whether a 2D (tiling) or 1D (striping) approach is used, an additional input parameter is required for parallel implementation of window operators: this is a list indicating data decomposition and can be calculated at run time from the know processing resources.

Output from the window operator application process consists of an image (or images) representing the results of the window application process and some from of metric information which can be used to inform the next processing stage. For example, an edge detection process might output an image containing edge points with the number of edge points as a metric. For parallel processing, each worker process would produce a sub-image and metric for the region of the image it was considering. The metric can then be used to guide further processing of the image on the worker processes, in this case the output image would remain in the worker network. To continue the edge detection example, the next stage could be line detection by Hough Transform. Since the HT considers only edge points, the data in the worker network should be re-partitioned to ensure that each Hough worker has roughly the same number of edge points in its new image partition.

For more complex window operators, e.g. surface fitting, the metrics provided will be more complex and will include things like fitting errors and percentage of bad data as well as measures of the features detected.

5.3 Operators

Initially, the two basic types of window operator can be supported by a single high-order function, the functionality of the operator being supplied in the mapping function. If more convoluted calculations are required then specific versions of this high-order function can be constructed:-

```
local_window_op:
  (image:'a HIPS) (window_x:int) (window_y:int) (func:'a HIPS -> 'b)
  -> 'b HIPS
```

It is likely that more than one window operation will be applied to an image. All the operations can, initially, be catered for by this function and a single top-level function created by composing several of these operators together (this is possible since the output is type-compatible with the input).

The output could be in the form of several processed images, for instance H-K measures, but this is catered for by the polymorphism of the HIPS abstype. The HIPS equivalent of the `unzip` list processing routine could be defined if individual images are required, for instance to allow output of intermediate images for debug purposes.

5.4 Control variables

Little can be said about controlling these routines until they assume a more concrete form in this specification.

Chapter 6

abstype 'a EMLHM

```
abstype 'a EMLHM: (Edge Maps/Local Homogeneity Measures)
Informal Specification:v1.00

Version history
Date       By   Comment
18-Apr-94  nrs  First version created
```

6.1 Overview

This abstype provides storage for results from local window operations, including the Canny edge detector. The format of the data will be in the form of one or more 2D arrays of values (which could themselves be complex data structures) which can be stored using the `'a IMAGE` abstype. This abstype can itself be easily converted into a `'a HIPS` abstype for file storage.

Initially, this abstype will simply mirror all the `'a IMAGE` methods but may be enhanced if more complex operations are required. Due to the potential size of intermediate data, it is possible that some form of hashing will be required. This is particularly so of local homogeneity measures in the depth route where a large number of identical (possibly large) values are stored for each disinct region.

A simple form of compression has been suggested whereby a separate table of values is stored corresponding to the local homogeneity measures for particular patches and the image is then represented as an abstype of pointer values into the table (say, `pointer IMAGE`). The table of measures will itself be implemented as an auxiliary abstype which will then be bound to the IMAGE abstype giving a single EMLHM abstype, for instance:-

```
type pointer = int
abstype 'a EMLHM = pointer IMAGE * (pointer,'a) POINTER_TABLE
with
  ...
end
```

Where the abstype `('a,'b) POINTER_TABLE` defines a table of `'b` values indexed by a value of type `'a`. This obviously places an extra level of processing between the client routines and the abstype internal representation. This could be implemented as a set of *decompression* routines operating on partial data or could be implemented as a layer of higher order functions (within the EMLHM abstype) with built-in decompression, shadowing the IMAGE routines. This maintains the purity of the EMLHM abstype unlike *in situ* decompression of the data by client routines (even although this is likely to be more efficient and would be used in the parallel version).

6.2 Edge maps

Edge maps are 2D arrays of basic numerical values (`int` or `real`) and present no difficulty to the `'a EMLHM` abstype. The only possible complication could arise if feedback from the segmentation phase to the edge detector were implemented in which case more complex values might be used, for instance an image of values generated by the segmentation algorithm. These would still be stored in the `'a EMLHM` abstype, however.

6.3 Local Homogeneity Measures

Local homogeneity measures provide a means of classifying pixels during regional segmentation by pixel grouping or region growing. They can also be accumulated by a Hough techniques for feature detection. Homogeneity measures can be calculated by the application of window operators. Examples of homogeneity measures include: partial derivatives, surface normals, curvature measures, and surface parameters. Before the measures can be calculated, it may be necessary to smooth the input depth image to reduce the effect of noise. Parameter calculation and smoothing can be performed simultaneously by least squares fitting methods however, these assume a single data distribution and are sensitive to added noise.

While smoothing of the input image can improve parameter estimates for the majority of image points, over-smoothing of the data at depth discontinuities is a problem. Over-smoothing occurs because a fitting window placed over a boundary pixel contains more than one surface, leading to inaccurate parameter estimation. Segmentation of images breaks down at sites of over-smoothing: this affects pixels within a half window width of the discontinuity.

To improve parameter fitting (and therefore segmentation) close to parameter boundaries and in the presence of other outlier data, robust parameter fitting methods have been proposed, [51, 52]. These methods are computationally expensive however, and hybrid techniques using least squares *and* robust methods were proposed during the Laird project [40, 41], these are discussed in section 8.2.

Local homogeneity measures have also been used to guide Hough transformation methods. In [4], planar parameter estimates based on [67] were accumulated using an Adaptive hough Transform (AHT) [38].

6.4 Operators

Initially the operators can be considered identical to the operators in the IMAGE abstype but with the `image_` prefix or suffix replaced by `emlhm_`, for instance:-

```
create_emlhm: (fname:string) -> 'a EMLHM
```

To create an EMLHM abstype from a file or:-

```
emlhm_window:
```

```
(image:'a EMLHM) (window_x:int) (window_y:int) (func:'a EMLHM -> 'b)
-> 'b EMLHM
```

as an equivalent window operator.

Chapter 7

structure Hough

```
structure Hough:
Informal Specification:v1.01

Version history
Date       By   Comment
08-Feb-94  nrs  First version created
24-Mar-94  nrs  minor modifications
```

7.1 Overview

The Hough transform is a general technique for solving sets of underdetermined equations where the number of solution classes is unknown. The solutions are arrived at by forming clusters of potential solutions in the solution space and attributing dense clusters of possible solutions to the most likely solutions.

This technique has widespread uses in vision related tasks, for instance, curve detection in intensity and depth images and model matching using non-parametric representations of models.

An example of one of these tasks is locating 2D lines in an intensity image. In this case the system of equations is the normal form for the equation of a straight line:-

$$\rho \ = \ x \ \cos\theta \ + \ y \ \sin\theta \tag{7.1}$$

The potential solutions are plotted in solution space (Hough space), which in this case consists of (ρ, θ) space. For each point in the image (x, y), a sinusoid is plotted in Hough space and points through which large numbers of these sinusoids pass represent the likely (ρ, θ) value of straight lines in the image. Practically, this can be achieved by quantising (ρ, θ) space using a suitable range of ρ and θ values and using a voting system to determine the most likely values of ρ and θ.

Formulated in this way, the Hough transform technique is a partially robust method with good resistance to noise and occlusion. This is because the method uses global information concentrated at single possible solutions. It is also a very general method, being applicable to any curves for which parametric forms are available and can even be generalised (at the expense of computational complexity) to any representable object.

There are, however, a range of problems and limitations associated with the technique. The main limitation is the amount of memory and calculation required. For a system of equations with n variables, each quantised to l levels, the size of the accumulator array is l^n values. For instance, for an ellipse with 5 parameters, if each parameter is accumulated to 200 levels, each of which is a four byte real number, the amount of memory required is 1250 MBytes.

Another major problem is finding the clusters (or peaks when accumulators are used) in Hough space. Quite often, in Hough space the peaks themselves have a definite structure, for instance the *four-leaf clover* patterns produced by an ellipse detector [20] and the *butterfly* shapes produced by some line detectors [45] although special filters can be developed for the efficient detection of peaks if their structure is known. It is also possible for optical effects such as shadows to result in multiple peaks where a single object is more desirable. Neighbouring peaks tend to distort one another and are difficult to separate in any case. A consequence of this is that relatively sophisticated statistical methods are required to ensure robust detection of peaks in Hough space, rejection of false peaks and separation of neighbouring peaks, for example [21].

Another issue is the quantisation of Hough space. Coarsely quantised Hough space results in poor accuracy of the resulting features and a higher chance of neighbouring peaks being taken as a single peak. Finely quantised Hough space gives greater accuracy in feature determination but exacerbates the problem of quantisation errors [71].

If angular data is available, as produced by some types of edge detectors, then this can be used to reduce the discretisation errors.

Finally, there is a problem associated with geometric primitives that degenerate to one another under certain conditions. Examples of this are large radius ellipses that approximate straight lines over short segments and large radius cylinders that approximate planes. This causes a problem when the model matching stage uses classified primitives where there is no provision for comparing, say, cylinders and planes.

7.2 Speedup methods

The high complexity and huge memory requirements of the standard Hough transform (SHT) have spawned a wide variety of methods to reduce the overall demands on the system. Some of the more relevant ones are as follows:-

1) *Dynamic accumulators.* In this method (the Adaptive Hough transform AHT), Hough space starts off coarsely quantised and then regions containing potential peaks are successively more finely quantised. The main problems with this are that quantisation effects are exacerbated and that isolated, relatively weak features can be overlooked.

2) *Parameter space decomposition.* In this variation, a high dimension problem is split up into several lower dimension problems. For instance, ellipse detectors (a 5 parameter problem) often work by detecting the centre of the ellipses using a diameter bisection or chord-tangent method, followed by a three parameter Hough transform to calculate the two axes and the orientation [78]. This method results in considerable savings in computational load and memory requirements but there is a generic problem with this type of method in that errors in the earlier stages of processing tend to propagate along the chain.

3) *Parameter constraint.* If something is known about the nature of the image being analysed then limits can be set on the range of parameters that need to be scanned during the Hough process.

   One example of this is the range of ρ values required by the 2D line transformation in (7.1). If the image is 256 by 256 then the maximum distance any line can be from the origin is $\sqrt{(2 \times 256^2)} = 362$ which sets an upper limit on ρ.

Another example is the use of gradient information from edge detectors to reduce the complexity of the calculation by one. If the value of θ in (7.1) can be extracted from the edge detector then the Hough transform becomes a $1 \to 1$ mapping between image space and transform space which means that the sinusoids no longer need to be plotted.

Other more sophisticated constraints can be applied, although these are mostly dependent on the particular transform being considered.

7.3 Post-processing

1) *Least squares fitting.* Due to the inaccuracies and potential pit-falls in the Hough transform process, most practical implementations of Hough techniques improve the quality of the detected features by referencing back to the original data. One common method is to collect the original points in the data that are associated with a given feature and perform a least-squares fit or some other kind of statistical analysis to the original data [22]. The reason for this is to minimise the effect of errors in the angular component of detected edges on the accuracy of the detected features although this method does allow outliers to contribute to the overall feature.

2) *Split and merge.* Another method of cleaning up spurious features is to apply a split and merge process to the features. If two features co-incide by a given metric or are somehow contiguous with one another then they can be combined into a single feature. Conversely, if a given feature can be shown to have two distinct sub-features, then the sub-features can be extracted and handled separately. Usually, these two measures are combined into a single measure for global optimisation.

3) *Filtering Hough space.* Due to inaccuracies in the detected edges, compounded by quantisation errors during the transform and accumulation processes, peaks in Hough space can be difficult to separate. There is the possibility of more than one peak being found in a single accumulation cell and, conversely, a single peak may be spread across several cells.

   One method of improving peak detection is to use filtering in Hough space. For maximum effect, the size of the features and the degree of quantisation of both the image and Hough space can be used to design an optimal filter for a given image [45]. This information is not always available, however.

   Another improvement in peak detection is the use of the median position of approximate peaks [21]. In this method a coarse Hough transform is performed to provide the approximate location of peaks, followed by finding the median position of the peaks along two perpendicular strips through the approximate peak.

4) *Gerig and Klein Backmapping.* In this technique, the Hough transform is carried out in the usual way. A second Hough space is then generated by re-scanning the points plotted in Hough space for each image point and accumulating the point which occurs at the highest accumulator value [30].

   This doubles the overall complexity of the Hough process but simplifies peak detection. This is because the second Hough space is much cleaner with fewer noise points. Points accumulated in the second Hough space are therefore attributed to the peak for which they have the greatest probability of belonging.

   This method does not work for transformations which are $1 \to 1$ mappings.

5) *Connectivity analysis.* Due to the global nature of the transform process, the features distinguished tend to be either whole features or features of infinite extent. While this is one of the strengths of the method in that partial or occluded features can be easily recognised it is more useful from the point of

view of the model matching process to have individual features of finite extent such as a segment from a line or an elliptical arc.

To achieve this, the original points in the image that contributed to a given peak in Hough space are subjected to a connectivity analysis, usually based on a fixed distance between neighbouring points within a given feature [22].

6) *Co-operation with edge detectors.* In the original paper by Canny on edge detectors [15], it was suggested that the image should be convolved at different *sigma* values for the filters, followed by feature synthesis to combine the edges detected at different filter widths. This takes into account the tradeoff between localisation of the edges and the probability of detecting false edges. This analysis would increase the computational complexity of the edge detector and may only provided limited additional information since it was observed that the larger widths tended to pick up features such as shadows and textural boundaries.

   One possibility, however, is to use the output from the Hough transform to guide the edge detector to perform more thorough analysis in the region of suspected lines. A similar method might improve the range of edges that can be detected by removing features that have already been identified from the original image, thereby potentially unmasking weaker lines in the neighbourhood of stronger ones.

7) *Special feature detection.* Once lines have been identified in an image, it may be desirable to more accurately place the junctions of lines using a dedicated corner detector. There is a simple technique for doing this based on the difference between a median filtered image and the original image [57], but this is highly non-mathematical and may only provide supporting evidence for the conjunction of edges rather than more accurate placement of the junctions. Another approach uses the generalised Hough transform [19]. Similar supporting evidence can be gained from a small hole detector, which may pick up small features that could be missed by a circle Hough transform. There are a number of variations on this theme, mostly based on lateral histograms followed by ambiguity resolving [76].

7.4 Transformation equations

1) **2D line**

   The normal form for the equation of a 2D straight line is given in Eqn. (7.1). The (x, y) values for each point in the image are used to plot ρ versus the range of θ values required, which is actually a series of sinusoids.

   Alternatively θ may be known from the edge detector orientation in which case the transform process becomes a $1 \rightarrow 1$ mapping from (x, y) to (ρ, θ).

2) **2D ellipse**

   The parametric equations for an ellipse are:-

$$x = a\cos\theta \tag{7.2}$$

$$y = b\sin\theta \tag{7.3}$$

   Most ellipse detectors are two or three stage processes. A common method is to find the centre of the ellipse using a chord-tangent method followed by a three parameter Hough transformation for the remaining parameters.

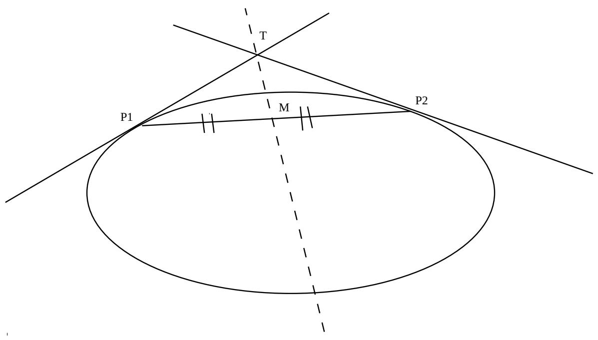

Figure 7.1: The chord tangent method

The chord-tangent method uses the edge orientation for *each pair* of points in the image to plot a line through the centre of the ellipse using the line between the intersection of the tangents and the mid-point of the two points, the transform space is accumulated in the usual manner for Hough transforms, see Fig. 7.1.

The remaining parameters can be found by substituting the centre point into the general ellipse equation and differentiating:-

$$Ax^2 + 2Hxy + By^2 + 2Gx + 2Fy + C = 0 \tag{7.4}$$

where $AB > H^2$ and $A \neq 0$.

Setting $x = x_c$ and $y = y_c$ gives:-

$$x^2 + 2Hxy + By^2 + 2Gx + 2Fy + C' = 0 \tag{7.5}$$

Differentiating this gives:-

$$x + Byy' + H(y + xy') = 0 \tag{7.6}$$

This defines an (H, B) parameter space for the hough transform which can be followed by a 1D parameter space for the final parameter C. The a, b and θ values for the ellipse can be calculated from H, B and C.

An alternative formulation uses the equation:-

$$\frac{(u + c + \frac{d^2}{2c})}{d^2} + \frac{v^2}{4d^2} = 1 \tag{7.7}$$

where:-

$$c = \frac{a + b}{2} \quad and \quad d = \frac{a - b}{2} \tag{7.8}$$

This allows the centre of ellipses of known a and b to be detected by plotting ellipses in hough space.

3) **2D corner/hole**

The *lateral histogram* method has been applied to both corner and small hole detectors [18],[76]. This method involves calculating an intensity histogram for both horizontal and vertical directions in an intensity image. This gives a series of peaks along each edge of an image corresponding to the x and y co-ordinates of potential features. The main problem is then to resolve the ambiguity resulting from combining the two sets of co-ordinates.

Both *template matching* and the *generalised Hough transform* approaches have been applied to hole and corner detectors.

For corner detection there is also an unusual method based on median filtering [57], in which a median filtered image is subtracted from the original image and also a number of methods based on the properties of the second order differentials of the intensity image, eg. [43].

4) **3D plane**

One form of the equation of a plane in 3D is:-

$$z = ax + by + c \tag{7.9}$$

Previous work included the HoughPlanes programme by the University of Surrey. This method works by calculating the gradients of the depth image in the x and y directions, $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$, using interpolation to fill in missing data. The gradients are then accumulated in Hough space with Gaussian smoothing and using the Gerig and Klein backmapping method.

This picks out sets of parallel planes ie. the a and b terms in Eqn. 7.9 and to fix the position of the plane in space a 1D Hough transform is performed in 'c' space.

Another method developed at the University of Surrey is the HaRASS programme which fits biquadratic surface patches to small sections of a depth image using robust statistical methods. Regions are then grown by grouping neighbouring patches with compatible curvatures.

This method can also be applied to cylinders, cones and spheres although at present, only cylinders are catered for and problems of patch classification arise.

7.5 Operators

Due to the large number of possible applications of the Hough technique and the large numbers of methods of improving the computational complexity and memory requirements it is proposed to build a framework upon which simple or advanced methods could be developed as required.

The key components of any Hough process are the transformation and accumulation step (assuming the input data to be filtered or thresholded as necessary), the peak detection phase (which may include processing of Hough space) and finally a post-processing phase in which the data in the original image is referenced to produce more accurate estimates or verification of the features identified.

The whole process can be iterative (eg. the adaptive Hough transform) or could be composed of a finite number of discrete steps (parameter space decomposition). In addition, the input data could be in the form of an intensity, depth or other form of image.

To cater for all these possibilities, the high-level function call for a hough process should be passed functions to carry out all of the above phases; transform, peak detect and post-process. Due to the variable number of steps, each function is specified in the form of a list of functions, one element for each stage of the Hough process.

The following is a rough approximation to the top level call for a generic Hough process. Due to the complexity of the data types involved, *type* declarations have been included to make the types of the parameters more readable:-

```
type 'a indata = (int * int) * 'a list
type params = real list
```

```
type 'a stagedata = params * 'a indata list
type GKplane = (params * real) list
type 'a HoughPlane = (params * real * 'a indata list) list
type outdata = params * (int list) list
hough:
  (input_data:'a HIPS list)
  (hough_pre_process:'a HIPS list -> 'a indata list)
  (no_cells:int list) (min:real list) (max:real_list)
  (nest_depth:int) (adapt_flag:bool)
  (stage_pre_process: ('a stagedata -> 'a stagedata) list)
  (stage_pre_transform: (('a stagedata -> GKplane) list)
  (stage_post_transform: (('a stagedata -> GKplane -> HoughPlane) list)
  (stage_peak_detect: ('a HoughPlane -> 'a stagedata list) list)
  (stage_post_process: ('a stagedata list -> 'a stagedata list) list)
  (hough_post_process: ('a stagedata list) list -> outdata list)
  -> outdata list
```

Overall this function takes a set of HIPS images as input, performs the necessary transformation and detection stages and outputs a list of parameters for each identified feature along with the original image points (x,y) that correspond to each feature. The input is in the form of several images to take into account the output from prior processing stages such as edge detectors or gradient computations.

The **hough_pre_process** function takes the input data and converts it into the form required by the *stage* functions although this could be used to perform other pre-processing functions such as thresholding.

The general philosophy in this scheme is to store co-ordinates as lists which makes, for instance, a 2D parameter space [a,b] type compatible with a 3D parameter space [a,b,c]. This allows the different stages to be supplied as lists themselves, which in turn allows variable numbers of stages to be supplied in a single high-level function call. This applies to all data of variable width, for instance the values associated with each point in the image [v1,v2,v3,...] (`'a list`).

The parameters **no_cells**, **min** and **max** define the initial quantisation of Hough space. These are in the form of lists since one set of (min,max,no_cells) is required for each dimension in Hough space. It is assumed, for adaptive versions, that the quantisation of subsequent levels has the same number of cells in each dimension as the previous step so the no_cells parameter is passed down to nested levels and new versions of min and max are computed at each level.

The **nest_depth** and **adapt_flag** parameters control the overall behaviour of the adaptive feature. Initially, a single maximum nest depth is specified but more sophisticated methods of terminating nest recursion could be developed. Another point is that the above scheme only caters for peak detection by accumulation, other methods such as clustering algorithms could also be built into the framework. The adapt_flag parameter determines whether the adaptive algorithm is applied at each stage or applied to all the stages as a single process.

The **stage_pre_process** function is applied at the start of each stage in the Hough process and is intended to provide simple operations such as thresholding. The form of the input is a list of parameters that have been identified in previous stages of computation (`params`) with a list of co-ordinates (`int * int`) with the value of the image at those co-ordinates (`int list`). The input to the first stage_pre_process is generated from the output from the hough_pre_process function by adding a nil `params` list. Note also that if any of the *stage* functions are not required, null equivalents can be supplied.

The **stage_pre_transform** step provides the intermediate parameter space required by the Gerig and Klein backmapping procedure and as such only contains the parameter definitions (`params`) and an accumulator value (`real`). This intermediate space is passed on to the next step.

The **stage_post_transform** function is the main transformation step (it can be passed a nil `GKplane` if no Gerig and Klein procedure is required). The output from this function is a list of accumulator values (`real`) along with the original data points (`'a indata list`) and the parameters for the Hough cells (`params`). The **stage_peak_detect** function performs any filtering or other processing in Hough space and picks out peaks. It is assumed that the parameters for the peak detection are built into the function. Notice that the output from this function is a list of elements of the same type (`'a stagedata list`) as the input to the stage_pre_process function.

The **stage_post_process** function is intended to allow the features identified by the peak detection to be referenced back to the original data. This may be a least squares fit or some other more robust fit which excludes outliers.

Once the list of features from a single stage has been generated, the next stage in line is applied to each element of this list in such a way that the parameters identified at that stage are appended to the parameters identified at previous stages. This is continued until all the *stage* functions are used up and all the parameters have been accumulated.

The **hough_post_process** function gathers all the data from the successive stages and generates a single list of output features (`params`) with their original data points, which could be 2D [x,y] or 3D [x,y,z] and so are stored as lists (`'a list list`). It is possible to incorporate some processing into this phase to eliminate multiply defined features such as lines that coincide, this keeps the adaptive process simple in that the effect of multiple features can be ignored during the transform process.

One of the advantages of structuring the programme in this manner are that it highlights areas where parallelism can be exploited. Each of the stage functions is approximately a *map* operation and the different components of a single stage are connected approximately in a *pipeline* configuration. The separate stages are themselves arranged in a pipeline. This means that prototyping of a given Hough transform, defined by the stage functions, could be partially automated since the different Hough transforms will have a similar structure. All the prototyping needs to do is time the individual components to give efficient processor allocations in the final versions.

Another advantage is that the generality of the Hough process is not lost. The above framework will not cover all of the possible Hough schemes or applications of the Hough transform method but by keeping the details of the processing separate a large number of different Hough transforms could be catered for, at the possible expense of the overall efficiency of the implementation.

7.6 Control variables

The output from a Hough transform could be quantified by the following properties:-

1) The total number of features detected. This is a very crude measure but could be used, for instance, to spot when there are obviously too many or too few features.

2) The statistical analysis of the dimensions of the features generated. This might give some clues as to which features are likely to be useful in later stages of processing, for instance, a textured background may produce a large number of short lines which could be filtered out. Another example is that constraints on the features likely to be found in an image could be provided by the model matching system.

3) The ratios of numbers of different types of features. Some models will have more lines than curves others may contain predominantly cylinders or planes. Since different features are likely to be generated from separate Hough transforms (to allow special attributes of the transformation equations to be exploited) these ratios, derived from the model, could be useful in balancing separate Hough transforms although assumptions about extraneous data in the image would have to be made.

4) The accumulator values for each feature. These could be used to rank the detected features in order of strength so that if too many features are presented to the model matching phase it could make its own decisions about how many to use. It may be useful to scale these values according to the dimensions of the features since a weak, long line may ultimately be more useful than a short, strong one.

5) The variances and co-variances of the parameters for each feature. These could be calculated outside of the transform process if the output includes the list of original data points contributing to the feature, in fact this might be desirable if the grouping process is capable of splitting or merging features. These values are used by some model matching algorithms but could have implications for the control system.

Present implementations of Hough transforms usually rely on thresholds at some point, particularly during peak detection and in processes which involve combining data such as calculating end points from sets of co-linear points. In addition, there are other operational parameters that can be used to control Hough transforms such as parameter space quantisation and other factors which affect the mode of operation (eg. the `adapt_flag` parameter in the above scheme). In general terms the factors which could be used to control a Hough process might be:-

1) Input filtering. In general, this can be assumed to have been carried out by previous processing of the image, for instance the hysteresis thresholding phase of the Canny edge detector or the ComputeDerivatives phase of the HoughPlanes algorithm.

2) Parameters for the adaptive aspect of the transform. These could include the number of cells in each dimension at each level, although in practice this will simply be the same as the overall parameter space quantisation factors below. A more important parameter is the nest depth. This could be a global maximum nest level or could be a heuristic measure based on the amount of data at each level. In either case there is a greater probability of a single peak being accidentally *split* by going to too fine a quantisation. This parameter might therefore be more useful for reducing the number of features when there are far too many being detected than for fine control of the Hough process. In the parallel AHT, *peak structure* was used to avoid this problem.

3) Parameter space decomposition. This is relatively fixed for a given algorithm but for any given set of parameters there could be several ways of splitting them up. It is possible that some data sets will work better with one partition of parameters than with another and so two algorithms could be run side by side with the choice of algorithm behaving like a control parameter. This is highly speculative, however.

4) Parameter space quantisation. This is probably one of the most important parameters in controlling the Hough process. The peaks in Hough space will be of a specific average size and thus will be best detected when the dimensions of the Hough cells match the dimensions of the peaks. This is also important for rejecting noise such as spurious peaks due to geometric co-incidence which will be smaller and weaker. It could also alleviate the problem of less well defined features such as shadows which will be spread over a larger area.

   The main problem with this parameter is that there are both upper and lower limits which can be used. If Hough space is too finely divided then the memory requirements will be very large, particularly with high dimensional Hough spaces. Conversely, if Hough space is too coarsely divided then features will be merged together which could lead to problems in later phases of processing, particularly segmentation algorithms.

5) Peak detection parameters. Finding the peaks in Hough space reliably is probably the most difficult task during a Hough transform and is the main source of errors on real data. Peaks in Hough space are very often modelled using Gaussian shaped peaks which leads to Gaussian filtering as a means of smoothing in Hough space.

   One problem is that peaks in Hough space are very often structured which requires matching the filtering to the peak shapes. Other peaks have random structures for no apparent reason and are even more difficult to legislate for.

   Another problem is that peaks of widely varying dimensions and strengths are required to be separated. This leads to complicated analysis, possibly recursive over the whole Hough process and has resulted in a number of statistical methods being applied.

   Yet another problem is that peak detection is dependent on the quantisation of Hough space. As mentioned above peaks can be split or merged depending on how the Hough cells are related to the peak dimensions.

   The simplest scheme merely detects all cells in Hough space which are maxima in each dimension but this is unlikely to be useful in a practical system. Global thresholds can go a long way towards reducing noise floors but there is always the danger of excluding useful information accidentally.

   Considering the complexity of this process it might be best, in an initial system, to define a reasonably robust peak detector and to leave the peak detection parameters out of the control system.

6) Post-processing parameters. In most Hough transforms there are parameters which are used in controlling the process that tidies up the data after peaks have been detected. This could be, for instance, a threshold distance and angle for merging of neighbouring features or a parameter that controls the exclusion of outliers during a least squares analysis.

   These are in general feature specific but should be included in the control variables since they are potentially very useful in controlling the quality of the data and are *close* to the controlling metric produced by the model matching process.

An initial set of parameters that could be used to define the output from a Hough process could be:-

1) The total number of each type of feature in the image.

2) The normalised accumulator value for each feature.

3) The variances and co-variances in the parameter estimates.

4) The ratios between each type of feature when multiple Houghs are used.

7.7 Notes

1) The comments in the Control variables section are only rough ideas. The initial system will probably be much simpler than indicated in this document, particularly with regards to the control system. It may be desirable, for instance, to reduce the number of thresholds and heuristic values in the system by replacing them with locally derived measures or simply ignoring them in the initial system.

2) This chapter is structured such that the sections from **Overview** to **Transformation equations** give an account of the features that might be required in the final version of the parallel vision system. The **Operators** section suggests a framework upon which to build successive versions of the system starting from an initial simplified version. The **Control variables** section gives some very general ideas that could be useful in designing the control system for the Hough component of the overall system but, as indicated, this aspect is likely to be very complicated indeed.

Chapter 8

structure RegSeg

```
structure RegSeg:
Informal Specification:v1.02

Version history
Date       By   Comment
06-Apr-94  nrs  First version created
03-May-94  wja  Removed AHT, added more on segment + firmer indication of alg.
19-Jul-94  nrs  Added algorithmic details, depth/intensity commonality
```

8.1 Overview

This chapter reviews some range image segmentation algorithms. The Laird segmentation algorithms are summarised in some depth: both the Hough and robust approaches give some insight into the types of control parameters that can be made available to any overall control process.

In specifying algorithms for the parallel system, a number of design parameters should be taken into account to ensure that the most reliable and flexible methods are used within the bounds of the programmer development time and the system architecture available to the project.

One very important parameter is that all algorithms should be capable of efficient parallelisation, even if they do not eventually end up being implemented in parallel due to the limits of the Meiko system. Some algorithms are easily realised in parallel, such as convolution with filters, whereas others, for instance the Hough transform, present more of a challenge for achieving respectable speedup figures.

The most robust methods should also be selected, where appropriate. Some shortcuts may have to be made, however, in terms of the total processing time of the system and the difficulty of implementation. It may be possible to use hybrid methods such as performing a least squares fit and then applying robust statistics when the residuals become large.

The methods used should be able to work under the proposed control system. Methods that are conducive to iterative improvement, such as adaptive filtering as a smoothing method, or are capable of being guided by later stages of processing, for instance the combining the Canny and Hough methods, are particularly suitable. Methods that provide intermediate results and metrics should also be preferred to methods that have to run to completion before being evaluated.

Methods that have general applicability to different parts of the system are also worth special consideration since they could reduce the total amount of programmer effort in building the system. Methods that fall into this category are convolution integral methods for smoothing and derivative calculation, the Hough transform for feature determination and SVD as a means of surface/curve fitting.

8.2 Calculation of Homogeneity Measures and Image Segmentation

Besl and Jain's survey of 3D object recognition [6] considers the acquisition and segmentation of range images. This is complemented by a more recent survey by Arman and Aggarwal that considers object recognition from dense range images [2]. The problem of range image segmentation was considered during the Laird project and these results are summarised in sections 8.2.1 and 8.2.2.

There are two basic approaches to range image segmentation, those based on similarity between pixels and those based on the difference between pixels. In the first, pixels with similar properties are grouped together to partition the image. A number of different grouping properties have been used, these include surface normals [67], curvature measures [8] or a combination of curvature and surface parameters recovered by fitting a surface to the pixels [9]. In the second approach, discontinuities in the depth and normal data are used to segment the image, e.g. [69, 70].

Fan *et al.* [69] detect discontinuities and extrema of curvature in range images using Gaussian smoothing and detection of zero crossings and extrema in directional curvature and principal curvatures. Once such points have been identified they are grouped to form jump and roof edges and lines of curvature extrema. The edge and extrema features are used to form closed regions to which biquadric surfaces can be fitted. The authors argue that it is easier to detect local discontinuities and reconstruct by surface fitting that to consider global smoothness measures. The results of segmentation are used in a graph matching algorithm for model detection [70]. Surface parameters are not used directly in matching, instead measures of surface orientation (normals and axes) and the average principal curvatures are used. Edge classification is used as a link attribute between surface nodes in the graph.

Al-Hujazi and Sood [1] detect discontinuities in range images using absolute residuals: the image is Gaussian smoothed and the and the difference between the smoothed and original pixel values form the residuals. Different Gaussian scales are used to detect jump and roof edges, jump edges are detected first and subtracted from the image before roof edge detection begins. Edges are extended and linked to form closed contours. A quadratic surface is fitted inside the closed contours using least-squares. Region growing is implemented by testing the edge pixels against the least-squares fit, if their residuals are less than the average fit error, they are accepted.

Edge based approaches to depth image segmentation are less common than surface based approaches. The edges detected are often not continuous and some interpolation stage is required to close contours before surface fitting can take place. In dense range images there are normally sufficient data points to allow local surface parameters to be determined and grouping can be more effectively carried out on the basis of these.

Besl and Jain advocate the use of curvature measures for the classification of pixels in range images [6, 8]. They described differential convolution operators which can be used to determine the principal curvatures of underlying surfaces in the range image. From these the mean (H) and Gaussian (K) curvatures of every pixel in the image can be determined. H and K are local surface properties which are invariant to rotation and transformation making them useful as shape descriptors in range image understanding. The actual values of H and K are not determined accurately enough to be used in model matching but the signs of H and K can be used to classify pixels into one of eight possible types: this is sufficient to allow an initial regional segmentation to be produced. In [9] this initial segmentation is used as input to a variable order region growing algorithm. The surface type image provides seed points for iterative region growing based on the fitting of low order bivariate surface to the data. Initially a planar region is fitted to the data, when the error residuals are too large, higher order polynomials (up to maximum order 4) are fitted instead. Iterative fitting proceeds until the parameter values converge within some tolerance threshold. During region growing, connected pixels are added to the seed region if they satisfy the approximating surface to the desired tolerance. After the pixels are added to the seed region the surface parameters are updated and the whole fit-and-grow process continues until no more points can be added to the seed region. The final segmentation consists of a set of smooth surfaces that allow the original range image to be reconstructed.

Gupta and Bajcsy [34] use surface based segmentation to guide volumetric segmentation. An iterative region growing approach is used to recover surfaces in the depth image. At each iteration a region is grown by extending the patch and updating its parameters. If a patch cannot be grown as a plane higher order surfaces are used instead, as in Besl and Jain's approach. 3D edges between the regions are recovered and used to create a surface adjacency graph: connected regions are linked by roof edges, disconnected regions by jump edges. In [34] this information is used to inform a volumetric segmentation algorithm, the same information would also be useful in a 3D surface based matching algorithm.

Quek *et al.* [60] use variable order surface fitting to provide an initial segmentation from which a more abstract (closer to the model) description can be derived. The segmented image and the model base are used to form hypotheses about the presence of model data in the scene which can then be verified. To extract features from the scene when the parameters are non-linear with respect to the orientation parameters, companion linear functions (CLFs) are proposed. For cylinders a biquadric surface is used as the CLF approximation that guides recovery of the actual cylinder parameters. The CLF provides a starting point for a non-linear fit to recover the actual cylinder parameters. This hypothesise and test approach can also be used to predict and verify the presence of smooth compound surfaces in the scene, e.g. the smooth merger of planes and cylinders.

Yokoya and Levine [77] also use curvature to segment depth images. They fit biquadric surfaces to local windows in the image and the equation of these surfaces can be differentiated to generate an H-K curvature sign map. A local window operator is used to determine the parameters of the surface at a given pixel but these parameters are not necessarily the ones that are assigned to the pixel, the surface fit of the overlapping windows is also considered. The surface equation derived from each neighbouring pixel is applied to the central pixel and the surface which produces the lowest residual error is assigned to the pixel. This use of overlapping windows attempts to address the problems associated with surface fitting close to a discontinuity. As well as determining surface curvature, the derivatives of the fitted equation are used to detect roof and jump edges in the depth map. The edge information allows segmentation into homogeneous patches which have no internal jump or roof discontinuities. Edges also allow neighbouring regions with the same surface classification to be discriminated. To identify planar surfaces, zero-thresholds for H and K must be determined, in [77] this is achieved by scanning a flat reference plane to provide flatness limits for a given scanner.

Hoffman and Jain describe a segmentation algorithm based on clustering [37]. Surface normal estimates for each pixel are determined using a local planar fit. Normal information, position and the depth value at each point are input to the clustering algorithm. The image is deliberately over-segmented to try to ensure that clusters do not contain points from different surfaces, over-segmented surfaces are combined in a later merging stage. Following clustering, a connected components algorithm is applied to the clusters to ensure that only distinct patches are used during classification. Normal information is used to classify patches as convex, concave or planar, the authors argue that noise in the image means that quadric based classification is inaccurate. Three different planarity tests are applied to possible planar surfaces. After classification, neighbouring patches with the same classification are considered for merging: an attempt is made to identify fold edges between the patches and if none exists the patches are merged. Edge detection is implemented by considering "slices" through the depth data of the regions under consideration.

Bhandarkar and Siebert [12] also use clustering of surface normals to over-segment range images. Following clustering, the contours of the cluster are determined using a scan-line algorithm that classifies the contour as a jump or roof discontinuity. Roof edges are verified by re-examining the scene data, if verification is not possible, neighbouring patches can be merged. The ρ-θ parameters of jump edges are determined by Hough transformation. Vertices of jump and roof edges are refined to provide a final segmentation that identifies polyhedral patches whose boundaries are classified as jump or roof edges. The paper describes segmentation of polyhedral scenes though it might be possible to generalise the approach to other surface classes.

Sabata *et al.* [64] determine the surface normal at each point using a planar fit and use this and the actual depth value to create six intensity images (one for each axis direction) using Lambertian shading. These, the authors argue, can be used to highlight differences between pixels ensuring that incompatible pixels are not included in the same image cluster. Pyramidal clustering is carried out independently for each the depth image and each of the intensity images. The individual partitions found by clustering are then combined to provide maximal clusters which over-segment the images. Two different methods for merging neighbouring clusters (both based on fitting bivariate surface patches) are described. The authors find that there are some problems with this approach close to actual boundaries, this can be partly attributed to fixed non-overlapping windows in the pyramidal clustering algorithm.

Nagata and Zha [55] used extended Gaussian images to simplify a Hough transform approach to the detection of surfaces in a scene. A modified three stage Hough transform is used. In the first two stages the orientation parameters of a surface are determined by finding planes in the EGI (the distance from the plane to the center of the sphere is recovered by the first stage, slant and tilt angles by the second). The final stage recovers any remaining parameters for the surface being investigated, the method is applied to the detection of spheres, cones and cylinders.

Roth and Levine [62] describe the extraction of geometric primitives from the scene using implicit equations of the primitives involved. The technique is based on the random sample and consensus approach of Fischler and Bolles [24]. Random minimal subsets are used to determine the equation of the primitives being sought and the remaining data points tested against this equation. If a subset of the data satisfies the equation, these are recorded as an occurrence of the primitive and the points are removed before a new minimal subset is taken to search for another primitive in the scene. The technique is interesting in the context of the earlier work on functional prototyping [75] and can be used to detect 2D and 3D lines and curves and 3D surfaces, i.e. is is applicable in both the depth and intensity routes. The main disadvantage of the approach is that it can only recover one geometric primitive per pass. Recovery could be prioritised according to the expected features in the scene, but in many cases (and for the new parallel system) this cannot be predicted in advance. This serial extraction of features makes the approach extremely costly to implement, though it can be parallelised, the solution requires global data access and is not ideally suited to MIMD processing.

Newman *et al.* [56] describe model driven classification of quadric surfaces that can be used to recover planes, cylinders, spheres and cones. The approach assumes the availability of segmented point data and surface normal information (this could be recovered by any of the techniques described above). The approach considers each patch in turn and tries to fit each type of surface to the data, the one with the lowest RMS error is selected as the actual classification. Hough based techniques are used to detect spheres and cylinders, for each possible sphere / cylinder from the model base, the radius is "injected" into the accumulation strategy and the centre position / axis direction are determined. For cones, tangent planes for each point are used to determine the vertex and then estimates for the angle and axis direction can be determined. Planes are fitted using a principal components analysis. The need to try all possible model features may make the procedure slower than other classification approaches that only consider curvature based measures. The model driven approach could be more useful during verification where single model features have been paired with regions of the image, model based matching can then be used to refine the segmentation of the scene. Such refinement could in turn lead to increased accuracy in the pose estimation.

The least-squares estimators used in many of the above techniques are known to provide poor parameter estimates when the main data distribution is contaminated by other distributions. In computer vision this problem occurs when a fitting window contains data from more than one surface. The same problem occurs when window based smoothing operators are applied during pre-processing, leading to over-smoothing close to surface discontinuities that makes edge location from the soothed data difficult. Recently, the vision community has started to consider the use of robust statistical methods [35] to address these problems. A review of robust estimation techniques can be found in [52] and a description of the LMedS operator that can be used to smooth without over-smoothing can be found in [51]. The use of robust operators was also considered during the Laird project, see section 8.2.2.

8.2.1 Surrey Planar Segmentation

The limitations of the least squares method for parameter fitting are discussed in a Laird technical report [40]. This report also introduces the use of robust estimators for fitting parameters to image data. In particular, the use of influence functions [35] which modify the behaviour of the robust estimator is discussed. Influence functions are designed to directly reduce the effect of outlying data on the robust estimator and require a robust estimation of scale to be available. The Surrey report notes that robust estimators are more computationally expensive than least squares estimators and proposes a hybrid method combining the two.

In the hybrid approach, a least squares method is applied to each pixel and the RMS error computed. Robust estimation is only applied to those pixels where the RMS error is high (large errors generally indicate discontinuities). Two robust methods can be applied: a median plane fit scheme is applied where there are low levels of contamination; and a robustly seeded fit of a higher order surface is applied where error values are higher.

The hybrid approach involves two thresholds on the RMS error: the lower one indicates that the median scheme should be used; the higher one that the robustly seeded approach should be used. These thresholds are selected by considering a percentage of the data points that should be examined by each method. A histogram of the RMS error is calculated and the threshold chosen to implement the required percentage split.

Planar segmentation as defined by the early Surrey Laird work is a two stage process. In the first stage robust estimates of the partial derivatives in the x and y directions are estimated, in the second a Hough process identifies groupings of planar pixels.

Computing Derivatives

The behaviour of the Surrey program for computing derivatives can be modified by changing the following parameters.

- The size of the fitting window (x and y can be changed independently).
- The fraction of points that should be fitted using the median estimator.
- The fraction of points that should be fitted using the robustly seeded estimator.
- The maximum number of iterations allowed per pixel for the robust stage (perhaps different values should be allowed for the median and robustly seeded stages).

- An initial estimate of the standard deviation of the noise. This can be used to reduce processing effort and may be useful if a noise estimate has been calculated during earlier processing.
- The order of the surface that should be used by the robustly seeded fitting algorithm.
- The convergence measure for the algorithm. This specifies the difference between iterative results which at termination should occur.

Within the context of the parallel system, each of these might be modified from the default value by the overall control process. Since the parallel system should allow a mixture of top-down and bottom-up processing, some of these parameters should use different default values than would be used by a serial implementation. This would delay the amount of data-driven processing which was carried out until higher level, top-down, processing indicated that the data driven approach should be restarted.

For example, the fraction of points considered by the robust methods could be reduced to minimise expensive robust processing at an early stage. Higher-level matching algorithms can re-invoke the low-level process only for those planes which match with a model. The robust estimators could be applied to improve the planar segmentation before the pose calculation took place. Where a single model is expected in the scene, this approach might be expected to improve processing times. Where more than one model was expected in the scene, it might be better to robustly process all plane boundaries anyway since most of the planes could be matched to a model. The convergence parameter can be reduced in a similar manner, with further processing to improve the parameter estimates being carried out one a plane had been matched to the model.

As well as higher-level guidance, the process can receive boundary information from the intensity based route. Edges in the intensity can be used to infer discontinuities in the depth image and could therefore be used to guide the application of robust operators in the depth image.

The compute derivatives program outputs images representing smoothed depth, RMS error and the x and y derivative maps, these are input to a Hough based grouping process. The RMS error map can be used during later processing to guide re-segmentation along the lines suggested above.

Hough Plane Detection

Output from the compute derivatives program is supplied to the Surrey find Hough planes program, [39]. To reduce parameter space complexity, surface normals are clustered in a 2D accumulator and, once peaks have been detected, a separate 1D accumulation is used to determine the distance component of the plane. A noise estimate supplied to the Hough process determines the region surrounding a parameter cell that must be incremented to account for noise in the derivatives. A connected components algorithm is applied to the pixels associated with each segmented plane to identify separate patches within the plane, these may arise from occlusion of a single surface, or from separate object surfaces.

The behaviour of the Hough plane detection algorithm can be modified by altering the following parameters.

- The upper and lower bounds on the parameter spaces.
- The resolution of the parameter space together with the bounding information determines how many cells should exist in parameter space.
- The standard deviation of the Gaussian convolved with the parameter space to model the effects of noise on the surface derivatives.
- The half-width of the window used to normalise the parameter space.

- The half-width of the back-mapping neighbourhood.
- The resolution of the 1D parameter space.
- The half width of the incrementation window for the 1D transform.
- The threshold above which peaks in the 1D accumulator should be considered.

The primary output from the Hough process is a region map of the planes in the image. Secondary output consists of the 1D and 2D accumulator arrays. Information on fitting errors from the derivatives stage should be combined with the region map.

The limits on the parameter space could be determined during derivative computation by a simple histogram of the derivative values. The main effect of this would be to save accumulator space, this is of less relevance in an occam implementation where the array sizes must be fixed at some maximum value. The standard deviation of the noise in the image could also be determined during the compute derivatives process. Parameter spaces resolution can be set by the user, but is ultimately determined by the sensor that is used.

8.2.2 Cylinder and Plane Detection: HaRASS

The later Surrey approach to surface segmentation did not use a Hough based process. Instead a region growing algorithm based on robust statistical fitting of bi-quadratic patches to data windows was implemented. The HaRASS program for planar and cylindrical surface segmentation has three main stages.

1. Bi-quadratic surfaces are fitted to non-overlapping windows in the image. From the parameters of these patches, the principal curvatures can be determined. This allows the surface to be classified as plane-only, plane-cylinder or general.
2. The surface classifications are built into plane and cylinder graph structures that associate all image windows with the appropriate classification. Within these graphs, windows are merged using a nearest neighbour algorithm that checks that neighbouring patches have compatible classifications and surface parameters.
3. Stages 1 and 2 can be viewed as a robust but coarse pre-segmentation of the data. The final stage takes each hypothesis from the graphs and verifies them using a robust iterative procedure that rejects outliers from the surface. The remaining data points are then formed into connected surfaces and boundary information is extracted (this is output in FIC format).

The HaRASS program also performs interpolation of missing data though this is not described in the accompanying documentation.

The following parameters can be used to vary the performance of the HaRASS program.

- The dimensions of the fitting window.
- A measure of the scale of the data distribution in the windows.
- The number of standard deviations (of the noise estimate within a window) that should be used to determine the outliers.
- The minimum number of data points in a segmented surface.

The dimensions of the fitting window could be determined on the basis of the model which was being searched for. However, for an extensive database of models, it might not be possible to derive such rules. A similar argument can be applied to the minimum number of data points which can appear as a surface in the output. The scale of the noise could be calculated by some pre-processing stage or could simply be based on the known characteristics of the range sensor.

In the proposed parallel system, boundaries detected in the intensity image could be used to guide the merging of windows during stage 2. Rather than using a nearest neighbour merge, boundaries from the intensity image could indicate windows to merge. For example, all windows within some closed boundary might be merged (although closed boundaries would probably be rare). Similarly windows along the line of some intensity discontinuity could be merged. Whether this would be useful is not clear, presumably the the nearest neighbour approach is relatively simple to implement (though this might not be the case in a parallel system where boundary effects must be considered).

A number of the control parameters that were available in the compute derivatives and Hough programs are no longer supported by the HaRASS program. It might be useful to make some parameters available again to allow a combination of bottom-up and top-down processing in the parallel system. For example, a convergence measure in the iterative verification stage would allow less accurate surface parameters to be used initially, these would be tightened up once it was known that the surface could be matched to a model from the model base, this would depend on the availability of processing and memory resources.

8.3 Parallel Implementation

Most of the surface segmentation techniques described in the preceeding sections can be split into three main stages:

1. Estimation of surface normals or curvature at each depth point.
2. Region growing or clustering using the results of stage 1.
3. Merging of compatible regions or clusters, possibly taking into account edge classifications derived during the earlier processing stages.

Processing during stage 1 requires local information only and each pixel can be examined independently. In theory local window operators could be applied at all points simultaneously. During region growing, there is no way to predict in advance those regions of the image that will lie within the same region: whatever date decomposition is selected for curvature estimation, it is likely that the data within any given partion will contain points from different regions and regions that extend beyond the processor boundary. Clustering is an iterative process: at each iteration the distance from each data point to a each cluster center is determined and the point is assigned to the cluster that is closest or a new cluster is created. At the end of the iteration, new cluster centres are calculated before the next iteration begins, processing stops when the clusters become stable. While the distance calculations can be carried out in parallel, a global communications phase is required after each iteration to compute and distribute the new cluster centres: this places a considerable communications overhead on the clustering technique. The merging of compatible regions / clusters is probably best views as a serial stage. While it would be possible for processors handling neighbouring patches to communicate directly, the communications pattern would be more complex than if region descriptions were passed to a single processor for merging.

8.3.1 A Proposed Solution

Memory limitations in the transputer network mean that it is unlikely that all processors will be competent to carry out all possible processing tasks. The main memory restrictions arise from the need to retain the results of each processing stage to allow a combination of top-down and bottom-up processing, Besl and Jain refer to this as "stimulus bound processing" [9]. It is suggested that data related processors (those dealing with low-level segmentation and verification against the scene) are performed by a dedicated partition within the main processor network. During early processing, this partition produces a segmented scene description that is broadcast to all appropriate matching processes. During verification, the data partition becomes a server for scene-based verification of models or model features. [56] and [53] give examples of scene based verification that may be useful in the parallel system.

Segmentation is based on the H-K approach of Besl and Jain [6, 8, 9]. Calculation of the H-K sign map can be parallelised by geometric decomposition of the data: where the task size is known and there are a fixed number of processors, this will generally perform better than process farming. Since the original image is required during verification, all processors in te low-level partition should have access to it. The image is distributed to each processor in the partition before curvature calculations begin. Decomposition is achieved by telling each processor the start and rows of the strip of data it should process. To achieve effective load balancing, the number of depth points in each processor handles shold be roughly the same, this can be achieved by counting the null depth points in each image row and partitioning the data accordingly, in [16] this is referred to as uniform partitioning. The data strips in the geometric decomposition overlap by the half width of the convolution operators to avoid boundary effects when the full curvature map is created.

At the end of the curvature calculation phase, each processor contains a single strip of the H-K sign map used to classify the image into homogeneous regions for surface fitting. Where a patch on a particular processor extends to the boundary of a data strip, the patch may extend across the boundary. This can be identified by examining a single scan line of data from each side of the boundary. This could be achieved by swapping a single row of data in one direction at the boundary however, this would only link patches on neighbouring processors: for n boundaries, n communication steps would be required to identify any region that spans all processors. During region growing it is desirable that all data points within a region are accessible on the same processor, this simplifies the control strategy and ensures that data does not have to be communicated during region growing. This requirement suggests an n step communications phase at the end of curvature estimation to place the complete curvature sign map (and any required intermediate results) on all processors in the low level network partition.

Once the complete curvature sign map has been placed on each processor in the low-level partition, process farming can be employed for region growing: each worker accepts a patch identifier and grows a region by surface fitting. In [9] the seed region is determined by applying a connected components algorithm to the patch and then eroding the largest connected region until a suitably sized seed region is obtained. Rather than applying an erosion operator, it may be possible to use patch centres in the parallel system, these can be determined during the communication phase that places the map on each processor. A connected components algorithm is still desirable to detect individual patches within the curvature classified patch, this may alter the seed region.

A region is grown by iteratively fitting and extending the seed region. A surface is fitted to the seed region and neighbouring pixels tested against this surface. Those pixels which are compatible are added to the region and a new surface fit is determined. A variable order fitting process could be used but initially only biquadric fitting will be tried. Regions are allowed to extended beyond the original patch classification provided the differently classified pixels still satisfy the region equation to an acceptable tolerance: the original classification at the boundary may be incorrect where the mask operators contained data from more than one surface.

After region growing, it is common to attempt to merge compatible neighbouring patches. This will not be attempted in the parallel version where it is assumed that this can be handled by region growing alone. If it becomes apparent that region growing is not sufficient to merge compatible patches, a serial merge process may be introduced.

Load Balancing

Load balancing during the initial curvature based classification is achieved by considering the number of depth points in the data partitions of the geometric decomposition. The unpredictable and variable size of patches during region growing makes this phase more difficult to load balance. As a general rule-of-thumb, large patches should be grown first, this tries to ensure that termination of the processor farm is not blocked by examining a large region after the other regions have been grown. If load balancing still proves to be a problem, it will be necessary to consider ways of splitting processing of a single patch across more than one processor. This might be done by parallel testing of boundary pixels, e.g. left and right boundary pixels could be examined by different processors.

Speedup and efficiency curves for surface based segmentation on the meiko are unlikely to approach the theoretical limits. The depth segmentation process should only be viewed as a way of testing mixed top-down and bottom-up processing.

Robust Fitting

So far, no mention of robust fitting has been made. Surrey used robust fitting to determine surface normals in HoughPlanes and Surface parameters in HaRASS. Robust fitting could be used throughout region growing in the parallel segmentation but this would be computationally expensive. Instead, robust fitting can be carried out to determine the seed region parameters and again when the region has been extended. Robust fitting could also be carried out periodically during the region growing process to stabilise the surface parameters every few fits: this will be investigated during implementation. The number of iterations between robust fitting stages could be one of system control parameters.

Control Parameters

The behaviour of the parallel depth segmentation process can be modified by changing the following parameters.

- The size of the fitting windows used to determine the H-K parameters (probable defaults: 5×5 or 7×7). There will be a trade off between the window size, accuracy of patch boundary, processing speed and computational overlap. This trade-off can be investigated to determine the usefulness of window size as a control parameter.
- The minimum number of data points in a patch. This could be set reasonably high to begin with and revised downwards to find further patches under the direction of higher level processes. Patches below the threshold value will not have regions grown but will entered on a list of patches to expand later if required.
- The number of iterations between robust fitting during region growing. This will be used only if experiments show that robust fitting during region growing is useful.

- The maximum number of iterations of the robust fitting algorithm for a given patch. Once a default value has been set, it is unlikely this should be changed. If the default value is exceeded, this might indicate that a higher order surface fit is required or that there are problems with the data.
- A convergence measure for the robust fitting stage. This allows robust fitting to be terminated when the surface parameters converge to the required tolerance. A control strategy might be to set this value to a low value to allow an initial estimate to be determined, this could then be refined at a later point.
- An estimate of noise for outlier discrimination. This will provide a value to override an estimate of noise obtained from the date.

Other control parameters may be determined once trials of the basic segmentation process are complete. For example, it may be possible to base the surface fit during region growing on a sample of points in the region, this would introduce a parameter to defined the sampling rate (a real number with maximum value 1.0).

Outputs

The region segmentation process instantiates the SCENE abstype. If the classification approach of Newman *et al* [56] is employed, regions will be classified as planes, cylinders, cones or spheres. If this classification is not used, general quadric patches will be output, these may be further qualified by one of the eight curvature based classifiers described by Besl and Jain. It will also be possible to output only the curvature sign map if this is required.

8.4 Operators

There are two top-level operations required during the depth segmentation process. The first is the H-K curvature sign map. This can be generated either by differential convolution operators or by fitting biquadric surfaces to local windows. In either case the basic operation is effectively a local window operator and as such are likely to be implemented using the window operations supplied by the **HIPS** abstype.

The second operation is the regional segmentation algorithm. Due to the constraints of parallelism this will initially be a region growing algorithm since this is a local operation. Region growing is likely to be implemented using processor farms. The farmer processor will be responsible for assigning seed regions to processors which then iteratively grow the seed region by adding compatible neighbouring pixels followed by recalculation of the region's parameters. Initially, it is not proposed to implement merging of neighbouring regions which would have to be carried out sequentially.

```
csm:
  (depth_data:'a HIPS)
  (window_x:int) (window_y:int)
  (HKcalc:'a IMAGE -> real * real)
  -> (real * real) HIPS
```

This function takes the raw depth data in the form of a **HIPS** depth map. A window of dimensions `window_x` by `window_y` is scanned over the depth data, the resulting windows are then passed to the `HKcalc` function which calculates the mean and gaussian curvature for that window. The resulting curvature values are then built into a **HIPS** output image.

```
regseg:
  (HKmap:real * real HIPS)
  (min_data_pts:int)
  (grow_iterations:int)
  (robust_iterations:int)
  (robust_tolerance:real)
  (noise_estimate:real)
  -> SCENE
```

This is the main regional segmentation algorithm which takes the `HKmap` produced by the `csm` function, performs the segmentation and converts the resulting patches into **SCENE** format. The parameters here are explained in the section on **Control Parameters** above, others may be required as the implementation is developed.

8.5 Algorithmic Details

This section adds some extra detail to the algorithms discussed in Section 8.3.1. The initial image smoothing phase is discussed, then the calculation of the mean and Gaussian curvature and then the curve fitting using the SVD method.

8.5.1 Image Smoothing

Nearly all range image segmentation algorithms are preceded by some method of smoothing due to the noise generated by range sensors. Simple Gaussian smoothing has been used in the past [14] but the problem with range image noise is that it is often not Gaussian in nature. The problem is quite often due to spikes at discontinuities, particularly with laser based sensors. The effect of Gaussian smoothing on such data is to spread the effect of the noise across a wider area giving poorer results than the underlying data are capable of producing.

One alternative is the adaptive smoothing filter of Saint-Marc and Medioni [65]. In this method the first derivative is iteratively smoothed by a monotonically decreasing function of probability of discontinuity. This method has the advantage of being applicable to both range and intensity images but has the disadvantage of the loss of spatial information. Re-integration to get the spatial information is likely to introduce additional errors.

The most desirable method for the parallel system is probably the iLMedS algorithm which is an improved version of the Least Median of Squares method (LMedS) [79]. This method removes outliers, notably the spikes introduced by the range sensor, and is thus a statistically robust method. With robust smoothing it may not be necessary to introduce robust fitting methods in later processing given sufficiently good underlying data. The other advantage of this method is that it can easily be implemented in parallel using geometric decomposition and an existing implementation is available in parallel C. For real data a following Gaussian filter with a small σ value would also be applied to help with any residual statistical noise.

8.5.2 Coarse Segmentation by Signs of Mean and Gaussian Curvature

Mean and Gaussian Curvature

This section is a summary of [7] and [10].

It is proposed to start the segmentation process by a coarse segmentation based on the mean H and Gaussian K curvatures. These are based on results from differential geometry which describes surfaces (or curves) in terms of the properties of the differentials of the surfaces rather than their position in space. The general parametric form of a surface is:-

$$S = \left\{(x, y, z) : x = d(u, v), y = e(u, v), z = f(u, v), (u, v) \in \mathbf{R}^2\right\} \tag{8.1}$$

where u and v define parameter space and $\mathbf{R}$ is the set of real numbers. For depth images, this form can be simplified so that $d(u, v) = u$ and $e(u, v) = v$, called graph surfaces.

There are two mathematical constructs used to describe such surfaces in differential geometry, one is the First Fundamental Form:-

$$I(u, v, du, dv) = d\mathbf{x} \cdot d\mathbf{x} = \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \tag{8.2}$$

where:-

$$g_{11} = E = \mathbf{x}_u \cdot \mathbf{x}_u \quad g_{12} = g_{21} = F = \mathbf{x}_u \cdot \mathbf{x}_v \quad g_{22} = G = \mathbf{x}_v \cdot \mathbf{x}_v \tag{8.3}$$

$$\mathbf{x}_u = \tfrac{\partial \mathbf{x}}{\partial u} \quad \mathbf{x}_v = \tfrac{\partial \mathbf{x}}{\partial v} \tag{8.4}$$

which measures the rate of change of $|d\mathbf{x}|^2$ with $d\mathbf{u}$. The Second Fundamental Form is:-

$$II(u, v, du, dv) = -d\mathbf{x} \cdot d\mathbf{n} = \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \tag{8.5}$$

where:-

$$b_{11} = L = \mathbf{x}_{uu} \cdot \mathbf{n} \quad b_{12} = b_{21} = M = \mathbf{x}_{uv} \cdot \mathbf{n} \quad b_{22} = N = \mathbf{x}_{vv} \cdot \mathbf{n} \tag{8.6}$$

$$\mathbf{x}_{uu} = \tfrac{\partial^2 \mathbf{x}}{\partial u^2} \quad \mathbf{x}_{uv} = \tfrac{\partial^2 \mathbf{x}}{\partial u \partial v} \quad \mathbf{x}_{vv} = \tfrac{\partial^2 \mathbf{x}}{\partial v^2} \tag{8.7}$$

and $\mathbf{n}$ is the unit normal to the surface at (u, v). This measures the correlation between the normal $d\mathbf{n}$ and the surface plane $d\mathbf{x}$ at (u, v).

Note that these quantities can be defined (and calculated) by the partial derivatives at the surface and can also be defined in terms of the $[g]$ and $[b]$ fundamental form matrices. The E, F, G, L, M and N Gaussian notation is also common in the literature. Note also that if $d\mathbf{n}$ and $d\mathbf{x}$ are aligned for a particular $d\mathbf{u}$ then this defines a principal direction and that the extrema of the normal curvature function defined by $II(u, v, du, dv)/I(u, v, du, dv)$ are in this direction and define the principal curvatures κ_1 and κ_2.

The six functions in the two fundamental forms characterise surfaces uniquely but it is desirable to summarise this information in a more convenient form. The matrix product $\left[g^{-1}\right][b]$ is called the Weingarten mapping and defines the shape operator, the Gaussian curvature K is the determinant of this matrix:-

$$K = det\left(\left[g^{-1}\right][b]\right) \tag{8.8}$$

The mean curvature H is half the trace of the shape operator:-

$$H = \frac{1}{2} tr\left(\left[g^{-1}\right][b]\right) \tag{8.9}$$

These are the natural scalar algebraic invariants of the shape operator and, when used in combination, can be used to classify surfaces. One of the most important properties of these values is that they are invariant to the (u, v) parameterisation (provided the Jacobian remains non-zero) and are invariant under rotation and translation and thus view independent.

Due to the difficulty of accurately determining differentials (particularly second order differentials) from noisy, discrete representations of surfaces the absolute values of H and K obtained from such data are not reliable enough to do matching but the signs (+,-,0) of the H and K curvatures are accurate enough to classify a surface point into one of eight types (K cannot be positive when H is strictly zero); peak $(H < 0\, K > 0)$, flat $(H = K = 0)$, pit $(H > 0\, K > 0)$, minimal $(H = 0\, K < 0)$, ridge $(H < 0\, K = 0)$, saddle ridge $(H < 0\, K < 0)$, valley $(H > 0\, K = 0)$ and saddle valley $(H > 0\, K < 0)$.

Evaluating Mean and Gaussian Curvature

There are a number of definitions of the mean and Gaussian curvatures that can be used to compute these curvatures. Some are more useful for calculation from parametric equations for surfaces, others are more useful for estimating the curvatures from range data.

Probably the most useful one for range data is the definition of the curvatures in terms of the elements of the fundamental form matrices:-

$$K = \frac{det(b)}{det(g)} = \frac{b_{11}b_{22} - (b_{12})^2}{g_{11}g_{22} - (g_{12})^2} = \frac{LM - N^2}{EG - F^2} \tag{8.10}$$

$$H = \frac{g_{11}b_{22} + g_{22}b_{11} - 2g_{12}b_{12}}{2\left(g_{11}g_{22} - (g_{12})^2\right)} = \frac{EN + GL - 2FM}{2\,(EG - F_2)} \tag{8.11}$$

Since range data is a graph surface of the form $\mathbf{x}(u, v) = [u\, v\, f(u, v)]^T$ the derivatives are quite easily calculated:-

$$\mathbf{x}_u = [1\, 0\, f_u]^T \quad \mathbf{x}_v = [0\, 1\, f_v]^T \tag{8.12}$$

$$\mathbf{x}_{uu} = [0\, 0\, f_{uu}]^T \quad \mathbf{x}_{vv} = [0\, 0\, f_{vv}]^T \quad \mathbf{x}_{uv} = [0\, 0\, f_{uv}]^T \tag{8.13}$$

$$\mathbf{n} = \frac{1}{\sqrt{1 + f_u^2 + f_v^2}}[-f_u\ -f_v\ 1]^T \tag{8.14}$$

These definitions can be used in conjunction with the dot product definitions of the matrix elements (Eqn.s 8.3 and 8.6) to give:-

$$\begin{array}{lll} g_{11} = 1 + f_u^2 & g_{12} = f_u f_v & g_{22} = 1 + f_v^2 \\ b_{11} = \frac{f_{uu}}{\sqrt{1+f_u^2+f_v^2}} & b_{12} = \frac{f_{uv}}{\sqrt{1+f_u^2+f_v^2}} & b_{22} = \frac{f_{vv}}{\sqrt{1+f_u^2+f_v^2}} \end{array} \tag{8.15}$$

Using these definitions for the matrix elements allows the curvatures to be expressed in terms of the differentials of f, the graph surface definition function:-

$$K = \frac{f_{uu} f_{vv} - f_{uv}^2}{(1 + f_u^2 + f_v^2)^2} \tag{8.16}$$

$$H = \frac{f_{uu} + f_{vv} + f_{uu} f_v^2 + f_{vv} f_u^2 - 2 f_u f_v f_{uv}}{2(1 + f_u^2 + f_v^2)^{3/2}} \tag{8.17}$$

These equations reduce the problem of evaluating H and K to evaluating the surface partial derivatives. These could be calculated using numerical differentiation but there is a much simpler method that makes use of convolution integrals.

To avoid calculating a full surface fit, a small $(N \times N)$ window is fitted locally, giving estimates of the derivatives from the equation of the fitted curve. Masks have been used to perform a quadratic least squares surface fit [7], with simple equations to extract the partial derivatives.

Given a $N \times N$ mask, with N odd, indexed by values from $u = -(N-1)/2$ to $u = (N-1)/2$, the masks are defined by the normalised polynomials:-

$$b_0(u) = \frac{1}{N} \tag{8.18}$$

$$b_1(u) = \frac{3}{M(M+1)(2M+1)} u \tag{8.19}$$

$$b_2(u) = \frac{1}{P(M)} \left(u^2 - \frac{M(M+1)}{3} \right) \tag{8.20}$$

where $M = (N-1)/2$ and:-

$$P(M) = \frac{8}{45} M^5 + \frac{4}{9} M^4 + \frac{2}{9} M^3 - \frac{1}{9} M^2 - \frac{1}{15} M \tag{8.21}$$

Using these particular masks, the coefficients of the quadratic fit a_{ij}, $(0 \leq i, j \leq 2)$ can be calculated from:-

$$a_{ij} = \sum_{u,v=-M}^{M} f(u,v) b_i(u) b_j(v) \tag{8.22}$$

and hence the partial derivatives from:-

$$f_u = a_{10} \quad f_v = a_{01} \quad f_{uu} = 2a_{20} \quad f_{uv} = a_{11} \quad f_{vv} = 2a_{02} \tag{8.23}$$

Since the expression for the coefficients is separable in u and v each partial derivative can be calculated by a separate convolution.

There are several disadvantages of this approach. Firstly the least squares fit is unweighted which could lead to problems with outliers and will cause the operator to break down at discontinuities. Secondly, a different fit is applied at each point on the surface with no continuity constraints. Another point is that the window size could be an important parameter without obvious means to guide the size selection.

In addition to providing a means of calculating the mean and Gaussian curvature the surface partial derivatives can also be used to calculate other, potentially useful, properties:-

1. Critical points. These are first or second order discontinuities on the surface, such as ridges or ramp edges. They can be calculated quite simply by spotting the zero crossings of the first derivative and can be combined with the mean and Gaussian curvatures to improve the coarse segmentation achieved by $H - K$ classification.
2. Metric determinant. This is the square root of the determinant of the first fundamental form matrix:-

$$\sqrt{g} = \sqrt{EG - F^2} = \sqrt{1 + f_u^2 + f_v^2} \tag{8.24}$$

   This can be summed over a surface to give the surface area and can also be thresholded to give an edge map.
3. Quadratic variation. This is defined by:-

$$Q = f_{uu}^2 + 2f_{uv}^2 + f_{vv}^2 \tag{8.25}$$

   This can be integrated over a surface to give a measure of the flatness of a region.

8.5.3 General Multidimensional Linear Least Squares Fitting by Singular Value Decomposition

The following section is primarily a summary of the Numerical Recipes book chapter 15 [58].

The general least squares method can be summarised as follows: Assume there are N data points (x_i, y_i), $i = 1, \ldots, N$ and a model $y(x; a_1, \ldots, a_M)$ with M adjustable parameters a_j, $j = 1, \ldots, M$. The least squares fitting method to find the best set of parameters a_j to fit the data points (x_i, y_j) under the given model is that which minimises:-

$$\sum_{i=1}^{N} [y_i - y(x_i; a_1, \ldots, a_M)]^2 \tag{8.26}$$

with respect to a_j.

This corresponds to maximum likelihood estimation, assuming independent and normally distributed measurement errors with constant σ. The Gaussian distribution assumption has problems with certain types of low-probability events ("tail" events) which are ascribed lower probabilities than they deserve. Range and intensity data are likely to suffer from this type of anomaly. However, since this phenomenon tends to manifest itself as large numbers of outliers, robust methods can be called into play if there are such effects during image processing. One point to note is that fitting only removes statistical errors, not systematic ones such as calibration errors.

One immediate generalisation is the relaxation of the constant σ assumption leading to weighted least squares or chi-squared fitting:-

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - y(x_i; a_1, \ldots, a_M)}{\sigma_i} \right]^2 \tag{8.27}$$

Note that this is not to be confused with the chi-squared distribution which assumes linearity in $a_1, \ldots, a_M$. If the σ_i values are not known in advance then an arbitrary single value for $\sigma_i = \sigma$ can be assumed, χ^2 can be minimised and then σ recomputed using:-

$$\sigma^2 = \sum_{i=1}^{N} \frac{[y_i - y(x_i)]^2}{N - M} \tag{8.28}$$

This, however, assumes a good fit between the model and the data and so prohibits any goodness of fit assessment. For the image data to be analysed, here, it is hoped that reasonably reliable estimates of error are available from the sensor project.

The generalised problem is obtained by differentiating Eqn. 8.27, giving the M equations:-

$$\sum_{i=1}^{N} \left(\frac{y_i - y(x_i)}{\sigma_i^2} \right) \left(\frac{\partial y(x_i; \ldots, a_k, \ldots)}{\partial a_k} \right) = 0 \tag{8.29}$$

where $i = 1, \ldots, M$.

One potential problem with image data is that both the dependent and independent variables are subject to errors. In this case, Eqn. 8.27 should strictly be:-

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - y(x_i; a_1, \ldots, a_M)}{Var(y_i - y(x_i; a_1, \ldots, a_M))} \right]^2 \tag{8.30}$$

Even for simple models such as straight lines, this introduces non-linearities into the differential which require sophisticated, iterative methods for minimising general functions.

General Linear Least Squares

A common specialisation of Eqn. 8.27 is to assume that $y(x_i; a_1, \ldots, a_M)$ is a linear combination of M, potentially non-linear, functions $X_k(x)$:-

$$y(x) = \sum_{k=1}^{M} a_k X_k(x) \tag{8.31}$$

The $X_k(x)$ functions are called the basis functions and in this case the merit function is defined as:-

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{M} a_k X_k(x_i)}{\sigma_i} \right]^2 \tag{8.32}$$

Rearranging the summations allows this expression to be written in matrix form:-

$$\chi^2 = \mathbf{b} - \mathbf{A} \cdot \mathbf{a} \tag{8.33}$$

where:-

$$A_{ij} = \frac{X_j(x_i)}{\sigma_i} \tag{8.34}$$

$$b_i = \frac{y_i}{\sigma_i} \tag{8.35}$$

The vector $\mathbf{a}$ is composed of the model parameters a_i. The matrix $\mathbf{A}$ is known as the design matrix of the problem and consists of the basis functions applied in columns to the data points in rows.

Differentiating Eqn. 8.32 gives the following M equations for the minimum of χ^2:-

$$\sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{M} a_j X_j(x_i) \right] X_k(x_i) = 0 \tag{8.36}$$

where $k = 1, \ldots, M$. Interchanging the summation order allows the matrix form for this equation to be written:-

$$(\mathbf{A}^T \cdot \mathbf{A}) = \mathbf{A}^T \cdot \mathbf{b} \tag{8.37}$$

This can be solved by numerical inversion of the $\mathbf{A}^T \cdot \mathbf{A}$ equations, however, a much better solution involves the use of singular value decomposition.

8.5.4 Least Squares by Singular Value Decomposition

One of the advantages of the SVD is that singular matrices can be handled very effectively. If there is any degeneracy in the basis functions then the columns of the design matrix can be linearly dependent making the design matrix singular. Methods that solve Eqn. 8.37 by inversion methods such as LU decomposition or Gauss-Jordan elimination cannot solve truly singular systems at all. For matrices that are almost singular, relative to the floating point representation of the machine, solutions are prone to high degrees of roundoff error due to unstably balanced fitting parameters a_k.

The singular value decomposition of a $M \times N$ matrix $\mathbf{A}$ is the product of three matrices $\mathbf{U}$, $\mathbf{W}$ and $\mathbf{V}$ such that:-

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T \tag{8.38}$$

where $\mathbf{U}$ is a $M \times N$ column-orthogonal matrix, $\mathbf{V}$ is a $N \times N$ orthogonal matrix:-

$$\mathbf{U}^T \cdot \mathbf{U} = \mathbf{V}^T \cdot \mathbf{V} = \mathbf{I} \tag{8.39}$$

and $\mathbf{W}$ is a diagonal $N \times N$ matrix with elements $w_1, \ldots, w_N$ known as the singular values of matrix $\mathbf{A}$. It turns out that all matrices have a singular value decomposition, whether N is less than, equal to or greater than M. For square matrices, the inversion is trivial and additionally, approximate inversions of singular matrices can be carried out by setting $1/w_i$ to zero instead of infinity. This corresponds to the addition of zero multiples of linear combinations of degenerate basis functions, rather than the large multiples introduced by inversion methods, thus alleviating instability and roundoff problems.

To understand the relevance to least squares methods, consider the simultaneous equations:-

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{8.40}$$

This set of equations can be solved by computing the SVD of matrix $\mathbf{A}$ and solving for $\mathbf{x}$ by:-

$$\mathbf{x} = \mathbf{V} \cdot [diag(1/w_j)] \cdot \left(\mathbf{U}^T \cdot \mathbf{b}\right) \tag{8.41}$$

If the equations are underdetermined then it can be shown that the process of setting $1/w_j = 0$ is equivalent to minimising the length of the $\mathbf{x}$ vector and that for overdetermined systems the solution vector $\mathbf{x}$ is the best approximation in the least squares sense, ie. it finds the vector $\mathbf{x}$ such that:-

$$|\mathbf{A} \cdot \mathbf{x} - \mathbf{b}| \tag{8.42}$$

is a minimum. Compare this with the linear least squares problem where it is required to find the parameters $\mathbf{a}$ that minimise:-

$$\chi^2 = |\mathbf{A} \cdot \mathbf{a} - \mathbf{b}|^2 \tag{8.43}$$

If $\mathbf{U}_{(i)}$ is the i-th column of $\mathbf{U}$ and $\mathbf{V}_{(i)}$ is the i-th column of $\mathbf{V}$ then the solution to the general linear least squares is:-

$$\mathbf{a} = \sum_{i=1}^{M} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{w_i} \right) \mathbf{V}_{(i)} \tag{8.44}$$

One incidental benefit of this is that the errors in the a_k parameters are also linear combinations of the columns of $\mathbf{V}$ so that the variances can be calculated by:-

$$\sigma^2(a_j) = \sum_{i=1}^{M} \left(\frac{V_{ji}}{w_i} \right)^2 \tag{8.45}$$

and the covariances can be calculated:-

$$Cov(a_j, a_k) = \sum_{i=1}^{M} \left(\frac{V_{ji} V_{ki}}{w_i^2} \right) \tag{8.46}$$

Another benefit is that singular values that are strictly large enough not to cause problems with roundoff error can also be edited out preventing combinations of variables that do not contribute to reducing χ^2 from increasing the errors in the a_k values.

The main disadvantage of SVD methods over other methods is that it requires extra storage space and runs significantly slower. Considering the benefits obtained, the performance is more than compensated for.

8.5.5 SVD Fitting in Intensity and Range Data

The equations above relate directly to intensity data where straight lines, ellipses and other parametric shapes can be handled with simple basis functions.

The situation for range data is slightly complicated by having basis functions of two variables. The foregoing comments, however, can be directly extended to functions of any number of variables:-

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{z_i - \sum_{k=1}^{M} a_k X_k(x_i, y_i)}{\sigma_i} \right]^2 \tag{8.47}$$

There is also a potential problem with errors in the independent variables x_i and y_i. The merit function in this case is:-

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{z_i - \sum_{k=1}^{M} a_k X_k(x_i, y_i)}{Var(z_i - \sum_{k=1}^{M} a_k X_k(x_i, y_i))} \right]^2 \tag{8.48}$$

The problem with this is that the differentials $\frac{\partial \chi^2}{\partial a_k}$ are non-linear in the a_k terms. To solve non-linear systems such as this requires advanced methods for minimising non-linear functions such as the Levenberg-Marquardt method, which are iterative in nature. It also means giving up SVD as a means of solving the equations as they are no longer in matrix form and the Levenberg-Marquardt method is itself an extension of the Normal Equations method with all the attendant problems of singular matrices.

A simple but partial solution might be to assume initial values for the a_k parameters and calculate:-

$$\sigma_i^2 = Var(z_i - \sum_{k=1}^{M} a_k X_k(x_i, y_i)) \tag{8.49}$$

The SVD fitting procedure could be run to give more accurate values of a_k and the process repeated until there is no further reduction in χ^2. The stability and convergence of this method are not guaranteed but the merit function ensures that divergence will not occur and the SVD method ensures at least partial stability of the intermediate solutions.

For image processing using the SVD approach the ability to spot degenerate basis functions is potentially very useful so it would be desirable to stick with the SVD method if at all possible.

8.5.6 Commonality Between Depth and Intensity Routes

Convolution Operations

The convolution integral forms the basis for several of the operations on raw images. These include Gaussian filtering, surface normal calculation and surface partial derivative calculation. This code could be abstracted using `CASE` statements in Occam2 to select the low-level operation required. The additional processing overhead should be minimal but there may be complications, depending on how the processors in the system are distributed among the different activities and on the total amount of data that can be stored on a single processor.

An additional requirement is parameterising the parallel implementation to ensure efficient parallel implementations with different sequential workloads and message sizes. In most cases, calculation will dominate over communication, assuming image preloading but there could be large variations in the sequential workload at each worker processor for the different applications. This will not affect the efficiency of the parallel implementation but it might preclude the possibility of running the two routes together through the same processors where one would be substantially delayed by a slower process.

It may also be desirable to "parameterise" the decomposition, since range images are more dense than intensity images and it might be prudent to use a load-balanced decomposition in the case of intensity images where simple geometric decomposition would suffice for range images. The "degree" of decomposition could also prove to be a useful parameterisation in this context. Past experience with parallel convolution operations has shown that saturation can occur with very low numbers of processors if the parallelism is too fine grained.

Region Growing and Edge Tracking

In range data, the initial classification will be improved by region growing based on the residuals of the fitted surface. It is possible to visualise a conceptually similar process for intensity data whereby edge tracking is used to extend an initial segmentation, also based on the residuals of the fitted curves.

The fitting process and residual calculations will be essentially similar for both depth and intensity data but the overall control of the growing process will be substantially different. For instance, when deciding which pixel to check next the tangent to curves in intensity data could guide the tracking process but for surfaces in range data the entire boundary of the patch should be checked.

Surface/Curve Fitting

The fitting procedures described in this document apply equally well to both intensity and range data. The only difference is in the basis functions. In fact, these form degenerate sets, for instance the basis functions for a plane are included in the general quadric equation.

While the SVD method will allow redundant parameters to be edited out, it will probably be better to include separate fitting for known surfaces and to use this ability in awkward cases, for instance a large radius cylinder approximating a plane. Thresholds could be set during the fitting procedure to force a classification one way or another or marginal parameters used to trigger "dual" representations. Two and three dimensional fitting should also use different basis functions.

If robust methods become necessary it may prove more difficult to provide a single routine to handle both depth and intensity data. It may turn out, however, that robust methods will only be needed on the depth route.

Chapter 9

abstype SCENE

```
abstype SCENE:
Informal Specification:v1.02

Version history
Date      By   Comment
11-Apr-94 nrs  First version created
09-May-94 wja  Added info relevant to natural quadrics
27-Jun-94 wja  Minor modifications and clarifications
```

9.1 Overview

This abstype is intended as a general purpose scene description data type. It is effectively a catch-all for output from segmentation routines in all three parallel system routes (intensity, depth, fusion) and will form the scene component for the input to the matching code. Since all routes feed into the same matcher a unified scene description will be necessary at some level of computation, although it may not exist in a concrete form outside the matching algorithm.

A number of desirable properties for the scene description can be identified, these include: ease of generation and storage; no redundant information; speed of access under the conditions prevalent during matching; and the ability to represent 2D (intensity) and 3D (depth / fused) data. To allow segmentation results to be examined using existing tools, existing data formats should be supported. It will be possible to generate segment files in XYZ format for visualisation. FIC codes [29] can be generated to allow the output from the segmentation process to be processed by existing vision software. If required features are not already represented in FIC, new codes will be introduced.

Satisfying these requirements may mean using multi-level data types or, alternatively, using different internal representations with conversion routines between. Each internal representation could be optimised for a particular access requirement. It is also likely that the internal representation will be highly structured this may require the definition of auxiliary abstypes. The types of surfaces and space curves to be represented are described here.

9.2 Scene Description

The surface segmentation algorithm identifies the parameters of the bi-quadric equation:

$$a_0x^2 + a_1y^2 + a_2z^2 + a_3xy + a_4yz + a_5xz + a_6x + a_7y + a_8z + a_9 = 0 \tag{9.1}$$

for each patch in the scene. For this project the natural quadrics (planes, cylinders, spheres and cones) are considered, these are sufficient to represent many man-made objects. For each of the natural quadrics, some of the parameters a_0 to a_9 in equation 9.1 will be zero, e.g. for a plane only a_6 to a_9 are defined. Both the type definition and quadric parameters of a surface are produced by the segmentation processes, in the first instance only type specific information will be supplied to the matching processes. The parameters are not likely to be accurate enough to allow direct matching of quadrics. Where a segmented patch cannot be classified as a natural quadric, the quadric parameters are given and the patch is assigned the type indeterminate.

As well as surfaces, space curves must be represented. All space curves can be approximated by a set of linear segments, this avoids the need to use more complicated (e.g. spline-based) representations. Circular and elliptical arcs will be represented provided stable segmentation algorithms can be developed.

Since there will be a dedicated processor partition for the segmentation processes, it will not be necessary to represent all possible data in the scene description. Using the "stimulus bound" approach, the complete segmentation data will be available in the low-level partition and can be specifically requested if required, the low-level server process will provide access to this information to matching or other processes as requested. If the segmented description is required by an external process, this will be generated by the low-level partition.

9.2.1 Surface Descriptions

Surfaces in the system are described by some type-specific information and a boundary description. As in FIC, boundaries are represented as "polylines", i.e. a list of (x, y, z) coordinates representing the vertices of a piecewise linear approximation to the boundary. The same polyline representation can be used for boundaries in each of the identified types. There are five possible surface types: plane, cylinder, cone, sphere, and indeterminate. Indeterminate is used when one of the natural quadrics cannot be fitted to a surface, the sign of curvature classification and surface parameters will be available but may not be used in the first instance. If indeterminate quadrics are not used in the matching process, they will not be communicated by the low-level partition.

Although the scene description passed to the matching process does not include variance and covariance information, this *is* available at the segmentation stage. If the matching algorithm requires it, e.g. to prioritise the surfaces for matching, it can be passed to the matching process. The information is represented when an external FIC file is created. A single measure of "goodness of fit" is desirable at the matching process to allow surfaces to be prioritised, RMS fitting error will be investigated for this purpose.

Polyline

Following the FIC definition, the following information is stored for polyline:

- **Polyline identifier:** this is a unique identifier for a surface boundary. It can be used to identify the boundary in the low-level partition and in any external FIC files produced.

- **Polyline type:** two types of boundary are possible: external boundaries representing the maximum extent of a surface, and internal boundaries, e.g. holes in the surface and poorly segmented regions due to surface characteristics.
- **Number of points:** specifies how many (x, y, z) points are included on the boundary.
- **Coordinate list:** the (x, y, z) coordinates of the vertices in the linear approximation of the boundary.

Plane

The following information is supplied for planes:

- **Identifier:** the unique identifier of a patch in the scene.
- **Area:** two area measures are represented in FIC, the scene description passed to the matching process(es) will use the actual area in mm^2.
- **Normal:** a unit vector describing the surface normal of the plane.
- **Centre Point:** the point nearest to the centre of the patch.
- **Surface Nearest Point:** the point in the plane nearest to the origin of the coordinate system.
- **Polylines:** a list of polyline identifiers representing the boundaries of the plane.

The quadric parameters represented in FIC are converted into a surface normal, this and a point in the plane are sufficient to classify the patch for matching purposes.

Cylinders

Cylindrical patches are represented by the following information.

- **Identifier:** the unique identifier of a patch in the scene.
- **Area:** the actual area of the patch in mm^2.
- **Radius:** the radius of the cylinder.
- **Radius Variance:** the spread in the radius estimate, this allows matching to tolerate inaccurate estimates of radius.
- **Type:** classifies the cylinder patch as convex (solid) or concave (hole).
- **Axis:** a unit vector representing the axis direction.
- **End Points:** specify the maximum extent of the cylinder. The end points are determined by considering the perpendicular projection of the surface onto the axis.
- **Nearest Point:** the point on the extended axis that is nearest to the origin of the coordinate system.
- **Surface Nearest Point:** the point on the actual surface that is nearest to the origin of the coordinate system.

- **Centre Point:** the point nearest to the centre of the patch.
- **Polylines:** a list of polyline identifiers representing the boundaries of the cylinder.

Again, no quadric parameters are supplied to the matching process: the axis, axis point and radius are sufficient to classify the cylinder.

Cones

Cone patches are represented by the following information.

- **Identifier:** the unique identifier of the patch in the scene.
- **Area:** the actual area of the patch in mm^2.
- **Angle:** the vertex-axis angle.
- **Type:** classifies the cone patch as convex (solid) or concave (hole).
- **Axis:** a unit vector representing the axis direction (from the vertex point in FIC).
- **End Points:** specify the maximum extent of the cylinder. The end points are determined by considering the perpendicular projection of the surface onto the axis. Where the vertex has been recovered from the scene, one of the endpoints will coincide with this.
- **Vertex:** the projected vertex of the cone, this may not actually exist in the scene.
- **Nearest Point:** the point on the extended axis that is nearest to the origin of the coordinate system.
- **Surface Nearest Point:** the point on the cone that is nearest to the origin of the coordinate system.
- **Centre Point:** the point nearest to the centre of the patch.
- **Polylines:** a list of polyline identifiers representing the boundaries of the cone.

Spheres

Sphere patches are represented by the following information.

- **Identifier:** the unique identifier of the patch in the scene.
- **Area:** the actual area of the patch in mm^2.
- **Centre:** the centre point of the sphere.
- **Radius:** the radius of the sphere
- **Radius Variance:** spread in the radius estimate allowing matching to tolerate inaccurate estimates of radius.
- **Type:** classifies the sphere patch as convex (solid) or concave (hole).
- **Surface Nearest Point:** the point on the sphere that is nearest to the origin of the coordinate system.
- **Centre Point:** the point nearest to the centre of the patch.
- **Polylines:** a list of polyline identifiers representing the boundaries of the sphere.

Notes

Centre points are determined using the external contour of the surface and give a rough approximation to the patch centroid. Nearest points are intended to give a stable point for calculation of the object translation.

No adjacency information is provided in the SCENE. This may be added if surfaces are merged at the end of the segmentation process. No auxiliary indices, e.g. based on the angle between feature vector pairs, are provided. Such indices may be added if the matching process turns out to be scene driven.

The surface classification process may not be able to distinguish between two types, e.g. it may be difficult to tell whether a surface patch should be a plane or a cylinder. When this occurs, both interpretations will be included in the scene abstype. During matching only one of the possible types should be present in a hypothesis - a means of linking ambiguous types is required to allow this. The simplest mechanism is to introduce a list of alternative features for each patch. If the list is empty, no special action is required by the matching process, when alternatives are available the matcher ensures different interpretations of the same patch do not appear in the same hypothesis.

The segmentation process may detect two or more patches that belong to the same surface. No attempt is made to detect this and two separate patches will be represented in the SCENE abstype. When this occurs the hypothesis generation stage of the match process is likely to select the largest of the related patches, the others can be added during the verification phase.

9.2.2 Space Curves

Three types of 3D space curves can be represented: lines, elliptical arcs (including circular arcs) and polylines. Polylines are used to approximate space curves that cannot be represented by the other two types. The definition of a polyline to represent space curves is the same as the one used in boundary representation for surfaces.

Linear Features

Linear features are represented by the following information.

- **Identifier:** the unique identifier of the feature in the low-level processor partition.
- **Length:** the length of the curve in mm^2.
- **Direction:** a unit vector giving the direction of the line.
- **Start:** (x, y, z) coordinate of start point on line.
- **End:** (x, y, z) coordinate of end point on line.
- **Centre:** (x, y, z) coordinate of the line's mid-point.
- **Nearest Point:** (x, y, z) coordinate of the point on the extended line nearest to the origin.

Elliptical Arcs

Elliptical and circular arcs are represented by the following information.

- **Identifier:** the unique identifier of the feature in the low-level processor partition.
- **Length:** the length of the curve in mm^2.
- **Centre:** (x, y, z) coordinates of the arc centre.
- **Major axis:** unit direction vector for major axis.
- **Minor axis:** unit direction vector for minor axis.
- **Radii:** the major and minor radii of curvature.
- **Normal:** the unit normal of the plane containing the arc.
- **Start:** (x, y, z) coordinate of start point of the arc.
- **End:** (x, y, z) coordinate of end point of the arc.
- **Bisector:** (x, y, z) coordinate of the point half way along the arc. This can be used to identify the portion of the arc that has been detected. For a complete ellipse or circle (start and end are the same) this point should be ignored.

When a circular arc is represented, the major and minor radii are equal and the axes are represented by zero vectors.

9.2.3 2D Lines and Arcs

2D lines and arcs for the intensity route can be represented using the 3D space curve subclasses defined in the previous section. For 2D features, some of the fields of the 3D feature would be set to zero. The z component of coordinates could be set to zero or some value associates with the position of the image plane in relation to the scene. Similarly, vectors in the 2D representation would be reduced to (x, y) components. It may be appropriate to introduce separate 2D sub-classes that take these differences into account.

9.2.4 Fused Data

Data from the fusion route will be represented by the 3D sub-classes described above. It will be necessary to augment the space curve classes with semantic labels identifying, e.g. blade, fold, mark, and shadow edges. Semantic labels will be introduced following further research carried out as part of the range finder project.

9.3 Operators

9.3.1 Basic generators

```
create_scene (filename:string) -> SCENE
```

Storing SCENE data in a file will prove useful for debug purposes. This function allows a stored SCENE in FIC format to be read into an abstype. This allows scene descriptions to be derived from external sources if required and means that the matching processes can be developed independently of the segmentation processes.

`empty_scene: () -> SCENE`

This returns an empty SCENE abstype with no other information.

9.3.2 Non-basic generators

Non-basic generators will be defined for each subclass of the SCENE description, i.e. planes, cylinders, spheres, cones, arcs, lines, and polylines. These functions allow complete scene descriptions to be built up by the segmentation processes.

9.3.3 Inquiry operations

Separate inquiry functions will be provided for each of the SCENE subclasses. When the matching process is implemented, the exact SCENE access requirements will become more obvious and separate inquiry functions will be introduced for commonly accessed data, this will avoid the need to retrieve the complete list of patch or space curve information.

9.3.4 Show functions

`commit_scene: (scene:SCENE) (filename:string) -> unit`

This function creates a file from a SCENE abstype, FIC format is used to allow visualisation using existing tools and interchange of data with other processes, not necessarily parallel.

`print_scene: (scene:SCENE) -> unit`

This is for debug purposes and prints out a summary of the contents of the SCENE abstype. This will produce an "English" description for readability. An `fprint_scene` function may be introduced to allow the symbolic representation to be saved to disk for further examination.

Chapter 10

abstype MODEL

```
abstype MODEL:
Informal Specification:v1.01

Version history
Date       By   Comment
12-Apr-94  nrs  First version created
18-Apr-94  nrs  Added hwmod2 description
11-May-94  wja  Tabularised description & added section on index
28-Jun-94  wja  Section on commonality between MODEL and SCENE
```

10.1 Overview

hwmod2 forms the basis of the MODEL abstype. Any information not required by the matching processes will be filtered out by an auxiliary program that converts hwmod2 files into a simpler numerical format for use by either functional prototypes or by the parallel implementation.

Corresponding to this simplified file structure will be an internal representation optimised for use in the model matching phase of the parallel system, this may mean restructuring the data to minimise access times. A specification of the parameters to be included in the MODEL abstype and the internal data structure for the abstype should be defined separately from hwmod2: this would then be viewed as the file format corresponding to the MODEL abstype.

10.2 Existing Model Formats

During the Laird project three different Heriot-Watt model formats were defined. The contents of each model reflect the understanding of representational requirements and matching algorithms at different stages in the project. None of the model formats was backward compatible with earlier versions.

The first format (hwmod1) contains infinite primitives for planes, cylinder, cones, and lines with closed primitives for spheres and ellipses (including circles). Finite sections of these identify actual surfaces and segments in the model. The model format is highly redundant and it is often necessary to follow several levels of pointer to gather all the relevant information for any given patch. Each patch explicitly contains a measure of surface area and pointers to the enclosing contour and infinite primitive from which the patch is derived. These pointers must be followed to gather information about the surface normal (or axis) and contour points and segments for each individual patch. Normals and line directions are represented by a pointer to a vector value. The abundance of pointers in the model format was intended to avoid duplication of data but often acts to make the existing information more difficult to access.

In the second format (hwmod2) infinite primitives are removed and the information accessed via these is made explicit in the patch representation. The vector, radius and angle entities are also made explicit in the patch. Hwmod2 introduces a new global entity that summarises the model structure, this includes: bounding volume, centre of mass, radius of bounding sphere, default surface properties, and a list of the surface and segment entities that make up the model. Each of the surface primitives can define its own surface properties to over-ride the global defaults. The model includes surface facets derived from the original CAD model that can be used to produce a ray traced image. All circular surfaces and curves include planar or linear approximations from the CAD model. The contour primitives from hwmod1 are replaced by 3 separate lists: linear segments, circular arcs, and approximating segments. The introduction of facet information leads to models files that, on average, are about ten times larger than hwmod1 representations of the same object. The removal of infinite surfaces and rationalisation of representation makes the stored information easier to access in hwmod2.

The hwmod3 format returned to a simpler model representation that supports matching based on surface patches without extensive boundary information. Hwmod3 represents planar and cylindrical patches and linear and circular segments. The surface entities include an optional list of graphical segments representing the surface boundary but these are only used during model visualisation, not during matching. Facets and adjacency information are no longer represented. The hwmod3 format can be used to represent scenes as well as models. Uncertainty domains in the model format allow the uncertainty in the scene segmentation to be made explicit in the model. This scene representation ignores much of the information available in the FIC codes, e.g. adjacency and surface area.

The cooperative program uses a model structure derived from hwmod1 but with most of the redundancy removed. A 'C' program was written to convert hwmod1 text into a numerical format that could be more easily read by an occam program. Separate files are produced for: planar patches, cylindrical patches, points, constraints, contours, segments, and elliptical arcs. In the occam program the last three of these are not used. The constraint file contains pairwise constraints between surfaces in the model which are used during tree search, these are compiled off-line to avoid unnecessary processing in the parallel system. The 'C' program makes the centroid, bounding box, extremity points and adjacencies for each surface available to avoid searching the model structure or performing unnecessary calculations at runtime. The connectivity information for points is made available to allow triples to be generated in the parallel PIA. The combination of surface and point information used by the cooperative model structure allows the programs to convert from junctions to surfaces and *vice versa*, cooperation between the depth and intensity information is based on this ability.

The parallel system detects surfaces and space curves during segmentation, suggesting that the MODEL abstype should support both feature types: hwmod1, hwmod2 and the cooperative model all provide this support. Hwmod2 also provides facet information that could be used to provide full ray-tracing during object verification if required. Hwmod2 is the most comprehensive of the existing model structures and will be used as the basis for the MODEL abstype. As in the cooperative system, an auxiliary 'C' program will be provided to generate occam model files. To allow system resources to be re-allocated during parallel model matching, it is desirable that the model base be available on all match processors. This implies that the model representation should be as compact as possible both to conserve memory and to reduce the initial distribution overhead: any redundant data will be filtered out by the 'C' program.

Feature	**Components**
Global summary	object, area, bounding box (xmin, ymin, zmin, xmax, ymax, zmax), radius of minimum bounding sphere, gravity-centre, color†, diffusion†, specularity†, transmittance†
Planar patches	area, normal, ref-point, points, adj-faces, segments, carc-edges, color†, diffusion†, specularity†, transmittance†
Cylindrical patches	area, sign (convex or concave), dir, ref-point, radius, segments, carc-edges, p1, p2, color†, diffusion†, specularity†, transmittance†
Conical patches	area, sign (convex or concave), dir, ref-point, vertex, slope, segments, carc-edges, p1, p2, color†, diffusion†, specularity†, transmittance†
Spherical patches	area, sign (convex or concave), centre, radius, segments, carc-edges, p1, p2, color†, diffusion†, specularity†, transmittance†
Line segments	length, dir, ref-point, extr1, extr2
Circular arcs	length, centre, normal, radius, angle
Points	coord

Table 10.1: Features represented by **hwmod2**

The transformation of the model into the scene for verification purposes could be encapsulated within the MODEL abstype. It is debatable whether this is actually part of the MODEL abstype, SCENE abstype, POSE abstype or is part of some system process that is unrelated to an abstype.

10.3 Hwmod2 Features

Table 10.1 identifies features from hwmod2 that may be used in the parallel system. Those features marked †should be considered optional, they will be filtered of the actual abstype unless a definite need for them is identified during implementation of the matching phase. Full details of the features and their components can be found in [68]. Any additional information, e.g. adjacency related, that is required by the matching process will be added to the MODEL definition as required. The model may be augmented by an index structure and pre-compiled feature constraints, see the next section.

10.4 Commonality with SCENE abstype

Table 10.2 compares the features that are represented in the MODEL and SCENE abstypes. It shows that there is a considerable degree of commonality between the two abstypes. The main differences are in the representation of surface boundary information. The SCENE abstype uses polylines, i.e. a chain of linear features to represent patch boundaries. MODEL represents boundaries using circular arcs and line segments. It would be possible to approximately represent circular arcs in the MODEL boundary using linear segments to create polyline boundaries however, circular arcs are desirable when dealing with space curves from the scene. Another possibility is to have dual boundary representations in the MODEL but this would increase the storage requirements. Initially different representations will be allowed in the SCENE and MODEL abstypes: boundary information is mainly used by the verification process and this should take the different representations into account.

Feature	Common	Model Only	Scene Only
Planes	area, reference / nearest point normal	surface characteristics adjacent faces linear & circ. arcs	centre point polylines
Cylinders	area, radius sign / type axis & ends reference / nearest point	surface characteristics linear & circ. arcs	radius variance surface nearest pt. centre point polylines
Cones	area sign / type axis & ends reference / nearest point angle / slope vertex	surface characteristics linear & circ. arcs	 surface nearest pt. centre point polylines
Spheres	area sign / type centre	surface characteristics linear & circ. arcs	radius variance surface nearest point centre nearest point polylines
Line Segs.	length direction reference point endpoints		
Arcs Segs.	length centre normal end points	only one radius	min. and maj. radii bisector

Table 10.2: Comparison of model and scene features

Another difference between the model and scene abstypes is the presence of centre points, surface points, and radius variances in the scene. It is not clear how useful these will be in the matching process and it is likely that they will be dropped in the actual SCENE implementation. Similarly the surface characteristics represented in the MODEL will not be used in the first instance and can be omitted from the implementation of that abstype. This leaves boundary representation as the main difference between the two abstypes.

The commonality between the features represented in the MODEL and SCENE abstypes allows them to be implemented using the same basic data structures and access functions. Differences between them are accommodated by extending the basic representation to include linear and arc segments for the MODEL and polylines for the SCENE. Further extensions to the MODEL abstype are necessary to represent auxiliary indices though these may be better implemented as an auxiliary abstype, see next section.

10.5 Model-Based Constraints

As well as comparing individual model features to scene features, model-based vision systems tend to use geometric information from the model to constrain the model-scene match. Wallace and McAndrew used binary relations between 2D scene features to rapidly index the model structure [72, 73, 48]. The relations between model features are encoded in a "cellular index", when indexed by the same relation between two scene features, a list of possible model pairs that match the scene features is made available. Grimson and Lozano-Perez introduced the geometry constrained tree search for 3D model-based matching [33, 32]. In this approach binary angle and distance constraints from the model-base are used to constrain the growth of the search tree. A model index based on the values of the binary constraints can be constructed to simplify the process of tree expansion, the index is accessed by values of the binary constraints used.

Both of these indexing approaches use constraints between scene data to identify a model and features within that model that may explain the presence of features in the scene. They suggest that the model structure should be extended to include a feature index based on binary constraints between features in the model. If individual models are being examined in parallel, it will only be necessary to return plausible features for the model being examined. A feature based index appropriate to the matching processes used will be built for the parallel system. If there are different processes for 2D and 3D matching, separate model indices may be created. Construction of the model index can take place off-line, e.g. in the program that generates the model files, or on-line, this would give rise to a one-off processing overhead to create the indices. The structure of the indices will become clearer once the matching processes have been specified.

Note: A feature index for the model is appropriate where scene-driven processing is carried out. If the processing is model-driven, it may be more appropriate to build similar index for the *scene*.

Another form of model-based indexing, geometric hashing [13], can be used as a complete matching process. In this approach, model features are used to a create local coordinate system and the positions of all other features are measured against this basis to create a hash table. This is repeated for all possible model feature pairings and for all models. Recognition is achieved by creating a basis system for two randomly selected scene features and using the positions of the other features to vote into hash table. The approach has been parallelised on the connection machine but is probably too expensive to run on a general purpose parallel architecture.

10.6 Operators

10.6.1 Basic generators

`create_model (filename:string) -> MODEL`

This function reads a model file (or files) and converts the data into MODEL format. The model file(s) will be based on hwmod2 but may be converted into a numerical format by an auxiliary program. There may be one model file for each possible model feature.

`empty_model: () -> MODEL`

This returns an empty MODEL abstype with no other information and may be useful as part of a set of routines used to create simple test data.

10.6.2 Non-basic generators

`model_add_feature: (model:MODEL) (feature:FEATURE) -> MODEL`

This function could be used to generate simple test models. The `feature` parameter is in the form of a FEATURE abstype which would be designed to represent individual features such as OBJECT name. Some means of handling the hierarchical structure of the model format would have to be devised.

10.6.3 Inquiry operations

```
model_get_feature:
  (model:MODEL) (object_name:string) (patch_name:string) (parameter_name:string)
  -> FEATURE
```

This returns the value of the given parameter for the named patch for the given object. In practice, separate auxiliary abstypes would be used at each level in the data structure. In addition to the `mode_get_feature` function, a `model_index` function will be defined to access the model via a binary feature index, this function will return a list of all possible model features that match the given scene index features.

10.6.4 Show functions

```
commit_model: (model:MODEL) (filename:string) -> unit
```

This function creates a file from a MODEL abstype.

```
print_model: (model:MODEL) -> unit
```

This is for debug purposes and prints out a summary of the contents of the MODEL abstype.

Chapter 11

structure Match

```
structure Match:
Informal Specification:v1.00

Version history
Date       By   Comment
11-Apr-94  nrs  First version created
12-May-94  wja  More information on matching and verification.
23-Jun-94  wja  Expanded tree search, added pose clustering.
```

11.1 Overview

The matching process attempts to associate a scene description with a model (or models) from the model base. For each model identified in the scene, the parameters of the model-scene transformation and a list of matched model and scene features are determined. Ideally some measure of the error in the match parameters will also be available. Where appropriate, the matching process should guide low-level processes in the system, e.g. feedback from the matcher may be used to re-segment or refine the estimate of certain regions in the scene.

The model-based matching process can be split into three distinct sub processes:

1. Generalisation of hypotheses associating scene and model features.
2. Pose determination for each hypothesis.
3. Verification of the pose estimate (and possible refinement).

After an initial hypothesis is generated, processing in stages 2 and 3 can proceed in parallel with processing in stage 1.

11.2 Hypothesis Generation

Hypotheses can be generated by random selection of model and scene features, Hough transformation, or geometry constrained tree search. The RANSAC approach of Fischler and Bolles [24] can be applied to model-based vision, this was done by McAndrew to create the Perspective Inversion Algorithm [50, 49]. In the PIA, minimal subsets of the model and scene data are selected and used to determine the viewing parameters. After inversion, the model is projected into the scene and a measure of goodness is determined. The select-invert-test process continues until a good enough fit has been achieved. It has been noted that generation and testing of minimal subsets can be carried out in parallel [25] and this fact was exploited by a functional prototype and parallel implementation of the PIA [75]. Although parallelisation of RANSAC based approaches is very straightforward, unless some suitable heuristic can be developed to guide selection of the minimal subsets, the approach can be expensive.

Model-based vision can be based on the generalised Hough Transform proposed by Ballard [5]. In this approach scene features are transformed into a parameter space and this is followed by peak detection to identify likely model occurrences. The main drawbacks of the approach are the parameter space storage requirements and the difficulty of peak detection in high dimension parameter spaces. It should also be noted that intermediate results are difficult to recover during GHT.

Geometry constrained searching processes have been widely used in model-based computer vision [73, 54, 32, 28, 31, 63, 26]. In this approach geometric constraints derived from the model are used to prune the search space of model-scene correspondences. The search space is normally referred to as the interpretation tree: each node represents a model-scene feature pair and each path through the tree represents a model scene hypothesis. Nodes below the root level can only be added if certain binary constraints between the scene feature and all parent scene features agree with the corresponding model constraints. Each path through the tree is locally consistent but may not be globally consistent, this is determined by the pose estimation and verification processes.

11.2.1 Geometry Constrained Tree Search

Geometry constrained tree search is proposed for the hypothesis generation stage of the depth matching process. The basic technique is well documented and understood and tree search algorithms have previously been parallelised [61, 44, 63]. Geometry constrained tree search can produce intermediate results that can be used to gauge the progress of hypothesis generation. The basic approach can be modified to accumulate an estimate of pose as matching progresses though this would probably not be robust. Tree search can be used to combine 3D surface and curved features in the same process, constraints between surfaces and curves can be developed to allow this.

Grimson and Lozano-Perez [33] identify three binary constraints for tree searches involving planar model and scene features.

1. An angles constraint that measures the angle between the surface normals of the features.
2. A distance constraint that measures the range of possible distances from one feature to the other. This can be measured in the model by considering a vector between points on the boundaries of the features. For a pair of scene features, it is sufficient to construct a vector between a point on each surface and verify that this falls within the model's predicted range.
3. A component constraint that measures the range of possible strengths of the distance component in each of three different directions in a local coordinate system derived from the vectors used in constraint 1 and their cross product.

Feature	**Stable vector**	**Distance Points**
Plane	Normal	Extremity points
Cylinder	Axis	End points
Cone	Axis	Vertex and axis end
Sphere	None	Centre point
Line	Direction vector	End points
Arc	None	End points (centre?)

Table 11.1: Vectors and points used in constraint construction

	Plane	**Cylinder**	**Cone**	**Sphere**	**Line**	**Arc**
Plane	θ, d, c	θ, d, c	θ, d, c	d	θ, d, c	θ?, d, c?
Cylinder	θ, d, c	θ, d, c	θ, d, c	d	θ, d, c	θ?, d, c?
Cone	θ, d, c	θ, d, c	θ, d, c	d	θ, d, c	θ?, d, c?
Sphere	d	d	d	d	d	d
Line	θ, d, c	θ, d, c	θ, d, c	d	θ, d, c	θ?, d, c?
Arc	θ?, d, c?	θ?, d, c?	θ?, d, c?	d	θ?, d, c?	θ?, d, c?
θ = angle, d = distance, c = component						

Table 11.2: The availability of constraints for the represented feature types

This basic approach can be generalised to non-planar surfaces where some *stable* direction measure can be determined, e.g. the direction of the axis of principal curvature, and where boundary or other points to provide some distance measure can be determined. Direction and distance measures can be determined for three of the four surface types considered in this project, a reliable direction measure cannot be determined for spheres. For partial spheres direction information is available, e.g. from the centre point to the centre of mass, but this is unlikely to be determined accurately from the scene data so cannot be used. The same set of constraints can be applied to 3D lines, for circular arcs, it may not be possible to determine a direction for use in constraints. Table 11.1 shows the vectors and points used to construct the angle and distance (including component) constraints. Table 11.2 identifies the binary constraints that can be applied for the different feature combinations.

In addition to binary constraints between matched feature pairs, unary constraints are applied when individual model and scene features are matched. The most obvious unary constraint is on the feature type, when a scene patch gives rise to two or more scene features (e.g. because the classification process cannot distinguish between a cylinder and a plane) only one interpretation is permissible on any given route from the root node to a leaf of the interpretation tree. Other unary constraints can be based on feature size (e.g. radius) and extent. Though these can provide an upper limit, occlusion means that lower limits on acceptable scene features can be difficult to determine.

There are two basic ways that a geometry constrained tree search can be parallelised.

1. Nodes are expanded in parallel under the control of a farmer process that maintains the tree structure and passes complete hypotheses to the verification phase.
2. Complete tree branches are expanded in parallel and hypotheses are passed to the verification phase as they become available.

In the first approach, node expansion involves identifying all new model-scene matches that can be children of some given node using unary constraints and confirming that these are compatible with earlier matches using binary constraints. Each worker requires an existing path through the tree to a current leaf node to be passed from the farmer and provides a list of new model-scene matches that can be used to expand the path to the farmer. This mode of processing is relatively fine-grained and involves a high communications overhead: lists of feature matches are supplied in the response *and* work messages. Control parameters could be set to alter the search behaviour, e.g. depth-first or breadth-first. The former would generate hypotheses that could be passed to the pose-determination process as they became available, allowing cascaded processing. The later would provide a complete list of hypotheses at the end of tree-expansion and would not allow cascading with the pose determination phase.

The second approach is less fine-grained and is probably more suitable for implementation on a transputer based machine. Complete tree branches are examined in parallel using depth-first search and messages are only exchanged when a hypothesis becomes available, allowing the pose-determination process to run in parallel. Process farming can be used to distribute tree branches for initial investigation. Process load can be redistributed by allowing workers to examine sub-trees from other workers if no new work is available. Processing of the reallocated work is just the same as expanding an initial tree branch but there is a greater set-up overhead as context information has to be passed between workers. Initial work distribution could be determined by considering characteristic views in the model, the Sheffield system uses a similar approach for task allocation [63].

The coarse-grained approach should provide hypotheses more quickly than the first, there is a lower communications overhead and the depth first approach is naturally focussed towards hypothesis generation (one node is expanded at each iteration rather than all possible children nodes). Heuristics can be used during node expansion to give the best node to try next, e.g. connectivity information in the model can be used to predict features to try next; uniqueness in the model or scene can be used to prioritise features for matching. The main difficulty with this approach is redistribution of work if one branch fails earlier than the others, much context information has to be supplied to allow processing to be shared.

Input to the tree search algorithm consists of a SCENE abstype and a MODEL instance, each model can be examined in parallel.

Output is a list of paired model and scene features that are locally compatible. If desired, the normal and point information can be included at this stage though this would increase the communication overhead between the hypothesis generator and the pose determination process.

11.2.2 Pose Clustering (Rotation)

Pose clustering can be used as an alternative to tree search for hypothesis generation. Given a set of model vectors, $v_1, v_2, ..., v_n$ and a set of scene vectors, $s_1, s_2, ..., s_m$, it is possible to determine the axis and angle of rotation (A and θ) for each possible combination of model and scene vector pairs using the equations

$$A = (v_{i1} - s_{j1}) \otimes (v_{i2} - s_{j2}) \tag{11.1}$$

and

$$\theta = \arccos \frac{(A \otimes v_{i1}).(A \otimes s_{j1})}{|A \otimes v_{i1}||A \otimes s_{j1}|}. \tag{11.2}$$

Details of this axis angle computation can be found in [42, 26]. To generate hypotheses, the calculated axis and angle for each model-scene pair combination must be clustered in parameter space together with the matched model and scene feature identifiers. The main advantage of this approach is that the vector information can be used independently of typing information (though reliable vectors may not be available for spheres and arcs).

The main disadvantages of the approach are the storage requirements of the parameter space / feature index and the possible difficulties of detecting peaks in the parameter space. To resolve the storage problem, it is necessary to implement the accumulator using sparse arrays. Shankar and Ranka describe a sparse array representation for SIMD machines [66], but this is not suitable for transputer implementation: it requires lock-step operation of the processors and good point-to-point communication between all processors in the network. Instead a sparse parameter array must be maintained at each processor where axis-angle calculations are carried out. All parameter spaces must be merged before peak detection is performed to produce hypotheses.

An attractive feature of the approach is that it is embarrassingly parallel, axis-angle calculations can be carried out independently for each possible combination of model and scene features. If no attempt is made to reduce the computational complexity of the approach, the work can be split evenly among the available processors using geometric decomposition. Since the task size is known exactly, all processors can be supplied with the same amount of work and farming is not necessary to provide load balancing. Resources can be allocated according to model complexity ensuring that all models receive the same processing resources. Once processing is complete, a single parameter space is created for each model and peak detection is carried out on a per model basis.

Although the approach is embarrassingly parallel, it is still computationally expensive. For a scene and model containing 30 vectors each, it would be necessary to apply equations 11.1 and 11.2 some 1.9×10^5 times. A test using a single transputer performed the required calculations in about 25 seconds (excluding storage in the accumulator and peak detection). One way to reduce the computational complexity would be to introduce typing information to ensure that only compatible vector pairs were considered in the accumulation process. Feature indexing might also be used but this introduces index calculations so the potential savings could be quite small.

One effect of trying to reduce computational expense is to make task size unpredictable (without examining the vectors in a task, it is not possible to say how often the equations will be applied). Process farming would be necessary to load-balance the task among the available processors. Work messages can consist of a pair of model vectors to be compared with all possible scene pairs. A sparse accumulator must be maintained at each worker and these are combined at the farmer before peak detection is carried out (requiring an additional control message to initiate the transfer). As in the non-farmed approach, the approximate computational resources per model can be determined and used to fix the size of processor farm associated with each model.

The pose clustering approach is a global method, i.e. all calculations must be complete before hypotheses can be generated. This has two implications for the parallel system.

1. Since clustering is a global process, this stage cannot be pipelined with pose determination and verification.
2. No intermediate results are produced so it is not possible to measure the progress towards hypothesised solutions

Input to the clustering algorithm is a list of model and scene vectors, this could be derived from the MODEL and SCENE abstypes or supplied as simple lists.

Output from the pose clustering algorithm is a list of hypotheses, each associating model and scene vectors which were clustered at the same point in the axis-angle parameter space. This is the same basic form as the output from the tree search algorithm.

11.2.3 Intensity Matching

The appearance of a 3D object projected into a 2D image plane is influenced by perspective effects and is viewpoint dependent. This means that it is not possible to use explicit geometric information to constrain the matching process: the angle and distance between features are not preserved under perspective projection. Connectivity is preserved and has been used to guide image understanding, e.g. early work on consistent labeling by Clowes [17] and Waltz [74], and later work on perceptual grouping by Lowe [47, 46]. Perceptual grouping was used by McAndrew to constrain the RANSAC based perspective inversion algorithm [50].

Fischler and Bolles [24] show that a minimum of 3 point correspondences are required to invert the perspective transformation (though more than one solution may be produced). This allows a clustering method similar to that described for depth data to be employed for hypothesis generation. The inversion process recovers rotation and transformation but the clustering process will allow some variation in these when a match is accepted as part of a cluster, this means that it may still be necessary to carry out pose estimation before verification is attempted.

The selection of model and scene points to invert can be limited by using junction information. Some consideration should be given to using junctions that are more complex than the 'L' junctions used in the PIA if this is indicated by the model. More complex junctions are likely to provide better constraints for the inversion process. The use of arc data will also be considered: such features often have high significance in man-made objects.

Input to the intensity matching process is a list of scene junctions and the MODEL abstype.

Output is a list of hypothesis, each associating scene and model features that were clustered at the same point in the transformation space.

11.3 Pose Determination

Once a model-scene hypothesis has been determined, an estimate of the transformation parameters (or pose) can be estimated. For the depth route this is a 3D-3D transformation, for the intensity route a 2D-3D perspective transformation is required. This implies that the pose calculation must handle depth and intensity data separately before computing some measure of agreement. A pose estimate for the intensity route will be available as a side effect of hypothesis generation, experiments will be carried out to determine if this must be refined in the pose determination stage.. For the fused route, 3D-3D transformations can be calculated.

To determine the rotational component of pose, a list of matched model and scene vectors is required, relatively stable vectors such as surface normal and axis information should be chosen. For the translational component, stable point matches are required, this is more problematic given the variety of segmentation errors that can arise. In theory, the matched point data could be used for both rotation and translation calculation, but is seems more sensible to perform an initial rotation calculation using stable intrinsic vectors.

Two approaches to the representation of 3D rotation are common: those using quaternions, and those using a rotation matrix derived from rotation about an axis by some angle. Both approaches and a formulation of the rotation calculation are described in the book by Kanatani [42]. The quaternion approach has been used in the registration of 3D point sets [11] and in a model based vision system [23]. The axis-angle approach has been used to register 3D point data [36] and in model based vision [32].

The cooperative system represented the rotational component of pose using Euler angles. This was largely historical, both the original PIA and the functionally prototyped version used this system. Euler angles are difficult to work with and do not lead to an intuitive representation of rotation. For the purposes of the new parallel system, the angle-axis approach is will be used.

While the quaternion approach may offer a faster time to solution than the angle-axis approach, there may be no unique solution and the matrix equations can be singular. Using the axis-angle approach, least squares estimation with a singular value decomposition (SVD) of the matrix equation is applied. The SVD approach is unaffected by singularities in the data and will give a unique solution provided there are at least as many data points as there are parameters to be estimated. Descriptions of SVD applied to least squares problems can be found in [27] and [59], these references also give implementations of the SVD algorithm. An initial implementation of Kanatani's rotation calculation and the SVD procedure already exist in occam.

If a robust estimation of rotation is required, the standard least squares solution may be required to seed the robust procedure. This suggests that the parallel system should first implement the least squares solution and then examine robust techniques as necessary. Robustness can also be achieved by a consensus based approach, this can be used to reject outliers with the same or greater proportion of contamination as the theoretical maximum for robust techniques [62].

Once the rotation matrix is known, the translational component of pose is determined. Another least squares estimation can be performed to determine this. If outliers have been removed during the rotation calculation and stable point information is available, normal LSE should suffice.

Input to the pose estimation process is a list of matched model and scene surfaces. This assumes that the pose estimation process has access to the model and scene representations: a list of matched vectors could be supplied this would require no further access to the model and scene structures but would increase the communication overhead. If as suggested, rotation and translation are examined separately, two scene vectors are required for each model feature matched: the first is a stable measure of orientation (e.g. surface normal) used during the rotation calculation; the second is a stable vector list which can be used to determine position.

Output from the pose estimation process includes a rotation matrix, translation vector and estimates of the error in each. If robust estimation is applied, a list of model-scene matches will also be output, this will be the original list without those matches that were identified as outliers.

Pose estimation can be viewed as a method within the scene description ADT of figure 1.1. Before pose estimation, the scene description would be partially complete, only hypothesised model-scene correspondences would exist. After pose estimation, a pose and (possibly modified) correspondence list would exist. For multi-object scenes, this data would be duplicated.

11.4 Pose Verification

A parallel approach to pose verification in the Bonsai system [26] is described in [53]. Each hypothesis is used to generate a synthetic depth image that is compared with the original depth image. Six measures are used to classify each hypothesis and determine the best model-scene match. Pose verification can be parallelised in a number of ways, but in [53], this is done on a per hypothesis basis using a task farming approach. The authors note that for limited numbers of hypotheses (as might be expected for constrained tree search object identification), speedup is limited. To improve speedup figures, they increased the number of object hypothesis, this may give an artificially high estimate of speedup (more hypotheses are used than might normally be expected). Another way to achieve better speedup is to increase parallelism by examining each hypothesis at the feature level, thus increasing the number of tasks available for process farming. The variation in size of the surfaces in a model might still lead to a processor farm that is difficult to balance. Surfaces could be divided further to make load balancing easier but this might complicate control processing.

Input data for the Bonsai verification process includes the following.

- The original depth image.
- A faceted model for creation of the synthetic depth image.
- A pose hypothesis.

The matched list of model-scene features is also useful during verification and is one of the required inputs in the transputer based system. This is used to decide the model features to project into the scene, avoiding the need to consider surfaces that the hypothesis assumes are not visible anyway.

Ray tracing of a model for verification is easily parallelised using geometric decomposition but is expensive, perhaps too expensive for practical vision applications given current hardware limitations. In the parallel system, pose verification will initially be done by projecting only wire-frame, i.e. boundary based, information into the scene and checking that this aligns with the data. Projection of individual surface boundaries can be used to parallelise verification, though tests will be necessary to verify that some advantage is obtained over the one processor per hypothesis approach. Unfortunately boundary information from the segmentation process is unlikely to be completely reliable. Problems with occlusion and surface fitting close to discontinuities mean that the wire-frame approach can only really ensure that the segmented surfaces are contained within the appropriate boundary limits. Once the basic approach has been implemented, it is likely that this will be extended to allow comparison between points projected using the calculated pose and the actual depth data. Points for projection could be sampled randomly or could be based on some concept of critical points, some further literature work and experiments will be necessary.

Once a hypothesis has been verified, features in the scene that have not yet been matched to the model can be considered to extend the match list. This might allow small surface patches and curves that were not considered in the original hypothesis generation to be considered.

The verification process requires access to the original depth image and the segmented data. Verification should therefore be carried out in the low-level partition to avoid unnecessary transmission of scene data. During verification, the low level network partition acts as a server to matching processes requesting hypothesis verification. The results of verification may be returned to the original matching process, some higher level control process, or both.

Input to the verification process is a list of model-scene matches, a pose estimate, and the original scene and segmented data. This may be derived from the intensity or depth routes.

Output is confirmation of the hypothesis (possibly extended to include further compatible model-scene matches) and a measure of the confidence or error in the pose.

Verification can be viewed as a method within the model abstract type. The model would be given a list of matches and instructed to transform itself into the scene.

11.5 Operators

The overall functionality of the matching process is provided by a single high-level call:-

```
match: (scene:SCENE) (model:MODEL) -> POSE list
```

The main inputs to this function are the scene description (in the form of a SCENE abstype) and the model database (MODEL abstype). The output is a pose estimate (POSE abstype), or a list of pose estimates of decreasing probability, together with the list of model-scene matches that gave rise to the estimate. The final ranking of the pose estimates will be determined by the control process using the estimate of error in the pose.

Other information that may be required includes initial estimates for the control parameters and the termination conditions. These could be supplied in the external environment or could be included in the parameter list. Additional output might include a trace of the system's progress including intermediate results, timing information, data flow calculations and profile information. In addition, the matching process will receive control parameters from the control system and return progress reports to the control system.

11.6 Control variables

The performance of the matching process can be modified by adjusting some of the parameters from earlier levels of processing, e.g. the minimum size of the patch or curve supplied to the matcher. Feedback from the matching process allows the earlier control parameters to be adjusted if sufficient match progress is not made.

One possible interaction between the matcher and the low-level processes occurs if a coarse segmentation was provided initially. Once a hypothesis has been verified using this initial segmentation, the low-level processes can be re-invoked to provide a more accurate estimate of the surface parameters. This refinement process may involve "injecting" parameters from the model into the segmentation process as suggested by Roth and Levine in [62].

The behaviour of the match process can be adjusted by changing the tolerance of the constraint application procedures to error. Relaxing this tolerance will increase the number of possible hypotheses. The level at which the tolerance is set should depend on the variance information that is available from the low-level processes and information about the sensor accuracy, some work will be required to identify the appropriate relationships. Similarly, some measure of the acceptable error during verification is required.

If the number of objects in the scene is known, it may be possible to use this to determine processor allocation, e.g. if there are two good hypotheses and two objects are expected most of the processing power in the verification phase should focus on these hypotheses.

Chapter 12

abstype POSE

```
abstype POSE:
Informal Specification:v1.00

Version history
Date       By   Comment
12-Apr-94  nrs  First version created
```

12.1 Overview

This is a simple abstype designed to hold a pose definition and is intended to make the specification of the parallel system logically complete. Thus the overall system behaves like a single function of type: `'a HIPS -> MODEL -> POSE list`.

The pose definition will have to store both a 3D to 2D perspective transformation for the intensity route and a 3D to 3D transformation for the depth and fusion routes.

The most likely form of the internal representation is a 3D homogeneous affine transformation matrix. Quaternions and Euler angles are also possibilities but it should be possible to store the data in one of these formats and provide conversion routines to the others if they are needed. A possible representation might be:-

```
abstype POSE = Pose of real MATRIX * (FEATURE * FEATURE) list
with
  ...
end
```

Since the pose data will be used during the verification phase, routines to actually apply the pose transformation to a vector should be included in the abstype. It may also be useful to save the pose data to files and read it back again. This would be accomplished using a convenient text file format.

It might also be desirable to include feature correspondence lists in the `POSE` abstype. These would be in the form of a list of `FEATURE` pairs.

12.2 Operators

12.2.1 Basic generators

```
create_pose
  (rotn:real list list) (trans:real list)
  (feature_list:(FEATURE * FEATURE) list)
  -> POSE
```

This function generates a POSE abstype from the rotation and translation data represented as lists and a list of FEATURE pairs.

```
create_file_pose (filename:string) -> POSE
```

This function reads the given file and converts the data into POSE format.

12.2.2 Inquiry operations

```
pose_apply_trans: (pose:POSE) (vec:real VECTOR) -> real VECTOR
```

This function applies the given pose definition to a 3D vector.

```
pose_euler_angles: (pose:POSE) -> real list
```

This function returns the Euler angles calculated from the internal representation of the given POSE abstype.

```
pose_get_feature: (pose:POSE) (feature:FEATURE) -> FEATURE list
```

This function takes a given FEATURE and returns a list of all the other features to which it corresponds.

12.2.3 Show functions

```
commit_pose: (pose:POSE) (filename:string) -> unit
```

This function creates a file from a POSE abstype.

```
print_pose: (pose:POSE) -> unit
```

This and prints out the rotation and translation transformations and the feature correspondence list in a suitable format.

Chapter 13

System Control

```
structure Control:
Informal Specification:v1.00

Version history
Date       By   Comment
04-Jul-94  wja  Initial version based on Andy's notes and discussion
```

13.1 Overview

There are two levels of control in the parallel system: *global* control is concerned with the system as a whole; *local* control is concerned with the efficient execution of individual methods in the system. Both levels of control are based on the use of metrics and parameters. Metrics measure how well a particular method is performing, parameters for a given method modify processing and performance within the method.

Figure 13.1 shows the basic control architecture for the parallel system. Metric information is passed from a given method to the local controller *and* to the global controller. This allows the local controller to monitor the method and, where necessary, adjust the method parameters. The global controller monitors metric information from all active methods and can over-ride locally set parameters or reallocate resources to improve performance at the system level. Reallocation may involve transferring resources between the routes in the system, e.g. if low-level processing on the intensity route runs to completion before the depth it may be appropriate to transfer resources to the depth segmentation processes (see section 13.4 however).

It is not clear how the control layer relates to the ADT based design of the rest of the system. Some aspects of control relate to individual methods while others relate to the interaction of methods. It is probably best to view control as distinct from the methods, perhaps the term meta-method could be applied.

13.2 Local Control

In local control, metric values are used to trigger changes in the parameters that are set for the local method. Parameters are assigned default values by the local controller at each invocation. As processing progresses, the parameters are changed by rules of the form

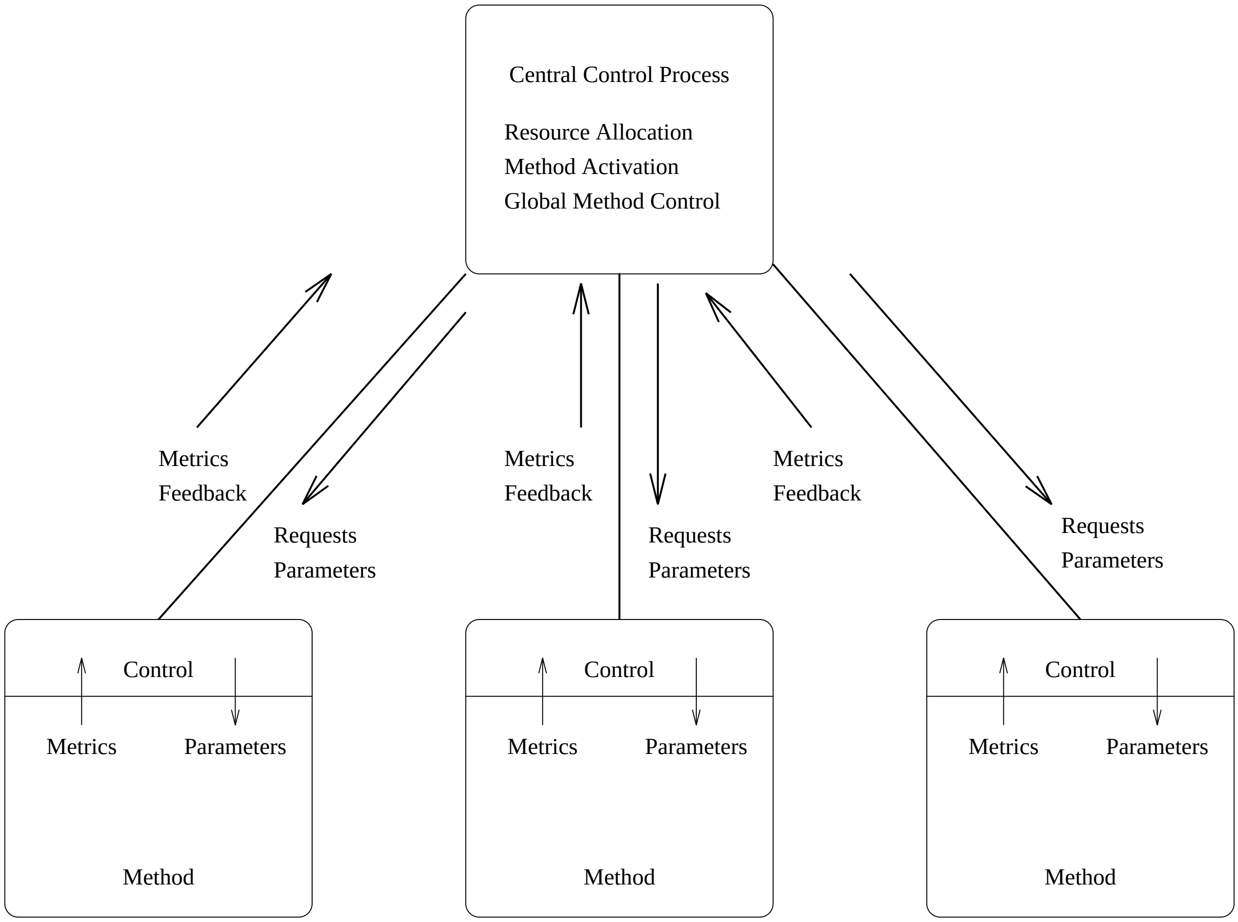

Figure 13.1: The levels of control in the parallel architecture

```
IF (metric, condition) THEN set parameter
```

In this rule, the condition can be as simple or complex as required. For example the condition could implement a simple threshold, at the other end of the scale, it might trigger some statistical analysis before the new parameter value is set. This form of control requires knowledge of the relationship between the parameters and the metrics. There is unlikely to be much information in the literature about this so testing of each method in the system will be required before control rules can be devised.

Although figure 13.1 shows control as a separate process for each method, it is likely that control rules will be incorporated within the method code. In a method where metrics are produced as evaluation proceeds the trigger rules should be invoked as the metrics appear, in a processor farm this would be done when the result messages were gathered from the workers. Where metrics are produced once the method results are complete, the trigger rules should be checked before the result is passed on to the next processing stage. If necessary the method would be re-invoked with a new set of parameters.

The impact of trigger rules checks on method evaluation should be low and should not greatly increase the latency of the method. Building trigger rules into the method means that the metrics do not have to be communicated to a separate local control process: this communications saving should more than compensate for the time taken to check the rules. It is still necessary to pass the metric to the global controller, this will introduce a communications overhead. To help minimise this, the method will pass the metric to a local high priority message handler for despatch to the global control process. There will be sufficient buffering to allow processing to continue once the message has been queued for despatch, i.e. synchronised communication with the global controller is not required. The same message handler will deal with parameters and requests from the global controller.

13.3 Global Control

The main tasks of the global controller are the scheduling methods and allocation of resources to the various methods in the system. At start up time this involves the static allocation of resources between the depth and intensity routes (the fusion route cannot be activated until initial processing in the depth and intensity routes is complete, and is not considered further in this report). Where the processes within a route can be cascaded, e.g. edge detection can provide edgels to Hough processes before a complete edge map is produced, this should be taken into account by the initial resource allocation.

This static allocation of resources represents the initialisation phase of the global control process. To be truly novel, the global controller must respond *dynamically* to the changing situation in each of the processing routes. For example if processing in the intensity route produces few identifiable features, this can be taken as an indication that the intensity route should be suspended and resources transferred to the depth route. This mode of control means that methods can be activated or re-activated at any time provided some previously identified set of conditions are satisfied. To allow dynamic control, the global controller maintains rules of the form:

```
IF precondition THEN activate method
IF metric THEN set parameter
IF metric & precondition THEN reallocate resources
```

This is in many ways analogous to the blackboard model of control where a central process is responsible for method activation and scheduling.

In [3] some shortcoming of parallel blackboard designs were identified, the main criticism was that synchronised access to resources maintained on a central blackboard reduces the available parallelism, processes spend much of their time waiting for access to the central resources. The metric and parameter driven model of control used here is based on asynchronous message passing. As in the cooperative system, no data essential to method evaluation is stored at the global control process. This should help to reduce any tendency towards a central bottleneck. The global controller monitors system processing until intervention is necessary. When a control rule is fired, a new set of parameters is sent to the appropriate method or resources are reallocated. The controller then resumes its monitoring task until the next intervention is necessary.

It is possible that the global controller will be distributed throughout the system, e.g. one part for each route, but it is more likely that a single process will exist. This gives a single address for communication and avoids the need for different control processors to communicate with each other.

13.3.1 Feedback

The parallel architecture should support feedback within each of the processing routes (the fusion route may provide feedback to either the intensity or the depth routes). This allows 'later' methods to re-invoke 'earlier' methods with new parameters, e.g. to selectively re-segment some areas of the image. This introduces the need for feedback messages that allow the method to identify that additional or modified information is required.

Feedback messages must be dealt with by the global controller since there is no guarantee that earlier methods are still active, these may have been de-scheduled to allow new methods to be invoked. Upon receipt of a feedback message, the global controller determines whether or not the required method is still available. If it is not, it is rescheduled with new parameters derived from the feedback message.

13.3.2 Intermediate-Level Control

Having stated that there will be one global controller, there may be a case for introducing intermediate-level controllers within the system. If it is known that a method will persist, it should be possible to send feedback messages directly to the method to reduce the load on the global controller. If a group of related methods persist, there may be a case for an intermediate level controller to deal with these. Intermediate control could be based on the concept of an estate manager responsible for coordinating a set of local methods. The estate manager can be used where methods persist and localised re-allocation of resources between methods is possible. For example, an estate manager could be used to control a low-level partition in the processor network. This would deal initially with segmentation issues but as processing progressed, would deal with requests for re-segmentation and verification of pose.

The estate manager could be associated with one of the existing method controllers or, if the computational expense was too high, created as a separate process. In either case, messages to and from the estate manager should be both short and infrequent. As for the global controller, asynchronous message passing will be used.

13.4 Resource Allocation

The initial static allocation of resources may be based on the results of functional prototypes applied to 'typical' input images. This identifies in advance the 'normal' processing requirements of the algorithms in the system. The processing effort required for many vision algorithms depends on the complexity of the input image, this implies that dynamic reallocation of resources may be required to deal with scenes that vary from the 'average' case. Such reallocation of resources must be carried out within the limitations of the implementation platform, this section considers some of the issues involved.

The main costs of resource reallocation on the Meiko are incurred when data is moved between processors. This involves interrupting the processes and setting up the communications, often with some degree of synchronisation between the processes. Additional housekeeping activities are required to ensure that sufficient processing context is available at the reallocated processors and that no potentially useful information is lost when the context switch takes place.

The high cost of distributing image data in the transputer network means that image data will probably be distributed to the appropriate processors *before* low-level processing begins. Any attempt to farm using image data as the task message is doomed to poor performance characteristics unless the processing demands are very high. To avoid moving data during resource allocation and incurring high communication overheads, each processor must have access to both images: this may not be possible given the memory restrictions at each processor. Swapping some of the data to disk is not a viable option given the restricted host to transputer system bandwidth and need to transfer all swap data via a single host connected transputer. Even if it is possible to store both images at all low-level processors, the impact of reallocation on the algorithm control paradigm must be considered.

During the initial low-level processing stages, the processing load is known in advance and geometric decomposition can be used to avoid farm-worker messages. Any attempt to reallocate processors during decomposition involves 'splitting' existing tasks to provide work for the reallocated processor(s). Unless there are sufficient resources to split all tasks, no increase in throughput will be achieved. If process farming is used instead of geometric decomposition, some increase in throughput can be expected, but there will be an increased message passing overhead: for the same number of processors, process farming is less efficient than geometric decomposition when the task size is known in advance.

Once the initial low-level processes have been carried out processing becomes data dependent, some regions of the image will contain more feature points than others. Choudhary and Ponnusamy [16] discuss dynamic run-time data decomposition for iconic data, this uses the number of feature in each processor's data partition to redistribute data and processing. Such feature information will exist at the end of the initial processing phase and can be used to further guide processing. Since the data is still iconic, it is probably worth redistributing the initial data to allow any processor within a given route to examine any part of the image. (This redistribution is also desirable during verification when the low-level partition may be asked to verify that some model feature falls within any specified section if the image.) Allocation of tasks to processors during the data dependent processing can be based on process farming so the reallocation of resources can be viewed as changing a worker's method. Since processing still uses iconic data however, it is necessary to store data from both routes at the worker processes: as identified above, there may not be sufficient memory to allow this.

During intermediate-level processing features identified in the iconic data are grouped to give a more compact feature representation that is suitable for matching, e.g. edge pixels can be grouped to form lines. In the intensity route, it is feasible to extract edge points and farm this out to the grouping processes. In the depth route the density of feature points is higher and it makes some sense to retain the iconic data produced during low-level processing as the input data, the grouping processes can then be directed to examine certain regions of the scene. A processor farm could be used to identify regions of the depth image worth examining and this would allow intensity resources to be added to the processor farm: again it would be necessary to ensure that the intensity processors had access to the depth data.

After intermediate level processing the scene data structures are more compact and can be supplied to each of the matching processors. The model database is placed on each match processor and the amount of data required to effect resource reallocation is considerably lower than during lower-level processing. There are two possibilities for resource reallocation at the match level: between depth and intensity processes examining the same model, and between processes examining different models. In both cases, allocation depends on being able to split a task to provide work to reallocated resources, this requires the transfer of sufficient contextual information to allow the task to proceed. This will be considered further during the development of the matching process(es).

Appendix A

Notes

A.1 Conventions

1) Where an informal algebraic specification is defined, **CAPITAL** letters, in general, refer to abstract datatype names, **Case Capitalised** words refer to external existential entities such as files or file structures (and is also used for type constructors), **lower** case is used for source code function formal parameters. Thus, for instance, **HIPS** refers to a HIPS abstype, **Hips** refers to Hips files and **hips** is used for programme formal parameters.

Bibliography

[1] E. Al-Hujazi and A. Sood. Range Image Segmentation with Applications to Robust Bin-Picking Using Vacuum Gripper. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1313–1324, 1990.

[2] F. Arman and J.K. Aggarwal. Model-Based Object Recognition in Dense Range Images: A Review. *ACM Computing Surveys*, 25(1):5–43, 1993.

[3] W.J. Austin. Message passing control for high level vision algorithms. Technical report, Department of Computer Science, Heriot-Watt University, 1991. Report number HW-91/006.

[4] W.J. Austin, A.M. Wallace, and V. Fraitot. Parallel Algorithms for Plane Detection using an Adaptive Hough Transform. *Image and Vision Computing*, 9:371–384, 1991.

[5] D.H. Ballard. Generalising the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(1):111–122, 1981.

[6] P.J. Besl and R.C. Jain. Three-dimensional object recognition. *ACM Computing Surveys*, 17(1):75–145, 1985.

[7] P.J. Besl and R.C. Jain. Invariant surface characteristices for 3D object recognition. *Computer Vision, Graphics and Image Processing*, 33:33–80, 1986.

[8] P.J. Besl and R.C. Jain. Invarient surface characteristices for 3D object recognition. *Computer Vision, Graphics and Image Processing*, 33:33–80, 1986.

[9] P.J. Besl and R.C. Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:167–192, 1988.

[10] P.J. Besl and R.C. Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:167–192, Mar 1988.

[11] P.J. Besl and N.D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:239–256, 1992.

[12] S.M. Bhandarkar and A. Siebert. Integrating edge and surface information for range image segmentation. *Pattern Recognition*, 25(9):947–962, 1992.

[13] O. Bourdon and G. Medioni. Object recognition using geometric hashing on the connection machine. In *Proceedings of the 10th International Conference on Pattern Recognition*, volume 2, pages 596–600, 1990.

[14] M. Brady, J. Ponce, A. Yullie, and H. Asada. Describing Surfaces. *CVGIP*, 32:1–28, 1985.

[15] J. Canny. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-8:679–698, 1986.

[16] A.N. Choudhary and R. Ponnusamy. Run-time data decomposition for parallel implementation of image processing and computer vision tasks. *Concurrency: Practice and Experience*, 4(4):313–334, 1992.

[17] M.B. Clowes. On seeing things. *Artificial Intelligence*, 2:79–116, 1971.

[18] E. R. Davies. Lateral histograms for efficient object location: speed vs ambiguity. *Pattern Recog. Letters*, 6:189–198, 1987.

[19] E. R. Davies. Application of the generalised Hough transform to corner detection. *IEE Proc. E*, 135:49–54, 1988.

[20] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities.* Academic Press, 1990.

[21] E. R. Davies. Simple two-stage method for the accurate location of Hough transform peaks. *IEE Proc. Comput. and Digital Tech.*, 139(3):242–248, 1992.

[22] S. A. Dudani and A. L. Luk. Locating straight-line edge segments on outdoor scenes. *Pattern Recognition*, 10:145–147, 1978.

[23] O.D. Faugeras and M. Herbert. The Representation, Recognition, and Positioning of 3-D Shapes from Range Data. In A. Rosenfeld, editor, *Techniques for 3-D machine Perception*, pages 13–51. North-Holland, 1986.

[24] M.A. Fischler and R.C. Bolles. Random Sample Concensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. ACM*, 24(6):381–395, 1981.

[25] M.A. Fischler and O. Firschein. Parallel Guessing: A Strategy for High-Speed Computation. *Pattern Recognition*, 20(2):257–263, 1987.

[26] P.J. Flynn and A.K. Jain. BONSAI: 3D Object Recognition Using Constrained Tree Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1066–1075, 1991.

[27] G.B. Forsythe, M.A. Malcolm, and C.B. Moler. *Computer Methods for Mathematical Computations.* Prentice-Hall, 1977.

[28] P.C. Gaston and T. Lozano-Perez. Tactile recognition and localization using object models: The case of polyhedra on a plane. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(3):257–266, 1984.

[29] M. Gay. Feature Identification Codes for LAIRD. Technical Report LAIRD-TR-BAE-MG911022 (Version 3), LAIRD Consortium: British Aerospace Plc, October 1991.

[30] G. Gerig and F. Klein. Fast contour identification through efficient Hough transform and simplified interpretation strategy. In *Proc. 8th Int. Conf. on Pattern Recognition*, pages 498–500, Paris, Oct 1986.

[31] W.E.L. Grimson. On the recognition of curved objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):632–643, 1989.

[32] W.E.L. Grimson, editor. *Object Recognition by Computer: The Role of Geometric Constraints.* MIT Press, 1990.

[33] W.E.L. Grimson and T. Lozano-Perez. Localizing overlapping parts by searching the recognition tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):469–482, 1987.

[34] A. Gupta and R. Bajcsy. Volumetric Segmentation of Range Images of 3D Objects Using Superquadric Models. *Computer Vision, Graphics and Image Processing: Image Understanding*, 3(58):302–326, 1993.

[35] F.R. Hampel, P.J. Rousseeuw, E.M. Ronchetti, and W.A. Stahel. *Robust Statistics: The Approach Based on Influence Functions.* Wiley-Interscience, 1986.

[36] R.M. Harlick, H. Joo, C-N Lee, X. Zhuang, V.G. Vaidya, and M.B. Kim. Pose Estimation from Corresponding Point Data. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1426–1446, 1989.

[37] R. Hoffman and A.K. Jain. Segmentation and Classification of Range Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):608–620, 1987.

[38] J. Illingworth and J. Kittler. The adaptive Hough transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):690–698, 1987.

[39] G.A. Jones. 3D Plane Segmentation from Range Data using the Hough Transform Technique. Technical Report LAIRD-MAN-UOS-GAJ-911201, LAIRD Consortium: University of Surrey, Feb. 1991.

[40] G.A. Jones. Robust Window Fitting: The Limitation of the Least Squares Method. Technical Report LAIRD-TR-UOS-GAJ-910201-1, LAIRD Consortium: University of Surrey, Feb. 1991.

[41] G.A. Jones. HaRASS: Curved Surface Segmentation. Technical Report LAIRD-MAN-UOS-GAJ-930201-1, LAIRD Consortium: University of Surrey, Feb. 1993.

[42] K. Kanatani. *Geometric Computation for Machine Vision.* Oxford Science Publications, 1993.

[43] L. Kitchen and A. Rosenfeld. Grey level corner detection. *Pattern Recog. Letters*, 1:95–102, 1982.

[44] V. Kumar and V.N. Rao. Parallel Depth First Search. Part II: Analysis. *International Journal of Parallel Programming*, 16(6):501–519, 1987.

[45] V. F. Leavers and J. F. Boyce. The Radon transform and its application to shape parameterization in machine vision. *Image Vision Comput.*, 5:161–166, 1987.

[46] D.G. Lowe. *Perceptual organisation and Visual Recognition.* Kluwer Academic Publishers, 1985.

[47] D.G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Journal of Artificial Intelligence*, 31:355–395, 1987.

[48] P. McAndrew and A.M. Wallace. Interpretation of 2d scenes using a general relational model. In *Proceedings of the 3rd ALVEY Vision Conference*, pages 107–115, 1987.

[49] P. McAndrew and A.M. Wallace. Rapid invocation and matching of 3D models to 2D images using curvilinear data. In *Proc. of 3rd Int. Conf. on Image processing and its Applications*, Warwick, UK, 1989.

[50] Patrick McAndrew. *Recognising and locating objects in two dimensional perspective views.* PhD thesis, Dept. of Computer Science, Heriot-Watt University, September 1990.

[51] P. Meer, D. Mintz, and A. Rosenfeld. Least Median of Squares Based Robust Analysis of Image Structure. In *Proceedings DARPA Image Understanding Workshop,1990*, pages 231–254, September 1990.

[52] P. Meer, D. Mintz, A. Rosenfeld, and D.Y. Kim. Robust Regression Methods for Computer Vision: A Review. *International Journal of Computer Vision*, 6(1):59–70, 1991.

[53] J. Moody, P.J. Flynn, and D.L. Cohn. Parallel Hypothesis Verification. *Pattern Recognition*, 26(10):1521–1527, 1993.

[54] D.W. Murray. Model based recognition using 3D shape alone. *Computer Vision, Graphics and Image Processing*, 40:250–266, 1987.

[55] T. Nagata and H.B. Zha. Determining orientationm location and size of primitive surfaces by a modified Hough transform technique. *Pattern Recognition*, 21(5):481–491, 1988.

[56] T.S. Newman, P.J. Flynn, and A.K. Jain. Model-Based Classification of Quadric Surfaces. *Computer Vision, Graphics and Image Processing: Image Understanding*, 28(2):235–249, 1993.

[57] K. Paler, J. Foglein, and J. Illingworth. Local ordered grey levels as an aid to corner detection. *Pattern Recog.*, 17:535–543, 1984.

[58] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. CUP, 2nd edition, 1992.

[59] W.H. Press, B.P. Flannery, S.A.Teukolsky, and W.T. Vetterling. *Numerical Recipes in Pascal*. Cambridge University Press, 1989.

[60] F. Quek, R. Jain, and T.E. Weymouth. An Abstration-Based Approach to 3-D Pose Determination From Range Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(7):722–736, 1993.

[61] V.N. Rao and V. Kumar. Parallel Depth First Search. Part I: Implementation. *International Journal of Parallel Programming*, 16(6):479–499, 1987.

[62] G. Roth and M.D. Levine. Extracting Geometric Primitives. *Computer Vision, Graphics and Image Processing: Image Understanding*, 1(58):1–22, 1993.

[63] M. Rygol, S. Pollard, and C. Brown. MARVIN and TINA: A Multiprocessor 3D Vision System. *Concurrency: Practice and Experience*, 3(4):444–356, 1991.

[64] B. Sabata, F. Arman, and J.K. Aggarwal. Segmentation of 3D Range Images Using Pyramidal Data Structures. *Computer Vision, Graphics and Image Processing: Image Understanding*, 57(3):373–387, 1993.

[65] P. Saint-Marc and G. Medioni. Adaptive Smoothing for Feature Extraction. In *Proceedings of the Image Understanding Workshop*, pages 1100–1113, Cambridge, Mass., Apr 1988.

[66] R.V. Shankar and S. Ranka. Parallel Vision Algorithms Using Sparse Array Representations. *Pattern Recognition*, 26(10):1511–1519, 1993.

[67] R.W. Taylor, M. Savini, and A.P. Reeevs. Fast segmetation of range imagery into planar regions. *Computer Vision, Graphics and Image Processing*, 45(1):42–60, 1989.

[68] E. Thirion and A.M. Wallace. Image synthesis and vision model creation. Technical report, Department of Computer Science, Heriot-Watt University, 1992. LAIRD Reference LAIRD/MAN/ET/920518, contact point A.M. Wallace.

[69] Ting-Jun Fan, G. Medioni, and R. Nevatia. Segmented Descriptions of 3-D Surfaces. *IEEE Journal of Robotics and Automation*, 3(6):527–538, 1987.

[70] Ting-Jun Fan, G. Medioni, and R. Nevatia. Recognising surfaces using surface descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1140–1157, 1989.

[71] T. M. van Veen and F. C. A. Groen. Discretization Errors in the Hough Transform. *Pattern Recognition*, 14(1-6):137–145, 1981.

[72] A.M. Wallace. An informed strategy for matching models to images of fabricated objects. *Pattern Recognition*, 3(20):349–363, 1987.

[73] A.M. Wallace. Matching segmented scenes to models using pairwise relationships between features. *Image and Vision Computing*, 5(2):114–120, 1987.

[74] D. Waltz. Understanding line drawings of scenes with shadows. In P.H. Winston, editor, *The psychology of computer vision*. McGraw-Hill, 1975.

[75] K.G. Waugh and G.J. Michaelson. Parallel process placement from functional prototype instrumentation. In *Proceedings TENCON91. Energy, computer communications and control systems*, volume 2, pages 275–282, 1991.

[76] Z.-Q. Wu and A. Rosenfeld. Filtered projections as an aid to corner detection. *Pattern Recogn.*, 16:31–38, 1983.

[77] N. Yokoya and M.D. Levine. Range Image Segmantation Based on Differential Geometry: A Hybrid Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):643–649, 1989.

[78] H. K. Yuen, J. Illingworth, and J. Kittler. Ellipse detection using the Hough transform. *Proc. 4th Alvey Vision Conf.*, pages 265–271, 1988.

[79] G. Zhang, M. Umasuthan, and A. M. Wallace. An Efficient Outlier Removal Algorithm. In *Proceedings SPIE 92, Non-linear Image Processing*, 1992.