REAL-TIME LINE DETECTION AND LINE-BASED MOTION STEREO

by

Danpo Zhang

B.S.E.E. Tsinghua University, Beijing, China, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in the School
of
Computing Science

SIMON FRASER UNIVERSITY
December 1992

APPROVAL

Name: Danpo Zhang

Degree: Master of Science

Title of thesis: **Real-time Line Detection and Line-based Motion Stereo**

Examining Committee: Dr. Fred Popowich , Chairman

Dr. Ze-Nian Li,
Senior Supervisor

Dr. Brian Funt,
Supervisor

Dr. Binay Bhattacharya,
SFU Examiner

Date Approved: Dec. 15, 1992

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Real-Time Line Detection and Line-Based Motion Stereo

Author:

(signature)

Danpo Zhang

(name)

12-16-92

(date)

ABSTRACT

Recognition of shape is one of the fundamental problems in computer vision. A number of fast line detection and line-based depth recovery algorithms have been developed on various parallel architectures to meet the requirement of real-time robotic vision. This thesis describes a parallel and hierarchical (pyramidal) approach to fast Hough line detection and line-based motion stereo. The lines are represented using their Hough parameters ρ and θ. The processes of line merging and matching are integrated in the pyramidal framework.

To reduce the complexity of the line merging process, the Dynamically Allocated Quadtree (DAQ) is introduced to represent the Hough parameter space. The line merging algorithm using the DAQ is appreciably more efficient than the original algorithm presented by Jolion and Rosenfeld. A line-based motion stereo algorithm is also developed to recover the depth information along linear features in the scene. Live images are obtained from a single static camera and a moving belt. Line segments from the multiple motion stereo images are matched while they are merged hierarchically in the pyramid. It is shown that the problem of matching lines among the multiple images can be effectively carried out as a straight-line detection problem in a well-defined three dimensional parameter space.

Experimental results from the SFU hybrid pyramid are presented for both line detection and line-based motion stereo algorithms. The pyramid architecture capitalizes on advantages of both the mesh and the pipeline architectures. It is demonstrated that the new algorithms combined with the hybrid pyramid hardware environment are suitable for real-time object recognition.

ACKNOWLEDGMENTS

I sincerely thank my senior supervisor Dr. Ze-Nian Li for his supervision and management of my thesis. He has provided valuable advice and guidance throughout this project. I also would like to thank Dr. Brian Funt, Dr. Binay Bhattacharya, for helping with the revision of my thesis, and Dr. Fred Popowich for organizing the thesis defense. I am grateful to Frank Tong and Dr. John Ens for there help when I encountered problems in my projects, and for their useful suggestions. Special thanks to my friend and roomate Bo Wang for his kindness help to formatting this thesis and preparing the defense. Finally, I would like to thank all my friends in the School of Computing Science, Engineering Science who certainly contributed to the work I have done in SFU.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Vision is one of the most powerful perceptual mechanisms of human beings. It can provide us with a great amount of information about the outside world and enable us to interact intelligently with the outside world. It is no wonder that many attempts have been made to give machines a sense of vision since the time when digital computers were invented. On the other hand, human vision is also very complex. It is not surprising, that attempts of giving machines a sense of vision have mostly ended with little success. However, significant progress has been made, mainly in industrial applications in which the "visual environment can be controlled and the task faced by the machine vision system is clear-cut"[1]. A very typical example could be a vision system used for directing a robot arm to do some operations on a moving conveyor belt.

One of the essential goals of a computer vision system is object recognition. To

achieve this goal, it is very important to estimate the properties of the object's surfaces. This surface information can be analyzed through the edge locations of the images, because edges arise from surface discontinuities or from reflectance or illumination boundaries of the object [2].

Symbolic descriptions of edges are often used as building blocks to construct the final description of the sensed objects. For example, a polyhedra object can be interpreted through the line descriptions of its edges. The most widely used symbolic description in computer vision is straight lines, not only because they are the most commonly appeared shape in this world, but also because they are easy to define and interpret by computer vision systems. Line detection is used as the major approach to get the description of straight lines. Lots of research has been conducted in this area [3 - 12].

Line detection techniques are usually applied in the two-dimensional image space. To interpret the three-dimensional world, depth information must be recovered. One of the commonly used methods for depth recovery is *stereo vision* which is implemented by matching the corresponding point of pair of images. Stereo technology has been widely used to recover the depth information [4, 13 - 20]. A detailed discussion of stereo mechanisms will be presented in Chapter 5.

A lot of good line detection and depth recovery algorithms have been developed. But most of them were initially implemented on sequential machines. The implementation of vision algorithms involves processing large two-dimensional arrays of data

on the Von Neumann machine. This kind of data processing is inefficient. It has been known that the operation on image array can be done concurrently on each element of the array, which may remarkably speed up the vision process.

To break the Von Neumann bottleneck and perform the vision process in real-time, different forms of parallel architecture have been proposed [3]. A parallel machine contains concurrently running processors and the communication links among those processors. In [4], Stout defined the *Communication radius* of a parallel machine as the minimum value of $R(p)$ over all processors p in the machine, where $R(p)$ is the largest number of communication links needed from processor p to any other processor in the parallel machine. In the same paper, he also pointed out that for a non-trivial vision problem, "any algorithm must take time proportional to the communication radius".

The mesh strutted computer contains a two-dimensional grid of processors where each processor is connected to its four nearest neighbour, Figure 1.1(a) depicts the mesh parallel processors. The communication radius of the mesh architecture can be achieved at the four center processors, where each of them will take N steps to go to the furtherest corner processor [4].

Since the communication radius of the mesh computer is N, it is apparent that $O(N)$ time complexity is incurred to deal with the $N \times N$ square images on the $N \times N$ mesh architecture. Applications such as the $O(N)$ complexity line-detection algorithm, have been developed on a $N \times N$ mesh architecture [5].

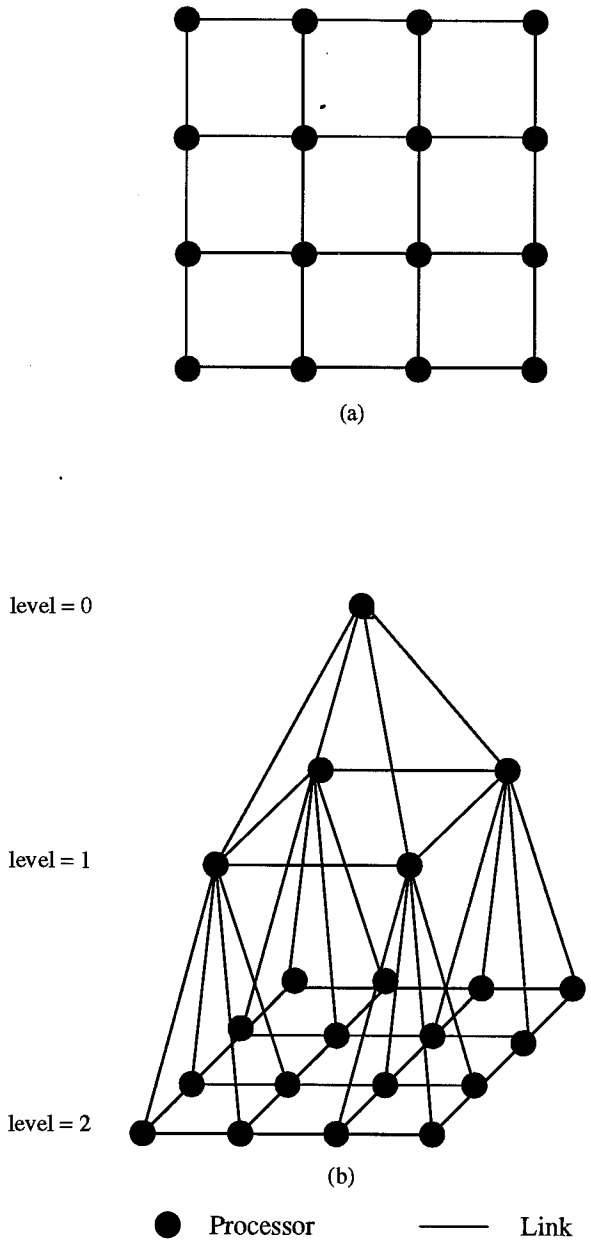

Figure 1.1: Mesh and Pyramid Structure.

The pyramid structure was first introduced by S. Tanimoto and T. Pavlidis in 1975 [6] for parallel image processing. Figure 1.1(b) gives a picture of a 3 level pyramid. From this figure it can be seen that level N of the pyramid is constructed by $2^N \times 2^N$ processors connected as a mesh. Every level has one fourth as many processors as the level just below it, and each non-base processor is connected to four children from the level below. The communication radius of the pyramid is $log(N)$ because it takes $log(N)$ hops for the top level processor to go to any base level processor.

From the communication radius theory, only $O(logN)$ complexity is incurred in processing $N \times N$ square images on pyramidal parallel architectures. Besides that, a pipeline technique is very much suitable for the hierarchical structure of the pyramid. Pipelining can be applied within the pyramid by assigning different functions to the multiple levels of the pyramid.

Since the pyramid architecture capitalizes on the advantages of both the mesh and the pipeline architecture, there have been many attempts of solving the vision problems by using this pyramid architecture. Many basic image operations can be efficiently calculated using this pyramid architecture. As the pioneer of the pyramid structure, Tanimoto and Pavlidis [6] believe that the pyramid structure can a) quickly find a relevant part of the image, and b) ignore irrelevant detail of the original image. P.J. Burt [7] [8] pointed out that one of the primary advantages of pyramid is that " they are hierarchically organized and locally connected." It is easy to find that the

connection between levels of the pyramid serves as a link between pixel-level representation and symbolic/object level representation.

Most parallel machines are organized either in SIMD (Single Instruction Multiple Data) or MIMD (Multiple Instruction Multiple Data) mode. SIMD machines are ideal for iconic data processing because all image pixels are to be processed identically. On the other hand, MIMD architectures tend to work effectively on symbolic data processing because they can concurrently process many different pieces of data independently. To achieve optimum performance at all levels of the vision problem, hybrid architectures consisting of both SIMD and MIMD have been proposed. [9] [10]

Several researchers have suggested the implementation of hybrid pyramidal machines for parallel computer vision [11]. The most ambitious development is the Image Understanding Architecture at the University of Massachusetts-Amherst. The large pyramid will eventually consist of three levels, with 512×512 SIMD processors at the bottom, 64×64 MIMD processors in the middle and 8×8 MIMD processors at the top [12] [9].

At Simon Fraser University, a hybrid pyramid multiprocessor vision machine was built for real-time vision applications [13]. The pyramid has 512 one-dimensional SIMD array processors at the bottom and a 63-node transputer-based multiprocessor system on the top. Parallel hardware links (PARLink) are developed to enable parallel data communications between the array processors and the transputers.

The motivation of this thesis is to develop fast line detection and line-based depth recovery algorithms on the SFU hybrid pyramid vision machine and to demonstrate that these algorithms combined with the hybrid pyramid hardware environment have potential in real-time object recognition.

This thesis first presents algorithms and implementations of fast Hough line detection in the hybrid pyramid. Live input images are preprocessed in the array processors. Transputers on the top merge edge pixels into short line segments and then into longer lines hierarchically in the hybrid pyramid. The lines are represented using their Hough parameters ρ and θ. To reduce the complexity of the line merging process, a Dynamically Allocated Quadtree (DAQ) is introduced to represent the Hough parameter space.

Line detection only gives two-dimensional descriptions of the objects. To interpret the three-dimensional world, stereo technology is used to recover the depth information. To get larger disparity and lower error rate during the matching, *Multiple-baseline stereo* [14] was proposed by using multiple cameras to take more than two snapshots. Since we are concentrated on a manufacturing environment with assembly lines where the belts are moving at a relatively constant speed, instead of moving the camera, multiple snapshots of the moving objects can be taken in rapid succession by a static camera. The controlled belt movement guarantees that any disparity only occurs along the epipolar line. We named this method *Motion Stereo.*

To reduce the complexity of pixel-based corresponding points matching, line-based

stereo matching was introduced. In the second part of this thesis, an algorithm for line-based motion stereo will be introduced. Input data is obtained from a single camera and a moving belt. A parallel and hierarchical (pyramidal) algorithm for line merging and matching is described. It is shown that the problem of matching lines among the multiple motion stereo images can be effectively carried out in a straight-line finding problem in a well defined 3D parameter space.

The organization of this thesis is as follows: The next chapter will give an overview of existing Hough line detection approaches. Chapter 3 describes the architecture of SFU hybrid pyramid vision machine. Chapter 4 presents the fast pyramidal Hough line detection algorithm. Chapter 5 details a line-based motion stereo algorithm. Chapter 6 of the thesis describes our experimental results, and the conclusion as well as discussion of future work are given in chapter 7.

CHAPTER 2

HOUGH BASED LINE DETECTION APPROACHES

2.1 Hough Transformation for Detecting Straight Lines

Detection of straight lines is one of the recurring problems in computer vision and image processing. As a straight line is just a group of collinear points in an image space, the task of finding a line is equivalent to finding that group of collinear points. Many approaches have been presented, among them, one of the most widely used is the Hough Transformation[15].

By using some mathematical transformation, a group of collinear points on the

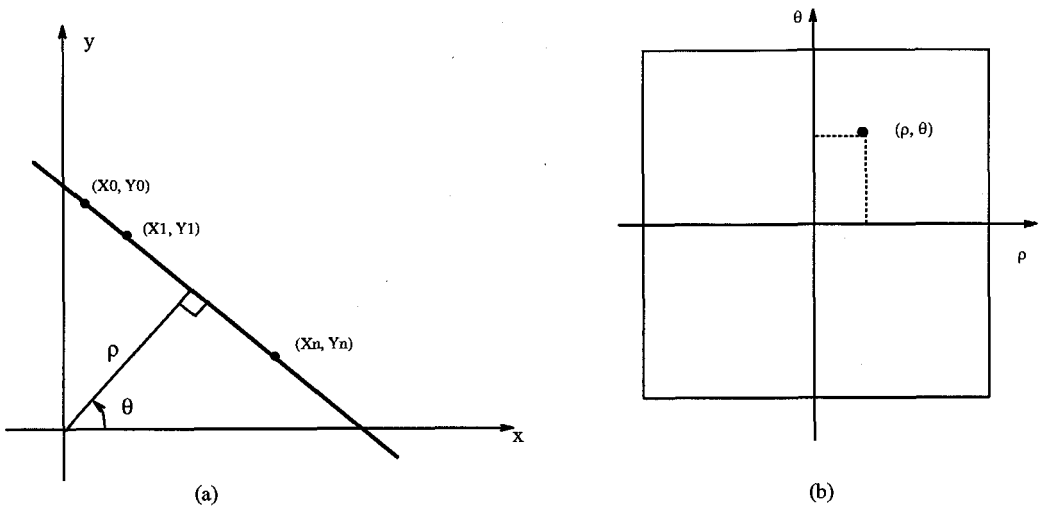

Figure 2.1: Image space and Hough space.

image space can be translated into a single point on the transformed space (often referred to as *parameter space*). This parameter space can be represented as an integer array to accumulate the transformed points. After applying the mathematical transformation, each group of collinear points (edge pixels) will be mapped into a "peak" point in the parameter space. This reduces the image-based line finding problem to a peak detecting problem in the parameter space. The reason for adopting this transform-based line detection approach is that detecting peaks is usually much easier than detecting collinear points.

Any 2D line in the image can be represented uniquely by two parameters such as slope-intercept, intercept(x)-intercept(y), etc. In [15], a straight line is specified by the angle θ of its normal and its algebraic distance ρ from the orgin. The equation of this transformation is: $\rho = xcos\theta + ysin\theta$. From Figure 2.1 it can be seen that the collinear points (x_0, y_0), (x_1, y_1), ... (x_n, y_n) share the same $\rho - \theta$ value in which:

$$\rho = x_0 cos\theta + y_0 sin\theta,$$
$$\rho = x_1 cos\theta + y_1 sin\theta,$$
$$\cdots$$
$$\rho = x_n cos\theta + y_n sin\theta.$$

Thus these collinear points are transformed into a single point in the $\rho - \theta$ parameter space, which is also known as the *Hough Space*.

2.2 Improvement of the Hough Transformation

The traditional Hough Transformation implementation involves the following steps:

- **Step0** Parameter space is defined at a proper quantization and represented by an integer array *para_space*. Initialize all the elements of *para_space* to zero.
- **Step1** Scan the edge image, for each edge pixel (x_i, y_i) with the orientation θ calculate: $\rho = xcos\theta + ysin\theta$. Update the accumulate array *para_space*: $para\_space[\rho][\theta] = para\_space[\rho][\theta] + 1$.
- **Step2** Scan the parameter space *para_space*. Detecting every peak point (ρ_i, θ_i). To avoid excessive number of lines or noise being detected, a practical threshold K can be set.

We can find there are two scanning processes involved in the traditional Hough transformation implementation. The operation of 2D array scanning is very costly

and these two scanning processes comprise the bottleneck of the implementation of a fast Hough line detection.

Various approaches have been taken to deal with the above problems. Illingworth and Kitttler [16] provide an extensive survey on various Hough techniques. Among these techniques, parallel architectures are widely accepted for concurrently processing the elements of arrays. More recently, a number of fast Hough transform algorithms have been developed for different parallel architectures including the shared-memory [17], mesh-connected arrays[5][18], hypercube[19] and pyramid[20].

From the discussion of communication radius of the parallel machine, we know that $O(N)$ can be reached by using mesh connected architecture. Cypher et. al. [5] developed an $O(N)$ complexity Hough Transform on $N \times N$ mesh-array architecture in 1987. Basically, Hough Transform is implemented on mesh-connected processors by rotating the columns of the image until all the collinear pixels (pixels in same band) for a given projection angle are moved into the same row of processors. After horizontal shift and add, collinear points with this projection angle can be calculated. $O(PN)$ complexity is needed while P is the number of projections to be calculated. Improvement was made so that $O(N + P)$ complexity can be achieved [5].

To achieve higher performance, pyramid architectures can be used since they have a smaller communication radius. Actually, $O(logN)$ algorithms were developed on pyramid architectures [20] [10].

Researchers also found out that the overhead of the second scanning process (peak finding) could be reduced by designing some efficient peak searching mechanisms. Li and Lavin developed a fast Hough Transform algorithm based on bintree data structure [21], the parameter space is represented with Bintree. To reduce the searching time, an algorithm was developed on bintree. It has the flexibility of searching the parameter space along each dimension independently. Illingworgh and Kittler introduced their Adaptive Hough Transform mechanism in 1987 [22]. Instead of just using one fine-grained Hough space, they defined a multiple level coarse-to-fine Hough space and voted simultaneously on these multiple spaces. The effort of Hough space searching can be greatly reduced by applying an efficient coarse-to-fine searching strategy on these multiple Hough spaces.

The advantage of parallel architecture is that it is possible to divide the whole image array into various parts and let the parallel processors process these parts. However, another problem appears: Each processor only gets a relatively small part of the image, this image is still mapped into the whole parameter space and the searching for full-parameter space is still needed. For a parallel architecture, it is not economical to store the whole parameter space (represented as a multi-dimensional array) in each processors's local memory and to process that array.

To alleviate this dilemma, a new approache has proposed by Jolion and Rosenfeld in 1989 [20] which essentially does parallel and hierarchical line merging so as to avoid the use of any accumulator arrays and to fully utilize the power of the parallel architecture.

A small block of image is first examined by the base processors of the pyramid. Since this small block of image may not contain too many edge pixels, they will form a sparse set of parameter-space pixels. The list of these pixels will be checked and grouped into some clusters which will then be sent to the parent processors. Each higher level processor will recursively group the clusters it receives from its children. This processing work only concerns the cluster of parameter pixels rather than mapping the pixels into the full-size parameter space array. Implementations of this algorithm can be found [20] [10].

Our first simple algorithm is similar to [20], in which a time complexity of O(log N) is claimed where N is the size of the image. However, this simple algorithm places a severe limitation on the number of lines K, which each processor may process. Additionally, a major assumption is that the algorithm is implemented on a massive 3D pyramid with a 2D array processor at its base that is nearly as large as the image size. The algorithm will slow down drastically if a large K is allowed while working with a smaller pyramid such like our short (half-scale) pyramid.

To reduce the complexity of the simple line merging algorithm, a Dynamically Allocated Quadtree (DAQ) is introduced to represent the Hough parameter space. A detailed discussion of this DAQ based line detection algorithm is presented in Chapter 4.

CHAPTER 3

A HYBRID PYRAMIDAL VISION MACHINE

Since pyramid structure supports multiresolution approaches and capitalizes on the advantages of both mesh and pipelined, it is suitable for parallel computer vision [7, 8, 23].

The transputer is a suitable processing element for a pyramidal machine because of its flexibility as a MIMD(multiple-instruction multiple-data) processor. However, implementing a full pyramid of transputers is too expensive and not necessary. In most cases, the processes applied at the bottom of the pyramid are just kernel convolutions for edge detection, or other similar filtering which can be done efficiently on SIMD(single-instruction multiple-data) processors. A good solution is to combine these two types of the machine together, which yields a hybrid pyramidal machine [24].

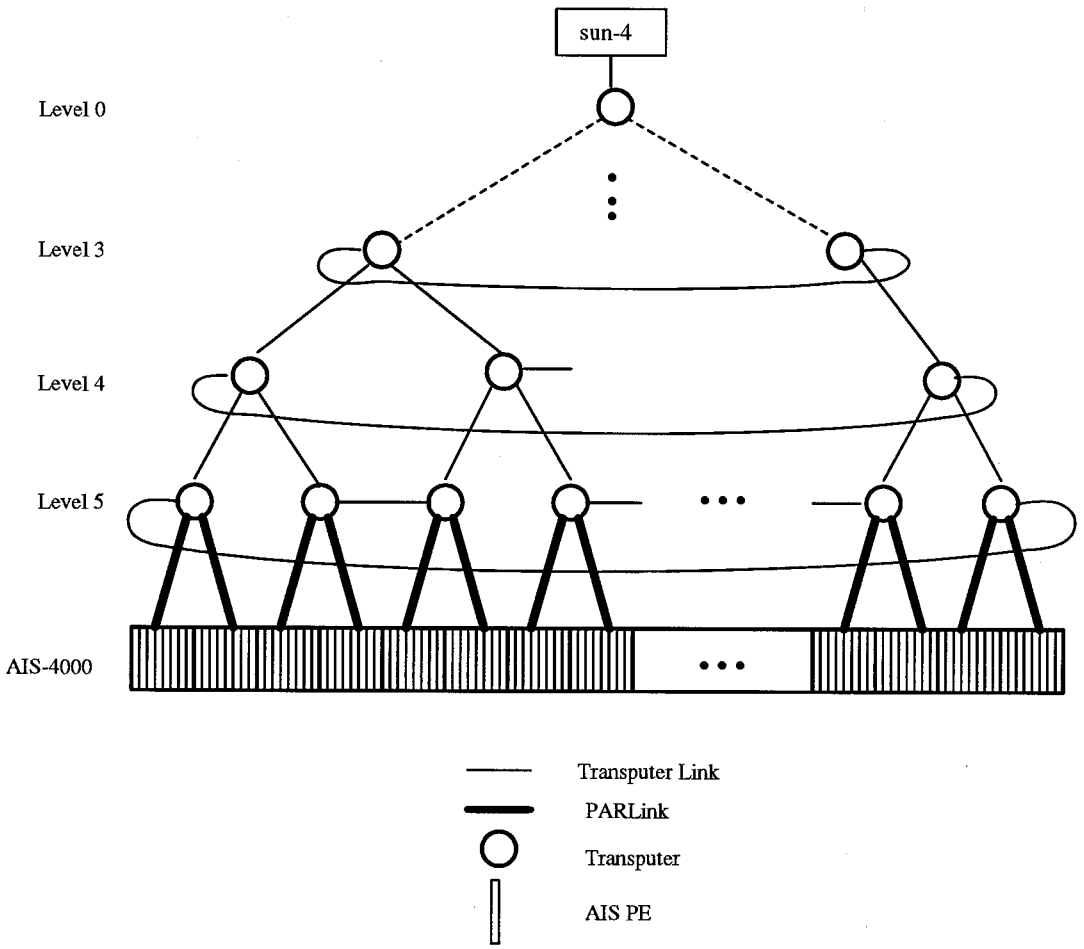

Figure 3.1: The SFU hybrid pyramid vision machine.

Figure 3.1 depicts the SFU hybrid pyramid vision machine. The AIS-4000 has fine-grained parallelism with 512 processors arranged in a single-instruction multiple-data (SIMD) 1-D array architecture. It can readily simulate a 2-D array and handle over 3 billion operations per second. Each of the sixty three transputers is a T-800 with 2 MBytes memory. Three of the four links of each T-800 node are used to form a binary tree. The remaining single link of each T-800 is connected to a programmable cross-bar switch which provides flexibility for additional desirable connectivity. For now, it is used to form horizontal links between the cousin nodes to augment the binary tree as shown in Figure 1. The single root node at level 0 interfaces with a host Sun-4 workstation. The AIS-4000, at the bottom layer of the pyramid, acquires live images and performs lower level processing, with one processor working on each column of the image, whereas the transputers which form the top of the pyramid perform higher level image processing and analysis.

The key to combining the AIS-4000 and the transputer nodes into a pyramidal architecture is a high speed communication link between the AIS-4000 and the transputers. After some initial study, the use of any single fast link was ruled out because of its limitation of data transfer rate, and more importantly, its inability to transfer data to more than one transputer (or AIS processor) simultaneously.

A parallel link, called PARLink was built in the Parallel Vision lab at SFU [13]. The AIS-4000 has eight processors packaged into one custom VLSI chip called a Pixie. Each Pixie chip interfaces to one 32K by eight bit static RAM chip for local image storage. Since the AIS-4000 has 512 SIMD nodes, this means there are 64 clusters of

Pixie and RAM chips. This architecture lends itself to the construction of 64 parallel links connecting each cluster up to the transputer pyramid. Two of these links are connected to one transputer node for a binary tree configuration. The transputers are ordinarily connected with 20 Mbit/sec serial links according to an INMOS protocol. Therefore the hardware requirements for each PARLink are to read eight bit parallel data from the static RAM chips of the AIS-4000 and translate this into the INMOS serial format. This requirement is reversed for communication from the transputers to the AIS-4000. The core of each PARLink is the INMOS C012 chip, with only a few additional buffers and control signals required. All PARLinks are controlled by some custom microcode subroutines supplied by the AIS Inc. Connections are made directly to the data lines of the static RAM chips. Control signals are obtained from the AIS-4000 digital I/O channels as well as direct connections to the microcontroller. A throughput of 1 MByte/sec can be achieved for each PARLink, which allows an image to be passed in 4 msec, with a total bandwidth of 64 MBytes/sec for all of the PARLinks.

The completion of the PARLink enabled the integration of two parallel subsystems, i.e. the AIS SIMD array processor and the MIMD transputer network, into a hybrid pyramid. The massive parallelism characterized by the array processors is naturally embedded in this pyramid machine, because each level of the pyramid is itself an array. Moreover, more interesting and complex algorithms can be implemented in the hybrid pyramid, since the transputers are powerful computing machines and they operate in a hierarchical MIMD mode.

CHAPTER 4

FAST PYRAMIDAL LINE DETECTION ALGORITHM

The pyramid vision machine supports the multiresolution approaches and capitalizes on advantages of both the mesh and the pipeline architectures. For a real-time system in which image data come in from the bottom continuously, this multilevel bottom-up information flow and abstraction can provide substantial speedup.

For example, the pyramidal Hough algorithm for line detection by Jolion and Rosenfeld [20] can readily be implemented in this fashion. Briefly, each transputer at the bottom level of the pyramid examines a vertical stripe of the edge image and merges edge pixels into short lines. The short lines are then passed up to the parent nodes and merged recursively up in the pyramid. A practical threshold K for the number of lines reported by each transputer can be set to avoid excessive number of

short lines and noise being detected.

4.1 A Simple Line Merging Algorithm

The central part of the Jolion and Rosenfeld [20] pyramidal line detection algorithm is line merging. The original paper uses parameters ρ, θ and two L-coordinates λ's to represent line segments and to calculate distances between them. As noted in Jolion and Rosenfeld [20]: "this representation treats collinear segments as a single segment even if they are far apart." It is because under that representation checking whthin the vicinity of two line segments with slightly different θ's is a bit complicated. In our prototype implementation, a modification is made so that coordinates of the end points are used to check the vicinity of the line segments before any merging process can take place. The coordinates of the end points are also used for calculating distances between the line segments, which involves only simple linear equations for the calculation of point-to-line distance. Hence, the use of computationally expensive trigonometric functions as in [20] is mostly avoided. The memory requirement is slightly increased, instead of two L-coordinates, four coordinates are now stored for each line segment. the parameters ρ, θ are still used, but only for the purposes of quick checking and indexing to the dynamically allocated Hough space as shown later. The following explains our modified version of the original line merging algorithm.

4.1.1 Line Segment Representation

A line segment S is denoted by a pair of Hough parameters (ρ, θ) and coordinates (x_1, y_1, x_2, y_2) of its end points P_1 and P_2. A line having an orientation of $45° \leq \theta < 135°$ or $225° \leq \theta < 315°$ is considered as x-major, otherwise, it is y-major. Another parameter *num* records the current number of pixels on the line segment. In summary, a line segment is represented by:

$$S = (\rho, \theta, x_1, y_1, x_2, y_2, num, flag).$$

4.1.2 Vicinity Checking of Line Segments

Use the x-major line segments S and S' in Figure 4.1 as an example, where x_1, x_2, x_1', and x_2' are the x-coordinates of P_1, P_2, P_1', and P_2', and $x_2 > x_1$, $x_2' > x_1'$. If $(x_1' - x_2) > \tau$ or $(x_1 - x_2') > \tau$ (e.g., $\tau = 32$ in 512×512 images), then S and S' are far apart. Similar measures are developed for vicinity checking of y-major lines, or x-major and y-major lines. Line segments far apart are not allowed to merge even if they have very similar ρ and θ.

4.1.3 Collinearity Checking of Line Segments

The collinearity of two similarly oriented line segments is determined by simply measuring the maximum distance between them. Consider $S \subset L$ and $S' \subset L'$, the distance between the two line segments is defined by:

$$d(S, S') = max[d_1(P_1, L'), d_2(P_2, L'), d_1'(P_1', L), d_2'(P_2', L)],$$

where P_1, P_2, and P_1', P_2' are end points of S and S', and d_1, d_2, and d_1', d_2' are their distances to the other line, respectively (see Figure 4.1).

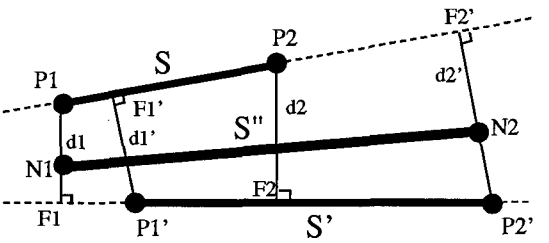

Figure 4.1: Detection of collinearity and merging

4.1.4 Simple Merging Algorithm

Following is the detail of simple merging algorithm for pyramidal line detection:

ALGORITHM Simple-Merging(S, S')

IF two line segments S and S' have almost identical θ and ρ

THEN trivial-merge;

ELSE if their θ's differ too much

THEN trivial-reject;

ELSE /* Collinearity checking */

IF $d(S, S')$ is less than a pre-determined threshold

/* non-trivial merge */

THEN merge S and S' into a new line segment S";

END{Simple-Merging}

Figure 4.1 shows an example of merging x-major lines. The relative position of the new end points N_1, N_2 of $S"$ with respect to S and S' is affected by the numbers of the pixels (num and num') on S and S'. Namely, the line with a larger num will possess a heavier weight and hence pull the new end point toward it. For example, N_1 is between point P_1 and its foot-of-normal F_1 to line S'. Let (x_1, y_1) and (x_{10}, y_{10}) be the coordinates of P_1 and F_1, (x, y) for N_1 can be derived as:

$$x = \frac{\text{num}}{\text{num} + \text{num}'} x_1 + \frac{\text{num}'}{\text{num} + \text{num}'} x_{10}$$

$$y = \frac{\text{num}}{\text{num} + \text{num}'} y_1 + \frac{\text{num}'}{\text{num} + \text{num}'} y_{10}$$

Similar calculation can be applied to P_2-F_2, P_1'-F_1', and P_2'-F_2' to generate a total of four candidates for N_1 and N_2. N_1 is the one that has the smallest x-coordinate value among the four candidates and N_2 the largest. The sum $num + num' = num"$ is recorded as the total number of pixels on $S"$ which is approximately equal to its length. Finally, a pair of ρ, θ for $S"$ is calculated using N_1 and N_2. Similar methods will be used for other situations, i.e., merging y-major lines, or merging x-major and y-major lines. It is shown in [20] that the accuracy of ρ and θ is improved when a merged line segment becomes longer and longer.

It should be emphasized that the cost of trivial-merge or trivial-reject is minimal. If two lines cannot be trivially merged or rejected, then the collinearity checking had to be invoked which is computationally expensive.

The Jolion and Rosenfeld paper [20] claims that their line merging algorithm has a time complexity of O(log N), where N is the size of the image. Besides the severe

limitation on the number K of lines allowed to be reported by each processor, a major assumption is that the algorithm is implemented in a massive pyramid with a 2D array processor at its base that is nearly as large as the image size, i.e., the height H of the pyramid is not much less than $M = \log_2 N$. While working with a smaller pyramid, e.g., the short "half-scale") pyramid as ours, the algorithm will slow down drastically. Assume the image has a resolution $2^M \times 2^M$ and the base of the pyramid has $2^H \times 2^H$ nodes, then $h = M - H$. If h is not small, then the leaf nodes are overloaded and become the bottleneck.

Several effective and flexible pyramidal pipeline techniques were developed and proved especially suitable for our 2D pyramid [25]. Under the flexible pipelining, processors at each level can have relatively balanced load. However, the increase of the speed has a upper bound of 100%, because there are not quite 100% more parent nodes that can share the work of the leaf nodes in a binary tree configuration.

The simple merging algorithm is implemented and the results will be shown in the Chapter 6. As some of the timing results show, when the number of lines in the test image increases, the time for line detection increases quadratically. This is because the worst case time complexity for merging two groups of lines is O(PQ) where P and Q are numbers of lines in these two groups. The algorithm will be especially slow if each pair of the lines must undergo a collinearity check.

4.2 An Improved Line Merging Algorithm Using DAQ

As pointed out in the previous section, the original simple line merging algorithm has several drawbacks. One of its major problems is the unstructured organization of the partially detected line segments. When a parent processor attempts to merge partial results from its two child processors, it has to deal with two potentially long linked lists. Since lines are represented by points in the Hough parameter $(\rho - \theta)$ space, a potential cure is to partition the parameter space into subspaces of smaller sizes. For instance, if the number of the lines in each of the two child parameter spaces can be split evenly into M subspaces, the worst case time complexity will become $M \times (P/M \times Q/M) = PQ/M$. Nevertheless, in order to quickly construct and traverse the subspaces, an efficient and dynamic indexing mechanism for the Hough space must be developed.

4.2.1 The Dynamically Allocated Quadtree (DAQ)

Quadtree has become a popular data structure. Its various applications and variations were reviewed extensively by Samet in his 1984 paper [26]. A quadtree for representing the Hough parameter space is a tree where each node represents a certain range of the parameter space $(\rho_{min}, \rho_{max}, \theta_{min}, \theta_{max})$. A finer resolution of this parameter space can be found in 4 children of this node, each child represents one of four subspaces of the current node:

$$\text{Child a: } (\rho_{min}, \tfrac{\rho_{min}+\rho_{max}-1}{2}, \theta_{min}, \tfrac{\theta_{min}+\theta_{max}-1}{2})$$

Child b: $(\rho_{min}, \frac{\rho_{min}+\rho_{max}-1}{2}, \frac{\theta_{min}+\theta_{max}+1}{2}, \theta_{max})$

Child c: $(\frac{\rho_{min}+\rho_{max}+1}{2}, \rho_{max}, \theta_{min}, \frac{\theta_{min}+\theta_{max}-1}{2})$

Child d: $(\frac{\rho_{min}+\rho_{max}+1}{2}, \rho_{max}, \frac{\theta_{min}+\theta_{max}+1}{2}, \theta_{max})$

An index to the quadtree that contains a string composed of characters a, b, c, and d can be used for each node is showed in Figure 4.2. Root (level 0) node has an empty string of index, while 4 children of the root (level 1) have index a, b, c, d, respectively. The nodes at the next level (level 2) will have index $aa, ab, ac, ad, ba, ..., dc, dd$. Thus, a unique index is associated with each node for fast access of a certain node in the tree. A list of line segments is allocated within each leaf node of the quadtree, whereas no line segment is associated with any internal tree nodes.

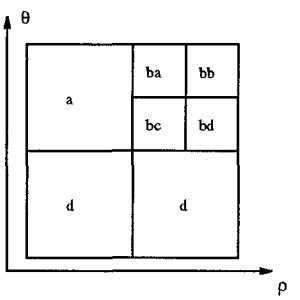

Figure 4.2: Indexing the DAQ

In their early papers [27] [28], O'Rourke and Sloan presented Dynamically Quantized Hough Spaces and Dynamically Quantized Pyramids. Both focused on the quantization problem, i.e., using a limited number of cells to represent the parameter space so that fine precision is maintained where it is needed. In this thesis, quadtrees are used for different purposes, namely, they are used for dynamic construction of Hough subspaces and, subsequently, for quick indexing in the merging process. We call them Dynamically Allocated Quadtrees (DAQ's) because they are dynamically

constructed by the transputer leaf nodes. We always equally divide a 2D Hough space into four quadrants when it is necessary to split a DAQ node. In this way, an efficient algorithm can be developed for combining two DAQ's at the parent node.

4.2.2 Building initial DAQ's

The following recursive procedure INSERT is performed by the transputer leaf nodes in the pyramid when they combine edge pixels into short line segments and build the initial DAQ's for these line segments in their subimages. Max-depth is a adjustable parameter which defines the maximal depth of the DAQ. Since the collinearity checkings for non-trivial merges are only performed among line segments at max-depth level of the DAQ, the insertion routine is reasonably fast.

It should be pointed out that the recursive procedure INSERT provides a concise description. In real implementation, some modifications/optimizations can make the procedure more efficient.

Procedure INSERT (S, R)
Use the parameters ρ and θ of S to traverse the DAQ from the root R downwards, until it encounters a leaf node Q of the DAQ

IF Q is a virtual node /* No line resides in it yet. */
THEN leave S in node Q;
ELSE /* Q is an actual node, there are lines in it already. */
IF S has almost identical θ and ρ with any line S' already in Q

THEN trivial-merge;
ELSE IF current-level = max-depth;
THEN IF S is "collinear" with line S' already in the Q
THEN non-trivial merge;
ELSE link S into the list of the lines at Q;
ELSE /* split Q */
generate 4 virtual child nodes for Q;
for all lines $NewS \in \{S$, all lines in $Q\}$
INSERT $(NewS, Q)$;

END{INSERT}

Figure 4.3 illustrates a sequence of insertions taken place in a transputer leaf node. Numbers in boxs indicate ρ and θ of lines. Initially, the root of the DAQ is a virtual node which has no line associated with it yet. A line (60, 250°) is inserted at Step 1. Step 2 sees the second inserted line which causes the split of the root node. A new line (61, 250°) is trivially merged with the old line (60, 250°) at Step 3/3.1, and a new longer line (60, 250°) is generated. At Step 4, another line (12, 170°) is inserted. Because its ρ value 12 is fairly close to ρ value 40 of the previous line in node b, new children nodes are recursively generated until the two lines reach the deepest allowable level (assume max-depth = 4) and fit in nodes bccc and bccd. Finally, the last inserted line (36, 171°) finds its place at node bccd at Step 5/5.1. Since it is almost collinear with line (40, 170°), the two lines are merged into a new line (39, 170°).

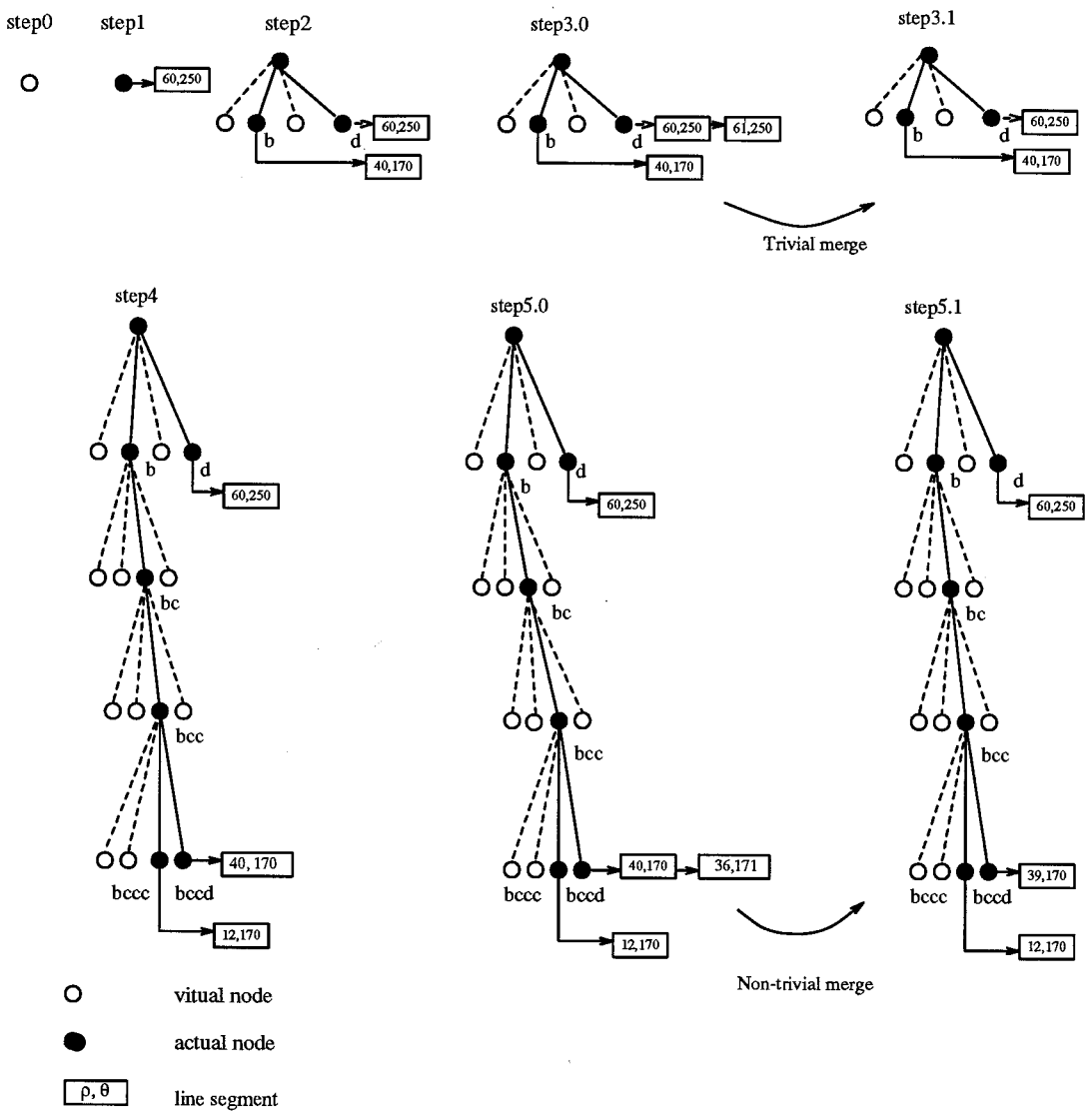

Figure 4.3: Building a DAQ at a transputer leaf node

4.2.3 Combining two DAQ's

Procedure COMBINE (DAQ1, DAQ2)

Traverse DAQ2 and examine each leaf node Q_2 in sequence;

FOR each Q_2

IF there is a (real, virtual or internal) node Q_1 in DAQ1 at the corresponding position as Q_2 in DAQ2

THEN $Q = Q_1$;

ELSE

find Q_2's youngest parent node Q_2' that has

a corresponding node Q_1' in DAQ1;

$Q = Q_1'$;

FOR all lines S in Q_2

INSERT (S, Q);

END{COMBINE}

The function of a parent node is to combine the two DAQ's from their two children to generate a new DAQ. In the combining process, collinear short line segments are merged into longer lines. Suppose there are two trees DAQ1 and DAQ2, an efficient algorithm is to first traverse only one tree, say DAQ2, and examine each leaf node Q_2 in sequence. The procedure INSERT is used to place the lines in Q_2 into their corresponding nodes in DAQ1. As a result, DAQ2 is appended to DAQ1.

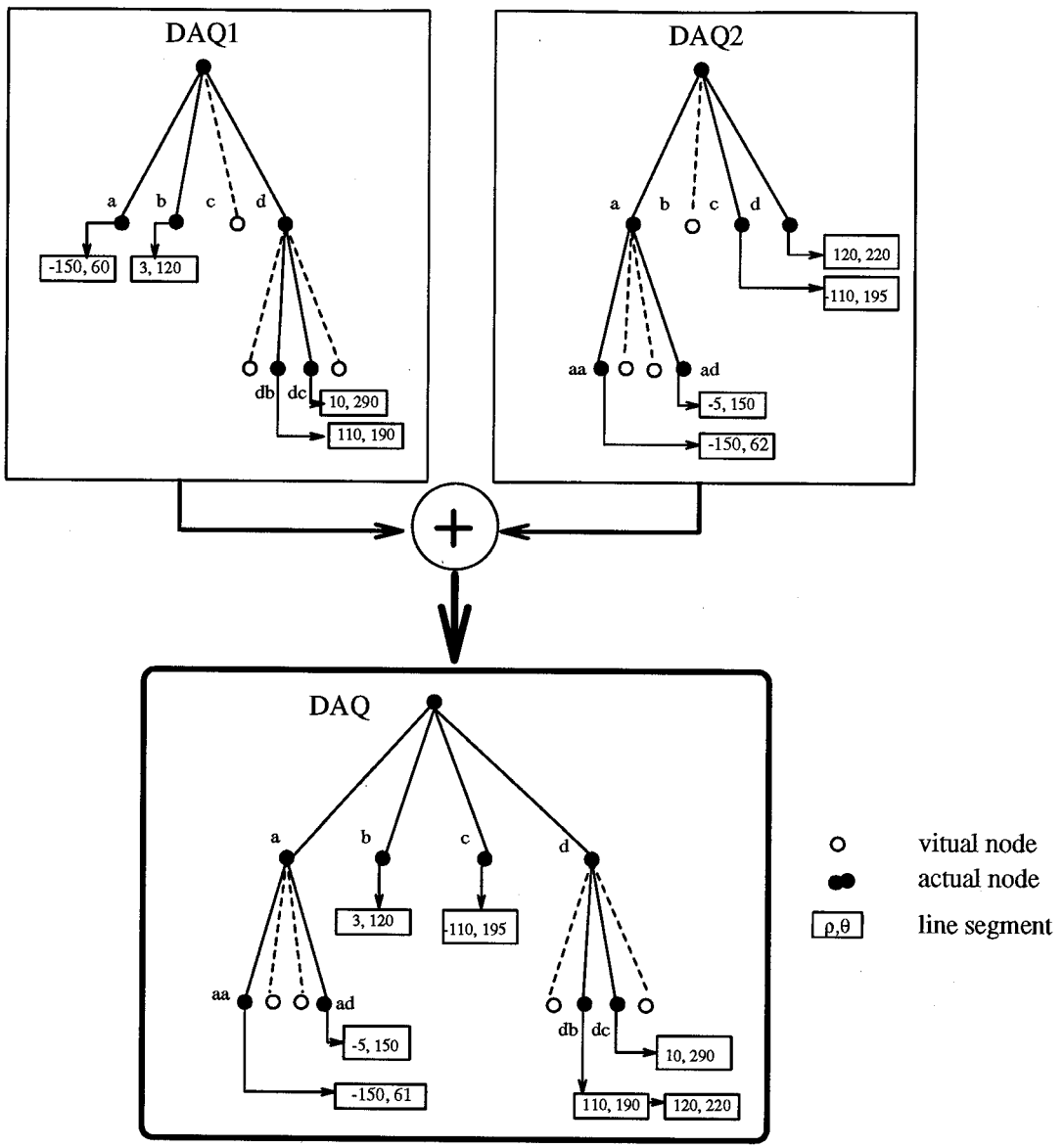

Figure 4.4: An example of combining two DAQ's

Figure 4.4 shows the combination of two DAQ's at a parent node. DAQ1 and DAQ2 from its two children will be combined into DAQ. Leaf nodes in DAQ2 are examined. Node aa in DAQ2 does not have a corresponding node in DAQ1, so the youngest parent node a in DAQ1 is found and line (-150, 62°) is inserted into node a, which causes a trivial merge of lines (-150, 60°) and (-150, 62°) into a new line (-150, 61°). (The new line stays in node a for now, this intermediate result is not shown in the figure.) Similarly, line (-5, 150°) in DAQ2 does not have a corresponding node in DAQ1 and is inserted to node a in DAQ1, which cause the split of node a as shown in Figure 4.4 Line (-110, 195°) in DAQ2 finds its corresponding virtual node c in DAQ1, and is inserted into it. Finally, line (120, 220°) is inserted to node d in DAQ1, which causes it to be linked to the other line (110, 190°) in node db.

4.2.4 Choosing the Depth of DAQ

The choice of the value for max-depth of the DAQ is an important issue. For all practical purposes, max-depth = 8 is sufficiently deep for a very fine partitioning of the Hough parameter $(\rho - \theta)$ space. One of the concerns is the overhead for building the DAQ, especially when it is deep. Experiments are conducted by varying max-depth at multilevels in the pyramid. Our results in Chapter 6 will show that the overhead for building a deep DAQ is low and acceptable and has a very good payoff.

Another concern is the detectability of lines in the image. If deep DAQ's are maintained at all levels in the pyramid (including the root transputer), then the Hough space is always divided into many fine subspaces. Excessive number of lines

might be reported by the root transputer, even if many of them only deviated by a tiny $d\rho$ and/or $d\theta$. It is because they are separated by too many boundaries of fine divisions. A suggested method is to reduce the max-depth gradually while the line merging process ascends the transputer pyramid. This will greatly improve the line detectability while maintaining the effectiveness of the DAQ algorithm. In real implementation, the four quadrants of each Hough subspace are overlapped to prevent mergable lines from being separated by the boundaries of the adjacent quadrants.

CHAPTER 5

LINE-BASED MOTION STEREO ALGORITHM

We use our left and right eyes to judge the depth of what we see because the images captured by our left eye are different with those captured by the right eye. Moreover, the correspondence problem is defined as to select a particular location on a surface in the scene from one image and to identify that same location in the other image [2]. This technology can also be applied to the computer vision systems. *Stereo Vision* is used for depth recovery in which pairs of images from horizontally and/or vertically displaced camera are used.

5.1 Stereo Disparity

A simple example can illustrate the mechanism of how to recover the depth information from a pair of images. Figure 5.1 shows two cameras where their optical axes are parallel and separated by a distance of b. The line connecting two lens centers is defined as the *Baseline*. The baseline is perpendicular to the optical axes of both lens. The correspondence points in the left and right images lie somewhere on a particular line, because these two points have the same y-coordinate. This line is defined as the *Epipolar line*. Let the coordinates of the point P be (X, Y, Z) and the image coordinates in the left and right images be (X_l, Y_l) and (X_r, Y_r). Then we have:

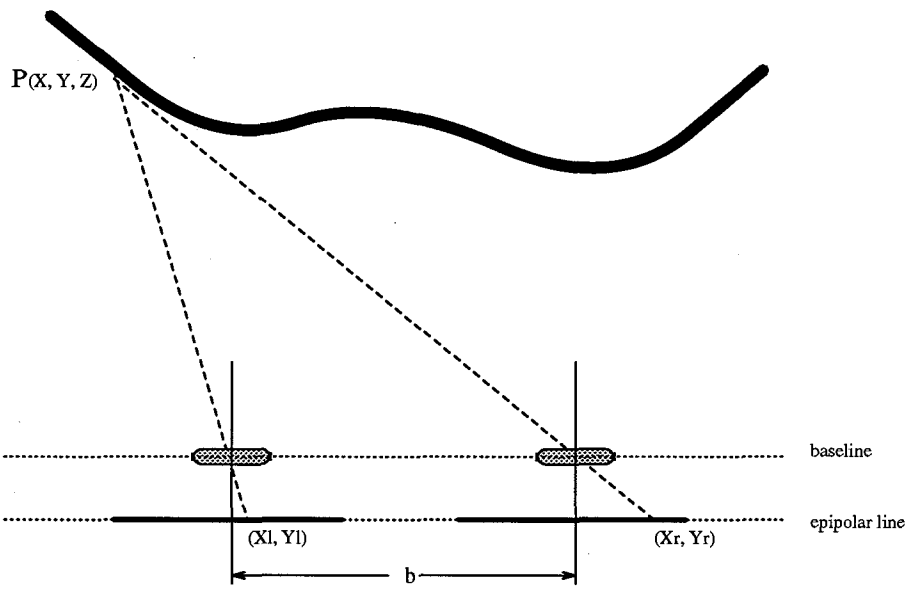

Figure 5.1: Recover Depth from A Pair of Images

$$\frac{X_l}{f} = \frac{X + b/2}{Z} \quad \text{and} \quad \frac{X_r}{f} = \frac{X - b/2}{Z} \tag{5.1}$$

where

$$\frac{Y_l}{f} = \frac{Y_r}{f} = \frac{Y}{Z}, \tag{5.2}$$

f is defined as the distance from the lens center to the image plane. The coordinates X, Y, Z can be solved from the equations 5.1 and 5.2:

$$X = b\frac{(X_l + X_r)/2}{X_l - X_r}, \quad Y = b\frac{(Y_l + Y_r)/2}{X_l - X_r}, \quad Z = b\frac{f}{X_l - X_r}. \tag{5.3}$$

The coordinate Z can be used as the depth information. From equation 5.1, we get:

$$X_l - X_r = \frac{bf}{Z}. \tag{5.4}$$

Disparity is defined as the difference in the image coordinates $(X_l - X_r)$. It is easy to see that the depth is inversely proportional to disparity.

5.2 Some Existing Stereo Approaches

As we have discussed above, the central part of depth recovery using the stereo method is to identify the corresponding points in the pair of images. Once the corresponding points in the pair of images are identified, their disparity values can be calculated and be used to recover the depth information.

To formulate the correspondence computation, its basis in the physical world must be examined and two constraints were identified [2]:

- *Uniqueness:* "No more than one disparity value can be assigned to each item from the image."
- *Continuity:* "Disparity varies smoothly, since the surfaces of a object are generally smooth compared with their distance from the viewer."

To solve the stereo vision problems, the above constraints are usually inadequate. The constraint of *Figure continuity* is proposed by Mayhew and Frisby in 1981 [29] in which they stated, "... the edges of surfaces and surface markings such as lines and blobs will be spatially continuous and that is the ultimate justification for relying on figural grouping rules to guide binocular combination." In other words, disparities are continuous along the edges of the surfaces. Since then, this has been accepted as a more powerful disparity-continuous constraint.

The correspondence problem is known to be very difficult. As Okutomi and Kanade had pointed out in [14], the distance between the pair of cameras greatly affects the accuracy and error rate of the correspondence process. A small distance (baseline) will provide less precision in the depth estimate due to narrow triangulation, whereas a longer distance (baseline) indicates a larger disparity range and hence higher error rate due to false matchs. To alleviate the dilemma, they proposed the *multiple-baseline stereo* method where different baselines are generated by lateral displacements of a camera.

We are dealing with a manufacturing environment with assembly lines, where the

belts are moving at a relatively constant speed. Instead of moving the camera, multiple snapshots of the moving objects will be taken in rapid succession. The controlled belt movement guarantees the disparity only occurs along the epipolar line. We named this method *Motion Stereo.*

In order to reduce the complexity of pixel-based matching, line-based stereo matching algorithms have been developed. Medioni and Nevatia [30] used linear features in their work. In 1988, McIntosh and Mutch [31] presented a system for matching straight lines. Multiple (eight) features were used. These lines failed to match only when three or more of their features differed significantly in two images. The line-based method for correspondence has the merits of naturally enforcing the constraint of figural continuity and speeding up the matching process. Moreover, it could be used to simultaneously yield certain boundary descriptions in Hough space which can be very useful for surface interpolation as suggested, for instance, by Hoff and Ahuja [32] in their integrated approach for Surface from Stereo.

The line matching problem in the image space can readily be converted into a peak point matching problem in Hough space [33]. Furthermore, the search process in the Hough space can be accomplished effectively by using the technique of Dynamic Programming [1].

[1]The term "line matching" has been loosely used here. Strictly speaking, "collinear points" are being matched. Peaks in the Hough spaces represent the collinear points (whether they are connected or not). Optimal matches between the corresponding peaks are obtained in the Hough space. Afterwards, it is not difficult to obtain the disparity values, with a refined higher accuracy, for the individual corresponding points on each epipolar line.

In [34] Bolles, Baker, and Marimont proposed a technique of epipolar-plane image analysis for determining structure from motion. They took images in a rapid succession to obtain a solid block of image data. The technique used knowledge of the camera motion to form and analyze slices (epipolar-planes) of the solid. It was pointed out that for straight-line camera motions, simple linear structures will be formed in the epipolar-planes. This same technique can apparently be applied to our motion stereo method, where objects only move along the epipolar lines. Moreover, it will be shown latter in this chapter that, if the solid block of the image data is mapped onto a 3D Hough space, then the peaks that represent the lines in the original stereo images can be connected by a straight line in the Hough space. This forms the basis of our fast Hough line-based motion stereo technique, since the problem of matching lines among the multiple motion stereo images is effectively reduced to an easier problem of finding collinear points in the Hough space.

Many state-of-the-art vision systems [35, 36] primarily use line features for vision-guided navigation and object recognition, especially in man-made environments. Therefore, the study of line-based motion stereo is still of importance and practical use. Moreover, there have been many attempts of employing parallel hardware to speed up the depth recovery process. For example, papers [37, 38] showed how pipeline architectures could be used in stereo matching process.

The stereo algorithm described in this chapter has been implemented in the pyramid machine. It can quickly render a non-dense depth map along the linear contours

in the scene.

5.3 Hough Line-based Motion Stereo

As we mentioned early in chapter 2, a line in the image space can be mapped into a point in the parameter space by using Hough Transformation [15]. Since any 2D line in the image can be represented uniquely by two parameters, parameter spaces such as slope-intercept, or $\rho - \theta$ have been used. As shown in Figure 5.2(a), we instead will use parameters m_1, m_2 to represent a line in the image space, where m_1 and m_2 are the distances from the bottom of the image to the intersection points between the extension of the line and the left or right border of the image, respectively. A single peak value at (m_1, m_2) in the parameter space can be seen in Figure 5.2(a).

For convenience of the implementation in our pyramid machine, the stereo images of a time sequence are taken so that the epipolar lines are vertical. In other words, the disparity values between corresponding pixels in the multiple stereo images are measured by their Δy. For the lines represented by their parameters m_1 and m_2, the disparity will be reflected by Δm_1 and Δm_2. Suppose a line segment at time T_0 has two end points $(x_1(0), y_1(0))$ and $(x_2(0), y_2(0))$. If the belt moves along the Y-axis, at time T_1 the previous line segment will be seen at a new location with $(x_1(0), y_1(1))$ and $(x_2(0), y_2(1))$, and so on. By superimposing these multiple images taken at different time T, Figure 5.2(b) shows the movement of that line segment.

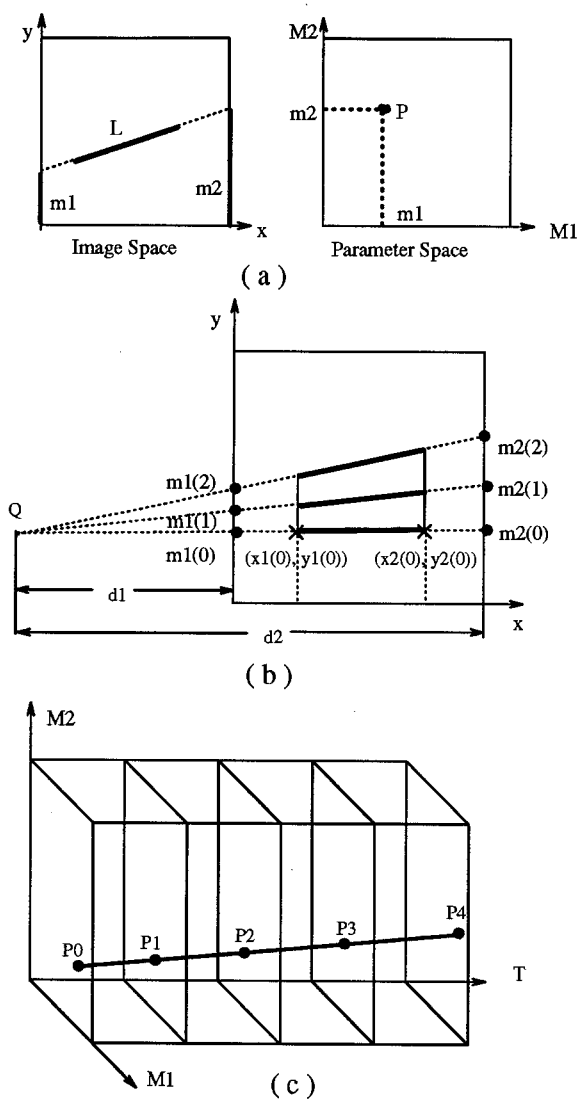

Figure 5.2: M1-M2-T 3D parameter space.

Given the constant speed of the vertical motion, it is guaranteed that at any moment t in the time sequence:

$$y_1(t) - y_1(t-1) = y_1(t+1) - y_1(t) \tag{5.5}$$

$$y_2(t) - y_2(t-1) = y_2(t+1) - y_2(t) \tag{5.6}$$

Suppose line segments $t-1, t, t+1$ are not parallel. They will all intersect at a common point Q after being extended (mathematical proof can be found in Appendix A). Figure 5.2(b) depicts this situation. Since

$$\frac{d_1}{d_1 + x_1(0)} = \frac{m_1(t) - m_1(t-1)}{y_1(t) - y_1(t-1)} \tag{5.7}$$

$$\frac{d_1}{d_1 + x_1(0)} = \frac{m_1(t+1) - m_1(t)}{y_1(t+1) - y_1(t)} \tag{5.8}$$

$$\frac{d_1 + x_2(0)}{d_2} = \frac{y_2(t) - y_2(t-1)}{m_2(t) - m_2(t-1)} \tag{5.9}$$

$$\frac{d_1 + x_2(0)}{d_2} = \frac{y_2(t+1) - y_2(t)}{m_2(t+1) - m_2(t)} \tag{5.10}$$

From equations 5.7, 5.8, 5.9, 5.10:

$$m_1(t) - m_1(t-1) = m_1(t+1) - m_1(t) \tag{5.11}$$

$$m_2(t) - m_2(t-1) = m_2(t+1) - m_2(t) \tag{5.12}$$

Lines in the multiple motion stereo images are transformed into multiple M1-M2 parameter spaces. A three-dimensional M1-M2-T parameter space can be constructed by stacking the multiple parameter spaces together. As in Figure 5.2(c), a line segment L in the nth image is represented as a point P_n in the M1-M2-T parameter space. Let $\Delta m(t) = m(t) - m(t-1)$, because at any moment t, $\Delta m_1(t) = \Delta m_1(t+1)$ and $\Delta m_2(t) = \Delta m_2(t+1)$, the N points in N M1-M2 planes can be connected by a straight line in the 3D parameter space. That is how the problem of matching lines

among the multiple motion stereo images is effectively being reduced to an easier problem of finding collinear points in the M1-M2-T space. This is also an important reason why M1-M2 is chosen to replace the well-known ρ, θ Hough representation. Lines in multiple motion stereo images will not be mapped to a straight line in the $\rho - \theta - T$ parameter space.

5.4 Pyramidal Line-based Motion Stereo Algorithm

In this section, a pyramidal algorithm for line detection and line matching will be described. Most existing stereo line matching algorithms detect lines first, followed by stereo matching. In this section, the processes of line detection and line matching are not separated. Instead, it will be shown that the integration of these two processes will benefit each other.

5.4.1 Pyramidal Line Detection

We have already presented a fast pyramidal line detection algorithm in the previous chapters. Basically, edge pixels are merged into short line segments at the lower level of the pyramid. The short lines are then passed to the parent nodes and merged recursively into longer lines up in the pyramid. To speed up the line merging process, a Dynamically Allocated Quadtree (DAQ) is used. This DAQ based pyramidal line

detection algorithm forms the basis of the line-based motion stereo algorithm.

Compared with the previous chapter, a line segment S is represented by its parameters m_1 and m_2. For hierarchical merging, coordinates of its endpoints and the current number (num) of pixels on the line segment are recorded. Besides, the disparity values at both endpoints (D_1 and D_2) are also kept for the purpose of stereo matching. These two parameters are defined as type array which is capable of keeping disparity information for more than one (up to a constant K) potential match. As a result, a line segment S is represented as:

$$S = (m_1, m_2, x_1, y_1, x_2, y_2, num, flag, D_1[K], D_2[K])$$

The DAQ pyramidal line merging algorithm is used for line detection. The only addition to that algorithm is that the disparity information is now also taken into account while merging the lines. To merge two line segments, not only their positions but also their disparity values will be checked. As for disparity, (a) the disparity values of the two line segments at the joint point must be very close, and (b) their disparity gradient $\frac{\partial D}{\partial l}$ must be similar.

5.4.2 Line Matching

As discussed before, every line will appear as a point in the M1-M2-T 3D parameter space. Also, the same line L appears in the sequence of N images will form N peak points in their respective M1-M2 planes. The task is to find the collinear peak points

that can be connected by a straight line in the 3D Hough space.

The N image frames come in succession. Frame 0 and Frame 1 are matched first. After Frame 2 comes, it will be matched against Frame 0, and so on, until all the frames are processed.

Procedure MATCH($DAQ[0], DAQ[i]$)

IF(i = 0) RETURN;

IF(i = 1)

FOR each leaf node $N(m_1, m_2)$ of DAQ[0]

define an area $A(m_1 \pm d, m_2 \pm d)$ on the M1-M2

plane for T=1; /* e.g., $d = 20$ */

match_node(N, A);

IF(i > 1)

FOR each leaf node $N(m_1, m_2)$ of DAQ[0]

FOR each potential match in MatchList[N], calculate the

searching area A' by using previous Δm_1 and Δm_2;

match_node(N, A');

END{MATCH}

Procedure match_node(N, A)

FOR each node N' in area A /* using DAQ[i'] */

IF N' is a matching candidate for N, link it to MatchList[N];

Keep only the K best candidates according to their similarity
in (a) disparity gradient, and (b) line length;
Store the MatchList[N] into m1[N][K], m2[N][K] array;
END{match_node}

For first match (Frame 0 to Frame 1), no information is available except the data in the two frames. For every line L in Frame 0 whose peak point in 3D parameter space is $(m_1, m_2, 0)$, search will be performed in the area $(m1 \pm d, m2 \pm d, 1)$ in the M1-M2 space for Frame 1, where d is a selected constant value. It is possible that more than one matchable peak point can be found in the searching area. Although the eventual task is to find out the best candidate, there is not enough information to decide at this point. The K best candidates are kept in $D_1[K], D_2[K]$. They will be used to guide later searches.

After the initial match between the pair of lines in the first and second images is obtained, a hypothetical line segment P0-P1 can be drawn in the M1-M2-T space. It is used to guide the future match. Figure 5.3 shows an example of the matching process in the 3D parameter space. Since the corresponding peaks from the sequence of the multiple M1-M2 planes can be connected with a straight line, the correct location for P2 in the M1-M2 plane at T = 2 can be readily calculated by extending the line connecting P0 and P1. In calculation, Δm_1 and Δm_2 derivable from the match P0-P1 are used to find out the center of the search area A' for P2. Similarly, the process can be carried out further for later images in the sequence. In the real implementation

some heuristic steps are also accommodated to make up the early misses. A simple example will illustrate how this heuristic works. Figure 5.3 depicts another situation in which line segment Q0 in image(0) doesn't match any correspondence line segments in image(1). Q0 is not discarded right away, instead, it is kept for another try and finally, Q0 finds its corresponding line segment Q2 and Q3 in image(2) and image(3).

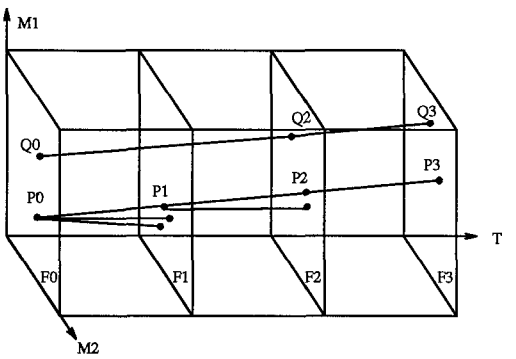

Figure 5.3: Matching in 3D parameter space.

In the worst case, K^2 potential match points could be found at each step and the K best among them will be retained. The number of potential match usually drops significantly after several steps. Since K is a relatively small number (4 in our experiment), the search process is very fast.

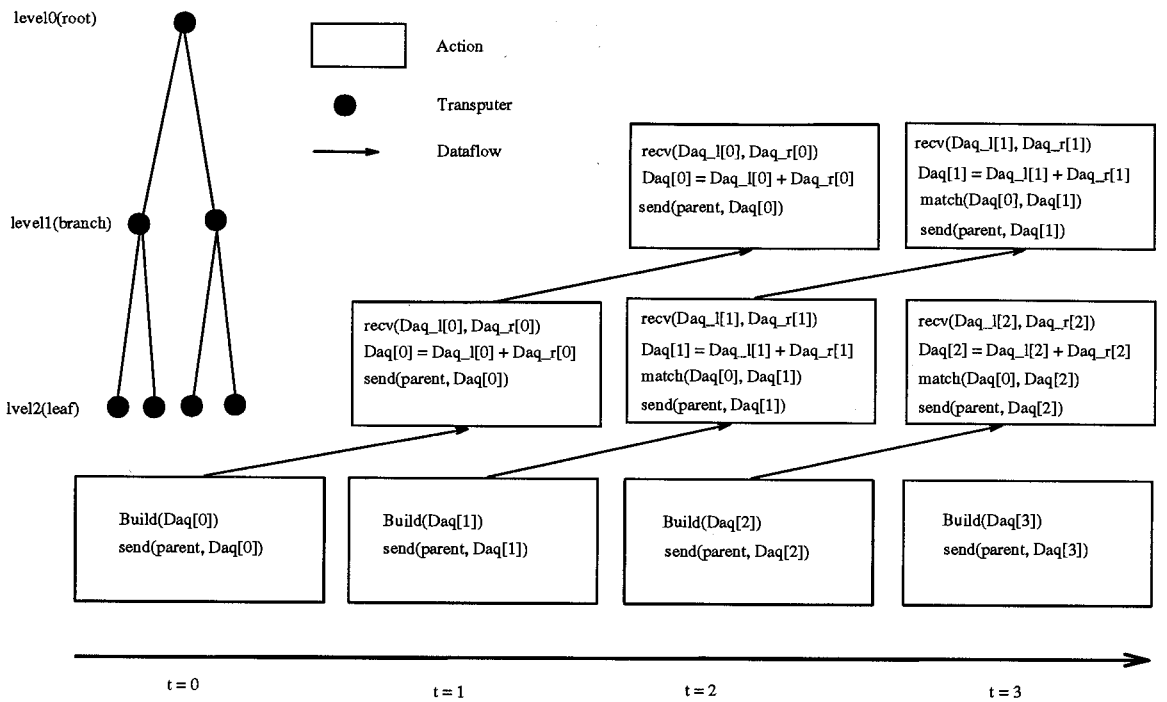

Figure 5.4: Data flow for line merging and matching in a three level pyramid.

5.4.3 Integrating Merging and Matching

The hierarchical line merging and matching are not independent processes. On one hand, merging short line segments into longer ones in the pyramidal hierarchy will gradually reduce the amount of lines to be matched. On the other hand, the disparity information from matching will impose another constraint on the merging, and thus yields a more reliable result.

The integration of merging and matching is accomplished in one bottom-up data flow process. The motion stereo images are first fed to the leaf nodes. They will then be pipelined through the pyramid with both merging and matching performed at each level.

In the pyramid, the bottom level transputer nodes will receive the edge frames from AIS. For each image, the initial DAQ's for short lines will be built by the leaf transputer nodes. The DAQ's will be sent to the parent transputers. Each parent transputer node receives two DAQ's from its two children and combines them into a new DAQ (merging). After that, it will match that DAQ with the DAQ of Frame 0, update the disparity information of the line segments and send it to its own parent, and then wait for the next frame. This process will stop at the top level (root) after all the frames are processed. Figure 5.4 illustrates the data flow for this integrated process in a simple three level pyramid.

According to the different tasks assigned to the different level of transputer, we divided the transputer into 3 groups: LEAF are those level 5 transputers, ROOT is the level 0 one, while all the remaining are belongs to BRANCH. We present those different tasks as follows:

The main data structures used:

1. FrameNumber; //total number of frames.
2. Frame[FrameNumber]; //input image from AIS.
3. DAQ[FrameNumber]; //DAQ tree of current node.
4. DAQ_Left[FrameNumber]; //DAQ received from left child.
5. DAQ_Right[FrameNumber]; //DAQ received from right child.

```
MotionStereo_LEAF(FrameNumber)
    FOR( i = 0; i < FrameNumber; i ++)
        FOR every edge pixel P of Frame[i]
                Traverse DAQ[i] from its root R downwards
            UNTIL it encounters a leaf node Q of DAQ[i];
            INSERT(S, R);
        SEND(PARENT, DAQ[i]);
END{MotionStereo_LEAF}

MotionStereo_BRANCH(FrameNumber)
    FOR( i = 0; i < FrameNumber; i ++)
        RECEIVE(DAQ_Left[i], DAQ_Right[i]);
        DAQ[i] = COMBINE(DAQ_Left[i], DAQ_Right[i]);
        MATCH(DAQ[0], DAQ[i]);
        SEND(PARENT, DAQ[i]);
END{MotionStereo_BRANCH}

MotionStereo_ROOT(FrameNumber)
    FOR( i = 0; i < FrameNumber; i ++)
        RECEIVE(DAQ_Left[i], DAQ_Right[i]);
        DAQ[i] = COMBINE(DAQ_Left[i], DAQ_Right[i]);
        MATCH(DAQ[0], DAQ[i]);
END{MotionStereo_BRANCH}
```

5.5 Compare Motion Stereo with Optical Flow

Extracting 3-dimensional information from time-varying images is very useful. We have already shown how to recover the depth of lines by using the line-based motion stereo approach. *Optical Flow* is defined as: "The apparent motion of brightness patterns observed when a camera is moving relative to the objects being imaged." [1]

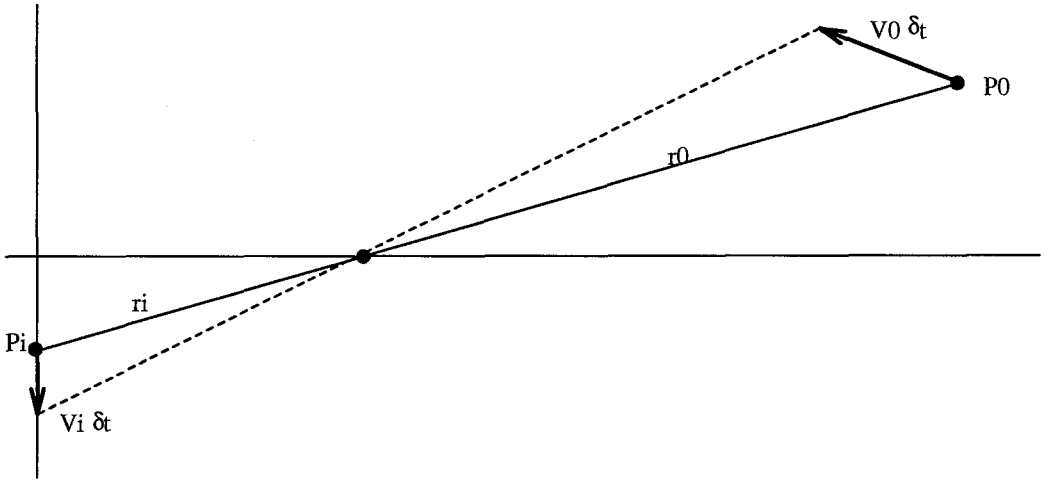

Figure 5.5: Perspective projection

Motion Field is the velocity vector for every image pixels. At a particular time t, a point P_i in the image corresponds to some point P_0 on the object's surface. These two points are connected by the projection equation(see Figure 5.5). Suppose the object point P_0 has velocity $\mathbf{v}_0$ which cause a velocity $\mathbf{v}_i$ for its corresponding image point P_i. The movement of P_0 over a time period δt is $\mathbf{v}_0\delta t$ while its image point P_i moves $\mathbf{v}_i\delta t$. We can have:

$$\mathbf{v}_o = \frac{d\mathbf{r}_o}{dt} \qquad \text{and} \qquad \mathbf{v}_i = \frac{d\mathbf{r}_i}{dt}, \tag{5.13}$$

$$\frac{1}{f_i}\mathbf{r}_i = \frac{1}{\mathbf{r}_o \cdot \hat{\mathbf{z}}}\mathbf{r}_o. \tag{5.14}$$

$$\frac{1}{f_i}\mathbf{v}_i = \frac{(\mathbf{r}_o \cdot \hat{\mathbf{z}})\mathbf{v}_o - (\mathbf{v}_o \cdot \hat{\mathbf{z}})\mathbf{r}_o}{(\mathbf{r}_o \cdot \hat{\mathbf{z}})^2} = \frac{(\mathbf{r}_o \times \mathbf{v}_o) \times \hat{\mathbf{z}}}{(\mathbf{r}_o \cdot \hat{\mathbf{z}})^2}. \tag{5.15}$$

From the perspective projection equation yielded above, it is obvious that the velocity vector can be assigned to every image pixels thus constitute the motion field. According to Horn [1], "Neighboring points on an object have similar velocities." So that it incurs that the motion field of the image is also continuous in most of the place.

In discrete digital images, an approximate continuous solution of motion field calculation can be reached by using a finite-difference schema. In particular, an iterative schema is used to implement the above theory. The number of iteration will affect the accuracy of the final result. In general, more iteration can give more precisely calculated motion field.

A wide variety of methods have been used to the measure of optical flow [39]. Some methods calculate the instantaneous motion field directly while the others track features across the image and then find a correspondence between features between one

moment to the next. The measurement of optical flow is known to be difficult because of the following two points. First, the changing pattern of intensity of the image only offers partial information about the true motion. Second, there does not exist a unique optical-flow field among changing images when the general motion of objects is allowed [39].

The optical flow mechanism can be mapped to a wide range of application domains, from the simple tracking of moving objects on the conveyor belt in industrial environment to the analysis of much more complex motions such as the cells in cell culture. The motion stereo approach we proposed deals with a subset of the optical-flow problems by imposing more constraints in a certain application domain. In the industrial environment where the objects are moving along the well-controlled conveyor belt, more useful constraints can be set to speed up the optical-flow process. To make the object moving along a straight epipolar line, we can use multi-baseline stereo technology to solve the correspondence problem. Moreover, for objects with linear features, it is convenient to impose the line-based stereo instead of pixel-based calculation to speed up the process. It is obvious that the motion stereo approach works well for the above application domain. For complicated object movements and less constrainted working environment, optical-flow is still a good choice.

CHAPTER 6

EXPERIMENTAL RESULTS

This chapter gives some experimental results of our pyramidal based algorithms. We start with the pyramidal line detection algorithms, followed by the line-based motion stereo algorithm.

6.1 Pyramidal line detection

This section reports the comparative timing results for the simple merging algorithm and the improved algorithm that uses DAQ. Several real images and synthetic images are tested. All the test images are of the size of 512×512.

Live input images are digitized by the AIS-4000. The sobel edge operator as adopted by Rosenfeld, et al. [18] is used to find the edge pixels and their directions

θ's. The detected edges are initially more than one pixel wide. A non-maximum suppression technique is used to reduce their edge widths to one. All these preprocessings take place in the AIS-4000 in less than 200 milliseconds. They will not be included in the following timing results.

6.1.1 Simple Merging Algorithm

Table 6.1: Timing results for detecting lines in real images

	Time Required (ms) for Simple Algorithm						
	level5	level4	level3	level2	level1	level0	Total
cube	95.7	5.8	8.6	5.6	1.4	2.7	119.8
pyramid	104.6	11.5	9.3	5.8	1.6	12.1	144.9
Rubik	437.60	132.4	248.6	560.3	210.2	1626.2	3215.3

The results in Table 6.1 suggest that the simple pyramidal line merging algorithm can detect all lines in several real images (containing simple objects such as cubes or pyramids) at the order of 100 ms. They also indicate the following:

- when the lines of the larger objects are longer, the parent nodes will need a little more time to do some real merging job instead of simply passing the results up to the root;
- when the object has more lines, e.g. the "Rubik's cube", it will take longer to merge them.

To better illustrate the later point, four synthetic grid images are used. The first image has an 8×8 grid and hence $8 + 8 = 16$ lines. Similarly, the others have

$16 + 16 = 32, 32 + 32 = 64$, and $64 + 64 = 128$ lines.

Table 6.2: Timing results for detecting lines in synthetic images

	Time Required (ms) for Simple Algorithm						
	level5	level4	level3	level2	level1	level0	Total
8x8 grid	272.6	6.6	8.7	7.7	8.5	25.6	329.7
16x16 grid	368.0	25.8	33.2	30.7	32.8	95.3	585.8
32x32 grid	648.8	118.4	130.4	124.7	131.3	389.5	1543.1
64x64 grid	1287.4	473.9	524.1	504.3	519.5	1485.3	5434.5

Table 6.2 shows the significant increase in the total time required to detect all lines when the number of lines increases from 16 to 128. Notably, for the non-leaf transputer nodes (level 4 to level 0), the time increase is approximately quadratic. This is consistent with the analysis in Chapter 4(4.1.3) which states the worst case time complexity for the simple merging algorithm is O(PQ) where P and Q are now the numbers of line segments detected by the child nodes. For the leaf nodes, however, the increase is not nearly as drastic. It is because that many trivial merges happened to occur at the leaf level for grid images and hence avoided the expensive procedure of checking for collinearity.

6.1.2 DAQ-based Merging Algorithm

Table 6.3 presents the comparative results of the DAQ merging algorithm and the simple merging algorithm. Apparently, the DAQ merging algorithm is more efficient than the simple algorithm. The speed increase range from 1.62 to 8.15 times for the

Table 6.3: Comparative Timing Results for Line Detection

	Timing (in milliseconds) for the Simple Algorithm						
	8 × 8 grid	16 × 16 grid	32 × 32 grid	64 × 64 grid	Cube	Pyramid	Rubik
Level 5	272.6	368.0	648.8	1287.4	95.7	104.6	437.6
Level 4	6.6	25.8	118.4	473.9	5.8	11.5	132.4
Level 3	8.7	33.2	130.4	524.1	8.6	9.3	248.6
Level 2	7.7	30.7	124.7	504.3	5.6	5.8	560.3
Level 1	8.5	32.8	131.3	519.5	1.4	1.6	210.2
Level 0	25.6	95.3	389.5	1485.3	2.7	12.1	1626.2
Total	329.7	585.8	1543.1	5434.5	119.8	144.9	3215.3
	Timing (in milliseconds) for the DAQ Algorithm						
	8 × 8 grid	16 × 16 grid	32 × 32 grid	64 × 64 grid	Cube	Pyramid	Rubik
Level 5	176.2	208.5	281.9	553.2	59.8	65.2	278.3
Level 4	7.5	11.4	32.4	65.8	12.4	13.2	72.3
Level 3	6.3	6.8	10.6	14.9	4.3	10.6	25.8
Level 2	4.8	7.3	8.4	12.7	4.6	5.9	19.8
Level 1	5.9	7.8	9.6	10.2	1.1	1.7	7.1
Level 0	3.1	4.0	8.8	10.4	3.2	5.8	33.2
Total	203.8	245.8	351.7	667.2	85.4	102.4	436.5
Speedup	1.62	2.38	4.39	8.15	1.40	1.42	7.37

grid images. Figure 6.1 depicts this comparative result.

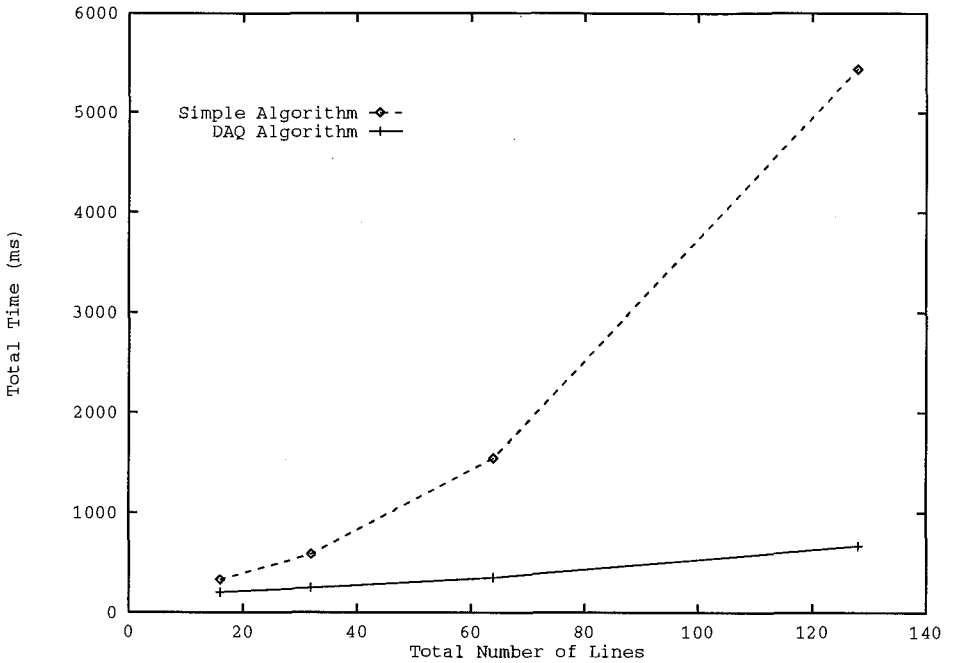

Figure 6.1: Comparative result for the grid images

As for real images, the cube and pyramid are simple polyhedral objects (images not shown) with only a few boundary lines, whereas the Rubik cube image has 72 extended lines. As shown in Figure 6.3, the edge map of "Rubik" has some noise that can be expected to slow down somewhat the line merging process. Indeed, Table 6.3 shows that the total line detection time of 436.5ms for "Rubik" by the DAQ algorithm is between the time required by the 32×32 and the time required by the 64×64 grid images. Moreover, a good speed up of 7.37 is achieved. Figure 6.4 is a reconstructed cube scene for all the line detected by the DAQ algorithm.

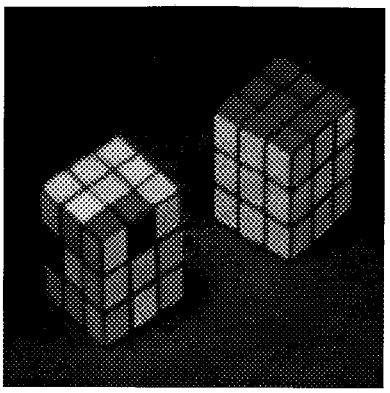

Figure 6.2: Live image of rubik cube

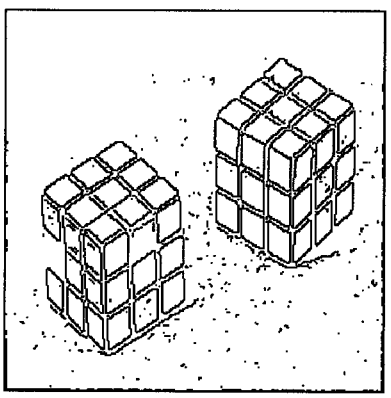

Figure 6.3: Edgemap of rubik-cube

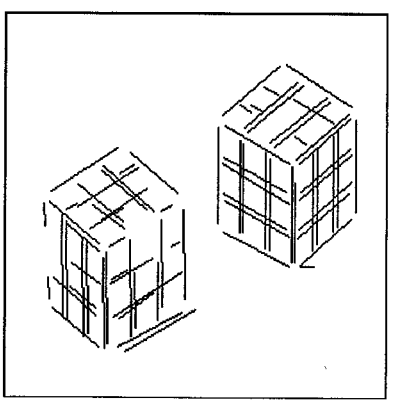

Figure 6.4: All detected lines

As shown in all the tables the leaf nodes are clearly overloaded. The application of flexible pyramidal pipelining technique [25] can easily balance the load of the transputer nodes at all levels of the pyramid and thus shorten the total time needed for the entire pyramidal line detection process. However, the main purpose of this section is to show the comparative results from the simple merging algorithm and the one using DAQ. For the simplicity and clarity for a fair comparison, the pipeline technique is not used.

6.1.3 Choosing the depth of DAQ

Table 6.4: Timing results for using various max-depth's

max-depth	*Time Required (ms)*			
	level5	level4	level3	subtotal
0-0-0	1413.7	365.4	370.2	2149.3
1-0-0	800.0	364.8	370.2	1535.0
2-1-0	431.4	180.9	382.8	995.1
3-2-1	256.8	94.1	342.4	693.3
6-4-4	157.9	34.7	40.3	232.9
6-5-4	157.9	20.2	46.2	224.3
6-6-4	157.9	10.9	53.8	222.6
7-4-4	124.2	65.9	40.3	230.4
7-5-4	124.2	44.7	46.2	215.1
7-6-4	124.2	39.4	53.8	217.4
7-7-4	124.2	19.6	103.9	247.7
8-4-4	66.1	110.8	40.3	217.2
8-5-4	66.1	85.7	46.2	198.0
8-6-4	66.1	79.6	53.8	199.5
8-7-4	66.1	64.1	103.9	234.1
8-8-4	66.1	34.2	214.8	315.1

The timing for various combinations of max-depth at different levels is measured.

A 256 × 256 synthetic image which contain huge amount of lines is used. Due to the image size, only 31 transputer nodes(5 levels) involved in this testing. Table 6.4 lists a number of their timing results. The notation n_1-n_2-n_3 indicates that max-depth is n_1, n_2, and n_3 at level 4, 3, and 2, respectively.

The first row in the table (max-depths: 0-0-0) shows an extreme case in which the DAQ algorithm degenerates to the simple merging algorithm. The next three rows show some examples when the max-depth's are still not sufficiently large (i.e., only 1, 2 or 3). All timing results for the first four rows are unsatisfactory, although there is already a clear decreasing trend in their subtotals. For the other rows, the max-depth at level 3 is chosen as 4, the max-depth at level 5 ranges from 8 to 6, and the max-depth at level 4 is somewhere in between. Apparently, the DAQ algorithm works well as long as the max-depth's are reasonably large (i.e., $\geq$ 4 at all levels). The overhead for building the DAQ's is never overwhelmingly heavy. In fact, the best timing results are from 8-5-4 and 8-6-4 when the leaf nodes at level 5 have the deepest DAQ of max-depth = 8 and only require 66.1 ms, which yield the minimal subtotals (198.0 or 199.5 ms) for all three levels.

6.1.4 Real-time Blocking Sorting - Integrating with Robotic Workcell

The fast pyramidal line detection algorithms have been integrated into a robotic workcell, which consists of a moving belt, a robot arm set beside the belt and a camera mounted above the mobile belt.

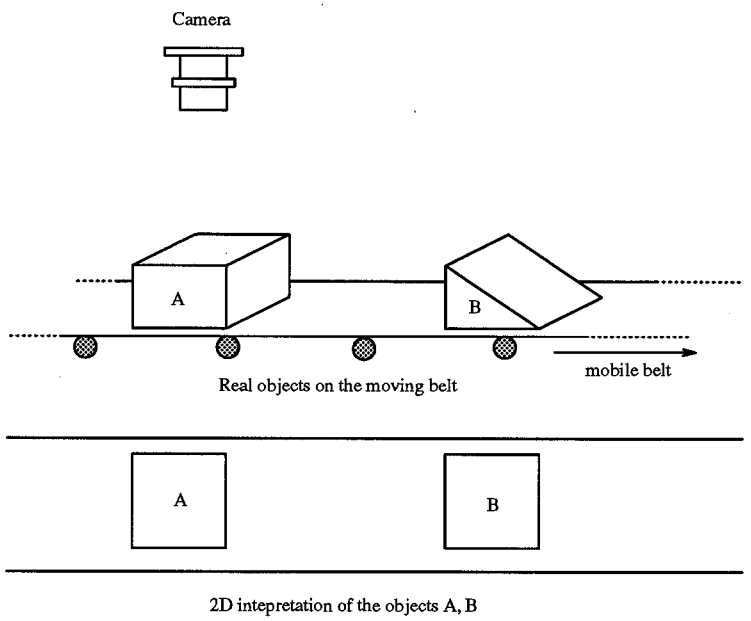

Figure 6.5: Mis-Interpretation of objects using only 2D information

A stream of varied polyhedral wooden blocks are placed on the moving belt. Live images are taken by the camera mounted above the belt and are sent to the hybrid pyramid for line detection. The vision system detects all the lines of the incoming images and decides whether it is a cube(square block from the image). Cubes are removed from the belt by the robot arm while the other shapes are left on the belt.

For ordinary block scenes, the algorithm takes less then 150 ms for line detection in the transputer pyramid. A simple heuristic algorithm was adopted for cube recognition which is basically taking the number of edges of the block and its orientation into consideration. The entire vision process is completed in less than 500 ms and thus enables real time sorting of the blocks.

Since the pyramidal line detection algorithm only interprets the 2D lines, it may misinterpret some objects, Figure 6.5 illustrates this situation. In a 2D scene, both objects will appear as a "cube" but actually only object A is. This problem will not be solved as long the depth information of the lines is missing. In other words, using line detection alone is not enough for recovering the objects. As we can see, this problem can be solved by combining the motion-stereo mechanism.

6.2 Motion Stereo Depth Recovery

In our experiment, polyhedral wooden blocks are placed on the moving belt, and eight consecutive frames of images were taken by the camera. Figure 6.6 shows eight live images.

Images are sent to the AIS-4000 array processor frame by frame for edge detection. Approximately 40 milliseconds (ms) is needed for each frame. Edge images are then sent to transputer leaf nodes via the PARLink. The pyramidal line merging and matching algorithm is applied to derive the final disparity map. Due to the limitation of the internal storage of the transputers (2MB on each transputer), not all edge images are kept. In fact, only edges for Frame 0 and the current Frame i are kept, all other frames ($0 < k < i$) are discarded after they are matched against Frame 0.

Figure 6.7 shows all the matched lines of the moving object from 8 frames (Notice: all static lines, e.g., the edges of the sensor mounted on the frame of the belt, are correctly not matched.) The final disparity map of Frame 7 is shown in Figure 6.8

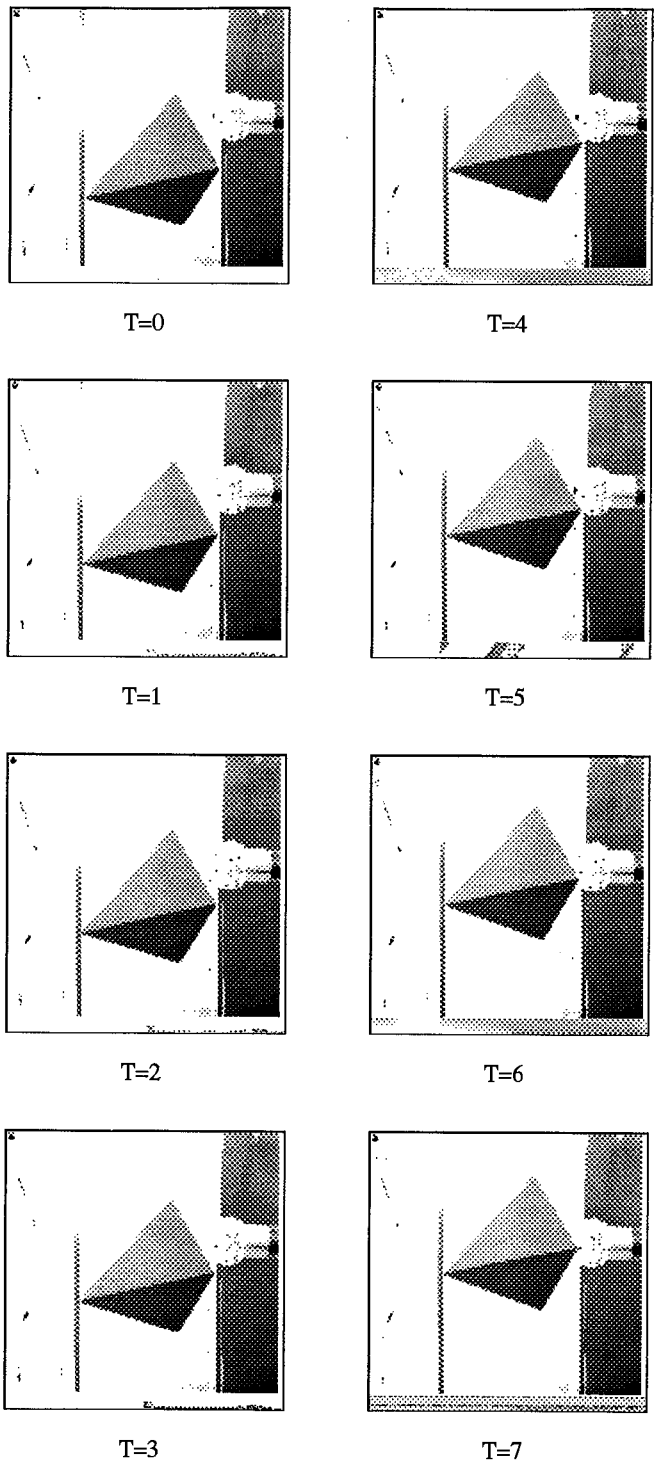

Figure 6.6: Live images captured at different time T

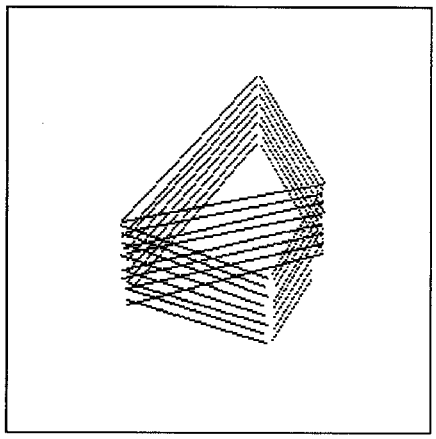

Figure 6.7: All matched lines of 8 frames of the moving object

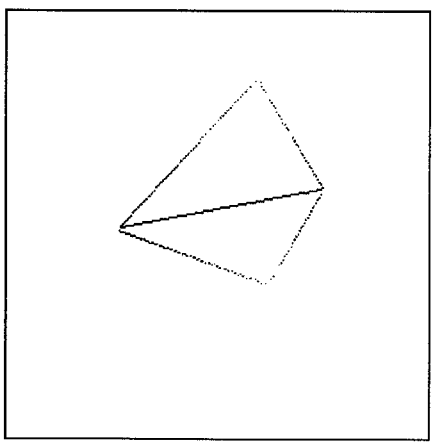

Figure 6.8: Disparity map of frame 7

(for display purposes, the disparity values are shown with coded gray level intensities).

Table 6.5: Timing results for recovering lines in real images.

	Time Required (ms)						
	AIS	level5	level4	level3	level2	level1	level0
Edge Detection	40.0	X	X	X	X	X	X
Finding Short Lines	X	70.1	X	X	X	X	X
Merging & Matching	X	X	13.0	5.9	3.0	6.1	5.7
Sub Total	40.0	70.1	13.0	5.9	3.0	6.1	5.7

Table 6.5 shows the time needed to work on one frame of image "pyramid". The matching process of one image can be done at the order of 100 ms. Since the pipeline method is applied, the total time for 8-frame motion stereo is:

$$t = 40.0 + 8 \times 70.1 + 13.0 + 5.9 + 3.0 + 6.1 + 5.7 = 634.5(\text{ms})$$

As shown in the table, the level 5 nodes are clearly overloaded. The load can be readily balanced by using the pipeline technique as we mentioned in the previous section. Again, in this preliminary experiment the pipeline technique is not used.

CHAPTER 7

CONCLUSIONS AND DISCUSSION

This thesis describes a parallel and hierarchical (pyramidal) approach to fast Hough line detection and line-based motion stereo. The lines are represented using their Hough parameters ρ and θ. The processes of line merging and matching are integrated in the pyramidal framework.

Fast Hough line detection algorithms are first presented in which the lines are represented using their Hough parameters ρ and θ are merged hierarchically by all the transputers in the pyramid. To reduce the complexity of the line merging process, a Dynamically Allocated Quadtree (DAQ) is introduced to represent the Hough parameter space. The comparative experimental results show that the line merging algorithm using DAQ is significantly more efficient than the simple merging algorithm.

The recovery of depth information of an object is important for 3D vision since it is essential for robotics and many manufacturing automation applications. An algorithm for line-based motion stereo is presented in the second part of this thesis. It is applied to a testbed of a manufacturing environment where an assembly belt is moving at constant speed. Only one camera is needed to capture a succession of motion stereo images. A parallel and hierarchical (pyramidal) algorithm for line merging and matching is described. It is shown that the problem of matching lines among the multiple, motion stereo images can be effectively carried out in a 3D parameter space.

A robotic workcell has been integrated into the overall system setup and it realizes the real-time vision process on the SFU hybrid pyramid vision machine. Preliminary experimental results show that our pyramidal line detection mechanism is capable of detecting all lines of a simple real image in the order of 100 ms, and the line-based motion stereo system is being able to producing a depth map along the linear features in about half a second.

As Horn [1] described, the purpose of the vision system is to produce symbolic descriptions of the imaged objects; to make decisions based on the descriptions; and finally, to manipulate the objects by moving the robot arm according to the decision made. Our approach demonstrated this kind of vision system (A diagrammatic representation of this kind of vision system is shown in Figure 7.1) has potential in real-time object recognition.

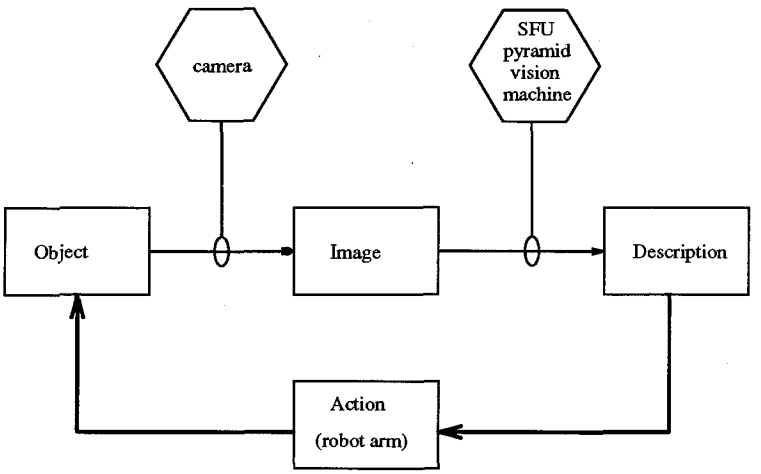

Figure 7.1: Vision system

It is now well-known that the pyramid architecture provides an unique parallel and hierarchical platform, and has great potential for parallel vision [7, 8, 23]. It supports the multiresolution approach and capitalizes on the advantages of both the mesh and the pipeline architectures. However, a full-scale 3D pyramid with thousands of nodes has yet to be created, and its availability will be limited for years to come. We have shown that, a 2D"half-scale" hybrid pyramid is not only economical, but can also be quite efficient and adequate for many parallel vision tasks.

Since we are developing vision systems for industrial purposes, further work is needed to improve the current algorithms and implementations and to make them as practical as possible. However, some limitations have been identified:

- A load balance mechanism should be integrated into the current system. The goal can be achieved by choosing the proper max-depth of the DAQ at different levels and by using the pipeline technique. One limitation of the above methods is that they are all statically managed. A dynamical load-balancing mechanism

which can determine the load of different nodes and balance them on run-time is needed.

- The current motion stereo algorithm has difficulty dealing with vertical lines because the epipolar line is parallel to y-axes. To get a full description of both x-major and y-major lines, trinocular stereo can be applied [40]. Instead of a simple camera, a pair of cameras can be placed side by side in the direction perpendicular to the belt movement. The additional sequence of the motion stereo images will provide the necessary information for conducting matching in trinocular stereo.

Appendix A

ADDITIONAL PROOF

Figure A.1 depicts 3 non-parallel line segments ab, cd and ef. As shown in in Chapter 5, $ac = cd = D_1$, $bd = df = D_2$. $D_1 \neq D_2$ as these lines are non-parallel. The proof below shows that these three non-parallel lines will intersect at a common point.

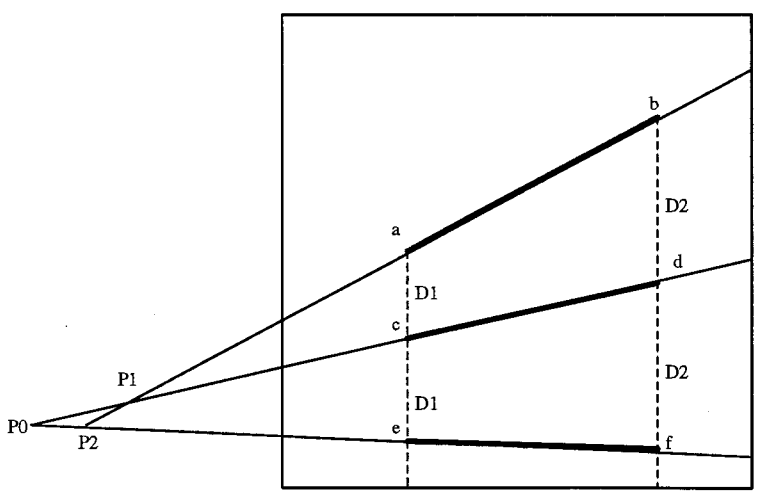

Figure A.1: Unparallel line segments

Proof by contradiction:

First extend the line segments ab, cd and ef and assume that they don't intersect at a common point. The intersection points are defined as follows:

P_0 is the intersection point of lines ef and ab.

P_1 is the intersection point of lines ab and cd.

P_2 is the intersection point of lines ab and ef.

From similar triangles, the ratio relationships are given:

$$\frac{P_2e}{P_2f} = \frac{ae}{bf} = \frac{2D_1}{2D_2} = \frac{D_1}{D_2} \tag{A.1}$$

$$\frac{P_0e}{P_0f} = \frac{ce}{df} = \frac{D_1}{D_2} \tag{A.2}$$

From Figure A.1:

$$Pe = P_0P_2 + P_2e \tag{A.3}$$

$$Pf = P_0P_2 + P_2f \tag{A.4}$$

From equation A.1, A.2, A.3 and A.4:

$$\frac{P_0P_2 + P_2e}{P_0P_2 + P_2f} = \frac{D_1}{D_2} = \frac{P_2e}{P_2f} \tag{A.5}$$

Assume $P_0P_2 \neq 0$ and $D_1 \neq D_2$:

$$P_2f \times (P_0P_2 + P_2e) = P_2e \times (P_0P_2 + P_2f) \tag{A.6}$$

So we have:

$$P_2f \times P_0P_2 + P_2f \times P_2e = P_2e \times P_0P_2 + P_2e \times P_2f$$

$$P_2 f = P_2 e \tag{A.7}$$

From Figure A.1 it is easy to see that $P_2 f \neq P_2 e$. This conclusion shows that our assumption about $P_0 P_2 \neq 0$ and $D_1 \neq D_2$ can not stand. Since $D_1 \neq D_2$, the only possible solution is $P_0 P_2 = 0$ which means $P_0 = P_1 = P_2$ so that all 3 non-parallel lines meet at a common intersection point.

REFERENCES

[1] B.K.P. Horn. *Robot Vision.* The MIT Press, 1986.

[2] D. Marr and T. Poggio. A computational theory of human stereo vision. In *Royal Society of London*, pages 301–328, 1979.

[3] M. Maresca, M.A. Lavin, and H. Li. Parallel architectures for vision. *Proceedings of the IEEE*, pages 1006–1015, 1988.

[4] Q.F. Stout. Mapping vision algorithms to parallel architectures. *Proceedings of the IEEE*, pages 982–995, 1988.

[5] R.E. Cypher, J.L.C. Sanz, and L. Synder. The hough transform has o(n) complexity on simd nxn mesh array architectures. In *Proc. IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 115–121, 1987.

[6] S.L. Tanimoto and T. Pavlidis. A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, 4:104–119, 1975.

[7] P.J. Burt. The pyramid as a structure for efficient computation. In Azriel Rosenfeld, editor, *Multiresolution Image Processing and Analysis*, pages 6–35, New York, 1984. Springer-Verlag.

[8] P.J. Burt. Smart sensing within a pyramid vision machine. *Proceedings of the IEEE*, 76(3):1006–1015, 1988.

[9] C.C. Weems and J.H. Burrill. The image-understanding architecture and its programming environment. In V.K. Prasanna Kumar, editor, *Paralle Architectures and Algorithms for Image Understanding*, pages 525–562, San Diego, 1991. Academic Press.

[10] G. Nudd, N. Francis, T. Atherton, D. Kerbyson, R. Packwood, and H. Vaudin. Hierarchical multiple-simd architecture for image analysis. *Machine Vision and Applications*, 5:85–103, 1992.

[11] M.H. Sunwoo and J.K. Aggarwal. Vista-an image understanding architecture. In V.K. Prasanna Kumar, editor, *Parallel Architectures and Algorithms for Image Understanding*, pages 121–154, San Diego, 1991. Academic Press.

[12] E.M. Riseman and A.R. Hanson. Progress in computer vision at the university of massachusetts. In *Proceedings of Image Understanding Workshop*, pages 86–96. DARPA, September 1990.

[13] John Ens. A parallel link between an AIS-4000 and a transputer pyramid. Technical Report CSS-IS TR91-09, Simon Fraser University, Centre for Systems Science, Burnaby, B.C., Canada V5A 1S6, October 1991.

[14] M. Okutomi and T. Kanade. A multiple-baseline stereo. In *Proc. IEEE Conf. on Vision and Pattern Recognition*, pages 63–69, 1991.

[15] R.O. Duda and P.E. Hart. Use of the hough transform to detect lines and curves in pictures. *Communications of the ACM*, 15:11–15, 1972.

[16] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision Graphics and Image Processing*, 44:87–116, 1988.

[17] A.N. Choudhary and R. Ponnusami. Implementation and evaluation of hough transform algorithms on a shared-memory multiprocessor. *Journal of Parallel and Distributed Computing*, 12(9):178–188, 1991.

[18] A. Rosenfeld, J. Ornelas, and Y. Hung. Hough transform algorithms for mesh-connected simd parallel processors. *Computer Vision Graphics and Image Processing*, 41:293–305, 1990.

[19] S. Ranka and S. Sahni. Computing hough transform on hypercube multicomputers. *Journal of Supercomputing*, 4:169–190, 1990.

[20] J. Jolion and A. Rosenfeld. An o(log n) pyramid hough transform. *Pattern Recognition Letters*, 9:343–349, 1989.

[21] H. Li and M.A. Lavin. Fast hough transform based on bintree data structure. pages 640–642, 1986.

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

[23] A. Rosenfeld. Some useful properties of pyramids. In Azriel Rosenfeld, editor, *Multiresolution Image Processing and Analysis*, pages 2–, New York, 1984. Springer-Verlag.

[24] J. Ens, Z. Li, F. Tong, D. Zhang, S. Atkins, and W. Luk. A hybrid pyramidal vision machine for real time object recognition. In *Transputer Research and Applications 5*, pages 90–103. IOS Press, 1992.

[25] Z.N. Li. Vision in pyramids - object recognition in real-time. In *Proc. 6th Int. Conf. on CAD/CAM, Robotics, and FOF*, pages 344–349, 1991.

[26] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Survey*, 16(2):187–260, 1984.

[27] J. O'Rourke. Dynamically quantized spaces for focusing the hough transform. In *Proceedings 7th IJCAI*, pages 737–739, 1981.

[28] K.R. Sloan. Dynamically quantized pyramids. In *Proceedings 7th IJCAI*, pages 734–736, 1981.

[29] J.E.W. Mayhew and J.P. Frisby. Psychophysical and computational studies towards a theory of human stereopsis. *Artificial Intelligence*, 17:349–385, 1981.

[30] G. Medioni and R. Nevatia. Segment-based stereo matching. *Computer Vision, Graphics, and Image Processing*, 31:2–18, 1985.

[31] J.H. McIntosh and K.M. Mutch. Matching straight lines. *Computer Vision, Graphics, and Image Processing*, 43:386–408, 1988.

[32] W. Hoff and N. Ahuja. Surfaces from stereo: integrating feature matching, disparity estimation, and contour detection. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 11(2):121–136, 1989.

[33] Z.N. Li. Dynamic programming in hough space for line matching in stereopsis. *ISPIE Symposium on Advances in Intelligent Robotics Systems '89*, SPIE Vol. 1195 (Mobile Robots IV):209–220, 1989.

[34] R.C. Bolles, H.H. Baker, and D.H. Marimont. Epipolar-plane image analysis: an approach to determining structure from motion. *Int. Journal of Computer Vision*, 1:7–55, 1987.

[35] E. Triendl and D.J. Kriegman. Stereo vision and navigation within buildings. In *Proc. IEEE Conf. on Robotics and Automation*, pages 1725–1730, 1987.

[36] P. Kahn, L. Kitchen, and E.M. Riseman. A fast line finder for vision-guided robot navigation. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 12(11):1098–1102, 1990.

[37] A.E. Kayaalp and J.L. Eckman. Near real-time stereo range detection using a pipeline architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:1461–1469, 1990.

[38] A.F. Laine and G.G. Roman. A parallel algorithm for incremental stereo mathing on simd machines. *IEEE Transactions on Robotics and Automation*, 7:123–134, 1991.

[39] E. Hildreth. Optical flow. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence, Volume2*, pages 684–687. John Wiley and Sons, 1987.

[40] C.V. Stewart and C.R. Dyer. The trinocular general support algorithm: a three-camera stereo algorithm for overcoming binocular matching errors. In *Proceeding of 2nd International conference on Computer Vision*, pages 134–138, 1988.