ABSTRACT

Automatic Motion Analysis of Colliding Spheres

by

John Vijayakumar Caesar

Motion analysis is useful to compute linear and angular velocities and acceleration of an object from a sequence of images. This thesis is part of an investigation to compute the translation and rotation velocities needed to determine the collision parameters of two colliding spheres. This involves the tracking of the spheres and feature points on the spheres over a time interval. An experimental setup releases two spheres such that they collide and a high speed imaging system, i.e., Kodak Ektapro 1000 is utilized to record the motion of the spheres. The imaging system is capable of recording at a speed of 1000 frames/sec with an image resolution of 239 x 192 for each frame. Selected images are analyzed in a PC 486 using programs developed with the Visilog software from Noesis[1]. Edge data from the images allow the feature points and the locations of the spheres to be detected and their locations recorded. Centers of the circles are computed using the Hough transform technique. Correspondence of the feature points from frame to frame is achieved using the proximal uniformity constraint. Suggestions for future work are given.

1. Visilog is a trademark of Noesis S.A.R.L.

AUTOMATIC MOTION ANALYSIS OF COLLIDING SPHERES

by

John Vijayakumar Caesar

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science
Department of Mechanical and Industrial Engineering
October 1992

APPROVAL PAGE

Automatic Motion Analysis of Colliding Spheres

by

John Vijayakumar Caesar

May 8, 92

Dr. Rajesh N. Dave, Thesis Advisor

Assistant Professor of Mechanical Engineering, NJIT

May 8, 1992

Dr. Anthony Rosato, Committee Member

Assistant Professor of Mechanical Engineering, NJIT

May 8, 1992

Dr. Ian S. Fischer, Committee Member

Associate Professor of Mechanical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: John Vijayakumar Caesar

Degree: Master of Science in Mechanical Engineering

Date: October, 1992

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Mechanical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1992
- Bachelor of Engineering in Mechanical Engineering,
  R. V. College of Engineering, Bangalore, India, 1990

Major: Mechanical Engineering

Presentations:

Caesar, John. "Automatic Tracking of Multiple Spheres Using a High Speed Imaging System." Presented at *Region-II Graduate Student Technical Conference,* Steven's Institute of Technology, April, 1992.

This Thesis is Dedicated

To my Parents

ACKNOWLEDGMENT

I take this opportunity to express my gratitude to Dr. Rajesh N. Dave, Assistant Professor, Mechanical Engineering Department of N.J.I.T. for his encouragement and valuable guidance throughout the course of this thesis work. His helpful hints, suggestions and patience were of immense help.

I am also very thankful to Dr. Anthony Rosato and Dr. Ian Fischer for reviewing my work and providing me with their useful suggestions.

I also thank my colleagues Summit Sen, Kimberly Hrnciar, Jerry Volcy, Avadhani Ashok and Kurra Bhaswan who have lent a helping hand at various times during the course of this thesis work.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1
- 1.1 Overview Of The Problem 1
- 1.2 Automatic Motion Analysis 2
- 1.3 Statement Of The Problem 3
- 1.4 Overview Of The Remaining Chapters. 4

2. LITERATURE REVIEW 6
- 2.1 Edge Detection 6
  - 2.1.1 First Derivative Edge Operators 6
    - 2.1.1.1 L. G. Robert's Operator 6
    - 2.1.1.2 Sobel's Operator 7
    - 2.1.1.3 Canny's Operator 8
  - 2.1.2 Second Derivative Edge Operators 11
    - 2.1.2.1 Marr & Hildreth 11
- 2.2 Circle Detection Techniques 12
  - 2.2.1 Hough Transform Based Techniques 12
- 2.3 Motion Analysis And Feature Point Correspondences 14

3. DESCRIPTION OF THE VISION SYSTEM 17
- 3.1 Hardware Used 17
- 3.2 Software Used 19
- 3.3 Description And Working Of The Set-up 23
  - 3.3.1 Description 23
  - 3.3.2 Working 23

4. PREPROCESSING USING VISILOG 27
- 4.1 Introduction To Visilog 27
- 4.2 Accessing Images In Visilog 29
  - 4.2.1 Image 29
  - 4.2.2 Acessing Images 29
- 4.3 Program Description 31

4.3.1 A brief description of the algorithm and functions used 31
4.3.2 Initialize( ) 32
4.3.3 Edge(char* name,int frame) 32
4.3.4 Hough(input,frame) 36
4.3.5 Marker(char* name,int frame) 39

5. ESTABLISHING CORRESPONDENCE 42
5.1 Definition 42
5.2 Proximal Uniformity Function 42
5.3 Implementation of the Algorithm 44
5.4 The Occlusion Problem 45

6. RESULTS AND CONCLUSION 47
6.1 Results 47
6.2 Conclusion 64

APPENDIX 66

REFERENCES 89

LIST OF TABLES

Table **Page**

1 Coordinates of the markers on the left sphere before achieving correspondence 57

2 Coordinates of the markers on the left sphere after achieving correspondence 57

3 Coordinates of the markers on the right sphere with a point missing in the 3rd frame before correspondence 62

4 Coordinates of the markers on the right sphere after achieving correspondence 62

LIST OF FIGURES

Figure **Page**

2.1 Canny's Non-Maximum Suppression Operator 10

3.1 A Typical System Configuration 21

3.2 System with Imaging Board 22

3.3 Schematic view of the communications and experimental setup 26

4.1a Shows the grey level image 35

4.1b Image after Sobel's operator 35

4.1c Image after Non-Maximum Suppression 35

4.1d Edge image after thresholding 35

4.2 Relationship between (x,y,θ) and center parameters (a,b) for a circle . . . 37

4.3 Shows the circle center using Hough transform 37

4.4 Shows multiple circle detection 38

4.5 Shows the markers and displays the number located 41

6.1a Shows the grey level image of the two spheres in the collision space . . . 49

6.1b Shows the edge image obtained from the grey level image using the Sobel and Canny edge operators. The centers of the spheres are also marked using the Hough transform technique 50

6.1c Shows that all the markers have been successfully located and the number of markers found is displayed on the image 51

6.2 Shows a sequence of grey level images recorded at 1000 frames/sec . . . 52

6.3 Shows the edge image for the sequence of frames 53

6.4 Shows that the markers have been successfully located and the number of markers found is displayed 54

6.5 Shows the first and last frames superimposed. The trajectories for each of the markers are displayed 55

6.6a Shows the trajectories of the spheres and its mirror image. The spheres were released at an angle of 15° 56

6.6b Shows the trajectories of the markers on the spheres 56

6.7a Shows the trajectories of the spheres and its mirror image. The spheres were released at an angle of 25° 58

6.7b Shows the trajectories of the markers on the spheres 58

6.8 Shows the sequence of edge images when the spheres were released at an angle of 10° 59

6.9 Shows the marker locations found in all the frames except in the 3rd frame, where only 9 have been found 60

6.10 Shows the superimposed image of the first and last frames. The trajectories for the feature points are displayed 61

6.11a Shows the trajectories of the spheres and its mirror image. The spheres were released at an angle of 10° 63

6.11b Shows the trajectories of the markers on the spheres 63

CHAPTER I
INTRODUCTION

1.1 Overview of the Problem

Analysis of a sequence of images is of primary importance to certain machine vision applications, where objects are to be detected and tracked over a period of time for later analysis. Meaningful measurements of their movements, such as linear and angular velocities, rotation and acceleration[1], can be obtained from these analysis. Some of the fields of application involving motion tracking include:

(a) **Computer vision:** detection, recognition and tracking of moving objects.

(b) **Industry:** robot vision and vehicle navigation.

(c) **Communications:** video signal compression using motion-compensated coding.

(d) **Biomedical imaging:** study of the heart and cell motion.

(e) **Meteorology:** examination of atmospheric processes using satellite images.

(f) **Transportation:** highway traffic monitoring.

Some difficult problems are encountered as the automatic analysis of moving images is highly dependent on the recording capabilities, facilities and environment in which the vision system is used. A few of the problems are listed below.

1) Moving objects tend to have a blurring effect on the recording tape or film and detecting the boundaries of these objects is

difficult.

2) As a large number of frames are to be processed, analyzing procedures must be efficient.

3) Shapes of objects could change from frame to frame and efficient algorithms are to be developed to identify it and achieve correspondence.

4) An inherent problem in the analysis of moving images is occlusion, since different objects move in different directions at different velocities.

1.2 Automatic Motion Analysis

Motion analysis can be conducted in a controlled, as well as uncontrolled environment. In either case, one needs extensive image processing techniques for the analysis of the images, which include pre-processing, edge detection, segmentation, Hough transform techniques for line and curve detection[2], finding features points to track and finding the correspondence of the points. The image processing task can be made simpler by using a high speed recording system and good lighting techniques.

The moving light display (MLD) technique[3] is quite popular in many motion analysis schemes. These can be produced by attaching small glass reflectors or fluorescent markers on the points of interest. A new technique is being developed where a remote tracer is tracked[4]. An emitter is embedded in the particle to be tracked. Receivers are used to pick up signals from the emitter, which is used to track the particle in 3-D. The emitter is an rf transmitter and the receivers are loops of coil (receiving antenna), which are magnetically coupled. Another approach is to mark white objects with some distinct black geometries like circles (spots), rectangles or any random shape which can be easily discernible by the vision system. Such strategies enable high contrast images to be obtained. Images recorded with a single camera provide the study of motion analysis in 2-D. Three dimensional motion analysis is possible if a second camera is strategically placed to record the motion.

Motion analysis can be generalized to consist of the following:

- Setting up the experiment, including lights, lenses, camera location, etc...
- Recording the movement of objects in space.
- Analysis of each frame to identify and locate the position of the object.
    - a) Noise filtering by applying a smoothing operator
    - b) Edge detection operator
    - c) Search the edge image for required objects
    - d) Find location information (position and orientation) of the object

1.3 Statement of the Problem

This thesis is part of an investigation into the study of particle collision, where the collision parameters are to be studied. Tracking of a particle in free space has some constraints due to the particle moving too fast or in an unsuitable environment.

The problem here is to detect and track, in a controlled environment, the motion of two spheres colliding in 3-D space. A high speed imaging system i.e., the Kodak Ektapro 1000, is used to record the collision of the two spheres at various angles of incidence. The imager is capable of recording at a speed of up to 1000 frames per second, with a maximum recording time of 35 seconds. The playback capability is at the rate of 30 frames per second, which allows for visually checking the quality of the images obtained, before further processing.

The purpose of the thesis was to set-up the imaging system with lights and mirrors so that good dual images are obtained. The spheres selected were white with randomly marked black spots on them. Programs were developed on the PC so that images acquired sequentially from the Kodak system are pre-processed to compute edges. This edge information is utilized to find the sphere centers using the Hough transform technique[5]. The marker locations were detected by a simple algorithm. Correspondence of the markers from frame to frame was achieved by minimizing the proximal uniformity constraint[6], which limits the search space. Trajectories of the markers are displayed on the monitor to check that correct correspondence has been achieved. Information of the locations of the markers and spheres obtained from frame to frame can be used to analyze the translational and rotational distances, velocities and accelerations.

1.4 Overview of the Remaining Chapters

The chapter 2 briefly surveys other techniques in the area of edge detection, Hough transform techniques, Fuzzy clustering methods and correspondence of feature points. Emphasis has been placed on the methods related to what has been done in this thesis.

Chapter 3 gives a little insight into the type of vision system used. Some salient features of the hardware involved and software used have been highlighted and a brief description of the experimental setup has been given.

Chapter 4 discusses the capabilities of the software Visilog, for accessing images, and processing images at the pixel level. A brief description has been given on some of the routines used in the program, to obtain the edge image, multiple circle detection using Hough transform, and finding the feature points or markers on the spheres.

Chapter 5 deals with the correspondence problem and its implementation. An algorithm proposed by Rangarajan and Shah[6], has been discussed in detail and a solution to occlusion has also been studied.

Chapter 6 presents the results of the work conducted. Conclusions drawn from the experiments have also been discussed. A note on direction for future research is also given.

Appendix A gives specifications of some of the high speed imaging equipment used.

Appendix B gives a brief description of the Visilog functions used in the programs.

Appendix C contains a listing of some of the programs used.

CHAPTER II
LITERATURE REVIEW

2.1 Edge Detection

Edge Detection algorithms drastically reduce the image content thus making post-processing of these images computationally less expensive. Extracting edges from digital images is an important aspect in any machine vision system, where it is necessary for object recognition, feature extraction, or for other image processing applications. A brief survey was made on some of the popular edge detection operators before choosing an appropriate combination. The two major classifications of edge detectors are based depending on whether they use first or second derivative properties.

2.1.1 First Derivative Edge Operators

2.1.1.1 L.G. Robert's Operator: L.G. Robert[7] was one of the first few who developed an edge operator which used a 2 x 2 region of pixels at each point. Estimates of the magnitude of the image gradient over a 2 by 2 region are perhaps the simplest edge operators. The Robert's magnitude operator R_1, estimates the derivatives diagonally, i.e.,

$$d_1 = f(i,j) - f(i+1, j+1) \quad \text{(E2.1)}$$

$$d_2 = f(i+1, j) - f(i, j+1) \quad \text{(E2.2)}$$

$$R_1 = [d_1^2 + d_2^2]^{1/2} \quad \text{(E2.3)}$$

Instead of calculating the square root of the sum of squares in the equation(E2.1), a computationally simpler operator is the Robert's absolute value estimate R_2 of the gradient given by

$$R_2 = |d_1| + |d_2| \tag{E2.4}$$

2.1.1.2 Sobel's Operator:

The Sobel's Operator is probably the most wide-spread image processing operator of all categories. According to Sobel[8,9], the idea is to estimate the gradient (f_x, f_y) employing the eight neighbor pixels with equal weight. However, because the corner pixels are further apart (with a factor $\sqrt{2}$) and because their difference vectors make 45° with the two main directions (another factor $\sqrt{2}$) they should contribute to both f_x and f_y with a factor of 2 less than the four pixels in the main directions. Therefore the Sobel operator pair (S_x, S_y) written as a convolution kernel is the following:

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

A digital gradient may be computed by convolving the two windows with an image, one window giving the x component f_x of the gradient and the other giving the y component f_y.

$$f_x(i, j) = S_x * n(i, j) \tag{E2.5}$$

$$f_y(i, j) = S_y * n(i, j) \tag{E2.6}$$

where n(i, j) is some neighborhood of (i, j) and $*$ represents the sum of the products of the corresponding terms. The Sobel operator is a nonlinear computation of the edge magnitude at (i, j) defined by

$$g_m(i, j) = \sqrt{(f_x^2 + f_y^2)} \tag{E2.7}$$

and its angle is defined by

$$g_a(i, j) = \tan^{-1}\left(\frac{f_x}{f_y}\right) \tag{E2.8}$$

2.1.1.3 Canny's Operator: Canny[10] has formulated an edge operator that follows three main performance criteria for edge detection. Assuming that edge detection is performed by convolving the noisy edge with a spatial function $f(x)$ and by marking edges at the maxima in the output of this convolution, three performance criteria are specified.

- Good Detection: There should be a low probability of failing to mark real edge points and low probability of falsely marking non-edge points. This criterion corresponds to maximizing signal to noise ratio.
- Good Localization: The points marked as edges by the operator should be as close as possible to the center of the true edge
- Only One Response to a Single Edge i.e., multiple response to edges must be avoided.

The mathematical forms for the first two criteria are easily obtained, which gives a product of the localization term and signal to noise ratio. Using the criteria for edge detection as a set of functionals of the unknown operator f, variational techniques are used to find the function that maximizes the criteria. These two criteria on their own are inadequate to produce a useful edge detector and the addition of the third criterion gives an operator that has a very low probability of giving more than one response to a single edge. It also provides a finite limit for the product of localization and signal to noise ratio. An analytic form for the operator was found, which is a sum of four complex exponentials and is approximated by the first derivative of a Gaussian. In one dimension the maxima in the output of this first derivative operator correspond to the zero-crossings in the output of a second derivative operator. The detector should be directional, the more directional the better. To obtain a good edge the directional output is used and the output of less directional neighbors is suppressed.

A non-maximum suppression scheme[11] was described, where the gradient magnitude is non-maximum suppressed in the gradient direction. This algorithm uses a nine pixel neighborhood as shown in the figure 2.1. The Interpolated Gradient is given as

$$G_1 = \frac{u_x}{u_y} G(x+1, y+1) + \frac{u_y - u_x}{u_y} G(x, y+1) \qquad \text{(E2.5)}$$

and in the opposite direction it is

$$G_2 = \frac{u_x}{u_y} G(x-1, y-1) + \frac{u_y - u_x}{u_y} G(x, y-1) \qquad \text{(E2.6)}$$

$P_{x,\,y}$ is considered maximum if $G(x,y) > G_1$ and $G(x,y) > G_2$.

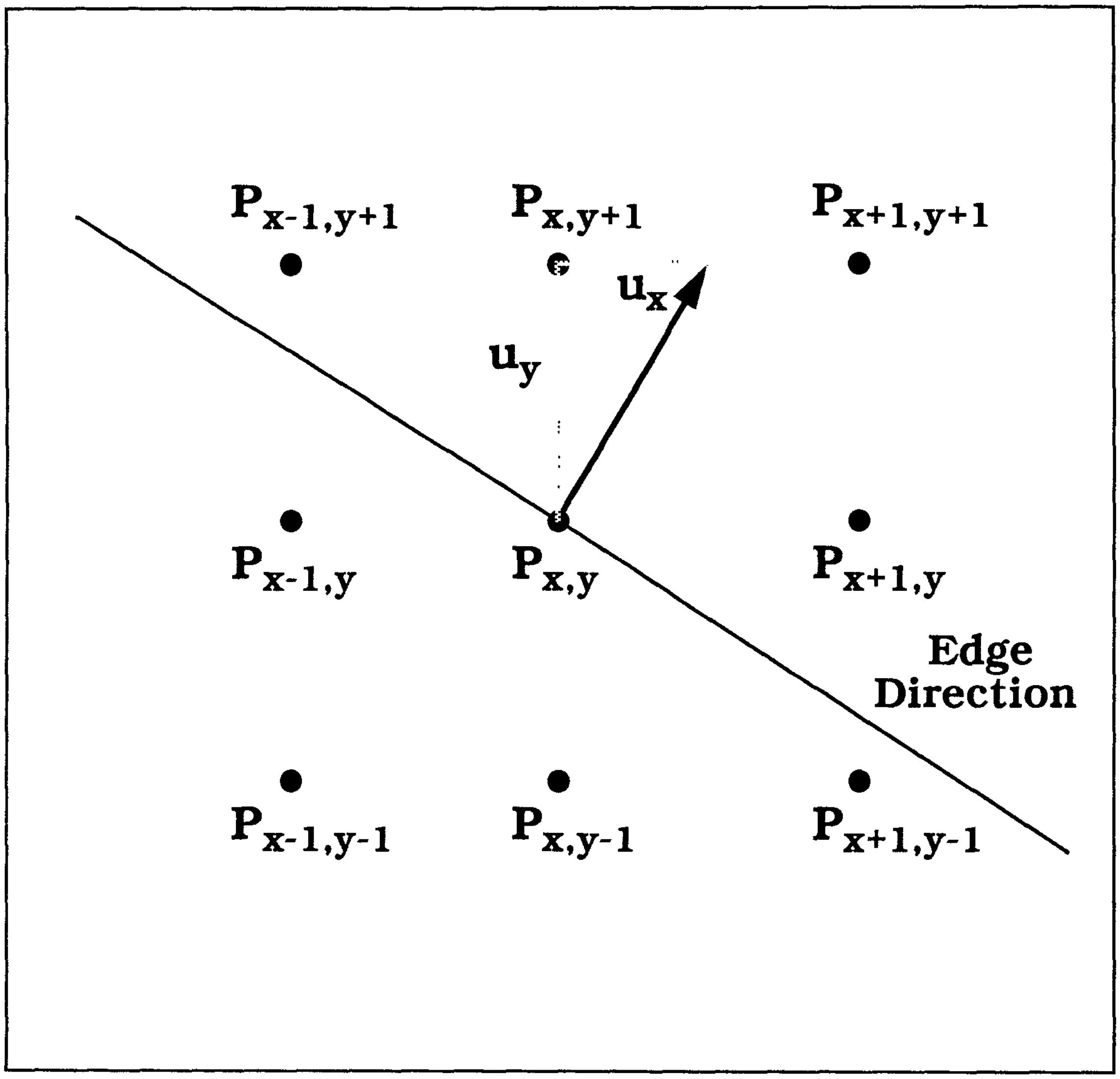


Figure 2.1 Canny's Non-Maximum Suppression Operator

2.1.2 Second Derivative Edge Operators

In this case, edges are detected as the point where the second derivatives of the image crosses zero. Marr operators[12] also called the “Mexican hat” operators are filters of the form $\nabla^2 G$ where ∇^2 is the Laplacian and G is the two dimensional Gaussian distribution. The main idea behind these filters is to first smooth the image with a Gaussian shaped filter and then find the edges (using the laplacian) in the smoothed image. One of the widely used operators in this class has been proposed by Marr & Hildreth[13].

2.1.2.1 Marr & Hildreth[13]**:** They propose using zero crossings of the operator D^2G (x, y) on the given image, where G (x, y) is a two-dimensional (2-D) Gaussian distribution and D^2 is the second derivative operator for detecting intensity changes in the image. The Gaussian operator is used to satisfy localization requirements in both the spatial and the spectral domains. Localization in the spatial domain arises because most intensity changes are spatially localized; hence, the output of the filter should be a smooth function of the nearby points. Localization in the spectral domain arises to reduce the range over which intensity changes take place. The Gaussian filter satisfies both these requirements. For the sake of reducing computations, the operator D^2G is replaced by the rotation invariant operator $\nabla^2 G$, where ∇^2 is the Laplacian. By using $\nabla^2 G$ with different widths, zero crossings of $\nabla^2 G$ at different scales are obtained.

2.2 Circle Detection Techniques

Some of the methods available for circle detection are Least square approach, Hough transforms and a more recent method is the Fuzzy clustering technique. These techniques are efficient in detecting circles but each has its own constraints. If edge data are available in the form of a list of the circle itself the least square technique can be applied, but the problem occurs when there is noise or if the objects are spread out over the image space. Fuzzy clustering techniques may be used to find circles. In objective functional based fuzzy clustering algorithms the weighted sum of the distances of the feature vectors from cluster prototypes are minimized. The fuzzy memberships are utilized as weighting factors. The cluster prototype can be a point or a line or a plane, etc.[28]. The Fuzzy c-shell clustering (FCS) method as introduced by Dave[29] assumes a cluster structure that is of some s-dimensional hyperspherical shells which are simply circles when s=2. Hypersphers refer to boundaries (surfaces for s>2). The prototypes do not include interiors-whence the word "shells" to describe the cluster prototypes. The FCS algorithm uses the Euclidean norm and as a result a certain measure of error results from measuring distances that are not Euclidean. This technique does have some problems like, it is sensitive to noise, sensitve to outliers, etc. The Hough transform is more robust and extra data in the image are not too much of a problem.

2.2.1 Hough Transform Based Techniques

The Hough Transform, HT was first introduced by P.V.C. Hough[14] as a method of detecting complex patterns of points in binary image data. Some of the many desirable features of this technique are described. Each image point is treated independently and therefore the method can be implemented using more than one processing unit. It can recognize partial or slightly deformed shapes. The HT method is very robust and random image points are unlikely to contribute much to the accumulator bin. The HT can also simultaneously accumulate evidence for several examples of a particular shape occurring in the same image. For the most part HT has been used to detect straight lines.

Kimme, Ballard and Sklansky[15] have shown that circular arcs can be detected and applied their technique to a medical image processing task. Essential to their method was the use of edge direction information to constrain the range of parameters addressed. Parabolas and ellipses have also been investigated by several authors[16,17].

Circles are usually described by three parameters their center coordinates (a,b) and its radius r. However, if we use extra information concerning the edge direction of image features then the circle detection problem can be decomposed into a two stage process which involves a 2-D HT to find estimates of the center parameters(a,b) [5], followed by a trivial 1-D HT or histogramming step, to find the best value of the circle radius. The circle center finding involves intersection of straight lines in the 2-D parameter space. Chapter 4 explains this concept in a little more detail.

There has been a reasonable amount of progress in techniques to achieve storage and computation savings. Illingworth and Kittler[5,18] have developed an iterative coarse-to-fine search strategy for detecting lines and circles in 2-D and 3-D parameter spaces. Their implementation is called the adaptive Hough transform, AHT. It uses a small accumulator array which is thresholded and then analyzed by a connected components algorithm. The shape and extent of the connected components determine the parameter limits which are used in the next iteration. Limits can be decreased, expanded, rotated or translated depending on the distribution of the counts in the accumulator. A simple example[5] involving searching for circles in a 3-parameter space showed that it was several hundred times faster than the standard method.

The HT has proved a valuable method in a large number of machine vision and related areas. It is a very robust method in the presence of extra data and can cope well with situations where some data are missing. Its major problems have been that in its simplest implementation it requires a lot of computation and a lot of storage for high dimensional arrays. However, it can be seen that a lot of research has been focussed on approaches that have led to very fast and space efficient digital implementations.

2.3 Motion Analysis and Feature Point Correspondences

Motion analysis and matching of feature points has always been an intriguing aspect of image processing. Motion analysis and matching of points are inter-related and the former cannot do without the latter. Motion analysis is possible only when feature points in one frame can be matched accurately with corresponding points in the next frame. Deriving motion information of an object from a sequence of images is a challenging research area in computer vision. Processing a sequence of images is definitely complex but the amount of information obtained, far outweighs its computational disadvantages. In this thesis more emphasis has been placed on feature point correspondence rather than the analysis of motion.

Ullman[19] has proposed a minimal mapping theory for correspondence. His approach is probabilistic in nature and he assumes that each point is moving independent of every other point. One limitation of this assumption, is that if the points being tracked belong to the same object, they move as a rigid structure, thus violating the independence assumption. But, when the points in a frame belong to different moving objects, the points on different objects move independently of each other and the minimal mapping theory works well. At low velocities the cost function used by Ullman reduces to the distance.

Jenkin[3] presented a method for tracking the 3-D motion of points from their 2-D images as viewed from a nonconvergent binocular vision system. This scheme used the concept of velocity smoothness. Position and velocity of points in 3-D were tracked, given the initial 3-D positions and velocity of the points and a sequence of images. Two frames were considered at any instant of time and each frame had a left stereo image and a right stereo image. The stereo correspondence and the velocity in the first frame are known and the stereo correspondence and the velocity in the next frame are to be determined. A greedy strategy was used to obtain the best solution.

Barnard and Thompson[20] use an iterative algorithm to match feature points selected in two different frames taken in a small time interval. Initial probabilities for matches between pairs of points are made based on a common motion heuristic. It restricts the potential match for a point, assuming that a point does not move a large distance between frames. The probabilities are then refined to strengthen common motion of neighboring points. The algorithm terminates when all the probabilities of matches between pairs of points are close either to 1 or to 0.

Sethi and Jain[21] have proposed two iterative algorithms GE (Greedy Exchange) and MGE (Modified Greedy Exchange), which minimize the cost function called path coherence. Initial correspondence is assumed to be known between frame 1 and 2 in GE. Then it extends the trajectories frame by frame starting with an assumption that the nearest match is a correspondence. An iterative loop exchanges correspondences which improves the cost function value. This process continues till correspondence in all the frames is established. The difference between GE and MGE is that in MGE initial correspondence is not assumed and the process is repeated in the forward and backward directions, which could alter the initially assumed correspondence.

One of the more recent advances in this area has been by Rangaragan and Shah[6] who propose a proximal uniformity constraint to solve the correspondence problem. According to this constraint, most objects in the real world follow smooth paths and cover a small distance in a small time. Therefore, given a location of a point in a frame, its location in the next frame lies in the proximity of its previous location. An efficient non-iterative algorithm is proposed which minimizes the proximal uniformity cost function and establishes correspondence over a sequence of frames. The assumption in this approach is that initial correspondence in the first two frames is known. A more detailed description of this approach is discussed in chapter 5.

CHAPTER III
DESCRIPTION OF THE VISION SYSTEM

3.1 Hardware Used

The **Advanced Frame Grabber (AFG)** is a high performance member of Imaging Technology Incorporated's VISON*plus*-AT family of board-level image processors. The AFG is a two-slot, PC AT based image processor that combines many important elements for advanced image processing applications. The AFG is a high-resolution video digitizer, frame memory and image processor capable of digitizing and displaying RS-170, CCIR and non-standard video images. The digitized images are stored in a special on-board image memory. The AFG includes the Texas Instruments TMS34010 Graphics Subsystem Processor (GSP) for fast graphics, image processing and display control.

The AFG is a two board set, that plugs directly into two adjacent expansion slots in the PC AT. The AFG digitizes the incoming video signal to eight bits of resolution. The AFG supports non-standard image sources and line scanners, as well as standard RS-170 and CCIR video sources. The AFG memory is 1024 by 1024 pixels by 16-bits and stores 8-bit, 12-bit and 16-bit image data. The AFG provides a separate 512K byte auxiliary memory for the GSP. The image memory and auxiliary memory are both mapped in the GSP address space. This allows the images to move from one memory to the other to gain additional processing space or continue processing in parallel with acquisition.

The **Kodak Ektapro 1000** motion analyzer consists of an imager, processor, controller and a monitor. The **imager** has an image intensifier assembly behind the lens and in front of the sensor. The image intensifier functions as an electronic shutter and light amplifier. This increases the imagers ability to capture events in low light and to reduce the blurring of objects moving through the field of view rapidly. The imager converts the light entering the lens into an electrical or video signal. The video signal created in the imager is amplified and processed so that it can be transmitted through the imager cable to the processor. The main component of the imager is the sensor which is a "solid state imaging array". The motion analyzer achieves a frame rate of 1000 frames per second by scanning sixteen rows of pixels simultaneously, for a full frame image of resolution 239 x 192.

The **Ektapro 1000 processor** contains an electronics card bin, a tape transport and the power supplies to the system. The processor card bin contains eleven printed circuit cards which are required to control the system and process the video. The keypad is used to set the operating parameters and configure the system for the desired mode of operation. The system operates in either the live, record or play mode.

The **controller** provides power and control signals to the image intensifier assembly. The controller is powered by 110 volts AC and receives a synchronizing signal from the strobe trigger output of the processor. The gain and gate can be adjusted from the controller to vary the light amplification and amount of time the electronic shutter is open during each frame respectively.

The **monitor** is connected to the processor using the coaxial cable supplied with the system. The resolution of the monitor is 239 x 192

The **Kodak Communication Interface (CI)** sets the communication protocol for control and access with a computer or a terminal which supports the RS-232-C protocol or IEEE-488 (GPIB) protocol. The CI board supports both serial communication and parallel communication. The CI provides a means of externally controlling the EKTAPRO 1000 motion analyzer and accessing the internal data.

A **486 PC** is used for communicating with the processor i.e., to issue commands, download and upload images, pre-processing the images using Visilog and to compute the correspondence.

3.2 Software Used

Visilog[22] is a computer vision software from Noesis which incorporates both image processing and analysis libraries and a command monitor. The standard Visilog distribution includes

- Image processing and analysis algorithms libraries
- A set of development tool libraries, to create or access images, control display, etc...
- The source code of a command monitor, which can generate the executable task associated with this monitor
- A number of source code of basic programs to illustrate the development of new functions of Visilog
- Technical documentation

This software package can run on a large number of system configurations, from simple PC or PC-compatible systems to scientific workstations and mainframe computers. A typical system configuration is shown in the figure 3.1. The system configuration to run Visilog and its monitor is composed of:

- a computer running on MS-DOS, a Microsoft C compiler and a disk to store data
- a display device which is an additional image display board supported by NOESIS
- a control terminal from which to issue commands to the system with a mouse device

- An acquisition device, i.e., camera input and image digitization

The system configuration for the system used in this set-up is shown in figure 3.2. The Visilog package manipulates images, a set of two-dimensional data stored on disk files or in the image memory. These images are visualized on the display memory and its associated display screen. This display memory is just a window of the image memory available to display data and processed results. In our case the display memory is identical to the image memory as they are actually the same additional board.

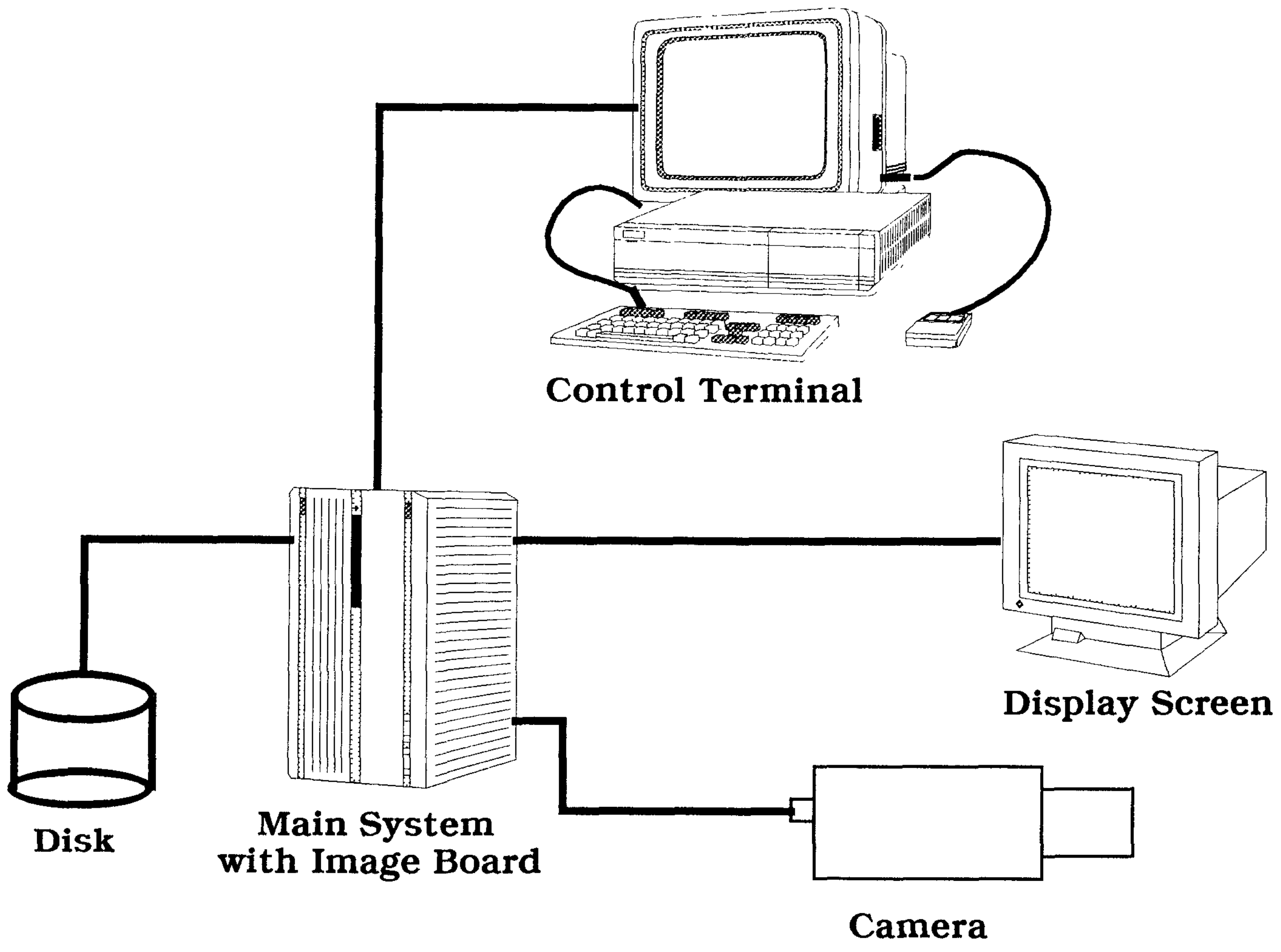


Figure 3.1 A Typical System Configuration

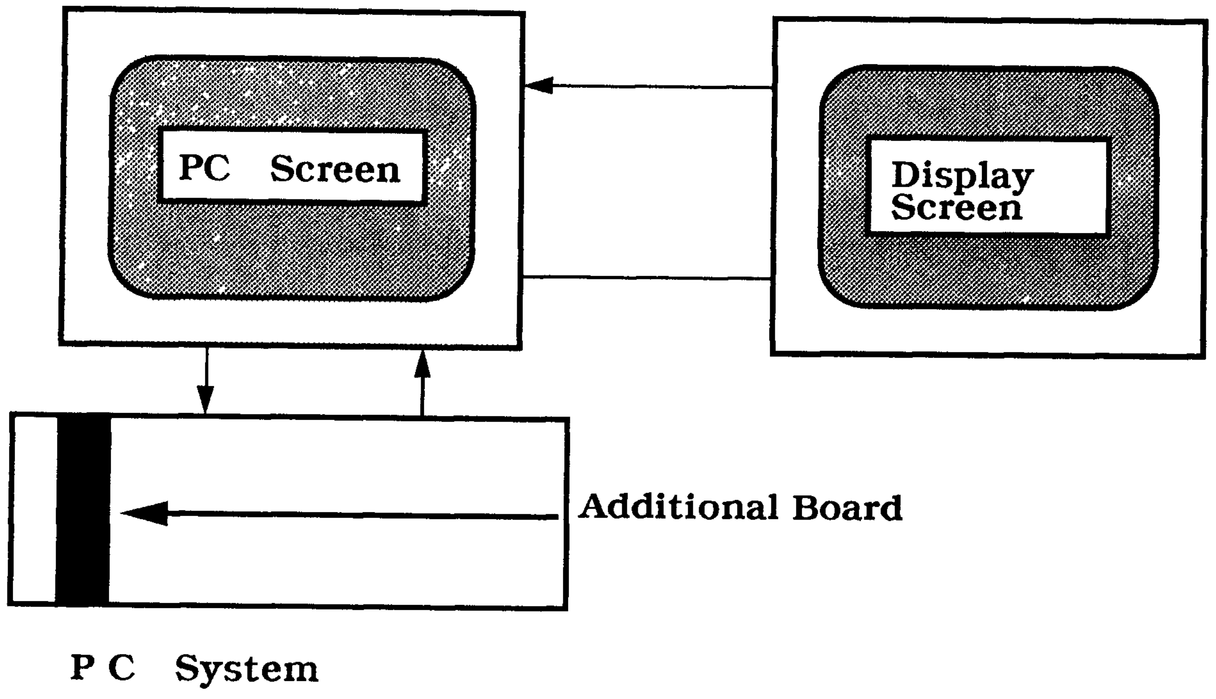


Figure 3.2 System with Imaging Board

3.3 Description and Working of the Setup

3.3.1 Description

Giving a brief description of the experimental set-up, it consists of the following:

- Two tubes wider than the spheres with settings to vary the angle.
- Two spheres with randomly placed markers on it.
- Two solenoids mounted on the tubes and connected to a triggering device.
- A mirror placed behind the expected collision space at an angle of approximately 45^0.
- Suitably placed lights to illuminate the collision area.
- A computer to control the triggering device, recording and to process the downloaded images.
- A high speed imaging system i.e., Kodak Ektapro 1000, with its accessories like the camera, processor, controller, etc.

3.3.2 Working:

The experimental set-up has been designed keeping certain constraints in view. Collision of the two spheres is expected every time they are released in an obstacle free environment. There must be provision to vary the angle at which the spheres are released. The spheres must roll through the tubes smoothly with little or no friction from the tubes. The triggering mechanism must release both the spheres at the same time instant to enable collision.

The tubes are made of plastic and they are wider than the spheres to provide smooth rolling. The tubes are mounted on stands that are capable of rotating about their base and the tubes themselves can also be rotated and set at the desired angle. A mirror is placed right behind the collision space at an angle of 45^0 to obtain the top view of the spheres. Solenoids mounted on the end of the tube allow the sphere to rest at the beginning of the tube before release. The solenoids are connected to a triggering device with a +24 volt supply. The triggering device can be activated by the computer using the software VID2 [23]. This triggers the solenoids which releases the spheres and after a short time interval activates the recorder.

The imager is placed such that the collision area is in focus and the mirror image is also within the field of view. The lighting arrangements are made so as to provide a clear image with little or no glare and proper distribution of light over the set-up. The camera is connected to the processor and the controller. A communication link exists between the PC486 and the processor. Images can be observed on the monitor. The communication network of the various components are shown in a schematic view in the figure 3.3. The recording speed of the imaging system is set at 1000 frames per second and it has a playback capability of 30 frames per sec. It can be further slowed down by jogging the frames step by step. This allows the collision of the spheres to be observed at an extremely slow rate.

After the collision of spheres has been recorded, it is replayed and a sequence of image frames before and after collision are selected for image processing. These image frames are transferred from the tape to the PC486 disk using VID2[23]. Now the images are processed individually to find the edges, 2-D circle and marker coordinates, which is the main area of interest in this thesis. The marker coordinates are later matched from frame to frame to track the trajectories of these markers.

The Kodak Ektapro imager has a frame resolution of 239 x 192, which is scanned from left to right, bottom to top, row by row. As a result the coordinates of the image begin at the lower left hand corner, with the origin starting at (1, 1).

Images in Visilog are read differently from the imager i.e., the image is read from left to right, top to bottom, row by row. This puts the origin at the top left hand corner, with the origin starting at (1, 1). This has got to be kept in mind when transferring images from the Kodak Ektapro to the Visilog system.

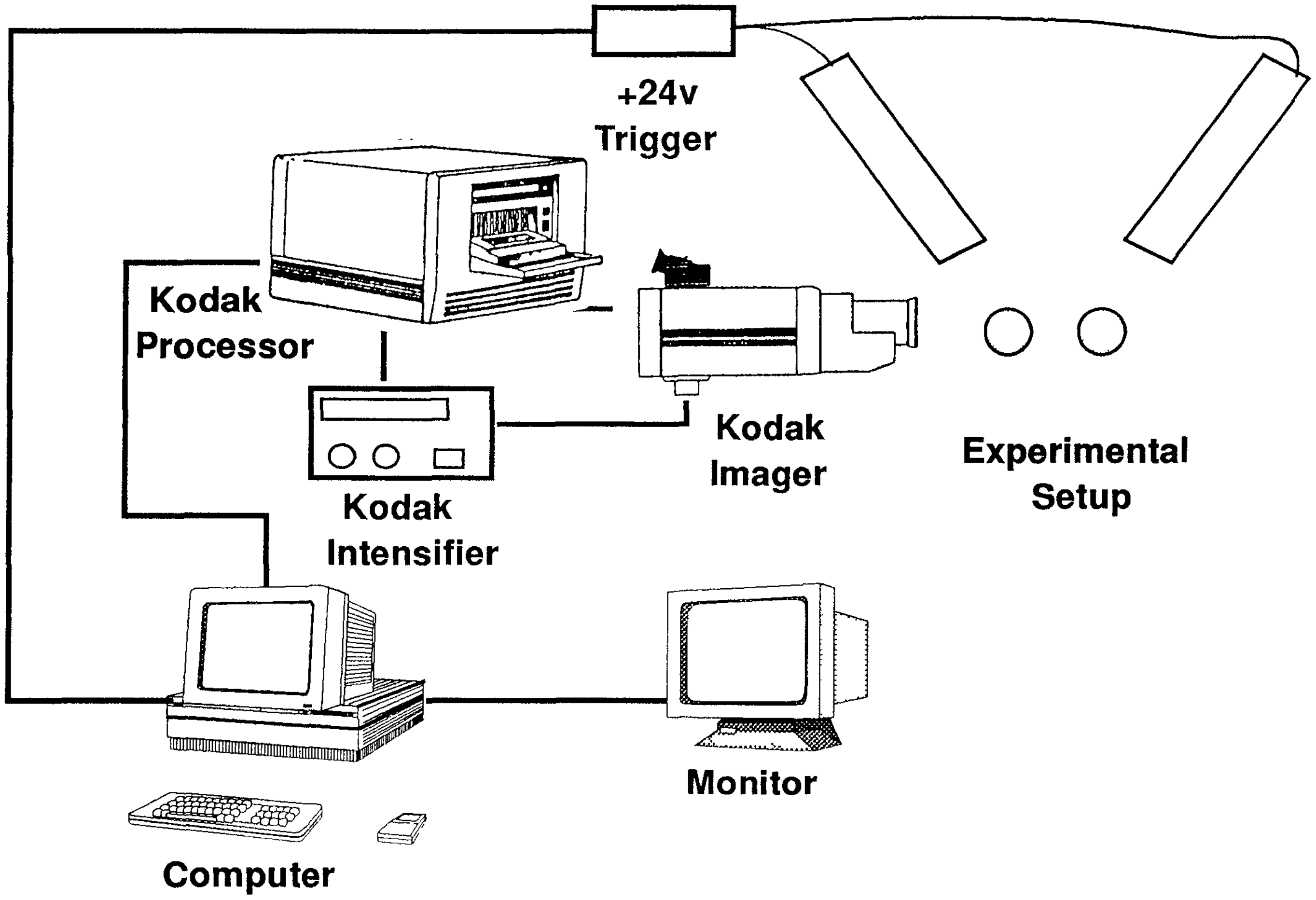


Figure 3.3 Schematic view of the communications and experimental setup

CHAPTER IV
PREPROCESSING USING VISILOG

4.1 Introduction to Visilog

Visilog[22] is a computer vision software package which incorporates both image processing and analysis libraries and a command monitor. This package can be used as a set of libraries, which can be combined by the user to perform certain specific vision tasks and to develop standalone programs more specific to the users requirement or area of interest. It can also be used as a Computer Vision development environment, where over two hundred Vision operations are activated using the command interpreter.

Standard image processing tools, such as arithmetic and logic operations between images, display and acquisition control, convolution and elementary high or low pass filtering and standard edge detection algorithms are included in this package. It also incorporates some of the most advanced vision tools, ranging from *Mathematical Morphology* operations to *Modern Edge Detection* schemes. A more detailed description of Visilog is given in the appendix.

Standalone programs have been developed in this Thesis which provide a greater flexibility in performing the vision functions, allows new image processing tasks to be coded in 'C' and also removes the necessity of the user to key in commands after every step through the interpreter. Standalone programs are designed to run outside the interpreter context, with a minimum of interaction with the user. There is a set of global variables which describe the display configuration, the image memory size and content, etc... These variables are necessary and must be initialized prior to any call to a Visilog routine. Such initialization takes place in the starting phase of Visilog and has been grouped in one function, which must be called prior to any image access or operation in a standalone program.

Giving a brief description of the computer implementation of the Visilog functions, the program can be considered as divided into three main subroutines with other supporting subroutines. The three main subroutines being

- Edge( )
- Hough( )
- Marker( )

Before these subroutines can be executed there are certain procedures and functions to be followed which are a vital part of the program. As mentioned earlier the function to be called before any image processing operation can be performed is *stdaln_( )*. When this function is called a set of global variables are automatically allocated and initialized. The declaration of this function is:

```
stdaln_(flag,argc,argv)
long *flag;
int argc;
char *argv[ ];
```

The flag pointer points to 0 for standalone applications. The argc, argv arguments are the standard command line processing arguments of 'C'. Once *stdaln_( )* is called, the application program knows of the image memory, the display memory, their type or configuration and will permit access to images, acquisition display and processing functions.

4.2 Accessing Images in Visilog

4.2.1 Image

Images are formed of picture elements or *pixels* of a specific arithmetic format. The data is organized according to some pre-defined rules which must be specified. In Visilog the images are considered to gather three classes of information:

- **Image Header,** describing the arrangement of the image information, type of arithmetic format, image size, etc.
- **Image User Header,** user stored customized information.
- **Image Pixels,** stored as rows of consecutive pixels.

Whenever image data is to be accessed, the headers and the pixel information are also simultaneously accessed.

4.2.2 Accessing Images

Accessing images is very similar to the standard file access routines in 'C'. The task is to open the files, read the input file, modify the information, write the new information to the output file and then close all the open files. Images are referenced by their name from the user point-of-view. To access images in programs an *image handle* is obtained, which is associated to the image name. An *image handle* is a new 'C' object which is declared as

IMAGE   nf;

Each time a valid *image handle* is retrieved, information to the image header and pixels is simultaneously accessed. *Image handles* are opaque structures defined in the *visilog.h* include file. An image handle returns a pointer to a structure of type image, that is a pointer to some place in the CPU memory of the system. This pointer is not initialized and as such cannot be used. When functions like *image_( )*, *xgrab_( )* or *dupnf_( )*, are called enough memory space to hold a structure of type image is first dynamically allocated, then the structure fields are filled with proper values, then the valid pointer to that allocated space is returned, which can be used. One of the functions which returns a valid *image handle* important to the program is *image_( ).* The declaration of this function is:

```
IMAGE      image_(name,mode,verify,control)
char       *name;
char       *mode;
char       *verify;
IMAGE      *control;
```

The argument *name* points to the name of the image.

The *mode* argument specifies the type of access required for the image and contains one of the list **[e,s,c,t]**.

- "**e**" specifies that the image already exists.
- "**s**" specifies image to be created if it does not exist or over write if it does exist.
- "**c**" specifies image will be created even if it already existed.
- "**t**" specifies that a temporary image is created and will be destroyed when the handle is released.

The *verify* argument specifies type of verification to perform on the size and arithmetic format of the image, an empty string " " checks nothing.

The *control* argument is a free *IMAGE handle* used to specify the image header information. Free IMAGE handles are obtained by calling the dupnf_( )function, such as

control = dupnf_(NULL);

Once the *IMAGE* handles have been obtained it is important to release these handles. To release the *IMAGE* handle the name of the image is passed by address to the function *fermnf_( ).*

Accessing images in Visilog can be generalized by paying attention to the following points:

- Declare image handles

  IMAGE handle;

- Retrieve proper image handles

  handle = image_(..........);

- Release handles when no longer required

  fermnf_(&handle);

4.3 Program Description

4.3.1 A brief description of the algorithm and functions used

1) Initialize( ): Initializes and clears the image memory.
2) Check(name): Assuming that a maximum of 20 frames are stored in the hard-disk a check is made to see if the sequence of images named exists or not.
3) Starting(name): Starts a loop which sends the name of the images to be processed one at a time till all the frames are processed for a maximum of 20 frames.
4) Process(name,name1,frame): This passes the name of the image to functions where the images are called and processed.
5) Flush_marker( ): Array locations which store the coordinates of the centers of the spheres and markers are initialized to 0.
6) Edge(name,frame): Image from the disk is called and its edges are obtained.
7) Hough(image,frame): Multiple circle detection is done using the hough transform technique for a maximum of 4 spheres.

8) Marker(name,frame): Markers in the image are detected in the lower two spheres.
9) Check_points(name,frame): Markers obtained in the previous function are superimposed on the original image to make a visual check for accuracy.
10) Storing(name,frame): In this function the coordinates obtained for the centers of the spheres and the markers are stored in two separate output files.

A listing of the programs used is available in the appendix. Of all the functions used in this program only a few need to be explored in detail.

4.3.2 Initialize( )

Initialization must be done before proceeding onto the image accessing/processing routines. This is accomplished by the function *initialize( ).* The 256 x 256 image memory partitioning is selected and the Visilog default image format and image tables are initialized. The display memory and the CPU memory are cleared and any images stored in this location are lost.

4.3.3 Edge(char* name,int frame)

The name of the image and the frame number being processed are passed as arguments to this function. The first step performed in this function is to declare all the *image handles* necessary within the function. A control image structure is created with a call to *dupnf_( ),* and since no reference image is needed the argument to this function is NULL and default image parameters are assigned to that structure. An *image handle* is declared to obtain an existing image from the disk and another is created to store this image into the display memory with calls to the *image_( )* function. Linear black and white look-up tables are defined with a call to *xlutst_( )* and the zoom factor is set at two using *xzoom_( )* i.e., a 256 x 256 window is set. This allows the image to be displayed over the whole screen of the Visilog monitor, as the image size obtained from the disk is 239 x 192. The image from the disk is copied into the image memory with the *scopy_( )* function and is displayed onto the monitor with the function *xvisu_( )*. As the control structure with default parameters is no longer required the control image handle is released with a call to *fermnf_( )*. Another control structure is created, with the image obtained from the disk passed as an address to the *dupnf_( )* function so that its physical parameters are referenced. Other image handles are created with this control structure and passed as addresses to the function *sedge3_( )*. This function performs an elementary edge detection with the Sobel[9] mask. The X gradient and Y gradient images are computed from the input image. This function does not yield a gradient amplitude image which is then obtained by using the function *scmpas_( )*.

Next a non-maximum suppression function is applied to get a good edge image following a method proposed by Canny[11]. This function is called *smxsup_( )*, it takes the X gradient image, Y gradient image and the gradient amplitude image as the input and returns a pointer to the output edge image. The X and Y gradient images are used to determine the gradient orientation and only points for which the gradient amplitude is a local maxima are marked as an edge. Resulting edges retain the original gradient amplitude and form curves and lines of thickness one. The resulting edge image can be displayed on the monitor with *xvisu_( )*. A threshold is set on the image so that only relevant edge points are retained and the rest can be deleted. This helps to remove all unwanted data and any extraneous information that is not required can be eliminated, drastically reducing the image content, thus making post-processing of these images computationally less expensive. The function used in this case is called *sthr_( )*, which returns a binary output. As the output is binary, the control handle must be reset with the correct arithmetic format using *set_code( )* and an image handle assigned to the output image. The input image is the edge image from the previous function and an array of low and high thresholds are the parameters to this function. The output points of the binary image obtained are defined as:

O(n,m) = 1 iff low <= I(n,m) <= high, *I(n,m) is input image*

O(n,m) = 0 otherwise. *O(n,m) is output image*

This gives a good edge image of the spheres with the markers on it. Since the images have been recorded in 2-D the spheres are represented as circles in the image. The edge image obtained in the above procedure is stored onto the disk so that it can be accessed by other routines and the image memory is not overloaded. The subroutine *hough( )* is now called for the detection of circles. All the image handles which were used within this function now have to be released with individual calls to the *fermnf_( )* function to allow the memory locations to be free for the next operation. The figure 4.1 shows a good example of an image and the results obtained using the Sobel's operator, non-maximum suppression and thresholding.

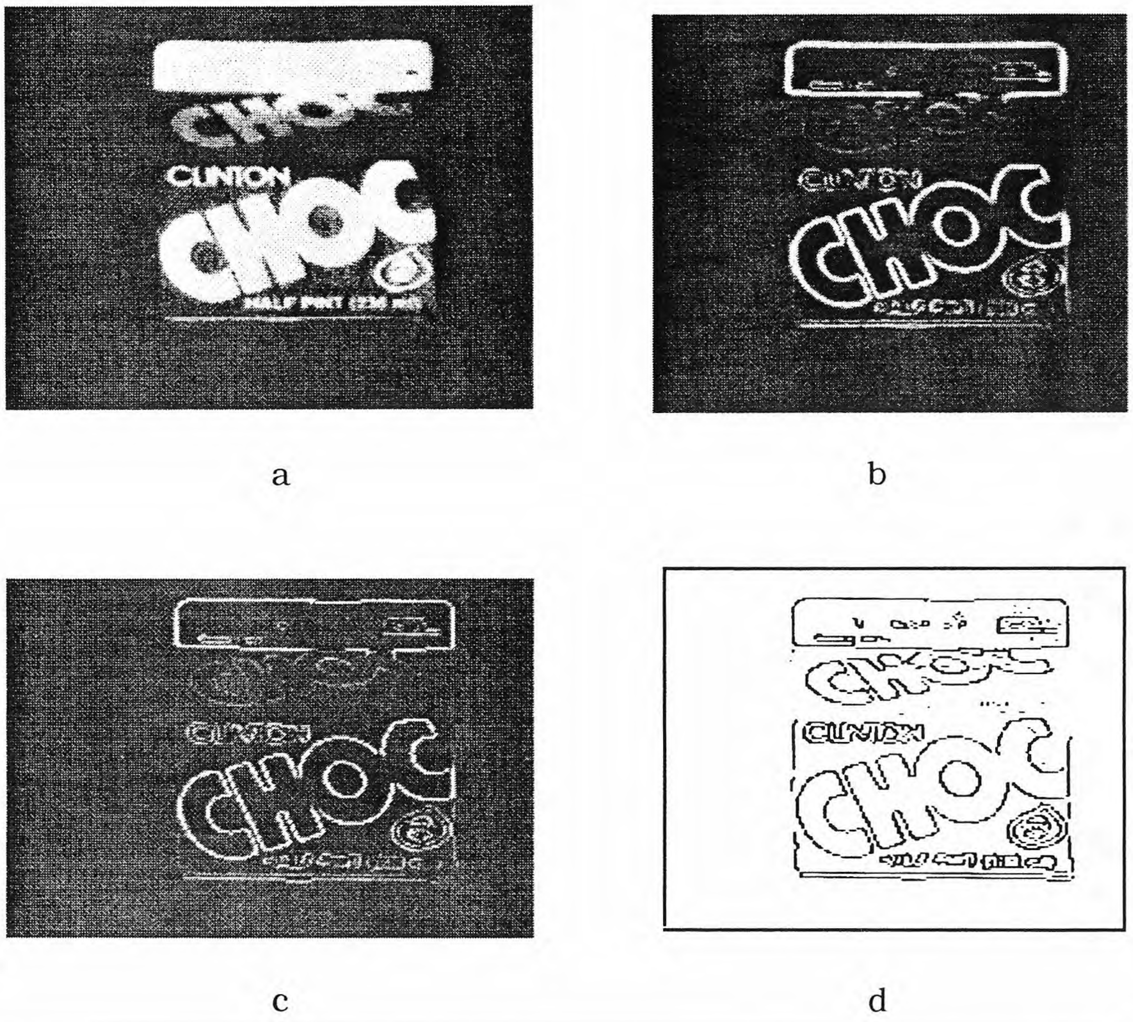


Figure 4.1 a) Shows the grey level image
b) Image after Sobels operator
c) Image after Non-Maximum Suppression
d) Edge image after thresholding

4.3.4 Hough(input,frame)

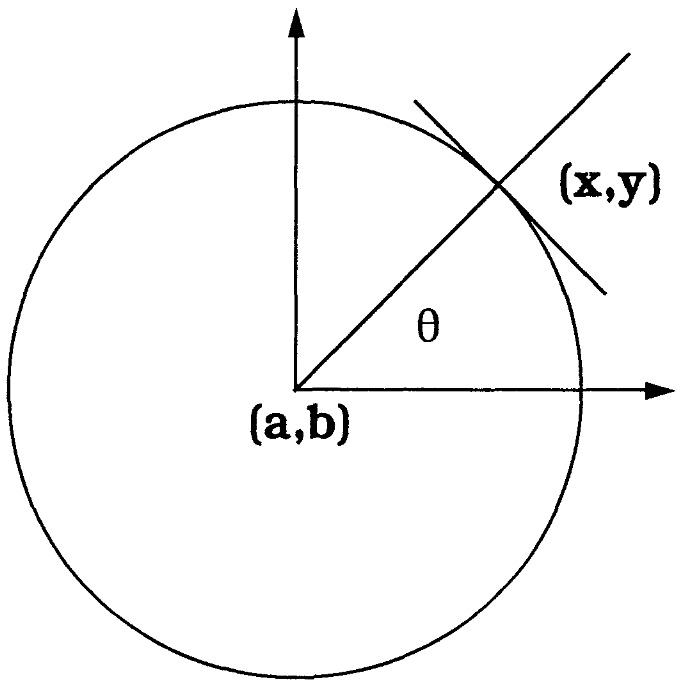
The edge image and the frame number being processed are passed as arguments to this function. This subroutine detects the centers of circles using the Hough Transform technique[5,24]. In this routine the images are accessed at the pixel level as compared to the functions used so far which take advantage of the elementary Visilog routines. Image handles are declared and a null control structure is created. The edge image stored in the disk in the previous function is recalled for further processing.

To locate the circle centers we incorporate a constraint that the vectors which are normal to the circle boundary must all intersect at the circle center(a_0, b_0). Estimates of these normal directions are obtained from local gray-level edge detection operators, i.e., the Sobel operator. The figure 4.2 illustrates how the knowledge of (x, y, θ) leaves only (a, b) as parameters. Mapping (x, y, θ) triplets into 2-D parameter space produces a straight line. The intersection of many of these lines identifies the circle center coordinates.

Since the circle finding problem has been formulated to produce a two stage algorithm which involves a 2-D HT to find the estimates of the center parameters (a, b), the accumulator array can be considered as a 2-D image of the Hough transform. This algorithm has been implemented in this thesis.

An image is created to store the 2-D accumulator array and initialized to zero. A check is made in the edge image for the edge points using the function *readpx_( )* and at every edge point the X and Y gradients are computed from the original image and the direction angle is found. For the image parameter range the HT is accumulated and every increment is updated in the accumulator image using the *writpx_( )* function. The accumulator image is analyzed for local maxima and its coordinates give the center of the circle.

This method can be extended to detect multiple spheres in an image. After the first center has been found, a region of 10 x10 pixels are reduced to zero around that center, and a search for maxima is made again. This can continue for a predetermined threshold value below which no circle is assumed to be detected. The robust nature of the HT algorithm is seen in figure 4.3 where the presence of noise does not affect the center of the circle and figure 4.4 shows that multiple circles are detected quite accurately



$$\mathrm{Tan}\ \theta = \frac{y - b}{x - a}$$

$$b = \mathrm{Tan}\ \theta\ \ a + (y - x\,\mathrm{Tan}\ \theta)$$

Figure 4.2 Relationship between (x, y, θ) and center parameters (a, b) for a circle.

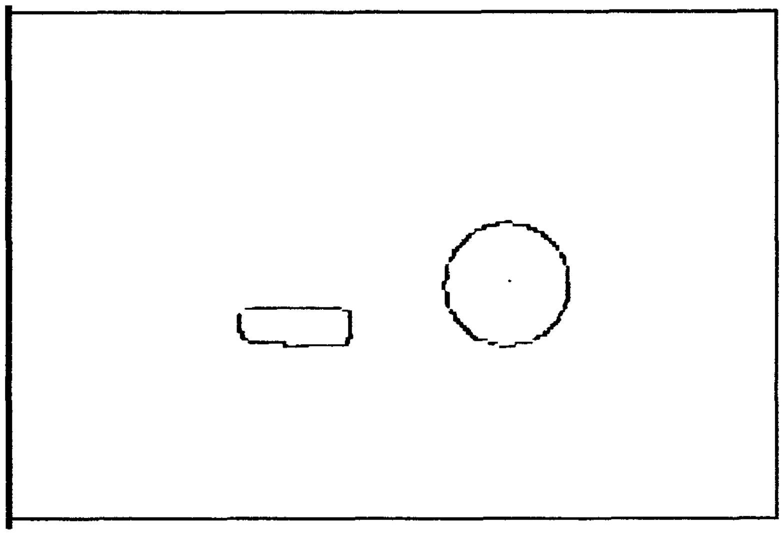


Figure 4.3 Shows the circle center using Hough transform

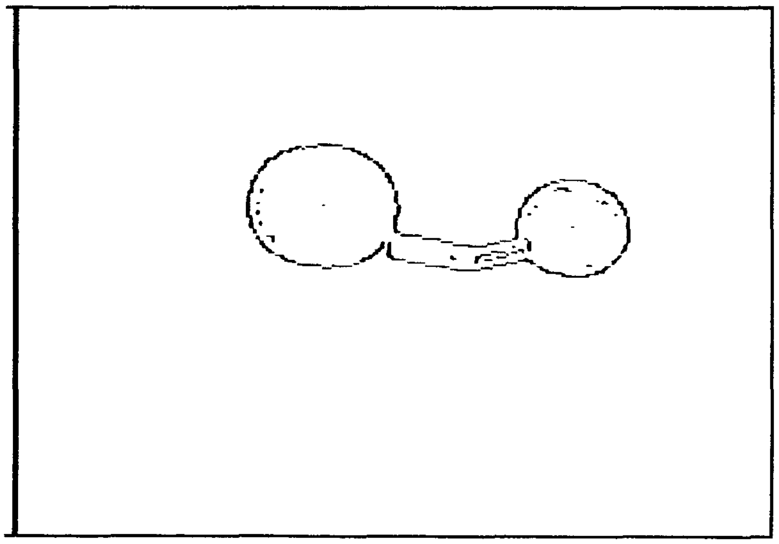


Figure 4.4 Shows multiple circle detection

4.3.5 Marker(char* name,int frame):

The name of the image and the frame number being processed are passed as arguments to this function. All the *image handles* and variables necessary within the function are declared. The main aim of this routine is to locate all the markers on the lower two spheres in the image which is accomplished with a simple heuristic approach. The display memory is cleared with a call to *visclr_( )* and a control image structure is created with a call to *dupnf_(NULL)*. The thresholded image stored in the disk, in the function edge, is accessed and displayed. Extra data in the image is eliminated with a call to *elimin( )* which deletes all information outside the radius of the circles whose coordinates were found in the function *hough( )*. Labelling of the image is done for which the parameters of the control structure is changed by *set_code( )*. The labelling operation is performed by *slabel_( )* with a binary image as its input. Starting from the binary image, it gives a grey level image where all the pixels of a connected component have a unique grey level value, thus labelling each connected component. The number of connected components is computed from the labelled image using the function *snumbe_( )*.

Now that the connected components are labelled, each labelled object is analyzed separately to find the markers. Separating every labelled object for analysis, from the main image poses a problem. This problem is alleviated due to the fact that every labelled component in the labelled image has a unique grey level which can be thresholded. A simple algorithm is incorporated at this point to locate the markers.

- For i = 1 to number of components do
- Threshold to separate the object of component number i
- Find area of the component i
- If area lies within the specified range do
- Compute inertia moments of component i
- Check if the component lies within the bounds specified

- Check if the component location is within the radius of the lower two spheres
- If above conditions are satisfied the component is one of the required markers
- Increment component number and begin loop again

A temporary image is created which can store a binary image. The labelled image is thresholded with a call to *sthr_( )* and the number of the component itself is used as the low and high level of the threshold. This gives us an image of that particular numbered component stored in the temporary image handle as a binary image. The first component is now separated from the labelled image and is available for further processing. The area of the component is found by using the function *sarea_( )* which computes the surface area of the binary image as the sum of non-zero pixels. Area of the component is now checked against the range of size of the markers. If within the range then *sinert_( )* gives the inertia moments of the binary image which are computed as below:

$$\text{moment}[0] = \frac{1}{N\Sigma X_n}$$

$$\text{moment}[1] = \frac{1}{N\Sigma Y_n}$$

Where N is the number of non-zero pixels, (X_n, Y_n) the coordinates of such pixels. A check is made to make sure the component is bounded within a certain box and the location of their X and Y coordinates are within the radius of the two spheres. When these conditions are satisfied, the component is termed as a marker and its location is stored. The figure 4.5 shows that marker locations have been located.

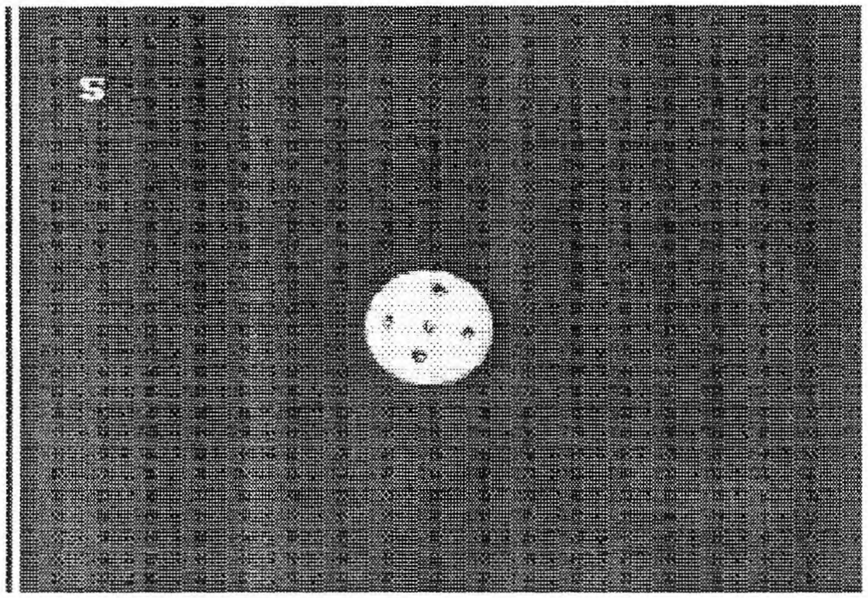


Figure 4.5 Shows the markers and displays the number located

CHAPTER V
ESTABLISHING CORRESPONDENCE

5.1 Definition

Given n frames taken at different time instants and m points in each frame, the problem of motion correspondence is to map a point in one frame to another point in the next frame such that no two points map onto the same point[6]. This definition of the problem and the method for its solution as proposed by Rangarajan and Shah has been studied in this thesis.

5.2 Proximal Uniformity Function

The aim is to obtain a one to one correspondence Φ^k between points of the kth frame and the $(k + 1)$th frame. It is assumed that objects in space move a small distance in a small amount of time and their motion is smooth or uniform. Therefore the location of a point from one frame to the next will be in the proximity of the previous location and the objects are assumed to follow a proximal uniform path. A *proximal uniformity function* δ, was proposed which obeys the following criteria.

- In the two successive frames selected the speed of the objects does not change much.
- Direction in the two successive frames do not change much.

- There is very small displacement between any two successive frames.

The proximal uniformity function is defined as follows:

$$\delta(X_p^{k-1}, X_q^k, X_r^{k+1}) = \frac{\left\| \overline{X_p^{k-1}, X_q^k} - \overline{X_q^k, X_r^{k+1}} \right\|}{\sum_{x=1}^{m} \sum_{z=1}^{m} \left\| \overline{X_x^{k-1}, X_{\Phi^{k-1}(x)}^k} - \overline{X_{\Phi^{k-1}(x)}^k, X_z^{k+1}} \right\|}$$

$$+ \frac{\left\| \overline{X_q^k, X_r^{k+1}} \right\|}{\sum_{x=1}^{m} \sum_{z=1}^{m} \left\| \overline{X_{\Phi^{k-1}(x)}^k, X_z^{k+1}} \right\|} \qquad \text{(E5.1)}$$

where $1 \le p, q, r \le m$; $2 \le k \le m - 1$; $q = \Phi^{k-1}(p)$; $\overline{X_q^k, X_r^{k+1}}$ is the vector from point q in frame k to the point r in frame $k + 1$ and $\|X\|$ denotes the magnitude of the vector X. The ith point in the jth frame is denoted by the vector X_i^j in 2-D coordinates.

The first term in the proximal uniformity function represents a relative change in velocity and the second term represents a relative displacement. The numerator in each term is an absolute quantity and the denominators represent sums of absolute quantities for all possible matches. The first term takes care of smooth and uniform trajectories and the second term forces proximal matches.

A major assumption being made with this proximal uniformity function is that Φ^1, an initial correspondence, is known. Knowing the initial correspondence the algorithm correctly marks the trajectories of the points from frame to frame. The correspondence Φ^k is determined by minimizing the cost function $\sum \delta(X_p^{k-1}, X_q^k, X_r^{k+1})$.

5.3 Implementation of the Algorithm

It is a non-iterative greedy algorithm **A**, assigning correspondence of points in one frame to the points in the next frame.

Algorithm A

```
■ For $k$ = 2 to $n$ - 1 do
    ● A matrix M ($m * m$) is constructed, where
      $M[i, j] = \delta(X_p^{k-1}, X_q^k, X_r^{k+1})$, when $\Phi^{k-1}(p) = i$. Points
      from the $k$th frame are along the rows and points from
      ($k$+1)th frame along the columns.
    ● For $a$ = 1 to $m$ do
        i.   The minimum element $[i, l_i]$ in each row $i$ of $M$ is
             identified.
        ii.  A *priori matrix* $B$ is computed, such that for each $i$,
             $B[i, l_i] =$
             $$\sum_{j=1, j \neq l_i}^{m} M[i, j] + \sum_{k=1, k \neq i}^{m} M[k, l_i]$$
        iii. The $[i, l_i]$ pair with highest priority value $B[i, l_i]$ is
             selected and $\Phi^k(i) = l_i$ is assigned.
        iv.  The row $i$ and the column $l_i$ in $M$ is masked.
```

5.4 The Occlusion Problem

The algorithm **A** above does not provide for the occlusion of points. Occlusion is an inherent problem in the tracking of feature points. An object is considered to be occluded if it does not appear in the image due to other objects overlapping on it and feature points in that part are missed. Another reason for missing feature points is because they may not have been detected by the image processing algorithm even though they were present in the image. The different cases of occlusion can be due to the following occurrences:

- Points visible in frame k may not be visible in frame k+1
- Points occluded in frame k become visible in frame k+1
- Points visible in frame k get occluded in frame k+1 and some points occluded in frame k become visible in frame k+1.

An assumption is made that there is no occlusion in the first two frames that are being analyzed and all points are visible in these frames. The first time a case of occlusion occurs it is a case 1 occlusion. The modified algorithm detects this case of occlusion and fills up for the missing point in the (k+1)th frame and hence the algorithm never comes across the cases 2-3. Modifications to the algorithm **A** yields the algorithm **B**.

Algorithm B

```
■ For $k$ = 2 to $n$ - 1 do
        ●   A matrix M ($m * m_{k+1}$) is constructed, where
            $M[i, j] = \delta(X_p^{k-1}, X_q^k, X_r^{k+1})$, when $\Phi^{k-1}(p) = i$.
            $m_k$ points from the $k$th frame are along the rows and
            $m_{k+1}$ points from ($k$+1)th frame along the columns.
        ●   If ($m_{k+1} < m_k$) then it is a case of occlusion so do
            i. For $a$ = 1 to $m_{k+1}$ do
            ⋆ The minimum element $[l_j, j]$ in each column $j$ of $M$ is
              identified.
```

⋆ The priority matrix B is computed such that

$$B[l_j, j] = \sum_{i=1, i \neq l_j}^{m} M[i, j] .$$

⋆ The pair $[l_j, j]$ with the highest priority value $B[l_j, j]$ is selected and assignment $\Phi^k(l_j) = j$ is made.

⋆ Row l_j and column j are masked in M.

ii. The points m_k - m_{k+1} for which correspondence has not been found are identified. New feature points are created in frame k+1 for the missing points by extrapolating the correspondence from frame k - 1 to frame k.

iii. Set $m_{k+1} = m$

else there is no occlusion

i. Algorithm **A**

Points in the frame k that do not have corresponding points in the frame k + 1 are identified and new points are created corresponding to those missing points. If a point a in frame k with coordinates (x_a^k, y_a^k) which correspond to a point c in frame k - 1 with coordinates (x_c^{k-1}, y_c^{k-1}) does not have a corresponding point in the frame k + 1, then a point b with coordinates (x_b^{k+1}, y_b^{k+1}) is created to correspond to point a with the equations,

$$x_b^{k+1} = x_a^k + (x_a^k - x_c^{k-1})$$

$$y_b^{k+1} = y_a^k + (y_a^k - y_c^{k-1})$$

This extrapolation ensures smoothness in velocity, both in magnitude and direction.

CHAPTER VI
RESULTS AND CONCLUSIONS

6.1 Results

In this chapter, the results are presented for the experiments conducted on two spheres colliding in space at various angles of incidence. Good images were recorded by varying the gain and gate to control the light intensity and the electronic shutter speed. Selected images were processed and excellent edge images and marker locations were obtained. The algorithm proposed by Rangarajan and Shah[6], was implemented and found to work quite efficiently. The figures, tables and graphs presented in this chapter give a good illustration of automatic image analysis.

Figure 6.1 illustrates the effectiveness of the procedure adopted and program developed using Visilog. Figure 6.1b, is a typical example of obtaining an edge image of single pixel width, using the Sobel and Canny edge operators. It also shows the efficiency and robust nature of the Hough transform technique in finding multiple circle center locations. Figure 6.1c is the result of locating the markers and their positions.

Figures 6.2, 6.3, 6.4, 6.5 show a sequence of 7 frames that are processed to find sphere centers, marker locations and trajectories of the markers. Figure 6.6 is a graph to show the trajectory set for the spheres and markers. The numbers on the graph give the frame number at which the coordinates were found. It is observed that collision has occured in the 4th frame.

An example of correspondence is shown using another set of data where, Table 1 and Table 2 show the marker locations for the sphere on the left side before and after correspondence and its related graph is shown in figure 6.7. The various markers are matched correctly without any false matches taking place.

Figures 6.8, 6.9, 6.10, show another sequence of 7 frames. In figure 6.9 it is observed that in the 3rd frame a marker in the right sphere is not located. The tabulations for marker locations on the spheres at the right side of the frame, before correspondence is shown in Table 3, where the missing marker in the 3rd frame has been assigned (0, 0) as its coordinates. Tabulations for the corresponded marker locations are shown in the Table 4. We find that the matching algorithm correctly interpolates the coordinates for the missing point and assigns it to its proper position. This an excellent illustration of the correspondence algorithm where the problem of occlusion is taken care of. Figure 6.11 is a graphical representation of the trajectories of the spheres and markers for this sequence of images.

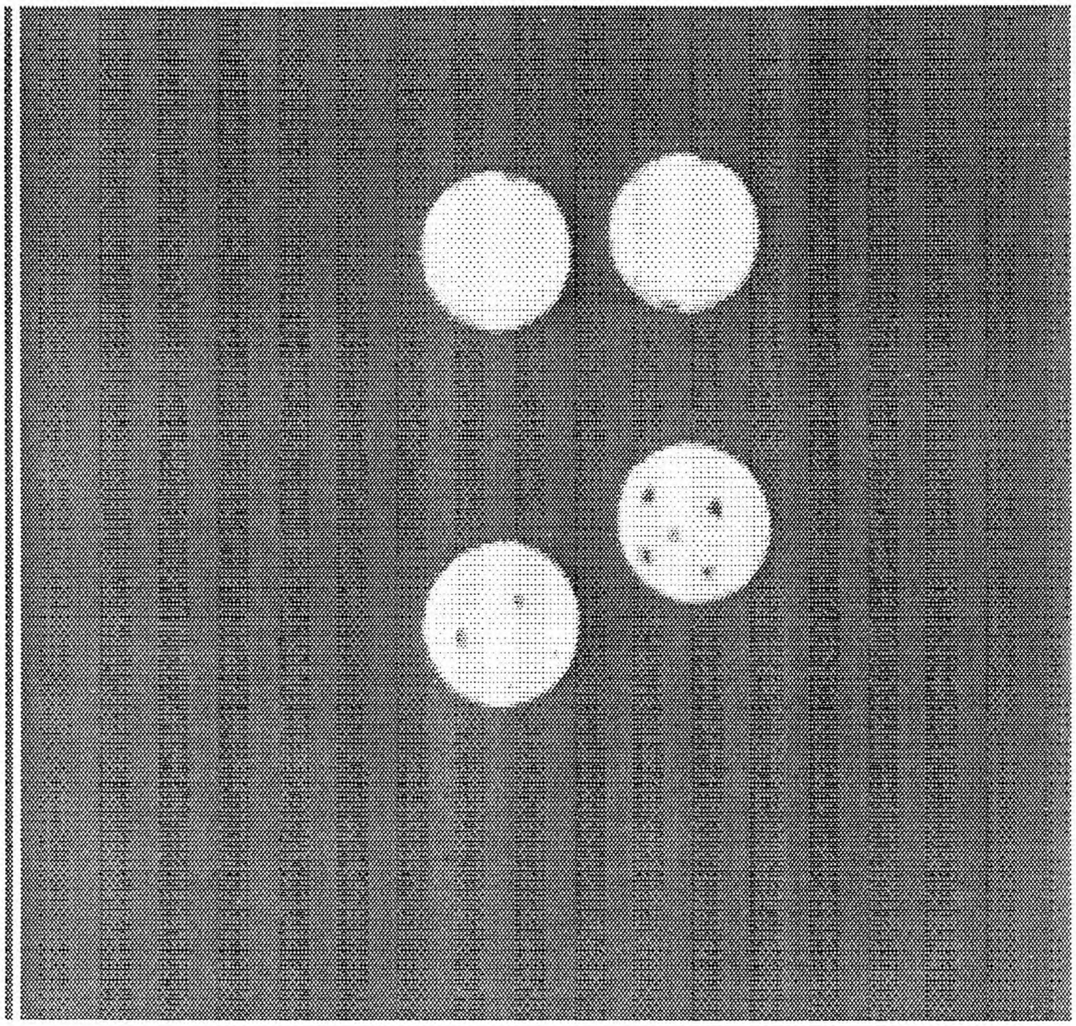


Figure 6.1a Shows the grey level image of the two spheres in the collision space.

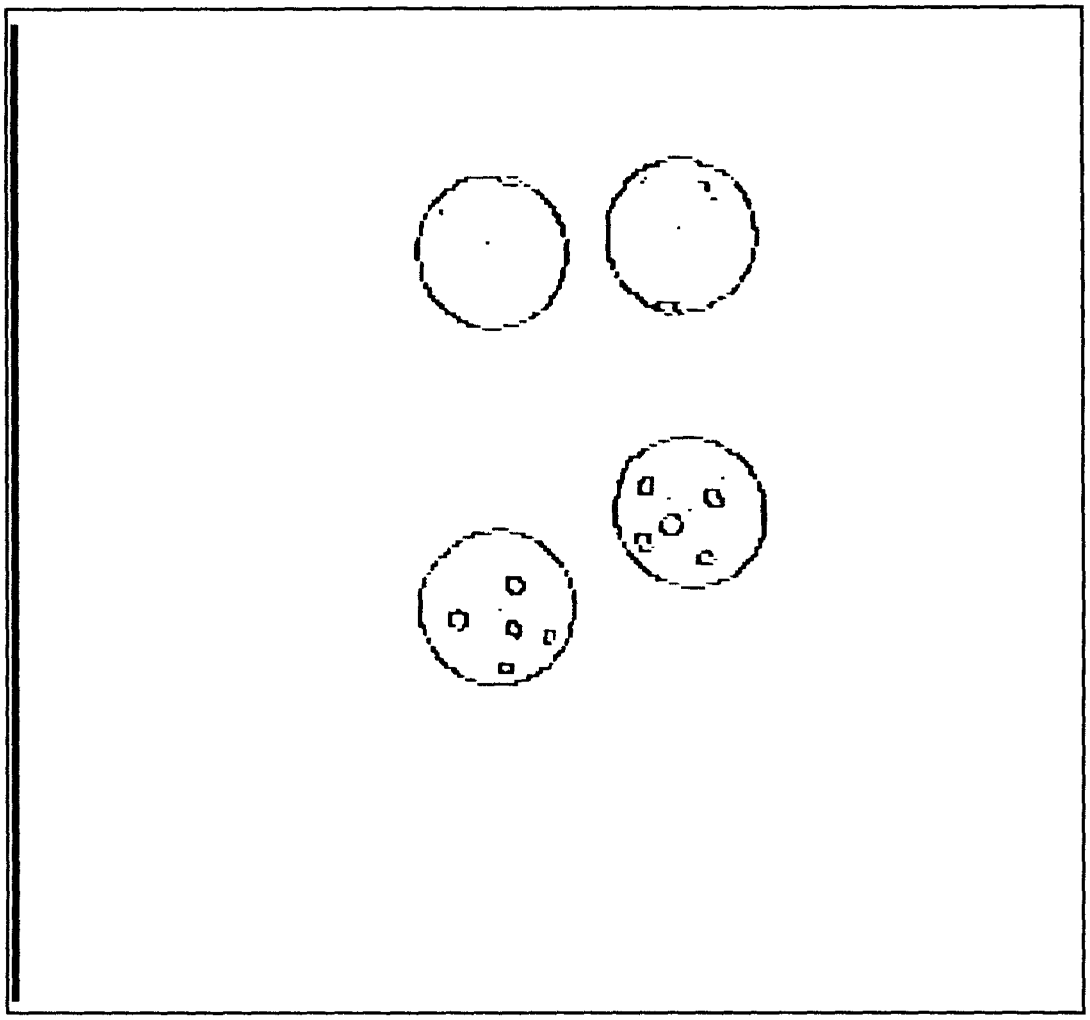


Figure 6.1b Shows the edge image obtained from the grey level image using the Sobel and Canny edge operators. The centers of the spheres are also marked using the Hough transform technique. The robust nature of this technique to multiple center detection in the presence of noise i.e., edge points other than that of the circles, is clearly seen.

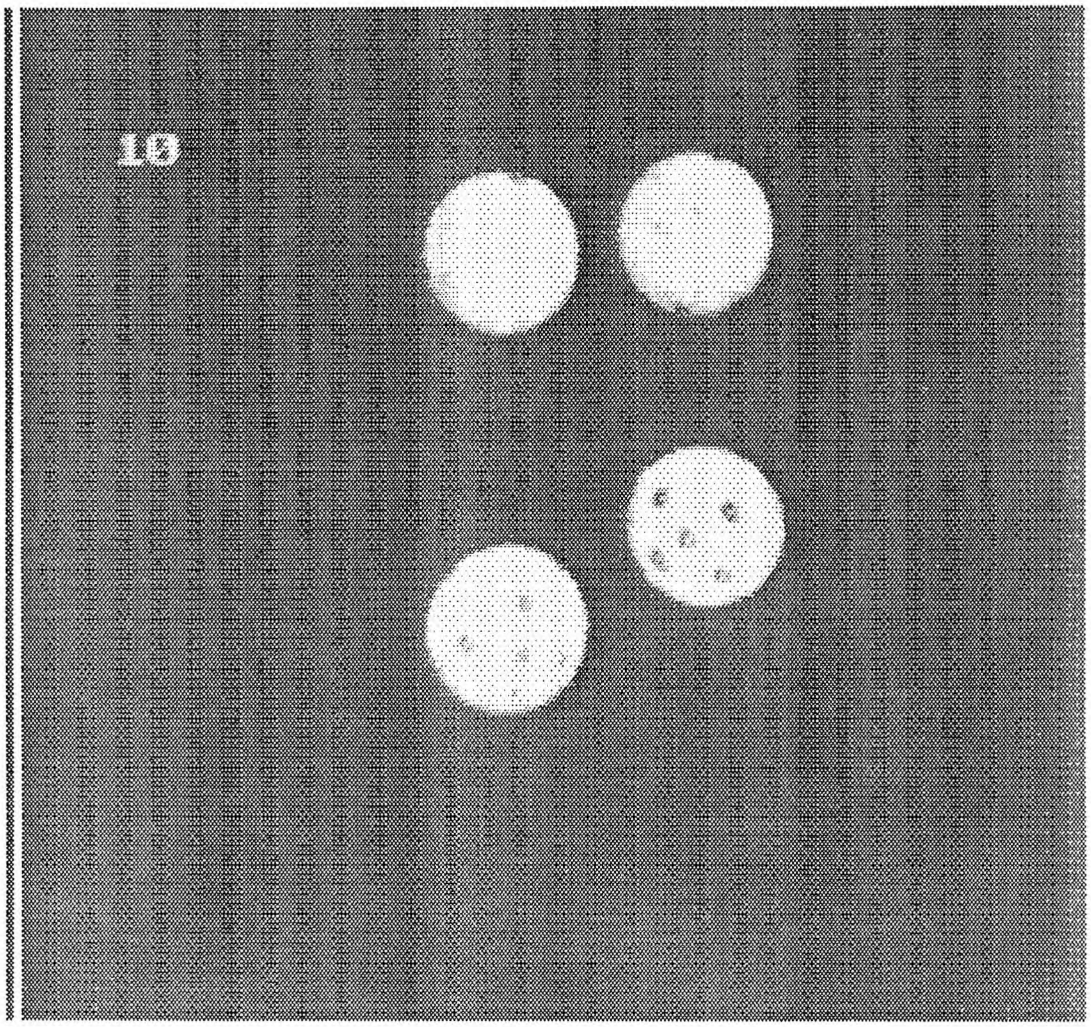


Figure 6.1c Shows that all the markers have been succesfully located and the number of markers found is displayed on the image. The algorithm used is quite accurate.

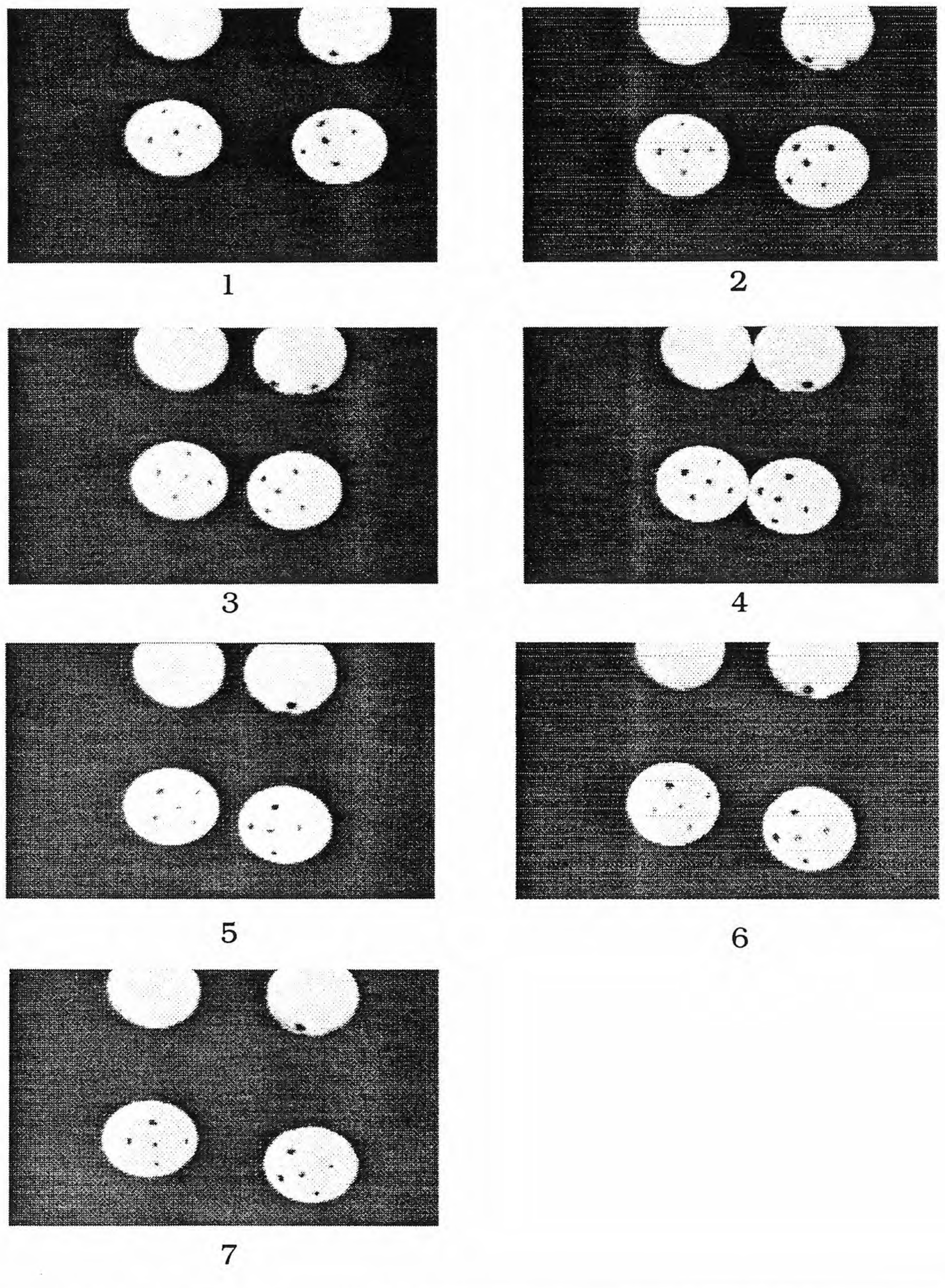
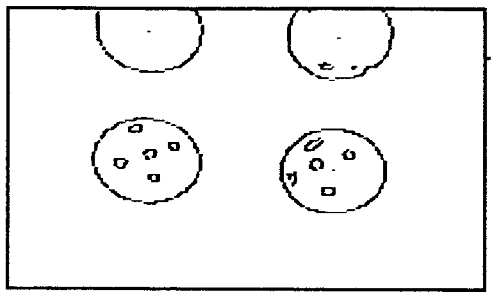
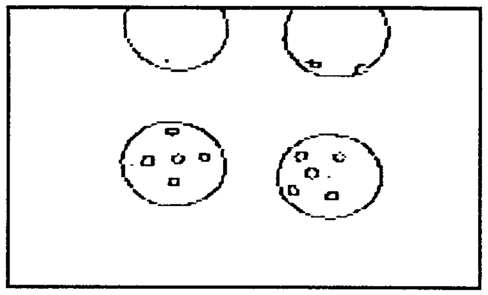
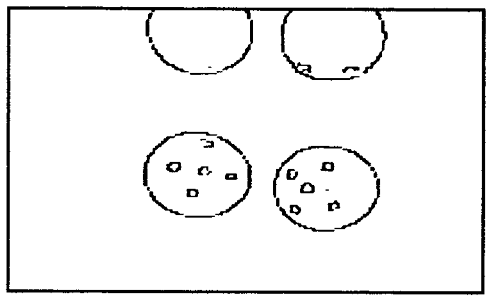
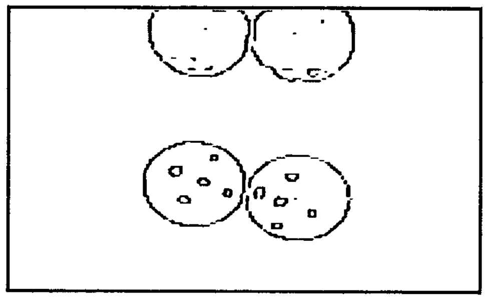


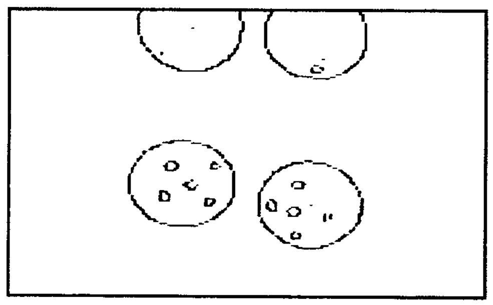
Figure 6.2 Shows a sequence of grey level images recorded at 1000frames/sec

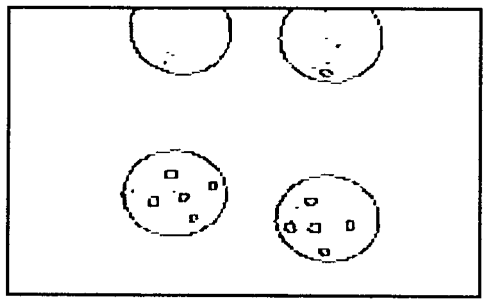

1


2


3


4


5


6

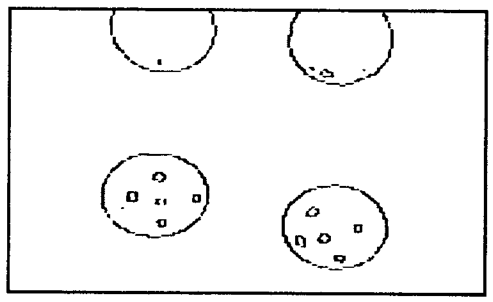

7

Figure 6.3 Shows the edge image for the sequence of frames

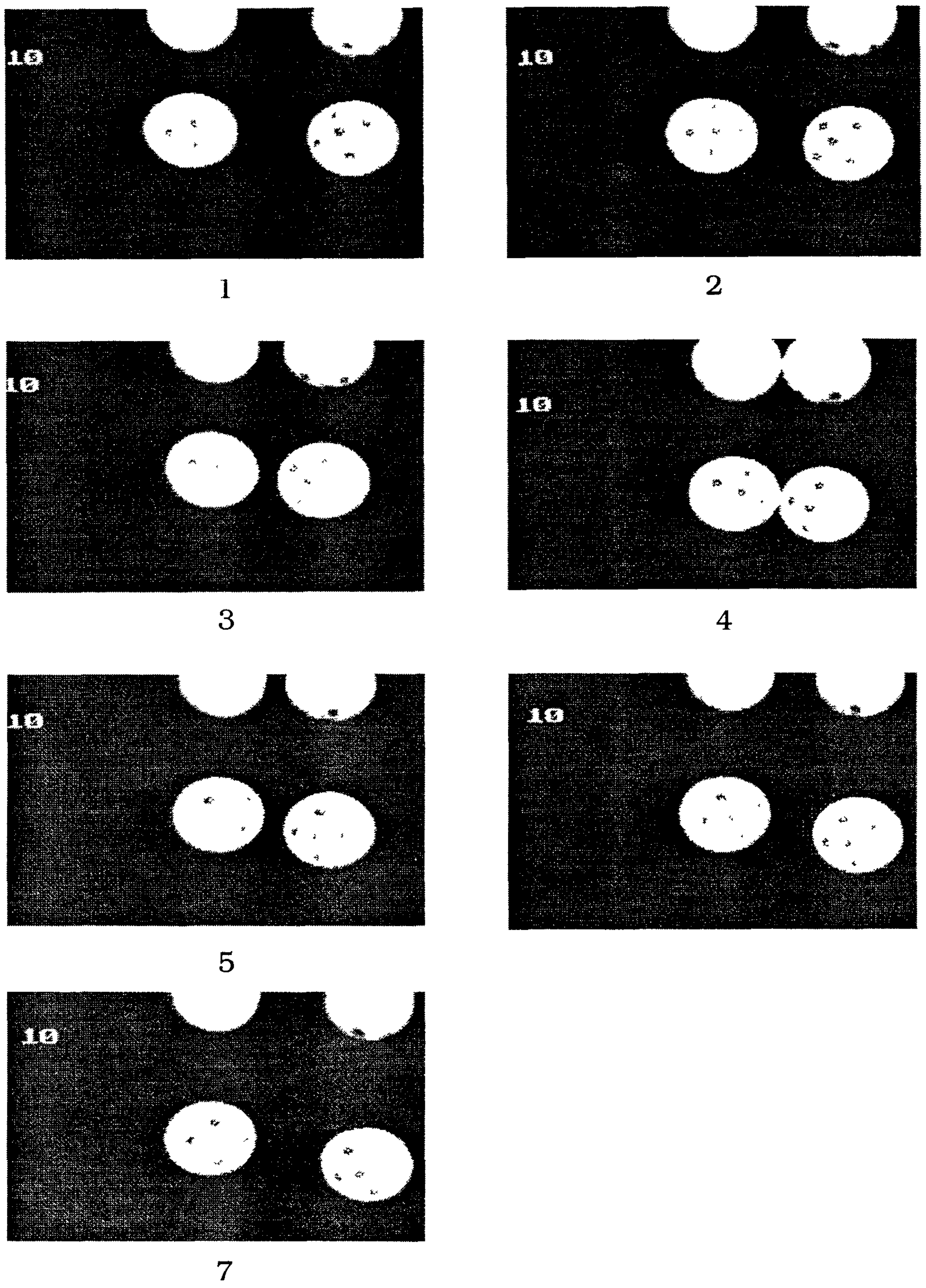


Figure 6.4 Shows that the markers have been successfully located and the number of markers found is displayed.

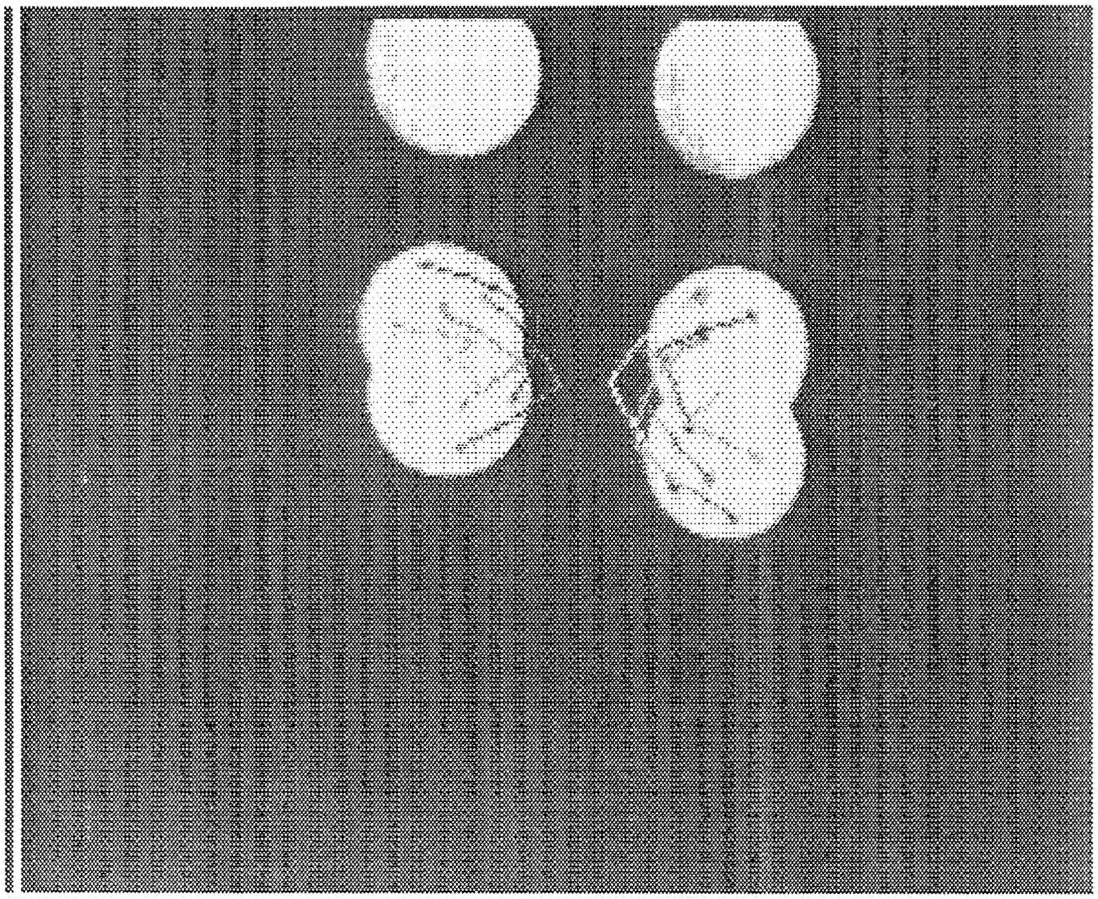


Figure 6.5 Shows the first and the last frames superimposed. The trajectories for each of the markers are displayed.

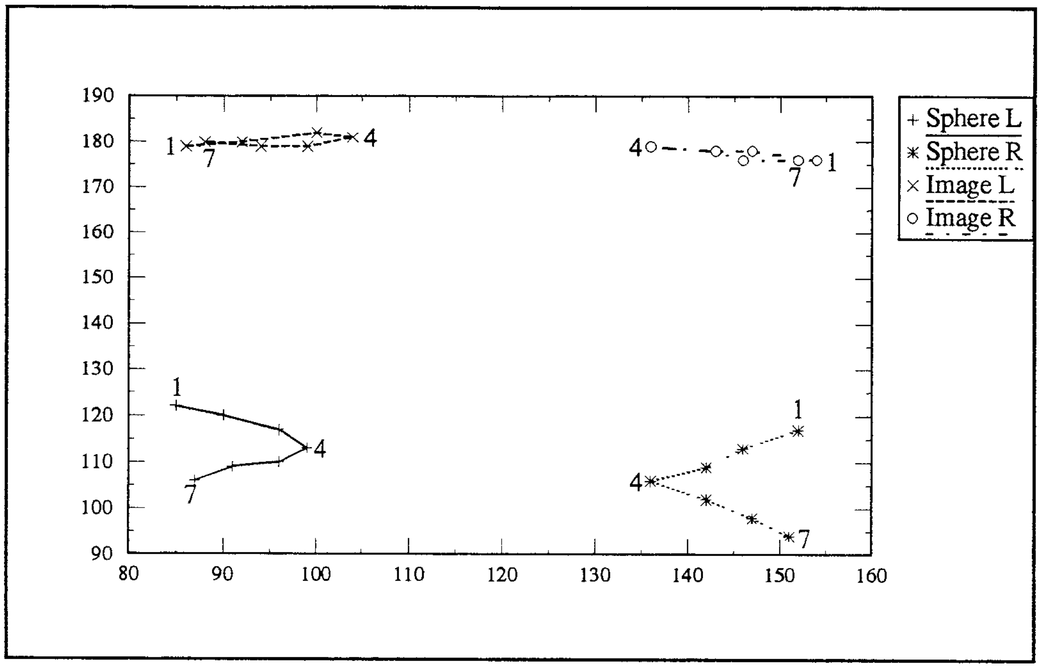


Figure 6.6a Shows the trajectories of the spheres and its mirror image. The spheres were released at an angle of 15°

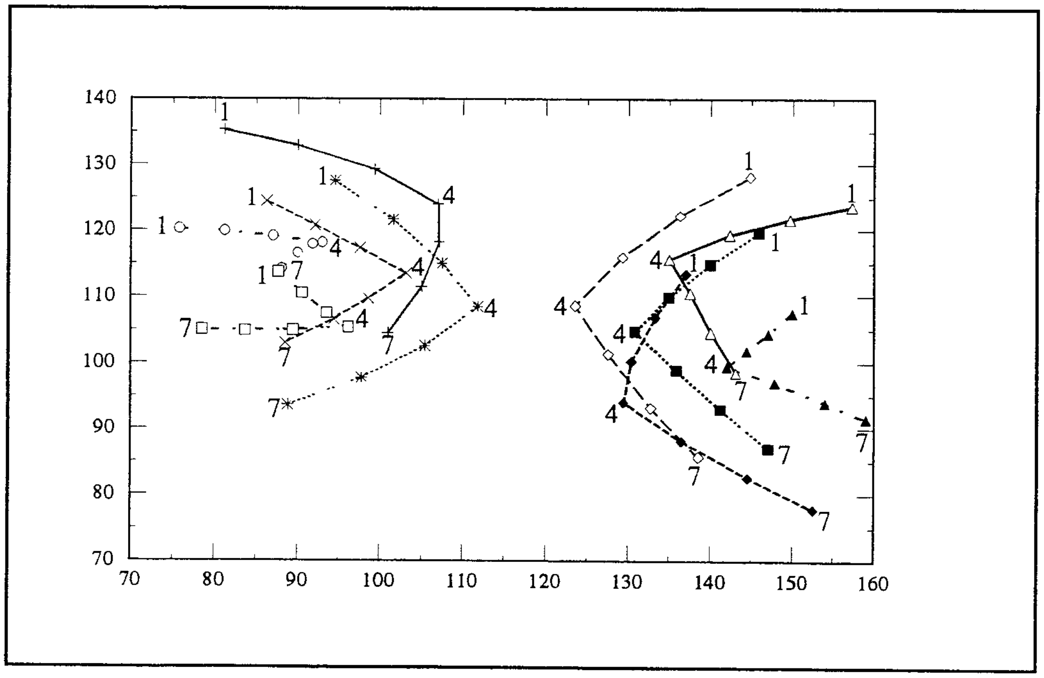


Figure 6.6b Shows the trajectories of the markers on the spheres.

Table 1: Coordinates of the markers on the left sphere before achieving correspondence

frame	marker 1	marker 2	marker 3	marker 4	marker 5
1	(95.14, 91.36)	(81.07, 93.71)	(87.22, 99.67)	(81.57, 109.79)	(96.00, 108.88)
2	(105.07, 97.29)	(92.00, 95.21)	(95.24, 102.47)	(86.07, 109.36)	(100.07, 114.36)
3	(102.62, 97.85)	(103.47, 106.13)	(115.00, 105.33)	(92.20, 109.47)	(104.13, 118.80)
4	(113.42, 102.67)	(99.86, 109.79)	(111.13, 111.00)	(122.58, 114.42)	(108.00, 123.00)
5	(112.64, 111.27)	(97.07, 114.71)	(108.13, 118.67)	(118.20, 124.50)	(101.47, 129.29)
6	(95.00, 120.50)	(110.67, 121.42)	(104.20, 127.07)	(94.76, 135.59)	(112.45, 135.55)
7	(93.07, 126.57)	(107.87, 132.07)	(100.00, 135.57)	(88.94, 141.35)	(105.77, 146.62)

Table 2: Coordinates of the markers on the left sphere after achieving correspondence

frame	marker 1	marker 2	marker 3	marker 4	marker 5
1	(95.14, 91.36)	(81.07, 93.71)	(87.22, 99.67)	(81.57, 109.79)	(96.00, 108.88)
2	(105.07, 97.29)	(92.00, 95.21)	(95.24, 102.47)	(86.07, 109.36)	(100.07, 114.36)
3	(115.00, 105.33)	(102.62, 97.85)	(103.47, 106.13)	(92.20, 109.47)	(104.13, 118.80)
4	(122.58, 114.42)	(113.42, 102.67)	(111.13, 111.00)	(99.86, 109.79)	(108.00, 123.00)
5	(118.20, 124.50)	(112.64, 111.27)	(108.13, 118.67)	(97.07, 114.71)	(101.47, 129.29)
6	(112.45, 135.55)	(110.67, 121.42)	(104.20, 127.07)	(95.00, 120.50)	(94.76, 135.59)
7	(105.77, 146.62)	(107.87, 132.07)	(100.00, 135.57)	(93.07, 126.57)	(88.94, 141.35)

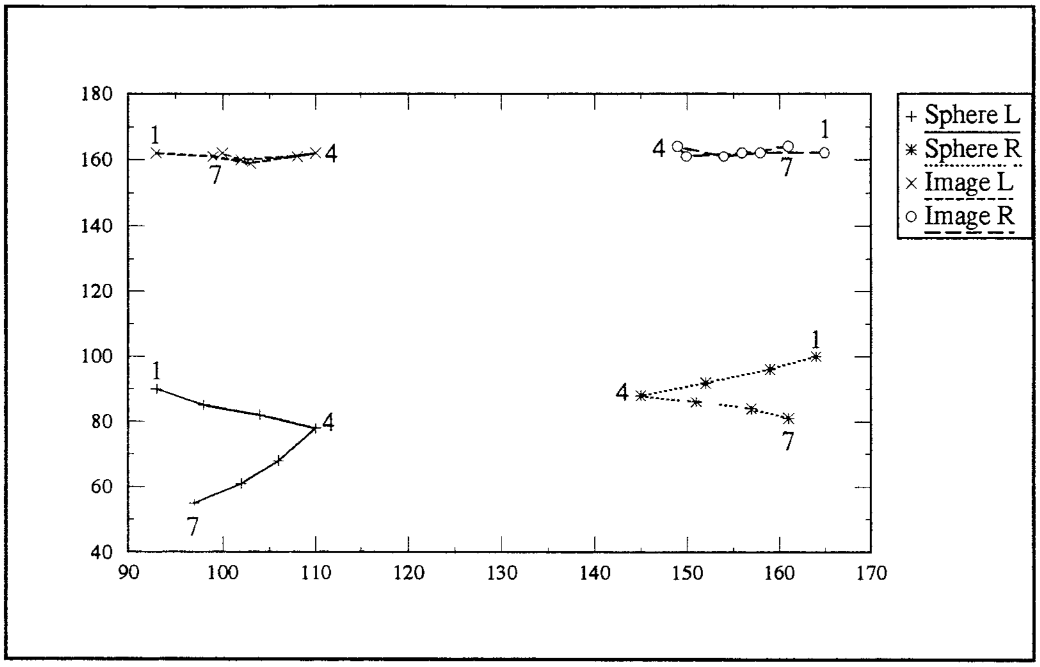


Figure 6.7a Shows the trajectories of the spheres and its mirror image. The spheres were released at an angle of 25^0

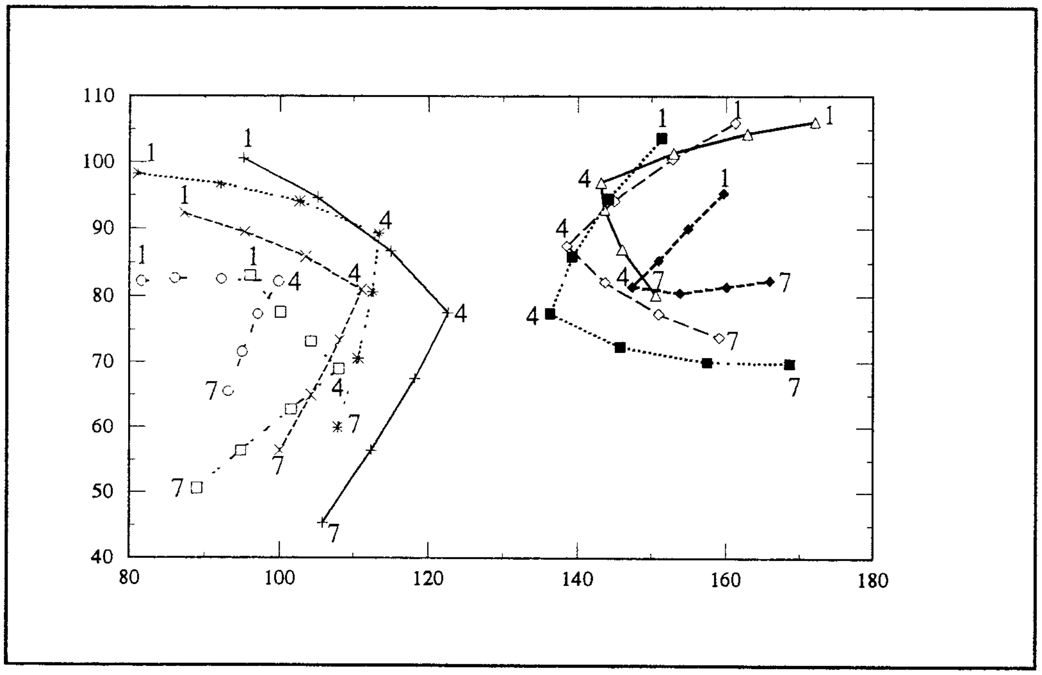


Figure 6.7b Shows the trajectories of the markers on the spheres.

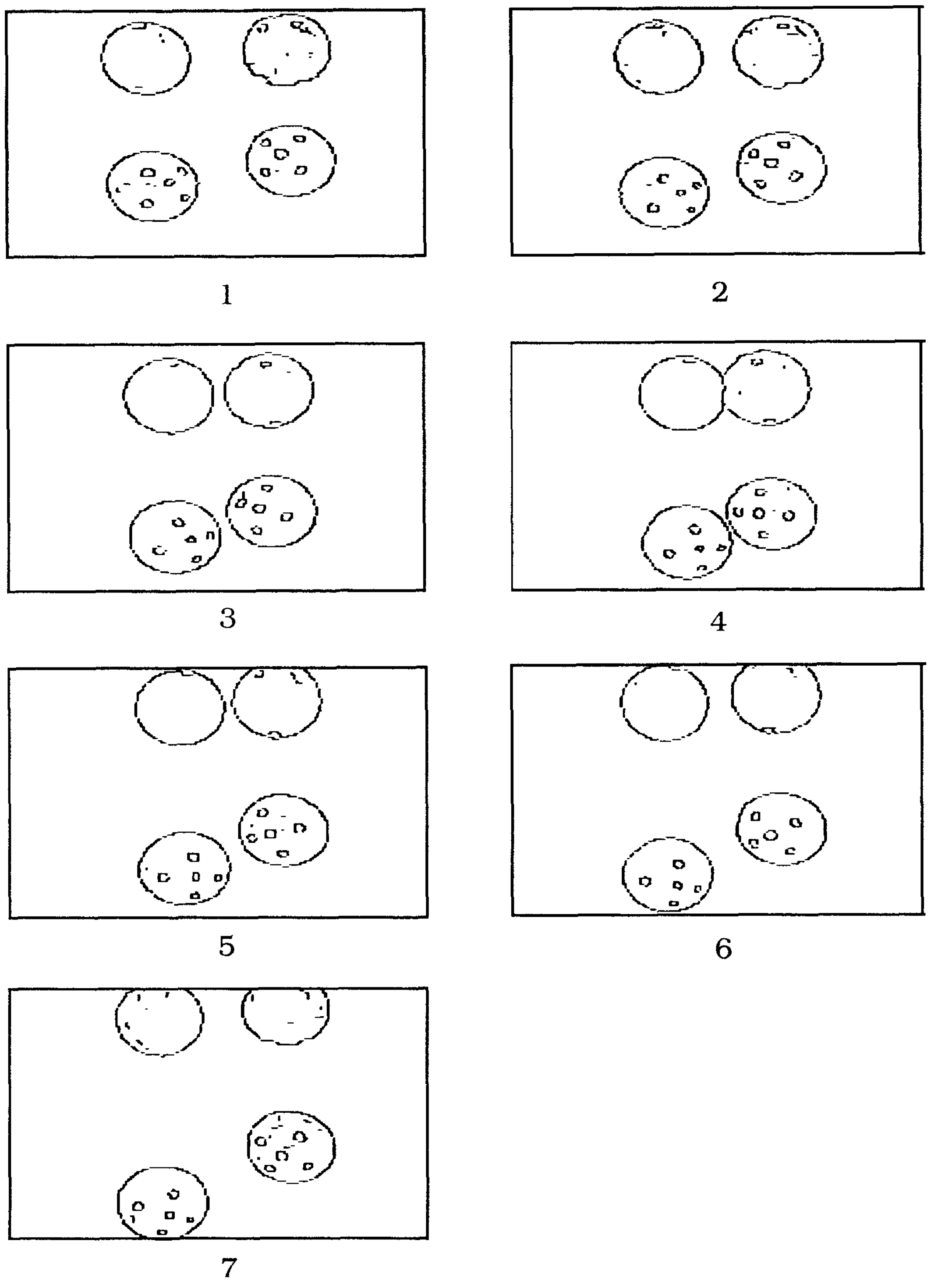


Figure 6.8 Shows a sequence of edge images when the spheres were released at an angle of 10°

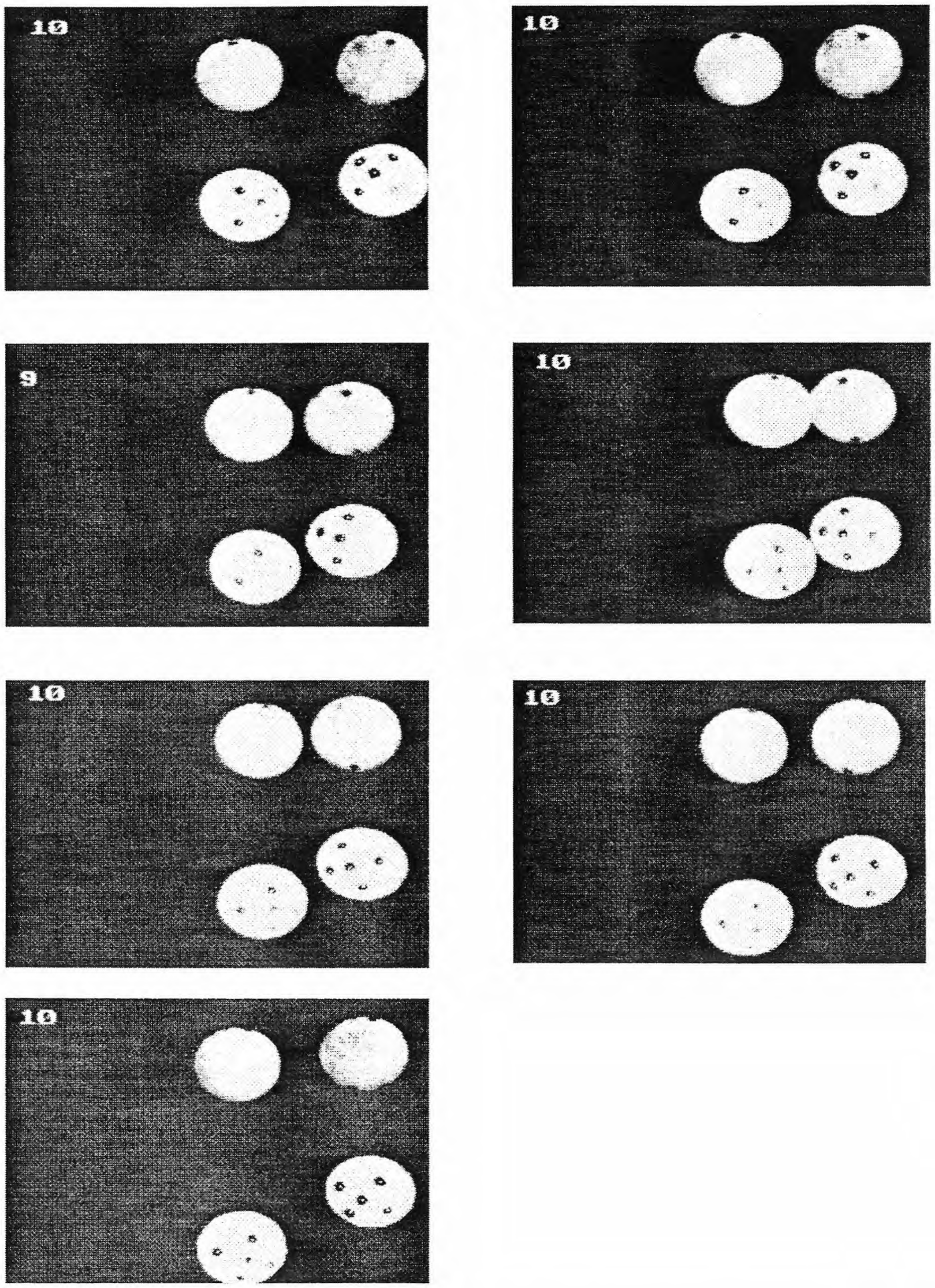


Figure 6.9 Shows the marker locations found in all the frames except in the 3rd frame, where only 9 have been found.

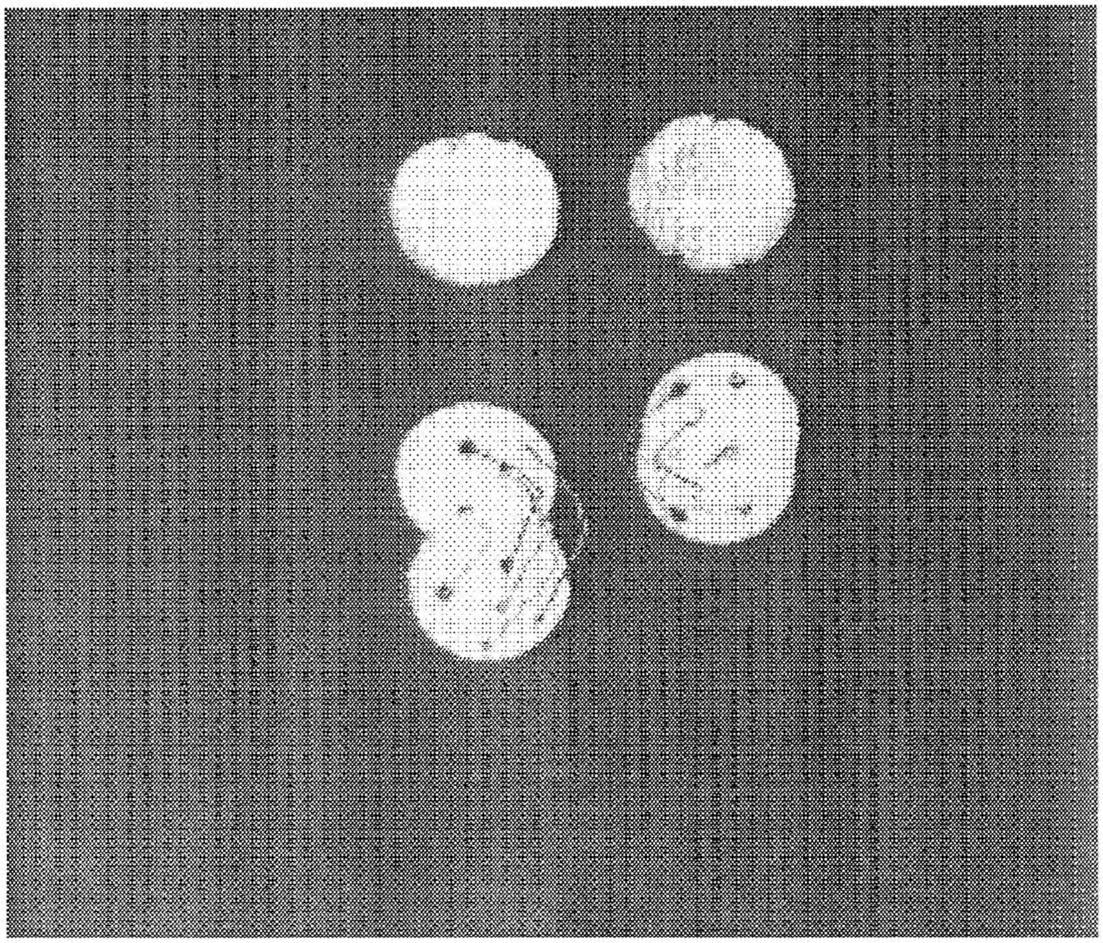


Figure 6.10 Shows the superimposed image of the first and last frames. The trajectories for the feature points are displayed.

Table 3: Coordinates of the markers on the right sphere with a point missing in the 3rd frame before correspondence

frame	marker 1	marker 2	marker 3	marker 4	marker 5
1	(156.07, 92.57)	(142.46, 90.69)	(148.26, 84.68)	(156.13, 75.93)	(141.57, 75.07)
2	(149.07, 90.57)	(136.62, 85.85)	(143.35, 81.18)	(153.71, 74.41)	(139.14, 70.14)
3	(142.08, 87.77)	(138.56, 77.06)	(150.07, 72.93)	(137.23, 65.69)	(0.00, * 0.00)
4	(136.58, 84.50)	(127.57, 74.14)	(136.00, 73.50)	(148.06, 72.35)	(137.45, 62.36)
5	(136.00, 81.80)	(150.88, 73.94)	(139.00, 71.00)	(131.00, 69.00)	(143.92, 60.69)
6	(136.57, 77.93)	(152.94, 74.94)	(142.59, 68.18)	(135.77, 64.15)	(150.73, 60.36)
7	(154.42, 75.47)	(138.07, 72.93)	(146.94, 65.87)	(157.42, 60.58)	(154.42, 75.47)

Table 4: Coordinates of the markers on the right sphere after achieving correspondence

frame	marker 1	marker 2	marker 3	marker 4	marker 5
1	(156.07, 92.57)	(142.46, 90.69)	(148.26, 84.68)	(156.13, 75.93)	(141.57, 75.07)
2	(149.07, 90.57)	(136.62, 85.85)	(143.35, 81.18)	(153.71, 74.41)	(139.14, 70.14)
3	(142.08, 87.77)	(130.78, * 81.01)	(138.56, 77.06)	(150.07, 72.93)	(137.23, 65.69)
4	(136.58, 84.50)	(127.57, 74.14)	(136.00, 73.50)	(148.06, 72.35)	(137.45, 62.36)
5	(136.00, 81.80)	(131.00, 69.00)	(139.00, 71.00)	(150.88, 73.94)	(143.92, 60.69)
6	(136.57, 77.93)	(135.77, 64.15)	(142.59, 68.18)	(152.94, 74.94)	(150.73, 60.36)
7	(138.07, 72.93)	(142.08, 59.54)	(146.94, 65.87)	(154.42, 75.47)	(157.42, 60.58)

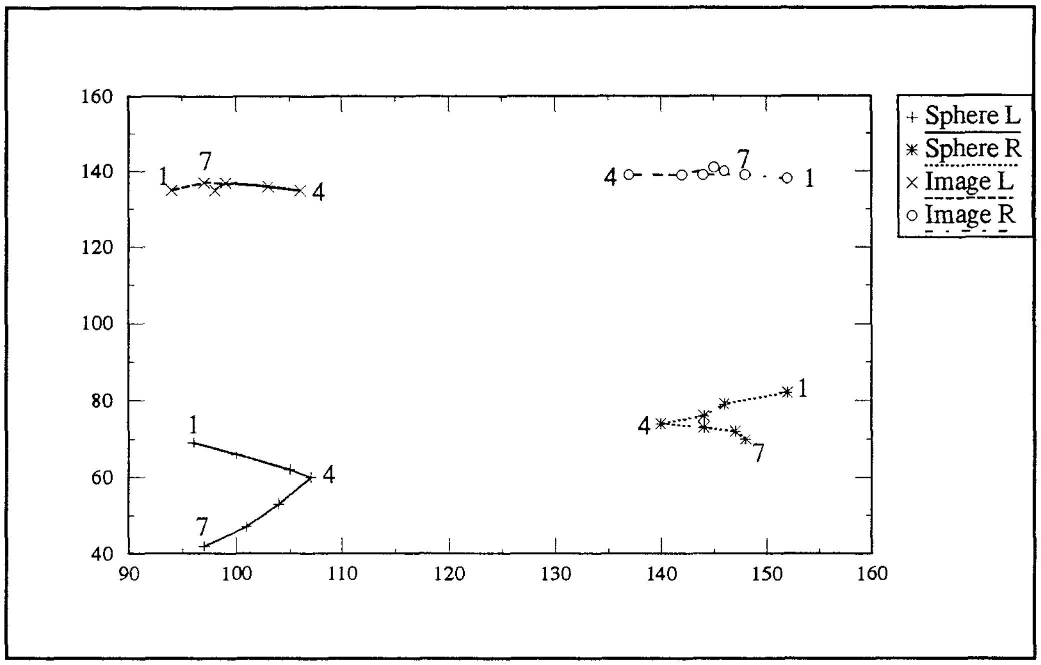


Figure 6.11a Shows the trajectories of the spheres and its mirror image. The spheres were released at an angle of 10°

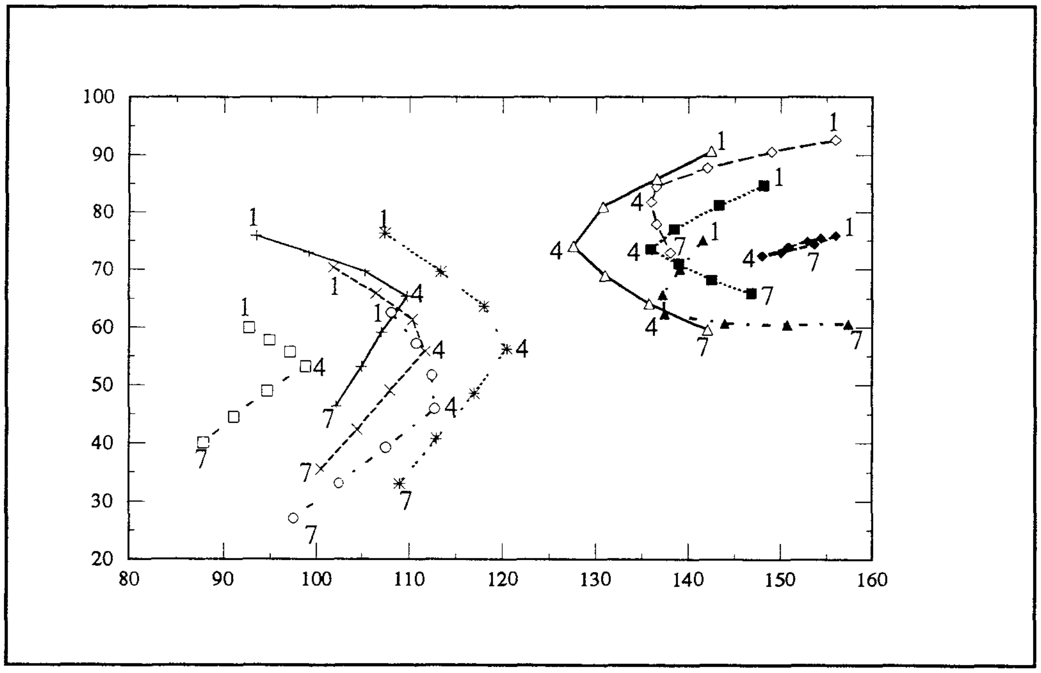


Figure 6.11b Shows the trajectories of the markers on the spheres.

6.2 Conclusions

The high speed imaging system i.e., Kodak Ektapro 1000 has been effectively utilized to obtain clear images of the two colliding spheres. The Sobel and Canny edge detection techniques have been successfully used to get good edge images as well as gradient images.

The Hough transform technique has proved once again to be robust and the detection of multiple circle centers poses no problem even in the presence of noise. This technique has been implemented in a PC 486 with the help of Visilog and the AFG card. The computational excesses of HT are significantly reduced by using the gradient information from the edge image and the parameters being reduced to two. This makes the accumulator array to be just 2 dimensional. Implementation of this technique in the PC was made possible due to the fact that Visilog considers images as 2-D objects, hence the accumulator array was converted into an image and stored in the image memory provided by the AFG card.

The method used in this thesis gives a fairly good result for the detection of multiple circle centers, but still the accuracy is limited to a pixel. Subpixel accuracy can be obtained by using the adaptive Hough transform technique and further research must be conducted in this area.

Once the sphere centers are found, the markers were easily located by searching the space within the spheres for connected components having a specified area range. Centroids of these connected components were then found, which give the coordinates of the markers. One of the disadvantages of this method is that if the markers are a little blurred due to poor lighting and focussing, its area may lie outside the specified range and fail to be counted as a marker.

Correspondence is achieved using the proximal uniformity constraint[6], proposed by Rangarajan and Shah, the assumptions being that initial correspondence is given and that all points are visible in the first two frames. This algorithm is found to work quite well for most of the sessions conducted. Tracking of the markers was possible when they were well spread over the sphere, but when the markers were too close to each other in a cluster the algorithm failed to make a correspondence. Occlusion of any marker is not a problem as the algorithm first makes a correspondence of all the markers that are found and then interpolates a value for the one that is not found.

If the algorithm is used by itself then the assumptions themselves are also found to be the limitations of this algorithm. This is true considering that human intervention is necessary to give an initial correspondence between the first two frames and the idea of automation is lost. A gradient based optical flow[30] method can be used for establishing the correct initial correspondence which was not incorporated in this thesis. Another limitation or assumption that was missed by the authors[6] was that if some other feature points are detected other than the ones required, then the algorithm fails completely i.e., no new points must be found in the entire sequence of images. This is a major limitation as the main focus is to track all the points observed in the entire sequence.

Overall it can be summarized that all the methods implemented performed reasonably well and allowed us to track the markers over a series of image frames with correct correspondence. Further work is necessary in motion tracking to incorporate the third dimension, i.e., to track 3-D coordinates of the feature points.

APPENDIX A

SPECIFICATIONS OF THE KODAK SYSTEM

PROCESSOR

Controls

Menu-driven Keypad:	LCD display provides user access to all system functions. Includes six dedicated functions keys and ten-multi-function keys.
Power Switch:	Easily accessible.
Eject Switch:	Ejects tape cassette.

Operating Features:

Recording Technique:	Linear FM.
Recording Medium:	1/2" high density tape.
Tape Handling:	Cassette (700ft.)
Frame Rates:	Records at 30, 60, 125, 250, 500, 1000 full frames/sec. Up to 6000 pictures/sec.
Frame Formats:	1, 2, 3, 4 or 6 pictures/frame
Recording Time:	A minimum of 16 minutes at 30 fps and a minimum of 30 seconds at 1000 fps.
Normal Playback:	30 frames per second.
Single Step:	Displays one frame at a time, forward or reverse.
Jog:	Displays successive frames, forward or reverse, at a slow, continuous rate.
Fast Forward/Rewind:	Moves tape at 300 ips forward or reverse. This

	rate is faster than the highest recording speed.
BOT/EOT:	Optically senses the beginning of tape and the end of tape to prevent overruns.
Search:	Moves the tape to a given video frame.
Heads	
Record & Playback:	Two Microgap heads, each providing 19 channels-- 16 video, 2 timing, and 1 unsupported.
Erase:	Permanent magnet.
Video Output	
Compatible with:	NTSC or PAL
Gamma Correction:	Variable from 0.1 to 1.0
Grey Scale:	256 levels.
Size:	17"x22"x12 1/4".
Weight:	Approximately 80 lbs.
Power:	110/220 VAC, 60/50 Hz, 8 amps/4 amps.

IMAGER

Control Keys:	Live, Record & Stop
I/O Jacks:	Video, Audio, & Remote Trigger.
Sensor:	192x240 pixel NMOS array.
Lens Mount:	C-Mount, with electronic remote control capability for zoom, focus and exposure.
Tripod Mount:	1/4-20 and 3/8-16 with standard ANSI hole pattern.
Cables:	15 ft. standard.
Size:	Approximately 9"x4"x5"
Weight:	Approximately 5 lbs.

Power:	Derived from processor.

KEYPAD

Dedicated-function Keys

Live:	Displays live image on viewfinder and monitor.
Record:	Starts recording.
Stop:	Stops recording or playback and freezes the last image in frame store.
Replay:	Moves tape to first frame of most recent recording session and plays back at 30 fps.
Play:	Plays a recording in any selected playback mode.
Help:	Provides short cut paths through menu tree.

System Software Menu

System Setup:	Controls Imager selection, overlay format, position and size, frame rate and division factor, automatic lens functions and session numbers.
Move Tape:	Controls playback mode and event markers.
Video Display:	Enables reticle, gamma adjustment, interlaced video and saved image.
Environment:	Controls time and date.

APPENDIX B

SOME DEVELOPMENT TOOLS IN VISILOG

Image Access Routines

dupnf_( )	Retrieves a free image handle		
C Definition	*IMAGE*	*dupnf_(nf);*	
	IMAGE	**nf;*	Pointer to reference image.
image_( )	Retrieves *IMAGE* handle of a given image object		
C Definition	*IMAGE*	*image_(name,mode,verify,nfc);*	
	char	**name;*	Pointer to the image name.
	char	**mode;*	Pointer to the access type.
	char	**verify;*	Pointer to verification flags.
	IMAGE	*nfc;	Pointer to the control image.
readpx_( )	Read a pixel		
C Definition	*void readpx_(nf,x,y,buffer);*		
	IMAGE	**nf;*	Pointer to the image.
	long	**x, *y;*	Pointer to the position.
	char	**val;*	
writpx_( )	Writes a pixel		
C Definition	void writpx_(nf,x,y,buffer);		Pointer to the image.
	IMAGE	**nf;*	Pointer to the image.
	long	**x, *y;*	Pointer to the position.
	char	**val;*	

Display And Acquisition Routines

visclr_( )	Clears the display memory		
C Definition	*int visclr_( );*		
xlutst_( )	Sets the lookup tables to pre-defined tables		
C Definition	*int xlutst_(type);*		
	long	**type;*	Pointer to table type.
xvisu_( )	Positions the display window on an image		
C Definition	*IMAGE*	*nf;	Pointer to the image.
xzoom_( )	Sets the hardware zoom factor		
C Definition	*int xzoom_(n);*		
	long	**n;*	Pointer to the zoom factor.
zgrtxt_( )	Draws text onto an image		
C Definition	*int zgrtxt_(nf,test,coor,lcol,size,bcol,orient);*		
	IMAGE	**nf;*	Pointer to the image.
	char	**text;*	String to write.
	long	*corr[2];*	X & Y left upper coordinates
	long	**lcol;*	Pointer to the letter level.
	long	*size[2];*	Pointer to the X,Y letter size.
	long	**bcol;*	Pointer to the background level.
	long	**orient;*	Pointer to the orientation flag.

Point To Point Operations

sarit_()	Arithmetic operations		
C Definition	*void sarit_(nfi1,nfi2,nfo,topp,tfg);*		
	IMAGE	**nfi1;*	Pointer to input image 1.
	IMAGE	**nfi2;*	Pointer to input image 2.
	IMAGE	**nfo;*	Pointer to the output image.
	long	**top;*	Pointer to the operation code.
	long	**tfg;*	Pointer to the operation type.
sthr_()	Thresholding of images		
C Definition	*void sthr_(nfi,nfo,level,value,option);*		
	IMAGE	**nfi;*	Pointer to the input image.
	IMAGE	**nfo;*	Pointer to the output image.
	long	*level[2];*	Low and high threshold.
	long	*value[3];*	Low, mid & high output values
	long	**option;*	Pointer to the operation code.

Analysis Operations

sarea_()	Surface of binary image		
C Definition	*void sarea_(nfi,area);*		
	IMAGE	**nfi;*	Pointer to the input image.
	long	**area;*	Pointer to the surface parameter.
sinert_()	Inertia moments		
C Definition	*void sinert_(nfi,moment);*		
	IMAGE	**nfi;*	Pointer to the input image.

	float	*moment[5];*	Array of inertia moments.
snumbe_( )	Number of connected components		
C Definition	*void snumbe_(nfi,nfw,number);*		
	IMAGE	**nfi;*	Pointer to the input image.
	IMAGE	**nfw;*	Pointer to the work image.
	long	**number;*	Pointer to the number of cells.

Morphology Operations

slabel_( )	Labelling of connected components		
C Definition	*void slabel_(nfi,nfo);*		
	IMAGE	**nfi;*	Pointer to the input image.
	IMAGE	**nfo;*	Pointer to the output image.

Edge Detection Operations

scmpas_( )	Compass gradient		
	void scmpas_(nfi,nfg,nfo,ty,scale,red);		
	IMAGE	**nfi;*	Pointer to the input image.
	IMAGE	**nfg;*	Pointer to gradient amplitude.
	IMAGE	**nfo;*	Pointer to orientation image.
	long	**typ;*	Pointer to the kernel indicator.
	long	**scale;*	Pointer to the scaling factor.
	long	**red;*	Pointer to the reduction factor.
sedge3_( )	3 x 3 Edge detectors		
C Definition	*void sedge3_(nfi,nfx,nfy,ty,scale,red);*		

IMAGE	**nfi;*	Pointer to the input image.
IMAGE	**nfx;*	Pointer to X gradient image.
IMAGE	**nfy;*	Pointer to Y gradient image.
long	**typ;*	Pointer to the kernel indicator.
long	**scale;*	Pointer to the scaling factor.
long	**red;*	Pointer to the reduction factor.

smxsup_( ) Detection of the local crest lines in gradient

C Definition *void smxsup_(nfx,nfy,nfm,nfe);*

IMAGE	**nfx;*	Pointer to X gradient image.
IMAGE	**nfy;*	Pointer to Y gradient image.
IMAGE	**nfm;*	Pointer to gradient amplitude.
IMAGE	**nfe;*	Pointer to the output edge.

APPENDIX C

PROGRAM LISTING USING VISILOG

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "../h/terminal.h"                  /* Tty manager          */
#include "../h/visilog.h"                   /* Visilog structures   */
#include "../h/inter1.h"                    /* External variables   */

    /***************************************************************/
    /*                  THE MAIN PROGRAM                          */
    /***************************************************************/

FILE *out_file1, *out_file2;
long  coor[2], size[2], *level;
int radix = 10;
char buf[3];
char *str;
float point[20][2];
int center[4][2];
int sphere, sph1, sph2;
float dist[20];
int files = 0;
int no_markers;
int colsn = 0;


main(argc,argv)
int argc; char **argv;
{
            long ttyflag = 0; argc = 0;
            stdaln_(&ttyflag,argc,argv);
                                    /* Initializes VISILOG context        */
            initialize();
```

```
          starting(argv[1]);
          chdir(Old_Dir);
          exit(0);

}

initialize()
{
          M_MMODE = 0;
          vlgini_();          /* Mode 256x256 is set        */
          if (V_ID != NO_DISPLAY)
            visclr_();
          if (M_ID == CPU_MEMORY)
            memclr_();
          printf ("\nCleared display memory.\n");

}

/*    Check if the file exists                        */

check(char* name1)
{
          FILE *in_file;
          char name[15];
          int  i;

          for (i = 1; i <= 20; i++)     /* Assume max of 20 frames to process */
          {
            sprintf (name, "%s.%d", name1, i);
            in_file = fopen (name, "r");
            if (in_file == NULL)
            {
              if (i == 1)
              {
                printf ("\n\n\n\nImage files do not exist.\n");
              }
              else
              {
                fclose (in_file);
                break;
              }
            }
            else
            {
              fclose (in_file);
              files = files + 1;
```

```
                }
            }
            printf ("\n\n\nNumber of image files to process are %d\n", files);
}

starting(char* name1)
{
            FILE *in_file;
            char name[15];
            int  i;

            check(name1);
            sprintf (name, "%s1.dat", name1);
            out_file1 = fopen(name, "w");
            sprintf (name, "%s2.dat", name1);
            out_file2 = fopen(name, "w");

            for (i = 1; i <= files; i++)      /* Assume max of 20 frames to process */
            {
              sprintf (name, "%s.%d", name1, i);
                process(name,name1,i);
            }
}

/*     Image processing functions start                        */

process(char* name,char* name1,int frame)
{
            flush_marker();
            edge(name,frame);               /* Edge detection process  */
            marker(name,frame);             /* Labelling is done         */
            check_points(name);
            storing(name1,frame);
}

/*     Function to detect the edge of the image                */

edge(char* name,int frame)
{
            IMAGE nffile ,          /* File image from CPU          */
                nforig ,            /* Input memory image           */
                nfg,                /* Gradient amplitude image     */
                nfo,                /* Gradient orientation image*/
                nthr,               /* Threshold image              */
                nflab,              /* Labelled image               */
                nfx  ,             /* X Gragient image              */
```

```
     nfy,              /* Y Gradient image          */
     nfe;              /* Edge image                */
IMAGE nfc;             /* Control structure         */
long typ  = 0;         /* Kernel indicator          */
long red  = 0;         /* Reduction indicator       */
long scale = 1;        /* Scaling factor            */
long level[1];         /* Array of low & high thres */
long value[3];         /* Array of low, mid & high  */

/* Create a control image structure */

nfc = dupnf_(NULL);

/* Get an existing image from file */

strcpy(nfc->file, name);
nffile = image_(nfc->file,"e","",&nfc);

/* Copy the image into memory & display */

strcpy(nfc->file,"init");
nforig = image_(nfc->file,"c","",&nfc);
xlutst_(&zero);                 /* Linear B&W lut's        */
xzoom_(&two);                   /* Zoom factor of 2        */
xvisu_(&nforig);                /* Show the input image */
scopy_(&nffile,&nforig);        /* Does the copying        */
fermnf_(&nfc);
printf ("\nProcessing image %s.\n", name);

/* Create a gradient image */

nfc = dupnf_(&nforig);
nfg = image_("nfg1","s","",&nfc);
nfo = image_("nfo1","s","",&nfc);
nfx = image_("nfx1","s","",&nfc);
nfy = image_("nfy1","s","",&nfc);
sedge3_(&nforig,&nfx,&nfy,&typ,&scale,&red);
scmpas_(&nforig,&nfg,&nfo,&three,&one,&zero);
xvisu_(&nfg);
printf ("\nAmplitude image.\n");

/* Create an edge image */

nfe = image_("nfo1","s","",&nfc);
smxsup_(&nfx,&nfy,&nfg,&nfe);
xvisu_(&nfe);
```

```
          printf ("\nEdge Detection is Done.\n");
          /*sleep(3);*/

          /* To obtain a thresholded image */

          set_code(nfc,I_BIN,16,2);
          nthr = image_("^thr","s","",&nfc);
          level[0] = 18;
          level[1] = 184;
          sthr_(&nfe,&nthr,level,value,&zero);
          printf ("\nThresholded image.\n\n");

          /* Hough Transform     */

          hough(&nforig,frame);

          /* Close all image handles */

          fermnf_(&nforig); fermnf_(&nfx);
          fermnf_(&nfy); fermnf_(&nfe); fermnf_(&nfc);
          fermnf_(&nfg); fermnf_(&nfo); fermnf_(&nthr);
}

/*    Flush all previous marker & center coordinates        */

flush_marker()
{
          int i;

          for (i = 0; i < 4; i++)
          {
           center[i][0] = 0;
           center[i][1] = 0;
          }
          for (i = 0; i < 20; i++)
          {
           point[i][0] = 0;
           point[i][1] = 0;
          }
}


/*    Function to label the image and to find the centers
          of the connected components    */

marker(char* name1,int frame)
```

```
{
          IMAGE nffile ,           /* Threshold image from CPU  */
               nforig ,            /* Input memory image        */
               nthr,               /* Threshold image           */
               nfo,
               temp,
               nflab;              /* Labelled image            */
          IMAGE nfc;               /* Control structure         */
          char *name = "thr";      /* Image file from disk      */
          int  i, j, k, l, m, val, sum, mark, dad;
          int  cx1, cx2, cx3, cx4, cy1, cy2, cy3, cy4;
          float x, y;
          float moment[5];
          long x1, y1, x2, y2;
          long number;
          long level[1];        /* Array of low & high thres */
          long value[3];        /* Array of low, mid & high  */
          long area;            /* Area of the image         */
          coor[0] = 10; coor[1] = 30;
          size[0] = 1; size[1] = 1;
          (*level) = 255;

          /* Get the thresholded image from the disk */
          visclr_();
          nfc = dupnf_(NULL);
          strcpy(nfc->file,name);
          nffile = image_(nfc->file,"e","",&nfc);

          /* Copy the image into memory & display */
          strcpy(nfc->file,"ajax");
          nforig = image_(nfc->file,"c","",&nfc);

          xlutst_(&two);                /* Linear B&W lut's        */
          xzoom_(&two);                 /* Zoom factor of 2        */
          xvisu_(&nforig);              /* Show the input image */
          scopy_(&nffile,&nforig);      /* Does the copying        */
          printf ("\nThresholded image.\n");
          /*sleep(3);*/

          /* Labelling the connected components */

          set_code(nfc,I_LABEL,16,2);
          strcpy(nfc->file,"nfg1");
          nflab = image_(nfc->file,"s","",&nfc);

          /* Eliminate extra data */
```

```
elimin(&nforig,&nflab);

xvisu_(&nflab);
slabel_(&nflab,&nflab);
printf ("\nLabelling done.\n");
scopy_(&nflab,&nforig);

/* To count the number of components */
snumbe_(&nflab,NULL,&number);
printf ("\nNumber of connected components = %ld\n", number);

/* Finding the center of each component */
set_code(nfc,I_BIN,16,2);
temp = image_("ajax","s","",&nfc);

mark = 150;
printf ("Finding markers for image %s.\n", name1);
k = 0;
for (i = 1; i <= number; i++)
{
 level[0] = i;
 level[1] = i;
 sthr_(&nflab,&temp,level,value,&zero);
 area = 0;
 sarea_(&temp,&area);
 if (area > /*8*/7)
 {
  for (j = 0; j < 5; j++)
   moment[j] = 0.0;
  sinert_(&temp,moment);

  x = moment[0];
  y = moment[1];
  x2 = x + 0.5;
  y2 = y + 0.5;

  if (area <= 19/*20*/)
  {
   sum = 0;
   for (l = x-3/*4*/; l <= x+3/*4*/; l++)
          for (m = y-3/*4*/; m <= y+3/*4*/; m++)
          {
           x1 = l;
           y1 = m;
           readpx_(&temp,&x1,&y1,&val);
           sum = sum + val;
```

```
                        }
                  if ((sum==area) && (distanc((int)x,(int)y)))
                  {
                        point[k][0] = moment[0];
                        point[k][1] = moment[1];
                        scopy_(&temp,&nforig);
                        xvisu_(&nforig);
                        k++;
                        printf ("\nMarker : %d   (%6.2f,%6.2f)\n", k, x, y);
                        /*printf ("Area of the marker =  %ld\n", area);
                        printf ("X = %7.2f,  Y = %7.2f\n", x, y);
                        printf ("x = %7.2f,  y = %7.2f  xy = %7.2f\n",
                                    moment[2], moment[3], moment[4]);*/
                        str = itoa(k,buf,radix);
                        writpx_(&nforig,&x2,&y2,&mark);
                        zgrtxt_(&nforig,str,coor,level,size,&zero,&zero);
                        sleep(2);
                        zgrtxt_(&nflab,str,coor,level,size,&zero,&zero);
                        writpx_(&nflab,&x2,&y2,&mark);
                        xvisu_(&nflab);
                  }
                 }
                }
               }
               printf ("\nThe number of markers visible are :  %2d\n", k);
               if (frame == 1)
               {
                 fprintf (out_file1, "%d %d \n\r", files, sphere);
                 fprintf (out_file2, "%d %d \n\r", files, k);
               }
               xvisu_(&nflab);
               zgrtxt_(&nflab,str,coor,level,size,&zero,&zero);

               /* Close all image handles */
               fermnf_(&nffile); fermnf_(&nforig); fermnf_(&nflab);
               fermnf_(&nfc);fermnf_(&nthr); fermnf_(&nfo); fermnf_(&temp);
}

/*     Eliminate extra data from the image before labelling    */
elimin(input,output)
IMAGE *input, *output;
{
               int i, j, k, val;
               int cx, cy;
               long x, y;
```

```
          for (i = 0; i < sphere; i++)
          {
            cx = center[i][0];
            cy = center[i][1];
            for (j = cx-21; j <= cx+21; j++)
              for (k = cy-21; k <= cy+21; k++)
              {
                x = j;
                y = k;
                readpx_(input,&x,&y,&val);
                writpx_(output,&x,&y,&val);
              }
          }
}

/*     To check if all points found are correct                    */

check_points(char* name)
{
          IMAGE nffile ,          /* File image from CPU        */
                nforig ;          /* Input memory image         */
          IMAGE nfc;
          int i, val;
          long x, y;
          char beep;

          /* Create a control image structure */

          nfc = dupnf_(NULL);

          /* Get an existing image from file */

          strcpy(nfc->file, name);
          nffile = image_(nfc->file,"e","",&nfc);

          /* Copy the image into memory & display */

          strcpy(nfc->file,"init");
          nforig = image_(nfc->file,"c","",&nfc);
          xlutst_(&zero);              /* Linear B&W lut's       */
          xzoom_(&two);                /* Zoom factor of 2       */
          xvisu_(&nforig);             /* Show the input image */
          scopy_(&nffile,&nforig);     /* Does the copying       */

          val = 255;
          beep = 7;
```

```
         for (i = 0; i < 20; i++)
         {
          x = point[i][0] + 0.5;
          y = point[i][1] + 0.5;
          if ((x == 0) && (y == 0))
            break;
          writpx_(&nforig,&x,&y,&val);
         }
         printf ("\nProcessing of image %s completed.\n", name);
         printf ("Hit any key to continue .... %c\n", beep);
         sleep(1);
         printf ("%c", beep);
         getch();

         fermnf_(&nffile); fermnf_(&nforig); fermnf_(&nfc);
}


/* Detects the centre of circles using the Hough Transform technique */

hough(input,frame)
IMAGE *input;
{
         IMAGE nffile,        /* Threshold image from CPU   */
               nforig,        /* Input memory image         */
               nfacm,         /* Accumulator image          */
               h_t;           /*    "      "    in CPU      */
         IMAGE nfc;
         int i, j, k, l, n, val, val1, val2, val3, val4, val5, val6, val7, val8;
         int maxx, imax, jmax, sumx, sumy, val9, val10, val11, val12;
         int p, q, r, c1, c2;
         int maxy1, maxy2;
         long x, y, z, x1, y1, x2, y2;
         double value1, value2, dir, dir1;
         char beep;

         /* Get the thresholded image from the disk */
         /*visclr_();*/
         nfc = dupnf_(NULL);
         strcpy(nfc->file,"thr");
         nffile = image_(nfc->file,"e","",&nfc);

         /* Copy the image into memory & display */
         strcpy(nfc->file,"ajax");
         nforig = image_(nfc->file,"c","",&nfc);
```

```
xlutst_(&two);              /* Linear B&W lut's       */
xzoom_(&two);               /* Zoom factor of 2       */
xvisu_(&nforig);            /* Show the input image */
scopy_(&nffile,&nforig);    /* Does the copying      */

strcpy(nfc->file,"nfg1");
nfacm = image_(nfc->file,"s","",&nfc);
sarit_(&nfacm,&zero,&nfacm,&two,&zero);

printf ("Hough Transform in progress.\n");
printf ("BE PATIENT   Computations being done\n");
n = 0;
z = 0;
for (i = 1; i <= 192; i++)
  for (j = 1; j <= 239; j++)
  {
    x = j;
    y = i;
    readpx_(&nforig,&x,&y,&val);
    if (val == 1)
    {
      x1 = x-1;
      y1 = y-1;
      x2 = x+1;
      y2 = y+1;
      readpx_(input,&x1,&y1,&val1);
      readpx_(input,&x1,&y,&val2);
      readpx_(input,&x1,&y2,&val3);
      readpx_(input,&x2,&y1,&val4);
      readpx_(input,&x2,&y,&val5);
      readpx_(input,&x2,&y2,&val6);
      readpx_(input,&x,&y1,&val7);
      readpx_(input,&x,&y2,&val8);
      sumx = val3 + 2 * val8 + val6 - (val1 + 2 * val7 + val4);
      sumy = val1 + 2 * val2 + val3 - (val4 + 2 * val5 + val6);
      n++;
      for (k = 1; k <= 239; k++)
      {
             if (sumx != 0)
             {
               value1 = sumx;
               value2 = sumy;
               dir = atan(value2/value1);
               dir1 = (180.0/3.142) * dir;
               if (dir1 < 0.0)
               {
                    dir1 = dir1 + 180.0;
                    dir = dir1 * 3.142/180.0;
                  }
                  if (dir != 0)
                  {
                    l = - k/tan(dir) + (i + j/tan(dir));
                    if ((l > 0) && (l <= 192))
                    {
                      z++;
                      printf ("%ld\r", z);
                      x1 = k;
                      y1 = l;
                      readpx_(&nfacm,&x1,&y1,&val9);
                      val10 = val9 + 1;
                      writpx_(&nfacm,&x1,&y1,&val10);
                    }
                  }
                }
    }
  }
}
xvisu_(&nfacm);

h_t = image_("^trough","s","",&nfc);
scopy_(&nfacm,&h_t);
beep = 7;

printf ("\n");
printf ("Number of edge points = %d\n", n);
printf ("Finding centers \n");

sph1 = 0;
maxy1 = 0;
for (r = 0; r < 4; r++)
{
  maxx = 0;
  for (i = 1; i <= 192; i++)
    for (j = 1; j <= 239; j++)
    {
                x = j;
                y = i;
                readpx_(&nfacm,&x,&y,&val5);
                if ((maxx < val5) && (val5 > 16))
                {
                    maxx = val5;
                    imax = j;
                jmax = i;
            }
    }
  if (maxx != 0)
  {
    printf ("center(%3d,%3d) maxx = %d \n", imax, jmax, maxx);
    x = center[r][0] = imax;
    y = center[r][1] = jmax;
    if (jmax > 90) /*(maxy1 <= jmax)*/
    {
            /*maxy1 = jmax;*/
            sph2 = sph1;
            sph1 = r;
    }
    val1 = 250;
    writpx_(&nforig,&x,&y,&val1);
    for (p = imax - 16; p <= imax + 16; p++)
            for (q = jmax - 16; q <= jmax + 16; q++)
            {
              x1 = p;
              y1 = q;
              writpx_(&nfacm,&x1,&y1,&zero);
            }
    if (r == 3)
    {
            sphere = 4;
            printf ("\nNumber of spheres found is %d\n", sphere);
    }
  }
  else
  {
    sphere = r;
    printf ("\nNumber of spheres found is %d\n", sphere);
    break;
  }
}

if (sphere >= 2)
{
printf ("lowest 2 spheres are: (%d %d)  (%d %d)\n", center[sph1][0],
 center[sph1][1], center[sph2][0], center[sph2][1]);

value1 = abs(center[sph1][0] - center[sph2][0]);
value2 = abs(center[sph1][1] - center[sph2][1]);
dist[frame] = sqrt(pow(value1,2) + pow(value2,2));
printf ("dist[%d] = %6.2f\n", frame, dist[frame]);
```

```
              if ((dist[frame] > dist[frame-1]) && (colsn != 1) && (frame != 1))
              {
                printf ("Collision occured at the %2d frame.\n", frame-1);
                colsn = 1;
              }
              }
              xvisu_(&nforig);
              printf ("Hit any key to continue ....%c\n", beep);
              sleep(1);
              printf ("%c", beep);
              getch();

              fermnf_(&nfacm); fermnf_(&nfc); fermnf_(&nforig); fermnf_(&nffile);
              fermnf_(&h_t);
}

/*    Check if the markers are within the lowest 2 spheres only       */

distanc(x,y)
{
             int  e, f, value1, value2;
             double a, b, c, d;

             e = x;
             f = y;
             a = abs(center[sph1][0] - e);
             b = abs(center[sph1][1] - f);
             c = abs(center[sph2][0] - e);
             d = abs(center[sph2][1] - f);
             value1 = sqrt(pow(a,2) + pow(b,2));
             value2 = sqrt(pow(c,2) + pow(d,2));

             /*if (((a < 21) && (b < 21)) || ((c < 21) && (d < 21)))*/
             if ((value1 < 21) || (value2 < 21))
               return 1;
             else
               return 0;
}

/*    To place the centers and the markers in two seperate files     */

storing(char* name1,int frame)
{
             int  i, j;
             int maxy = 0;
             char name[15];
```

```
        if (frame > 1)
        {
          sprintf (name, "%s1.dat", name1);
          out_file1 = fopen(name, "a");
          sprintf (name, "%s2.dat", name1);
          out_file2 = fopen(name, "a");
        }
        for (i = 0; i < sphere; i++)
        {
          fprintf (out_file1, "%d %d  ", center[i][0], center[i][1]);
          /*if (maxy < center[i][1])
            maxy = center[i][1];*/
        }
        fprintf (out_file1, "\n\r");
        for (i = 0; i < 20; i++)
        {
          if ((point[i][0] == 0) && (point[i][1] == 0))
            break;
          /*if (point[i][1] < maxy-40)
            continue;*/
          fprintf (out_file2, "%6.2f %6.2f  ", point[i][0], point[i][1]);
        }
        fprintf (out_file2, "\n\r");
        fclose (out_file1);
        fclose (out_file2);
}
```

References

1. Tsai, R. Y., and T. S. Huang. "Uniqueness and Estimation of Three Dimensional Motion Parameters of Rigid Objects with Curved Surfaces." *IEEE Trans. on Pattern Analysis and Machine Intelligence.* PAMI-6(1) (1984):13-27.
2. Duda, R.O., and P.E. Hart. "Use of the Hough Transformation to detect lines and curves in pictures." *Communications of ACM.* 15 (1972):11-15.
3. Jenkin. "Tracking three dimensional moving light displays." in *Proceedings, Workshop Motion: Representation Contr. Toronto.* (1983): 66-70.
4. Rosato, A., R. Dave, I. Fischer, W. Carr. "Methodology of a Non-Intrusive Particle Tracing Technique for Inclined Chute Flows." *Proceedings of the Third DOE/NSF Workshop. Flow of Particulates and Fluids.* (1991):Oct. 23-25. Worcester, MA.
5. Illingworth, J., and J. Kittler. "The adaptive Hough transform." *IEEE T-PAMI.* 5 (1987):690-697.
6. Rangarajan, K., and M. Shah. "Establishing Motion Correspondence." *CVIGP: Image Understanding* 54(1) (1991):56-73.
7. Roberts, L. G. "Machine Perception of Three Dimensional Solids." *Optical and Electro-Optical Information Processing.* ed. Tippet, J. T., et al., Cambridge, Mass., MIT Press. (1965):150-197.
8. Nevatia, R. "*Machine Perception.*" Princeton Hall. (1982).
9. Rosenfeld, A., and A. C. Kak. "*Digital Picture Processing.*" Academic Press, 2 (1982).
10. Canny, J. "A Computational Approach to Edge Detection." *IEEE Trans. on Pattern Analysis and Machine Intelligence,* 8 (1986): 679-698.
11. Canny, J. F. "Finding Edges and Lines in Images." *MIT Artificial Intelligence Laboratory Technical Report TR-720,* (1983).
12. Marr, D. *Vision,* W. H. Freeman and Co., San Francisco, (1982).
13. Marr, D. and E. C. Hidreth. "A Theory of Edge Detection." *Phil. Trans. Roy. Soc. London B207,* (1980):187-217.

14. Hough, P.V.C. "Method and Means for Recognizing Complex Patterns." U.S. Patent 3069654, (1962).

15. Kimme, C., D. H. Ballard, and J. Sklansky. "Finding circles by an array of accumulators." *CACM,* 18 (1975):120-122.

16. Wechsler, H., and J. Sklansky. "Automatic Detection of ribs in chest radiographs." *Pattern Recognition,* 9 (1977):21-30.

17. Tsuji, S., and F. Matsumoto. "Detection of ellipses by modified Hough transformation." *IEEE T-COMP,* 27 (1978):777-781.

18. Illingworth, J., and J. Kittler. "A survey of Hough transform." *Pattern Recognition,* (1988):88-116.

19. Ullman, S. *Interpretation of Visual Motion,* MIT Press, Cambridge, MA, (1979).

20. Barnard, S. T., and W. B. Thompson. "Disparity Analysis of Images." *Technical Report 79-1,* Computer Science Dept., University of Minnesota, (1979).

21. Sethi, I. K., and R. Jain. "Establishing Correspondence of Non-rigid Objects Using Smoothness of Motion," *Technical Report CRL-TR-10-84,* University of Michigan Center for Research on Integrated Manufacturing, (1984).

22. Visilog, *Programmers Guide,* Noesis, (1988).

23. Volcy, J., "Kodak Ektapro 1000 Serial Interface for IBM PCs," *Technical report,* NJIT, (1991).

24. Ballard, D.H. "Generalizing the Hough transform to detect arbitrary shapes." *Pattern Recognition,* 13 (1988):111-122.

25. Kodak Ektapro 1000, Reference Manual.

26. Ballard, D.H., and Brown, C. M. *Computer Vision,* Prentice Hall, Englewood Cliffs, New Jersey, (1982).

27. Horn, B. K. P. *Robot Vision,* The MIT Press, Cambridge, Mass. (1986).

28. Bezdek, J. C., Coray, C., Gunderson, R., and Watson, J. Detection and characterization of cluster substructure, *SIAM J. Appl. Math.* 40 (1981):339-372.

29. Dave, R. N. "Fuzzy Shell Clustering and Applications to circle Detec tion in Digital Images." *International J. of General Syatems*, (1990).

30. Little, Henirich, and Poggio. "Parallel optical flow using local voting." *Proceedings of Second ICCV*, (1988):454-459.