2010

SPHEROID DETECTION IN 2D IMAGES USING CIRCULAR HOUGH TRANSFORM

Priyanka Chaudhary
University of Kentucky, pchau2@uky.edu

ABSTRACT OF THESIS

SPHEROID DETECTION IN 2D IMAGES
USING CIRCULAR HOUGH TRANSFORM

Three-dimensional endothelial cell sprouting assay (3D-ECSA) exhibits differentiation of endothelial cells into sprouting structures inside a 3D matrix of collagen I. It is a screening tool to study endothelial cell behavior and identification of angiogenesis inhibitors. The shape and size of an EC spheroid (aggregation of ~ 750 cells) is important with respect to its growth performance in presence of angiogenic stimulators. Apparently, tubules formed on malformed spheroids lack homogeneity in terms of density and length. This requires segregation of well formed spheroids from malformed ones to obtain better performance metrics. We aim to develop and validate an automated imaging software analysis tool, as a part of a High-content High throughput screening (HC-HTS) assay platform, to exploit 3D-ECSA as a differential HTS assay. We present a solution using Circular Hough Transform to detect a nearly perfect spheroid as per its circular shape in a 2D image. This successfully enables us to differentiate and separate good spheroids from the malformed ones using automated test bench.

KEYWORDS: Circular Hough Transform, Pattern Recognition, 3D-ECSA, HC-HTS, Image Processing.

Priyanka Chaudhary

May, 19th 2010

SPHEROID DETECTION IN 2D IMAGES USING CIRCULAR HOUGH TRANSFORM

By,

Priyanka Chaudhary

Dr. Laurence Hassebrook
Director of Thesis

Dr. Stephen G Gedney
Director of Graduate Studies

May, 19th 2010

RULES OF THE THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature of each user.

Name Date

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

THESIS

Priyanka Chaudhary

The Graduate School

University of Kentucky

2010

SPHEROID DETECTION IN 2D IMAGES USING CIRCULAR HOUGH TRANSFORM

THESIS

A thesis submitted in partial fulfillment of the
Requirements for the degree of Master of Science in
Electrical Engineering in the College of Engineering
at the University of Kentucky

By

Priyanka Chaudhary
Lexington, Kentucky

Director: Dr. Laurence G. Hassebrook, Professor of Electrical Engineering
Lexington, Kentucky

2010

Dedicated to ….

God, my loving family & friends

ACKNOWLEDGEMENTS

I would like to express heartfelt gratitude to Dr. Laurence Hassebrook, my advisor for believing in my work and guiding me at every turn of the stone. I will always cherish and honor the privilege to have worked under his guidance and the opportunity to develop multiple skills. My sincere thanks and love to my parents Dr. Ravindra Chaudhary and Mrs. Anuradha Chaudhary , and my elder brother, Piyush Chaudhary for their blessings and for being so kind and supportive of my decisions. They have always given me the spirit, strength and the dedication to fulfill my dreams.

My sincere thanks to Akshay Pethe for being the best friend that he has always been and helping me out at each step, in successfully finishing this ordeal. Special thanks to Yongchang Wang and Charles Casey who took out time from their busy endeavors and helped me at some of the most crucial moments.

Last but never the least, my Satguru Shri Saibaba, who has relentlessly cared for me, listened to me and salvaged my life and my beliefs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FILES

Priyanka_Chaudhary_EE_Thesis.pdf……2.7MB

Chapter 1 Introduction

High Throughput Screening (HTS) is a way of drug discovery where a large number of biological or chemical samples are tested against some measure. HTS can be the first step for a new drug discovery[1]. HTS is often used along with an 'assay'. Assay is a procedure used to generally perform tests on an organism or an organic compound[2]. There are different types of assays. Out of those 'cell counting' assay is of interest in this research. Cell counting assays can be used to perform various functions such as either counting number of living cells or dead cells, or to detect a healthy cell or a malformed cell. HTS is then essentially an automated system used to perform assay.

One of the key components of a HTS system is a microtiter plate. This small plate has grid of small indentations called 'wells'. The samples to be tested are present in these wells and the automated system is generally designed to scan through and test the samples in each well. In many applications manual measurements might be required, such as determining whether the embryonic growth is proper or not etc[2]. But it is difficult to perform manual inspection of millions of samples and defeats the purpose of building a HTS system. HTS system is synonymous with automation. Generally the system will be automated to perform all the steps, such as preparing the samples, building a library of microtiter plate, and supplying new plates to the analysis stage etc.

The research performed in this thesis is mainly concerned with High Content HTS (HC-HTS) of colony of endothelial cells. The research also addresses the need for software having versatility and sensitivity required to perform spheroid analysis. Spheroid is an aggregation of about ~750 cells. Current commercial systems have robust robotic systems but lack a good software[3].

Presence of a malformed spheroid has effects on the performance and efficiency of a HTS system. It is required that the well formed spheroids be segregated from the malformed spheroids. To do this manually would require lots of effort and time. This research focuses on developing image analysis algorithm and software along with the required robotics to automate this process of segregation of good spheroids from the bad spheroids.

As a part of the HTS, a motorized three axis XYZ table was adapted with a camera. This apparatus is programmed to scan the microtiter plate in a specific sequence and capture the images using the camera. Software was written in C++ to control the motors and the camera and scan in a specific pattern. We also developed and implemented a basic spheroid recognition algorithm for identifying spheroids and coarsely categorizing them into good versus bad classes. The spheroid detection is based on Hough Transform. The spheroids being tested are not perfectly circular. The research aims at separating almost circular spheroids from sprouted spheroids, having developed tubules. Hough Transform implementation performed in this research is very robust in this regard and was tested on a very large data set.

Another part of the research involved generating the huge dataset required to validate the spheroid detection algorithm. Since actual data from the lab was not available, a data synthesizer was also developed to generate more data from the available original data of a good and a bad spheroid. The algorithm was designed to generate infinite number of variations of the original data, but still maintaining its statistical characteristics of either being a good or a bad spheroid. The data generated from this synthesizer was used to validate the spheroid detection algorithm. We synthesized very realistic data, and even more promising results from our spheroid detection algorithm.

1.1 Thesis Organization

The thesis is organized in 6 chapters. Chapter 1 gives the introduction and the aim of thesis. Chapter 2 gives a background on HTS and Hough transform. Chapter 3 gives details of the HTS hardware, experimental setup and software written to control the same. Chapter 4 gives details about the Hough transform, the stochastic data Synthesizer, the Spheroid detection algorithm. Chapter 5 gives results and detailed figures of each step of the spheroid detection algorithm. The thesis is concluded in Chapter 6.

Chapter 2 Background

High throughput screening[4] (HTS) is a method of experimental analysis used specifically for drug discovery and other biology and chemistry related experiments. It is an assembly of hardware control devices ran by special softwares , enhanced by sensitive detectors and robotics . It involves screening of an assay plate[5] i.e. testing of a set of specific compounds to determine a particular activity or attribute. Automation is a key element in HTS that enables screening of thousands of compounds in each unit of an assay and hence the term "High" throughput screening.

2.1 High throughput screening (HTS) [4]

HTS is essentially an automated procedure for testing over a million compounds at once for a specific and desired attribute for e.g. shape, inhibition, desired reaction to a biochemical etc. For the assay, a set of reagents are placed in a microtiter plate which is made of plastic, with a grid of wells. These wells are in the multiples of 96 and are prepared for test by filling them up with the biochemical compounds that need to be examined. Some of the wells are often left empty for control measures. The matter inside the wells is then allowed to undergo desired transformation and then calibration of the plates is done either manually or by a machine. Further refined observations can be achieved on the narrowed down results obtained from the first assay by following up another assay on a second plate containing liquid from former that gave "hits" or nearly desirable results. The entire process including calibration, and generating thousands of data values becomes a matter of minutes due to a high speed data processing and control machine.

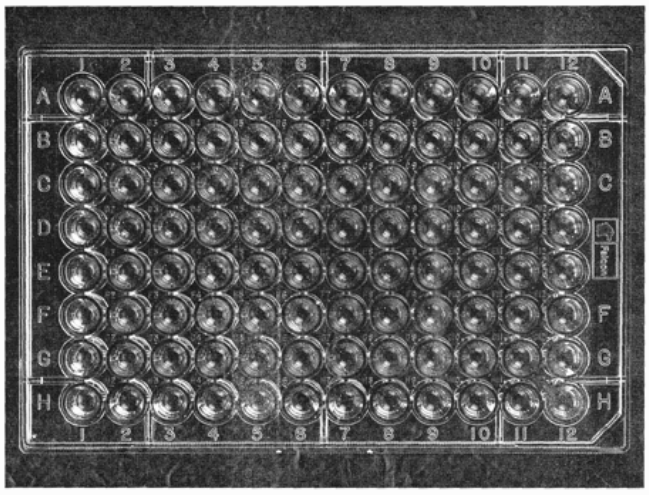


Figure 2-1: Microtiter plate (96 wells)

HTS as a powerful technology in the field of biochemistry and drug discovery has been discussed in [6][22]. Being a fairly new technology, HTS is seeing developments like use of imaging systems[7], fluorescence techniques[8], capillary electrophoresis[10], programmable plates with optical sectioning[11] etc. for higher throughput. Ultra –high[9] throughput screening refers to screening of more than 100,000 compounds per day where it is exceeding the normal screening speed of a regular HTS.

2.2 Hough Transform

The Hough Transform[12][15][18] is a feature extraction technique used to determine features of any particular shape in an image. It is widely used in digital image processing, computer vision and image analysis for isolating features of lines, circles, ellipses etc. Even arbitrary shapes can be isolated using the generalized form of the Hough transform. This technique is tolerant to gaps in the boundary descriptions and relatively unaffected by image noise as compared to the edge detectors[13]. The need for machine analysis for bubble chambers lead to its early invention in 1959[14]. Later in 1962, it was patented and assigned with the name "Method and Means for Recognizing Complex Patterns" to the U.S Atomic Energy Commission. Hough transform determines instances of objects within a set of shape parameters by a voting procedure carried out in a parameter space called the "accumulator" space[12]. Accumulator is generated by the algorithm chosen for the transform wherein the object instances/ candidates are acquired as local maxima.

The Hough transform is very useful in determining the parameters of a curve with given edge description. The edges can be located through pre-processing by either using a Canny, Sobel, Robert Cross edge detectors or any separability filter. Such a description may have considerable noise in it, since the above mentioned feature detectors only describe where in an image is a feature located, whereas the Hough Transform determines what shape the feature has and how many such features exist[13].

Consider a point (x_p,y_p) and the following two equations[21]

$$y_p = ax_p + b \qquad (2.1)$$

$$b = -x_p\, a + y_p \qquad (2.2)$$

Equation (2.1) gives the general equation of a line in the slope-intercept form. This ab plane is the parameter space associated with (x_p,y_p). There are infinite lines passing through (x_p,y_p) but all satisfying this equation for different values of a and b. Equation (2.2) yields a single line for a fixed pair of coordinates x_p and y_p.. Consider another point (x_q,y_q) that also has a line in the given parameter space which intersects the line through (x_p,y_p) at (a', b') where a' is the slope of the line making an intercept b' and containing (x_p,y_p) and (x_q,y_q). Similarly, all the points lying on this line will have corresponding lines in parameter space intersecting at (a',b').

Subdivision of parameter space into accumulator cells: We consider the normal representation of the line to avoid the problem of the slope reaching infinity.

$$x\cos\varphi + y\sin\varphi = \rho \qquad (2.3)$$

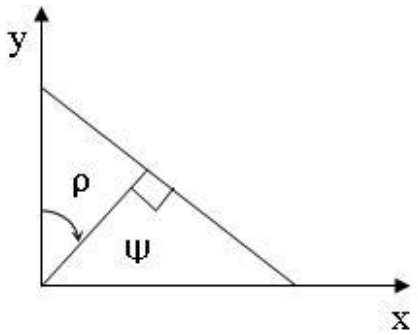


Figure 2-2: Normal representation of a line.

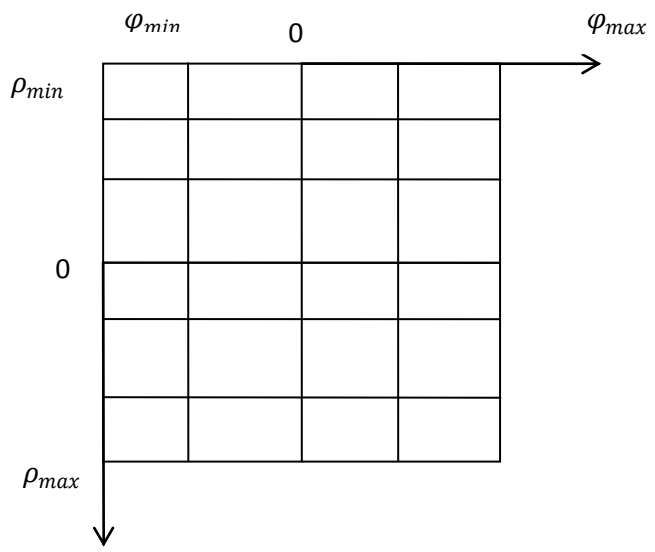


Figure 2-3: Subdivision of $\boldsymbol{\rho\varphi}$ plane into accumulator cells[21]

Where (ρ_{max} , ρ_{min}) and (φ_{max} , φ_{min}) are the expected ranges of distance and argument values. The cell at (*i*,*j*), with accumulator value A(*i*,*j*) corresponds to the square associated with (ρ_i, φ_j). Initially these cells are set to zeros. Then for every (x_m,y_m) in image plane , ρ is allowed to have all the possible values on ρ axis and the corresponding φ is calculated, which is rounded off to the nearest possible value on the φ axis. If ρ_k solves for φ_l, then A(*k*,*l*) = A(*k*,*l*) +1. A value V in A(*m*,*n*) corresponds to V points in the $\rho\varphi$ plane lying on the line given by

$$xcos\varphi_n + ysin\varphi_n = \rho_m \qquad (2.4)$$

When analyzing the image, the coordinates of the points in the edges determined by the filter are known (x_{ic} , y_{ic}) and serve as constants within the parametric equation of the curve for e.g. a circle.

$$(x_{ic} - x_o)^2 + (y_{ic} - y_o)^2 = R^2 \qquad (2.5)$$

Where x_o and y_o are the coordinates of the center of the circle of radius *R*.

The accumulator in the above case will be 3-D i.e. the parameter space will have three coordinates. The algorithm runs transforming every (x_{ic},y_{ic}) into a (x_o,y_o,*R*) curve in 3D space. The accumulator cells falling on these curves are incremented and as with each

iteration their value increases, the resulting maximas substantiate the presence of a corresponding curve, here circle, in the image. The circular Hough transform[17] is discussed in detail in chapter 4.

The following figure illustrates a common variation of the Hough transform called the Generalized Hough Transform used for arbitrary shapes.

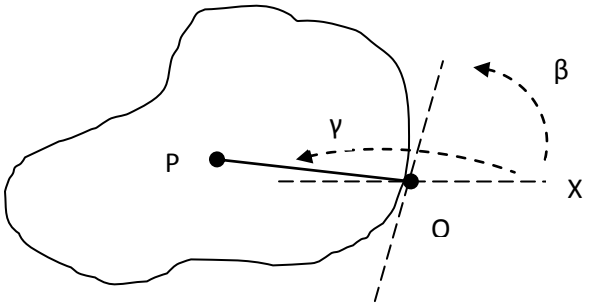


Figure 2-4: Generalized Hough Transform

In the case of arbitrary shapes[13], a look up table is used to find the relation between the Hough parameters and the shape's boundary points. This lookup table is constructed before hand by the help of a prototype shape as above. Let us suppose we know the shape and orientation , and say take a point P (x_p,y_p) inside it. The shape of the feature can then be defined by the distances of boundary points from this point and the angles made by perpendiculars drawn to the joining lines. i.e. if there is a point O (x,y) on the boundary such that OP is given by d and the orientation of OP is given by γ while the orientation of a normal drawn to it is given by β. The look up table will consist of the pair of d and γ indexed by β for all its values.

The transformation is then defined as[13]

$$x_p = x + d\cos(\gamma) \quad (2.6)$$

$$y_p = y + d\sin(\gamma) \quad (2.7)$$

The values of d and γ are determined from the look up table for known values of β. If the values of β are not known , then the complexity of the accumulator increases

by one more parameter to account for variation in the values of β. Other variations of Hough transform include the kernel[12] based Hough transform and the use of gradients[16] to reduce voting . Tahir[19] et al discuss the implementation of Hough transform for the detection of cylinders while Vosselman[20] et al discuss the 3D implementation.

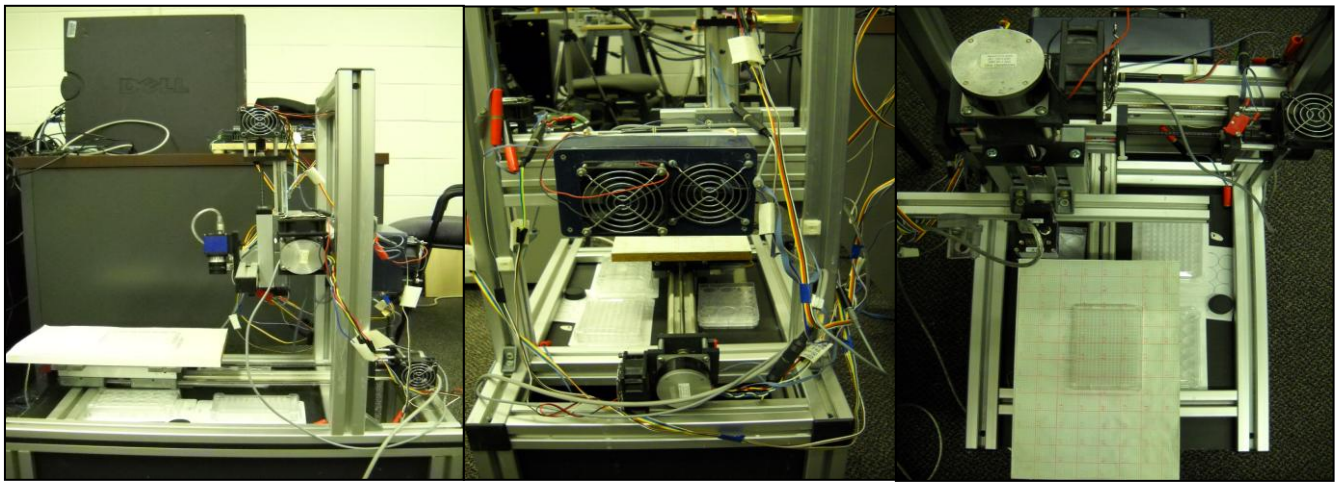
Chapter 3 High Throughput Screening:Scanning Protocol

High throughput screening (HTS) is used to conduct biological or chemical tests on a very large number of samples. It generally consists of some kind of sensor, controlling software and a post processing logic to conduct the test and give results.

In this research we build a HTS system to distinguish good and bad endothelial spheroids. A perfectly spherical spheroid is considered a good endothelial spheroid. A malformed spheroid is considered bad because they lack uniformity in density and length leading to deteriorated performance in the presence of angiogenic stimulators.

3.1 Experimental Setup

A motorized three axes XYZ table was adapted with a camera such that it can translate along the Z and Y directions. Figure 3-1 (a) through (c) show the machine from different profiles. An electronic DI/DO (Fig. 3.2) using a USB interface is used to control the stepper motor drivers. The camera used is a 2 megapixel MatrixVision BlueFox with a USB interface. The operating system is Microsoft XP. We used a desk top system but a laptop computer would also function with this system .



(a) **(b)** **(c)**

Figure 3-1: HTS Machine (a) side view (b) back motor (c) top view

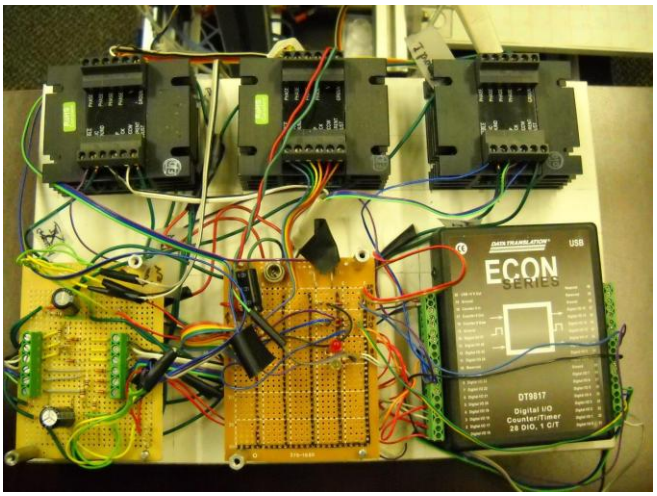


Figure 3-2: DIDO

As seen from Figure 3-1(c), a microtiter plate is placed on the calibrated platform which rests on the Y axes. The user can determine the position of the starting end and the length and breadth of the plate by noting down the markings for input in the screening process.The motors take step input so given home as the {0,0,0} coordinate, a positioning function would position the axes to absolute step coordinates. Functions for conversion from millimeter values to “step” coordinate values were also implemented.

3.2 Software Development

Controlling functions for the motors were written in .NET by Yonchang Wang. The software for scanning and capturing the images was developed as a part of this research in VC++.

Figure 3-3 shows the interface for inputting dimensions (mm) of input/output plates for screening. The user can choose between the input and output dishes to scan. With the help of the calibrations marked on the platform, one can easily find the starting locations for the camera as well as the length and breadth for the dishes. The information about dimensions is used to calculate the approximate size of the well so that the camera may stop over each well at the right distance. Figure 3-4 shows the camera in a live mode.

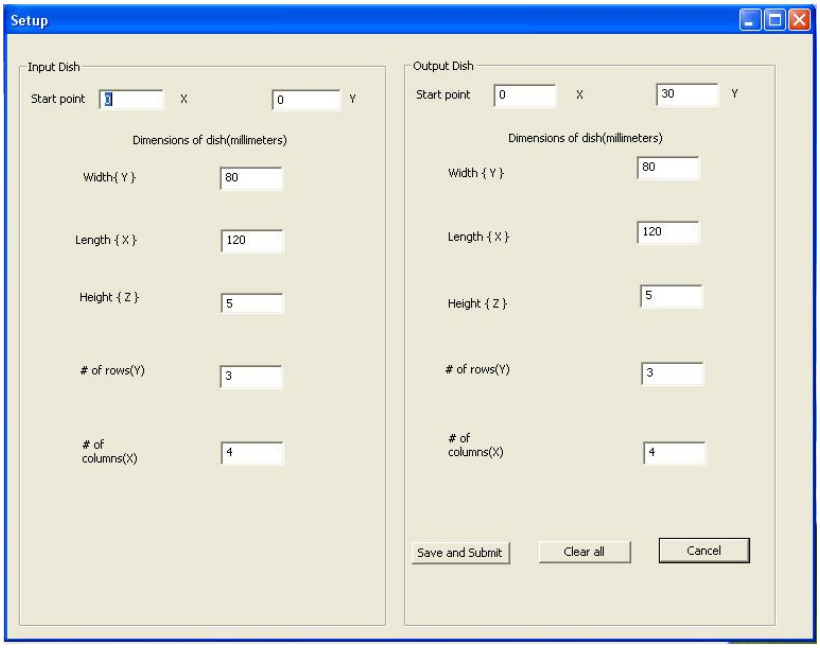


Figure 3-3: setup dialog box.

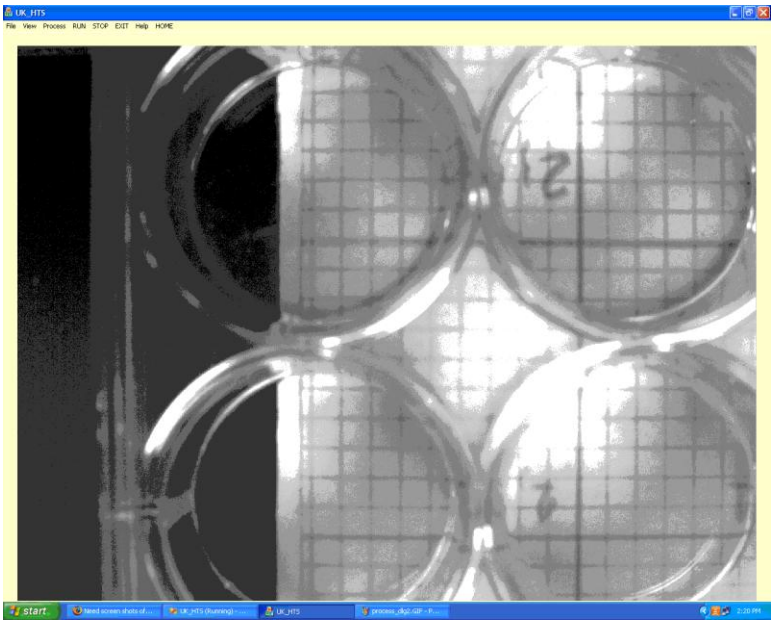


Figure 3-4: Scanned Template

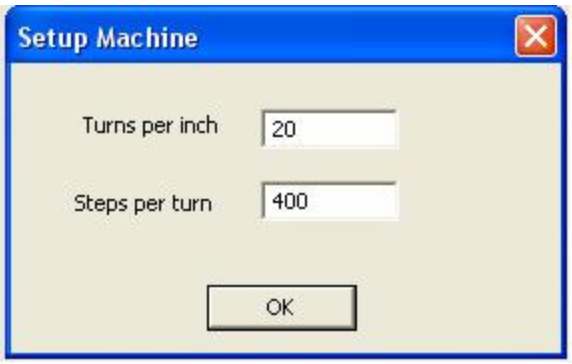


Figure 3-5: Machine Setup

As shown in **Figure 3-5**, every turn of the screw moves the axes by 400 steps and 20 such turns were needed to move them by an inch. However if the motors were to be replaced by different kind, the user only needs to change these values through the dialog box shown above and not in the code.

3.2.1 Scanning Algorithm

The scanning/screening process is guided by a motion pattern that minimizes total travel distance. This zig zag pattern is shown by the red arrows in Figure 3-6. The "home" position is set by driving the axes until mechanical switches are activated, thereby establishing a home or {0,0,0} step coordinate. Figure 3-7 shows the flow chart implementation of the scanning algorithm.

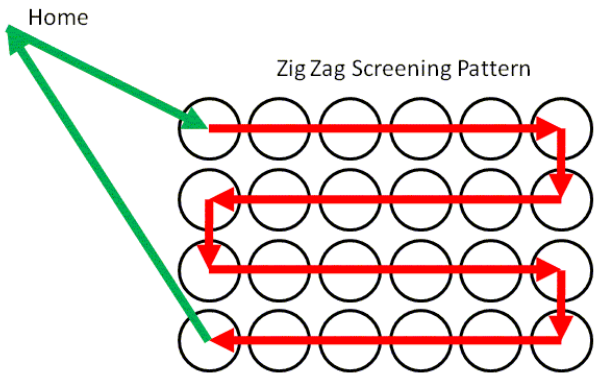


Figure 3-6:Zig Zag screening trajectory

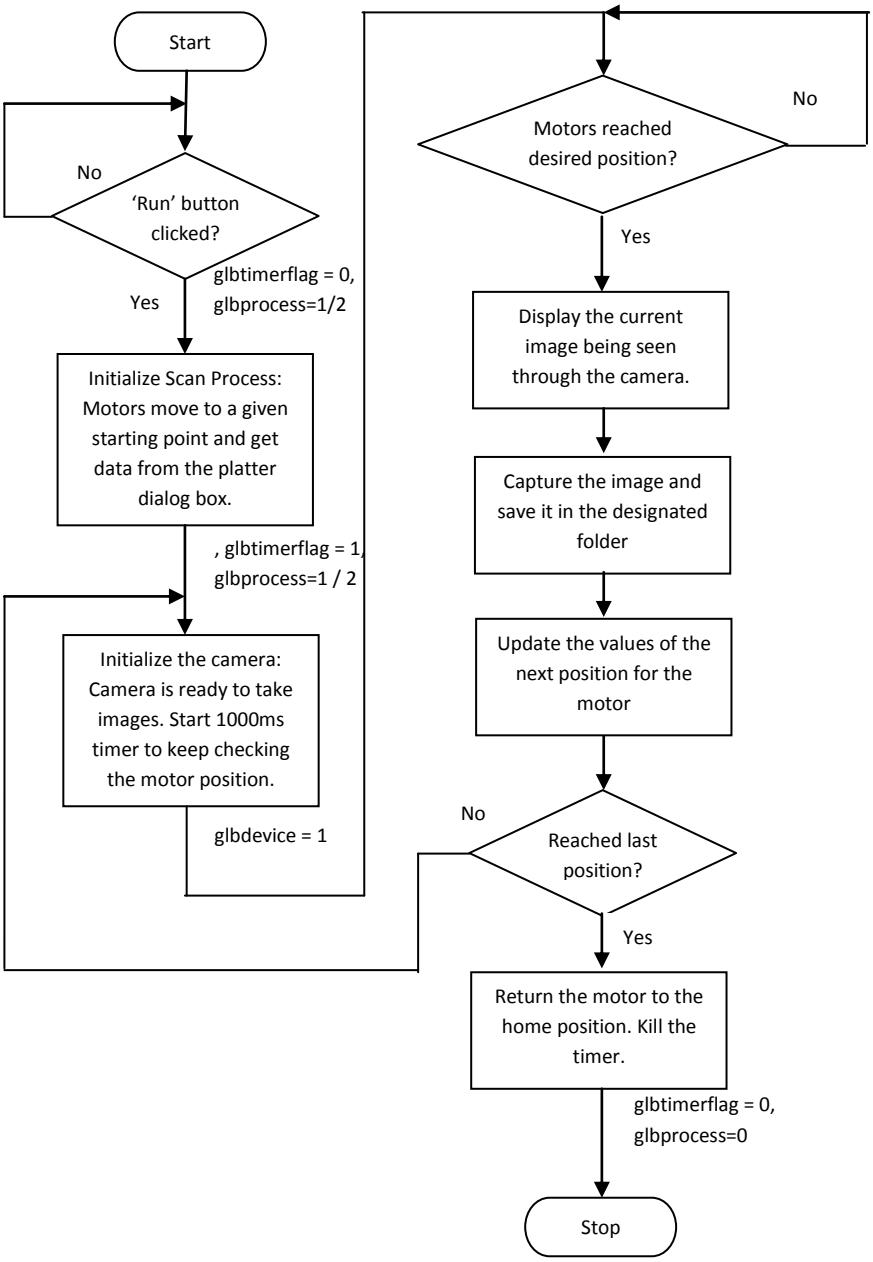


Figure 3-7: Scanning Algorithm flowchart

Scanning Algorithm:

1. If user clicks on **Run**, set glbtimerflag(shows timer status)=0,glbprocess(type of dish to scan;1 for input ,2 for output)=0.
2. **Initialize scan process**: Start timer thread, get data from setup platter dialog and process the starting point values and steps to move. Move motors to the updated position. Set glbprocess=1 or 2 (chosen by user), glbtimerflag=1.

3. Initialize camera,open device and set glbdevice(status flag for camera)=1.
4. **Timer thread**(1000msec):This timer thread checks the position of the motors every 1 sec and also does the following.

a. Wait to reach the approx. position. For any glbprocess, setup display dialog and display current camera image.
b. Capture image and save in designated folders.
c. Update with next calculated position.
d. If motor reached end position , return home position and set glbtimerflag=0,glbprocess=0. Else go to next position and repeat from a to d.

Chapter 4 Pattern detection using Circular Hough Transform

4.1 Circular Hough Transform

As discussed in the background chapter of this thesis, Hough transform is a means of extracting features of specified shape in an image. A circular hough transform pertains to detecting circular shaped objects or features in an image. For spheroid detection in 2D images, we look for circular shapes and then set a threshold parameter that separates a well-formed circular shape from the malformed one.

We know that a circle can be represented by the following equations, where equation (1) relates to Cartesian space while equation (2) and equation (3) to Parametric space[17].

$$(\mathrm{x}-\mathrm{m})^2+(\mathrm{y}-\mathrm{n})^2=R^2 \qquad (4.1)$$

$$x=m+R\cos\emptyset \qquad (4.2)$$

$$y=n+R\sin\emptyset \qquad (4.3)$$

Where *m,n* are the coordinates of the center of the circle of radius *R*,in *x* and *y* directions respectively. Thus the parameter space for a circle will have three parameters namely, *R*, *m* and *n* which means that the accumulator space will be of the order 3. For simplicity, the radius can be treated as a constant, and transforms can be performed for various values of radii.

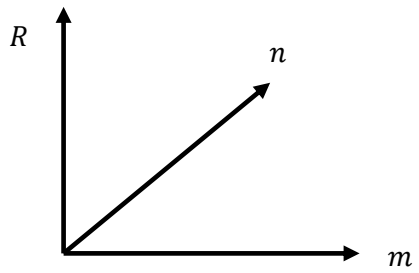


Figure 4-1: Parameter space for a circle

For a constant radius *R*, the accumulator space division for a circle will look similar to that shown in chapter 2.

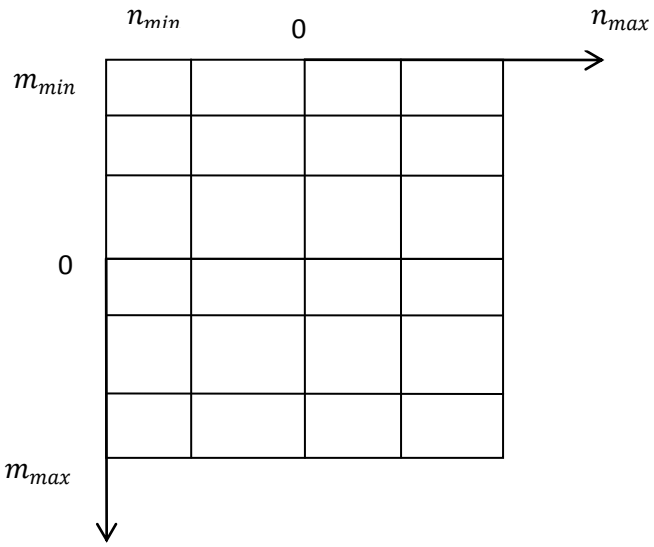


Figure 4-2: Accumulator space[21]

The range of values of *m* and *n* is generally the number of edge points detected as a pre-processing by any of the edge detectors or morphological operations.

Once the edge points are known, a circle is drawn keeping the edge points as centres and *R* as their radius. All the coordinates that lie on the perimeter of this circle are then incremented in the accumulator space, where the initial value in the accumulator cells is zero.

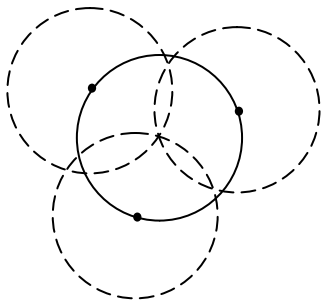


Figure 4-3: Location of center is given by the peak in the accumulator space.The dark points are the edges detected prior CHT[17].

As seen in the figure above, the point that is likely to have the highest value in the accumulator space for being on the perimeter of these circles (dotted circles) is the centre of the original circle (bold circle).This is also a mathematical fact, that two or more circles cutting a single circle, all of the same radii, have the centre of the circle they cut, on their perimeter. Hence one should expect a maxima in the accumulator cell that corresponds to the coordinates of the centre of the circle being detected.

4.2 Synthesis of data

We generated our own dataset for experimentation purpose. This synthesis process was an attempt towards saving time while waiting on a second party for generating actual data from their process. We originally had one bitmap images of a single good spheroid and a single bad spheroid. The synthesis of our databank was simply a stochastic model of a good spheroid (in terms of shape), so far that it still looks a good spheroid but with different pixel coordinates ,texture and orientation. The same was done with the bad spheroid. Thus, by simply changing the orientation and shape of the original image, we were able to generate an arbitrarily large databank of good and bad spheroids. We call them 'masks', and each mask is stochastically similar to its parent image but NOT identical to it.

Mathematically,

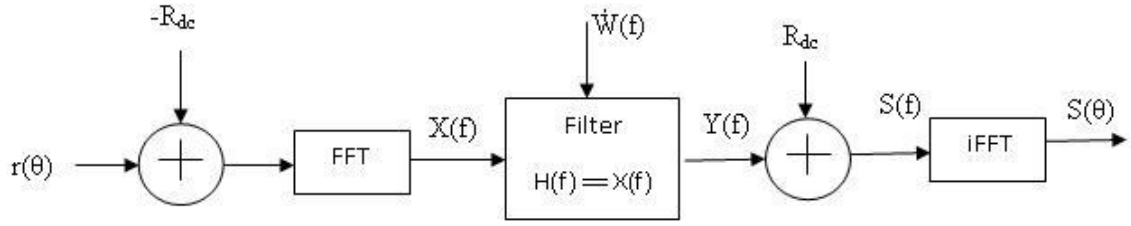


Figure 4-4: Data Synthesizer Model

Where,

$r(\theta)$ = Distance of each edge point as a function of theta

R_{dc} = Mean of the distances of all the edge points

X(f) = Filter Transfer function

H(f) = Fourier Transform or r(θ)

W(f) = Zero mean white Gaussian noise

Y(f) = Filtered white noise

S(f) = Resultant Fourier Transform of the filtered noise

S(θ) = Resultant synthesized distance as a function of theta

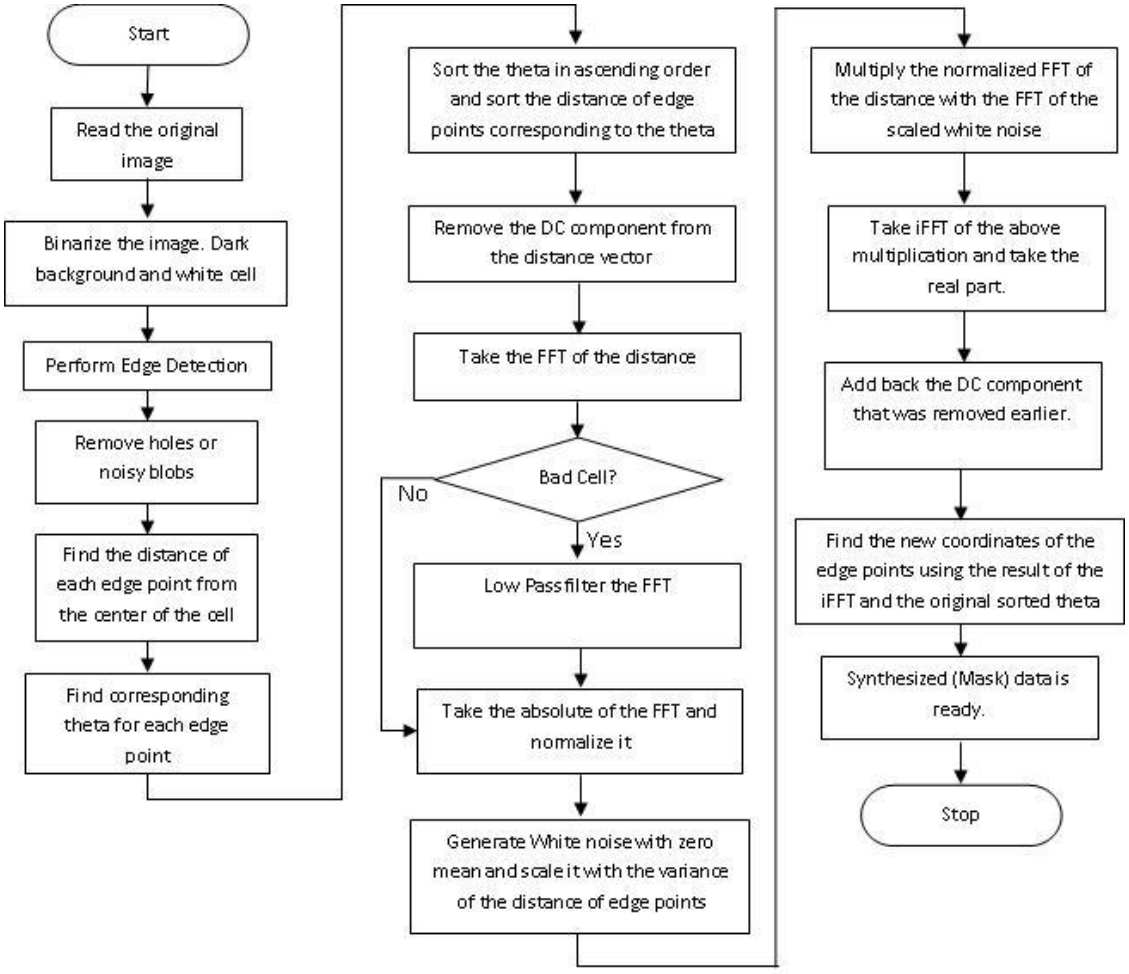


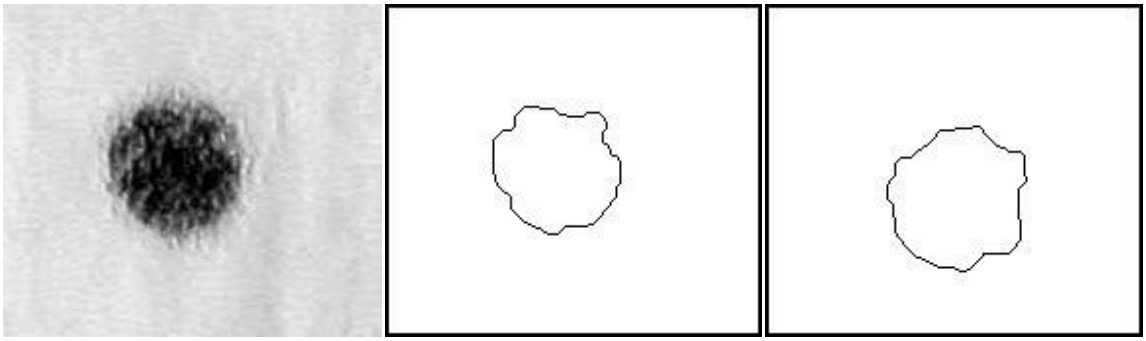
Figure 4-5: Flow Chart for Synthesis of Data

4.2.1 Algorithm for the synthesis of 'masks'

1. The original Goodspheroid or Badspheroid bmp image was binarized to separate the background noise from the spheroid in the foreground.
2. The edges of the spheroid were detected. For both the good and the bad spheroid, there were a few holes in the detected boundary as well as inside the spheroid. These were filled using some filling operations.
3. In addition to the holes, for a bad spheroid, there were additional smaller blobs in the binarized image,that were present as tubules in the original picture. These were identified and subsequently removed.
4. This final edge detected image with no holes was ready for processing.
5. The coordinates of the edges were found. The distance of each edge point from the center of the spheroid was calculated.(The center is determined by taking the mean of x and y coordinates resp.).
6. The value of angle, theta, for each corresponding point is calculated and then sorted in ascending or descending order. The distance value for each edge point is then arranged w.r.t the sorted theta. This removes any ambiguity caused due to multiple values or concavities. The dc part from the distance vector is removed.
7. The FFT of this vector of distance values is taken .If the original image is a good spheroid, the following is skipped, while for a bad spheroid, the vector is passed through a low pass filter to remove any frequencies higher than 20Hz.
8. The magnitude of the fourier transformed vector is taken and normalized.
9. A zero-mean (unity variance) white noise is generated and scaled with variance of the distance vector. This scaled noise is then multiplied with the fourier transformed distance vector.
10. The inverse FFT of the above product is taken and its imaginary parts are removed. The previously removed dc , is now added to this vector.
11. This new distance vector renders new coordinates. These are plotted using the original sorted theta values.
12. The new image with new boundary is checked for any holes again, both inside and outside. These are removed by the same operations as in step 2.

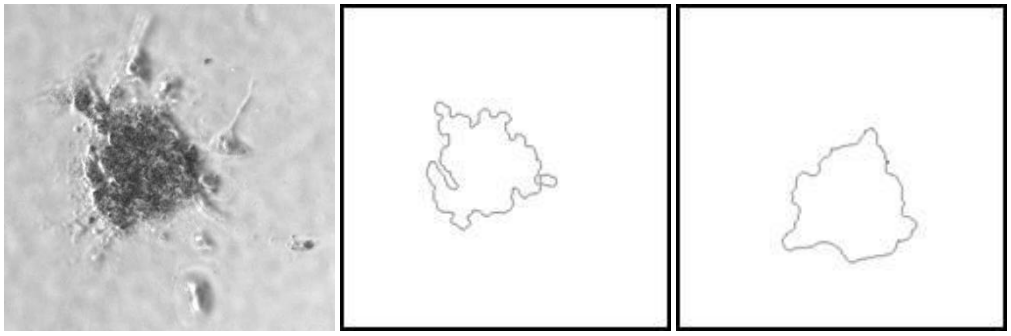
13. Thus by changing, the white noise scaling factor, we were able to synthesize infinite data. The white noise generated had different values each time, therefore it also contributed to the uniqueness of the shape. Of the lot, we selected the images that looked similar, but NOT identical to the parent image for our experimentation. The examples are illustrated below.

Synthesis of data from the original sample



(a) **(b)** **(c)**

Figure 4-6: (a) GoodSpheroid.bmp (b) After edge detection (c) New mask



(a) **(b)** **(c)**

Figure 4-7: (a) BadSpheroid.bmp (b) After edge detection (c) New Mask

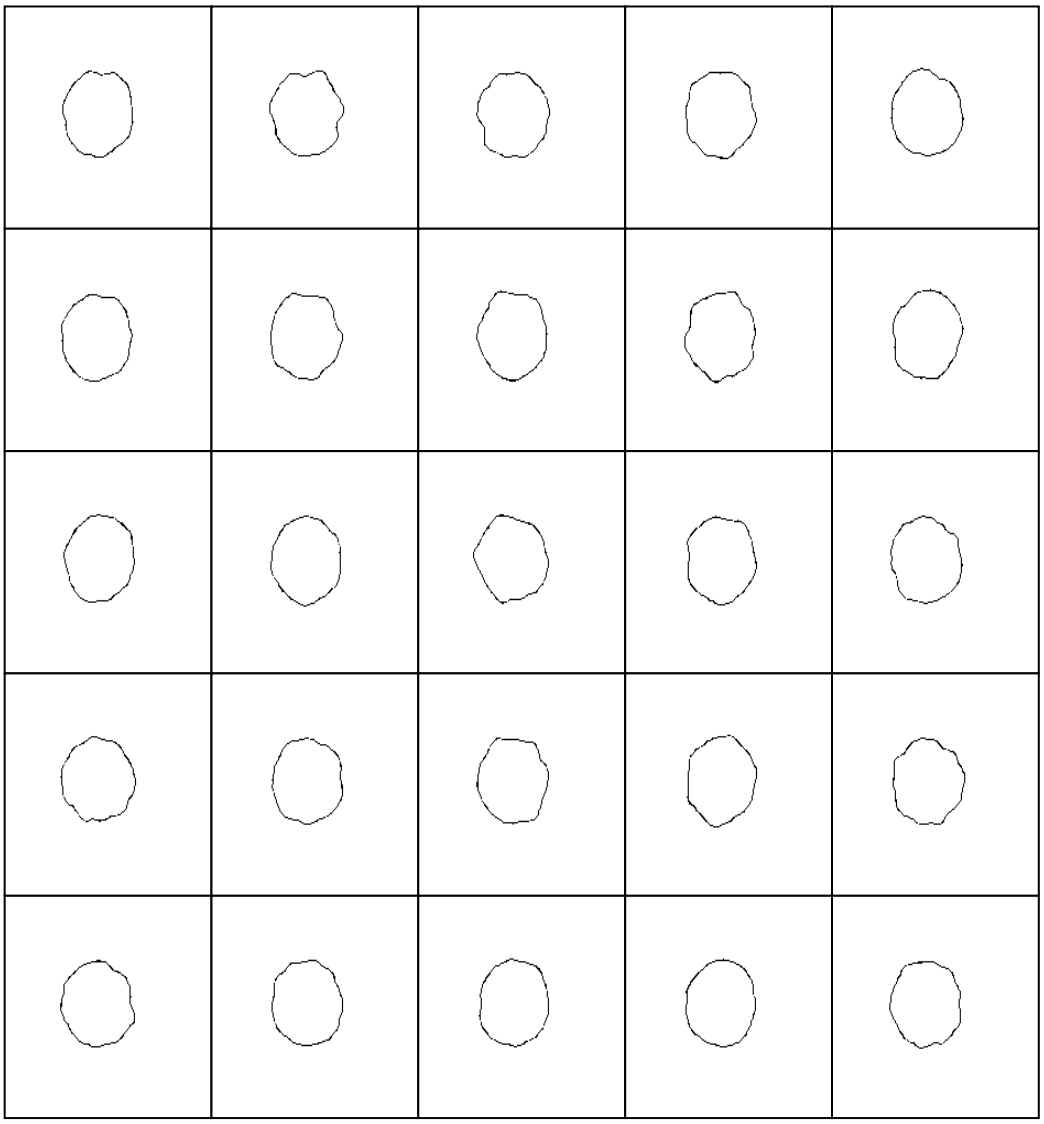


Figure 4-8: 25 Synthesized Good Spheroids

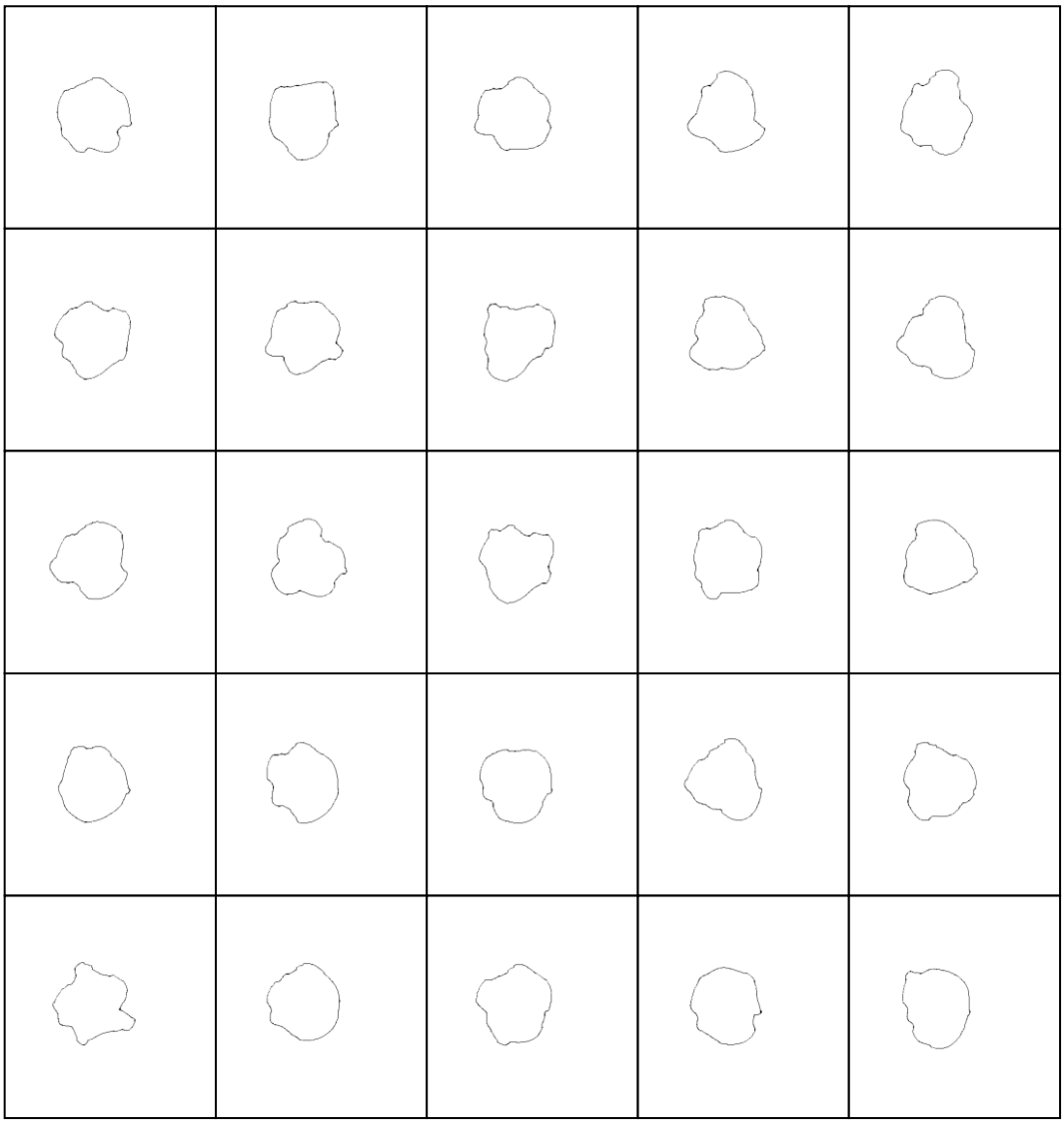


Figure 4-9: 25 Synthesized Bad Spheroids

4.2.2 Scaling factor

The white noise used here carried zero mean or unity variance. If we simply multiplied the distance vector's fft with the white noise, we achieved little to no change in the output. The variance was not high enough to bring out any noticeable difference. Hence, the white noise was scaled to match that of the real spheroid. For the constant variance of the distance vector, the variance of the white noise needed to be scaled to have the output look similar to the original distance vector, by virtue of higher variance.

The white noise *h(t)* had to be normalized given that $\tilde{y}(t) = h(t) * \tilde{s}(t)$, so that

$\text{Var}\{\tilde{y}(t)\} \longrightarrow \text{Var}\{\tilde{s}(t)\}$, where $\tilde{y}(t)$ is the synthesized distance vector, $\tilde{s}(t)$ is the original distance vector.

By experimentation, we realized that the more the scale value approached the variance of the distance vector, the better results we found. At approx. the variance value, the results started taking their desired shape. Hence, the scale was taken as a multiple of variance of the distance vector.

4.3 Algorithm for circle detection in images

1. Define global parameters.
    1. **PreProcessing** =0; {false if using synthesized images,true if using actual data} .
    2. **NUM_OF_FILES**=<number> say i.
    3. **FINAL_RESULTS**= array with 4 columns and i rows such that

Image number	Radius	Peak to edge ratio	Mean Square error

    4. **index**= <starting file num>. radii={25,….35} pixel units.
2. **Begin loop1** for all files. For index ≤ file_name ≤ i.
    1. Check if preprocessing required. If PreProcessing=1 goto **PreProc** else image_to_use= file_name.
    2. Define **Peak2Edge**= array that stores peak to edge points ratio for as many elements as there are radii. **MSE**= array that stores mean square error values for the corresponding elements.
    3. Find coordinates of edge points. Each edge point => (x,y).
    4. **Begin loop 2** for all radii.
        i. For radii(first) ≤ j ≤ radii(last). Perform CHT.

ii. **CHT**: create accumulator space of same size as the image being processed. Initialize A(k,l)=0 ∀ (k,l ∈ *R*).

1. **Begin loop 3** for voting in the parameter space per edge point. For every edge point i.e. (x,y), make a circle of radii(j) with (x,y) as center.
2. Find coordinates of all points that lie on the perimeter of this circle, say (m,n). then A(m,n) = A(m,n) +1;
3. Repeat for all edge points.
4. **End loop 3** (loop ends after processing last edge point).

iii. Locate peak value P_k , in the accumulator cell => location of center of spheroid. (if more than 1 peaks , choose the one with min mean square error (see **msqer**) . if same mean square error, choose any point as center.)

iv. **Measure 1**: Peak to edge ratio: Peak2Edge(j) = P_k /num of edge points.

v. **Measure 2**: Mean Square Error: MSE(j) = **msqer** for current data.

5. **End loop 2**.
6. Find radii with least mean square error and the corresponding peak to edge ratio.
7. Store values for every file in **FINAL_RESULTS**.
8. **Plot figures**: Draw circle of above found radius on the image that was being processed centered at the peak coordinates
9. Repeat for next file.

3. **End loop 1**.(loop ends after processing last file).
4. Display **FINAL_RESULTS**.

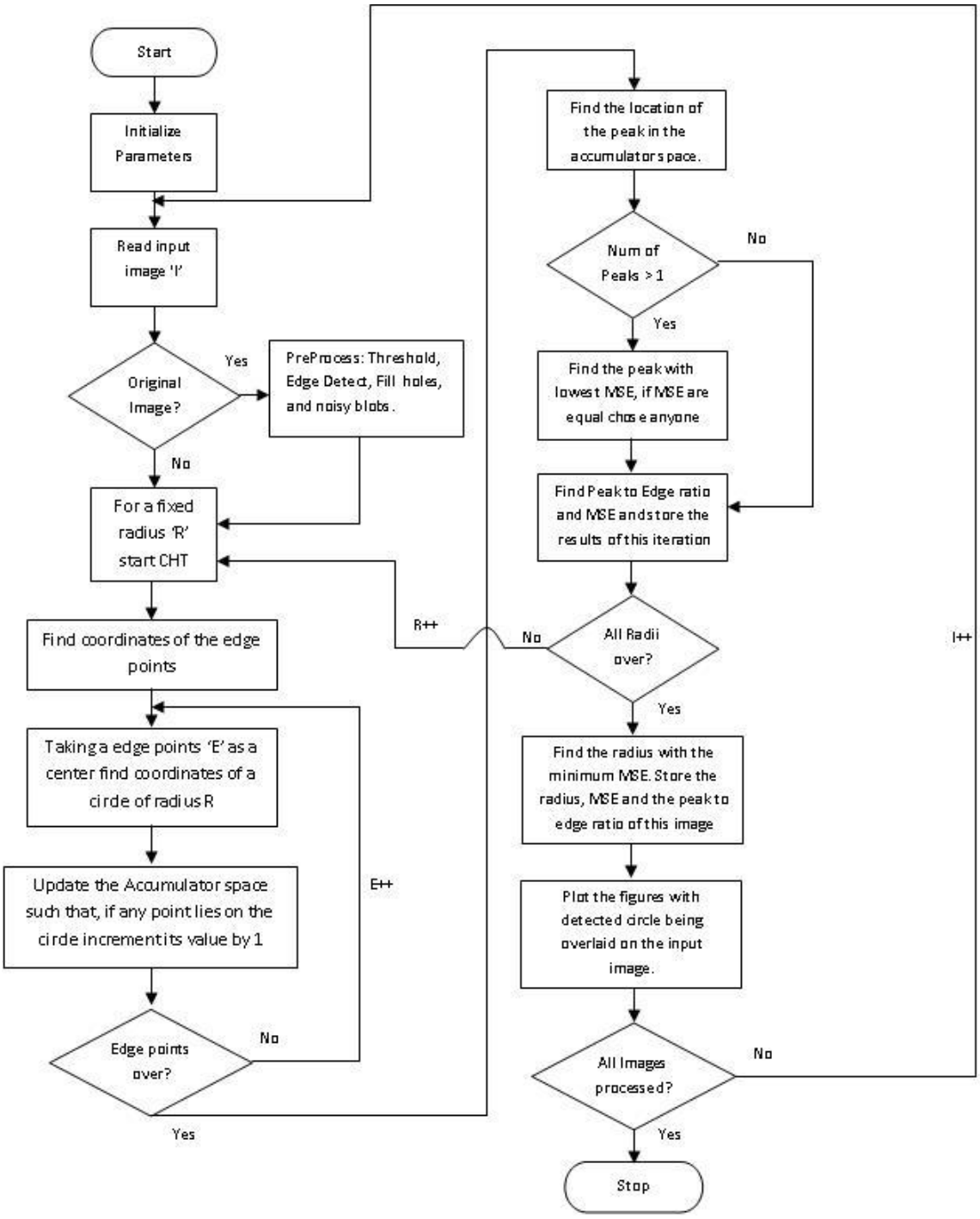


Figure 4-10: Circular Hough Transform

4.4.PreProc algorithm

1. Read input file (file_name).
2. Binarize to separate the background from the spheroid.
3. Remove holes ,if any.
4. Detect the edges. (file_name_proc)
5. image_to_use= file_name_proc.

4.5.Msqer algorithm

1. Input arguments (center coordinates, radius, edge coordinates, number of edge points)
2. For each edge point P_i, find $r_{hi}=\sqrt{(x_c - x_i)^2 \quad + (y_c - y_i)^2}$
3. $error_i = \ (radius - r_{hi})^2 \ \forall$ i.
4. return msqer= mean ($error_i$).

4.6 Results of above algorithm implemented on a perfect circle illustrated with different stages.

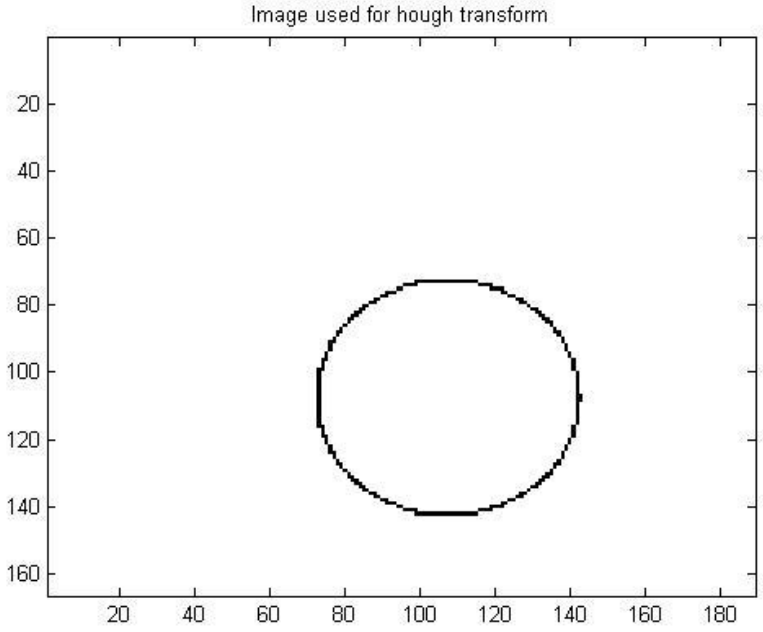


Figure 4-11: Stage I: Image used for Hough transform

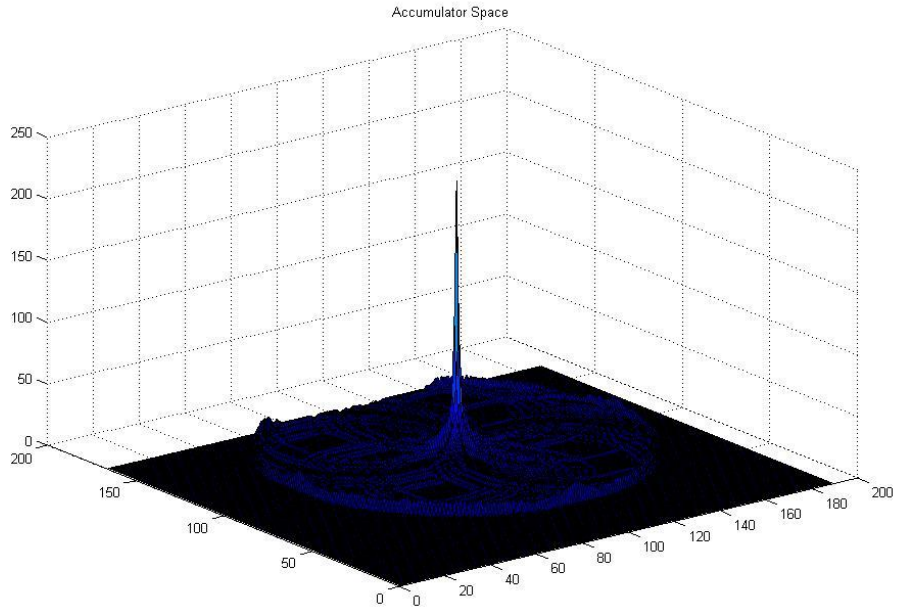


Figure 4-12.Stage II: Accumulator space with the peak shown at center.

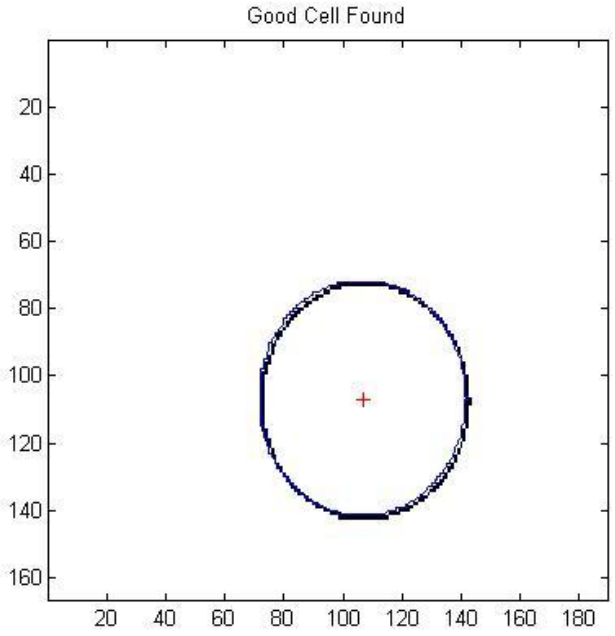


Figure 4-13:Stage III: Good Spheroid detected (boundary and center demarcated)

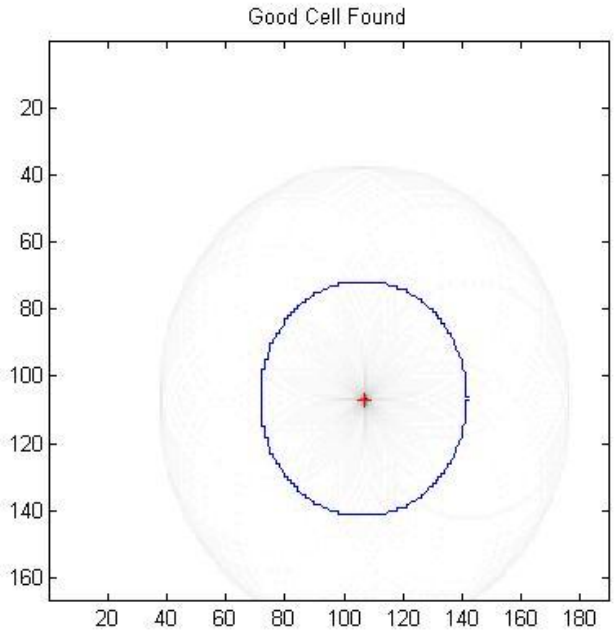


Figure 4-14:Stage IV: Circle detected is shown in the accumulator space. Peak is shown with red spot, highest concentration of votes. The blue rings represent distribution of votes,with the innermost one showing the boundary of the spheroid detected.

4.7 Measures of goodness of a spheroid

In order to determine whether the spheroid is a good spheroid or a bad spheroid, we measure 2 measures of quality of a spheroid, namely

1. Peak to Edge Ratio (PER)
2. Mean Square Error

4.7.1 Peak to Edge Ratio (PER)

To determine whether the spheroid is a good spheroid(closer to a perfect circle) or not an easy measure is the ratio of the value of the peak in the accumulator space to the total number of edge points. In an ideal circle , considering there are 360 points on the circle the ratio would be 1. For any shape not a perfect circle this ratio would be less than 1, since not all points would contribute a vote at the center of the circle. Peak to Edge ratio is given by

PER =(Value of the Maximum Peak)/ (Total number of points on the edge of the spheroid) (4.4)

4.7.2 Mean Square Error (MSE)

Mean Square Error is used as an estimator for the difference between the expected value and the actual value. MSE is useful when the error can be either positive or negative. In this case we measure the error between the computed radius (distance of an edge point from the center of the circle found using Hough Transform) and the radius of a perfect circle most closely matched with the spheroid. MSE is calculated as follows

$$\text{MSE} = \frac{\sum_{i=1}^{i=Num\ \_Edge\ \_Pts} (Radius\ _i - R\ )^2}{(Num\ \_Edge\ \_Pts)} \quad (4.5)$$

Where

Num_Edge_Pts = Total Number of edge points on the spheroid

$Radius_i$ = Distance of Each edge point from location of the peak.

R = Radius of the perfect circle most closely matched to the spheroid.

Chapter 5 Results

Large data set can be produced using the data synthesizer algorithm. In this case we synthesized 25 good spheroids and about 25 bad spheroids. All the synthesized images were processed using the circular Hough transform described in chapter 4. Parameters such as Mean Square Error (MSE) and Peak to Edge Ratio (PER) were measured to make a decision whether the spheroid was a good spheroid of a bad spheroid.

In this chapter we show the entire processing of the original good spheroid and the original bad spheroid with results of the intermediate spheroids. In the end we present the results obtained for all the rest of the synthesized data, in a tabular format.

5.1 Detection of a Good Spheroid

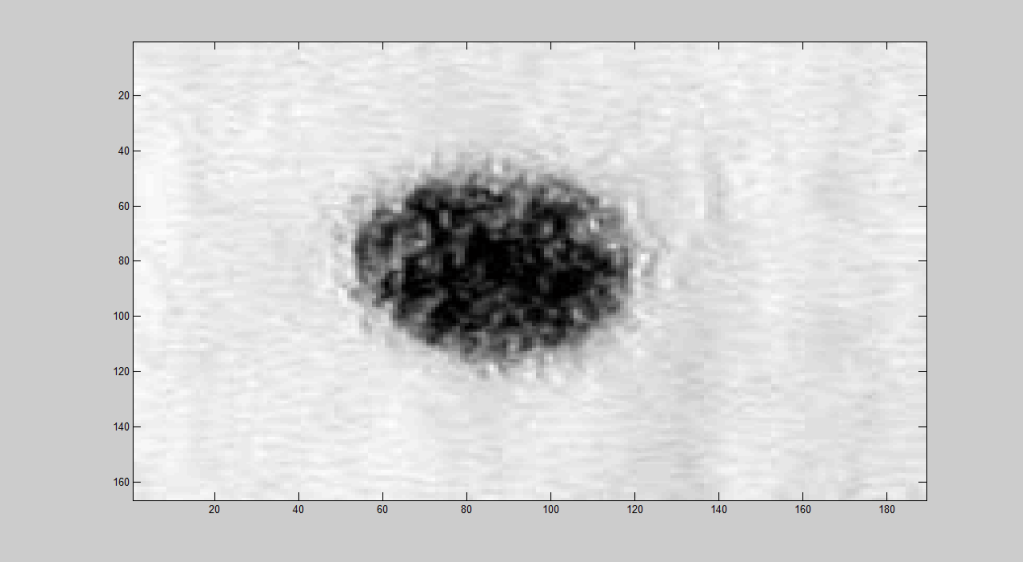


Figure 5-1: Stage I: Original bitmap image of a good spheroid.

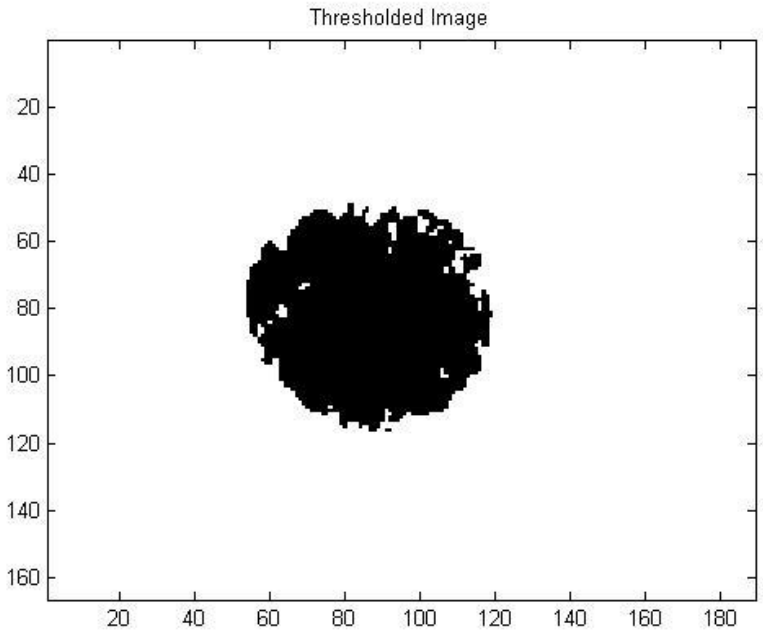


Figure 5-2: Stage II: Thresholded image. Observe some holes inside and on the boundary of the spheroid.

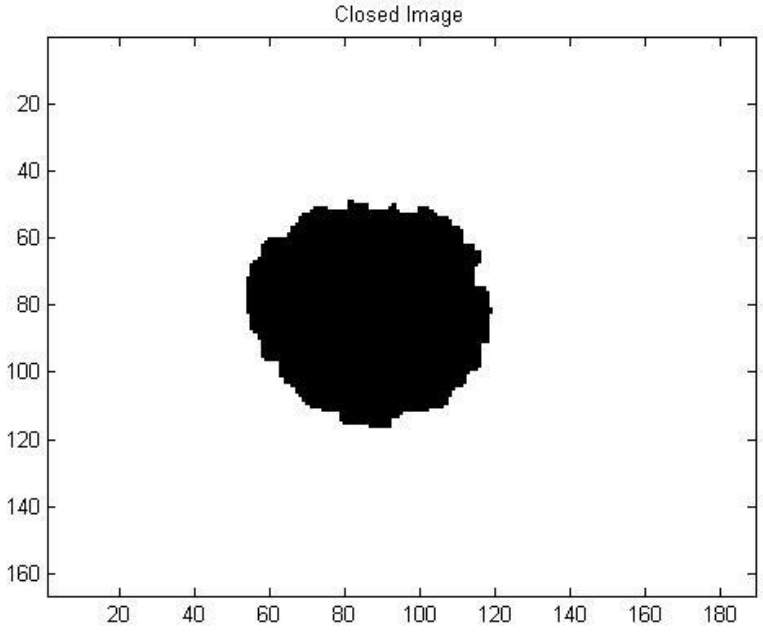


Figure 5-3:Stage III: After removing the holes.

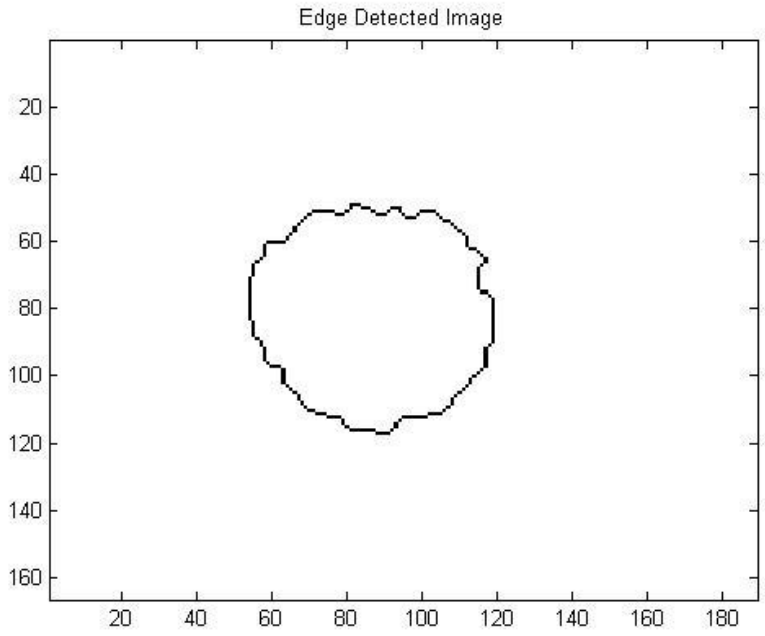


Figure 5-4: Stage IV: Edges detected. This image is then used for hough transform.

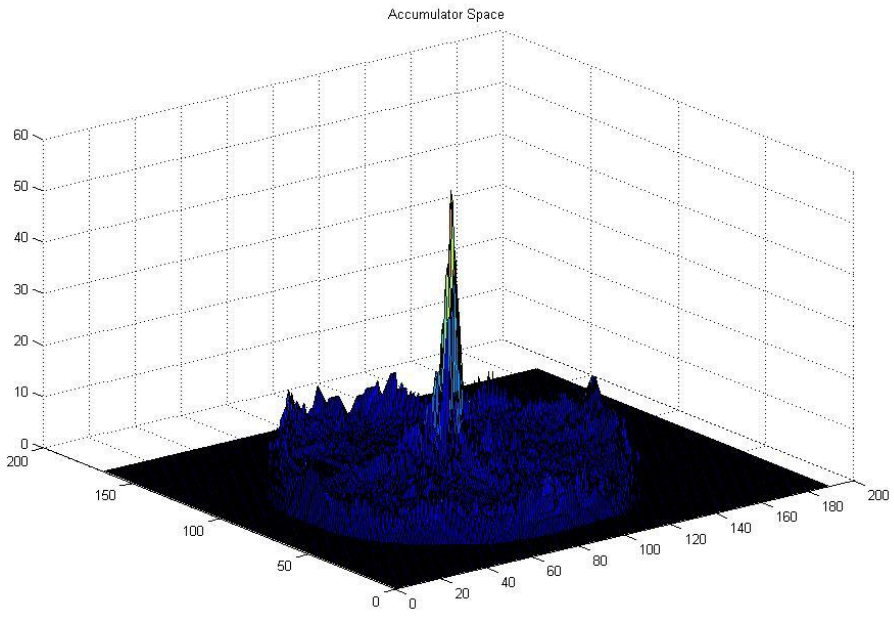


Figure 5-5: Stage V: Accumulator space with one maxima and several small peaks.

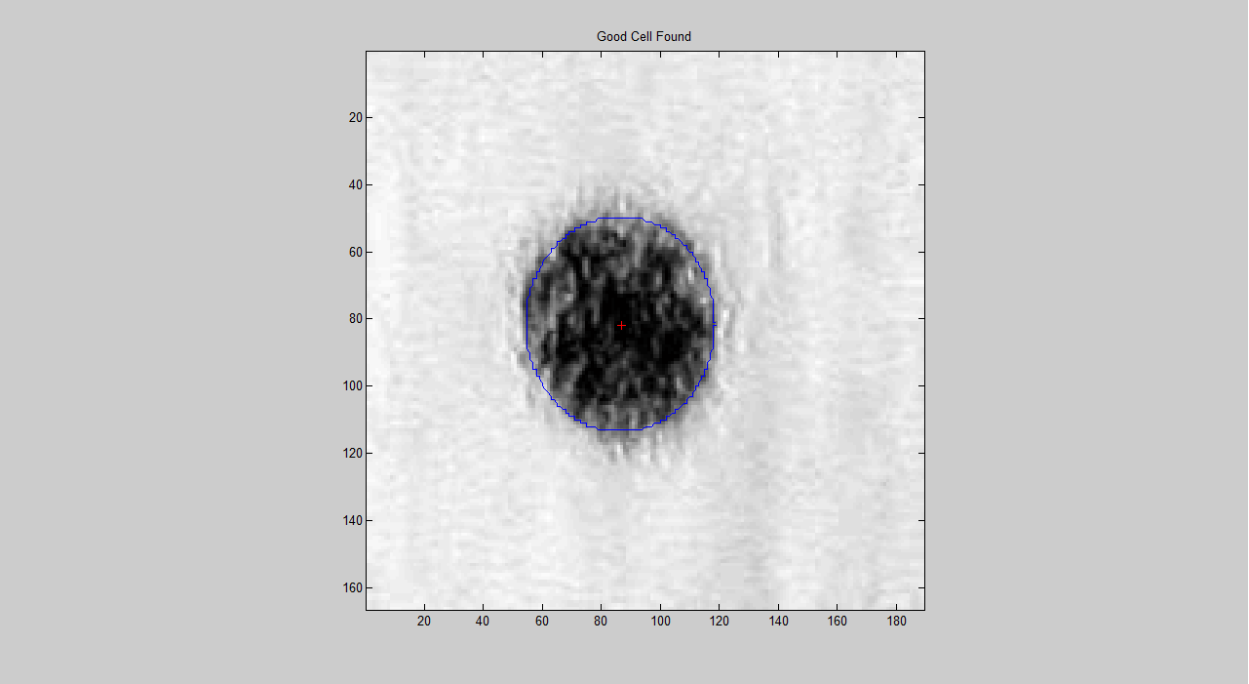


Figure 5-6 :Stage VI: Circle detected and overlaid on original image. Boundary and center demarcated.

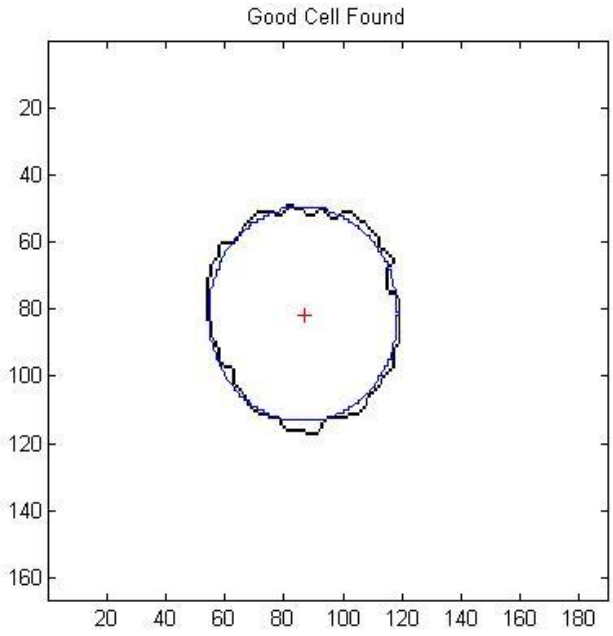


Figure 5-7 :Stage VII: Circle detected and overlaid on the edge detected image.

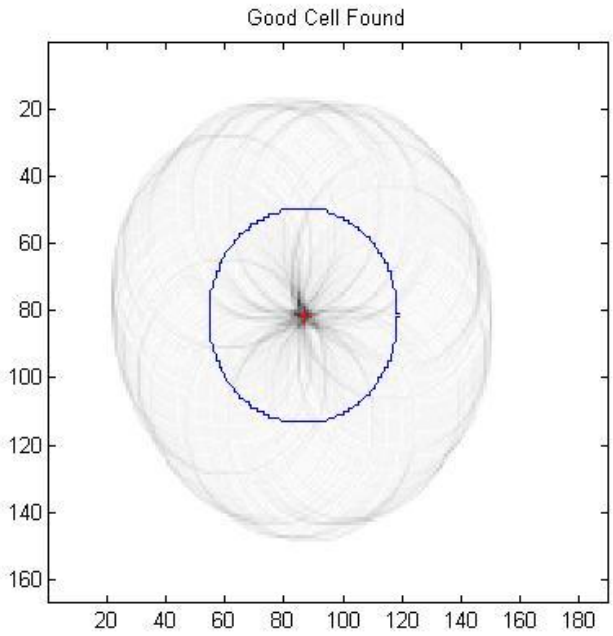


Figure 5-8 :Stage VIII: Overlaying the detected circle in accumulator space. Detected circle is shown with the innermost blue ring and center at the peak shown by red spot. Red spot denotes highest concentration of votes.

5.2 Detection of a Bad Spheroid

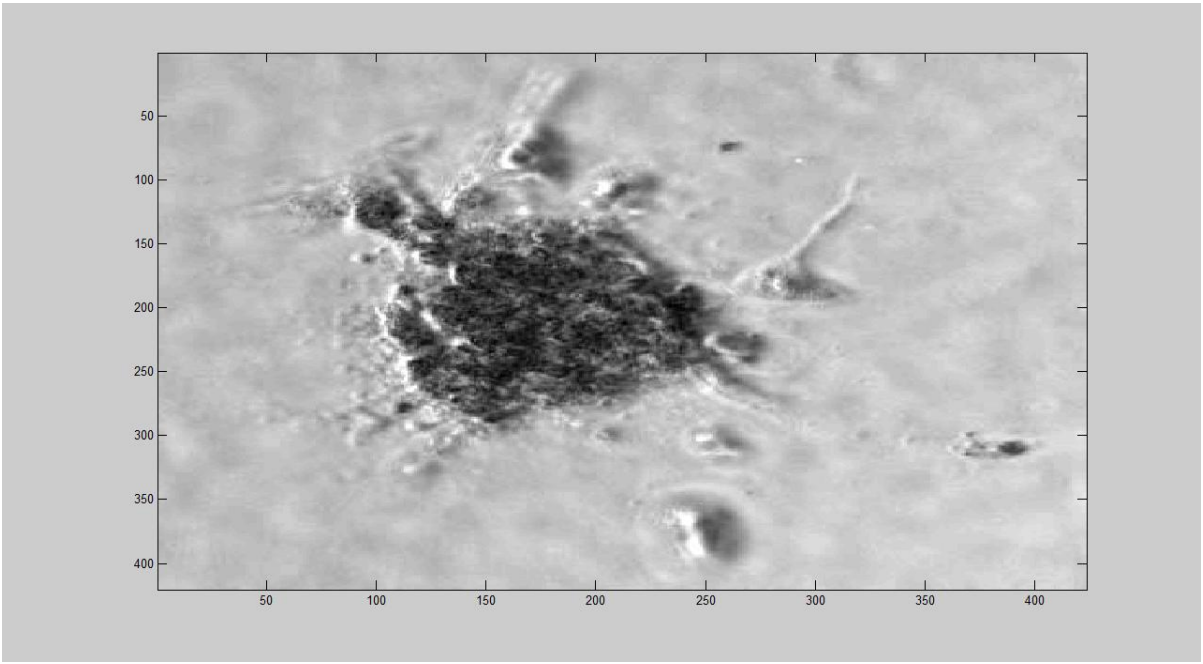


Figure 5-9 :Stage I: Original bitmap image of a bad spheroid.

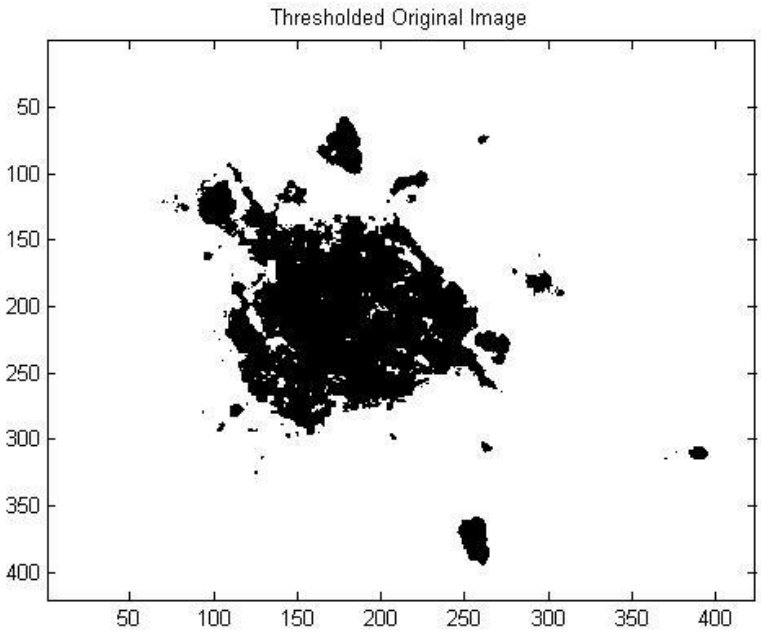


Figure 5-10 :Stage II: Thresholded image

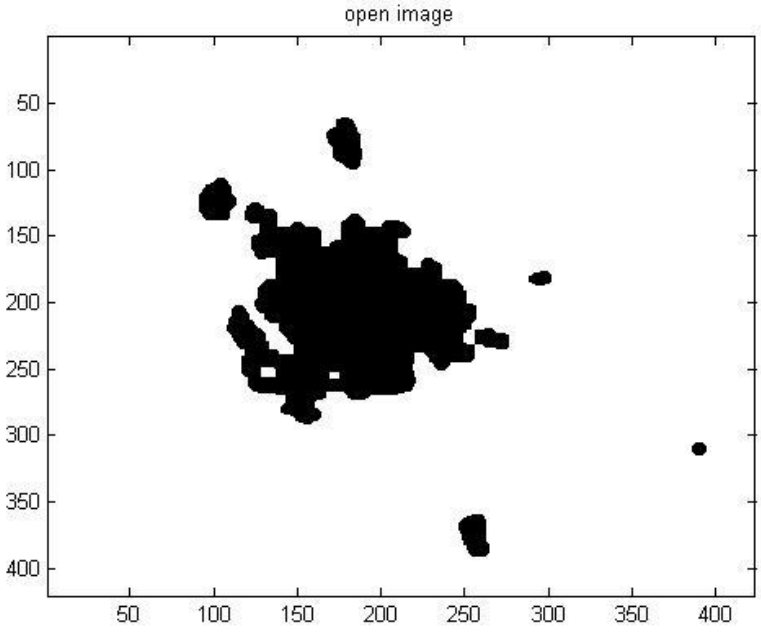


Figure 5-11 :Stage III: Holes were partially removed.

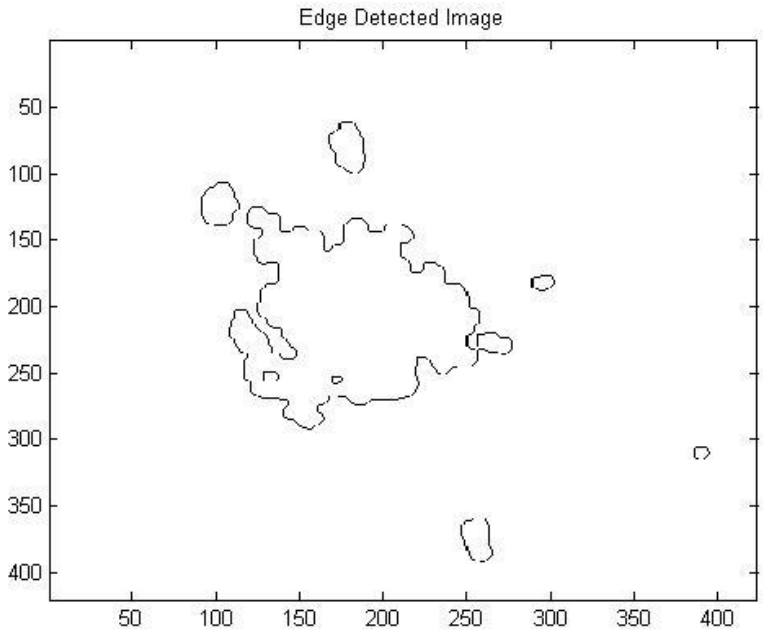


Figure 5-12 :Stage IV: Edges were detected. Observes some blobs inside and outside the spheroid.

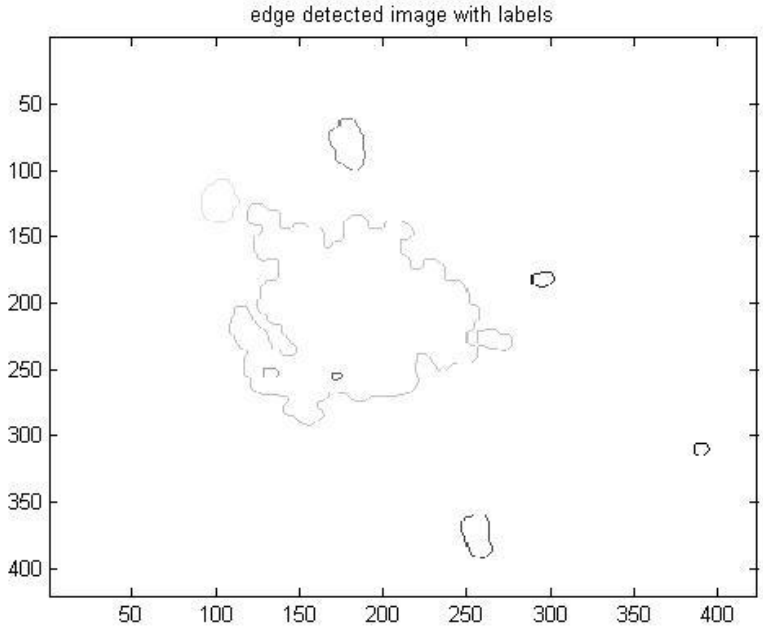


Figure 5-13 :Stage V: Labeling all the objects before removing. This distinguishes the spheroid from unwanted blobs, and then they can be removed.

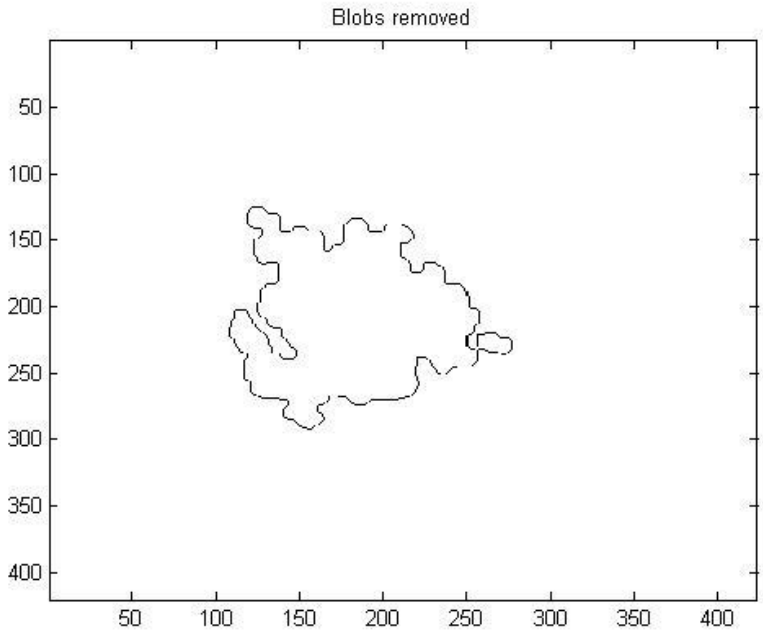


Figure 5-14 :Stage VI: After removing the blobs.This image is used for CHT.

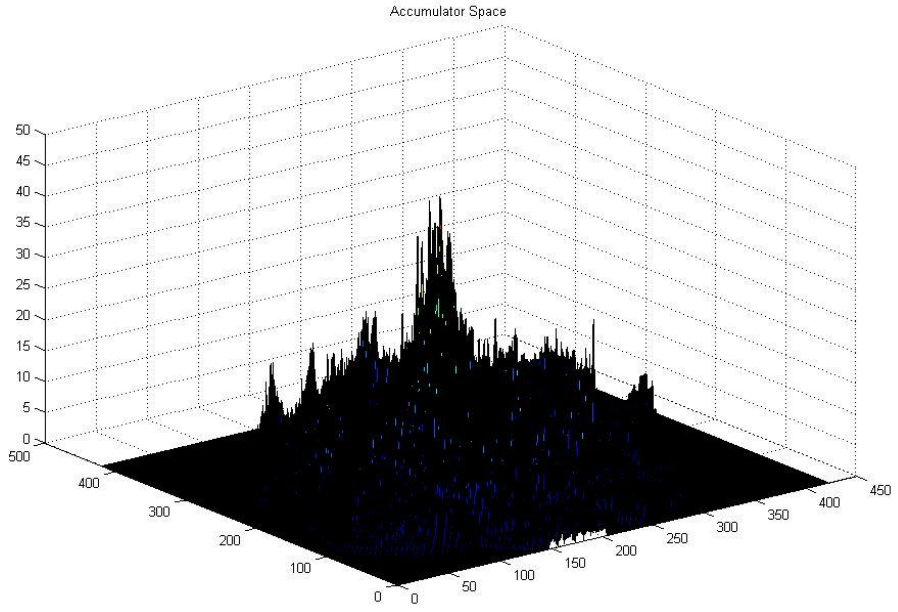


Figure 5-15 :Stage VII: Accumulator space for a bad spheroid. Notice various peaks with almost same value in different locations. Based on the peak to edge ratio and MSE, the algorithm determines that it is a bad spheroid.

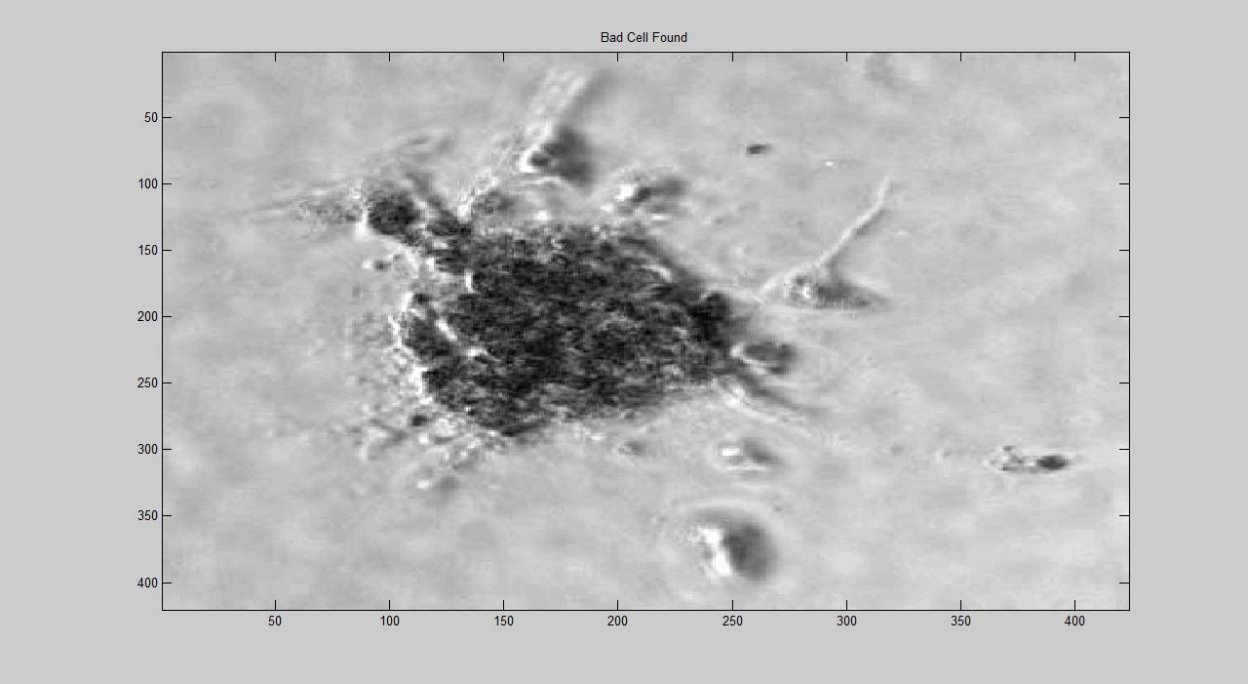


Figure 5-16 :Stage VIII: Bad spheroid found.

5.3 Total Results

Table 1.FINAL RESULTS FOR GOOD SPHEROID MASKS

Image type	Expected	Opt. radius	PER	MSE	Found
Perfect Circle	Good	35	0.91667	0.35658	Good
Goodspheroid1	Good	32	0.38916	2.2398	Good
Goodspheroid2	Good	32	0.37931	2.5148	Good
Goodspheroid3	Good	32	0.45	2.1898	Good
Goodspheroid4	Good	32	0.40201	1.0001	Good
Goodspheroid5	Good	32	0.4604	1.4387	Good
Goodspheroid6	Good	32	0.475	0.99632	Good
Goodspheroid7	Good	32	0.35025	1.711	Good
Goodspheroid8	Good	31	0.35294	2.756	Good
Goodspheroid9	Good	32	0.32864	2.205	Good
Goodspheroid10	Good	32	0.36139	2.2016	Good
Goodspheroid11	Good	32	0.36275	1.5642	Good
Goodspheroid12	Good	32	0.47236	1.063	Good
Goodspheroid13	Good	31	0.31683	3.2356	Good
Goodspheroid14	Good	32	0.37019	1.7871	Good
Goodspheroid15	Good	32	0.39487	2.1883	Good
Goodspheroid16	Good	32	0.35294	1.9306	Good
Goodspheroid17	Good	32	0.44103	0.97463	Good
Goodspheroid18	Good	32	0.38384	1.5742	Good
Goodspheroid19	Good	32	0.35377	2.7898	Good

Table 1 Continued					
Goodspheroid20	Good	32	0.38571	1.8752	Good
Goodspheroid21	Good	32	0.40385	2	Good
Goodspheroid22	Good	32	0.44554	1.4765	Good
Goodspheroid23	Good	32	0.42927	1.4821	Good
Goodspheroid24	Good	32	0.45455	1.5069	Good
Goodspheroid25	Good	32	0.43842	1.2857	Good

Table 2.FINAL RESULTS FOR BAD SPHEROID MASKS

Image type	**Expected**	**Opt. radius**	**PER**	**MSE**	**Found**
Perfect Circle	Good	35	0.91667	0.35658	Good
BadSpheroid1	Bad	65	0.11837	89.446	Bad
BadSpheroid 2	Bad	65	0.11373	67.8	Bad
BadSpheroid 3	Bad	65	0.10763	112	Bad
BadSpheroid 4	Bad	65	0.088757	108.6	Bad
BadSpheroid 5	Bad	58	0.076772	237.63	Bad
BadSpheroid 6	Bad	65	0.09899	150.67	Bad
BadSpheroid 7	Bad	65	0.090909	61.163	Bad
BadSpheroid 8	Bad	64	0.088235	155.91	Bad
BadSpheroid 9	Bad	65	0.10421	224.81	Bad
BadSpheroid 10	Bad	65	0.10638	336.88	Bad
BadSpheroid 11	Bad	65	0.11943	118.95	Bad
BadSpheroid 12	Bad	65	0.094118	179.97	Bad

Table 2 Continued					
BadSpheroid 13	Bad	65	0.094862	117.03	Bad
BadSpheroid 14	Bad	65	0.11618	49.969	Bad
BadSpheroid 15	Bad	65	0.097713	51.788	Bad
BadSpheroid 16	Bad	65	0.12391	33.216	Bad
BadSpheroid 17	Bad	65	0.11983	55.823	Bad
BadSpheroid18	Bad	65	0.10381	75.851	Bad
BadSpheroid 19	Bad	64	0.08805	121.75	Bad
BadSpheroid 20	Bad	65	0.12146	62.115	Bad
BadSpheroid 21	Bad	64	0.059259	156.37	Bad
BadSpheroid 22	Bad	65	0.16269	57.681	Bad
BadSpheroid 23	Bad	65	0.10288	86.278	Bad
BadSpheroid 24	Bad	65	0.12065	62.969	Bad
BadSpheroid 25	Bad	65	0.13136	78.651	Bad

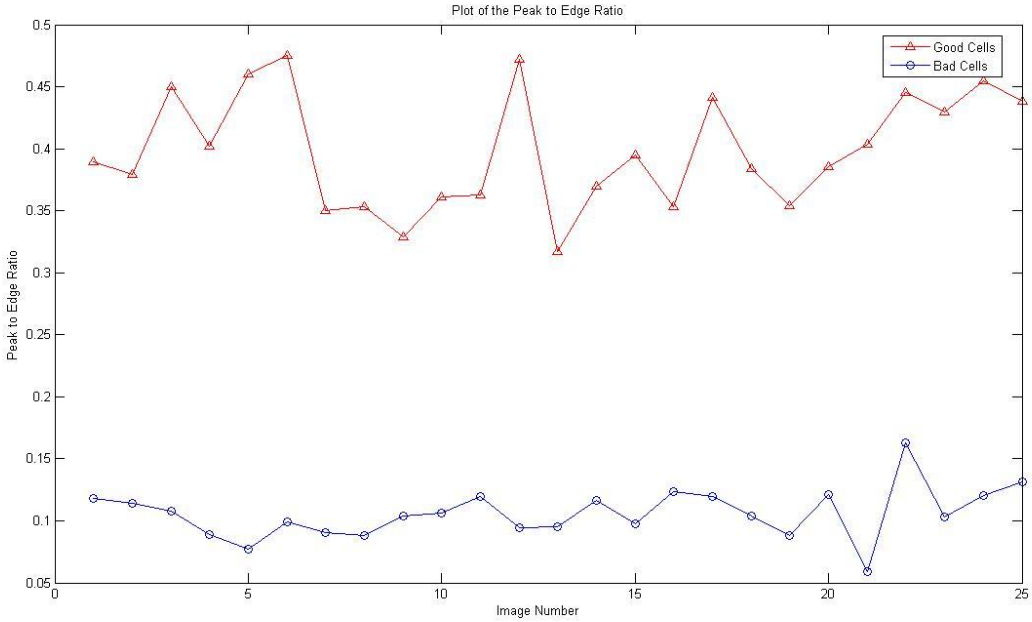


Figure 5-17: Comparison of Peak to Edge Ratio between Good and Bad spheroids.

(X axis-File number 1-25,Y axis-PER, Red-Good, Blue-Bad)

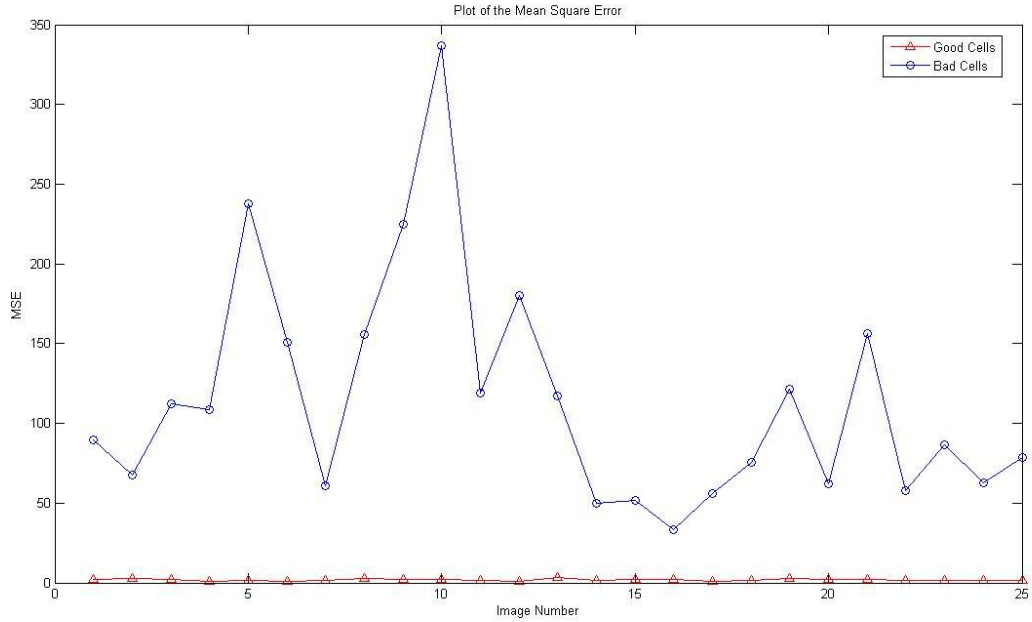


Figure 5-18: Comparison of MSE values between Good and Bad spheroids.

(X axis-File number 1-25,Y axis-MSE, Red-Good, Blue-Bad)

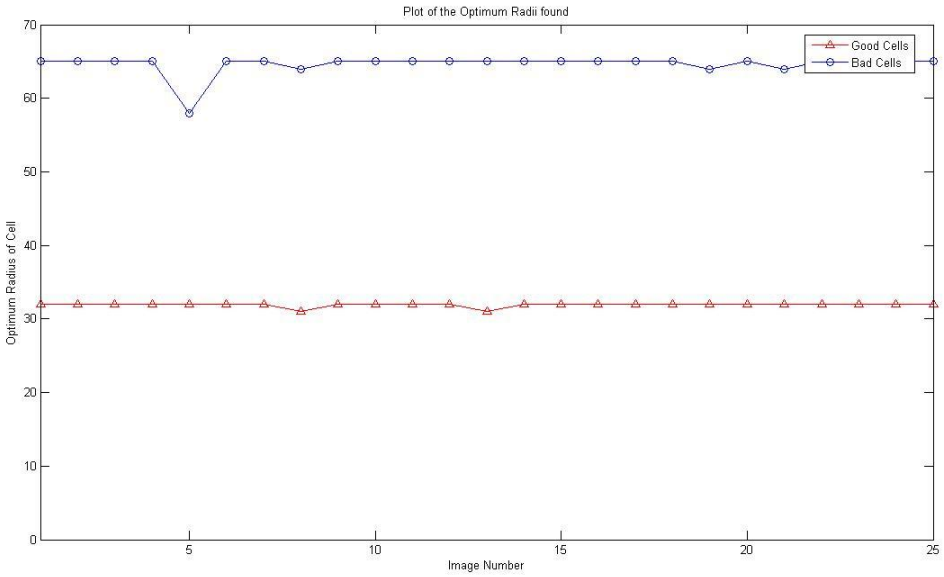


Figure 5-19: Comparison of Optimum Radii values between Good and Bad spheroids.

(X axis-File number 1-25,Y axis-Radius, Red-Good, Blue-Bad)

Chapter 6 Conclusion and Future Work

The High Throughput Scanning system prototype under study looks very promising and enables the testing of endothelial cells to be automated. We were successfully able to demonstrate that the process of capturing the images of the spheroids can be completely automated using the motor assembly and the camera, interfaced with software. The software can be further improved by making it a multithreaded application wherein one thread can run the motor, other the can control the camera and take pictures, while a third one can be used to run the spheroid detection algorithm.

We have demonstrated that Hough Transform can be successfully used for detecting spheroids in a 2D image. We presented two measures of determining whether the spheroids were good (circular) or bad (non-circular). The algorithm currently only searches for circles in a given range of radii. It can be further improved to detect circular shapes of any radii in a given image.

6.1 Numerical Costs

Following are the execution times:

- Processing 25 bad spheroids for 10 radii : 34.1170 sec
- Processing 25 good spheroids for 10 radii : 13.338 sec
- Processing Original Good spheroid : 1.3420 sec
    - Hough Transform logic : 0.0483 sec
- Processing Original bad spheroid : 5.0540 sec
    - Hough Transform logic : 0.3361 sec.

Currently the execution times required are less than 1 sec per spheroid. This algorithm can be further improved to have optimized Hough transform logic to reduce the execution time even further. Also it should be noted that the execution times are also dependent on the file size. The file size for the good spheroid was 5KB,and that for the bad spheroid was 15KB. Also the file size for synthesized good spheroid was 4KB and that for the synthesized bad spheroid was 23 KB.

Appendix A

A.1 Data Synthesizer Code

A.1.1 Synthesizer for Good Spheroid

```
%  Priyanka Chaudhary
%  Synthesizer to generate data for processing through the cell
detection
%  algorithm.
%  University of Kentucky Spring 2010
%  The image is thresholded to get only the cell
%  The image is inverted to get cell as a white portion
%  White noise is generated and band pass filtered
%  The FFTs of the image and the noise are added together, this creates
the
%  mask.
%  Add color to the white part of the noise to create texture.
%  taking fft of distance versus theta plot.

clc;
clear all;
close all;

%Parameters
threshold = 150; % for thresholding the image
center_x  = 107;
center_y  = 101;

%Read Good Cell Image
orig_image = imread('Picture1.bmp');

%Take one channel
orig_img_ch1 = orig_image(:,:,1);
[Nx My] = size(orig_img_ch1);
orig_img_thresh = zeros(Nx,My);

% Threshold the image
for i = 1:Nx
    for j = 1:My
        if(orig_img_ch1(i,j)>threshold)
           orig_img_thresh(i,j)= 1;
        else
           orig_img_thresh(i,j)= 0;
        end
    end
end


figure;
imagesc(orig_img_ch1),colormap('gray');
title('Original Image');

figure;
imagesc(orig_img_thresh),colormap('gray');
```

```
title('Thresholded Original Image');

% Remove the white circles inside: morphologically close and open the
image to fill
% any holes
se_close= strel('disk',5);% a strucutring element of disk type having
radius of 5 pixels.
se_open= strel('disk',3);% a strucutring element of disk type having
radius of 3 pixels.

orig_img_thresh_close= imclose(orig_img_thresh,se_close);
figure;                                                  %closing the image
imagesc(orig_img_thresh_close),colormap('gray');
title('closed image');

orig_img_thresh_open= imopen(orig_img_thresh_close,se_open);
figure;                                                  %opening the image
imagesc(orig_img_thresh_open),colormap('gray');
title('opened image');

%Invert the image to get cell as white portion
orig_img_thresh_inv = 1-orig_img_thresh_open;

figure;
imagesc(orig_img_thresh_inv),colormap('gray');
title('Thresholded and Inverted Original Image');

%Edge Detect the image
orig_img_thresh_inv_edge = edge(orig_img_thresh_inv,'sobel');

%Rotate the image
%====================
%Rotation is one more way to add variation in the original image.
%If rotation is not desired pass 0 in the imrotate function
orig_img_thresh_inv_edge_rotate= imrotate(orig_img_thresh_inv_edge,0);
figure;
subplot(1,2,1)
imagesc(orig_img_thresh_inv_edge),colormap('gray');
title('Edge Detected Image');
subplot(1,2,2)
imagesc(orig_img_thresh_inv_edge_rotate),colormap('gray');
title('Rotated Edge Detected Image');

%Invert the image
orig_img_thresh_inv_edge_rotate_inv = 1-
orig_img_thresh_inv_edge_rotate;
figure;
imagesc(orig_img_thresh_inv_edge_rotate_inv),colormap('gray');
title('Mask for Good cell');

%Frequency domain noise addition
%====================================

%Finding the coordinates of the edge points.

num_of_edge_points = sum(sum(orig_img_thresh_inv_edge_rotate));
```

```
x_coord = zeros(num_of_edge_points,1);
y_coord = zeros(num_of_edge_points,1);
num_count = 1;
[Ix,Iy]=size(orig_img_thresh_inv_edge_rotate);
for i=1:Ix
    for j = 1:Iy
        if(orig_img_thresh_inv_edge_rotate(i,j)==1)
            x_coord(num_count,1)=j;
            y_coord(num_count,1)=i;
            num_count = num_count+1;
        end
    end
end

xy_coord = [x_coord,y_coord];
%These are fixed for the synthesizer but would be found out in the
hough
%transform
center_x=87;
center_y=79;
center_x=round(mean(x_coord));
center_y=round(mean(y_coord));

%Finding distance of the edge points from the center of the circle
distance = zeros(num_of_edge_points,1);

for i= 1:num_of_edge_points
    dist = sqrt((x_coord(i)- center_x).^2 + (y_coord(i) -
center_y).^2);
    distance(i,1) = (dist);
end


%Generating white noise of same size as boundary.
%=====================================================
%defining theta
%finding theta for corresponding point
theta = zeros(num_of_edge_points,1);
for i= 1:num_of_edge_points
    dx = x_coord(i,1)-center_x;%finding dx
    dy = y_coord(i,1)-center_y;%finding dy
    theta(i,1)= atan2(dy,dx);
end

figure;
[Ylgh,Ilgh]=sort(theta,1,'ascend');
distlgh(1:num_of_edge_points,1)=distance(Ilgh);
% Ylgh and distlgh are in order from -pi to +pi but they are not
uniformly
% distributed along the angular rotation
% One way is to map to a vector using theta as a estimate of
% index=1+(round)(theta+pi)*(num_edge_points-1)/(2*pi);
% this will leave some holes. So you need to detect the holes and
linearly
% interpolate across them. Given dleft, indexleft and dright,
indexright
% dleft=indexleft*a+b, dright=indexright*a+b so
```

```
% a=(dright-dleft)/(indexright-indexleft) and b=dright-indexright*a;
plot(distlgh);
title('distance');

%Removing the dc component
mean_dist   = mean(distlgh);
distance    = distlgh-mean_dist;
figure;
plot(distance);
title('distance zero mean');

% taking fft of distance.
distance_fft=fft(distance);
figure;
plot(distance_fft);
title('fft of distance');

%Taking Abs of the fft
%This part is commented cause it is giving wrong results.
%=======================================================
abs_distance_fft=abs(distance_fft);
% normalize
abs_distance_fft=abs_distance_fft/sqrt(abs_distance_fft'*abs_distance_f
ft);
% abs_distance_fft=distance_fft;
figure;
plot(abs_distance_fft);
title('FFT of distance');

%Multiplying distance with white noise
% %=====================================
%Generating the white noise
white_noise=randn(num_of_edge_points,1);

%Scaling the white noise to get variance on output signal
scale = 3.*var(distlgh);

distance_white= (abs_distance_fft.*fft(scale*white_noise));

%Taking the ifft and only the real part
distance_ifft=real(ifft(distance_white));
figure;
plot(distance_ifft);
title('distance after adding noise');

%Adding back the dc component
distance_ifft= distance_ifft + mean_dist;
figure;
plot(distance_ifft);
title('noisy distance');

%Mapping back the X and Y coordinates
%====================================
Xc= zeros(num_of_edge_points,1);
Yc= zeros(num_of_edge_points,1);
%getting the coordinates
 for i = 1: num_of_edge_points
```

```
     [Xc(i),Yc(i)]= pol2cart(Ylgh(i),distance_ifft(i));
 end

%plotting back
%====================================
mask=zeros(Nx,My);
for i= 1:num_of_edge_points
    m=(round(Xc(i))+center_x);
    n=round(Yc(i))+center_y;
    mask(n,m)=1;
end

figure;
imagesc(mask),colormap('gray');
title('mask');

%filling up the holes
%================================
se=strel('disk',4);
mask_close=imclose(mask,se);
figure;
imagesc(mask_close),colormap('gray');
title('closed mask');

%Writing the files
%==================
mask_logical = logical(mask_close);
imwrite((1-mask_logical),'GoodCells/GoodCell.bmp','bmp');
```

A.1.2.Synthesizer for Bad Spheroid

```
%  Priyanka Chaudhary
%  Synthesizer to generate data for processing through the cell
detection
%  algorithm.
%  University of Kentucky Spring 2010
%  The image is thresholded to get only the cell
%  The image is inverted to get cell as a white portion
%  White noise is generated and band pass filtered
%  The FFTs of the image and the noise are added together, this creates
the
%  mask.
%  Add color to the white part of the noise to create texture.
%  taking fft of distance versus theta plot.

clc;
clear all;
close all;

%Parameters
threshold = 150; % for thresholding the image
center_x  = 171;
center_y  = 204;

%Read Bad Cell Image
```

```
orig_image = imread('Picture2.bmp');

%Take one channel
orig_img_ch1 = orig_image(:,:,1);
[Nx My] = size(orig_img_ch1);
orig_img_thresh = zeros(Nx,My);

% Threshold the image
for i = 1:Nx
    for j = 1:My
        if(orig_img_ch1(i,j)>threshold)
           orig_img_thresh(i,j)= 1;
        else
           orig_img_thresh(i,j)= 0;
        end
    end
end


figure;
imagesc(orig_img_ch1),colormap('gray');
title('Original Image');

figure;
imagesc(orig_img_thresh),colormap('gray');
title('Thresholded Original Image');

% Remove the white circles inside: morphologically close and open the
image to fill
% any holes
se_close= strel('disk',5);% a strucutring element of disk type having
radius of 5 pixels.
se_open= strel('disk',3);% a strucutring element of disk type having
radius of 3 pixels.

orig_img_thresh_close= imclose(orig_img_thresh,se_close);
figure;                                                  %closing the image
imagesc(orig_img_thresh_close),colormap('gray');
title('closed image');

orig_img_thresh_open= imopen(orig_img_thresh_close,se_open);
figure;                                                  %opening the image
imagesc(orig_img_thresh_open),colormap('gray');
title('opened image');

%Invert the image to get cell as white portion
orig_img_thresh_inv = 1-orig_img_thresh_open;

figure;
imagesc(orig_img_thresh_inv),colormap('gray');
title('Thresholded and Inverted Original Image');

% Remove the white circles inside: morphologically close and open the
image to fill
% any holes
se_close= strel('disk',3);% a strucutring element of disk type having
radius of 5 pixels.
```

```
se_open= strel('disk',3);% a strucutring element of disk type having
radius of 3 pixels.

orig_img_thresh_inv_edge_rotate_close=
imclose(orig_img_thresh_inv,se_close);
figure;                                                  %closing the image
imagesc(orig_img_thresh_inv_edge_rotate_close),colormap('gray');
title('closed image');
orig_img_thresh_inv_edge_rotate_open=
imclose(orig_img_thresh_inv_edge_rotate_close,se_open);
figure;
imagesc(orig_img_thresh_inv_edge_rotate_open),colormap('gray');
title('open image');

%Edge Detect the image
orig_img_thresh_inv_edge =
edge(orig_img_thresh_inv_edge_rotate_open,'sobel');
figure;
imagesc(orig_img_thresh_inv_edge),colormap('gray');
title('Edge Detected Image');

%labeling the objects,removing the blobs
L = bwlabel(orig_img_thresh_inv_edge,8);
figure;
imagesc(L);
title('edge detected image with labels');

for i=1:Nx
    for j=1:My
        if(L(i,j)==2)
            L(i,j)=1;
        else
                L(i,j)=0;
        end
    end
end

figure;
imagesc(L),colormap('gray');
title('Blobs removed');

%Rotate the image
%====================
%Rotation is one more way to add variation in the original image.
%If rotation is not desired pass 0 in the imrotate function
orig_img_thresh_inv_edge_rotate= imrotate(L,0);
figure;
subplot(1,2,1)
imagesc(orig_img_thresh_inv_edge),colormap('gray');
title('Edge Detected Image');
subplot(1,2,2)
imagesc(orig_img_thresh_inv_edge_rotate),colormap('gray');
title('Rotated Edge Detected Image');

%Invert the image
orig_img_thresh_inv_edge_rotate_inv = 1-
orig_img_thresh_inv_edge_rotate;
```

```
figure;
imagesc(orig_img_thresh_inv_edge_rotate_inv),colormap('gray');
title('Mask for Good cell');

%Frequency domain noise addition
%===================================

%Finding the coordinates of the edge points.

num_of_edge_points = sum(sum(orig_img_thresh_inv_edge_rotate));

x_coord = zeros(num_of_edge_points,1);
y_coord = zeros(num_of_edge_points,1);
num_count = 1;
[Ix,Iy]=size(orig_img_thresh_inv_edge_rotate);
for i=1:Ix
    for j = 1:Iy
        if(orig_img_thresh_inv_edge_rotate(i,j)==1)
            x_coord(num_count,1)=j;
            y_coord(num_count,1)=i;
            num_count = num_count+1;
        end
    end
end

xy_coord = [x_coord,y_coord];
%These are fixed for the synthesizer but would be found out in the
hough
%transform
center_x=87;
center_y=79;
center_x=round(mean(x_coord));
center_y=round(mean(y_coord));

%Finding distance of the edge points from the center of the circle
distance = zeros(num_of_edge_points,1);

for i= 1:num_of_edge_points
    dist = sqrt((x_coord(i)- center_x).^2 + (y_coord(i) -
center_y).^2);
    distance(i,1) = (dist);
end


%Generating white noise of same size as boundary.
%=====================================================


%defining theta
%finding theta for corresponding point
theta = zeros(num_of_edge_points,1);
for i= 1:num_of_edge_points
    dx = x_coord(i,1)-center_x;%finding dx
    dy = y_coord(i,1)-center_y;%finding dy
    theta(i,1)= atan2(dy,dx);
end
```

```
figure;
[Ylgh,Ilgh]=sort(theta,1,'ascend');
distlgh(1:num_of_edge_points,1)=distance(Ilgh);
% Ylgh and distlgh are in order from -pi to +pi but they are not
uniformly
% distributed along the angular rotation
% One way is to map to a vector using theta as a estimate of
% index=1+(round)(theta+pi)*(num_edge_points-1)/(2*pi);
% this will leave some holes. So you need to detect the holes and
linearly
% interpolate across them. Given dleft, indexleft and dright,
indexright
% dleft=indexleft*a+b, dright=indexright*a+b so
% a=(dright-dleft)/(indexright-indexleft) and b=dright-indexright*a;
plot(distlgh);
title('distance');

%Removing the dc component
mean_dist   = mean(distlgh);
distance    = distlgh-mean_dist;
figure;


plot(distance);
title('distance zero mean');

% taking fft of distance.
distance_fft=fft(distance);
%Create irect
%This is an ideal low pass filter.
H = irect(1,20,1,num_of_edge_points);
%Filter the image
filter_distance = H'.*distance_fft;
distance_lp=real(ifft(filter_distance));
figure;


plot(distance_lp);
title('distance filtered');



%Taking Abs of the fft

%=========================================================
abs_distance_fft=abs(filter_distance);
% normalize
abs_distance_fft=abs_distance_fft/sqrt(abs_distance_fft'*abs_distance_f
ft);
% abs_distance_fft=distance_fft;
figure;
plot(abs_distance_fft);
title('FFT of distance');

%Multiplying distance with white noise
% %=====================================
%Generating the white noise
```

```
white_noise=randn(num_of_edge_points,1);

%Scaling the noise to get variance on the output signal
scale = var(distlgh);

%Add noise
distance_white= (abs_distance_fft.*fft(scale*white_noise));

%Taking the ifft and only the real part
distance_ifft=real(ifft(distance_white));

%Adding back the dc component
distance_ifft= distance_ifft + mean_dist;
figure;
plot(distance_ifft);
title('noisy distance');

%Mapping back the X and Y coordinates
%==================================
Xc= zeros(num_of_edge_points,1);
Yc= zeros(num_of_edge_points,1);
%getting the coordinates
 for i = 1: num_of_edge_points
     [Xc(i),Yc(i)]= pol2cart(Ylgh(i),distance_ifft(i));
 end

%plotting back
%====================================
mask=zeros(Nx,My);
for i= 1:num_of_edge_points
    m=(round(Xc(i))+center_x);
    n=round(Yc(i))+center_y;
    mask(n,m)=1;
end

figure;
imagesc(mask),colormap('gray');
title('mask');

%filling up the holes
%================================
se=strel('disk',3);
mask_close=imclose(mask,se);
figure;
imagesc(mask_close),colormap('gray');
title('closed mask');

%Writing the files
%=================
mask_logical = logical(mask_close);
imwrite(mask_logical,'./BadCells/BadCell.bmp','bmp');
```

A.2 Hough Transform Implementation

```
%  Priyanka Chaudhary
%==============================
%  Script to perform cell detection using Hough transform.
%  University of Kentucky Spring 2010
%  The image is thresholded to get only the cell
%  The image is inverted to get cell as a white portion
%  Hough transform is performed on the image to detect circles of known
%  radii. Then Peak to Edge ratio, MSE are used to decide whether the
%  detected cell is a good cell or a bad cell


%close all;
%clear all;

%Starting timer for total execution time
%tic

%For either turning on/off the figures
DISPLAY_FIGURES = 0;

%To tell whether the data is either the original image of the
synthesised
%data.
ORIGINAL_IMAGE  = 0;

%Total number of files to be read
NUM_OF_FILES    = 26;

%For storing pictures for animation
ANIMATION = 0;
if(NUM_OF_FILES >1)
    ANIMATION = 0;
end

%this array is in the form of a table like this
% Filename  optimum_radii   peak2edge   MSE
FINAL_RESULTS_BAD   = zeros(NUM_OF_FILES,4);

%To denote the file number
index = 0;

%For loop for reading number of files at once.
%Set the NUM_OF_FILES to 1 if you want to just read one file
for FILE_NO = 1:NUM_OF_FILES

    Pathname='.\BadCells'; % path or folder name in reference to your
default path
    Filename='BadCell'; % name of files not including the index or
suffix
    Filesuffix='bmp'; % suffix or image type

Fullname=sprintf('%s%c%s%d%c%s',Pathname,'\',Filename,index,'.',Filesuf
fix);

    %read the image
```

```
    rawimg = imread(Fullname);
    rawimg_1= rawimg(:,:,1);
    [My Nx] = size(rawimg_1);


    %This will be used in the rest of the script
    image_to_use = zeros(My,Nx);

    %Approx radius for Picture1 = 35, Picture2 =25 and Picture4 = 17
    %Use for Good Cell Data
    %radii  = [25,26,27,28,29,30,31,32,33,34,35];
    %Use for Bad Cell Data
    radii  = [55,56,57,58,59,60,61,62,63,64,65];
    [num_of_radii] = size(radii);

    if(ORIGINAL_IMAGE == 1)
        %Approx threshold for Picture1 = 130, Picture2 = 150 and
Picture4 = 230
        threshold = 130;

        rawimg2 = zeros(My,Nx);
        %Threshold
        for i=1:My
            for j = 1:Nx
                if(rawimg_1(i,j)>threshold)
                    rawimg2(i,j)=0;
                else
                    rawimg2(i,j)=1;
                end
            end
        end

        %Close the Image
        se_close = strel('disk',3);
        se_open = strel('disk',5);
        img_clo = imclose(rawimg2,se_close);

        %Edge Detect
        img_edge = edge(img_clo,'sobel');

        %Image to use in the rest of the script
        image_to_use = img_edge;
    else
        image_to_use = rawimg_1;
    end

    %Variables to store the results
    PEAK2EDGE_ARRAY = zeros(1,num_of_radii(2));
    MSE_ARRAY       = zeros(1,num_of_radii(2));

    %Finding the coordinates of the edge points
    num_of_edge_points = sum(sum(image_to_use));
    x_coord = zeros(num_of_edge_points,1);
    y_coord = zeros(num_of_edge_points,1);
    num_count = 1;
    for i=1:My
        for j = 1:Nx
```

```
            if(image_to_use(i,j)==1)
                x_coord(num_count,1)=j;
                y_coord(num_count,1)=i;
                num_count = num_count+1;
            end
        end
    end
    xy_coord = [x_coord,y_coord];


    %hough_processing_time = zeros(num_of_radii,1);

    %Looping through all the radii to get the one with least MSE
    for z = 1:num_of_radii(2)

    %assigning the radius
    radius = radii(z);


    %Starting timer for hough transform execution time
    %tic
    %Hough Transform
    %Accumulator space
    Accumulator = zeros(My,Nx);% same size as image

    for i = 1:num_of_edge_points
        cent_x = x_coord(i,1);
        cent_y = y_coord(i,1);
        %Get the circle coordinates
        [X_VAL,Y_VAL]=GetCircleCoords([cent_x,cent_y],radius);
        %Update the accumulator space
        %Total 360 points are returned from the GetCircleCoords
fucntion.
        %But due to the 'floor' function used many points are being
mapped
        %to the same point. Hence in the accumulator space the peak
might
        %get more votes than the number of edge points.
        [cx cy]=size(X_VAL);

        %Defining variable to take care of repeating points
        j_prev = 0;
        k_prev = 0;
        for l=1:cy
            j= X_VAL(1,l);
            k= Y_VAL(1,l);
            %Taking care of repeating points
            if((j==j_prev)&&(k==k_prev))
                continue
            else
                %Check to make sure indices are not outside the image
                if((j>=1) && (k>=1))
                    if((j<=Nx) && (k<=My))
                        Accumulator(k,j) = Accumulator(k,j)+ 1;
                        j_prev           = j;
                        k_prev           = k;
                    end
```

```
                end
            end
        end
        if(ANIMATION)
            %Storing the Accumulation images

Accum_file_name=sprintf('%s%d%c%s','Animation\accum',i,'.','jpg');
            H1 = surf(Accumulator);
            colormap('default')
            saveas(H1,Accum_file_name,'jpg');

            %Storing the circle

Circle_file_name=sprintf('%s%d%c%s','Animation\circ',i,'.','jpg');
            H2 = imshow(image_to_use);
            colormap('gray'),hold on
            plot(X_VAL,Y_VAL,'b');
            plot(cent_x,cent_y,'r+');
            hold off
            saveas(H2,Circle_file_name,'jpg');
        end

    end


    %Finding the center
    max_peak = max(max(Accumulator));
    [Center_y,Center_x] = find(Accumulator == max_peak);
    %Just Taking one of many peaks if detected in rare case
    [cx cy] = size(Center_y);
    cnt = 1;
    if((cx>1))
        MSE_PK = zeros(cx,1);
        RMSE_PK = zeros(cx,1);
        MSE_WRT_THETA_PK = zeros(cx,1);
        for i = 1:cy
            [MSE_PK(i) RMSE_PK(i) MSE_WRT_THETA_PK(i)] =
mean_square_errors(num_of_edge_points,x_coord,y_coord,Center_x(1,i),Cen
ter_y(1,i),radius);
        end
        min_mse = min(min(MSE_PK));
        cnt = find(MSE_PK == min_mse);
        circen_x = Center_x(cnt(1));
        circen_y = Center_y(cnt(1));
    else
        cnt = 1;
        circen_x = Center_x;
        circen_y = Center_y;
    end
    [X_VAL,Y_VAL]=GetCircleCoords([circen_x,circen_y],radius);
%     hough_processing_time(z,1) = toc

    %Measure 1
    %=============
    %Ratio of max peak value to the num of edge points
    %Total 360 points are returned from the GetCircleCoords fucntion.
    %But due to the 'floor' function used many points are being mapped
```

```
    %to the same point. Hence in the accumulator space the peak might
    %get more votes than the number of edge points.Thus ideal
theoritical
    %peak2edge should be 1, but due to above explanation it might be
more
    %than 1.
    peak2edge = max_peak/num_of_edge_points;

    %Measure 2
    %==========
    % %Root Mean square error
    [MSE RMSE MSE_WRT_THETA] =
mean_square_errors(num_of_edge_points,x_coord,y_coord,circen_x,circen_y
,radius);


    %Store the Values of this iteration
    %===================================
    PEAK2EDGE_ARRAY(z) = peak2edge;
    MSE_ARRAY(z)       = MSE;

    %Measure 3
    %==========
    %Fractal Dimensions
    %Generating a perfect circle
    % perfect_circle = zeros(My,Nx);
    % [py px] = size(X_VAL);
    % for i = 1:px
    %     x = X_VAL(i);
    %     y = Y_VAL(i);
    %     perfect_circle(y,x) = 1;
    % end
    %
    % %Finding Self Similarity of a perfect circle
    % [n r] = boxcount(perfect_circle);
    % figure
    % loglog(r, n,'bo-', r, (r/r(end)).^(-2), 'r--')
    % xlabel('r')
    % ylabel('n(r)')
    % legend('actual box-count','space-filling box-count');
    % title('Perfect Circle');
    %
    % figure
    % boxcount(perfect_circle);
    % figure
    % boxcount(perfect_circle,'slope');
    %
    % %Finding Self Similarity of the cell
    % [n r] = boxcount(img_edge);
    % figure
    % loglog(r, n,'bo-', r, (r/r(end)).^(-2), 'r--')
    % xlabel('r')
    % ylabel('n(r)')
    % legend('actual box-count','space-filling box-count');
    % title('Cell');
    %
```

```
    %Creating a derivative history
%     derivative_image = diff(image_to_use);
%     figure;
%     imagesc(derivative_image);
%     colormap('gray');
%
%     %Taking the Fourier Transform
%     spectral_info = fft(image_to_use);
%     abs_spectral_info = abs(spectral_info);
%     figure;
%     imagesc(abs_spectral_info);
%     colormap('gray');
%
%     %Creating a histogram of the spectral info
%     max_val = ceil(max(max(abs_spectral_info)));
%     min_val = floor(min(min(abs_spectral_info)));
%     range = min_val:1:max_val;
%     figure;
%     hist(abs_spectral_info,range)




    %DISPLAY SECTION
    %==========================

    if(DISPLAY_FIGURES == 1)


        %PREPROCESSING FIGURES
        %======================

        if(ORIGINAL_IMAGE)
            %Read Image
            figure;
            imagesc(rawimg_1),
            colormap('gray');


            %Thresholded Image
            figure;
            imagesc(rawimg2),
            colormap('gray');
            title('Thresholded Image');

            %Closed image
            figure;
            imagesc(img_clo),
            colormap('gray');
            title('Closed Image');

            %Edge Detected Image
            figure;
            imagesc(img_edge),
            colormap('gray');
            title('Edge Detected Image');
```

```
end

    %Final Image being used for transform
    figure;
    imagesc(image_to_use),
    colormap('gray');
    title('Image used for hough transform');

%RESULTS FIGURES
%=========================

%Accumulator Space
figure;
surf(Accumulator);
colormap('gray')
title('Accumulator Space');


if(peak2edge > 0.2)

    %Plotting the results
    figure;
    imagesc(rawimg_1);
    colormap('gray');
    hold on;
    plot(X_VAL,Y_VAL),
    axis square;
    plot(circen_x,circen_y,'r+');
    hold off
    title('Good Cell Found');

    %Plotting the results on edge detected image
    figure;
    imagesc(image_to_use);
    colormap('gray');
    hold on;
    plot(X_VAL,Y_VAL),
    axis square;
    plot(circen_x,circen_y,'r+');
    hold off
    title('Good Cell Found');


    %Plotting the results on accumulator image
    figure;
    imagesc(Accumulator);
    colormap('gray')
    hold on;
    plot(X_VAL,Y_VAL),
    axis square;
    plot(circen_x,circen_y,'r+');
    hold off
    title('Good Cell Found');

else
    %Plotting the results
    figure;
```

```
            imagesc(rawimg_1);
            colormap('gray');
            title('Bad Cell Found');

        end

    end


    end %End of For loop for radii

    %Finding the radii with the lease MSE

    min_mse = min(MSE_ARRAY);
    loc = find(MSE_ARRAY == min_mse);

    FINAL_RESULTS_BAD(FILE_NO,1) = FILE_NO;
    FINAL_RESULTS_BAD(FILE_NO,2) = radii(1,loc);
    FINAL_RESULTS_BAD(FILE_NO,3) = PEAK2EDGE_ARRAY(1,loc);
    FINAL_RESULTS_BAD(FILE_NO,4) = MSE_ARRAY(1,loc);

    %Read the next file
    index = index + 1;
end %End of For loop for reading files

%Display the results
FINAL_RESULTS_BAD

%Plotting the results
x = 1:1:NUM_OF_FILES;

figure;
plot(x,FINAL_RESULTS_BAD(:,2),'r+');
xlabel('Image Number');
ylabel('Radius');
title('Optimum Radius');
axis([1,26,0,70]);

figure;
plot(x,FINAL_RESULTS_BAD(:,3),'bd');
xlabel('Image Number');
ylabel('Peak to Edge Ratio');
title('Peak to Edge Ratio');
axis([1,26,0,1.5]);

figure;
plot(x,FINAL_RESULTS_BAD(:,4),'-');
xlabel('Image Number');
ylabel('Mean Square Error');
title('MSE');

%total_processing_time = toc

%Average Hough Processing time
%mean(hough_processing_time)
```

A.3 DIDO Schematic

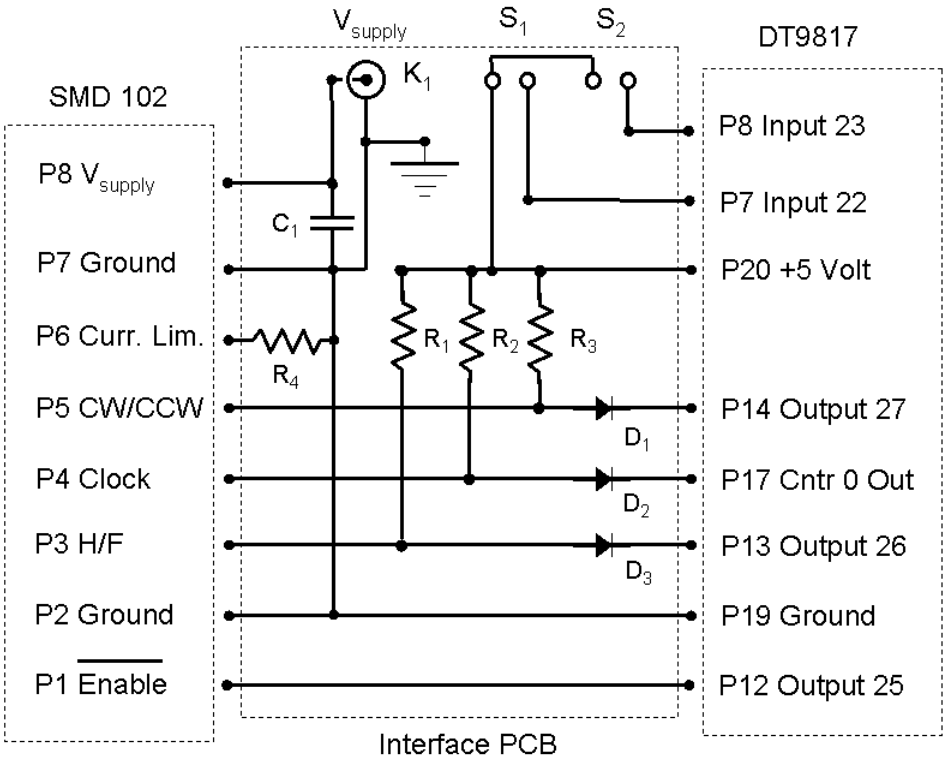


Figure A-1: Interface Printed Circuit Board schematic. Interfaces Stepper Driver, Limit Switches and DI/DO device.

Table A.1: Parts list for Interface PCB

Part Number	Part Specification
K1	
S1	Limit Switch N.C.
S2	Limit Switch N.C.
R1	510 Ω
R2	510 Ω
R3	510 Ω
R4	

C1	1000 µf 50V
D1	1N916
D2	1N916
D3	1N916

A.4 Motor and Camera Control Code

```
// UK_HTSDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UK_HTS.h"
#include "UK_HTSDlg.h"
#include <fstream>
#include <windowsx.h>
#include <afxwin.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <sys/timeb.h>
#include <math.h>
#include <afxstr.h>
#include <atlimage.h>
#include <gdiplus.h>
#include <Gdiplusimaging.h>

#include <assert.h>                    /* for asertion testing */
#include <memory.h>                    /* for _fmemset() */
#include <ctype.h>                     /* for ascii test macros */
#include <mmsystem.h> // sound classes
#include "..\CORE\CoreStructures.h"
#include "..\CORE\display.h"
#include "..\CORE\CORE1\fftlib1.h"
#include "..\CORE\fileioall.h"
#include "..\CORE\fileiobmp.h"
#include "..\CORE\fileioparam.h"
#include "..\CORE\fileiomat5.h"
#include "..\CORE\filter.h"
#include "..\CORE\CORE1\filter1.h"
#include "..\CORE\funcbyte.h"
#include "..\CORE\funccmplx.h"
#include "..\CORE\funcflt.h"
#include "..\CORE\funcrgb.h"
#include "..\CORE\CORE1\CORE2\funcrgb2.h"
#include "..\CORE\funcmat5.h"
#include "..\CORE\funcsnake.h"
#include "..\CORE\CORE1\imbeddedgraphics.h"
```

```
#include "..\CORE\icon5d\imbed_icon5d_0.h"
#include "..\CORE\jpeg.h"
#include "..\CORE\CORE1\Minv1.h"
#include "..\CORE\matrixMxNflt.h"
#include "..\CORE\matrix2x2.h"
#include "..\CORE\matrix3x3.h"
#include "..\CORE\matrix4x4.h"
#include "..\CORE\CORE1\openhb1.h"
#include "..\CORE\CORE1\CORE2\funcmat5_2.h"
#include "..\CORE\pathio.h"
#include "..\CORE\slpconvert.h"
#include "..\CORE\sort.h"
#include "..\CORE\MSdeprecated.h"
#include "..\..\CORE\OEM\mvBlueFOX.h"



#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CAboutDlg dialog used for App About
// global variables
short glbprocess,glbtimerflag;
float glbxstrt, glbystrt,glbzstrt,glbxsteps,glbysteps;
float glboutxstrt,glboutystrt;
float glblen,glbwdth,Positions[3];
short  glbrows,glbcol,row_num,col_num;
bool thrdflag,snap,thrd2timer,left_to_right;
void timeclock(double *timenow);
bool Scan_done,Pic_taken,camdiag,end_scan;

// Threads added by Priyanka
DWORD WINAPI Runthread(LPVOID dummy);
DWORD WINAPI Camthread(LPVOID dummy);
//variables
short glbi,glbj,index_x;
float glbactual_pos;
static HANDLE glbhrunthread;
static HANDLE glbhcamthread;
short glbrunthreadactive=0,glbrunthreadon=0,glbrunthreadread=0;
short glbcamthreadactive=0,glbcamthreadon=0,glbcamthreadread=0;
DWORD glbdwrunthread;
DWORD glbdwcamthread;
LPCONFIG lpconfig;
LPMOTORPARA lpmotorpara;
//classes
CSetupDlg m_SetupDlg;
MvCamera Mv;
class CAboutDlg : public CDialog
{
public:
      CAboutDlg();

// Dialog Data
      enum { IDD = IDD_ABOUTBOX };
```

```
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support

// Implementation
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()


// CUK_HTSDlg dialog



CUK_HTSDlg::CUK_HTSDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CUK_HTSDlg::IDD, pParent)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CUK_HTSDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CUK_HTSDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
	ON_COMMAND(ID_RUN_GO, &CUK_HTSDlg::OnRunGo)
	ON_COMMAND(ID_STOP_STOP, &CUK_HTSDlg::OnStopStop)
	ON_COMMAND(ID_EXIT_EXIT, &CUK_HTSDlg::OnExitExit)
	ON_WM_CLOSE()
	ON_WM_SIZE()
	ON_WM_TIMER()
	//ON_COMMAND(ID_FILE_OPEN32771, &CUK_HTSDlg::OnFileOpen32771)
	ON_COMMAND(ID_HOME_HOME, &CUK_HTSDlg::OnHomeHome)
	ON_COMMAND(ID_SETUP_SETUPPLATTER,
&CUK_HTSDlg::OnSetupSetupplatter)
	ON_COMMAND(ID_SETUP_SETUPMACHINE,
&CUK_HTSDlg::OnSetupSetupmachine)
```

```
      ON_COMMAND(ID_DIAGNOSTICS_CAMERA,
&CUK_HTSDlg::OnDiagnosticsCamera)
      ON_COMMAND(ID_VIEW_DISPLAYSETTINGS,
&CUK_HTSDlg::OnViewDisplaysettings)
      ON_COMMAND(ID_DIAGNOSTICS_MOTORS,
&CUK_HTSDlg::OnDiagnosticsMotors)
      ON_COMMAND(ID_SCREENINGPROCESS_INPUTDISH,
&CUK_HTSDlg::OnScreeningprocessInputdish)
      ON_COMMAND(ID_SCREENINGPROCESS_OUTPUTDISH,
&CUK_HTSDlg::OnScreeningprocessOutputdish)
END_MESSAGE_MAP()


// CUK_HTSDlg message handlers

BOOL CUK_HTSDlg::OnInitDialog()
{ CDialog::OnInitDialog();
      GetCurrentDirectory(512,glbpath);
CUK_HTSDlg::Update(0,glbpath);

      UpdateData(FALSE);
      lpconfig = new CONFIG;
      lpmotorpara = new MOTORPARA;

      InitialECON(lpconfig);
      InitialPositionDistancePara(lpmotorpara);



      // Add "About..." menu item to system menu.

      // IDM_ABOUTBOX must be in the system command range.
      ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
      ASSERT(IDM_ABOUTBOX < 0xF000);

      CMenu* pSysMenu = GetSystemMenu(FALSE);
      if (pSysMenu != NULL)
      {
            CString strAboutMenu;
            strAboutMenu.LoadString(IDS_ABOUTBOX);
            if (!strAboutMenu.IsEmpty())
            {
                  pSysMenu->AppendMenu(MF_SEPARATOR);
                  pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
            }
      }

      // Set the icon for this dialog.  The framework does this
automatically
      //  when the application's main window is not a dialog
      SetIcon(m_hIcon, TRUE);             // Set big icon
      SetIcon(m_hIcon, FALSE);            // Set small icon
      // TODO: Add extra initialization here
      // get default path

      UpdateData(FALSE);
```

```
        glbtimeron=0;
            glbtimerflag=0;
            glbprocess=0;
            glballocate=0;
            glbbuttonx=0;
            glbbuttony=20;
        thrdflag= false;
            thrd2timer= false;
        row_num=col_num=1;
        glbi=1;
        glbj=0;
            Scan_done=Pic_taken=false;
            left_to_right=true;
            end_scan=false;
            camdiag=false;
            index_x=1;
      return TRUE;  // return TRUE  unless you set the focus to a
control
}

void CUK_HTSDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
      if ((nID & 0xFFF0) == IDM_ABOUTBOX)
      {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
      }
      else
      {
            CDialog::OnSysCommand(nID, lParam);
      }
}

// If you add a minimize button to your dialog, you will need the code
below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.

void CUK_HTSDlg::OnPaint()
{
      if (IsIconic())
      {
            CPaintDC dc(this); // device context for painting

            SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

            // Center icon in client rectangle
            int cxIcon = GetSystemMetrics(SM_CXICON);
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;

            // Draw the icon
```

```
			dc.DrawIcon(x, y, m_hIcon);
		}
		else
		{
			CDialog::OnPaint();
		}
}

// The system calls this function to obtain the cursor to display while
the user drags
//  the minimized window.
HCURSOR CUK_HTSDlg::OnQueryDragIcon()
{
		return static_cast<HCURSOR>(m_hIcon);
}


void CUK_HTSDlg::OnRunGo()
{ UpdateData(TRUE);
CClientDC dc(this);
int ldummy=1;

  // start Run thread

			if(glbrunthreadactive==0)
			{ glbrunthreadactive=1;

		glbhrunthread=CreateThread(NULL,0,Runthread,(LPDWORD)ldummy,(DWOR
D)0,&glbdwrunthread);
			}

		// call Runthread
AfxBeginThread((AFX_THREADPROC)Runthread,
			(LPVOID)this,
			0,
			0,
			0,
			NULL );

//Initialize scan
CUK_HTSDlg::Init_Scan();

//open camera
    short iresult;
      BOOL result;
      iresult=0;
      CUK_HTSDlg::OnStopStop();
      //glbprocess=1;
      Invalidate();
      Mv.pDev=Mv.MvCamera_open(Mv.devMgr); //open the camera
      result=Mv.MvCamera_getSize(Mv.devMgr,Mv.pDev,&glbNx1,&glbMy1);

      CUK_HTSDlg::allocate();
      if(glbtimerflag==0) {glbtimerflag=1;iresult =
(short)SetTimer(1,1000,NULL);}
      if (iresult == 0) {MessageBox ("cannot install timer!");}
```

```
/* Sleep(100);

//check to see if camera needs to be initialized
  if(thrdflag==true)
   {
         // start Cam thread
            if(glbcamthreadactive==0)
            { glbcamthreadactive=1;

      glbhcamthread=CreateThread(NULL,0,Camthread,(LPDWORD)ldummy,(DWOR
D)0,&glbdwcamthread);
            }

       AfxBeginThread((AFX_THREADPROC)Camthread,
            (LPVOID)this,
            0,
            0,
            0,
            NULL );
   }
  Sleep(200);
  if(thrdflag==false)
  {
   // KILL Cam THREAD

      if(glbcamthreadactive==1)
      { glbcamthreadread=0;
      //wait for thread to exit
        while(glbcamthreadon==1)
        {Sleep(1000);
      MessageBeep((WORD)-1);
         }
     // terminate the thread
       TerminateThread(glbhcamthread,1);
       CloseHandle(glbhcamthread);
       glbcamthreadactive=0;
      }

  }*/
UpdateData(FALSE);

}


void CUK_HTSDlg::OnStopStop()
{
      //kill timer
      if(glbtimerflag==1) {KillTimer(1); glbtimerflag=0;}
      while(glbtimeron==1) {MessageBeep((WORD)-1);}//wait for timer to
exit

      //KILL Run THREAD
      if(glbrunthreadactive==1)
      { glbrunthreadread=0;
      //wait for thread to exit
        while(glbrunthreadon==1)
```

```
        {Sleep(1000);
      MessageBeep((WORD)-1);
         }
     // terminate the thread
       TerminateThread(glbhrunthread,1);
       CloseHandle(glbhrunthread);
       glbrunthreadactive=0;
      }

      UpdateData(TRUE);
       //update variables
      // TODO: Add your command handler code here
      CUK_HTSDlg::Update(1,glbpath);
      CUK_HTSDlg::deallocate();
      glbprocess=0;
}

void CUK_HTSDlg::OnExitExit()
{
      UpdateData(TRUE);
      CUK_HTSDlg::OnStopStop();
      delete lpconfig;
      delete lpmotorpara;
      OnOK();
}

void CUK_HTSDlg::OnClose()
{
      // TODO: Add your message handler code here and/or call default

      CUK_HTSDlg::OnStopStop();
      CDialog::OnClose();
}

void CUK_HTSDlg::OnSize(UINT nType, int cx, int cy)
{
      CDialog::OnSize(nType, cx, cy);
     glbwidth=cx;
       glbheight=cy;
      Invalidate();
}

/*void CUK_HTSDlg::OnFileOpen32771()
{
      m_OpenDlg.DoModal();
}*/

void CUK_HTSDlg::OnHomeHome()
{ UpdateData(TRUE);
      GoHome(lpconfig, lpmotorpara);

}

void CUK_HTSDlg::Update(short iotype,char *path)
{
      char filename[512],stemp[512];
      short iresult;
```

```
      short Nnames,Nbolvalues,Nshtvalues,Nfltvalues;
      CHARSTRING512 filenames[19];
      bool bolvalues[80];
      short shtvalues[80];
      float fltvalues[80];
      short iotypeb4,n;
      strcpy_ansi(filename,path);
      strcat_ansi(filename,"\\UK_HTS.hed");
      iotypeb4=iotype;
RESET:;
      switch(iotype)
      {
      case -1:
            Nnames=20;Nbolvalues=80;Nshtvalues=80;Nfltvalues=80;
            for(n=0;n<Nnames;n++)
strcpy_ansi(filenames[n].name,"blank");
            for(n=0;n<Nbolvalues;n++) bolvalues[n]=FALSE;
            for(n=0;n<Nshtvalues;n++) shtvalues[n]=0;
            for(n=0;n<Nfltvalues;n++) fltvalues[n]=0.0;
            //
            strcpy_ansi(filenames[0].name," ");
            strcpy_ansi(filenames[1].name," ");
            //
            //bolvalues[0]=FALSE;//m_FileInputDlg.m_FileInputRefCheck;
            bolvalues[0]=FALSE;//glbmainaverage;
            // short
            shtvalues[0]=20;//m_SetupDlg.m_in_row;
            shtvalues[1]=20;//m_SetupDlg.m_in_col;
            shtvalues[2]=20;//m_SetupDlg.m_out_row;
          shtvalues[3]=20;//m_SetupDlg.m_out_col;
            shtvalues[4]=20;//glbbuttony;
            // float
            fltvalues[0]=0.0;//m_SetupDlg.m_in_x_strt;
            fltvalues[1]=0.0;//m_SetupDlg.m_in_y_strt;
         fltvalues[2]=3.5;//m_SetupDlg.m_in_width;
         fltvalues[3]=0.0;//m_SetupDlg.m_in_len;
         fltvalues[4]=0.0;//m_SetupDlg.m_in_ht;

        fltvalues[4]=0.0;//m_SetupDlg.m_out_x_strt;
        fltvalues[5]=0.0;//m_SetupDlg.m_out_y_strt;
        fltvalues[6]=0.0;//m_SetupDlg.m_out_width;
        fltvalues[7]=0.0;//m_SetupDlg.m_out_len;
        fltvalues[8]=0.0;//m_SetupDlg.m_out_ht;
        fltvalues[9]=20;//m_SetupmachDlg.m_turnpin;
        fltvalues[10]=400;//m_SetupmachDlg.m_stepperturn;
            //
            iresult=NnameNbolNshtNfltio(1,filename,filenames,&Nnames,

      &bolvalues[0],&Nbolvalues,&shtvalues[0],&Nshtvalues,&fltvalues[0]
,&Nfltvalues);
            if(iresult!=1)
            {
                  strcpy_ansi(stemp,"Not able to write to
");strcat_ansi(stemp,filename);
                  //MessageBox (stemp);
                  goto EXIT;
            }
```

```
            if(iotypeb4!=-1) {iotype=iotypeb4;goto RESET;}
            break;
      case 0:
            iresult=NnameNbolNshtNfltio(0,filename,filenames,&Nnames,

      &bolvalues[0],&Nbolvalues,&shtvalues[0],&Nshtvalues,&fltvalues[0]
,&Nfltvalues);
            if(iresult==0) {iotype=-1;goto RESET;}
            strcpy_ansi(filenames[0].name,"Version.092107"); //
original date: DONT CHANGE
            glbVersionOriginal=filenames[0].name;
            // location for testing version, prior to updating new
version
            strcpy_s(filenames[1].name,"Version.091608"); // update
after editting
            glbVersionNew=filenames[1].name;
            // names
            // bool
            glbmainaverage=bolvalues[0];
            // short
            m_SetupDlg.m_in_row=shtvalues[0];
        m_SetupDlg.m_in_col=shtvalues[1];
          m_SetupDlg.m_out_row=shtvalues[2];
          m_SetupDlg.m_out_col=shtvalues[3];
            glbbuttony=shtvalues[4];
            // float
            m_SetupDlg.m_in_x_strt=fltvalues[0];
          m_SetupDlg.m_in_y_strt=fltvalues[1];
         m_SetupDlg.m_in_width=fltvalues[2];
         m_SetupDlg.m_in_len=fltvalues[3];
         m_SetupDlg.m_in_ht=fltvalues[4];

        m_SetupDlg.m_out_x_strt=fltvalues[4];
        m_SetupDlg.m_out_y_strt=fltvalues[5];
        m_SetupDlg.m_out_width=fltvalues[6];
        m_SetupDlg.m_out_len=fltvalues[7];
        m_SetupDlg.m_out_ht=fltvalues[8];
        m_SetupmachDlg.m_turnpin=fltvalues[9];
        m_SetupmachDlg.m_stepperturn=fltvalues[10];
      break;
      case 1:
            iresult=NnameNbolNshtNfltio(0,filename,filenames,&Nnames,

      &bolvalues[0],&Nbolvalues,&shtvalues[0],&Nshtvalues,&fltvalues[0]
,&Nfltvalues);
            Nnames=20;Nbolvalues=80;Nshtvalues=80;Nfltvalues=80;
            strcpy_s(filenames[0].name,glbVersionOriginal);
            strcpy_s(filenames[1].name,glbVersionNew);
            //bool
            bolvalues[0]=glbmainaverage;
            // short
            shtvalues[0]=m_SetupDlg.m_in_row;
            shtvalues[1]=m_SetupDlg.m_in_col;
            shtvalues[2]=m_SetupDlg.m_out_row;
          shtvalues[3]=m_SetupDlg.m_out_col;
            shtvalues[4]=glbbuttony;
```

```
            // float
            fltvalues[0]=m_SetupDlg.m_in_x_strt;
            fltvalues[1]=m_SetupDlg.m_in_y_strt;
          fltvalues[2]=m_SetupDlg.m_in_width;
          fltvalues[3]=m_SetupDlg.m_in_len;
          fltvalues[4]=m_SetupDlg.m_in_ht;

         fltvalues[4]=m_SetupDlg.m_out_x_strt;
         fltvalues[5]=m_SetupDlg.m_out_y_strt;
         fltvalues[6]=m_SetupDlg.m_out_width;
         fltvalues[7]=m_SetupDlg.m_out_len;
         fltvalues[8]=m_SetupDlg.m_out_ht;
         fltvalues[9]=m_SetupmachDlg.m_turnpin;
         fltvalues[10]=m_SetupmachDlg.m_stepperturn;
            //
            //bolvalues[0]=m_FileInputDlg.m_FileInputRefCheck;

            //
            //shtvalues[0]=m_CropDlg.m_XulEdit;

            //
            //fltvalues[0]=m_TrimFillDlg.m_TrimDzEdit;


            //
            iresult=NnameNbolNshtNfltio(1,filename,filenames,&Nnames,

      &bolvalues[0],&Nbolvalues,&shtvalues[0],&Nshtvalues,&fltvalues[0]
,&Nfltvalues);
            break;
      }
EXIT:;
}
void CUK_HTSDlg::OnSetupSetupplatter()
{m_SetupDlg.DoModal();
      // TODO: Add your command handler code here
}
void CUK_HTSDlg::OnTimer(UINT_PTR nIDEvent)
{
      CClientDC dc(this);
      CFont MyFont;
      CFont* pOldFont=dc.SelectObject(&MyFont);
      static short Ncornerb4=-1;
      short Nx1,My1,Ncol,Nrow;
      float bright;
      BMPPIXEL red,green;
      double timeA,timeB;
      //
      glbtimeron=1;
            //snap=false;
      timeclock(&timeA);
      red=rgbcolor(255,0,0);
      green=rgbcolor(0,255,0);

      if(camdiag==true)//timer called by camera diagnostics
      {
        if((glbwidth<10)||(glbheight<10)) goto SKIP;
```

```
        bright=m_MainDisplayDlg.m_BrightEdit;
      // select process
        //switch(glbprocess)
        //{
            //case 1: // full screen
            // setup display
            glbNcorner=0;
            Nx1=glbNx1;My1=glbMy1;
            // display the camera image so increment the Ncorner
counter
            ++glbNcorner;
//          if(Ncornerb4!=glbNcorner) Invalidate();
            Ncornerb4=glbNcorner;
            //
            if(glbNcorner>0)
rgb2Ndisplay(&glbcornermap[0],&Ncol,&Nrow,glbNcorner,glbbuttonx,glbbutt
ony,glbwidth,glbheight,Nx1,My1,5);
            // grab image
            //if(glbdevice==1)
            //{
                  //
      bimage1=Mv.MvCamera_capture8(Mv.pDev,&Nx1,&My1);
                  Mv.MvCamera_capture8(Mv.pDev,glbbimage1,&Nx1,&My1);
                  byte2bmp(glbbmpimage,glbbimage1,Nx1,My1);

            glbncorner=0;

      bmp2display2(&dc,glbbmpimage,Nx1,My1,glbcornermap[glbncorner].ulx
,glbcornermap[glbncorner].uly,1,1,glbcornermap[glbncorner].Nx,glbcorner
map[glbncorner].My,&glbscalexy1);
            ++glbncorner;
            //break;
         //} // end of switch
      //
SKIP:;
      timeclock(&timeB);
      m_TimeEdit=(long)(timeB-timeA);
      UpdateData(FALSE);
      glbtimeron=0;
      CDialog::OnTimer(nIDEvent);

      }//if ends

      else// if timer was called by Run button
      {    CUK_HTSDlg::MM2Steps();
            if(glbactual_pos>Positions[1]-1 &&
glbactual_pos<=Positions[1])//wait to reach approx. position
           {
                  //if(end_scan==true)
                  //{

                  //}
                  thrdflag=false;
                if((glbwidth<10)||(glbheight<10)) goto SKIP0;
              bright=m_MainDisplayDlg.m_BrightEdit;
      // select process
        //switch(glbprocess)
```

```
        //{
            //case 1: // full screen
            // setup display
                 glbNcorner=0;
                 Nx1=glbNx1;My1=glbMy1;
            // display the camera image so increment the Ncorner
counter
                 ++glbNcorner;
            if(Ncornerb4!=glbNcorner) Invalidate();
                 Ncornerb4=glbNcorner;
            //
                 if(glbNcorner>0)
rgb2Ndisplay(&glbcornermap[0],&Ncol,&Nrow,glbNcorner,glbbuttonx,glbbutt
ony,glbwidth,glbheight,Nx1,My1,5);
            // grab image
            //if(glbdevice==1)
            //{
                  //
      bimage1=Mv.MvCamera_capture8(Mv.pDev,&Nx1,&My1);
                   Mv.MvCamera_capture8(Mv.pDev,glbbimage1,&Nx1,&My1);
                   byte2bmp(glbbmpimage,glbbimage1,Nx1,My1);
             CUK_HTSDlg::SaveImages();//save images

                 glbncorner=0;

bmp2display2(&dc,glbbmpimage,Nx1,My1,glbcornermap[glbncorner].ulx,glbco
rnermap[glbncorner].uly,1,1,glbcornermap[glbncorner].Nx,glbcornermap[gl
bncorner].My,&glbscalexy1);
                 ++glbncorner;
                   // Update position
                   CUK_HTSDlg::Update_Pos_X();
                   if(glbj==glbrows)
                  {
                         GoHome(lpconfig, lpmotorpara);
                     CUK_HTSDlg::OnStopStop();
                   }
                   //thrdflag=true;//flag set to start thread running
                   else
                      {
                         Go(lpconfig, lpmotorpara, Positions);
                       }


            //break;
          //} // end of switch
      //
SKIP0:;
      timeclock(&timeB);
      m_TimeEdit=(long)(timeB-timeA);
      UpdateData(FALSE);
      glbtimeron=0;
      CDialog::OnTimer(nIDEvent);
      }//if ends

   }//else ends
```

```
}



void CUK_HTSDlg::OnSetupSetupmachine()
{   m_SetupmachDlg.DoModal();
      // TODO: Add your command handler code here
}

void CUK_HTSDlg::OnDiagnosticsCamera()
{
      camdiag=true;//to access timer
      //thrd2timer=false;//timer not called by Runthread
      short iresult;
      BOOL result;
      iresult=0;
      CUK_HTSDlg::OnStopStop();
      glbprocess=1;
      Invalidate();
      Mv.pDev=Mv.MvCamera_open(Mv.devMgr); //open the camera
      result=Mv.MvCamera_getSize(Mv.devMgr,Mv.pDev,&glbNx1,&glbMy1);

      CUK_HTSDlg::allocate();
      if(glbtimerflag==0) {glbtimerflag=1;iresult =
(short)SetTimer(1,200,NULL);}
      if (iresult == 0) {MessageBox ("cannot install timer!");}

//SKIP:;
}
void CUK_HTSDlg::allocate()
{
      if(glballocate==1) deallocate();
      if(glballocate==0)
      {
            glbNindex1=(long)glbNx1*(long)glbMy1;
            glbbimage1=new unsigned char [glbNindex1];
            glbbmpimage=new BMPPIXEL [glbNindex1];
            glbbmpimage1=new BMPPIXEL [glbNindex1];
            glbbmpimageF=new BMPPIXEL [glbNindex1];
            glballocate=1;
      }
}
void CUK_HTSDlg::deallocate()
{
      if(glballocate==1)
      {
            delete [glbNindex1] glbbimage1;
            delete [glbNindex1] glbbmpimage;
            delete [glbNindex1] glbbmpimage1;
            delete [glbNindex1] glbbmpimageF;
            glballocate=0;
      }
}
void timeclock(double *timenow)
{
      // delaytime is in milliseconds with precision to milliseconds
```

```
      struct _timeb tstruct;
      double seconds,mseconds;
      // time now
      _ftime64_s(&tstruct);seconds=tstruct.time;mseconds=tstruct.millit
m;*timenow=(double)(1000*seconds+mseconds);
}
void CUK_HTSDlg::OnViewDisplaysettings()
{
      m_MainDisplayDlg.DoModal();
}

void CUK_HTSDlg::OnDiagnosticsMotors()
{
      m_motordlg.DoModal();
}

void CUK_HTSDlg::OnScreeningprocessInputdish()
{ glbprocess=1;
      // TODO: Add your command handler code here
}

void CUK_HTSDlg::OnScreeningprocessOutputdish()
{ glbprocess=2;
      // TODO: Add your command handler code here
}

DWORD WINAPI Runthread(LPVOID dummy)
{  //declaring variables

   glbrunthreadon=1;
   glbrunthreadread=1;


 // Runthread processing
      while(glbrunthreadread==1)
      {
            if(glbrunthreadread==0)
            { goto JUMPOUT;}
            else
            {
            if(thrdflag==true)
            {
                        Go(lpconfig, lpmotorpara, Positions);
                  }

        }//else ends
        JUMPOUT:
            //if(Scan_done)

        glbrunthreadread=0;
    }//while ends

   glbrunthreadon=0;
    return 0;


}
```

```
//CamThread definition
/*DWORD WINAPI Camthread(LPVOID dummy)
{
     CUK_HTSDlg nobj;
      nobj.Capture_image();
return 0;
}

// function to capture images
void CUK_HTSDlg::Capture_image()
{
    CClientDC dc(this);
    short iresult;
      BOOL result;
      iresult=0;
      CUK_HTSDlg::OnStopStop();
      Invalidate();
      Mv.pDev=Mv.MvCamera_open(Mv.devMgr); //open the camera
      result=Mv.MvCamera_getSize(Mv.devMgr,Mv.pDev,&glbNx1,&glbMy1);

      CUK_HTSDlg::allocate();
      if(glbtimerflag==0) {glbtimerflag=1;iresult =
(short)SetTimer(1,1000,NULL);}
      if (iresult == 0) {MessageBox ("cannot install timer!");}

SKIP:;
            UpdateData(FALSE);
        thrdflag=false;


}*/

void CUK_HTSDlg::SaveImages()
{
      char
filename1[512],filename2[512],filename3[512],filereturn[512];
      strcpy_ansi(filename1,"C:\\2009cprog\\UK_HTS_data\\Images\\imgR#"
);
      //if((row_num)<=glbrows)
      row_num=glbj+1;
      col_num=index_x;
      {
            fileindexall(filename1,row_num,filename2);
            strcpy_ansi(filename3,"C#.bmp");
            strcat_ansi(filename2,filename3);
            //if((col_num)<=glbcol)
              {
                    fileindexall(filename2,col_num,filereturn);

bmp2bmp24file(filereturn,glbbmpimage,glbNx1,glbMy1);
            }
            //else
              //{
            //      col_num=1; row_num++;
              //}
      }
```

```
}


void CUK_HTSDlg::Kill_timer()
{
      if(glbtimerflag==1)
      {
            KillTimer(1); glbtimerflag=0;
      }
      while(glbtimeron==1) {MessageBeep((WORD)-1);}//wait for timer to
exit
  UpdateData(FALSE);
}

void CUK_HTSDlg::Init_Scan()
{
  //Getting data from setup platter dialog box
   if (glbprocess==1)//if screening input dish
   {
     glbxstrt  = m_SetupDlg.m_in_x_strt ;
     glbystrt  = m_SetupDlg.m_in_y_strt;
     glblen    =   m_SetupDlg.m_in_len;
     glbwdth   = m_SetupDlg.m_in_width;
     glbcol    = m_SetupDlg.m_in_col;
     glbrows   = m_SetupDlg.m_in_row;
     glbxsteps = glblen/glbcol;
     glbysteps = glbwdth/glbrows;
   }
   else//if screening output dish
   {
     glbxstrt  = m_SetupDlg.m_out_x_strt ;
     glbystrt  = m_SetupDlg.m_out_y_strt;
     glblen    = m_SetupDlg.m_out_len;
     glbwdth   = m_SetupDlg.m_out_width;
     glbcol    = m_SetupDlg.m_out_col;
     glbrows   = m_SetupDlg.m_out_row;
     glbxsteps = glblen/glbcol;
     glbysteps = glbwdth/glbrows;
   }

  //Ready the motors for scanning
   Positions[0] = glbystrt;// Y axis motor updated
   Positions[1] = glbxstrt;// X axis or camera motor updated
   Positions[2] = 15;//fixed height of camera ,but can be made user
defined also.
   Go(lpconfig, lpmotorpara, Positions);// motors go to initial
position defined by user
}

void CUK_HTSDlg::Update_Pos_X()
{
      switch(left_to_right)
      {
          case true:
             { index_x=glbi+1;
           Positions[1]=Positions[1] + glbxsteps;
```

```
                glbi++;

               if(glbi>glbcol)
                {
                     Positions[1]=Positions[1] - glbxsteps;
                     left_to_right=false;//toggle order of screening
                     glbj++;
                     CUK_HTSDlg::Update_Pos_Y();
                     glbi--;
                     index_x=glbi;
              }//if ends
              }//case true ends
             break;
             case false:
              {
                index_x=glbi-1;
                Positions[1]=Positions[1] - glbxsteps;
                glbi--;

               if(glbi==0)
               {
                     Positions[1]=Positions[1] + glbxsteps;
                     left_to_right=true;//toggle order of screening
                     glbj++;
                     CUK_HTSDlg::Update_Pos_Y();
                     glbi++;
                     index_x=glbi;
               }//if ends
              }//case false ends
              break;
      }//switch ends

}

void CUK_HTSDlg::Update_Pos_Y()
{
 Positions[0]= Positions[0] + glbysteps;
}
void CUK_HTSDlg::MM2Steps()
{
      float Stp, Tpi,Ipmm;
      Stp= 400; Ipmm =1/(25.4);
      Tpi=20;
      glbactual_pos= (lpmotorpara-
>currentposition2)*(1/Ipmm)*(1/Stp)*(1/Tpi);

}
```

A.5 3D Feature Tracking Algorithm and Implementation

Problem: Animation for movies (special effects SFX) and video games is a time consuming, largely manual process requiring skilled animators.

Solution: We have preliminary prototype software applications, to satisfy the "Expected Product," that will allow us to conduct surround merging of patches without the use of contact markers. We also have the necessary tracking algorithms to perform feature tracking of an object in motion. These tracked features can then be used to stretch or move a polygonal mesh across time.

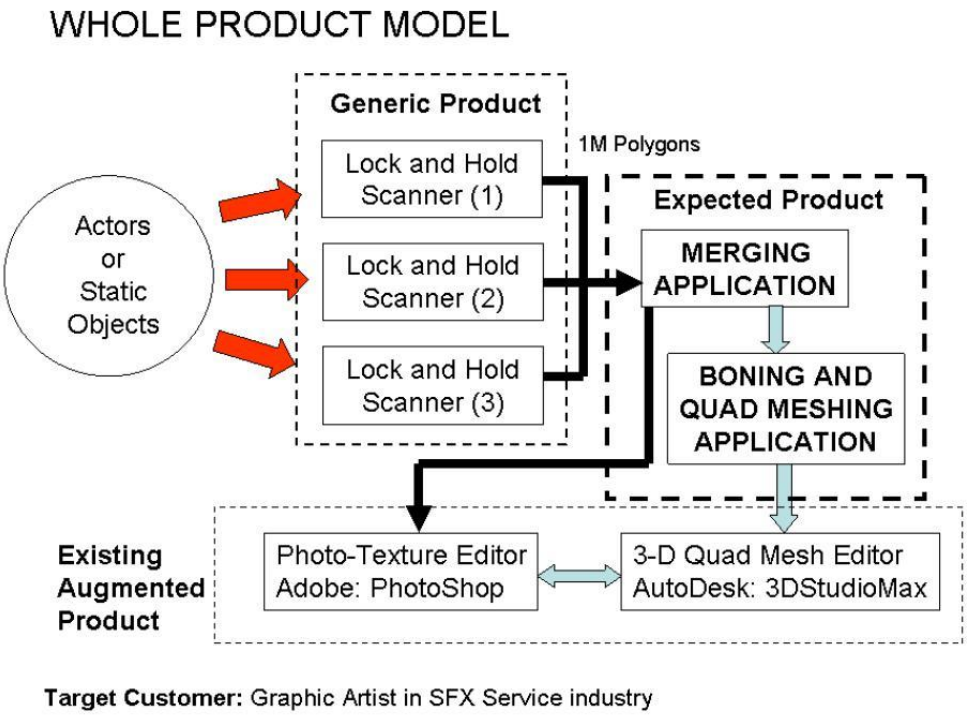


Figure A-3 Product Model

Algorithm:-

1.Read all the C,X,Y,Z,I files into Matlab and store them appropriately.

2.Convert X,Y,Z .byt files from 1D to 2D.

3.Make a rectangular tracker TR with crosshair: Take an all zero rectangular matrix and make its boundary and cross hair edges 1s. Now open Lock mat5 image(C file).

4.Place the tracker on the desired feature (here nose) with the centre of the cross hair on the nose tip. This will be used for correspondence hence called correspondence point.

5.Make a matrix TRn of same size as tracker for each X, Y, Z, C &I and load respective parameters in each for every element of TRn.

6.Note u,v values of the correspondence point . We know its i,j values & X,Y,Z,C,I values.

7.Take next frame or first Hold mat5 image. Place the tracker (multiply C file with tracker) at same u,v as in Lock mat5.

8.Make TR_{n+1} for each X,Y,Z,C & I.

9.Find $\Delta x_t = X_{t,n+1} - X_{t,n}$, same for Y and Z etc.

10.Find $x_{off} = \frac{1}{N}\sum_{t=0}^{N-1} \Delta x_t$, same for Y and Z etc.

11.Add these offsets to corresp. Coordinates i.e. X, Y, and Z for each element of TRn. This aligns TRn in Lock image to new world coordinates.

Steps 9 through 11 give translation correction and can be clubbed as TC.

Rotation correction

12. Now since TRn is at new world coordinates, find its u,v. Then for same u,v but in Hold image find X,Y,Z for all the elements.

13.Repeat 9.

14.Perform affine transformations or rotation about X,Y,Z axes with correspondence point of TRn as centre of rotation. This is given under Affine transformations below. Find min. Euclidean distance $\varepsilon^2 = \sum_{t=0}^{N-1}(\Delta x_t^2 + \Delta y_t^2 + \Delta zt2$. Rotate about X,Y and Z successively and perform TC to converge $\varepsilon 2$.

15.Now make a 3x3 kernel around the correspondence point of TR_{n+1}. Find the min. euc. Distance of each point of this kernel from the correspondence point of TRn. The u,v which give the min. euc. Distance are the true u, v points we were looking for.

Affine transformation:

The following are rotation matrices about origin for X,Y and Z axes respectively in counter clockwise direction.

$$R_x(\theta) = \begin{matrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{matrix}$$

$$R_y(\alpha) = \begin{matrix} cos\alpha & 0 & sin\alpha \\ 0 & 1 & 0 \\ -sin\alpha & 0 & cos\alpha \end{matrix}$$

$$R_z(\emptyset) = \begin{matrix} cos\emptyset & -sin\emptyset & 0 \\ sin\emptyset & cos\emptyset & 0 \\ 0 & 0 & 1 \end{matrix}$$

Combined rotation is given by $R_{00} = R_z . R_y . R_x$

$$= \begin{matrix} cos\emptyset cos\alpha & cos\emptyset sin\theta sin\alpha - sin\emptyset cos\theta & cos\emptyset cos\theta sin\alpha + sin\theta sin\emptyset \\ sin\emptyset cos\alpha & sin\emptyset sin\theta sin\alpha + cos\theta cos\emptyset & sin\alpha sin\emptyset cos\theta - sin\theta cos\emptyset \\ -sin\alpha & sin\theta cos\alpha & cos\theta cos\alpha \end{matrix}$$

The affine transformation about X,Y,Z at an offset from origin is given by

$$\begin{matrix} r_{00} & r_{01} & r_{02} & x - r_{00}x - r_{01}y - r_{02}z \\ r_{10} & r_{11} & r_{12} & y - r_{10}x - r_{11}y - r_{12}z \\ r_{20} & r_{21} & r_{22} & z - r_{20}x - r_{21}y - r_{22}z \\ 0 & 0 & 0 & 1 \end{matrix}$$

Where $r_{i,j}$ are the elements of R_{00}.

Translation correction(performed on 2 consecutive images) Algorithm

Please see appendix IV for code and results

1 .Read all image and .byt files.

2. Generate a rectangular crosshair and place it on the first image.

Crosshair placed on the Lock image

3. Store values of correspondence point.

4. Place tracker at same u,v in the second image and store difference in X, Y,Z as dx, dy,dz.

5.Find the X , Y and Z offsets as given by the formula in the guidelines.

6. Update the tracker on image 1 by adding to it the offsets : *Translation correction.*

7. For optimization: generate 3x3 kernels centered around each point in tracker 1.

   a. Find differences in X,Y,Z between a point in the image 0 and every point in the kernel of its corresponding point.

   b. For every pair find the Euclidean distance ; find the minimum Euclidean distance for a given kernel.

   c. Locate the u,v of the point in every kernel, that gave the min Euclidean distance.

   d. Update the tracker 1 with new coordinates.

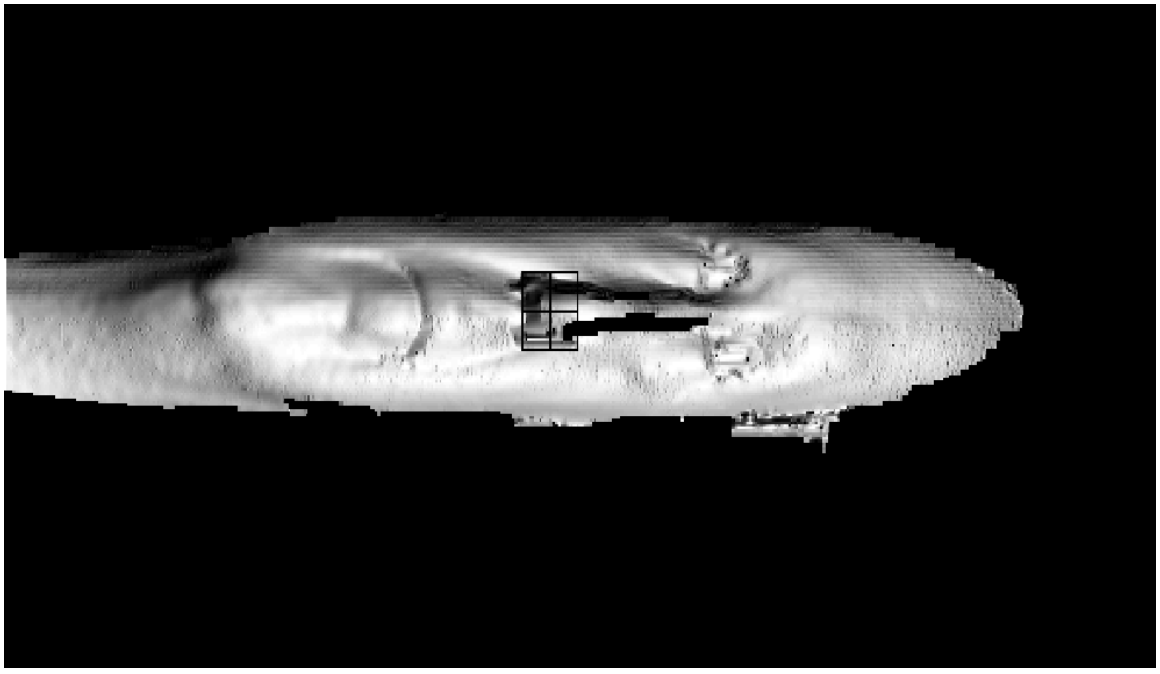


Figure A-2 : CrossHair placed on the lock iamge

Rotation correction Algorithm

1. Take the image 0 and translation corrected image and make several copies of them.

2. Each copy will be processed to assess the angle of rotation about the coordinate axes.

3. A copy is rotated along each axis using 3D projection principle [2]

(by centering image 0 and corrected copy at origin) from +/- 0.25 to +/- 5 degrees in steps of 0.25.

4. On each rotation, the crosshair is placed on the rotated image at same u, v, the corresponding X, Y,Z stored.
5. Now the Euclidean distance between each point in image 0 and the corresponding point on this rotated image is stored.
6. The angle which gives the minimum Euclidean distance is the desired angle to be taken.
7. Once these are found the translation corrected image is simultaneously rotated about all the 3 axes following the Euler order.

Code:

```
%Clear all the memory and close all figures
clear all;
clc;
close all;
% defining a matrix to access files.
%reading any .byt file
fid1= fopen('snk3cnv\4Meglgh112007_0X.byt','rb' );
Img1=fread(fid1,'float32');
fclose(fid1);
[R1,C1,Z1]=size(Img1);
% introducing a matrix to store all X images with same size as the byt file
% with an addition of the third parameter that gives the # of images to be
% read i.e 5(here).
Acc_im1 =zeros(R1,C1,5);
% reading all Xmatrix files
path_name='snk3cnv\4Meglgh112007_'%pathname w.r.t default directory; this
remains constant
while reading all files.
for i= 0:4 % reading 5 files.
fullname1= sprintf('%s%d%c%c%s' ,path_name,i,'X','.','byt' );
Fullname1=fopen(fullname1,'rb' );
img1=fread(Fullname1,'float32');
fclose(Fullname1);
Acc_im1(:,:,[i+1])=img1(:,:); % storing all X files in the above created
matrix.
end
%=====================================================================
%Reading images of Y-files:
% proceeding same as above.
% defining a matrix to access files.
fid2= fopen('snk3cnv\4Meglgh112007_0Y.byt','rb' );
Img2=fread(fid2,'float32');
fclose(fid2);
[R2,C2,Z2]=size(Img2);
Acc_im2 =zeros(R2,C2,5);
% reading all Ymatrix images
for i= 0:4
fullname2= sprintf('%s%d%c%c%s' ,path_name,i,'Y','.','byt' );
Fullname2=fopen(fullname2,'rb' );
img2=fread(Fullname2,'float32');
fclose(Fullname2);
Image2= ['Image#',int2str(i), '.byt'];
Acc_im2(:,:,[i+1])=img2(:,:);
%imwrite(img,'Image','BYT');
figure;
12/22/08 7:44 PM C:\MATLAB7\work\Kernel_for_each_point.m 2 of 11
imagesc(Acc_im2(:,:,(i+1)));
colormap('GRAY' )
end
%=====================================================================
%Reading images of Z-files:
% defining a matrix to access images
```

```
fid3= fopen('snk3cnv\4Meglgh112007_0Z.byt','rb' );
Img3=fread(fid3,'float32');
fclose(fid3);
[R3,C3,Z3]=size(Img3);
Acc_im3 =zeros(R3,C3,5);
% reading all Zmatrix images
for i= 0:4
fullname3= sprintf('%s%d%c%c%s' ,path_name,i,'Z','.','byt' );
Fullname3=fopen(fullname3,'rb' );
img3=fread(Fullname3,'float32');
fclose(Fullname3);
Image3= ['Image#',int2str(i), '.byt'];
Acc_im3(:,:,[i+1])=img3(:,:);
%imwrite(img,'Image','BYT');
figure;
imagesc(Acc_im3(:,:,(i+1)));
colormap('GRAY' )
end
%=========================================================================
%Reading images of C-file:
% defining a matrix to access images
Img4= double(imread('snk3cnv\4Meglgh112007_0C.bmp'));
[R4,C4,Z4]=size(Img4);
Acc_im4 =zeros(R4,C4,5);
% reading all Cmatrix images
for i= 0:4
Fullname4= sprintf('%s%d%c%c%s' ,path_name,i,'C','.','bmp' );
img4=double(imread(Fullname4));
Image4= ['Image#',int2str(i), '.bmp'];
%Storing all the three channels to retain the color information
Acc_im4(:,:,[i+1])=img4(:,:,1);
Acc_im4_ch2(:,:,[i+1]) = img4(:,:,2);
Acc_im4_ch3(:,:,[i+1]) = img4(:,:,3);
%imwrite(img4,'Image4','BMP');
figure;
imagesc(Acc_im4(:,:,(i+1)));
colormap('GRAY' )
end
%=========================================================================
%Reading images of I-file:
% defining a matrix to access images
Img5= double(imread('snk3cnv\4Meglgh112007_0I.bmp'));
[R5,C5,Z5]=size(Img5);
Acc_im5 =zeros(R5,C5,5);
% reading all Imatrix images
for i= 0:4
Fullname5= sprintf('%s%d%c%c%s' ,path_name,i,'I','.','bmp' );
img5=double(imread(Fullname5));
Image5= ['Image#',int2str(i), '.bmp'];
Acc_im5(:,:,[i+1])=img5(:,:,1);
%imwrite(img5,'Image5','BMP');
figure;
imagesc(Acc_im5(:,:,(i+1)));
colormap('GRAY' )
end
%=========================================================================
%Creating 2D from 1D for X Y Z matrices:
%The third index represents the number of images being read.
%right now its 5
Mat_X_2D= zeros(R4,C4,5);
Mat_Y_2D= zeros(R4,C4,5);
Mat_Z_2D= zeros(R4,C4,5);
for k= 1:5
for j= 1:C4
for i= 1:R4
index = i*C4+j;
if(index>C4*R4)
index=C4*R4;
```

```
end
Mat_X_2D(i,j,k)=Acc_im1(index,1,k);%((i-1)*C4)+j);
Mat_Y_2D(i,j,k)=Acc_im2(index,1,k);%((i-1)*C4)+j);
Mat_Z_2D(i,j,k)=Acc_im3(index,1,k);%((i-1)*C4)+j);
end
end
end
close all;
%=====================================================================
% Tracker rectangle on first image of C-file:
%crosshair 31x29 matrix %
tr_col=29; %dimensions of the crosshair
tr_row=31;
crosshair=ones(tr_row,tr_col);
crosshair(1,:)=0;
crosshair(:,1)=0;
crosshair(31,:)=0;
crosshair(:,29)=0;
crosshair(16,:)=0;
12/22/08 7:44 PM C:\MATLAB7\work\Kernel_for_each_point.m 4 of 11
crosshair(:,15)=0;
%imshow(crosshair);
%reading the image%
img=double(imread('C:\MATLAB7\work\snk3cnv\4Meglgh112007_0C.bmp' ));
%placing the crosshair at the nose%
img(104:134,265:293,1)=img(104:134,265:293,1).*crosshair;
figure;
imagesc(img(:,:,1))
colormap('GRAY' )
%=====================================================================
corr_pt_r= 120; corr_pt_c=280;%Correspondence Point
%Tracker 0
TR0_C= Acc_im4(104:134,265:293,1);
TR0_I= Acc_im5(104:134,265:293,1);
TR0_X_2D= Mat_X_2D(104:134,265:293,1);
TR0_Y_2D= Mat_Y_2D(104:134,265:293,1);
TR0_Z_2D= Mat_Z_2D(104:134,265:293,1);
corr_pt_x= TR0_X_2D(16,15);
corr_pt_y= TR0_Y_2D(16,15);
corr_pt_z= TR0_Z_2D(16,15);
%Creating 1D from 2D for X Y Z matrices:
TR0_X_1D= zeros(1,tr_col*tr_row);
TR0_Y_1D= zeros(1,tr_col*tr_row);
TR0_Z_1D= zeros(1,tr_col*tr_row);
for j= 1:tr_col
for i= 1:tr_row
index=i*tr_col+j;
if(index>tr_col*tr_row)
index=tr_row*tr_col;
end
TR0_X_1D(index)=TR0_X_2D(i,j);%((i-1)*tr_col)+j)=TR0_X_2D(i,j);
TR0_Y_1D(index)=TR0_Y_2D(i,j);%((i-1)*tr_col)+j)=TR0_Y_2D(i,j);
TR0_Z_1D(index)=TR0_Z_2D(i,j);%((i-1)*tr_col)+j)=TR0_Z_2D(i,j);
end
end
%Writing the MAT5 format for Tracker 0
mat5file='Tracker0';
tr0_c(:,:,1)= uint8(TR0_C);
tr0_c(:,:,2)= uint8(Img4(104:134,265:293,2));
tr0_c(:,:,3)= uint8(Img4(104:134,265:293,3));
tr0_i(:,:,1)= uint8(TR0_I);
tr0_i(:,:,2)= uint8(Img5(104:134,265:293,2));
tr0_i(:,:,3)= uint8(Img5(104:134,265:293,3));
result=mat5write(mat5file,tr0_c,tr0_i,TR0_X_1D,TR0_Y_1D,TR0_Z_1D);
%=====================================================================
%Processing image 2 i.e. with index 1
temp_img_c=Acc_im4;
temp_img_c(104:134,265:293,2)=Acc_im4(104:134,265:293,2).*crosshair;
```

```
delta_x= zeros(tr_row,tr_col);
%Taking the difference between the X coordinates between TR0 and TR1
delta_x= Mat_X_2D(104:134,265:293,2)-TR0_X_2D;
%X Offset
X_off= mean(mean(delta_x));
%Taking the difference between the Y coordinates between TR0 and TR1
delta_y= zeros(tr_row,tr_col);
delta_y= Mat_Y_2D(104:134,265:293,2)-TR0_Y_2D;
%Y Offset
Y_off= mean(mean(delta_y));
%Taking the difference between the Z coordinates between TR0 and TR1
delta_z= zeros(tr_row,tr_col);
delta_z= Mat_Z_2D(104:134,265:293,2)-TR0_Z_2D;
%Z Offset
Z_off= mean(mean(delta_z));
%Tracker 1
TR1_C= Acc_im4(104:134,265:293,2);
TR1_I= Acc_im5(104:134,265:293,2);
TR1_X_2D= Mat_X_2D(104:134,265:293,2);
TR1_Y_2D= Mat_Y_2D(104:134,265:293,2);
TR1_Z_2D= Mat_Z_2D(104:134,265:293,2);
%Tracker 1 for making kernels
TR1_X_2D_kernel= Mat_X_2D(103:135,264:294,2);
TR1_Y_2D_kernel= Mat_Y_2D(103:135,264:294,2);
TR1_Z_2D_kernel= Mat_Z_2D(103:135,264:294,2);
TR1_C_k= Acc_im4(103:135,264:294,2);
TR1_I_k= Acc_im5(103:135,264:294,2);
%Tracker 1 with updated world coordinates
TR1_X_2D_up= TR1_X_2D + X_off;
TR1_Y_2D_up= TR1_Y_2D + Y_off;
TR1_Z_2D_up= TR1_Z_2D + Z_off;
%Tracker 1 for kernels updated
TR1_X_2D_kernel_up= TR1_X_2D_kernel + X_off;
TR1_Y_2D_kernel_up= TR1_Y_2D_kernel + Y_off;
TR1_Z_2D_kernel_up= TR1_Z_2D_kernel + Z_off;
% Their row and column size
row_k=33; col_k=31;
%=========================================================================
%Corresspondence point World coordinates of the updated Tracker 1
%corr_pt_new_x= TR1_X_2D_up(16,15);
%corr_pt_new_y= TR1_Y_2D_up(16,15);
%corr_pt_new_z= TR1_Z_2D_up(16,15);
%Defining a 3X3 matrix for searching the next correspondence point
%(u,v) in the C image
%kernel_x= TR1_X_2D_up(15:17,14:16);
%kernel_y= TR1_Y_2D_up(15:17,14:16);
%kernel_z= TR1_Z_2D_up(15:17,14:16);
%Kernels for each point in tracker1
points= tr_row*tr_col;
kernel_x=zeros(3,3,points);
kernel_y=zeros(3,3,points);
kernel_z=zeros(3,3,points);
[M,N,Z]= size(kernel_x);
for k= 1 :points
for i= 1: tr_row
for j= 1: tr_col
if(i==1)
row_beg_k=1; row_end_k= 3;
elseif(i==tr_row)
row_end_k= row_k; row_beg_k= 31;
else
row_beg_k=i-1; row_end_k= i+1;
end
if(j==1)
col_beg_k=1; col_end_k= 3;
elseif(j==tr_col)
col_beg_k= 29; col_end_k= col_k;
else col_beg_k=j-1; col_end_k= j+1;
```

```
end
kernel_x(1:M,1:N,k)=
TR1_X_2D_kernel_up(row_beg_k:row_end_k,col_beg_k:col_end_k);
kernel_y(1:M,1:N,k)=
TR1_Y_2D_kernel_up(row_beg_k:row_end_k,col_beg_k:col_end_k);
kernel_z(1:M,1:N,k)=
TR1_Z_2D_kernel_up(row_beg_k:row_end_k,col_beg_k:col_end_k);
end
end
end
%Finding the Euclidean distance
%diff_x= zeros(3,3);
%diff_y= zeros(3,3);
%diff_z= zeros(3,3);
%for j=1:3;
% for i=1:3
% diff_x(i,j)= corr_pt_x - kernel_x(i,j);
% diff_y(i,j)= corr_pt_y - kernel_y(i,j);
% diff_z(i,j)= corr_pt_z - kernel_z(i,j);
% end
%end
%subtracting each point in the kernel of tracker1 from tracker0 points
diff_x_kernel=zeros(3,3,points);
diff_y_kernel=zeros(3,3,points);
diff_z_kernel=zeros(3,3,points);
for i= 1: tr_row
for j= 1: tr_col
pos= (i-1)*tr_col + j;
diff_x_kernel(:,:,pos)=TR0_X_2D(i,j)- kernel_x(:,:,pos);
diff_y_kernel(:,:,pos)=TR0_Y_2D(i,j)- kernel_y(:,:,pos);
diff_z_kernel(:,:,pos)=TR0_Z_2D(i,j)- kernel_z(:,:,pos);
end
end
% euclidean distance
%euc_d_sq= zeros(3,3);
%for j=1:3;
% for i=1:3
% euc_d_sq(i,j)=(diff_x(i,j).^2) + (diff_y(i,j).^2) + (diff_z(i,j).^2);
% end
%end
%euc_d=sqrt(euc_d_sq);
% finding the euclidean distances of the difference kernels
euc_d_sq_k=zeros(3,3,points);
euc_d_k= zeros(3,3,points);
for k= 1 : points
for i= 1: 3
for j= 1:3
euc_d_sq_k(i,j,k)=(diff_x_kernel(i,j,k).^2)+ (diff_x_kernel(i,j,k).^2) +
(diff_x_kernel(i,j,k).^2);
end
end
euc_d_k(:,:,k)= sqrt(euc_d_sq_k(:,:,k));
end
%Find the minimum Euclidean distance
%min_d=min(min(euc_d));
%for j=1:3
% for i= 1:3
% if euc_d(i,j)== min_d;
% I= i; J=j;
% end
%end
%end
%finding min euc distance in kernels
I_rel=zeros(1,points);
J_rel=zeros(1,points);
K_rel=zeros(1,points);
min_d_k= zeros(1,points);
for k= 1: points
```

```
min_d_k(k)= min(min(euc_d_k(:,:,k)));
end
for k= 1:points
for j=1:3
for i= 1:3
if euc_d_k(i,j,k)== min_d_k(k);
I_rel(k)= i; J_rel(k)=j;K_rel(k)=k;
end
end
end
end
%Finding the indices for the new (u,v)
%row= I- 2;
%col= J- 2;
%new_corr_pt_r= row + corr_pt_r;
%new_corr_pt_c= col + corr_pt_c;
% finding the location of the correspondence points
rel_row=zeros(1,points);
rel_col=zeros(1,points);
I_el= zeros(1,points);
J_el=zeros(1,points);
%TR1_X_2D_kernel_up_new=TR1_X_2D_kernel_up;%updating new correspondence
points
%TR1_Y_2D_kernel_up_new=TR1_Y_2D_kernel_up;
%TR1_Z_2D_kernel_up_new=TR1_Z_2D_kernel_up;
for k=1:points
rel_row(k)= 2-I_rel(k); rel_col(k)= 2-J_rel(k);
for i =1:tr_row
for j= 1: tr_col
if(k-j==(i-1)*tr_col)
I_el(k)=i;
J_el(k)=j;
end
end
end
end
Rel_row= [0,rel_row,0];
Rel_col=[0,rel_col,0];
U= zeros(1, points);
V= zeros(1, points);
k=1;l=2;x=2;i=2;
while(k<points+1)
U(k)= i- Rel_row(l);
V(k)= x- Rel_col(l);
l=l+1;k=k+1;x=x+1;i=i+1;
if(i==33)
i=2;
end
if(x==31)
x=2;
end
end
TR1_X_2D_kernel_up_new=zeros(tr_row+2,tr_col+2);
TR1_Y_2D_kernel_up_new=zeros(tr_row+2,tr_col+2);
TR1_Z_2D_kernel_up_new=zeros(tr_row+2,tr_col+2);
n=1;
for i= 2: 32
for j= 2:30
TR1_X_2D_kernel_up_new(i-1,j-1)=TR1_X_2D_kernel_up(U(n),V(n));
TR1_Y_2D_kernel_up_new(i-1,j-1)=TR1_Y_2D_kernel_up((U(n)),(V(n)));
TR1_Z_2D_kernel_up_new(i-1,j-1)=TR1_Z_2D_kernel_up((U(n)),(V(n)));
n=n+1;if(n==points+1)
i=33;j=31;
end
end
end
TR1_X_2D_k_up_new=TR1_X_2D_kernel_up_new(1:31,1:29);
TR1_Y_2D_k_up_new=TR1_Y_2D_kernel_up_new(1:31,1:29);
```

```
TR1_Z_2D_k_up_new=TR1_Z_2D_kernel_up_new(1:31,1:29);
%Defining range of the Tracker 1
%start_row= new_corr_pt_r - 0.5*(tr_row-1);
%start_col= new_corr_pt_c - 0.5*(tr_col-1);
%end_row= new_corr_pt_r + 0.5*(tr_row-1);
%end_col= new_corr_pt_c + 0.5*(tr_col-1);
%Finding the new C and I images wrt to the new corresspondence point
%TR1_C_up= Acc_im4(start_row:end_row,start_col:end_col,2);
%TR1_I_up= Acc_im5(start_row:end_row,start_col:end_col,2);
TR1_C_ker_up=zeros(tr_row+1,tr_col);%Acc_im4((I_el-rel_row),(J_el-
rel_col),2);
TR1_I_ker_up=zeros(tr_row,tr_col);%Acc_im5((I_el-rel_row),(J_el-
rel_col),2);
for i= 1:31
for j= 1:29
k= (i-1)*tr_col+j;
TR1_C_ker_up(i,j)=TR1_C_k((U(k)),(V(k)));
TR1_I_ker_up(i,j)=TR1_I_k((U(k)),(V(k)));
end
end
%Creating 1D from 2D for X Y Z matrices:
tr_col=29;
tr_row=31;
%TR1_X_1D= zeros(1,tr_col*tr_row);
%TR1_Y_1D= zeros(1,tr_col*tr_row);
%TR1_Z_1D= zeros(1,tr_col*tr_row);
TR1_X_1D_k=zeros(1,tr_col*tr_row);
TR1_Y_1D_k=zeros(1,tr_col*tr_row);
TR1_Z_1D_k=zeros(1,tr_col*tr_row);
for j= 1:tr_col
for i= 1:tr_row
index=i*tr_col+j;
if(index>tr_col*tr_row)
index=tr_row*tr_col;
end
%TR1_X_1D(index)=TR1_X_2D_up(i,j);%((i-1)*tr_col)+j)=TR0_X_2D(i,j);
%TR1_Y_1D(index)=TR1_Y_2D_up(i,j);%((i-1)*tr_col)+j)=TR0_Y_2D(i,j);
%TR1_Z_1D(index)=TR1_Z_2D_up(i,j);%((i-1)*tr_col)+j)=TR0_Z_2D(i,j);
TR1_X_1D_k(index)=TR1_X_2D_k_up_new(i,j);
TR1_Y_1D_k(index)=TR1_Y_2D_k_up_new(i,j);
TR1_Z_1D_k(index)=TR1_Z_2D_k_up_new(i,j);
end
end
%Writing MAT5 for Tracker 1
mat5file='Tracker1';
%tr1_c(:,:,1)= uint8(TR1_C_up);
tr1_c_k(:,:,1)= uint8(TR1_C_ker_up);
%tr1_c(:,:,2)= uint8(Acc_im4_ch2(start_row:end_row,start_col:end_col,2));
tr1_c_k(:,:,2)= uint8(Acc_im4_ch2(min(U):max(U),min(V):max(V),2));
12/22/08 7:44 PM C:\MATLAB7\work\Kernel_for_each_point.m 11 of 11
%tr1_c(:,:,3)= uint8(Acc_im4_ch3(start_row:end_row,start_col:end_col,2));
tr1_c_k(:,:,3)= uint8(Acc_im4_ch3(min(U):max(U),min(V):max(V),2));
%tr1_i(:,:,1)= uint8(TR1_I_up);
tr1_i_k(:,:,1)= uint8(TR1_I_ker_up);
%tr1_i(:,:,2)= uint8(TR1_I_up);
tr1_i_k(:,:,2)= uint8(TR1_I_ker_up);
%tr1_i(:,:,3)= uint8(TR1_I_up);
tr1_i_k(:,:,3)= uint8(TR1_I_ker_up);
%result=mat5write(mat5file,tr1_c,tr1_i,TR1_X_1D,TR1_Y_1D,TR1_Z_1D);
result=mat5write(mat5file,tr1_c_k,tr1_i_k,TR1_X_1D_k,TR1_Y_1D_k,TR1_Z_1D_k)
;
```

References

1. http://www.htscreening.org/
2. http://en.wikipedia.org/wiki/Assay
3. Novel Modular Vascular Patterning Assay for HTS. PI: D.L. Lau, Co-PI: L.G Hassebrook. National Cancer Institute. 5R01CA131059-02. July 1, 2008 through June 30,2010.
4. http://en.wikipedia.org/wiki/High-throughput_screening
5. http://www.ncgc.nih.gov/guidance/section1.html#introduction
6. High-throughput screening: new technology for the 21st century *Current Opinion in Chemical Biology*, Volume 4, Issue 4, Pages 445-451 R.Hertzberg.
7. P. Ramm, Imaging systems in assay screening. *Drug Discov Today* **4** (1999), pp. 401–410 Review of CCD-based systems for rapid measurement of scintillation, luminescence and fluorescence for HTS
8. A.J. Pope, U. Haupts and K.J. Moore, Homogeneous fluorescence readouts for miniaturized high-throughput screening; theory and practice. *Drug Discov Today* **4** (1999), pp. 350–362
9. Z. Li, S. Mehdi, I. Patel, J. Kaooya, M. Judkins, W. Zhang, K. Diener, A. Lzada and D. Dunnington, An ultra-high throughput screening approach for an adenine transferase using fluorescence polarization. *J Biomol Screen* **5** (2000), pp. 31–38 HTS application paper using fluorescence anisotropy including 1536-well plate work.
10. I. Gibbons, Microfluidic assays for high-throughput submicroliter assays using capillary electrophoresis. *Drug Discov Today* **1** suppl (2000), pp. 33–37 Applications of microchannel-based separation and detection for ultra low volume HTS enzyme assays.
11. Q.S. Hanley, P.J. Verrbeer and T.M. Jovin, Optical sectioning fluorescence spectroscopy in a programmable array microscope. *Appl Spectroscopy* **52**(1998), pp. 783–789.
12. http://en.wikipedia.org/wiki/Hough_transform
13. http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm

14. Hart, P. E., "How the Hough Transform was Invented", IEEE Signal Processing Magazine, Vol 26, Issue 6, pp 18 - 22 (November, 2009) .
15. Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Comm. ACM, Vol. 15, pp. 11–15 (January, 1972
16. http://matwbn.icm.edu.pl/ksiazki/amc/amc18/amc1818.pdf
17. Circular Hough Transform, Simon Just Kjeldgaard Pedersen Aalborg University, Vision, Graphics, and Interactive Systems,November 2007.
18. Fernandes, L.A.F. and Oliveira, M.M., "Real-time line detection through an improved Hough transform voting scheme," Pattern Recognition, Elsevier
19. Tahir Rabbani and Frank van den Heuvel, "Efficient hough transform for automatic detection of cylinders in point clouds" in Proceedings of the 11th Annual Conference of the Advanced School for Computing and Imaging (ASCI '05), The Netherlands, June 2005.
20. Vosselman, G., Dijkman, S: "3D Building Model Reconstruction from Point Clouds and Ground Plans", International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol 34, part 3/W4, October 22-24 2001, Annapolis, MA, USA, pp.37- 44.
21. Digital image processing by Gonzalez,second edition.
22. Priya Ravindranath, “Process optimization and validation of an assay for drug screening”, Center for Systems Manufacturing,College of Engineering,University of Kentucky,(2005-2006).

Vita

Priyanka Chaudhary was born on 30th May 1986 in Indore, India. She was awarded an Honors degree for Bachelors in Electronics and Communications Engineering from Rajiv Gandhi Technical University in 2007. She was nominated for a Summer Intensive Program at the Nagoya State University, Japan in 2008 and as an IAESTE intern at Wise Automotives, Seoul, South Korea in 2009. She worked as a Research Assistant at the University of Kentucky from June 2008 to September 2009.

Priyanka Chaudhary