A novel approach for determination of tree age using Learning Automata fast algorithm

KUMBHARE PRATHMESH RAJENDRA
Dept. of Computer Science
Missouri University of Science & Technology
Rolla, USA
pkxvd@mst.edu

***Abstract*—— calculating age of tree from images as a case of computer vision is attempted. The problem is constructed as a case of feature extraction with tree rings as feature of interest. Motivation for the problem is found in the detection of circles using Hough Transformation. For practical purposes, the problem is also presented a constraint of optimization. Fast algorithm utilizing Learning Automata is implemented for this purpose. Shape of tree rings used for this study is taken from plants of division conifers (Pinophyta) according to tree nomenclature system. Though this algorithm can be modified and applied to any plant. Comparative analysis for algorithm is made by changing parameters.**

***Index Terms*—Hough Transform, Learning Automata (LA), Pinophyta, Superellipse, Dendrochronology.**

I. INTRODUCTION

Growth of the plant across period of one year is recorded in the form of tree ring. So, these are also known as growth rings. By determining the number of such rings formed, one can predict the age of the plant with great accuracy. For this study, this problem is considered in computer vision or more correctly in feature extraction. Hough transform is the most commonly used technique for shape based feature extraction. Knowledge of tree ring shape is required for this purpose. In generality, there is no single equation governing the shape for every type of plant. But, recent studies on the spiral radial growth of conifers suggest that shape of tree ring can be more accurately approximated by equations (1), (2), (3) of superellipse [1].

Dendrochronology is the scientific process of dating tree rings. By analyzing the growth pattern, one can deduce environmental conditions during the past. A timeline can be created using these results to obtain more information regarding evolution of the tree [2]. Formal process includes acquiring a sample by drilling a hole in tree from bark to the core or cutting a cross section of the tree trunk (Fig. 1). This sample is then mounted on the slide using some chemicals. The slide is studied and compared with known patterns to make conclusions. This method involves lot of manual work and it is time consuming. Motivation for this study comes from the fact that feature extraction methods can be employed to images of these samples for fast and accurate analysis. Since the first step in this study is determination of tree age, two different algorithms are implemented to deduce the age from given images. A brief overview for both these approaches is given below.

First approach is the Hough Transform algorithm. This method is most common for feature extraction problems. Basic approach was constructed for detection of straight lines in the images, but a generalization can be made for detecting any arbitrary curve like circle [3]. A practical issue with this algorithm is that it utilizes voting in the parameter space. This implies that as more parameters are required to describe a curve, the algorithm consumes more storage size and becomes more computationally intensive. A different method which has constraints of optimization cost is needed. To overcome this issue, another approach utilizing Learning Automata is implemented [4]. Motivation for this approach is that it can detect complex curves with speed and accuracy. As this method deals in the probability space, it solves our computation issue and analysis of the results becomes relatively easier. It uses linear reward inaction scheme for optimal convergence.

The aim of this report is to present an alternative method for dating tree rings as a problem in computer vision. It can also be argued from this study that given certain prior knowledge, appropriate feature extraction techniques can be employed to make more detailed pattern analysis. A similar analysis is made in a recent study which utilizes a different approach [5]. The foundation basis for this method is more rigorous in mathematics and hence provides more generality and flexibility in deciding required accuracy and computation time limits. The report is organized as follows. The basic mechanism used for algorithm construction is covered in Section II named methodology. Simulation results obtained from the case study of conifers are presented and analyzed in Section III. Three dimensional plots are also provided for visualization in this section. Conclusions based on Hough Transformation and Learning Automata approach are included in Section IV. Matlab codes used for simulation are included in the appendix.

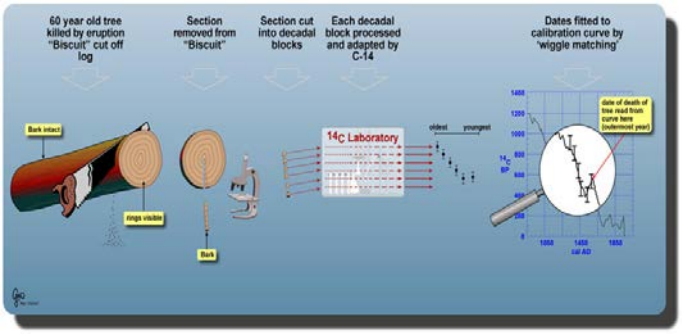

Fig. 1. Dendrochronology Process

II. METHODOLOGY

A. Tree Ring Equations

Geometric shape estimation of the tree rings is an active research area. Most common assumption is that these rings are circularly shaped. But, in reality they are hardly perfect circles. Further study indicates that they can be better approximated by superellipses [1]. This curve is a generalized version of ellipse. It can be used to represent circle, square, ellipse & rectangle by appropriate parameter selection. The equation describing curves of this nature can be mathematically written as:

$$\left|x/a\right|^n + \left|y/b\right|^n = 1 \quad (1)$$

Where:

x, y are the Cartesian coordinates,
a is the major semi-axis radius,
b is the minor semi-axis radius & $(0 < b \leq a)$,
n is a power.

Equation (1) can be modified to a more generalized version by using polar coordinates instead of Cartesian.

$$r = \left(\left|\cos \Phi/a\right|^n + \left|\sin \Phi/b\right|^n\right)^{-1/n} \quad (2)$$

$$r = a \,.\, \left(\left|\cos \Phi\right|^n + \left|\sin \Phi/k\right|^n\right)^{-1/n} \quad (3)$$

Where:

r is the radial distance between pole and point on curve,
φ is the angle of radial vector,
x coordinate is given by r cos φ,
y coordinate is given by r sin φ,
k is the ratio of semi-axis radius $(k=b/a \leq 1)$.

B. Hough Transform Algorithm

Under the assumption that tree ring is circular shaped, an algorithm to detect circles in image is designed. Hough Transform is a shape detection feature extraction method which can detect any arbitrary shape in theory. Basic model is derived for detection of straight line. By replacing the straight-line equation with the one for circle, it is possible to modify the algorithm for detection of circles. The parametric equation used in the Hough Transform algorithm is given by:

$$x = a + R\cos(\theta) \quad (4)$$

$$y = b + R\sin(\theta) \quad (5)$$

Where:

x, y are the circle point coordinates,
R is the radius of the circle,
a, b are the circle center coordinates,
θ is the angle with varies (0,360) degrees.

At the initialization stage, the image is properly scaled and converted into grayscale. 'Canny' edge filter is applied to this image and edge points are acquired. Then, parameter space is defined appropriately for voting purposes. For circle detection, the important parameters are center coordinates and radius. So, the parameter space will be present in 3 dimensions and its size will depend upon image size. The program loops through every edge point in the image and treats it as a point on the circle. By plugging this point coordinate values in the above mentioned parametric equation, center coordinates and radius value are determined. Vote for these combination of values is updated by one in the parameter space. In the inference part of the algorithm, the program searches for candidate with maximum number of votes in the parameter space and then plots circle with corresponding values. The parameter space which is also known as accumulator, stores information about points where overlapping occurs. Therefore, the final values obtained from the algorithm are the center points and radius distance on which maximum number of edge points would lie.

Two of the most important drawbacks for this algorithm observed are:

- As the number of parameters increase, the size of parameter space and hence computation time increases.
- Problems in detection can arise when objects are placed near each other.

The second issue can be limited by appropriate selection of bin size for the parameter space. Smaller size ensures greater accuracy but tends toward increase in computation time. For our study, superellipse curve is considered for detection. This requires minimum of 5 different parameters to describe the shape of rings. From computationally point of view, this increases the complexity of the parameter space greatly. The rings present in the image are concentric in nature. Determining the exact location of these rings would be very difficult using this approach. Hence, another algorithm is necessary for this task which emphasizes on optimality as well as accuracy.

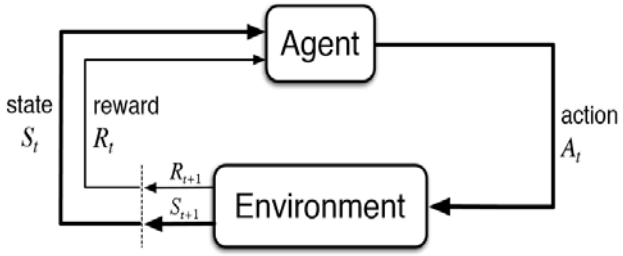

Fig. 2. Block diagram for reinforcement learning

C. Learning Automata Algorithm

Learning Automata (LA) is a type of Machine Learning algorithm whose foundations are based on reinforcement learning (Fig. 2). This method ensures that the solution will converge to a global minimum in finite number of iterations. The algorithm starts by assigning equal probability to all the actions. One action from all possibilities is selected randomly and then response of the environment to this action is observed. According to some learning law, probability values associated with all actions is updated [6]. These steps complete a single iteration. After certain iterations, a single action prevails with highest probability and the system is said to be converged successfully.

Detection of circles is attempted using this approach. For obtaining all possible actions, firstly the image is converted into grayscale and is passed through edge filter. From these edge points, only five percent are randomly selected. All possible combinations of three points from this set are made. It can be mathematically shown that three points lying on the circumference of circle are sufficient to represent it. Assuming that these points lie on the circumference, centre coordinates and radius of that circle are calculated. This set of three points are the required parameters for describing circle. Similarly, parameters for every possible combination is obtained. Each of this set is the potential candidate for circle. By deleting repeated or non-required candidates, we finally get all the actions for the algorithm.

For initialization purposes, all the actions are assigned equal probability. Then, a set of actions is selected randomly. A reinforcement signal β is computed for each action. From parameter values of an action, all the coordinates of the points lying on the circumference of a circle are calculated. For all these points presence of edge points at that location are checked. Amount of points present on the edge point pixels are computed in a metric called reinforcement signal. A value of β near 1 indicates that higher number of pixels are edge points for considered action. Depending upon value of reinforcement signals for each action, probability associated with each one is updated. This variable can be represented mathematically as:

$$\beta(A_1) = \sum E_1(x_i, y_i) / N_s \quad (6)$$

Where:

x_i, y_i are the point coordinates computed from parameters,
E equal to 1 if pixel is located at (xi, yi) or 0 otherwise,
N_s is the number points to check.

For updating the probability of actions after each iteration, linear reward/inaction (LRI) scheme is utilized. If the value of β is one, then maximum reward is given to that action and similarly if the value is zero, then null reward is given. In this scheme, a quantitative reward is provided to one action and zero penalty (no reward) is provided to all other actions in that iteration. So, the probability value associated with the action having greatest reinforcement signal value is increased, while values for every other action are kept as it is. The probability update laws are given mathematically as:

$$p_i(k+1) = p_i(k) + \theta.\beta(x_i).(1 - p_i(k)) \quad (7)$$

$$p_j(k+1) = p_j(k) - \theta.\beta(x_i).p_j(k), \quad if\ j \neq 1 \quad (8)$$

Where:

x_i represents action i from set of n actions,
$\beta(.) \in [0,1]$ is the reinforcement signal for that action,
$p(k)$ is the probability at k^{th} iteration.

As the iterations advance, the probability values for successful actions will evolve and approach to one. At the end of iterations, the action with largest probability will be detected circle. Amount of iterations taken are equal to one half of number of possible actions as a rule. Further analysis on this probability distribution curve will be needed to locate multiple circles in the image. So, overview of this algorithm can be given in following main points:

- Convert image into grayscale & apply edge filter.
- Randomly select 5 % of the edge points.
- From this pool create all possible combinations of set of points.
- Obtain parameter values from these set of points. Eliminate all the repeated and non-interest combinations to get all possible actions.
- Assign equal probabilities to each action. Set number of iterations to half of number of possible actions. Follow the next 3 steps until these pre-assigned iterations are completed.
- Randomly select a set of actions from the pool of all possible actions.
- Evaluate reinforcement signal for each selected action by using equation (6).
- Update the probability distribution by using the equations (7) & (8).
- Analyze the probability distribution curve in order to determine number of rings and their parameter values.

Since the shape of tree rings is rarely circle, error in the results start to occur. This is very much evident when the rings are elliptically shaped. By using the earlier argument that superellipse equations better fit these rings than circles, the Learning Automata algorithm is implemented for these curves. The process explained earlier remains same for this model, but few modifications are necessary. In case of circles, only 3 edge points are required to describe the curve. Similarly, 5 points are required to describe a superellipse curve. (or ellipse in general) Additional parameters like power of the curve and ratio of semi-axis radius are needed to be defined. Results for different images are obtained from these two models. Results are discussed in the next section.

III. RESULTS & DISCUSSIONS

A. Hough Transform Result

Few images consisting single circle in them are provided to the algorithm. This method detects the circular curve accurately (Fig. 3 and Fig. 4). The Hough space consists accumulation of votes for parameter values. The three-dimensional plot is shown for a specific value of radius. (detected circle radius value) So, it contains all the accumulated votes for required radius spread across possible combinations of coordinates of the circle centre (Fig. 5). Values along x and y axis for corresponding peak (along z axis) gives the centre coordinates for detected circle. This model can be used for detection of tree rings as they can reasonably detect multiple circles in images as shown by some studies. The reasons for which this model is not implemented on multiple circle or tree ring images are:

- When size of image becomes larger, (like 600*600 pixels) then storing parameter space becomes difficult. In case of superellipse, minimum of 5 parameters are required. Hence, this model is not much useful.
- Tree rings can be more accurately described by superellipse than circle curve.

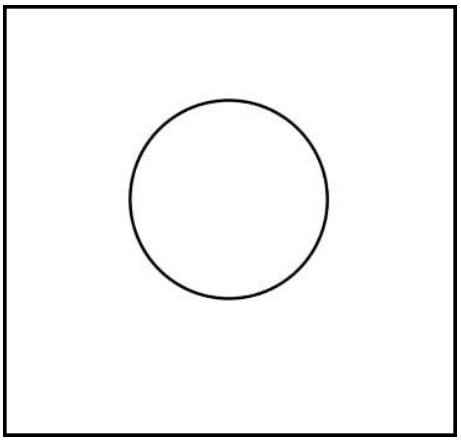

Fig. 3. Image used for testing Hough Transform model

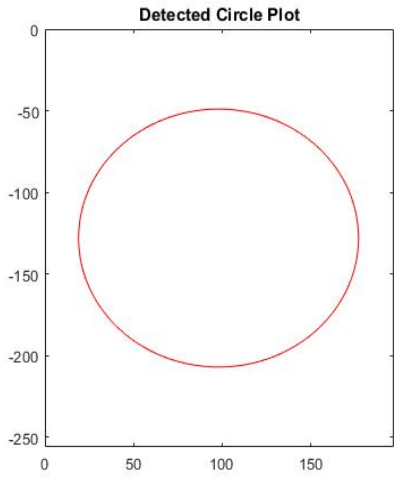

Fig. 4. Predicted circle for Hough Transform model

The range of radius given to the model for this specific image is from 25 pixels to 161 pixels. (one fourth of image diagonal size) Providing appropriate range ensures that the model executes faster and occupies

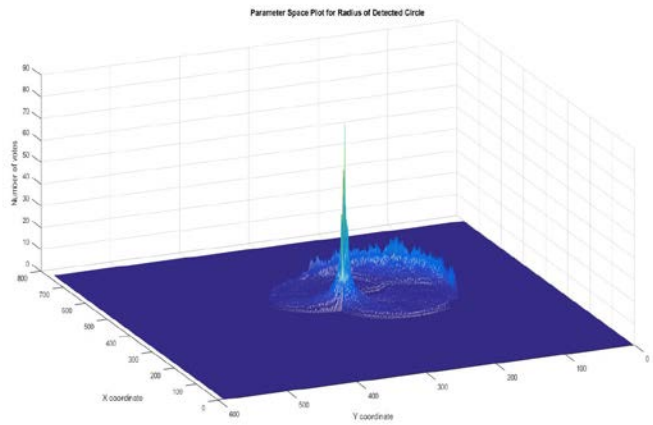

Fig. 5. Distribution of votes in parameter space

B. Learning Automata Circle Result

The algorithm is given images with multiple concentric circle and ellipse. Before running the model, few values of parameter must be defined. For these set of images, learning rate of 0.005 is assigned. For relatively fast computation speeds, the image is scaled down until number of selected edge points are around 1600. Range of radius is selected according to the image. For the application of tree ring measurement, radius range from few pixels to one half of image diagonal length must be taken. For ensuring that repeated candidates or similar candidates do not compete with each other, additional conditions are employed. Number of iterations performed are equal to one half of

number of actions remaining after these conditions. In order to obtain coordinates for potential candidates from centre values and radius size, “get mid-point circle” algorithm is utilized. (It is standard algorithm) The model predicts circular curves with satisfactory accuracy (Fig. 6). The result is given as the circle with highest probability. In case of multiple curves, by analysing the probability distribution plot, we can extract information about them (Fig. 7). Number of peaks in this curve gives us the total number of circles in this image. Locations of the peaks are also plotted to get an idea (Fig. 8).

Fig. 6. Concentric circles with predicted circle

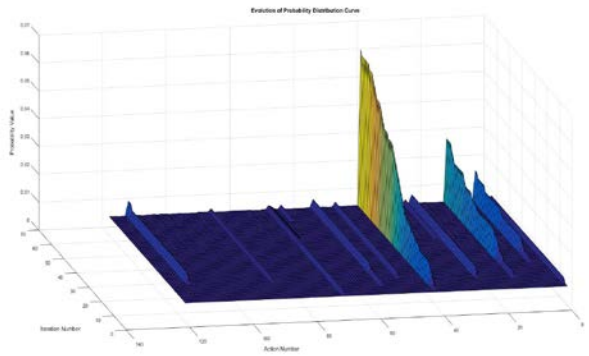

Fig. 7. Evolution of probability distribution space

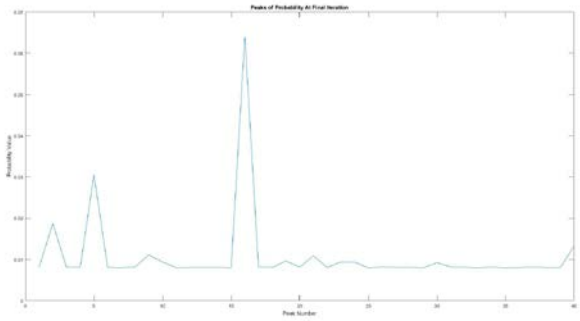

Fig. 8. Locations of peak in the probability curve

Another important point to mention is the value of threshold until which the probability value can be considered as peak. In other words, we must decide how much tall the peak must be in order to consider it. Normally it is taken as 10 percent of maximum peak value. This model is applied to sample tree image (Fig. 9). It predicts the number of rings satisfactorily. But, as the ring is not circle exactly, this can create errors in further analysis.

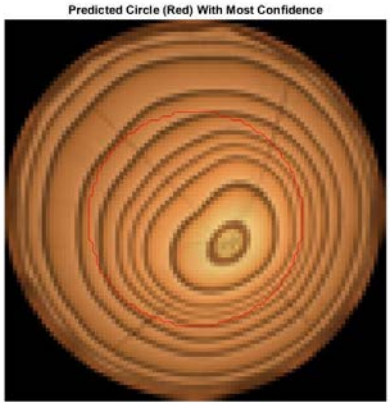

Fig. 9. Result for sample tree image

C. Learning Automata Superellipse Result

Computationally, this model is a bit intensive than earlier one. The Learning Automata algorithm ensures that converged solutions are obtained in minimum possible time. All the potential candidates are selected from combination of 6 edge points. Following the results of recent studies as a guideline, power of the curve (n) is taken as 1.90 and the ratio of semi-radius axis (k) is taken as 0.95. Actual tree sample images are given to this model for prediction. The predicted superellipse curve fits better than the circle *(Fig. 10).* Also, from the probability distribution curve, similar conclusion can be made as the magnitude of peaks are greater. So, the model is more confident in predicting these rings *(Fig. 11).*

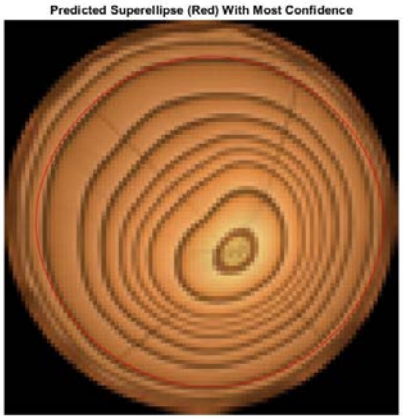

Fig. 10. Prediction result of tree ring

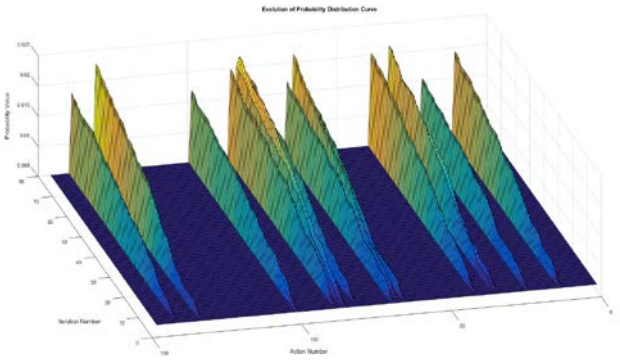

Fig. 11. Prediction results in probability space

An important point to consider while analyzing results from Learning Automata algorithm is that the predictions are not fixed. Since, the initialization for action sets is random, the final probability values are different each time by a bit. To get even better results, just execute the algorithm multiple times and average them out.

Table 1: Results for some tree images

Image Name	Manually Detected Rings	LA model Rings	Error
tree_img_1.jpg	13	14	1
tree_img_2.jpg	14	16	2
tree_img_3.jpg	17	19	2
tree_img_4.jpg	25	28	3
tree_img_5.jpg	24	22	2
tree_img_6.jpg	23	26	3
tree_img_7.jpg	22	20	2

IV. CONCLUSIONS

The Learning Automata model using superellipse predicts the tree rings satisfactorily (Table 1). Hough Transform is a good method for detection of simple curves and its limitation are evident as complexity of curves increases. Study of growth pattern of tree rings is active research area and new analysis are still going on. To achieve further accuracy, following factors in this algorithm can be improved upon:

- For the sake of simplicity, the algorithm implemented for getting parameters of curve from edge points is assumed as ellipse. Then the predictions are made for superellipse. By calculating these parameters for superellipse, accuracy can be increased.
- Some of the rings in the image (Fig. 10) have orientation. The model calculates these orientation angle but does not use them in predictions. Addition of this factor can also improve the results.

It can be concluded that both the Learning Automation algorithms provide satisfactory results in determining tree age from images. The superellipse model gains an advantage due to the fact that its predictions fit better and are more confident. This method can be implemented on almost any plant species just by knowing the power of the curve (n) and ratio of semi-axis radius (k) for its tree ring. Better predictions are useful when further analysis on tree rings is required. This includes studies like determining width distance between two rings to get information regarding growth pattern of the tree in that specific year. Improvements in the computation time for program execution can be made by writing more efficient codes or implementing them in python. As a part of future modification, this algorithm can be integrated into a mobile application.

REFERENCES

[1] "Capturing spiral radial growth of conifers using the superellipse to model tree-ring geometric shape" Internet: https://doi.org/10.3389/fpls.2015.00856 [May. 10, 2017]

[2] "Dendrochronology"Internet:https://en.wikipedia.org/wiki/Dendrochronology[May. 10, 2017]

[3] "Feature Extraction Using the Hough Transform" Internet: http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1260&context=spacegrant[May. 10, 2017]

[4] Cuevas, E., Wario, F., Osuna-Enciso, V., Zaldivar, D., Pérez-Cisneros, M. Fast algorithm for multiple-circle detection on images using learning automata, IET Image Processing 6 (8) , (2012), pp. 1124-1135

[5] "Method of tree-ring image analysis for dendrochronology" Internet:http://opticalengineering.spiedigitallibrary.org/article.aspx?articleid=1351720[May. 10, 2017]

[6] IEEE Transactions on systems, man, and cybernetics, vol. smc-4, no. 4, July 1974

APPENDIX

1. Hough Transform algorithm code:

```
clc;
clear all;
close all;

disp(' ## PROGRAM FOR CIRCULAR HOUGH TRANSFORM : PROJECT ## ');

%file directory

directory=char(pwd);
ImageDirectory = 'Hough_Test_Images\';
ImageFiles = dir(ImageDirectory);

origIm=imread([ImageDirectory ImageFiles(7).name]);  % change the number in
ImageFiles(3) from 3 to 7 for all images

%origIm=imresize(origIm,0.5);  % scale image appropriately if neccessary
origIm1=rgb2gray(origIm);      % converting into grayscale
BW = edge(origIm1,'canny');    % getting edge points
showIm=origIm;
imshow(showIm);


[rows, columns]=size(BW);

% Creating parameter space

RADIUS=(25:floor(sqrt((rows*rows)+(columns*columns))/2));    % Select appropriate
range of radius to search
THETA=(0:90);
A=(1:floor(3*columns));
B=(1:floor(3*rows));
PARAMETER_SPACE=zeros(size(A,2),size(B,2),size(RADIUS,2));

for i = 1:rows

    for ii = 1:columns

        if BW(i,ii)==1

            x=ii;
            y=-i;

            for r = RADIUS


                for t = 1:(size(THETA,2))

                    a(1,t)=(x-r*cosd(-(t-1)))+(columns);   % Shifting the values by
factor of 1
```

```
                    b(1,t)=-(y+r*sind(-(t-1)))+(rows);     % Parametric equations
taken as mentioned in report

                    % updating votes in parameter space


                    PARAMETER_SPACE(floor(a(1,t)),floor(b(1,t)),(r-(RADIUS(1)-
1)))=PARAMETER_SPACE(floor(a(1,t)),floor(b(1,t)),(r-(RADIUS(1)-1)))+1;

                end
            end
         end

    end

end

final_point_count=max(PARAMETER_SPACE(:));  % Determining the detected circle
final_radius=[];

for c= (RADIUS-(RADIUS(1)-1))

    % Obtaining parameter values for detected circle

    d=PARAMETER_SPACE(:,:,c);
    point_count=max(d(:));

    if point_count==final_point_count;

        final_radius=[final_radius; c];
        [aa,bb]=find(d==final_point_count);
        aa=aa-(columns);
        bb=-(bb-(rows));

        break

    else
        continue
    end

end

final_radius=max(final_radius)+(RADIUS(1)-1);
aa=max(aa);
bb=max(bb);
figure;
mesh(d);    % Plotting the parameter surface for determined circle
title('Parameter Space Plot for Radius of Detected Circle');
xlabel(' X coordinate ');
ylabel(' Y coordinate ');
zlabel(' Number of votes ');

for t = 1:(size(THETA,2)*4)

     xunit(1,t) = final_radius*cosd(t) + aa;
```

```
     yunit (1,t)= final_radius*sind(t) + bb;

end

figure;
h = plot(xunit,yunit,'r'); % plotting the detected circle
title('Detected Circle Plot');
axis([0 columns -rows 0]);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2. Learning Automata (Circle) Code:

```
clc;
%clear all;
%close all;

disp(' ## PROGRAM FOR CIRCLE DETECTION LEARNING AUTOMATA : PROJECT ## ');

%file directory

directory=char(pwd);
ImageDirectory = 'LA_Circle_Test_Images\';
ImageFiles = dir(ImageDirectory);
scale=0.5;
Edge_Pt_Num = 10000;

while Edge_Pt_Num>1200;   % Adjusting Image Size

    origIm=imread([ImageDirectory ImageFiles(10).name]);  % change the number in
ImageFiles(3) from 3 to 10 for all images
    origIm=imresize(origIm,scale);
    origIm1=rgb2gray(origIm);
    BW = edge(origIm1,'canny');

    [rows, columns] = size(BW);
    Edge_Pt_Num = sum(BW(:));
    Edge_Pt_Data = zeros(Edge_Pt_Num,2);
    Radius_Range = [2, floor(sqrt((rows*rows)+(columns*columns))/2)]; % Rmin & Rmax
for algorithm
    scale=scale-0.01;

end

showIm=origIm;
imshow(showIm);
count=1;

% Creating Edge Point Data Matrix
```

```
for ii = 1:columns

    for i = 1:rows

        if BW(i,ii)==1

            y=-i;
            x=ii;

            Edge_Pt_Data(count,1) = x;
            Edge_Pt_Data(count,2) = y;
            count=count+1;
        end
    end
end

% Randomly Selecting 5 Percent Of Total Edge Points

Edge_Pt_Sel_Num=round((Edge_Pt_Num*5)/100);
Edge_Pt_Sel_Val=randperm(Edge_Pt_Num,Edge_Pt_Sel_Num);

% Forming Matrix For All Combinations Of Possible Candidate Circles (3 Pts)

Circle_Candidates_Val = nchoosek(Edge_Pt_Sel_Val,3);

% Computing Parameters For All The Circle Candidates (Center & Radius)

Circle_Candidates_Param=[];
Circle_Candidates_Param_final=[];

for num =1:size(Circle_Candidates_Val,1)

    % Co Ordinates Of 3 Non Colinear Points

    Pt_A = [Edge_Pt_Data(Circle_Candidates_Val(num,1),1),
Edge_Pt_Data(Circle_Candidates_Val(num,1),2)];
    Pt_B = [Edge_Pt_Data(Circle_Candidates_Val(num,2),1),
Edge_Pt_Data(Circle_Candidates_Val(num,2),2)];
    Pt_C = [Edge_Pt_Data(Circle_Candidates_Val(num,3),1),
Edge_Pt_Data(Circle_Candidates_Val(num,3),2)];

    A = [(Pt_B(1)^2 + Pt_B(2)^2 - (Pt_A(1)^2 + Pt_A(2)^2)), 2*(Pt_B(2)-Pt_A(2));
(Pt_C(1)^2 + Pt_C(2)^2 - (Pt_A(1)^2 + Pt_A(2)^2)), 2*(Pt_C(2)-Pt_A(2))];
    B = [2*(Pt_B(1)-Pt_A(1)), (Pt_B(1)^2 + Pt_B(2)^2 - (Pt_A(1)^2 + Pt_A(2)^2));
2*(Pt_C(1)-Pt_A(1)), (Pt_C(1)^2 + Pt_C(2)^2 - (Pt_A(1)^2 + Pt_A(2)^2))];

    Den = (4*((Pt_B(1)-Pt_A(1))*(Pt_C(2)-Pt_A(2))-(Pt_C(1)-Pt_A(1))*(Pt_B(2)-
Pt_A(2))));
    X_Zero = floor(det(A)/Den);
    Y_Zero = floor(det(B)/Den);
    Radius = floor(sqrt(((X_Zero-Pt_A(1))^2) + ((Y_Zero-Pt_A(2))^2)));

    % Eliminating The Candidates Outside The Interested Radius Range
```

```
    if X_Zero>0.45*columns && X_Zero<0.55*columns && Y_Zero<(-0.45)*rows && Y_Zero>(-
0.55)*rows && Radius>Radius_Range(1) && Radius<Radius_Range(2)

        Circle_Candidates_Param=[Circle_Candidates_Param; [X_Zero Y_Zero Radius]];

    end
end

Circle_Candidates_Param = unique(Circle_Candidates_Param,'rows');

% Eliminating The Candidates Which Are Similar To Each Other

num=1;

while num <= size(Circle_Candidates_Param,1)

    X_Zero_Range = ((Circle_Candidates_Param(num,1)-
1):(Circle_Candidates_Param(num,1)+1));
    Y_Zero_Range = ((Circle_Candidates_Param(num,2)-
1):(Circle_Candidates_Param(num,2)+1));
    Radius_Range1 = ((Circle_Candidates_Param(num,3)-
1):(Circle_Candidates_Param(num,3)+1));
    Repeat_Param_Vector = combvec(X_Zero_Range,Y_Zero_Range,Radius_Range1);
    Repeat_Param_Vector = Repeat_Param_Vector';
    %Repeat_Param_Vector =
removerows(Repeat_Param_Vector,'ind',[round(size(Repeat_Param_Vector,1)/2)]);
    X_Zero=Circle_Candidates_Param(num,1);
    Y_Zero=Circle_Candidates_Param(num,2);
    Radius=Circle_Candidates_Param(num,3);


Circle_Candidates_Param(any(ismember(Circle_Candidates_Param,Repeat_Param_Vector,'rows
'),2),:) = [];
    Circle_Candidates_Param_final = [Circle_Candidates_Param_final; [X_Zero Y_Zero
Radius]];

    num=num+1;
end


Remaining_Actions = size(Circle_Candidates_Param_final,1);
K_max = round(Remaining_Actions/2);

% Learning Automata Algorithm

Inital_Probability = (1/Remaining_Actions);
Prob_Update_Matrix = Inital_Probability*(ones(Remaining_Actions,1));
Prob_Action_Matrix = Prob_Update_Matrix;
%Beta = zeros(Remaining_Actions,1);
Learning_Rate = 0.005;


for i=1:K_max

     disp(i);
```

```
    % 1. Selectiion of actions 'Av' for the iteration

    z=rand; % Selecting Pseudo Random Number Between 0-1
    Prob_Update_Step = [];
    count=1;
    n=randperm(Remaining_Actions)';

    while sum(Prob_Update_Step)<z

         Prob_Update_Step(count,1)=Prob_Update_Matrix(n(count,1),1);
         count=count+1;

    end

    if count>=3

        n = n(1:(count-2),1);
        Prob_Update_Step = Prob_Update_Step(1:(count-2),1);

    end

    % 2. Calculation of reinforcement signal for the actions 'Av'

    Beta_Update_Vector=zeros(size(n,1),2);

    for ii = 1:size(n,1)

        [x1, y1] = getmidpointcircle(Circle_Candidates_Param_final(n(ii,1),1),
Circle_Candidates_Param_final(n(ii,1),2), Circle_Candidates_Param_final(n(ii,1),3));
        Error_Beta_Cal=zeros(size(x1));


        for m = 1:size(x1,1)

            if ismember([x1(m,1), y1(m,1)],Edge_Pt_Data,'rows') == true

                 Error_Beta_Cal(m,1) = 1;

            end

        end

        Beta_Cal = (sum(Error_Beta_Cal))/size(x1,1);
        Beta_Update_Vector(ii,1) = n(ii,1);
        Beta_Update_Vector(ii,2) = Beta_Cal;
        %Beta(n(ii,1),1) = Beta_Cal;

        %Prob_Update_Matrix(n(ii,1),1) = Prob_Update_Matrix(n(ii,1),1) +
(Learning_Rate*Beta(n(ii,1),1)*(1-Prob_Update_Matrix(n(ii,1),1)));

    end

    % Probability Update Law
```

```
    [row_max] = find(Beta_Update_Vector(:,2)==max(Beta_Update_Vector(:,2)));
    Action_Num_Max = Beta_Update_Vector(row_max,1);
    Beta_Max = Beta_Update_Vector(row_max,2);

    % A. Rewarding The Action With Highest Reinforcement Signal

    %for ii = 1:size(row_max,1)

        %Prob_Update_Matrix(Action_Num_Max(ii,1),1) =
Prob_Update_Matrix(Action_Num_Max(ii,1),1) + (Learning_Rate*Beta_Max(ii,1)*(1-
Prob_Update_Matrix(Action_Num_Max(ii,1),1)));
        %Beta_Update_Vector = removerows(Beta_Update_Vector,'ind',[row_max(ii,1)]);

     Prob_Update_Matrix(Action_Num_Max(1,1),1) =
Prob_Update_Matrix(Action_Num_Max(1,1),1) + (Learning_Rate*Beta_Max(1,1)*(1-
Prob_Update_Matrix(Action_Num_Max(1,1),1)));
     Beta_Update_Vector = removerows(Beta_Update_Vector,'ind',[row_max]);


    %end

    % B. Penalizing Rest Of The Actions

    for ii = 1:(size(n,1)-size(row_max,1))

        Action_Num = Beta_Update_Vector(ii,1);
        Beta_Val = Beta_Update_Vector(ii,2);

        Prob_Update_Matrix(Action_Num,1) = Prob_Update_Matrix(Action_Num,1) -
(Learning_Rate*Beta_Val*Prob_Update_Matrix(Action_Num,1));

    end

    Prob_Action_Matrix = [Prob_Action_Matrix, Prob_Update_Matrix];

end

surf(Prob_Action_Matrix);
title(' Evolution of Probability Distribution Curve ');
xlabel(' Iteration Number ');
ylabel(' Action Number ');
zlabel(' Probability Value ');
figure;

% plotting estimated circle

Final_Action_Prob=max(Prob_Action_Matrix(:,size(Prob_Action_Matrix,2)));
[action_row_max] =
find(Prob_Action_Matrix(:,size(Prob_Action_Matrix,2))==Final_Action_Prob);

[xf, yf] =
getmidpointcircle(Circle_Candidates_Param_final(action_row_max,1),(rows+Circle_Candida
tes_Param_final(action_row_max,2)),Circle_Candidates_Param_final(action_row_max,3));
```

```
showIm=origIm;
imshow(showIm);
hold on;
plot(xf,yf,'r');
title(' Predicted Circle (Red) With Most Confidence ');

% For analysis:

[sortedX,sortingIndices] =
sort(Prob_Action_Matrix(:,size(Prob_Action_Matrix,2)),'descend');

figure;
pks = findpeaks(Prob_Action_Matrix(:,size(Prob_Action_Matrix,2)'));
plot(pks);
title(' Peaks of Probability At Final Iteration ');
xlabel(' Peak Number ');
ylabel(' Probability Value ');

max_val=max(sortedX);
```

%%%

3. Learning Automata (Superellipse) Code:

```
clc;
%clear all;
%close all;

disp(' ## PROGRAM FOR TREE RINGS DETECTION LEARNING AUTOMATA : PROJECT ## ');

%file directory

directory=char(pwd);
ImageDirectory = 'LA_Superellipse_Test_Images\';
ImageFiles = dir(ImageDirectory);
scale=0.5;
Edge_Pt_Num = 10000;

while Edge_Pt_Num>3000;   % Adjusting Image Size

    origIm=imread([ImageDirectory ImageFiles(12).name]);    % change the number in
ImageFiles(3) from 3 to 12 for all images
    origIm=imresize(origIm,scale);                           % for
ImageFiles(3),ImageFiles(4),ImageFiles(5) take N=2 and K=0.5 below
    origIm1=rgb2gray(origIm);                                % for rest of the images
take N=1.9 and K=0.95 below
    BW = edge(origIm1,'canny');
    disp(Edge_Pt_Num);
```

```
    [rows, columns] = size(BW);
    Edge_Pt_Num = sum(BW(:));
    Edge_Pt_Data = zeros(Edge_Pt_Num,2);
    Radius_Range = [2, floor(sqrt((rows*rows)+(columns*columns))/2)]; % Rmin & Rmax
for algorithm
    scale=scale-0.01;

end

showIm=origIm;
imshow(showIm);
count=1;

% Creating Edge Point Data Matrix

for ii = 1:columns

    for i = 1:rows

        if BW(i,ii)==1

            y=-i;
            x=ii;

            Edge_Pt_Data(count,1) = x;
            Edge_Pt_Data(count,2) = y;
            count=count+1;
        end
    end
end

% Randomly Selecting 5 Percent Of Total Edge Points

Edge_Pt_Sel_Num=round((Edge_Pt_Num*5)/100);
Edge_Pt_Sel_Val=randperm(Edge_Pt_Num,Edge_Pt_Sel_Num);
Edge_Pt_Data1=Edge_Pt_Data;
Edge_Pt_Data1=removerows(Edge_Pt_Data1,'ind',Edge_Pt_Sel_Val');
Edge_Pt_Num1=size(Edge_Pt_Data1,1);
Edge_Pt_Sel_Val1=randperm(Edge_Pt_Num1,Edge_Pt_Sel_Num);

% Forming Matrix For All Combinations Of Possible Superellipse Candidates (6 Pts)

Circle_Candidates_Val1 = nchoosek(Edge_Pt_Sel_Val,3);
Circle_Candidates_Val2 = nchoosek(Edge_Pt_Sel_Val1,3);

Circle_Candidates_Val = [Circle_Candidates_Val1, Circle_Candidates_Val2];


% Computing Parameters For All The Circle Candidates (Center & Radius)

Circle_Candidates_Param=[];
Circle_Candidates_Param_final=[];
warning('off');
```

```
for num =1:size(Circle_Candidates_Val,1)

    points=Circle_Candidates_Val(num,:);

    if size(points,2)==size(unique(points),2)


Xp=[Edge_Pt_Data(points(1,1),1);Edge_Pt_Data(points(1,2),1);Edge_Pt_Data(points(1,3),1
);Edge_Pt_Data1(points(1,4),1);Edge_Pt_Data1(points(1,5),1);Edge_Pt_Data(points(1,6),1
)];

Yp=[Edge_Pt_Data(points(1,1),2);Edge_Pt_Data(points(1,2),2);Edge_Pt_Data(points(1,3),2
);Edge_Pt_Data1(points(1,4),2);Edge_Pt_Data1(points(1,5),2);Edge_Pt_Data(points(1,6),2
)];
        E=fit_ellipse(Xp,Yp);

         X_Zero=E.X0_in;
         Y_Zero=E.Y0_in;
         Radius=round(E.long_axis/2);

        % Eliminating The Candidates Outside The Interested Radius Range

        if sum(X_Zero)~=0 || sum(Y_Zero)~=0 || sum(Radius)~=0

            if X_Zero>0.46*columns && X_Zero<0.54*columns && Y_Zero<(-0.46)*rows &&
Y_Zero>(-0.54)*rows && Radius>Radius_Range(1) && Radius<Radius_Range(2)

                Circle_Candidates_Param=[Circle_Candidates_Param; [X_Zero Y_Zero
Radius]];

            end
        end
    end
end

Circle_Candidates_Param = unique(Circle_Candidates_Param,'rows');
Circle_Candidates_Param=floor(Circle_Candidates_Param);

% Eliminating The Candidates Which Are Similar To Each Other

num=1;

while num <= size(Circle_Candidates_Param,1)

    X_Zero_Range = ((Circle_Candidates_Param(num,1)-
1):(Circle_Candidates_Param(num,1)+1));
    Y_Zero_Range = ((Circle_Candidates_Param(num,2)-
1):(Circle_Candidates_Param(num,2)+1));
    Radius_Range1 = ((Circle_Candidates_Param(num,3)-
2):(Circle_Candidates_Param(num,3)+2));
    Repeat_Param_Vector = combvec(X_Zero_Range,Y_Zero_Range,Radius_Range1);
    Repeat_Param_Vector = Repeat_Param_Vector';
    %Repeat_Param_Vector =
removerows(Repeat_Param_Vector,'ind',[round(size(Repeat_Param_Vector,1)/2)]);
    X_Zero=Circle_Candidates_Param(num,1);
    Y_Zero=Circle_Candidates_Param(num,2);
```

```
    Radius=Circle_Candidates_Param(num,3);


Circle_Candidates_Param(any(ismember(Circle_Candidates_Param,Repeat_Param_Vector,'rows
'),2),:) = [];
    Circle_Candidates_Param_final = [Circle_Candidates_Param_final; [X_Zero Y_Zero
Radius]];

    num=num+1;
end

Remaining_Actions = size(Circle_Candidates_Param_final,1);
K_max = round(Remaining_Actions/2);

% Learning Automata Algorithm

Inital_Probability = (1/Remaining_Actions);
Prob_Update_Matrix = Inital_Probability*(ones(Remaining_Actions,1));
Prob_Action_Matrix = Prob_Update_Matrix;
%Beta = zeros(Remaining_Actions,1);
Learning_Rate = 0.0005;

% Ellipse properties:

N=1.9;
K=0.95;



for i=1:K_max

     disp(i);

     % 1. Selectiion of actions 'Av' for the iteration

     z=rand; % Selecting Pseudo Random Number Between 0-1
     Prob_Update_Step = [];
     count=1;
     n=randperm(Remaining_Actions)';

     while sum(Prob_Update_Step)<z

          Prob_Update_Step(count,1)=Prob_Update_Matrix(n(count,1),1);
          count=count+1;

     end

     if count>=3

         n = n(1:(count-2),1);
         Prob_Update_Step = Prob_Update_Step(1:(count-2),1);

     end
```

```
    % 2. Calculation of reinforcement signal for the actions 'Av'

    Beta_Update_Vector=zeros(size(n,1),2);

    for ii = 1:size(n,1)

        [x1, y1] = getsuperellipse(Circle_Candidates_Param_final(n(ii,1),1),
Circle_Candidates_Param_final(n(ii,1),2),
Circle_Candidates_Param_final(n(ii,1),3),N,K);
        Error_Beta_Cal=zeros(size(x1));


        for m = 1:size(x1,1)

            if ismember([x1(m,1), y1(m,1)],Edge_Pt_Data,'rows') == true

                Error_Beta_Cal(m,1) = 1;

            end

        end

        Beta_Cal = (sum(Error_Beta_Cal))/size(x1,1);
        Beta_Update_Vector(ii,1) = n(ii,1);
        Beta_Update_Vector(ii,2) = Beta_Cal;
        %Beta(n(ii,1),1) = Beta_Cal;

        %Prob_Update_Matrix(n(ii,1),1) = Prob_Update_Matrix(n(ii,1),1) +
(Learning_Rate*Beta(n(ii,1),1)*(1-Prob_Update_Matrix(n(ii,1),1)));

    end

    % Probability Update Law

    [row_max] = find(Beta_Update_Vector(:,2)==max(Beta_Update_Vector(:,2)));
    Action_Num_Max = Beta_Update_Vector(row_max,1);
    Beta_Max = Beta_Update_Vector(row_max,2);

    % A. Rewarding The Actions With Highest Reinforcement Signal

    for ii = 1:size(row_max,1)

         Prob_Update_Matrix(Action_Num_Max(ii,1),1) =
Prob_Update_Matrix(Action_Num_Max(ii,1),1) + (Learning_Rate*Beta_Max(ii,1)*(1-
Prob_Update_Matrix(Action_Num_Max(ii,1),1)));
         %Beta_Update_Vector = removerows(Beta_Update_Vector,'ind',[row_max(ii,1)]);

     %Prob_Update_Matrix(Action_Num_Max(1,1),1) =
Prob_Update_Matrix(Action_Num_Max(1,1),1) + (Learning_Rate*Beta_Max(1,1)*(1-
Prob_Update_Matrix(Action_Num_Max(1,1),1)));
     %Beta_Update_Vector = removerows(Beta_Update_Vector,'ind',[row_max]);


    end
```

```
    Beta_Update_Vector = removerows(Beta_Update_Vector,'ind',[row_max]);

    % B. Penalizing Rest Of The Actions

    for ii = 1:(size(n,1)-size(row_max,1))

        Action_Num = Beta_Update_Vector(ii,1);
        Beta_Val = Beta_Update_Vector(ii,2);

        Prob_Update_Matrix(Action_Num,1) = Prob_Update_Matrix(Action_Num,1) -
(Learning_Rate*Beta_Val*Prob_Update_Matrix(Action_Num,1));

    end

    Prob_Action_Matrix = [Prob_Action_Matrix, Prob_Update_Matrix];

end

surf(Prob_Action_Matrix);
title(' Evolution of Probability Distribution Curve ');
xlabel(' Iteration Number ');
ylabel(' Action Number ');
zlabel(' Probability Value ');
figure;

% plotting estimated circle

Final_Action_Prob=max(Prob_Action_Matrix(:,size(Prob_Action_Matrix,2)));
[action_row_max] =
find(Prob_Action_Matrix(:,size(Prob_Action_Matrix,2))==Final_Action_Prob);

showIm=origIm;
imshow(showIm);
hold on;


for i=1:size(action_row_max,1)

   [xf, yf] =
getsuperellipse(Circle_Candidates_Param_final(action_row_max(i,1),1),(rows+Circle_Cand
idates_Param_final(action_row_max(i,1),2)),Circle_Candidates_Param_final(action_row_ma
x(i,1),3),N,K);
   plot(xf,yf,'r');
   title(' Predicted Tree Ring(Red) With Most Confidence ');
   hold on;

end

% For analysis:

[sortedX,sortingIndices] =
sort(Prob_Action_Matrix(:,size(Prob_Action_Matrix,2)),'descend');

max_val=max(sortedX);
```

```
Tree_Ring_Num=0;

for val=1:size(sortedX,1);

    peak_val=sortedX(val,1);

    if peak_val > max_val*0.50; % Peaks below 50% of max value are rejected

        Tree_Ring_Num=Tree_Ring_Num+1;

    end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```