Circle detection using Hough Transform

Faraz Mazhar
BCSF14M529

Thursday, January 25, 2018

Table of Contents

Table of Figures

Introduction

This project explores the possible solution to the problem that is detection of approximate circles and circular objects of two types, ones with known radius and others with unknown radius. For the solution, a basic feature extracting technique in Digital Image Processing is used known as Circular Hough Transform.

A circle, on a plane, has two key components i.e. center and its radius.

Circular Hough Transform (CHT) is a specialization of Hough Transform which is employed to find circles imperfect input images. Candidate circles are produced with the help of finding the local maxima of votes in an accumulator matrix which is in Hough parameter space. [1]

Coming up in following sections contain review of the relevant literature, how everything was implemented, results of the implementation, and conclusion and findings of the report.

Literature Review

For the understanding and implementation, article which was the most helpful was "Circle Detection Using Hough Transforms Documentation" by Jaroslav Borovička from 2013 [2]. Another helpful literature was "A Method to Detect Circle Based on Hough Transform" by Mengji Wu *et al.*, from 2015 [3]

Other than above mentioned literatures, lecture slides of "Lecure 11: Line detection via Hough Transform" by Nazar Khan of PUCIT, and "Finding 2D Shapes and the Hough Transform" by Aaron Bobick of School of Interactive Computing were also studied.

Some of the helpful online articles and documentations were "Hough Circle Transform tutorial" in OpenCV documentation [4] and "Circle Hough Transform" on AI shack [5].

Implementation

Starting from scratch, there needs to be certain steps performed before getting on with the circle detection. The steps help to improve detection accuracy, quality and overall efficiency. Following is a list of essential steps were performed for circle detection [2]:

- Smoothing using Gaussian filtering
- Edge detection using Sobel operators
- Calculating magnitude and gradient of each pixel
- Thresholding to only keep desired edges
- Non-maxima suppression
- Accumulation into [a, b] space
- Finding local maxima for possible center points
- Accumulation into [r] space
- Marking detected circles on the input image (optional)

Following image will be used as input:

Figure 1 Input image (coins.png)

Smoothing

Script(s): my2DConvolution.m, my2DGaussian.m

Smoothing of the image is done by using Gaussian kernel of size 3x3 but user has an option to increase the size of the kernel. The purpose of the step is to reduce noise and Gaussian kernel is used to keep the edges as sharp as possible because of it's weight distribution i.e. high at near center points and low at near border points. Following formula is used to calculate 2D Gaussian kernel [6]:

$$Gaussain(i,j) = \frac{1}{(\sqrt{2\pi\sigma})^2}\, e^{-\frac{i^2+j^2}{2\sigma^2}}$$

The resultant kernel is divided by the sum of all of its elements and then convolved with the given image using following formula [7]:

$$h(k,l) = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f(i,j) * g(k-i, l-j)$$

Following is the smoothed image:

Figure 2 Smoothed input image

Sobel Edge Detection

Script(s): sobelFilter.m

After obtaining smoothed image, it's time to perform edge detection. Using Sobel edge detection provides an additional benefit of being able to calculate direction map. Importance of direction map will be explained later. For Sobel detection we need vertical and horizontal operators known as Sobel Operators. [8]

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Input image is convolved with *Gx* and *Gy* and then Euclidean norm is applied to obtain edge map. [2]

$$sobel_{merged}(i,j) = \sqrt{(sobel_{vertical}(i,j))^2 + (sobel_{horizontal}(i,j))^2}$$

For direction map, a built-in function 'atan2(.)' was used and this yield result in [-π, π] interval but for this implementation, I used 'atan(.)' which yields result in [-π/2, π/2] because Θ and Θ + π vote in same direction. [2] [9]

Edge map:

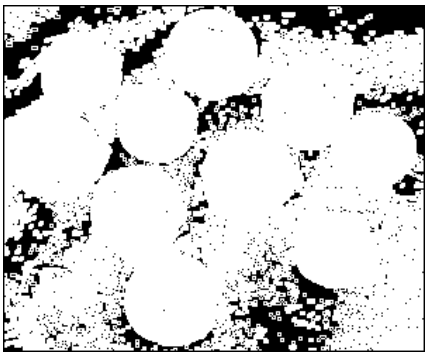

Figure 3 Edge map of the input image

Direction map:

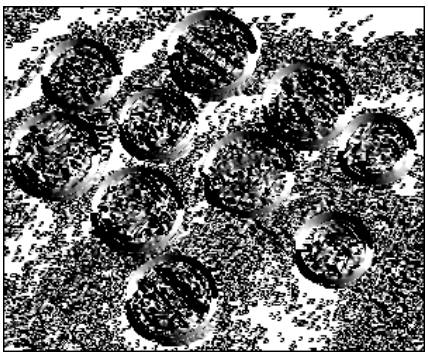

Figure 4 Direction map of the input image

Thresholding

Script(s): thresholding.m

Thresholding the edge map returns an image that has non-zero values where the edge magnitude is greater than the threshold and zeros everywhere else. Binary as an output also work just fine.

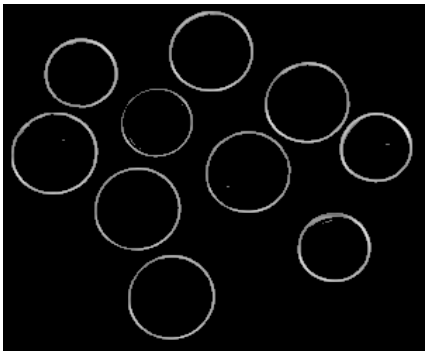

Figure 5 Threshold edge map

Non-maxima suppression

Function: `Inms = bwmorph(I, 'skel', Inf);`

Edges are required to have single pixel width so the local maxima at each pixel in width is kept and rest are suppressed. For this implementation, built-in MATLAB function was used. [10]

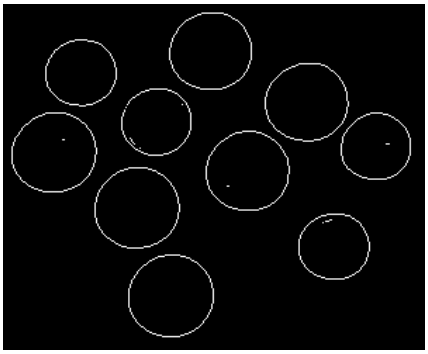

Figure 6 Non-maxima suppression on threshold edge map

Accumulation in [a, b] space

Script(s): accumulation.m

This is the first major step towards finding the circles in an image. As, stated earlier, a circle has two key components when it's in a plane i.e. center and radius. This step will help in discovering the center(s) of the circle(s) present in the image.

For a given circle of a radius 'r' and center at 'a' and 'b' and 'x' and 'y' being points on the edge of the circle, we can state the following:

$$x = a + r\sin\Theta$$

$$y = b + r\cos\Theta$$

So, reorganizing the equations, we get:

$$a = x - r\sin\Theta$$

$$b = y - r\cos\Theta$$

Now, these reorganized equations can be used to calculate the center point. Now, the benefit of calculating direction map can be stated. If it hadn't been calculated, Θ would have to be calculated for all 360° but since direction map has been calculated, direction of each point is already known. This will result in each point voting for just one direction and the point where all the votes meet would be the center of the circle. Following formula is used calculate:

$$A(a,b,r) = A(a,b,r) + \frac{1}{r}$$

The 1/r is essential as it will diminish votes as the distance from the point increases, this is helpful as not doing 1/r might result in votes from different point from different circles meet and generate a false center.

Accumulator matrix A(a, b, r) is used which is a 3D matrix but this can be implemented using 2D matrix with just using A(a, b) but it would be a trade off of accuracy. In this project, A(a, b, r) as well as A(a, b) was used.

Accumulator matrix in 2D:

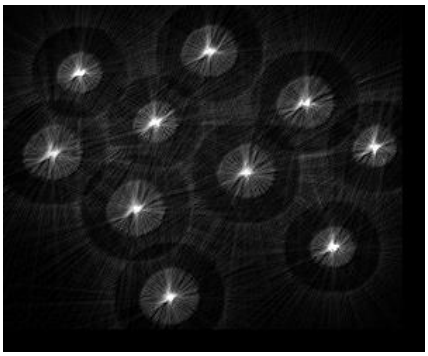

Figure 7 Accumulator matrix

Accumulator matrix threshold:

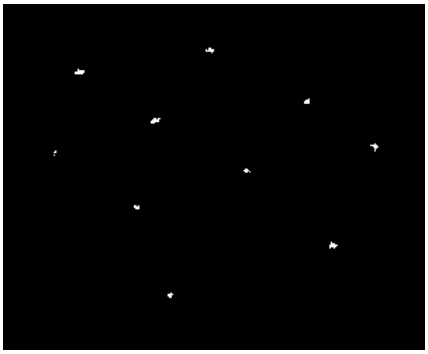

Figure 8 Threshold accumulator matrix

Local maxima for possible centers

Local maxima to find centers, for this built-in MATLAB function `imregionalmax(.)` was used there were some false centers were detected due to unknown reasons:

Figure 9 Possible centers of candidate circles

Accumulation in [r] space

Script(s): accumulationrspace.m

Now, that centers are known, these center points can be used to find the circles. Iterating from minimum radius to maximum radius, at each radius point, a vote is casted for that radius then the circle with votes more than threshold are considered as good circles. Equations discussed earlier will be used here:

$$x = a + r\sin\Theta$$

$$y = b + r\cos\Theta$$

For each center, votes are casted for each radius that lies between minimum radius and maximum radius. These votes are then accumulated. So, for all possible Θ at point (a, b), following is performed [2]:

$$R(r) = \sum Edge(x, y)$$

Finally, desired information is collected after thresholding R for good circles and so, centers and radius have been calculated. This information can be used to mark the detected circles.

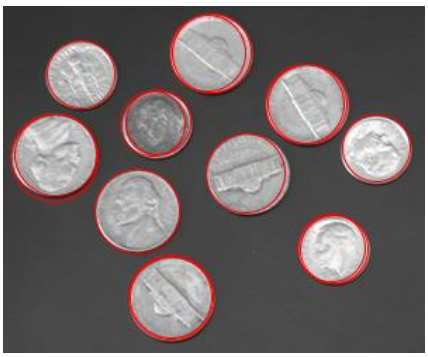

Figure 10 Marked detected circles

It should be noted that, for detection of a circle with known radius, giving 'main.m' same value for 'minr' and 'maxr' would work as intended.

Results

Following are the results on executing on a circle which basically ends being a disc and this script work on it as concentric circles.

Figure 11 circle.png Input image

Figure 12 circle.png smoothed

Figure 13 circle.png edged

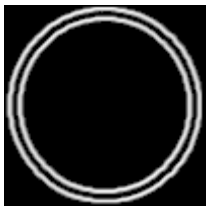

Figure 14 circle.png edged and thresholded

Figure 15 circle.png non maxima suppressed

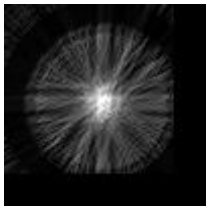

Figure 16 circle.png [a, b] space

Figure 17 circle.png [a, b] space thresholded

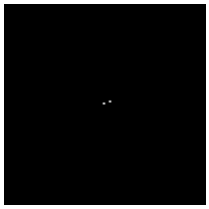

Figure 18 circle.png local maxima of [a, b] space

Figure 19 circle.png detected circles

Conclusion

To summarize, circle detection in imperfect images have been performed to a degree of accuracy. A basic set of steps were followed to achieve this goal of finding centers and radii of circles. This report explorers a way of achieving the goal via Hough Transform. Although, the project can be marked as a success, there are still a lot to be optimized and improved. Many issues with varying difficulties were face and for which some unorthodox solutions were implemented which did induce some uncertainties and inaccuracies.

Although, most of the concepts involved have been discussed in the MATLAB code, in its entirety, is present at the end of this report.

References

[1] "Circle Hough Transform," 17 November 2017. [Online]. Available: https://en.wikipedia.org/wiki/Circle_Hough_Transform.

[2] J. Borovička, "Circle Detection Using Hough Transforms," pp. 1-9, 4 March 2003.

[3] Z. S. B. L. F. L. B. L. C. S. Mengjie Wu, "A Method to Detect Circle based on Hough Transform," in *International Conference on Information Sciences, Machinery, Materials and Energy*, Xi'an, 2015.

[4] A. K. R. Alexander Mordvintsev, "Hough Circle Transform," 2013. [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html.

[5] U. Sinha, "Circle Hough Transform," 2017. [Online]. Available: http://www.aishack.in/tutorials/circle-hough-transform/.

[6] "Gaussian Blur," 19 January 2018. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian_blur#Mathematics.

[7] "Spatial Correlation and Convolution," in *Digial Image Processing 3rd ed.*, Pearson Education International, 2006, p. 171.

[8] "An Implementation of Sobel Edge Detection," [Online]. Available: https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection. [Accessed 2018 January 14].

[9] "atan," MATLAB, [Online]. Available: https://www.mathworks.com/help/matlab/ref/atan.html. [Accessed 2018 Janruay 14].

[10] "Skeleton," [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm.

Scripts

main.m

```
%% Circle detection using hough transfomration
%  * author: Faraz Mazhar, BCSF14M529
% ASSUMPTIONS:
%  * Images are in grayscale.

I = imread('circle.png');
[r, c, d] = size(I);

if (d ~= 1)
    I = rgb2gray(I);
end

GaussSize = 3; % Adjust this for the size of Guassian mask
threshold = 200; % Adjust threshold for Sobel edge detection here.
radiiThreshold = 200; % Adjust this threshold for radius calculation.

% Radii for unknown or set both with same value for known.
minr = 10;
maxr = 100;

% Connectivity for finding local maxima. [Valid: 1, 4, 6, 8, 18, and 26.]
conn = 26;

% Generating Gaussian Mask.
Mgauss = my2DGaussian(GaussSize);

% Convolving Guassian Mask for smoothing for noise reduction.
Iconv = my2DConvolution(I, Mgauss);

% Sobel filtering for edge detection.
[Isobel, gradient] = sobelFilter(Iconv);

% Thresholding.
Iedged = thresholding(Isobel, threshold);

% Skeletonization
Ithin = bwmorph(Iedged, 'skel', Inf);

% Accumulation in (a, b) space for unknown radius.
% Give same value to minr and maxr for known radius.
[A] = accumulation(minr, maxr, Ithin, gradient);

% Finding local maxima.
localMaxima = imregionalmax(A, conn);
Amax = max(max(A));
Aglobalmax = localMaxima;
localMaximaXY = localmaxXY(Aglobalmax);

% Accumulation in (r) space and then draws circles on the image.
[R, Irgb] = accumulationrspace(minr, maxr, Ithin, localMaximaXY, I,
radiiThreshold);

figure,imshow(Irgb);
```

my2DConvolution.m

```
%% This function convolves a given image.
%  * author: Faraz Mazhar, BCSF14M529
% ASSUMPTIONS:
%  * I is an image. (8-bit image)
%  * M is a mask. (2D Matrix / Normalized / 'n' x 'n' / 'n' is odd)

function [I] = my2DConvolution(I,M)
    [img_r, img_c] = size(I);
    [msk_r, msk_c] = size(M);
    msk_mid = (msk_r + 1)/2;

    for i = 1:img_r
        for j = 1:img_c
            conv = 0;

            for m = 1:msk_r
                img_r_msk = i-msk_mid+m;

                % Checking row borders.
                if img_r_msk < 1
                    % Mirroring rows on the left.
                    img_r_msk = img_r_msk + ((-2 * img_r_msk) + 1);
                elseif img_r_msk > img_r
                    % Mirroring rows on the right.
                    img_r_msk = img_r_msk - ((2 * (img_r_msk - img_r)) + 1);
                end

                for n = 1:msk_c
                    img_c_msk = j-msk_mid+n;

                    % Checking column borders.
                    if img_c_msk <= 0
                        % Mirroring columns on the top.
                        img_c_msk = img_c_msk + ((-2 * img_c_msk) + 1);
                    elseif img_c_msk > img_c
                        % Mirroring rows on the right.
                        img_c_msk = img_c_msk - ((2 * (img_c_msk - img_c)) +
1);
                    end

                    % Applying mask on the apt pixels.
                    conv = conv + (I(img_r_msk, img_c_msk) * M(msk_r-m+1,
msk_c-n+1));
                end
            end

            I(i, j) = conv;
        end
    end
```

my2DGaussian.m

```
%% This function generates a 2D gaussian mask of given size.
%  * author: Faraz Mazhar, BCSF14M529
% ASSUMPTIONS:
%  * None.

function [M] = my2DGaussian(S)
    M = zeros(S, S);
    M_mid = (S+1)/2;
    SD = sqrt (-(S * S) / (2 * log10(1.0 / 255.0)));

    for i = 1:S
        for j = 1:S

            % Figuring out weight of the specific coordinate.
            x = i - M_mid;
            y = j - M_mid;

            M(i,j) = (1/(2 * pi * SD^2)) * exp(-((x^2 + y^2)/(2 * SD^2)));
        end
    end
    M = M/sum(M(:)); % Normalization.
```

sobelFilter.m

```
%% This function implements Sobel filters on an image by convolution.
%  * author: Faraz Mazhar, BCSF14M529
%  * source link: https://angeljohnsy.blogspot.com/2013/07/sobel-edge-
detection-part-2.html
% ASSUMPTIONS:
%  * Image is grayscale.
%  * Image is smoothed or has insignificant ammont of noise.

function [sobelmerged, gradient] = sobelFilter(I)
    Gx = [1, 0, -1; 2, 0, -2; 1, 0, -1]; % Vertical Sobel filter mask.
    Gy = [1, 2, 1; 0, 0, 0; -1, -2, -1]; % Horizontal Sobel filter mask.

    I = double(I);
    sobelmerged = zeros(size(I));
    gradient = zeros(size(I));

    % Convolution for magnitude and gradient.
    for i=1:size(I,1)-2
        for j=1:size(I,2)-2
            % Gradient operations
            Ix=sum(sum(Gx.*I(i:i+2,j:j+2)));
            Iy=sum(sum(Gy.*I(i:i+2,j:j+2)));

            % Magnitude of vector
            sobelmerged(i+1,j+1)=sqrt(Ix.^2+Iy.^2);
            % Direction map
            gradient(i+1,j+1) = atan(Ix/Iy);
        end
    end
end
```

thresholding.m

```
%% This function will threshold am image.
%  * author: Faraz Mazhar, BCSF14M529
%  * source link: https://angeljohnsy.blogspot.com/2013/07/sobel-edge-detection-part-2.html
% ASSUMPTIONS:
% * None.

function [Ithreshold] = thresholding(I, threshold)
    Ithreshold = max(I,threshold);
    Ithreshold(Ithreshold==round(threshold))=0;

%     Ithreshold = im2bw(Ithreshold);
    Ithreshold = mat2gray(Ithreshold);
end
```

lapOfGauss.m

```
%% This function calculates and applies Laplacian of Guassian.
%  * author: Faraz Mazhar, BCSF14M529
% ASSUMPTIONS:
%  * Size is odd.

function [Imexican] = lapOfGauss(I, S)
    LoG = zeros(S, S);

    SD = 1.4;

    Imexican = zeros(size(I));

    for x = 1:S
        for y = 1:S
            LoG(x, y) = - (1/(pi * SD^4)) * (1 - (x^2 + y^2)/(2 * SD^2)) * exp(-(x^2 + y^2)/(2 * SD^2));
        end
    end

    LoG = LoG/(sum(sum(LoG)));

    for i=1:size(I,1)-(S-1)
        for j=1:size(I,2)-(S-1)
            Imexican(i, j)=sum(sum(LoG.*I(i:i+(S-1),j:j+(S-1))));
        end
    end
end
```

accumulation.m

```
%% This function will perform accumulation in [a, b] space.
%  * author: Faraz Mazhar, BCSF14M529
% ASSUMPTIONS:
%  * None.

function [A] = accumulation(minr, maxr, magnitude, gradient)
    [rows, cols, ~] = size(magnitude);

    A = zeros(rows, cols);
%     A3 = zeros(rows, cols, maxr);

    for row = 1:rows
        for col = 1:cols
            for r = minr:maxr
                a = row - (r * cos(gradient(row, col)));
                b = col - (r * sin(gradient(row, col)));
                c = row + (r * cos(gradient(row, col)));
                d = col + (r * sin(gradient(row, col)));

                a = ceil(a);
                b = ceil(b);
                c = ceil(c);
                d = ceil(d);

                if (a > 0 && a <= rows && b > 0 && b <= cols)
                    A(a, b) = A(a, b) + (magnitude(row, col)/r);
%                     A3(a, b, r) = A(a, b, r) + (magnitude(row, col)/r);
                end
                if (c > 0 && c <= rows && d > 0 && d <= cols)
                    A(c, d) = A(c, d) + (magnitude(row, col)/r);
%                     A3(c, d, r) = A(c, d, r) + (magnitude(row, col)/r);
                end
            end
        end
    end
    figure, imshow(A);
    A = lapOfGauss(A, 17);
    A(A < 1) = 0;
end
```

accumulationinrspace.m

```
%% This function will perform accumulation in [r] space.
%  * author: Faraz Mazhar, BCSF14M529
% ASSUMPTIONS:
%  * None.

function [R, Irgb] = accumulationrspace(minr, maxr, Ithin, localMaximaXY, I,
radiiThreshold)
    [LMX, ~, ~] = size(localMaximaXY);

    Irgb = I;

    for lmX = 1:LMX
            R = zeros(maxr);

        for r = minr:maxr
            for theta = 1:360
                thetaRad = deg2rad(theta);
                x = localMaximaXY(lmX, 1) + r * cos(thetaRad);
                y = localMaximaXY(lmX, 2) + r * sin(thetaRad);

                x = ceil(x);
                y = ceil(y);


                if (x > 0 && x <= size(Ithin,1) && y > 0 && y <=
size(Ithin,2))
                    R(r) = R(r) + Ithin(x, y);
                enda
            end

            if (R(r) > radiiThreshold)
               rX = localMaximaXY(lmX, 1);
               rY = localMaximaXY(lmX, 2);

               Irgb = insertShape(Irgb, 'circle', [rY rX r], 'color', 'red');
            end
        end
    end
end
```