# Table of Contents



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

# taking image

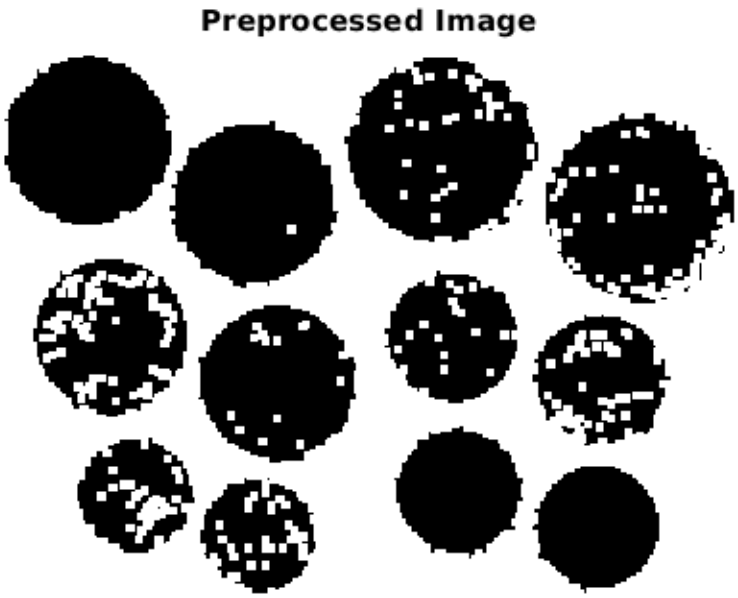
```
img = imread("coin 2.jfif");
copy_image = img;
%this image is color image

%taking size of image in the variables
dim = size(img);
rows = dim(1);
cols = dim(2);

% converting into greyscale image
img = rgb2gray(img);
imshow(img);
title('Original image Converted To GreyScale Image')
```

Original image Converted To GreyScale Image

# pre-processing the image

```
%pre-processing is done to remove the stamps/marks.figures on the coin
%which may lead to the false edges of the coin
th = 245;
for i = 1 : rows
    for j = 1 : cols
        if(img(i, j) > th)
            img(i, j) = 255;
        else
            img(i, j) = 0;
        end
    end
end

%mean filter
n = 3; %size of the kernel
k = (n - 1) / 2;
%my kernel for mean filter
kernel = [1,1,1;1,1,1;1,1,1];
kernel = kernel / 9;
% convolution
for ci = 1 : rows
    for cj = 1 : cols
        csum = 0;
        for i = ci - k : ci + k + 0.5
            for j = cj - k : cj + k + 0.5
                % as good as origianl img padded with zeros
                if(i < 1 || i > rows || j < 1 || j > cols)
```

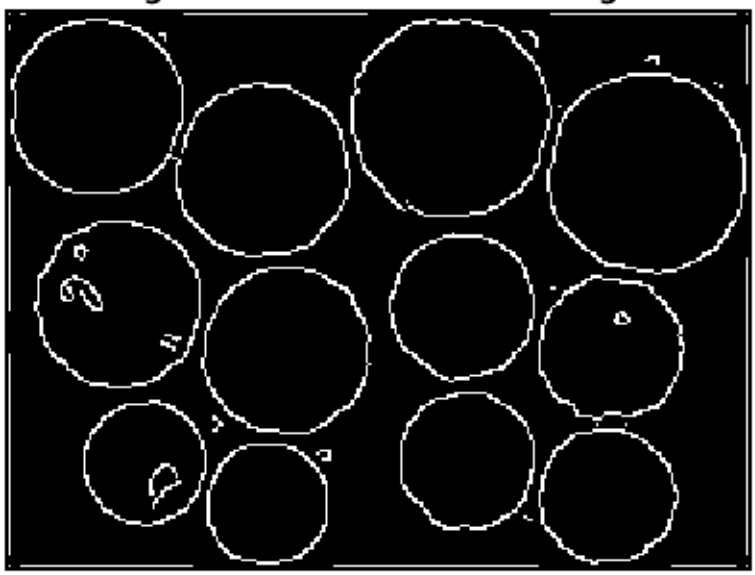
```
                    csum = csum + 0;
                else
                    csum = csum + double(img(i, j)) * kernel(i - ci +
 k + 1, j - cj + k + 1);
                end
            end
        end
        if csum > 255
            timg(ci, cj) = 255;
        elseif csum < 0
            timg(ci, cj) = 0;
        else
            timg(ci, cj) = csum;
        end
    end
end

figure();
imshow(timg);
title('Preprocessed Image')
```




# edge detection

```
%canny edge detection on the pre-processed image
e = edge(timg, 'canny');
figure();
imshow(e);
title('Edge Detection of the Coin-Edges')
```

Edge Detection of the Coin-Edges



# getting circular hough transform

```
%checking for coin radii ranging from 15 to 40 pixels
radii = 15 : 1 : 40;
%returns the pixels where edge points are present
[edgePtX, edgePtY] = find(e);
%create the transformed space (rows*cols) for each radius value
space = zeros(rows, cols, length(radii), 'uint8');
tempR = zeros(1, 0);
tempC = zeros(1, 0);
tempRad = zeros(1, 0);
for i = 1 : length(radii)
    %get the circle points of the circle with given radii centered at
 (0,0)
    [tR, tC] = circlepoints(radii(i));
    tempR = [tempR tR]; %concat the temp indices (centered at 0,0)
    tempC = [tempC tC]; %concat the temp indices (centered at 0.0)
    tempRad = [tempRad repmat(i, 1, length(tR))]; %concat the
 correspoinding radius for each point
end
%loop over all edge points and move the circle of each radius to the
%edge-point center and increment the count in the HT space
for i = 1 : length(edgePtX)
    for j = 1 : length(tempR)
        %shift the center of the circle to the edge point
        x = -tempC(j) + edgePtX(i);
        y = tempR(j) + edgePtY(i);
        %increment the count in the HT space
        if(x > 0 && x < rows && y > 0 && y < cols)
```

```
            space(x, y, tempRad(j)) = space(x, y, tempRad(j)) + 1;
        end
    end
end

%divide the count in each radius plane by its radius for normalization
%as circles with greater radius have more count at the center
for i = 1 : length(radii)
    space(:, :, i) = space(:, :, i) ./ radii(i);
end
```

# peak detection

```
%if a peak is found at particular location then dont find another peak
 in
%the neighbour(nbr) of that peak, limits of the 'nbr' are below
nbrXY = 15; nbrR = 21;
%on one side of the point, vector will be
nbrSide = ([nbrXY nbrXY nbrR] - 1) / 2;
%specify the number of circles to detect, if unknown initialize to 0
maxpks = 0;
%threshold for the count for detection of center of the circle
%we can tune its value
threshold = 0.5 * max(space(:));

if (maxpks == 0)
    peaks = zeros(3, round(numel(space)/100));  % preallocate
else
    peaks = zeros(3, maxpks);  % preallocate
end

np = 0;

while true
        %get the location and value of global maximum of a 3D array
        [r, c, k, v] = max3(space); %(row, column, radii(k), value)
        % stop if peak height below threshold
        if v < threshold || v == 0
            break;
        end
        np = np + 1;
        peaks(:, np) = [c; r; radii(k)]; %putting values of x, y, k
        % stop if done enough peaks
        if np == maxpks
            break;
        end
        % suppress the peaks in the neighbourhood
        r0 = max([1 1 1], [r c k]-nbrSide);
        r1 = min(size(space), [r c k]+nbrSide);
        space(r0(1):r1(1), r0(2):r1(2), r0(3):r1(3)) = 0;
end
peaks(:, np+1:end) = [];   % trim
```

# plot

```
%plots the detected the circles on the actual image
figure();
%used inbuilt function to insert the detected circle on the image
img = insertShape(copy_image, "Circle", peaks', "Color", "red");
imshow(img);
title('Detected Circles on the Original Image')
```


# print the output

```
ncoins = size(peaks, 2);
msg = ['number of coins :',num2str(ncoins)];
disp(msg);
disp('location and radius of each coin is given below in the format
 (x, y, radius)');
disp(peaks');
```

```
number of coins :12
location and radius of each coin is given below in the format (x, y,
 radius)
    49   157    21
    92   171    21
   161   156    23
   209   169    23
   159   103    25
    32    34    30
    90    56    30
   210   118    24
```

```
    40   102    28
    98   118    28
   155    37    34
   223    57    34
```

# function max3

```
function [r, c, k, v] = max3(h)
% location and value of global maximum of a 3D array
[vr, r] = max(h);
[vc, c] = max(vr);
[v, k] = max(vc);
c = c(1, 1, k);
r = r(1, c, k);
end
```

*Published with MATLAB® R2020a*