Lab 3

Albert Segarra Roca (S3255050), Carlos Humberto Paz Rodríguez (S3040577)

Group 16

Pattern Recognition

FMNS • RUG

November 23, 2017

1 Assignment 1

1.1 Exercise 1

For this exercise we are using the cameraman.tif image (Figure 1), first we calculate the edges of the image using the Canny's algorithm, because it is the most used algorithm and most of the times has the best results detecting edges of images than other algorithms, we use the default values of threshold for this exercise. The threshold value of the Canny's algorithm let's us decide the level of detail or resolution we want to use on the transformation, if we use a low threshold value, we would detect too many edges that sometimes are caused by noise in the image, on the contrary, if we use a high value, we may end up missing some edges on the image.

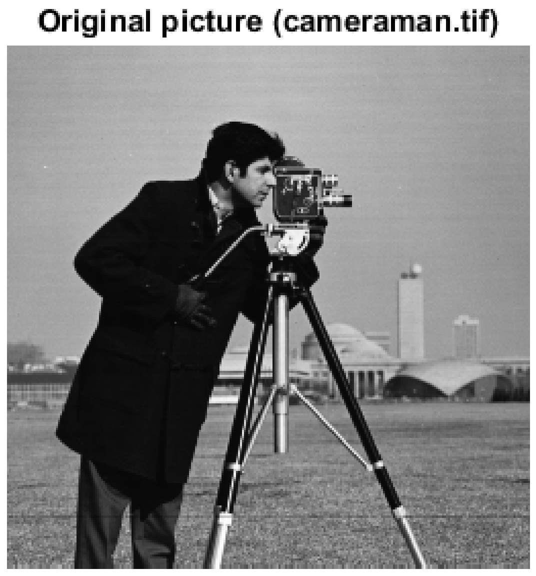

Figure 1: Original image (cameraman.tif)

As we can see on Figure 2, the canny's algorithm was able to detect most of the edges of the distinct elements.

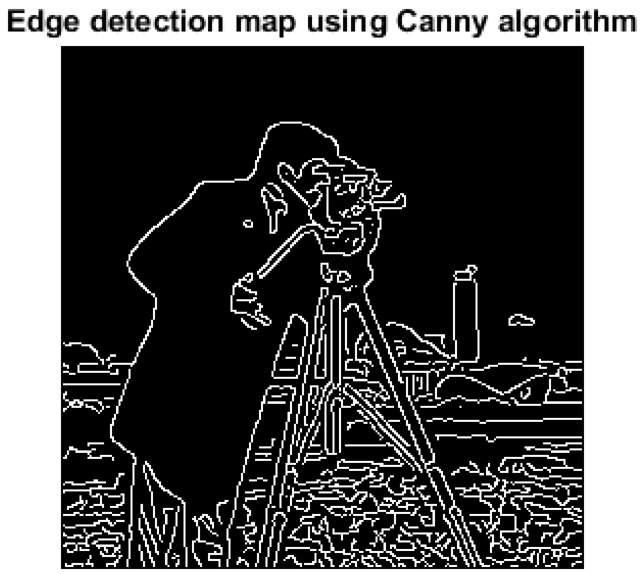

Figure 2: Image after applying the edge detection algorithm with default values (Canny's algorithm)

Then we use the Hough's transformation and plot the data using the equation:

$$\rho = xcos\theta + ysin\theta \tag{1}$$

In our exercise the local maximum value is 110, this tells us that 100 different points form the same and longest line detected on the image.

The peak threshold is the minimum amount of points that we want to have to consider the detection of a line.

In our exercise we used a proportion of 83.5% times of the highest value, so we can detect the five longest lines.

On the Figure 3, we can see the local maxima array and the strongest five points detected, this way we can go back to the cartesian space, and plot the five longest detected lines above the initial image,(Figure 4).

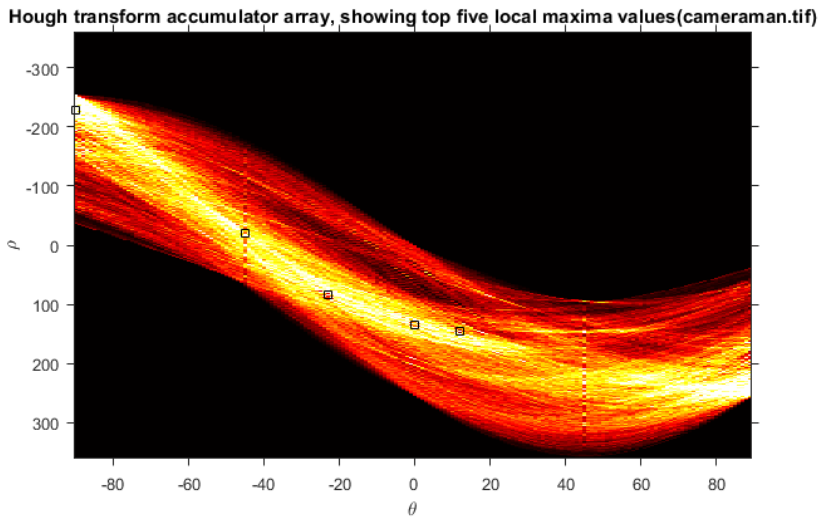

Figure 3: Hough's accumulator array

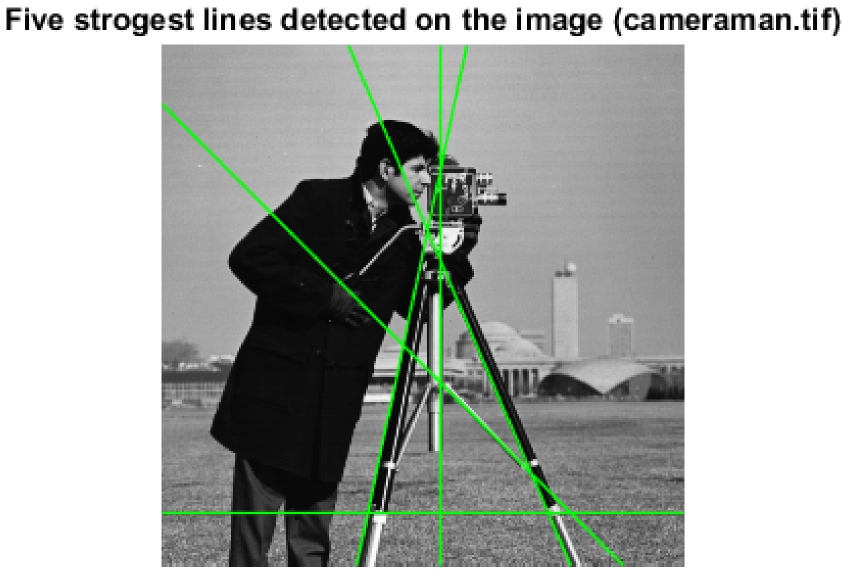

Figure 4: Initial image with overlaid lines, of the top five longest lines detected

The houghlines() function returns a structure with the initial and end point, also with the ρ and θ values of each line, so we can design our own function to plot a line over an image (view script 5.2), the strongest line detected is shown on the Figure 5.

The code is available at the script number 5.1

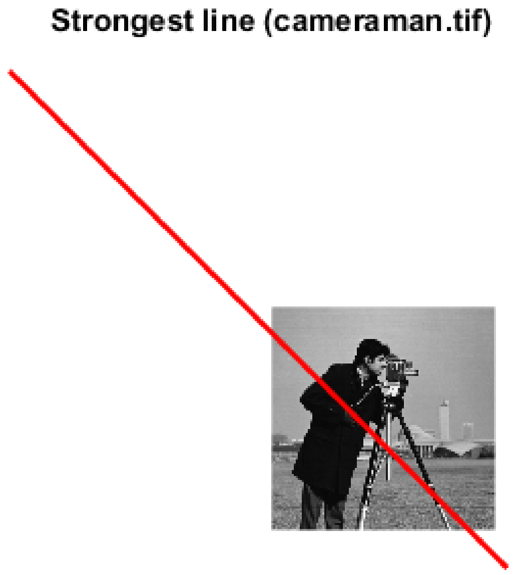

Figure 5: Strongest line detected

2 Assignment 2

2.1 Exercise 1

If we want to restrict p to positive values, then we can use the range $0 < \theta \leq 2\pi$, which ensures a positive p because it covers also the top-right quadrant (Figure 1 in the statement), allowing lines crossing this quadrant to have positive p. p must then have the range $0 \leq p \leq d$, where d is the diagonal size of the image. We can discretize this ranges by considering a certain threshold ϵ, $0 < \epsilon < 2\pi$ for and consider all θ values such that $\theta = k\epsilon, \ k \in \mathbb{Z}$.

2.2 Exercise 2

The implementation of the algorithm can be found in Section 5.8.

3 Assignment 3

3.1 Exercise 1

We created a 50x50 black image with one white pixel (figure 6) and calculated its hough transformation (figure 7). The code is available at script number 5.2

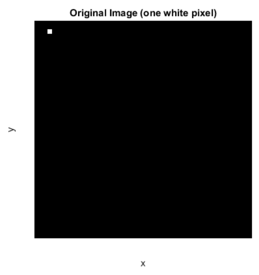

Figure 6: Original Image (one white pixel)

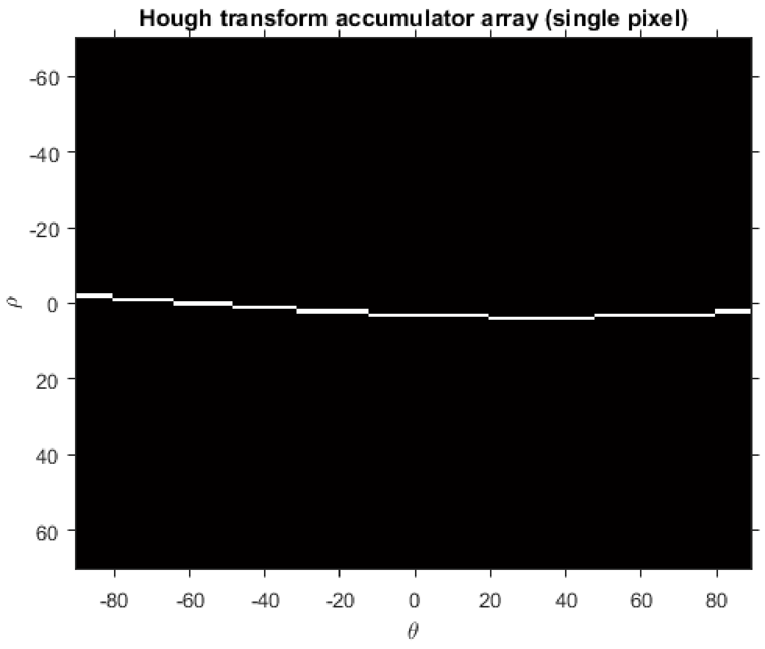

Figure 7: Hough map, one pixel

3.2 Exercise 2

Then we created another 50x50 image with three unalligned points (figure 8) and calculated its hough transformation (figure 9). The code is available at script number 5.3

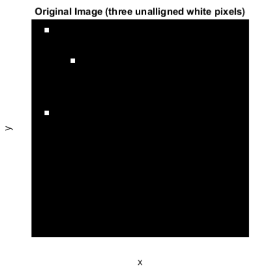

Figure 8: Original Image (three unalligned pixels)

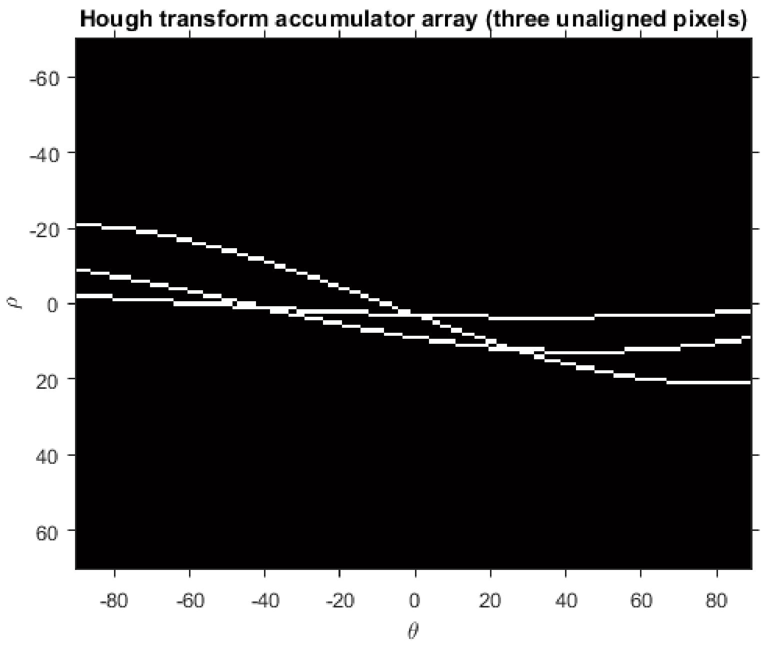

Figure 9: Hough map (three unalligned pixels)

3.3 Exercise 3

Finally we created another 50x50 image with three alligned points (figure 10) and calculated its hough transformation (figure 11). The code is available at script number 5.4

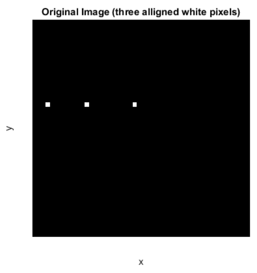

Figure 10: Original Image (three alligned pixels)

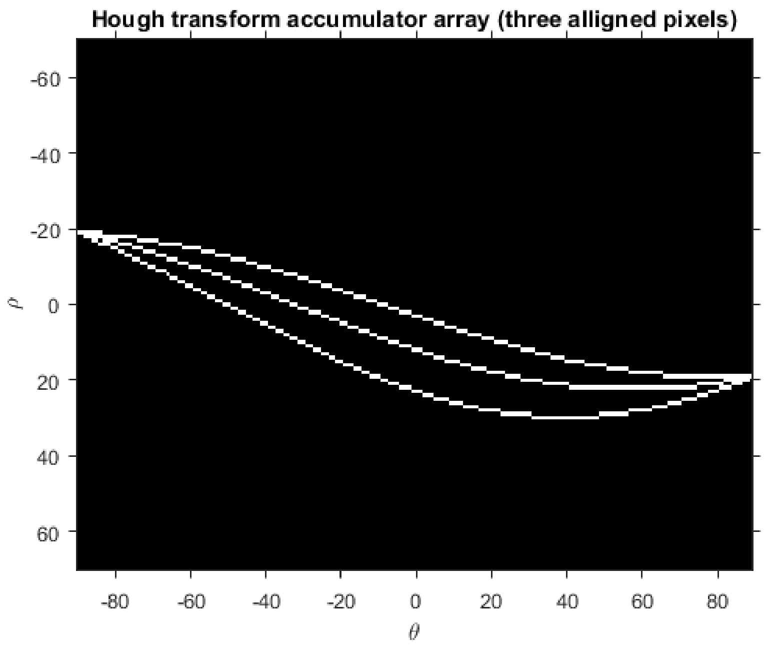

Figure 11: Hough map (three alligned pixels)

3.4 Exercise 4

We can conclude from this experiment that when the three lines intersect on the hough transformation map, it means the three lines are aligned, when they are not, even for a small amount, they wouldn't intersect on the same place in the hough map. We can also observe that on the second scenario, each line would cross only with one other at a specific θ value, which means we could draw a line only between two points, and also that there are many different combination options.

3.5 Exercise 5-6

If we calculate the local maxima of the image with three pixels aligned (figure 12), we can see that the local maxima is the point where the three curves intersect, we can also plot this line on the initial image and we see that the three dots connect perfectly (figure 13). The code is available at script number 5.5

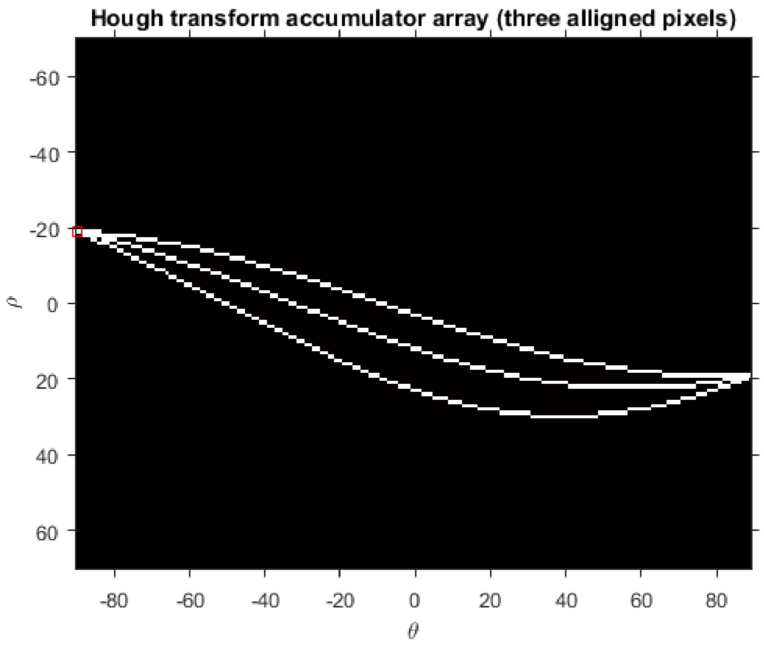

Figure 12: Hough map (three alligned pixels), with local maxima

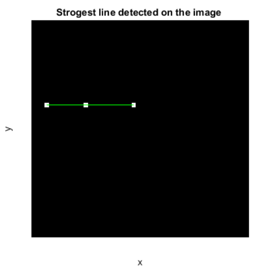

Figure 13: Initial image with the line connecting all the points

3.6 Exercise 7-8

Now we are going to use an external image that we load to matlab first, then we use the same procedure and calculate the fifteen strongest lines, we needed to adjust the threshold of the peaks detector, to half the value of the biggest number, in this case because most of the lines have the same length. On figure 14, we can see the first fifteen local maxima values and we finally plot them on the figure 15, we can learn from this exercise that when many lines have the same value, a high threshold value excludes many lines. The code is available at script number 5.6

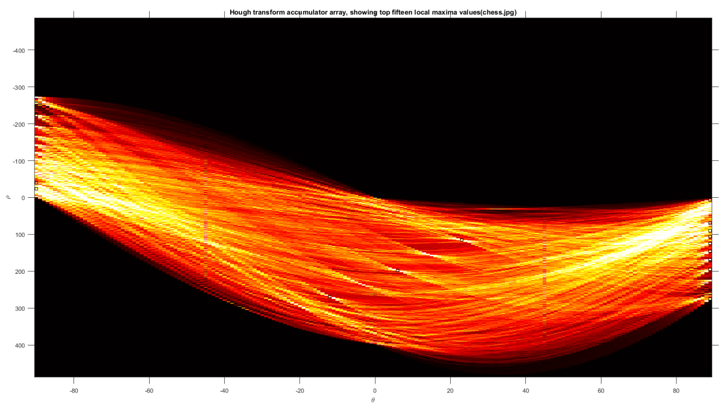

Figure 14: Hough map fifteen highest values

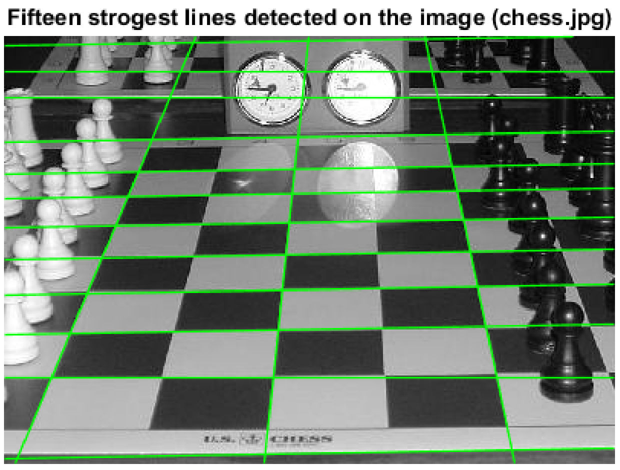

Figure 15: Initial image with the lines plotted over

4 Assignment 4

The script implementing the steps described in the statement can be found in Section 5.9. As you can see, we have used the `imfindcircles` method to compute the centers and radius of the circles in the enhanced image. We've used a radius range $20 \leq r \leq 40$ and a sensitivity of 0.897967, which has been determined by performing a sort of bisection algorithm on the number of circles found.

In order to delete possible duplicated circles for the same screw, we perform a filtering which searches for pairs of intersecting circles and deletes one of them. Note that they are not deleted during the iteration (which would mess up the indices and possibly lead to an out of range access), but instead they are deleted afterwards.

To get the two strongest circles we just need to access the two first elements of each of the arrays (`centers, radii`), because they are already sorted decreasingly by strength.

As we can see in Figure 16, we obtain 6 circles, which are highlighted in red. In Figure 17 we can see the two strongest circles detected by the algorithm. This is consistent with the fact that they are the ones which are facing best to the camera, and thus have a much more clear circular shape and contrast that eases the detection by the algorithm.

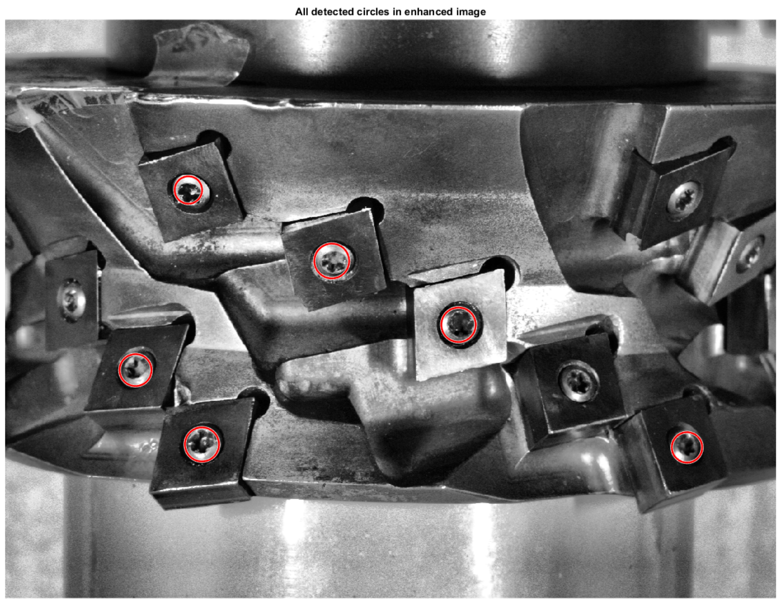

Figure 16: All detected circles in enhanced image

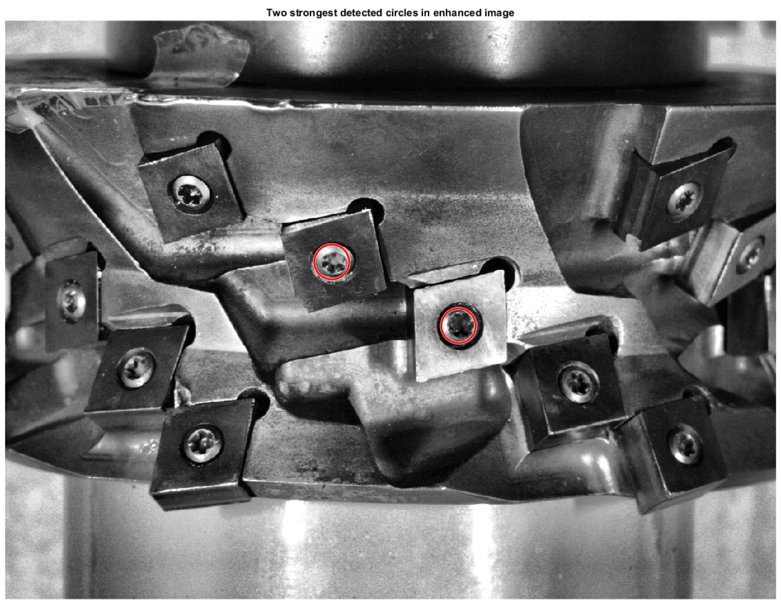

Figure 17: Two strongest detected circles in enhanced image

5 Appendix

5.1 Script for loading and calculating the five longest lines of the sample image (cameraman.tif)

```matlab
function hough1()
    % Load and display the Cameraman.tiff image
    BW = imread('Cameraman.tiff');
     figure
     imshow(BW);
     title('Original picture (Cameraman.tiff)');

    %Calculate the edges using canny's algorithm
     trans = edge(BW,'canny');
     figure
     imshow(trans);
     title('Edge detection map using Canny algorithm');

    %Generate the accumulator array to find the local maxima values
     [H,T,R] = myhough(trans);
      figure
      imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,...
             'InitialMagnification','fit');
      title('Hough transform accumulator array, showing top five local maxima values(
          cameraman.tif)');
      xlabel('\theta'), ylabel('\rho');
      axis on, axis normal, hold on;
      colormap(gca,hot);

      %Calculate the local maxima value
      disp(max(H(:)))

      %Top five
      %Plot the maxima value on the accumulator array map
      P = houghpeaks(H,5,'threshold',ceil(0.835*max(H(:))));
      x = T(P(:,2));
      y = R(P(:,1));
      plot(x,y,'s','color','black');

      %Auto detect lines on the image, using
      lines = houghlines(BW,T,R,P,'FillGap',100,'MinLength',80);

      %DISPLAY ORIGINAL IMAGE WITH LINES %SUPERIMPOSED
      figure, imshow(BW), hold on
      max_len = 0;
      for k = 1:length(lines)
          xy = [lines(k).point1; lines(k).point2];
          plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','green');
      end
      title('Five strogest lines detected on the image (cameraman.tif)');

      %Highest value
      %We can use our own function to plot the strongest line
      myhoughline('Cameraman.tiff', lines(1).rho, lines(1).theta);
      title('Strongest line (Cameraman.tiff)');
end
```

5.2 Script for ploting a line over an image, given the initial image, theta and rho values

```
function myhoughline(img,rho, theta)
    figure
   BW = imread(img);
    imshow(BW);
    title('Original picture');
    hold on;

    theta=degtorad(theta);
   %Having the initial data in polar coordinates, we convert them to
   %cartesian values
    [x,y]=pol2cart(theta, rho);

   %Considering the initial point is the center (0,0) we calculate the
   %slope of the rho line using the two points
   m = (y)/(x);

   %We calculate the perpendicular slope
   m2=(-1/m);

   %Given that we know a point in the perpendicular line, we calculate
   %its b value
    b2=y-m2*x;

    %Now  we generate a number of x values and plot the line
    xx=-500:10:500;
    yy=m2*xx+b2;
    plot(xx,yy,'Color','r','LineWidth',2)

    hold off;
    axis([-300,300,-300,300])
end
```

5.3 Script to create a 50x50 black image with one white pixel and calculate its hough map

```
function image31()
    %Create a black image with one white pixel
    img = false(50,50);
    img(3:3,4:4) = true;

    %# show the image
    figure,imshow(img)
    title('Original Image (one white pixel)')
    xlabel('x'), ylabel('y');

    %Generate the accumulator array to find the local maxima values
    [H,T,R] = hough(img);
     figure
     imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,...
           'InitialMagnification','fit');
     title('Hough transform accumulator array (single pixel)');
     xlabel('\theta'), ylabel('\rho');
     axis on, axis normal, hold on;
     colormap(gca,hot);
end
```

5.4 Script to create a 50x50 black image with three white pixels unalligned and calculate its hough map

```
function image32()
    %Create a black image with one white pixel
    img = false(50,50);
    img(3:3,4:4) = true;
    img(10:10,10:10) = true;
    img(22:22,4:4) = true;

    %# show the image
    figure,imshow(img)
    title('Original Image (three unalligned white pixels)')
    xlabel('x'), ylabel('y');

    %Generate the accumulator array to find the local maxima values
    [H,T,R] = hough(img);
     figure
     imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,...
           'InitialMagnification','fit');
     title('Hough transform accumulator array (three unaligned pixels)');
     xlabel('\theta'), ylabel('\rho');
     axis on, axis normal, hold on;
    colormap(gca,hot);
end
```

5.5 Script to create a 50x50 black image with three white pixels alligned and calculate its hough map

```
function image33()
    %Create a black image with one white pixel
    img = false(50,50);
    img(20:20,4:4) = true;
    img(20:20,13:13) = true;
    img(20:20,24:24) = true;

    %# show the image
    figure,imshow(img)
    title('Original Image (three alligned white pixels)');
    xlabel('x'), ylabel('y');

    %Generate the accumulator array to find the local maxima values
    [H,T,R] = hough(img);
     figure
     imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,...
           'InitialMagnification','fit');
     title('Hough transform accumulator array (three alligned pixels)');
     xlabel('\theta'), ylabel('\rho');
     axis on, axis normal, hold on;
     colormap(gca,hot);
end
```

5.6 Script to create a 50x50 black image with three white pixels alligned and calculate its hough map, detecting the local maxima, and printing the line on the initial image

```
function image36()
    %Create a black image with one white pixel
    img = false(50,50);
    img(20:20,4:4) = true;
    img(20:20,13:13) = true;
    img(20:20,24:24) = true;

    %Generate the accumulator array to find the local maxima values
    [H,T,R] = hough(img);
     figure
     imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,...
           'InitialMagnification','fit');
     title('Hough transform accumulator array (three alligned pixels)');
     xlabel('\theta'), ylabel('\rho');
     axis on, axis normal, hold on;
     colormap(gca,hot);

     %Plot the maxima value on the accumulator array map
     P = houghpeaks(H,1,'threshold',ceil(0.1*max(H(:))));
     x = T(P(:,2));
     y = R(P(:,1));
     plot(x,y,'s','color','red');
     hold off

     %Auto detect lines on the image, using
     lines = houghlines(img,T,R,P,'FillGap',100,'MinLength',5);

     %DISPLAY ORIGINAL IMAGE WITH LINES %SUPERIMPOSED
     figure , imshow(img), hold on
     max_len = 0;
     for k = 1:length(lines)
         xy = [lines(k).point1; lines(k).point2];
         plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','green');
     end
     title('Strogest line detected on the image ');
     xlabel('x'), ylabel('y');
end
```

5.7 Script to load an chessboard picture, calculate its hough map, detecting the local maxima, and printing the strongest fifteen lines on the initial image

```
function image37()
    BW = imread('chess.jpg');
    BW=rgb2gray(BW);

    %Calculate the edges using canny's algorithm
    trans = edge(BW,'canny');

    %Generate the accumulator array to find the local maxima values
    [H,T,R] = hough(trans);
     figure
     imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,...
           'InitialMagnification','fit');
     title('Hough transform accumulator array, showing top fifteen local maxima values(chess
         .jpg)');
     xlabel('\theta'), ylabel('\rho');
     axis on, axis normal, hold on;
     colormap(gca,hot);

    %Calculate the local maxima value
    disp(max(H(:)))

    %Top fifteen
    %Plot the maxima value on the accumulator array map
    P = houghpeaks(H,15,'threshold',ceil(0.5*max(H(:))));
    x = T(P(:,2));
    y = R(P(:,1));
    plot(x,y,'s','color','black');

    %Auto detect lines on the image, using
    lines = houghlines(BW,T,R,P,'FillGap',100,'MinLength',80);

    %DISPLAY ORIGINAL IMAGE WITH LINES %SUPERIMPOSED
    figure , imshow(BW), hold on
    max_len = 0;
    for k = 1:length(lines)
        xy = [lines(k).point1; lines(k).point2];
        plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','green');
    end
    title('Fifteen strogest lines detected on the image (chess.jpg)');

end
```

5.8 Script for myhough

```
function acc_array = myhough(edge_map)
    % Step a) Determine values of theta
    EPSILON = 0.05; % This determines the size of each theta step for the discretization
    theta = EPSILON : EPSILON : 2*pi; % Notice theta 0 < theta <= 2*pi

    % Step b) Find the foreground pixels (those with value 1, black)
    [y, x] = find(edge_map); % Notice column index is y axis position, row index is x axis

    % Step c) Calculate all values of p rounding to the nearest integer less or equal
    p = floor([x, y]*[cos(theta); sin(theta)]);

    % Step d) Initialize 2D accum array
    acc_array = zeros(size(theta(:)), max(p(:)) + 1);

    % Step e) Fill accumulator array
    curr_theta_ind = 1;
    for theta_column = p
      for v = theta_column(1,:)
        if v >= 0
          ++acc_array(curr_theta_ind, v + 1); % Need to add 1 for 0 values
        end
      end
      ++curr_theta_ind;
    end
```

5.9 Script for circle Hough transform

```
function houghcircles()
   % Step 1: Read image & convert to double precision
   BM = im2double(imread('HeadTool0002.bmp'));

   % Step 2: Apply contrast-limited adaptative histogram equalization
   BMEQ = adapthisteq(BM);

   % Step 3: Find 6 circles keeping only one for each screw
   [centers, radii, ~] = imfindcircles(BMEQ, [20 40], 'Sensitivity', 0.89767);

   % Remove circles belonging to the same screw by finding circles that intersect
   % and removing one of the pair.
   N = size(centers);
   deleteIndexList = [];
   for i = 1:N
     for j = i+1:N
        x1 = centers(i,1);
        y1 = centers(i,2);
        r1 = radii(i);
        x2 = centers(j,1);
        y2 = centers(j,2);
        r2 = radii(j);

        if pdist([x1 y1; x2 y2]) - r1 - r2 <= 0 % Check if they intersect
            deleteIndexList = [deleteIndexList, j]; % If they intersect, add to delete list
        end
     end
   end

   % We need to delete them later because otherwise we would mess with the
   % iteration
```

```
    deleteIndexList = unique(deleteIndexList);
    centers(deleteIndexList, :) = [];
    radii(deleteIndexList) = [];

    % Step 4: Show enhanced image with circles
    figure
    imshow(BMEQ);
    title('All detected circles in enhanced image');
    viscircles(centers, radii);

    % Step 5: Show enhanced image with two strongest circles
    figure
    imshow(BMEQ);
    title('Two strongest detected circles in enhanced image');
    % Already sorted decreasingly by the strength
    strongest2CirclesCenters = centers(1:2, :);
    strongest2CirclesRadii = radii(1:2);
    viscircles(strongest2CirclesCenters, strongest2CirclesRadii);
```