AMME4710 Tutorial 4 - Image Features

Alan KHOURY - 310208408 - akho7890@uni.sydney.edu.au

Anirban GHOSE - 311193218 - agho8615@uni.sydney.edu.au

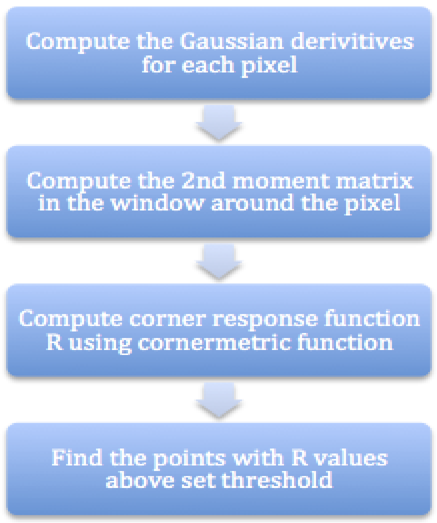
Michael HOLMES - 311246044 - mhol7078@uni.sydney.edu.au

Q1) Harris Corner Detector

AIM:
Our aim in this question is to investigate the effectiveness of the Harris Corner Detector (based upon the Harris-Stephens corner model/algorithm) on a variety of images exhibiting varying amounts of contrast, resolution and image complexity. Corner detectors such as the Harris variant rely on pixel intensity variation in multiple directions for the classification of strong corners, invariant to translation and rotation of the image.

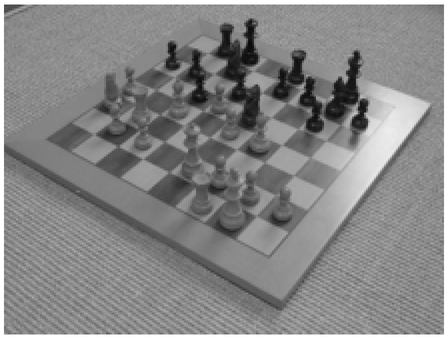
METHOD:
The harris corner detector works by monitoring the surrounding changes in intensity of a pixel. A search window is used to distinguish corners as regions within an images where there is a strong intensity gradient in two or more directions. The R value is an indicator of the strength of the gradient change in particular direction which is due to a corner. Positive R values indicate a corner and negative values indicated edges.

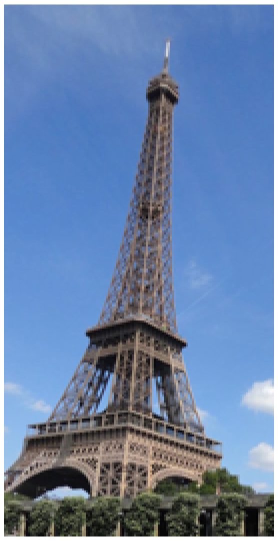
The following process was used and the corresponding matlab code is shown in the appendix under Question 1.


Compute the Gaussian derivitives for each pixel
Compute the 2nd moment matrix in the window around the pixel
Compute corner response function R using cornermetric function
Find the points with R values above set threshold

RESULTS:

The following images were used to investigate the effectiveness the Harris corner detector. Image A was selected to study the effect of how noisy backgrounds can be picked up as corners. Image B was selected to see how minute and complex details affect the detection of corners. Image C was selected to determine the how a similar intensity background affects the detection of corners around objects.





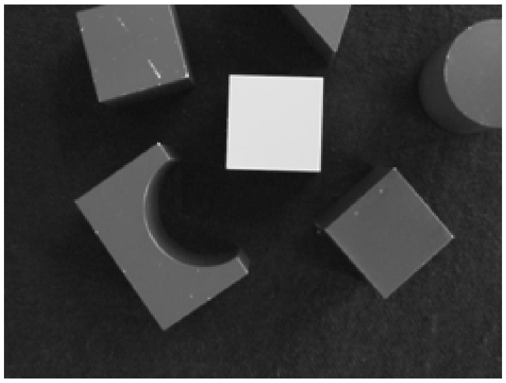


Image A Image B Image C

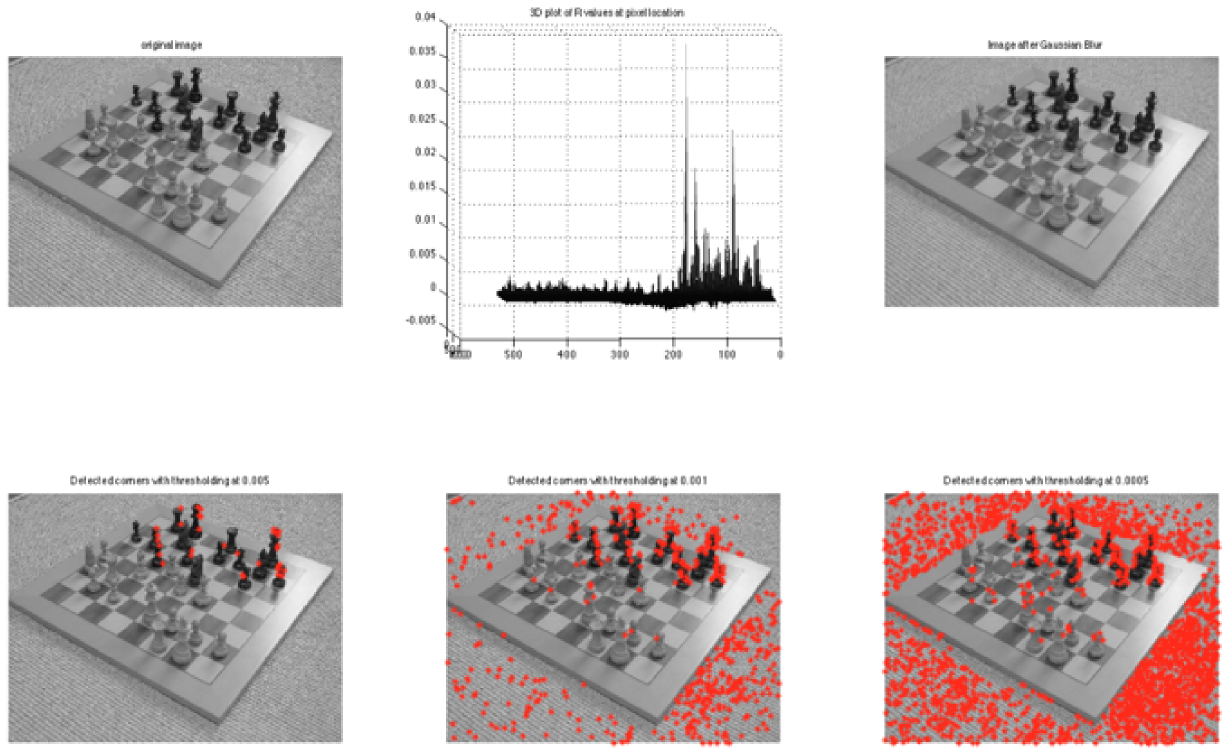


Figure 1.1

The top left image shows the converted gray scale image and the top right image shows the image after the Gaussian filter is applied. The top middle image is of a 3D plot of R value at pixel location. It shows a much high concentration of corners toward the bottom of the image. The bottom left image with a threshold value of 0.005 shows corners detected only on the black chess pieces. The bottom right image with a threshold value of 0.0005 shows many detected corners on the carpeted background around the chessboard.

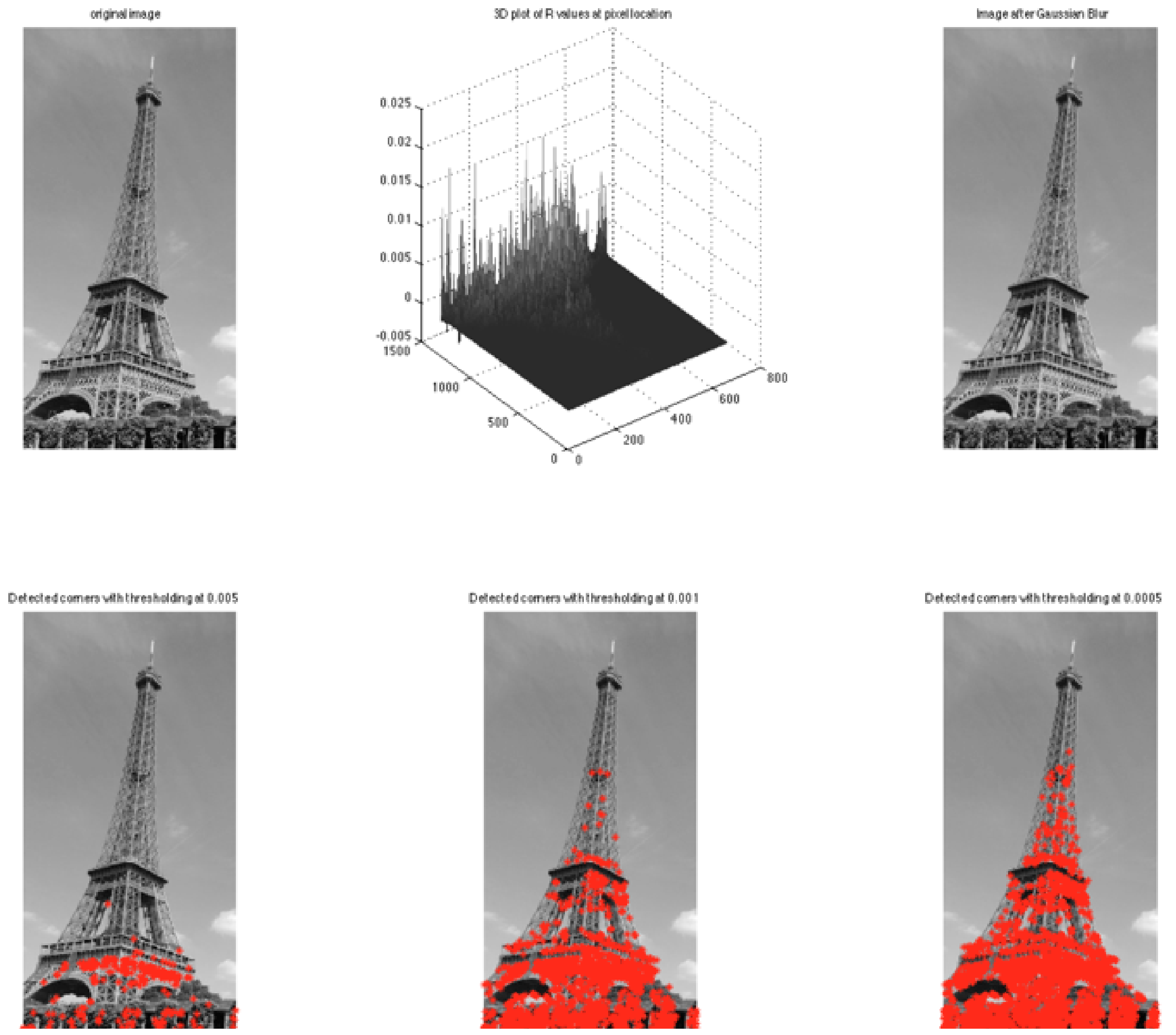


Figure 1.2

Figure 1.2 sticks to the same layout of images as figure 1.1. Figure 1.2 shows the results of using the Harris detector with threshold R values set at 0.005, 0.001 and 0.0005. The bottom images show the detected corners with decreasing threshold values of 0.005, 0.001, and 0.0005 from left to right.

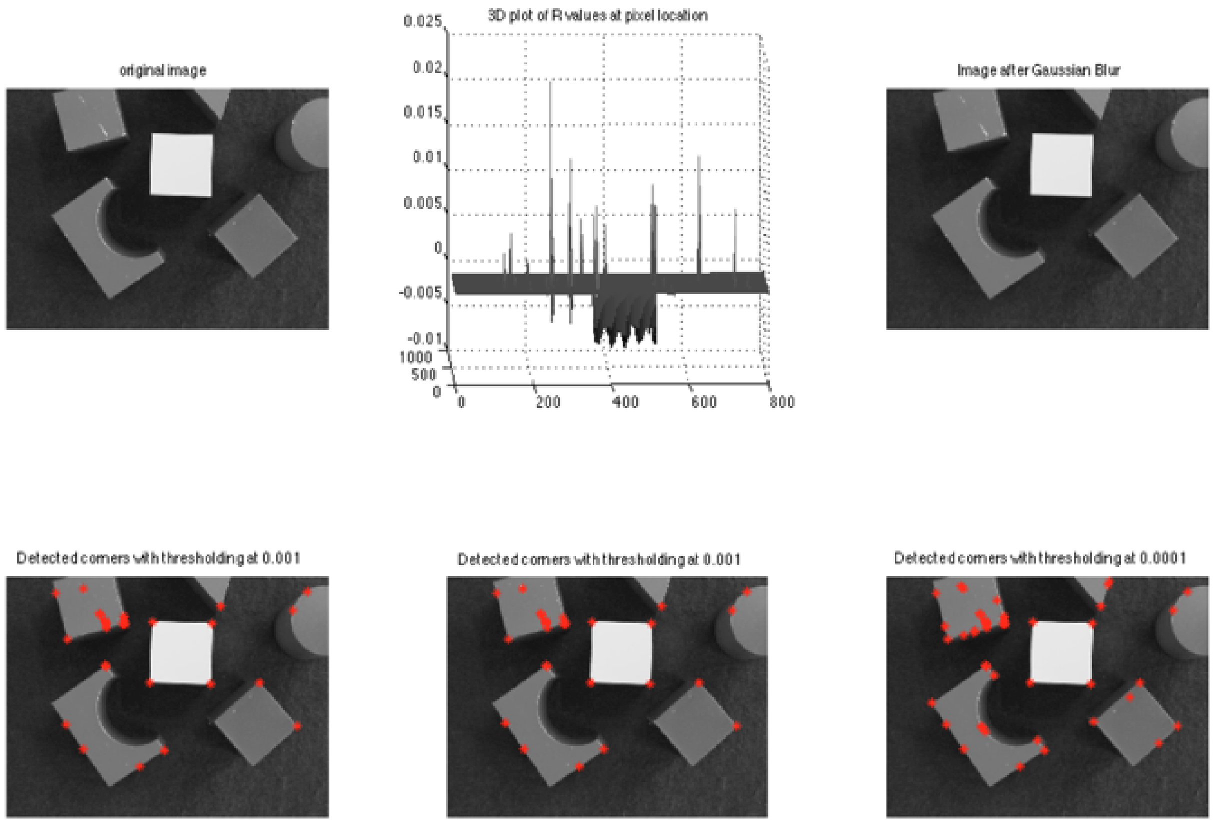


Figure 1.3

Figure 1.3 shows the same layout of images as figures 1.1 and 1.2 but the bottom right image has a lower threshold value of 0.0001 to attempt to detect more corners. The R value peaks on image C are far more pronounced than images A and B as shown as lone peaks on the 3d plot.

DISCUSSION:

In the first set of images of the chess board, we observe that there have been alot of false corners detected for a threshold value of 0.005. This indicates that the Harris corner detector can be very susceptible to noise. At a threshold of 0.005 however, the number of corners is much less and focus on the black chess pieces. The Harris corner detector uses gradient changes to detect corner and thus is very good when the interested corner has a great change in greyscale values. However as seen in the chess pieces image where the white piece would be similar in attributes to the black pieces however were not detected as there is greater change in greyscale value for the black pieces. Hence other image manipulation techniques would be needed if we wanted to detect a corner where there isn't such a great gradient change.

In the second set of images, much more of the bottom of the eiffel tower was detected to have corners present with the top of it having very little. This result was somewhat unexpected. Upon closer inspection however, the bottom of the image has higher contrasting intensities surrounding the bottom. This is due to the fact that the gaps in the bottom of the tower are larger and therefore show high contrast than the top. Also because of the low quality of the image, the low pixel density of the does not allow a higher contrast in pixel intensity near the top of the tower to come through and, as a result, this washes out the detail and thus making it more difficult to detect corners without lower the R value threshold which would introduce more fale corners to be detected.

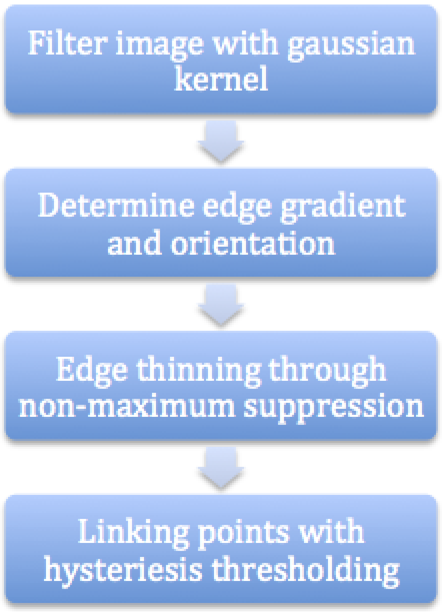
In the third set of images, not all corners were detected for the range of threshold values set. Rather than identifying the corners of the blocks as expected, the white scuff marks on the cubes were detected. This is due to the nature of the Harris detector looking for high gradient changes. The corners of the white cube showing up strong support this hypothesis. To detect the corners more effectively, perhaps segmentation or intensity thresholding can be performed prior to applying the corner detector to increase the intensity gradient magnitude for better results.

Q2) Canny Edge Detector

AIM:
To use different edge detection techniques and understand their differences. Edges can be formed due to depth discontinuity, material change, colour change and change in texture. A canny edge detector, a sobel edge detector and laplacian of Gaussian detectors can be very helpful in detecting edges.

Edge detection is an important tool in computer vision applications as it can reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant, while preserving the important structural properties of an image.

METHOD:
The Canny edge detection method is a four step process outlined below. It uses two parameters for fine tuning which are the size of the gaussian filter as well as the upper and lower thresholds.


Filter image with gaussian kernel
Determine edge gradient and orientation
Edge thinning through non-maximum suppression
Linking points with hysteriesis thresholding

The following code was used to apply the sobel and laplacian of Gaussian edge detectors. The thresholding for the sobel and LOG function was done automatically to increase the usability of the code.

```
H = fspecial('average',1);
I2 = imfilter(I,H,'replicate');

BW = edge(I2, 'sobel',[] );
subplot(2,2,3), imshow(BW)
title('binary image created by the sobel edge detector')
```

```
H = fspecial('average',7);
I3 = imfilter(I,H,'replicate');

BW = edge(I3, 'log', [],'vertical');
subplot(2,2,4), imshow(BW)
title('binary image created by the laplacion of gaussian(LOG) edge detector')
```

RESULTS:

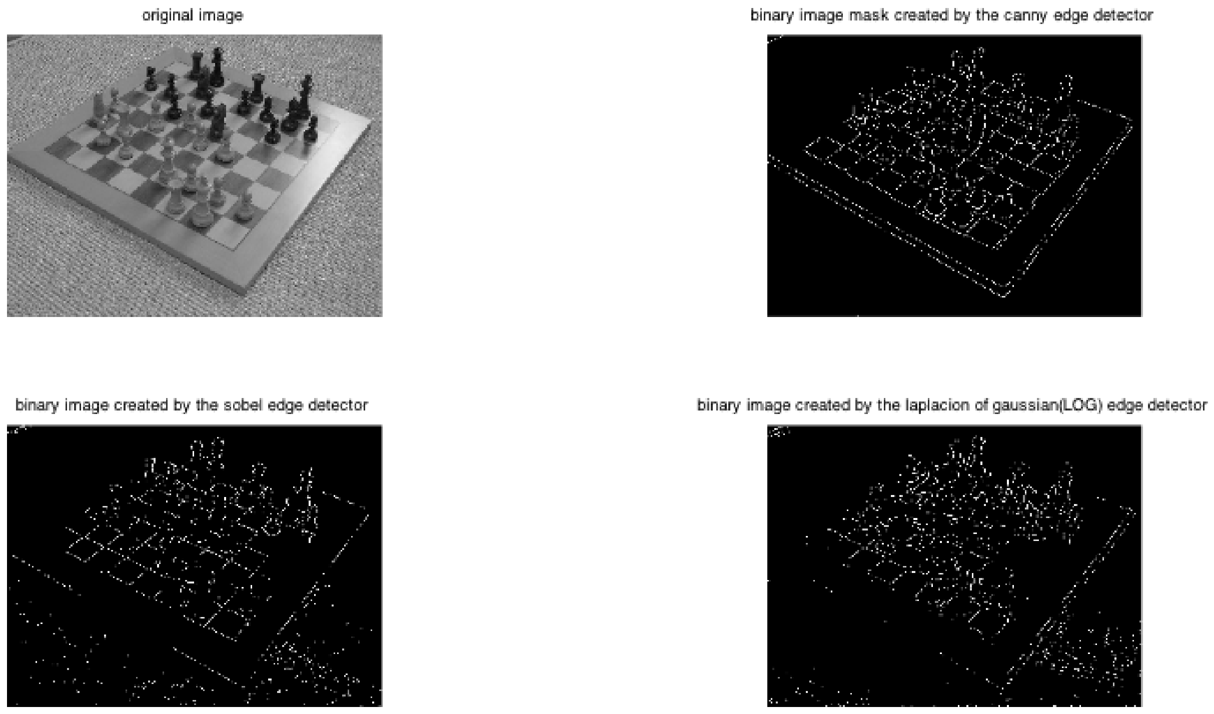

original image
binary image mask created by the canny edge detector
binary image created by the sobel edge detector
binary image created by the laplacion of gaussian(LOG) edge detector

Figure 2.1

A filter mask size of 4 by 4 was used for both the LOG and sobel filters. It can be seen that the sobel filter detects the desired edges while the sobel and LOG didn't pick up all the line of the chess board. The canny edge detector clearly shows the most accurate representation of the edged around the chessboard.

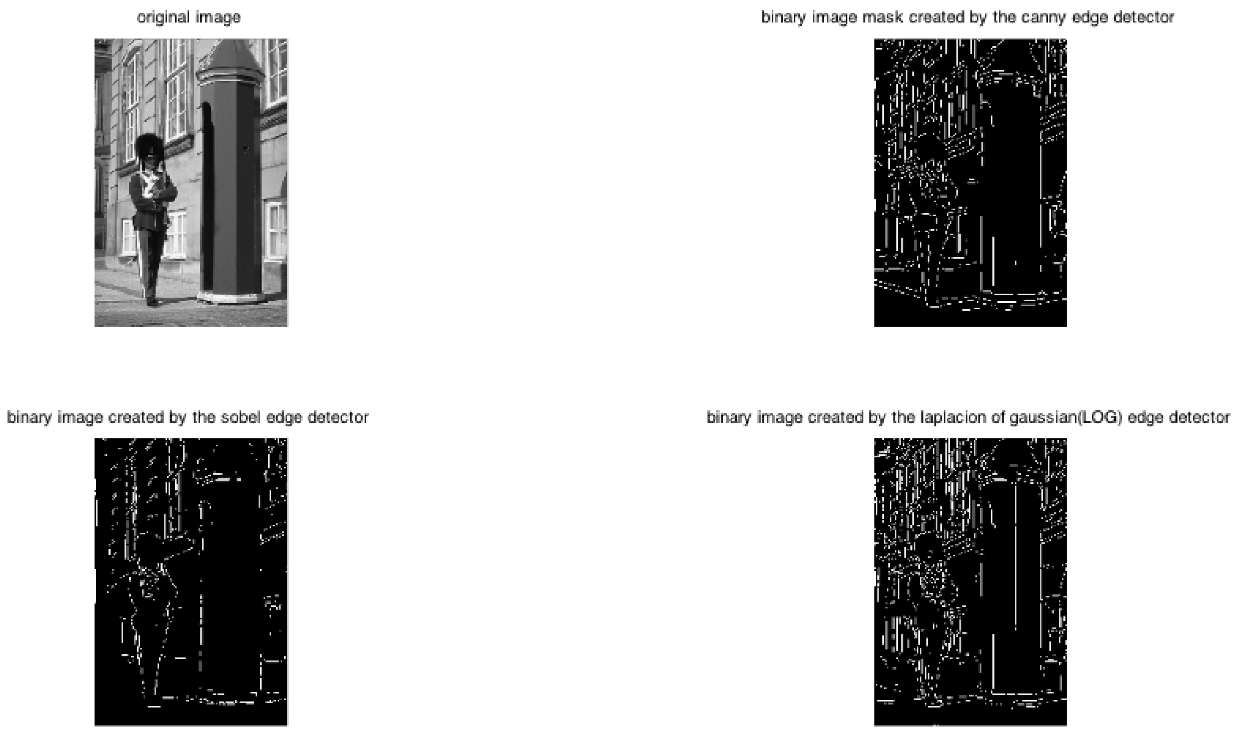

original image
binary image mask created by the canny edge detector
binary image created by the sobel edge detector
binary image created by the laplacion of gaussian(LOG) edge detector

Figure 2.2

A filter mask size of 4 by 4 was used. The sobel detector picked up fewer edges than the canny edge detector while the LOG picked up the greatest number of edges.

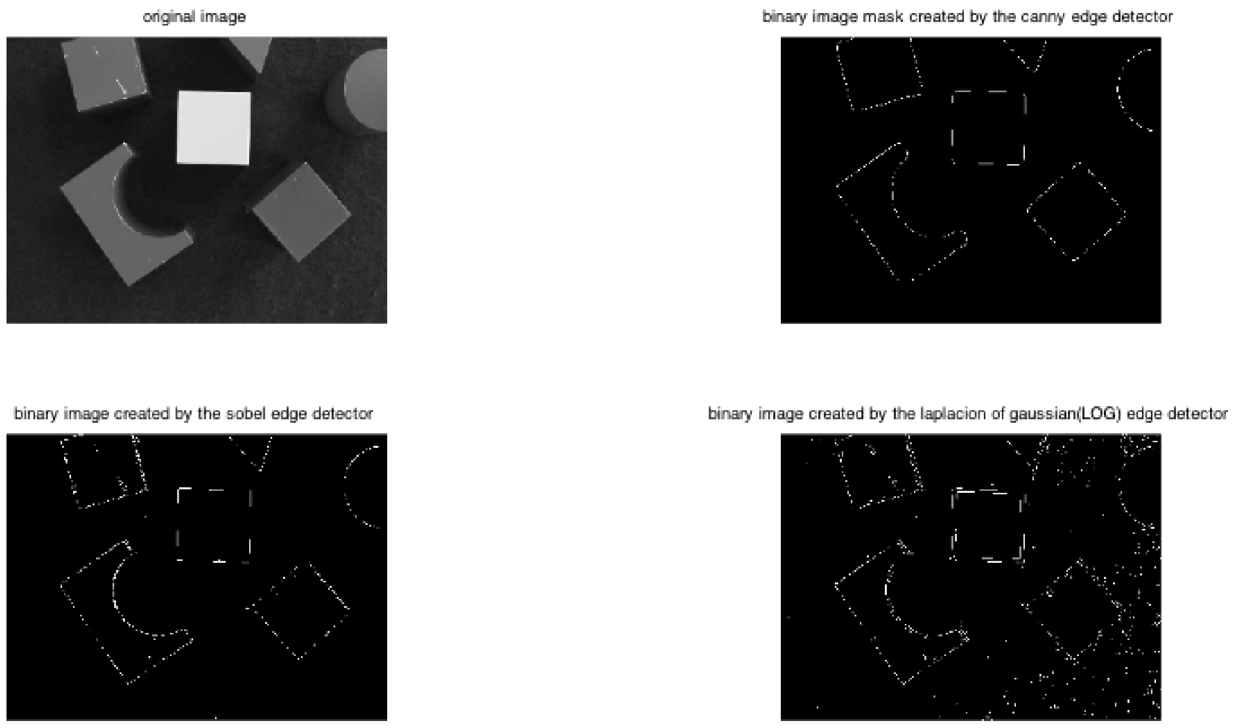

original image
binary image mask created by the canny edge detector
binary image created by the sobel edge detector
binary image created by the laplacion of gaussian(LOG) edge detector

Figure 2.3

No filtering was needed for the sobel detector in this case and a filter size of 7x7 was used for the LOG detector. While the canny edge and sobel detectors were able to pick up the edge of the image the LOG detector picked up a lot of noise from the background. This noise was of the similar level to the edges and thus could not be removed.

DISCUSSION:

The canny edge detector was the easiest edge detector to use as it didn't require a smoothing filter to be used. Instead the sigma value could be changed. It finds the local maximum for the gradient where the gradient is calculated using the derivative of a Gaussian filter.The Canny detector takes two inputs: one for the threshold value and the other to threshold the noise/weak edge. In the output only the strong edges are displaced unless the weak edges are connected to them.

The Laplacian of Gassian edge detector is the hardest one to edge detectors to use as it is very susceptible to noise. As a result larger averaging filters needed to be used on the LOG image to try and achieve a better result. This could be due to the use of the laplacian technique which highlights noise as well as the edge.The sobel edge detector shows quite decent results compared to the canny method. It uses two kernels, one for both the x and y directions and then calculates the intensity gradients in both directions. These are then combined to give the magnitude of the gradient at each point.

If the other filters are to be used a good starting value for an averaging filter size is 5 by 5. Letting the filters automatically choose their own threshold worked well and was far more efficient than changing values for each case.

Q3) Hough Transform

AIM:

The Hough transform is an invaluable method of extracting quantifiable data from a line-segmented image. Whilst image processes such as the Canny edge detector from question 2 are able to isolate the most prominent edge features in an image quite successfully, this detection only serves to filter out undesirable edge data. Whilst as humans we can clearly identify the edges in the refined image, we require use of a method such as the Hough transform to extract key metrics such as pixel location and length of found edges in the image, which can be used for a variety of applications.

Knowing that the Hough transform is a method of quantizing the position of lines by voting for line combinations, our aim in this question was to gain a better understanding of how the Hough transform is performed from pixel-space to rho-theta-space, along with its effectiveness on a variety of images and how the extracted peak data can be inverted for the identification of prime line combinations.

METHODS:

- First the image is loaded into MATLAB, and converted to grayscale. This is necessary as the Hough transform is a 2-dimensional, single space transform whilst a colour image is composed of the concatenation of 3 different colour spaces.
- With the image loaded, it is passed through a Canny edge detector to filter out the undesirable line segments in the image, which is a significant proportion of the detail, leaving a prominent outline of the sharpest edges.
- The filtered image is then passed through the Hough transform, which converts the pixel points from X-Y pixel space to the rho-theta space. The Hough transform is simplest to understand with the following transform from X-Y space to gradient-intercept (M-B) space as follows:

  $$y = m(x) + b \;{-}{>}\; b = -x(m) + y$$

  Such that any point (X,Y) in a line, maps to a particular line gradient and intercept in M-B space. Just as many points comprise a single line of particular gradient and intercept, multiple points will map to the same point in (M-B) space. Thus in this transform we will be able to locate the hotspots (or peaks) where many (X,Y) points converge, suggesting a line. Choosing which points can vote depends on the intensity thresholding, this technique is most effective on binary images. The problem with the M-B transform is that it cannot account for highly vertical lines, for which the gradient M is undefined. Thus we use the more robust rho-theta space:

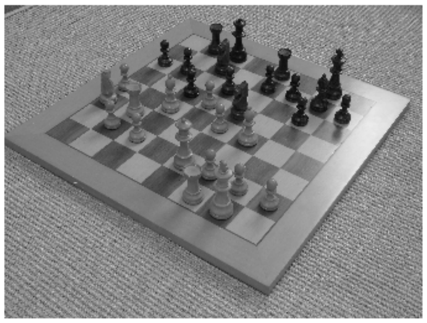
  $$r = x\,cos(\theta) + ysin(\theta)$$

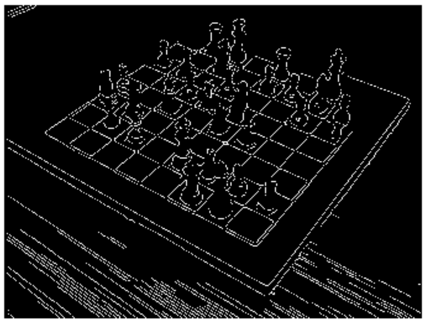
That parameterises the line in terms of rho-theta, where rho is the perpendicular distancefrom the line to the origin, and theta is the angle this perpendicular line makes with the x axis. Thus all lines in X-Y space can be represented by a theta of +90 to -90 degrees and varying rho distance from the origin. The voting behaviour of this transform is identical.

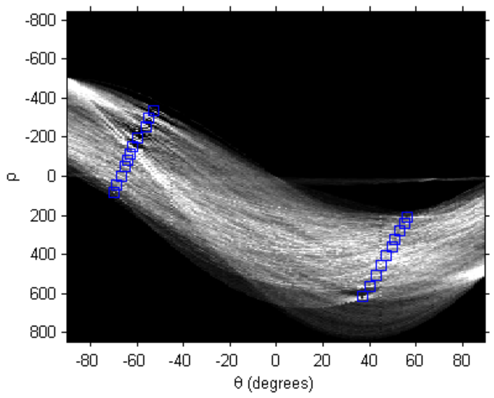
- From the resulting Hough transform, peaks can be identified as the most prominent lines in the image. Because of the nature of the transform, the area around a given peak will also feature significant votes, as there are an infinite number of lines that roughly describe the points of interest. Thus a ‘separation window’ must be nominated, to remove any false peaks, which can also serve to mask/hide less prominent peaks if one is simply choosing the ‘n’ most voted for line combinations. Alternatively, a voting threshold can be implemented to immediately discount peaks that did not acquire a minimum number of votes. This is achieved using the ‘houghpeaks’ function which encapsulates window separation, threshold and maximum peaks to identify.

- With the peaks identified, their rho-theta combinations have to be transformed back into X-Y pixel space to be usable. In this case we are using the inverted peaks to overlay the most prominent lines back on the original image. It is important to use the original grayscale image, and not the Canny image, for this purpose. The endpoints are determined by first gathering all candidate (X,Y) non-zero pixel points that describe the hough peak bin, and compute the square of the distance between them all. This distance metric is then used to identify ‘gaps’ in the line. If a gap is above the defined line threshold of the function, then each of the candidate points comprising the gap become endpoints of 2 different lines returned by the function.
- With all the candidate lines returned by the ‘houghlines’ function, the pairs of points are simply plotted onto the original image to show the results.

RESULTS:







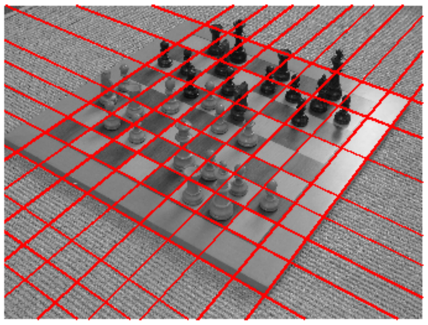
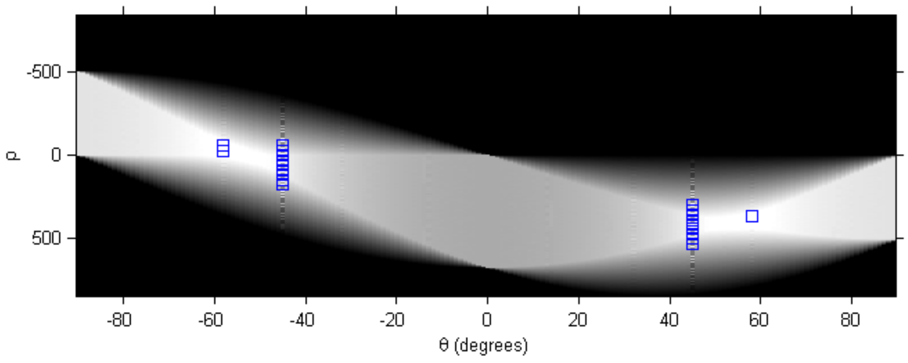


Figure 3.1: Resulting output of the transform process showing the original grayscale image (top left), the result of the canny edge detector removing much of the detail (top right), the resulting hough transform into rho-theta space with highlighted peaks in squares (bottom left), and the resulting re-map of these peaks onto the original grayscale image (bottom right). Note the large amount of lines grouped in the bottom left corner of the edge detection, resulting in 3 detected peak lines in the carpet. Note also missed chessboard lines in the image.



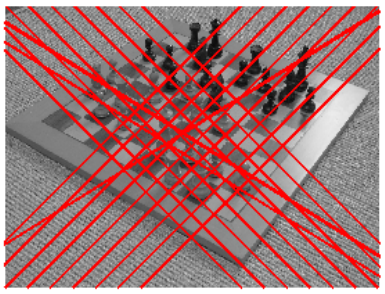


Figure 3.2: An alternative result where no Canny edge detection is performed prior to using the hough transform. The higher amount of detail acts to 'thicken' the hough transform (top) as the image consists of a larger number of high intensity pixels and edges. These additional lines have caused a skewing of the peak detection, as evidenced in the output (bottom) where few lines correlate to significant edges in the image.

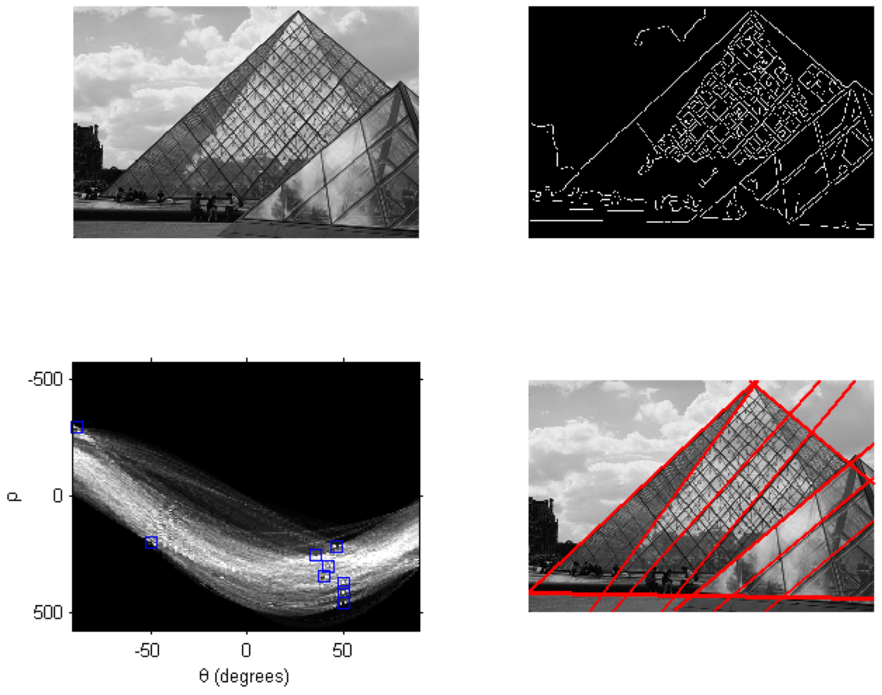


Figure 3.3: The result of a different image, the Louvre, being passed through the Hough transform for line identification. The stock Canny edge detection parameters have been less effective in this case, with the shadowed left region of the pyramid failing to yield any binary edges with the current threshold settings. Despite this, the prominent edges of the pyramid have been identified by the peaks (bottom right) along with several major interior pane edges.

DISCUSSION:

The Hough transform has been an effective method of extracting quantifiable edge positions from the Canny edge output. However the effectiveness of the line extraction is highly dependent on the effectiveness of the edge detector. This is seen in the comparison of figures 3.1 and 3.3, where the higher segmentation of lines in the Louvre image has resulted in less detection of lines. However smaller lines could be identified if the 'houghlines' line length threshold parameter was relaxed.

In figure 3.1, it was noted that 3 carpet lines featured prominently in the max peak detection. This has arisen from the high concentration of carpet lines detected in the lower left corner of the image in the Canny edge detector stage. The clustering of these lines acts to concentrate votes in that region of the Houghspace. This causes the 3 carpet lines to feature more prominently than the remaining lines of the chessboard, an issue that can only be resolved by better refinement of the edge detection.

The missed chessboard lines most likely came about due to the segmentation of the main chess edges by the cluster of chess pieces towards the top of the image. Relaxation of the line length parameters along with an increase to the maximum number of identified peaks would help to improve the detection, although their positions could easily be inferred from the position of the other lines.

Whilst this application of the Hough transform has been used to determine lines in an image, similar logic can be applied to find a variety of different shapes from edges; such as circles, squares, rectangles etc. by applying the appropriate coordinate transform.

Q4) SIFT

AIM:

SIFT is a commonly used feature matching algorithm which is unaffected by transformations to an image. These transformations can either be geometric (rotation, scale or affine) or photometric (affine). The two images being matched are compared based on 128 properties of their features. A feature match is determined by the calculation of the Euclidean distance between the 128 properties of each feature and checking if the distance is below a given threshold. This question experiments with different ways of determining a feature match and testing the SIFT method on a variety of transformed images.

METHOD:

- Acquire images for matching and convert to greyscale as SIFT only works on 2-dimensional arrays.
- Extract features from images using the compiled SIFT library
- Match features using either the dot, ssd or ratio methods
- Show the matched keys.

Dot Method:
This method approximates the euclidean distance using the dot product as it is computationally faster in Matlab.

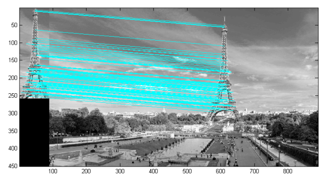
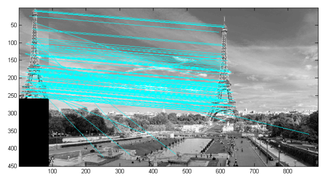
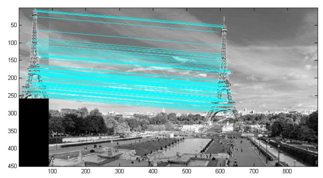
- Find the dot product of the two descriptor arrays for each image
- Take the acos of the dot products
- Check if the distance between two descriptors agrees more than the threshold times the second most probable match

Sum Square Difference (SSD) Method:
This takes the SSD of each descriptor vector and checks if it below a certain threshold.

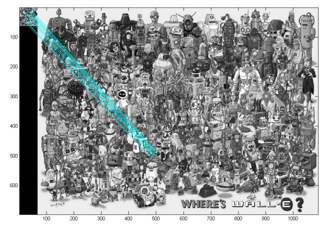
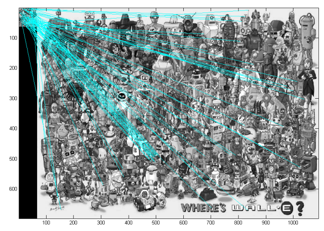
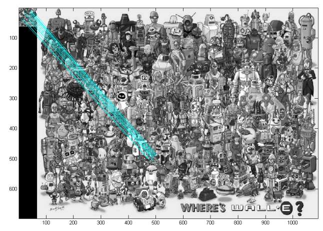
- Using the first extracted feature from the first image, take each parameter from the descriptor vector and find the difference of the squares with the corresponding parameter in each descriptor for all features in image 2.
- For each feature the SSD is summed across all parameters to give the SSD value.
- Compare the SSD value with the threshold to determine if it a matched feature.

Ratio of SSD Method:
This takes the ratio of the most likely SSD match and the second most likely SSD match and checks if the ratio of the two is less than a given threshold.

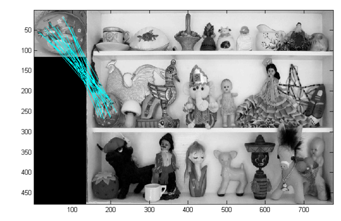
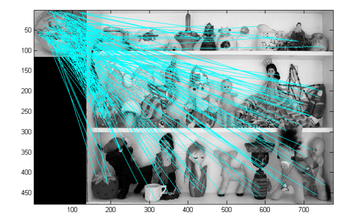
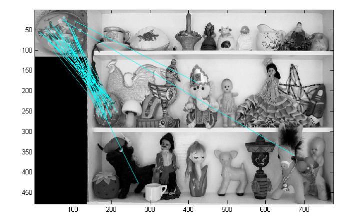
RESULTS:

DOT Method	SSD Method	Ratio Method
		

This image shows a cropped image of the eiffel tower on the left and a full landscape shot of the eiffel tower on the right. The DOT method has many features linked from image one to two without any outliers. All the features have been correctly identified. The SSD method has a few outliers or false positives, however the Ratio method has corrected for this and there is again a strong correlated link between the two images.

DOT Method	SSD Method	Ratio Method
		

This image shows a cropped image of wall-e on the left and a larger image with wall-e obscured in a crowd of cartoon characters on the right. The DOT method has many features linked from image one to two without any outliers. All the features have been correctly identified. The SSD method has a few outliers or false positives, however the Ratio method has corrected for this and there is again a strong correlated link between the two images.

DOT Method	SSD Method	Ratio Method
		

This image shows a cropped image of an object on the left and a shelf full of objects on the right. The object has been rotated relative to its orientation in the image on the right.The DOT method has many features linked from image one to two without any outliers. All the features have been correctly identified. The SSD method has a many outliers or false positives, however the Ratio method has corrected for this and there is again a strong correlated link between the two images.

DISCUSSION:

The SIFT method has overall been able to successfully match a variety of features of two images to each other. The method is slow, as there are 128 parameters that it is comparing, but is straightforward to implement. The dot product and ratio methods were comparable and gave very few false positives. They were robust to transformations, as seen in the third example, and partial obstruction, as evidenced in the second example.

The SSD method had significant problems as it did not compare the difference between the two most likely matches. This reduced the likelihood of picking the most likely match and instead gave far more matches. There was not any appreciable computational advantage to using the SSD method over the ratio method so it is recommended to use the ratio method over the SSD method.

Appendix

Question 1 Matlab Code

```
clc
clear all
close all

ImageDir='imdemos/';%directory containing the images

Irgb = imread([ImageDir 'figures.jpg']);
if ndims(Irgb) == 3;
   I = rgb2gray(Irgb);
elseif ndims(Irgb) == 2;
   I = Irgb;
end

figure, set(gca,'FontSize',30,'fontWeight','bold'),
subplot(2,3,1), imshow(I)
set(gcf,'color','w');
title('original image')
%% wweewew

[x,y]=size(I);
X=1:x;
Y=1:y;
[xx,yy]=meshgrid(Y,X);

%% Harris corner detection
filtCoeff = fspecial('gaussian',[9 1],1);
sensFactor = 0.05;
R = cornermetric(I,'Harris','FilterCoefficients', filtCoeff ,'SensitivityFactor',sensFactor);
i=im2double(R);
subplot(2,3,2),mesh(xx,yy,i);colormap(jet)
title('3D plot of R values at pixel location')

R1 = R;
R2 = R;
R3 = R;

maxR = max(max(R(:,:)))
minR = min(min(R(:,:)))
```

```
blurred = imfilter(I,filtCoeff);
subplot(2,3,3), imshow(blurred)
title('Image after Gaussian Blur')



threshold = 0.001
idx = find(R1 < threshold);
R1(idx) = 0;
corner_peaks = imregionalmax(R1);
[cornerY,cornerX]  = find(corner_peaks == true);
subplot(2,3,4), imshow(I), hold on
plot(cornerX, cornerY, 'r*');
title('Detected corners with thresholding at 0.001')



threshold = 0.001
idx = find(R2 < threshold);
R2(idx) = 0;
corner_peaks = imregionalmax(R2);
[cornerY,cornerX]  = find(corner_peaks == true);
subplot(2,3,5), imshow(I), hold on
plot(cornerX, cornerY, 'r*');
title('Detected corners with thresholding at 0.001')



threshold = 0.0001
idx = find(R3 < threshold);
R3(idx) = 0;
corner_peaks = imregionalmax(R3);
[cornerY,cornerX]  = find(corner_peaks == true);
subplot(2,3,6), imshow(I), hold on
plot(cornerX, cornerY, 'r*');
title('Detected corners with thresholding at 0.0001')
```

Question 2 Matlab Code

```
%% clear all
close all

ImageDir='imdemos/';%directory containing the images

Irgb = imread([ImageDir 'liftingbody.png']);
if ndims(Irgb) == 3;
   I = rgb2gray(Irgb);
elseif ndims(Irgb) == 2;
   I = Irgb;
end
subplot(2,2,1), imshow(I)
title('original image')

%%
canny_thresh = [0.1 0.2];
canny_sigma = 5;
BW = edge(I,'canny', canny_thresh, canny_sigma);
%BW = edge(I, 'canny', 0.2 , canny_sigma);

subplot(2,2,2), imshow(BW)
title('binary image mask created by the canny edge detector')

H = fspecial('average',1);
I2 = imfilter(I,H,'replicate');

[gv,t] = edge(I2,'sobel', 'vertical');
sobel_thresh = [0 0.7];
%BW = edge(I, 'sobel', []);
BW = edge(I2, 'sobel',[] );
subplot(2,2,3), imshow(BW)
title('binary image created by the sobel edge detector')

H = fspecial('average',7);
I3 = imfilter(I,H,'replicate');


[gv,t2] = edge(I3, 'log', 'vertical');
t2
sobel_thresh = [1.5*t 2*t];
BW = edge(I3, 'log', [],'vertical');
```

```
subplot(2,2,4), imshow(BW)
title('binary image created by the laplacion of gaussian(LOG) edge detector')
```

Question 3 Matlab Code

```
clc
clear all
close all

ImageDir='imdemos/';%directory containing the images

I = imread([ImageDir 'chess.png']);
I = rgb2gray(I);
subplot(2,2,1), imshow(I)
title('original image')

%% canny edge detection

canny_thresh = [0.1 0.25];
canny_sigma = 2;
BW = edge(I,'canny', canny_thresh, canny_sigma);
subplot(2,2,2), imshow(BW)
title('output from the canny edge detector')


%% hough transform

[H,theta,rho] = hough(BW);

subplot(2,2,3), imshow(imadjust(mat2gray(H)),[],'XData',theta,'YData',rho,...
        'InitialMagnification','fit');
xlabel('\theta (degrees)'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(hot)
title('output from the hough transform');


maxPeaks = 21;
peakSepar = [55 23];
peakThresh = 100;
P = houghpeaks(H, maxPeaks, 'Threshold', peakThresh, 'NhoodSize', peakSepar);
```

```
x = theta(P(:,2));
y = rho(P(:,1));
plot(x,y,'s','color','blue');

lineGap = 20;
lineMinL = 7;
lines = houghlines(I,theta,rho,P,'FillGap',lineGap,'MinLength',lineMinL);

subplot(2,2,4), imshow(I), hold on

for k = 1:length(lines)
xy = [lines(k).point1; lines(k).point2];
plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');
end
title('orginal image with the detected edges')
```

Question 4 Matlab Code

```
% num = match(image1, image2)
%
% This function reads two images, finds their SIFT features, and
%   displays lines connecting the matched keypoints.  A match is accepted
%   only if its distance is less than distRatio times the distance to the
%   second closest match.
% It returns the number of matches displayed.
%
% Example: match('scene.pgm','book.pgm');

function num = match(image1, image2,method)

% Find SIFT keypoints for each image
[im1, des1, loc1] = sift(image1);
[im2, des2, loc2] = sift(image2);

% For efficiency in Matlab, it is cheaper to compute dot products between
%  unit vectors rather than Euclidean distances.  Note that the ratio of
%  angles (acos of dot products of unit vectors) is a close approximation
%  to the ratio of Euclidean distances for small angles.
%
% distRatio: Only keep matches in which the ratio of vector angles from the
%   nearest to second nearest neighbor is less than distRatio.
```

```
distRatio = 0.6;

% For each descriptor in the first image, select its match to second image.
match = matchFeature(des1, des2, distRatio, method);

% Create a new image showing the two images side by side.
im3 = appendimages(im1,im2);

% Show a figure with lines joining the accepted matches.
figure('Position', [100 100 size(im3,2) size(im3,1)]);
colormap('gray');
imagesc(im3);
% imshow(im3);
% imshow(imread('wall-e.jpg'));
hold on;
cols1 = size(im1,2);
for i = 1: size(des1,1)
  if (match(i) > 0)
    line([loc1(i,2) loc2(match(i),2)+cols1], ...
         [loc1(i,1) loc2(match(i),1)], 'Color', 'c');
  end
end
hold off;
num = sum(match > 0);
fprintf('Found %d matches.\n', num);




function match = matchFeature(des1, des2, distRatio, method)

switch method
    case 'dot'
        des2t = des2';                          % Precompute matrix transpose
        for i = 1 : size(des1,1)
           dotprods = des1(i,:) * des2t;        % Computes vector of dot products
           [vals,indx] = sort(acos(dotprods));  % Take inverse cosine and sort results

           % Check if nearest neighbor has angle less than distRatio times 2nd.
           if (vals(1) < distRatio * vals(2))
              match(i) = indx(1);
           else
              match(i) = 0;
```

```
            end
          end
       case 'ssd'
          for I = 1:size(des1,1)
            for  J = 1:size(des2,1)
               rowSum(J) = sum((des1(I,:)-des2(J,:)).^2);
            end
            [vals,indx] = sort(rowSum);
            % Check if nearest neighbor has angle less than distRatio times 2nd.
            if (vals(1) < distRatio)
               match(I) = indx(1);
            else
               match(I) = 0;
            end
          end
       case 'ratio'
          for I = 1:size(des1,1)
            for  J = 1:size(des2,1)
               rowSum(J) = sum((des1(I,:)-des2(J,:)).^2);
            end
            [vals,indx] = sort(rowSum);
            % Check if nearest neighbor has angle less than distRatio times 2nd.
            if (vals(1)/ vals(2) < distRatio )
               match(I) = indx(1);
            else
               match(I) = 0;
            end
          end
end
end
```