

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ4003

Lab 2 Report

Name : Chua Zi Jian

Matriculation No.: U2022354J

Date : 1/11/2022

Content	<i>Page</i>
1 Edge Detection	3
2 Line Finding using Hough Transform	15
3 3D Stereo	22

1. Edge Detection

Edge detection is used to extract discontinuities or sharp gradients in the image. There are many different methods available, ranging from the basic Sobel operators with filter masks of

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

to the more complex Canny algorithm which involves separate steps of Gaussian derivative filtering, non-maximal suppression and hysteresis thresholding. Each edge point extracted is also known as an *edge element*, or “edgel”.

- a) Download ‘macritchie.jpg’ from NTU Learn and convert the image to grayscale. Display the image.

Code:

```
P=imread("macritchie.jpg");  
macritchie=rgb2gray(P);  
imshow(macritchie);
```

Fig 1.1.1 Code to run

Result:



Fig 1.1.2 Macritchie

- b) Create 3x3 horizontal and vertical Sobel masks and filter the image using conv2. Display the edge-filtered images. What happens to edges which are not strictly vertical nor horizontal, i.e. diagonal?

Code:

```
sobel_h = [  
    -1 -2 -1;  
     0 0 0;  
     1 2 1;  
];  
  
sobel_v = [  
    -1 0 1;  
    -2 0 2;  
    -1 0 1;  
];  
  
% convolute the sobel filter on the image, with vertical, horizontal and  
% both vertical and horizontal filter  
macritchie_sobel_h = conv2(macritchie, sobel_h);  
macritchie_sobel_v = conv2(macritchie, sobel_v);  
macritchie_sobel_all = conv2(macritchie_sobel_v, sobel_h);  
  
figure;  
imshow(uint8(macritchie_sobel_h));  
title("Sobel Filter Horizontal");  
figure;  
imshow(uint8(macritchie_sobel_v));  
title("Sobel Filter Vertical");  
figure;  
imshow(uint8(macritchie_sobel_all));  
title("Sobel Filter All");
```

Fig 1.2.1 Code to run

Result:

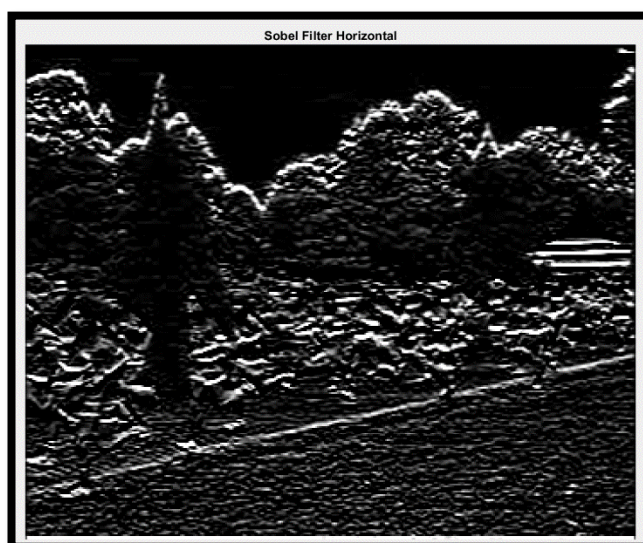


Fig 1.2.2 Sobel Filter Horizontal

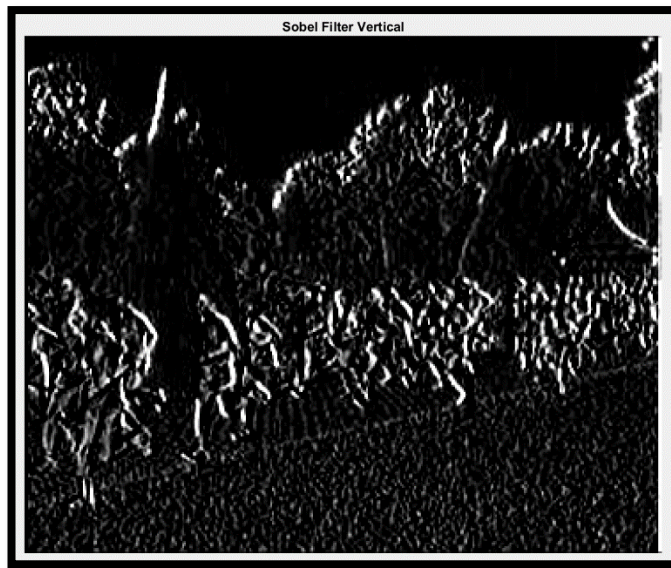


Fig 1.2.3 Sobel Filter Vertical

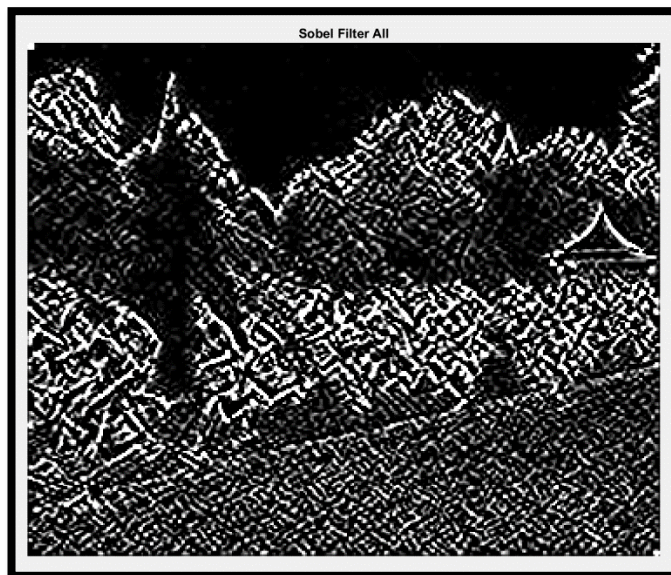


Fig 1.2.4 Sobel Filter All

Answer:

If we are using horizontal sobel filter, the vertical edges will be filtered out and if we are using vertical sobel filter, the horizontal edges will be filtered out. For both cases, **the edges which are not strictly horizontal nor vertical will become fainter** instead of filtered out like the strictly horizontal/vertical edges.

- c) Generate a combined edge image by squaring (i.e. \cdot^2) the horizontal and vertical edge images and adding the squared images. Suggest a reason why a squaring operation is carried out.

Code:

```
combinedEdge_img=macritchie_sobel_v.^2+macritchie_sobel_v.^2;  
figure;  
imshow(uint8(combinedEdge_img));  
title("Combined Edge Image");  
figure;  
imshow(uint8(sqrt(combinedEdge_img)));  
title("Squared Combined Edge Image");
```

Fig 1.3.1 Code to run

Result:

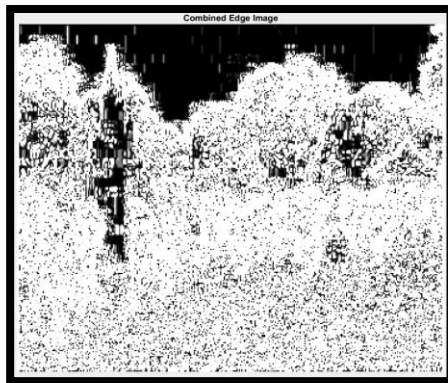


Fig 1.3.2 Combined Edge Image

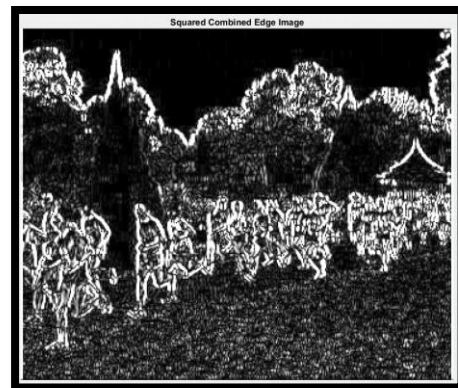


Fig 1.3.3 Squared Combined Edge Image

Reason:

Applying the horizontal sobel filter give the horizontal gradient vector and the vertical sobel filter gives the vertical gradient vector.

After the Sobel Filters have been applied, the gradient of each pixel can be negative and what we need is the resultant magnitude of the edges and the positive and negative is just an indication of direction so squaring operation will help us to obtain the resultant magnitude of the gradient.

And after that we will select the edges which has greater magnitude than the threshold.

d) Threshold the edge image E at value t by

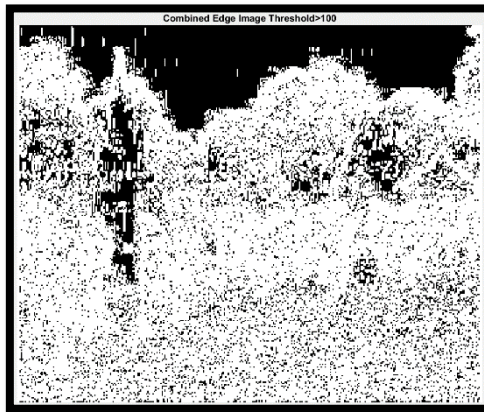
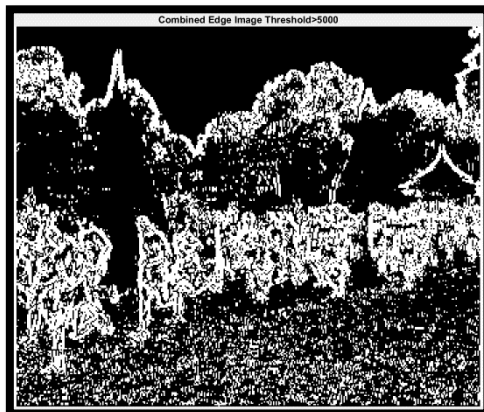
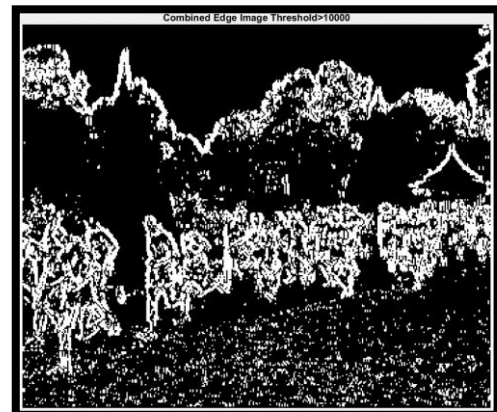
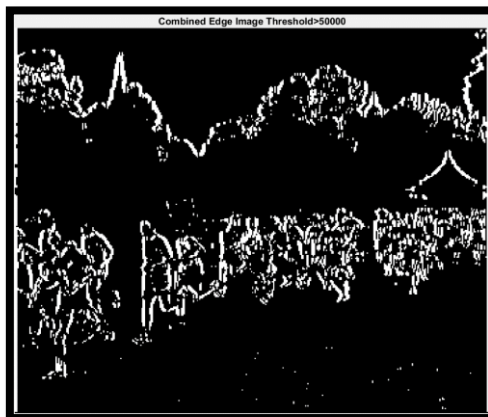
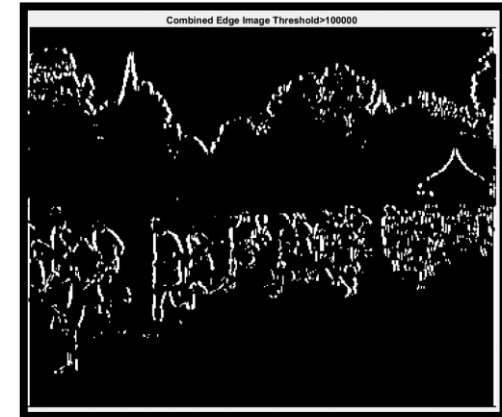
$\gg Et = E > t;$

This creates a binary image. Try different threshold values and display the binary edge images. What are the advantages and disadvantages of using different thresholds?

Code:

```
Et = combinedEdge_img > 100;
figure;
imshow(Et);
title("Combined Edge Image Threshold>100");
Et = combinedEdge_img > 1000;
figure;
imshow(Et);
title("Combined Edge Image Threshold>1000");
Et = combinedEdge_img > 5000;
figure;
imshow(Et);
title("Combined Edge Image Threshold>5000");
Et = combinedEdge_img > 10000;
figure;
imshow(Et);
title("Combined Edge Image Threshold>10000");
Et = combinedEdge_img > 50000;
figure;
imshow(Et);
title("Combined Edge Image Threshold>50000");
Et = combinedEdge_img > 100000;
figure;
imshow(Et);
title("Combined Edge Image Threshold>100000");
```

Fig 1.4.1 Code to run

Result:***Fig 1.4.2 Edge, Threshold > 100******Fig 1.4.3 Edge, Threshold > 1000******Fig 1.4.4 Edge, Threshold > 5000******Fig 1.4.5 Edge, Threshold > 10000******Fig 1.4.6 Edge, Threshold > 50000******Fig 1.4.7 Edge, Threshold > 100000***

Answer:

Advantages of high threshold:

1. The higher accuracy of getting the outline of the object in image.
2. Lesser noises in the edges detected.

Disadvantages of high threshold:

1. The image will have lesser edge compared to the lower threshold.

Advantages and disadvantages of using low threshold will be the other way around. So we can see that there will be a trade-off of details of edges and noises depends on your threshold, the higher the threshold the lesser the details but lesser noises also.

- e) Recompute the edge image using the more advanced Canny edge detection algorithm with $tl=0.04$, $th=0.1$, $\sigma=1.0$

```
>> E = edge(I,'canny',[tl th],sigma);
```

This generates a binary image without the need for thresholding.

- (i) Try different values of σ ranging from 1.0 to 5.0 and determine the effect on the edge images. What do you see and can you give an explanation for why this occurs? Discuss how different σ are suitable for (a) noisy edge removal, and (b) location accuracy of edges.
- (ii) Try raising and lowering the value of tl . What does this do? How does this relate to your knowledge of the Canny algorithm?

Code:

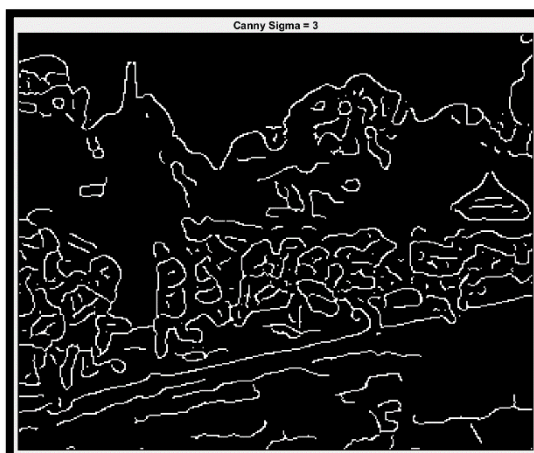
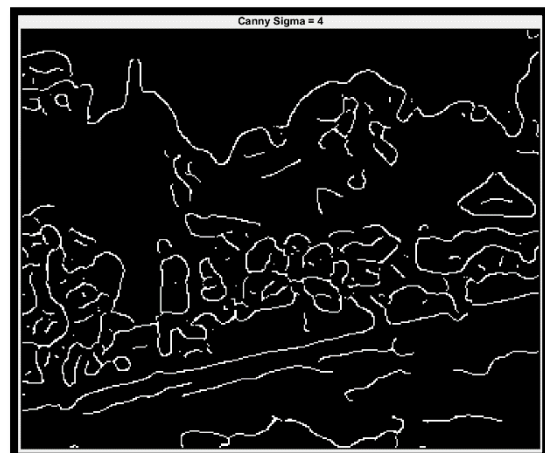
```
% (I)
tl=0.04; % Low threshold
th=0.1; % High threshold
sigma=1.0; % Standard deviation of the Gaussian filter

E = edge(macritchie, 'canny', [tl th], sigma); % apply Canny Edge detection
figure;
imshow(E);
title("Canny Edge Detector Used");

% trying sigma 1 to 5
s=5;
for sigma = 1:s
    canny_i = edge(macritchie, 'canny', [tl th], sigma);
    figure;
    imshow(canny_i);
    title("Canny Sigma = "+sigma);
end
% WHAT DO YOU SEE?*****

% (ii)
valuesList = {0.01, 0.04, 0.08};
%*****
% loop to generate image after applying Canny Edge at different tl
% values in the valuesList
for tl = 1:length(valuesList)
    sigma = 1.0;
    cannyDiffTL = edge(macritchie, 'canny', [valuesList{tl} th], sigma);
    figure, imshow(cannyDiffTL)
end
```

Fig 1.5.1 Code to run

Result:***I)*****Fig 1.5.2 Canny Edge Detector, Default****Fig 1.5.3 Canny Edge, Sigma = 1****Fig 1.5.4 Canny Edge, Sigma = 2****Fig 1.5.5 Canny Edge, Sigma = 3****Fig 1.5.6 Canny Edge, Sigma = 4**

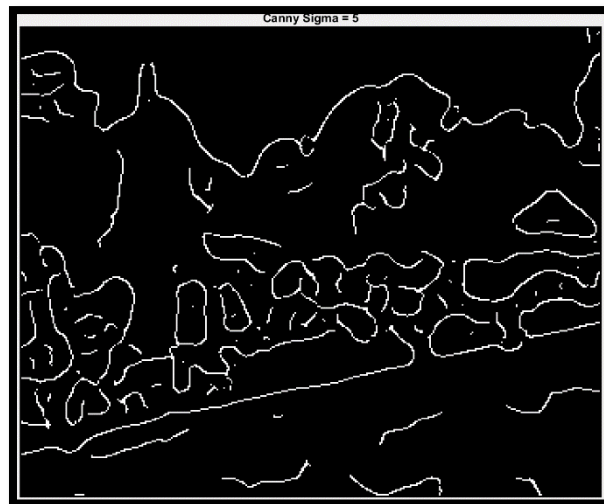


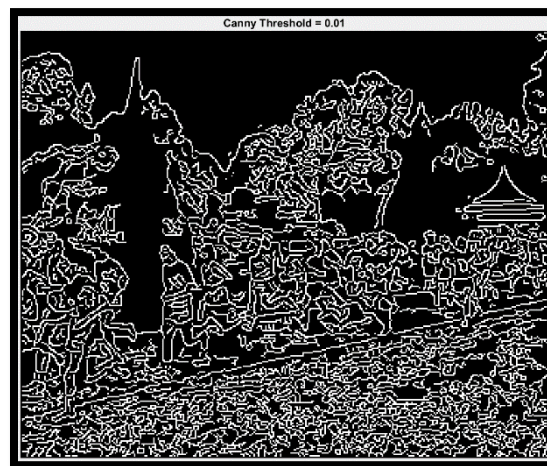
Fig 1.5.7 Canny Edge, Sigma = 5

Discussion:

Lower sigma will give better location accuracy of edges but will have more noises. As the sigma increases, although the noises is being removed but more details of the edges is also being removed and the location accuracy will drop.

I think the reasons is because canny algorithm uses gaussian filter to smooth the edges and when the sigma is increases the details of the edges will also be removed and we will get a lower accuracy of location of edges.

II)

**Fig 1.5.8 Canny Edge, Threshold = 0.01****Fig 1.5.9 Canny Edge, Threshold = 0.04****Fig 1.5.10 Canny Edge, Threshold = 0.08**

Discussion:

When the t_l is small, there will be more edges and noises. As the t_l increases, the noises will be lesser but the edges will also become less detailed.

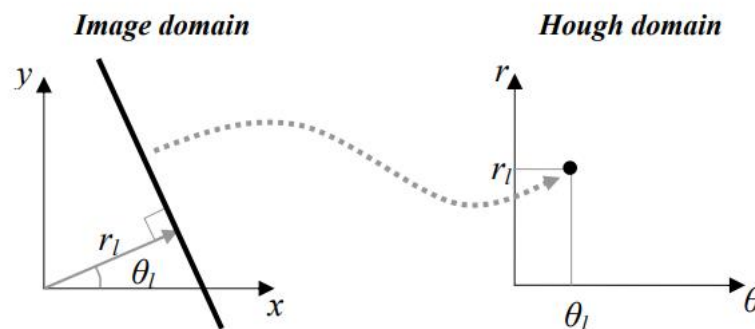
Canny algorithm uses hysteresis thresholding and the t_l is the lower bound threshold value, any value lesser than the t_l will be set to 0 so that weak edges or noises can be removed.

2. Line Finding using Hough Transform

Edge detection may not give you all the information that you want. While the above methods extract local edges, they are unable to determine the presence of long, consistent lines which have broken local edges.

Hough transform is a transformation which maps a straight line in the image domain to a single point in the Hough domain, based on the parameter mapping shown in the diagram below. Hence it can also be shown that a single point in image domain is mapped to a *sinusoidal curve* in Hough domain.

When all edgels in image space are mapped to Hough space, edgels that lie on a straight line will each map to sinusoidal curves that *intersect* at a single point in Hough space. These intersection points will have large values after the Hough transform, and can be easily detected. The coordinates of an intersection point are parameters that define the line in image space.



- a) Reuse the edge image computed via the Canny algorithm with $\sigma=1.0$.

Code:

```
tl=0.04;    % Low threshold
th=0.1;     % High threshold
sigma=1.0;  % Standard deviation of the Gaussian filter

P_macritchie=imread("macritchie.jpg");
macritchie=rgb2gray(P_macritchie);
E = edge(macritchie, 'canny', [tl th], sigma); % apply Canny Edge detection
figure;
imshow(E);
title("Edge");
```

Fig 2.1.1 Code to run

Result:**Fig 2.1.2 Edge detected**

- b) As there is no function available to compute the Hough transform in MATLAB, we will use the Radon transform, which for binary images is equivalent to the Hough transform. Read the help manual on Radon transform, and explain why the transforms are equivalent in this case. When are they different?

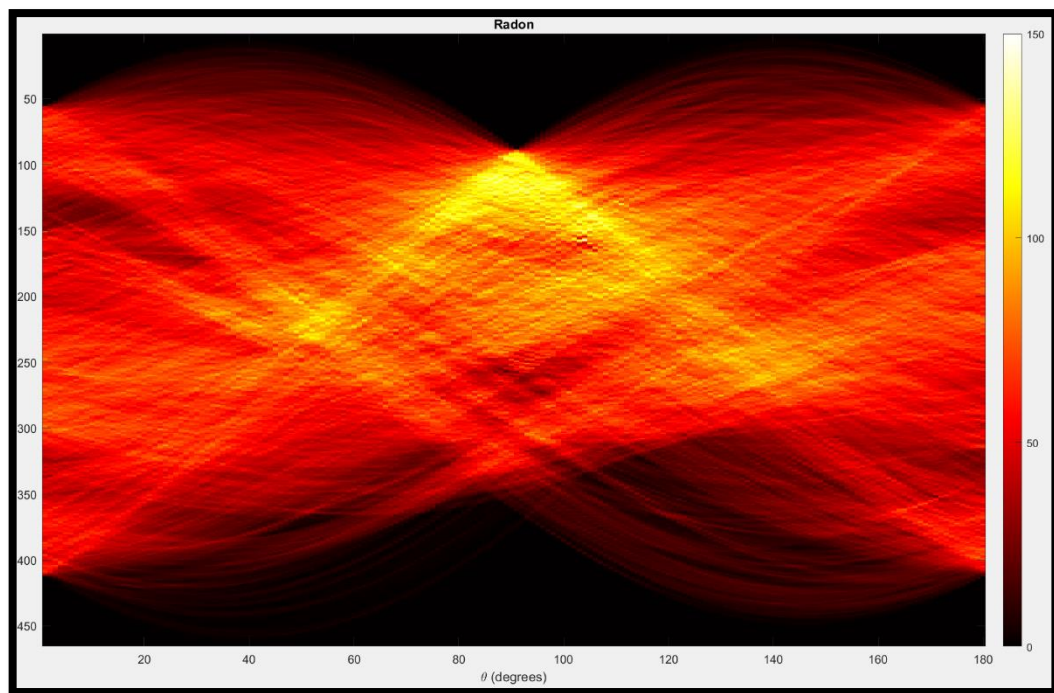
```
>> [H, xp] = radon(E);
```

Display H as an image. The Hough transform will have horizontal bins of angles corresponding to 0-179 degrees, and vertical bins of radial distance in pixels as captured in xp. The transform is taken with respect to a Cartesian coordinate system where the origin is located at the centre of the image, and the x-axis pointing right and the y-axis pointing up.

Code:

```
[H, xp] = radon(E);
figure, imagesc(uint8(H)) % to visualise the waveforms better
xlabel('\theta (degrees)');
ylabel('x');
colormap(gca, hot), colorbar;
title("Radon");
```

Fig 2.2.1 Code to run

Result:***Fig 2.2.2 Colormap*****Answer:**

The Hough transform and the Radon transform are indeed very similar to each other and their relation can be loosely defined as the former being a discretized form of the latter.

The Radon transform is a mathematical integral transform, defined for continuous functions on $R(n)$ on hyperplanes in $R(n)$. The Hough transform, on the other hand, is inherently a discrete algorithm that detects lines (extendable to other shapes) in an image by polling and binning (or voting).

In this case, Radon transform will have the same value as Hough transform because the image is applying discrete radon transform. With this both of the transform will map each pixel into equivalent sinusoidal function. However the Radon transform will result differently if we consider a continuous radon transform.

In continuous radon transform is obviously different because hough transform works discretely. Hence the density of the line will have different value.

- c) Find the location of the maximum pixel intensity in the Hough image in the form of [theta, radius]. These are the parameters corresponding to the line in the image with the strongest edge support.

Code:

```
maxH = max(H(:));
[radius, theta] = find(H == maxH);
radius = xp(radius); % obtain the radial coordinate for the maximum intensity
theta=theta-1;
disp(radius);
disp(theta);
```

Fig 2.3.1 Code to run

Result:

```
-76
103
```

Fig 2.3.2 Result of radius and theta

- d) Derive the equations to convert the [theta, radius] line representation to the normal line equation form $Ax + By = C$ in image coordinates. Show that A and B can be obtained via

```
>> [A, B] = pol2cart(theta*pi/180, radius);
>> B = -B;
```

B needs to be negated because the y-axis is pointing downwards for image coordinates.

Find C. Reminder: the Hough transform is done with respect to an origin at the centre of the image, and you will need to convert back to image coordinates where the origin is in the top-left corner of the image.

Code:

```
[A,B] = pol2cart(theta * pi/180, radius);
B = -B;

% Find center coordinate of the image
[numOfRows, numOfCols] = size(macritchie);
x_center = numOfCols / 2;
y_center = numOfRows / 2;

% Obtain the C value from Ax + By = C
C = A*(A+x_center) + B*(B+y_center);
|
disp(C);
```

Fig 2.4.1 Code to run**Result:**

```
1.9760e+04
```

Fig 2.4.2 Result of C

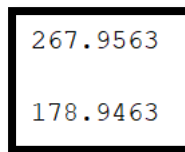
- e) Based on the equation of the line $Ax + By = C$ that you obtained, compute y_l and y_r values for corresponding $x_l = 0$ and $x_r = \text{width of image} - 1$.

Code:

```
xl = 0;
xr = numOfCols - 1;

% Equation to find y value: y = (C - Ax)/B
yl = (C - A*xl)/B;
yr = (C - A*xr)/B;
|
disp(yl);
disp(yr);
```

Fig 2.5.1 Code to run

Result:

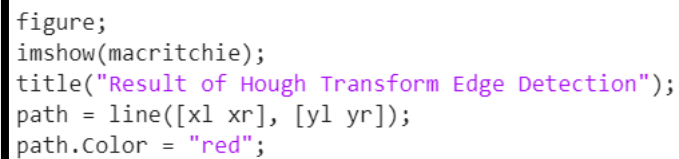
267.9563
178.9463

Fig 2.5.2 Result of yl and yr

- f) Display the original 'macritchie.jpg' image. Superimpose your estimated line by

```
>> line([xl xr], [yl yr]);
```

Does the line match up with the edge of the running path? What are, if any, sources of errors? Can you suggest ways of improving the estimation?

Code:

```
figure;  
imshow(macritchie);  
title("Result of Hough Transform Edge Detection");  
path = line([xl xr], [yl yr]);  
path.Color = "red";
```

Fig 2.6.1 Code to run

Result:

Fig 2.6.2 *Detected running track Edge*

Answer:

From the result, we can see that the estimated edge is aligned with the running track but if we see it closely we can found out that the edge is almost but not perfectly aligned to the running track.

These are the several possibilities that I can think of:

1. The line of the running track is not necessarily straight, maybe there will be some curve in between
2. There can be small precision error conversion from Radon transform parameters to the coordinate in image space.
3. There might be some noises that will affect to get the maximum intensity pixel.

Suggestion:

1. We can try to use non-linear function when the running track is not totally straight.
2. Try using a larger sigma value to reduce the noises on the picture

3. 3D Stereo

The pixel intensity sum-of-squares difference (SSD) between a small image patch T and a large image I at the (x,y) location given by

$$S(x, y) = \sum_{j=0}^M \sum_{k=0}^N (I(x+j, y+k) - T(j, k))^2$$

By computing the SSD at *all locations* of the image I , we can generate a SSD image $S(x,y)$. This allows the location in the image that best matches the image patch T to be found.

Evaluating the above equation directly can be very time consuming. An alternative approach is to decompose the equation into 3 parts

$$S(x, y) = \sum_{j=0}^M \sum_{k=0}^N I^2(x+j, y+k) + \sum_{j=0}^M \sum_{k=0}^N T^2(j, k) - 2 \sum_{j=0}^M \sum_{k=0}^N I(x+j, y+k) T(j, k)$$

The second term is constant with respect to different parts of the image, while the first and third terms can be expressed as convolution. Convolution on large images can be efficiently computed via FFTs.

In stereo vision, the goal is to compute the relative depth of a 3D point from the stereo cameras. If rectified images (i.e. where projected 2D points from the same 3D point have the same y coordinate in different images -- they lie on the same scanline) are used, we can express depth in terms of disparity. Disparity is the difference between the x coordinate of the projections in two images, and is inversely proportional to depth.

Given two images, stereo vision allows the computation of a *disparity map*. A disparity map is simply an array containing the disparities or depths that are associated with every pixel in one image (typically the left image).

This is a fairly substantial section as you will need to write a MATLAB function script to compute disparity images (or *maps*) for pairs of rectified stereo images P_l and P_r . The disparity map is inversely proportional to the depth map which gives the distance of various points in the scene from the camera.

Estimating Disparity Maps

The overview of the algorithm is:

for *each* pixel in P_l ,

- i. Extract a template comprising the 11x11 neighbourhood region around that pixel.
- ii. Using the template, carry out SSD matching in P_r , but *only along the same scanline*. The disparity is given by

$$d(x_l, y_l) = x_l - \hat{x}_r$$

where x_l and y_l are the relevant pixel coordinates in P_l , and \hat{x}_r is the SSD matched pixel's x-coordinate in P_r . You should also constrain your horizontal search to small values of disparity (<15).

Noted that you may use **conv2**, **ones** and **rot90** functions (may be more than once) to compute the SSD matching between the template and the input image. Refer to the equation in section 2.3 for help.

- iii. Input the disparity into the disparity map with the same P_l pixel coordinates.

- a) Write the disparity map algorithm as a MATLAB function script which takes two arguments of left and right images, and 2 arguments specifying the template dimensions. It should return the disparity map. Try and minimize the use of for loops, instead relying on the vector / matrix processing functions.

Code:

```

function [result] = disparityMap(pLeft, pRight, rowDim, colDim)
    % Convert image matrix to double
    pLeft = im2double(pLeft);
    pRight = im2double(pRight);

    % Obtain the dimension of left image
    [height, width] = size(pLeft);

    % Get the disparity range and the template dimensions
    rowLength = floor(rowDim/2);
    colLength = floor(colDim/2);
    dispRange = 15;

    % Initialise matrix of 0s
    result = zeros(size(pLeft));

    % Outer loop - each pixel along the matrix column
    for i = 1:height
        % variables to keep the range in check
        minRow = max(1, i - rowLength);
        maxRow = min(height, i + rowLength);

        % loop for each pixel along the matrix row
        for j = 1:width
            % variables to keep the range in check
            minCol = max(1, j - colLength);
            maxCol = min(width, j + colLength);

            % variables to keep the horizontal search range in check
            minDisp = max(-dispRange, 1 - minCol);
            maxDisp = min(dispRange, width - maxCol);
            |
            % Obtain the template from the right image
            dispTemplate = pRight(minRow:maxRow, minCol:maxCol);

            % Initialise the variables for SSD comparison
            minSSD = inf;
            leastDifference = 0;

            % Inner loop - to do the searching in the search range
            for k = minDisp:maxDisp
                % Get the difference between left and right images
                newMinCol = minCol + k;
                newMaxCol = maxCol + k;
                block = pLeft(minRow:maxRow, newMinCol:newMaxCol);

                % Perform SSD
                squaredDifference = (dispTemplate - block).^2;
                ssd = sum(squaredDifference(:));

                % Get the lowest SSD
                if ssd < minSSD
                    minSSD = ssd;
                    leastDifference = k - minDisp + 1;
                end
            end

            % Return the SSD result
            result(i, j) = leastDifference + minDisp - 1;
        end
    end
end

```

Fig 2.1 Disparity Map Function Code

- b) Download the synthetic stereo pair images of 'corridorl.jpg' and 'corridorr.jpg', converting both to grayscale.

Code:

```
P_corrleft = imread('corridorl.jpg');  
P_corrright = imread('corridorr.jpg');  
  
% Convert to grayscale  
P_corrleft_gray = rgb2gray(P_corrleft);  
P_corrright_gray = rgb2gray(P_corrright);  
  
figure;  
imshow(P_corrleft_gray);  
title("Corridor Left");  
  
figure;  
imshow(P_corrright_gray);  
title("Corridor Right");
```

Fig 3.2.1 Code to run

Result:



Fig 3.2.2 Corridor Left



Fig 3.2.3 Corridor Right

- c) Run your algorithm on the two images to obtain a disparity map D , and see the results via

```
>> imshow(-D,[-15 15]);
```

The results should show the nearer points as bright and the further points as dark. The expected quality of the image should be similar to 'corridor_disp.jpg' which you can view for reference.

Comment on how the quality of the disparities computed varies with the corresponding local image structure.

Code:

```
D = disparityMap(P_corrleft_gray, P_corrright_gray, 11, 11);  
figure;  
imshow(D,[-15 15]);  
title("Result: Disparity Map D of Corridor");
```

Fig 3.3.1 Code to run

Result:



Fig 3.3.2 Result of Corridor after Disparity Map D

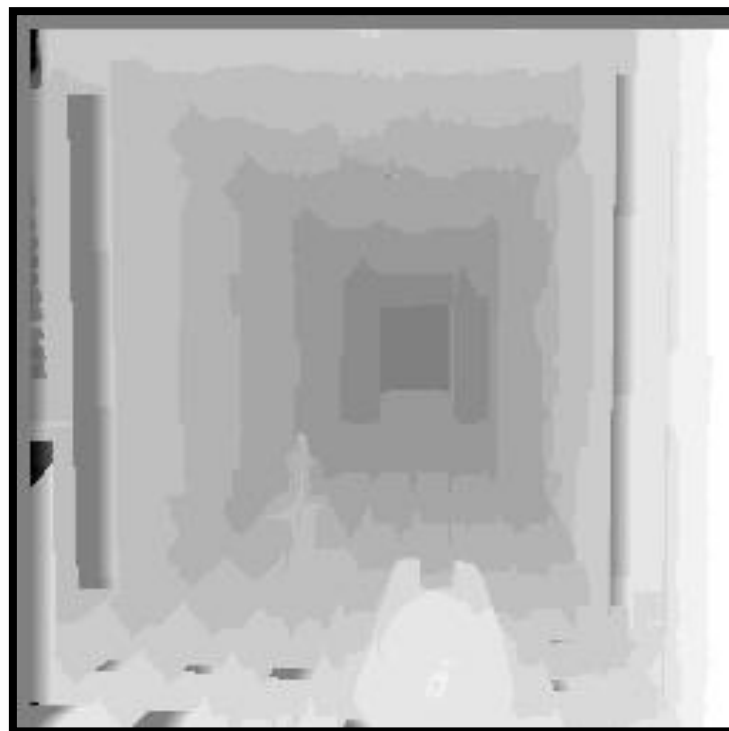


Fig 3.3.3 Expected Output of Corridor

Answer:

The result disparity map is similar to the expected disparity map. As expected, the pixel intensity decreases (gets darker) as we reach the centre of the image since the disparity decreases for far away objects. One area that can be improved upon is the centre of the image. Although the pixel intensities there are dark indicating low disparity which is correct, the results are not uniform across that section. This is due to the homogenous colour of that part of the image.

- d) Rerun your algorithm on the real images of 'triclops-i2l.jpg' and 'triclops-i2r.jpg'. Again you may refer to 'triclops-id.jpg' for expected quality. How does the image structure of the stereo images affect the accuracy of the estimated disparities?

Code:

```
P_triclops_left = imread('triclopsi2l.jpg');
P_triclops_right = imread('triclopsi2r.jpg');

% Convert to grayscale
P_triclops_left_gray = rgb2gray(P_triclops_left);
P_triclops_right_gray = rgb2gray(P_triclops_right);

D_triclops = disparityMap(P_triclops_left_gray, P_triclops_right_gray, 11, 11);
figure;
imshow(D_triclops, [-15 15]);
title("Result: Disparity Map D of Triclops");
```

Fig 3.4.1 Code to run

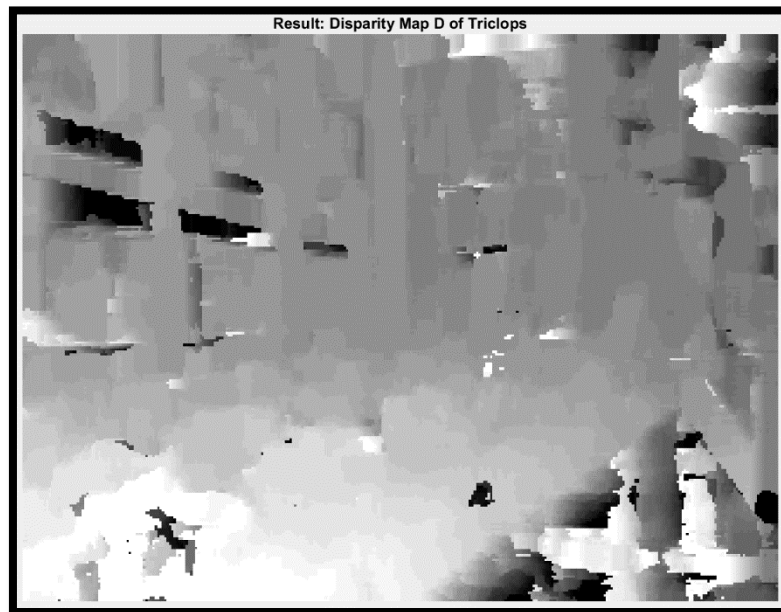
Result:

Fig 3.4.2 Result of Triclops after Disparity Map D



Fig 3.4.3 Expected Output of the Triclops

Answer:

The result disparity map we obtained is quite close to the expected disparity map. However, this result is not desirable as the original building structure edge information are lost in the disparity map. To conclude, appearance-based disparity maps like the one we have implemented have trouble calculating the correct disparity if there is a homogenous surface with the same intensity.