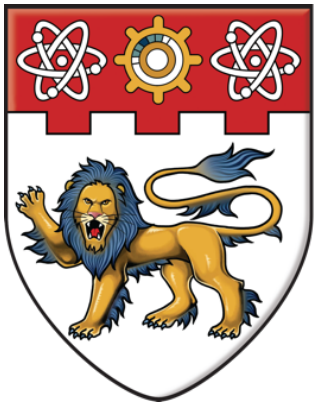


NANYANG TECHNOLOGICAL UNIVERSITY

CZ4003 Computer Vision

Laboratory 2

Edge Detection + Hough Transform + 3D Stereo

DAI JUNWEI

N902106B

Contents

1. Introduction

1.1 Objectives

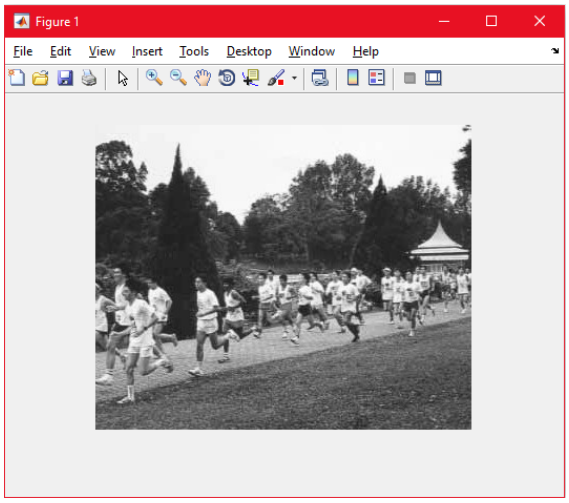
This laboratory aims to introduce image processing in MATLAB context. In this laboratory you will:

a) Explore different edge detection methods;
b) Employ the Hough Transform to recover strong lines in the image;
c) Experiment with pixel sum-of-squares difference (SSD) to find a template match within a larger image. Estimate disparity maps via SSD computation;
d) Optionally, compare the bag-of-words method with spatial pyramid matching (SPM) on the benchmark Caltech-101 dataset.

2. Experiments

2.1 Edge Detection

a) Download `macritchie.jpg' from edveNTUre and convert the image to grayscale. Display the image.
Result and Comments:




```
pic = imread('./images/macritchie.jpg');
pic_gray = rgb2gray(pic);
imshow(pic_gray)
```

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?

Result and Comments:

Before creating the 3x3 Horizontal and Vertical Sobel masks, we need to know what it is and how it looks like.

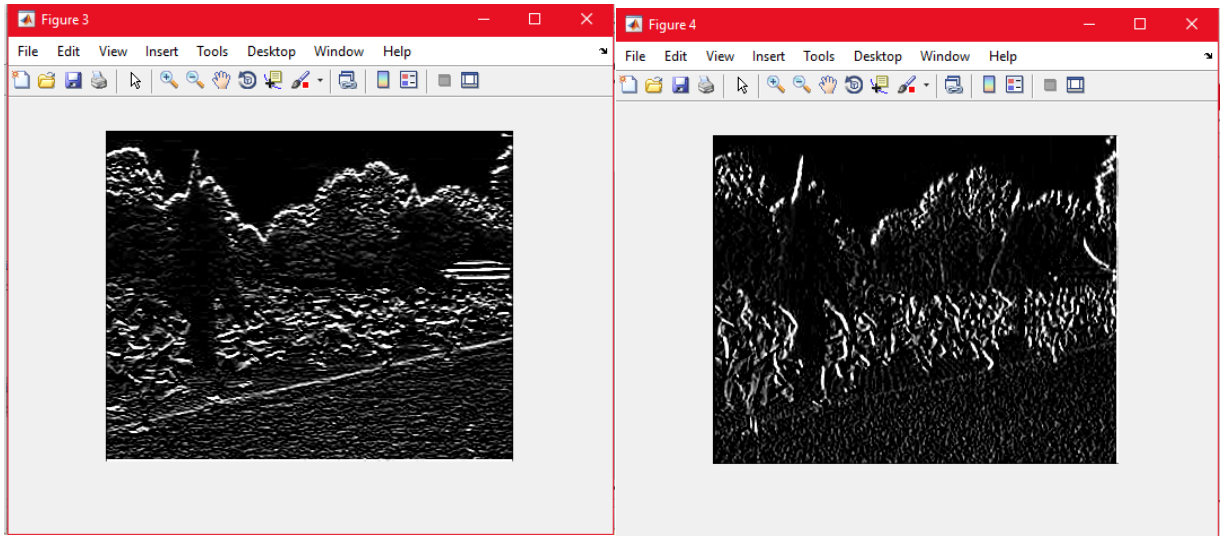
-1	-2	-1
0	0	0
+1	+2	+1

-1	0	+1
-2	0	+2
-1	0	+1

Vertical Jumps Horizontal Jumps

Therefore, just coding out the matrix for the vertical and horizontal jumps in shown below.

```
>> sobel_vertical = [-1 -2 -1; 0 0 0; 1 2 1];
>> sobel_vertical

sobel_vertical =

    -1    -2    -1
     0     0     0
     1     2     1

>> sobel_horizontal = [-1 0 1; -2 0 2; -1 0 1];
>> sobel_horizontal

sobel_horizontal =

    -1     0     1
    -2     0     2
    -1     0     1
```




```
conv_image_vertical = conv2(pic_gray, sobel_vertical);
conv_image_horizontal = conv2(pic_gray, sobel_horizontal);
figure, imshow(uint8(conv_image_vertical))
figure, imshow(uint8(conv_image_horizontal))
```

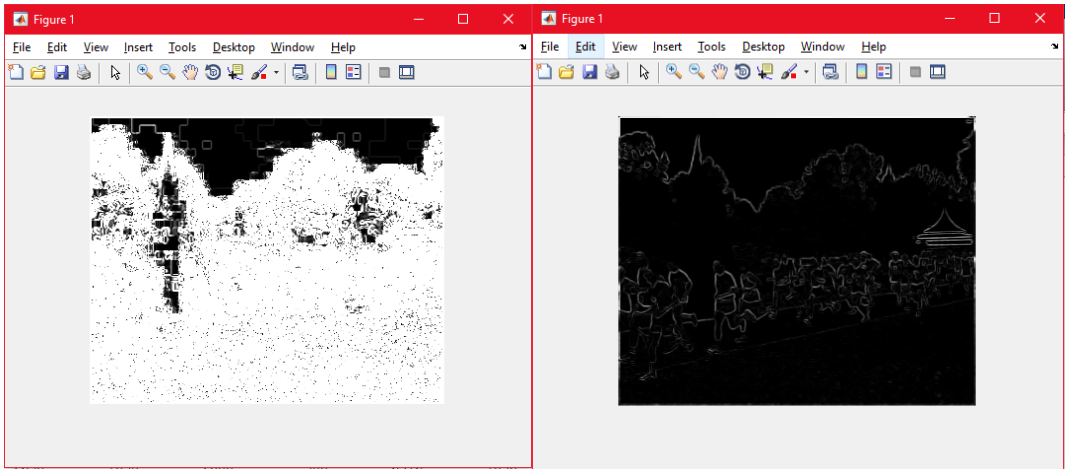
As we can see, the picture on the left is detecting vertical edges while the picture on the right is detecting horizontal edges. The diagonal edges in the image are removed as each filter is specifically only looking for a vertical or a horizontal jump.

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

Result and Comments:

```
conv_image_vertical = conv2(pic_gray, sobel_vertical);
conv_image_horizontal = conv2(pic_gray, sobel_horizontal);
combined_edge_image = conv_image_vertical.^2 + conv_image_horizontal.^2;
```

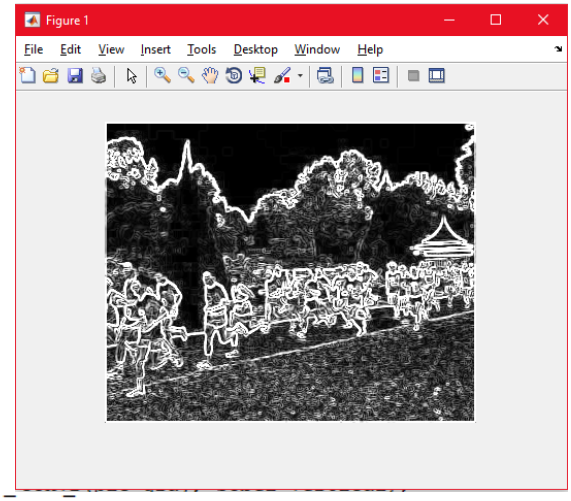
The image after combining them is as shown below. It seems we cannot make out what the picture is as some of the values in the matrix are way over 255. The values are huge.




```
imshow(uint8(combined_edge_image)) % The image is so bright. Cannot view anything
imshow((combined_edge_image),[]) % This function normalizes the image.
```

After using the imshow code as shown in the second line above, I was able to get how the picture look like when the vertical and horizontal sobel applied filters were combined which is shown on the right.

The squaring operation is done to get the gradient magnitude. The absolute gradient magnitude can be approximated by taking the element-wise squares of the filtered image and adding both together and finally taking the square root. Following the lecture notes we will get the following image as shown below and the codes are placed beside it.

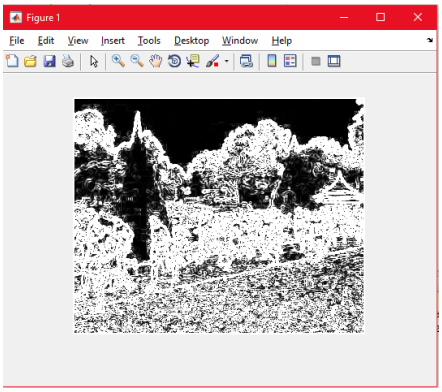



```
imshow(uint8(sqrt(combined_edge_image))) % Following Lecture Notes Squareroot(Horizontal^2 + Vertical^2)
```

d) Threshold the edge image E at value t by >> 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?

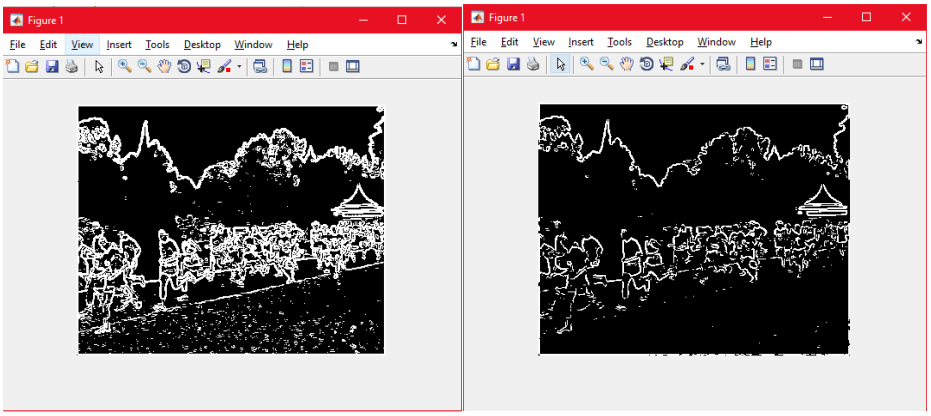
Result and Comments:

The original image after normalizing is shown below.

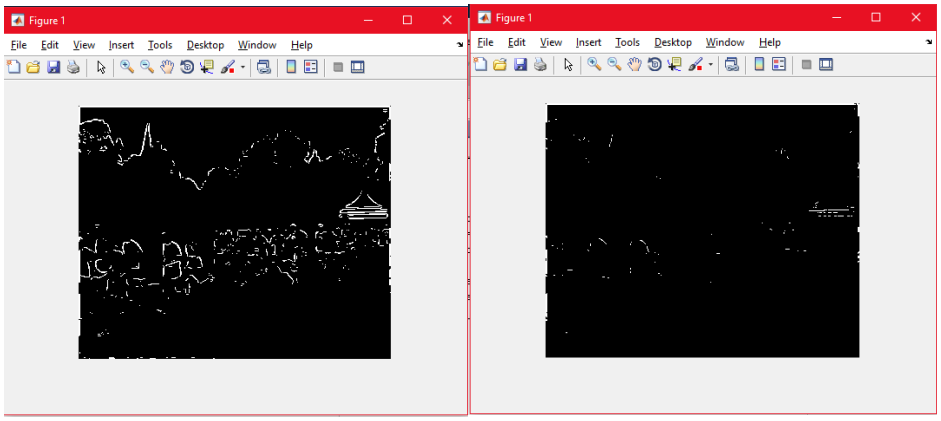



```
min_image = min(combined_edge_image(:));
max_image = max(combined_edge_image(:));
E = 255*(combined_edge_image - min_image)/(max_image-min_image);

Et = E > 20;
imshow(Et)
Et = E > 50;
imshow(Et)
Et = E > 100;
imshow(Et)
```

When threshold t = 5; and When threshold t = 20;



When threshold t = 50; and When threshold t = 100;

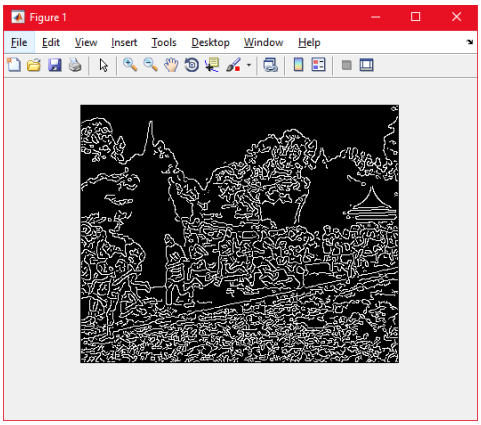


As the threshold value increases, the edges slowly disappear. However, if no threshold is done, it is hard to differentiate the edges from the noise. Thresholding it allows us to find a perfect balance between obvious edges and removing noise.

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);

Result and Comments:

This generates a binary image without the need for thresholding.

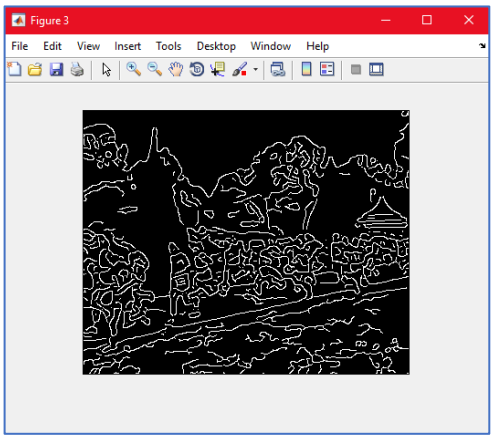



```
tl = 0.04;
th = 0.1;
sigma = 1.0;
CannyEdge = edge(pic_gray, 'canny', [tl th], sigma);
imshow(CannyEdge);
```

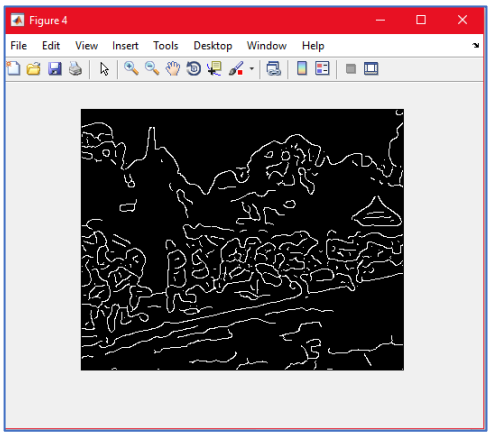
(i) Try different values of sigma ranging from 1.0 to 5.0 and determine the effect on the edge images. What do you see and can you give an explanation for why this occurs? Discuss how different sigma are suitable for (a) noisy edgel removal, and (b) location accuracy of edgels.

Result and Comments:

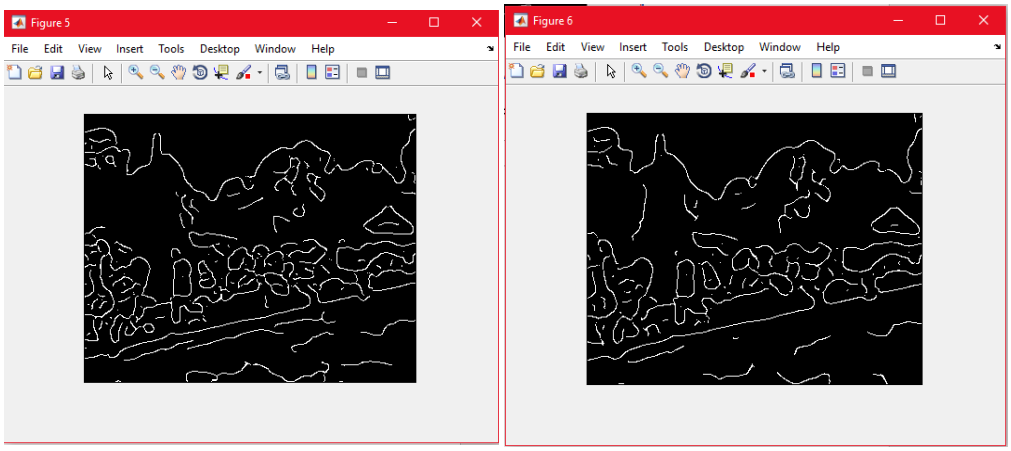
When sigma = 2.0;



When sigma = 3.0;



When sigma = 4.0; When sigma = 5.0;



As sigma increases, more edge details and noise disappear causing the location accuracy of the edges to be lower as well.

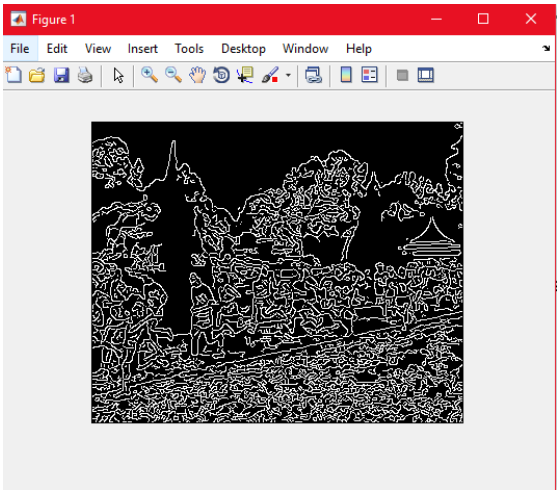
```
% Part E
tl = 0.04;
th = 0.1;
sigma = 1.0;
sigma_list = [1.0 2.0 3.0 4.0 5.0];
tl_list = [0.01 ,0.04, 0.09];

CannyEdge1 = edge(pic_gray, 'canny', [tl th], sigma_list(1));
CannyEdge2 = edge(pic_gray, 'canny', [tl th], sigma_list(2));
CannyEdge3 = edge(pic_gray, 'canny', [tl th], sigma_list(3));
CannyEdge4 = edge(pic_gray, 'canny', [tl th], sigma_list(4));
CannyEdge5 = edge(pic_gray, 'canny', [tl th], sigma_list(5));

figure, imshow(CannyEdge1);
figure, imshow(CannyEdge2);
figure, imshow(CannyEdge3);
figure, imshow(CannyEdge4);
figure, imshow(CannyEdge5);
```

(ii) Try raising and lowering the value of tl. What does this do? How does this relate to your knowledge of the Canny algorithm?

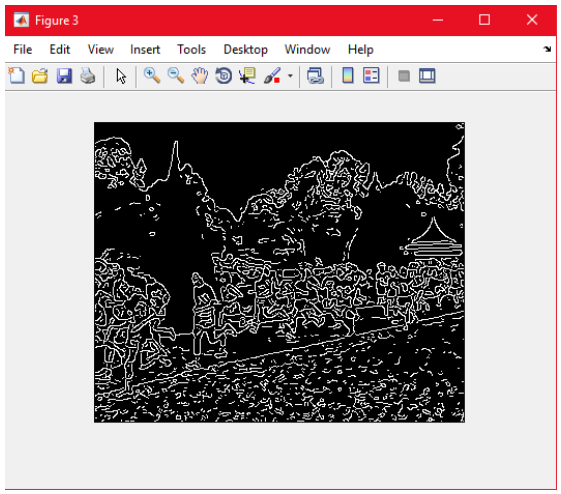
Result and Comments:

```
CannyEdge6 = edge(pic_gray, 'canny', [(tl_list(1)) th], sigma(1));
CannyEdge7 = edge(pic_gray, 'canny', [(tl_list(2)) th], sigma(1));
CannyEdge8 = edge(pic_gray, 'canny', [(tl_list(3)) th], sigma(1));

figure, imshow(CannyEdge6);
figure, imshow(CannyEdge7);
figure, imshow(CannyEdge8);
```

When tl = 0.01;



When tl = 0.09;



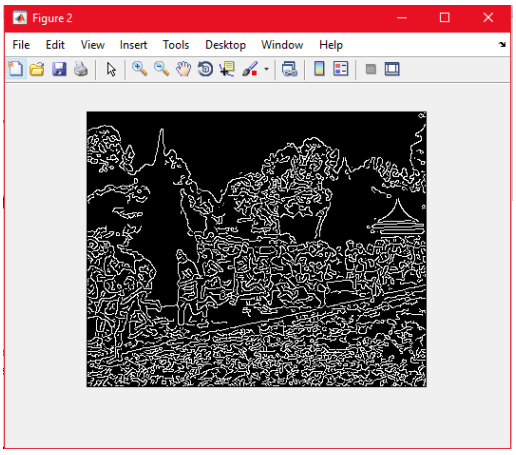
The thresholding Canny Edge Detector uses is known as Hysteresis Thresholding. It involves 3 different conditions to determine if the pixel should be considered an edge. Values below the threshold will be ignored. i.e. any value below tl=0.01 or tl=0.09 will not be considered and if a pixel value is above the threshold value, it will be considered an edge. The pixel values which fall in between the threshold_low and threshold_high will be selected as an edge based on the condition of it's neighboring pixels. If the neighboring pixels are considered as an edge, it will also be considered as an edge. If the pixel that is tested is isolated (not have any edge beside), it will not be considered as an edge and will be removed.

2.2 Line Finding using Hough Transform

In the section, the goal is to extract the long edge of the path in the `macritchie.jpg' image as a consistent line using the Hough transform.

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

Result and Comments:




```
% 2.2 Hough Transform

% Part A
    pic = imread('macritchie.jpg');
    pic_gray = rgb2gray(pic);
    tl = 0.04;
    th = 0.1;
    sigma = 1.0;
    E = edge(pic_gray, 'canny', [tl, th], sigma);
```

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?

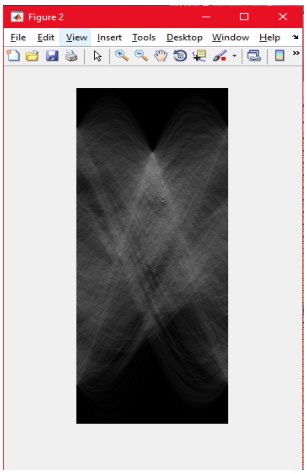
```
>> [H, xp] = radon(E);
```

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.

Result and Comments:

The Help Manual as shown in the console.

```
>> help radon
 radon Radon transform.
    The radon function computes the Radon transform, which is the
    projection of the image intensity along a radial line oriented at a
    specific angle.

    R = radon(I,THETA) returns the Radon transform of the intensity image I
    for the angle THETA degrees. If THETA is a scalar, the result R is a
    column vector containing the Radon transform for THETA degrees. If
    THETA is a vector, then R is a matrix in which each column is the Radon
    transform for one of the angles in THETA. If you omit THETA, it
    defaults to 0:179.

    [R,Xp] = radon(...) returns a vector Xp containing the radial
    coordinates corresponding to each row of R.

    Class Support
    -------------
    I can be of class double, logical or of any integer class and must be
    two-dimensional. THETA is a vector of class double.  Neither of the
    inputs can be sparse.
```



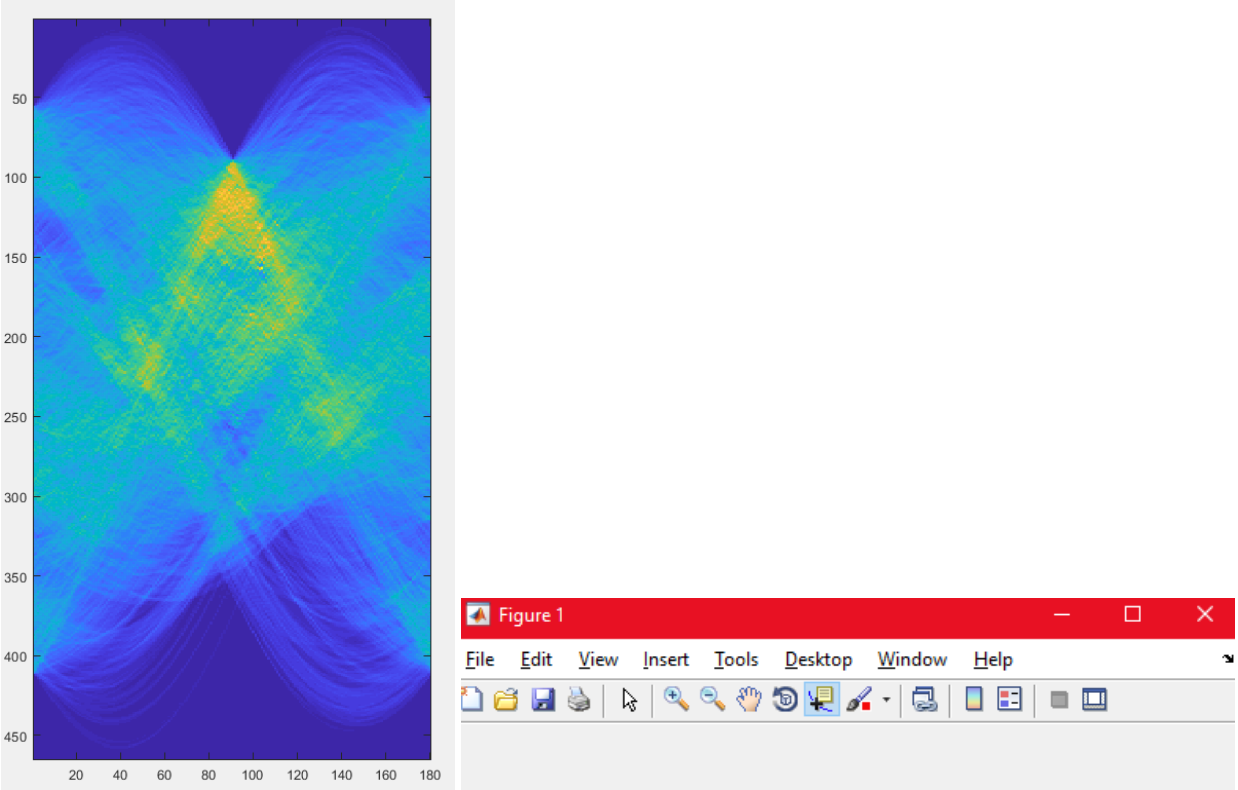

```
% Part B
    [H, xp] = radon(E);
    imshow(uint8(H))
```

The Radon transform can be used to detect lines in the image and is stated by the documentation that it is related to the Computer Vision operation known as the Hough Transform. Doing further research, it is said that the Hough Transform is Discrete while the Radon is a Continuous. Applying a Discrete or Continuous input into the Radon transform function will produce a different outcome. If you applied a Discrete input to the Radon method, it will produce the Discrete Radon Transform.

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.

Result and Comments:

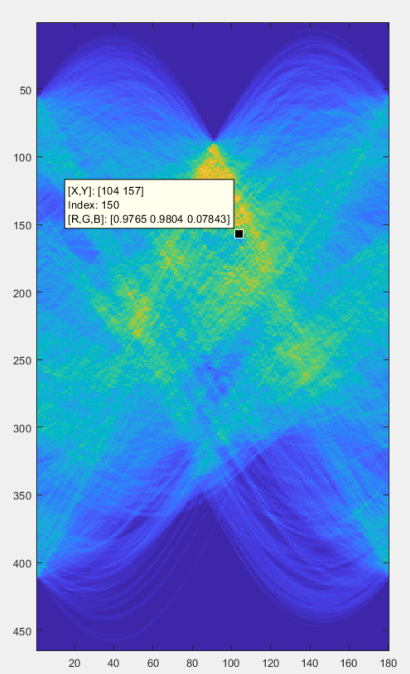
The X axis is represented by the values of theta, while the Y axis is represented by the radius value. Viewing it using imagesc(H) and we can view it in the default colormap.



The image is bigger if not it will be difficult to see where the peaks are. There are a lot of peaks centered around but to find the highest peak. I will look for the one with the highest intensity. Using the “Data Cursor” Icon (highlighted with a blue square) I was able to place a square and immediately know the X axis as well as the Y axis and the RGB intensities. The Icon can be found in the Figures toolbar when a new figure is shown using imagesc. So testing the multiple “Yellow” peaks. The one that has the highest intensity was the one that is at

Theta = 104; and Radius = 157; Therefore, [theta, radius] = [104, 157].

This can be shown in the image below with the black square after using “Data Cursor”.



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.

Result and Comments:

```
% Part D
    % Center of the pic is (290 / 2) = 145 by (358 / 2) = 179 - Y by X
    theta = 104;
    radius = xp(157);
    [A, B] = pol2cart(theta*pi/180, radius);
    B = -B;
    C = A*(A+179) + B*(B+145);
```

Since the formula for A and B is given, we can use it to find A and B. We need to first get the respective values of theta as well as the radius based on the highest pixel intensity found in the previous section. The method provided converts polar coordinates to cartesian coordinates. The center coordinates are gotten by taking the image NxM dimensions and dividing each individual dimension. Therefore, since the image is 290x358, the center of it will be 290/2 by 358/2 which then equals to (145, 179). In Matlab the image are read in as YxX therefore, the respective shift for X is 179 while the Y shift is 145.

e) Based on the equation of the line Ax+By = C that you obtained, compute yl and yr values for corresponding xl = 0 and xr = width of image - 1.

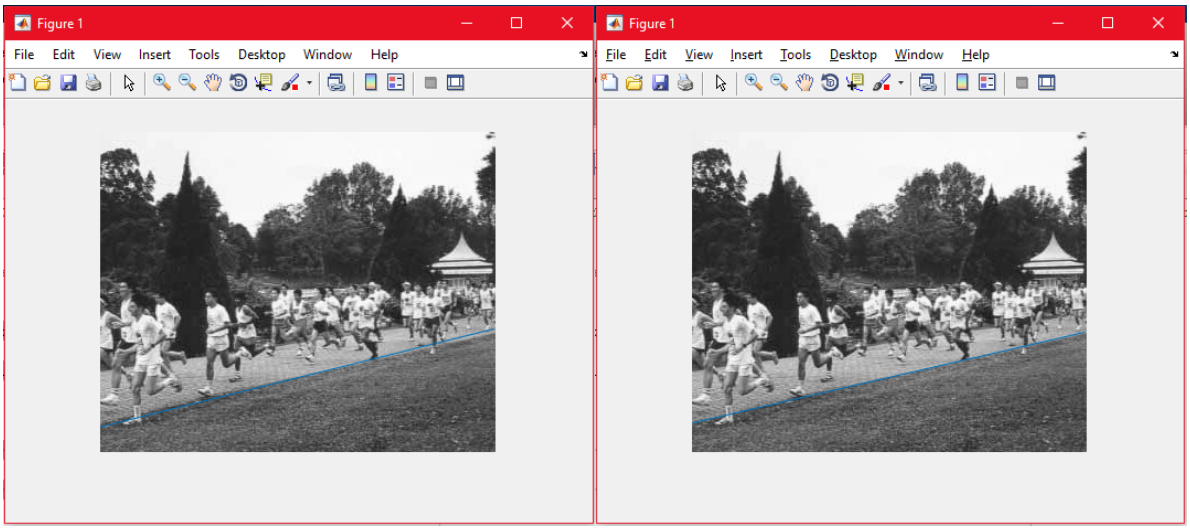
Result and Comments:

The width of X is 358. Therefore, xr = 358 – 1 = 357. Rearranging the equations we can solve for y1.

```
% Part E
    xl = 0;
    yl = (C - A * xl) / B;
    xr = 357;
    yr = (C - A * xr) / B;
```

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?

Result and Comments:



The line on the left picture does not match up on the line perfectly. This may be because at the early stage of estimating the X and Y coordinates of the highest pixel intensity, I may have misjudged or click on the one that I think was the highest and did not check the surrounding. After checking, I realized that the value of theta should be 103 instead. And re-writing the code, I was able to get the line to fit much better.

```
% Correction for part D to F.
theta = 103;
radius = xp(157);
[A, B] = pol2cart(theta*pi/180, radius);
B = -B;
C = A*(A+179) + B*(B+145);

xl = 0;
yl = (C - A * xl) / B;
xr = 357;
yr = (C - A * xr) / B;

imshow(pic_gray)
line([xl xr], [yl yr])
```

2.3 3D Stereo Vision

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 Pl and Pr. 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:

i. for each pixel in Pl,

ii. Extract a template comprising the 11x11 neighbourhood region around that pixel.

Using the template, carry out SSD matching in Pr, but only along the same scanline. The disparity is given by

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

where xl and yl are the relevant pixel coordinates in Pl, and x_hat is the SSD matched pixel's x-coordinate in Pr. You should also constrain your horizontal search to small values of disparity (<15).

Note 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 Pl 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.

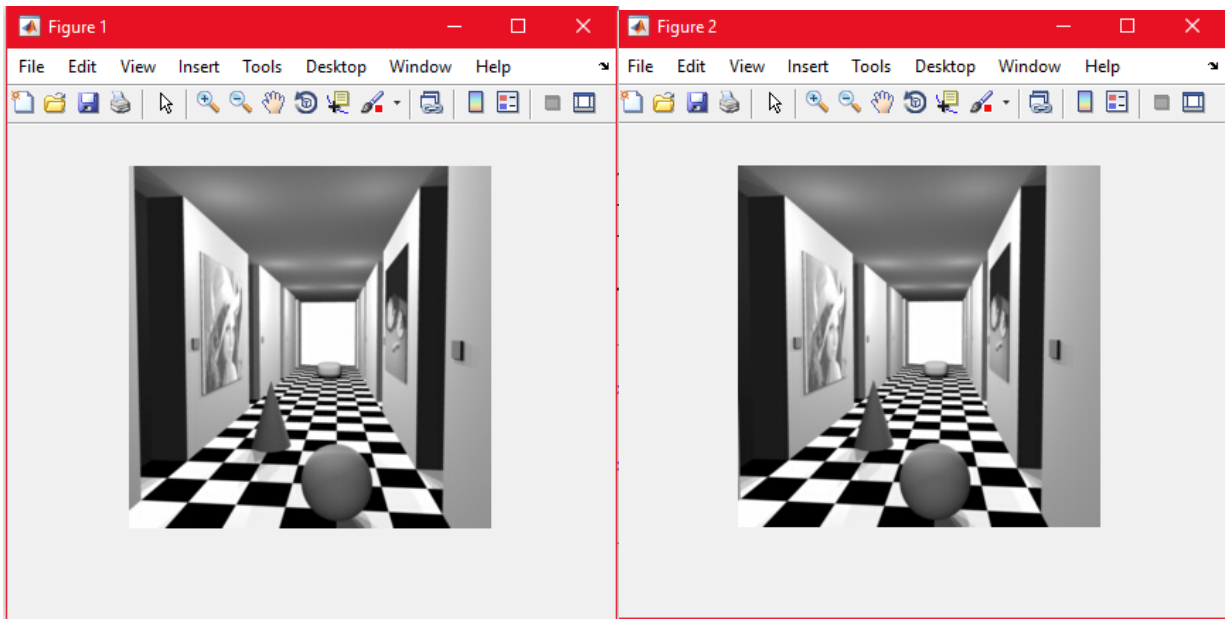
Result and Comments:

```
function disparity = map(left_image, right_image, x_temp, y_temp)
    % Divide template size by 2 as 2k+1 need to get k to form the KxK filter
    x_dim = floor(x_temp/2);
    y_dim = floor(y_temp/2);

    [x1, y1] = size(left_image);
    [x2, y2] = size(right_image);

    if size(left_image) ~= size(right_image)
        error("Please ensure that the left image and right image has the same size")
    else
        % Initialize the disparity map size
        disparity = ones(x1 - x_temp + 1, y1 - y_temp + 1);
        for i = 1 + x_dim : x1 - x_dim
            for j = 1 + y_dim : y1 - y_dim
                cur_r = left_image(i - x_dim : i + x_dim, j - y_dim : j + y_dim);
                cur_l = rot90(cur_r, 2);
                min_coor = j;
                min_diff = inf;

                for k = max(1 + y_dim, j - 14) : j
                    T = right_image(i - x_dim: i + x_dim, k - y_dim: k + y_dim);
                    cur_r = rot90(T, 2);

                    conv_1 = conv2(T, cur_r);
                    conv_2 = conv2(T, cur_l);

                    ssd = conv_1(x_temp, y_temp) - 2 * conv_2(x_temp, y_temp);
                    if ssd < min_diff
                        min_diff = ssd;
                        min_coor = k;
                    end
                end
                disparity(i - x_dim, j - y_dim) = j - min_coor;
            end
        end
    end
return
```

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

Result and Comments:

```
% Part B
    l = imread('./images/corridorl.jpg');
    l = rgb2gray(l);
    figure, imshow(l);
    r = imread('./images/corridorr.jpg');
    r = rgb2gray(r);
    figure, imshow(r);
```

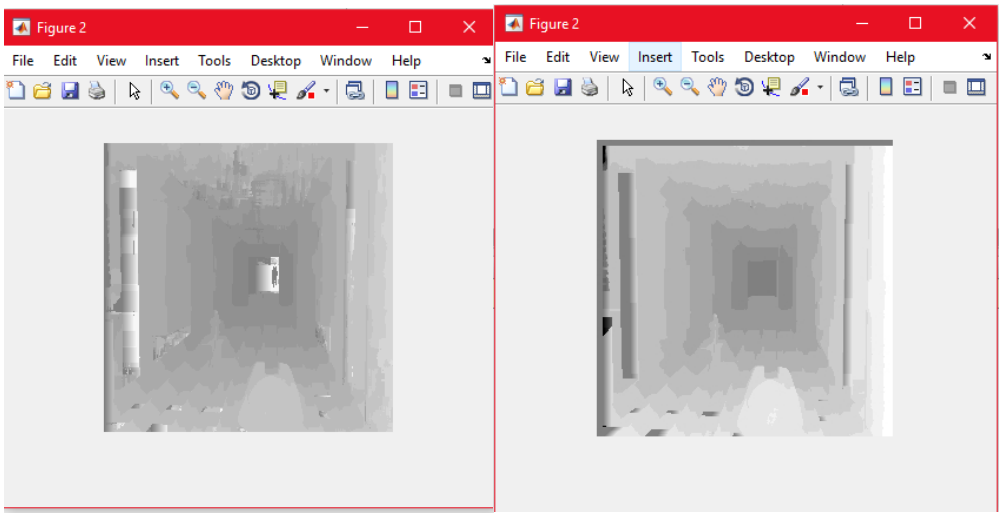


Left and Right images placed respectively

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.

Result and Comments:

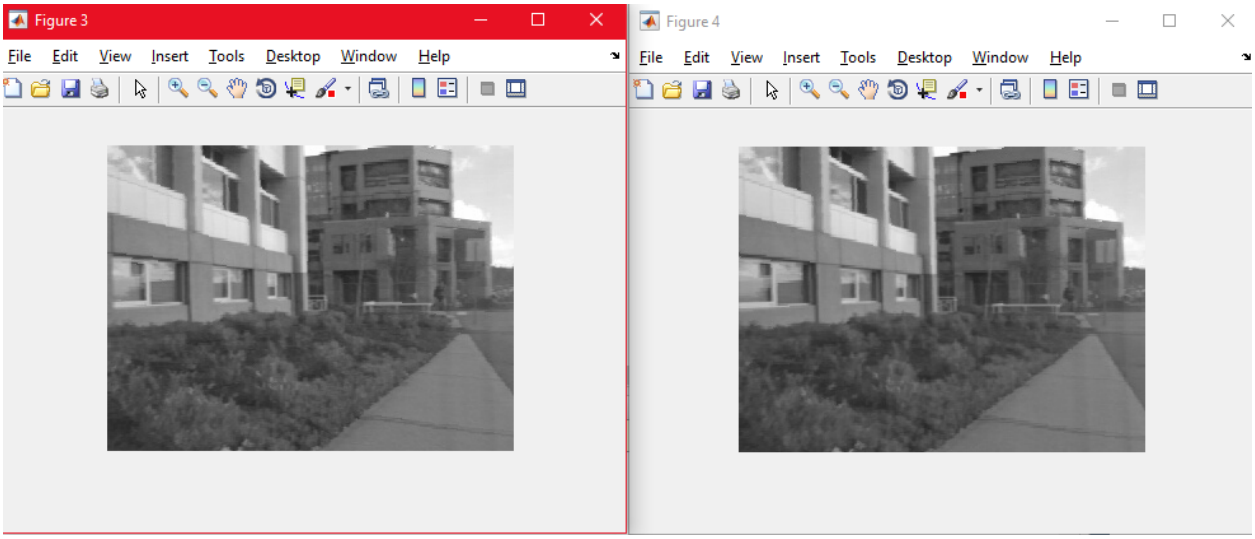
Comment on how the quality of the disparities computed varies with the corresponding local image structure.



The left image shows the disparity map created by the function map defined in Part A. We can see it is not perfect but as the image goes further into the distance, it becomes darker.

f) 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?

Result and Comments:



Original Left and Right images respectively

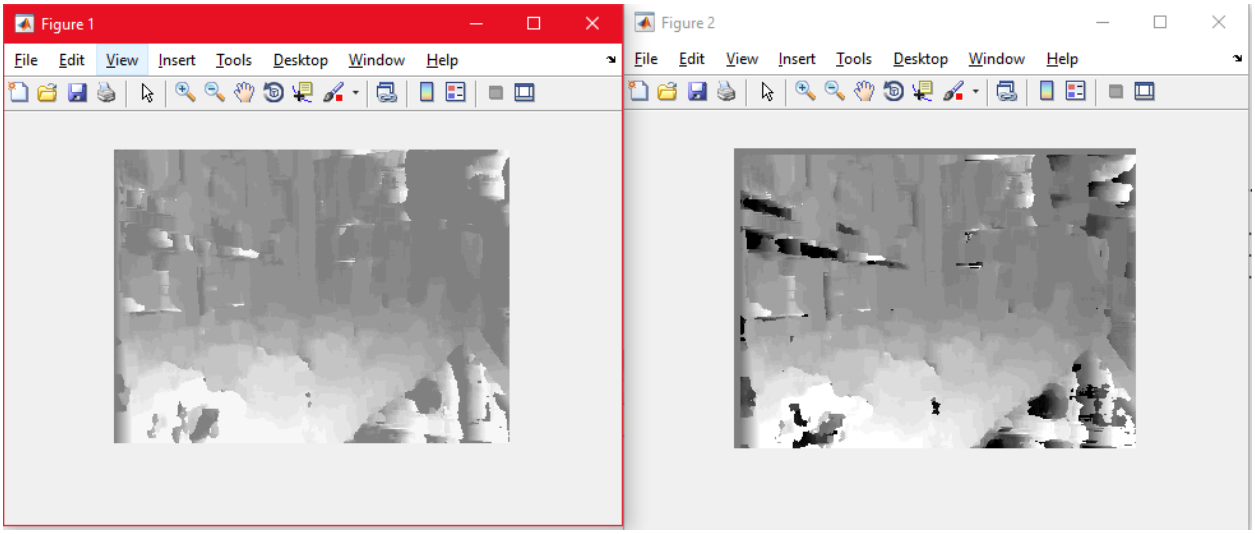


Figure 1 is the disparity mapped done by the function created in Part A. While Figure 2 shows the comparison disparity map. From the results we can see that the image may not seem to be correctly mapped at some regions. This may be due to the case that the quality of the image in Part C was much better than the images in this section. We do see a little bit of blur in the images in Part E i.e. this section.