NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CE4003 Computer Vision

Lab 2 Report

Elayne Tan Hui Shan

U1921730C

Table of Contents

1 Objective 3

2 Experiment 4

2.1 Edge Detection 4

2.2 Line Finding using Hough Transform 12

2.3 3D Stereo 19

2.4 Spatial Pyramid Matching and Bag-of-Words 23

1 Objective

This laboratory aims to provide further exposure to advanced image processing and computer vision techniques in MATLAB. In this laboratory, we 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 Experiment

2.1 Edge Detection

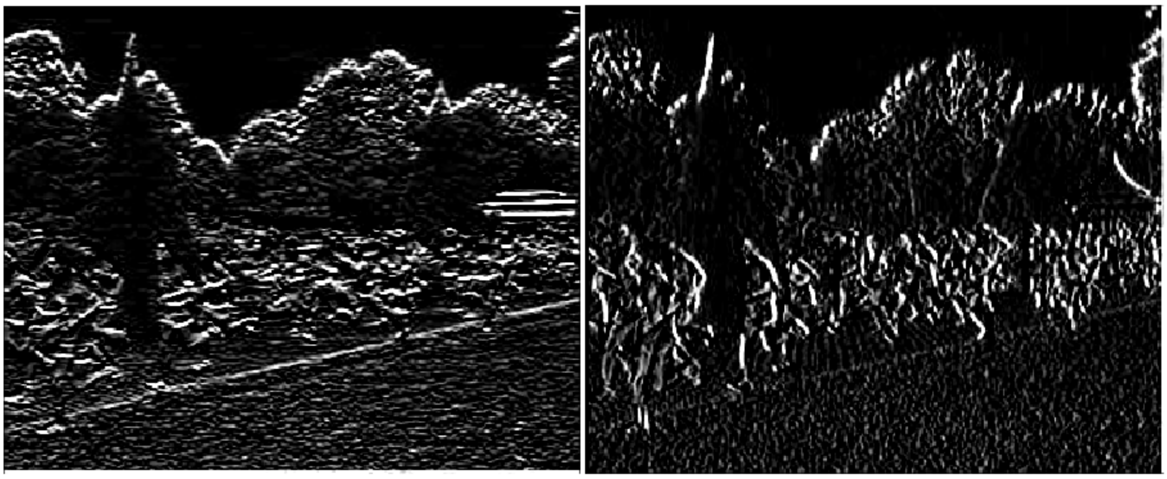
a) Download `macritchie.jpg' and convert the image to grayscale. Display the image.

```
macritchie = imread('Images/MacRitchie Cross-Country Race.jpg');
macritchie = rgb2gray(macritchie);
imshow(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?

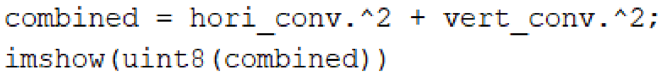
```
hori_sobel = [
    -1 -2 -1;
    0 0 0;
    1 2 1
];
vert_sobel = [
    -1 0 1;
    -2 0 2;
    -1 0 1
];
hori_conv = conv2(macritchie, hori_sobel);
vert_conv = conv2(macritchie, vert_sobel);
imshow(uint8(hori_conv))
imshow(uint8(vert_conv))
```

The edge-filtered images are shown below (left: hori_conv, right: vert_conv).

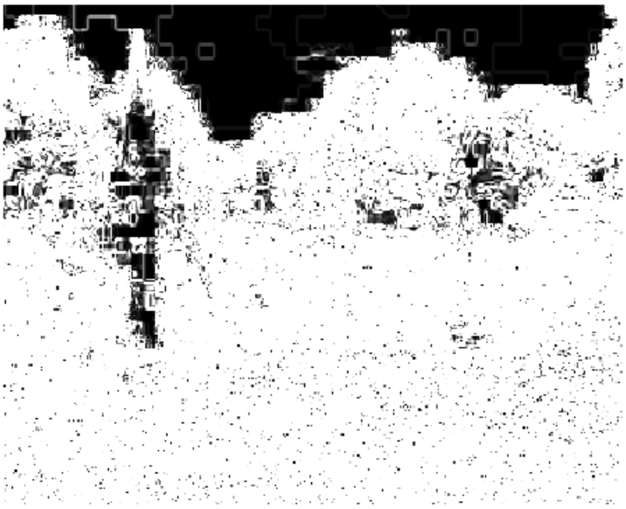


The horizontal and vertical Sobel masks detect horizontal and vertical edges respectively. Diagonal edges are removed as the masks do not detect diagonal edges and only look for vertical and horizontal jumps. Some diagonal edges are still present but fainter.

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.

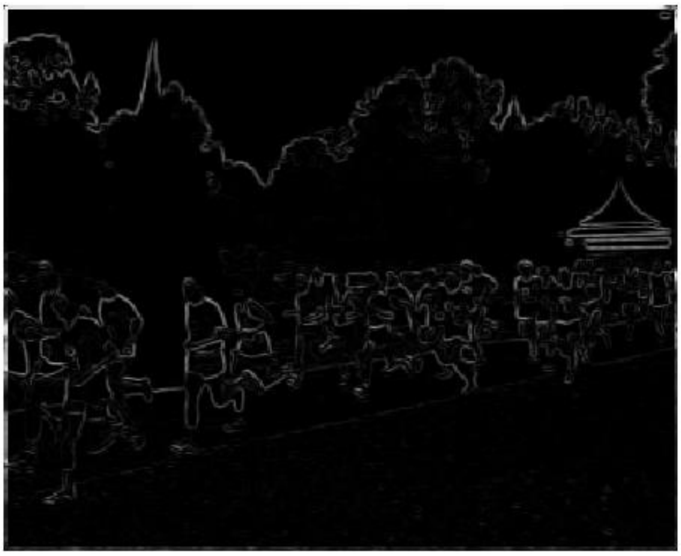


```
combined = hori_conv.^2 + vert_conv.^2;
imshow(uint8(combined))
```



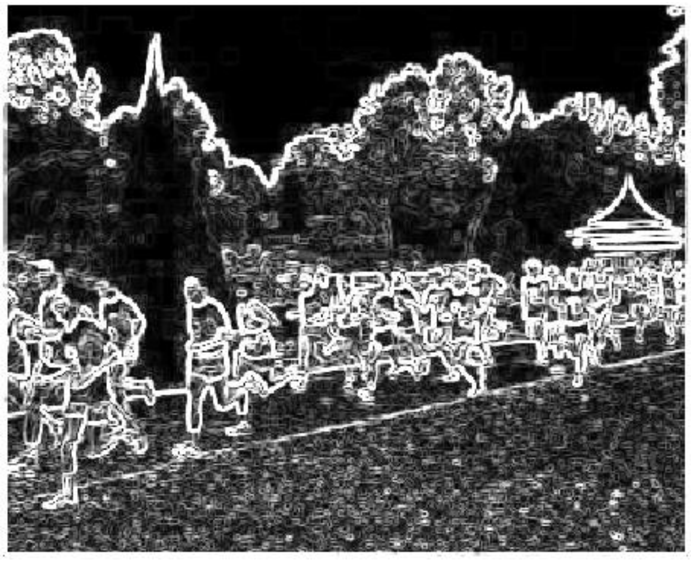
It is hard for us to conduct observations as the values are too large. Hence, we implement the code below to normalize the image.

```
imshow((combined), [])
```



We then take square root to approximate the absolute gradient magnitude.

```
imshow(uint8(sqrt(combined)))
```



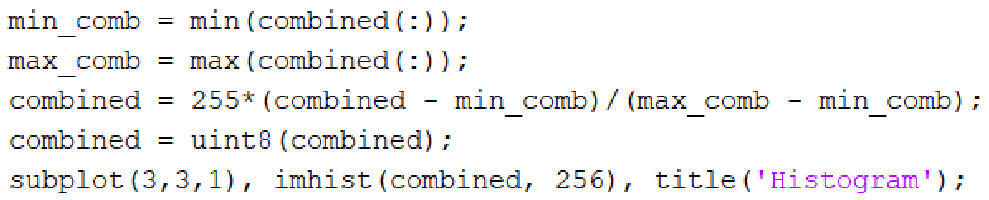
The squaring operation is carried out to calculate the gradient magnitude in its direction. This is done by taking the element-wise squares of hori_conv and vert_conv and adding them together, then taking the square root. The horizontal and vertical Sobel masks may generate negative values, hence, to take both positive and negative values into account, we perform squaring to obtain full information of the detected edges.

d) Threshold the edge image E at value t by

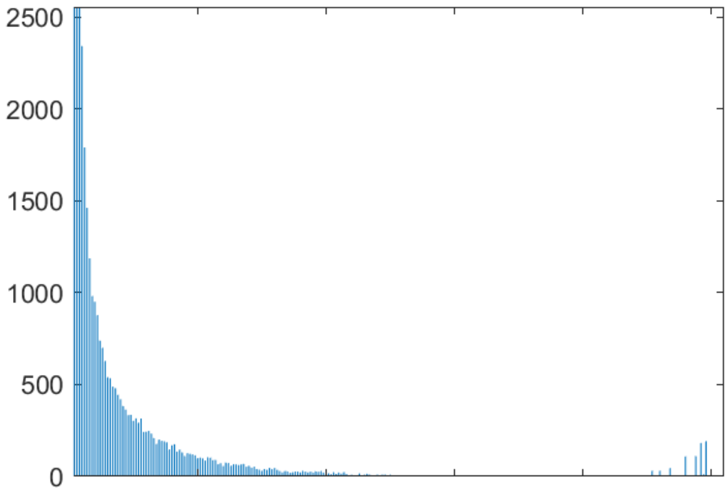
>> Et = E>t;

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?

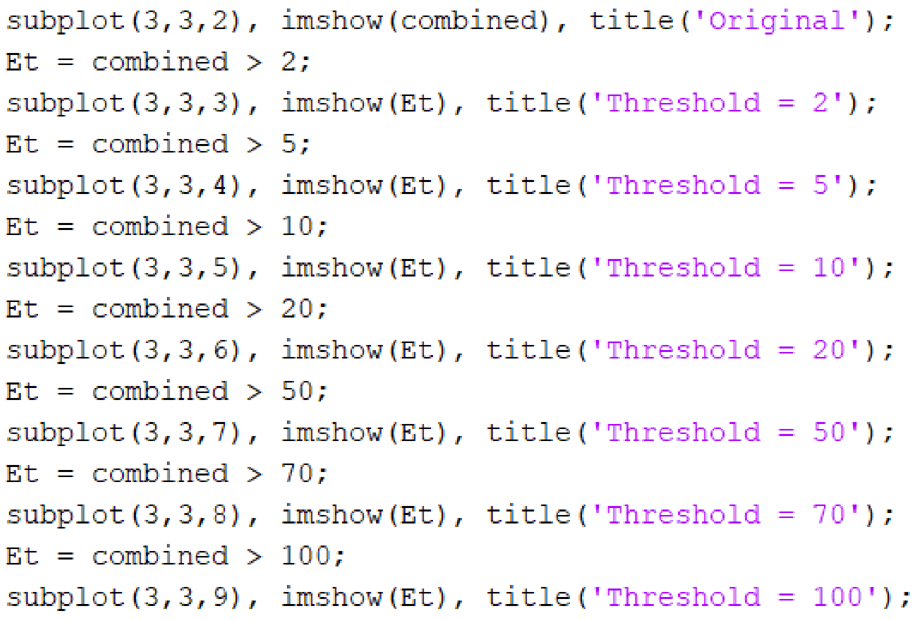
We perform contrast stretching to better facilitate our process in choosing an optimal threshold. We plot a histogram to observe the distribution of pixels, to select a suitable threshold.



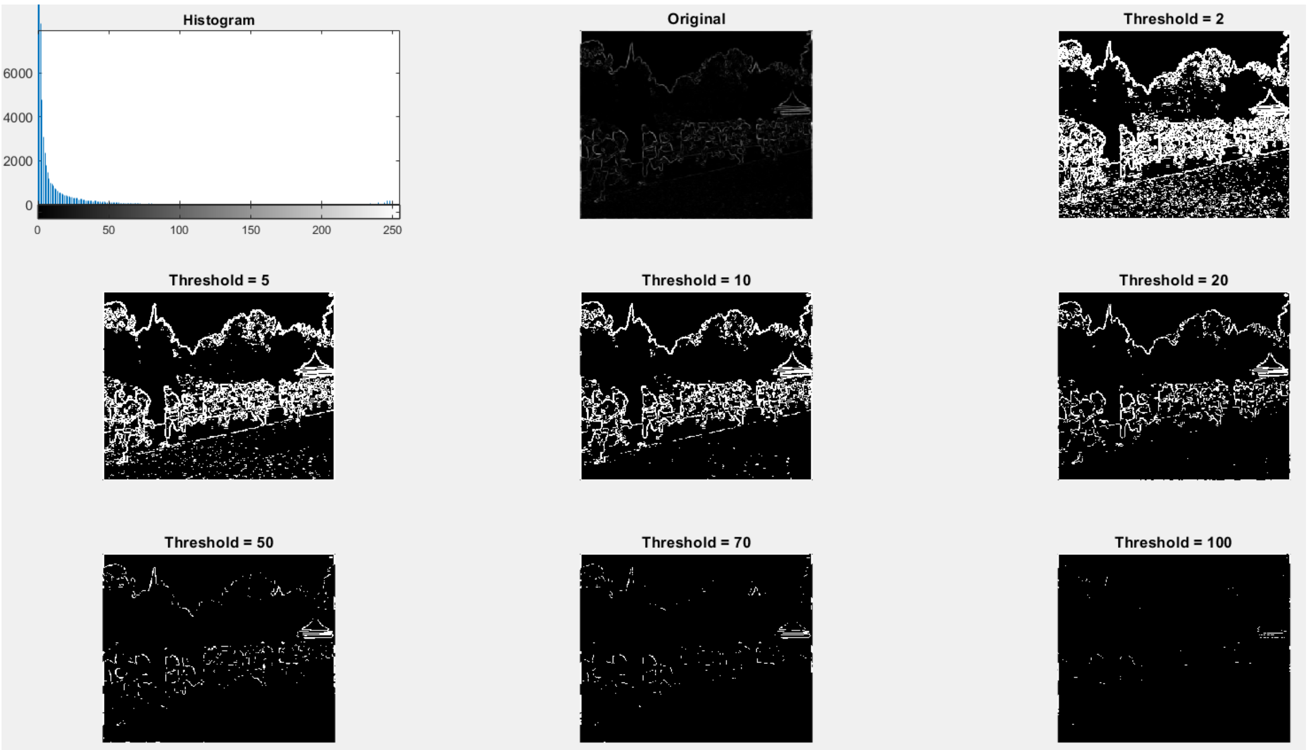
```
min_comb = min(combined(:));
max_comb = max(combined(:));
combined = 255*(combined - min_comb)/(max_comb - min_comb);
combined = uint8(combined);
subplot(3,3,1), imhist(combined, 256), title('Histogram');
```



From the histogram, we can observe that at a threshold value of 5, there is a steep decrease in pixel intensity, and that at a threshold value of around 70, there is a separation between the high pixel intensities and the low pixel intensities which could highlight the most distinct edges of the image. We plot the images with varying threshold values together to compare the effects of using different thresholds.



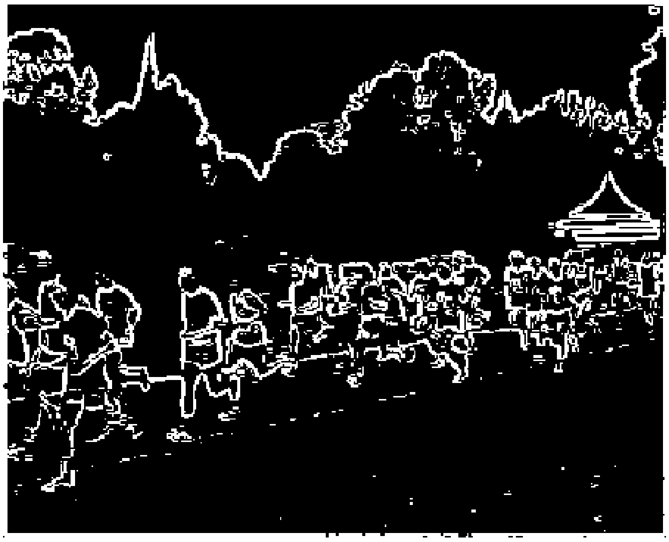
```
subplot(3,3,2), imshow(combined), title('Original');
Et = combined > 2;
subplot(3,3,3), imshow(Et), title('Threshold = 2');
Et = combined > 5;
subplot(3,3,4), imshow(Et), title('Threshold = 5');
Et = combined > 10;
subplot(3,3,5), imshow(Et), title('Threshold = 10');
Et = combined > 20;
subplot(3,3,6), imshow(Et), title('Threshold = 20');
Et = combined > 50;
subplot(3,3,7), imshow(Et), title('Threshold = 50');
Et = combined > 70;
subplot(3,3,8), imshow(Et), title('Threshold = 70');
Et = combined > 100;
subplot(3,3,9), imshow(Et), title('Threshold = 100');
```



We observe that having a threshold value of 5 retains majority of the important edges with slight noise whereas having a threshold value of 20 has little or no noise while also retaining some important edges.

As the threshold value increases, noise is removed, but at the same time the detected edges become fainter or disappear. Having a low threshold value will retain detailed information about edges. However, having too low a threshold value would make it difficult for us to differentiate between edges and noise. Having a high threshold value will allow us to only detect prominent edges. However, having too high a threshold value would make it difficult for us to detect edges, as seen when the threshold value was set to 100.

It is necessary for us to decide on an optimal threshold that will minimise noise while retaining important edges. After experimenting, we decided on a threshold value of 15.



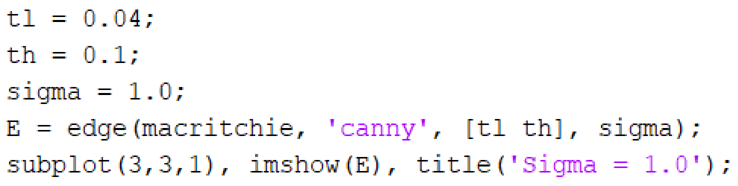
e) Recompute the edge image using the more advanced Canny edge detection algorithm with tl=0.04, th=0.1, sigma=1.0

>> E = edge(I,’canny’,[tl th],sigma);

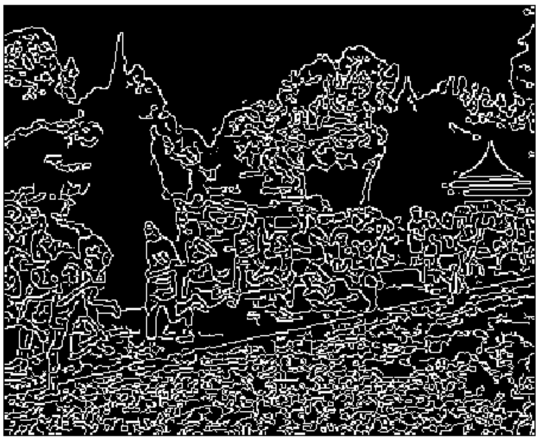
This generates a binary image without the need for thresholding.

(i) Try different values of 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.

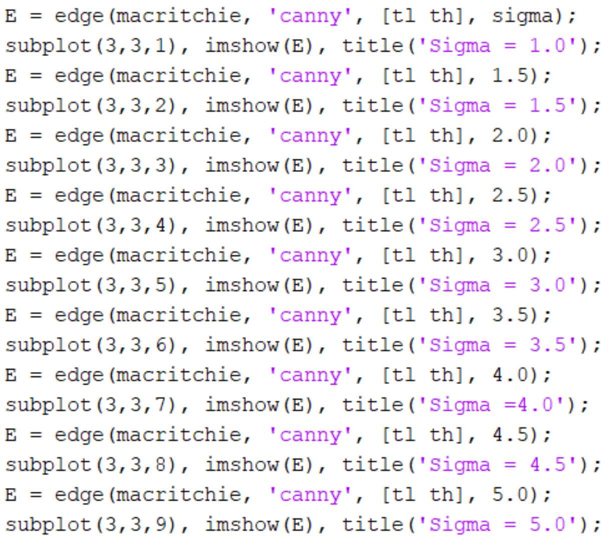
We first recompute the edge image with the original parameters (sigma = 1.0):



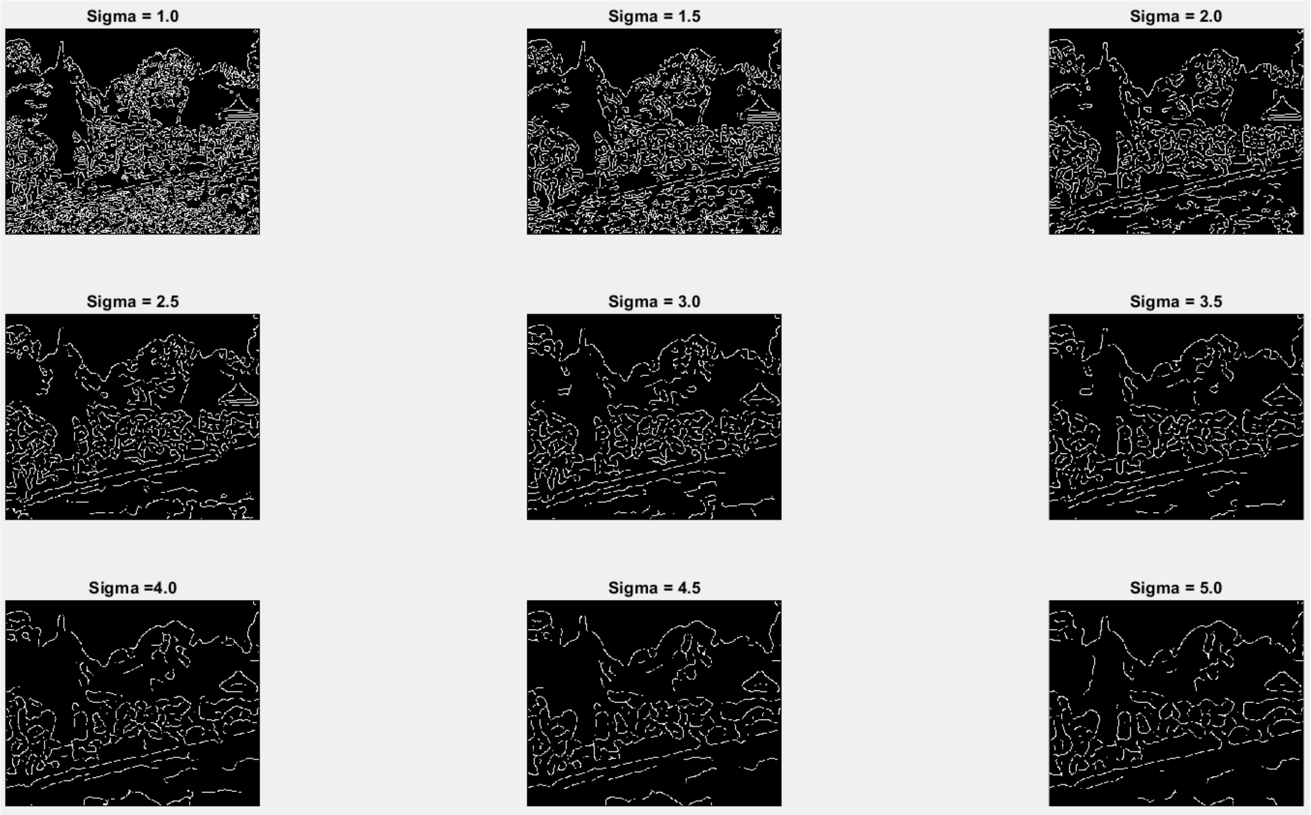
```
tl = 0.04;
th = 0.1;
sigma = 1.0;
E = edge(macritchie, 'canny', [tl th], sigma);
subplot(3,3,1), imshow(E), title('Sigma = 1.0');
```



We then recompute the edge image with varying values of sigma ranging from 1.0 to 5.0 in intervals of 0.5. We plot the images together to facilitate our comparison.



```
E = edge(macritchie, 'canny', [tl th], sigma);
subplot(3,3,1), imshow(E), title('Sigma = 1.0');
E = edge(macritchie, 'canny', [tl th], 1.5);
subplot(3,3,2), imshow(E), title('Sigma = 1.5');
E = edge(macritchie, 'canny', [tl th], 2.0);
subplot(3,3,3), imshow(E), title('Sigma = 2.0');
E = edge(macritchie, 'canny', [tl th], 2.5);
subplot(3,3,4), imshow(E), title('Sigma = 2.5');
E = edge(macritchie, 'canny', [tl th], 3.0);
subplot(3,3,5), imshow(E), title('Sigma = 3.0');
E = edge(macritchie, 'canny', [tl th], 3.5);
subplot(3,3,6), imshow(E), title('Sigma = 3.5');
E = edge(macritchie, 'canny', [tl th], 4.0);
subplot(3,3,7), imshow(E), title('Sigma =4.0');
E = edge(macritchie, 'canny', [tl th], 4.5);
subplot(3,3,8), imshow(E), title('Sigma = 4.5');
E = edge(macritchie, 'canny', [tl th], 5.0);
subplot(3,3,9), imshow(E), title('Sigma = 5.0');
```

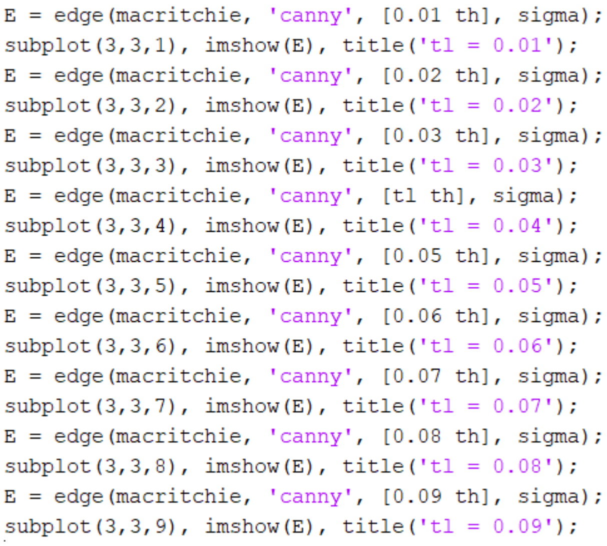


We observed that as sigma increases, the noise is removed. However, some important edges disappear, and the detected edges are less defined and loses its details.

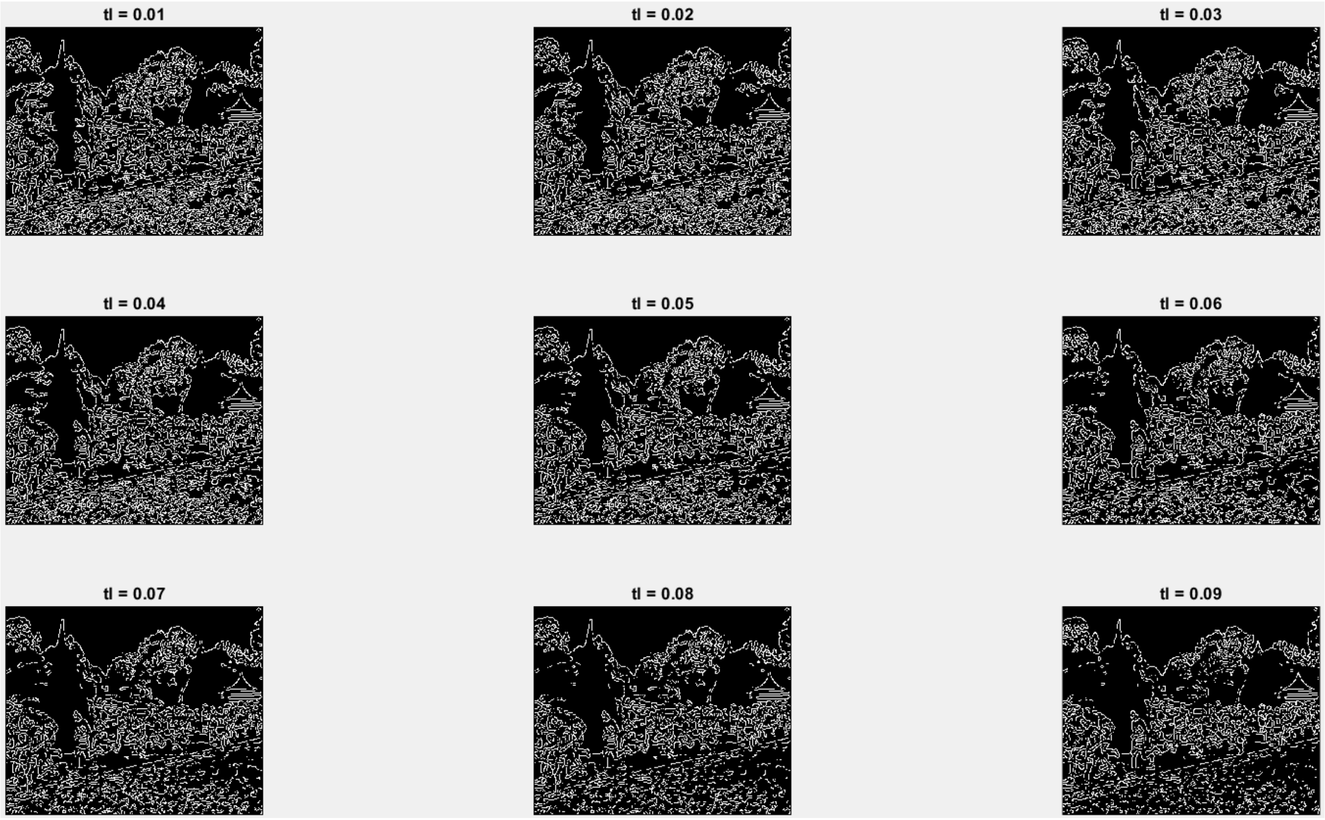
High sigma values are more suitable for noisy edgel removal whereas low sigma values are more suitable for retaining the location accuracy of edgels. This is because for Canny edge detection algorithm, the sigma relates to the strength of the gaussian filter applied and a larger sigma value would signify a flatter gaussian curve, which causes more smoothing.

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

Like previously, we plot the images together for better comparison, but this time with varying values of tl ranging from 0.01 to 0.09 in intervals of 0.01.



```
E = edge(macritchie, 'canny', [0.01 th], sigma);
subplot(3,3,1), imshow(E), title('tl = 0.01');
E = edge(macritchie, 'canny', [0.02 th], sigma);
subplot(3,3,2), imshow(E), title('tl = 0.02');
E = edge(macritchie, 'canny', [0.03 th], sigma);
subplot(3,3,3), imshow(E), title('tl = 0.03');
E = edge(macritchie, 'canny', [tl th], sigma);
subplot(3,3,4), imshow(E), title('tl = 0.04');
E = edge(macritchie, 'canny', [0.05 th], sigma);
subplot(3,3,5), imshow(E), title('tl = 0.05');
E = edge(macritchie, 'canny', [0.06 th], sigma);
subplot(3,3,6), imshow(E), title('tl = 0.06');
E = edge(macritchie, 'canny', [0.07 th], sigma);
subplot(3,3,7), imshow(E), title('tl = 0.07');
E = edge(macritchie, 'canny', [0.08 th], sigma);
subplot(3,3,8), imshow(E), title('tl = 0.08');
E = edge(macritchie, 'canny', [0.09 th], sigma);
subplot(3,3,9), imshow(E), title('tl = 0.09');
```

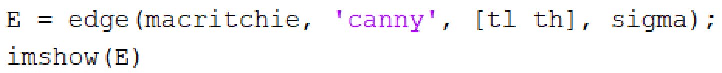


Canny edge detection algorithm uses hysteresis thresholding, with tl signifying the lower bound threshold value. This means that any value lesser than the defined tl value will not be detected as an edge. We observed that as the tl value increases, the number of detected edges decreases. This also means that there is less noise detected as weak edges are not retained.

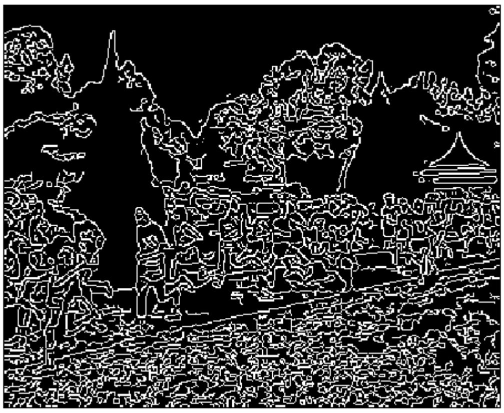
2.2 Line Finding using Hough Transform

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.

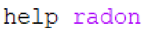


```
E = edge(macritchie, 'canny', [tl th], sigma);
imshow(E)
```

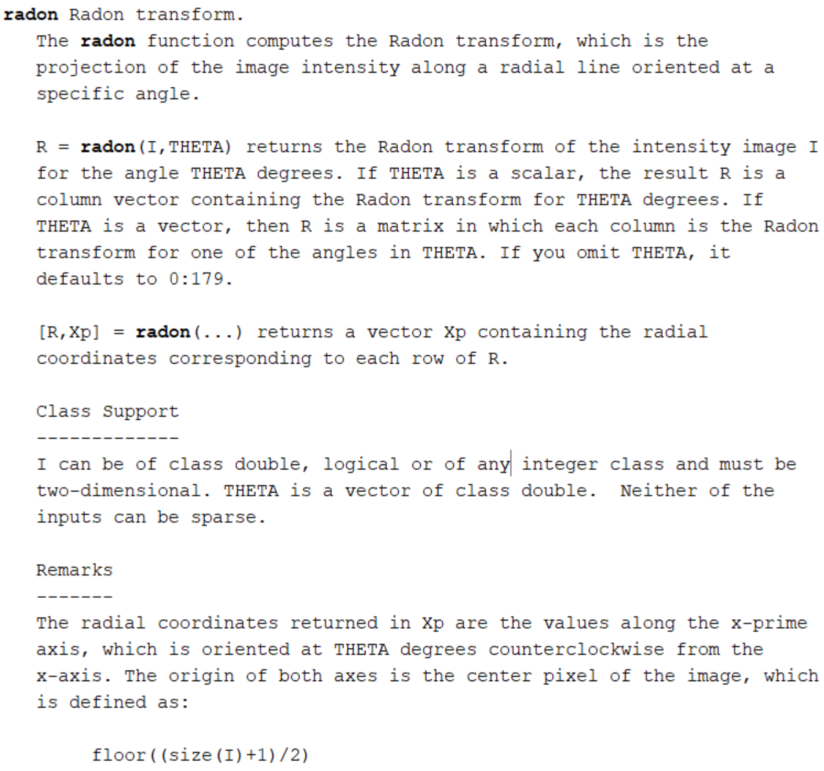


b) As there is no function available to compute the Hough transform in MATLAB, we will use the Radon transform, which for binary images is equivalent to the Hough transform. Read the help manual on Radon transform and explain why the transforms are equivalent in this case. When are they different?

First, we generate the help manual on Radon transform.



```
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.

   Remarks
   -------
   The radial coordinates returned in Xp are the values along the x-prime
   axis, which is oriented at THETA degrees counterclockwise from the
   x-axis. The origin of both axes is the center pixel of the image, which
   is defined as:

        floor((size(I)+1)/2)
```

Hough transform formula:

$$\rho = xcos\theta + ysin\theta$$

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x^*,y^*)\ \delta(x-x^*)\ \delta(y-y^*)\ dx^*dy^* \ \Rightarrow$$
$$\breve{g}(\rho,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x^*,y^*)\ \delta(\rho - x^*\cos\theta - y^*\sin\theta)\ dx^*dy^*$$

Discrete Radon transform formula:

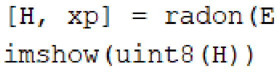
$$\begin{aligned}\breve{g}(\rho_r,\theta_t) &= \int_{-\infty}^{\infty} g(\rho_r\cos\theta_t - s\sin\theta_t, \rho_r\sin\theta_t + s\cos\theta_t)\ ds \\ &\approx \Delta s\sum_{j=0}^{J-1} g(\rho_r\cos\theta_t - s_j\sin\theta_t, \rho_r\sin\theta_t + s_j\cos\theta_t)\end{aligned}$$

Discrete Radon transform takes place when the input to the Radon transform function is discrete. By observing the equations above, we noticed that Radon transform takes the entire image, applies template matching based on θ and ρ, and sums them up. Hough transform takes each point in the image and applies template matching specific to each (θ, ρ) line. Both are equivalent in this case (when Radon transform is used for binary images).

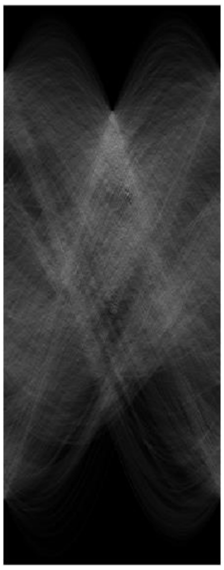
The difference between them is that the Radon transform is continuous whereas the Hough transform is discrete. As such, passing in discrete or continuous inputs into the Radon transform function will generate different results – passing in a discrete input into the function would produce discrete Radon transform.

>> [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.



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

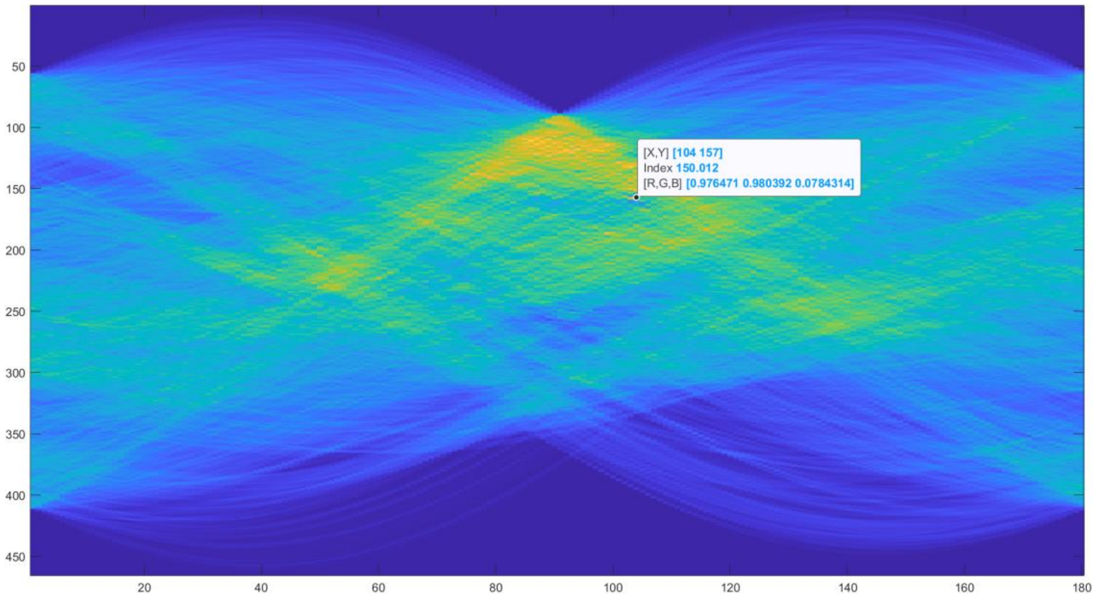


The colour intensity signifies the strength of line edges, and strong line edges are generally the intersection of sinusoidal projections.

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.

```
imagesc(H)
```

We can view the Hough image in the default colormap, where the x axis represents the theta values, and the Y axis represents the radius values. By using the “Data Tips” feature in MATLAB, we can find the peak with the highest intensity by testing the possible peaks.



We found the peak with the highest intensity to have a theta value of 104 and a radius value of 157.

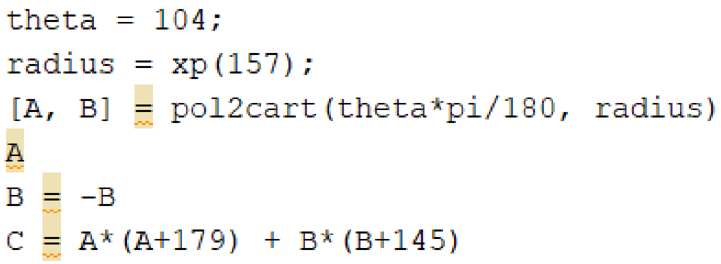
d) Derive the equations to convert the [theta, radius] line representation to the normal line equation form Ax + By = C in image coordinates. Show that A and B can be obtained via

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

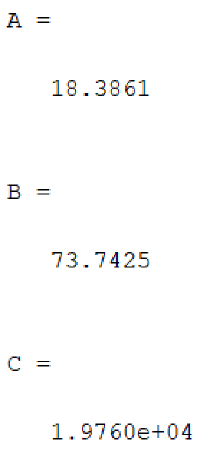
>> B = -B;

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.



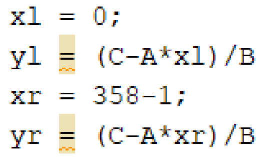
```
theta = 104;
radius = xp(157);
[A, B] = pol2cart(theta*pi/180, radius)
A
B = -B
C = A*(A+179) + B*(B+145)
```



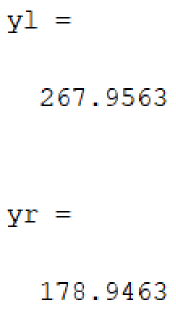
```
A =

   18.3861


B =

   73.7425


C =

   1.9760e+04
```

The image dimensions are (358, 290). Hence, the centre of the image would be (179, 145), which is obtained by dividing the dimensions by 2.

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.



```
xl = 0;
yl = (C-A*xl)/B
xr = 358-1;
yr = (C-A*xr)/B
```

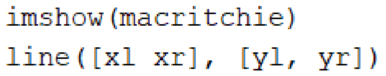


```
yl =

  267.9563


yr =

  178.9463
```

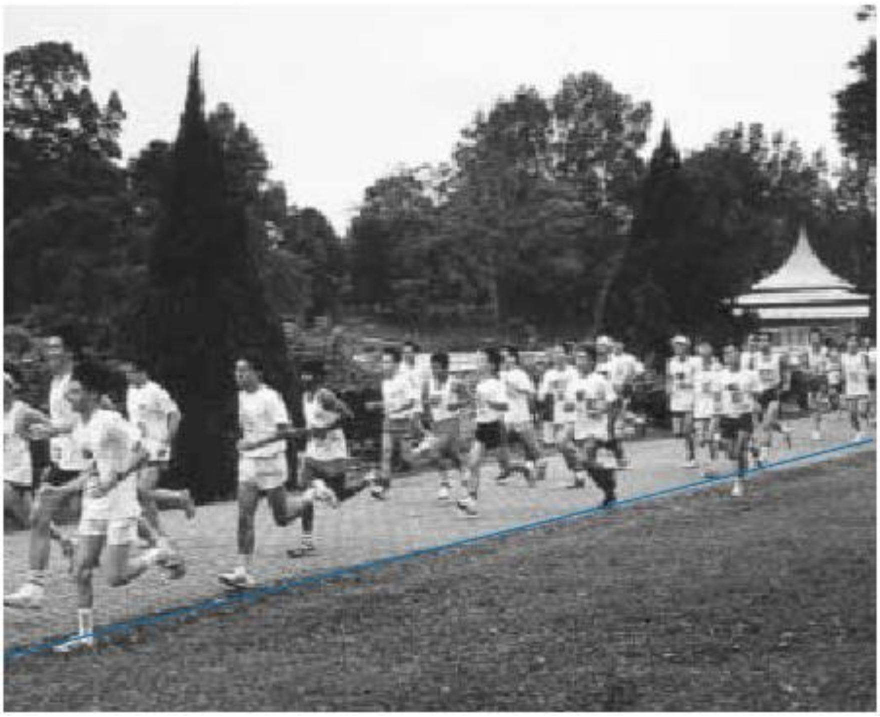
f) Display the original ‘macritchie.jpg’ image. Superimpose your estimated line by

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

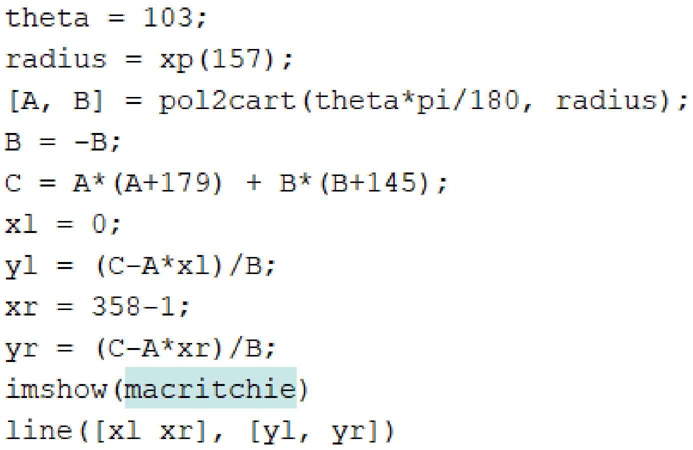
Does the line match up with the edge of the running path? What are, if any, sources of errors? Can you suggest ways of improving the estimation?



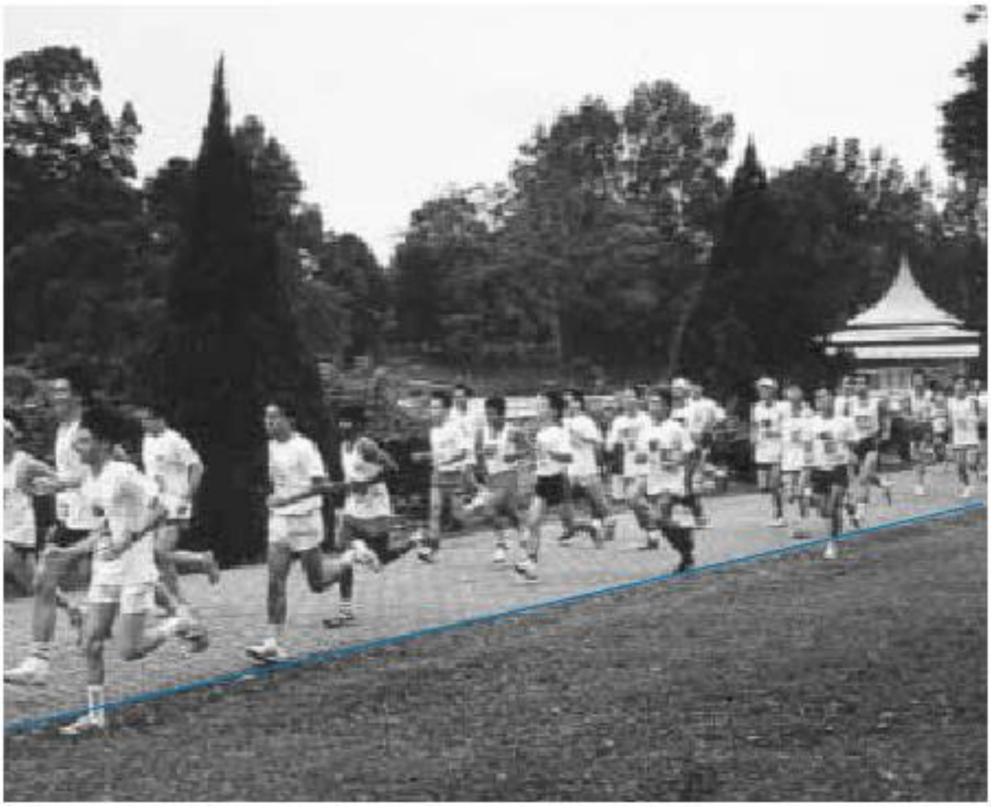
```
imshow(macritchie)
line([xl xr], [yl, yr])
```



The line did not match up with the edge perfectly. This might be due to human error from manually selecting the peak with the highest intensity earlier on and missing out on the actual peak with the highest intensity. After checking once again, the theta value should be 103 instead. After calculating everything once more with 103 as the theta value, the line fit the edge much better.

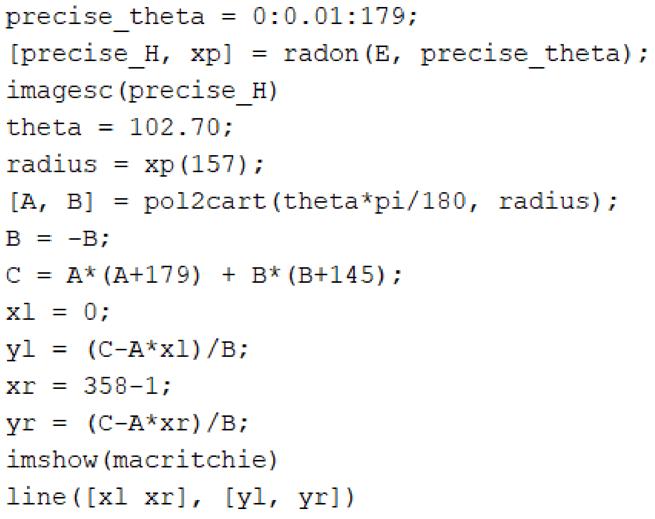


```
theta = 103;
radius = xp(157);
[A, B] = pol2cart(theta*pi/180, radius);
B = -B;
C = A*(A+179) + B*(B+145);
xl = 0;
yl = (C-A*xl)/B;
xr = 358-1;
yr = (C-A*xr)/B;
imshow(macritchie)
line([xl xr], [yl, yr])
```



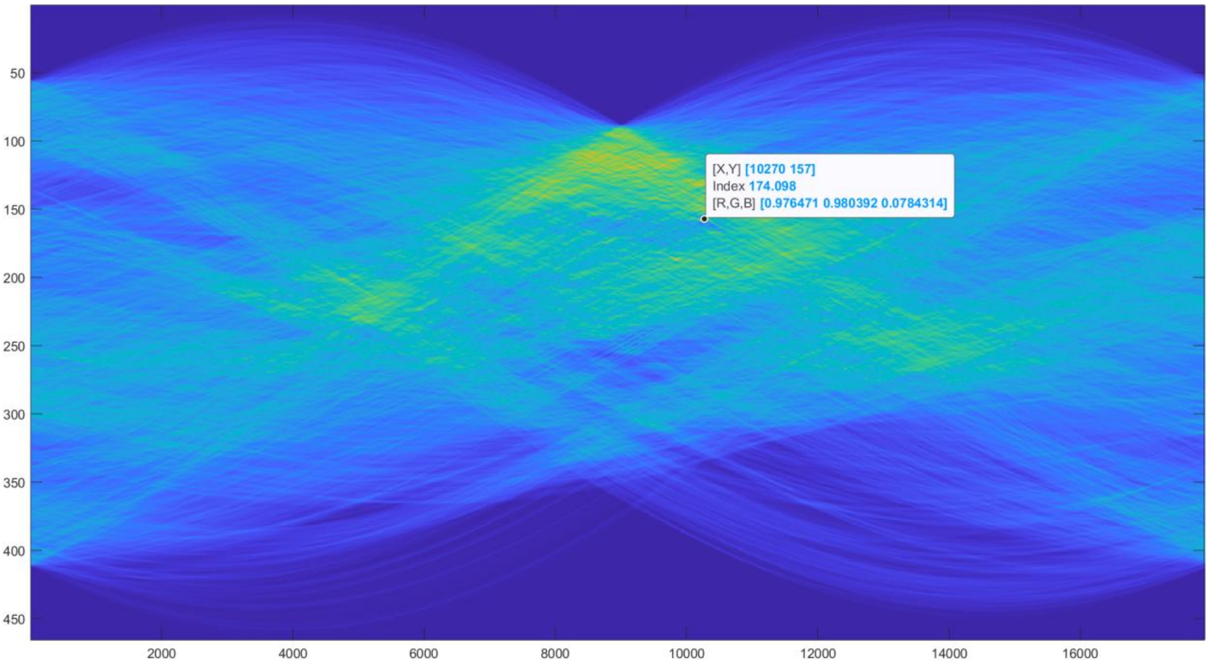
However, when a closer look is taken, the line does not perfectly match the edge towards the end. This might be due to the edge in the image not being a straight line (the track might be slightly curved). Hence, a linear function would not be able to cater to the edge and we would need a non-linear function. It might also be because our theta value is not precise enough. Our theta values are in the range of 0 to 179 in intervals of 1, and such intervals might be too large and not precise enough to capture the exact angle of the edge in the image.

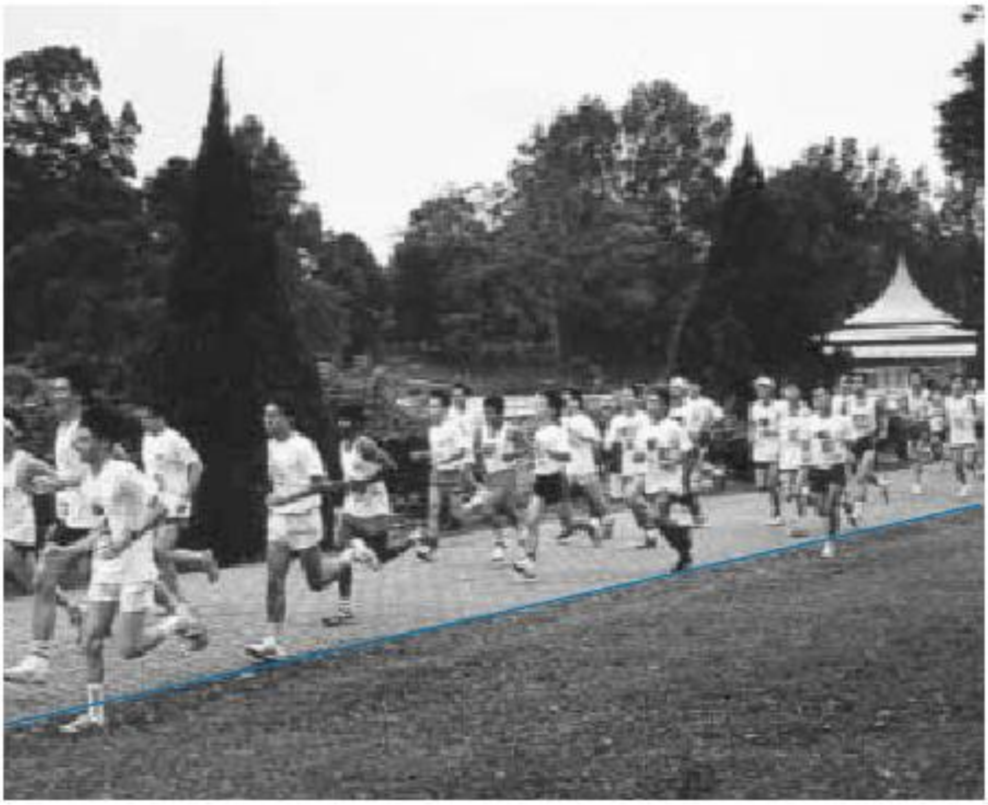
Hence, we regenerate our Radon transform with more precise theta values ranging from 0 to 179 but in intervals of 0.01. The same steps are repeated with these theta values. We now obtain a theta value of 102.70 for the peak with the highest intensity.



```
precise_theta = 0:0.01:179;
[precise_H, xp] = radon(E, precise_theta);
imagesc(precise_H)
theta = 102.70;
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 = 358-1;
yr = (C-A*xr)/B;
imshow(macritchie)
line([xl xr], [yl, yr])
```

The new theta values with more intervals also took us a longer time to compute, but a more accurate result was generated.





It is evident that the line now matches the edge much better.

2.3 3D Stereo

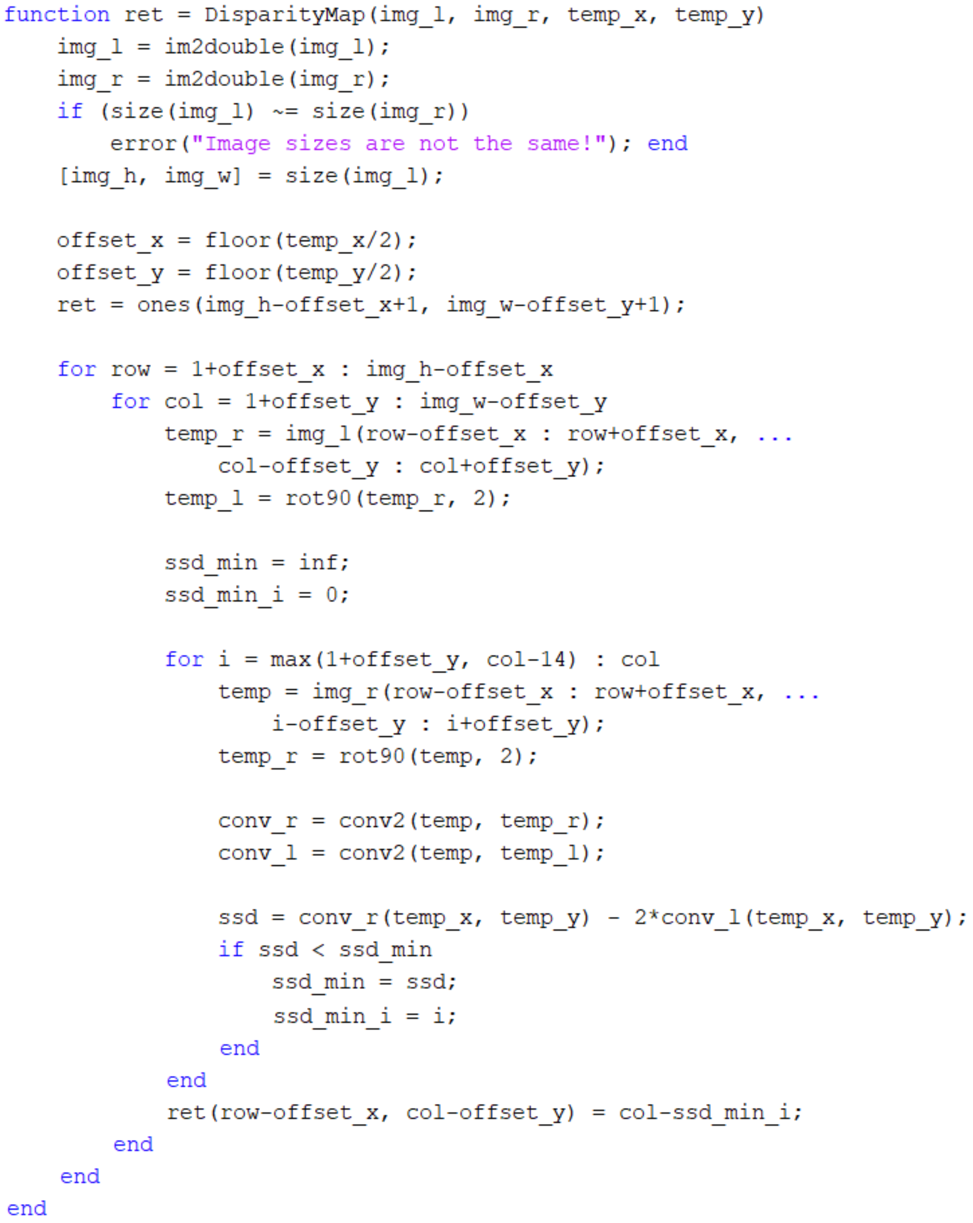
This is a fairly substantial section as you will need to write a MATLAB function script to compute disparity images (or maps) for pairs of rectified stereo images Pl and Pr. The disparity map is inversely proportional to the depth map which gives the distance of various points in the scene from the camera.

Estimating Disparity Maps

The overview of the algorithm is: for each pixel in Pl,

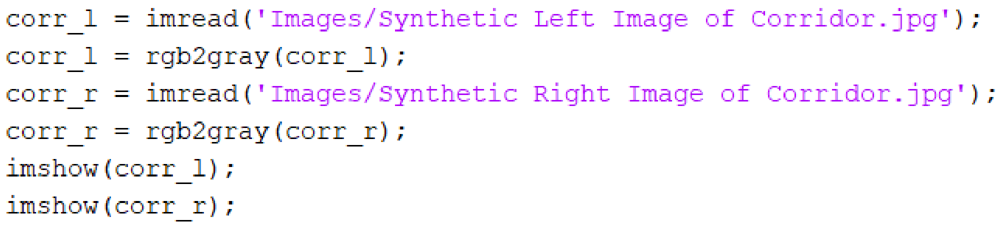
i. Extract a template comprising the 11x11 neighbourhood region around that pixel.
ii. Using the template, carry out SSD matching in Pr, but only along the same scanline.
iii. Input the disparity into the disparity map with the same Pl pixel coordinates.

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.

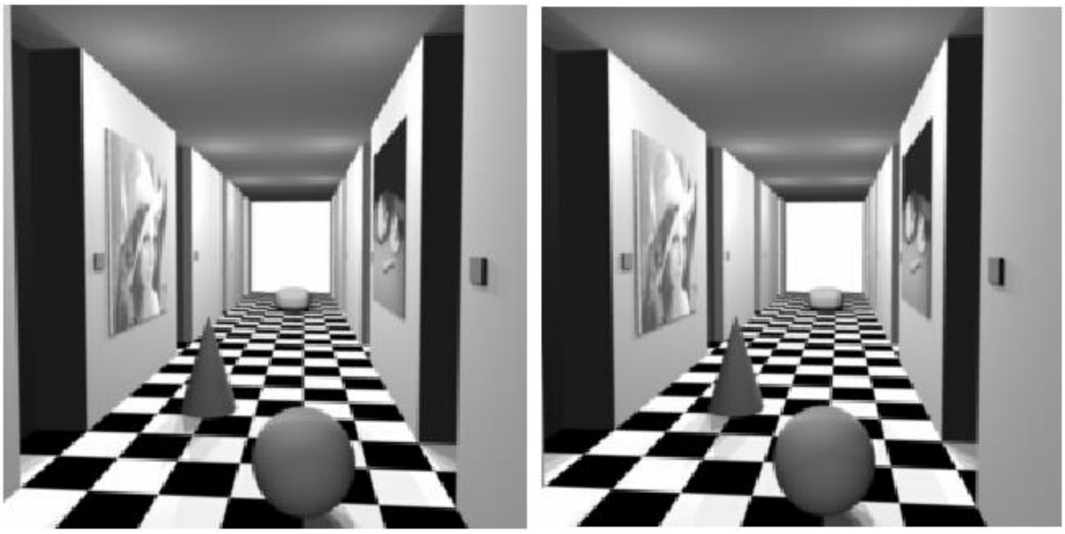


```
function ret = DisparityMap(img_l, img_r, temp_x, temp_y)
    img_l = im2double(img_l);
    img_r = im2double(img_r);
    if (size(img_l) ~= size(img_r))
        error("Image sizes are not the same!"); end
    [img_h, img_w] = size(img_l);

    offset_x = floor(temp_x/2);
    offset_y = floor(temp_y/2);
    ret = ones(img_h-offset_x+1, img_w-offset_y+1);

    for row = 1+offset_x : img_h-offset_x
        for col = 1+offset_y : img_w-offset_y
            temp_r = img_l(row-offset_x : row+offset_x, ...
                col-offset_y : col+offset_y);
            temp_l = rot90(temp_r, 2);

            ssd_min = inf;
            ssd_min_i = 0;

            for i = max(1+offset_y, col-14) : col
                temp = img_r(row-offset_x : row+offset_x, ...
                    i-offset_y : i+offset_y);
                temp_r = rot90(temp, 2);

                conv_r = conv2(temp, temp_r);
                conv_l = conv2(temp, temp_l);

                ssd = conv_r(temp_x, temp_y) - 2*conv_l(temp_x, temp_y);
                if ssd < ssd_min
                    ssd_min = ssd;
                    ssd_min_i = i;
                end
            end
            ret(row-offset_x, col-offset_y) = col-ssd_min_i;
        end
    end
end
```

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



```
corr_l = imread('Images/Synthetic Left Image of Corridor.jpg');
corr_l = rgb2gray(corr_l);
corr_r = imread('Images/Synthetic Right Image of Corridor.jpg');
corr_r = rgb2gray(corr_r);
imshow(corr_l);
imshow(corr_r);
```

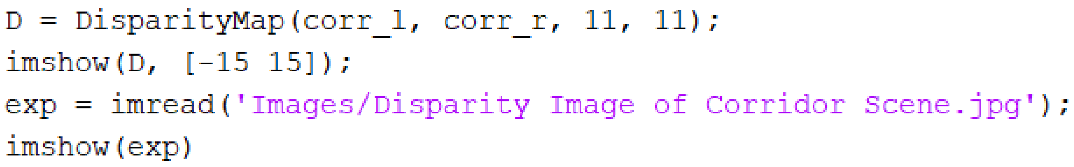


(Left: corr_l, Right: corr_r)

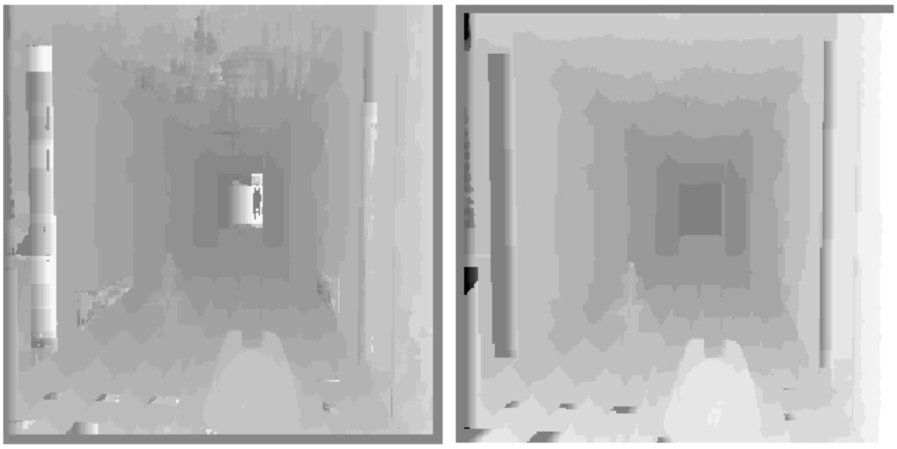
c) Run your algorithm on the two images to obtain a disparity map D, and see the results via

>> 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.



```
D = DisparityMap(corr_l, corr_r, 11, 11);
imshow(D, [-15 15]);
exp = imread('Images/Disparity Image of Corridor Scene.jpg');
imshow(exp)
```



(Left: D, results from DisparityMap, Right: expected image)

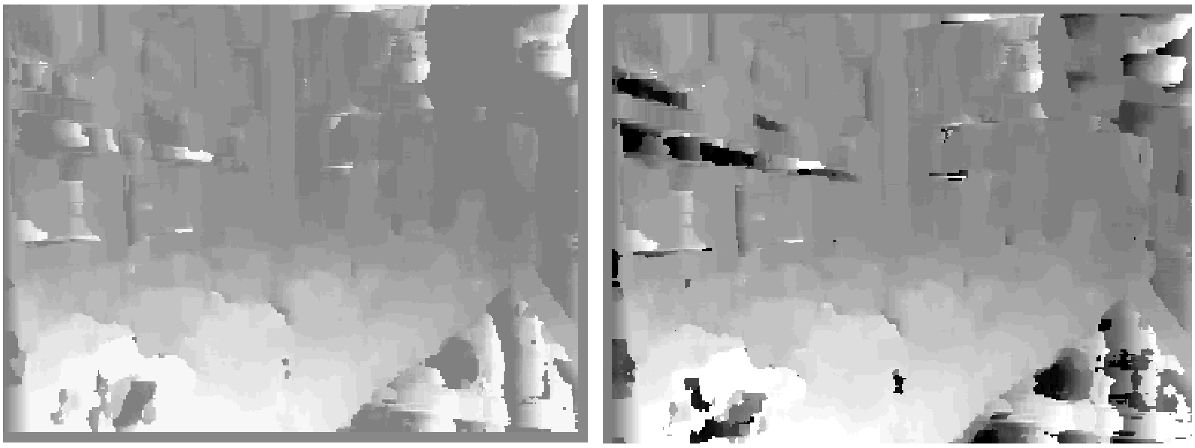
The result from DisparityMap is similar to the expected image. The corridor and the objects (cone, sphere) are generally mapped correctly across both images. The pixel intensity decreases towards the centre of the image as disparity decreases for objects further away. However, one thing to note is that the results at the centre of the image are not as desired since they are not uniform. This might be due to the homogenous colour for that portion of the image. The result might be slightly different due to us limiting the search to small values of disparity.

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?

We repeat the same steps for different images.

```
corr_l = imread('Images/Left Image of Triclops Stereo Pair.jpg');
corr_l = rgb2gray(corr_l);
corr_r = imread('Images/Right Image of Triclops Stereo Pair.jpg');
corr_r = rgb2gray(corr_r);
imshow(corr_l);
imshow(corr_r);
D = DisparityMap(corr_l, corr_r, 11, 11);
imshow(D, [-15 15]);
exp = imread('Images/Disparity of Triclops Stereo Pair.jpg');
imshow(exp)
```

(Left: corr_l, Right: corr_r)



(Left: D, results from DisparityMap, Right: expected image)

The results generated by the DisparityMap for this set of images performed worse. The image is not correctly mapped at certain regions. This might be because this set of images have lesser contrast compared to the first set of images, and this lack of contrast might have led to the decrease in accuracy of the disparity map. It might also be due to this set of images having a different scan line, hence resulting in certain portions not being mapped correctly. This might also be due to the difference in quality of the images – the previous set of images had higher quality whereas this set of images are more blurred. We observed that disparity maps are not as accurate when there is a homogenous portion with the same intensity.

2.4 Spatial Pyramid Matching and Bag-of-Words

You will need to implement the algorithm in the CVPR 2006 paper entiled “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”. You will need to use the benchmark Caltech-101 dataset and compare the classification results of Spatial Pyramid Matching (SPM) and the bag-of-words (BoW) method as in Table 2 of the paper by following the experimental setting in Section 5.2 of the paper.

Refer to spatial_pyramid_matching.ipynb for implementation and analysis