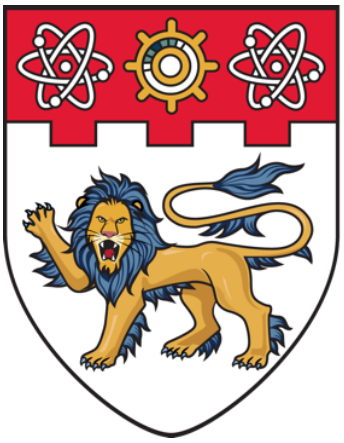


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4003 Lab 2

Report

Hans Tananda U1720251K

Experiments

3.1 Edge Detection

a. Load Image and change image to grayscale

```
%% a. Load Image
Pc = imread('macritchie.jpg');
% Change Image to grayscale
P = rgb2gray(Pc);
whos P
```

Output:

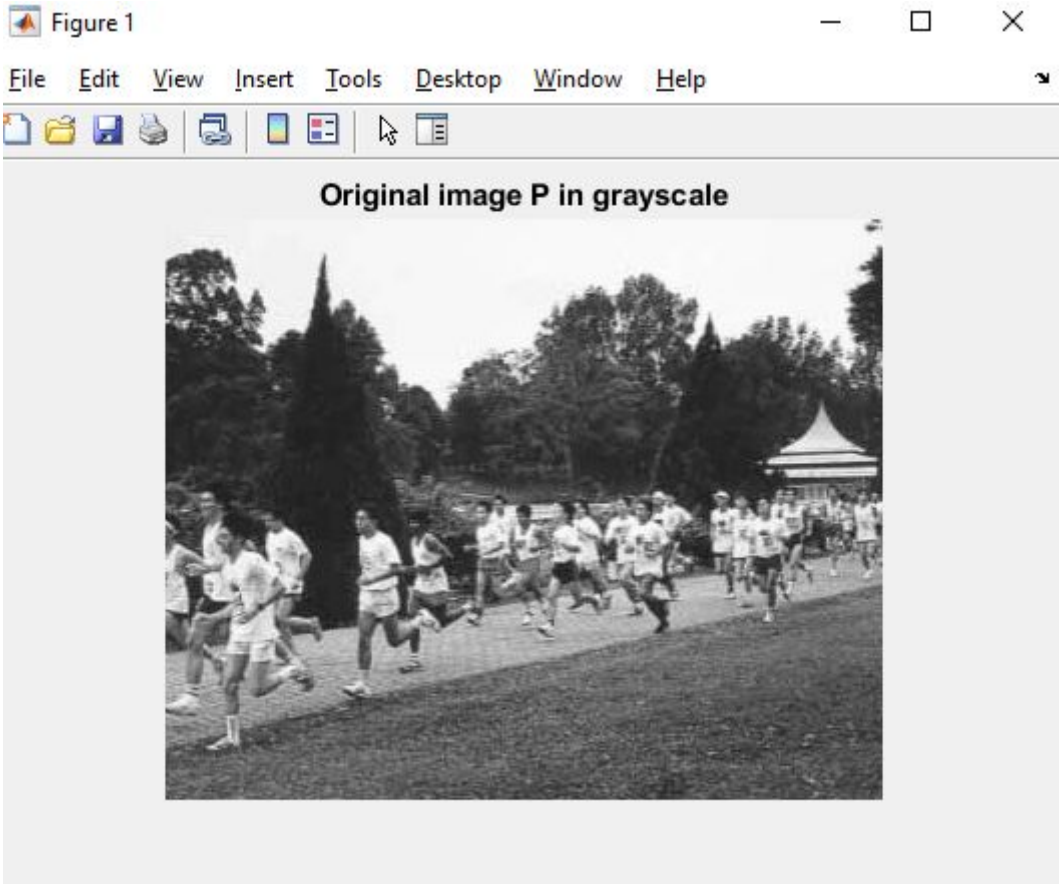
```
Name        Size              Bytes  Class    Attributes

P         290x358            103820  uint8
```

Explanation and comment: From the output above, it is confirmed that the image P has been transformed to grayscale.

Display the image

```
%% Display the Image
figure;
imshow(P);
title('Original image P in grayscale');
```

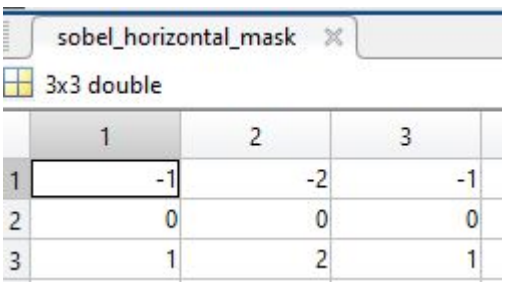
Output:



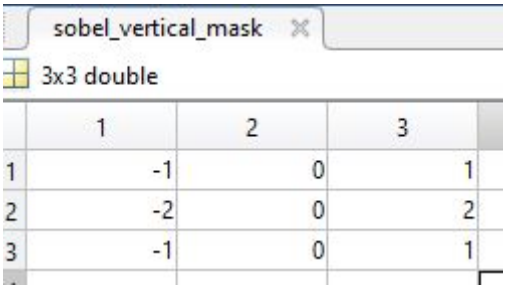
b. Create 3x3 horizontal and vertical Sobel masks and filter the image using conv2 then display the filtered images.

```
%% b. Create 3x3 horizontal and vertical Sobel masks
sobel_horizontal_mask = [-1 -2 -1; 0 0 0; 1 2 1];
sobel_vertical_mask = [-1 0 1; -2 0 2; -1 0 1];
```

Output:



	1	2	3
1	-1	-2	-1
2	0	0	0
3	1	2	1



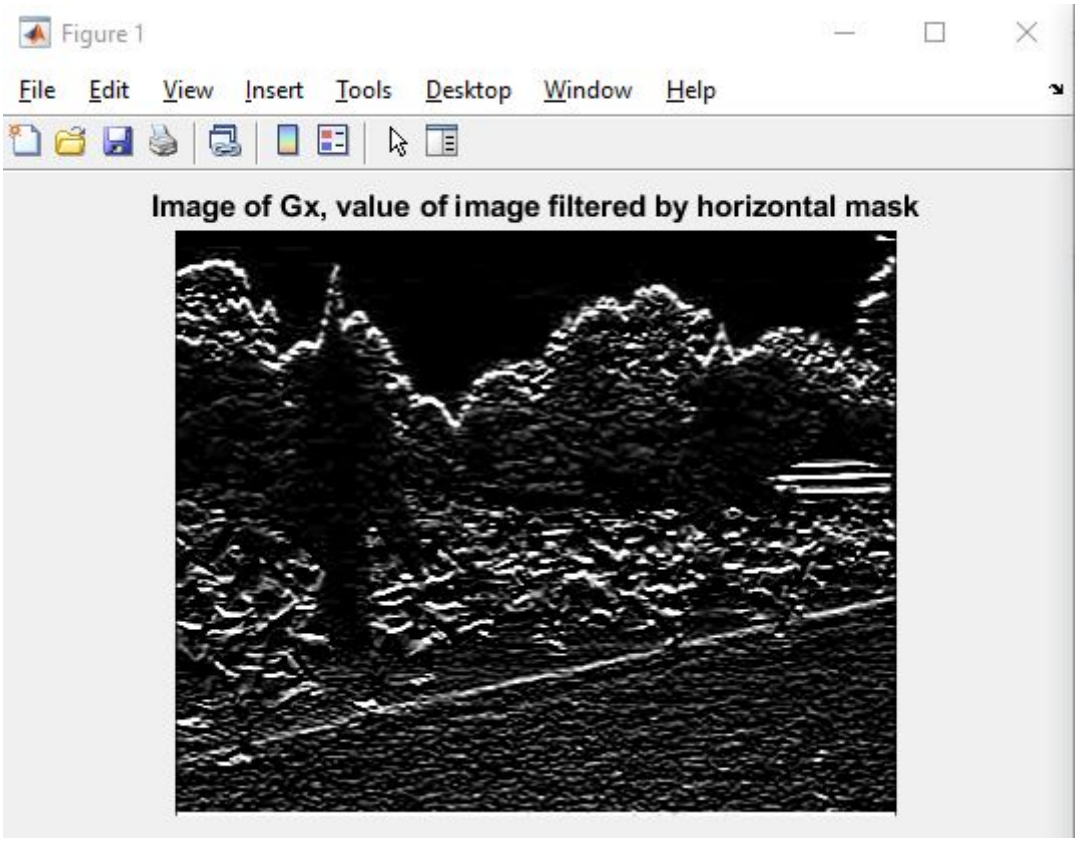
	1	2	3
1	-1	0	1
2	-2	0	2
3	-1	0	1

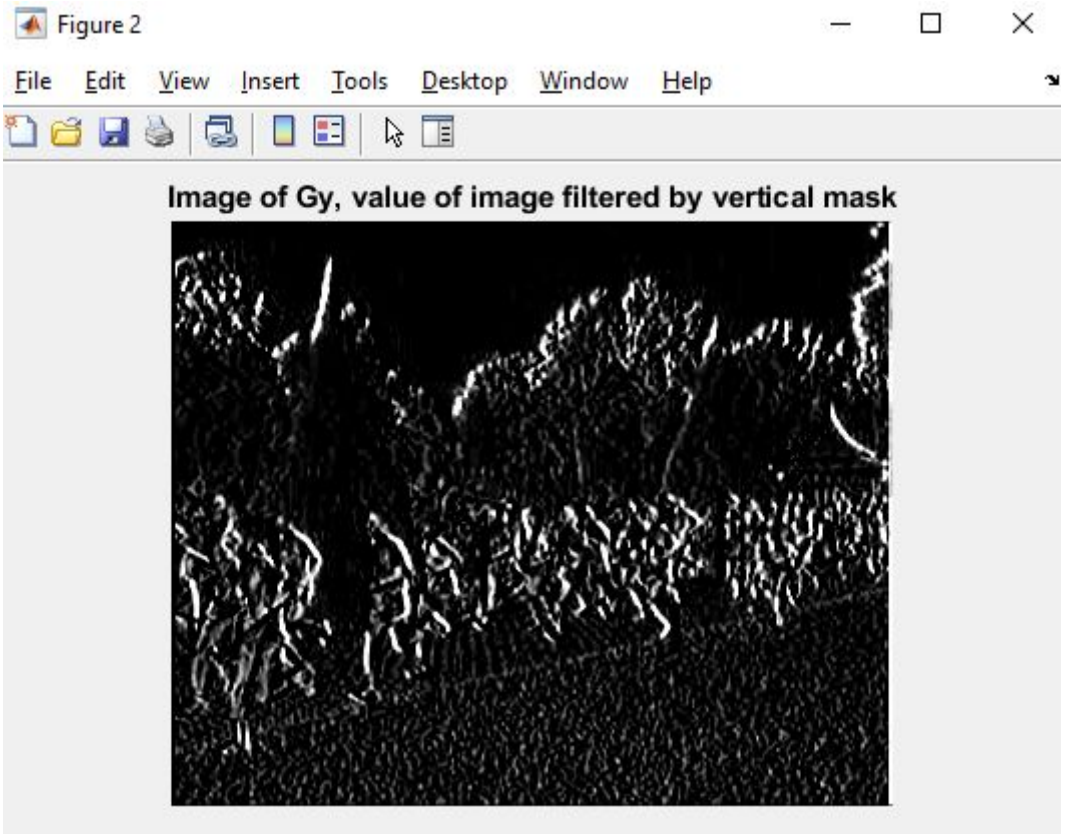
Note: It can be seen that the 3x3 Sobel masks have been created correctly.

Display the edge-filtered images

```
%% filter the image using conv2 and display the filtered images
Gx = conv2(P,sobel_horizontal_mask);

figure;
imshow(uint8(Gx));
title('Image of Gx, value of image filtered by horizontal mask');
figure;
imshow(uint8(Gy));
title('Image of Gy, value of image filtered by vertical mask');
```

Output:

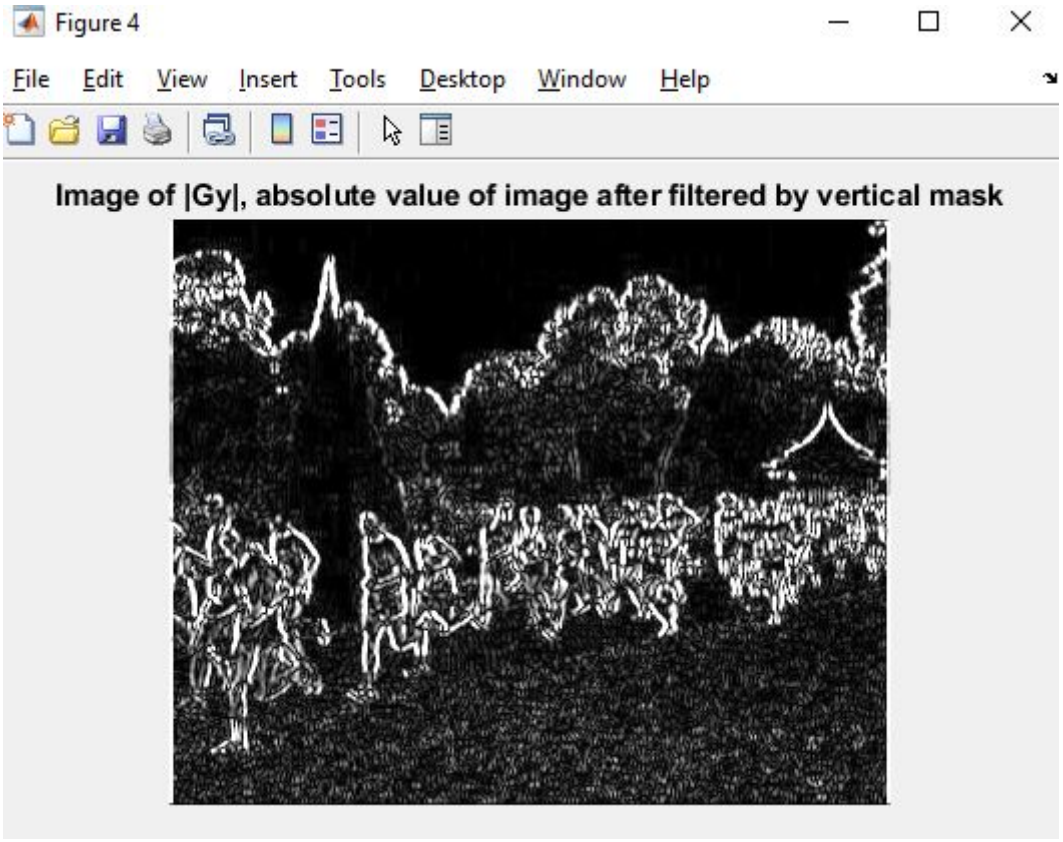




Explanation and comment: It can be seen from the images above that in the Image filtered by the horizontal mask, only pixels that resemble horizontal lines are shown, and only vertical lines are shown in the image filtered by the vertical mask. However, after further observation, it seems that around half of those horizontal/vertical lines are gone. After referring back to the lecture notes, it seems that we need to display the absolute value of the filtered images (i.e. |Gx| and |Gy| instead of Gx and Gy directly) to truly see all the edges of the image. Therefore, I tried to display the image using the absolute values of the filtered images.

```
%% Display the absolute value of the filtered images

figure;
imshow(uint8(abs(Gx)));
title('Image of |Gx|, absolute value of image after filtered by
horizontal mask');

Gy = conv2(P,sobel_vertical_mask);
figure;
imshow(uint8(abs(Gy)));
title('Image of |Gy|, absolute value of image after filtered by vertical
mask');
```

Output:



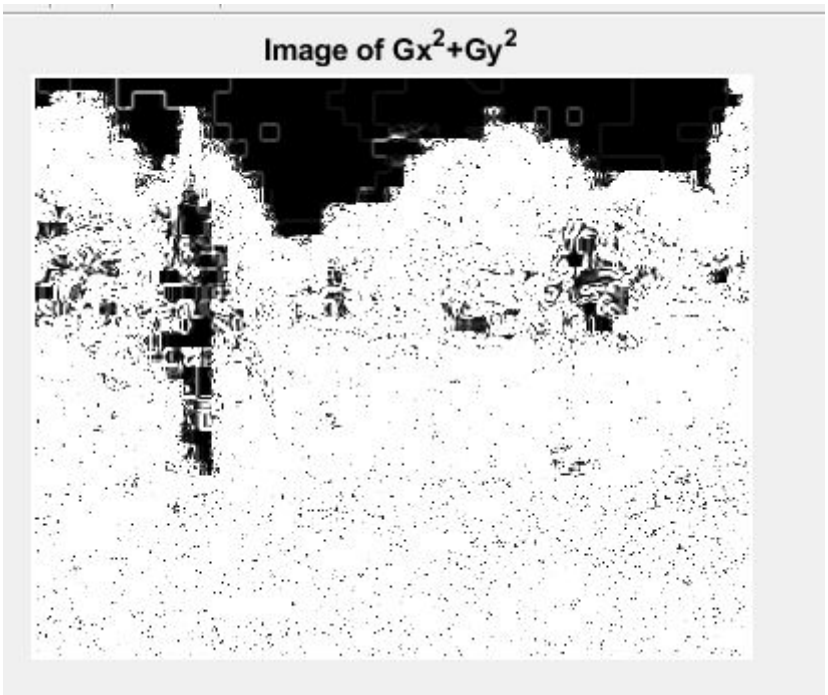
Explanation and comment: It can be seen from the images above that all pixels resembling a vertical line are displayed in the image filtered with vertical mask, and all pixels resembling a horizontal line are displayed in the image filtered with horizontal mask.

Meanwhile, for the lines which are not strictly vertical nor horizontal, it varies by the angle/ gradient of the line. If the gradient is small, then for most cases they still appear in the image filtered with the horizontal mask. For example, the line of the running path can be seen in Figure 3. Meanwhile, if the gradient is large, they appear in the image filtered with the vertical mask. This is the case for the tip of pine trees in Figure 4. However, the lines with angles around 45 degrees seem to disappear from the image, as can be seen from the bottom part of the pine trees, which appears dark in both figures.

c. Generate a combined edge image by squaring (i.e. .^2) the horizontal and vertical edge images and adding the squared images.

```
combined_Gx_Gy = Gx.^2 + Gy.^2;
figure;
imshow(uint8(combined_Gx_Gy));
title('Image of Gx^2+Gy^2');
```

Output:



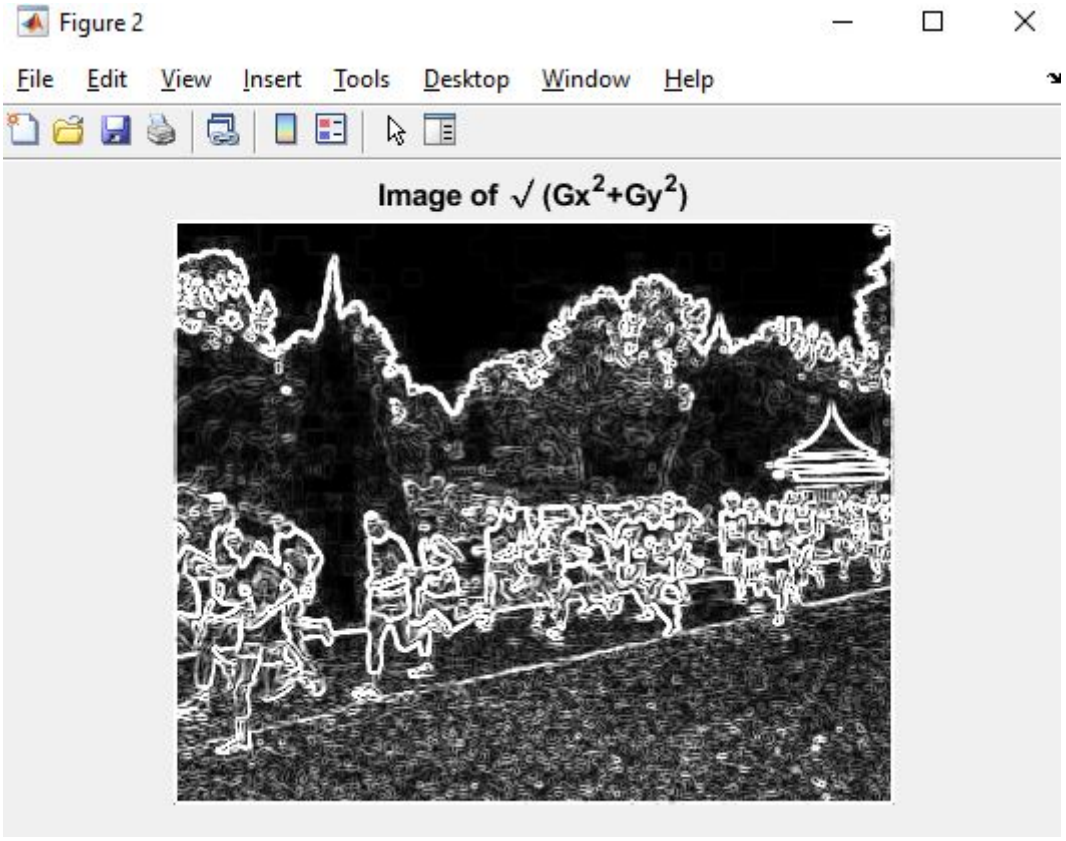
Explanation and comment: After following the exact steps instructed in the lab manual, we can see that everything appears white and there are strange square shapes in the image. This is due to the fact that the values can be really large , to be exact *1019592* after checked with command:

```
max(max(combined_Gx_Gy))
```

Therefore, I double checked with lecture notes and found that we should have taken the square root values instead to get the true absolute gradient magnitude.

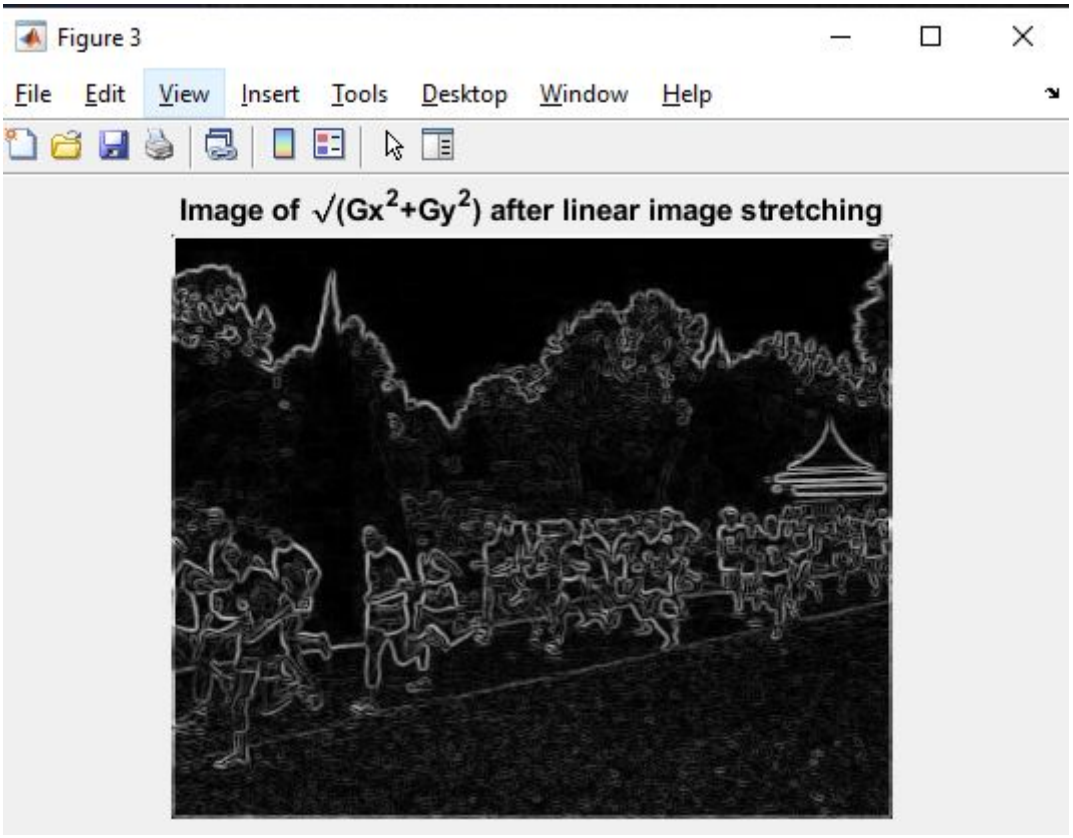
```
combined_Gx_Gy_sqrt = sqrt(combined_Gx_Gy);
figure;
imshow(uint8(combined_Gx_Gy_sqrt));
title('Image of \surd (Gx^2+Gy^2)');
```

Output:



Explanation and comment: After taking the square root values of the combined values, we can see that the pixels that represent a horizontal or vertical line are shown clearly in the image. The squaring operation is meant to get the absolute value of the magnitude, since squaring any negative number will result in a positive number. Meanwhile, the square root operation is done after addition of ‘Gx^2’ and ‘Gy^2’ so that we can get the absolute value of gradient magnitude.

Note: After further observation, it seems that the lines are overly thick. I suspected that this is due to the fact that the resulting pixel values are more than 255, and thus capped at 255. Thus, I tried to do a linear image stretching/ squeezing so that it fits between 0 and 255.

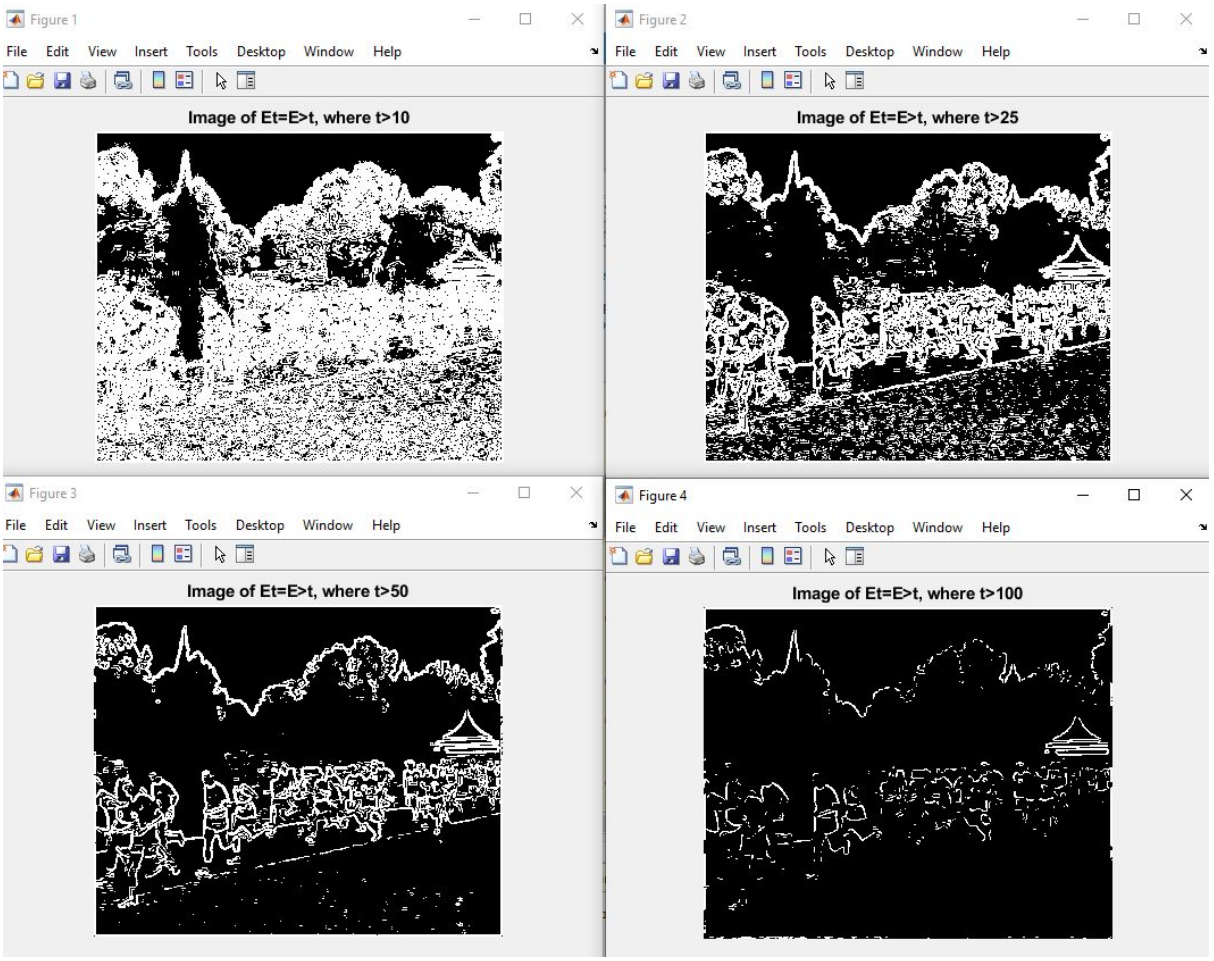


Comments: The result above looks much better than the previous one, so I decided to use this image as the base for further processing.

d. Threshold the edge image E at value t

```
Et = E > 10;
figure;
imshow(Et);
title('Image of Et=E>t, where t>10');

Et = E > 25;
figure;
imshow(Et);
title('Image of Et=E>t, where t>25');

Et = E > 50;
figure;
imshow(Et);
title('Image of Et=E>t, where t>50');

Et = E > 100;
figure;
imshow(Et);
title('Image of Et=E>t, where t>100');
```

Output:



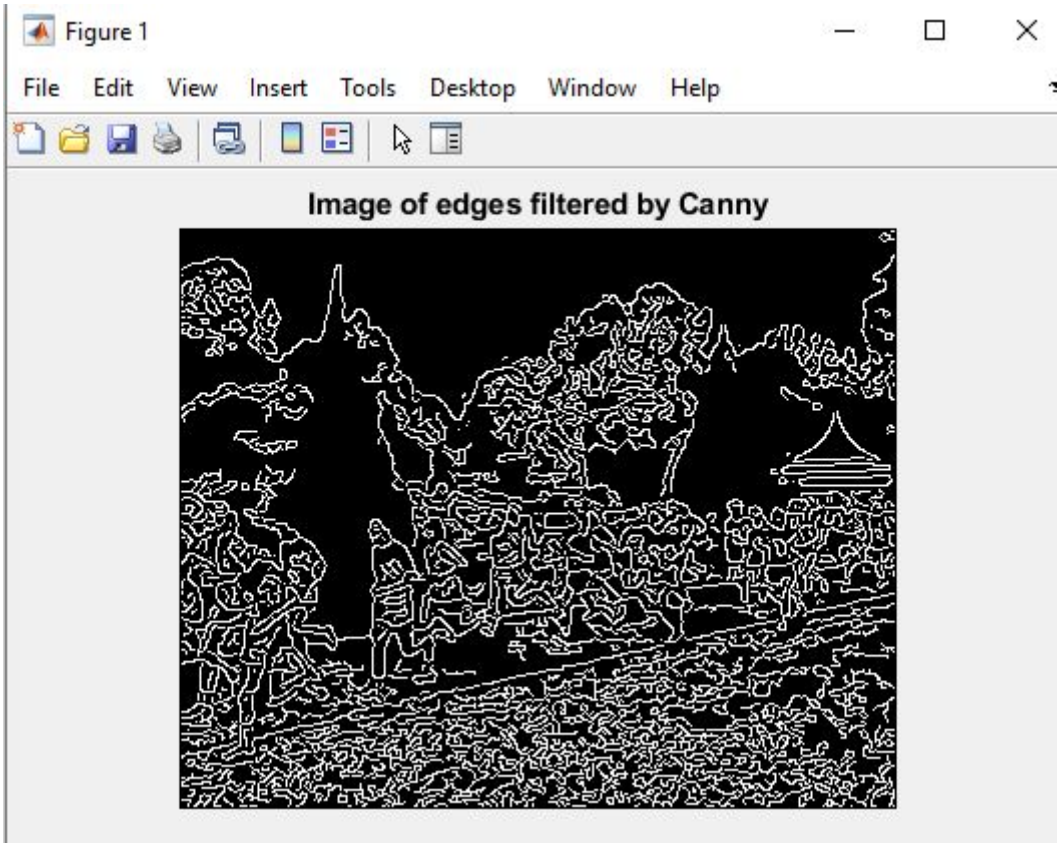
Explanation and comment:
Thresholding provides us with a binary image (pixel value only either 0 or 1). This is a simple and useful method to get an image with equal brightness.
It can be seen that when the threshold is increased, more edges disappear from the image. Therefore, this can be used as a way to remove noisy edgels. The higher the threshold is, the less noisy the image becomes. However, part of actual edgels may not be captured as well, and existing edgels may become disconnected and get broken up to smaller edgels. Thus, it can be said that one must find a good compromise between the noise and the edgels captured using this thresholding method.

e. Recompute the image using canny edge detection

```
tl = 0.04;
th = 0.1;
sigma = 1.0;

P_canny_s1 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_s1);
```

```
title('Image of edges filtered by Canny');
```

Output:



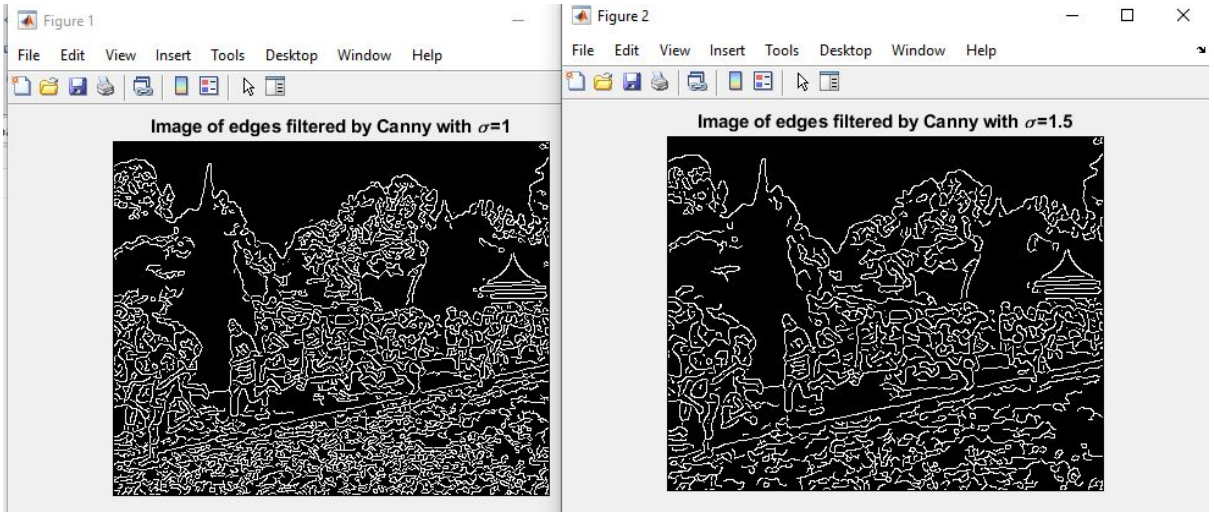
Explanation and comment: The canny edge detection resulted in a binary image with pretty much the same edge width without the need of thresholding.

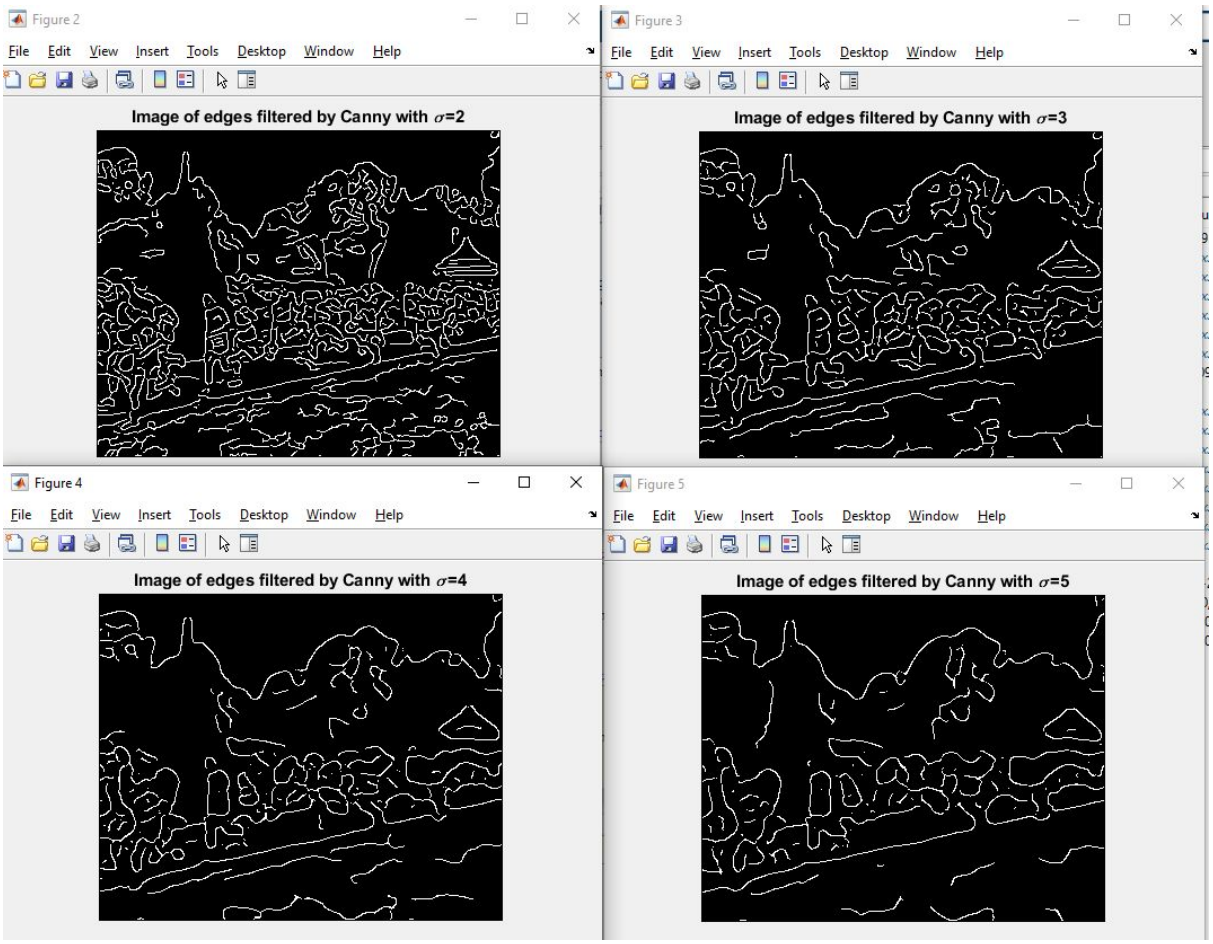
(i) Canny edge detection with different values of sigma ranging from 1.0 to 5.0

```
%% \sigma=1.5
sigma = 1.5;
P_canny_s1_5 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_s1_5);
title('Image of edges filtered by Canny with \sigma=1.5');
%% \sigma=2
sigma = 2.0;
P_canny_s2 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_s2);
title('Image of edges filtered by Canny with \sigma=2');
%% \sigma=3
sigma = 3.0;
```

```
P_canny_s3 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_s3);
title('Image of edges filtered by Canny with \sigma=3');
%% \sigma=4
sigma = 4.0;
P_canny_s4 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_s4);
title('Image of edges filtered by Canny with \sigma=4');
%% \sigma=5
sigma = 5.0;
P_canny_s5 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_s5);
title('Image of edges filtered by Canny with \sigma=5');
```

Output:





Explanation and comment:

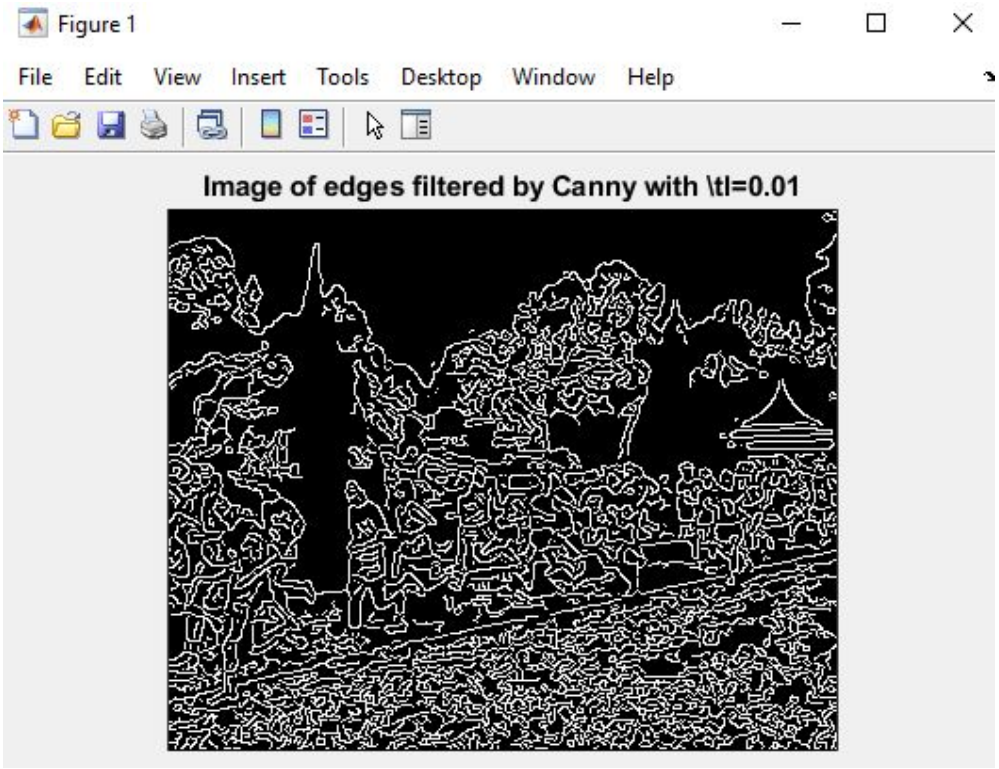
After observation, it can be seen that bigger sigma leads to less edgels, and the edgels become less and less detailed. Due to the fact that the edges' appearance become much more simplified, it can be seen that some of the edges shifted a bit from the original edge location as well when the sigma increases. Therefore, it can be concluded that a bigger sigma is more suitable to remove noisy edges, but with a tradeoff of edges being less detailed and the location accuracy of edges being less accurate as well.

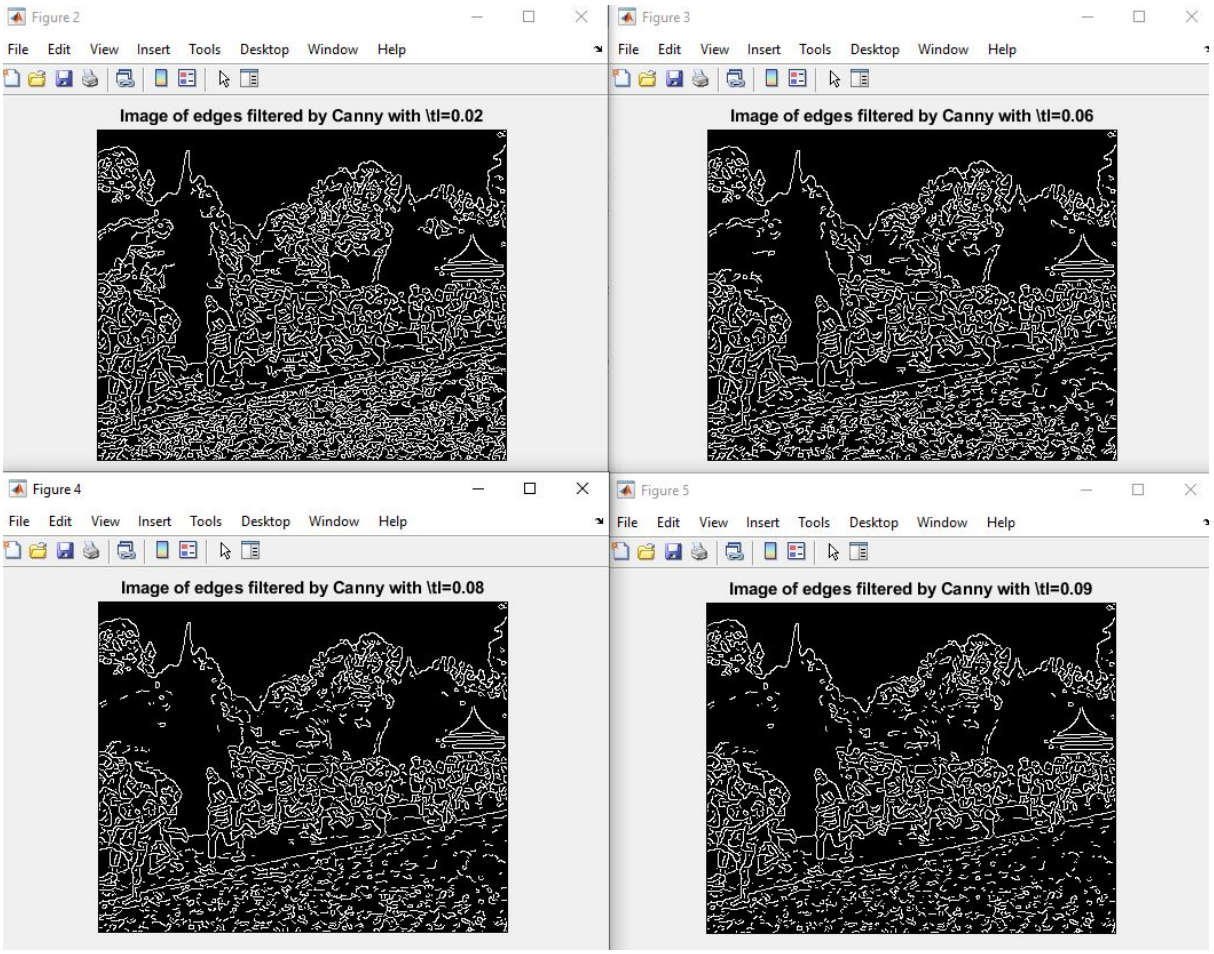
(ii) Canny edge detection with different values of tl

```
sigma = 1.0;
%% tl=0.01
tl = 0.01;
P_canny_tl_1 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_tl_1);
title('Image of edges filtered by Canny with \tl=0.01');
%% tl=0.02
tl = 0.02;
P_canny_tl_2 = edge(P, 'canny', [tl th], sigma);
```

```
figure;
imshow(P_canny_tl_2);
title('Image of edges filtered by Canny with \tl=0.02');
%% tl=0.06
tl = 0.06;
P_canny_tl_6 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_tl_6);
title('Image of edges filtered by Canny with \tl=0.06');
%% tl=0.08
tl = 0.08;
P_canny_tl_8 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_tl_8);
title('Image of edges filtered by Canny with \tl=0.08');
%% tl=0.09
tl = 0.09;
P_canny_tl_9 = edge(P, 'canny', [tl th], sigma);
figure;
imshow(P_canny_tl_9);
title('Image of edges filtered by Canny with \tl=0.09');
```

Output:





Explanation and comment:

It can be seen that when tl is lower, the edges looked more detailed and complex. As the tl gets closer to th, the edges become less detailed.

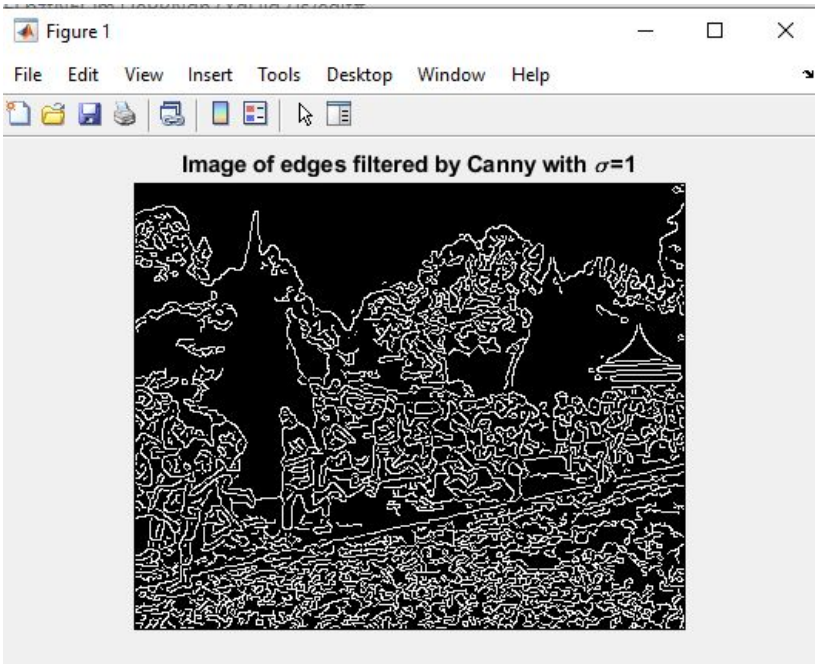
This can be explained by the fact that the Canny edge detector uses hysteresis thresholding to determine if a pixel should be considered as an edgel. If the computed value of a pixel is between tl and th, then it will be further verified with its neighboring pixels. It will be considered as an edge if the neighboring pixels are considered as an edge. Otherwise, it will not be considered as an edge. As a result, the bigger the threshold of tl and th, the more pixels will be considered as part of the edges. Conversely, the closer the threshold between tl and th, less pixels will be considered as edgels and thus the less complex the edges will be.

3.2 Line Finding using Hough Transform

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

```
Pc = imread('macritchie.jpg');
% Change Image to grayscale
P = rgb2gray(Pc);
% Image computed via the Canny algorithm with sigma=1.0
tl = 0.04;
th = 0.1;
sigma = 1.0;

E = edge(P, 'canny', [tl th], sigma);
figure;
imshow(E);
title('Image of edges filtered by Canny with \sigma=1');
```

Output:



Explanation and comment:
Since I decided to create separate files for each experiment, I decided to copy paste relevant parts from Experiment 3.1 and put it as part of Experiment 3.2. The image shown above is used to verify that the correct image has been used for this experiment.

c. Radon transform

Read 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)

    For example, in a 20-by-30 image, the center pixel is
    (10,15).
```

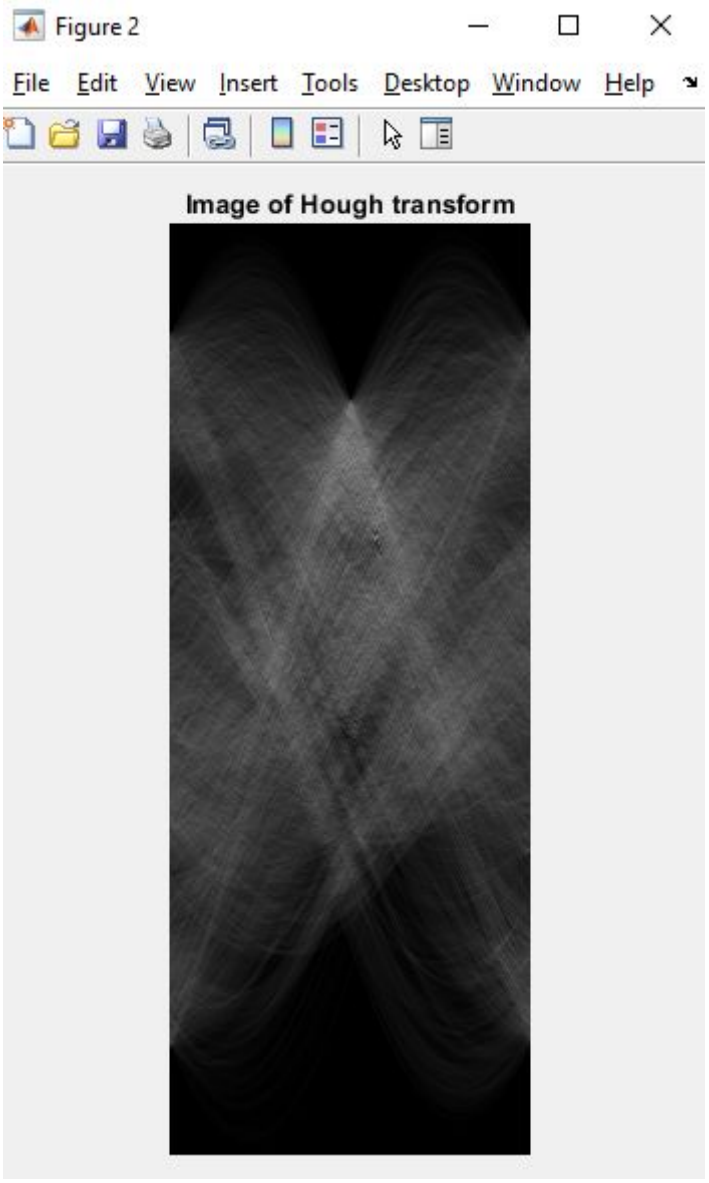
Explanation and comment:
From the definition of the radon function, we know that it computes “the projection of image intensity on a radial line at a specific angle”. Thus, points that lie on a line with a specific angle will be mapped to the same point in the theta domain. Eventually, this is pretty much the same with Hough Transform. The only difference is that since Radon transform takes into account pixel intensity of an image, if the image is not a binary image (thus have pixel intensity of 0-255 instead of 0-1) , then Radon transform may produce different results with Hough transform.

Apply Radon transform and display H as an image

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

```
title('Image of Hough transform');
```

Output:



Explanation and comment:
The image above shows the result of the Hough transformation of the edge image E. We can see that there are certain bright dots in the image, which suggests that there are a lot of points forming a line at that specific angle.

Find the location of the maximum pixel intensity in the Hough image in the form of [theta, radius].

```
[radius, theta]=find(H>=max(max(H)));
radius,theta
```

Output:

```
radius =

   157

theta =

   104
```

Explanation and comment:
I'm utilizing the matlab internal function find to find the indices in the image that have the maximum pixel intensity. It turns out that there is only one such pixel, which is located at [157,104]

d. Derive the equations to convert the [theta, radius] line representation to the normal line equation form Ax + By = C

Show that A and B can be obtained via the given MATLAB commands

```
>> help pol2cart
 pol2cart Transform polar to Cartesian coordinates.
    [X,Y] = pol2cart(TH,R) transforms corresponding elements of data
    stored in polar coordinates (angle TH, radius R) to Cartesian
    coordinates X,Y.  The arrays TH and R must the same size (or
    either can be scalar).  TH must be in radians.
```

Looking more closely at the `pol2cart` function guide, we found that it is used to transform polar to cartesian coordinates, given angle `TH` in radians. Meanwhile, our current definition of theta is in degree. Thus, we will have to convert it to radians.
Therefore, we have our definition of hough point transformation as follows(based on lecture notes):

```
C = x * cos(theta * pi/180) + y * sin(theta * pi/180)
```

Meanwhile, the value of A and B after `[A, B] = pol2cart(theta*pi/180, radius)` are as follows:

```
A=cos(theta * pi/180)
B=sin(theta * pi/180)
```

Putting the values of A and B back to our first equation, we get:

```
C = x * A + y * B
```

Therefore, it can be shown that A and B can be obtained with the given MATLAB commands. An additional step is added where B=-B since the y-axis is defined in bottom-left pointing upwards, whereas it is defined as top-left pointing upwards in the image. Therefore, we need to negate it to put it in the correct position and orientation in the target image.

Code:

```
radius = xp(radius);
[A, B] = pol2cart(theta*pi/180, radius);
% negated because the y-axis is pointing downwards for image
coordinates.
B = - B;
```

Find C

```
% We need to convert back from hough transform with respect to origin at
centre to image coordinates where the origin is in the top-left corner
of the image
% Calculate center of image
[sY, sX]= size(P);
tX = sX/2;
tY = sY/2;
C = A*(A+tX) + B*(B+tY);
C
```

Output:

```
C =

   1.9760e+04
```

Explanation and comment:
We can directly use the formula given in the lab manual to calculate the value of A and B. We used the `size` command to get the size of the image, and then calculate the center of the image. This is done since the Hough transform is done with respect to origin in the centre of the image, meanwhile the image origin is top-left corner of the image.

E. compute yl and yr values for corresponding xl = 0 and xr = width of image - 1

```
% define xl and xr
xl = 0;
xr = sX-1;
% calculate yl and yr
yl = (C - A * xl) / B;
yr = (C - A * xr) / B;
yl,yr
```

Output:

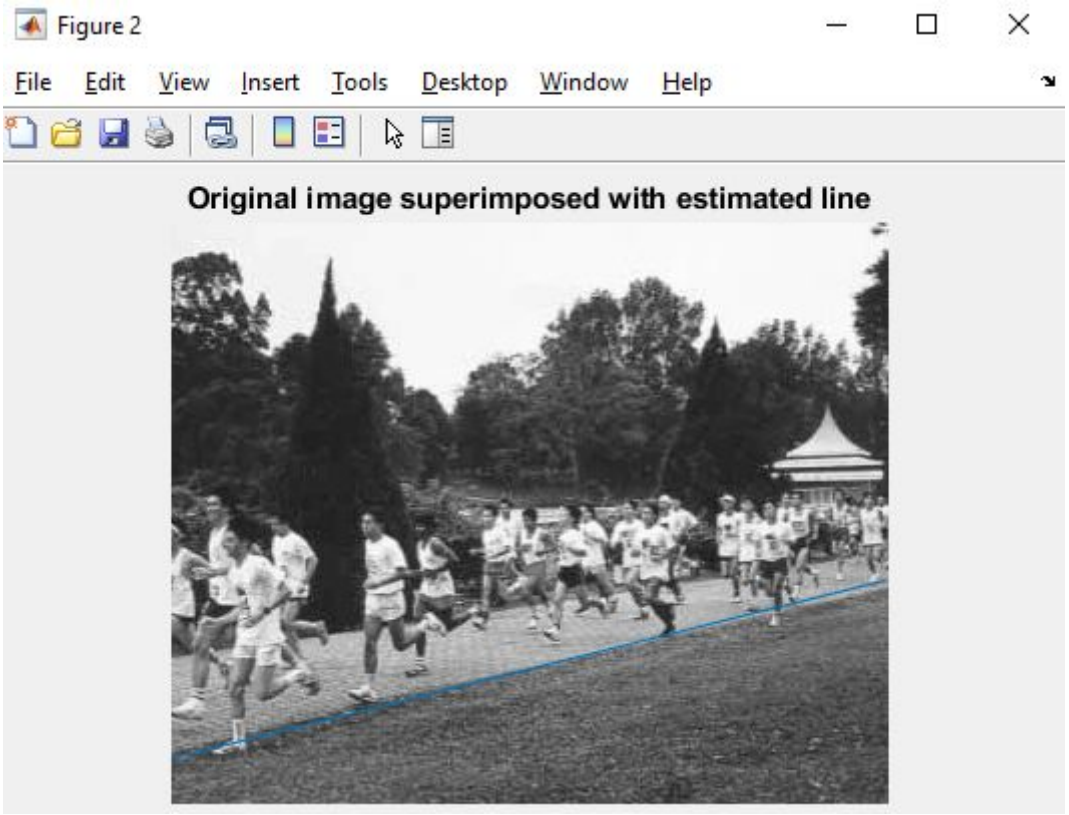
```
yl =

  267.9563


yr =

  178.9463
```

Explanation and comment:
We initially defined xl=0 and xr =Sx(width of image)-1 as instructed in the lab manual. To compute y, we can rearrange the equation Ax+By=C into By=C-Ax then dividing by B, we get y=(C-Ax)/B. Therefore, we can use this to calculate yl and yr.

F. Display the original ‘macritchie.jpg’ superimposed with estimated line

```
figure;
imshow(P);
line([xl xr], [yl yr]);
title('Original image superimposed with estimated line');
```

Output:



Explanation and comment:
It can be seen that the line matches up quite closely with one edge of the running path. However, it does not perfectly match with that line perfectly. One possible explanation for this is that since we are using integer bins of 0-179 degrees as our theta, the resulting theta

might not be that accurate. However, I don't think there is not much we can do regarding this limitation. Another possible explanation for this is that there is still some noise near the edge of the running path in the original Canny image. Therefore, I tried to rectify it by applying the same algorithm to the Canny Edge image with sigma=5. It seems that the resulting line fits more perfectly to the edge of the running path.
Code:

```
%% Fix the line to match up more closely with the edge

% Image computed via the Canny algorithm with sigma=5.0
tl = 0.04;
th = 0.1;
sigma = 5.0;

E = edge(P, 'canny', [tl th], sigma);

% Radon transform
[H, xp] = radon(E);

% Find maximum pixel intensity
[radius, theta]=find(H>=max(max(H)));
radius = xp(radius);
% Find C
[A, B] = pol2cart(theta*pi/180, radius);
B = - B;
[sY, sX]= size(P);
tX = sX/2;
tY = sY/2;
C = A*(A+tX) + B*(B+tY);

% compute yl and yr values for corresponding xl = 0 and xr = width of
image - 1
% define xl and xr
xl = 0;
xr = sX-1;
% calculate yl and yr
yl = (C - A * xl) / B;
yr = (C - A * xr) / B;

% Display the original 'macritchie.jpg' superimposed with estimated line
figure;
imshow(P);
line([xl xr], [yl yr]);
title('Original image superimposed with estimated line');
```

Output:

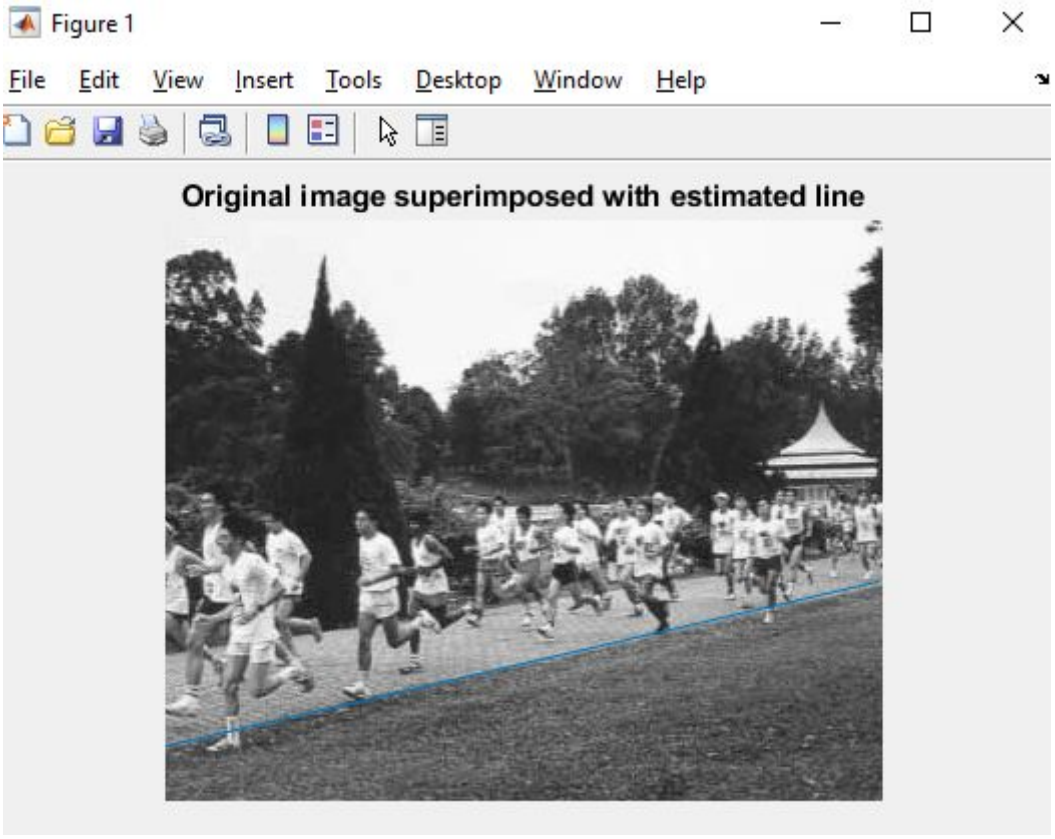


Figure 1
File Edit View Insert Tools Desktop Window Help
Original image superimposed with estimated line

3.3 3D Stereo

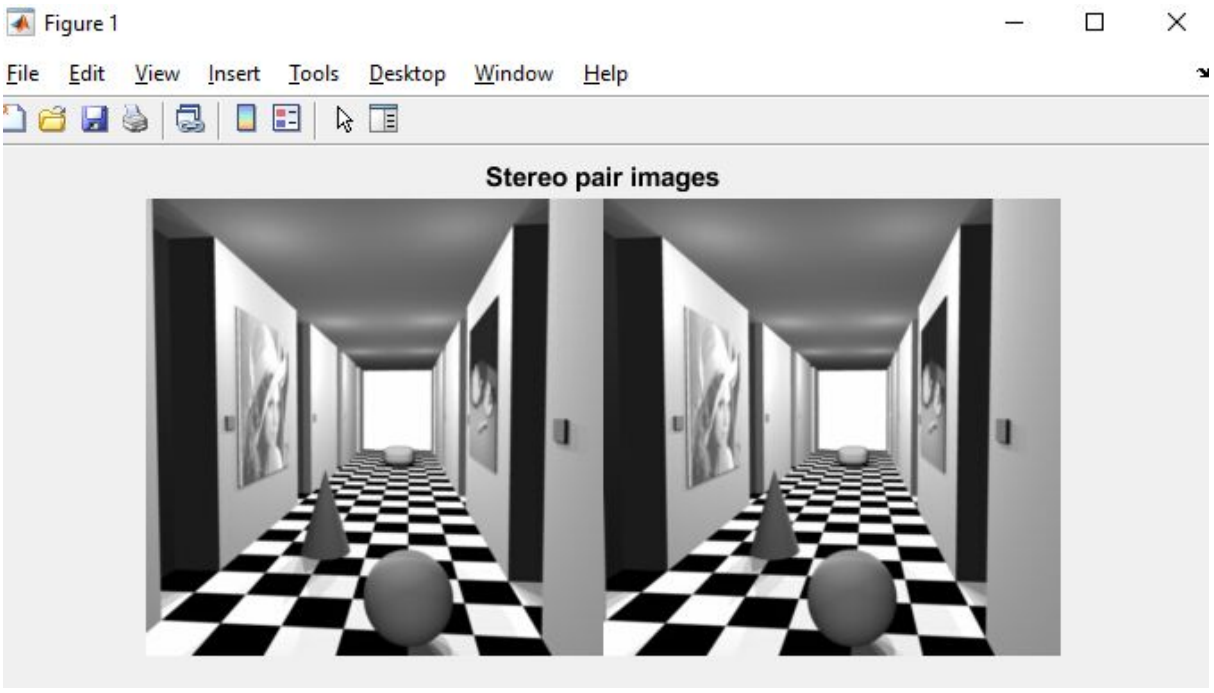
a. Write the disparity map algorithm as a MATLAB function script

```
function map_result = disparity_mapping(Pl, Pr, t_height, t_width)

[P_height, P_width] = size(Pl);
map_result = ones(P_height, P_width);

t_height_center = idivide(t_height,int32(2));
t_width_center = idivide(t_width, int32(2));

% define search constraint to small values of disparity (<15).
s_limit = 14;

% Iterate through all pixels in Pl
for y = t_height_center+1 : P_height-t_height_center
      for x = t_width_center+1 : P_width-t_width_center
      % Extract image patch T region around the pixel
      T = Pr(y-t_height_center : y+t_height_center, x-t_width_center :
x+t_width_center);
      T_flipped = rot90(T,2);
      % Extract template region of image I around the pixel
      I = Pl(y-t_height_center : y+t_height_center,
max(x-s_limit,t_width_center)  : min(x+s_limit,P_width));
      % Compute SSD using the alternative approach
      S = conv2(I.^2, ones(t_height, t_width), 'same') ... % 1st part
            + sum(sum(T_flipped.^2)) ...   % 2nd part
            - 2*conv2(I, T_flipped, 'same'); %3rd part
      % Use `find` function to get the indices (relative x-coord) of
      % minimum SSD within the specified search range
      xr = find(S(t_width_center+1, :) == min(S(t_width_center+1, :)),
1);
      % Get the disparity value, which in our case is how far our point
      % from (x-s_limit: x+s_limit) to midpoint indices
      map_result(y, x) = idivide(size(S,2),int8(2))-xr;
      end
end
```

b. Load stereo pair images and convert to grayscale

```
Pc_left = imread('corridorl.jpg');
P_left = rgb2gray(Pc_left);
Pc_right = imread('corridorr.jpg');
P_right = rgb2gray(Pc_right);
figure;
imshowpair(P_left, P_right, 'montage');
title('Stereo pair images');
```

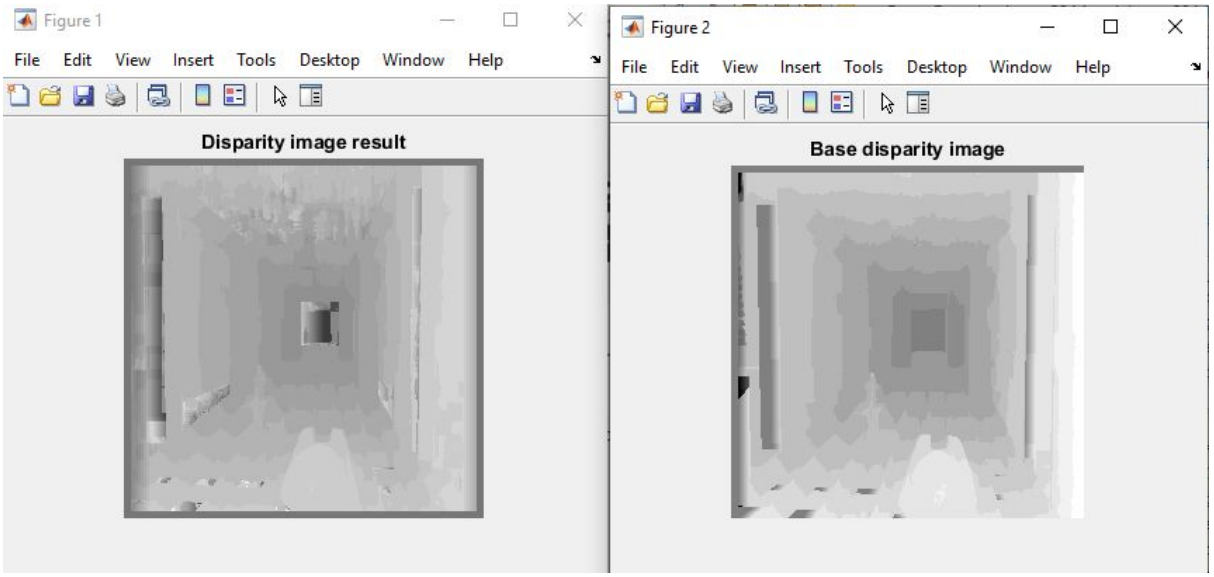
Output:



c. Obtain a disparity map D, and see the result

```
D = disparity_mapping(im2double(P_left), im2double(P_right), 11, 11);
% D = disparity_mapping(P_left, P_right);
figure;
imshow(-D, [-15 15]);
title('Disparity image result');
P_disp = imread('corridor_disp.jpg');
figure;
imshow(P_disp);
title('Base disparity image');
```

Output:



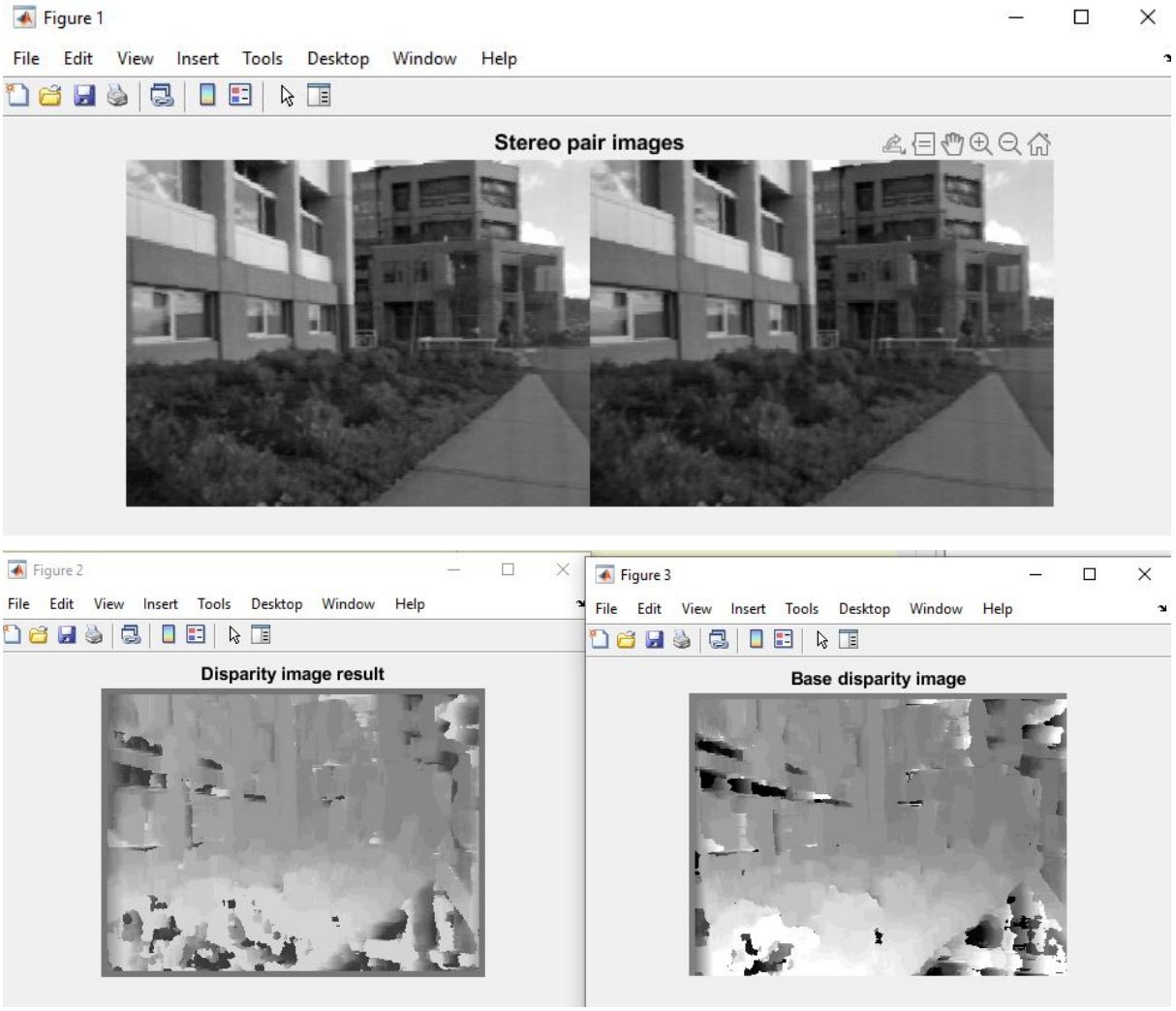
Explanation and comment:
At glance, the disparity image result looks rather similar with the base disparity image, which is a good indication that the algorithm has been implemented correctly.The qualities of the disparity image is also almost as good as the base image. It starts with bright parts in the borders and gets darker going into the center region.
The most noticeable difference is the fact that the center of the disparity image is far darker than the base disparity image. This may be caused by the fact that the center of the original images both has a plain white wall, which then might result in those areas having similar color, and thus almost no disparity is detected in the center. Similarly, it can also be seen that the black shadow area in the left side of the original images cannot be computed smoothly in the disparity image result.

d. Rerun algorithm on the real images of ‘triclops-i2l.jpg’ and ‘triclops-i2r.jpg’

```
Ptc_left = imread('triclops-i2l.jpg');
Pt_left = rgb2gray(Ptc_left);
Ptc_right = imread('triclops-i2r.jpg');
Pt_right = rgb2gray(Ptc_right);

figure;
imshowpair(Pt_left, Pt_right, 'montage');
title('Stereo pair images');

D = disparity_mapping(im2double(Pt_left), im2double(Pt_right), 11, 11);
figure;
imshow(-D, [-15 15]);
title('Disparity image result');
Pt_disp = imread('triclops-id.jpg');
figure;
```

```
imshow(Pt_disp);
title('Base disparity image');
```

Output:



Explanation and comment:

It can be seen that the disparity image result also looks quite similar with the base disparity image in general. However, it is not as similar as the result in part C. For example, the disparity mapping of the roadwalk in the obtained result is pretty different with the one in the base image. This might be caused by the fact that the roadwalk pretty much has similar colors and thus cannot be mapped properly with SSD. This also happens with parts of the building on the left side of the image.

3.4 Optional Task

Implementation of the algorithm in the CVPR 2006 paper entitled “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”

In order to implement the Spatial Pyramid Matching experiment, I decided to use Python as the programming language. We first need to download the dataset “Caltech-101”. Afterwards, we will load the images from the dataset and perform feature extraction using the SIFT algorithm. Fortunately, OpenCV already has an implementation for this, so we can use it directly. The following are the relevant code parts to perform the operation.

```
images_path = "./101_ObjectCategories"
categories = os.listdir(images_path)
# Number of train images and test images used based on paper
num_train = 5
num_test = 10
# Number of categories to be used
num_cat = 200

train_descriptors = dict()
test_descriptors = dict()
train_kp = dict()
test_kp = dict()
train_img_sizes = dict()
test_img_sizes = dict()

for cat_idx, category in enumerate(categories):
      if cat_idx >= num_cat:
      break
      print("Processing category {}".format(category))
      train_descriptors[category] = list()
      test_descriptors[category] = list()
      train_kp[category] = list()
      test_kp[category] = list()
      train_img_sizes[category] = list()
      test_img_sizes[category] = list()

      image_category_path = os.path.join(images_path, category)
      image_categories = os.listdir(image_category_path)
      for idx, image_name in enumerate(image_categories):
      if idx >= num_train + num_test:
            break
      # Load and change rgb to grayscale as per paper description
      image = cv.imread(os.path.join(image_category_path, image_name))
```

```
        gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
        width, height, channels = image.shape
        kp, descriptors = SpatialPyramidMatching.get_sift_features(image)
        if idx < num_train:
                train_descriptors[category].append(descriptors)
                train_kp[category].append(kp)
                train_img_sizes[category].append((height, width))
        else:
                test_descriptors[category].append(descriptors)
                test_kp[category].append(kp)
                test_img_sizes[category].append((width, height))
```

Get sift features implementation:

```
@staticmethod
        def get_sift_features(gray_image):
        """
        Generate sift features
        https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html
        """
        sift = cv.SIFT_create()
        kp, des = sift.detectAndCompute(gray_image, None)
        kp = [i.pt for i in kp]
        return kp, des
```

After we have finished extracting the features, we proceed with k-means clustering with M clusters to form a visual vocabulary. I decided to test with M = 200, which is the same number of features used in the Table 2 in Section 5.2 of the paper. I used the K-Means clustering implemented in sklearn to perform this method.

```
# Set visual vocabulary size, 200 is used in Table 2 as strong features
M = 200
clusters = KMeans(n_clusters=M)
clusters.fit(features)
print("Clustering completed!")
```

After the visual vocabulary has been constructed, I proceed with the extraction of features using the spatial pyramid matching algorithm described in the paper. Afterwards, I proceed with model training and testing using the SVM model which has also been implemented in sklearn. For the spatial pyramid matching algorithm, I used several publicly available source codes as reference [3][4] since I find it quite difficult to comprehend and implement the algorithms solely from the paper.

```
# Set the level L
for L in range(4):
        print("Starting building level {}".format(L))
```

```
      X_train, Y_train =
SpatialPyramidMatching.get_spatial_pyramid(train_descriptors, train_kp,
train_img_sizes, clusters, L, M)
      X_test, Y_test =
SpatialPyramidMatching.get_spatial_pyramid(test_descriptors, test_kp,
test_img_sizes, clusters, L, M)
      model = LinearSVC()
      model.fit(X_train, Y_train)
      print("Level {} accuracy result: {}".format(L,
accuracy_score(Y_test, model.predict(X_test))))
```

Below is the implementation for the spatial pyramid matching algorithm implementation:

```
@staticmethod
      def get_spatial_pyramid(descriptors, kp, image_sizes,
visual_vocabs, L=3, M=200):
      """
      Transforms the images into spatial pyramid
      """
      X = []
      Y = []

      for des in descriptors:
            for idx, descriptor in enumerate(descriptors[des]):
                  sigma_L = (-1 + 4 ** (L + 1)) // 3
                  if descriptor is None:
                  X.append([0]* (M * sigma_L))
                  Y.append(des)
                  continue
                  result_vector = []
                  channels = {}
                  channels_kp = {}
                  width, height = image_sizes[des][idx]

                  # Get K-means prediction of current image descriptor
based on the generated visual vocabularies
                  predictions = visual_vocabs.predict(descriptor)

                  for pred_idx, prediction in enumerate(predictions):
                  if prediction not in channels:
                        channels[prediction] = []
                        channels_kp[prediction] = []

channels[prediction].append(descriptors[des][idx][pred_idx].tolist())
                  channels_kp[prediction].append(kp[des][idx][pred_idx])

                  # Iterate through all channel m
```

```
                  for channel in range(M):
                  if channel not in channels:
                        result_vector += [0] * sigma_L
                        continue
                  for l in range(L + 1):
                        w = 1 / (2 ** min((L - l + 1), L))

                        # Define histogram
                        hist = [0] * (4 ** l)

                        # Fill the histogram
                        for position in channels_kp[channel]:
                              x, y = position
# Get the grid location of (x,y) position in image
                              grid = SpatialPyramidMatching.get_grid(l,
x, y, width, height)
                              try:
                              hist[grid] += 1
                              except IndexError:
                              hist[-1] += 1

                        hist = [it * w for it in hist]

                        result_vector += hist

                  X.append([it / (((M / 200) * 25) * (2 ** L)) for it in
result_vector])
                  Y.append(des)
      return X, Y
```

Results:

```
Starting building level 0
Level 0 accuracy result: 0.35
Starting building level 1
Level 1 accuracy result: 0.33
Starting building level 2
Level 2 accuracy result: 0.28
Starting building level 3
Level 3 accuracy result: 0.26
```

Explanations and comments:

Somehow in my implementation the level of accuracy drops when the depth L is increased from 0(original Bag-of words method) to 3(spatial pyramid matching with depth L=3). This might be due to some implementation error on my code. Nevertheless, I failed to find which part of my code introduces this bug. I suspected that there might be some issue during the feature extraction of the spatial pyramid algorithm.

Additional Information

The original source codes will be uploaded alongside the submission of this report, and are also available at https://github.com/hanstananda/CZ4003-Lab2.

References

1. S. Lazebnik, C. Schmid and J. Ponce, "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories," *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, New York, NY, USA, 2006, pp. 2169-2178, doi: 10.1109/CVPR.2006.68.
2. L. Fei-Fei, R. Fergus and P. Perona. *Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories*. IEEE. CVPR 2004, Workshop on Generative-Model Based Vision. 2004
3. Pentium, Gede Bagus Ayu. 'Spatial Pyramid Matching Implementation in Python'. *GitHub*
4. Image-Recognition - Recognize images using Spatial Pyramid Matching & Earth Mover's Distance
5. CZ4003 Lecture Notes