Lab Assignment 2 Report

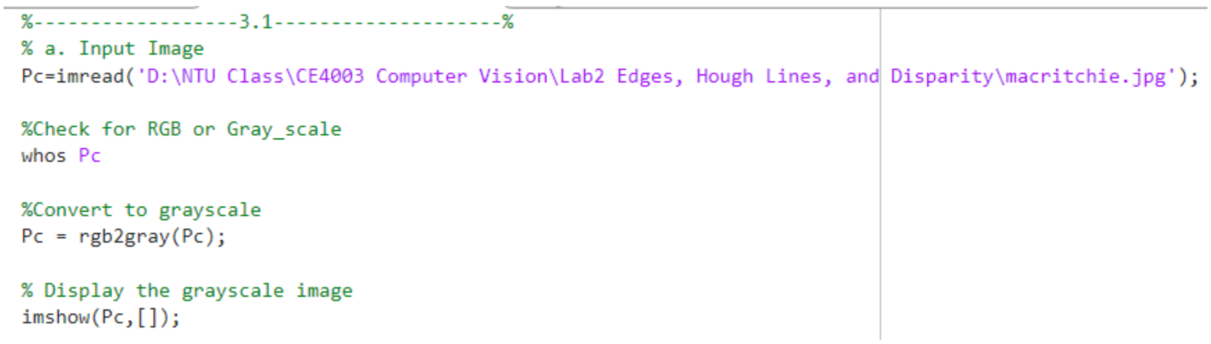
Name: Vedula Kartikeya

Matriculation Number: U1923891G

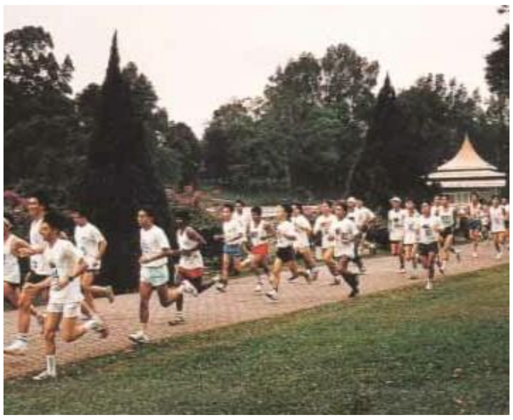
Subject Code: CE4003

3.1 <u>Edge Detection:</u>

a. The following Matlab code is used to convert “macritchie.jpg” image to gray scale and display it:



```
%------------------3.1-------------------%
% a. Input Image
Pc=imread('D:\NTU Class\CE4003 Computer Vision\Lab2 Edges, Hough Lines, and Disparity\macritchie.jpg');

%Check for RGB or Gray_scale
whos Pc

%Convert to grayscale
Pc = rgb2gray(Pc);

% Display the grayscale image
imshow(Pc,[]);
```



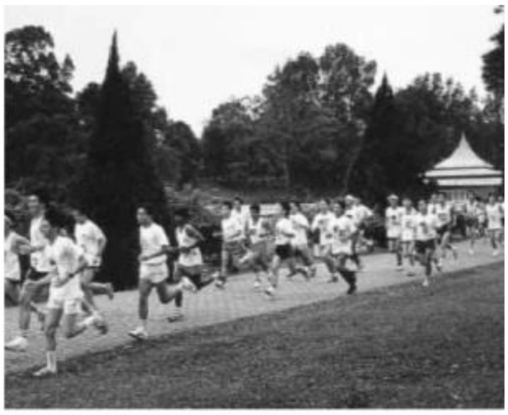


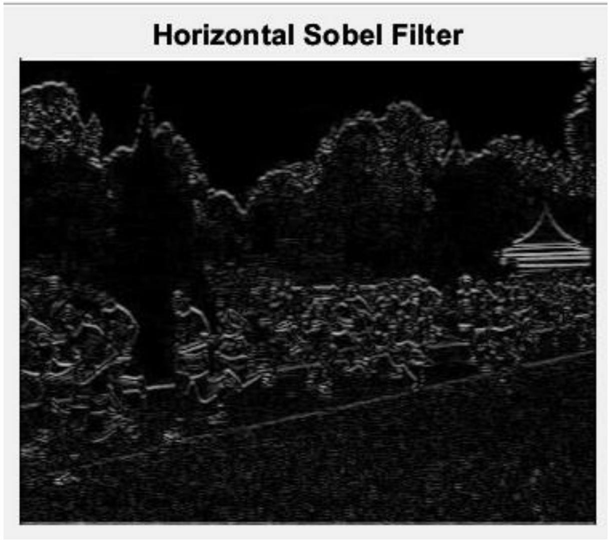
Figure
1: Original ‘macritchie.jpg’ image

Figure 2: Grayscale ‘macritchie.jpg’ image

b. The following Matlab code is sued to create a 3x3 horizontal and vertical Sobel masks and filter the image using conv2:

```
  % b. Create 3x3 horizontal and vertical sobel mask
  horiSobel = [-1 -2 -1;
               0 0 0;
               1 2 1];
  vertiSobel = [-1 0 1;
               -2 0 2;
               -1 0 1];

  % Convolution image with sobel mask created above
  image = conv2(double(Pc),double(horiSobel));
  imagehori =  abs(image)/4;

  imageverti= conv2(double(Pc),double(vertiSobel));
  imageverti = abs(imageverti)/4;

  % Display horizontal filter
  figure;
  imshow(imagehori, []);
  title('Horizontal Sobel Filter');
  % Display vertical filter
  figure;
  imshow(imageverti, []);
  title('Vertical Sobel Filter');

% diagonal edges detection
diagSobel = [ 0 1 2;
             -1 0 1;
             -2 -1 0];
imagediagonal= conv2(double(Pc),double(diagSobel));
imagediagonal = abs(imagediagonal)/4;

% Display horizontal filter
figure;
imshow(imagediagonal, []);
title('Detect Diagonal edges');
```

Resultant image after filtering with horizontal and vertical masks are as follows:



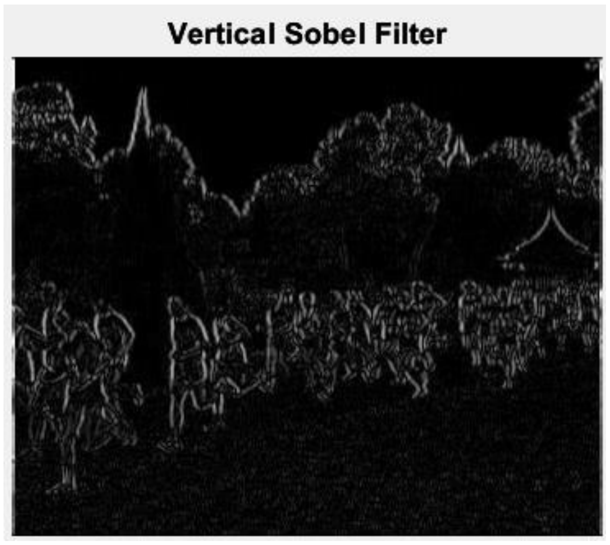


Figure 3: Filtered image with Horizontal Sobel Mask

Figure 4: Filtered image with Vertical Sobel Mask

However, we can see that there are some edges that are neither vertical nor horizontal, i.e., diagonal. As seen by the diagonal lines representing the tracks in the image, horizontal filters appear to detect these diagonal edges better than vertical filters. I employed the following kernel to detect diagonal edges with the Sobel operator: [0 1 2;-1 0 1;-2 -1 0].

c. Following Matlab code is used to generate a combined image by squaring the horizontal and vertical edge images and display the image:

```
% c. Squaring horizontal and vertical edge images and adding them
imgcombedge = sqrt(imagehori.^2+imageverti.^2);
figure;
imshow(imgcombedge, []);
title('Combined Sobel Filter');
```

The reason behind carrying out square operation is that there won’t be any negative values left

(-ve*-ve = +ve), and it is hence easier to calculate the Sobel matrix.

The resultant image is as follows:

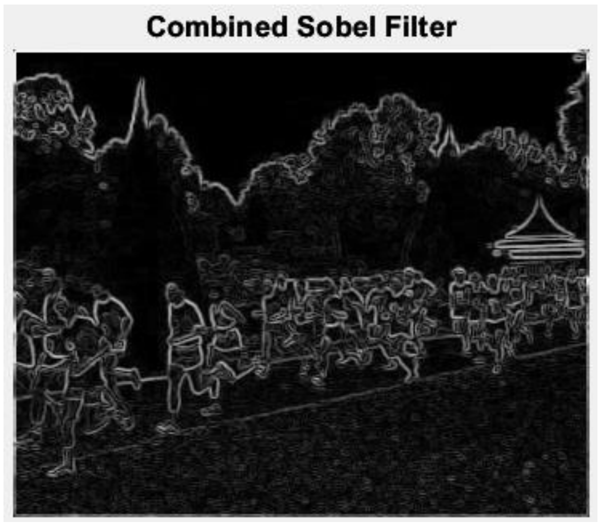
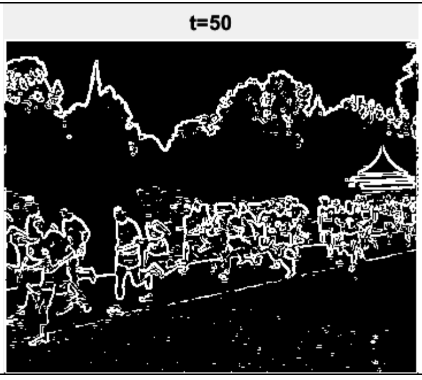
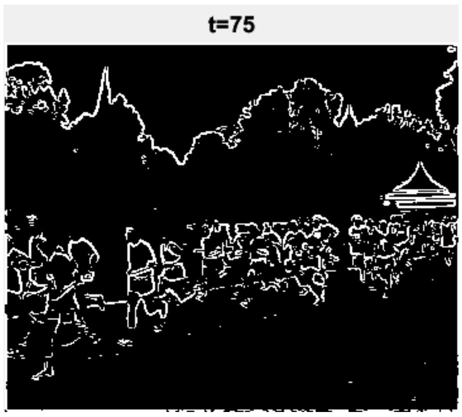


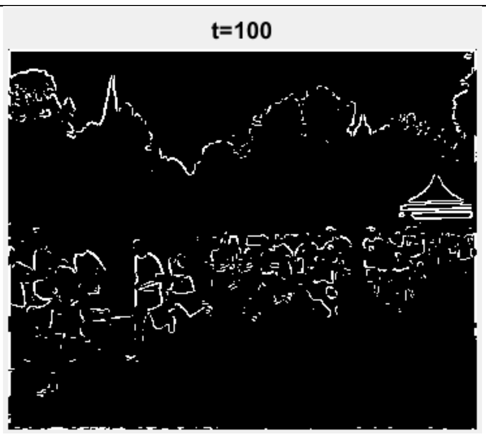
Figure 5: Filtered image after adding and squaring vertical and horizontal edge images

d. Following code experiments with different threshold values and displays the corresponding binary edge images:

```
% d. Trying different threshold values and displaing the resultant binary image
% threshold(t) = 100
threshold100 = imgcombedge>100;
figure;
imshow(threshold100, []);
title('t=100');

% threshold(t) = 75
threshold75 = imgcombedge>75;
figure;
imshow(threshold75, []);
title('t=75');

% threshold(t) = 50
threshold50 = imgcombedge>50;
figure;
imshow(threshold50, []);
title('t=50');
```

The results of the code are shown below:

	Figure 6: Binary image with threshold>100
	Figure 7: Binary image with threshold>100

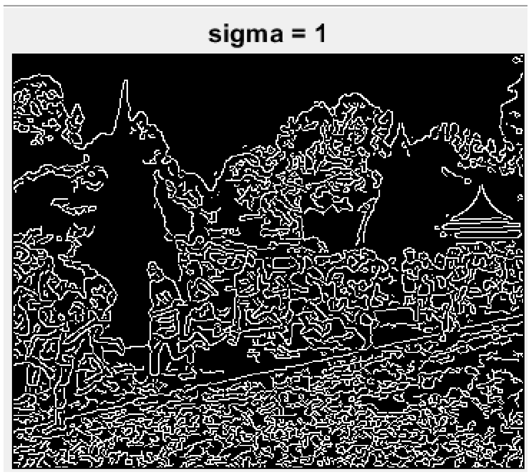
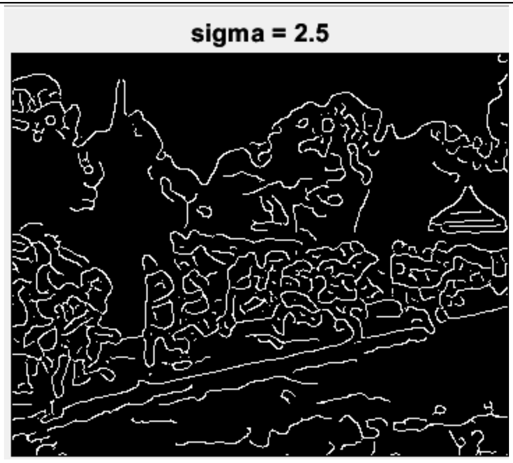
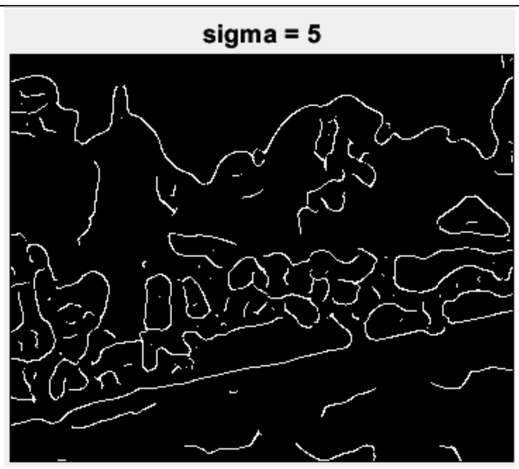
	Figure 8: Binary image with threshold>100

As seen from the table above, after applying threshold to the edge image, it will turn into a binary image. For example, by setting threshold value to 50, intensities less than or equal to 50 will be reduced to 0 (black) and those above will be increased to an intensity of 255 (white).

If the threshold is set too low, the binary image contains too many details, and the edges, which are clearly visible, become part of the extra noise. If the threshold is set too high, too many details are omitted, and edges are obscured. We can achieve the desired result by setting a suitable threshold.

e. i. Following code is used to recompute the edge image with different sigma values ranging from 1.0 to 5.0 and display the effect on images:

```
% e. Recomputing the edge image using the more advanced Canny edge
% detection algorithm
% i. Trying different sigma from 1.0 to 5.0
canny1 = edge(Pc, 'canny', [0.04 0.1], 1.0);
canny2_5 = edge(Pc, 'canny', [0.04 0.1], 2.5);
canny5 = edge(Pc, 'canny', [0.04 0.1], 5.0);
% showing results
figure;
imshow(canny1, []);
title('sigma = 1');
figure;
imshow(canny2_5, []);
title('sigma = 2.5');
figure;
imshow(canny5, []);
title('sigma = 5');
```

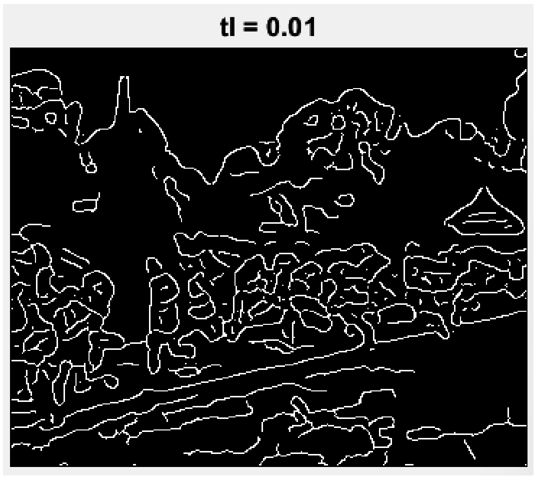
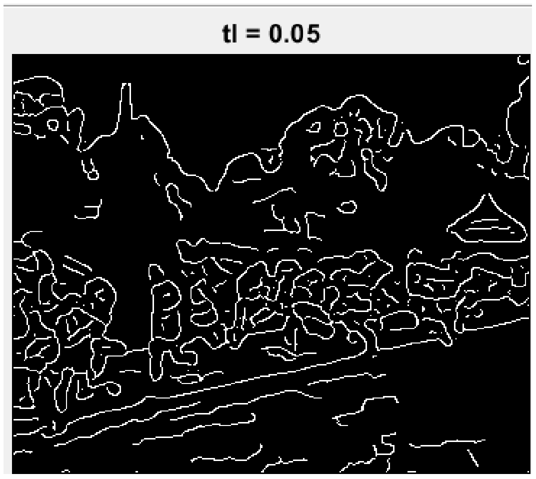
	Figure 9: Edge image with sigma=1.0
	Figure 10: Edge image with sigma=2.5
	Figure 11: Edge image with sigma=5.0

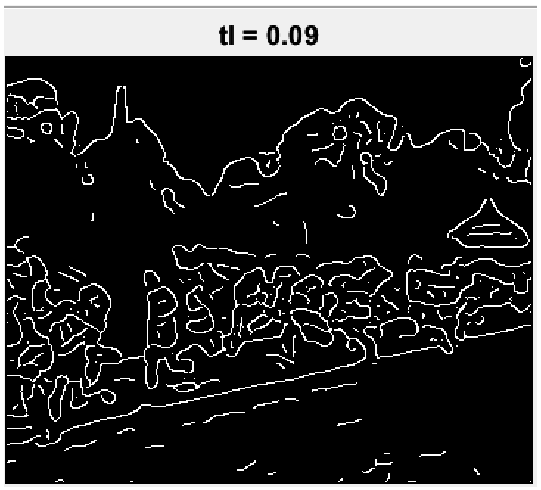
In the first stage of the Canny edge detection algorithm, the Gaussian Filter is used. The "sigma" employed in this filter has a direct impact on the canny detection algorithm's results. Furthermore, the sigma represents the filter's standard deviation, and thus the higher the sigma, the greater the influence on adjacent pixels.
According to the results in the table above, the lower the sigma, the smaller and sharper the edges are, as well as additional noise, such as grass. The higher the sigma, the more noises can be cancelled out; however, we can only detect larger and smoother edges. Hence, we can conclude the following:

- The higher sigma is appropriate for noise edgel removal because it blurs the image while cancelling the noise.
- The lower sigma is appropriate for location accuracy of edgels because we get a more detailed image.

e. ii. The code below is used to recompute the edge image while keeping the sigma constant and experimenting with the value of "tl" and displaying the effect on images:

```
% ii. Trying raising and lowering the value of tl, keeping sigma at 3
canny3_1 = edge(Pc, 'canny', [0.01 0.1], 3.0);
canny3_2 = edge(Pc, 'canny', [0.05 0.1], 3.0);
canny3_3 = edge(Pc, 'canny', [0.09 0.1], 3.0);
% showing results
figure;
imshow(canny3_1, []);
title('tl = 0.01');
figure;
imshow(canny3_2, []);
title('tl = 0.05');
figure;
imshow(canny3_3, []);
title('tl = 0.09');
```

	Figure 12: Edge image with tl=0.01
	Figure 13: Edge image with tl=0.05

	Figure 14: Edge image with tl=0.01

Hysteresis thresholding is used in the final stage of the Canny edge detection algorithm to set a low threshold tl and a high threshold th. Edge pixels less than tl will be discarded, while edge pixels greater than high threshold will be considered edges if they meet the condition.

More details and edges can be displayed as the tl decreases. Using the provided threshold value, the canny algorithm can divide each pixel into edges, non-edges, and in-between. The pixel in the middle could also be an edge pixel if it is connected to other edge pixels. As a result, tl represents the minimum threshold, and when tl is lower, the range is larger, allowing it to detect more edges.

3.2 Line Finding using Hough Transform:

a. The code below reuses the edge image computed earlier:

```
%------------------3.2-------------------%
% a. Reuse the edge image computed via the Canny algorithm with sigma=1.0
cannyRe = edge(Pc, 'canny', [0.04 0.1], 1.0);
figure;
imshow(cannyRe, []);
title('sigma = 1');
theta=0:179;
```

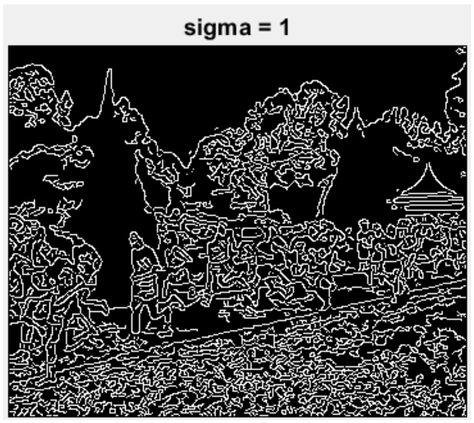


Figure 15: Reusing old image with sigma =1.0

b. Code below computes Hough transform in Matlab using Radon transform:

```
% b. Radon transform on binary image is equivalent to the Hough transform
[H, xp] = radon(cannyRe);
figure; imagesc(uint8(H));title('Image in Hough Space');
xlabel('Theta');
ylabel('Rho');
colormap("hot");
colorbar;
```

To visualize the result, we apply colormap:

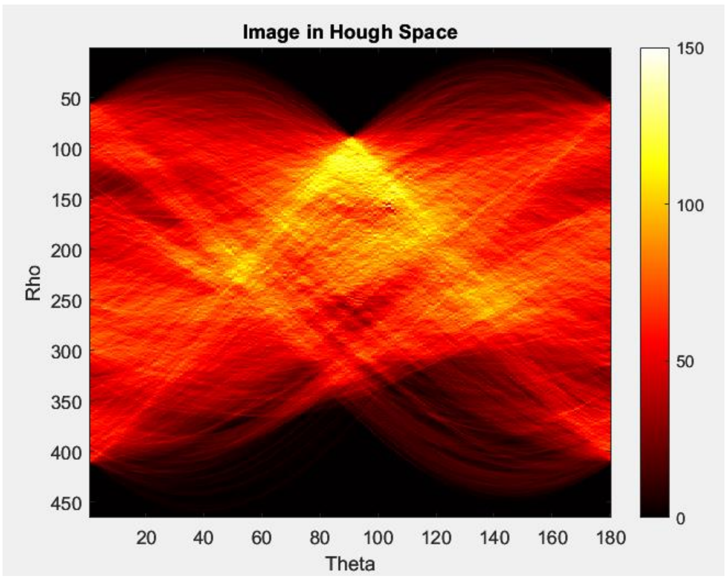


Figure 16: Image in Hough Space

A binary image is made up entirely of vectors of 1s and 0s. Hough transform is commonly used after normal or hysteresis thresholding to produce a binary image. Since this is a binary image, the Radon and Hough transforms are equivalent. When the image is not binary, it would not be the same because the intensity of the pixel may be biased and could not fully represent the maximum intensity as the edge, whereas the Hough transform has the intersection as its straight edge.

c. The location of the maximum pixel intensity in the Hough image in the form of [theta, radius] is as follows:

```
% c. Getting location of max pixel intensity in the Hough image
[max_value,idx]=max(H(:));
[x_max,y_max]=ind2sub(size(H),idx);
%Find line theta and radius with strongest edge support
theta=theta(y_max);
radius=xp(x_max);
fprintf('radius: %d\n', radius);
fprintf('theta: %d\n', theta);
```

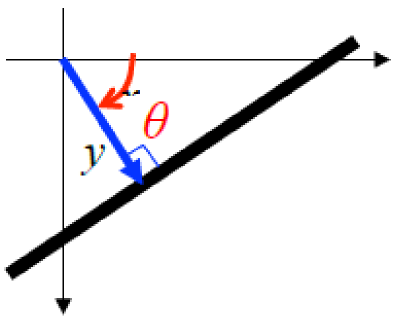
The theta and radius values obtained are:

```
radius: -76
theta: 103
```

d. Code to derive equations to covert [theta, radius] line representation to normal line equation form is as follows:

```
% d. Convert theta, radius to Ax+by=C and find C
[A, B] = pol2cart(theta*pi/180, radius);
B = -B;
```

C of the normal line equation Ax+By=C is calculated as follows,
C1 is the original intercept when the origin has not been shifted. Using figure from tutorial:



$$\rho = x\cos\theta + y\sin\theta$$
$$\Rightarrow y\sin\theta = \rho - x\cos\theta$$
$$\Rightarrow y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$
$$\text{or } y = -\cot\theta\, x + \rho\csc\theta$$

Figure 17: Figure and Equation from tutorial material

As seen from the equation above, original intercept = radius*cosec(theta*pi/180) = radius/sin(theta*pi/180), since y-axis here has changed direction, C1= -radius/sin(theta*pi/180)

```
C1=-(radius/sin(theta*pi/180));
```

C is the new intercept, and from the graph shown below you can see that C=L/2 + the original y value of the intersection of line and new y-axis:

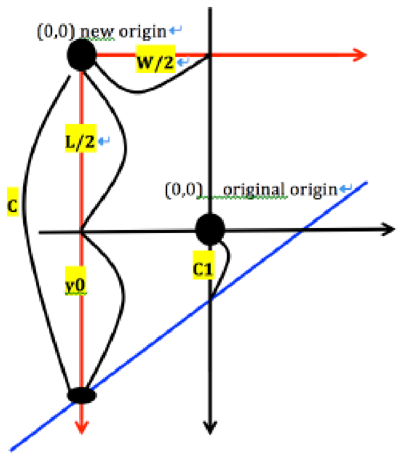


Figure 18: Figure representing the new C intercept

```
[L,W]=size(cannyRe);
x0=-W/2;
y0=(-A/B)*x0+C1;
C = y0+(L/2);
fprintf('A = %d\n', A);
fprintf('B = %d\n', B);
fprintf('C = %d\n', C);
```

The code to calculate A,B, C is shown to the left:

The results displayed are as follows:

```
A = 1.709628e+01
B = 7.405212e+01
C = 2.643245e+02
```

e. Code to compute yl and yr values based on Ax+By=C for corresponding xl=0 and xr=width of image-1 is as follows:

```
% e. Compute yl and yr values and corresponding xl, xr
xl = 0;
yl=(-A/B)*xl+C;
% Width of the image can be obtained from the whos command used in 3.1a
xr = 358-1;
yr=(-A/B)*xr+C;
fprintf('yl = %d\n', yl);
fprintf('yr = %d\n', yr);
```

yl and yr values obtained are as follows:

```
yl = 2.643245e+02
yr = 1.819046e+02
```

f. Following code displays the original 'macritchie.jpg' image superimposed with estimated line:

```
% f. Display the original 'macritchie.jpg by superimposing the line
figure;
imshow(P_original,[]);
line([xl xr], [yl yr], 'Color', 'red');
```

Image displayed is shown below:

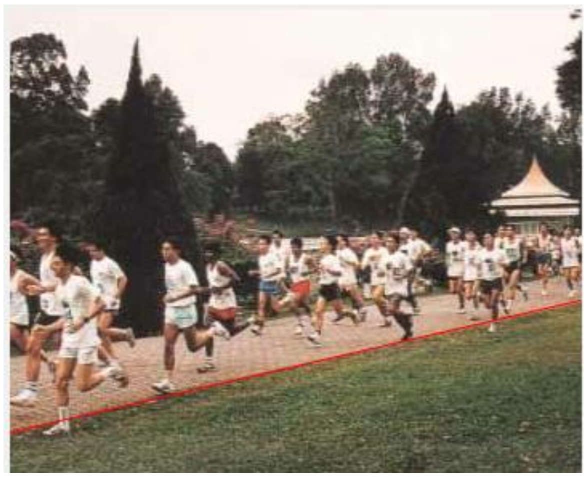


Figure 19: Original 'macritchie.jpg' image superimposed with estimated line

3.3 3D Stereo:

a. Explanation of the disparity map algorithm is as follows:-

i. Initializing variables:

```
% a. Write the disparity map algorithm
%dis_map function
function dispmap = disparitymap(Pl, Pr, m, n)
    [x,y]= size(Pl);
    xs=ceil((m+1)/2);
    ys=ceil((n+1)/2);
    dispmap=ones(x,y);
```

ii. Setting the area outside the boundary of the image to 0, to avoid edge problems:

```
    Pln=zeros(x+xs,y+ys+150);
    Prn=zeros(x+xs,y+ys+150);
    Pln(1:x,15+(ys-1):y+14+(ys-1))=Pl;
    Prn(1:x,15+(ys-1):y+14+(ys-1))=Pr;
```

iii. 15 pixels are moved to avoid any edge problem:

```
    for i=xs:x
        for j=ys+15+(ys-1):y+15+(ys-1)
            template=Pln(i-(xs-1):i+(xs-1),j-(ys-1):j+(ys-1));
            I=Prn(i-(xs-1):i+(xs-1),j-15-(ys-1):j+15+(ys-1));
            TD=double(template);
            TD=rot90(rot90(TD));
            Isquare=double(I).*(double(I));
```

iv. Creating F as an all 1 matrix, and further do a convolution with Isquare, to get the m by n matrix. To achieve this, equation from the sheet was utilized:

```
            F=ones(m,n);
            Ss=conv2(Isquare,F,'same')-2*conv2(double(I),TD,'same');
```

v. Generate the array with the SSD value:

```
            [~,sw]=size(Ss);
```

vi. Obtain only the array which have not been affected by the boundary and has been fully involved in the convolution:

```
            Ssd=Ss(xs,ys:sw-(ys-1));
```

vii. Now, we look for disparity which is the minimum SSD value:

```
            [~,I]=min(Ssd);
```

viii. Finally, we add the value to the disparity matrix. Since we moved the position earlier, we must move it back to preserve the position:

```
                dispmap(i,j-15-(ys-1))=I-15;
            end
        end
    end
```

b. Code shown below downloads the synthetic pair images of ‘corridorl.jpg’ and ‘corridorr.jpg’, and converts them to grayscale:

```
% b. Load the image
corridorl=imread('D:\NTU Class\CE4003 Computer Vision\Lab2 Edges, Hough Lines, and Disparity\corridorl.jpg');
corridorr=imread('D:\NTU Class\CE4003 Computer Vision\Lab2 Edges, Hough Lines, and Disparity\corridorr.jpg');
corridor_disp=imread('D:\NTU Class\CE4003 Computer Vision\Lab2 Edges, Hough Lines, and Disparity\corridor_disp.jpg');
corridorl=rgb2gray(corridorl);
corridorr=rgb2gray(corridorr);
```

c. We now create the disp_map function and run it on the 2 images downloaded before:

```
% c. running the dis_map function
dispmap=disparitymap(corridorl,corridorr,11,11);
imshow(-dispmap,[-15 15]);
```

The results are shown below:

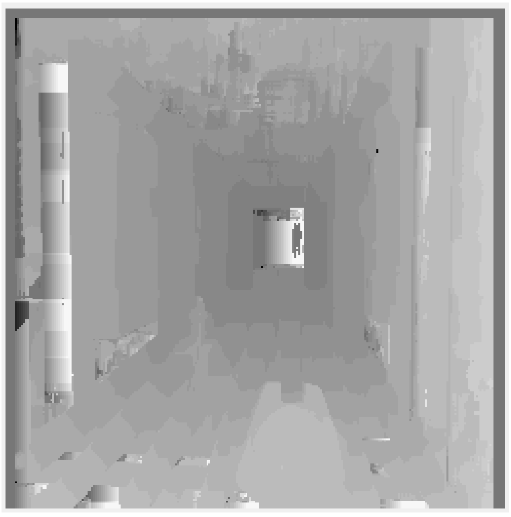


Figure 20: Result image

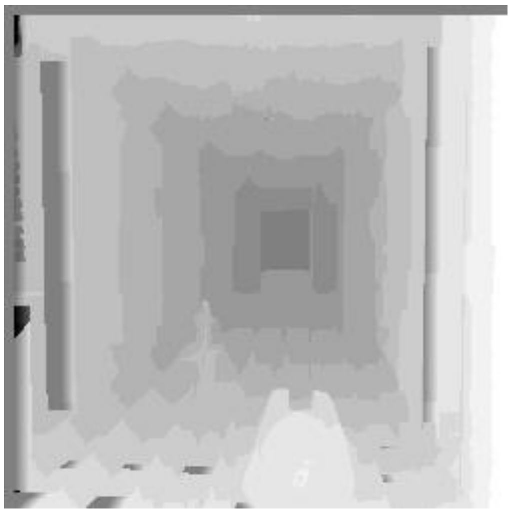
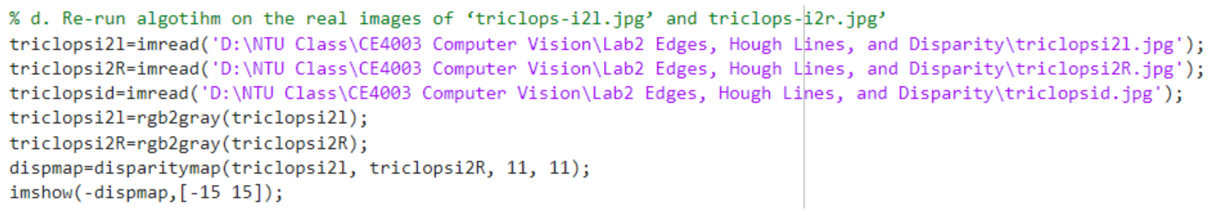


Figure 21: corridor_disp.jpg

The obtained result, corridor disp, is similar to the expected image. However, there is a difference in the deepest part, which also happens to be where the wall is. The disparities quality is poor because the wall should be the deepest part, which is the darkest color. It is difficult to find the match piel from left and right pictures when the part of the image is flat and does not have a lot of different components, such as the wall in this case. This could result in a false result because it matches the wrong pixel and considers it shallow rather than deep.

d. Rerunning algorithm on ‘triclops-i2l.jpg’ and triclops-i2r.jpg’:



```
% d. Re-run algotihm on the real images of ‘triclops-i2l.jpg’ and triclops-i2r.jpg’
triclopsi2l=imread('D:\NTU Class\CE4003 Computer Vision\Lab2 Edges, Hough Lines, and Disparity\triclopsi2l.jpg');
triclopsi2R=imread('D:\NTU Class\CE4003 Computer Vision\Lab2 Edges, Hough Lines, and Disparity\triclopsi2R.jpg');
triclopsid=imread('D:\NTU Class\CE4003 Computer Vision\Lab2 Edges, Hough Lines, and Disparity\triclopsid.jpg');
triclopsi2l=rgb2gray(triclopsi2l);
triclopsi2R=rgb2gray(triclopsi2R);
dispmap=disparitymap(triclopsi2l, triclopsi2R, 11, 11);
imshow(-dispmap,[-15 15]);
```

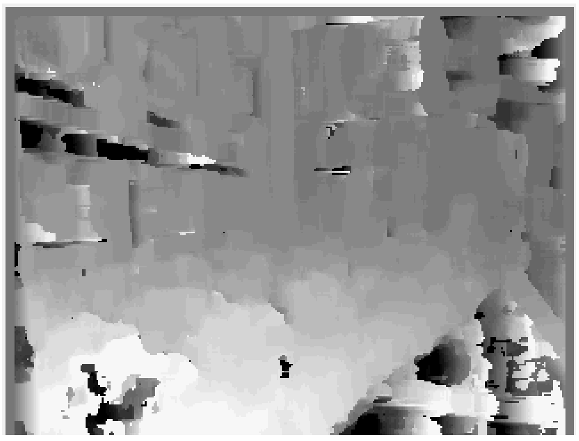


Figure 22: Result image

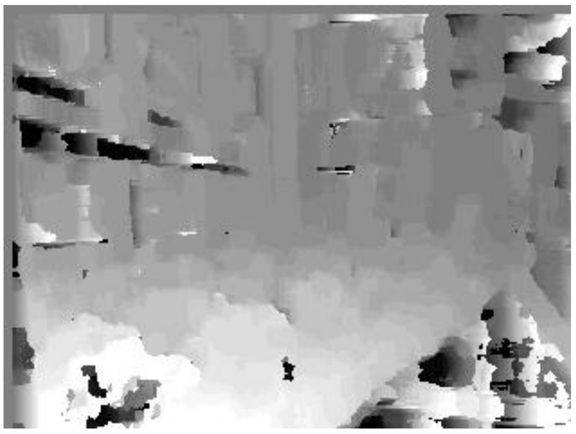


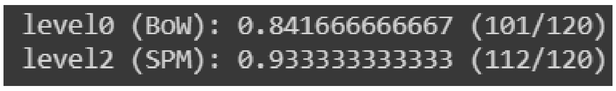
Figure 23: triclopsid.jpg

The resulting image, as seen above, is very similar to triclopsid.jpg. However, upon closer inspection, we can see that the pavement component is not very accurate. This is due to two factors: first, the pavement's components are very similar and could easily detect to the same SSD, and second, the angle of two photos for the pavement is slightly different, which could lead to low accuracy of estimated disparities.

3.4 Optional Part:

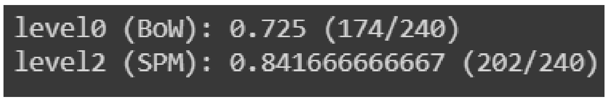
I attempted to completely adapt the algorithm to compare the classification results of SPM [1] [2] and Bag of Words[3] Method by referring to their source codes.

First, I attempted to apply the algorithm using four categories of images from the Caltech-101 dataset (aces, laptop, soccer ball, and starfish). And for each category, I chose 30 training images and 30 testing images. The following is the outcome:



```
level0 (BoW): 0.841666666667 (101/120)
level2 (SPM): 0.933333333333 (112/120)
```

As shown above, two-level SPM outperformed the BoW approach. Both classification rates, however, are high. One possible explanation is that the number of categories is insufficient (only 4). As a result, I added four new categories (Leopards, Brains, Lamp, and Cougar face) and reran the classification algorithm. The newly obtained result was:



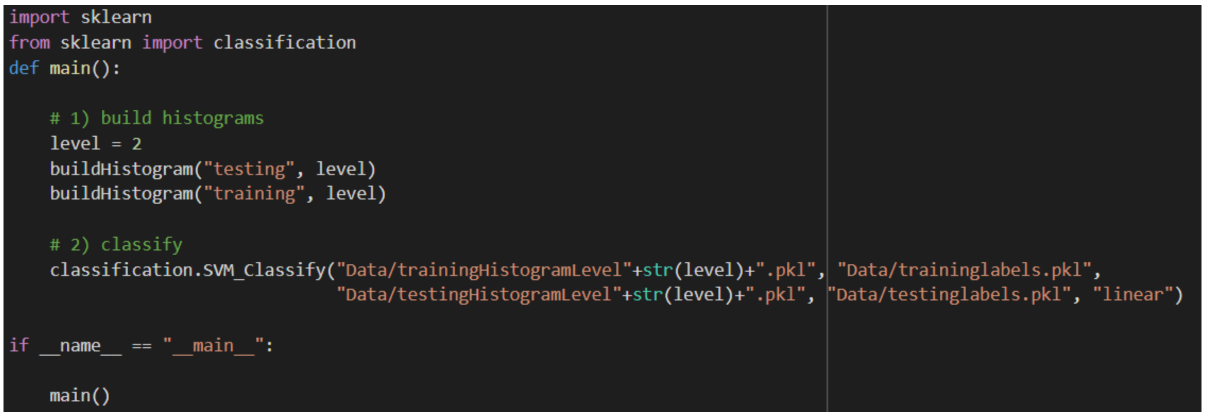
```
level0 (BoW): 0.725 (174/240)
level2 (SPM): 0.841666666667 (202/240)
```

It can be observed that the classification rates are very close to the experiment results reported in the research paper: "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories".

From the results and thorough understanding, I was able to notice that one disadvantage of Bag of Visual Words is that all local features are encoded into a single code vector without regard for the position of the feature descriptors, resulting in spatial information between words being discarded in the final code vector. Thus, to incorporate spatial information into the final code vector, we can use Spatial Pyramid Matching, a simple but effective idea proposed by CVPR 2006 paper.

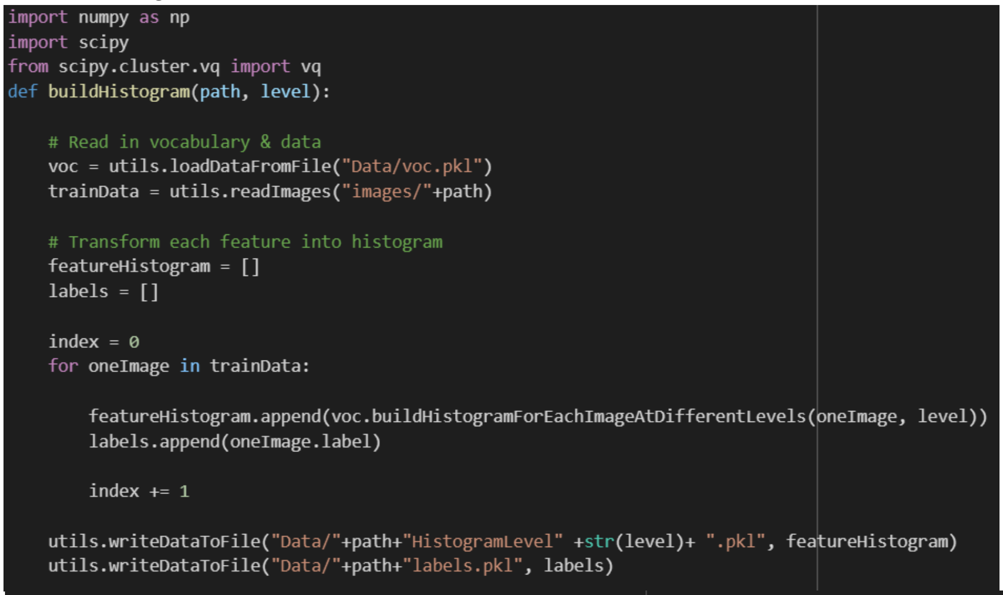
Source Code:

1. Main function:

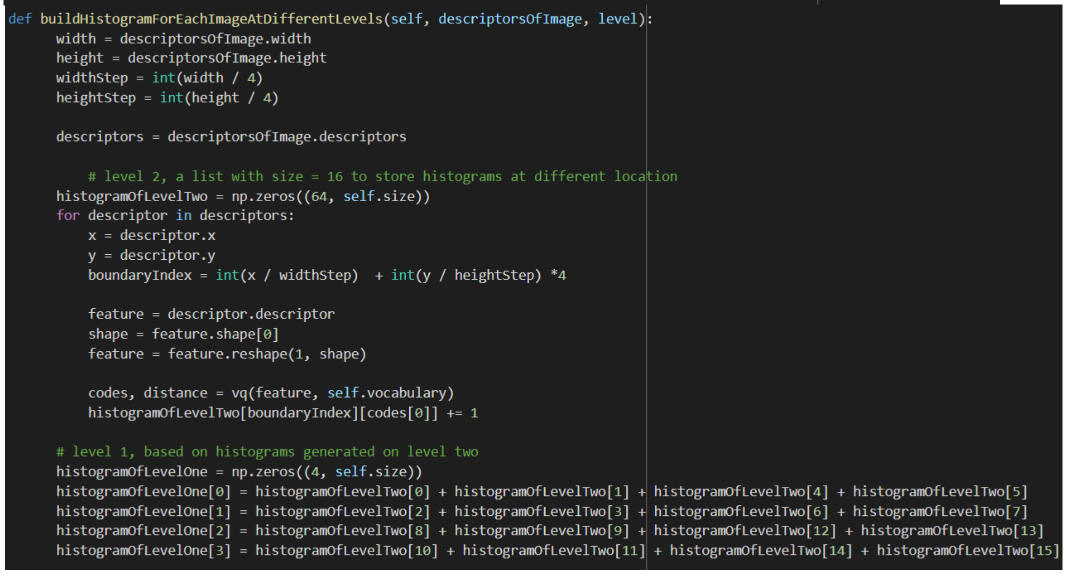


```python
import sklearn
from sklearn import classification
def main():

    # 1) build histograms
    level = 2
    buildHistogram("testing", level)
    buildHistogram("training", level)

    # 2) classify
    classification.SVM_Classify("Data/trainingHistogramLevel"+str(level)+".pkl", "Data/traininglabels.pkl",
                                "Data/testingHistogramLevel"+str(level)+".pkl", "Data/testinglabels.pkl", "linear")

if __name__ == "__main__":

    main()
```

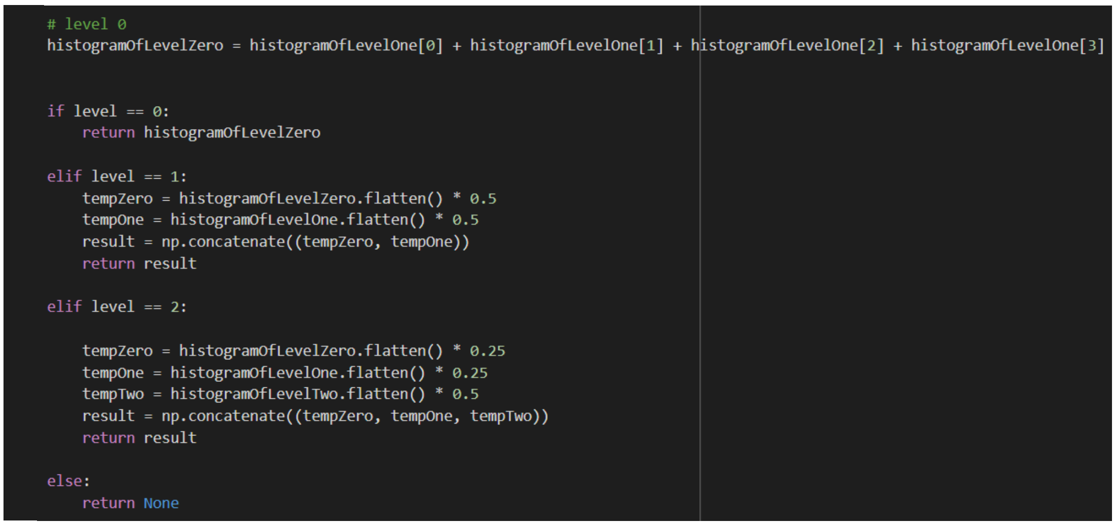
2. Build Histogram:



```
import numpy as np
import scipy
from scipy.cluster.vq import vq
def buildHistogram(path, level):

    # Read in vocabulary & data
    voc = utils.loadDataFromFile("Data/voc.pkl")
    trainData = utils.readImages("images/"+path)

    # Transform each feature into histogram
    featureHistogram = []
    labels = []

    index = 0
    for oneImage in trainData:

        featureHistogram.append(voc.buildHistogramForEachImageAtDifferentLevels(oneImage, level))
        labels.append(oneImage.label)

        index += 1

    utils.writeDataToFile("Data/"+path+"HistogramLevel" +str(level)+ ".pkl", featureHistogram)
    utils.writeDataToFile("Data/"+path+"labels.pkl", labels)
```

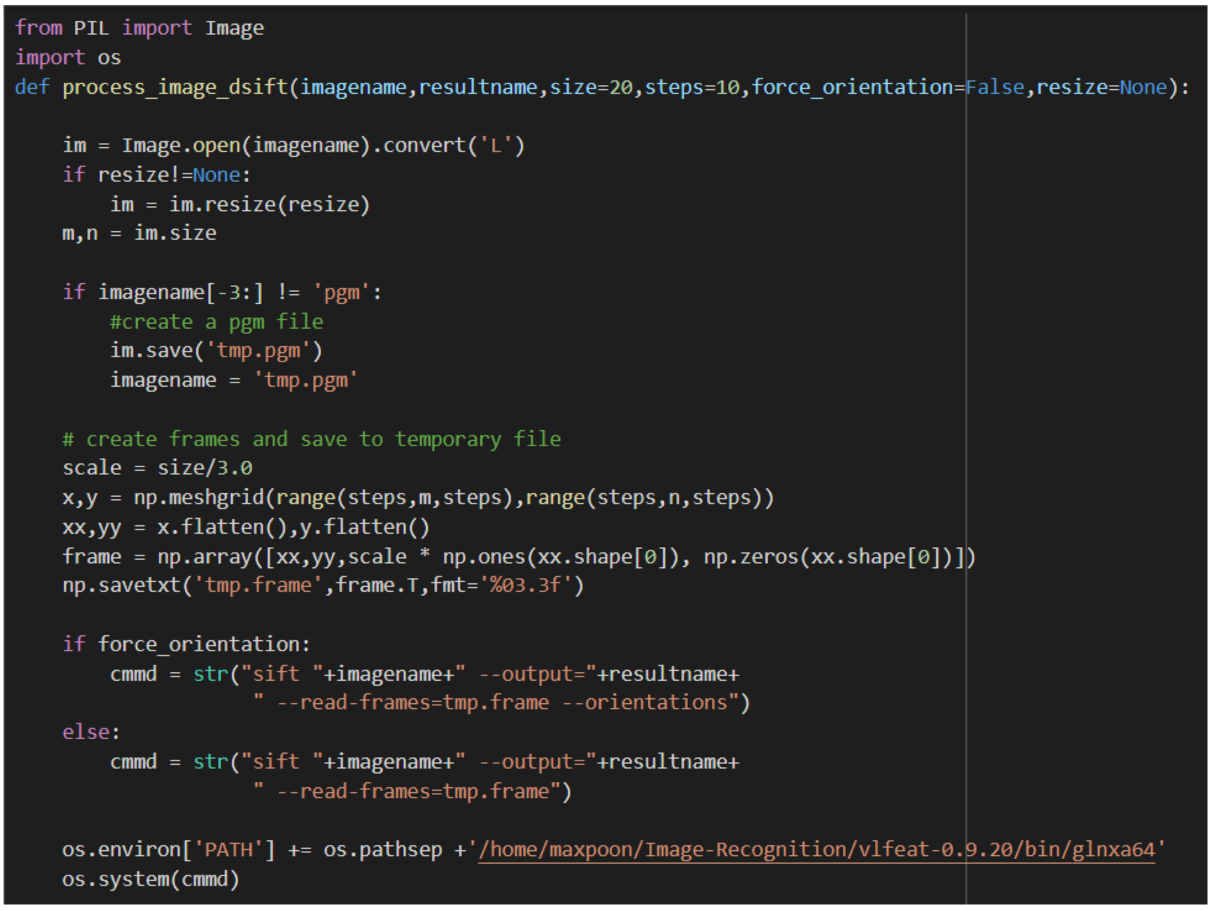


```
def buildHistogramForEachImageAtDifferentLevels(self, descriptorsOfImage, level):
      width = descriptorsOfImage.width
      height = descriptorsOfImage.height
      widthStep = int(width / 4)
      heightStep = int(height / 4)

      descriptors = descriptorsOfImage.descriptors

          # level 2, a list with size = 16 to store histograms at different location
      histogramOfLevelTwo = np.zeros((64, self.size))
      for descriptor in descriptors:
          x = descriptor.x
          y = descriptor.y
          boundaryIndex = int(x / widthStep)  + int(y / heightStep) *4

          feature = descriptor.descriptor
          shape = feature.shape[0]
          feature = feature.reshape(1, shape)

          codes, distance = vq(feature, self.vocabulary)
          histogramOfLevelTwo[boundaryIndex][codes[0]] += 1

      # level 1, based on histograms generated on level two
      histogramOfLevelOne = np.zeros((4, self.size))
      histogramOfLevelOne[0] = histogramOfLevelTwo[0] + histogramOfLevelTwo[1] + histogramOfLevelTwo[4] + histogramOfLevelTwo[5]
      histogramOfLevelOne[1] = histogramOfLevelTwo[2] + histogramOfLevelTwo[3] + histogramOfLevelTwo[6] + histogramOfLevelTwo[7]
      histogramOfLevelOne[2] = histogramOfLevelTwo[8] + histogramOfLevelTwo[9] + histogramOfLevelTwo[12] + histogramOfLevelTwo[13]
      histogramOfLevelOne[3] = histogramOfLevelTwo[10] + histogramOfLevelTwo[11] + histogramOfLevelTwo[14] + histogramOfLevelTwo[15]
```



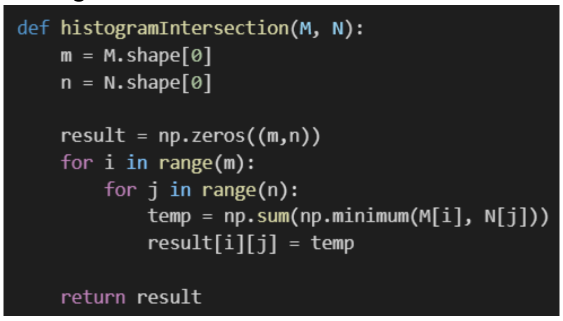
```
# level 0
histogramOfLevelZero = histogramOfLevelOne[0] + histogramOfLevelOne[1] + histogramOfLevelOne[2] + histogramOfLevelOne[3]


if level == 0:
    return histogramOfLevelZero

elif level == 1:
    tempZero = histogramOfLevelZero.flatten() * 0.5
    tempOne = histogramOfLevelOne.flatten() * 0.5
    result = np.concatenate((tempZero, tempOne))
    return result

elif level == 2:

    tempZero = histogramOfLevelZero.flatten() * 0.25
    tempOne = histogramOfLevelOne.flatten() * 0.25
    tempTwo = histogramOfLevelTwo.flatten() * 0.5
    result = np.concatenate((tempZero, tempOne, tempTwo))
    return result

else:
    return None
```

3. SIFT:



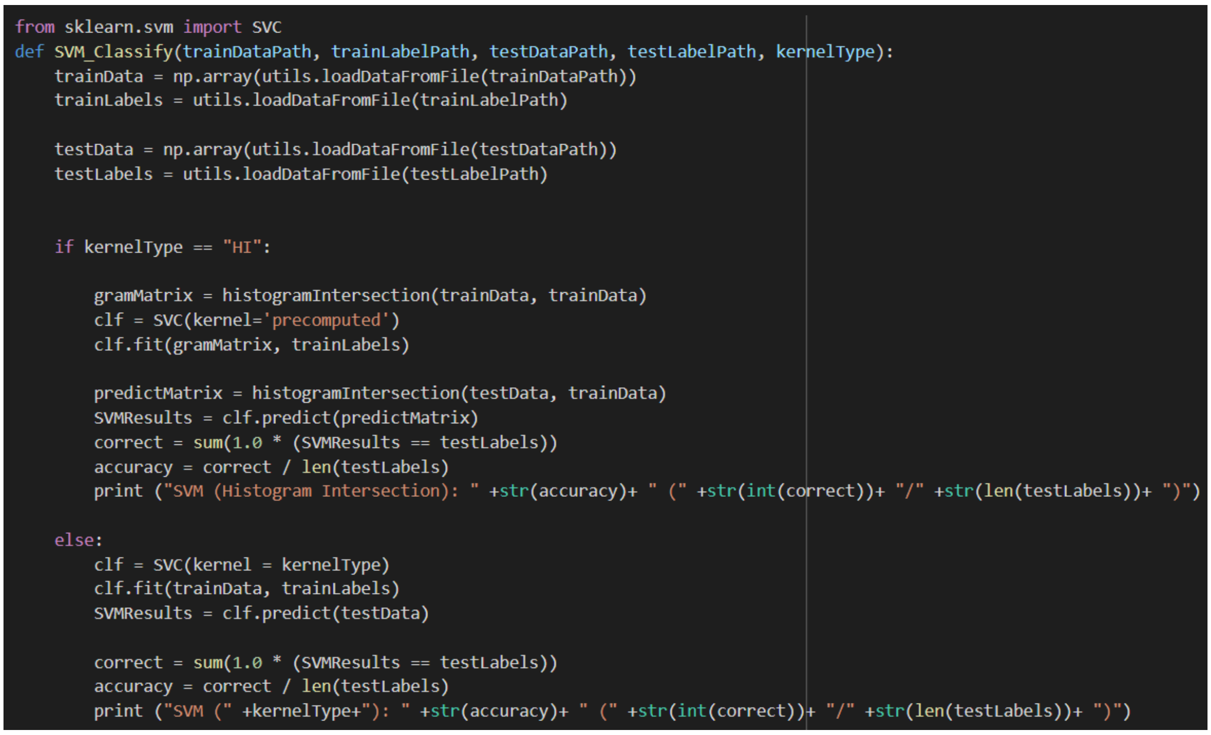
```
from PIL import Image
import os
def process_image_dsift(imagename,resultname,size=20,steps=10,force_orientation=False,resize=None):

    im = Image.open(imagename).convert('L')
    if resize!=None:
        im = im.resize(resize)
    m,n = im.size

    if imagename[-3:] != 'pgm':
        #create a pgm file
        im.save('tmp.pgm')
        imagename = 'tmp.pgm'

    # create frames and save to temporary file
    scale = size/3.0
    x,y = np.meshgrid(range(steps,m,steps),range(steps,n,steps))
    xx,yy = x.flatten(),y.flatten()
    frame = np.array([xx,yy,scale * np.ones(xx.shape[0]), np.zeros(xx.shape[0])])
    np.savetxt('tmp.frame',frame.T,fmt='%03.3f')

    if force_orientation:
        cmmd = str("sift "+imagename+" --output="+resultname+
                    " --read-frames=tmp.frame --orientations")
    else:
        cmmd = str("sift "+imagename+" --output="+resultname+
                    " --read-frames=tmp.frame")

    os.environ['PATH'] += os.pathsep +'/home/maxpoon/Image-Recognition/vlfeat-0.9.20/bin/glnxa64'
    os.system(cmmd)
```

4. Histogram Intersection:



```
def histogramIntersection(M, N):
    m = M.shape[0]
    n = N.shape[0]

    result = np.zeros((m,n))
    for i in range(m):
        for j in range(n):
            temp = np.sum(np.minimum(M[i], N[j]))
            result[i][j] = temp

    return result
```

5. Classification:



```
from sklearn.svm import SVC
def SVM_Classify(trainDataPath, trainLabelPath, testDataPath, testLabelPath, kernelType):
    trainData = np.array(utils.loadDataFromFile(trainDataPath))
    trainLabels = utils.loadDataFromFile(trainLabelPath)

    testData = np.array(utils.loadDataFromFile(testDataPath))
    testLabels = utils.loadDataFromFile(testLabelPath)


    if kernelType == "HI":

        gramMatrix = histogramIntersection(trainData, trainData)
        clf = SVC(kernel='precomputed')
        clf.fit(gramMatrix, trainLabels)

        predictMatrix = histogramIntersection(testData, trainData)
        SVMResults = clf.predict(predictMatrix)
        correct = sum(1.0 * (SVMResults == testLabels))
        accuracy = correct / len(testLabels)
        print ("SVM (Histogram Intersection): " +str(accuracy)+ " (" +str(int(correct))+ "/" +str(len(testLabels))+ ")")

    else:
        clf = SVC(kernel = kernelType)
        clf.fit(trainData, trainLabels)
        SVMResults = clf.predict(testData)

        correct = sum(1.0 * (SVMResults == testLabels))
        accuracy = correct / len(testLabels)
        print ("SVM (" +kernelType+"): " +str(accuracy)+ " (" +str(int(correct))+ "/" +str(len(testLabels))+ ")")
```

Since this code is an adaptation, it can be run with the Caltech-101 data from the reference repository directly.

The source code can be found in the following references:

[1] PG_BOW_DEMO - Image Classification using Bag of Words and Spatial Pyramid BoW

[2] Image-Recognition - Recognize images using Spatial Pyramid Matching & Earth Mover's Distance

[3] Spatial Pyramid Matching Scene Recognition

*************The End**************