Report

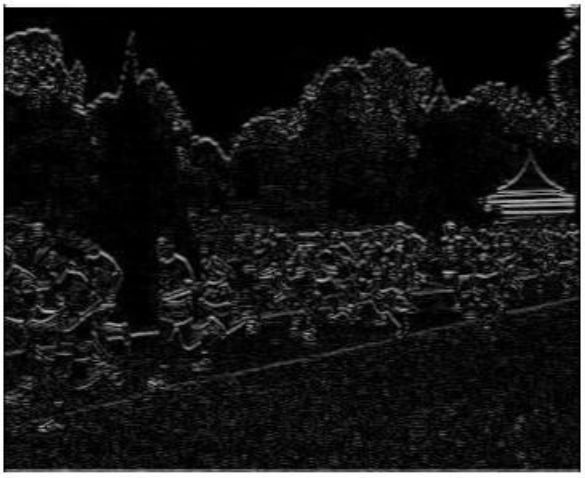
3.1 Edge Detection

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

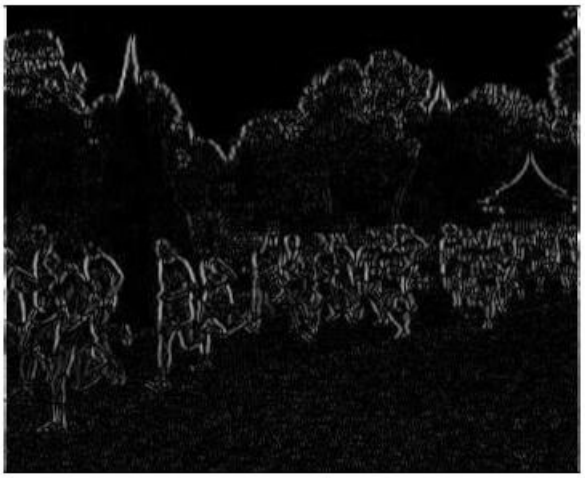
```
P = imread('resources/maccropped.jpg');
P = rgb2gray(P);
imshow(P)
```

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?

```
h1 = [-1 -2 -1; 0 0 0; 1 2 1]
h2 = [-1 0 1; -2 0 2; -1 0 1]
P2 = conv2(double(P),double(h1));
P2 = abs(P2);
P2 = P2*255/996;
imshow(uint8(P2))
```



```
P3 = conv2(double(P),double(h2));
P3 = abs(P3);
P3=P3*255/993;
imshow(uint8(P3))
```



The edges like diagonal disappear from the edge images.

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.

```
P2 = conv2(double(P),double(h1));
P3 = conv2(double(P),double(h2));
```

```
P4 = P2.^2+P3.^2;
P4 = P4*255/1019592;
imshow(uint8(P4))
```

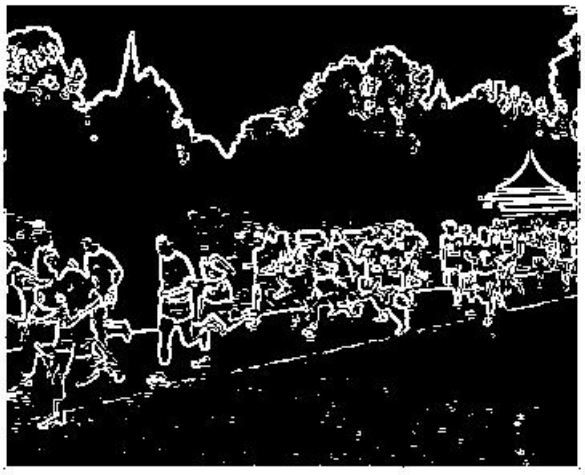
It is to calculate the magnitute of the gradient in its direction. That is why squaring operation is carried out.

d) Threshold the edge image E at value t

```
P5 = P4>50;
```

```
P5 = P4>20;
```

```
P5 = P4>10;
```



What are the advantages and disadvantages of using different thresholds?

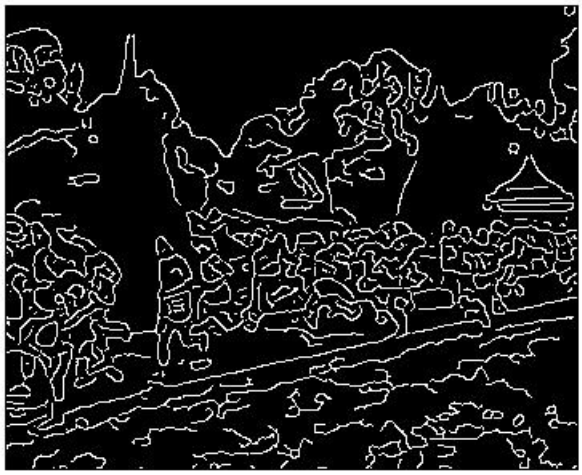
Using a high thresholds, noise could be removed, but some edges are removed at the same time. Using a lower threshold would keep the edges but there would be more noise pixels.

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

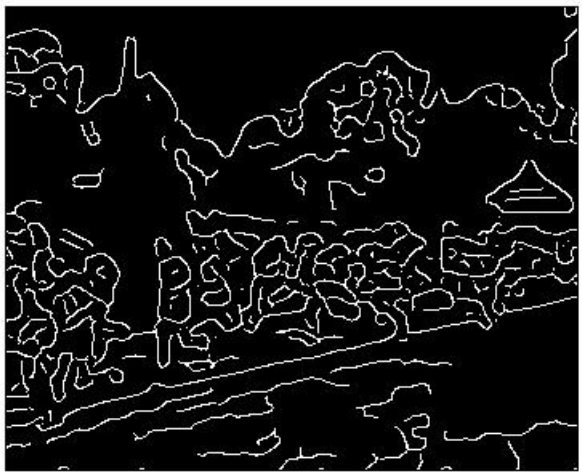
(i) Try different values of sigma ranging from 1.0 to 5.0 and determine the effect on the edge images. What do

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.

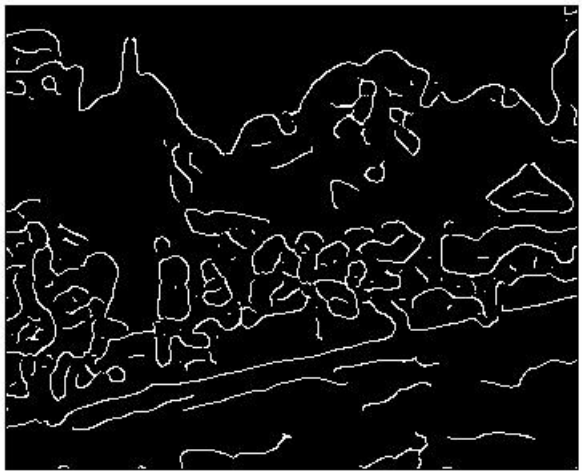
sigma=2.0



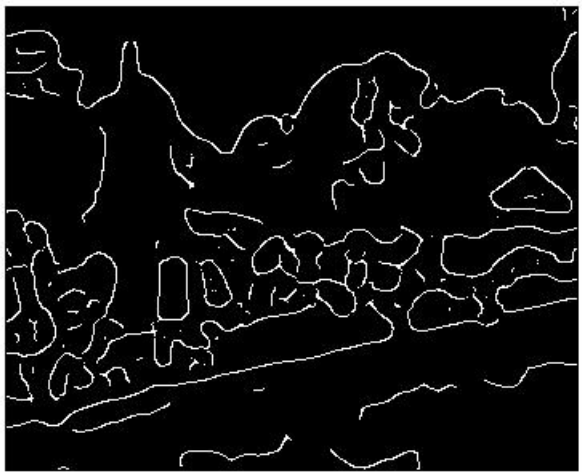
sigma=3.0



sigma=4.0



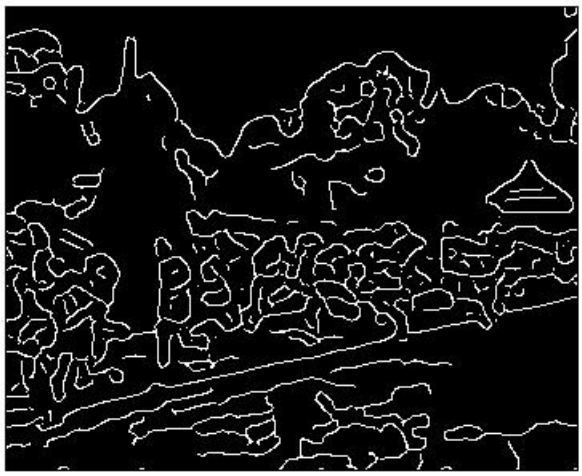
sigma=5.0



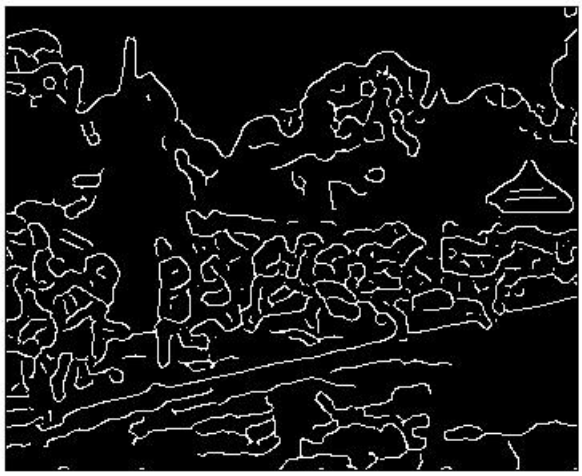
With a larger sigma value, the Canny edge detection is able to remove the noise, but the location accuracy is lower.

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

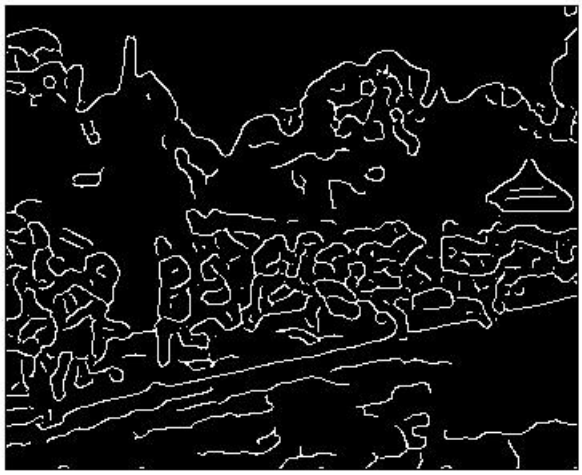
tl = 0.01, sigma = 3



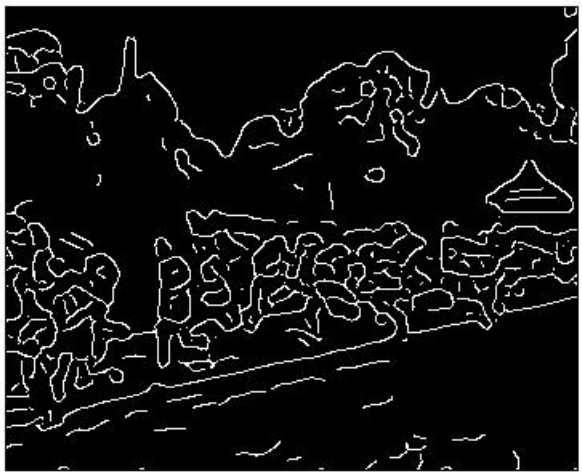
tl = 0.02, sigma = 3



tl = 0.04, sigma = 3



tl = 0.08, sigma = 3



When tl has a lower value, more pixel will be set to one. So among the above images, the first one has more complete edges and more noise.

3.2 Line Finding using Hough Transform

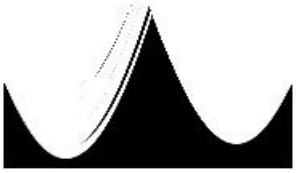
a) Reuse the edge image computed via the Canny algorithm with sigma=1.0.

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?

```
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.
```

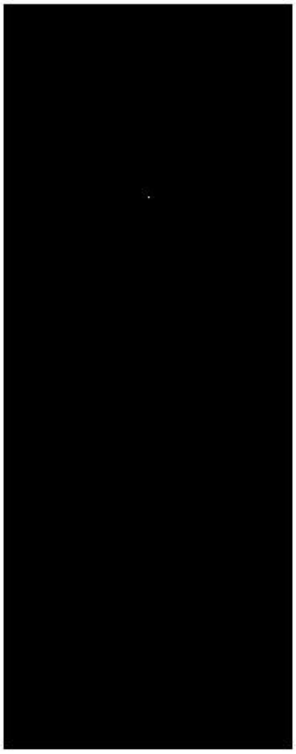
Using Radon transform, the points on the same straight line will be mapped to the same point in the theta domain, so the point in theta domain will be higher. So Radon transform is equivalent to Hough transform. These two method may be different when the image is not in binary format (i.e. gray-level image or RGB image), because the intensity will be taken into consideration in Radon transform.

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.



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.

```
H1 = H>=143.5;
imshow(H1)
```



```
[radius, theta]=find(H1>0);
radius = xp(radius)
```

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.

We have: cos(theta * pi/180) * x + sin(theta * pi/180) * y = radius

```
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.
```

When we call the function

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

A = radius * cos(theta * pi/180), B = radius * sin(theta*pi/180), which are corresponding to cos(theta * pi/180) * x + sin(theta * pi/180) * y = radius.

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.

C = radius^2 (in Hough transform coordinates)

A*(x-179)* + *B*(y-145) = radius^2

A*x*+*B*y = radius^2 + 179A + 145B

```
C = radius^2 + 179*A+145*B
```

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.

```
yl = (C-A*xl)/B
yr = (C-A*xr)/B
```

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

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

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

It does not match up with any edge of the running path. One of the possible sources of errors is that the value we chose for sigma is too small and it is not enough to remove the noise. I try to get the result with sigma=3. And this time, the line match up with the lower edge of running path.

3.3 3D Stereo

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

```
function map = disparity_map(Pl, Pr)
[height, width] = size(Pl);
map = ones(height-10, width-10);
for row = 6:height-5
    for xl = 6:width-5
        T = Pl(row-5:row+5,xl-5:xl+5);
        left = xl-14;
        right = xl;
        if left<6
            left = 6;
        end
        ssd_min = Inf;
        xr_min = left;
        for xr = left:right
            I = Pr(row-5:row+5,xr-5:xr+5);
            I_flipped = rot90(I,2);
            ssd_1 = ifft2(fft2(I).*fft2(I_flipped));
            ssd_1 = ssd_1(11,11);
            ssd_2 = ifft2(fft2(T).*fft2(I_flipped));
            ssd_2 = ssd_2(11,11)*2;
            ssd = ssd_1 - ssd_2;
            if ssd<ssd_min
                ssd_min=ssd;
                xr_min = xr;
            end
        end
```

```
        d = xl - xr_min;
        map(row-5, xl-5) = d;
    end
end
```

c) Run your algorithm on the two images to obtain a disparity map D, and see the results via imshow(-D,[-15 15]);

The output image is similar to ‘corridor_disp.jpg’. And the quality of the disparity is acceptable. The bright region at the center is supposed to be in dark color. The possible cause is that there is a large region having the same color at this location in the original image, so that the SSD values are the same regardless of the value of xr.

The quality of the image is good because there are not many large area having the same or similor colors, so that the SSD matching is working well for most of the region.

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?

The output image is similar to triclops-id.jpg. But the quality is not very good compared to the output for corridor image. This is because there are a lot of large areas in the image having the same color. This adversely affects the performance of SSD matching.

3.4 OPTIONAL

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.

Refering to some open source codes for SPM, I implement the algorithm in the paper in python.

Result

At first, I implemented the algorithm with four categories (Faces, laptop, soccer_ball and starfish) of images from Caltech-101 dataset. And I select 30 training images and 30 testing images for each category. The result is:

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

We can see that two-level SPM outperforms the BoW approach. But both classification rates are quite high. The reason might be that the number of categories is too small (only 4 categories). So I add four more categories (Leopards, brains, lamp and coagar_face) and re-run the classification program. This time the result is:

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

We can see that the classification rates are very close to the experiment results reported in

“Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”.

Key functions of the source codes

The codes repository is on my github: https://github.com/MaxPoon/Image-Recognition

Main script:

```
def main():

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

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

if __name__ == "__main__":

    main()
```

Build histogram:

```
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] + histogramOfLev
elTwo[4] + histogramOfLevelTwo[5]
        histogramOfLevelOne[1] = histogramOfLevelTwo[2] + histogramOfLevelTwo[3] + histogramOfLev
elTwo[6] + histogramOfLevelTwo[7]
        histogramOfLevelOne[2] = histogramOfLevelTwo[8] + histogramOfLevelTwo[9] + histogramOfLev
elTwo[12] + histogramOfLevelTwo[13]
        histogramOfLevelOne[3] = histogramOfLevelTwo[10] + histogramOfLevelTwo[11] + histogramOfL
evelTwo[14] + histogramOfLevelTwo[15]

        # level 0
        histogramOfLevelZero = histogramOfLevelOne[0] + histogramOfLevelOne[1] + histogramOfLevel
One[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
```

Sift:

```
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)
```

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
```

Classify:

```
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(tes
tLabels))+ ")"
```