**Assignment 5 Solution  
Computer Vision (CS-559)**

**Name: Dhaval Harish Sharma  
Red ID: 824654344**

1. **[12%] Suppose that the boundary of a closed region is represented by a 4-directional chain code. Write a function Area(ChainCode) in pseudo-code to compute the area of the region from its chain code representation.**

**Solution:**The pseudo-code for computing the area of the region from its chain code representation can be given as follows:def **Area(ChainCode)**:

*# Initializing the attributes*

x, y = 0, 0

area = 0

coord = []

*# Loop for finding the coordinates from the chain code*

for direction in ChainCode:

if direction == 0:

y += 1

elif direction == 1:

x -= 1

elif direction == 3:

x += 1

else:

y -= 1

if [x, y] not in coord:

coord.append([x, y])

*# Finding the area of the region from the coordinates*

for i in range(len(coord) - 1):

area += coord[i + 1][0] \* coord[i][1] - coord[i][0] \* coord[i + 1][1]

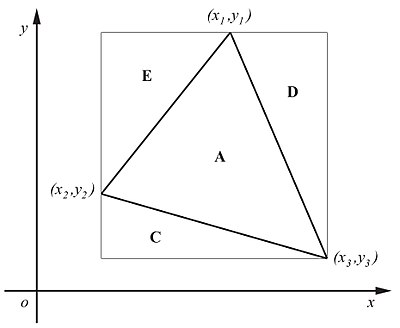
area = area // 2

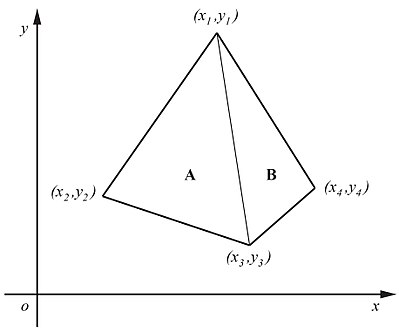
return area

The function takes the chain code as its parameter and returns the area of the region encoded within the chain code representation. Firstly, we find the coordinates from the chain code. Then, these coordinates can be input to the formula of the area enclosed by polyline which gives the area enclosed by the chain code. There is a Question1.py file attached to this document containing a full working code.

1. **[12%] Show that the area enclosed by the polyline,   
   (x0, y0), (x1, y1), …, (xn-1, yn-1), (x0, y0) is given by the**

**Solution:**

Proof for a Triangle:  
Let A be the area of the triangle whose vertices are given by the coordinates of the polyline (x1, y1), (x2, y2) and (x3, y3). Draw the minimum area rectangle around the triangle so its sides are parallel to the x or y axes. At least one vertex of the triangle will be on a corner of the rectangle. In the figure, the areas of the three surrounding triangles are C, D and E. Obviously, A is equal to the area of the rectangle (call it R) minus the areas of the other three triangles. The equation describing this relationship is,  
**A = R – C – D – E**  
  
By inspection of the figure it can be seen that the areas are given by,  
R = (x[3] – x[2]) (y[1] – y[3]) = (x[3]y[1] + x[2]y[3]) – (x[3]y[3] + x[2]y[1])  
-C = -(1/2) [(x[3] – x[2])(y[2] – y[3])] = (1/2)[-x[3]y[2] – x[2]y[3]] + (1/2)[x[3]y[3] + x[2]y[2]]  
-D = -(1/2) [(x[3] – x[1])(y[1] – y[3])] = (1/2)[-x[3]y[1] – x[1]y[3]] + (1/2)[x[3]y[3] + x[1]y[1]]

-E = -(1/2) [(x[1] – x[2])(y[1] – y[2])] = (1/2)[-x[1]y[1] -x[2]y[2]] + (1/2)[x[1]y[2] + x[2]y[1]]  
  
Collecting terms and rearranging yields,  
A = (1/2) [(x[2]y[3] – x[3]y[2]) – (x[1]y[3] – x[3]y[1]) + (x[1]y[2] – x[2]y[1])]  
  
Rearranging the another way,  
A = -(1/2)[x[1]y[2] + x[2]y[3] + x[3]y[1] – x[2]y[1] – x[3]y[2] – x[1]y[3]]  
  
This formula can be extended to find the area of any polygon since a simple polygon can be divided in to triangles.  
  
Proof for a quadrilateral and general Polygon:  
Finding the area of a quadrilateral demonstrates how the formula is generalized to any polygon by dividing the polygon into triangles. Consider the figure of a quadrilateral whose coordinates are labeled in counterclockwise order. The quadrilateral is divided into two triangles with areas A and B. Using the triangle formula on each triangle we get,  
A = (1/2)[x[1]y[2] + x[2]y[3] + x[3]y[1] – x[2]y[1] – x[3]y[2] – x[1]y[3]]  
B = (1/2)[x[1]y[3] + x[3]y[4] + x[4]y[1] – x[3]y[1] – x[4]y[3] – x[1]y[4]]  
  
  
  
Since both triangles were traced in a counterclockwise direction, both areas are positive and we get the area of the quadrilateral by adding the two areas. The last positive term and the last negative term of A cancel with the first positive term and the first negative term of B giving,  
Total Area = (1/2)[x[1]y[2] + x[2]y[3] + x[3]y[4] + x[4]y[1] – x[2]y[1] – x[3]y[2] – x[4]y[3] – x[1]y[4]]  
  
By the above proofs, we can say that for any area covered by a polyline, we can convert its area to a sum of different triangles as follows,

1. **[13%] Compare Hough transform and Canny edge detection for region detection in terms of (i) robustness (insensitivity) to noise, (ii) detection of regions with irregular shape, (iii) any common technique that is used both methods.  
   Solution:**The Canny edge detector is one of the most commonly used image processing tools, detecting edges in a very robust manner. It is a multi-step process, which can be implemented as a sequence of filters. The result can be used to detect regions in an image. The following steps are followed in the process of Canny edge detection:  
   0) Convert to Grayscale 1) Noise Reduction 2) Compute Gradient Magnitude and Angle 3) Non-maximum Suppression 4) Hysteresis Thresholding.  
     
   Hough Transform allows finding regions that can be described by mathematical relationships, such as lines, circles and ellipses. Consider a set of points on a line in x-y plane (image space) described by y = mx + b. The above line can be described in the parameter space as b = -xm + y. The above representation has some problems such as the parameters are bounded, vertical lines are infinite, etc. The alternative representation is polar coordinates: xcosƟ + ysinƟ = ƿ.  
     
   While traditional Canny edge detection provides relatively simple but precise methodology for edge detection problem, with more demanding requirements on the accuracy and robustness on the detection, the traditional algorithm can no longer handle the challenging edge detection task. The main defects of the traditional algorithm can be summarized as follows:  
   1) A Gaussian filter is applied to smooth out the noise, but it will also smooth the edge, which is considered as the high frequency feature. This will increase the possibility of missing weak edges, and the appearance of isolated edges in the result. 2) For the gradient amplitude calculation, the old Canny edge detection algorithm uses the center in a small 2×2 neighborhood window to calculate the finite difference mean value to represent the gradient amplitude. This method is sensitive to noise and can easily detect false edges and lose real edges. 3) In the traditional Canny edge detection algorithm, there will be two fixed global threshold values to filter out the false edges. However, as the image gets complex, different local areas will need very different threshold values to accurately find the real edges. In addition, the global threshold values are determined manually through experiments in the traditional method, which leads to complexity of calculation when many different images need to be dealt with. 4) The result of the traditional detection cannot reach a satisfactory high accuracy of single response for each edge - multi-point responses will appear.  
     
   The Hough transform is only efficient if a high number of votes fall in the right bin, so that the bin can be easily detected amid the background noise. This means that the bin must not be too small, or else some votes will fall in the neighboring bins, thus reducing the visibility of the main bin.   
   Also, when the number of parameters is large (that is, when we are using the Hough transform with typically more than three parameters), the average number of votes cast in a single bin is very low, and those bins corresponding to a real figure in the image do not necessarily appear to have a much higher number of votes than their neighbors. The complexity increases at a rate of **O(Am-2)** with each additional parameter, where A is the size of the image space and m is the number of parameters. Thus, the Hough transform must be used with great care to detect anything other than lines or circles.  
   Finally, much of the efficiency of the Hough transform is dependent on the quality of the input data: the edges must be detected well for the Hough transform to be efficient. Use of the Hough transform on noisy images is a very delicate matter and generally, a denoising stage must be used before.  
     
   As seen above, the Canny edge detector is way more robust than the Hough transform in comparison to detecting irregular shapes because it doesn’t depend on any kind of formula to detect shapes. The Hough transform needs a mathematical formula for the kind of objects or features in order to work properly whereas the Canny edge detector is independent of the same. If we want to find the true edges of the buildings, a canny edge detector cannot recover information very well, however, the Hough transform can detect some of the straight lines representing building edges even within obstructed region. Although the version of the Hough transform described above applies only to finding straight lines, a similar transform can be used for finding any shape which can be represented by a set of parameters. A circle, for instance, can be transformed into a set of three parameters, representing its center and radius, so that the Hough space becomes three dimensional. Arbitrary ellipses and curves can also be found this way, as can any shape easily have expressed as a set of parameters.  
     
   There are several common techniques in both the methods. The first is the noise reduction. Both methods use noise reduction techniques before performing computations. The Canny edge detector uses Gaussian filter whereas the Hough transform uses edge detection and thresholding. The second similarity is the gradient. Both the methods use gradient direction Ɵ in order to get the direction of the gradient.
2. **[13%] (a) Compare three losses compressions techniques in terms of their suitability for natural images.**

**(b) Explain which lossless compression technique use variable code length. Which technique results in the optimal code length?  
Solution:**There are several lossless compression techniques available for compressing a natural image without losing the image information. The comparison between Delta or Differential Coding, RLE and Huffman coding can be given as follows:  
Delta or Differential Coding is to record the values of the first pixel in a row, and the difference between graylevel (GL) of each pixel and the previous pixel in that row. The difference can be coded using fewer bits. The RLE exploits high interpixel redundancy in simple images. Provides the count of the number of pixels with the same GL value. It is most effective for binary images. For non-binary images, first bit plane slicing is applied and then for each bit plane the RLE is performed. Typical compression ratios are 0.5 to 1.2 so it is not a good compression method for complex images – it is only used for simple images such as graphic files. Alternatively, preprocessing can be applied to reduce GLvalues, but this amounts to lossy compression. Another method to extend RLE to GL images is to use two parameters to characterize a run (GL, RL) where GL is graylevel and RL is the run length. In Huffman Coding, the codewords are chosen in such a way that aveL is as close as possible to E. This code is minimum length, though variable length. Typical reduction is 10% to 50%, i.e. 1.1:1 to 1.5:1. The first step in Huffman Coding is to compute the normalized histogram and order the h(i) from highest to lowest. Then combine the two smallest and then again order h(i) from highest to lowest. Continue this process until two probabilities are left. Work backward along a Huffman tree to generate the code alternatively by alternatively assigning 0 and 1. The comparison ratio is 2.0:1.9, providing 5% compression.   
The compression ratios are dependent on the type of image, i.e. synthetic or natural. Typical ratios are given below,

|  |  |  |  |
| --- | --- | --- | --- |
| **Image** | **Delta** | **RLE** | **Huffman** |
| **Synthetic** | 1.95 | 60 | 6 |
| **Natural** | 1.8 | 1.1 | 1.6 |

Generally, the compression ratios are higher with synthetic images since there are much coding and interpixel redundancies in these types of images. In particular, RLE is most suitable for synthesis images since there are few graylevels and long runs of the same graylevel.

The Huffman Coding is a lossless compression technique that uses variable code length. In this coding the codewords are chosen in such a way that Lave is as close as possible to E. This code is minimum length, though variable length. Typical reduction is 10% to 50%, i.e. 1.1:1 to 1.5:1.

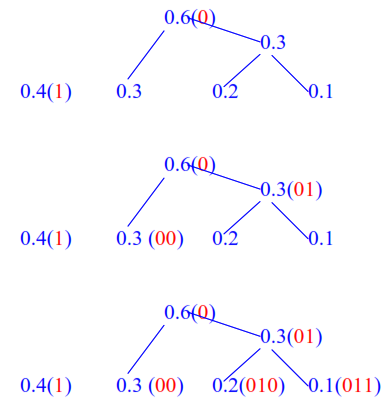
Algorithm

1. Compute normalized histogram

2. Order h(i) from highest to lowest

3. Combine two smallest

4. Go to Step 2 until two probabilities are left

5. Work backward along a Huffman tree to generate the code by alternatively assigning 0 and 1.  
  
Example:  
Let h(0) = 0.2, h(1) = 0.3, h(2)= 0.1, h(3) = 0.4. Order as 0.4, 0.3, 0.2, 0.1. Huffman tree,  


Note that two GL have 3 bits assigned to them, two with 2 bits and one GL has just 1 bit. The latter is the pixel most likely to occur (0.4 or 40% of the time).   
This is the average length with Huffman code. The compression ratio is 2.0:1.9, providing 5% compression. Thus, Huffman coding results in the optimal code length.

1. **[20%] Consider the 8 by 8 subimage,  
     
   Apply the JPEG compression algorithm and find the 1-D coefficient sequence.  
   Solution:**The Joint Photographic Experts Group (JPEG), has specified an algorithm based on a frequency domain transformation, the so called discrete cosine transform (DCT). This transform is similar to Fourier transform but contains only real data. The program for finding the 1-D coefficient sequence is as follows:  
   # Importing the necessary libraries

import numpy as np

import cv2

# Initializing the input 8X8 Image array

a = np.array([[56, 45, 51, 66, 70, 61, 64, 73],

[63, 59, 56, 90, 109, 85, 69, 72],

[62, 59, 68, 103, 144, 104, 66, 73],

[63, 58, 71, 132, 134, 106, 70, 69],

[65, 61, 68, 114, 116, 82, 68, 70],

[79, 65, 60, 67, 77, 68, 58, 75],

[85, 71, 54, 59, 55, 61, 65, 73],

[87, 79, 69, 58, 65, 66, 78, 94]])

# Shifting pixels

for i in range(8):

for j in range(8):

a[i][j] = a[i][j] - 128

# Computing the descrete cosine transform of the input image

dct\_a = np.zeros(shape = (8, 8))

cv2.dct(a / 255, dct\_a)

dct\_a = (dct\_a \* 255).astype('int32')

# Initializing the Quantization matrix

q = np.array([[16, 11, 10, 16, 24, 40, 51, 61],

[12, 12, 14, 19, 26, 58, 60, 55],

[14, 13, 16, 24, 40, 57, 69, 56],

[14, 17, 22, 29, 51, 87, 80, 62],

[18, 22, 37, 56, 68, 109, 103, 77],

[24, 35, 55, 64, 81, 104, 113, 92],

[49, 64, 78, 87, 103, 121, 120, 101],

[72, 92, 95, 98, 112, 100, 103, 99]])

# Dividing the input array by the Quantization matrix elementwise

divided\_a = np.divide(dct\_a, q).astype('int32')

# Traversing the image in a zigzag manner

result = np.empty([8\*8])

index = -1

bound = 0

for i in range(0, 15):

if i < 8:

bound = 0

else:

bound = i - 8 + 1

for j in range(bound, i - bound + 1):

index += 1

if i % 2 == 1:

result[index] = divided\_a[j][i - j]

else:

result[index] = divided\_a[i - j][j]

# Finding the EOB

result = result.astype('int32')

temp = 0

for i in range(len(result)):

if result[i] != 0:

temp = i + 1

final\_result = []

for i in range(temp):

final\_result.append(result[i])

print(final\_result)

The output for this code is,  
[-26, -2, 0, -3, -2, -5, 1, -4, 0, -3, 0, 0, 4, 0, 2, 0, 0, 0, 1]

There is a Question5.py file attached to this document containing the full working code of the JPEG Compression algorithm.