First Name: Siddharth

Last Name: Thorat

AuE 8200: Machine Perception and Intelligence
Instructor: Dr. Bing Li, Clemson University, Department of Automotive Engineering

.

Question 1) [Sampling/2D-Convolution] Download the image “Lenna.jpg” from the hyperlink. (Lenna or Lena image is a standard test image widely used for image processing since 1973.)

1-1) Convert the image from RGB to gray, using a standard RGB-intensity conversion approach like NTSC, and store the converted image “LennaGray.jpg” as an 8-bit gray image. (2 pts)

1-2) Down-sampling image “LennaGray.jpg” from size 256x256 to 64x64. (3 pts) Perform the down-sampling and visualize your result.

1-3) Implement the convolution (using basic arithmetic operations only, rather than build-in conv()) of Sobel kernel on the “LennaGray.jpg” for edge detection, visualize and comment your detection result. (10 pts)

1-1)

Figure 1(Convert the image from RGB to gray)

1-2)

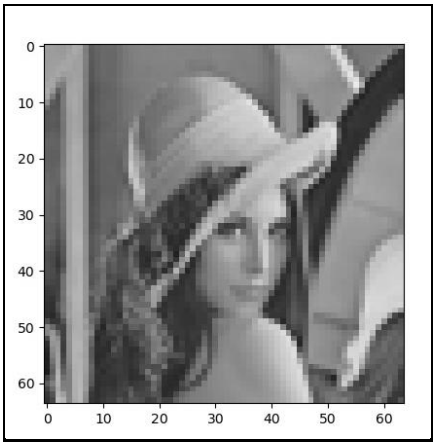


Figure 2(Down-sampling image “LennaGray.jpg” from size 256x256 to 64x64)

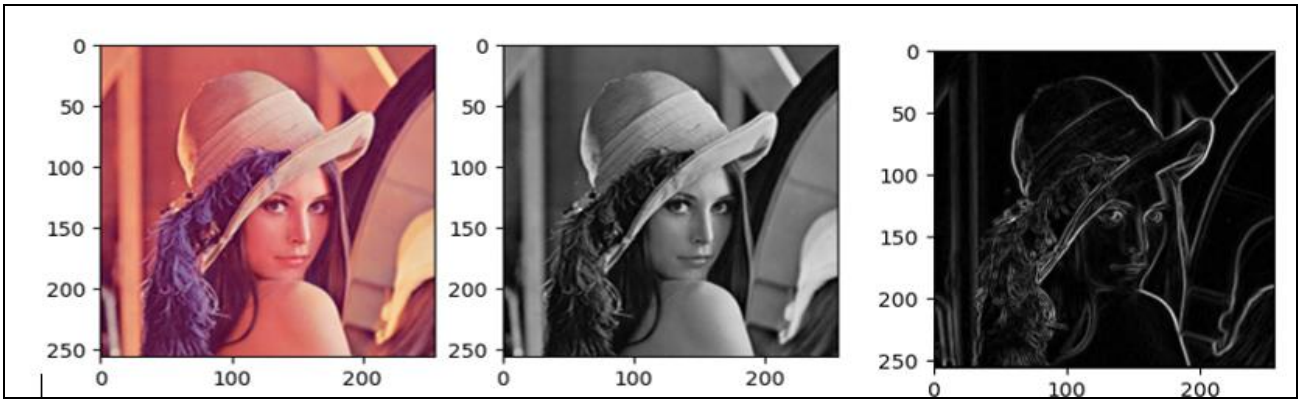


Figure 3

- The above Lenna image is converted into the gray scale image then the gray scale image is converted into Sobel kernel on the “LennaGray.jpg” for edge detection.

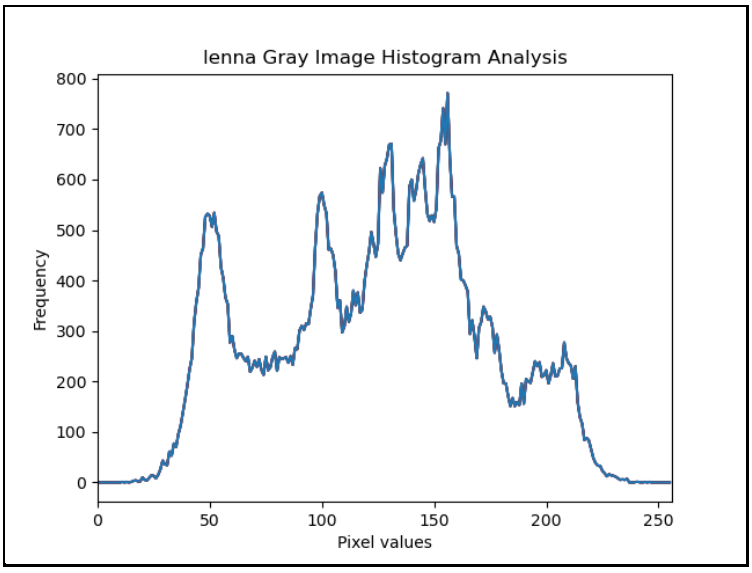
Question 2) [Histogram Equalization] Take the converted gray image “LennaGray.jpg”.

2-1) Perform histogram analysis and visualize histogram distribution.

2-2) Calculate and visualize accumulative histogram distribution.

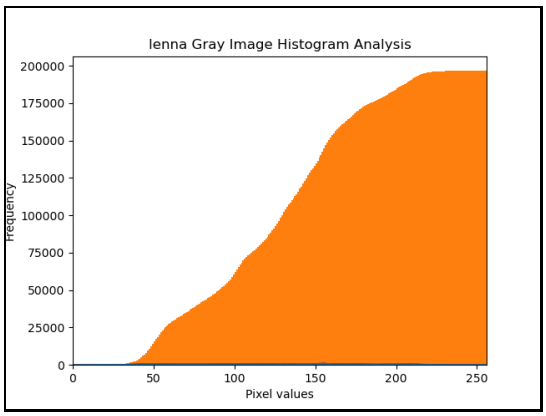
2-3) Implement a function to perform histogram equalization for this image, visualize your histogram-equalized image and its histogram distribution. Comments the difference between the two images before/after histogram equalization.

2-1)



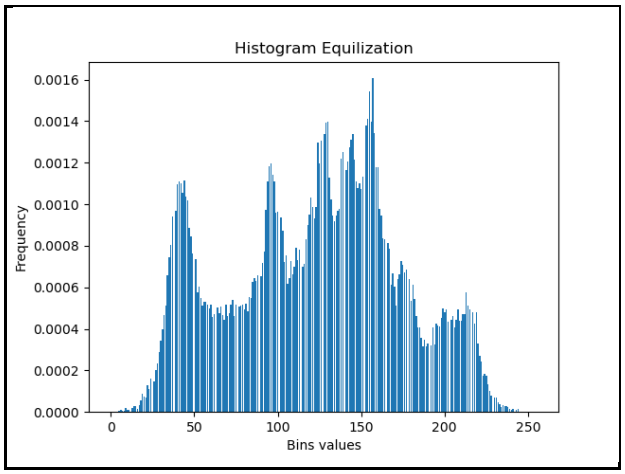
Plot 1

2-2)



Plot 2

2-3)



Plot 3

Figure 4

Question 3) [Line Detection] Download the image “ParkingLot.jpg” from the hyperlink. Note: For this question, you are free to use any 3rd party libraries.

3-1) Apply and visualize histogram analysis, then find a proper threshold to convert the image to a binary image.

3-2) Apply Hough transformation or other line detection approach to detect multiple lines in the image (You select a threshold for the voting matrix). Visualize the lines in the image space (just as: we saw lines there) and in the transformed space (like in Polar space that we introduced in the class) respectively.

3-3) Comment on: will the two lines as two sides of a particular park space be parallel or not, explain why?

3-4) Design and implement the approaches to find all parking space polygons with the four vertex points for each parking space. Describe your approaches and visualize all detected polygons with different colors overlaid on the original image. The TA will check your code.

3-1)

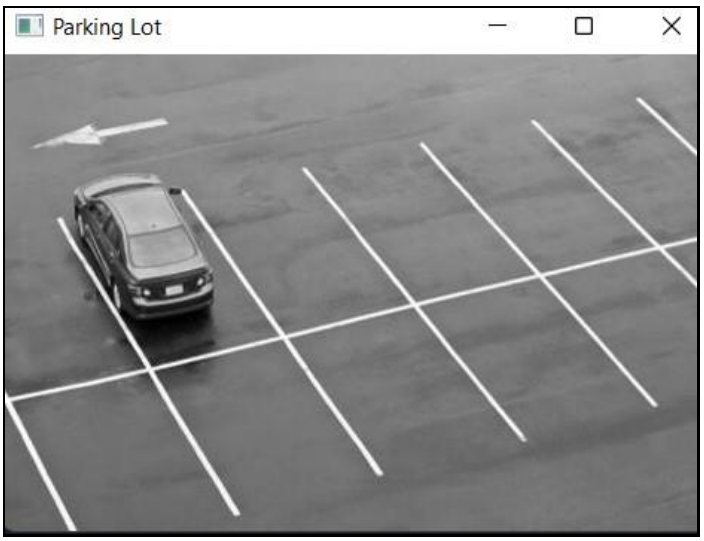


Figure 5

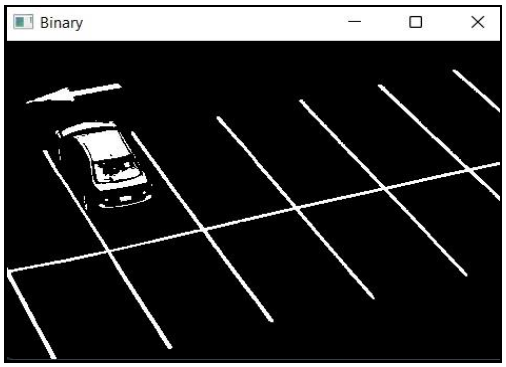
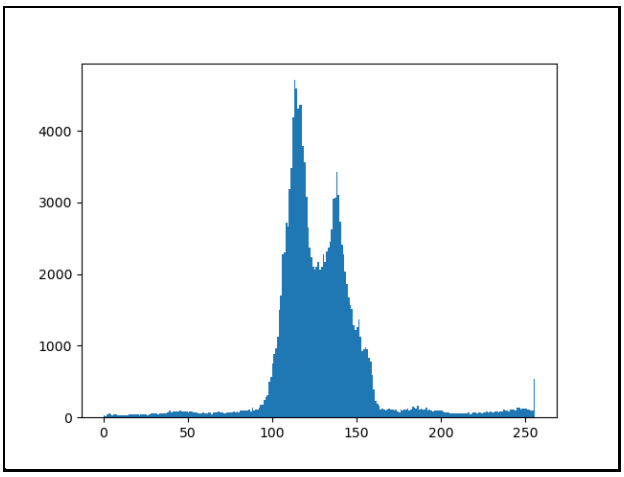


Figure 6



Plot 4

3-2)

Plot 5

```python
#3(2).....

import cv2
import numpy as np
# import math
from matplotlib import pyplot as plt

img = cv2.imread("ParkingLot.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 75, 150)
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 30, maxLineGap=250)

for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (0, 0, 128), 3)

cv2.imshow("linesDetected", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program 1

3-3)

- Yes, the two lines of two sides of a particular park space are parallel. The lines present are the limits of the parking space. Basically, the vehicle should be correspondingly parked between the lines so that the vehicle beside will have equal space to park.

3-4)

```
#3(4).....

import numpy as np
import cv2

# Reading image
img2 = cv2.imread('ParkingLot.jpg', cv2.IMREAD_COLOR)

# Reading same image in another variable and
# converting to gray scale.
img = cv2.imread('ParkingLot.jpg', cv2.IMREAD_GRAYSCALE)

# Converting image to a binary image
# (black and white only image).
_,threshold = cv2.threshold(img, 110, 255,
                            cv2.THRESH_BINARY)

# Detecting shapes in image by selecting region
# with same colors or intensity.
contours,_=cv2.findContours(threshold, cv2.RETR_TREE,
                            cv2.CHAIN_APPROX_SIMPLE)

# Searching through every region selected to
# find the required polygon.
for cnt in contours :
    area = cv2.contourArea(cnt)

    # Shortlisting the regions based on there area.
    if area > 400:
        approx = cv2.approxPolyDP(cnt,
                                  0.009 * cv2.arcLength(cnt, True), True)

        # Checking if the no. of sides of the selected region is 7.
        if(len(approx) == 7):
            cv2.drawContours(img2, [approx], 0, (0, 0, 255), 5)

# Showing the image along with outlined arrow.
cv2.imshow('image2', img2)

# Exiting the window if 'q' is pressed on the keyboard.
if cv2.waitKey(0) & 0xFF == ord('q'):
    cv2.destroyAllWindows()
```

Program 2

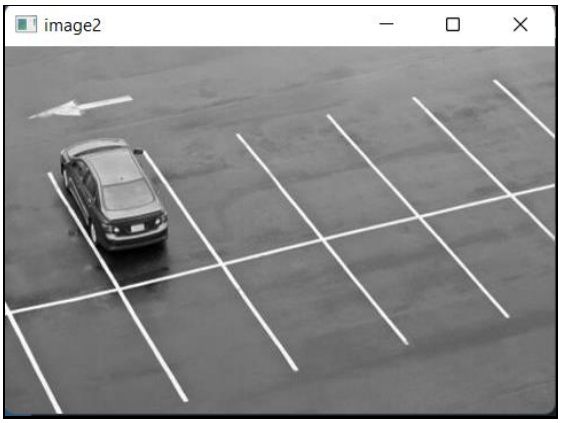


Figure 7