Alexander Gao
awg297
N17435149

Project: Canny Edge Detector

Source code is uploaded as a separate .py (text file) on NYUClasses:

canny-edge-detector-awg297.py

Executable program is uploaded as a separate file on NYUClasses:

canny-edge-detector-awg297-exec.command

Instructions:

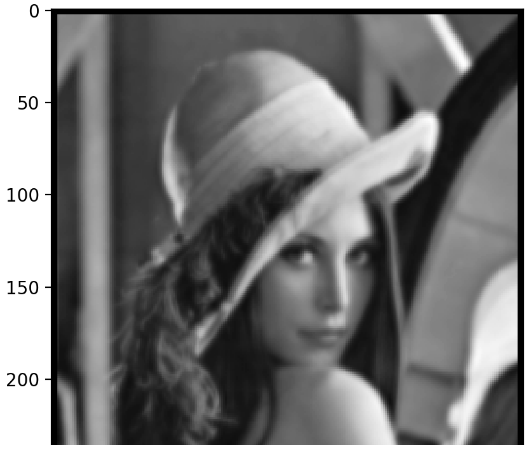
1.) Double-click the executable program, or run the .py program from Terminal.

2.) The user will be prompted to enter the **source path** of their input image.

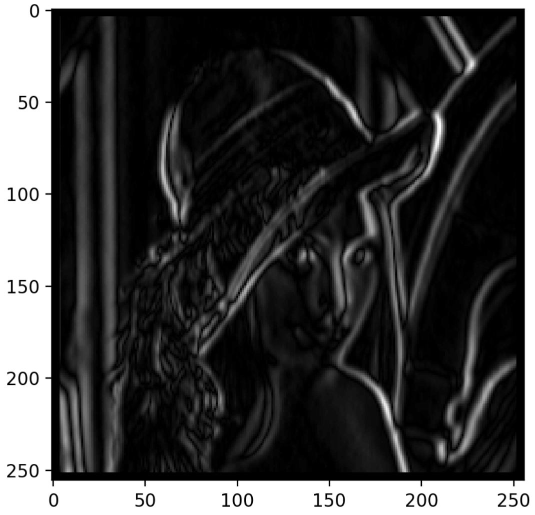
3.) Once they enter the path, the program will automatically run and display all of the requested, processed images in order.

Alexander Gao
awg297
N17435149
Project: Canny Edge Detector

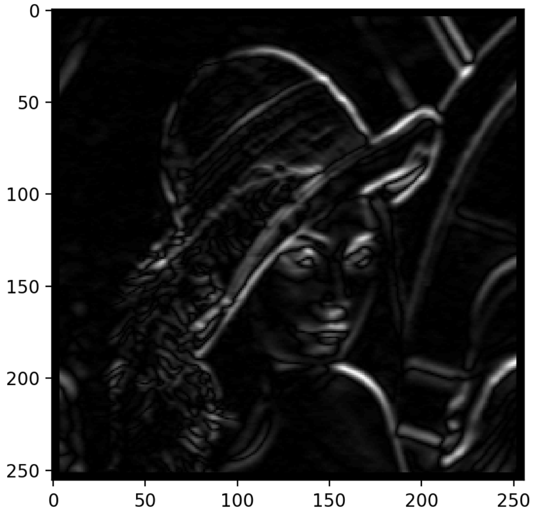
Output Images (Lena256.bmp)



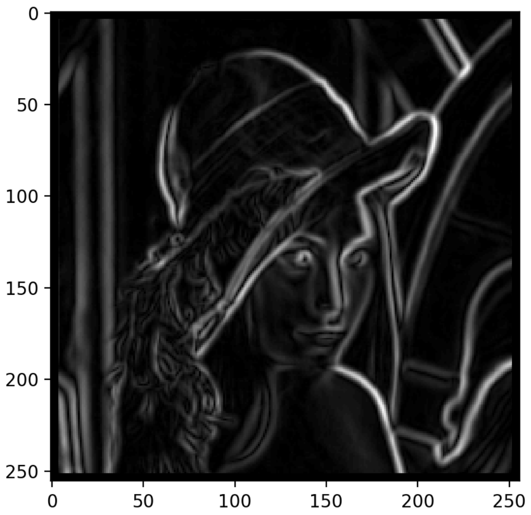
Gaussian



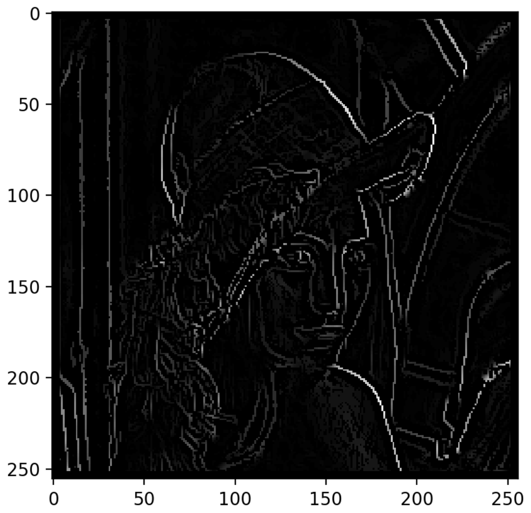
Gradient GX



Gradient GY

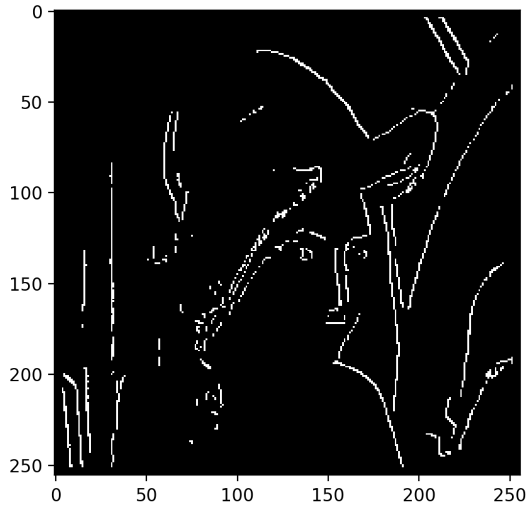


Gradient Magnitude

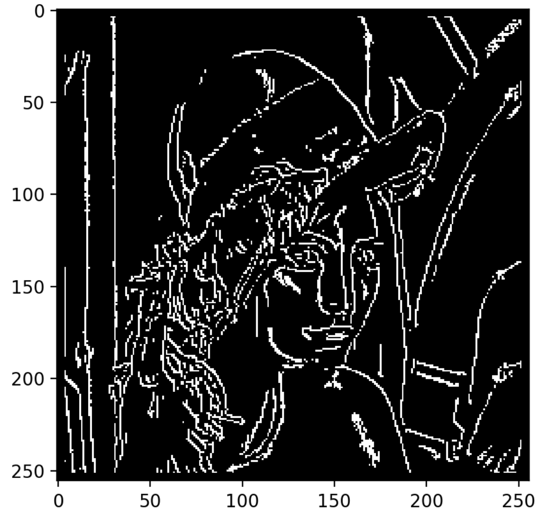


After Non-Maxima Suppression

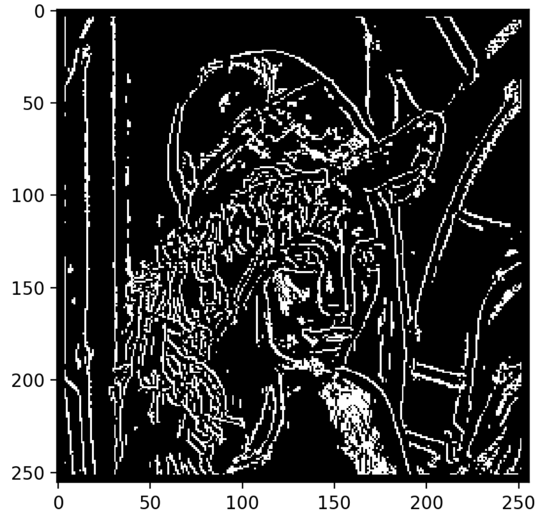
Alexander Gao
awg297
N17435149
Project: Canny Edge Detector



Threshold: 10%
T = 21
Edge Pixels: 2038

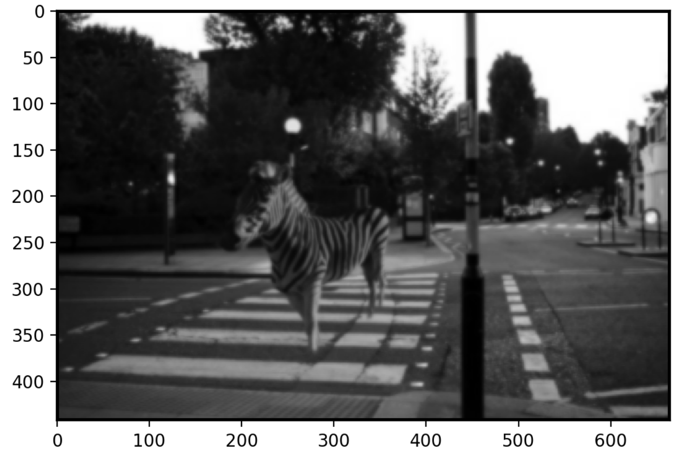


Threshold: 30%
T = 7
Edge Pixels: 6104

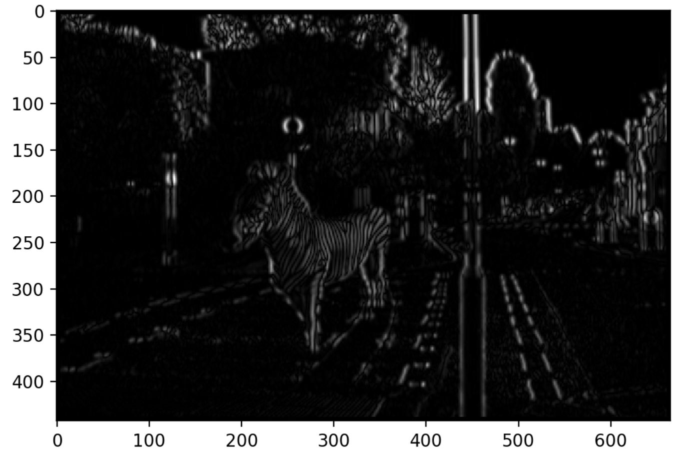


Threshold: 50%
T = 4
Edge Pixels: 9472

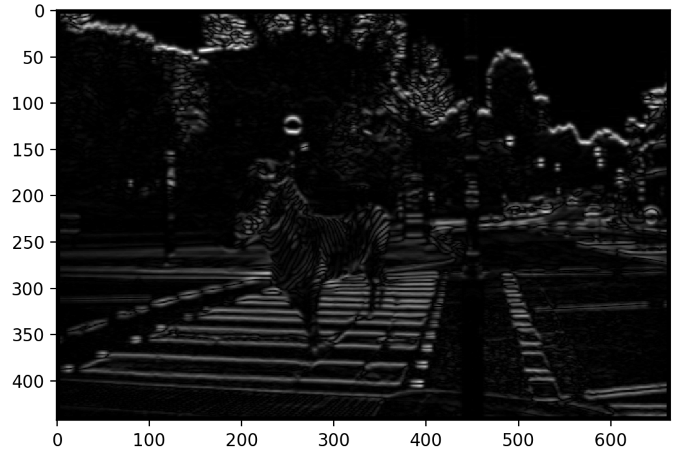
Output Images (zebra-crossing-1.bmp)



Gaussian

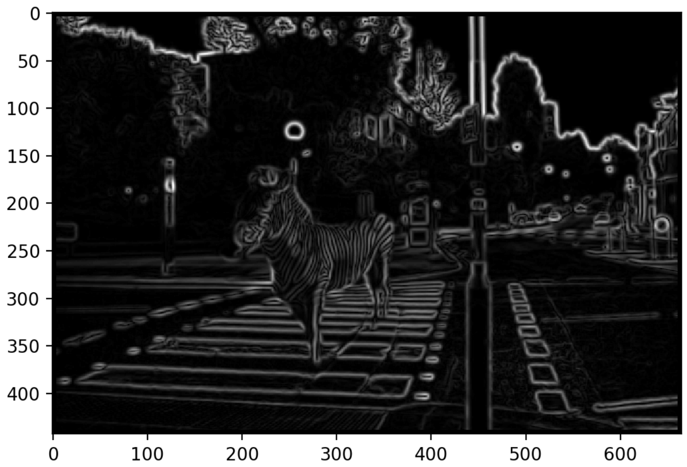


Gradient GX

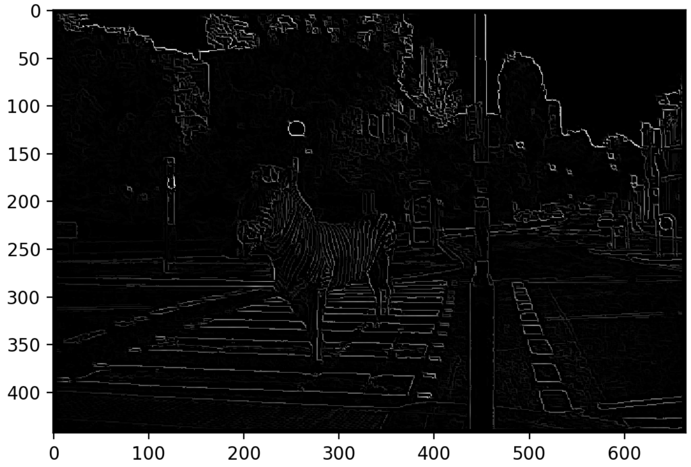


Gradient GY

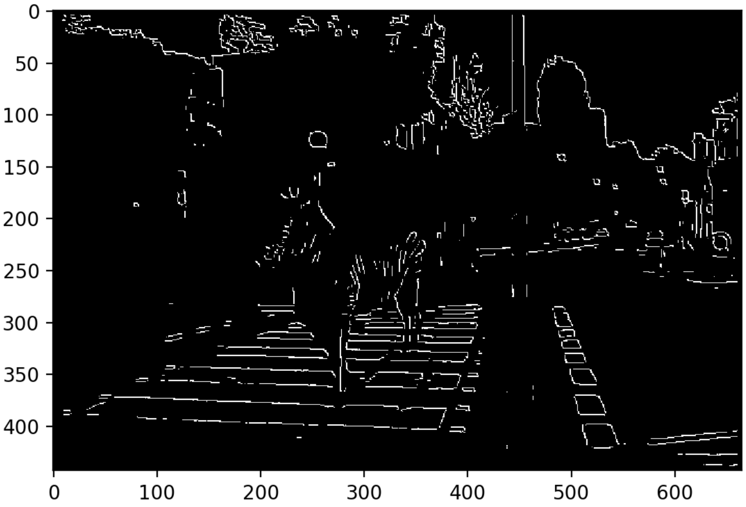
Alexander Gao
awg297
N17435149
Project: Canny Edge Detector



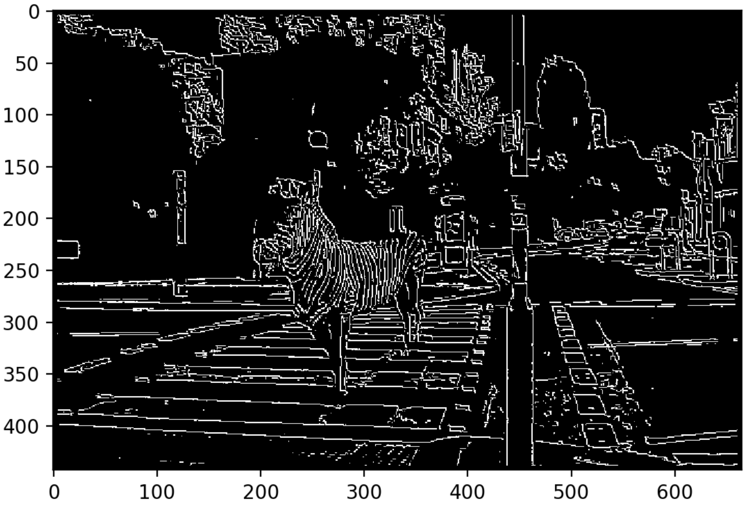
Gradient Magnitude



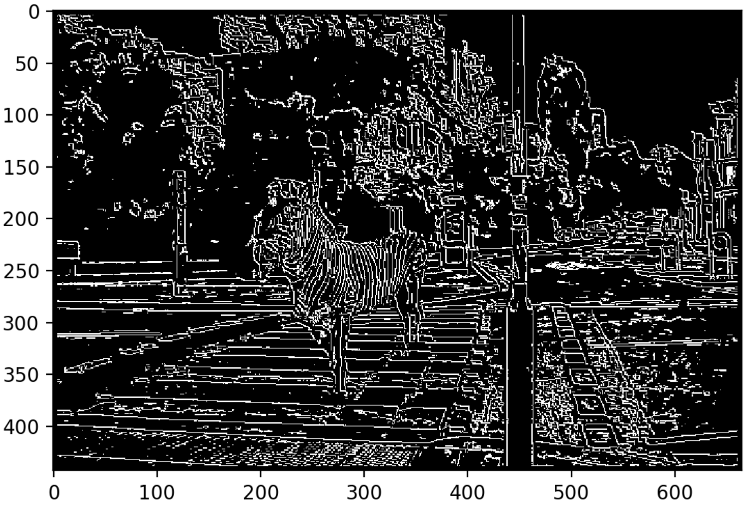
After Non-Maxima Suppression



Threshold: 10%
T = 25
Edge Pixels: 8510



Threshold: 30%
T = 7
Edge Pixels: 25110



Threshold: 50%
T = 3
Edge Pixels: 40922

Alexander Gao
awg297
N17435149
Project: Canny Edge Detector

Source Code

```
##############################
### IMPORT PYTHON MODULES ###
##############################
import math
from PIL import Image
import numpy as numpy
from matplotlib import pyplot as plt

###########################
#results## Import INPUT IMAGE ###     #REPLACE THE PARAMETER OF IMAGE.OPEN WITH THE PATH TO IMAGE THAT YOU WANT TO PROCESS
###########################

source_path = raw_input("Please enter the complete path of your source image (just path, no quotes): ")

try:
            input_image = Image.open(source_path)
except:    #Error case
            print("Error loading image.")
            quit()

####################################################################
### Convert INPUT IMAGE into MATRIX FORMAT using a NUMPY ARRAY ###
####################################################################
indexed_image = numpy.array(input_image)


##########################################
##########################################
#### FUNCTIONS/SUBPROGRAM DEFINITIONS ###
##########################################
##########################################


#################################################################
### CANNY EDGE DETECTOR STEP 1: GAUSSIAN CONVOLUTION FILTER ###
#################################################################
def gaussian(input_image):
            gaussianmask = numpy.array([[1,1,2,2,2,1,1],                                #Store the Gaussian Mask in a MATRIX structure
                                                                                                [1,2,2,4,2,2,1],
                                                                                                [2,2,4,8,4,2,2],
                                                                                                [2,4,8,16,8,4,2],
                                                                                                [2,2,4,8,4,2,2],
                                                                                                [1,2,2,4,2,2,1],
                                                                                                [1,1,2,2,2,1,1]])
            gaussianresult = numpy.zeros(input_image.shape)                             #Create a new matrix "gaussianresult" that will hold the values after gaussian
convolution

            for j in range(3,(input_image.shape[1]-3)):
                        for i in range(3,(input_image.shape[0]-3)):

                                    img_submatrix = input_image[i-3:i+4,j-3:j+4]        #submatrix of size 7x7 that gaussian mask will perform convolution operation
over

                                    convolution_sum = 0                                                                                     #convolution
operation
                                    for k in range(0,7):
                                                for l in range(0,7):
                                                            convolution_sum += img_submatrix[k,l]*gaussianmask[k,l]
                                    convolution = convolution_sum / 140
                                    gaussianresult[i,j] = convolution                                   #store the convolution results in "gaussianresult"

            plt.imshow(gaussianresult, cmap='gray')                                                #display the resulting image after gaussian filter has been applied
            plt.show()

            return gaussianresult                                                                                       #return "gaussianresult"
so it can be passed to next step of canny edge detector: "gradientify"


################################################################################
### CANNY EDGE DETECTOR STEP 2: GRADIENT OPERATION USING PREWITT OPERATOR ###
################################################################################

def gradientify(gaussianresult):
            dim = gaussianresult.shape

            #CREATE NEW, SEPARATE MATRICES FOR GX, GY, GRADIENT MAGNITUDE, and GRADIENT ANGLE
            gx_arr = numpy.zeros(dim, dtype=numpy.int)
            gy_arr = numpy.zeros(dim, dtype=numpy.int)
            g_magnitude_arr = numpy.zeros(dim, dtype=numpy.int)
            g_angle_arr = numpy.zeros(dim)

            # PREWITT OPERATOR USED TO CALCULATE GX AND GY
            prewitt_gx = numpy.array([[-1,0,1],
                                                                                        [-1,0,1],
                                                                                        [-1,0,1]])
            prewitt_gy = numpy.array([[1,1,1],
                                                                                        [0,0,0],
                                                        [-1,-1,-1]])

    # 3X3 SUBMATRIX THAT WILL BE TAKEN AT EACH IMAGE PIXEL, AND PREWITT OPERATOR APPLIED TO
    img_submatrix = numpy.zeros((3,3))

    for i in range(4,(gaussianresult.shape[0]-4)):
        for j in range(4,(gaussianresult.shape[1]-4)):

            img_submatrix = gaussianresult[i-1:i+2, j-1:j+2]
            gx_sum = 0
            gy_sum = 0
            for k in range(0,3):
                for l in range(0,3):
                    gx_sum += img_submatrix[k,l] * prewitt_gx[k,l]
                    gy_sum += img_submatrix[k,l] * prewitt_gy[k,l]

            abs_gx = abs(gx_sum)                    #take absolute value of gx
            normal_gx = abs_gx/3                    #normalize gx
            gx_arr[i,j] = normal_gx

            abs_gy = abs(gy_sum)                    #take absolute value of gy
            normal_gy = abs_gy/3                    #normalize gy
            gy_arr[i,j] = normal_gy

            g_magnitude_arr[i,j] = math.sqrt(normal_gx**2 + normal_gy**2)/math.sqrt(2)          #populate gradient magnitude matrix
            g_angle_arr[i,j] = math.degrees(math.atan2(normal_gy,normal_gx))   #populate gradient angle matrix

    plt.imshow(gx_arr, cmap='gray')
    plt.show()

    plt.imshow(gy_arr, cmap='gray')
    plt.show()

    plt.imshow(g_magnitude_arr, cmap='gray')
    plt.show()

    return (g_magnitude_arr, g_angle_arr)          #return a tuple containing:    1.) gradient magnitude matrix    &    2.) gradient angle matrix

########################################################
### CANNY EDGE DETECTOR STEP 3: NON-MAXIMA SUPPRESSION ###
########################################################

def nms((magnitude, angle)):

    nms_result = numpy.zeros(magnitude.shape, dtype=int)

    for j in range(1,(magnitude.shape[1]-1)):
        for i in range(1,(magnitude.shape[0]-1)):

            #ASSIGN REGION TO ALL APPLICABLE PIXELS BASED ON GRADIENT ANGLE
            gradient_angle = (angle[i,j])
            if gradient_angle > -22.5 and gradient_angle <= 22.5:
                region = 0
            elif gradient_angle > 22.5 and gradient_angle <= 67.5:
                region = 1
            elif gradient_angle > 67.5 and gradient_angle <= 112.5:
                region = 2
            elif gradient_angle > 112.5 and gradient_angle <= 157.5:
                region = 3
            elif gradient_angle > 157.5 and gradient_angle <= 202.5:
                region = 0
            elif gradient_angle > 202.5 and gradient_angle <= 247.5:
                region = 1
            elif gradient_angle > -112.5 and gradient_angle <= -67.5:
                region = 2
            elif gradient_angle > -67.5 and gradient_angle <= -22.5:
                region = 3

            #BASED ON REGION, COMPARE ALL PIXELS TO NEIGHBORS AND SUPPRESS ALL NON-MAXIMA VALUES (SET TO ZERO)
            if region == 0:
                if magnitude[i,j] >= magnitude[i,j+1] and magnitude[i,j] >= magnitude[i,j-1]:
                    nms_result[i,j] = magnitude[i,j]
                else:
                    nms_result[i,j] = 0
            elif region == 1:
                if magnitude[i,j] >= magnitude[i-1,j+1] and magnitude[i,j] >= magnitude[i+1,j-1]:
                    nms_result[i,j] = magnitude[i,j]
                else:
                    nms_result[i,j] = 0
            elif region == 2:
                if magnitude[i,j] >= magnitude[i-1,j] and magnitude[i,j] >= magnitude[i+1,j]:
                    nms_result[i,j] = magnitude[i,j]
                else:
                    nms_result[i,j] = 0
            elif region == 3:
                if magnitude[i,j] >= magnitude[i-1,j-1] and magnitude[i,j] >=  magnitude[i+1,j+1]:
                    nms_result[i,j] = magnitude[i,j]
                else:
                    nms_result[i,j] = 0

    plt.imshow(nms_result, cmap='gray')
    plt.show()
    return nms_result


#################################################################
### CANNY EDGE DETECTOR STEP 4: THRESHOLDING USING P-TILE METHOD ###
#################################################################

def threshold(nms_result):

    #CREATE A HISTOGRAM OF IMAGE
    histogram = [0 for c in range(0,256)]
    total_px_count = 0
    for i in range(0,(nms_result.shape[0])):
        for j in range(0,(nms_result.shape[1])):
            if nms_result[i,j] > 0:
                histogram[nms_result[i,j]] += 1
                total_px_count += 1

    r10 = numpy.array(nms_result)
    r30 = numpy.array(nms_result)
    r50 = numpy.array(nms_result)


    ########
    #P = 10%
    ########

    ptile10 = total_px_count / 10
    count10 = 0
    h10 = 255
    while count10 <= ptile10:
        count10 += histogram[h10]
        h10 -= 1

    #FIND BEST THRESHOLD (MIGHT BE ABOVE OR BELOW EXACT T, WHICHEVER IS CLOSEST)
    if abs(count10 - ptile10) <= abs(count10 - histogram[h10+1] - ptile10):
        h10 += 1
    else:
        count10 -= histogram[h10+1]
        h10 += 2

    t10 = h10
    print("10% Threshold Gray Level: ", t10)
    print("Number of edge pixels (assuming 10% Threshold): ", count10)

    for i in range(0,r10.shape[0]):
        for j in range(0,r10.shape[1]):
            if r10[i,j] < t10:
                r10[i,j] = 0
            else:
                r10[i,j] = 255

    plt.imshow(r10, cmap='gray')
    plt.show()

    ########
    #P = 30%
    ########

    ptile30 = total_px_count * 0.3
    count30 = 0
    h30 = 255
    while count30 <= ptile30:
        count30 += histogram[h30]
        h30 -= 1

    #FIND BEST THRESHOLD (MIGHT BE ABOVE OR BELOW EXACT T, WHICHEVER IS CLOSEST)
    if abs(count30 - ptile30) <= abs(count30 - histogram[h30+1] - ptile30):
        h30 += 1
    else:
        count30 -= histogram[h30+1]
        h30 += 2

    t30 = h30

    print("30% Threshold Gray Level: ", t30)
    print("Number of edge pixels (assuming 30% Threshold): ", count30)

    for i in range(0,r30.shape[0]):
        for j in range(0,r30.shape[1]):
            if r30[i,j] < t30:
                r30[i,j] = 0
            else:
                r30[i,j] = 255
    plt.imshow(r30, cmap='gray')
    plt.show()

    ########
    #P = 50%
    ########

    ptile50 = total_px_count * 0.5
    count50 = 0
    h50 = 255
    while count50 <= ptile50:
        count50 += histogram[h50]
        h50 -= 1

    #FIND BEST THRESHOLD (MIGHT BE ABOVE OR BELOW EXACT T, WHICHEVER IS CLOSEST)
    if abs(count50 - ptile50) <= abs(count50 - histogram[h50+1] - ptile50):
        h50 += 1
    else:
        count50 -= histogram[h50+1]
        h50 += 2

    t50 = h50
    print("50% Threshold Gray Level: ", t50)
    print("Number of edge pixels (assuming 50% Threshold): ", count50)

    for i in range(0,r50.shape[0]):
        for j in range(0,r50.shape[1]):
            if r50[i,j] < t50:
                r50[i,j] = 0
            else:
                r50[i,j] = 255

    plt.imshow(r50, cmap='gray')
    plt.show()


    return r30

###############################################
### CANNY EDGE DETECTOR: PRIMARY FUNCTION ###
###############################################

def cannyedges(q):
    step1 = gaussian(q)
    step2 = gradientify(step1)
    step3 = nms(step2)
    step4 = threshold(step3)

######################################
### END OF FUNCTIONS/SUBPROGRAMS ###
######################################



############################
#### ! MAIN PROGRAM ! ####
############################

cannyedges(indexed_image)
```