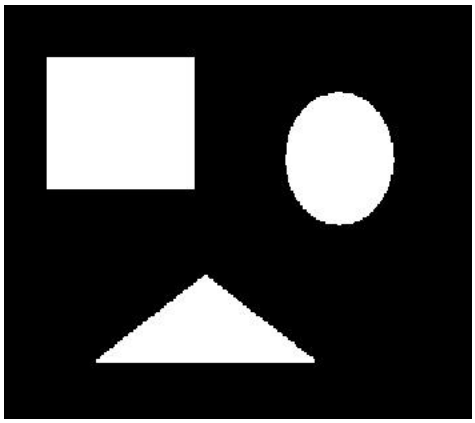
Project 2 of CSE 473/573

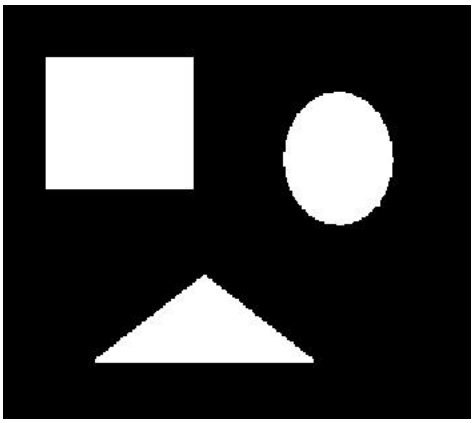
Shaoming Xu UB# 50247057

Task 1

Question1



res_noise1.jpg

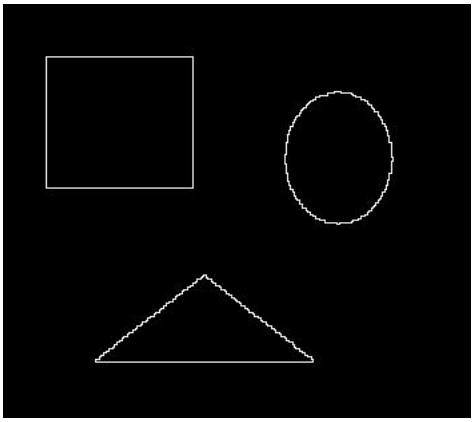


res_noise2.jpg

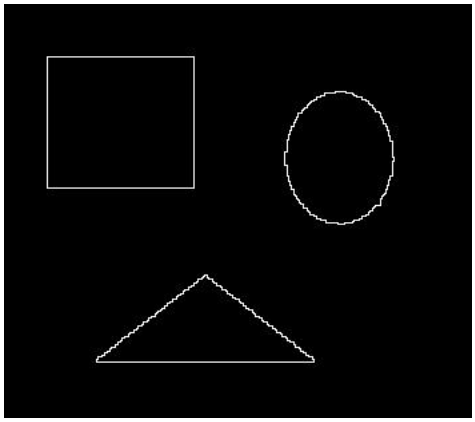
Question2

They are different in my report. The res_noise1.jpg is got from first opening then closing. The res_noise2.jpg is got from first closing then opening. Therefore, the size of the object in res_noise2 will be larger or equal to the res_noise1. We can find the size of ellipses and triangles in res_noise2 are larger than those size in res_noise1.

Question3



res_bound1.jpg



res_bound2.jpg

Task1 Source code:

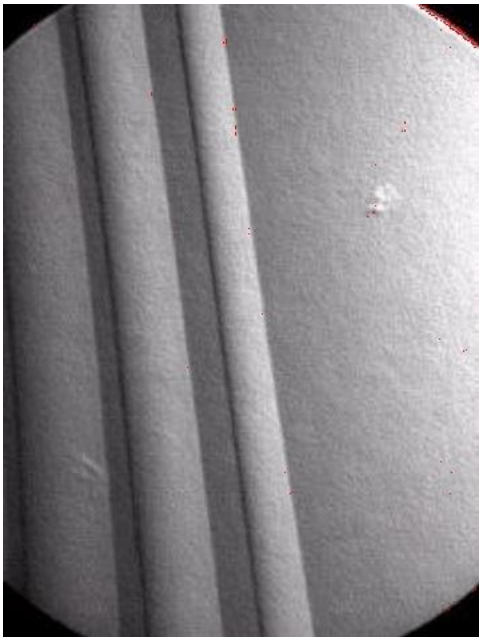
```
def dilation(binary_image, selem):
    row_pad = math.floor(selem.shape[0] / 2)
    col_pad = math.floor(selem.shape[1] / 2)
    res_img = np.zeros(binary_image.shape)
    check_img = np.pad(binary_image, ((row_pad,row_pad),(col_pad, col_pad)), 'constant')
    flipped_selem = np.flip(selem)
    for i in np.arange(row_pad, check_img.shape[0] - row_pad):
        for j in np.arange(col_pad, check_img.shape[1] - col_pad):
            patch = check_img[i-row_pad:i+row_pad+1, j-col_pad:j+col_pad+1]
            if np.any(patch & flipped_selem == 1):
                res_img[i-row_pad, j-col_pad] = 1
    return res_img.astype(np.uint8)

def erosion(binary_image, selem):
```

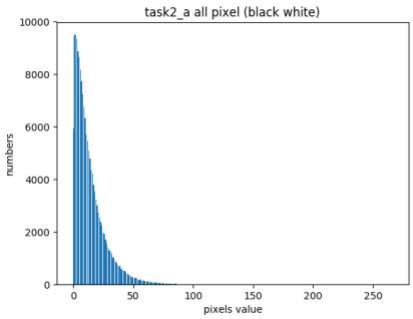
```
    row_pad = math.floor(selem.shape[0] / 2)
    col_pad = math.floor(selem.shape[1] / 2)
    res_img = np.zeros(binary_image.shape)
    check_img = np.pad(binary_image, ((row_pad,row_pad),(col_pad, col_pad)), 'constant')
    for i in np.arange(row_pad, check_img.shape[0] - row_pad):
        for j in np.arange(col_pad, check_img.shape[1] - col_pad):
            patch = check_img[i-row_pad:i+row_pad+1, j-col_pad:j+col_pad+1]
            if np.all(patch & selem == selem):
                res_img[i-row_pad, j-col_pad] = 1
    return res_img.astype(np.uint8)

def opening(binary_image, selem):
    return dilation(erosion(binary_image, selem), selem)

def closing(binary_image, selem):
    return erosion(dilation(binary_image, selem), selem)

def boundary(binary_image):
    selem = np.ones((3,3)).astype(np.uint8)
    return binary_image - erosion(binary_image, selem)

def denoising(method=1):
    def method_1(binary_image, selem):
        return closing(opening(binary_image, selem), selem)

    def method_2(binary_image, selem):
        return opening(closing(binary_image, selem), selem)

    if method == 1:
        return method_1
    else:
        return method_2

def threshold(gray_img, thresh):
    res_img = np.zeros(gray_img.shape)
    for i in range(gray_img.shape[0]):
        for j in range(gray_img.shape[1]):
            if gray_img[i,j] > thresh:
                res_img[i,j] = 1
    return res_img.astype(np.uint8)
if __name__ == "__main__":
    img = cv2.imread("../task1_img/noise.jpg")
    img = cv2.imread("../task1_img/noise.jpg")
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    thresh = 127
    struc_elem = np.ones((3,3)).astype(np.uint8)
    binary = threshold(gray, thresh)
    print("**** task1 question a ****")
    res_noise1 = denoising(method=1)(binary, struc_elem)
    name = "../task1_img" + "/res_noise1" + ".jpg"
    cv2.imwrite(name, res_noise1*255)
    res_noise2 = denoising(method=2)(binary, struc_elem)
    name = "../task1_img" + "/res_noise2" + ".jpg"
    cv2.imwrite(name, res_noise2*255)
    print("**** task1 question c ****")
    res_bound1 = boundary(res_noise1)
    name = "../task1_img" + "/res_bound1" + ".jpg"
    cv2.imwrite(name, res_bound1*255)
    res_bound2 = boundary(res_noise2)
    name = "../task1_img" + "/res_bound2" + ".jpg"
    cv2.imwrite(name, res_bound2*255)
```

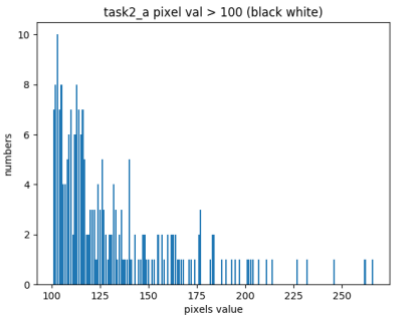
Task 2a:

In this part, I have done two set experiments. One is to do point detection for both white and black point as same as that in slide. The other is only detect the white point. It turns out in our case, only do white point detection is better. Below I will discuss these two separately.

Let's discuss white and black case First.

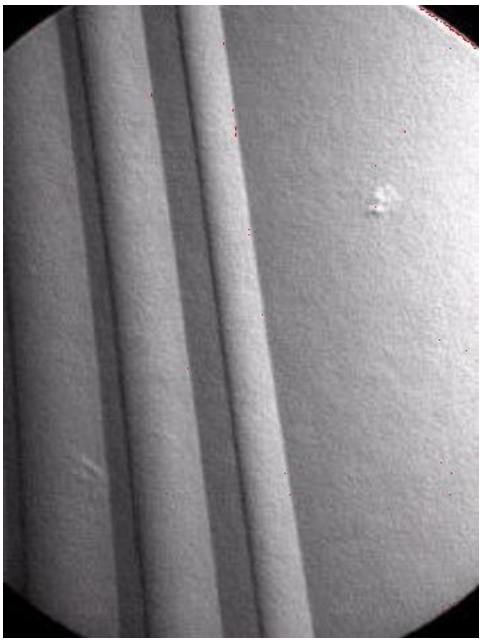


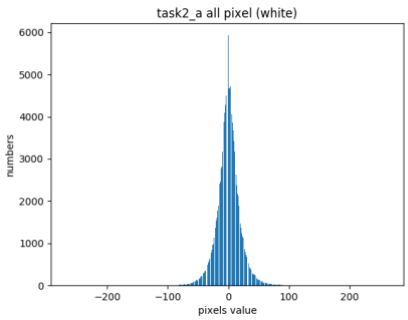


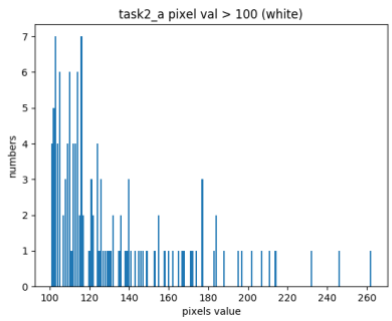


The left image upon show the result after white black point detection. The left two images are the histogram for the pixel value counts on absolute matrix of the original image convoluted with point detection mask. The threshold 110 is got from observation from histogram graph.

Then here we discuss the white case. The only difference of it is that we do not do absolute operation on the convoluted image. We can see the histogram has the pixel values smaller than zero.







The left image upon show the result in this case. We can see it has less noise than the first one. The threshold is 110, which is got from observation on histogram graph and experiments.

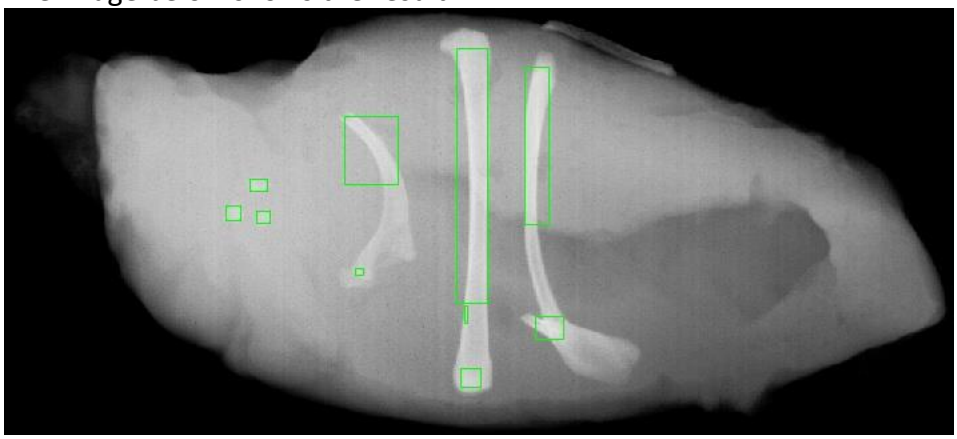
Task 2a code:

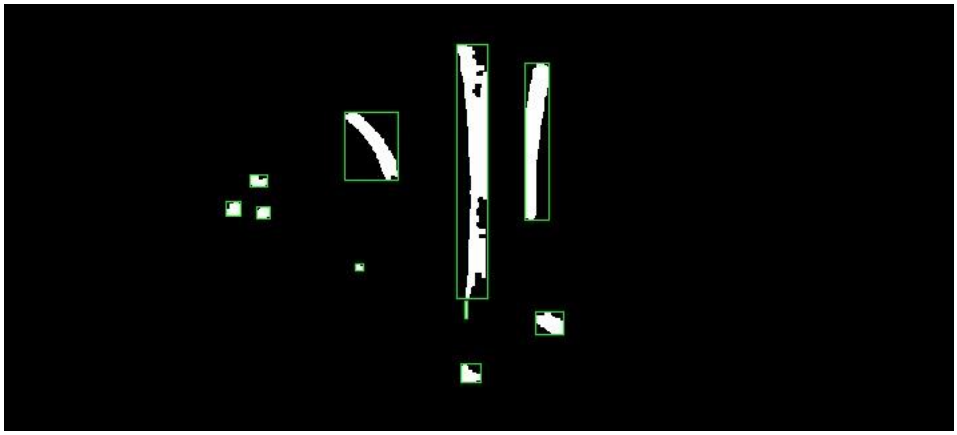
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from mylibrary import count_pixels

def get_masked_img(img_gray):
    """
    Purpose:
        measure the weighted difference between the center point and its neighbors.
    Input:
        img_gray: matrix, value type can be real.
    Output:
        res_img: matrix, value type can be real.
    """
    mask = np.array([[-1,-1,-1],
                     [-1,8,-1],
                     [-1,-1,-1]])
    pad_img = np.pad(img_gray, ((1,1),(1,1)), 'constant')
    res_img = np.zeros(img_gray.shape)
    for i in range(1, pad_img.shape[0] - 1):
        for j in range(1, pad_img.shape[1] - 1):
            res_img[i-1,j-1] = np.sum(pad_img[i-1:i+2, j-1:j+2] * mask)
    return res_img

def detect_point(img_gray, T, white_only=False):
    """
    Purpose:
        Detect points in images
    Input:
        img_gray: matrix with value type real
        T: threshold, used to record pixels which value greater T.
        white_only: boolean, if False, detect black and white points. Otherwise, only detect white.
    Output:
        res_img: matrix with value type np.uint8. use to record points.
        stat: a dictionary, key is pixel value, value is number of given pixel value.
    """
    masked_img = get_masked_img(img_gray)
    if white_only == False:
        masked_img = np.abs(masked_img)

    stat = count_pixels(masked_img, include_border=False)
    res_img = np.zeros(img_gray.shape).astype(np.uint8)
    #ignore border
    for i in range(1, masked_img.shape[0] - 1):
        for j in range(1, masked_img.shape[1] - 1):
            if masked_img[i,j] > T:
                res_img[i,j] = 1
    return res_img, stat


def mark_img(img, binary_img):
    res = img.copy()
    for i in range(binary_img.shape[0]):
        for j in range(binary_img.shape[1]):
            if binary_img[i,j] == 1:
                res[i,j] = np.array([0,0,255]).astype(np.uint8)
    return res

if __name__ == "__main__":
    ## detect both black and white points
    img = cv2.imread("../task2a_img/point.jpg")
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    binary, stat = detect_point(gray_img, 110)
    res_img = mark_img(img, binary)
    key_val_arr = np.asarray([[key, val] for key,val in stat.items()])
    check_all = key_val_arr.T
    plt.bar(check_all[0],check_all[1],align='center') # A bar chart
```

```
plt.title('task2_a all pixel (black white)')
plt.xlabel('pixels value')
plt.ylabel('numbers')
plt.savefig("../task2a_img/task2a_hist_all_black_white")
plt.close()

check_greater = np.asarray([x for x in key_val_arr if x[0] > 100]).T
plt.bar(check_greater[0],check_greater[1],align='center') # A bar chart
plt.title('task2_a pixel val > 100 (black white)')
plt.xlabel('pixels value')
plt.ylabel('numbers')
plt.savefig("../task2a_img/task2a_hist_100_black_white")
plt.close()

cv2.imwrite('../task2a_img/res_point.jpg', res_img)

## detect only white points
gray_img = cv2.imread("../task2a_img/point.jpg", 0)
binary, stat = detect_point(gray_img, 110, white_only=True)
res_img = mark_img(img, binary)
key_val_arr = np.asarray([[key, val] for key,val in stat.items()])
check_all = key_val_arr.T
plt.bar(check_all[0],check_all[1],align='center') # A bar chart
plt.title('task2_a all pixel (white)')
plt.xlabel('pixels value')
plt.ylabel('numbers')
plt.savefig("../task2a_img/task2a_hist_all_white")
plt.close()

check_greater = np.asarray([x for x in key_val_arr if x[0] > 100]).T
plt.bar(check_greater[0],check_greater[1],align='center') # A bar chart
plt.title('task2_a pixel val > 100 (white)')
plt.xlabel('pixels value')
plt.ylabel('numbers')
plt.savefig("../task2a_img/task2a_hist_100_white")
plt.close()

cv2.imwrite('../task2a_img/res_point_white.jpg', res_img)
```

Task 2b:

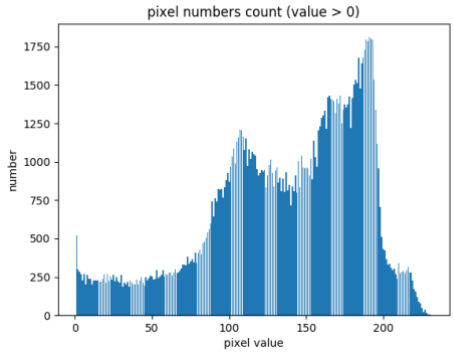
The image below shows the result.





I follow these steps to get the final results.

First, I count the pixel of gray scale original image, and use its result to draw the histogram graph.



The left is the histogram graph which ignore the pixel value of zero. By observation this graph, I do a set of experiments to get the optimal threshold as 205 to binary the image.

Second, I do the closing and then opening operation on the binary image to get the denoised image.

Third, I use DFS algorithm to recursively record all the pixel coordinations for every detected object, and save the record in the obj_bank list.

Finally, I use the obj_bank to draw the bounding box for each object.

Through all these steps, I got these results:

There are 10 objects

object 0 has 2314 numbers pixels

object 1 has 1273 numbers pixels

object 2 has 610 numbers pixels

object 3 has 103 numbers pixels

object 4 has 106 numbers pixels

object 5 has 89 numbers pixels

object 6 has 31 numbers pixels

object 7 has 42 numbers pixels

object 8 has 244 numbers pixels

object 9 has 140 numbers pixels

Sort by left_up's second value, it's the row index. You can corelate records with objects on image easily by rows from up to down.

object 0's cordinate: left_up=(339, 30), right,down=(362, 221)

object 1's cordinate: left_up=(390, 44), right,down=(408, 162)

object 2's cordinate: left_up=(255, 81), right,down=(295, 132)

object 3's cordinate: left_up=(184, 128), right,down=(197, 137)

object 4's cordinate: left_up=(166, 148), right,down=(177, 159)

object 5's cordinate: left_up=(189, 152), right,down=(199, 161)

object 6's cordinate: left_up=(263, 195), right,down=(269, 200)

object 7's cordinate: left_up=(345, 223), right,down=(347, 236)

object 8's cordinate: left_up=(398, 231), right,down=(419, 248)

object 9's cordinate: left_up=(342, 270), right,down=(357, 284)

Task 2b code:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import task1
from mylibrary import count_pixels
import sys

def thresh(gray_img, T):
    res_img = np.zeros(gray_img.shape).astype(np.uint8)
    for i in range(gray_img.shape[0]):
        for j in range(gray_img.shape[1]):
            if gray_img[i,j] > T:
                res_img[i,j] = 1
    return res_img

def numIslands(binary_img):
    if binary_img is None or binary_img.shape[0] == 0 or binary_img.shape[1] == 0:
        return 0, []

    obj_bank = []
    binary = binary_img.copy()

    for i in range(binary.shape[0]):
        for j in range(binary.shape[1]):
            if binary[i,j] == 1:
                obj = []
                dfs(binary, i, j, obj)
                obj_bank.append(obj)

    return obj_bank

def dfs(grid, x, y, obj):
    n = grid.shape[0]
    m = grid.shape[1]
    if x < 0 or x > n - 1 or y < 0 or y > m - 1 or grid[x, y] == 0:
        return
    obj.append([x,y])
    grid[x,y] = 0
    dfs(grid, x+1, y, obj)
    dfs(grid, x-1, y, obj)
    dfs(grid, x, y-1, obj)
    dfs(grid, x, y+1, obj)


def draw_box(img, binary, obj_bank):
    res_img = img.copy()
    binary_color = cv2.cvtColor(binary*255, cv2.COLOR_GRAY2BGR)
    res_dcolor = binary_color

    for i, obj in enumerate(obj_bank):
        obj = np.asarray(obj).T
        up = min(obj[0])
        down = max(obj[0])
        left = min(obj[1])
        right = max(obj[1])
        print("object {}'s cordinate: left_up={}, right,down={}".format(i, (left, up), (right, down)))
        res_img = cv2.rectangle(res_img,(left, up),(right,down),(0,255,0),shift=0)
        res_dcolor = cv2.rectangle(res_dcolor,(left, up),(right,down),(0,255,0),shift=0)
    return res_img, res_dcolor
```

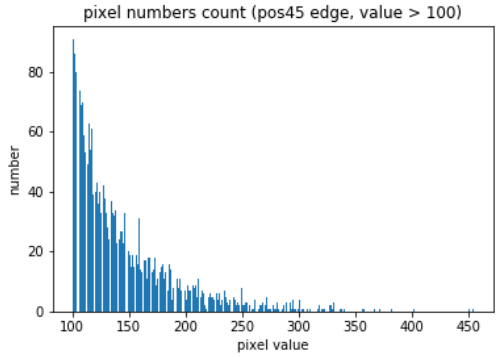
```
if __name__ == "__main__":
    sys.setrecursionlimit(1500)
    img = cv2.imread("../task2b_img/segment.jpg")
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    pixel_stat = count_pixels(img_gray)
    stat_list = np.asarray([[key,val] for key,val in pixel_stat.items() if key != 0]).T

    # save the histogram
    plt.bar(stat_list[0],stat_list[1],align='center') # A bar chart
    plt.title("pixel numbers count (value > 0)")
    plt.xlabel('pixel value')
    plt.ylabel('number')
    plt.savefig("../task2b_img/task2b_hist")
    plt.close()

    # observe the histogram to get T
    T = 205
    binary_img = thresh(img_gray, 205)
    #denoise the binary_img
    struc_elem = np.ones((3,3)).astype(np.uint8)
    denoised = task1.denoising(method=2)(binary_img, struc_elem)
    obj_bank = numIslands(denoised)
    print("There are {} objects".format(len(obj_bank)))
    for i in range(len(obj_bank)):
        print("object {} has {} numbers pixels".format(i, len(obj_bank[i])))

    img_res, dcolor_res = draw_box(img, denoised, obj_bank)
    cv2.imwrite('../task2b_img/res_segment.jpg', img_res)
    cv2.imwrite('../task2b_img/res_segment_2.jpg', dcolor_res)
```

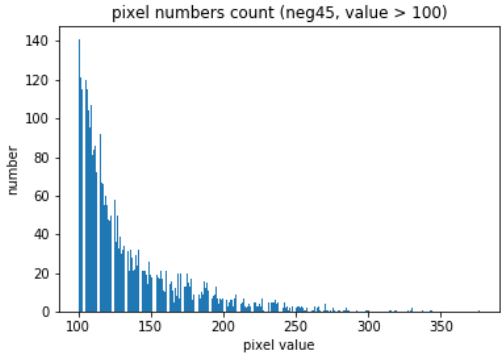
Task 3:

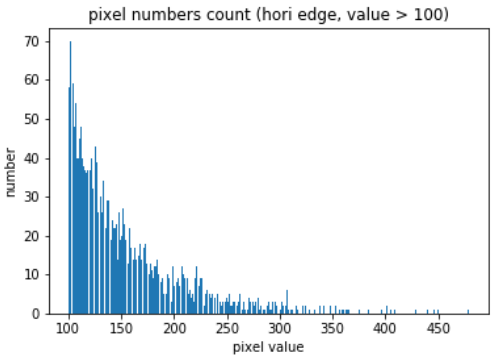
In the task3, I have implemented both edge detection algorithms and Hough transform algorithms.

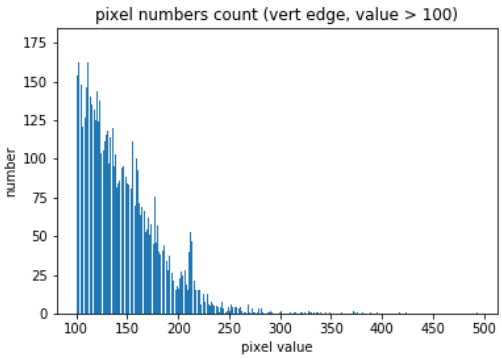
Different from the opencv Canny edge detection algorithm, my edge detection is based on Morphology operation. My algorithm is followed on these steps.

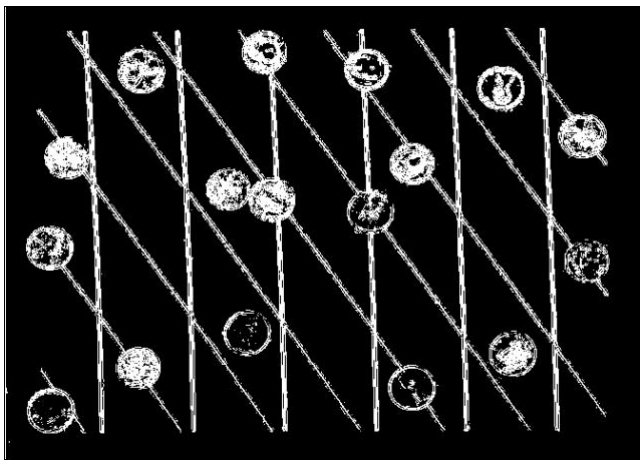
First, do the line detection on four direction, and generate four histograms.



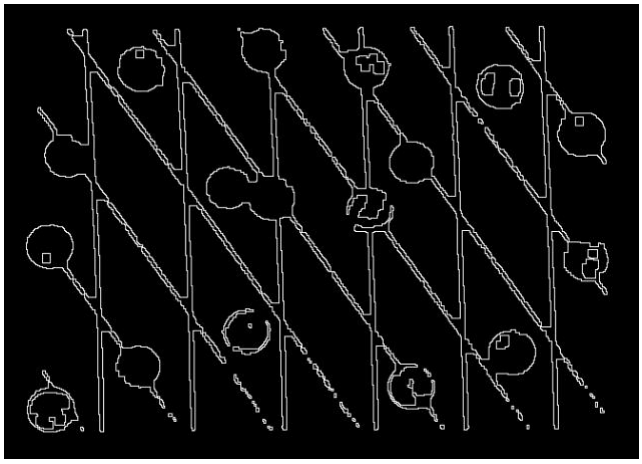






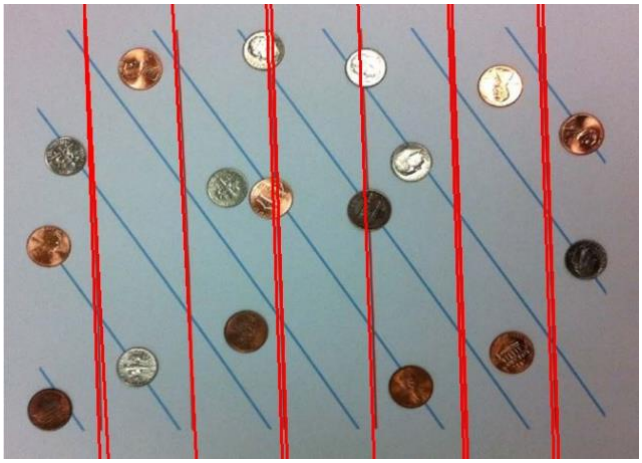


Second, based these histograms to find the optimal threshold to combine the four edge images to a single one.



Third, do morphology operation follow these sequences: closing -> opening -> closing -> boundary to extract the boundary of image.

Fourth, do the Hough Transform on the boundary to find information of all Lines. To save the memory space and speed up computation, my Hough transform use a python dictionary to record the line information. The key is a tuple (rho, radian), value is the number of its key. The idea of Hough Transform algorithm is simple and elegant. It is just do voting on rho and radian space for each candidate points. Then use threshold on voting to find lines.

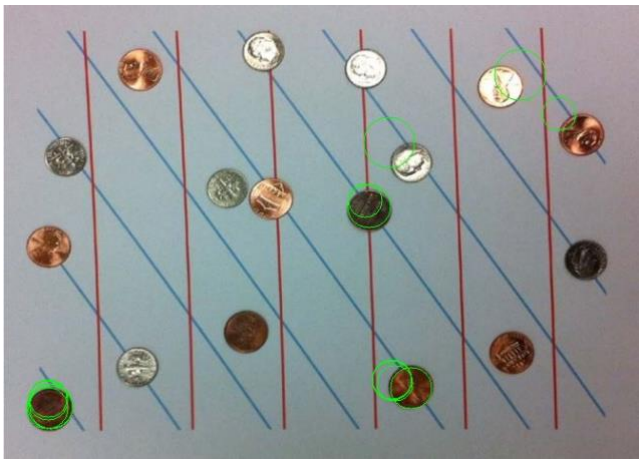


Sixth, we start to detect lines with given radians. To detect the red line, I set the radian to be 3.1 (in np.pi scale) and the gap as 0.2 to extract all line with radian in (2.9, 3.3) range. After that I set the threshold as 160 to extract lines with Hough rating over 160. Then I draw the lines on image. The left image shows I detect all red lines correctly.

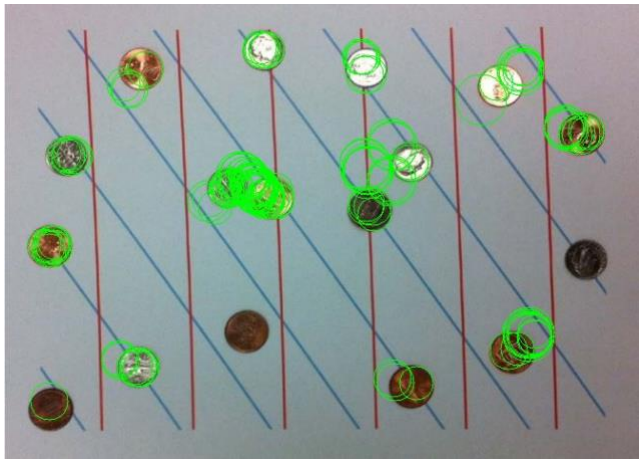
To detect the blue lines. I set the radian as 2.5 and gap as 0.2 to extract all lines in with radian in (2.3, 0.2) range. After that I set the threshold as 140 to extract lines with Hough rating over 140. Then I draw the lines on image. The left image shows I detect six blue lines.

Task 3 bonus:

I use two method to do circle detection. One is to do circle detection on my edge detection algorithm. The other is to directly do the circle detection on opencv canny edge. The canny edge method can detect more circle, but it is very slow. The idea behind Hough circle algorithm is also very simple and elegant. It is just do voting on (a, b, r) for each candidate points. Then use threshold on voting to find circle (a, b are the center of circle, r is the radius). There are two kinds of coins, I set the r range in 18 to 28 in my algorithm.



The left image shows the result of the circle detected by using the Hough circle on my own edge detection. It detects three coins correctly.



This left image shows the result of the circle detected by using on the canny edge detection. We can see it detect 15 coins, along with many incorrect circles. At last, this method is very computation intensive. I recommend you to comment it out in my code if you time is limited.

Task 3 code:

```
import numpy as np
import cv2
import math
import matplotlib.pyplot as plt
from mylibrary import count_pixels
import task1

## the edge detection part
# sober for edge detection
VERTICAL_SOBEL_3BY3 = np.array([[-1,2,-1],
                    [-1,2,-1],
                    [-1,2,-1]])

HORIZONTAL_SOBEL_3BY3 = np.array([[-1,-1,-1],
                    [2,2,2],
                    [-1,-1,-1]])
POS45_SOBEL_3BY3 = np.array([[-1,-1,2],
                    [-1,2,-1],
                    [2,-1,-1]])
NEG45_SOBEL_3BY3 = np.array([[2,-1,-1],
                    [-1,2,-1],
                    [-1,-1,2]])

def texture_filtering(img_gray, kernel):
    """
    Purpose:
        use to filter the gray image given the kernel
    Input:
        img_gray:
            an two dimension ndarray matrix, dtype:usually is uint8 representint the gray image.
        kernel:
            a two dimension ndarray matrix
    Output:
        The filtered image without padding around.
    """
    row_pad = math.floor(kernel.shape[0] / 2)
    col_pad = math.floor(kernel.shape[1] / 2)

    res_img = np.zeros(img_gray.shape)
    check_img = np.pad(img_gray, ((row_pad,row_pad),(col_pad, col_pad)), 'constant')
    flipped_kernel = np.flip(kernel)

    for i in range(row_pad, check_img.shape[0] - row_pad):
        for j in range(col_pad, check_img.shape[1] - col_pad):
            patch = check_img[i-row_pad:i+row_pad+1, j-col_pad:j+col_pad+1]
            res_img[i-row_pad, j-col_pad] = np.sum(patch * flipped_kernel)
    return res_img


def eliminate_zero(img, method=1):
    """
    Purpose:
        two ways to eliminate the negative value or the value out of 255.
    Input:
        img: two dimension matrix
            the raw image. dtype usually is float64 with pixel < 0 or pixel > 255
        method: int
            default is 1 which directs to first method
            the 2 will direct to the second method.
    Output:
        a matrix dtype range zero to one.
    """
    if method == 1:
        min_ = np.min(img)
        max_ = np.max(img)
        return (img - min_) / (max_ - min_)
    elif method == 2:
        abs_img = np.abs(img)
        return abs_img / np.max(abs_img)
    else :
        print("method is 1 or 2")
```

```
def combine_edge(hori,vert,pos45,neg45,T):
    """
    Purpose:
        to combine edges in four direction.
        Mark result image 1 in given loc if any of these four edge images pixels absolute value
        greater than T in that loc.
    Output:
        res_img: binary img, value type np.uint8
    """
    hori = np.abs(hori)
    vert = np.abs(vert)
    pos45 = np.abs(pos45)
    neg45 = np.abs(neg45)
    res_img = np.zeros(hori.shape).astype(np.uint8)
    for i in range(hori.shape[0]):
        for j in range(hori.shape[1]):
            if hori[i,j] > T or vert[i,j] > T or pos45[i,j] > T or neg45[i,j] > T :
                res_img[i,j] = 1
    return res_img

def ignore_border(img_gray):
    """
    Purpose:
        ignore the img bodre by setting value as 0.
        notice: Affect original img.
    Output:
        the pointer point to original image.
    """
    img_gray[0] = 0
    img_gray[-1] = 0
    for row in img_gray:
        row[0] = 0
        row[-1] = 0
    return img_gray

### the hough algorithms part
def hough(edge_img):
    """
    Purpose:
        Main function of hough transform.
        Notice: Here I use the rho = x * sin(theta) + y * cos(theta). It is different from the hough equation.
        I do this to meet the convention of opencv so that to use opencv library easily later.
    Input:
        edge_img: binary image, the boundary image after edge detection.
    Output:
        res_dic: a dictionary, key is a tuple (rho, radian), value is the number of its key.
    """
    m, n = edge_img.shape
    res_dic = {}
    for i in range(m):
        for j in range(n):
            if edge_img[i,j] == 1:
                for degree in range(0,180):
                    radian = np.radians(degree)
                    rho = int(i * np.sin(radian) + j * np.cos(radian))
                    if ((rho, radian)) not in res_dic:
                        res_dic[(rho, radian)] = 0
                    res_dic[(rho, radian)] += 1
    return res_dic

def pick_by_count(lines, T):
    """
    Purpose:
        filter the lines which number of (rho, radian) greater than threshold T.
    Input:
        lines: a matrix, col_1: rho, col_2: radian, col_2: number of (rho, radian) tuple. It can be got by res_dic.
    Output:
        matrix, numpy array, each row represent a selected line.

    """
    res = []
    for line in lines:
        if line[2] > T:
            res.append(line)
```

```
    return np.asarray(res)

def pick_by_theta(lines, theta, gap):
    """
    Purpose:
        filter the lines by theta,
    Input:
        lines: a matrix, col_1: rho, col_2: radian, col_2: number of (rho, radian) tuple.
        theta: real, is a radian, will compared radians from col_2
        gap: the maximum difference between two theta.
    Output:
        matrix, numpy array, each row represent a selected line.
    """
    res = []
    for line in lines:
        if abs(line[1] - theta) < gap:
            res.append(line)
    return np.asarray(res)


def draw_lines(img, lines, loc="../task3_img/", name = "hough_res.jpg"):
    """
    Purpose:
        draw lines in image
        not affect original imgage.
    """
    img = img.copy()
    for line in lines:
        rho = line[0]
        theta = line[1]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))
        cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)
    cv2.imwrite(loc+name, img)

def circle_hough(edge_img):
    res_dic = {}
    for x in range(edge_img.shape[0]):
        for y in range(edge_img.shape[1]):
            if edge_img[x,y] == 1:
                for r in range(18,28):
                    for t in range(0,360):
                        a = x - r * np.cos(t * np.pi / 180)
                        b = y - r * np.sin(t * np.pi / 180)
                        if (a,b,r) not in res_dic:
                            res_dic[(a,b,r)] = 0
                        res_dic[(a,b,r)] += 1
    return res_dic

def draw_circles(img, circles, loc="../task3_img/", name="coin.jpg" ):
    img = img.copy()
    for cir in circles:
        cv2.circle(img,(cir[1], cir[0]), cir[2], (0,255,0), 1)
    cv2.imwrite(loc+name, img)



if __name__ == "__main__":
    img = cv2.imread('../task3_img/hough.jpg')
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    print("Start to do edge detection")
    ## get edges from four directions
    vertical = texture_filtering(img_gray, VERTICAL_SOBEL_3BY3)
    horizontal = texture_filtering(img_gray, HORIZONTAL_SOBEL_3BY3)
    pos45 = texture_filtering(img_gray, POS45_SOBEL_3BY3)
    neg45 = texture_filtering(img_gray, NEG45_SOBEL_3BY3)

    ## Find threshold through observation
    # vertical
```

```
pixel_stat = count_pixels(np.abs(vertical))
stat_list = np.asarray([[key,val] for key,val in pixel_stat.items() if key >100]).T
plt.bar(stat_list[0],stat_list[1],align='center') # A bar chart
plt.title("pixel numbers count (vert edge, value > 100)")
plt.xlabel('pixel value')
plt.ylabel('number')
plt.savefig("../task3_img/vert_hist")
plt.close()

#horizontal
pixel_stat = count_pixels(np.abs(horizontal))
stat_list = np.asarray([[key,val] for key,val in pixel_stat.items() if key >100]).T
plt.bar(stat_list[0],stat_list[1],align='center') # A bar chart
plt.title("pixel numbers count (hori edge, value > 100)")
plt.xlabel('pixel value')
plt.ylabel('number')
plt.savefig("../task3_img/hori_hist")
plt.close()

#pos45
pixel_stat = count_pixels(np.abs(pos45))
stat_list = np.asarray([[key,val] for key,val in pixel_stat.items() if key >100]).T
plt.bar(stat_list[0],stat_list[1],align='center') # A bar chart
plt.title("pixel numbers count (pos45 edge, value > 100)")
plt.xlabel('pixel value')
plt.ylabel('number')
plt.savefig("../task3_img/pos45_hist")
plt.close()

#neg45
pixel_stat = count_pixels(np.abs(neg45))
stat_list = np.asarray([[key,val] for key,val in pixel_stat.items() if key >100]).T
plt.bar(stat_list[0],stat_list[1],align='center') # A bar chart
plt.title("pixel numbers count (neg45, value > 100)")
plt.xlabel('pixel value')
plt.ylabel('number')
plt.savefig("../task3_img/neg45_hist")
plt.close()

## Combined edges using threshold
T = 50
combined = combine_edge(horizontal,vertical,pos45,neg45,T)
combined = ignore_border(combined)
cv2.imwrite("../task3_img/combined.jpg", (combined*255).astype(np.uint8))

## preprocess binary image
print("Start to preprocess binary image")
#denoise
struc_elem = np.ones((3,3)).astype(np.uint8)
denoised = task1.denoising(method=2)(combined, struc_elem)
cv2.imwrite("../task3_img/denoised.jpg", (denoised*255).astype(np.uint8))
#closing
struc_elem = np.ones((7,7)).astype(np.uint8)
closed = task1.closing(denoised, struc_elem)
cv2.imwrite("../task3_img/closed.jpg", (closed*255).astype(np.uint8))
#extract boundary
boundary = task1.boundary(closed)
cv2.imwrite("../task3_img/boundary.jpg", (boundary*255).astype(np.uint8))

## Start using hough algorithms
print("Start using hough algorithms")
hres = hough(boundary)
lines = np.asarray([[key[0], key[1], val] for key, val in hres.items()])
#filter by theta and count
#red lines
T = 160
theta_gap = 0.2
theta_red = 3.1
lines_red = pick_by_theta(lines, theta_red, theta_gap)
res_red = pick_by_count(lines_red, T)
draw_lines(img, res_red, loc="../task3_img/", name = "red_line.jpg")

#blue lines
T = 140
theta_gap = 0.2
theta_blue = 2.5
```

```
lines_blue = pick_by_theta(lines, theta_blue, theta_gap)
res_blue = pick_by_count(lines_blue, T)
draw_lines(img, res_blue, loc="../task3_img/", name = "blue_lines.jpg")

#Bonus Circle hough
print("start hough circle")
chres = circle_hough(boundary)
circles = [item for item in chres.items() if chres[item[0]] >= 4]
circles = np.asarray([[x[0][0], x[0][1], x[0][2], x[1]] for x in circles]).astype(int)
draw_circles(img, circles)
#Bonus using Canny edge detection
print("hough circle on Canny edge")
img = cv2.imread('../task3_img/hough.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)
edges = (edges/255).astype(np.uint)
chres = circle_hough(edges)
circles = [item for item in chres.items() if chres[item[0]] >= 4]
circles = np.asarray([[x[0][0], x[0][1], x[0][2], x[1]] for x in circles]).astype(int)
draw_circles(img, circles, name="coin_edge.jpg")
```