**CPSC 470/570 – Artificial Intelligence**

**Problem Set #8 – Computer Vision**

**25 points**

**Due Friday April 26th, 11:59:59pm**

Some reminders:

* **Grading contact:** \*(\*) is the point of contact for initial questions about grading for this problem set.
* **Late assignments** are not accepted without a Dean’s excuse.
* **Collaboration policy:** You are encouraged to discuss assignments with the course staff and with other students. However, you are required to implement and write any assignment on your own. This includes both pencil-and-paper and coding exercises. You are not permitted to copy, in whole or in part, any written assignment or program as part of this course. You are not to take code from any online repository or web source. You will not allow your own work to be copied. Homework assignments are your individual responsibility, and plagiarism will not be tolerated.
* **Students taking CPSC570:** There is no extra section for this assignment. Your assignment is the same as CPSC470.

In this assignment, you will complete computer vision tasks. The main library to use is skimage in python. It comes pre-installed with several Python distributions. It is also available on the zoo machines if you don’t have it on your machine. However, we may not be able to help with library installation. Other library you may need is numpy, which is used in ps6.

# Problem #1 : Edge Detection (10 Points)

Given the image “yale.png” provided in the starter code folder, you can import this image into python using the command:

from skimage import io

img = io.imread('yale.png')

This will create a three-dimensional array called img, which is an numpy array, will have dimensions of **HxWx3** where **H** is the height of the image and **W** is the width of the image. The last index gives access to the red, green and blue components of each pixel. Thus, img[0, 1, 2] gives you the blue component of the pixel in the first row and the second column. You can view the dimension of img with np.shape(). You can view this image:

io.imshow(img)

io.show()

Take the color image and convert it to a grayscale image:

from skimage.color import rgb2gray

grey\_img = rgb2gray(img)

This grayscale image grey\_img is a 2-D array of dimensions **HxW**. For more information on the rgb2gray can be found here if you are interested in:

<https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_rgb_to_gray.html>

In this section, you will detect the edges on the grayscale image using the following methods mentioned in class (lecture slide 24):

* Sobel operator
* Robert’s cross
* the Canny edge detector

Here are the corresponding implementation details corresponding to each method:

Sobel Operator

* Documentation:

<https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.sobel>

* Sample code:

from skimage.filters import sobel

sobel\_edge = sobel(grey\_img)

Robot’s Cross

* Documentation:

<https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.roberts>

* Sample code:

from skimage.filters import roberts

robert\_cross\_edge = roberts(grey\_img)

The Canny Edge Detector

* Documentation:

<https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.canny>

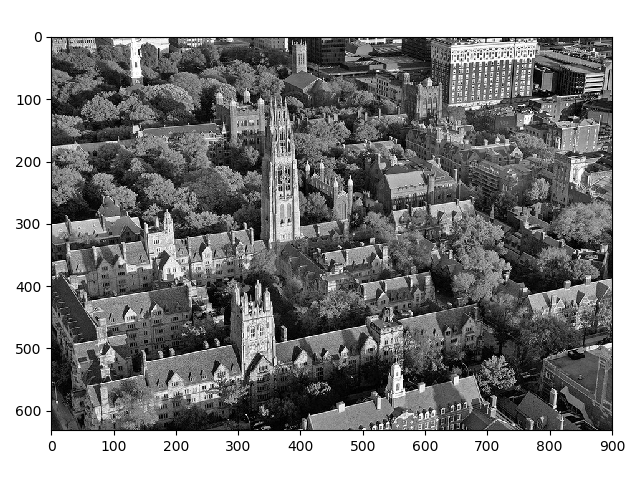
* Here in the documentation, there are several parameters you can set (e.g., low\_threshold, high\_threshold, and sigma). Try a few different values of each of these parameter settings and see how they impact the edge image.
* Sample code:

from skimage import feature

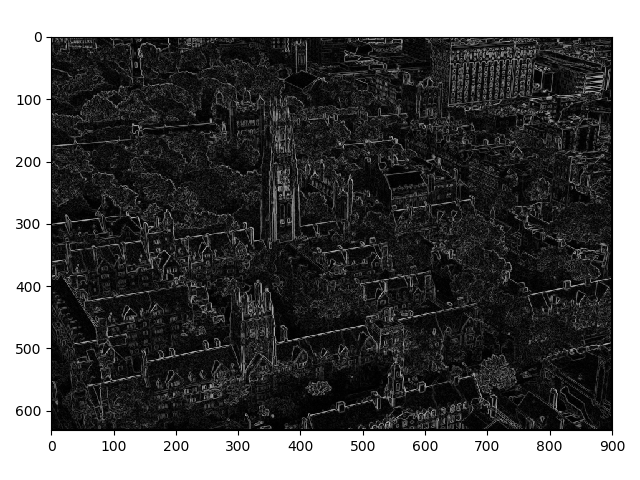
canny\_edge = feature.canny(grey\_img)

Question 1. Please get a picture of the greyscale image and the three edge images below. Please clearly label which is which (4 points)

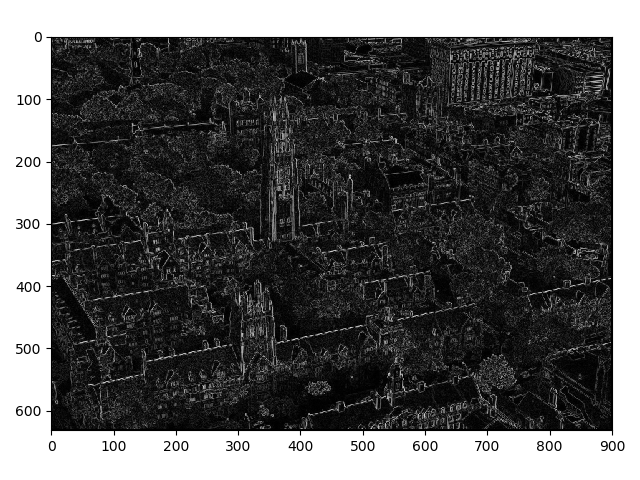
Greyscale:



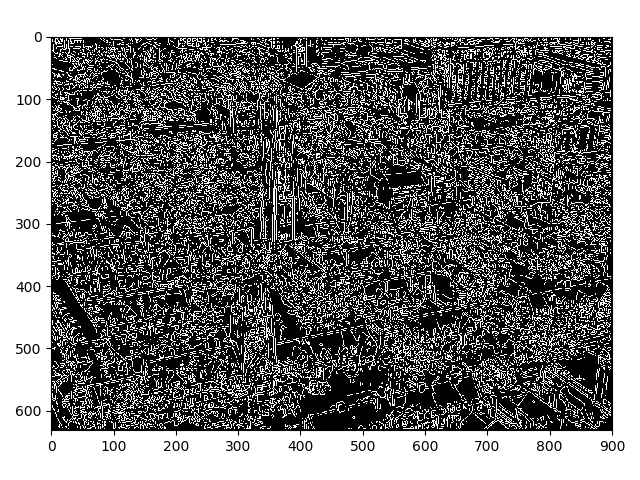
Sobel:



Robot’s cross:



Canny



Question 2. Please answer the following questions regarding the canny edge detection method. (3 points)

a) what impact does the “low\_threshold” have on the image? (1 point)

The larger the low\_threshold is, fewer edges are included.

b) what impact does the “high\_threshold” have on the image? (1 point)

The larger the high\_threshold is, fewer edges are included

c) what impact does the “sigma” have on the image? (1 point)

The larger sigma is, fewer edges are included.

Question 3. Please write a short paragraph that answers the following question: (3 points)

If a robot were to acquire a camera image that looked like your original image, and if that robot needed to navigate through the scene shown in the image, which of the edge detecting methods gives the best results? (Which method picks out the boundaries of major obstacles without providing too many details?) How do you judge this?

The technique that you choose for problems (like navigation) is often dependant on the image itself and on the parameter settings that you choose. This unfortunate fact makes it difficult to select an appropriate algorithm for real-world applications that must deal with a variety of environmental conditions. In general, you want an edge detector that picks up on the large, relevant edges (such as the side of a building) but tends to ignore excessive details (such as brickwork or the texture lines in concrete). More edges is not necessarily better.

# Problem #2 : Finding Color Blobs (15 Points)

In this problem, we will use region growing techniques on a color image to identify areas of an image that belong to several simple geometric shapes. We will use the image shown below (which you can also find in the start\_code folder).



Figure 1

Your goal is to identify the location of the centroid (the center of mass) and the extent (in the form of a bounding box) of each of the balls and rectangular blocks in the scene.

Your solution for doing this does **NOT** need to be elegant or general. It just needs to work on images that contain these same objects (although they might be in various positions.) You can assume that there will be little or no occlusion in any of the test images.

You are free to solve this problem any way that you want. Here is one idea on how to do it:

1. Divide the color image into three separate color-channel images (one for red, blue and green). You can do this with the command like:

redImage = img[:, :, 0]

2. Binarize the images by applying a threshold. For example, to get a binary image (consisting of zeros and ones only) that has a 1 anywhere the red value is greater than 125, you would use the command like (please note that this is just an example which may or may work. Also you may need to use np.logical\_and() ):

redBinary = redImage > 125;

3. Label connected components in the binary images using region growing. (We looked at region growing on slide 20 of the lecture slide 24.) The function measure.label()will do this calculation for you and produce a tagged image and the number of tags (n) used (you can find more information here: <https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label>):

from skimage import measure

redTagged, redN = measure.label(redBinary, neighbors = 8, return\_num = True)

Where neighbors is either 4 (for 4-connected regions) or 8 (for 8-connected regions). Although on the documentation it mentioned the argument is deprecated, you can still feel free to use it for the purpose of this assignment. If you found the proper threshold in step 2, you will find 2 regions for each color.

4. Extract a binary image showing the location of each tagged region. For example, to get the binary image of the region with tag 1, you could type:

yellow\_ball = yellowTagged == 1

5. Compute the boundary (the maximum and minimum row and column for the tagged region) and the centroid (the average row and column position of each pixel in the tagged region).

**Please answer the following questions.**

Question 1. Please provide a description of how your code works (5 points)

(it is exactly as the steps above)

Question 2. Please complete the table below. Please round to the nearest integar. (5 points)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Object | Centroid | | Maximum | | Minimum | |
| row | col | row | Col | row | col |
| Yellow ball (at left) | 24 | 41 | 39 | 58 | 9 | 25 |
| Green block (at left) | 70 | 50 | 96 | 57 | 43 | 36 |
| Yellow block (at left) | 101 | 48 | 112 | 79 | 89 | 16 |
| Red ball (in center) | 45 | 99 | 69 | 127 | 21 | 72 |
| Blue block (in center) | 102 | 97 | 129 | 114 | 75 | 80 |
| Green block (in center) | 136 | 96 | 145 | 134 | 125 | 60 |
| Red ball (at right) | 53 | 159 | 78 | 184 | 30 | 134 |
| Blue block (at right) | 110 | 160 | 136 | 177 | 84 | 144 |

Question 3. Please provide you code below (either pictures or copy/paster you code would work. You can free to use all the spaces below till the end of page 10. )

import numpy as np

from skimage import io

from skimage import filters

from skimage import measure

def get\_centroid\_max\_min(grey\_img, description):

row, col = np.shape(grey\_img)

row\_min, row\_max = row, 0

col\_min, col\_max = col, 0

row\_total, col\_total = 0, 0

num = 0

for i in range(row):

for j in range(col):

if grey\_img[i, j]:

row\_min = min(row\_min, i)

row\_max = max(row\_max, i)

col\_min = min(col\_min, j)

col\_max = max(col\_max, j)

row\_total += i

col\_total += j

num += 1

print(description, " centroid: [", str(row\_total/num), ", ", str(col\_total/num), "]")

print(description, " max: [", str(row\_max), ", ", str(col\_max), "]")

print(description, " min: [", str(row\_min), ", ", str(col\_min), "]")

print("==================================================")

return [row\_total/num, col\_total/num], [row\_max, col\_max], [row\_min, col\_min]

# step 1

img = io.imread('object.jpg')

#io.imshow(img)

#io.show()

redImage = img[:, :, 0]

greenImage = img[:, :, 1]

blueImage = img[:, :, 2]

# step 2

redBinary = np.logical\_and(redImage > 125, greenImage < 100)

greenBinary = np.logical\_and(np.logical\_and(greenImage > 125, redImage < 100), blueImage < 100)

yellowBinary = np.logical\_and(greenImage > 125, redImage > 125)

blueBinary = blueImage > 150

# step 3

redTagged, redN = measure.label(redBinary, neighbors = 8, return\_num = True)

greenTagged, greenN = measure.label(greenBinary, neighbors = 8, return\_num = True)

yellowTagged, yellowN = measure.label(yellowBinary, neighbors = 8, return\_num = True)

blueTagged, blueN = measure.label(blueBinary, neighbors = 8, return\_num = True)

# step 4

yellow\_ball = yellowTagged == 1

green\_block\_left = greenTagged == 1

yellow\_block = yellowTagged == 2

red\_ball\_center = redTagged == 1

blue\_block\_center = blueTagged == 1

green\_block\_center = greenTagged == 2

red\_ball\_right = redTagged == 2

blue\_block\_right = blueTagged == 2

# step 5

get\_centroid\_max\_min(yellow\_ball, "Yellow Ball")

get\_centroid\_max\_min(green\_block\_left, "Green Block(left)")

get\_centroid\_max\_min(yellow\_block, "Yellow Block")

get\_centroid\_max\_min(red\_ball\_center, "Red Ball(Center)")

get\_centroid\_max\_min(blue\_block\_center, "Blue Block(Center)")

get\_centroid\_max\_min(green\_block\_center, "Green Block(Center)")

get\_centroid\_max\_min(red\_ball\_right, "Red Ball(Right)")

get\_centroid\_max\_min(blue\_block\_right, "Blue Block(Right)")