CPSC 470/570 – Artificial Intelligence
Problem Set #8 – Computer Vision
25 points
Due Friday April 26th, 11:59:59pm

Some reminders:

- **Grading contact:** Camilla Gu (camilla.gu@yale.edu) 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. The other library you may need is numpy, which was used in ps6.

Problem #1: Edge Detection (10 Points)

We have provided an image "yale.png" along with this problem set. 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 a numpy array with dimensions of **(H, W, 3)** 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 then 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 (**H, W)**. More information on rgb2gray can be found here:
https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_rgb_to_gray.html

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

- Sobel operator
- Robert's cross
- the Canny edge detector

Here are the 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)
  ```

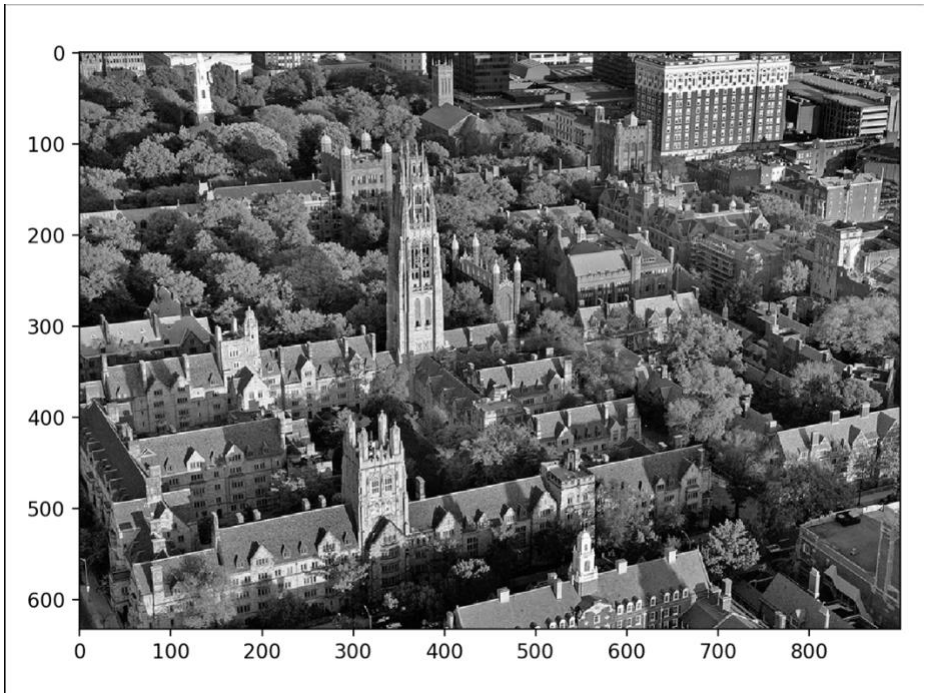
Robot's Cross

- 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)
  ```

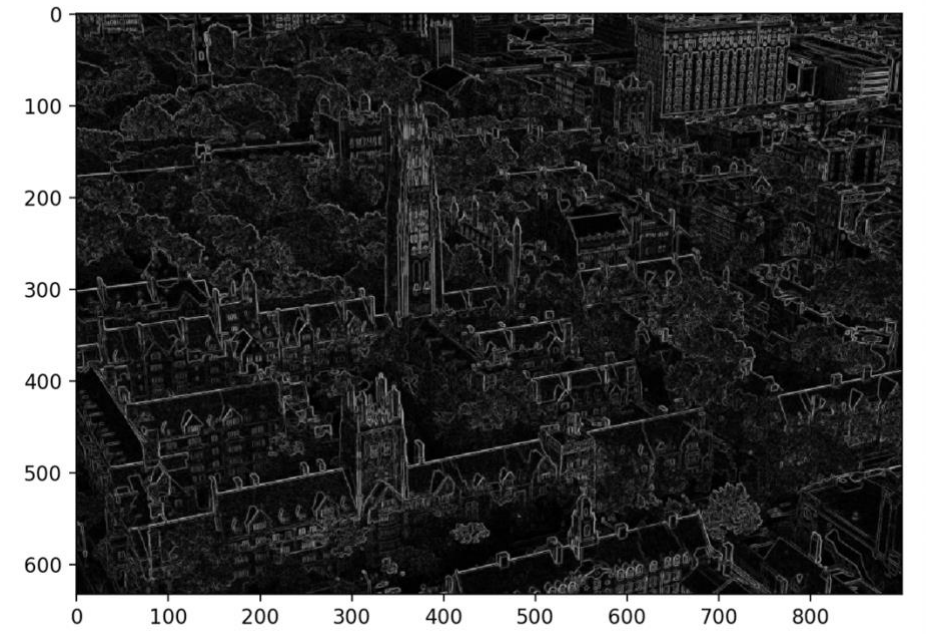
The Canny Edge Detector

- Documentation:
  https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.canny
- For this operator, 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)
  ```

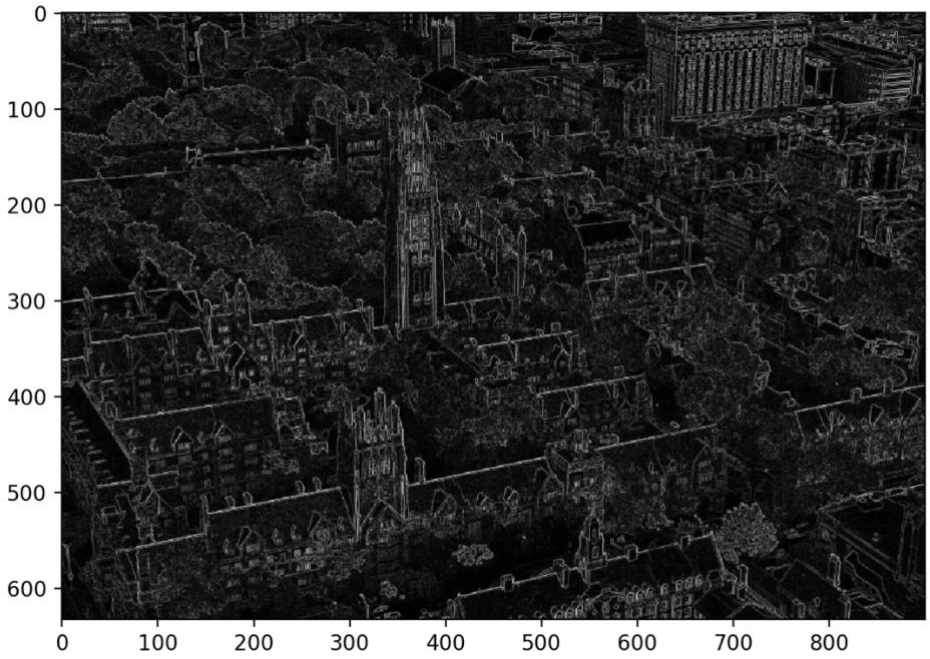
Question 1. Insert below pictures of your greyscale original image and the three edge images. Please scale each image to be roughly half of the page, and clearly label each (4 points).



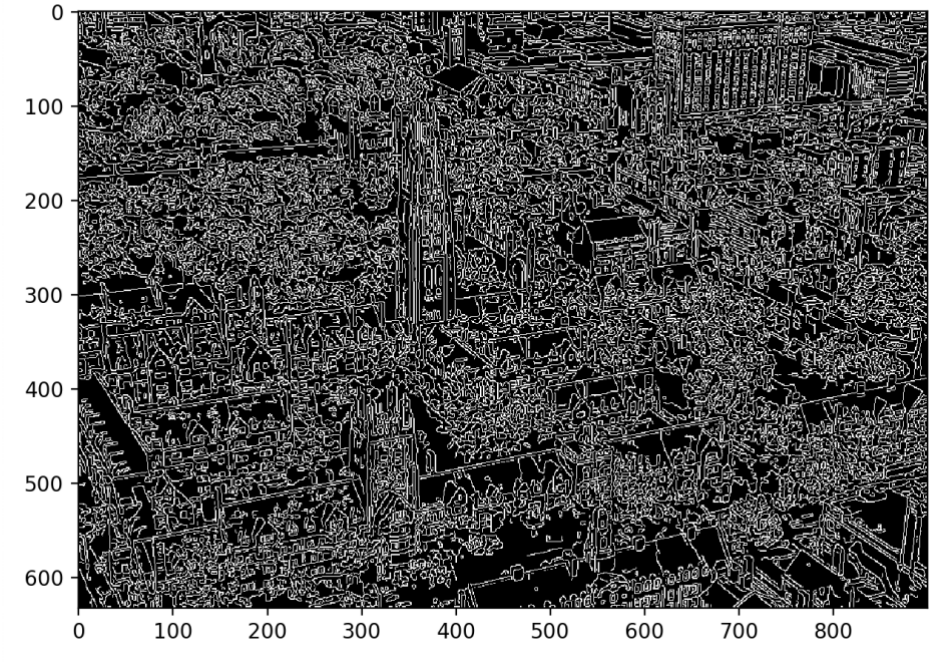
Original Image (Greyscale)



Sobel Operator



Robot's Cross



The Canny Edge Detector

Question 2. Answer the following questions regarding the Canny edge detection method.

a) what impact does the "low_threshold" have on the image? (1 point)

low_threshold decides whether an edge pixel gets rejected or not. If an edge pixel's gradient value is smaller than low_threshold, it will be suppressed.

b) what impact does the "high_threshold" have on the image? (1 point)

high_threshold decides whether an edge pixel gets accepted or not. If an edge pixel's gradient value is higher than high_threshold, it is marked as a strong edge pixel and will be accepted.

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

sigma is the standard deviation of the Gaussian filter. The purpose of sigma is to remove some of the noise before further processing the image, in order to prevent false detection caused by noise.

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

The Canny edge detector gives the best result.

This is because in the Canny edge detector, the parameter sigma can be tuned to control the amount of noise to be removed, which affects the level of details in the final output image. With a set of carefully chosen parameters (low_threshold, high_threshold and sigma), the Canny edge detector is able to pick out the boundaries of major obstacles without providing too many details.

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 problem set folder).

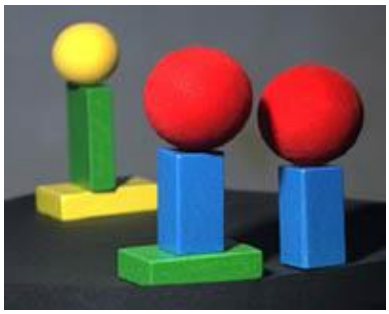


Figure 1

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.

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

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 not 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 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 the documentation mentions 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)

1. Read in the original image, and obtain the 3 RGB color-channel images.

2. Extract the binary images by applying some thresholds. Check that the binary images reflect the actual regions of the objects with each particular color (red, green, blue and yellow).

3. For each of the binary images, label the connected components. Check that the number of regions equals 2 for each particular color.

4. For each of the binary images and each of its 2 regions, create a row index array and a column index array, and output (avg. row, avg. col, max row, max column, min row, min column).

Question 2. Please complete the table below (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)	71	50	96	57	43	34
Yellow block (at left)	101	48	112	79	89	16
Red ball (in center)	45	99	69	127	21	72
Blue block (in center)	103	97	129	114	75	80
Green block (in center)	135	96	145	134	125	60
Red ball (at right)	53	159	78	184	30	134
Blue block (at right)	110	161	136	177	84	144

Question 3. Please copy and paste your code below. (5 points)

```
import numpy as np
from skimage import io
from skimage import measure

# original image
img = io.imread('object.jpg')
io.imshow(img)
io.show()

# binary images
redImage = img[:, :, 0]
greenImage = img[:, :, 1]
blueImage = img[:, :, 2]

redBinary = np.logical_and(redImage > 130, np.logical_and(greenImage < 100,
blueImage < 130))
greenBinary = np.logical_and(redImage < 121, np.logical_and(greenImage > 125,
blueImage < 130))
blueBinary = np.logical_and(redImage < 100, np.logical_and(greenImage < 170,
blueImage > 140))
yellowBinary = np.logical_and(redImage > 125, np.logical_and(greenImage > 125,
blueImage < 125))

io.imshow(redBinary)
io.show()
io.imshow(greenBinary)
io.show()
io.imshow(blueBinary)
io.show()
io.imshow(yellowBinary)
io.show()
```

```
# connected components (region growing)
redTagged, redN = measure.label(redBinary, neighbors = 8, return_num = True)
greenTagged, greenN = measure.label(greenBinary, neighbors = 8, return_num = True)
blueTagged, blueN = measure.label(blueBinary, neighbors = 8, return_num = True)
yellowTagged, yellowN = measure.label(yellowBinary, neighbors = 8, return_num =
True)

print(redN)
print(greenN)
print(blueN)
print(yellowN)

# compute the boundary (max/min row/col) and centroid (average row/col)
def computeRegionStatistics(taggedImage, region):
        rows = []
        cols = []
        for i in range(len(taggedImage)):
                for j in range(len(taggedImage[0])):
                        if taggedImage[i][j] == region:
                                rows.append(i)
                                cols.append(j)
        return round(sum(rows)/len(rows), 2), round(sum(cols)/len(cols), 2), max(rows),
max(cols), min(rows), min(cols)

print(computeRegionStatistics(redTagged, 1))
print(computeRegionStatistics(redTagged, 2))
print(computeRegionStatistics(greenTagged, 1))
print(computeRegionStatistics(greenTagged, 2))
print(computeRegionStatistics(blueTagged, 1))
print(computeRegionStatistics(blueTagged, 2))
print(computeRegionStatistics(yellowTagged, 1))
print(computeRegionStatistics(yellowTagged, 2))
```