CPSC 470/570 – Artificial Intelligence
Problem Set #8 – Computer Vision
25 points
Due Wednesday April 28th, 10:30AM

Some reminders:

- **Grading contact:** Nick Georgiou (nicholas.georgiou@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 **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 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 **HxW**. 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).

Figure 1. Greyscale original image

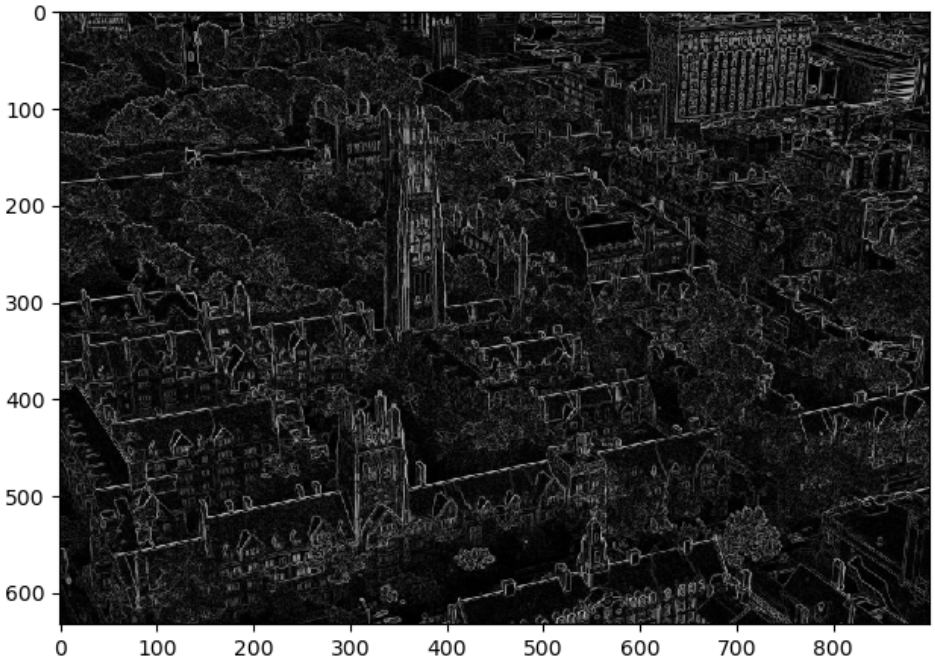

Figure 2. Sobel Operator

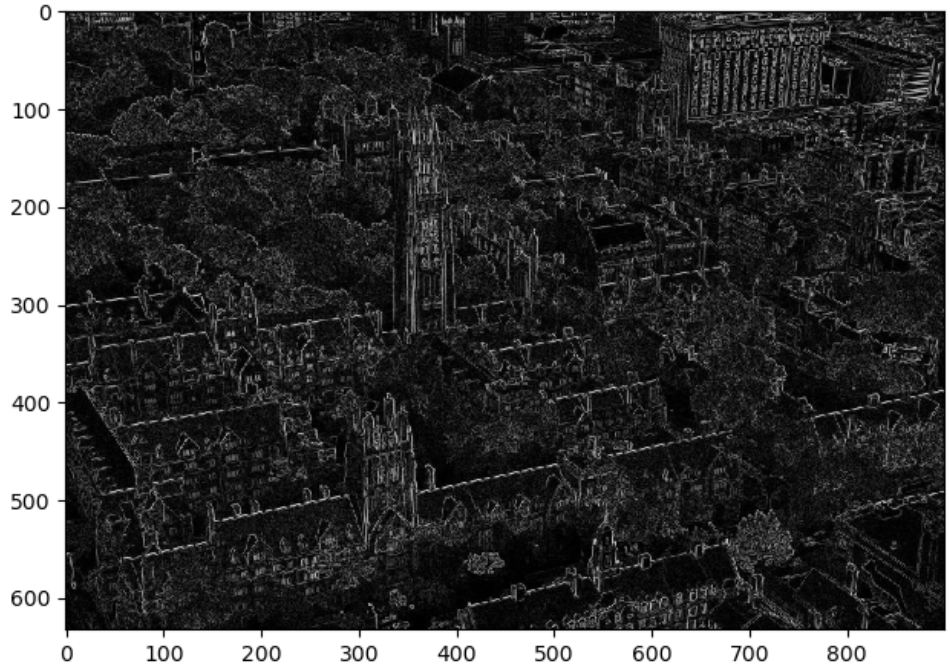

Figure 3. Robot's Cross

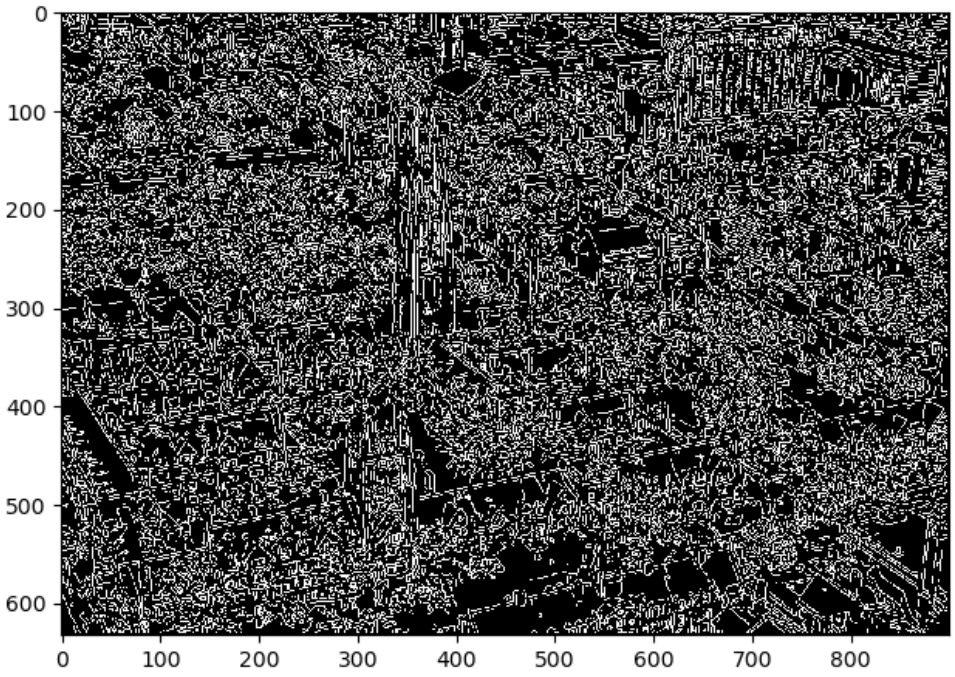

Figure 4. The Canny Edge Detector

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)

As the low threshold increases, the spurious edge (caused by noise) gradually disappears. This is because the edges with gradients below the low threshold are directly removed.

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

As we increase high threshold, fewer pixels are kept as true edges. This is because the high threshold controls the amount of true edges. The pixels with gradient larger than the high threshold are directly kept as true edges.

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

The larger the sigma is, the smaller the noises and the fewer the edges are. This is because the first step of Canny edge detection algorithm is to apply Gaussian filter which uses sigma. The larger the sigma is, the smoother the image. So the noises get smooth out and removed.

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)

Canny edge detector probably works best if the parameters are carefully tuned. Since it use multiple thresholds and hysteresis, it can remove most noisy edges while keeping the true edges. This will help the robot to detect the boundary of objects.

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).

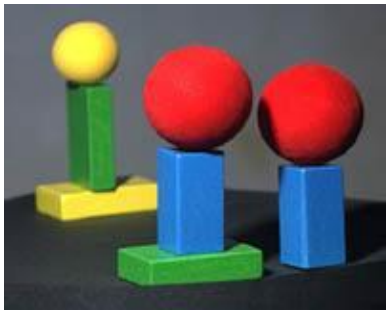

Figure 1

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 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)

First, I read the image object.jpg. Then I followed steps in the questions instruction as follows.

1. Divided the color image into three separate color-channel images.
2. Binarized the images by applying a threshold. Manually tuned the parameters (threshold) for each color and make sure each region is a connected region and there is no noisy regions.
3. For each binary images obtained in step 2, labelled connected components in the binary images using region growing. Manually checked the number of regions = 2 for each color.
4. For each tagged image, for each region, I compute the binary image. Then I take the maximum along the columns to create a row vector, which is used to find the column indices of the region. Similarly, I can find the row indices of the regions. The minimum, maximum, and mean (centroid) of the region can be obtained from these indices.

Question 2. Please complete the table below (5 points)

Object	Centroid		Maximum		Minimum	
	row	col	row	Col	row	col
Yellow ball (at left)	24	41.5	39	58	9	25
Green block (at left)	69.5	47	96	57	43	37
Yellow block (at left)	100.5	47.5	112	79	89	16
Red ball (in center)	45.5	100	69	127	22	73
Blue block (in center)	102	97	129	114	75	80
Green block (in center)	135	97	145	134	125	60
Red ball (at right)	54.5	159	78	184	31	134
Blue block (at right)	110	160.5	136	177	84	144

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

```
# Problem 2: Finding Color Blobs
img = io.imread('object.jpg')
io.imshow(img)
io.show()

# 1. Divide the color image into three separate color-channel images
redImage = img[:, :, 0]
greenImage = img[:, :, 1]
blueImage = img[:, :, 2]

# 2. Binarize the images by applying a threshold
redBinary = np.logical_and(redImage > 125, greenImage < 64, blueImage < 64)
yellowBinary = np.logical_and(redImage > 125, greenImage > 125)
blueBinary = blueImage > 143
greenBinary = np.logical_and(np.logical_and(greenImage > 130, redImage < 90),
blueImage < 110)

# io.imshow(redBinary.astype(np.float64))
# io.show()
# io.imshow(yellowBinary.astype(np.float64))
# io.show()
# io.imshow(blueBinary.astype(np.float64))
# io.show()
# io.imshow(greenBinary.astype(np.float64))
# io.show()

# 3. Label connected components in the binary images using region growing
# e.g. redTagged - 0: background 1: first region 2: second region
redTagged, redN = measure.label(redBinary, connectivity = 2, return_num = True)
```

```
yellowTagged, yellowN = measure.label(yellowBinary, connectivity = 2, return_num =
True)
blueTagged, blueN = measure.label(blueBinary, connectivity = 2, return_num = True)
greenTagged, greenN = measure.label(greenBinary, connectivity = 2, return_num = True)

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

# 4. Extract a binary image showing the location of each tagged region
# yellow1 = yellowTagged == 1 # return a T/F array
# yellow2 = yellowTagged == 2
# green1 = greenTagged == 1
# green2 = greenTagged == 2
# blue1 = blueTagged == 1
# blue2 = blueTagged == 2
# red1 = redTagged == 1
# red2 = redTagged == 2

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

def get_regions_rows_cols(tagged_img, region_idx):
  binary = tagged_img == region_idx
  row_vec = binary.max(0) # the row vec corresponds to the max element in each column
  cols = np.flatnonzero(row_vec)

  col_vec = binary.max(1)  # the col vec corresponds to the max element in each row
  rows = np.flatnonzero(col_vec)

  col_min = cols.min()
  col_max = cols.max()
  row_min = rows.min()
  row_max = rows.max()
  col_centroid = cols.mean()
  row_centroid = rows.mean()
  print(row_centroid, col_centroid, row_max, col_max, row_min, col_min)
  return row_centroid, col_centroid, row_max, col_max, row_min, col_min

get_regions_rows_cols(yellowTagged, 1)
get_regions_rows_cols(yellowTagged, 2)
get_regions_rows_cols(greenTagged, 1)
get_regions_rows_cols(greenTagged, 2)
get_regions_rows_cols(blueTagged, 1)
```

get_regions_rows_cols(blueTagged, 2)
get_regions_rows_cols(redTagged, 1)
get_regions_rows_cols(redTagged, 2)