HW 5 Lab Report

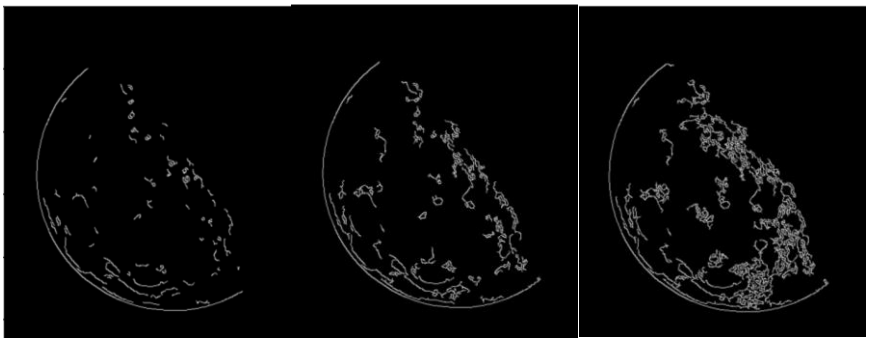
Jacob Alongi

Part 1: **Canny Edge detection**

- load an image
- Convolve the image with a Gaussian

```
img = cv2.GaussianBlur(img, (3, 3), 0)
```

I did do this in the previous assignment calling a singular method is easier
(3,3) sets the kernel size, 0 is sigma

- Set min an max threshold and apply edge detection
- 

Edge Threshold(100,200) (50,200) (20,200)

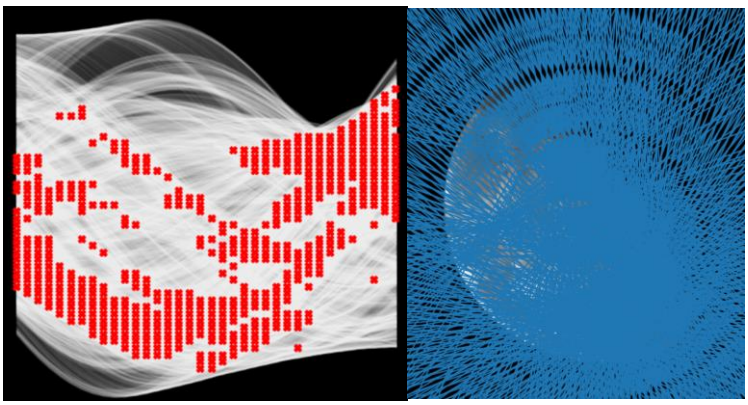
- Set a Global Threshold for accumulator and use for debugging

  The threshold is used as a cutoff for the counter, if below the threshold, the line is not strong enough and will not be considered for line detection.

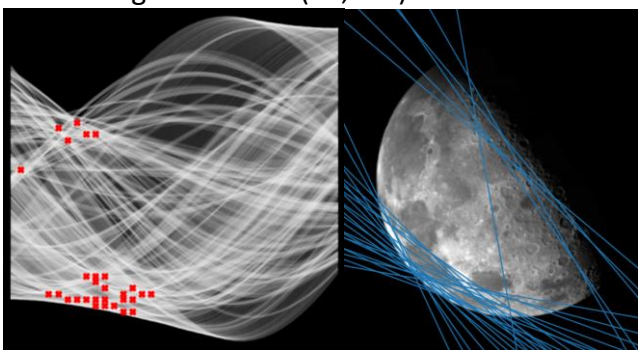
For each of the images if using the default Threshold (100,200) for edge detection, some of the example images will have a lot potential lines, causing the hough space diagram to become white and hard to discern. But keeping this flat makes it easier to set the threshold for the Accumulator.

```
#building = 1500
#campus = 400
#castle = 300
```

```
#road = 600
#rock = 1750
#roof-top = 220
```



Edge Threshold(20,200) Accumulator threshold 220



Edge Threshold (100,200) Accumulator Threshold, 150

Notice the cleaner hough space on the left, The threshold for the Accumulator had to be dropped since at +80 min threshold it only detected one line.
The point is altering the min edge thresh will produce cleaner hough space, but Accumulator threshold has much more control over the congestion in the line detection

Part 2: **Line Detection:**

- Take the edge image, the number of degrees you want to jump when looking for lines, the range of Rho and range of theta
- Create a 2D array called the accumulator representing the Hough Space with dimension (num_rhos, num_thetas) and initialize all its values to zero.
  ```
  accumulator = np.zeros((len(rho_array), len(rho_array)))
  ```
- For every pixel on the edge image, check whether the pixel is an edge pixel.

```
for y in range(edge_height):
    for x in range(edge_width):
        # binary image, check only non zero pixels
        # check whether the pixel is an edge pixel
        if edge_image[y][x] != 0:
```

- If it is an edge pixel, loop through all possible values of θ

```
for theta_val in range(len(theta_array)):
```

- calculate the corresponding ρ, find the θ and ρ index in the accumulator, and increment the accumulator base on those index pairs.

```
rho = (edge_point[1] * cos_thetas[theta_val]) + (edge_point[0] * sin_thetas[theta_val
theta = theta_array[theta_val]
rho_val = np.argmin(np.abs(rho_array - rho))
```

```
accumulator[rho_val][theta_val] += 1
```

- store rho in theta value into two separate arrays that will be used to generate the hough space diagram

```
    accumulator[rho_val][theta_val] += 1
    ys.append(rho)
    xs.append(theta)
plot1.plot(xs, ys, color="white", alpha=0.05)
```

Hough space diagrams are inverted on the x and y axis

```
plot1.invert_yaxis()
plot1.invert_xaxis()
```

Return the Accumulator, Rho, and Theta arrays that are used to compute all possible locations of the line

```
return accumulator, rho_array, theta_array
```

I used a method to help decide where the accumulator threshold is
(takes in any array of greyscale pixel values, and an optional threshold)
Prints the format to show its index location in stdout.
If the pixel value is below the threshold it will not be printed to see how many above are being reported, if to many just look at the values and raise the threshold

Threshold set to low, to many lines detected at this threshold

A lot less lines detected, line display will not be as congested

Part 3: **Accumulator:**

Takes a threshold, the original image, accumulator array returned from line detection, and the rho and theta array

```
check_accumulator(threshold_v, img, accumulator, rho_array, theta_array)
```

- Loop through all the values in the accumulator. If the value is larger than a certain threshold:

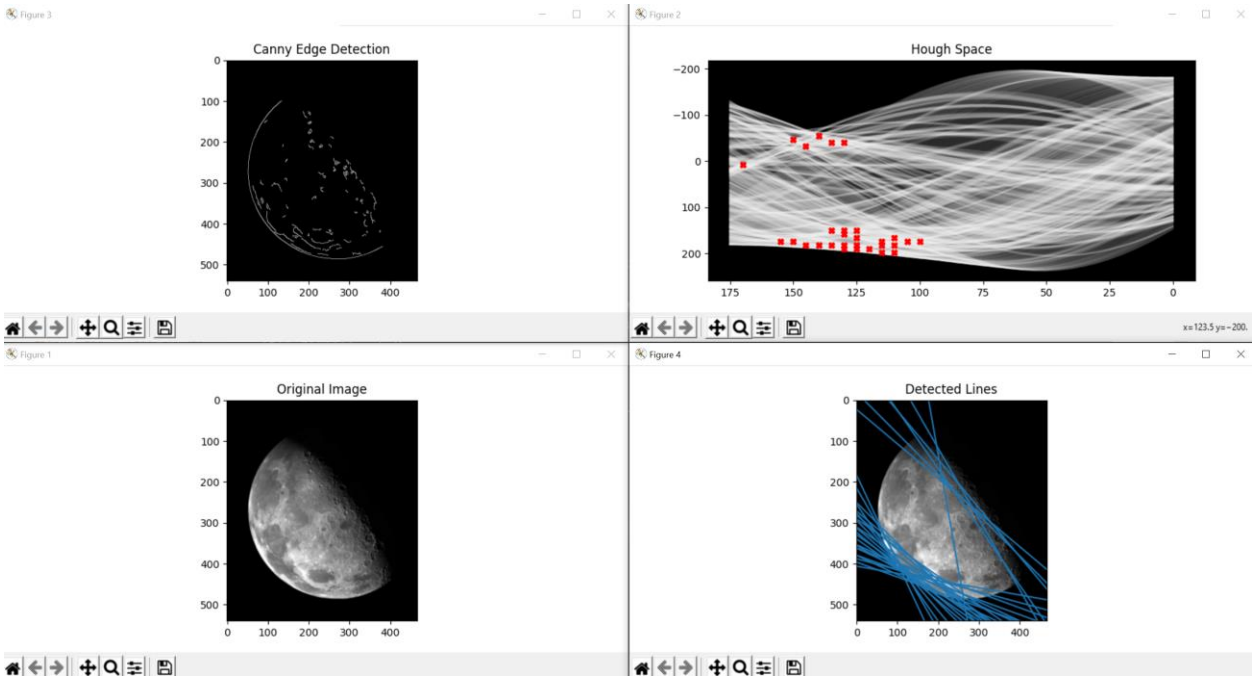
```
for i in range(accumulator.shape[0]):
    for j in range(accumulator.shape[1]):

        # Loop through all the values in the accumulator.
        # If the value is larger than a certain threshold:
        if accumulator[i][j] > threshold:
```

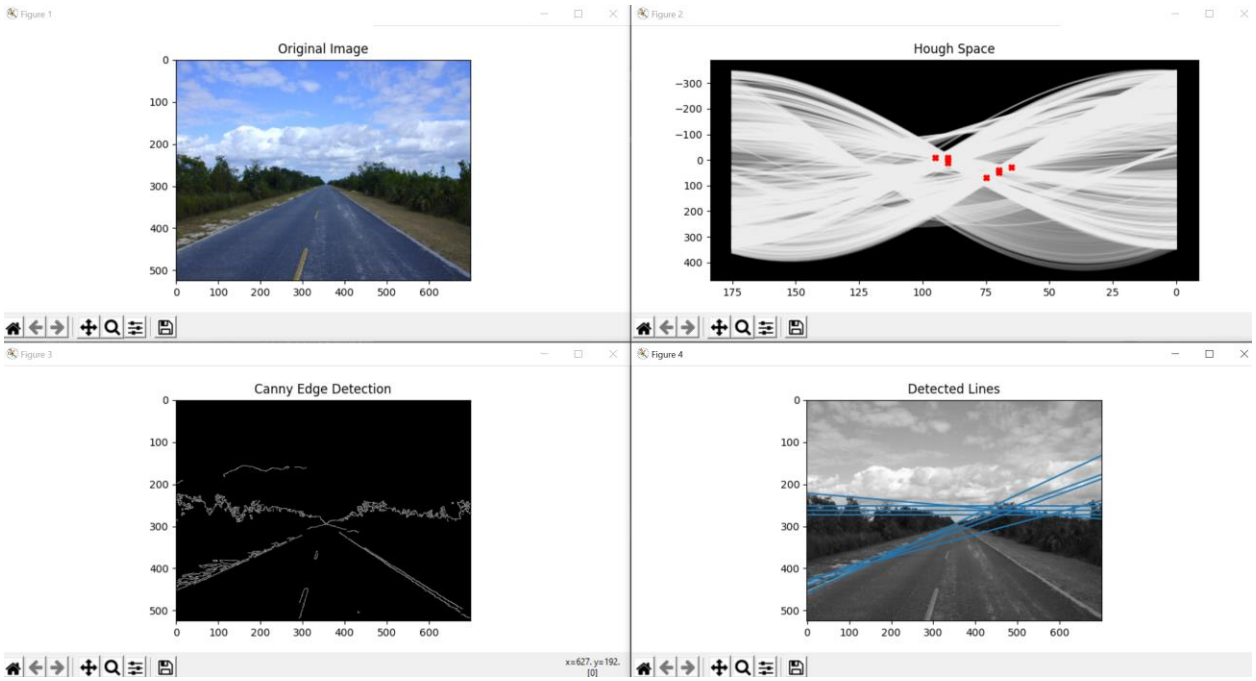
- get the ρ and θ index, get the value of ρ and θ from the index pair which can then be converted back to the form of y = ax + b

```
rho = rho_array[i]
theta = theta_array[j]
# get the ρ and θ index, get the value of ρ and θ from the index pair
# which can then be converted back to the form of y = ax + b.
a = np.cos(np.deg2rad(theta))
b = np.sin(np.deg2rad(theta))
x = (a * rho) + width_half
y = (b * rho) + height_half
x1 = int(x + 1000 * (-b))
y1 = int(y + 1000 * (a))
x2 = int(x - 1000 * (-b))
y2 = int(y - 1000 * (a))
plot1.plot([theta], [rho], marker='X', color="red")
plot2.add_line(mlines.Line2D([x1, x2], [y1, y2]))
```

Need to go back to the Hough lines plot and add the markers at the detected lines locations.

Plot 2 shows the lines detected overlayed on top of the original image

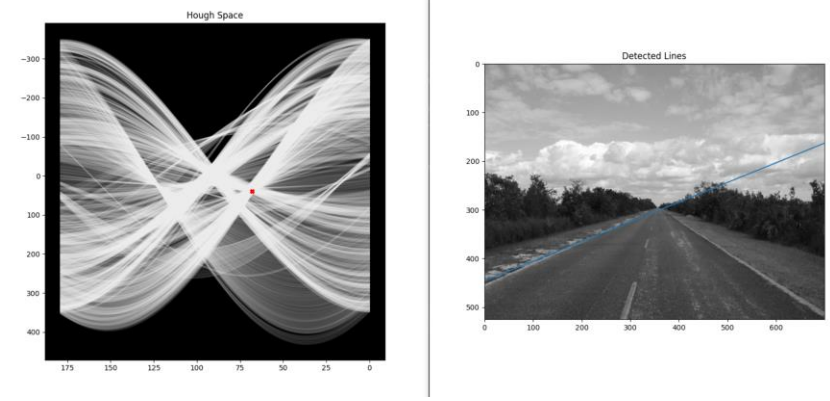




We can also change the parameters to find the strongest line in the image, currently theta steps by 5 degrees, Edge threshold (100, 200) Accumulator threshold 600

Changing to 1 degree, and changing the accumulator check from > threshold to the max value to:

```
if accumulator[i][j] > threshold:
# test method to find strongest line in image
if accumulator[i][j] == np.amax(accumulator):
```



Code:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import os
```

```
"""
@parameters detect_lines
edge_image: the edge detection image
num_rhos: 180 range of values to use for rho
num_thetas: 180 range of values to use for theta
"""
def detect_lines(edge_image, step_degrees, num_rhos, num_thetas):

    edge_height, edge_width = edge_image.shape[:2]
    edge_height_half, edge_width_half = edge_height / 2, edge_width / 2


    d = np.sqrt(np.square(edge_height) + np.square(edge_width))
    # print(d)



    drho = (2 * d) / num_rhos
    # print("dhro", drho)
    # Array of directions 1 - 180, only need to look 180 degrees
    theta_array = np.arange(0, num_thetas, step=step_degrees)

    rho_array = np.arange(-d, d, step=drho)
    # print("rhos", rho_array)

    #
    cos_thetas = np.cos(np.deg2rad(theta_array))
    # print("cos_thetas", cos_thetas)
    sin_thetas = np.sin(np.deg2rad(theta_array))
    #
    # print("sin_thetas", sin_thetas)
```

```
# Create a 2D array called the accumulator representing the Hough Space
# with dimension (num_rhos, num_thetas) and initialize all its values to zero.

# accumulator is 180x180 array
accumulator = np.zeros((len(rho_array), len(rho_array)))

# print("rhos length", len(rho_array))

global plot1
plot1 = figure.add_subplot(1, 1, 1)
plot1.set_facecolor((0, 0, 0))
plot1.title.set_text("Hough Space")

# print("edge_height_half", edge_height_half)
# print("edge_width_half", edge_width_half)

# For every pixel on the edge image
# iterate through the edge image
```

```
for y in range(edge_height):
    for x in range(edge_width):
        # binary image, check only non zero pixels
        # check whether the pixel is an edge pixel
        if edge_image[y][x] != 0:

            edge_point = [y - edge_height_half, x - edge_width_half]

            ys, xs = [], []
            # If it is an edge pixel, loop through all possible values of θ, calculate the corresponding ρ,
            # check every angle from 1 - 180
            for theta_val in range(len(theta_array)):
                # find the θ and ρ index in the accumulator, and increment the accumulator base on those index pairs.
                rho = (edge_point[1] * cos_thetas[theta_val]) + (edge_point[0] * sin_thetas[theta_val])
                theta = theta_array[theta_val]
                rho_val = np.argmin(np.abs(rho_array - rho))
                # print("rho_val", rho_val)
                # print("theta_val", theta_val)
                # add vote to accumulator
                accumulator[rho_val][theta_val] += 1
                ys.append(rho)
                xs.append(theta)
            plot1.plot(xs, ys, color="white", alpha=0.05)
plot1.invert_yaxis()
plot1.invert_xaxis()


print_save_display(accumulator, threshold_v)

return accumulator, rho_array, theta_array
```

```
"""
@parameters detect_lines
threshold_v: the threshold for the accumulator
img: original image to display lines on
accumulator: accumulator array givent from detect lines
rho_array: array of rho values from detect_lines
theta_array: array of theta values from detect_lines
"""
def check_accumulator(threshold, image, accumulator, rho_array, theta_array):

    figure2 = plt.figure(figsize=(8, 8))
    plot2 = figure2.add_subplot(1, 1, 1)
    plot2.imshow(image, cmap="gray")

    height, width = image.shape[:2]
    height_half, width_half = height / 2, width / 2
    # print(height, width)
```

```
# iterate through accumlator
for i in range(accumulator.shape[0]):
    for j in range(accumulator.shape[1]):

        # Loop through all the values in the accumulator.
        # If the value is larger than a certain threshold:
        # if accumulator[i][j] > threshold:
        # test method to find strongest line in image
        if accumulator[i][j] == np.amax(accumulator):
            rho = rho_array[i]
            theta = theta_array[j]
            # get the ρ and θ index, get the value of ρ and θ from the index pair
            # which can then be converted back to the form of y = ax + b.
            a = np.cos(np.deg2rad(theta))
            b = np.sin(np.deg2rad(theta))
            x = (a * rho) + width_half
            y = (b * rho) + height_half
            x1 = int(x + 1000 * (-b))
            y1 = int(y + 1000 * (a))
            x2 = int(x - 1000 * (-b))
            y2 = int(y - 1000 * (a))
            plot1.plot([theta], [rho], marker='X', color="red")
            plot2.add_line(mlines.Line2D([x1, x2], [y1, y2]))

plot2.title.set_text("Detected Lines")
```

```
# useful visualization of image, shows pixel value at pixel location, can only take up to 3 digit values
# threshold default 0, if raised will ignore index values lower than threshold
def print_save_display(img, threshold=0):
    saveFile = open("display.txt", "w+")
    width, height = img.shape
    for i in range(1, width - 1):
        for j in range(1, height - 1):
            if (img[i][j] != 0 and img[i][j] >= threshold):
                print('{:<3}'.format(img[i][j]), end=" ")
                saveFile.write('{:<4}'.format(img[i][j]))
            else:
                print("   ", end=" ")
                saveFile.write("    ")
        print()
        saveFile.write("\n")
    saveFile.close()
```

```python
# Function to change cartesian to polar
def cart2pol(x, y):
    rho = np.sqrt(x ** 2 + y ** 2)
    phi = np.arctan2(y, x)
    return (rho, phi)

# Function to change polar to cartesian
def pol2cart(rho, phi):
    x = rho * np.cos(phi)
    y = rho * np.sin(phi)
    return (x, y)

# shrink back down after thickening the lines
def erode(edge_image):
    edge_image = cv2.erode(
        edge_image,
        cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
        iterations=1
    )
    e1 = plt.figure()
    plt.imshow(edge_image, cmap="gray")
    plt.title('After erode')
    return edge_image

# use dilate to thicken the lines for easier edge detection
def dilate(edge_image):
    edge_image = cv2.dilate(
        edge_image,
        cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)),
        iterations=1
    )
    d1 = plt.figure()
    plt.imshow(edge_image, cmap="gray")
    plt.title('After Dilation')
    return edge_image
```

```python
# loads in an image, displays the original image then converts it to greyscale
def load(img_name):
    img = cv2.imread(img_name)
    l1 = plt.figure()
    # setting to make plt show original colors, default is distorted
    plt.imshow(img[:,:,::-1])
    plt.title('Original Image')
    # Load the image as greyscale
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img

# loads images into images array from a folder
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))
        if img is not None:
            images.append(img)

    return images
```

```python
def main():
    # load in the image
    img = load('5img/road.jpg')

    global figure
    figure = plt.figure(figsize=(6, 6))



    # images = load_images_from_folder('5img')
    # for image in images:

    # blur the image
    edge_image = cv2.GaussianBlur(img, (3, 3), 1)

    # use canny edge detection to create image of edges
    min_thresh = 100
    max_thresh = 200
    edge_image = cv2.Canny(edge_image, min_thresh, max_thresh)
    # display canny edge detection

    f1 = plt.figure()
    plt.imshow(edge_image, cmap="gray")
    plt.title('Canny Edge Detection')



    # use dilate to thicken the edges for easier edge detection
    edge_image = dilate(edge_image)
    # # use erode to thin lines
    # edge_image = erode(edge_image)
```

```
# threshold of voting is set by threshold currently = 150
global threshold_v
threshold_v = 700
#building = 1500
#campus = 400
#castle = 300
#road = 600
#rock = 1750
#roof-top = 220


"""
@parameters detect_lines
edge_image: the edge detection image
num_rhos: 180 range of values to use for rho
num_thetas: 180 range of values to use for theta
"""
# step_degrees is the number of degrees to step by when looking for lines point passes through
# increasing this value greatly improves processing speed
step_degress = 5.0
num_rhos = 180
num_theta = 180
accumulator, rho_array, theta_array = detect_lines(edge_image, step_degress, num_rhos, num_theta)
```

```
    """
    @parameters detect_lines
    threshold_v: the threshold for the accumulator
    img: original image to display lines on
    accumulator: accumulator array givent from detect lines
    rho_array: array of rho values from detect_lines
    theta_array: array of theta values from detect_lines
    """
    check_accumulator(threshold_v, img, accumulator, rho_array, theta_array)
    # once all steps are complete, show all images at once
    plt.show()


if __name__ == '__main__':
    main()
```