import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import numpy as np

import cv2

import math

image = mpimg.imread('homestreet3.jpg')

image = np.rot90(image, 3)

height=image.shape[0]

width=image.shape[1]

def region\_of\_interest(img, vertices):

mask = np.zeros\_like(img)

match\_mask\_color = 255

cv2.fillPoly(mask, vertices, match\_mask\_color)

masked\_image = cv2.bitwise\_and(img, mask)

return masked\_image

def draw\_lines(img, lines, color=[255, 0, 0], thickness=3):

if lines is None:

return img

img\_copy = np.copy(img)

line\_img = np.zeros(

(

img.shape[0],

img.shape[1],

3

),

dtype=np.uint8,

)

for line in lines:

for x1, y1, x2, y2 in line:

cv2.line(line\_img, (x1, y1), (x2, y2), color, thickness)

img\_copy = cv2.addWeighted(img\_copy, 0.8, line\_img, 1.0, 0.0)

return img\_copy

plt.figure()

plt.imshow(image)

plt.show()

gray\_image = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

region\_of\_interest\_vertices = [

(0, height//2),

(width//2, 0),

(width, height//2),

]

cropped\_gray = region\_of\_interest(gray\_image, np.array([region\_of\_interest\_vertices], np.int32))

plt.figure()

plt.imshow(cropped\_gray, cmap='gray')

plt.title('Cropped Grayscale Image')

plt.show()

cannyed\_image = cv2.Canny(gray\_image, 300, 600)

cropped\_canny = region\_of\_interest(cannyed\_image, np.array([region\_of\_interest\_vertices], np.int32))

plt.figure()

plt.imshow(cropped\_canny)

plt.title('Cropped Canny Edge Detection')

plt.show()

lines = cv2.HoughLinesP(

cropped\_canny,

rho=1,

theta=np.pi / 90,

threshold=200,

lines=np.array([]),

minLineLength=50,

maxLineGap=25

)

left\_line\_x = []

left\_line\_y = []

right\_line\_x = []

right\_line\_y = []

for line in lines:

for x1, y1, x2, y2 in line:

slope = (y2 - y1) / (x2 - x1) # <-- Calculating the slope.

if math.fabs(slope) < 0.5: # <-- Only consider extreme slope

continue

if slope <= 0: # <-- If the slope is negative, left group.

left\_line\_x.extend([x1, x2])

left\_line\_y.extend([y1, y2])

else: # <-- Otherwise, right group.

right\_line\_x.extend([x1, x2])

right\_line\_y.extend([y1, y2])

if left\_line\_y and right\_line\_y:

poly\_left = np.poly1d(np.polyfit(left\_line\_y, left\_line\_x, deg=1))

poly\_right = np.poly1d(np.polyfit(right\_line\_y, right\_line\_x, deg=1))

min\_y = int(height \* (2/ 5))

max\_y = height

left\_x\_start = int(poly\_left(max\_y))

left\_x\_end = int(poly\_left(min\_y))

right\_x\_start = int(poly\_right(max\_y))

right\_x\_end = int(poly\_right(min\_y))

line\_image = draw\_lines(

image,

[[

[left\_x\_start, max\_y, left\_x\_end, min\_y],

[right\_x\_start, max\_y, right\_x\_end, min\_y],

]],

thickness=5,

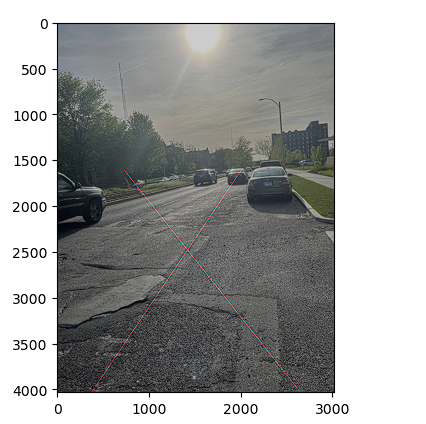
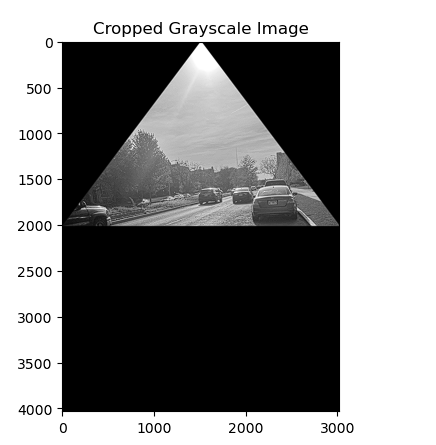
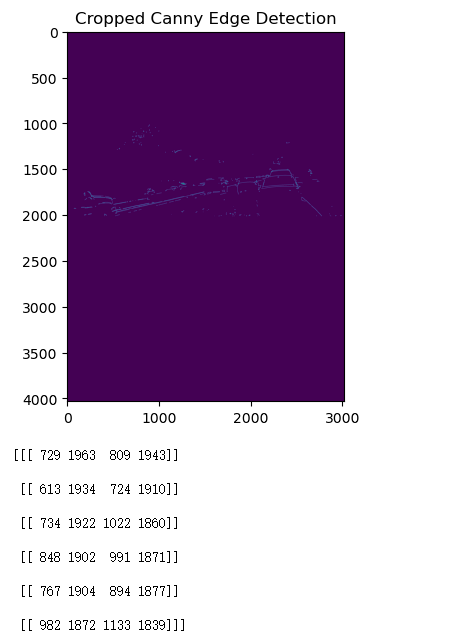
)

print(lines)

plt.figure()

plt.imshow(line\_image)

plt.show()



Change the region of interest to quadrilateral area

region\_of\_interest\_vertices = [

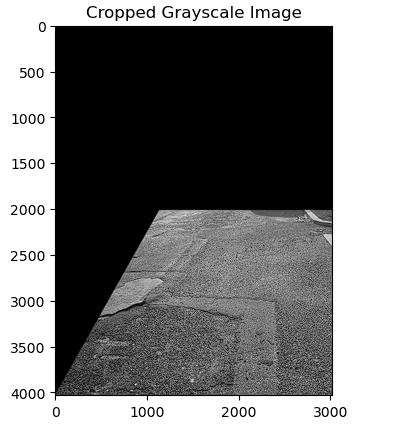
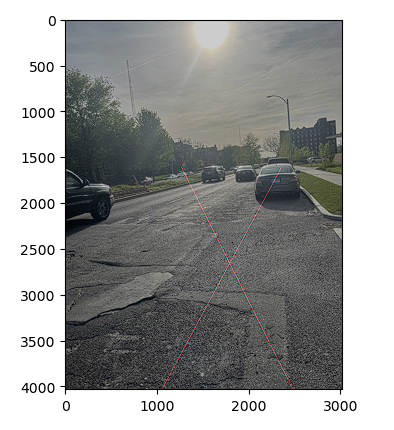
(0, height),

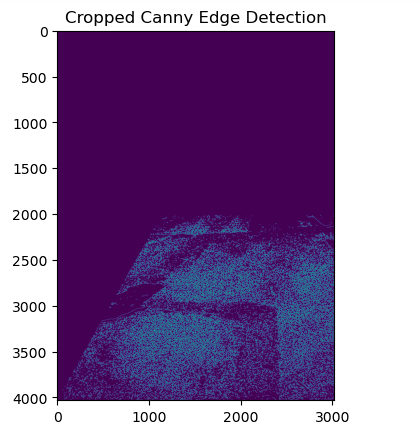
(width / 2 - width / 8, height / 2),

(width, height / 2),

(width, height),

]





Change the houghline and canny

lines = cv2.HoughLinesP(

cropped\_canny,

rho=4,

theta=np.pi / 90,

threshold=190,

lines=np.array([]),

minLineLength=50,

maxLineGap=25

)

cannyed\_image = cv2.Canny(gray\_image, 150, 300)

