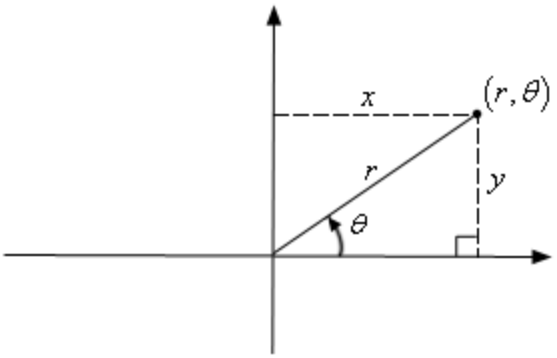
Hough Transform

(Notes From https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html)

a line in Cartesian space is usually represented with parameters (m, b). We may also apply the polar coordinate system and parameterize using (r, θ).



For Hough Transforms, we can express lines in the Polar Coordinate System. The equation for line Y can be expressed as

$$y = (-\frac{\cos\theta}{\sin\theta})\ x\ +\ (\frac{r}{\sin\theta})$$

- $x = r\cos\theta$
- $y = r\sin\theta$

we know that $x^2 + y^2 = r^2$ and so $r = \sqrt{x^2 + y^2}$. To obtain our angle θ, we apply

$$\frac{y}{x} = \frac{r\sin\theta}{r\cos\theta} = \tan\theta$$

so the inverse yields...

$$\theta = \tan^{-1}(\frac{y}{x})$$

we then construct a Hough accumulator matrix (H) in the following way:

- rows = different values of r
- columns = different values of θ

$$\begin{bmatrix} r_1\theta_1 & r_1\theta_2 & \dots \\ r_2\theta_1 & r_2\theta_2 & \ddots \end{bmatrix}$$

Since these (r, θ) values represent lines, points that are co linear will vote on the same values values of r and θ that correspond to their line in Hough Space.

Canny Edge Detection

We first apply Canny Edge Detection to find the edges in the given image. This follows an algorithm of blurring the image slightly and detecting areas with relatively high changes in their intensity gradients (a key property of edges). We can then sort these into real edges, potential edges, and non-edges. We then look at adjacent points on potential edges to see if they are similar.

Code

I implemented the the code to perform the above theory. the `detectLines()` function takes in the path to some image and then performs the hough lines detection. I had to tweak the sensitivity parameter of the `cv.HoughLines()` function for each image due to the distance each photo was taken from the road.

```
import cv2 as cv
import numpy as np
import math
import time



def detectLines(img_loc, sensitivity):

    # loading image
    img = cv.imread(img_loc, cv.IMREAD_GRAYSCALE) # convert to gray

        # canny edge dectetion
    canny = cv.Canny(img, 50, 200, None, 3)

    # save edges to be displayed later im final image
    img_edges = cv.cvtColor(canny, cv.COLOR_GRAY2BGR)
```

```
    # apply hough transform
    lines_detected = cv.HoughLines(canny, 1, np.pi /180, 125, None,
0, 0)

    # constructing accumulator matrix
    accum = []

    if lines_detected is not None:
        # make and draw lines
        for i in range(len(lines_detected)):
            rho_cur = lines_detected[i][0][0]
            theta_cur = lines_detected[i][0][1]

            # get x,y coordinates of each point in cartesian space
            a = math.cos(theta_cur)
            b= math.sin(theta_cur)
            x0 = a * rho_cur
            y0 = b * rho_cur

            # store values
            accum.append((rho_cur, theta_cur))

            # generate points and plot
            pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
            pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
            cv.line(img_edges, pt1, pt2, (15,230,230), 2,
cv.LINE_AA)

        print(f"Lines={len(accum)}")




    else:
        print("no lines")


    cv.imshow("Original:", img)
    cv.imshow("Detected Lines: {str(len(lines))}", img_edges)

    while True:
```

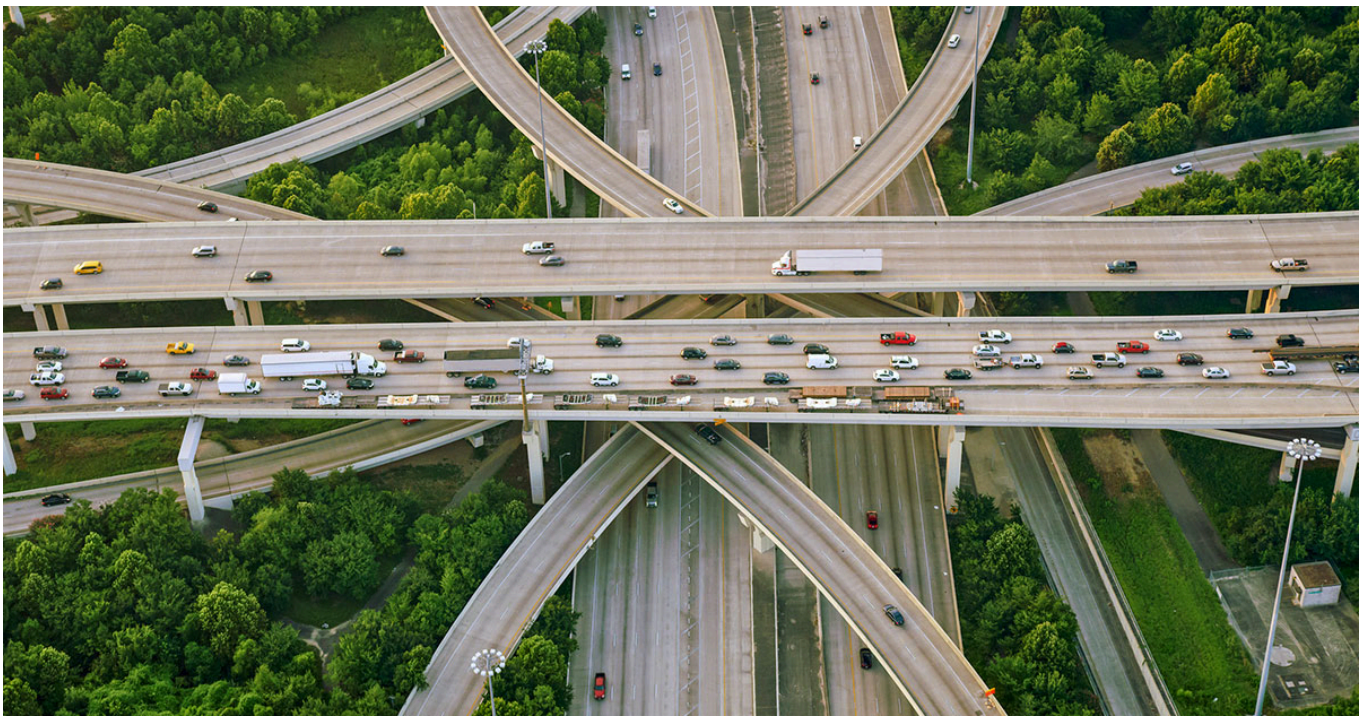
```
        k= cv.waitKey(1) & 0xFF
        if k== 27: # esc
            cv.destroyAllWindows()


if __name__ == "__main__":
    import os

    os.environ["QT_QPA_PLATFORM"] = "xcb" # linux issue
    IMG_LOC =
'/home/aaron/Projects/classWork/CMPSC497/learn_opencv/hw_labs/hough_
line_detection/test/highway.jpg'
    detectLines(IMG_LOC, sensitivity=440)
```

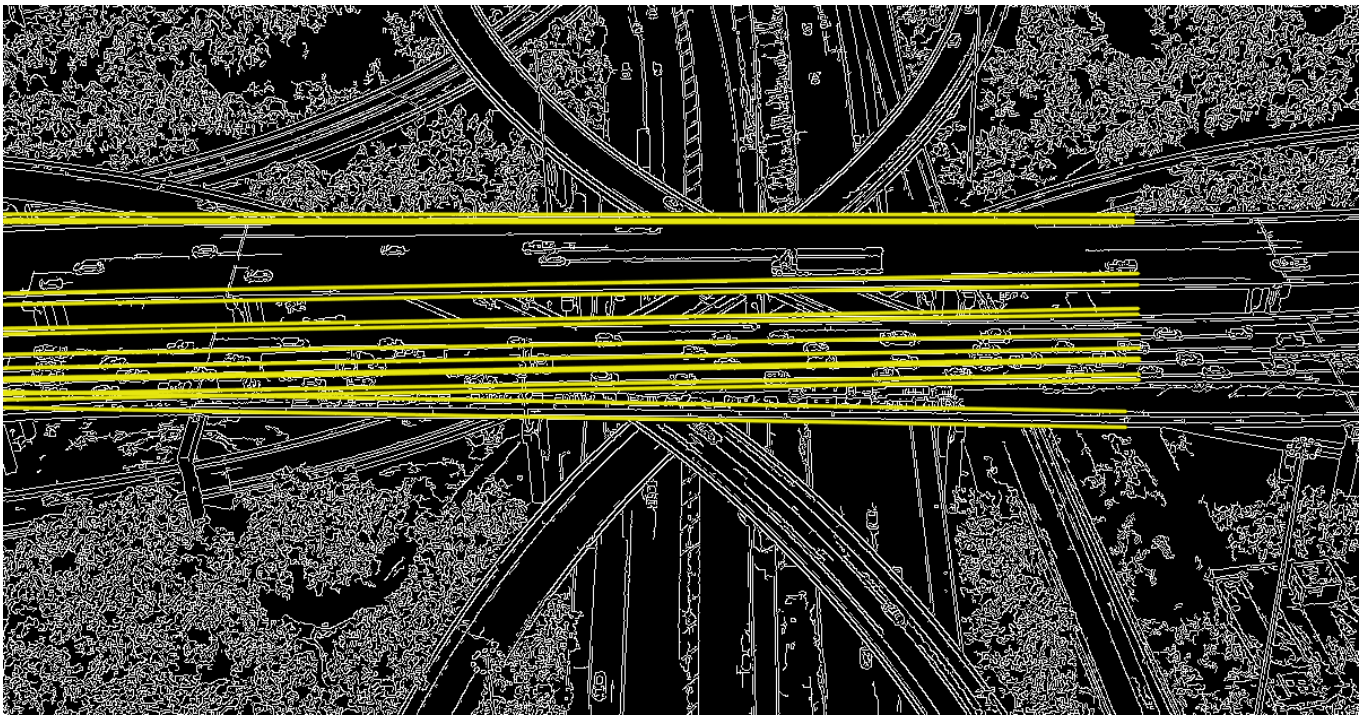
Images

Original Image



Gray Scale Image

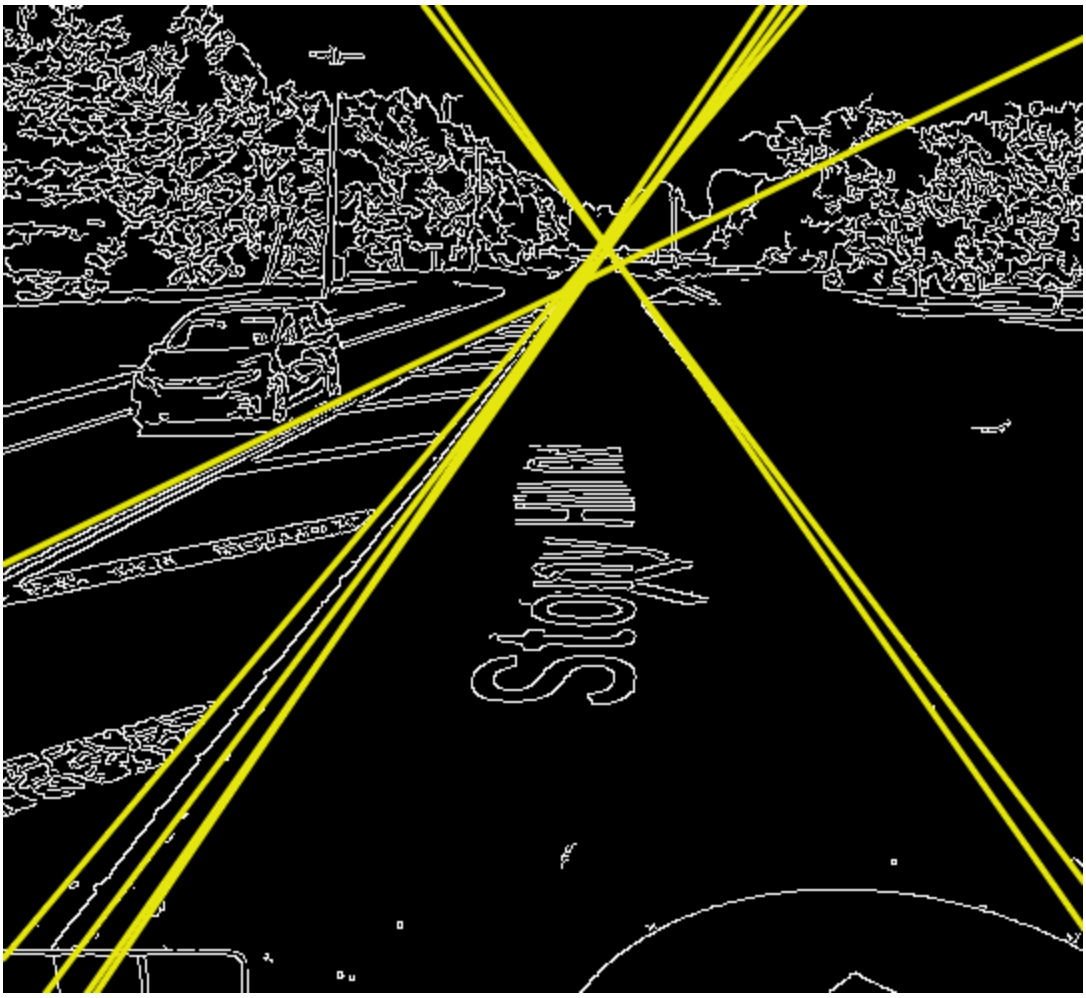
Canny Filtered and Lines Overlayed



Original Image

Gray scale Image

Canny Filtered and Lines Overlayed



Conclusion

Some issues I could note for using this for real road detection is the fact that the lines continue beyond the bounds of the road. I think this may be solved by cutting the frame at a certain height value -- meaning we'd only observe the road forward for a certain distance ahead. Another issue was the size of the road within the frame. For this application, a camera in a fixed position on a vehicle, this shouldn't be a problem. All in all, with some minor filtering and possibly applying some Gaussian blur this algorithm will serve this problem well.