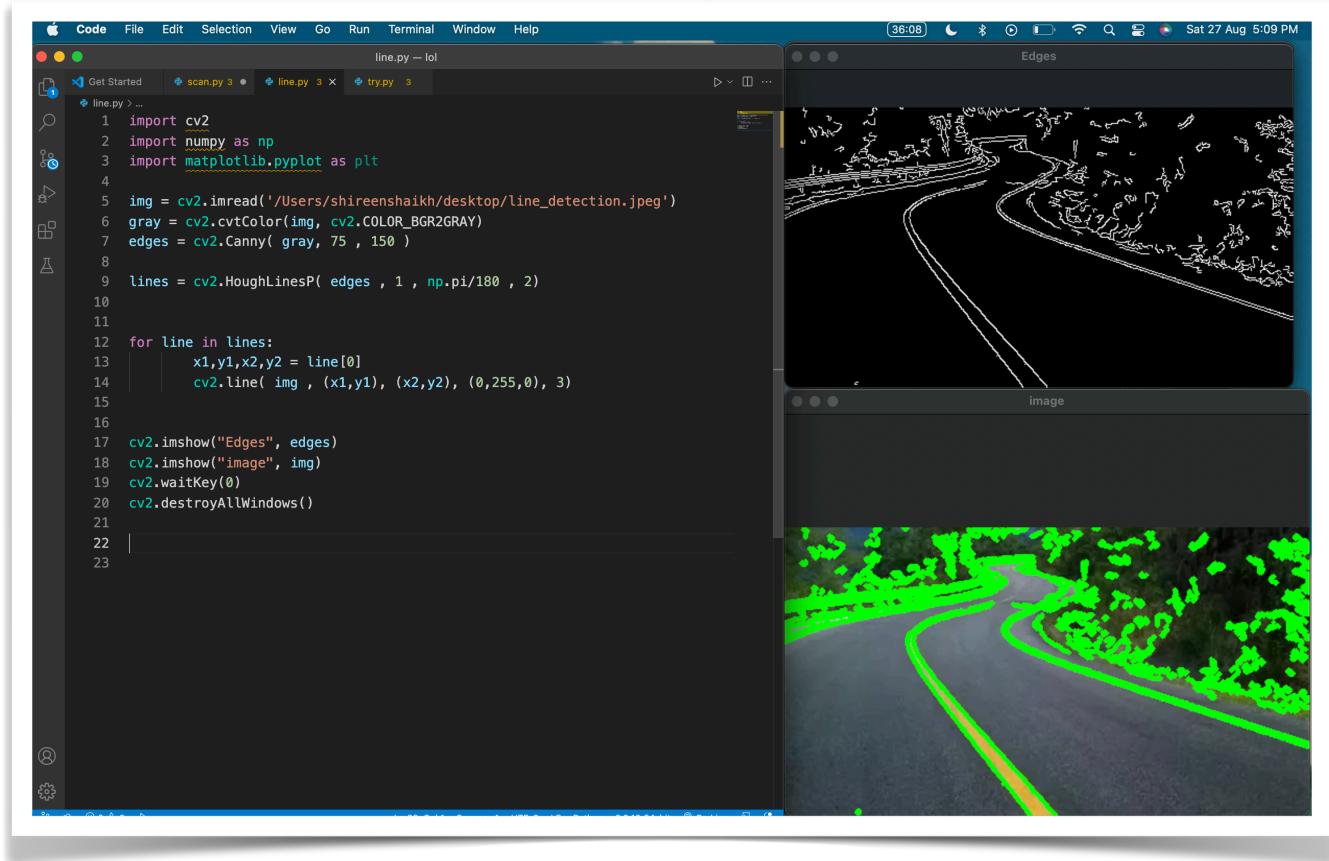


LINE DETECTION

- Without using maxLineGap and minLineLength, setting threshold to 2. Code and output.



The screenshot shows a Mac OS X desktop environment. On the left, a code editor window titled "line.py — lol" displays Python code for line detection using OpenCV. The code reads an image, converts it to grayscale, applies the Canny edge detector, and then uses the HoughLinesP function to find lines. It then draws these lines onto a copy of the original image. Two windows titled "Edges" and "image" are shown on the right, displaying the results. The "Edges" window shows a binary mask of detected edges in white on a black background. The "image" window shows the original image with green lines highlighting the detected road boundaries.

```
line.py > ...
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('/Users/shireenshaikh/desktop/line_detection.jpeg')
6 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7 edges = cv2.Canny( gray, 75 , 150 )
8
9 lines = cv2.HoughLinesP( edges , 1 , np.pi/180 , 2)
10
11
12 for line in lines:
13     x1,y1,x2,y2 = line[0]
14     cv2.line( img , (x1,y1), (x2,y2), (0,255,0), 3)
15
16
17 cv2.imshow("Edges", edges)
18 cv2.imshow("image", img)
19 cv2.waitKey(0)
20 cv2.destroyAllWindows()
21
22 |
23
```

2. Using maxLineGap and minLineLength, best I could adjust.

The screenshot shows a Jupyter Notebook environment with a code cell containing a Python script named `line.py`. The script uses OpenCV and NumPy to detect lines in a road image. It includes parameters for `minLineLength` and `maxLineGap` to filter the detected lines. The resulting image shows multiple green line segments drawn over the road, indicating the detected lanes. The notebook interface also shows other open files like `scan.py` and `try.py`.

```
line.py — lol
Get Started scan.py 3 line.py 3 try.py 3
line.py > ...
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('/Users/shireenshaikh/desktop/line_detection.jpeg')
6 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7 edges = cv2.Canny( gray, 75 , 150 )
8
9 lines = cv2.HoughLinesP( edges , 1 , np.pi/180 , 50 , minLineLength =
40, maxLineGap = 10)
10
11
12 for line in lines:
13     x1,y1,x2,y2 = line[0]
14     cv2.line( img , (x1,y1), (x2,y2), (0,255,0), 3)
15
16 cv2.imshow("image", img)
17 cv2.waitKey(0)
18 cv2.destroyAllWindows()
19
20
```

image

Ln 17 Col 15 Spaces: 4 LUTE 2 LE Python 3.9.10 64-bit @ Go Live ⌂ ⌂ ⌂

TRIED LANE DETECTION

The screenshot shows a development environment with two main panes. The left pane is a code editor titled "try.py — lol" containing Python code for lane detection. The right pane displays the results of the code execution.

Code (try.py):

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('/Users/shireenshaikh/desktop/line_detection.jpeg')
6
7 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
8 lower = np.array([0,0,168])
9 upper = np.array([172,111,255])
10 mask= cv2.inRange(hsv, lower, upper) # mask is region of interest
11 edges= cv2.Canny(mask, 75,150)
12 res = cv2.bitwise_and(img, img, mask = mask)
13
14 cv2.imshow('res', res)
15 cv2.imshow('edges', edges)
16 cv2.waitKey(0)
17 cv2.destroyAllWindows()
```

Results:

- edges:** A grayscale image showing the detected edges of the road lanes.
- res:** A color image showing the original road image with the detected lanes highlighted in yellow and white.