

**CSE365 Computer Vision**

**Phase 1 – Road Lane Line Detection**

**GitHub Repository:** [Link](https://github.com/zyad626/lane-detector)

Submitted To:

Dr. Mahmoud Khalil

Eng. Mahmoud Ahmed

Submitted by:

Zeyad Yasser Abdallah Ali – 18P4353

Moamen Soliman Moussa – 17P8247

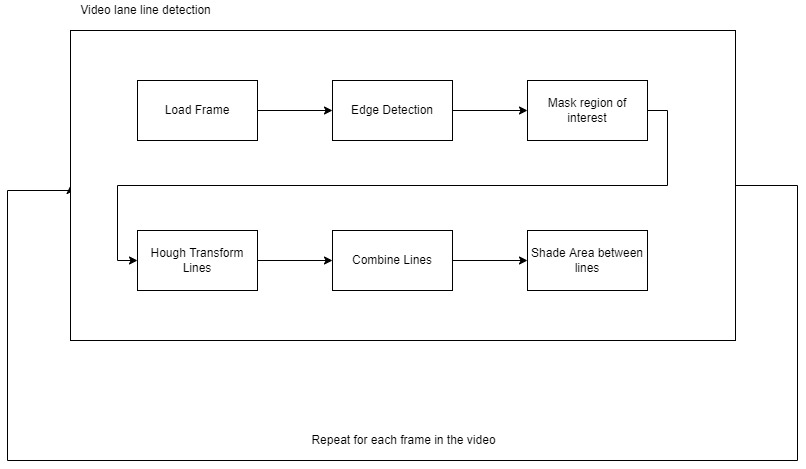
Contents

[Principle 3](#_Toc101634207)

[Results 8](#_Toc101634208)

[Code 10](#_Toc101634209)

# Principle



The First step the program is concerned with is loading an image. The app is capable of processing videos the same way since videos are nothing but a sequence of images (aka frames).

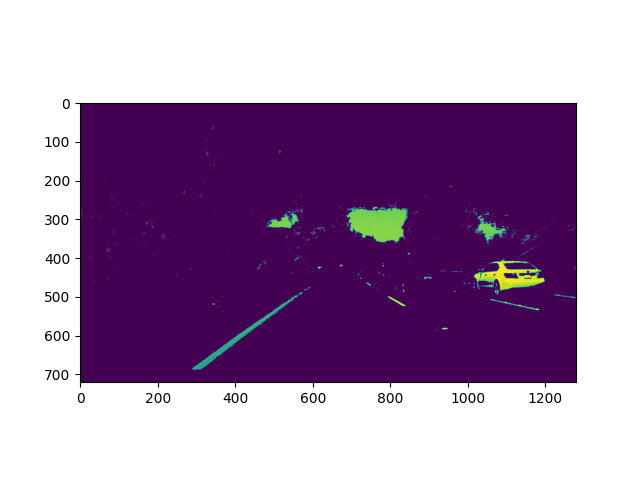
We will demonstrate the process on the following image



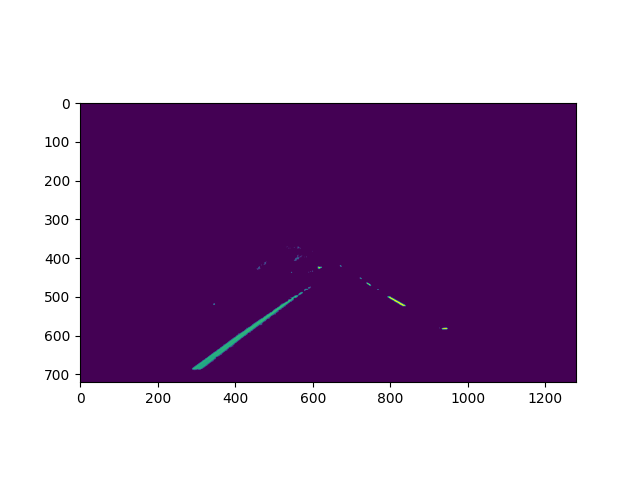
After the image is loaded, we will convert the image to grayscale and add an averaging filter (Gaussian) before applying edge detection in order to remove any noise that might affect out edge results.

We could have used canny function to directly get edges of the image, but we have noticed that canny does not perform well out of the box with the yellow lanes in the test images, and since most of the lanes in the world is yellow, we tweaked the edge detection function to support yellow variants by converting the image into HSV color space and defining a range of color values of yellows that should be included in our detection. Here is a snippet of the function that handles the edge detection with yellow lanes:

def edge\_detection(img, gray\_img, blurred\_img):  
  
hsv\_img = cv2.cvtColor(img, cv2.COLOR\_RGB2HSV)  
 lower\_yellow = np.array([20, 100, 100], dtype="uint8") # Yellow Ranges  
 upper\_yellow = np.array([100, 255, 255], dtype="uint8") # Yellow Ranges  
 yellow\_mask = cv2.inRange(hsv\_img, lower\_yellow, upper\_yellow)  
 white\_mask = cv2.inRange(gray\_img, 200, 255)  
 yw\_mask = cv2.bitwise\_or(white\_mask, yellow\_mask)  
 edges = cv2.bitwise\_and(gray\_img, yw\_mask)  
 return edges



After that, we will mask the image to only focus on the lane. We will crop the image to have 3 points of interest, which is a triangle

AAfter our edges are detected, we will apply Hough transform to detect the lines in the image. The values of rho, theta, threshold, and line length and gap were tested until we came out with the best results

lines = cv2.HoughLinesP(isolated\_lanes, 1, np.pi / 180, 35, np.array([]), minLineLength=25, maxLineGap=2)

The hough lines after that are averaged. The function average() finds *m* and *c* of line and outputs 2 solid lines on each side of the lane instead of the many hough lines. We can get from this function a starting and ending point for each line of the 2 sides.

For each of the 2 lines, we have (x1,y1) and (x2,y2). We can draw a line based on these 2 points, and same goes for the other lines. The logic goes as the following:

if lines is not None:  
 for line in lines:  
 x1, y1, x2, y2 = line  
 cv2.line(lines\_image, (int(x1), int(y1)), (int(x2), int(y2)), (255, 0, 0),10)

As for the filling the area between the lines, we already have 2 points for each line. As a result, we have 4 points from both lines that form a polygon. We can create a polygon based on these 4 points and fill them with a solid color. The logic comes as the following:

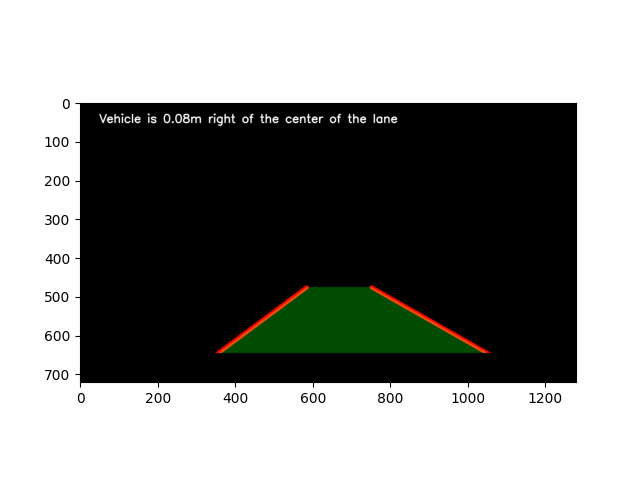
points = np.array(points)  
cv2.fillPoly(poly\_image, [points], (0, 255, 0))

To detect the vehicle variation from the center of the lane, we will assume a scale of 10px:1cm. We can find the lane center by subtracting highest point on the x-axis – lowest point, and then adding the offset from the start that we have definced from region of interest. The logic comes as the following:

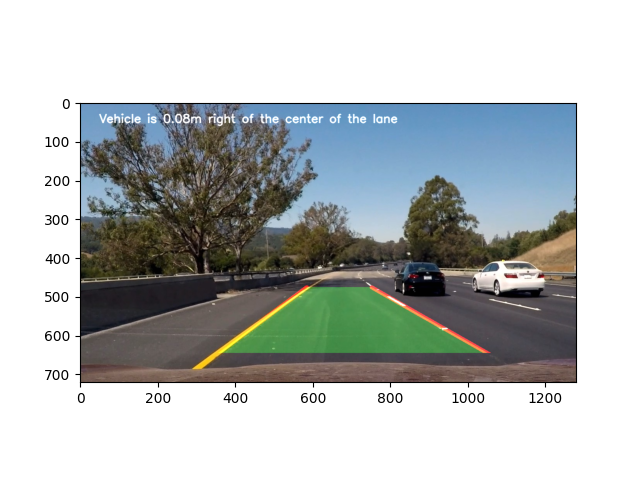
actual\_center\_x\_axis = image.shape[1] / 2  
car\_center\_x\_axis = points[0][0] + (points[-1][0] - points[0][0]) / 2  
center\_diff = actual\_center\_x\_axis - car\_center\_x\_axis # Scale is: 10 px = 1 cm  
  
if center\_diff > 0:  
 text = f"Vehicle is {abs(round(float(center\_diff / 1000), 2))}m left of the center of the lane"  
elif center\_diff < 0:  
 text = f"Vehicle is {abs(round(float(center\_diff / 1000), 2))}m right of the center of the lane"  
else:  
 text = f"Vehicle is at the center of the lane"

Finally for this step, we will overlay the results with the original image.

Here’s the result of the lines alone:



We will overlay the lines image with the original image to finally produce our final output:



As for the video, same process will be followed for each frame of the video

video = cv2.VideoCapture('project\_video.mp4')  
frame\_width = int(video.get(3))  
frame\_height = int(video.get(4))  
result = cv2.VideoWriter('RESULT.avi',cv2.VideoWriter\_fourcc(\*'MJPG'), 25, (frame\_width, frame\_height))  
while video.isOpened():  
 ret, frame = video.read()  
 if ret == True:  
 frame = frame\_process(frame)  
 result.write(frame)   
 cv2.imshow('Frame', frame)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 else:  
 break  
video.release()  
cv2.destroyAllWindows()

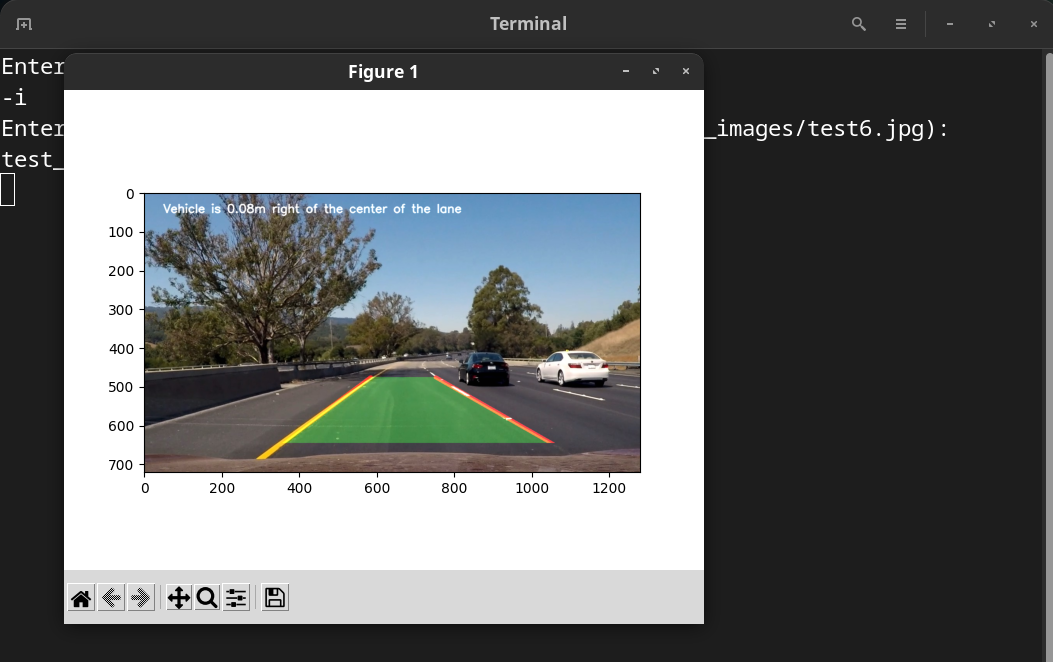
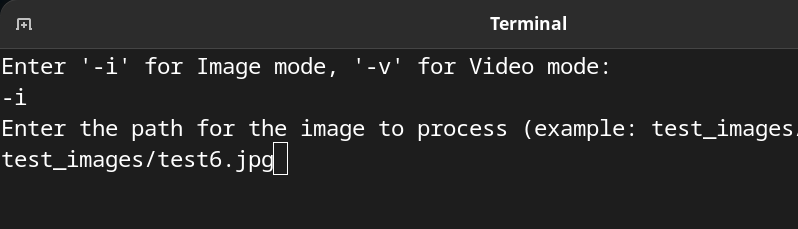
How to run the code

1. Download the code ZIP file from GitHub

2. Place the image/video you want to test within the code directory. You can use the already provided test images

3. Double click on "BASH\_TO\_RUN.sh"

4. Follow the instructions on the command line, you will be asked for which mode you want to run the code on (image mode '-i' or video mode '-v') and you will be asked to provide the path to your file.



# Results



Graphical user interface

Description automatically generated

# Code

**GitHub Repository:** <https://github.com/ahmed-fawzy99/Road-Lane-Line-Detection>

**import** matplotlib**.**pylab **as** plt

**import** cv2

**import** numpy **as** np

offset\_width **=** 100

**def** region**(**image**):**

"""This function isolates certain the region that contains the lane lines

it takes the image we want (We will pass the canny results) to isolate and

produces the masked image based on our defined region of interest"""

height**,** width **=** image**.**shape

region\_of\_interest **=** np**.**array**([[(**offset\_width**,** height**),** **(**550**,** 350**),** **(**1200**,** height**)]])**

mask **=** np**.**zeros\_like**(**image**)** # Mask now is black image

mask **=** cv2**.**fillPoly**(**mask**,** region\_of\_interest**,** 255**)** # Fill region with white

mask **=** cv2**.**bitwise\_and**(**image**,** mask**)** # Isolate edges that correspond with lane lines

**return** mask

**def** average**(**image**,** lines**):**

"""This function averages hough lines result. it finds m and c of line

and outputs 2 solid lines instead (one on each side)"""

left **=** **[]**

right **=** **[]**

**for** line **in** lines**:**

x1**,** y1**,** x2**,** y2 **=** line**.**reshape**(**4**)**

parameters **=** np**.**polyfit**((**x1**,** x2**),** **(**y1**,** y2**),** 1**)**

m **=** parameters**[**0**]**

c **=** parameters**[**1**]**

**if** m **<** 0**:**

left**.**append**((**m**,** c**))**

**else:**

right**.**append**((**m**,** c**))**

right\_avg **=** np**.**average**(**right**,** axis**=**0**)**

left\_avg **=** np**.**average**(**left**,** axis**=**0**)**

left\_line **=** make\_points**(**image**,** left\_avg**)**

right\_line **=** make\_points**(**image**,** right\_avg**)**

**return** np**.**array**([**left\_line**,** right\_line**])**

**def** edge\_detection**(**img**,** gray\_img**,** blurred\_img**):**

"""

# Canny Edge Detection

Note: Regular canny detection will not work if the lanes has yellow color.

You can use the following canny() function to reduce overhead \*\*IF\*\* the lanes are only white.

However, the yellow detection code also detects white lanes anyways.

edges = cv2.Canny(blurred\_img, 50, 150)

"""

hsv\_img **=** cv2**.**cvtColor**(**img**,** cv2**.**COLOR\_RGB2HSV**)**

lower\_yellow **=** np**.**array**([**20**,** 100**,** 100**],** dtype**=**"uint8"**)** # Yellow Ranges

upper\_yellow **=** np**.**array**([**100**,** 255**,** 255**],** dtype**=**"uint8"**)** # Yellow Ranges

yellow\_mask **=** cv2**.**inRange**(**hsv\_img**,** lower\_yellow**,** upper\_yellow**)**

white\_mask **=** cv2**.**inRange**(**gray\_img**,** 200**,** 255**)**

yw\_mask **=** cv2**.**bitwise\_or**(**white\_mask**,** yellow\_mask**)**

edges **=** cv2**.**bitwise\_and**(**gray\_img**,** yw\_mask**)**

**return** edges

**def** make\_points**(**image**,** average**):**

**try:**

m**,** c **=** average

**except** **TypeError:**

m**,** c **=** 0.001**,** 0

y1 **=** **int(**image**.**shape**[**0**])**

y2 **=** **int(**y1 **\*** **(**3.3 **/** 5**))**

x1 **=** **int((**y1 **-** c**)** **//** m**)**

x2 **=** **int((**y2 **-** c**)** **//** m**)**

**return** np**.**array**([**x1**,** y1**,** x2**,** y2**])**

**def** display\_lines**(**image**,** lines**):**

lines\_image **=** np**.**zeros\_like**(**image**,** 'uint8'**)**

poly\_image **=** np**.**zeros\_like**(**image**,** 'uint8'**)**

points **=** **[]**

**if** lines **is** **not** **None:**

**for** line **in** lines**:**

x1**,** y1**,** x2**,** y2 **=** line

cv2**.**line**(**lines\_image**,** **(int(**x1**),** **int(**y1**)),** **(int(**x2**),** **int(**y2**)),** **(**255**,** 0**,** 0**),** 10**)**

# Fill space between lines

**if** x1 **<** x2**:**

points**.**append**([**x1**,** y1**])**

points**.**append**([**x2**,** y2**])**

**else:**

points**.**append**([**x2**,** y2**])**

points**.**append**([**x1**,** y1**])**

# Measure distance from center

actual\_center\_x\_axis **=** image**.**shape**[**1**]** **/** 2

car\_center\_x\_axis **=** points**[**0**][**0**]** **+** **(**points**[-**1**][**0**]** **-** points**[**0**][**0**])** **/** 2

center\_diff **=** actual\_center\_x\_axis **-** car\_center\_x\_axis # Scale is: 10 px = 1 cm

**if** center\_diff **>** 0**:**

text **=** f"Vehicle is {**abs(round(float(**center\_diff **/** 1000**),** 2**))**}m left of the center of the lane"

**elif** center\_diff **<** 0**:**

text **=** f"Vehicle is {**abs(round(float(**center\_diff **/** 1000**),** 2**))**}m right of the center of the lane"

**else:**

text **=** f"Vehicle is at the center of the lane"

# Highlight the lanes

points **=** np**.**array**(**points**)**

cv2**.**fillPoly**(**poly\_image**,** **[**points**],** **(**0**,** 255**,** 0**))**

cv2**.**addWeighted**(**poly\_image**,** 0.3**,** lines\_image**,** 1**,** 0**,** lines\_image**)**

lines\_image **=** cv2**.**putText**(**lines\_image**,** text**,** **(**50**,** 50**),** cv2**.**FONT\_HERSHEY\_SIMPLEX**,** 1**,** **(**255**,** 255**,** 255**),** 2**,** cv2**.**LINE\_AA**)**

# Crop car engine hood

lines\_image **=** cv2**.**rectangle**(**lines\_image**,** **(**0**,** image**.**shape**[**0**]** **-** 75**),** **(**image**.**shape**[**1**],** image**.**shape**[**0**]),** **(**0**,** 0**,** 0**),** **-**1**)**

**return** lines\_image

**def** frame\_process**(**img**):**

# Read Image

# img = cv2.imread('test\_images/test1')

# img = cv2.cvtColor(img, cv2.COLOR\_RGB2BGR)

copy **=** np**.**copy**(**img**)**

# Convert to Grayscale

gray\_img **=** cv2**.**cvtColor**(**copy**,** cv2**.**COLOR\_BGR2GRAY**)**

# Add Gaussian blur to improve canny results

blurred\_img **=** cv2**.**GaussianBlur**(**gray\_img**,** **(**5**,** 5**),** 0**)**

# Edge Detection with support of yellow lanes

edges **=** edge\_detection**(**img**,** gray\_img**,** blurred\_img**)**

# Isolated lane lines edges

isolated\_lanes **=** region**(**edges**)**

# Hough line transform

lines **=** cv2**.**HoughLinesP**(**isolated\_lanes**,** 1**,** np**.**pi **/** 180**,** 35**,** np**.**array**([]),** minLineLength**=**25**,** maxLineGap**=**2**)**

averaged\_lines **=** average**(**copy**,** lines**)**

black\_lines **=** display\_lines**(**copy**,** averaged\_lines**)**

lanes **=** cv2**.**addWeighted**(**copy**,** 1**,** black\_lines**,** 1**,** 1**)**

**return** lanes

# img = cv2.imread('test\_images/test6.jpg')

# img = cv2.cvtColor(img, cv2.COLOR\_RGB2BGR)

# plt.imshow(frame\_process(img))

# plt.show()

video **=** cv2**.**VideoCapture**(**'project\_video.mp4'**)**

frame\_width **=** **int(**video**.**get**(**3**))**

frame\_height **=** **int(**video**.**get**(**4**))**

# result = cv2.VideoWriter('RESULT.avi',cv2.VideoWriter\_fourcc(\*'MJPG'), 25, (frame\_width, frame\_height)) # Already exported it once

**while** video**.**isOpened**():**

ret**,** frame **=** video**.**read**()**

**if** ret **==** **True:**

frame **=** frame\_process**(**frame**)**

# result.write(frame) # Already exported it once

cv2**.**imshow**(**'Frame'**,** frame**)**

**if** cv2**.**waitKey**(**1**)** **&** 0xFF **==** **ord(**'q'**):**

**break**

**else:**

**break**

video**.**release**()**

cv2**.**destroyAllWindows**()**