ECE – 510

PORTLAND STATE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Intelligent Robotics III

Project: Lane Detection for Autonomous Vehicles

Guidance: Dr. Marek Perkowski

Aniruddha Shahapurkar (anirud@pdx.edu)

Date: 05/20/2020

INDEX

INTRODUCTION:

- ❖ Advanced Driving Assistant Systems, intelligent and autonomous vehicles are promising solutions to enhance road safety, traffic issues and passengers' comfort.
- ❖ Such applications require advanced computer vision algorithms that demand powerful computers with high-speed processing capabilities.
- ❖ Keeping intelligent vehicles on the road until its destination, in some cases, remains a great challenge, particularly when driving at high speeds.
- ❖ The first principle task is robust navigation, which is often based on system vision to acquire RGB images of the road for more advanced processing.
- ❖ The second task is the vehicle's dynamic controller according to its position, speed and direction. In this project, we tried to implement an efficient road boundaries and detection algorithm for intelligent and autonomous vehicles.

INPUT IMAGE:

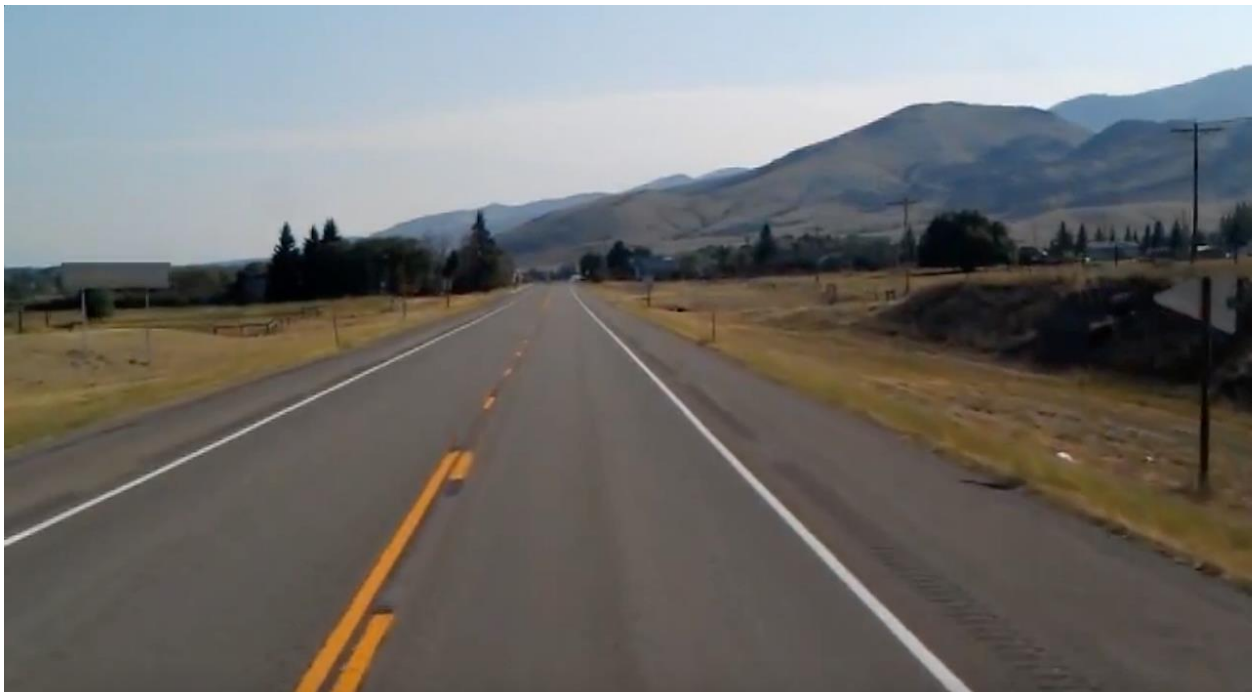


Figure. Test Image for Lane Detection

IMPPLEMENTATION METHODOLOGY FOR LANE DETECTION:

1. **Converting RGB image to Gray scale image:**
   - Using cvtColor transformation form cv2 (OpenCV)

2. **Smoothening Image**
   - Using Gaussian filter to reduce noise and Smoothen the image

3. **Edge Detection**
   - Using Canny Edge detection

4. **Finding Lane lines of interest**
   - Using Hough Transformation

5. **Optimization**
   - Using Average slope-intercept method

6. **Detecting Lanes in the image file**
   - Combining all the above methods to detect lane lines in an image file

7. **Lane Detection in the video file**
   - Applying same methodology but for a continuous set of images, .i.e., A Video file

Step1: Grayscale Conversion

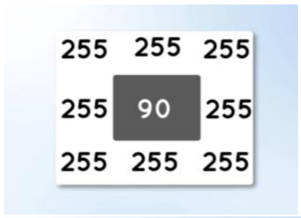
Converting RGB image to Gray scale image.

Reason for doing:

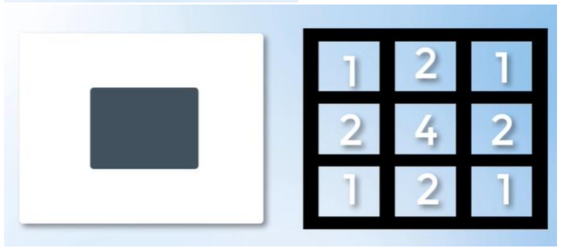
- A three-channel color image will have Red, Green and Blue channels.
- Thus, each pixel is a combination of three intensity values.
- Whereas a gray scale image will have only one channel and each pixel will have only one intensity value ranging from 0 to 255.
- The point is by using a gray scale image, processing a single channel will be faster than processing a three-channel color image and also there will be less computational intensity

Step2: Smoothening Image

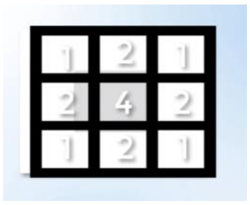
- Reduce noise and smoothen the image
- When detecting edges, it is important to accurately catch as many edges in the image as possible. To achieve that, we must filter out image noise.
- Because image noise can create false edges and ultimately effect edge detection.
- That’s why we need to remove noise and thus smoothen the image. This will be done with a Gaussian filter.
- Gaussian filter
    - An image is stored as a collection of discrete pixels.
    - Each pixel for a gray scale image is described by a single number which describes the brightness of the pixel.
    - So how does smoothening takes place?
    - It is done by modifying the pixel value by the average value of the pixel intensity values around it.



255	255	255
255	90	255
255	255	255



1	2	1
2	4	2
1	2	1

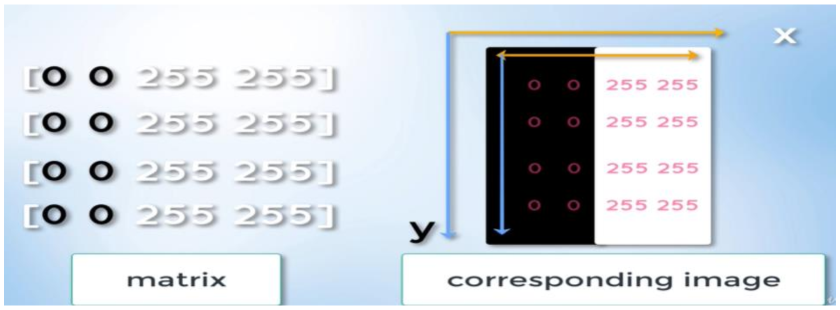


1	2	1
2	4	2
1	2	1

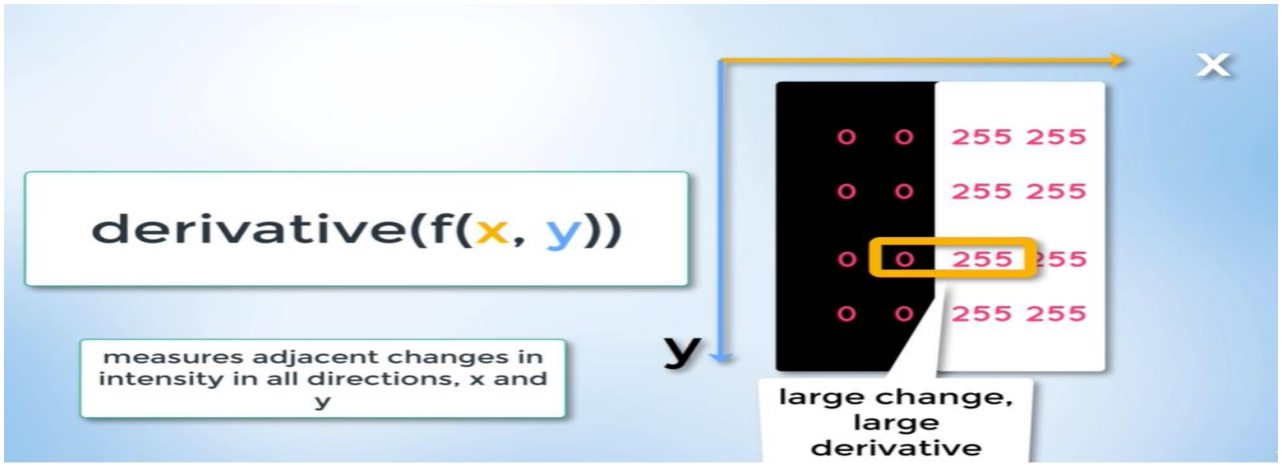
- Kernel *(matrix filter)* of normally distributed function is run across our image and thus sets each pixel value to the weighted average of its neighboring pixels.
- Size of the Kernel is dependent on specific situations.

Step3: Canny Edge Detection:

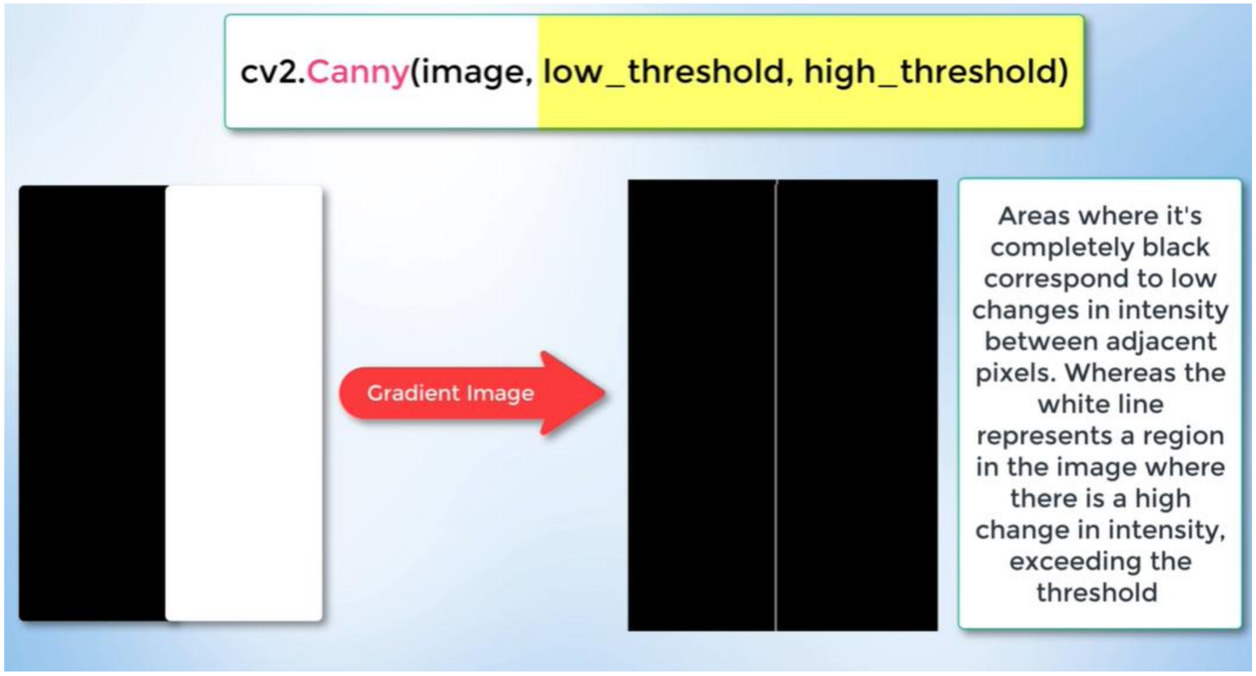
Purpose: Detect the edges.



Performs Derivative in x and y direction as shown below:



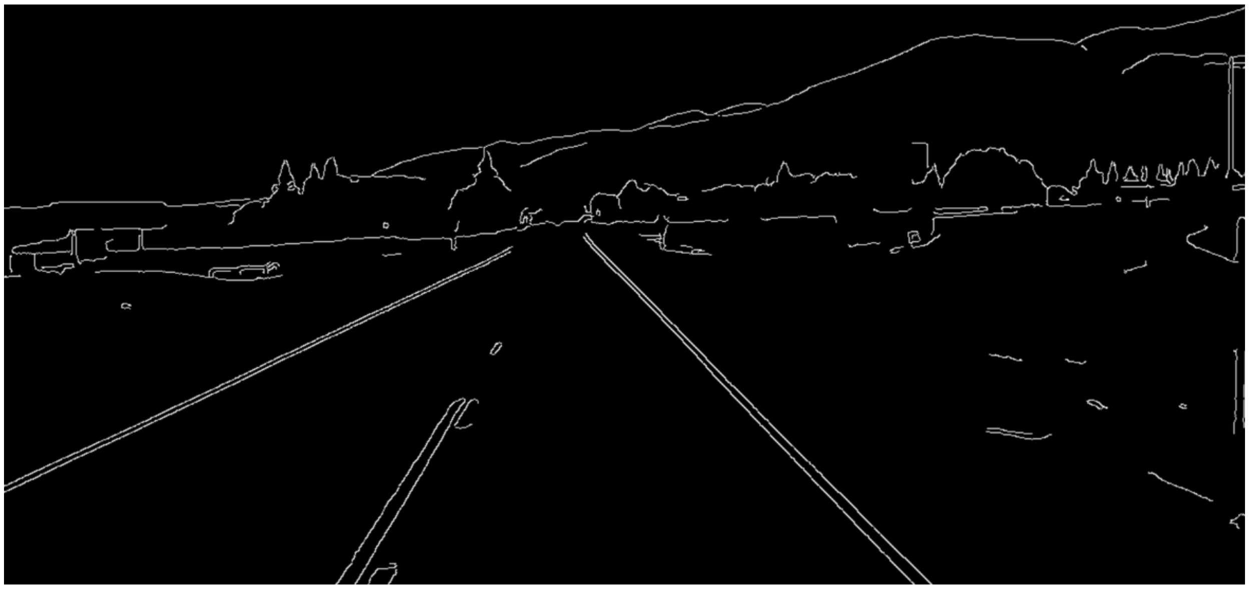
derivative(f(x, y))
measures adjacent changes in intensity in all directions, x and y
x
y
0 0 255 255
0 0 255 255
0 0 255 255
0 0 255 255
large change, large derivative



cv2.Canny(image, low_threshold, high_threshold)
Gradient Image
Areas where it's completely black correspond to low changes in intensity between adjacent pixels. Whereas the white line represents a region in the image where there is a high change in intensity, exceeding the threshold

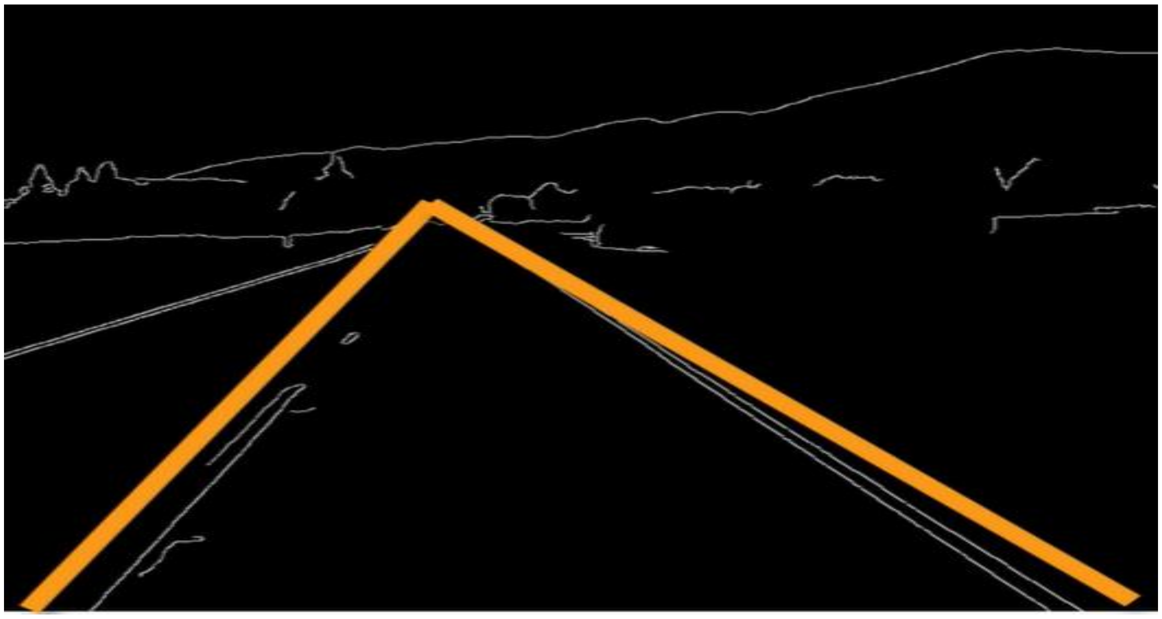
Images explain it well.

Result:

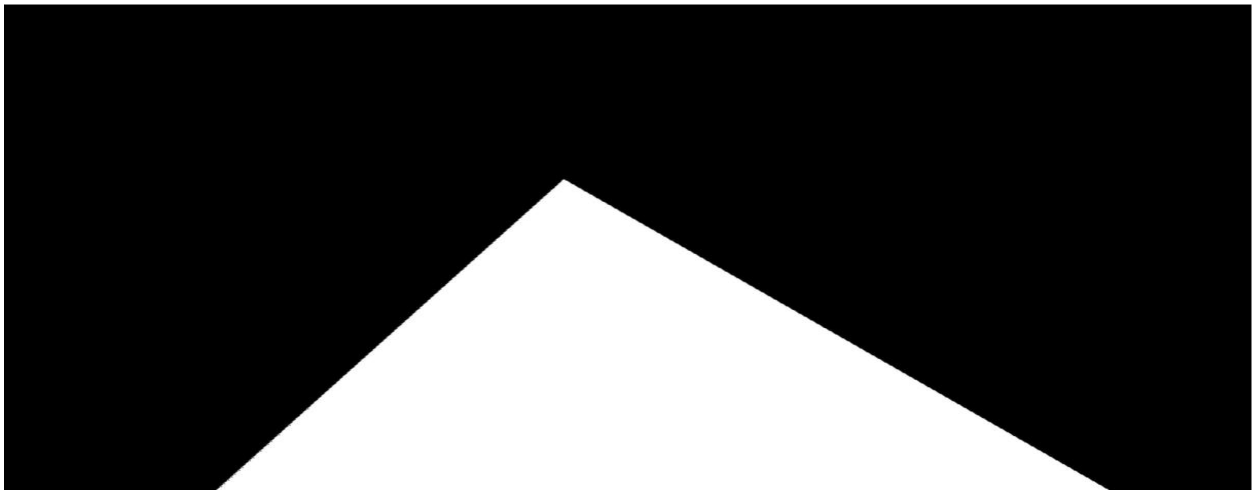


- The white lines indicate most rapid changes in the brightness of the image.
- Small changes in brightness are not traced at all, thus displayed as black color pixels as they fall below the lower threshold.
- Thus, we outlined the strongest gradient *(change in pixel colors)* in our image.

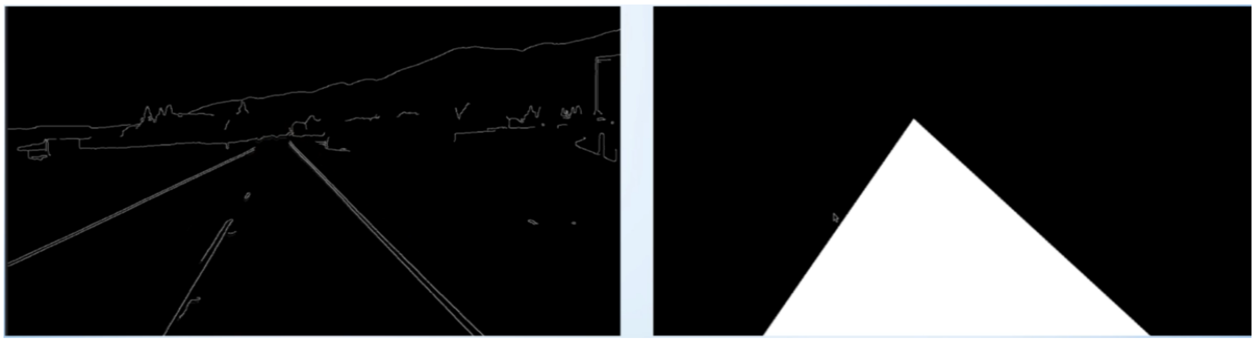
Step4: Finding Lane lines of interest using Hough Transformation



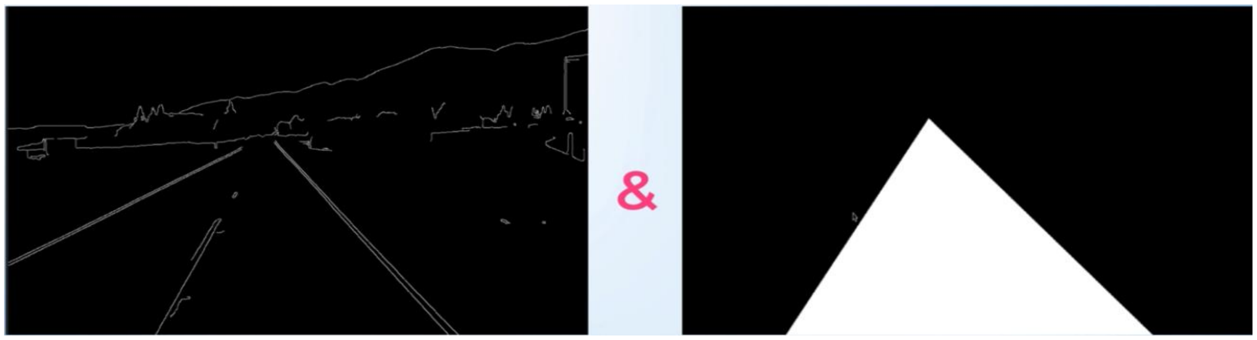
- Now, as we have identified the lane lines in the image, our next goal is to find region of interest, i.e., Lane lines in the image.
- For this, we defined a mask of zeros (Black pixels) of the same size of our image.
- After that, we defined a triangular shape of white pixels with the coordinates.
- Then we filled the mask of black image with the triangular region of white pixels. The result appeared as shown below.



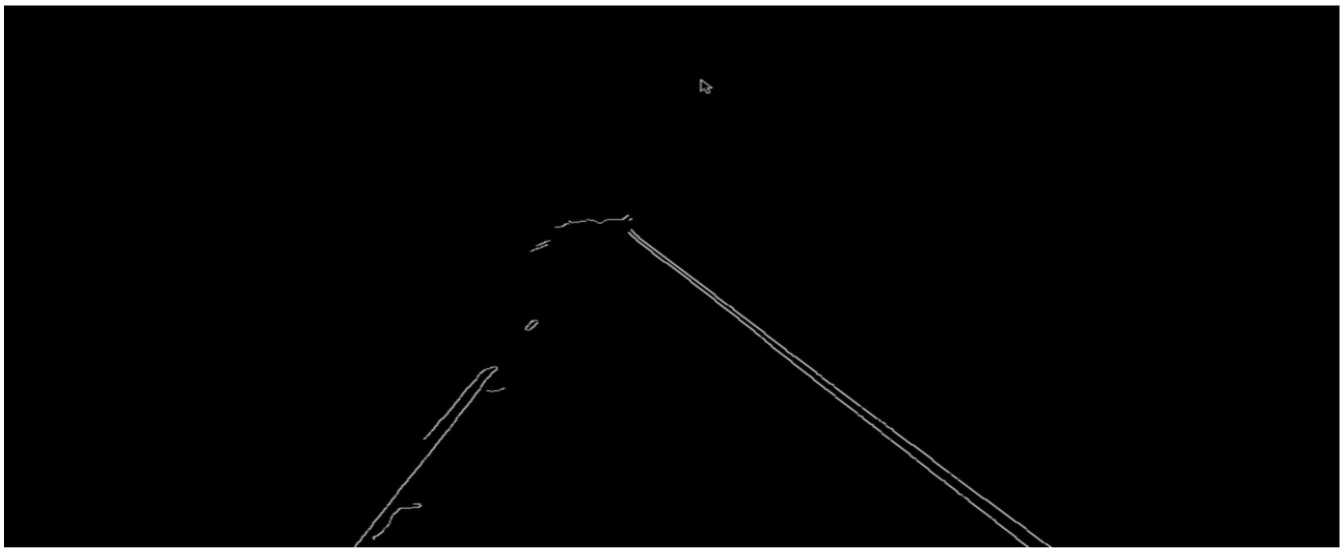
- The following image shows us two parts. The first part is our image of the road obtained after canny edge detection, and the second part shows us the triangular mask that we defined.
- One important thing is both the images of same size.



- Then we perform bit wise AND operation of the above two images.



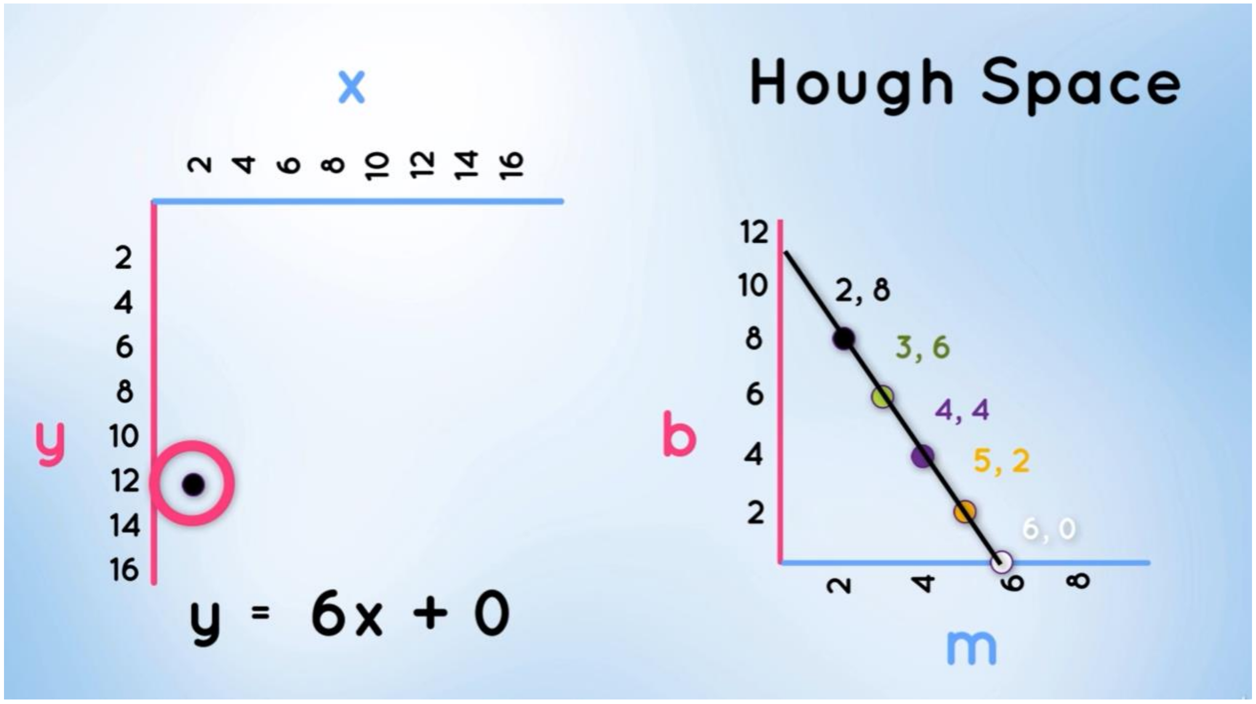
- In both the images all the black pixels will have pixel intensities of 0 and all the white pixels have pixel intensities of 255.
- Therefore if we will perform bitwise ANDing operation of the two images, only the white pixels present in the first image corresponding to the triangular shape of the second image will remain as it is, and all the remaining pixels will remain black thus highlighting the lane lines in the image.
- The result of the bitwise AND operation is as shown below.



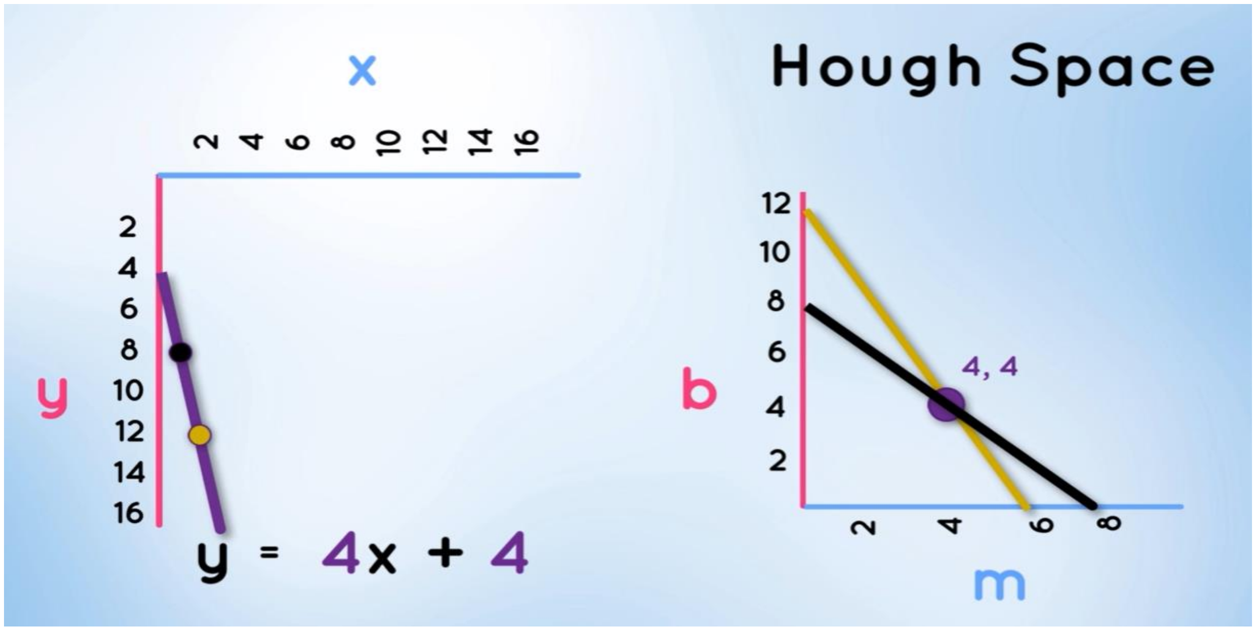
- Thus, we can see that the region of our interest i.e., the lane lines are highlighted.

Hough Transformation:

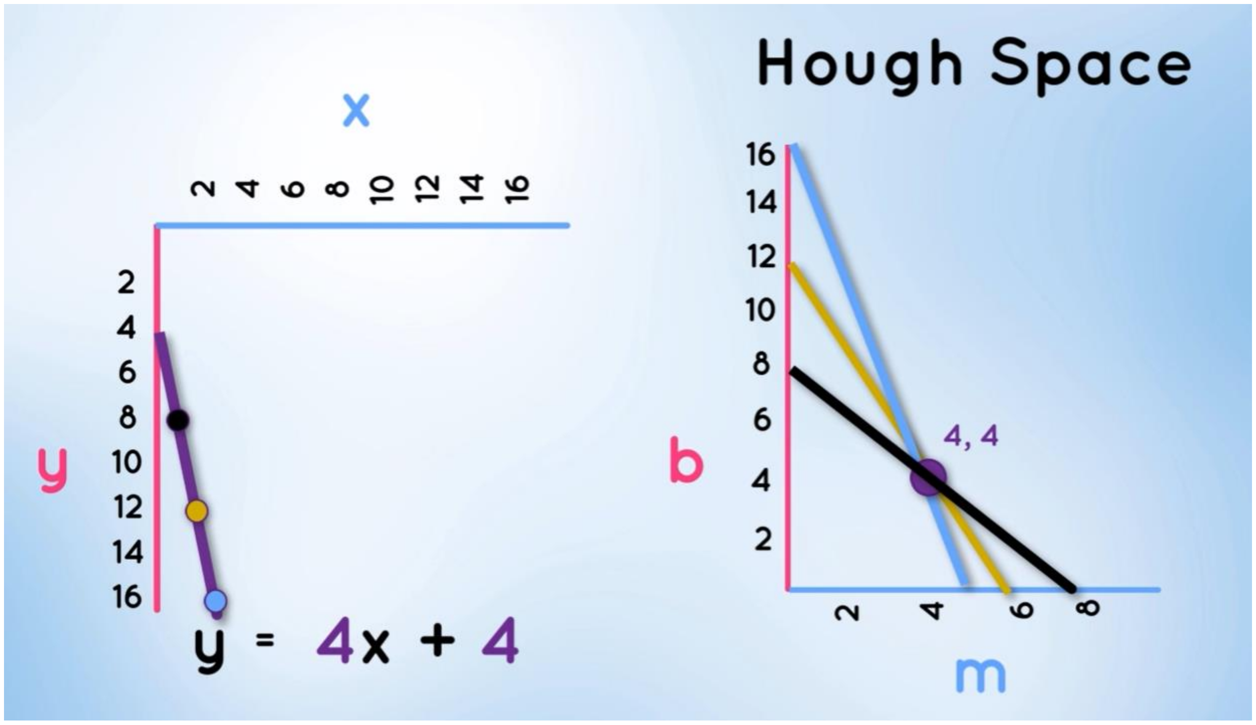
- To understand Hough Transformation, we need to understand Hough Space.
- In the Hough Space x-axis represents slope and y-axis represents y-intercept.



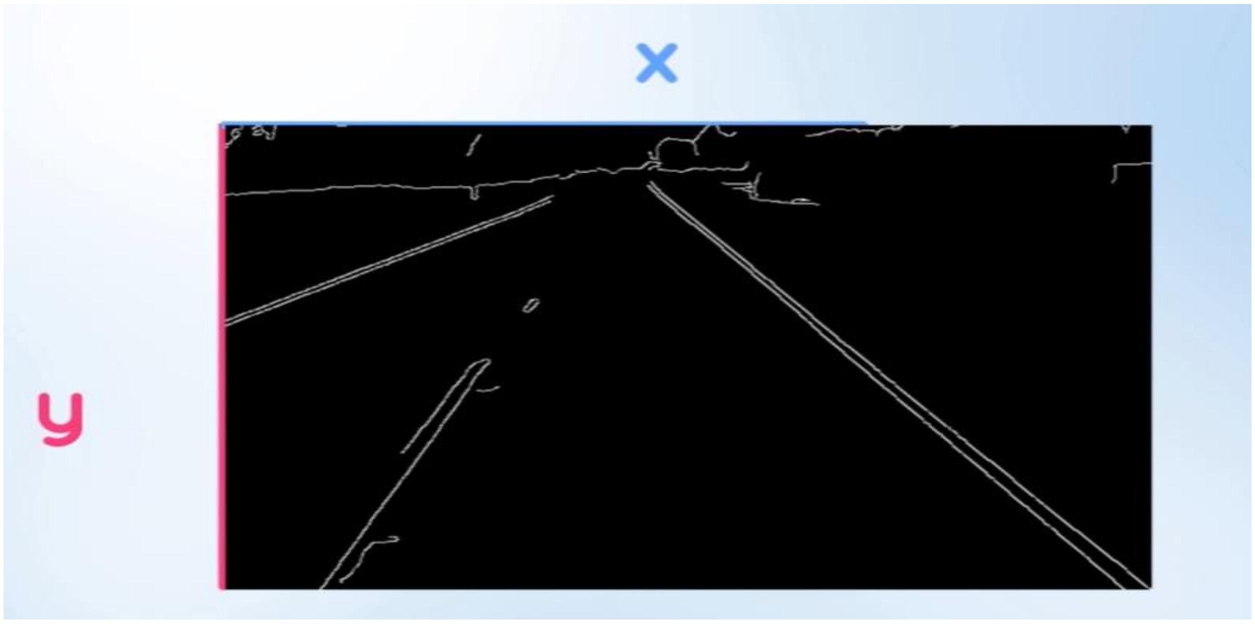
- We can see that a single point in Cartesian coordinate system is represented by a line in Hough space. This is because number of lines with distinct m and b values, passing through the same point in the Cartesian coordinate system will form a line in the Hough space as shown in the above figure.



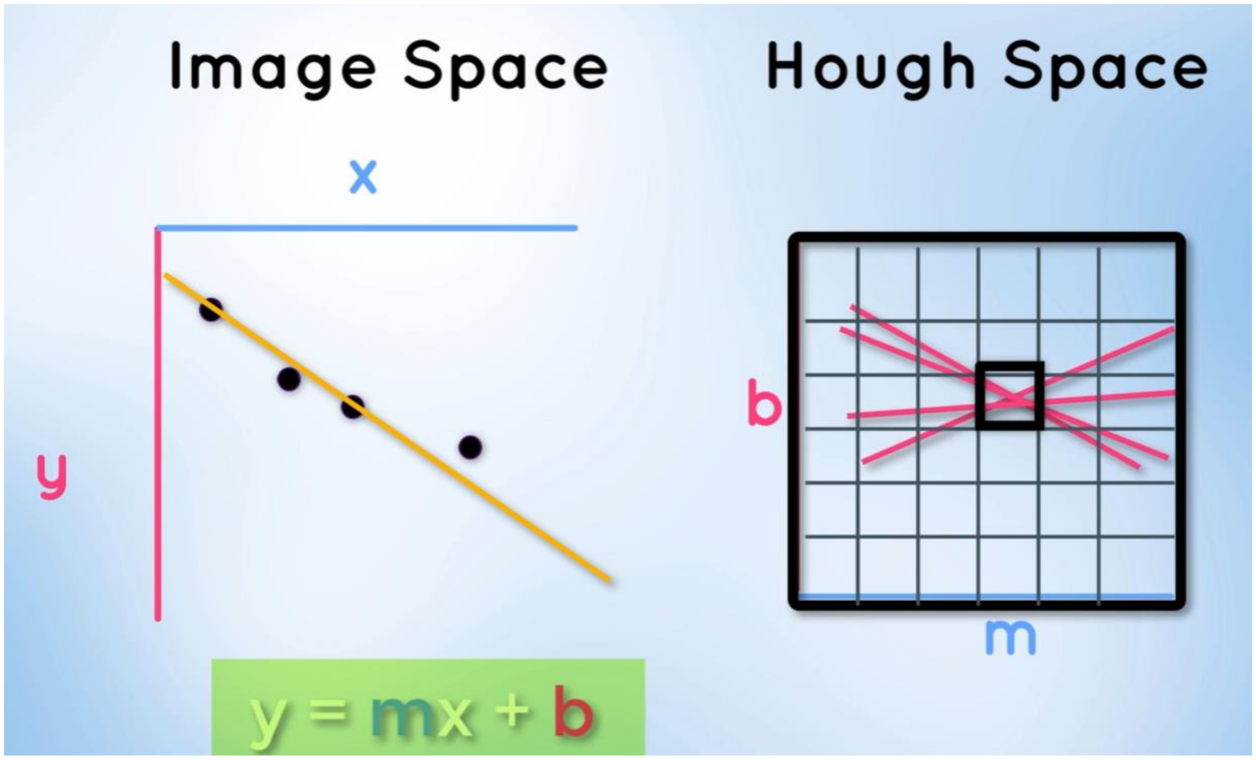
- Now as shown in above figure, if there are two distinct points in x-y plane as shown in the above figure, there will be many different lines passing through these lines separately. But there will be only one line passing through these points.
- We can determine that with the point of intersection in the Hough space. Because that point of intersection represents m and b values of the line consistent with crossing both of our points in x-y plane and thus forming a line
- Suppose we have one more point in our x-y plane.



- Now again, in case of three points, there will be a line passing through all three points which we can see with the help of intersection again at the same point in the Hough space having m = 4 and b = 4.
- We can use this concept for lane detection.
- Consider our gradient image shown below.

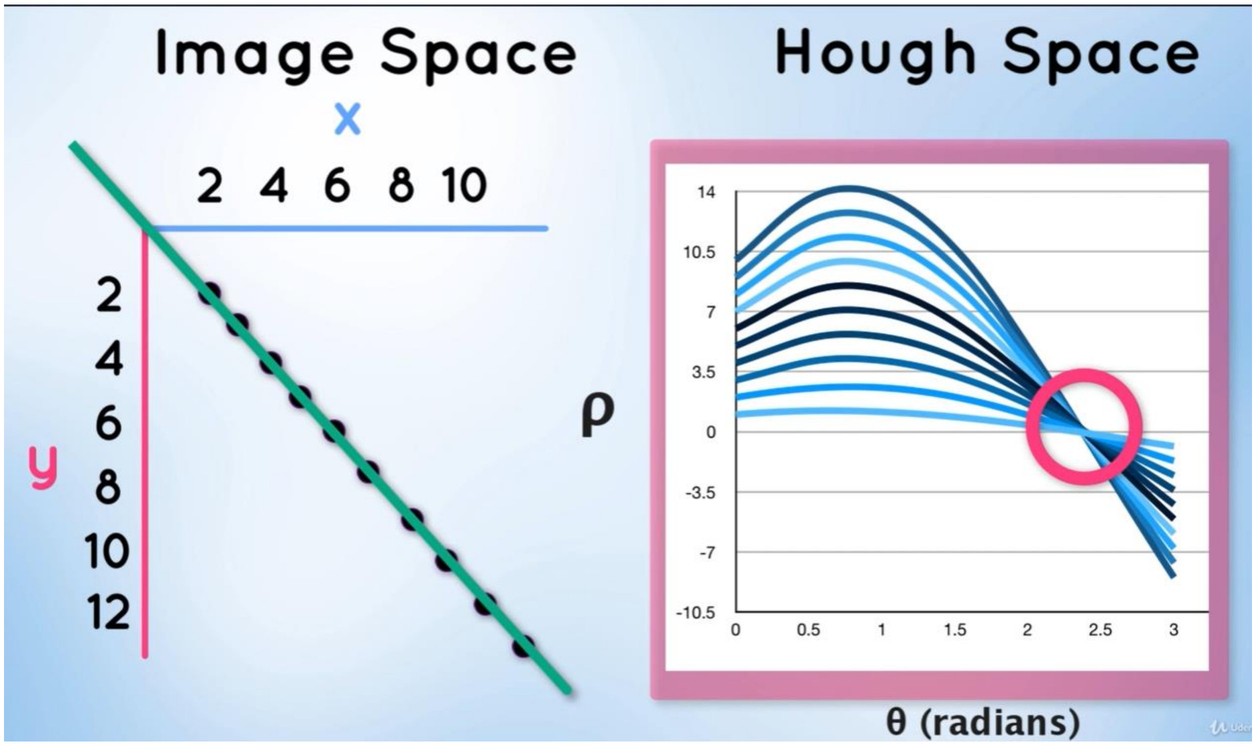


- We can see that the gradient image is nothing but series of white points which represent edges in our image space.
- Now how will detect these edges?

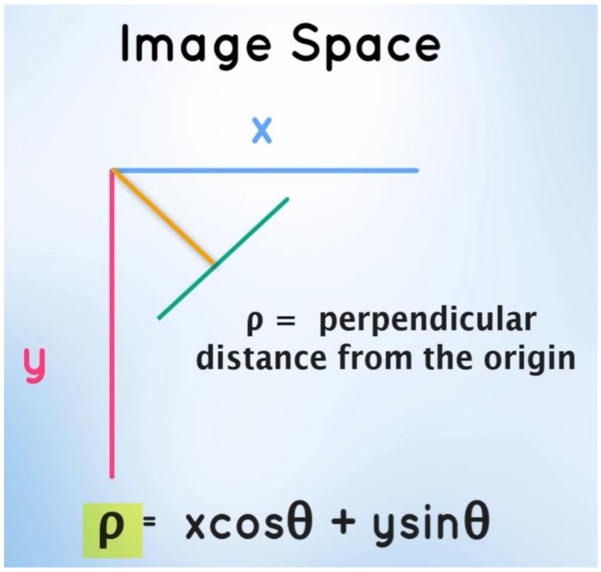


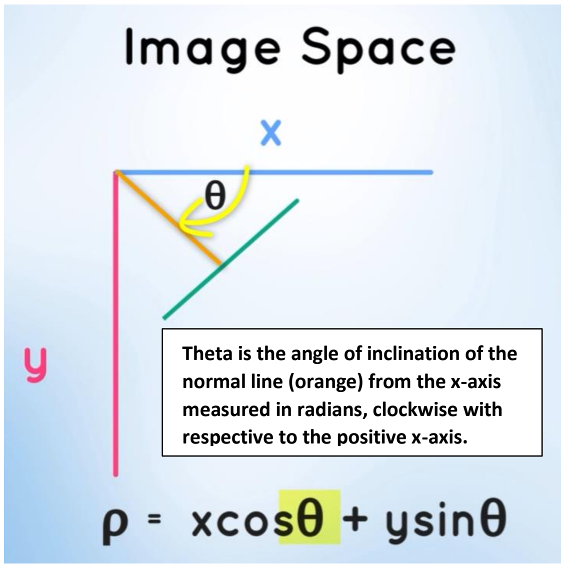
- In above figure, we can see that our Hough space is divided in a grid. We can see that the point of intersections in the Hough space are in a single bin. For every point of intersection, we will cast a vote inside the bin that it belongs to. The bin with the maximum number of votes or in other words, maximum number of intersections will represent our line in the x-y plane.

- So, with the m and b co-ordinates of the Hough space's most voted bin, we will draw a line in the x-y plane which will be of best fit for our data.
- This is how we will identify lines in our gradient image of the road.



- The above image shows the same concept in the form of sinusoidal curves. The points forming one line in the x-y plane will intersect in the Hough space at same point.
- What are rho and theta?

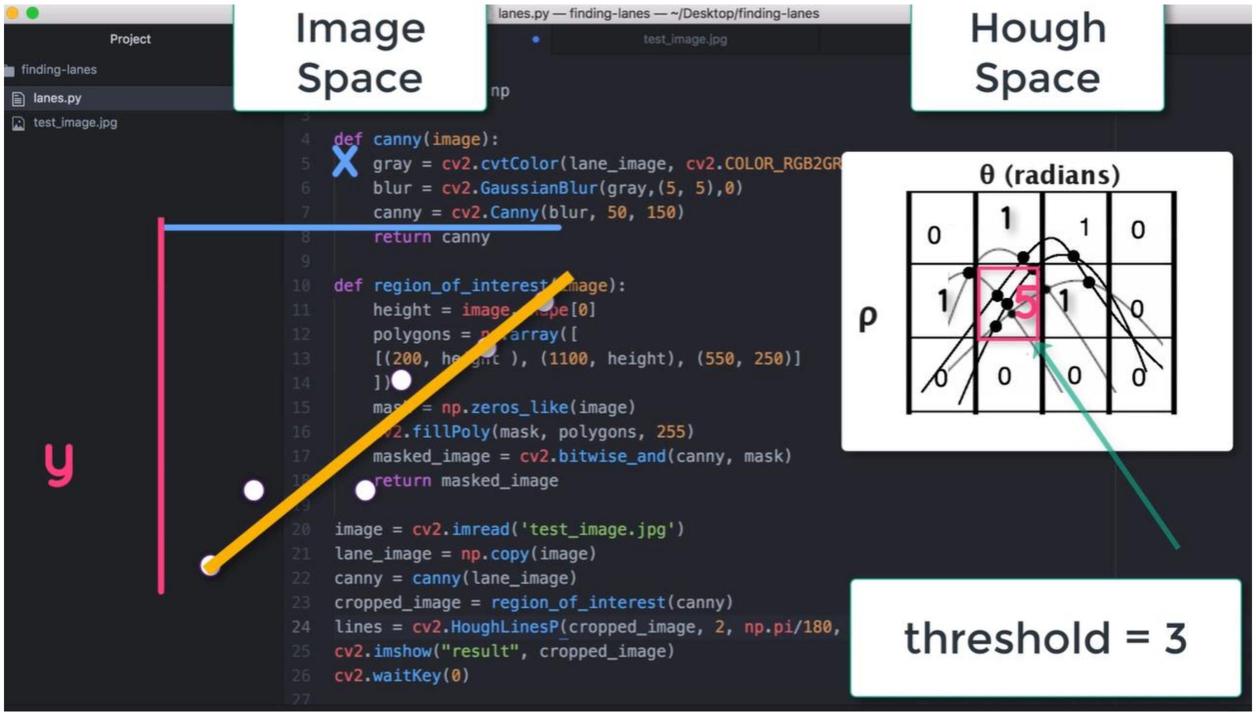




Theta is the angle of inclination of the normal line (orange) from the x-axis measured in radians, clockwise with respective to the positive x-axis.

- In our code, we implemented Hough Transform as follows:

lines = cv2.HoughLinesP(cropped_image, 2, np.pi/180, 100, np.array([]), minLineLength = 40, maxLineGap = 5)



```
np

def canny(image):
    gray = cv2.cvtColor(lane_image, cv2.COLOR_RGB2GR
    blur = cv2.GaussianBlur(gray,(5, 5),0)
    canny = cv2.Canny(blur, 50, 150)
    return canny

def region_of_interest(image):
    height = image.shape[0]
    polygons = np.array([
    [(200, height), (1100, height), (550, 250)]
    ])
    mask = np.zeros_like(image)
    cv2.fillPoly(mask, polygons, 255)
    masked_image = cv2.bitwise_and(canny, mask)
    return masked_image

image = cv2.imread('test_image.jpg')
lane_image = np.copy(image)
canny = canny(lane_image)
cropped_image = region_of_interest(canny)
lines = cv2.HoughLinesP(cropped_image, 2, np.pi/180,
cv2.imshow("result", cropped_image)
cv2.waitKey(0)
```

- 2nd and 3rd argument are for resolution, they determine the bin size. 2nd argument is rho which is distance resolution and 3rd is theta which is angle resolution Rho = 2 i.e., precision of 2 pixels; and Theta = 1 = pi/180. This provided us the best result, 100 is the threshold for minimum number of intersections in a bin, next is a place holder array and others are minimum length of the line before it breaks and maximum allowed gap in the line. Both are measured in pixels.

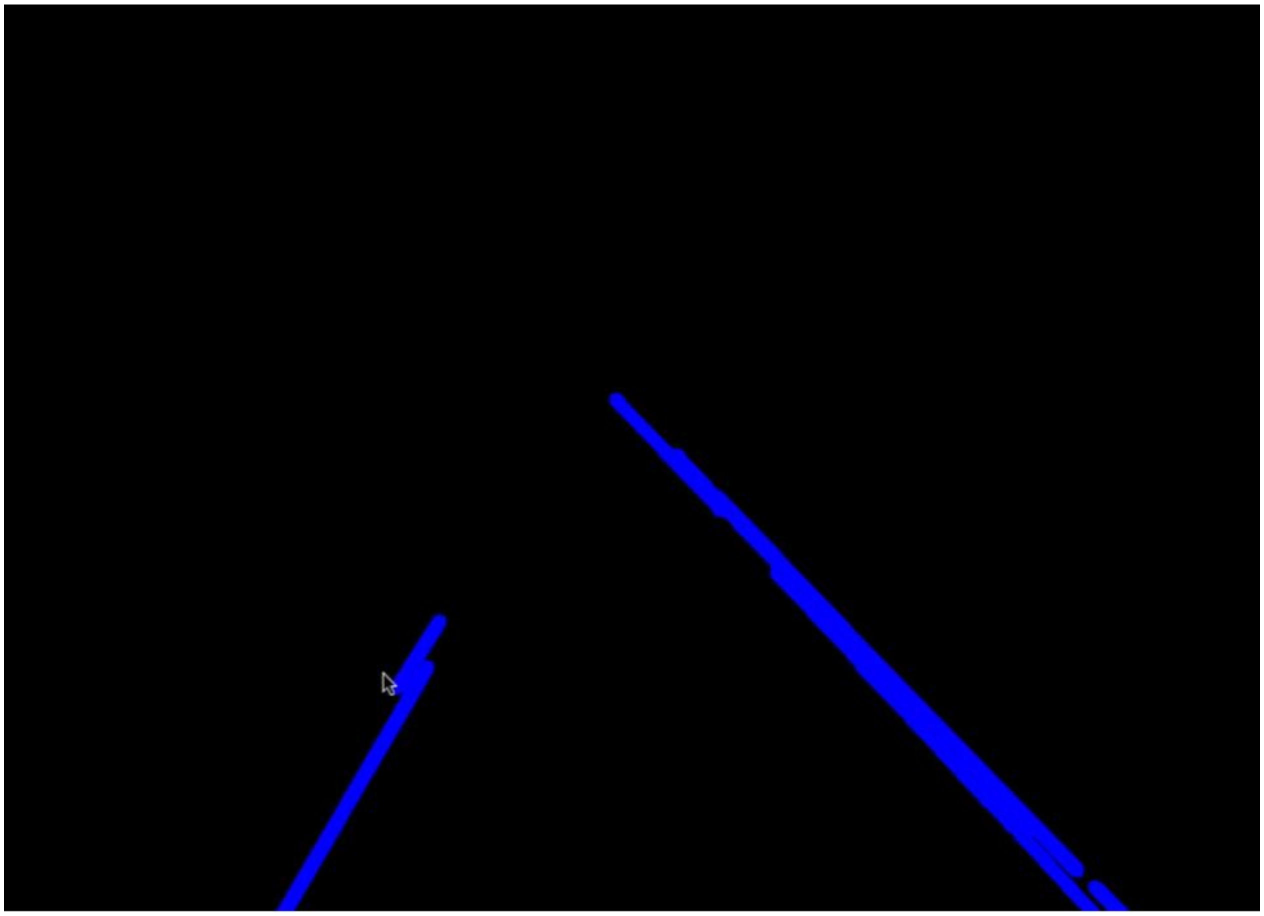
- After applying Hough Transform, we displayed the detected lane line using display_lines() function as shown below:

```
def display_lines(image, lines):
    line_image = np.zeros_like(image)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)
    return line_image
```

- It can be explained as follows:
    - First we defined an image having same size of our original image but with all black pixels
    - Then, we checked if the array of lines is empty or not

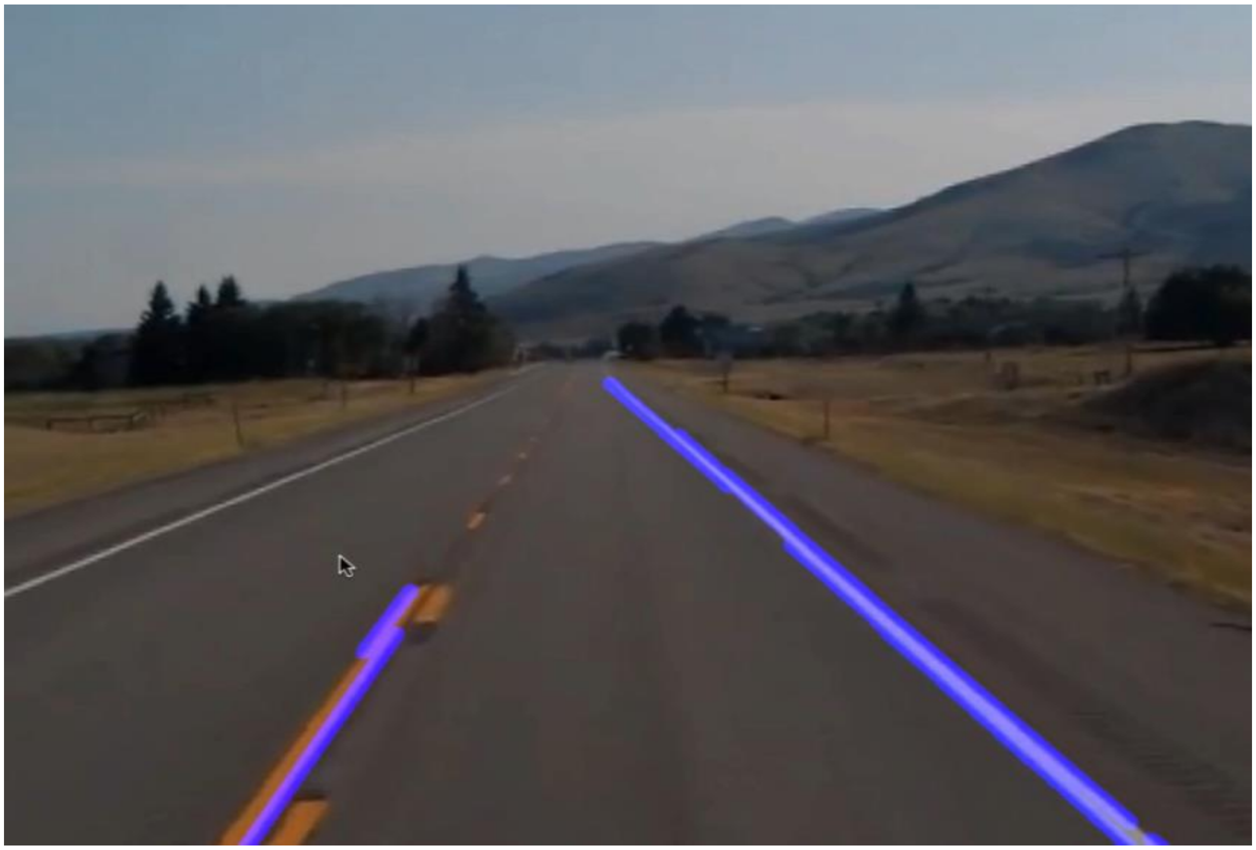
- After that we converted each line in the lines array from two-dimensional array to one dimensional array.
- By unpacking array elements into 4 different variables we converted them from 2-D to 1-D form.
- The line() function draws a line between two points. 1st argument is the name of the image. 2nd and 3rd arguments are Coordinates of first and second point. 4th is Color (BGR) and 5th is Line Thickness.

▪ When we display our detected lane lines, we obtained the following result:



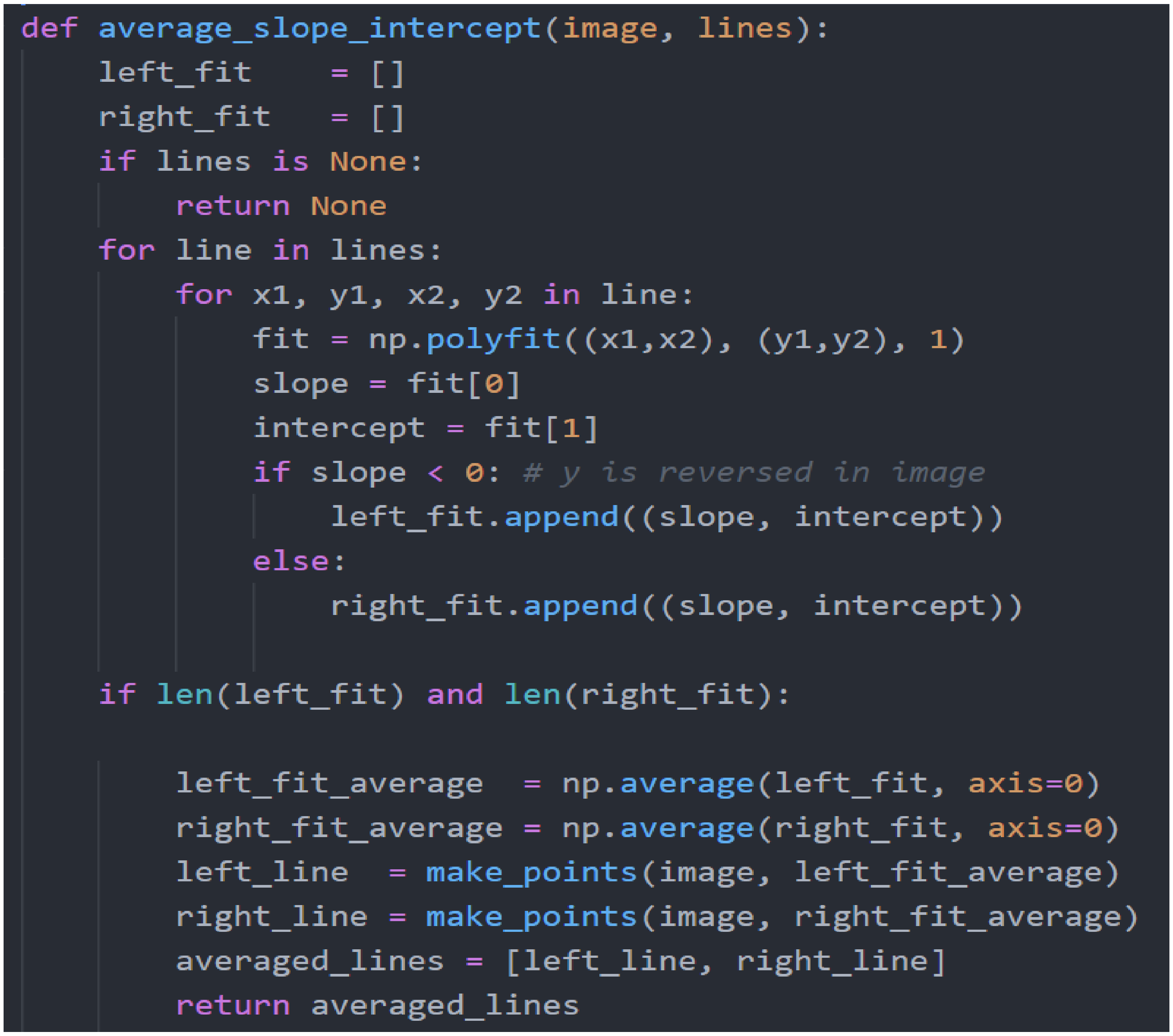
▪ Then, we added the above obtained image with the original image to see how correctly the Lane lines are detected.

▪ For that we observed the following results.



- As we can see in the above image, we successfully detected the lane lines, but we can notice that the detected lane lines are not smooth enough and their length is inconsistent.
- We can eliminate this issue by average_slope_intercept() function as shown below.

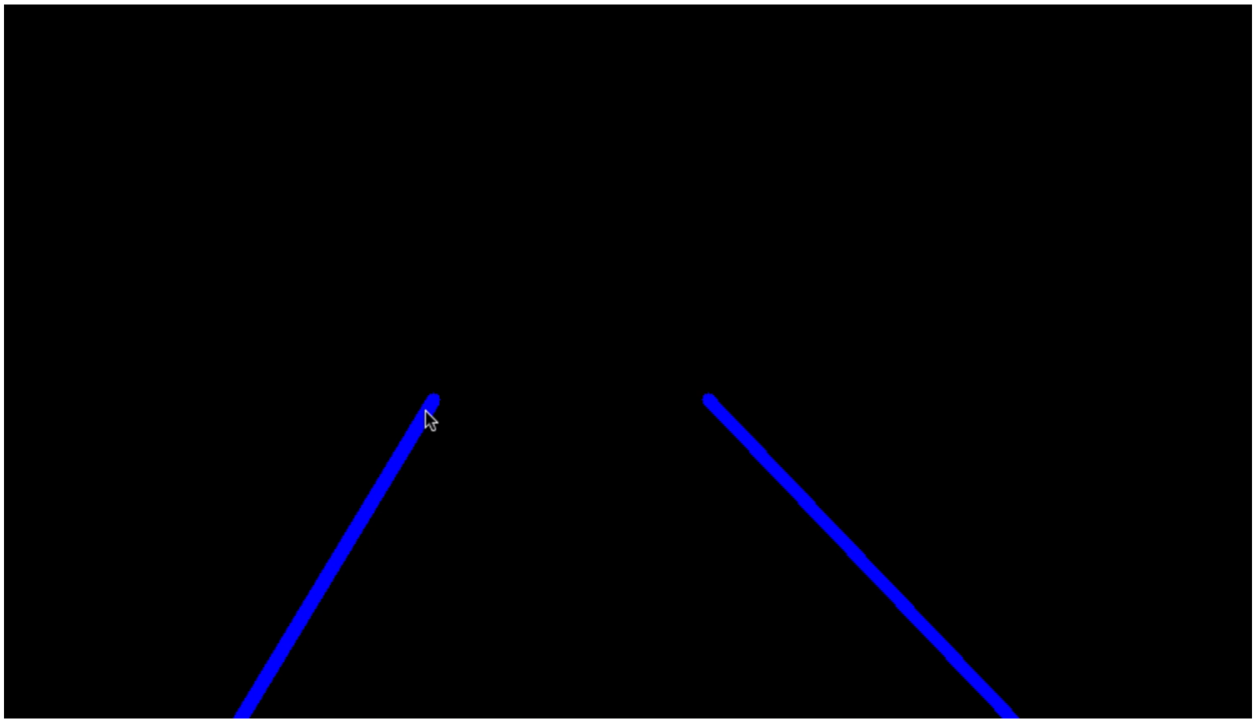
Step5: Optimizations



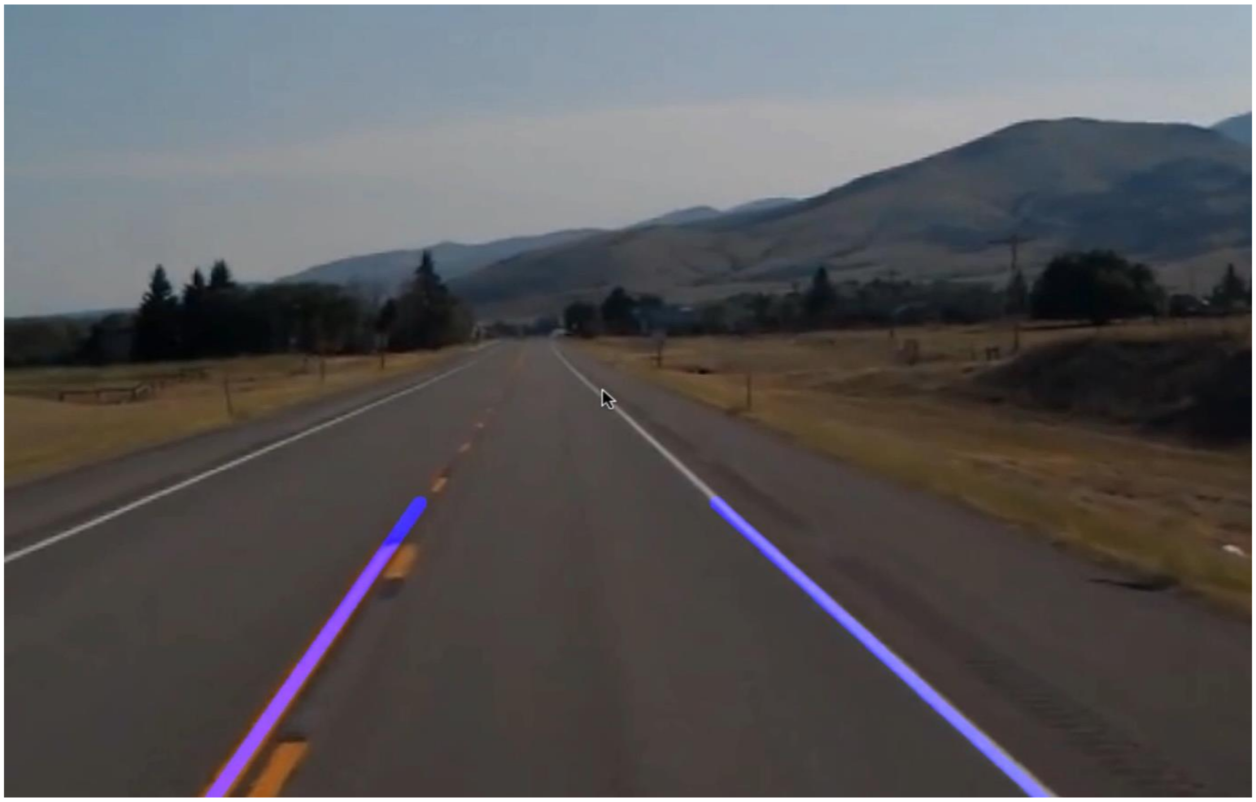
```python
def average_slope_intercept(image, lines):
    left_fit    = []
    right_fit   = []
    if lines is None:
        return None
    for line in lines:
        for x1, y1, x2, y2 in line:
            fit = np.polyfit((x1,x2), (y1,y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if slope < 0: # y is reversed in image
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))

    if len(left_fit) and len(right_fit):

        left_fit_average  = np.average(left_fit, axis=0)
        right_fit_average = np.average(right_fit, axis=0)
        left_line  = make_points(image, left_fit_average)
        right_line = make_points(image, right_fit_average)
        averaged_lines = [left_line, right_line]
        return averaged_lines
```

- left_fit will contain the coordinates of the line on left side lane.
- right_fit will contain the coordinates of the line on right side lane
- The polyfit() function gives us slope and the y-intercept for the given line, Last 1 indicates the ` degree of polynomianl.
- fit[0] returns slope at index 0
- fit[1] returns returns Y-intercept at index 1
- In our image, lines at the left side will have positive slope and lines in the right side will have positive slope
- Then we averaged out all the left and right side values using n.average() function.
- Function call to obtain new coordinates for averaged left line
- Then we passed these averaged (smoothened) lines to the make_points() function to obtain the new fixed length coordinates for the lane lines.

- After this optimization we obtained lane lines in the following form.



- And after combining with original image, we got the following result.



- Now as we successfully detected lane lines for the image file, let us try it for a video file.

For Video file:

- In this case also, we are going to use the same algorithm using VideoCapture() function of OpenCV.
- A video is nothing but continuous sequence of frames.
- Thus, we captured frames from the video file and used the same algorithm which we used for lane detection in image files and successfully detected the lane lines in the video file.

Conclusion:

In this project we accomplished our goal of detecting lane lines both in an image file as well as a video file, which is one of the up-most importance for cars which use the lane departure warning system and corrects the position of the vehicle with respect to the detected lane

References:

[1] S. Kammel and B. Pitzer, “Lidar-based lane marker detection and mapping,” *2008 IEEE Intelligent Vehicles Symposium*, Eindhoven, 2008, pp. 1137–1142. doi: 10.1109/IVS.2008.4621318

[2] Pan, X., Shi, J., Luo, P., Wang, X., & Tang, X. (2018). Spatial As Deep: Spatial CNN for Traffic Scene Understanding. In AAAI Conference on Artificial Intelligence (pp. 7276–7283). https://doi.org/10.2522/ptj.20090377

[3] Zhang, J., Xu, Y., Ni, B., & Duan, Z. (2018). Geometric Constrained Joint Lane Segmentation and Lane Boundary Detection. In European Conference on Computer Vision (pp. 37–47). https://doi.org/10.1007/978-3-030-01246-5_30

[4] M. Bertozzi and A. Broggi. Real-time lane and obstacle detection on the GOLD system. In Proceedings of Conference on Intelligent Vehicles, pages 213–218, 1996