LANE DETECTION

by

NAME: Shashank Pandey REGISTER NUMBER: 20MIA1147
NAME: Piyali Saha REGISTER NUMBER: 20MIA1066
NAME: Ayush Madurwar REGISTER NUMBER: 20MIA1009

A project report submitted to

Dr. Anushiya Rachel Gladston

SCOPE

in partial fulfilment of the requirements for the course of

SWE1010 – Digital Image Processing

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road

Chennai – 600127

NOVEMBER 2022

BONAFIDE CERTIFICATE

Certified that this project report entitled "**LANE DETECTION"** is a Bonafede work of

NAME: Shashank Pandey	REGISTER NUMBER: 20MIA1147
NAME: Piyali Saha	REGISTER NUMBER: 20MIA1066
NAME: Ayush Madurwar	REGISTER NUMBER: 20MIA1009

who carried out the Project work under my supervision and guidance for

SWE1010 – **Digital Image Processing**

Dr. Anushiya Rachel Gladston

School of Computer Science and

Engineering (SCOPE),

VIT University, Chennai

Chennai – 600127.

TABLE OF CONTENTS

SERIAL NO.	TITLE	PAGE NO.
1	ABSTRACT ACKNOWLEDEGMENT INTRODUCTION	4 5 6
2	METHODOLOGY	7 - 9
3	WORKING	10 - 14
4	CODE	15 - 23
5	FUTURE SCOPE	24 - 25
6	RESULT	26-27
7	CONCLUSIONS	28

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, Dr. Anushiya Rachel Gladston Professor, School of Computer Science and Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution

SHASHANK **PIYALI** **AYUSH**

ABSTRACT

Lane detection is a developing technology that is implemented in vehicles to enable autonomous navigation. Most lane detection systems are designed for roads with proper structure relying on the existence of markings. The main shortcoming of these approaches is that they might give inaccurate results or not work at all in situations involving unclear markings or the absence of them. In this study one such approach for detecting lanes on an unmarked road is reviewed followed by an improved approach. Both the approaches are based on digital image processing techniques and purely work on vision or camera data. The main aim is to obtain a real time curve value to assist the driver/autonomous vehicle for taking required turns and not go off the road.

INTRODUCTION

Intelligent Transportation System is the today's curious and advance technology the whole world is looking forward to. Many of the billion-dollar companies like Google, Uber, Tesla are trying in this field to invent fully autonomous vehicles. Image processing is one of the main drivers of automation, security and safety related application of the electronic industry. Image Significance of image processing and real time embedded systems applications are impacting the modern technology. Among the complex tasks of future self- driving safety vehicles is lane detection. Lane detection is the major module in self driving cars. Most of these implementations is actually of mixed domains where the coder needs to understand more advance technologies like AI or ML but in this paper, we tried using completely using image processing. If video is the intermediate part in lane detection, then obviously video is a sequence of images, if we can process images, it means indirectly, we are processing video, by the time we reach final processing of image then we directly apply it on each frame of the video, so the input video will be camera captured video and output will be lane Detected video. After researching many base papers, we found so many algorithms and techniques like Canny edge detection for edge detection and some filters to reduce noises occurs in images like Gaussian Filters and Gray Scale imaging for smoothing. Our Model is able to detect most straightened lines as well slight curvy lines, because fully curved lines is out of this scope. We will first take images and make a calculation which is feasible and working fine for all those images then we will pore it on to video stream. Lane line in the test images is in white or yellow we need to specify correct colour selections, so we need to mask the yellow and white colour in the process. The entire project is about how we are going to detect the road lanes lines So in this project it's all about how we can achieve this thing entirely using image processing without even knowing knowledge Machine Learning, Artificial Intelligence and any other extra concepts.

Objective of the Project:

This project is created to demonstrate how a lane detection system works on cars equipped with a front-facing camera. Finding a place in more and more vehicles, this system is an essential part of the advanced driver assistance systems (ADAS) used in autonomous/semi-autonomous vehicles. This feature is responsible for detecting lanes, measuring curve radius, and monitoring the offset from the center. With this information, the system significantly improves safety by making sure the vehicle is centered inside the lane lines, as well as adds comfort if it is also configured to control the steering wheel to take gentle curves on highways without any driver input. This is a simplified version of what is used in production vehicles, and the best functions if good conditions are provided (clear lane lines, stable light conditions). In this repository, it is included a dash cam footage for the script to work.

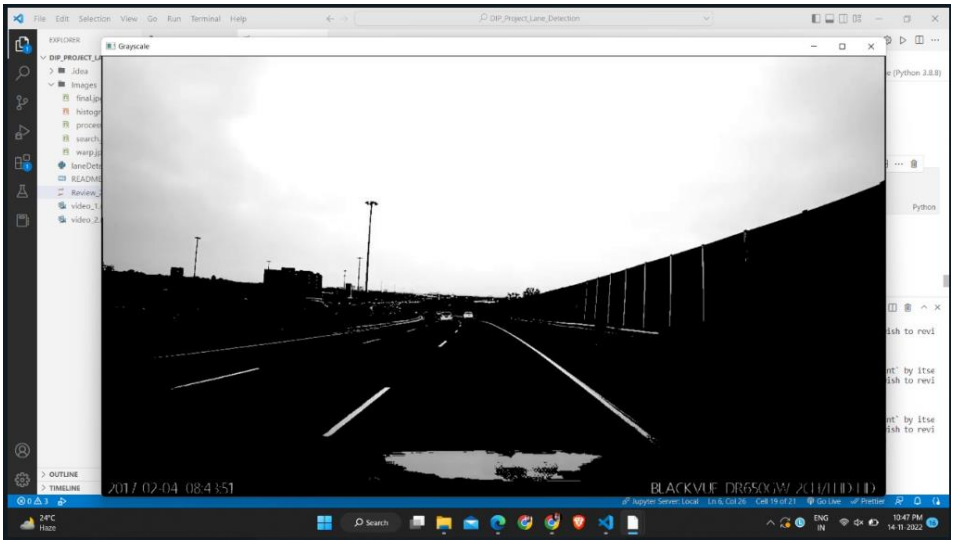
METHODOLOGY

After overall look up of so many techniques we choose a feasible, efficient, effective methodologies and algorithms we prepared a pipeline structure for the implementation part. Lane lines on roads are actually in white colour and yellow. We need to select the most fitting color space that clearly highlights the lane lines. We decided to apply HSL color selection because after researching all we found out that using HSL will be the best colour space to use and we mask their yellow and white color space. Then we apply Canny Edge detection for edge detection and the algorithm has been already explained in short above. We're interested in the area facing the camera, where the lane lines are found. So, we'll apply region of interest to cut out everything which we not required else.

Phases of Image processing:

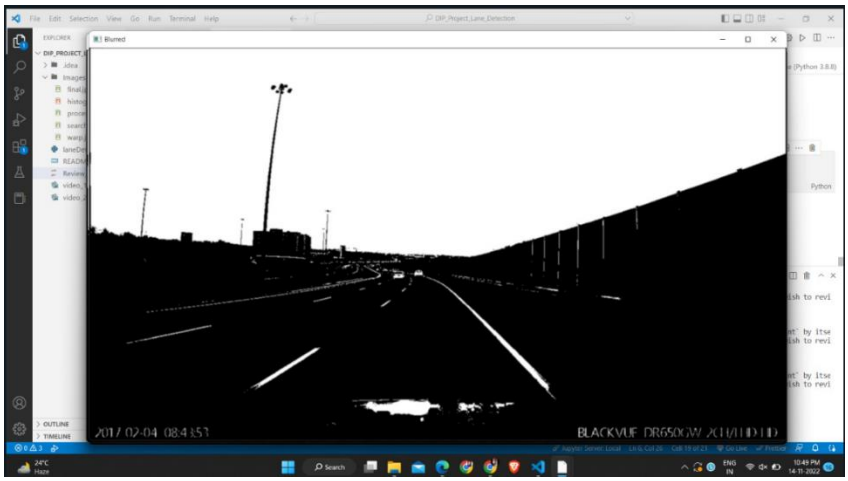
1. Gray Scale

For a grayscale image, Background Threshold is set to (0,0,0) by default, corresponding to the black color, and for a color edge a pixel is viewed as a pixel that is part of an object if any of the RGB values are greater than the corresponding value in the Background Threshold.



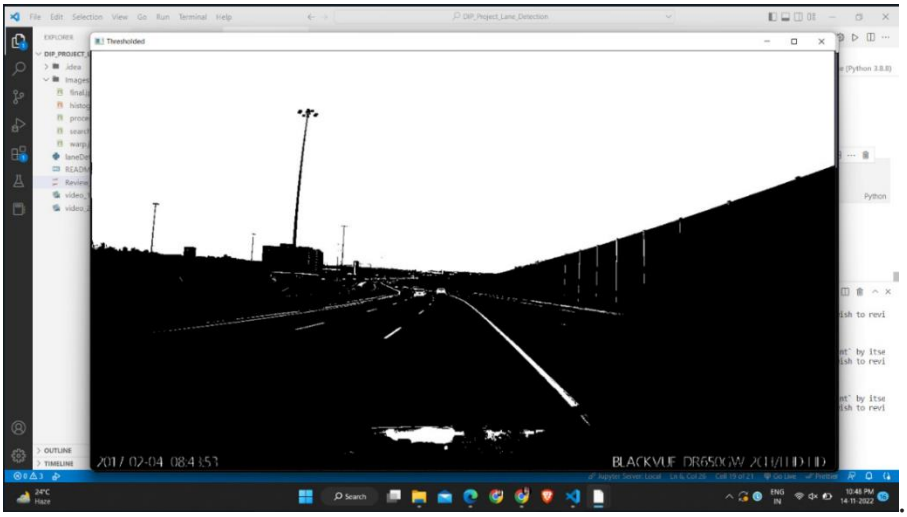
2. Gaussian Blur

Gaussian Blur is a pre-processing technique used to smoothen the edges in an image to reduce noise. This step is needed to reduce the number of lines we detect, as we only want to focus on the most significant lines, not on those other than road lanes. We must be careful as to not blur the images too much otherwise it will become hard to make up a line. In Open- cv built-in function of Gaussian Blur takes an integer kernel parameter that represents the intensity of the smoothing.



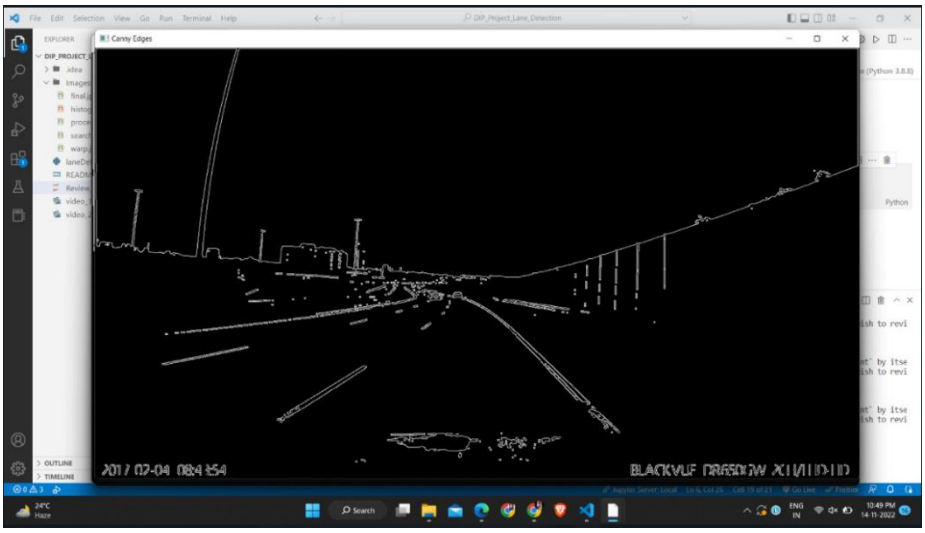
3. Threshold

Thresholding is a type of image segmentation, where we change the pixels of an image to make the image easier to analyze. In thresholding, we convert an image from colour or grayscale into a binary image, i.e., one that is simply black and white. Most frequently, we use thresholding as a way to select areas of interest of image, while ignoring the parts we are not concerned.



4. Canny Edge Detector

Canny Edge Detector finds edges (sharp brightness changes) and discards irrelevant data. The resulting image is wiry, which helps us focus on lane detection as we care about lines. The OpenCV built-in function requires a blurred image and two thresholds to include or exclude edges. A threshold measures a point's changing intensity. Any point beyond the high threshold will be included in our final image, and points between the thresholds will be included if they're near to edges beyond the high threshold. Low-quality edges are discarded. Low and high criteria are 50 and 150.



WORKINGS :

Image Processing

1.readVideo ()

- First up is the readVideo () function to access the video file which is located in the same directory.

2.processImage ()

- This function performs some processing techniques to isolate white lane lines and prepare it to be further analysed by the upcoming functions. Basically, it applies HLS color filtering to filter out whites in the frame, then converts it to grayscale which then is applied thresholding to get rid of unnecessary detections other than lanes, gets blurred and finally edges are extracted with cv2.Canny() function. The canny edge detection first removes noise from image by smoothening. It then finds the image gradient to highlight regions with high spatial derivatives.

3.perspectiveWarp ()

- Now that we have the image we want, a perspective warp is applied. 4 points are placed on the frame such that they surround only the area in which lanes are present, then maps it onto another matrix to create a Birdseye look at the lanes. This will enable us to work with a much-refined image and help detect lane curvatures. It should be noted that this operation is subject to change if another video is used. The predefined 4 points are calculated with this particular footage in mind. It should be returned if another video has a slightly different angled camera.

Lane Detection, Curve Fitting

1. Plot Histogram ()

- Plotting a histogram for the bottom half of the image is an essential part to obtain the information of where exactly the left and right lanes start. Upon analysing the histogram, one can see there is two distinct peaks where all the white pixels are detected. A very good indicator of where the left and right lanes begin. Since the histogram x coordinate represent the x coordinate of our analysed frame, it means we now have x coordinates to start searching for the lanes.

2. Slidewindow Search ()

- A sliding window approach is used to detect lanes and their curvature. It uses information from the previous histogram function and puts a box with lane at the center. Then puts another box on top based on the positions of white pixels from the previous box and places itself accordingly all the way to the top of the frame. This way, we have the information to make some calculations. Then, a second-degree polynomial fit is performed to have a curve fit in pixel space.

3. General Search ()

- After running the slide_window_search () function, this general_search () function is now able to fill up an area around those detected lanes, again applies the second-degree polyfit to then draw a yellow line which overlaps the lanes pretty accurately. This line will be used to measure radius of curvature which is essential in predicting steering angles.

4. measure_lane_curvature ()

- With information provided by the previous two functions, np. polyfit () function is used again but with the values multiplied by xm_per_pix and ym_per_pix variables to convert them from pixel space to meter space. xm_per_pix are set as 3.7 / 720 which lane width as 3.7 meters and left & right lane base x coordinates obtained from histogram corresponds to lane width in pixels which turns out to be approximately 720 pixels. Similarly, ym_per_pix are set to 30 / 720 since the frame height is 720.

Visualization and Main Function

1. draw_lane_lines ()

- From here on, some methods are applied to visualize the detected lanes and other information to be displayed for the final image. This particular function takes detected lanes and fills the area inside them with a green color. It also visualizes the center of the lane by taking the mean of left_fitx and right_fitx lists and storing them in pts_mean variable, which then is represented by a yellowish color. This variable is also used to calculate the offset of the vehicle to either side or of it is centered in the lane.

2. offCenter ()

- Off-Center() function uses pts_mean variable to calculate the offset value and show it in meter space.

3. addText ()

- Finally, by adding text on the final image would complete the process and the information displayed.

4. main ()

- Main function is where all these functions are called in the correct order and contains the loop to play video.

Requirements

Pre-requisites

- Python 3.6 or higher
- OpenCV
- Numpy
- Scipy
- matplotlib
- skimage

Finalized image

Code

```
import cv2
import numpy as np
import os
from scipy import optimize
from matplotlib import pyplot as plt, cm, colors

# Defining variables to hold meter-to-pixel conversion
ym_per_pix = 30 / 720
# Standard lane width is 3.7 meters divided by lane width in pixels which is
# calculated to be approximately 720 pixels not to be confused with frame height
xm_per_pix = 3.7 / 720

# Get path to the current working directory
CWD_PATH = os.getcwd()




# FUNCTION TO READ AN INPUT IMAGE
def readVideo():

    # Read input video from current working directory
    inpImage = cv2.VideoCapture(os.path.join(CWD_PATH, 'video_2.mp4'))

    return inpImage


# FUNCTION TO PROCESS IMAGE
def processImage(inpImage):

    # Apply HLS color filtering to filter out white lane lines
    hls = cv2.cvtColor(inpImage, cv2.COLOR_BGR2HLS)
    lower_white = np.array([0, 160, 10])
    upper_white = np.array([255, 255, 255])
    mask = cv2.inRange(inpImage, lower_white, upper_white)
    hls_result = cv2.bitwise_and(inpImage, inpImage, mask = mask)

    # Convert image to grayscale, apply threshold, blur & extract edges
    gray = cv2.cvtColor(hls_result, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 160, 255, cv2.THRESH_BINARY)
    blur = cv2.GaussianBlur(thresh,(3, 3), 0)
    canny = cv2.Canny(blur, 40, 60)

    # Display the processed images
##    cv2.imshow("Image", inpImage)
##    cv2.imshow("HLS Filtered", hls_result)
##    cv2.imshow("Grayscale", gray)
##    cv2.imshow("Thresholded", thresh)
##    cv2.imshow("Blurred", blur)
##    cv2.imshow("Canny Edges", canny)

    return image, hls_result, gray, thresh, blur, canny


#  FUNCTION TO APPLY PERSPECTIVE WARP
```

```
def perspectiveWarp(inpImage):

    # Get image size
    img_size = (inpImage.shape[1], inpImage.shape[0])

    # Perspective points to be warped
    src = np.float32([[590, 440],
                      [690, 440],
                      [200, 640],
                      [1000, 640]])

    # Window to be shown
    dst = np.float32([[200, 0],
                      [1200, 0],
                      [200, 710],
                      [1200, 710]])

    # Matrix to warp the image for birdseye window
    matrix = cv2.getPerspectiveTransform(src, dst)
    # Inverse matrix to unwarp the image for final window
    minv = cv2.getPerspectiveTransform(dst, src)
    birdseye = cv2.warpPerspective(inpImage, matrix, img_size)

    # Get the birdseye window dimensions
    height, width = birdseye.shape[:2]

    # Divide the birdseye view into 2 halves to separate left & right lanes
    birdseyeLeft  = birdseye[0:height, 0:width // 2]
    birdseyeRight = birdseye[0:height, width // 2:width]

    # Display birdseye view image
    # cv2.imshow("Birdseye" , birdseye)
    # cv2.imshow("Birdseye Left" , birdseyeLeft)
    # cv2.imshow("Birdseye Right", birdseyeRight)

    return birdseye, birdseyeLeft, birdseyeRight, minv


# FUNCTION TO PLOT THE HISTOGRAM OF WARPED IMAGE
def plotHistogram(inpImage):

    histogram = np.sum(inpImage[inpImage.shape[0] // 2:, :], axis = 0)

    midpoint = np.int(histogram.shape[0] / 2)
    leftxBase = np.argmax(histogram[:midpoint])
    rightxBase = np.argmax(histogram[midpoint:]) + midpoint

    plt.xlabel("Image X Coordinates")
    plt.ylabel("Number of White Pixels")

    # Return histogram and x-coordinates of left & right lanes to calculate
    # lane width in pixels
    return histogram, leftxBase, rightxBase


# APPLY SLIDING WINDOW METHOD TO DETECT CURVES
def slide_window_search(binary_warped, histogram):

    # Find the start of left and right lane lines using histogram info
    out_img = np.dstack((binary_warped, binary_warped, binary_warped)) * 255
```

```
midpoint = np.int(histogram.shape[0] / 2)
leftx_base = np.argmax(histogram[:midpoint])
rightx_base = np.argmax(histogram[midpoint:]) + midpoint

# A total of 9 windows will be used
nwindows = 9
window_height = np.int(binary_warped.shape[0] / nwindows)
nonzero = binary_warped.nonzero()
nonzeroy = np.array(nonzero[0])
nonzerox = np.array(nonzero[1])
leftx_current = leftx_base
rightx_current = rightx_base
margin = 100
minpix = 50
left_lane_inds = []
right_lane_inds = []

# Loop to iterate through windows and search for lane lines
for window in range(nwindows):
    win_y_low = binary_warped.shape[0] - (window + 1) * window_height
    win_y_high = binary_warped.shape[0] - window * window_height
    win_xleft_low = leftx_current - margin
    win_xleft_high = leftx_current + margin
    win_xright_low = rightx_current - margin
    win_xright_high = rightx_current + margin
    cv2.rectangle(out_img, (win_xleft_low, win_y_low), (win_xleft_high, win_y_high),
    (0,255,0), 2)
    cv2.rectangle(out_img, (win_xright_low,win_y_low), (win_xright_high,win_y_high),
    (0,255,0), 2)
    good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
    (nonzerox >= win_xleft_low) &  (nonzerox < win_xleft_high)).nonzero()[0]
    good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
    (nonzerox >= win_xright_low) &  (nonzerox < win_xright_high)).nonzero()[0]
    left_lane_inds.append(good_left_inds)
    right_lane_inds.append(good_right_inds)
    if len(good_left_inds) > minpix:
        leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
    if len(good_right_inds) > minpix:
        rightx_current = np.int(np.mean(nonzerox[good_right_inds]))

left_lane_inds = np.concatenate(left_lane_inds)
right_lane_inds = np.concatenate(right_lane_inds)

leftx = nonzerox[left_lane_inds]
lefty = nonzeroy[left_lane_inds]
rightx = nonzerox[right_lane_inds]
righty = nonzeroy[right_lane_inds]

# Apply 2nd degree polynomial fit to fit curves
left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx, 2)


ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0])
left_fitx = left_fit[0] * ploty**2 + left_fit[1] * ploty + left_fit[2]
right_fitx = right_fit[0] * ploty**2 + right_fit[1] * ploty + right_fit[2]

ltx = np.trunc(left_fitx)
rtx = np.trunc(right_fitx)
plt.plot(right_fitx)
```

```
    plt.show()

    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]

    plt.imshow(out_img)
    plt.plot(left_fitx,  ploty, color = 'yellow')
    plt.plot(right_fitx, ploty, color = 'yellow')
    plt.xlim(0, 1280)
    plt.ylim(720, 0)

    return ploty, left_fit, right_fit, ltx, rtx

#  APPLY GENERAL SEARCH METHOD TO DETECT CURVES

def general_search(binary_warped, left_fit, right_fit):

    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    margin = 100
    left_lane_inds = ((nonzerox > (left_fit[0]*(nonzeroy**2) + left_fit[1]*nonzeroy +
    left_fit[2] - margin)) & (nonzerox < (left_fit[0]*(nonzeroy**2) +
    left_fit[1]*nonzeroy + left_fit[2] + margin)))

    right_lane_inds = ((nonzerox > (right_fit[0]*(nonzeroy**2) + right_fit[1]*nonzeroy +
    right_fit[2] - margin)) & (nonzerox < (right_fit[0]*(nonzeroy**2) +
    right_fit[1]*nonzeroy + right_fit[2] + margin)))

    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0])
    left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
    right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]


    ## VISUALIZATION

    out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
    window_img = np.zeros_like(out_img)
    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]

    left_line_window1 = np.array([np.transpose(np.vstack([left_fitx-margin, ploty]))])
    left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx+margin,
                                  ploty])))])
    left_line_pts = np.hstack((left_line_window1, left_line_window2))
    right_line_window1 = np.array([np.transpose(np.vstack([right_fitx-margin, ploty]))])
    right_line_window2 = np.array([np.flipud(np.transpose(np.vstack([right_fitx+margin,
ploty])))])
    right_line_pts = np.hstack((right_line_window1, right_line_window2))

    cv2.fillPoly(window_img, np.int_([left_line_pts]), (0, 255, 0))
    cv2.fillPoly(window_img, np.int_([right_line_pts]), (0, 255, 0))
    result = cv2.addWeighted(out_img, 1, window_img, 0.3, 0)
```

```
    # plt.imshow(result)
    plt.plot(left_fitx,  ploty, color = 'yellow')
    plt.plot(right_fitx, ploty, color = 'yellow')
    plt.xlim(0, 1280)
    plt.ylim(720, 0)

    ret = {}
    ret['leftx'] = leftx
    ret['rightx'] = rightx
    ret['left_fitx'] = left_fitx
    ret['right_fitx'] = right_fitx
    ret['ploty'] = ploty

    return ret


# FUNCTION TO MEASURE CURVE RADIUS
def measure_lane_curvature(ploty, leftx, rightx):

    leftx = leftx[::-1]  # Reverse to match top-to-bottom in y
    rightx = rightx[::-1]  # Reverse to match top-to-bottom in y

    # Choose the maximum y-value, corresponding to the bottom of the image
    y_eval = np.max(ploty)

    # Fit new polynomials to x, y in world space
    left_fit_cr = np.polyfit(ploty*ym_per_pix, leftx*xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty*ym_per_pix, rightx*xm_per_pix, 2)

    # Calculate the new radii of curvature
    left_curverad  = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) /
np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5)
/ np.absolute(2*right_fit_cr[0])
    # Now our radius of curvature is in meters
    # print(left_curverad, 'm', right_curverad, 'm')

    # Decide if it is a left or a right curve
    if leftx[0] - leftx[-1] > 60:
        curve_direction = 'Left Curve'
    elif leftx[-1] - leftx[0] > 60:
        curve_direction = 'Right Curve'
    else:
        curve_direction = 'Straight'

    return (left_curverad + right_curverad) / 2.0, curve_direction

# FUNCTION TO VISUALLY SHOW DETECTED LANES AREA
def draw_lane_lines(original_image, warped_image, Minv, draw_info):

    leftx = draw_info['leftx']
    rightx = draw_info['rightx']
    left_fitx = draw_info['left_fitx']
    right_fitx = draw_info['right_fitx']
    ploty = draw_info['ploty']

    warp_zero = np.zeros_like(warped_image).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

    pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
```

```
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
    pts = np.hstack((pts_left, pts_right))

    mean_x = np.mean((left_fitx, right_fitx), axis=0)
    pts_mean = np.array([np.flipud(np.transpose(np.vstack([mean_x, ploty])))])

    cv2.fillPoly(color_warp, np.int_([pts]), (0, 255, 0))
    cv2.fillPoly(color_warp, np.int_([pts_mean]), (0, 255, 255))

    newwarp = cv2.warpPerspective(color_warp, Minv, (original_image.shape[1],
original_image.shape[0]))
    result = cv2.addWeighted(original_image, 1, newwarp, 0.3, 0)

    return pts_mean, result



#FUNCTION TO CALCULATE DEVIATION FROM LANE CENTER
def offCenter(meanPts, inpFrame):

    # Calculating deviation in meters
    mpts = meanPts[-1][-1][-2].astype(int)
    pixelDeviation = inpFrame.shape[1] / 2 - abs(mpts)
    deviation = pixelDeviation * xm_per_pix
    direction = "left" if deviation < 0 else "right"

    return deviation, direction




#FUNCTION TO ADD INFO TEXT TO FINAL IMAGE

def addText(img, radius, direction, deviation, devDirection):

    # Add the radius and center position to the image
    font = cv2.FONT_HERSHEY_TRIPLEX

    if (direction != 'Straight'):
        text = 'Radius of Curvature: ' + '{:04.0f}'.format(radius) + 'm'
        text1 = 'Curve Direction: ' + (direction)

    else:
        text = 'Radius of Curvature: ' + 'N/A'
        text1 = 'Curve Direction: ' + (direction)

    cv2.putText(img, text , (50,100), font, 0.8, (0,100, 200), 2, cv2.LINE_AA)
    cv2.putText(img, text1, (50,150), font, 0.8, (0,100, 200), 2, cv2.LINE_AA)

    # Deviation
    deviation_text = 'Off Center: ' + str(round(abs(deviation), 3)) + 'm' + ' to the ' +
devDirection
    cv2.putText(img, deviation_text, (50, 200), cv2.FONT_HERSHEY_TRIPLEX, 0.8, (0,100, 200),
2, cv2.LINE_AA)

    return img

######## END - FUNCTIONS TO PERFORM IMAGE PROCESSING ############################
```

```
# MAIN FUNCTION

# Read the input image
image = readVideo()

#
# LOOP TO PLAY THE INPUT IMAGE

while True:

    _, frame = image.read()


    # Apply perspective warping by calling the "perspectiveWarp()" function
    # Then assign it to the variable called (birdView)
    # Provide this function with:
    # 1- an image to apply perspective warping (frame)
    birdView, birdViewL, birdViewR, minverse = perspectiveWarp(frame)


    # Apply image processing by calling the "processImage()" function
    # Then assign their respective variables (img, hls, grayscale, thresh, blur, canny)
    # Provide this function with:
    # 1- an already perspective warped image to process (birdView)
    img, hls, grayscale, thresh, blur, canny = processImage(birdView)
    imgL, hlsL, grayscaleL, threshL, blurL, cannyL = processImage(birdViewL)
    imgR, hlsR, grayscaleR, threshR, blurR, cannyR = processImage(birdViewR)


    # Plot and display the histogram by calling the "get_histogram()" function
    # Provide this function with:
    # 1- an image to calculate histogram on (thresh)
    hist, leftBase, rightBase = plotHistogram(thresh)
    print(rightBase - leftBase)
    plt.plot(hist)
    plt.show()


    ploty, left_fit, right_fit, left_fitx, right_fitx = slide_window_search(thresh, hist)
    plt.plot(left_fit)
    plt.show()


    draw_info = general_search(thresh, left_fit, right_fit)
    plt.show()


    curveRad, curveDir = measure_lane_curvature(ploty, left_fitx, right_fitx)


    # Filling the area of detected lanes with green
    meanPts, result = draw_lane_lines(frame, thresh, minverse, draw_info)


    deviation, directionDev = offCenter(meanPts, frame)


    # Adding text to our final image
    finalImg = addText(result, curveRad, curveDir, deviation, directionDev)
```

```
    # Displaying final image
    cv2.imshow("Final", finalImg)


    # Wait for the ENTER key to be pressed to stop playback
    if cv2.waitKey(1) == 13:
        break



# Cleanup
image.release()
cv2.destroyAllWindows()
```

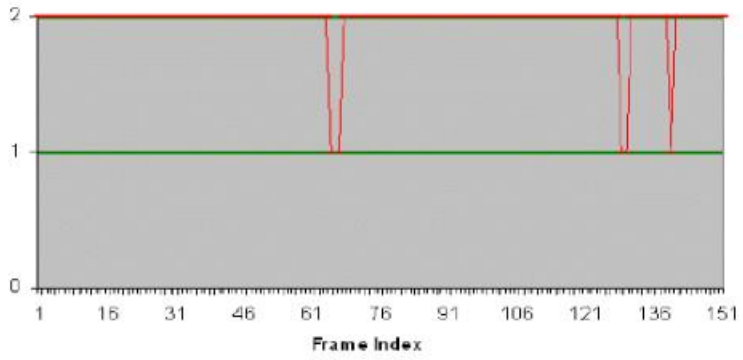
FUTURE SCOPE

As the image processing part of our project is kept very simple and can fail to identify lanes in poor or unstable light conditions. Lane lines must be clearly visible and tighter curves can cause problems with fitting curves and visualization. If we were to compare this project to one of the production grade versions, things get significantly more sophisticated as error rates should be extremely low and the system should be able to adapt to various situations. For example, unstable light conditions where there is huge amount of change in light density, or weather conditions that has impact on road surface visibility etc. Many variables are hardcoded in this project where any change could lead this setup to failure.

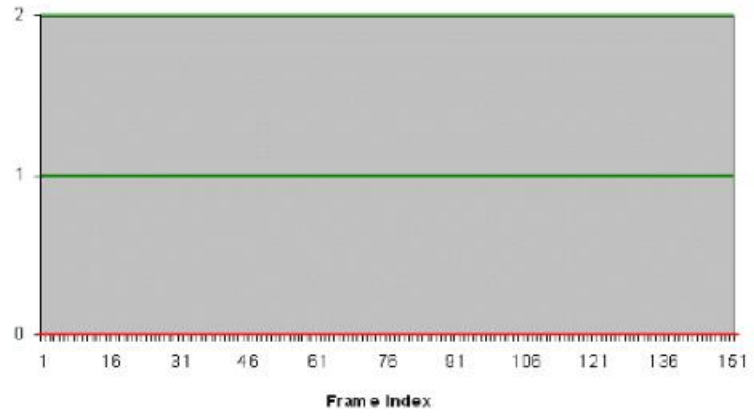
That's why methods like machine learning should be used to make this system more adaptive and less prone to fail in real life conditions so we can we reduce the errors as much as possible and train the data for every good and worst situation using Machine learning because till now, we made our project from the perspective of Image Processing. Further we will try to implement machine learning in this project.

RESULT

The performance of the workings is evaluated qualitatively in terms of accuracy in the localization of the lane boundaries for 150 frames in each case. This tier performance metric per input frame is strictly a pass/fail vote based on the likelihood that a vehicle could conceivably navigate with the output hyperbola pairs. It has been developed from the algorithm where accurate lane detection is marked by a red line at 2 and once it drops down to 1 it indicates the fault lane detection at that frame sequence. However, the evaluation ranged from straight highway to curved normal country roads, during day and night. A few frames had been shown as traffic in road types, Railway Bridge, and critical weather condition such as heavy rain and cloudy sky.



Lane detection in highway during day time per single frame



Lane detection in highway during night time per single frame

Curved and straight road lane detection in a highway during night

Curved and straight road lane detection in a highway during day

CONCLUSION

In this project, we proposed a new lane detection preprocessing and ROI selection methods to design a lane detection system. The main idea is to add white extraction before the conventional basic preprocessing. Edge extraction has also been added during the preprocessing stage to improve lane detection accuracy. We also placed the ROI selection after the proposed preprocessing. Compared with selecting the ROI in the original image, it reduced the non-lane parameters and improved the accuracy of lane detection.

We have chosen best technologies and learnt the importance of famous algorithms. The working scenario has been learnt and each research paper has its own uniqueness and has its own advantages and disadvantages as well. By considering all those things we wisely took the necessary technologies and methodologies required for our project .There are many challenges to be addressed, to overcome all these challenges we have proposed an pipeline architecture which will lead to the proper output and destiny of our lane detection .We choose canny Edge detection, Gray Scale, Gaussian Blur, Threshold, HSL transform , The reason to choose those algorithms is that they are very fast efficient and help us for real time embedded implementation. We also applied so many other image processing techniques like filters, color, selection, scaling images for different purposes in our project which determines the result. In order to search out for the left and right vector points that represent the road lanes, the lane scan boundary phase uses the edge image to effectively allocate the lane points.