
LANE DETECTION

TEAM MEMBERS

TEAM MEMBERS

SHASHANK PANDEY(20MIA1147)

PIYALI SAHA(20MIA1066)

AYUSH MADURWAR(20MIA1009)

PROJECT DESCRIPTION

This project demonstrates lane detection on autos with a front-facing camera. This system is vital to autonomous/semi-autonomous vehicles' advanced driver assistance systems (ADAS). This feature detects lanes, measures curve radius, and monitors offset. With this information, the system enhances safety by keeping the car centred within the lane lines and adds comfort if it's designed to negotiate easy turns on highways without human input. This simplified version of what's in production automobiles works best in good conditions (clear lane lines, stable light conditions). This repository contains dash cam footage for the script.

WORKING

Image Processing

1.readVideo()

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

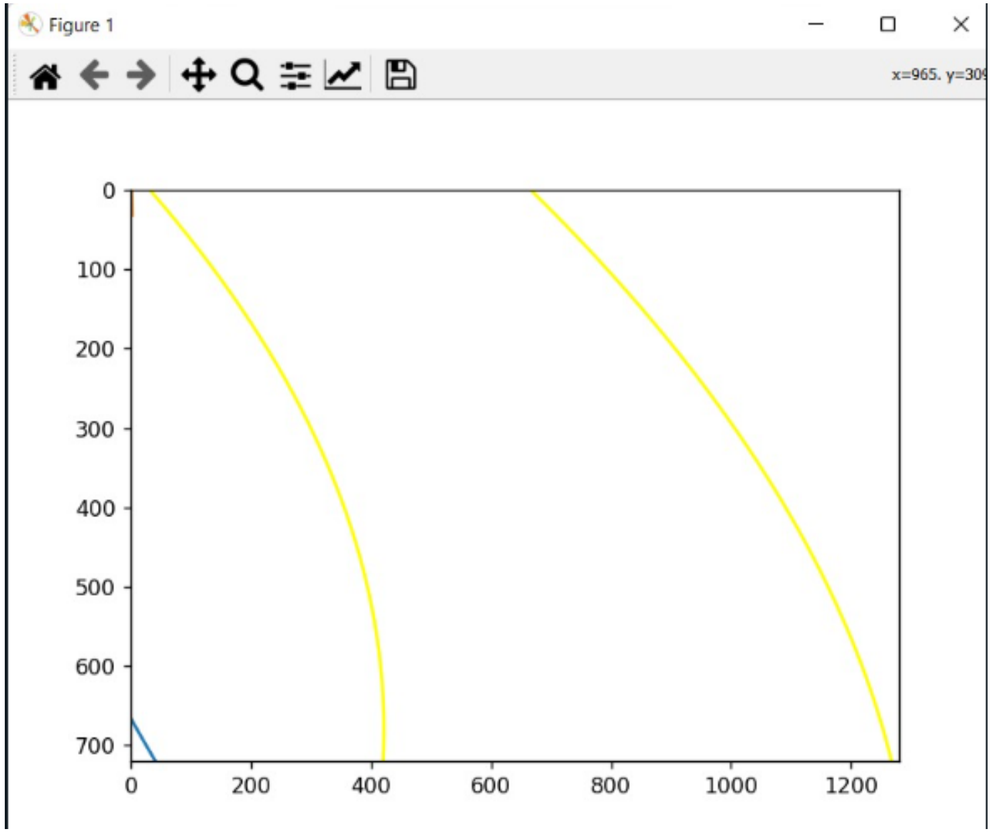
2.processImage()

- This function performs some processing techniques to isolate white lane lines and prepare it to be further analyzed 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.

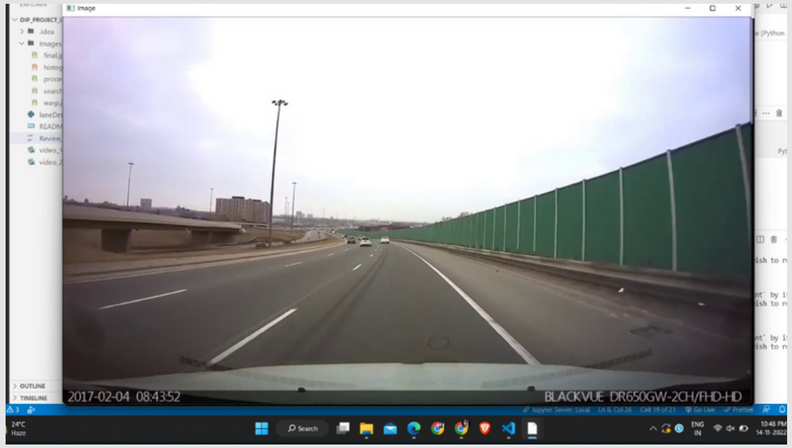
WORKING

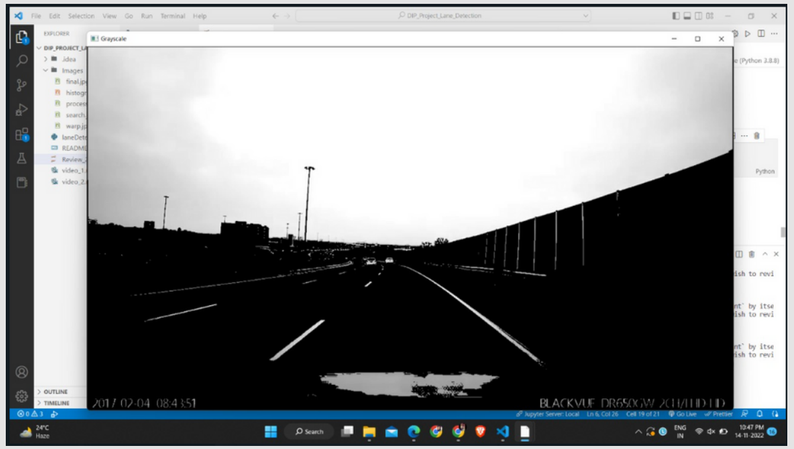
3.perspectiveWarp()

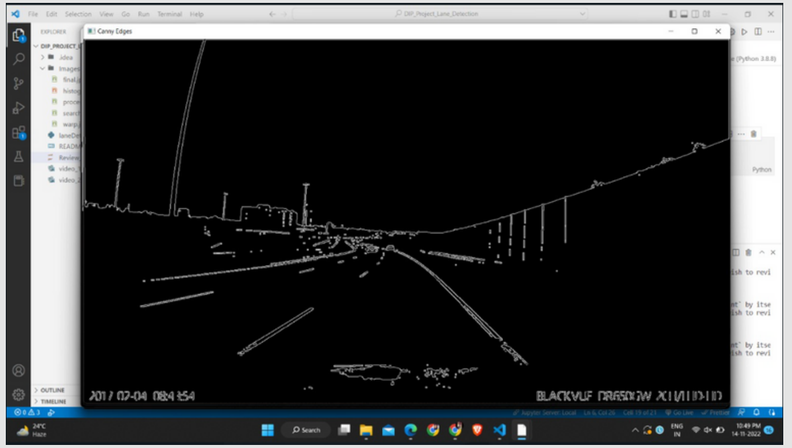
- Now that we have the image we want, a perspective warp is applied. 4 points are placed on the frame such that they surround only the area in which lanes are present, then maps it onto another matrix to create a birdseye look at the lanes. This will enable us to work with a much-refined image and help 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.



PHASES OF IMAGE PROCESSING







PHASES OF IMAGE PROCESSING

2. Gray Scale

- For a grayscale image, BackgroundTreshold 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 BackgroundTreshold.

PHASES OF IMAGE PROCESSING

3. 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 Opencv built-in function of Gaussian Blur takes an integer kernel parameter that represents the intensity of the smoothing.

PHASES OF IMAGE PROCESSING

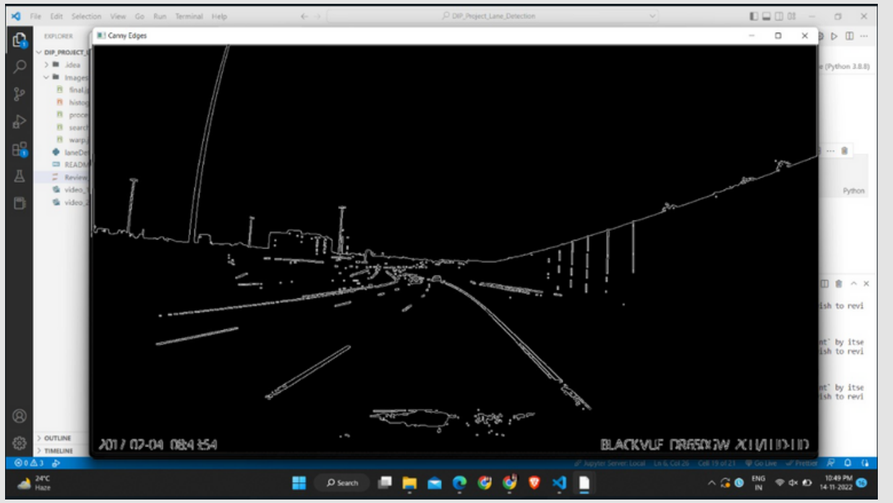
4. Threshold

- 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 an image, while ignoring the parts we are not concerned.

PHASES OF IMAGE PROCESSING

5. 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.

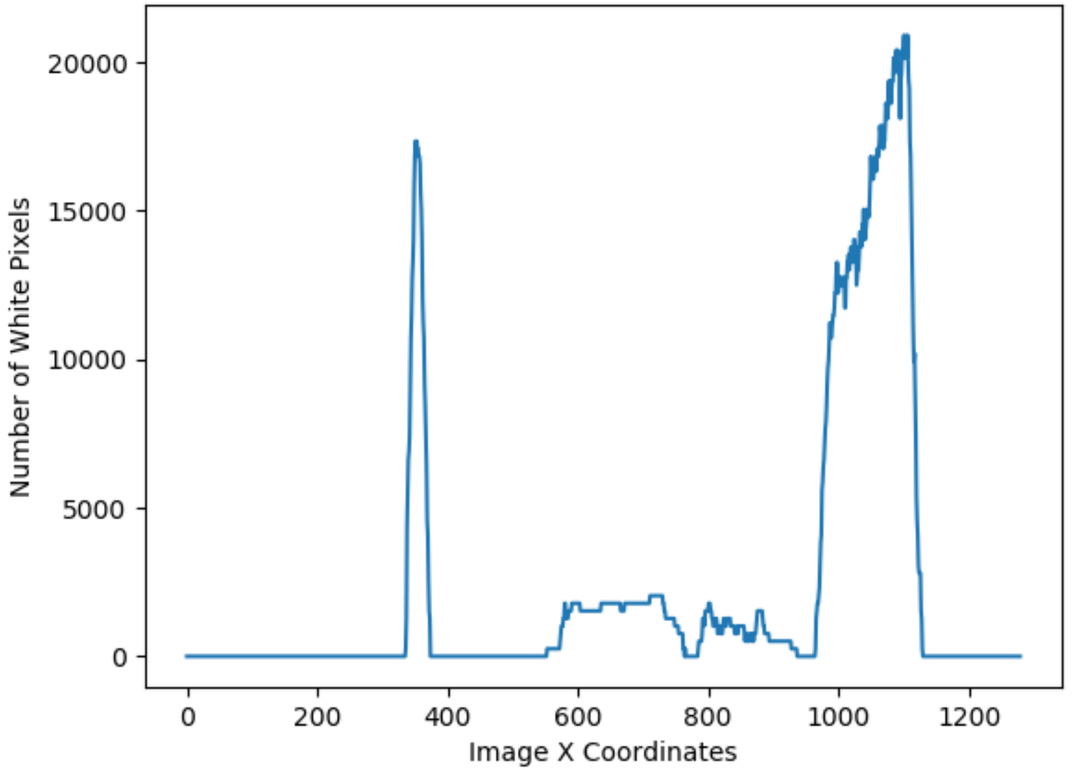


WORKING

Lane Detection, Curve Fitting

1. Plot Histogram()

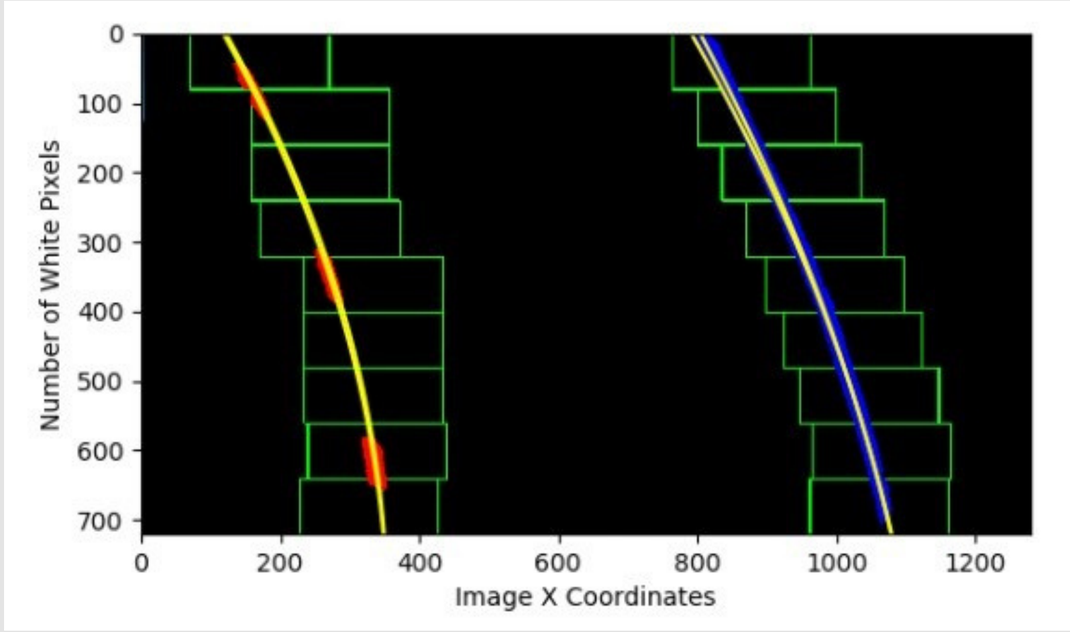
- Plotting a histogram for the bottom half of the image is an essential part to obtain the information of where exactly the left and right lanes start. Upon analyzing the histogram, one can see there is two distinct peaks where all the white pixels are detected. A very good indicator of where the left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



WORKING

2. Slidewindow Search()

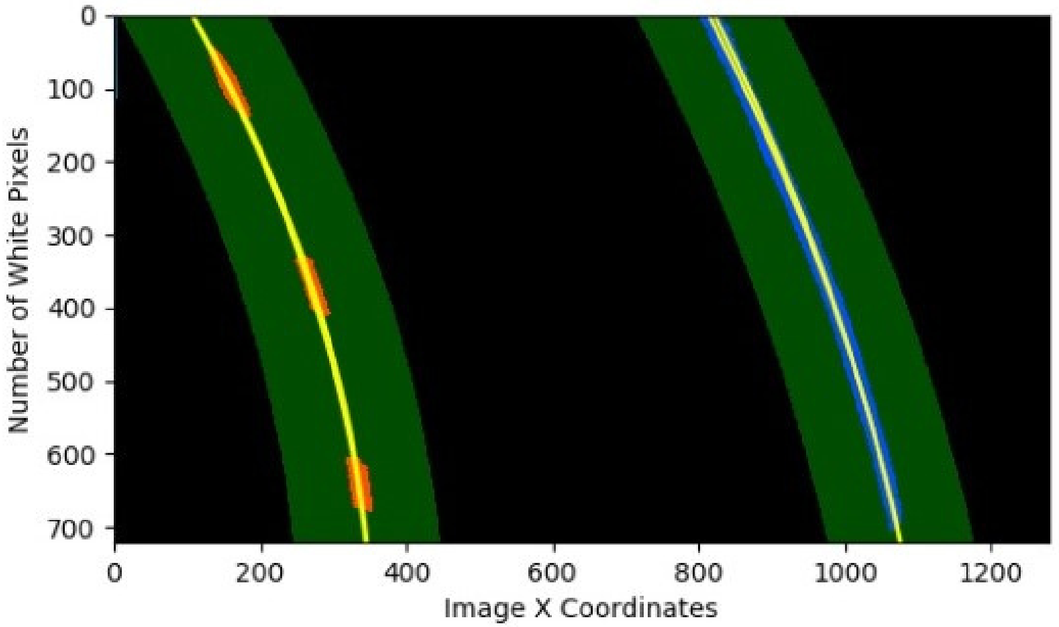
- A sliding window approach is used to detect lanes and their curvature. It uses information from the previous histogram function and puts a box with lane at the center. Then puts another box on top based on the positions of white pixels from the previous box and places itself accordingly all the way 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.



WORKING

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.



WORKING

4. Measuring 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 is 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 is set to 30 / 720 since the frame height is 720.

WORKING

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. Off Center

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

WORKING

3. Add Text

- 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

Prerequisites

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

THANK YOU