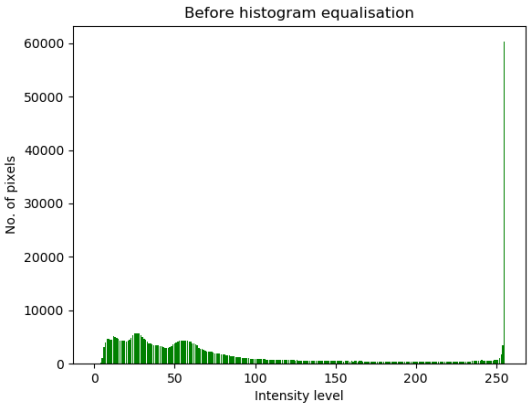
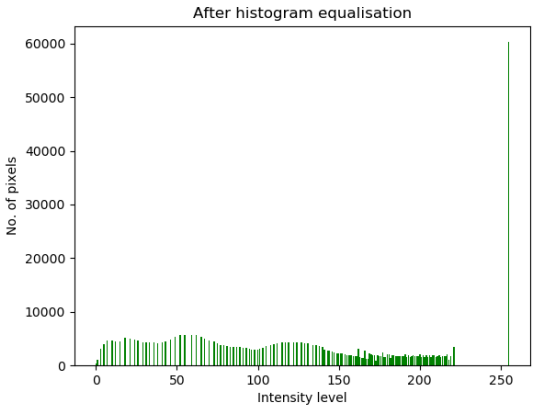
-Jeffin Johny Kachappilly
UID : 118293929

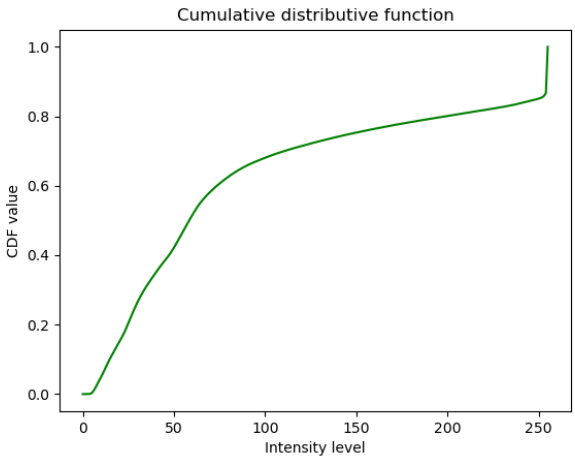
Solution 1.a : Histogram Equalization

- 'for' loop for accessing all given frames.
- Initialised numpy array (intensity_count) of size 256, to store number of pixels with same intensity (bin size = 255)
- Got the intensity value of each pixel using nested 'for' loop and incremented 'intensity_count' accordingly.
- Determined pixel fraction (pdf) for each intensity value, by dividing 'intensity_count' by total number of pixels of the image.
- Cumulative Distributive function(cdf), was calculated which represents the fraction of pixels with intensity less than or equal to current intensity.
- Multiplied 'cdf' with 255 and assigned the new intensity values to another image.

Here are the plots which represent the change in intensity values before and after equalisation.





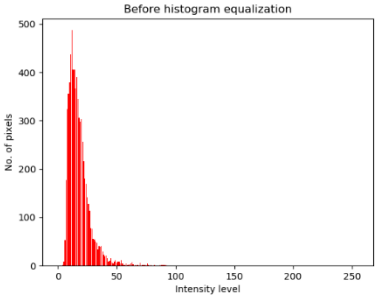


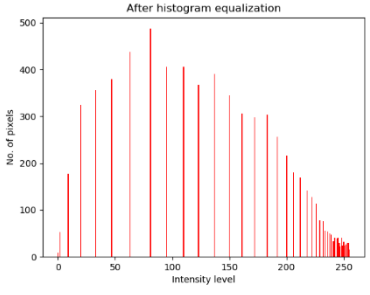
FINAL OUTPUT:

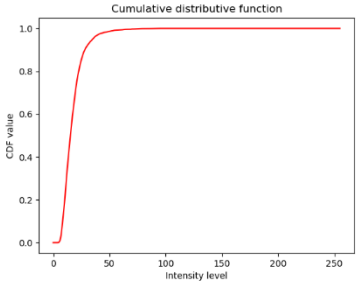
VIDEO_1 : https://youtu.be/6jTEr5xllfU

Solution 1.b : Adaptive Histogram Equalization

- ‘for’ loop for accessing all given frames.
- Split the image into 8 horizontal slices equally, followed by splitting of each slice into 8 equal vertical slice.
- Then each sub-array is considered for equalization.
- Initialised numpy array (intensity_count) of size 256, to store number of pixels with same intensity (bin size = 255)
- Got the intensity value of each pixel using nested ‘for’ loop and incremented ‘intensity_count’ accordingly.
- Determined pixel fraction (pdf) for each intensity value, by dividing ‘intensity_count’ by total number of pixels of the image.
- Cumulative Distributive function(cdf), was calculated which represents the fraction of pixels with intensity less than or equal to current intensity.
- Multiplied ‘cdf’ with 255 and assigned the new intensity values to same sub-array.
- All the sub-arrays are joined back into single image.







- The plots plotted above is for the first sub-array(0,0) , of an image.

FINAL OUTPUT :

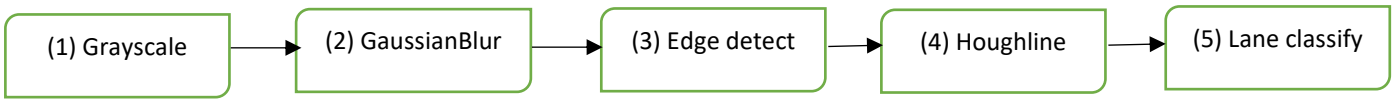
VIDEO_2 : https://youtu.be/CYxa9WVmDzI

COMPARISON between 1a and 1b :

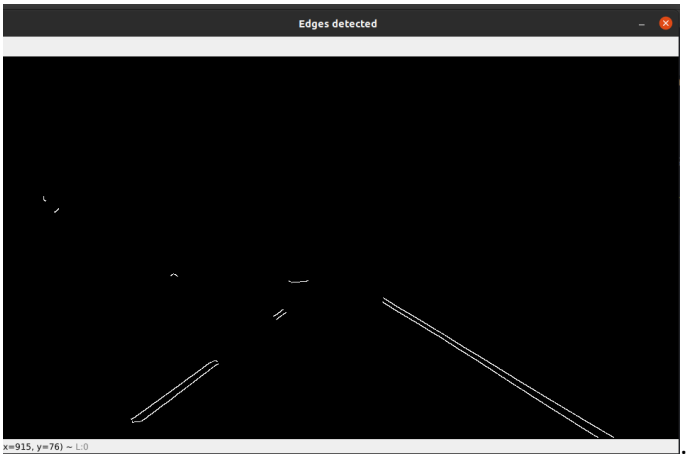
- The adaptive histogram, for the given frame contains more details of the road but appears to be washed out at different regions of the image.
- The results from normal histogram are preferrable, as adaptive histogram implemented for this problem is just a primitive version.

Solution 2 : Straight Lane Detection

Pipeline :



1) Read and converted frame into grayscale.
2) Performed blurring operation to filter out cars and unwanted lanes.
3) Parameter tuned CannyEdge to detect atleast the main lanes



4) Detected all the straight lines by using HoughlinesP, the parameters are finely tuned such that the first output line would always belong to 'solid lane' (worked for inverted image as well).
5) Using the first line output from Hough as solid, the dashed lanes were determined using the following approach:
    a. The slope direction of the solid lane was calculated(+ve or -ve).
    b. Using this, the equation of the line for the given slope was calculated and plotted green.
    c. Dashed lines were determined by selecting the lines with slopes ,which have opposite sign from that of the solid lane. (Dashed plotted in red)

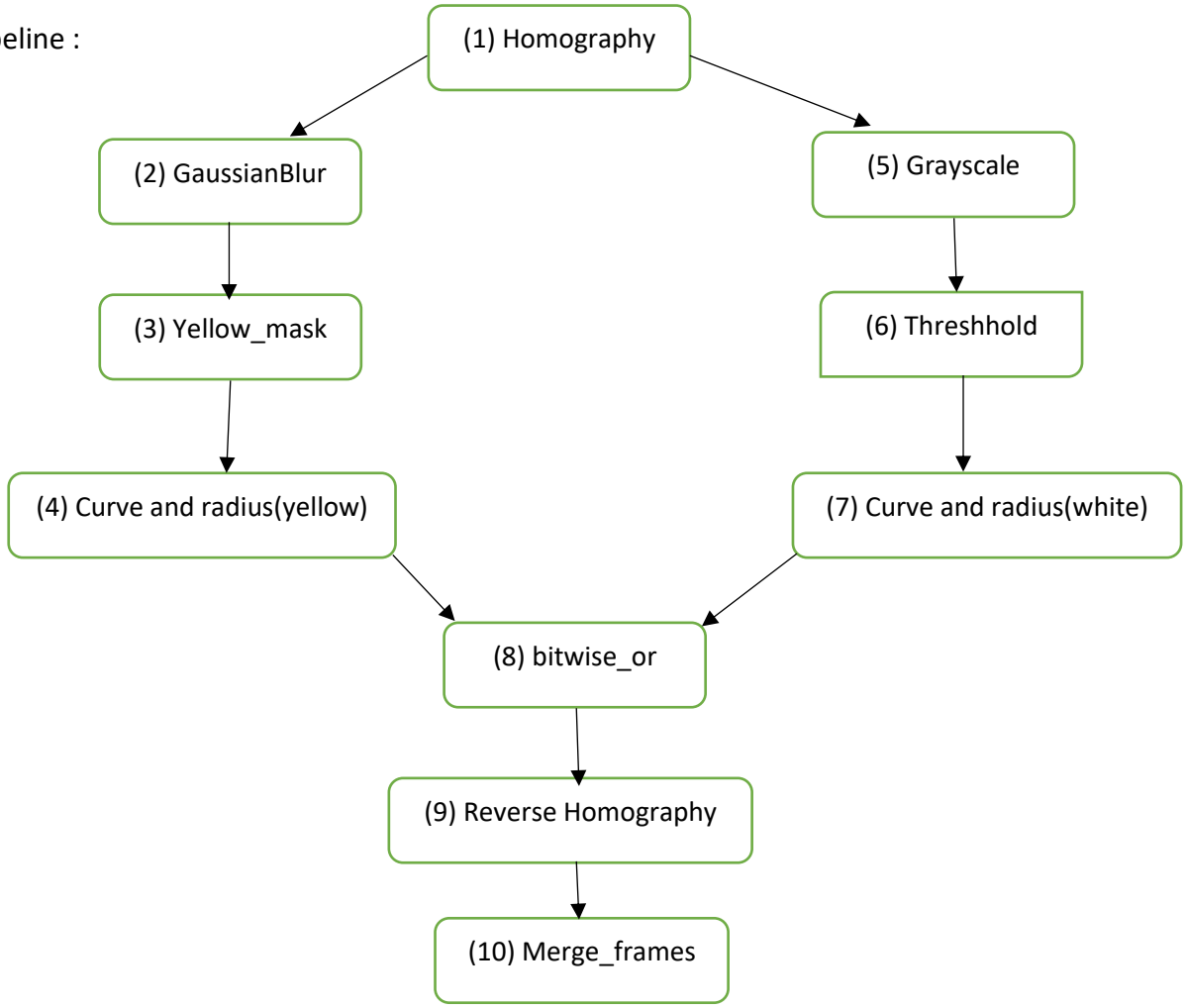
VIDEO_3 : https://youtu.be/ALf60N8Jc4w

CHALLENGES :

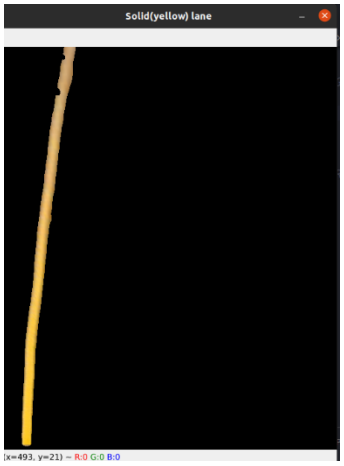
- Few extra lines(car edges..) were detected by hough, with same slope sign as that of the dashed lanes. This was a problem when storing dashed lanes as these lines were also classified as dashed lane. Issue solved by:
    - Defining a function ‘dashed_find’ , which would find the outliers if the slope didn’t belong to a specified range.
- Too many lines were detected when Standard Hough transform was used. Solved by:
    - By employing probabilistic hough transform.

Solution 3 : Predict Turn

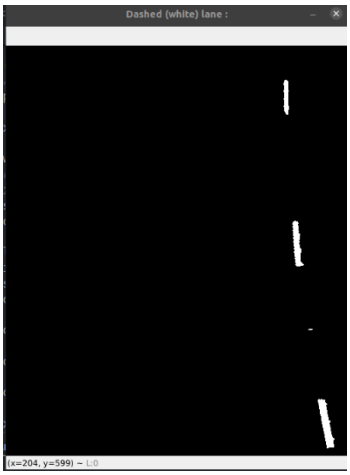
Pipeline :



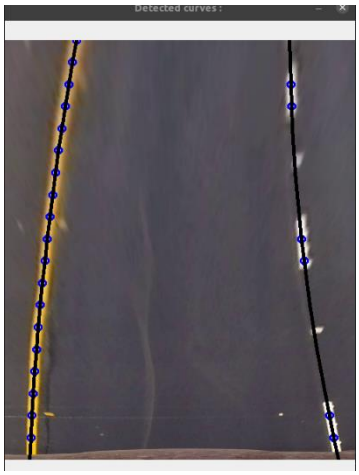
1) Inbuilt opencv function used to get top-view of required section.
2) GaussianBlur to remove noise from image.
3) Yellow lanes detected, by converting in ‘hsv' color format and applying mask.



4) The detected yellow pixels are passed into ‘np.polyift’ to get curve equation and radius of curvature is determined.
5) Top-view image converted into grayscale.
6) White lanes detected, applying threshold.



7) The detected white pixels are passed into ‘np.polyift’ to get curve equation and radius of curvature is determined.
8) Both the detected lanes are merged into a single image.



9) The top-view is converted back into original view.
10) The final image is generated by concatenating different images.

VIDEO_4 : https://youtu.be/swtyh6bpl-I

CHALLENGES :

- Wasn't able to detect white lanes properly using masking technique. Used threshold instead.

MISCELLANEOUS :

Homography :

- Homography, is a transformation that is occurring between two planes. In other words, it is a mapping between two planar projections of an image.
- It is represented by a 3x3 transformation matrix in a homogenous coordinates space.
- Atleast, 4 sets of points are required to perform homography. The 4 sets of points from source image is entered cyclically, followed by the destination images 4 sets of points.
- The common applications are image rectification, or camera motion(rotation and translation)between two images.
- The homography functionality was used for determining radius of curvature of the road. The orientation of the lane was changed such as to make it look like, we have placed the camera directly above the region of interest.

HoughLines :

- Based on houghTransform, houghlines work best on binary omage.
- Lines in image space are points in hough space and vice-versa.
- The line equation is stored in polar (pho, theta) format instead of slope intercept form as vertical lines .

- An important concept for the Hough transform is the mapping of single points. The idea is, that a point is mapped to all lines, that can pass through that point. This yields a sine-like line in the Hough space.
- To determine the areas where most Hough space lines intersect, an accumulator covering the Hough space is used. When an edge point is transformed, bins in the accumulator is incremented for all lines that could pass through that point. The resolution of the accumulator determines the precision with which lines can be detected.
- In opencv, HoughLines() outputs lines from images as vector of couples(r,theta). HoughLinesp() is a more efficient approach and outputs the endpoints of the line segment.

Pipeline Generalization :

Problem 1:

- The histogram implemented here is a basic one. If the input images had more noise, then the result would have been much worse.

Problem 2:

- The parameters are tuned for this specific condition. The pipeline would work on given inverted video though.

- If more white cars and aeroplanes were present. The pipeline would go haywire.
- The weather condition would also effect the detection process.

Problem 3:

- If the video was horizontally flipped, the pipeline wouldn't work properly as the current region of homography wouldn't cover the required region.
- If both the lanes are white, the pipeline wouldn't work.