PROJECT II

Perception for Autonomous Robots

Instructors:
DR. SAMER CHARIFA

Student:
RISHABH SINGH;
rsingh24@umd.edu

Semester:
SPRING 2022

Course code:
ENPM673

Contents

List of Figures

1 Introduction

This project includes topics like histogram equlisation (adaptive and non-adaptive), lane detection using a video from dashcam and segregating dashed and solid lines, and finally curve detection on a road's lane.

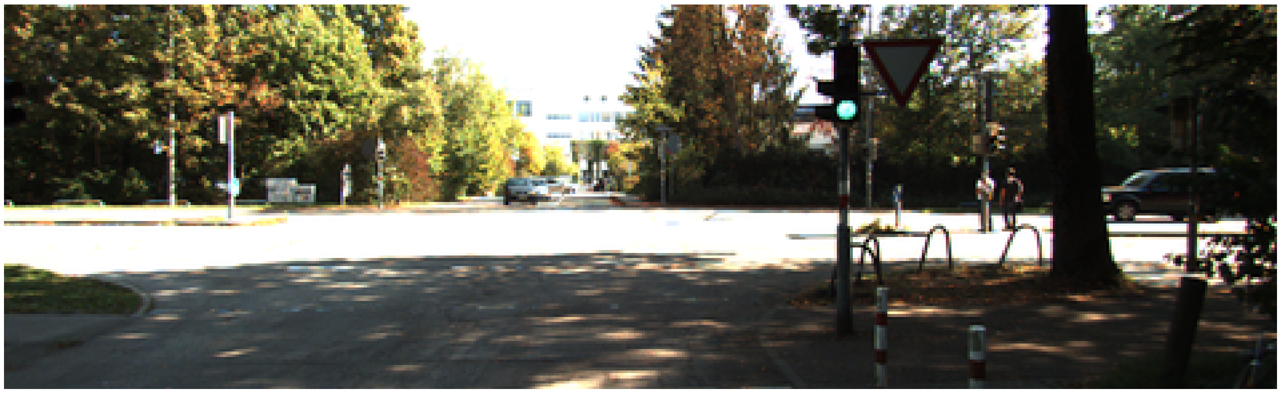
Figure 1: Sample input image to be equalised

2 Part 1

2.1 Histogram Equalisation

Problem Description: Here, we aim to improve the quality of the image sequence of which a sample is shown in 1. This was a video recording of a highway during night. Most of the Computer Vision pipelines for lane detection or other self-driving tasks require good lighting conditions and color information for detecting good features. A lot of pre-processing is required in such scenarios where lighting conditions are poor. Now, using the techniques taught in class our aim was to enhance the contrast and improve the visual appearance of the video sequence without using any inbuilt functions for the same. We had to use histogram equalization based methods, both histogram equalization and adaptive histogram equalization and compare the results.

Equalisation Algorithm:

A color histogram of an image represents the number of pixels in each type of color component. Histogram equalization cannot be applied separately to the Red, Green and Blue components of the image as it leads to dramatic changes in the image's color balance. However, if the image is first converted to another color space, like HSL/HSV color space, then the algorithm can be applied to the luminance or value channel without resulting in changes to the hue and saturation of the image.

- Check the maximum and minimum intensity of the image (in 8 bit, maximum = 256 and minimum = 1) and define number of bins. Store all the corresponding intensity values in their intensity bins.
- After getting the histogram we first find the PDF (Probability Distribution Function) for each intensity bin. Which is the number of pixels in each bin divided by the total number of pixels, given by the formula below:

$$PDF(i) = (n_i)/n \tag{1}$$

where, i = intensity value (bin), n_i = number of pixels in the i_{th} bin and n = total number of pixels

- After getting the PDF values, we need to calculate the CDF (Cumulative Distribution Function) for each bin, which is the summation of all the PDF below and equal to the intensity of the bin. This is an increasing function as give below:

$$CDF(i) = \sum_{j=0}^{i}[p_x(x = j] \tag{2}$$

- Using the CDF values we equalize the intensity values using the formula below which normalises the intensities according to the cumulative distribution. And the old values of intensity in each pixel location is replaced by equalised intensity values:

$$h(v) = round(\frac{CDF(v) - CDF_{min}}{n - CDF_{min}})(I_{max} - 1)$$

(3)

where cdf_{min} is the minimum non-zero value of the cumulative distribution function (in this case 1), n gives the image's number of pixels and I_{max} is the number of grey levels used (in most cases, like this one, 256).

- The output of the simple histogram equalisation is not normalised because of global normalisation, where some part of the image is underexposed and the other part is overexposed. As a result, the next good method is to perform the equalisation by distributing the image into small windows.

The CDF of the input image is plotted to check the distribution as shown in 2. It can be oberved that the distribution is not normalised and looks both under exposed and over exposed at certain places. To enhance the comparison I have performed equalisation in three different ways, first one is by equalising the pixels in all the channels at once, second one is by converting the image into LAB domain and then equalising the L values, and third one is by equalising each colour channel which is not recommended because the new colours formed are distorted.

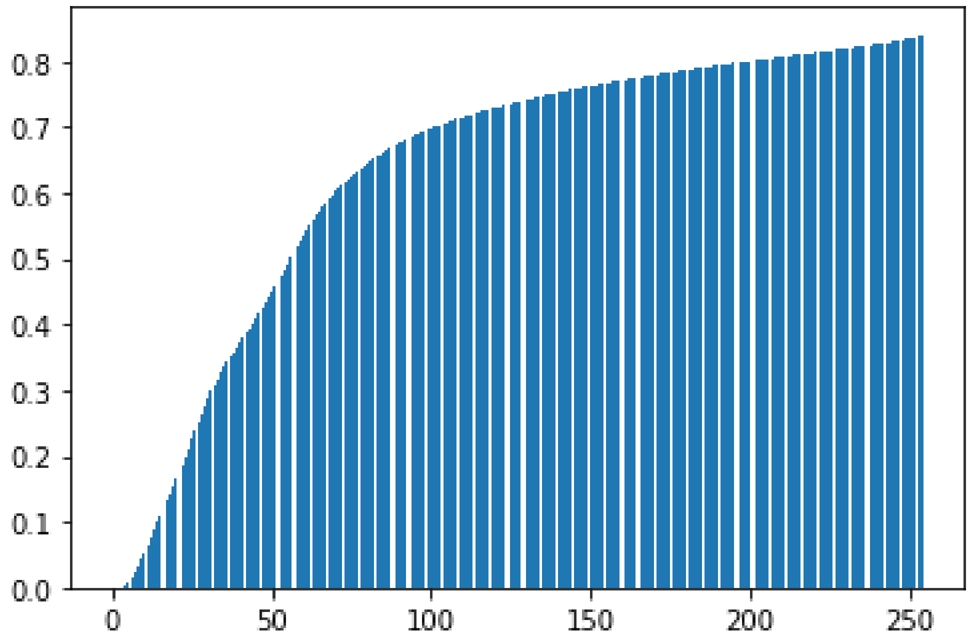

Figure 2: CDF histogram of input image

2.1.1 Equalisation in all RGB channels at once

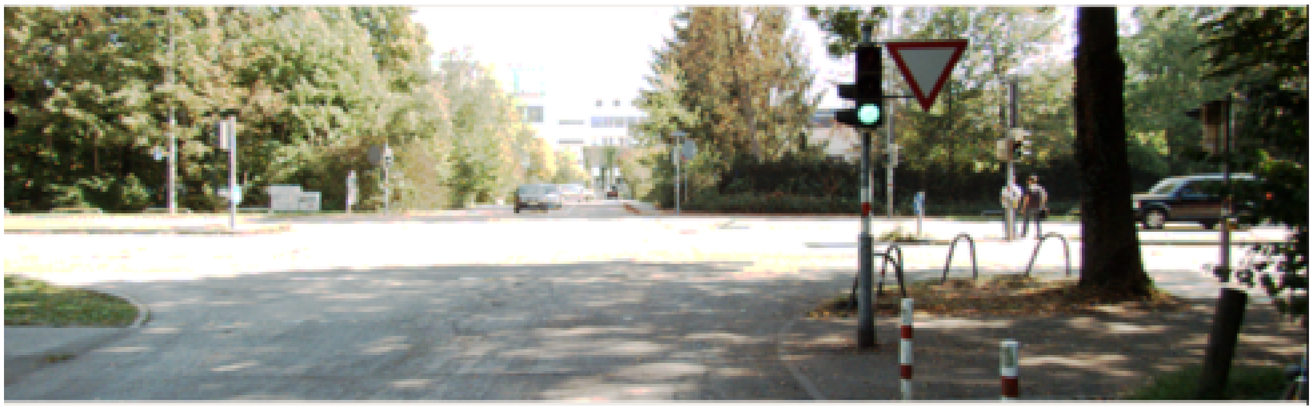

Figure 3: Equalised image using all channels at once (not recommended)

Though the output looks little normalised but still the image is overexposed at certain places. Also, since I have applied the equalisation at all channels at once this is not the recommended way to do it but strangely this gives the closest result to the inbuilt function. Even the CDF is as expected.

Link for Video

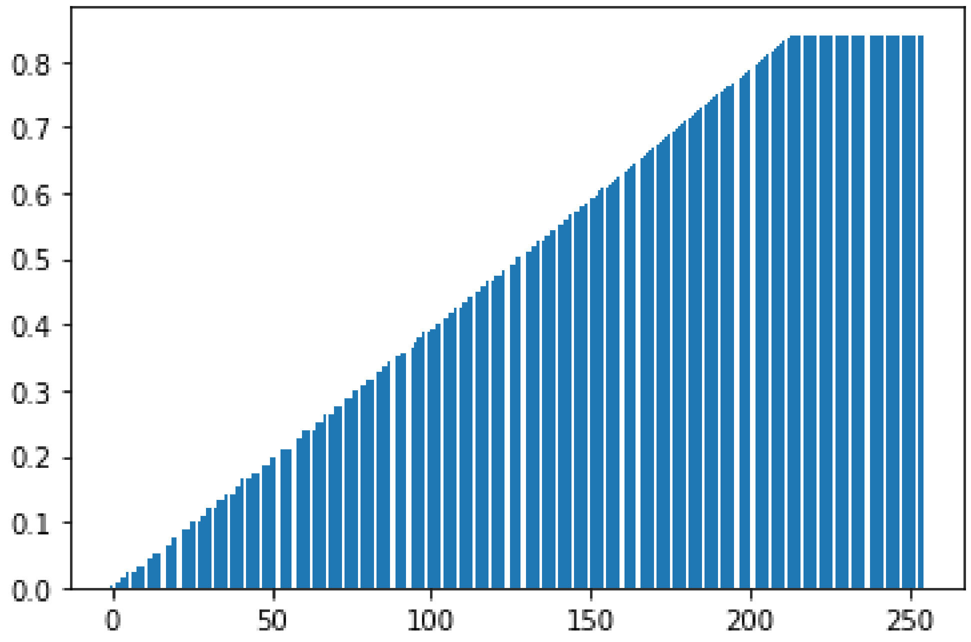

Figure 4: CDF of equalised image using all channels at once (not recommended)

2.1.2 Equalisation in L channel of the image

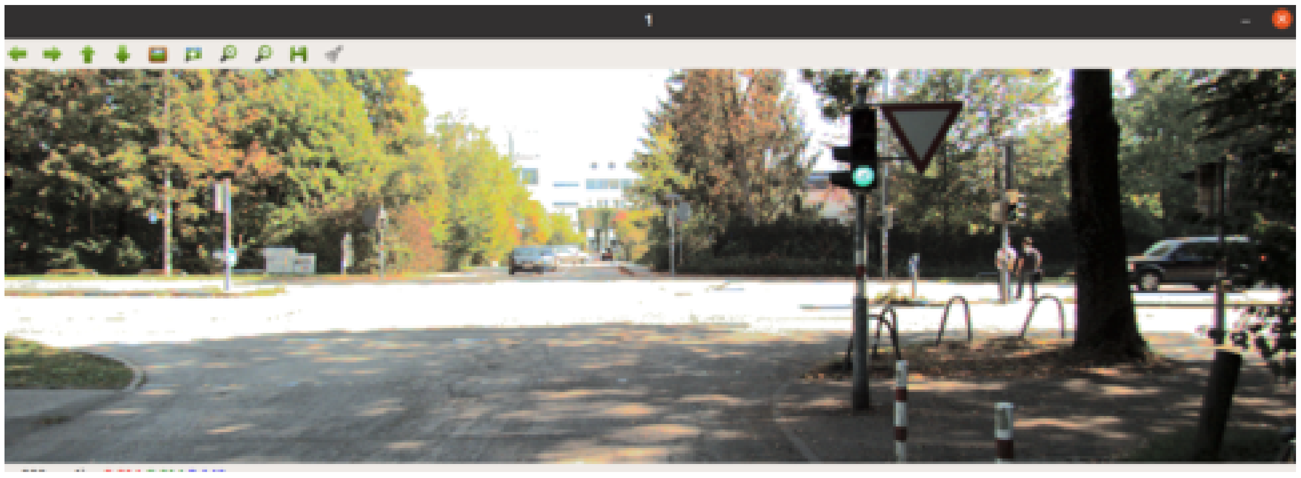

Figure 5: Image output with equalised L value channel (most recommended)

The only requirement for this part is to convert the image to LAB using cv2.COLOR˙BGR2LAB method and then applying the equalisation method to the L channel of the image. The CDF distribution in 6 also has a good distribution than the input image. Even the image in 5 looks better than the previous one in 3.

Link for Video

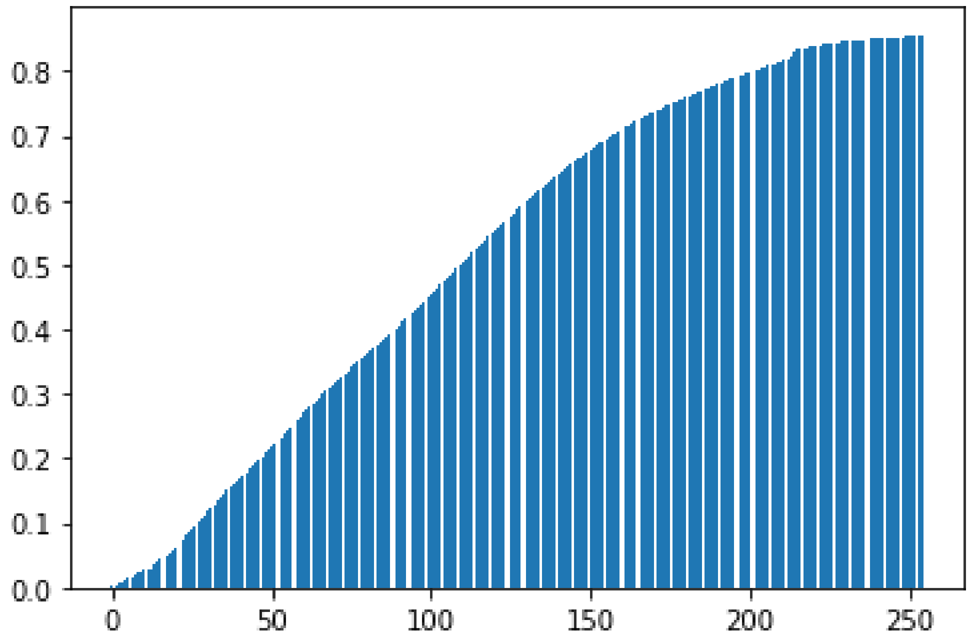

Figure 6: CDF of equalised image using L channel (most recommended)

2.1.3 Equalisation in each RGB channel of the image (not recommended at all)

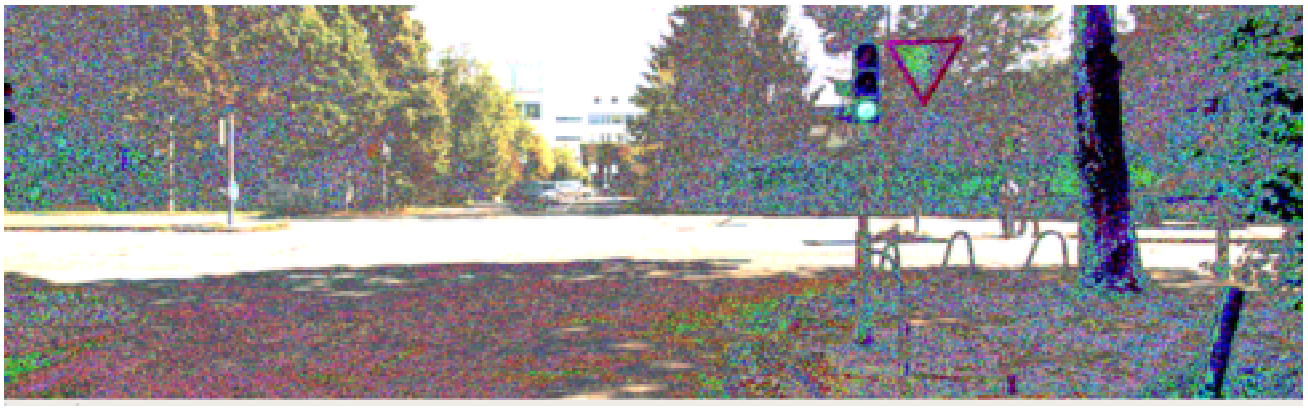

Figure 7: Image output with equalised RGB channels after splitting them (not recommended at all)

I am showing this just to show a comparison. This method is not recommended at all because of the colour distortion as shown by both 7 image and distribution in 8.

2.1.4 Adaptive Equalisation in all RGB channel of the image (not recommended)

The adaptive equalisation output is as shown by both 9 image and the distribution is shown in 10. The distribution looks better here but due to local windows created for local equalisation, we

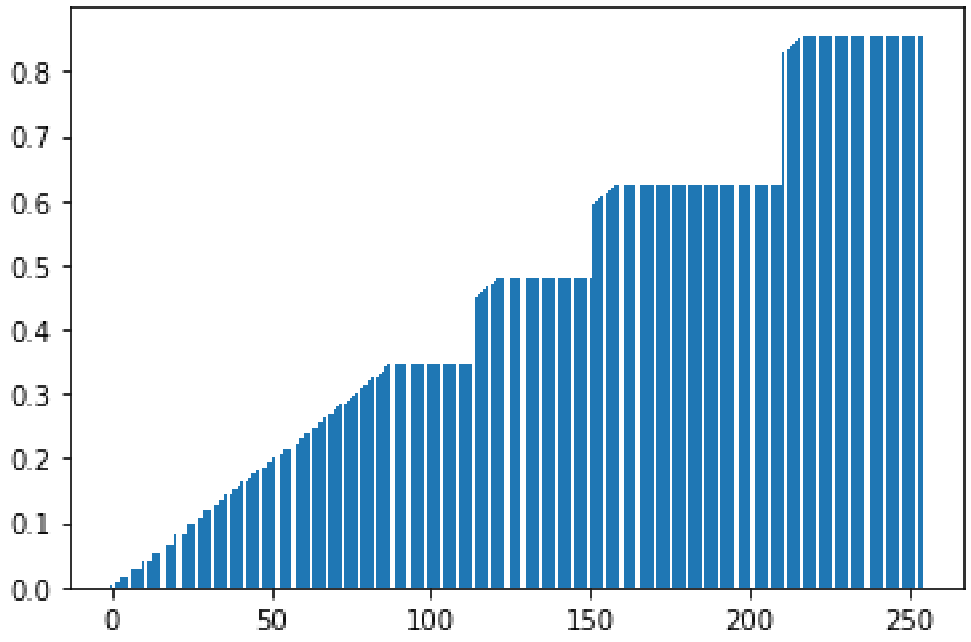

Figure 8: CDF of image with equalised RGB channels after splitting them (not recommended at all)

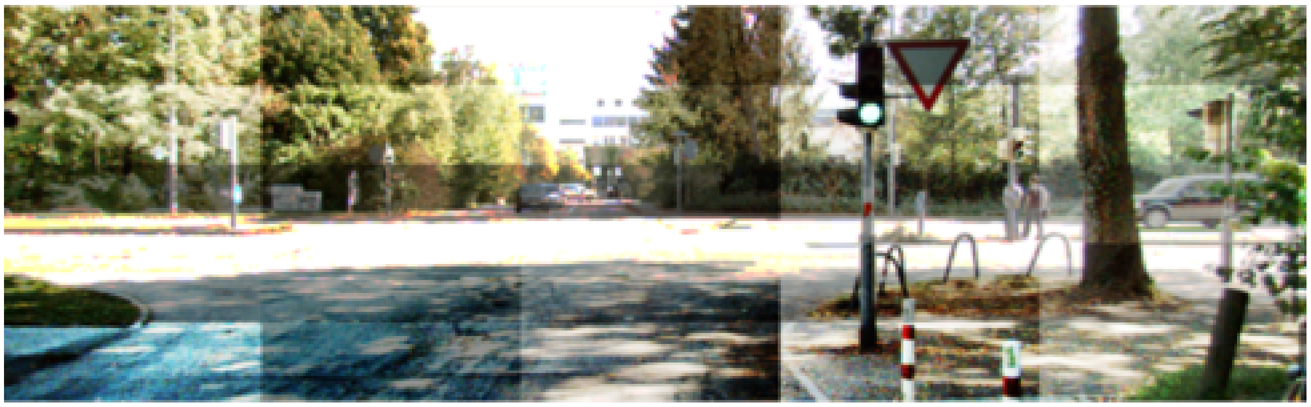

Figure 9: Image output with adaptive equalisation in all RGB channels at once (not recommended)

can see edges on the images. These edges can be removed using better methods such as contrast limiting.

Link for Video

2.1.5 Adaptive Equalisation in L channel of the image (most recommended)

The adaptive equalisation output is as shown by both 11 image and the distribution is shown in 12. The distribution looks better here but due to local windows created for local equalisation, we

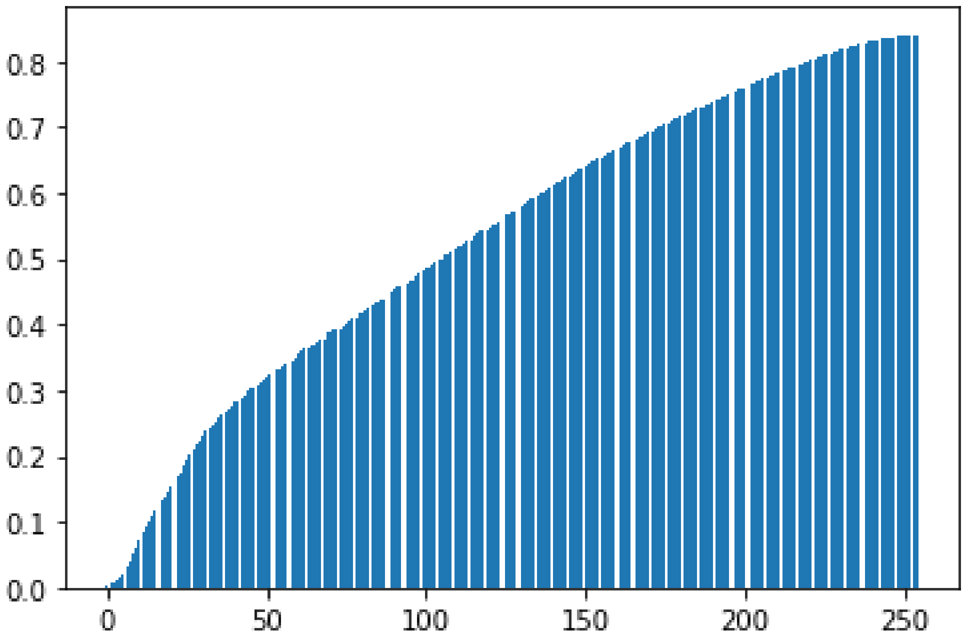

Figure 10: CDF of equalised image using L channel (not recommended at all)

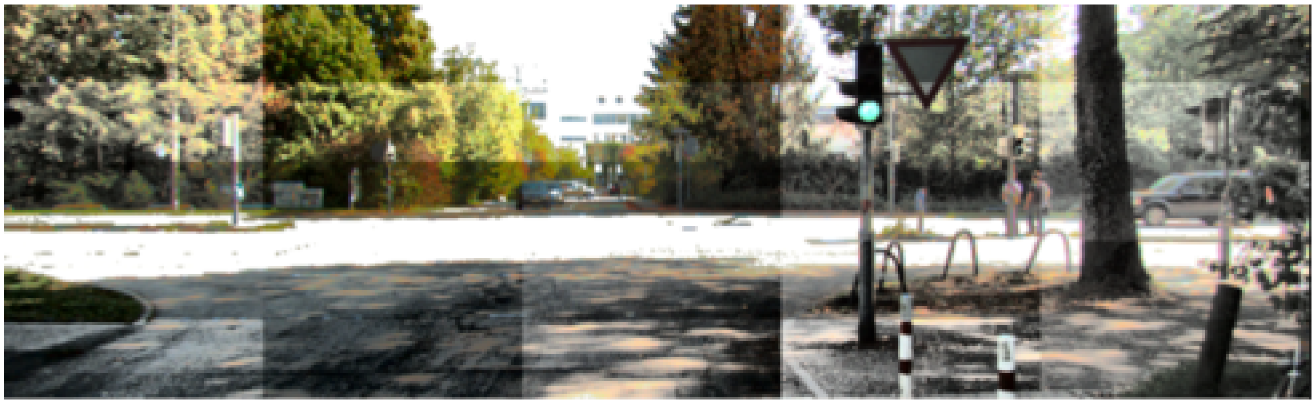

Figure 11: Image output with adaptive equalisation in L channel of the image (most recommended)

can see edges on the images. These edges can be removed using better methods such as contrast limiting.

Link for Video

Comparison between Adaptive Histogram Equalisation Output and Simple Histogram Output:

- Image Outputs:

  1) Though the contrast in the output images 9 and 11 using adaptive histogram equalisation has improved but the image is not smooth because of the kernels/window used. This can be

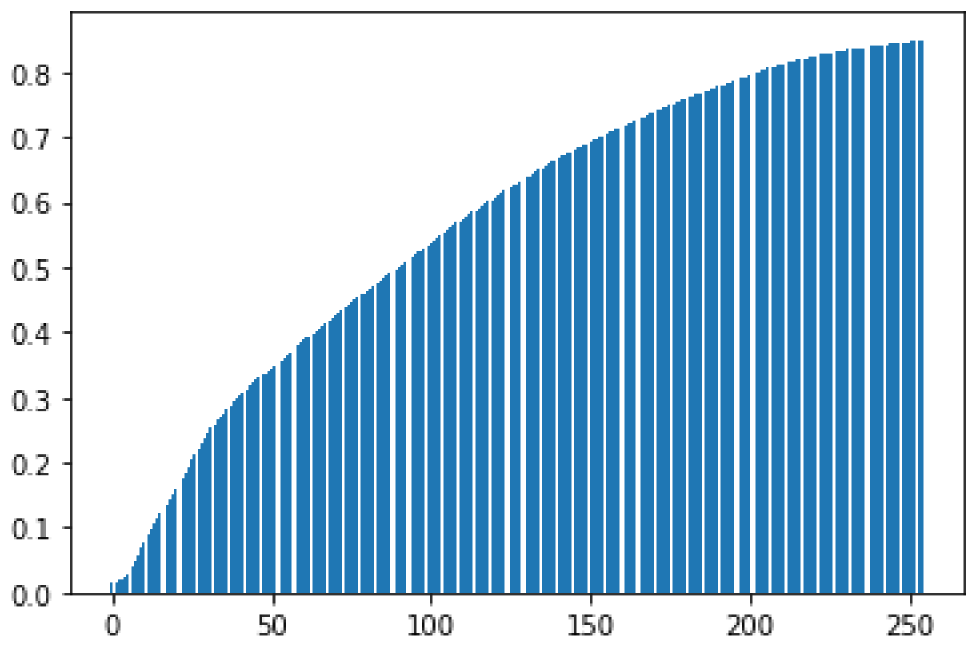

Figure 12: CDF of equalised image using L channel (most recommended of all)

overcome by applying the contrast limiting algorithm.

2) It can be seen, the middle part of [5] is overexposed which is making it worst than the input image, this has been taken care by [11].

3) It can be seen, the right part of [5] is still underexposed which has been beautifully taken care by [11].

4) It can be seen, the road in [5] looks better than [11].

- CDF Histogram plot Outputs:

  1) The CDF histogram of simple equalisation using the 3RGB channels at once looks very linear [4] and is closest to the inbuilt function.

  2) The histogram outputs from Lab image equalisation [6 is bit non-linear but makes more sense because the input image is both underexposed and overexposed at certain places. Also, this is very similar to the output of adaptive equalisation in [12] and [10]

  3) The histogram outputs from adaptive equalised image [12] and [10] is also non-linear and hence makes more sense because of the reason mentioned in point 2.

3 Part 2

3.1 Lane Detection

Problem Description: In this part, we were given an input video from a dashcam and using OpenCV functions we had to detect the lane and segregate dashed and solid line using different colour.

Lane detectionAlgorithm:

As mentioned in the problem, I came up with my own algorithm where I am defining my ROI and cropping the image directly using homography technique. The steps are described below in the flow chart [13].

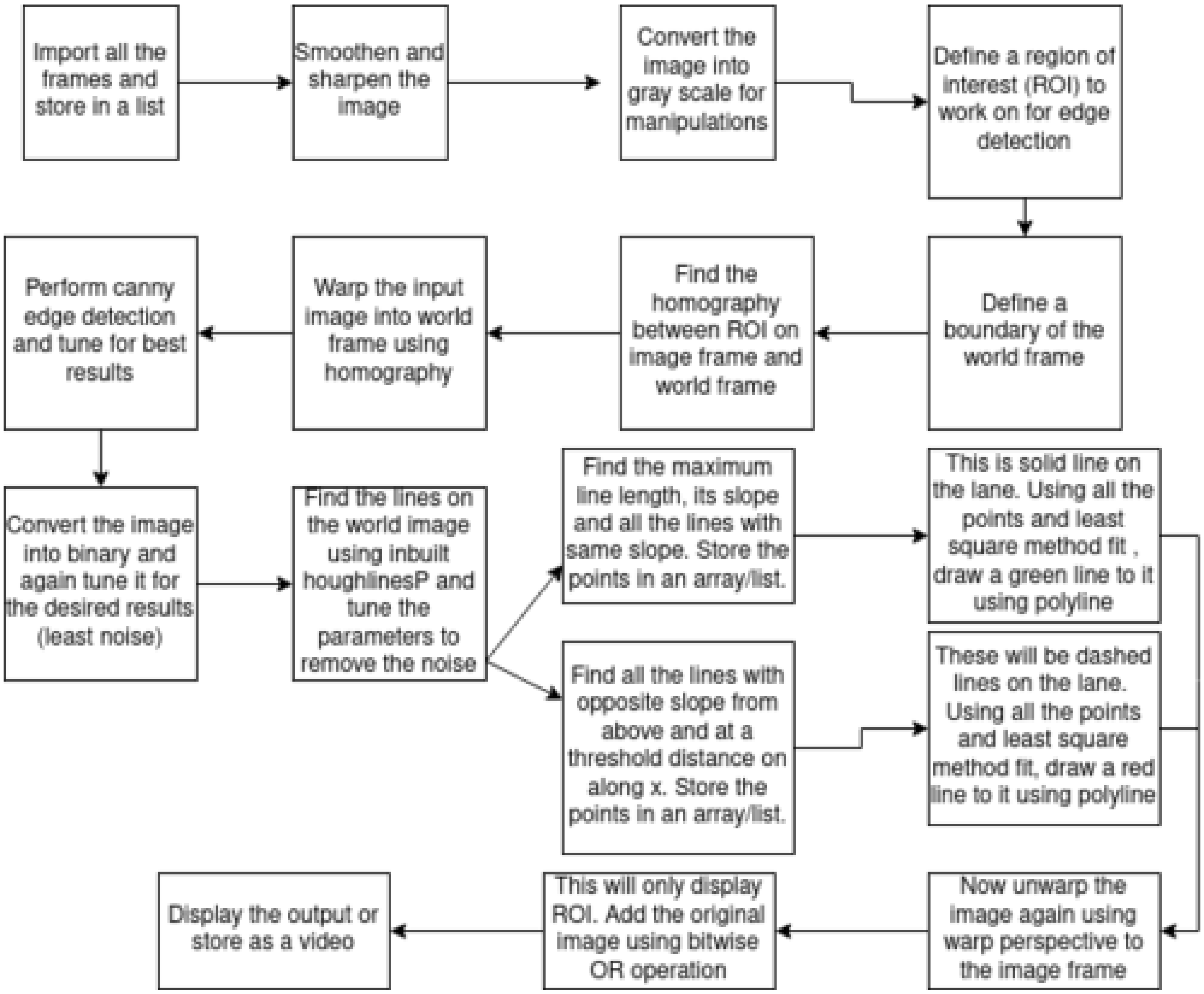
Import all the frames and store in a list
Smoothen and sharpen the image
Convert the image into gray scale for manipulations
Define a region of interest (ROI) to work on for edge detection
Define a boundary of the world frame
Find the homography between ROI on image frame and world frame
Warp the input image into world frame using homography
Perform canny edge detection and tune for best results
Convert the image into binary and again tune it for the desired results (least noise)
Find the lines on the world image using inbuilt houghlinesP and tune the parameters to remove the noise
Find the maximum line length, its slope and all the lines with same slope. Store the points in an array/list.
Find all the lines with opposite slope from above and at a threshold distance on along x. Store the points in an array/list.
This is solid line on the lane. Using all the points and least square method fit , draw a green line to it using polyline
These will be dashed lines on the lane. Using all the points and least square method fit, draw a red line to it using polyline
Now unwarp the image again using warp perspective to the image frame
This will only display ROI. Add the original image using bitwise OR operation
Display the output or store as a video

Figure 13: Flow diagram for pipeline followed for lane detection

Understanding of concept and problems faced:

1) Defining the correct ROI (Region of Interest) for homography required a lot number calibration in order to filter out the noise.

2) Finding the solid lanes was easy using the concept that it will always be longest line detected by houghlines, given the parameter tuning is correct. But it took me sometime to decide how segregate both the lines without hardcoding it. So, I came with the concept that the slope will always be opposite on both sides. So, first finding out the maximum line points using hough and then segre-

Figure 14: Sample input frame from video for lane detection

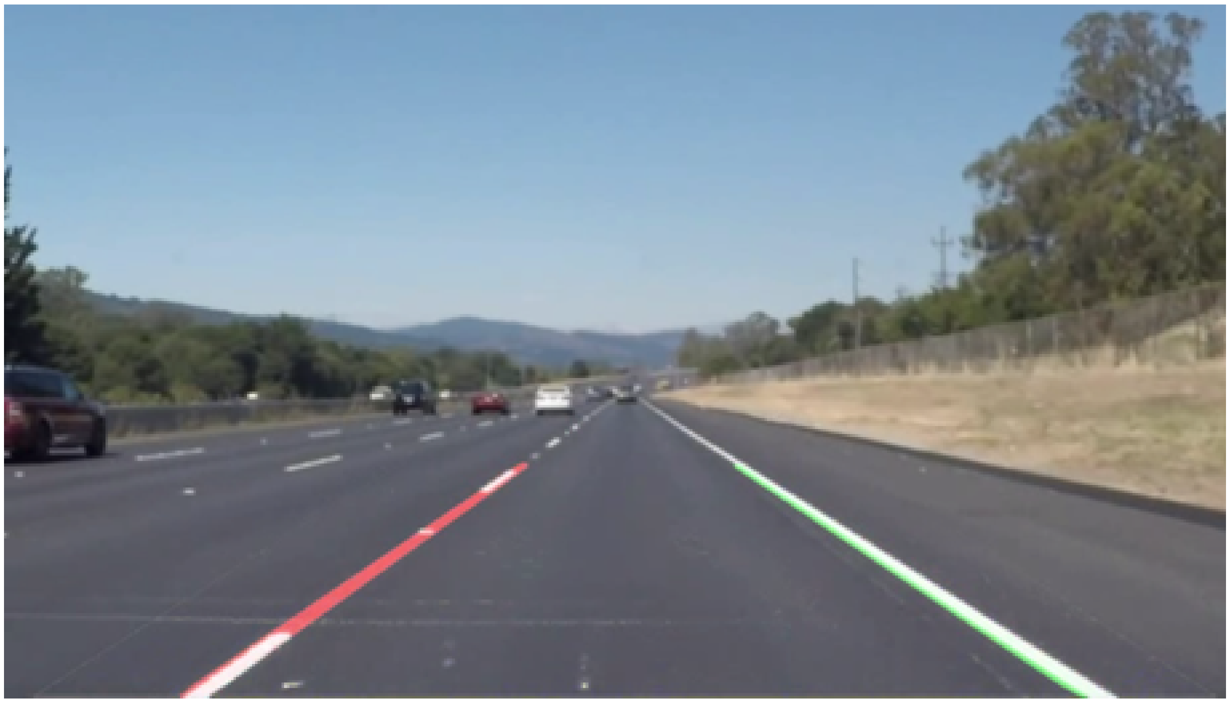

Figure 15: Output for lane detection Link for the video

gating the lines with same and opposite slop solved the case. Another thing I used to make the algorithm robust is in my homography frame both the lanes will have some distance, so I kept a thresh-

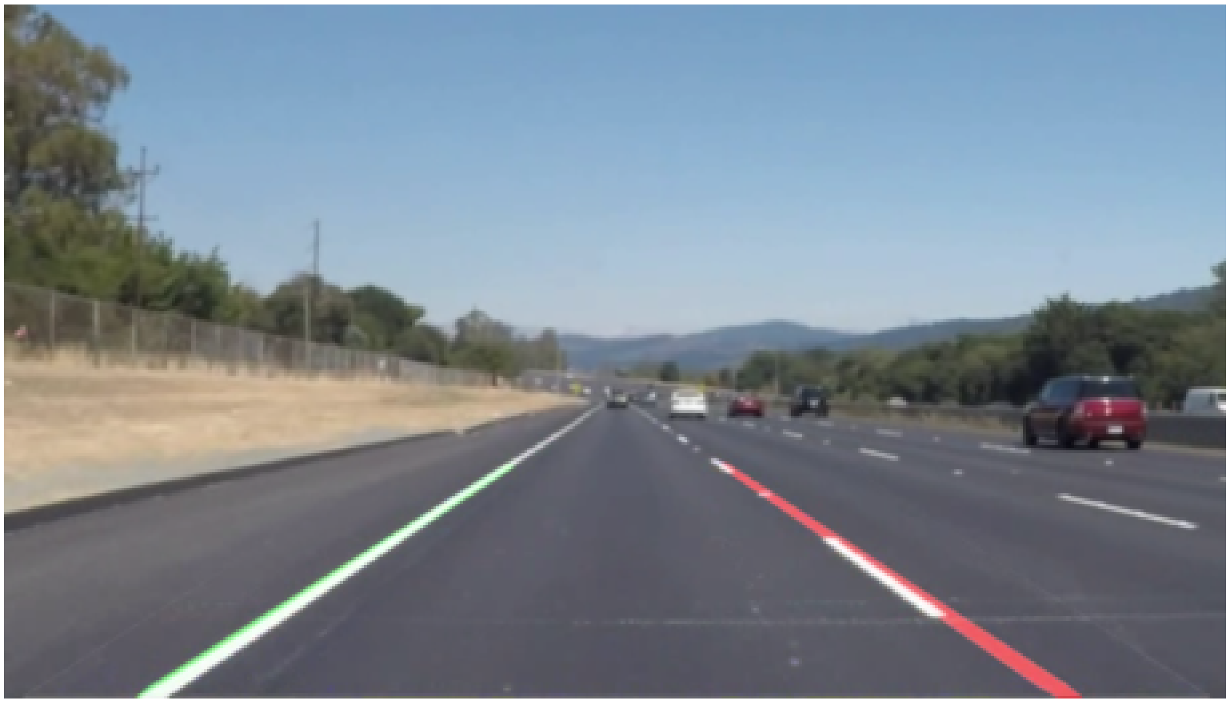

Figure 16: Output when the video is horizontally flipped Link for the video

old along the x direction for line and point detection, which will only flame if the image is vertically flipped but that doesn't make sense for a dash-cam until unless toppled :P.

3) While plotting the white line there was a lot of noise which was mostly because of less number of points detected either due to saturation, hence the plotted line was leaving the track of lane. To solve this issue, I used the points from previous frames which reduced the noise by a lot.

4) The ROI is hardcoded, the solution to this is to use the sliding window algorithm but since we were asked to develop our own algorithm I didn't use that method. But this area needs work because if the car leaves the lane for some reason, the algorithm may fail.

4 Part 3

4.1 Curve Detection

Problem Description: In this part, we aim to detect the curved lanes and predict the turn depending on the curvature: either left or right turn. The data-set provided has a yellow line and a while line. Our task was to design an algorithm to detect these lanes on the road, and predict the turn and compute the radius of curvature for the lane using the obtained polynomial equation. I separated the problem into two halves, first I performed detection of the yellow part using HSV range, found the side of yellow part on the frame and based on that kept a threshold to perform detection on the white dashed line (whether left part or right part). The technique I used for white part is just the intensity threshold to convert it into binary image and then performed the detection on that part.

The algorithms are as shown below in [17] and [18]

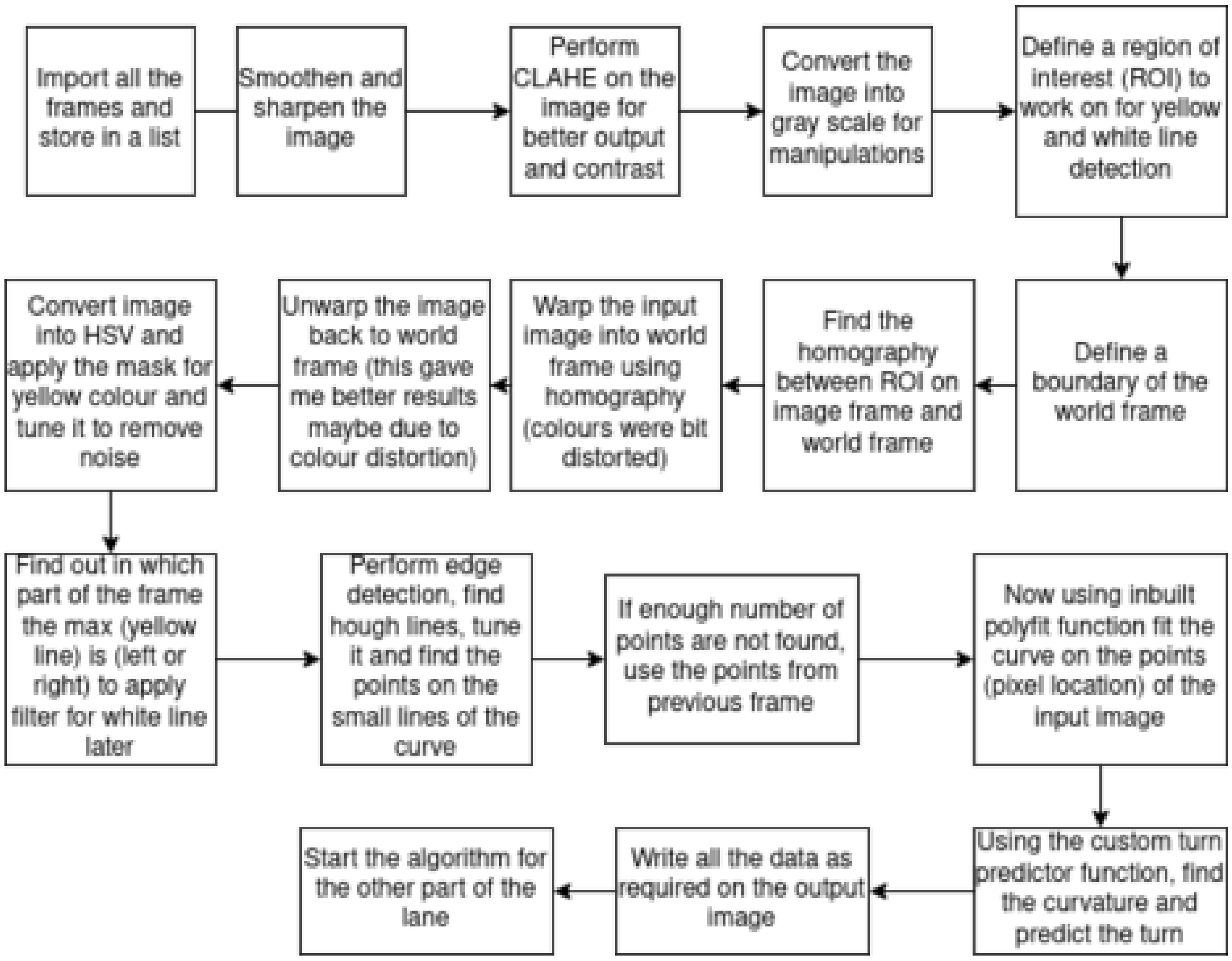
Import all the frames and store in a list
Smoothen and sharpen the image
Perform CLAHE on the image for better output and contrast
Convert the image into gray scale for manipulations
Define a region of interest (ROI) to work on for yellow and white line detection
Define a boundary of the world frame
Find the homography between ROI on image frame and world frame
Warp the input image into world frame using homography (colours were bit distorted)
Unwarp the image back to world frame (this gave me better results maybe due to colour distortion)
Convert image into HSV and apply the mask for yellow colour and tune it to remove noise
Find out in which part of the frame the max (yellow line) is (left or right) to apply filter for white line later
Perform edge detection, find hough lines, tune it and find the points on the small lines of the curve
If enough number of points are not found, use the points from previous frame
Now using inbuilt polyfit function fit the curve on the points (pixel location) of the input image
Using the custom turn predictor function, find the curvature and predict the turn
Write all the data as required on the output image
Start the algorithm for the other part of the lane

Figure 17: Algorithm to detect the yellow curve

Turn prediction algorithm: For turn prediction I wrote a simple algorithm to detect how the slope of the line is changing while going along the curve upwards, if the slope is continuously decreasing (in world frame) it will be a right turn otherwise it will be a left turn. This was continuously done for all the points on the curve and I kept a counter for right turn and left turn prediction at each point. The prediction with higher value is the average prediction for the curve. After that if the left curve and right curve agree with the prediction it is the output, otherwise the prediction of the curve with more number of points is considered.

textbfUnderstanding of concept and problems faced:

1) Defining the correct ROI (Region of Interest) for homography required a lot number calibration in order to filter out the noise.

2) Finding the solid lanes was easy using the concept that it will always be longest line detected by houghlines, given the parameter tuning is correct. But it took me sometime to decide how segregate both the lines without hardcoding it. So, I came with the concept that the slope will always be opposite on both sides. So, first finding out the maximum line points using hough and then segre-

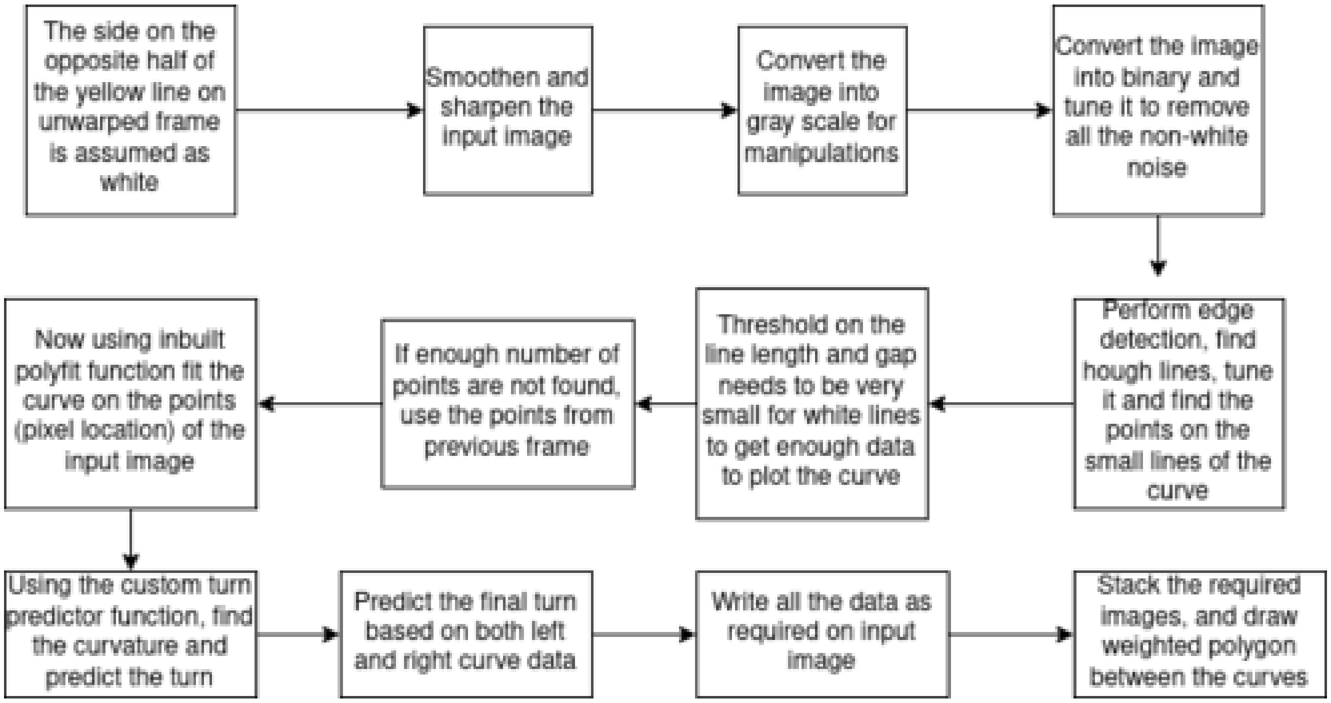
The side on the opposite half of the yellow line on unwarped frame is assumed as white
Smoothen and sharpen the input image
Convert the image into gray scale for manipulations
Convert the image into binary and tune it to remove all the non-white noise
Perform edge detection, find hough lines, tune it and find the points on the small lines of the curve
Threshold on the line length and gap needs to be very small for white lines to get enough data to plot the curve
If enough number of points are not found, use the points from previous frame
Now using inbuilt polyfit function fit the curve on the points (pixel location) of the input image
Using the custom turn predictor function, find the curvature and predict the turn
Predict the final turn based on both left and right curve data
Write all the data as required on input image
Stack the required images, and draw weighted polygon between the curves

Figure 18: Algorithm to detect the white curve

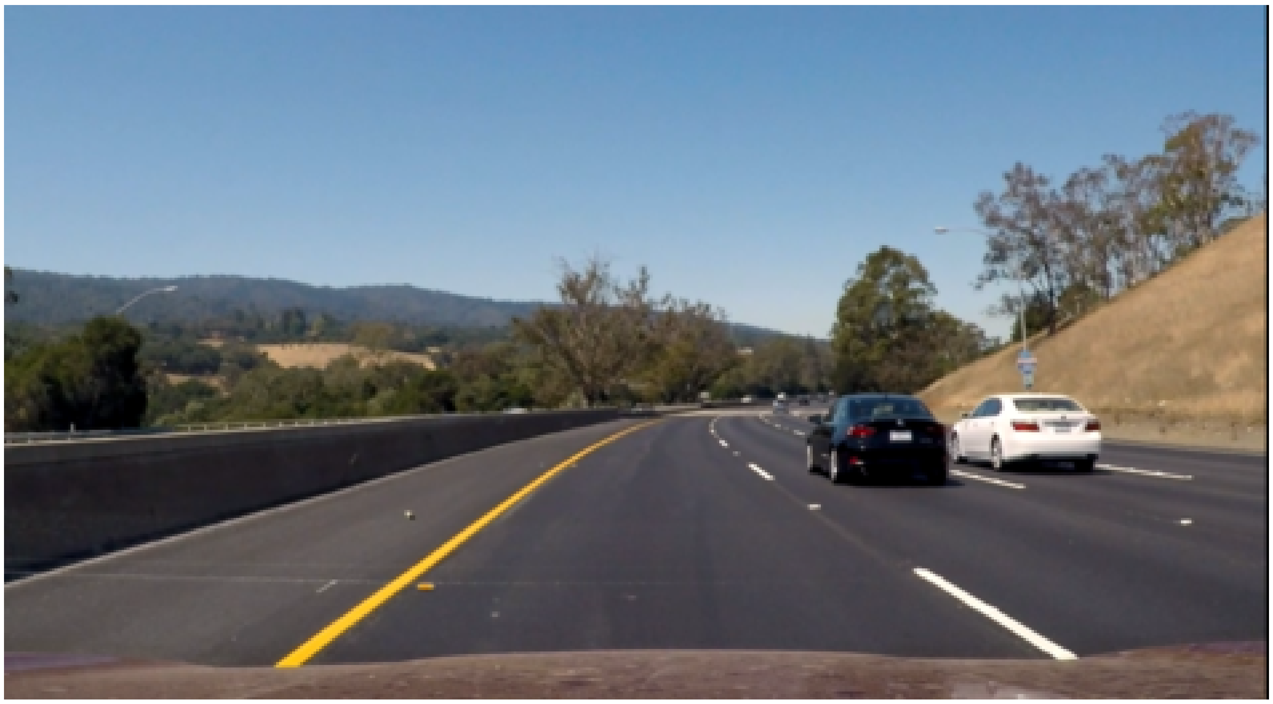

Figure 19: Sample input frame of the video

gating the lines with same and opposite slop solved the case. Another thing I used to make the algorithm robust is in my homography frame both the lanes will have some distance, so I kept a thresh-

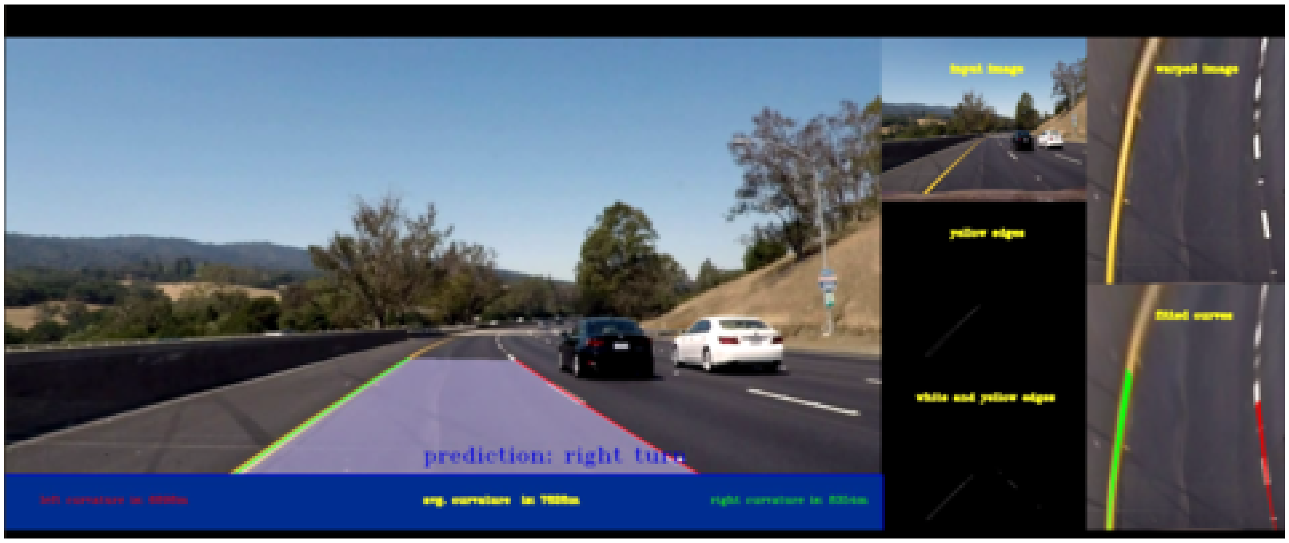

Figure 20: Desired output frame of the video Link for the video

old along the x direction for line and point detection, which will only flame if the image is vertically flipped but that doesn't make sense for a dash-cam until unless toppled :P.

3) While plotting the white line there was a lot of noise which was mostly because of less number of points detected either due to saturation, hence the plotted line was leaving the track of lane. To solve this issue, I used the points from previous frames which reduced the noise by a lot.

4) The ROI is hardcoded, the solution to this is to use the sliding window algorithm but since we were asked to develop our own algorithm I didn't use that method. But this area needs work because if the car leaves the lane for some reason, the algorithm may fail.

Understanding of used functions in all problems:

1) Defining the correct ROI (Region of Interest) for homography required a lot number calibration in order to filter out the noise.

2) Left (yellow) lane has an outlier [21] next to the line which I was flipping the curve whenever its in the frame. Because of this my left curve tends to give left turn prediction in these frames. I couldn't find a solution out of this and hence I had to give more weight-age to the right curve whenever the left curve tends to flip between frames.

3) While plotting the white line there was a lot of noise which was mostly because of less number of points detected either due to saturation, hence the plotted line was leaving the track of lane. To solve this issue, I used the points from previous frames which reduced the noise by a lot.

4) The homography points are hardcoded and because of the curve the algorithm is failing at some points when the curvature is small for the curve. The ROI is hardcoded, the solution to this is to use the sliding window algorithm but since we were asked to develop our own algorithm I didn't use that method. But this area needs work because if the car leaves the lane for some reason, the algorithm may fail.

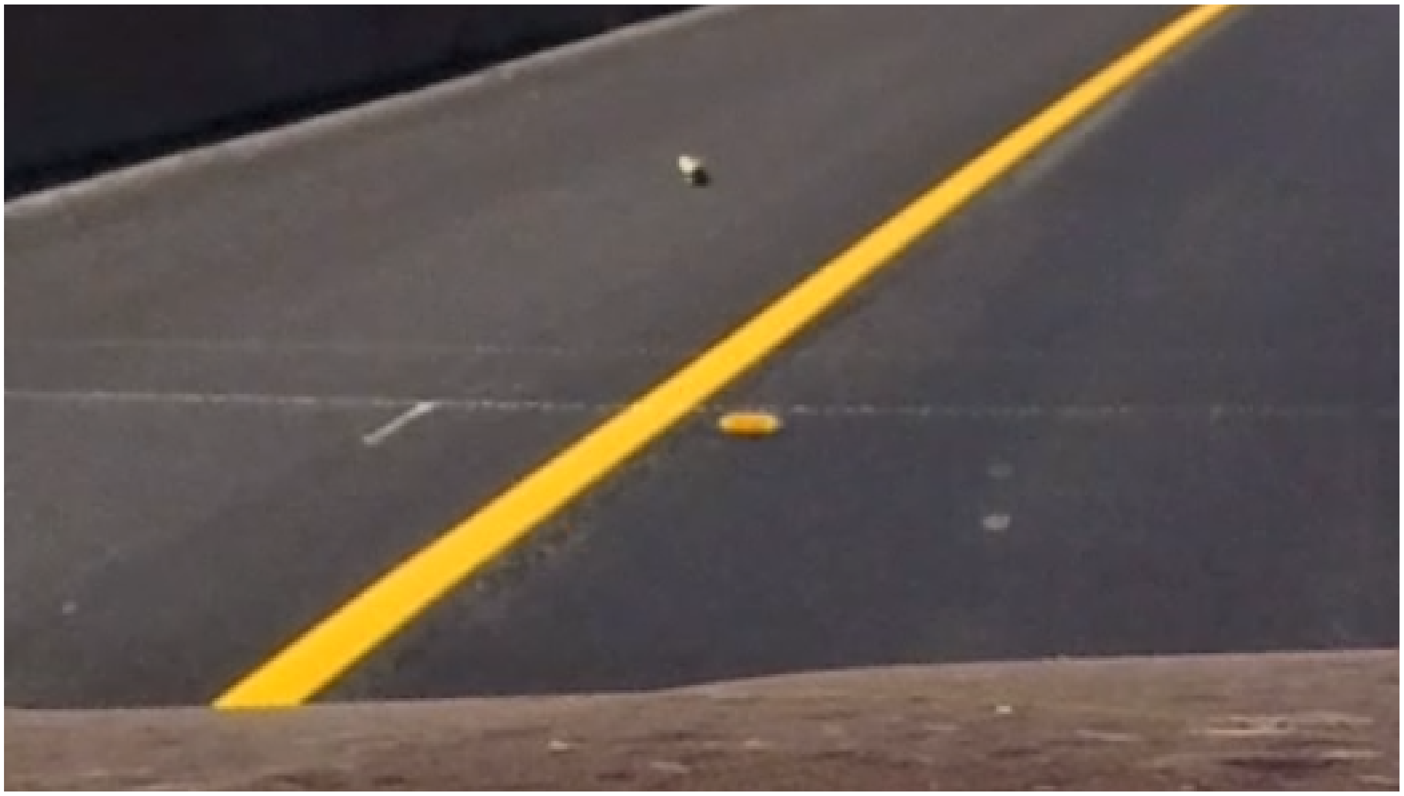

Figure 21: Outlier on the yellow line

Used techniques in all problems:

- **Understanding of homography and my use case:**

  Homography is a mathematical tool to map points between image frame to the world frame (planar homography). As shown in figure [22]. Basically two points can be mathematically mapped between two frames. It is represented by a 3x3 transformation matrix in a homogenous coordinates space. It is a very useful tool if we want to perform any kind of inspection or manipulation on the image. For e.g. it can be checked that whether a railway line shown in the image is parallel or not. In my case, I used the homography and warping to determine the region of interest. Now homography mathematically mapped the point and warping used the mapped in which each pixel location and was multiplied to the homography matrix to get the pixel location in the world frame. The only requirement for the map to build using the inbuilt function was to decide and feed the custom world frame corners. Now the intensities of the image are directly copied to the new locations on the world frame.

  Problem 2 and 3: I used the homography to define my region of interest and to see the lanes in the bird's eye view to perform edge detection, line detection and plotting. The homography matrix was fed into the warp perspective to create the bird's eye view and then again was unwarped using the inverse of homography matrix to see the plot on the image frame.

$$H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Figure 22: Homography tool

- Your understanding of how HoughLines work (whether you used them or not!)
- How likely you think your pipeline will generalize to other similar videos.