# Project 2 Report

## *Problem 1*

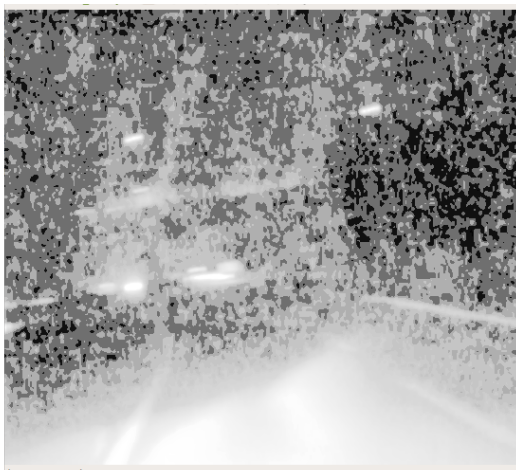
**Goal**: To improve the quality of the given video sequence using the Histogram Equalization Method.

**Approach**
I have tried different approaches with and without using inbuilt functions. The approach I shall cover is by implementing the histogram equalization method without any inbuilt function.

The steps for performing the histogram equalization are as follows:
1. The frames of the video are converted to grayscale and filtered by way of a Gaussian blur.
2. An empty array of zeros is created of size 256.
3. This array is filled with values of the pixel count representing the different pixel intensities of the image.
4. The CDF is computed using the pixel count of the different intensities obtained and dividing them by the total number of pixels in the image.
5. The CDF for each gray level is multiplied by 255.
6. The value obtained would correspond to the new pixel intensity. This way all the intensities are distributed across the image.

**Results**
The results obtained are as follows:

(a) Original Image



(b) Post Histogram Equalization

Figure 1: (a) Frame from the video provided (b) Histogram Equalization performed on the frame

When I try to implement the same code on another image. I get the following results.

Figure 2: (a) Original Image (b) Image after Histogram Equalization

We can clearly observe the quality of the image has improved thus suggesting that the code is working as expected. However for the video as part of this problem other techniques such as gamma correction or CLAHE can be implemented to improve the quality of the image.

Similarly using a colored image and performing the operations leads to the following result. But using the np.cumsum() function and using a masked array concept improved the quality but there is still a lot of noise

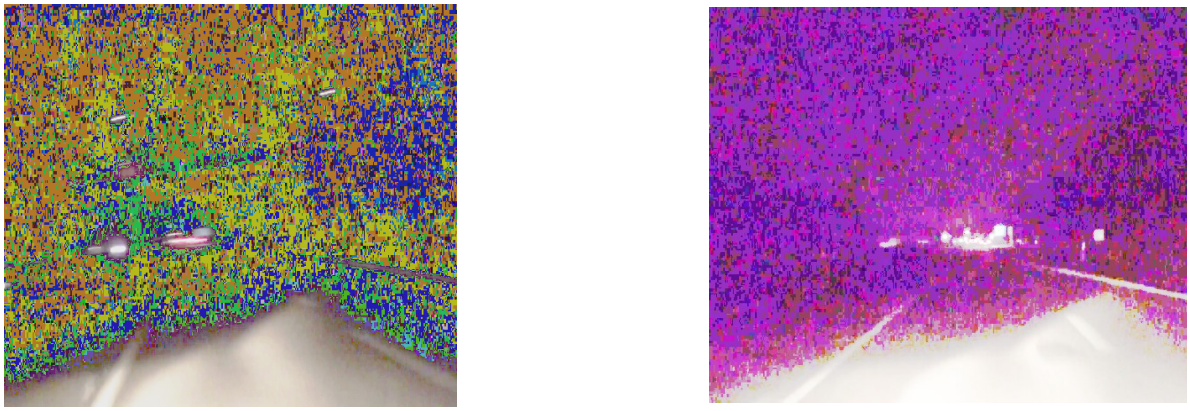


Figure 3: (a) Histogram equalization on a colored image (b) Usage of numpy function such as masked array and cumsum

The video for this problem can be viewed here

# *Problem 2*

## Dataset1

**Goals**:
(i) To stitch the images given to obtain a video.
(ii) To generate a pipeline for the lane detection algorithm.
(iii) To implement the algorithm based on the pipeline.
(iv) Documentation of problems and issues encountered when the algorithm runs.

**Initial Requirements**:
(i) The images given in the dataset 1 folder were stitched together in the order provided to generate a video of the vehicle driving in one lane (the right lane)
(ii) To perform homography 4 points needs to be selected, 2 points on each side of the lane to give a perspective of the image from the bird's eye view. This is done to track the movements of the vehicle as the video runs and predict the direction of the vehicle turn.

**Pipeline**

***Step1: Preparation of the input***

The steps to taken to prepare the input are as follows:
1. Distortion in the frames if any should be removed using the camera parameters provided which could remove some unwanted pixels and correct any lens distortion for the given camera matrix and distortion coefficients (k,d).
2. The process if noise removal is performed by way of a Gaussian blur.
3. The region of interest is extracted by manually selecting 4 points in total. 2 points on each side of the lane representing the top and bottom corner points. Homography is performed to project the 4 points on a new blank image. The warp perspective of the frame is found using the homogrpahy matrix to obtain the frame from a bird's eye view. This would show the 2 lines of the lane to be parallel and in the subsequent steps can help identify the lane candidate and to predict the turn as the video rolls on.

The images in Figure 1 show the frames from a bird's eye view when the lane is straight, curved to the left and curved to the right. In this video the lane is mostly curving to the left or is pointing stright. There are a very few instances when the lane curves to the right.

(a)

(b)

(c)

Figure 4: (a) Bird's eye view of a frame when the lane is pointing straight. (b) Warp perspective of a frame when the lane is curving to the left (c) perspective when the lane is pointing towards the right

***Step2: Detection of Lane Candidates***

This step of the pipeline focuses on identifying which pixels in a frame belong to the lane and store those coordinates and pixels for subsequent processing. The approach I have taken is as follows:

**1. Usage of a sobel filter**

Once the frames are obtained from a bird's eye view, a sobel filter is applied to obtain the pixels representing the edges. Before performing the sobel operation a threshold operation is performed to obtain a binary image. To threshold the image you can use any of the BGR channels, I obtained good results for all the three channels. The sobel filter is applied on the x and y direction on this image to obtain the edges. Figure 2 shows the output after the application of the sobel filter.

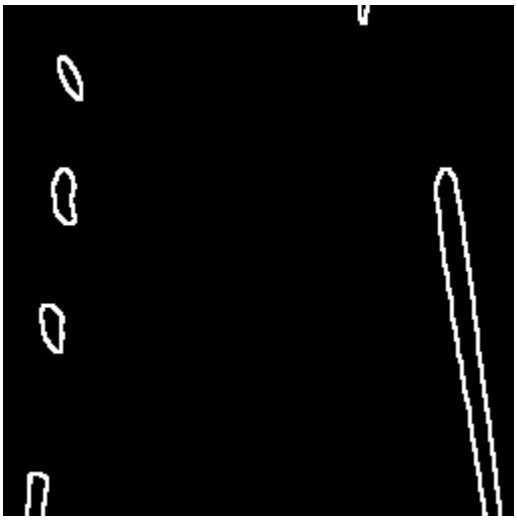


Figure 5: Implementation of sobel filter to extract the edges of the lane pixels

**2. Generating a histogram to obtain the white pixels on the left and right sides of the lane**

The histogram is computed by obtaining the number of white pixels in the y direction for a particular value of x in the image obtained after performing the sobel operation. I take the histogram of the whole image The peaks need to be obtained on the left and right half of the histogram. The image is split into 2 halves and a window is applied in each side which adjusts itself based on the mean values in the x direction and a window height specified in the y direction. The white pixels and their respective indices are obtained in each window. These pixels would represent the left and right x, y coordinates. The below image split into 10 sliding windows to obtain the white pixel coordinates.

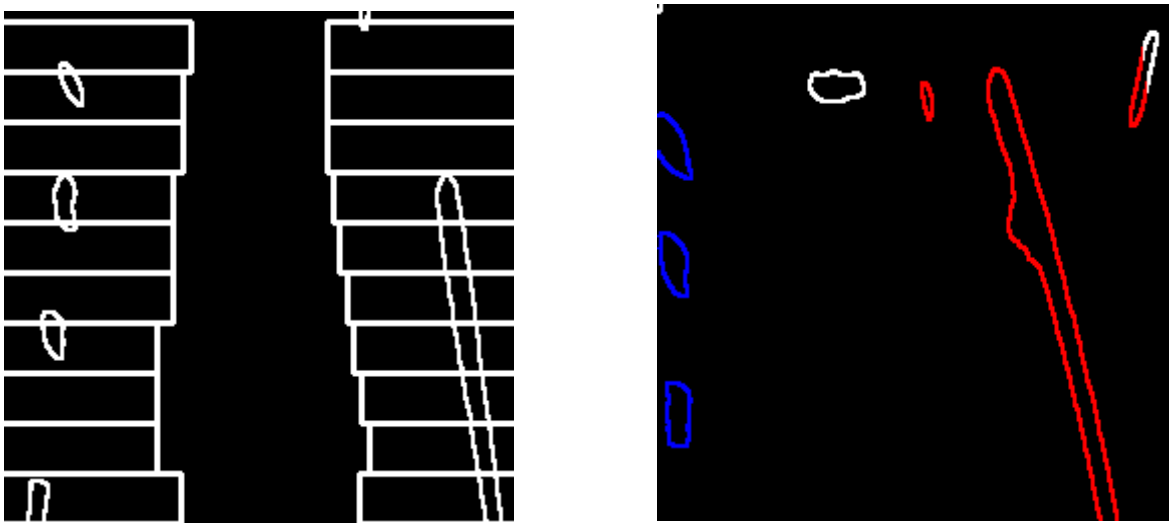


Figure 6: (a). Sliding window to obtain the x and y pixel coordinates in two peaks of the histogram (b) The histogram represented in red and blue for the left and right lanes.

***Step3: Refinement of the detected lanes using polynomial fitting***

Since the lanes are not always straight, the best way to connect all the points would be way of a polynomial equation. A second order polynomial has been fitted to the left and right points obtained to get the coefficients. The corresponding coefficients are multiplies with the the pixels along the line to obtain the new set of points which represent the lanes. The corresponding polynomial fit to the points are as displayed in figure 4 which shows the lane. cv2.fillPoly function has been used to draw a polygon with respect to the points obtained after the polynomial fitting.

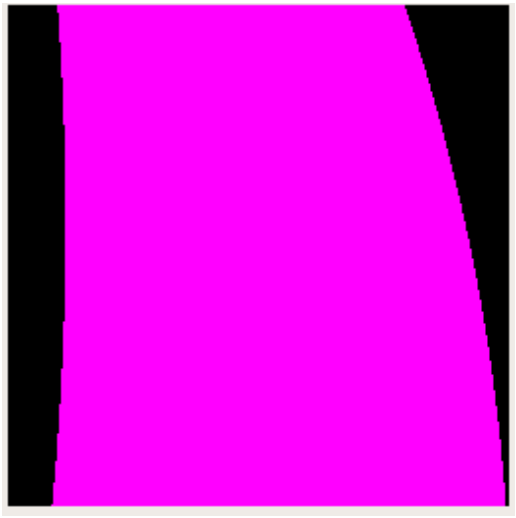


Figure 4: Polyfit

To overlay this image on the original frame, a warp perspective needs to be performed with respect to the original frame size and using the cv2.addWeighted the mesh is overlayed. The below figures show the result of a warp perspective and the subsequent overlaying of the mesh on the original image.

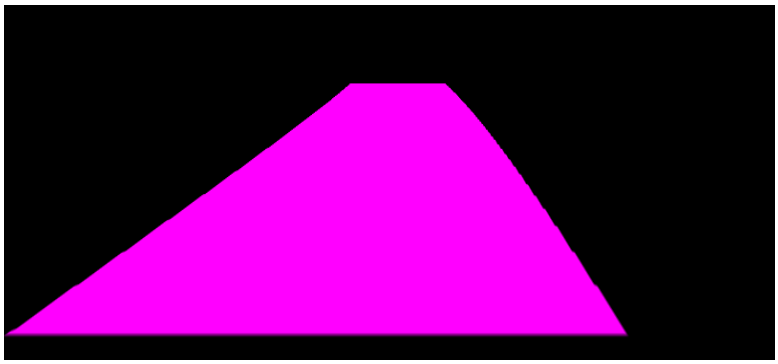


Figure 7: Warp perspective to project the polynomial fit on the original frame's plane

Figure 8: Overlaying of the mesh on the original image

**Step4: Prediction of turn**

Once we are able to find the area in the image which corresponds to the image where the vehicle in the video is moving. It is crucial to predict a turn based on the way the lane curves using the points obtained by fitting a polynomial. The method I am using to predict the turn is by taking the difference of the value which corresponds as the median of the warped image (my warped image has a height and width of 255, so the median value is 127) from the bird's eye view and the corresponding average of the bottom most points in the left and right lanes. If the difference between the 2 is below a certain threshold, the turn would be a left indicating a curve in the left direction, if it is above a particular threshold the turn would be right indicating a curve in the right direction and the values between these 2 thresholds indicate that the vehicle is moving straight.

The below images for the video correspond to a left, right and straight motions of the vehicle along the lane.


Figure 9: There is no curvature in the lane. Thus the vehicle is moving straight

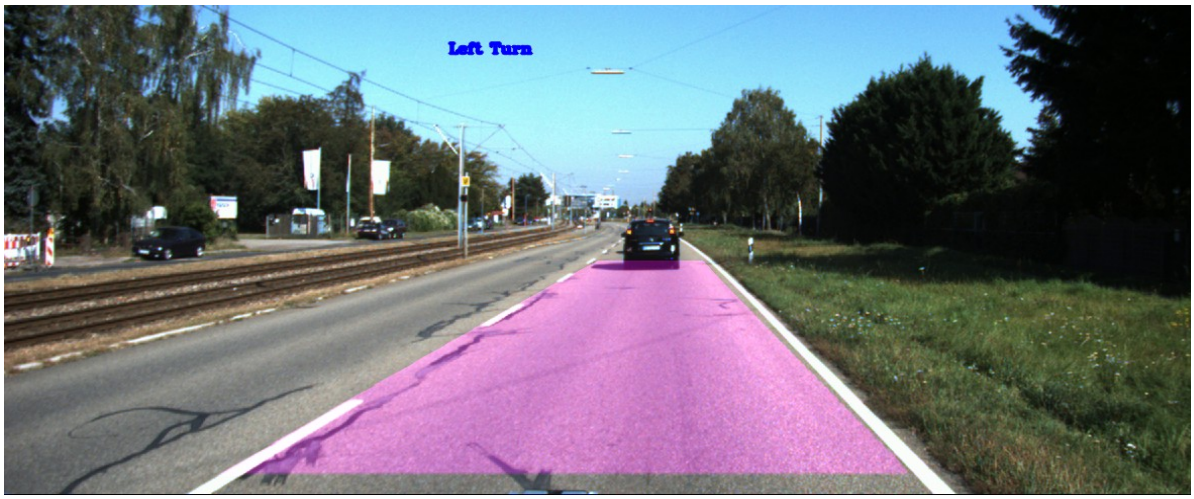



Figure 10: Road curving to the left

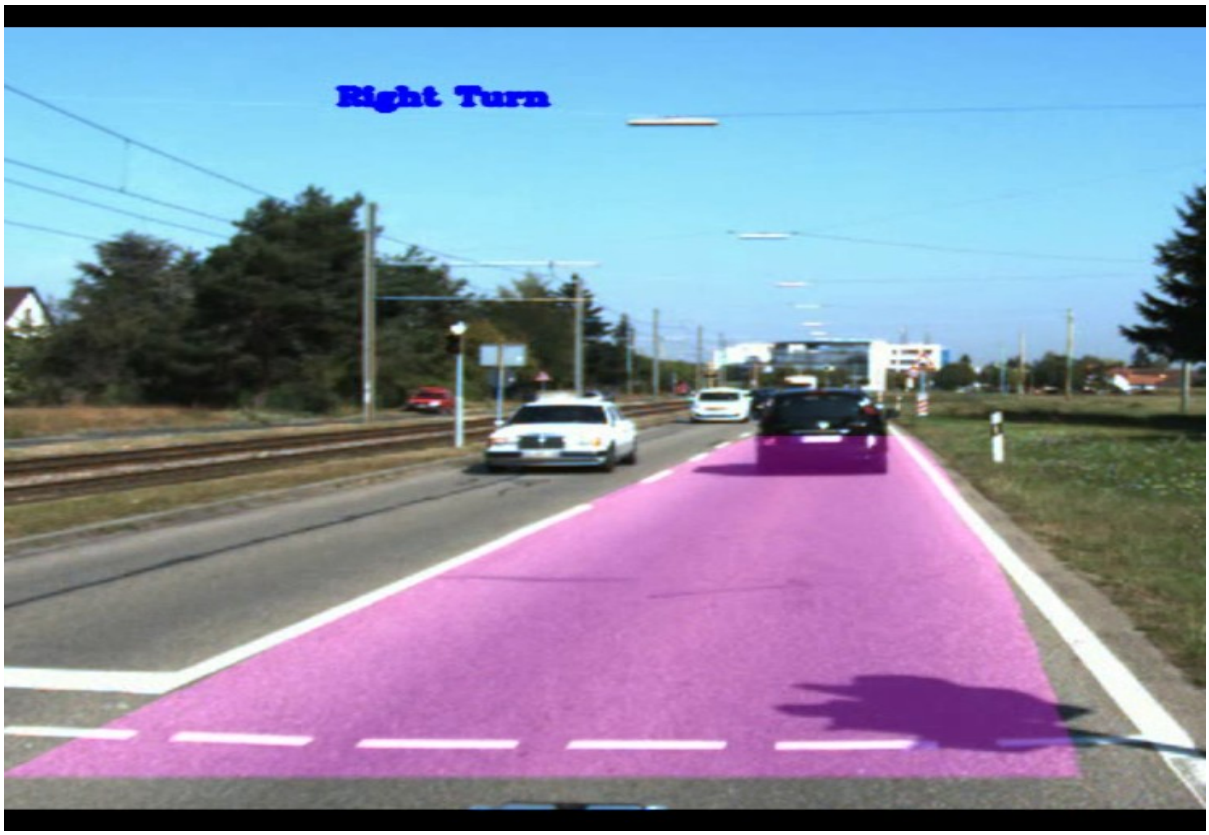



Figure 11: Road curving to the right

The video of the lane detection process can be viewed here

# Dateset2

The pipeline for this video is similar to the previous video but instead of implementing a sobel filter to obtain the pixels corresponding to the lane, color segmentation has been implemented to obtain the white yellow color candidates corresponding to the lane.

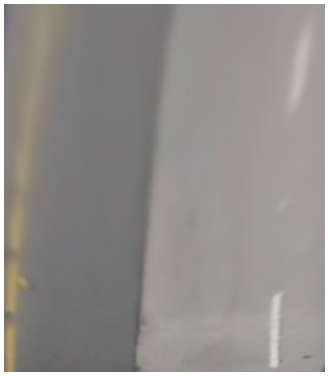
**HSL Color Segmentation**

I have considered using the HSL color segmentation method to obtain the pixel candidates because using the same method was not detecting white pixels which were of lower contrast during the beginning of the video and my program would terminate if no white pixels are found on the right side of the lane. Thus using a color segmentation method I was able to assign a range of values for the white and yellow regions to be detected. I was able to use these values to pot a histogram to obtain the peak values in the x direction and perform the other steps of the pipeline as for Dataset1.

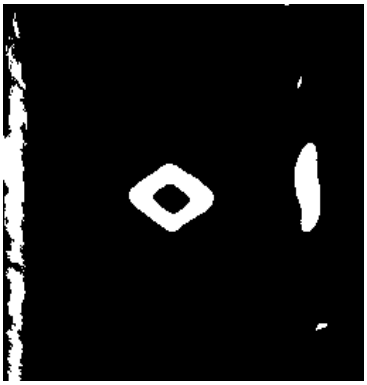
**Results**

(a)

(b)



(c)



(d)

Figure 12: (a) Bird's eye view of a frame for the lane with no curvature. (b) lane pointing slightly to the left (c) Lane curving towards the right (d) HSL segmentation to identify yellow and white pixels


Figure 13: Prediction of straight when there is no curve


Figure 14: Prediction of a right turn

(a)

(b)


(c)

Figure 15 (a) Bird's eye view of the lane slightly curving to the left (b) HLS color segmentation (c) Prediction of a left turn

The video of the lane detection process can be viewed here

# Problems Encountered for Problem 2

When the challenge_video,mp4 starts playing there is big gap on the right hand side between the 2 spots of white lanes as can been seen in the below image.


The next spot of the white lane was not being detected and the code was getting terminated since it could not find right vertices for the lanes. One fix for this is selecting the homography points of the bottom end of the lane a little bit further in the image. But even doing that was not causing the white pixels to be detected since its intensity was very low. Using the HSL color segmentation I was able to adjust the value of lightness for those pixels as well to be detected.

The other issue I faced was when the video shows the car going under the bridge.


From the images we can see that the points keep reducing as the vehicle goes under the bridge eventually it is just a small mesh. Once again setting homography points for the bottom of the mesh might help increasing the mesh size to go further down the bridge but there could still be problems in the prediction accuracy. In my case the program did not terminate because of the range of HLS values given and the program was able to detect the pixels outside the bridge and the program continued.

Another fix could be by using gamma correction which could increase the contrast of the pixels under the bridge. This could slightly improve the performance but there could be a case when pixels that are even slightly white or yellow could be recognized and the polynomial can be fitted based on those points. The below are the results when performing gamma correction. The mesh size has increased since more number of points are plotted in the histogram. Modifying the HLS ranges might also help in this detection. But in general there could be a case when in some portions the conditions might be different. So the system needs to be dynamic to adjust to these scenarios. Similarly when the vehicle comes out of the bridge the lighting conditions are a lot brighter in one side of the road and the polynomial mesh is unstable before finally adjusting back to the prescribed lane.


This was the reason I used color segmentation instead of sobel since it would allow me to obtain the respective pixels by entering a range of values.

The turn prediction has been user defined in both the cases based on the average of the bottom points of the polynomial. This would work in most cases. But using the radius of curvature to predict turns might be something to consider in the future.