ENPM-673 PERCEPTION FOR AUTONOMOUS ROBOTS

# **<u>Project-2</u>**
# **Lane Detection in Self Driving Cars**

Jeet Patel (117430479)

## Contents

## 1. Video Enhancement

The task here is to improve the quality of the video recording of the highway, which was captured during the night. Most of the lane detection applications requires good lighting and color information to detect lanes. Since it does not always satisfy these conditions, various computer vision techniques and pipeline are applied to enhance the quality of video. The input video is dark and contains large cluster of pixels as same color. A frame captured from the input video is shown in figure 1.

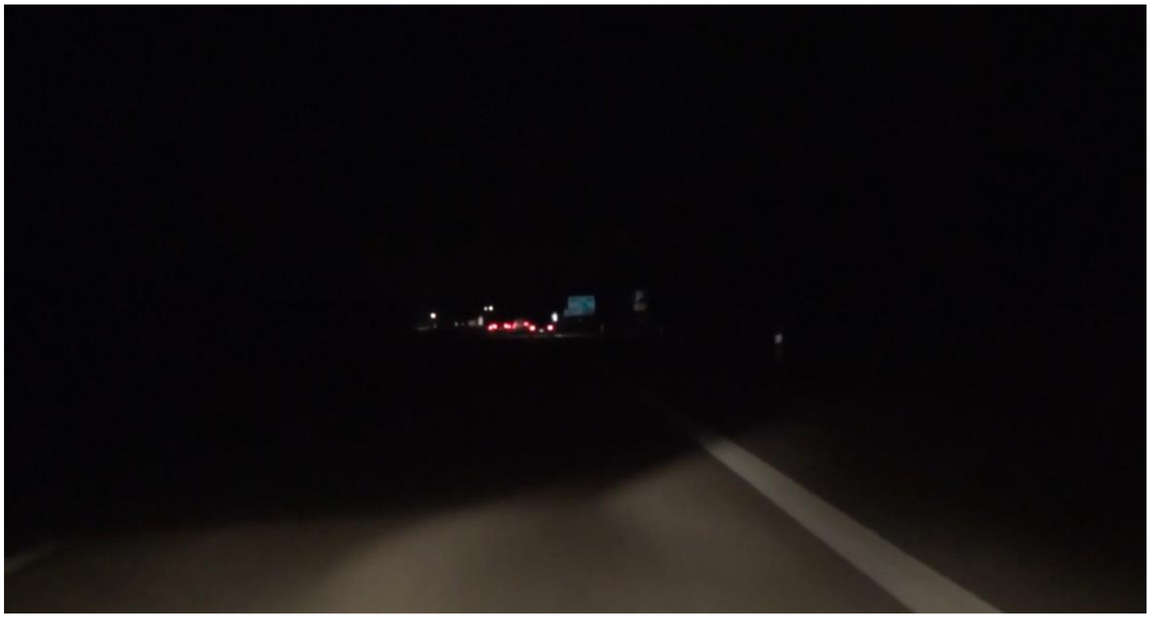

Figure 1: Original video

### 1.1 HISTOGRAM EQUILIZATION

The histogram is a graphical representation of an image's intensity distribution. It simply represents the number of pixels for each intensity value taken into account. It is technique used to improve contrast in images.

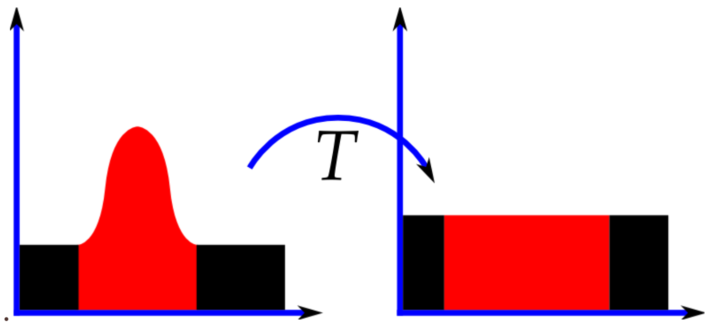

Figure 2: Histogram of image and after equalization.

Initially, I decided to improve the video quality by using histogram equalization. To correct the contrast, the non-uniform pixel values are uniformly stretched over the entire image. Initially, the video was converted into grayscale before being equalized, but the output was too noisy and contains grains in the image as shown in figure 3. I have also tried equalizing on an BGR image by separating 3 channels, but it distorts the color.

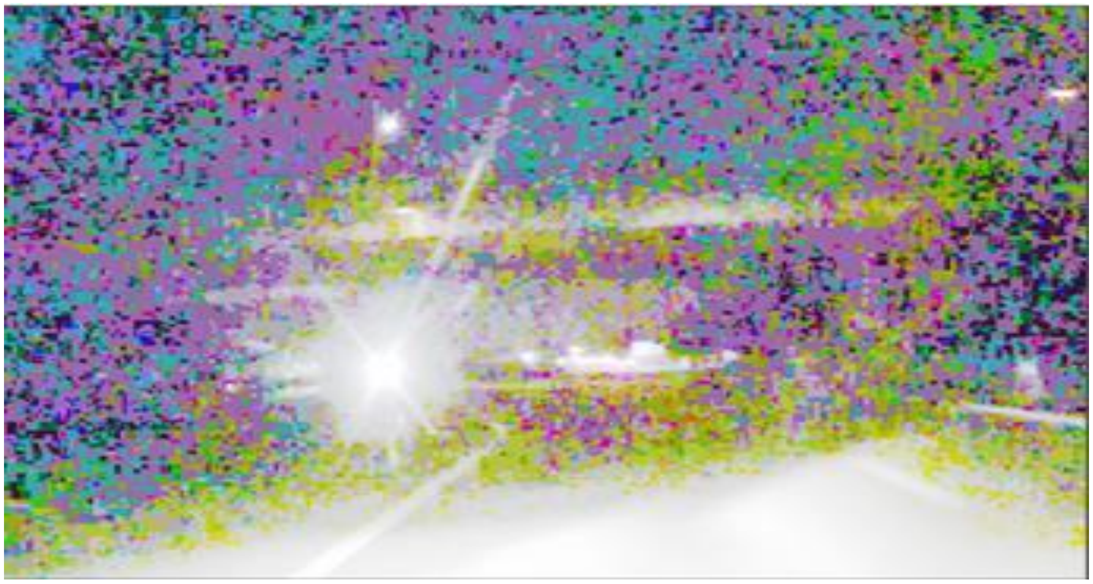
Figure 3: Output after Histogram Equalization

## 1.2 GAMMA CORRECTION METHOD

The output from histogram equalization is not sufficient to detect lanes, road signs, pedestrians in the autonomous vehicles. So, I decided to apply the gamma correction method to enhance the quality of video. Gamma correction is a nonlinear procedure used in video and still image systems to encode and decode luminance or tristimulus values. Now, every image capturing device cannot capture the luminance value perfectly. This value is between 0 to 1, where 0 means complete darkness (black), and 1 is brightest (white). The overall brightness of an image is regulated by gamma correction. Images that have not been corrected may appear bleached out or too dark.

The pipeline for solving this problem using gamma correction method is described as below:

1. Choosing the gamma value. Here, I have taken gamma = 2.
2. Then, pixel intensity values [0-255] are scaled in range of 0 to 1.
3. Now, gamma corrected image can be obtained by applying the equation:

   $O = I^{\frac{1}{G}}$ ; O = output Image, I = input image and G = gamma value
4. Return the output image's pixel values to their original range of 0 to 255.

5. Returning the original pixel can be done by implementing a lookup table. Firstly, build a lookup table that maps the input pixel values to the output gamma corrected values. This can be implemented using **cv2.LUT**. This is faster way to compute the output value as table determines the output in time complexity of O (1) time.

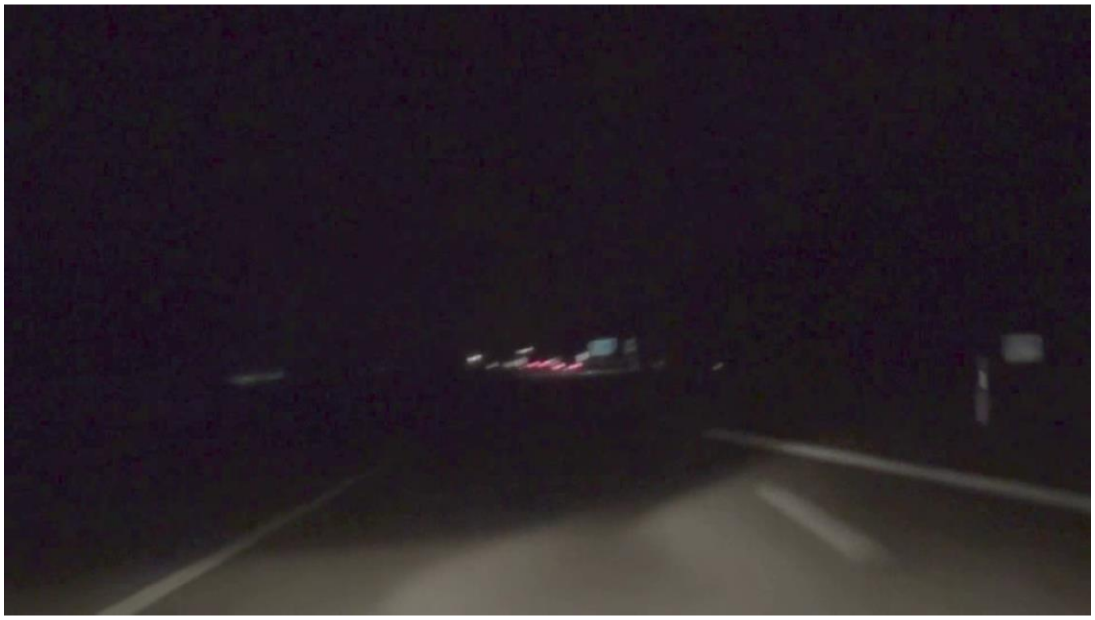

Figure 4: Enhanced Frame with gamma correction method

For gamma = 0.5, one can barely see the details on the road which is quite dark for the lane detection purposes. By increasing the gamma to 1.5, the image starts to lighten up and more details are observed. Setting the gamma to 2, all details are observed but after further increase in gamma value, the image becomes washed out.

## 2 Lane detection

### 2.1 OVERVIEW OF PIPELINE

1) Un-distort the image based on camera parameters
2) Removing the noise
3) Use color transform, gradients to create a threshold binary image
4) To get a bird's eye view, using a perspective transform.
5) Generate a histogram of the number of white pixels in each column of binary image
6) Find the histogram peaks that reflect the lane lines.
7) From the list of detected lanes, select the fit to find the boundary of lanes.
8) Warping the image back to camera plane
9) Predict the turning direction.

## 2.2 PREPROCESSING THE IMAGE

The given dataset contains distortions such as radial distortion, spherical aberration, etc. as the video is captured from camera. Due to this, the lanes sometimes look curve instead of straight, as move gradually from the center. The following preprocessing was applied to make image suitable for lane detection:

1) Undistort the Image was done using the given camera calibration matrix and distortion coefficients for each of the dataset. This was using the in-built function of OpenCV, cv2.undistort().
2) Gaussian filter is applied to remove the noise from the image. The kernel size was taken as 5 x 5, cv2.GaussianBlur(image, (5, 5), 5). Histogram equalization is required to equalize the abrupt changes in intensity values.

## 2.3 COLOUR SPACE TRANSFORMATION

Apart from the lanes, any other information is not needed to estimate the lanes on the route. Since the image's road lanes are white and yellow, masking out everything else in it except the white and yellow lanes is required. HSL color space is used here which deals with the Hue, Lightness and Saturation values of the pixels. The Hue is used to classify the color of object as combinations of red, green or blue. Saturation defines the vividness of the picture based on chromatic intensity. Lightness denotes the variable levels of black or white paint in combination.

For the data 1, extract the white pixel from the image using the highest RGB value. Because in RGB color space the intensity of the white pixel is 255, so it is easy to detect. For data 2, detecting lanes was not straightforward as yellow lanes were present. So, thresholding the image was done as per the values of the yellow, which was determined experimentally, as shown in table below.

| | | | Lower value | Upper value |
|---|---|---|---|---|
| Data-1 | White | RGB | (200, 200, 200) | (255, 255, 255) |
| Data-2 | Yellow | HLS | (10, 120, 120) | (70, 200, 250) |

Table-1: Threshold values for both datasets

## 2.4 PERSPECTIVE TRANSFORMATION

Almost all lanes look parallel to each other when looked from the top. But the actual camera on the autonomous cars does not look from the top. So, it is preferable to get the bird's eye view of the lane lines using perspective transformation to fit the curves accurately in the later stages.

### 2.4.1 what is homography?

The projective geometry of two cameras and a world plane is defined by homography. Homography is a technique for mapping images of points on a world plane from one camera view to another. Choose the four set of points from first image and another four points from second image. Homography maps the points from first image to the corresponding points in second image.

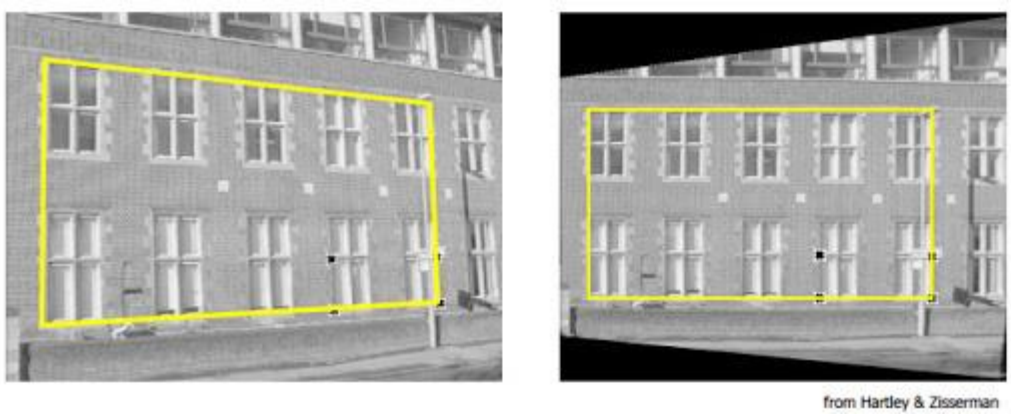

Figure 5: Concept of homography

Homography of a matrix can be solved using singular value decomposition (SVD) which is a factorization of a real or complex matrix that generalizes the eigen decomposition of a square normal matrix.

For any matrix of m x n (8 x 9) system,

$$A = U\sum V^T$$

Steps for finding Homography matrix H using calculations of SVD are:

1. Finding the eigenvalues and eigenvectors of $AA^T$ and $A^TA$
2. Eigenvectors of $AA^T$ and $A^TA$ makes up the columns of U and V respectively which is calculated by the in-built function np.linalg.eig() in NumPy.
3. Sorting the eigenvalues and the corresponding eigenvectors in descending order.
4. Singular values in $\sum$ are square roots of eigenvalues from U and V matrix.
5. The elements of the last of columns of the V matrix gives the values of the homography matrix H.

### 2.4.2 Bird's eye of view

To view lanes as parallel lines, computation of homography is required on the undistorted binary image. Four points are chosen manually to compute as shown in table 2, which were determined experimentally. Top-down images can be obtained using cv2.getperspectivetransform() and cv2.warpperspective() function.

| | Initial points | Final points |
|---|---|---|
| Data 1 | `np.float32([[150, 495], [950, 495], [530, 280], [740, 280]])` | `np.float32([[0, 400], [200, 400], [0, 0], [200, 0]])` |
| Data 2 | `np.float32([[150, 495], [950, 495], [530, 280], [740, 280]])` | `np.float32([[0, 400], [200, 400], [0, 0], [200, 0]])` |

Table-2: Considering coordinates for Top-down image

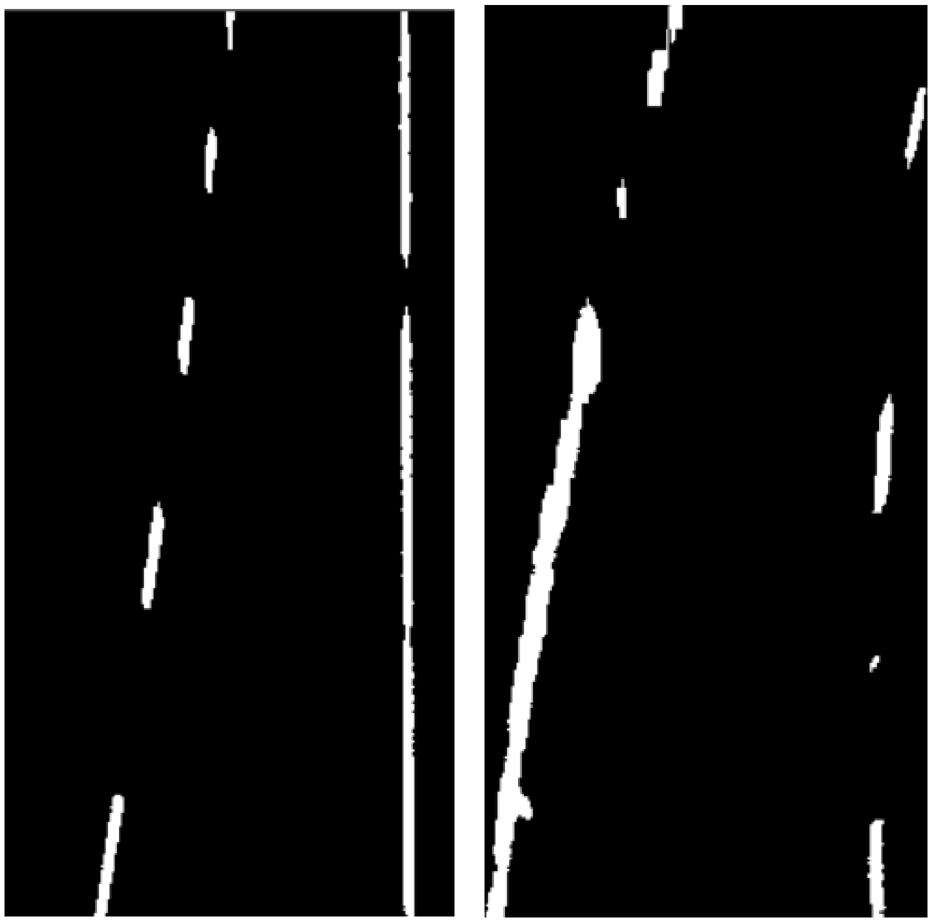

Figure 6: Parallel lanes after applying Homography transformation

## 2.5 DETECTING LANES AND CURVE FITTING

We need to extract the information about the respective lane pixels after we get the binary picture with only lane lines in it. The histogram is used to distinguish between the left and right lanes. The schematic representation of the tonal spectrum in a visual picture is called a histogram. It reveals the distribution of intensity values within an image.

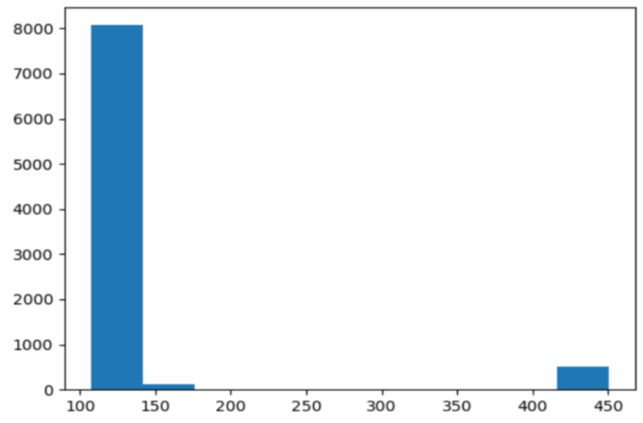

Figure 7: Histogram of detected lane

Choosing the lanes candidates is the essential step from the pixel of the individual lanes. For this I have used the sliding window approach. This is done as follows:

1) Creating a different window for each lane while retaining a certain margin from the lane center.
2) The windows can be visualized using only one frame in it. So, dividing the image into 20 horizontal strips.
3) Extracting the white from the individual window as there are already two separate regions. The non-zero function is used to append the indexes of those pixels into a separate list of left and right lanes by traversing over the height of the frame.
4) Updating the position of window center as the mean of the all-white pixels present in a window.
5) Two separate list of right and left candidates are obtained using this approach.

From the points obtained, A polynomial function is needed to match the pixel positions after having the white pixel indices for both lanes in different lists. The polynomial of degree 2 can be found using numpy.PolyFit() function, which returns the model parameters of equation. The reason behind choosing the polynomial of degree 2 is take the consideration the curve during turns.

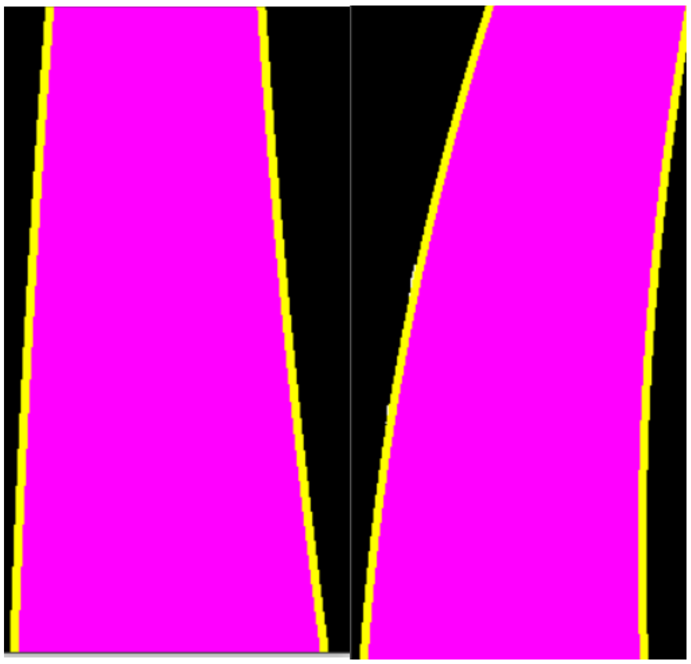

Figure 8: Generated polygons in bird’s eye view for both dataset-1 and 2.

Finally, applying inverse warping for transferring the picture back to the original and merge the two images together with cv2.addWeighted(). Here, images are given different weights to manipulate the transparency.

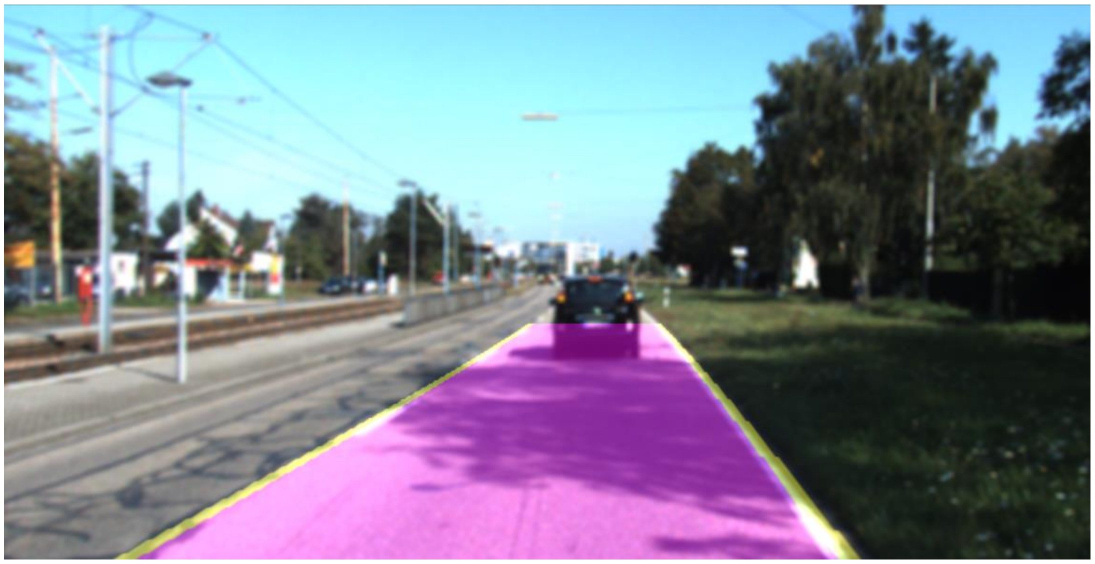

Figure 9(a): Overlaying polygon on data_1

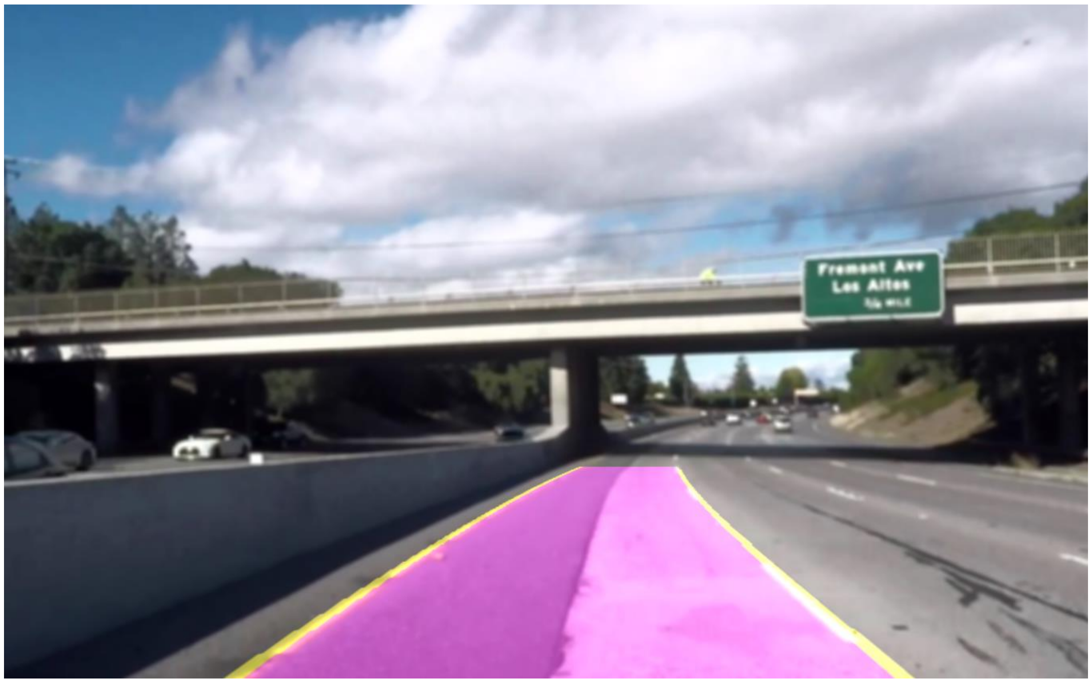

Figure 9(b): Overlaying polygon on data_2

## 2.6 TURN PREDICTION

The final step is to predict the turns for self-driving cars. The real problem is that one cannot make any of turn predictions around a static point like the frame base. So, Turns are predicted based upon the difference between vehicle center and lane center. By comparing the left and right lane polynomials at the maximum Y and finding the middle point, the lane center can be determined. At minimum value of y, center of vehicle can be taken be taken as the mid-point.

If difference of lane center and vehicle center is **+ve**, then vehicle is turning right,

If difference of lane center and vehicle center **-ve**, then vehicle is turning left,

If difference of lane center and vehicle center is **zero**, then vehicle is Going straight.

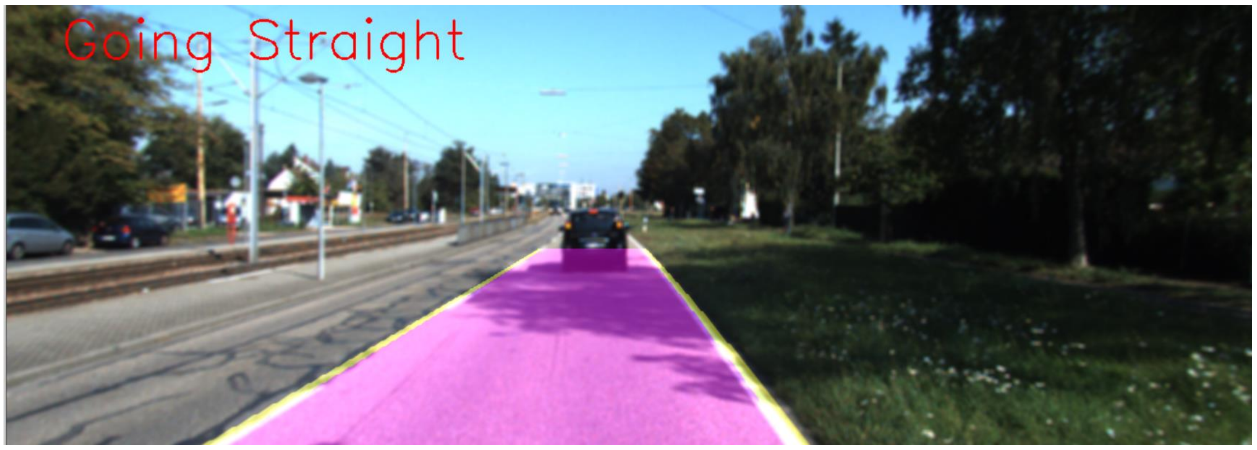

Figure 10(a): Turn Prediction for data 1

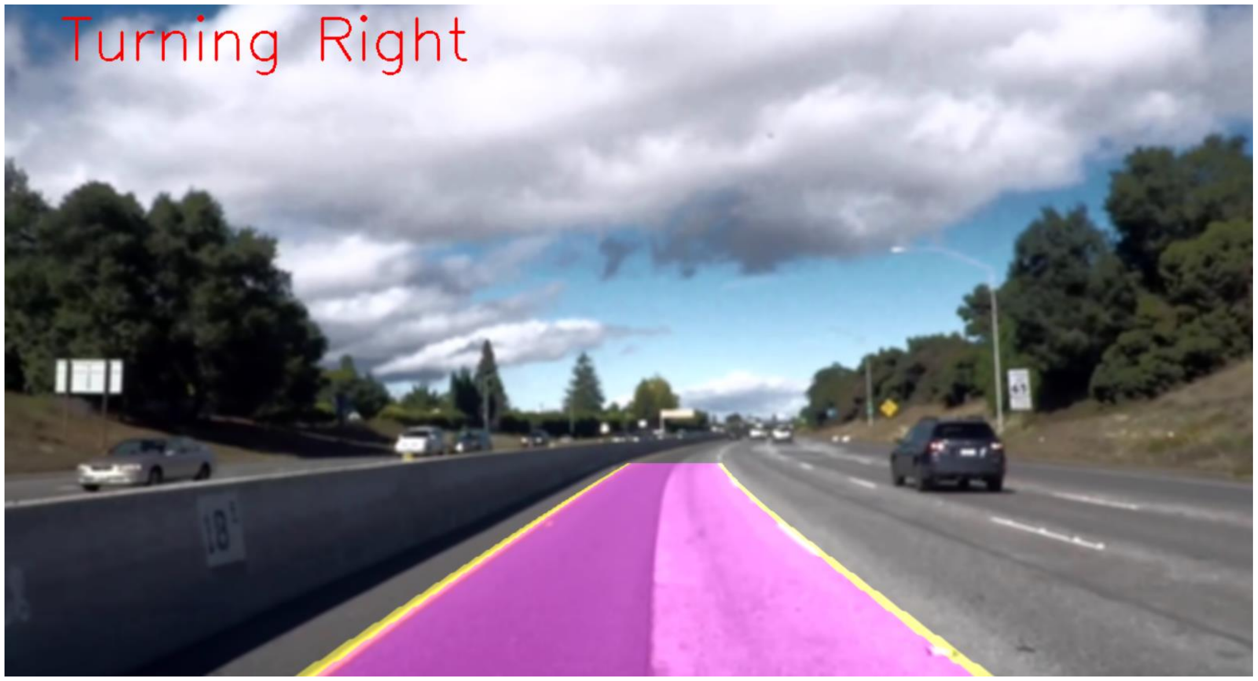

Figure 10(b): Turn prediction for data 2

Figure 10: Output of turn prediction

## 2.7 HOUGH LINES TRANSFORM

The Hough transform was initially designed to identify lines, but it has since been expanded to include arbitrary shapes like circle, ellipse, etc. It is a feature extraction technique that uses a voting scheme.

### 2.7.1 Line representation in Hough space

The line in the Hough space can be represented by two parameters a and b which correspond to slope and intercept,

$$y = ax + b$$

But this form of equation is not feasible for vertical lines. Therefore, the line can be represented using the pair (d, θ) in polar system,

$$d = x \cos\theta + v \sin\theta,$$
$$\theta = \text{angle made by the line},$$
$$d = \text{perpendicular distance}$$

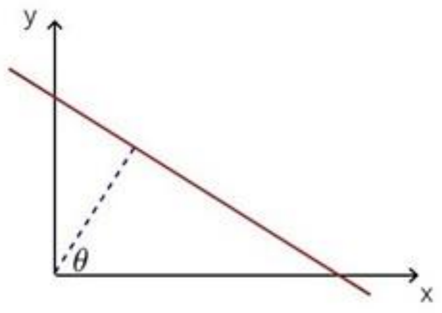

Figure 11: Representation of line in polar coordinate system.

The Hough space for lines has therefore these two dimensions; θ and d, and a line is represented by a single point, corresponding to a unique set of parameters (θ, d). The line-to-point mapping is shown in Figure 1.

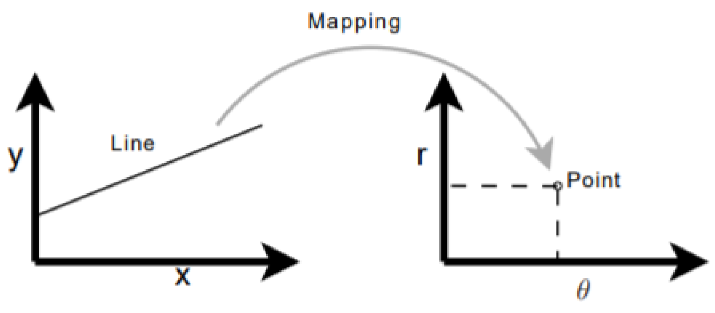

Figure 12- Line mapping to point in Hough space

In Hough space, drawing an infinite number of additional lines intersecting at this one point will yield a continuous sinusoid.

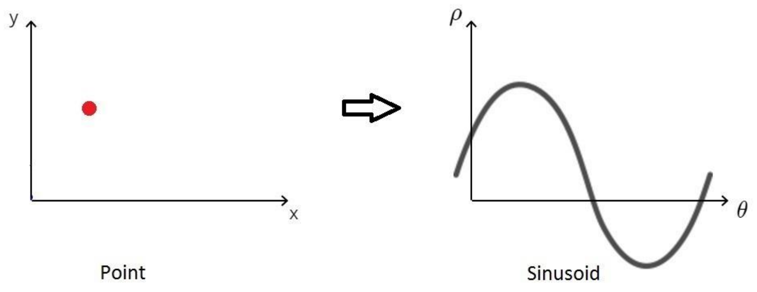

Figure 13– Point represents to sinusoid in Hough transformation

### 2.7.2 Lane detection using Hough Transform approach

Another approach for detecting lanes for self-driving cars is using Hough Transform. The process is as follows:

1) Applying smoothing filters to denoise the video frames.
2) Grayscale the image and determine the edges in the images using canny edge detection or any other approach.
3) Initialize the Hough matrix H [d, θ]=0
4) For each edge point in the image, check whether the pixel is an edge pixel. If the edge pixel is present, loop through all possible values of θ = 0 to 180. Find the equation of line in form of $d = x \cos \theta + v \sin \theta$ for each index and add the vote.
5) Find the highest value in the H matrix, get the d and θ index, get the value of d and θ from the index pair which fits the most edges in the image. For detecting lanes, determine the line of best fit by removing some extra lines using filters.

## 3 Conclusion

1) For Night Drive video, applying the histogram equalization resulted in a lot of noise and distortion in image.
2) The pipeline fails to estimate lanes during the sudden changes in brightness or illumination level which is clearly seen when vehicles pass under the bridge in data_2.
3) Initially, I tried applying sobel filter for segmentation. But it was inaccurate due to presence of noise.
4) To detect the curve of the road, the idea of traversing the sliding window over the image height was crucial.
5) Significant amount of time was spent about understanding various color spaces. Yellow lanes in challenge video were not detected easily. This was done effectively using the HLS color space.

6) The Hough lines approach did not detect curves along the lane (white and yellow) pixels, so it was not used with the Lane detection system. As a result, I used the Histogram of Pixels method to include the curves in the pixels around the frame.

## 4 References

- Fariss Abdo, "OpenCV Python equalizeHist colored image," *Stack Overflow*, Aug. 13, 2015. https://stackoverflow.com/questions/31998428/opencv-python-equalizehist-colored-image.
- A. Rosebrock, "OpenCV Gamma Correction - PyImageSearch," *PyImageSearch*, Oct. 05, 2015. https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction/.
- PyData, "Ross Kippenbrock - Finding Lane Lines for Self Driving Cars," *YouTube*. Jul. 26, 2017, Accessed: Apr. 05, 2021. [Online]. Available: https://youtu.be/VyLihutdsPk.
- J. He, S. Sun, D. Zhang, G. Wang, and C. Zhang, "Lane Detection for Track-following Based on Histogram Statistics," 2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), Jun. 2019, doi: 10.1109/edssc.2019.8754094.

## 5 Video links

- https://drive.google.com/drive/folders/1Y-VKErxDD6Hp_P6TR6ah5lAtS-rSJ1VG?usp=sharing