Project 2

PERCEPTION FOR AUTONOMOUS ROBOTS ENPM 673 SPRING 2021

UNIVERSITY OF MARYLAND, COLLEGE PARK

Siddharth Telang
stelang@umd.edu
116764520

PROBLEM 1.

Improving the Quality of Video Sequence

We are given a video sequence from a vehicle during the night where the lighting is very low. Our aim here is to increase the quality of the video sequence so that the features can be detected for autonomous driving. The method which we will be using is Histogram Equalization.

Histogram equalization is a method to increase the intensity of each pixel such that the final histogram of pixels is distributed over the complete range of the image and not just in one region. For this, we use a Cumulative distribution function, which is monotonic function. The CDF is given by the summation of the fraction of pixels with an intensity less than a specific value. CDF $C(i) = [\Sigma h(j)/N]$ for $j <= i$. Then we multiply each pixel intensity's corresponding CDF value with 255.

If we have a RGB image, we split the image into three channels – Red, Blue, and Green, apply histogram equalization to each channel individually and then combine the channels back together.

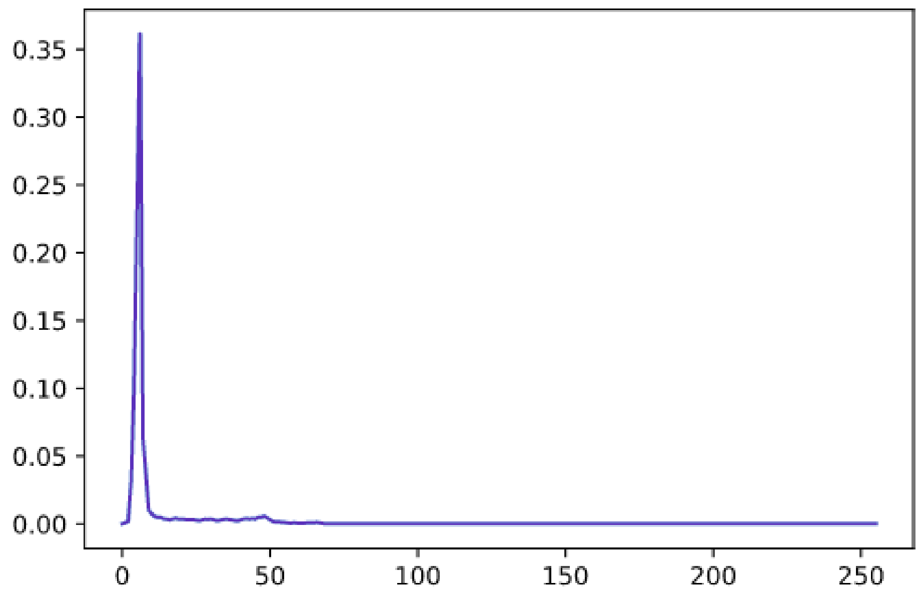
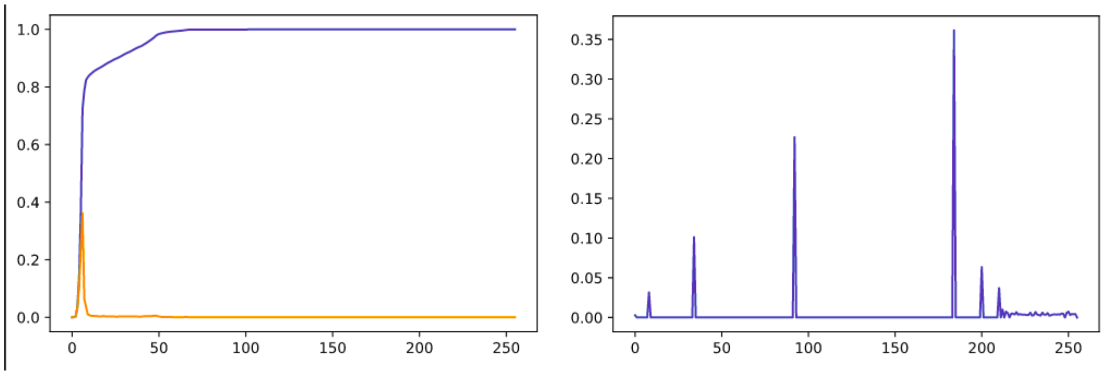


Figure showing Histogram for one frame in video.

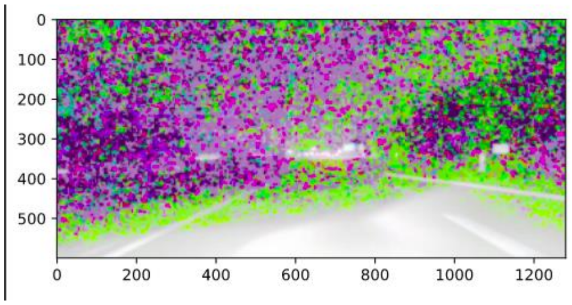
We can see from the above histogram that most of the pixels are in the low light area and the pixel count at 255 is very low.

The below figure shows the CDF function(left) and the histogram after equalization(right). We can clearly see that after equalization, the histogram has been shifted to right side.

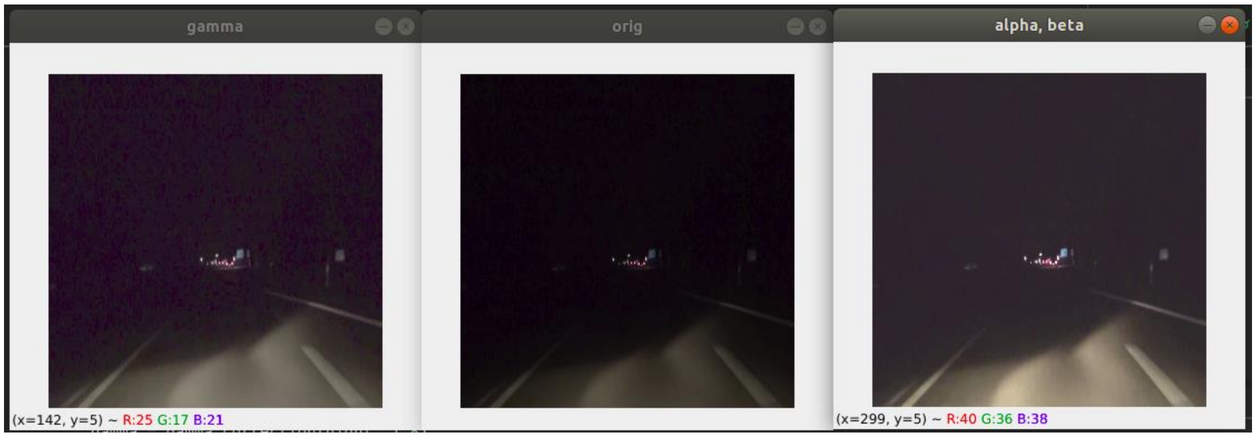
However, there is no such significant improvement in the quality of the image as there is minimal light in the frame. We can also observe this from the histogram on the right.



Moreover, the colors are also not proper after this equalization method. The figure below shows the final output of the method used.



We compare this output with other correction methods like Gamma and Alpha/beta and find that the gamma correction and the alpha/beta correction do a better job than our manual method, but still the quality is bad. **Output videos can be found** here



(a) Gamma Correction (b) Original Image (c) Alpha, Beta Correction

Problem 2

Lane Detection

In this task we are given two video sequences from self-driving cars and our aim is to detect the lanes. The camera mounted on the car captures the images and we have to process these images to get the lanes.

But the camera looks at the road at an angle and thus, the lanes do not look parallel in the camera image frame. We know that if we compute homography on a region of interest and get the perspective transform, we can make the lanes look parallel and work on them. In addition, we have been provided with the camera and distortion matrix to un-distort the image.

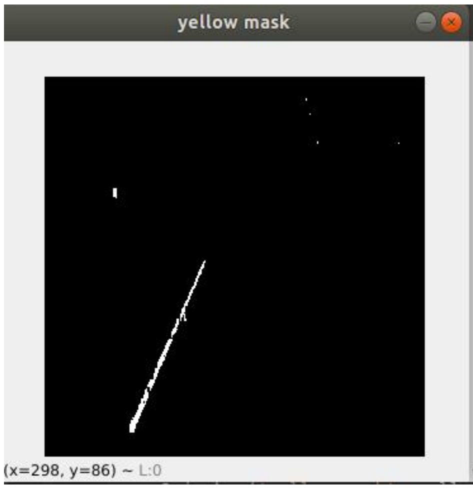
There are two methods which I used here to detect the lane

- Hough Lines
- Histogram of lane pixels

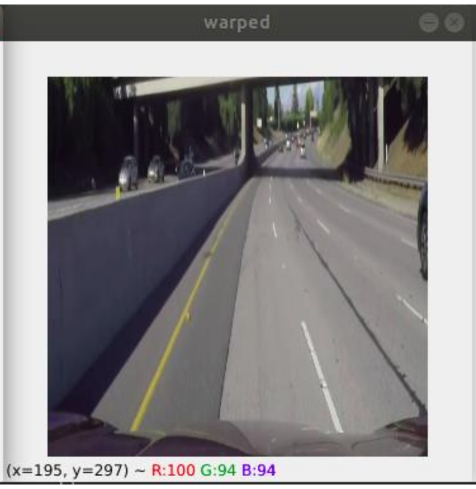
We will see the Hough lines method and its drawback and then move to Histogram of lane pixels method.

Color Thresholding

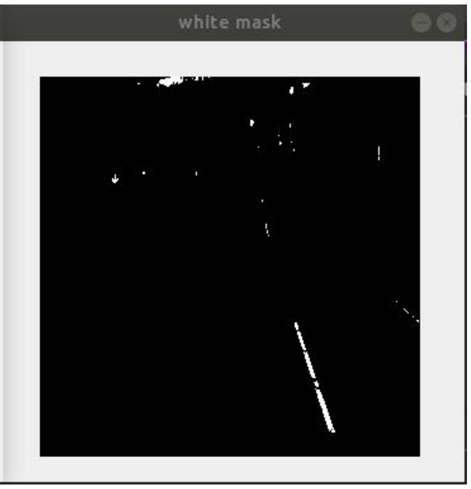
This is the first process that we do on the image to detect the yellow and white lanes. Our image is currently in RGB format. However, there exist various other formats like HSV, HSL, to name a few. We then check which one of these formats best isolates the yellow and white lanes in the image. On doing some research and tests with these formats, I chose HSL format (HLS in openCV) as it gave near perfect results and was able to isolate the colors pretty well by choosing an appropriate threshold.



Yellow HLS mask



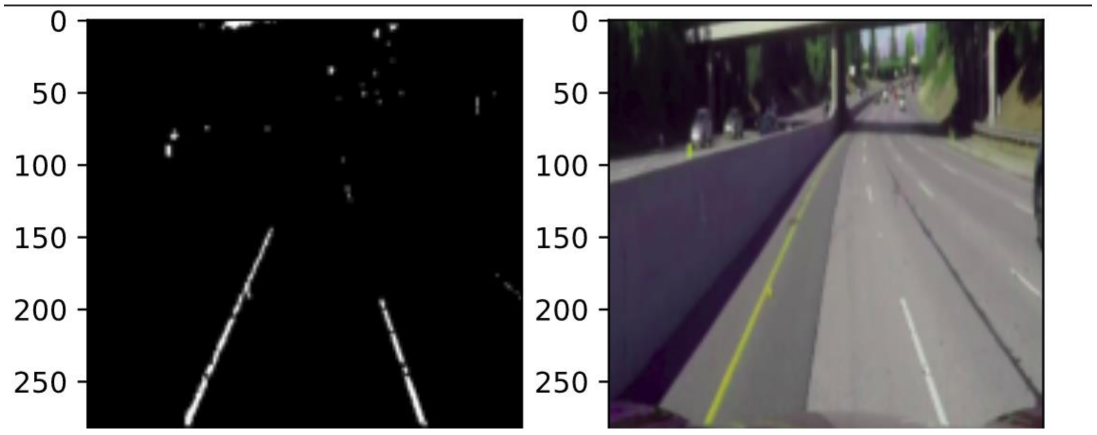
Original image



White mask

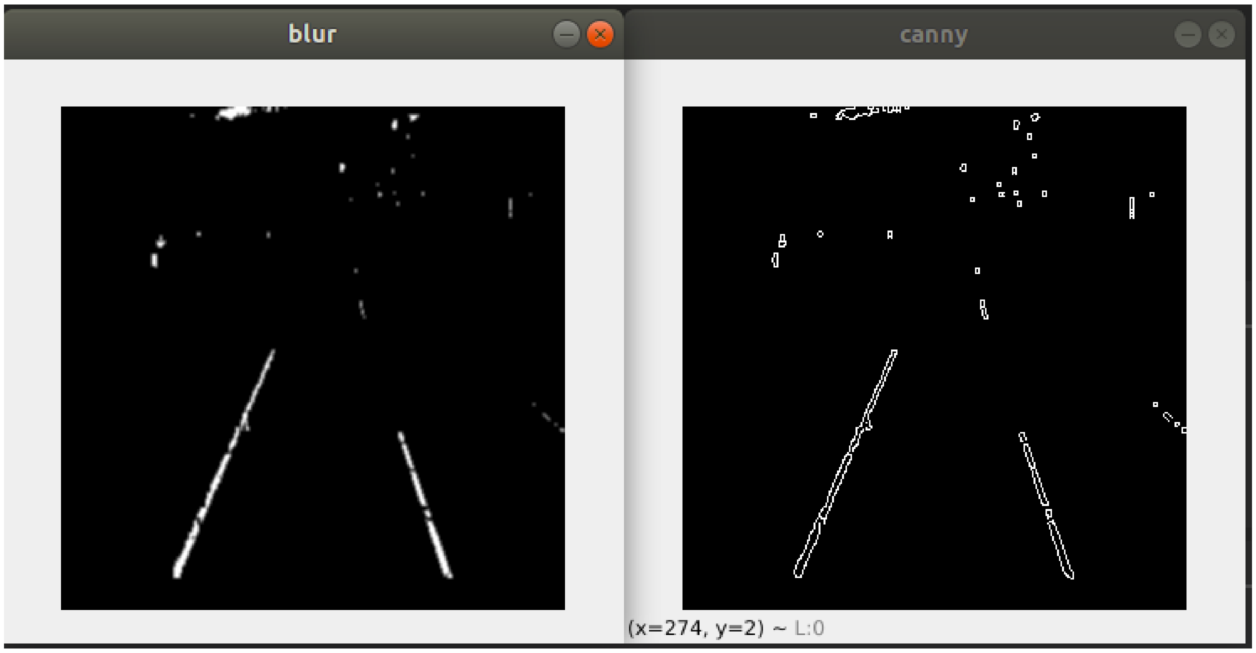
The above images show the yellow mask(left), original warped image using perspective transform(middle), and white mask(right). We can clearly see that the HLS thresholding is good at isolating the yellow and white colors, which are nothing but our lanes.

The below figure shows the combination of the masks.



Canny Edge Detection

Now, as we have the binary image with us, we can easily apply edge detection to detect the edges of the lane which will give us the coordinates. Before that we remove noise from the image by first converting to grayscale, blurring using Gaussian blur and finally apply Canny edge detector.



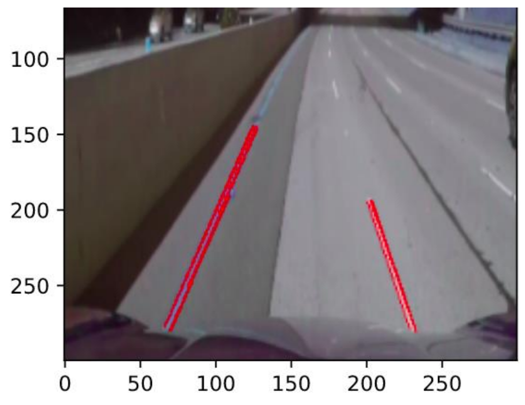
Hough Transform

A line in image space becomes a point in Hough space and point in image space transforms to a line in hough space. Using this principle, we can extract the desired lines from Hough space.

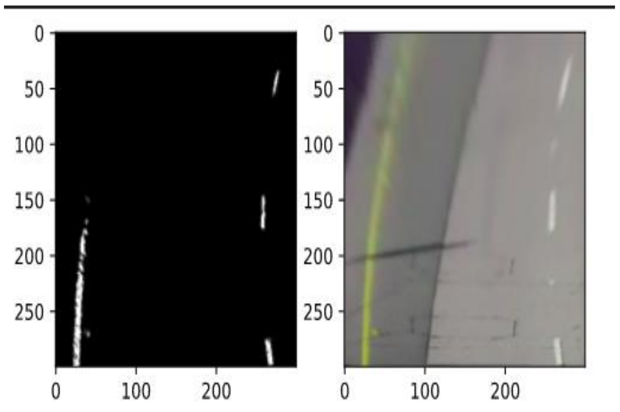
As we already have the points in image space from edge detection, we can find out the corresponding lines in Hough space. All the points on the specific lane will have only one slope and intercept and hence there would exist multiple intersecting lines for these points in the Hough space.

A greater number of intersecting lines will help to choose the points on the image space which lie on the lane. To do this, we move to polar coordinates as we cannot work on tan(x) - the slope in cartesian coordinates. A line in polar coordinates is made up of distance 'd' and angle 'θ'.

The algorithm here is to initialize an array of d and θ and check how many points in image space have the same d and θ values. If they are more than a specified threshold, we can say that those points lie on the line/lane. We use the inbuilt openCV funtion and get the below result.



(a)Hough transform application to detect lanes.

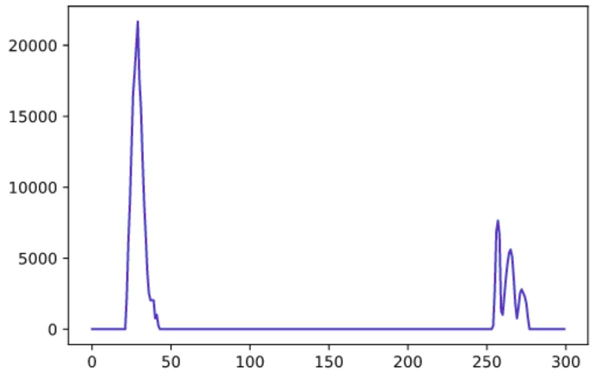


(b) Curved lanes

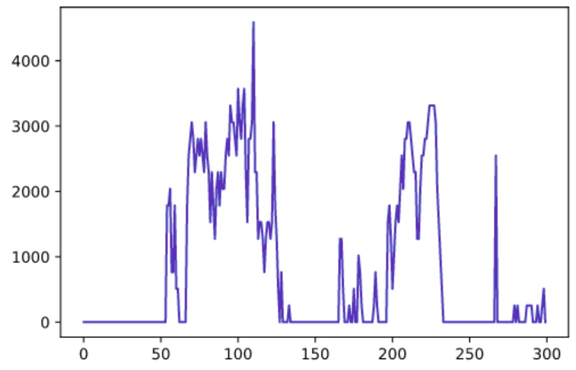
As we observe here, we can use Hough transform to detect straight lines. However, not every lane is straight and will have curves. In that case, this method will not give us the perfect result. For example, the above image on the right shows a road curvature, and Hough transform will not be able to detect the curved lane. Due to this, we will have to use another method for detecting the lane through the histogram of pixels.

Histogram of Pixels

Here, we need the homography step, taking any random four points on the lane and warping the image, having a bird eye view. A histogram of the processed warped threshold image will provide us with a measure of how many "white" pixels lie along the vertical axis, which will help to identify the lane points. If we take the histogram along the vertical direction, we get the output as per below – figure(a) is the histogram of the curved road (figure above), which is without any noise and does not have any other surrounding pixels, while figure(b) shows the histogram where there are lot of surrounding pixels other than the lanes which were in the "white" region.



Figure(a) No adjoining region pixels

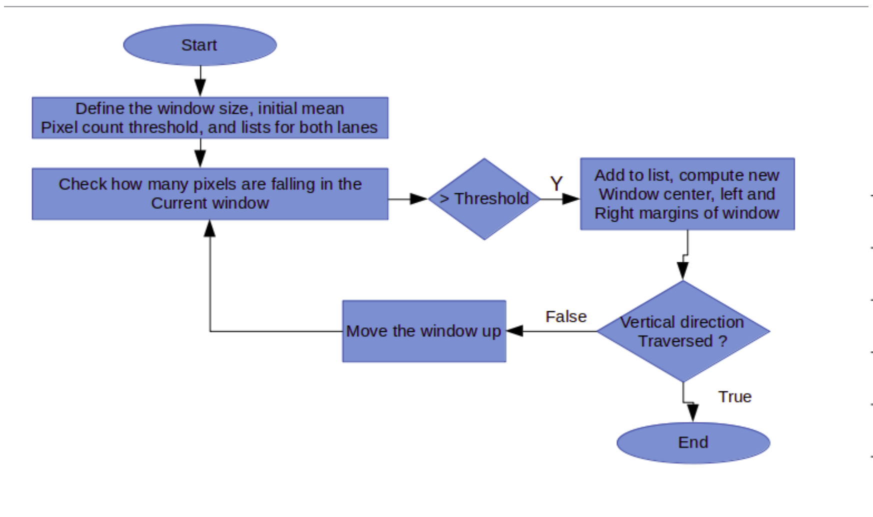


Figure(b) with Surrounding pixels

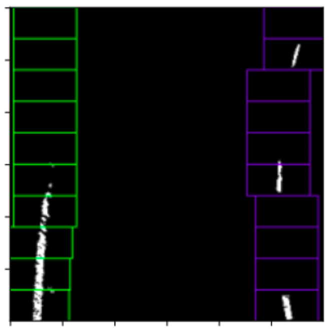
As we can see from the above two histograms, there are two peak values. These peak values are the lanes, and hence, we can get these two peak values coordinates. Once we get these, we can proceed with detecting the complete lane and curve fitting.

Lane Points Detection

From the above histogram, we have one point on both the lanes having the maximum pixel count. Our aim is to get all the points on the detected lane. For this, we will have to traverse through the entire vertical direction across the point detected point on horizontal axis (using histogram). We set up a window of a fixed area having the initial center as the horizontal point detected on both the lanes.
Then, we move this window vertically to check how many non-zero pixels are falling in the area and add that to a list. If for a particular window the pixel count is more than a specified threshold, we consider this as a point on the lane and shift the window's center point – the new mean of the non-zero-pixel points. This process is done for both the lanes for the complete vertical direction and at the end we get the non-zero lane coordinates. The flow chart below explains this step and the visualization of the same.


Start
Define the window size, initial mean Pixel count threshold, and lists for both lanes
Check how many pixels are falling in the Current window
> Threshold
Y
Add to list, compute new Window center, left and Right margins of window
Vertical direction Traversed ?
False
Move the window up
True
End

(a)Flow chart illustration of the lane coordinates detection

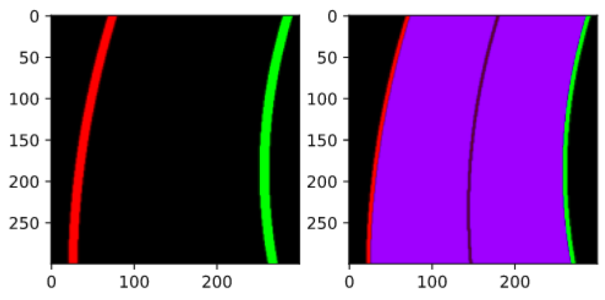


(b) Illustration of the process

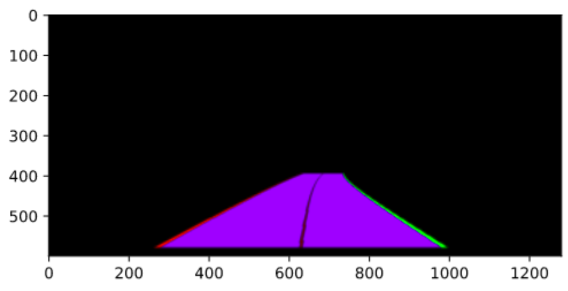
Polynomial Fitting

The next step is to fit a curve using the above detected points which will be our final lane. OpenCV already gives us an inbuilt function to estimate the polynomial giving the set of points and the degree. We use this to get the polynomial function – the coefficients a,b,c to be precise in $\mathbf{x = ay^2 + by + c}$.

Moreover, using the polynomial of the two lanes, we can compute the center curve which can be assumed as the mean of those curves. Next, we can easily draw a polygon and line using openCV functions on an empty image.

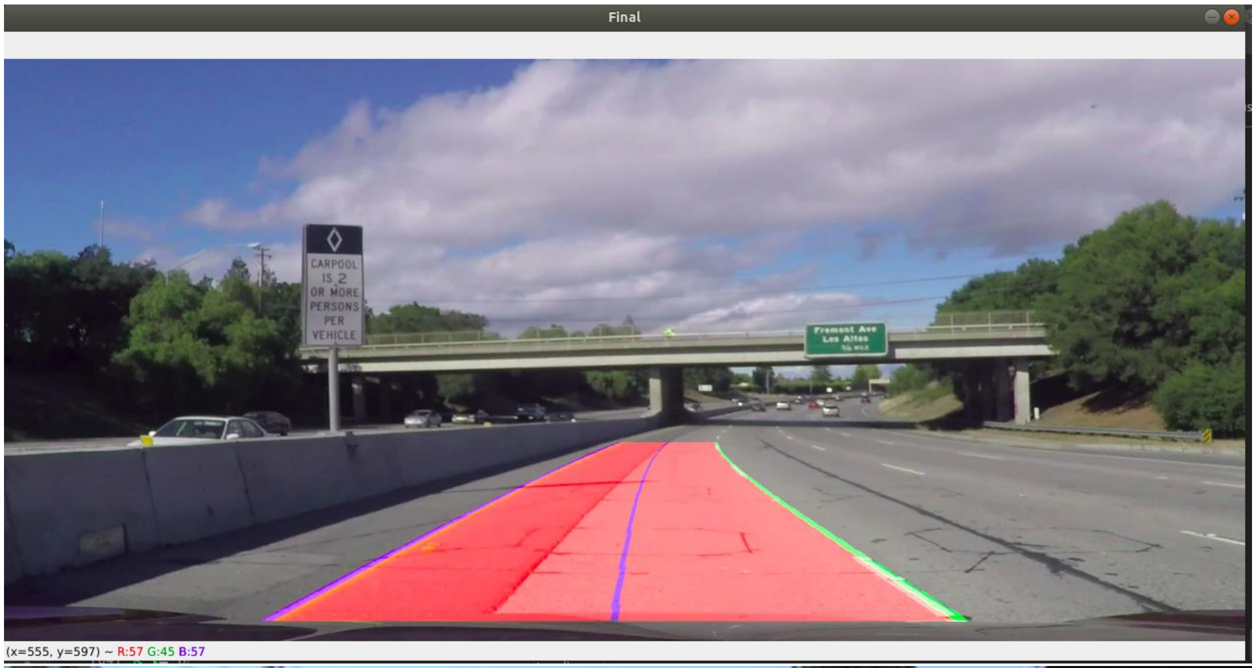


(a) Line and Polygon filling

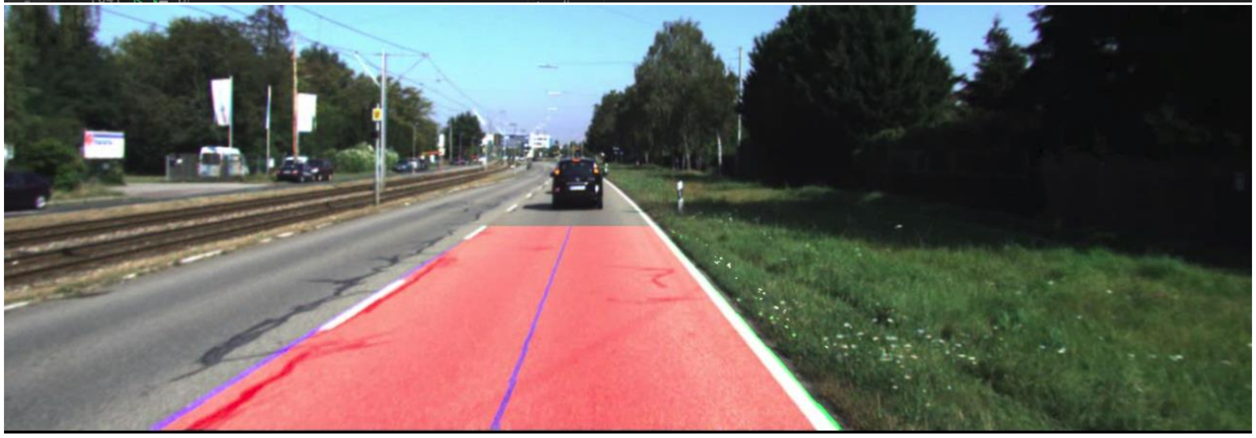


(b) Inverse warping onto the original image

Figure above shows the polynomial and line filling and the inverse warping of the filled polynomial onto the shape of the original image.



```
(x=555, y=597) ~ R:57 G:45 B:57
```

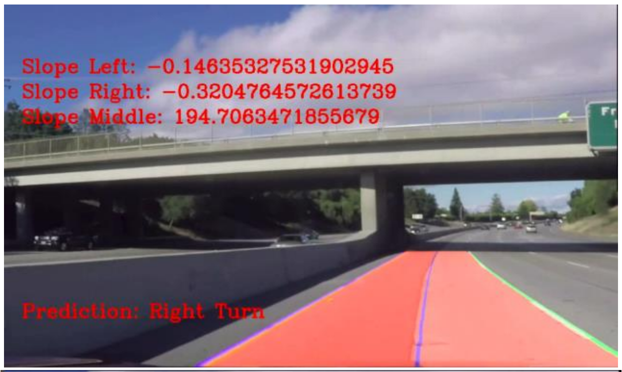


Figures showing the blending of the original frame with the filled warped image.

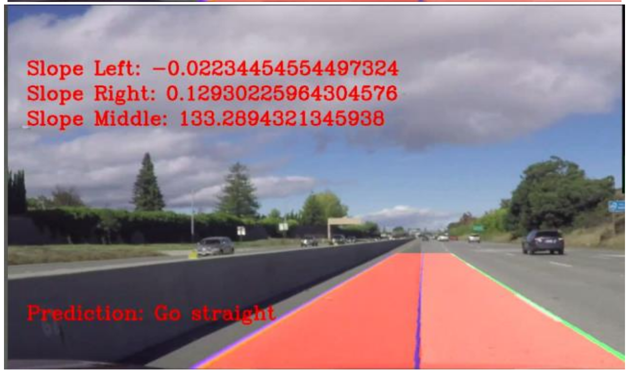
After this, we can blend the original frame and the image on which lanes are drawn to get the output as the image shown above.

Turn Prediction

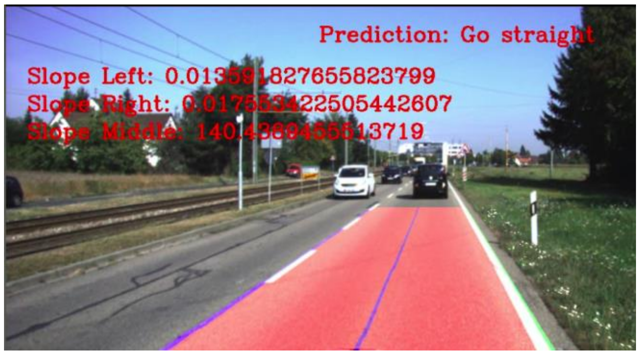
To predict the turn – either left or right we inspect the slope of the center curve. The coefficient corresponding to the first order of variable can be considered for this. Then, we define a threshold for this for left, right and straight motion and output on the frame. The



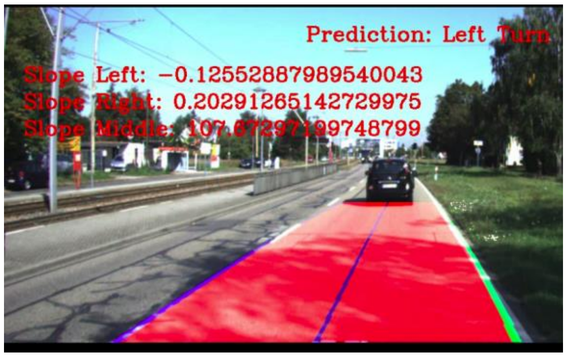
```
Slope Left: -0.14635327531902945
Slope Right: -0.3204764572613739
Slope Middle: 194.7063471855679
Prediction: Right Turn
```



```
Slope Left: -0.02234454554497324
Slope Right: 0.12930225964304576
Slope Middle: 133.2894321345938
Prediction: Go straight
```



```
Prediction: Go straight
Slope Left: 0.013591827655823799
Slope Right: 0.017553422505442607
Slope Middle: 140.4389455513719
```



```
Prediction: Left Turn
Slope Left: -0.12552887989540043
Slope Right: 0.20291265142729975
Slope Middle: 107.67297199748799
```

Output images while deciding the turn threshold.

The link for the final videos can be found here.

Problems encountered and solutions implemented

1) Canny edge detector and Hough Transform

It was seen that Canny edge detector gave the edges of the shadow in the image, which aggravated the image for detecting the lanes if the shadow falls in between the lanes.
Hough transform as we saw before, only gives us straight lines and does not give good results when the road has a curvature.
Due to this, I chose color thresholding and histogram of pixel method to detect the lanes.

2) Color thresholding

RGB thresholding was not holding good for the frames to detect the white and yellow lanes separately. Thus, I explored the HSV and HSL formats and found that HSL was able to isolate the yellow and white lanes perfectly for both the videos and a single threshold set was found using the tool on https://www.w3schools.com/colors/colors_hsl.asp

3) Polynomial Fitting

As color thresholding was used here, there are frames in the video where due to bad lighting and orientation conditions, very few or no points are detected. In that case if we try to fit a polynomial two things can happen:

- Crash in software due to null points
- Incorrect polynomial fit – drastic change in coefficients

To resolve these issues, I stored the last well-known polynomial fit and if no points are detected for a frame, use those coefficients.
In addition, if the polynomial shows a drastic change in the direction of curvature, neglect that and take the last well-known polynomial. This was done by making a class and storing the good configuration in the class variables.

Conclusion

This pipeline involves color thresholding and detecting the lane points using histogram of pixel method. It is a generalized pipeline, except for one homography step and will give good results where the lighting conditions are good. Moreover, the polynomial fitting considers the last well-known coefficients if there is a sudden distortion caused by improper thresholding, noise and other parameters.