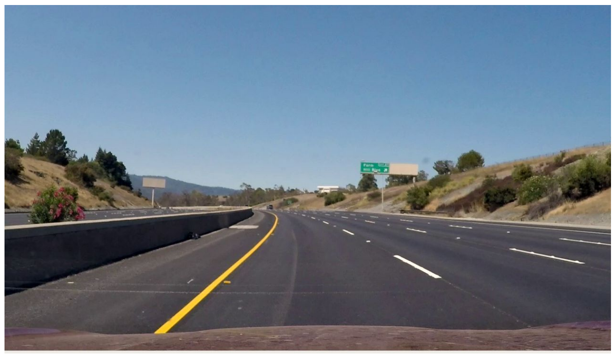
ENPM673 - Perception for Autonomous Robots
Project 2 - Lane Detection

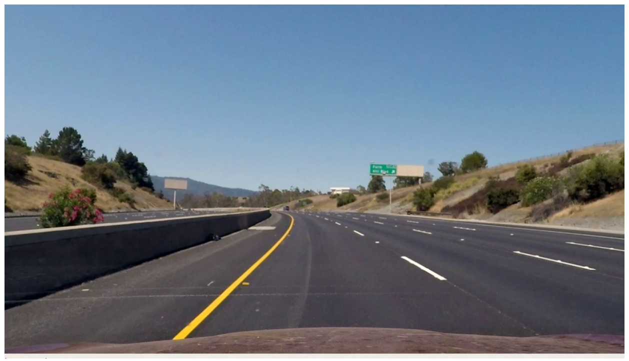
Abstract
Lane detection is vital to develop algorithms in autonomous driving robots and self-driving cars. In the real world scenario various obstacles like changing weather conditions, light conditions, irregular markings, dirty road surface, confusion of new/old lane markings, curvature and type of road, come into play and pose a serious challenge on lane detection requiring the lane detection pipeline to be robust.

1. Detection Pipeline

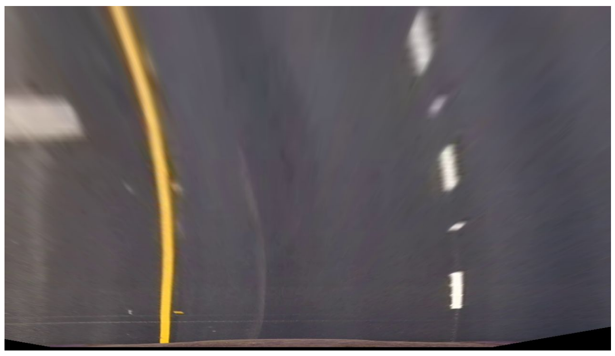
1.1 Preparing Input

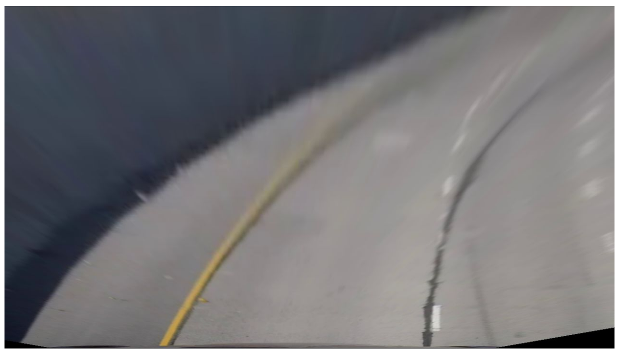
- In real time, the camera's lens does not capture the true image but a distortion of the original image sequence where the points have higher distortion from the center. This pincushion distortion can be caused by various factors such as usage of inexpensive lens, difference in distance from the center of camera or perspective distortion. Hence, the first step is to ***undistort*** the image.


Original Image


Undistorted Image

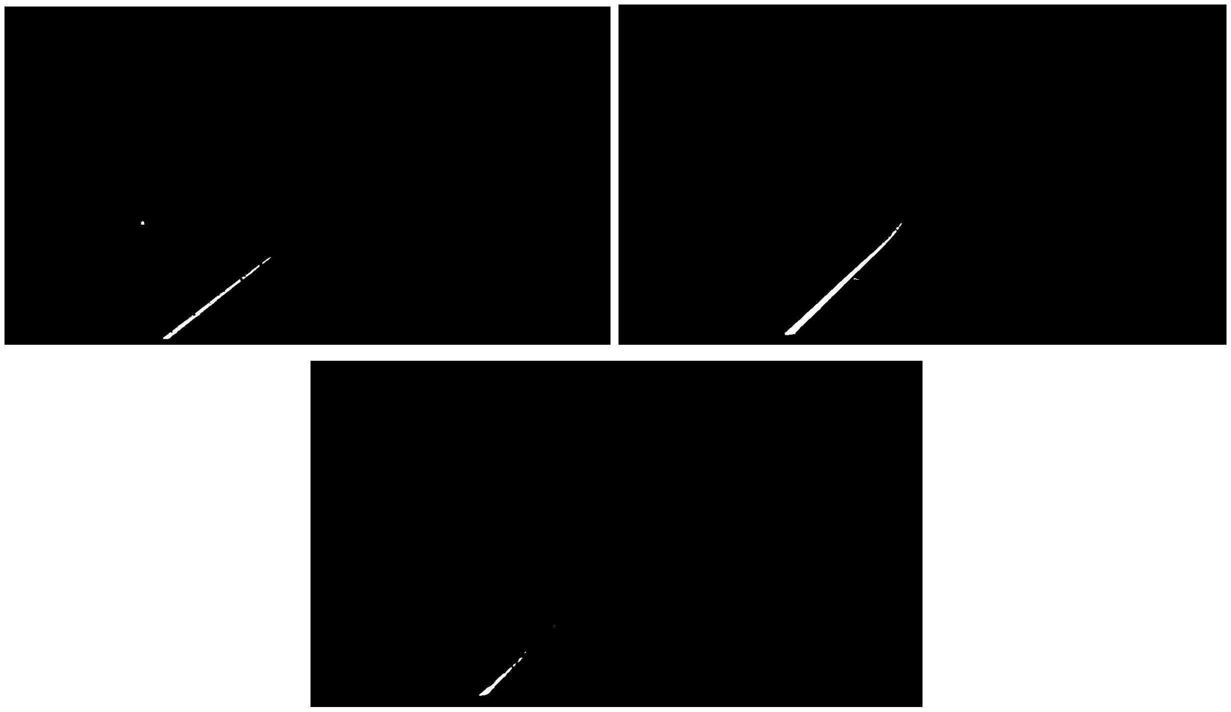
- To get a top/birds eye view of the image apply ***perspective transform***. We identify 4 points in the original camera image such that the top part of the image is disregarded and then warp the image such that the region between the 4 points forms a rectangular section. With the bottom part of the image as the region of interest and its perspective transform we proceed to detect the lanes.





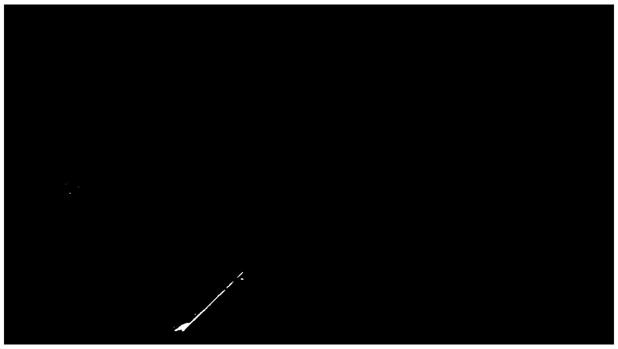
1.2 Detecting Lane Candidates

- In reality, color is a continuous phenomenon with infinite number of colors. We represent these in discrete values using color spaces. Next, we convert the bird's eye view RGB image to ***HLS*** color scheme. The Hue, Saturation and Lightness scheme is an intuitive way to represent colors where the hue represents the color, the saturation represents amount of color and the lightness stands for the average of largest and smallest color components and can be used to obtain good color discrimination. We used the HSL color space to detect the yellow and white color.



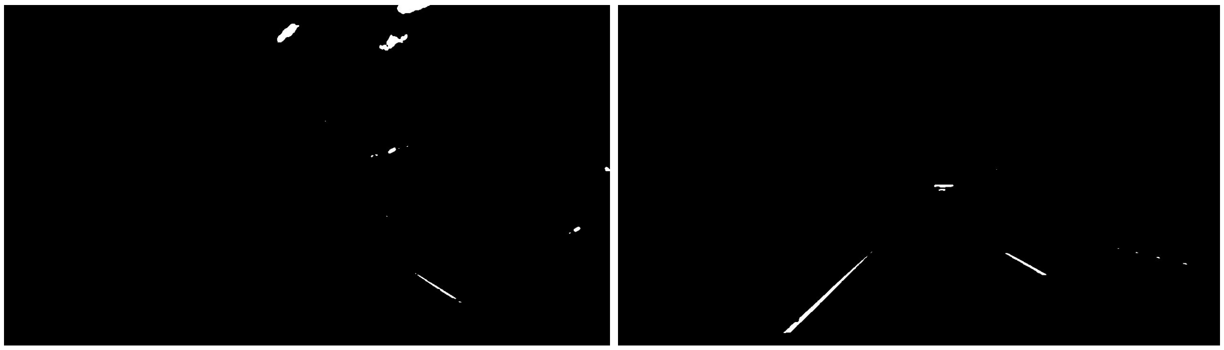
HLS Yellow filter

- We have also used yellow filter in RGB spectrum. Combination of yellow filter in HLS and RGB spectrum is used to filter out yellow color.



RGB Yellow Filter

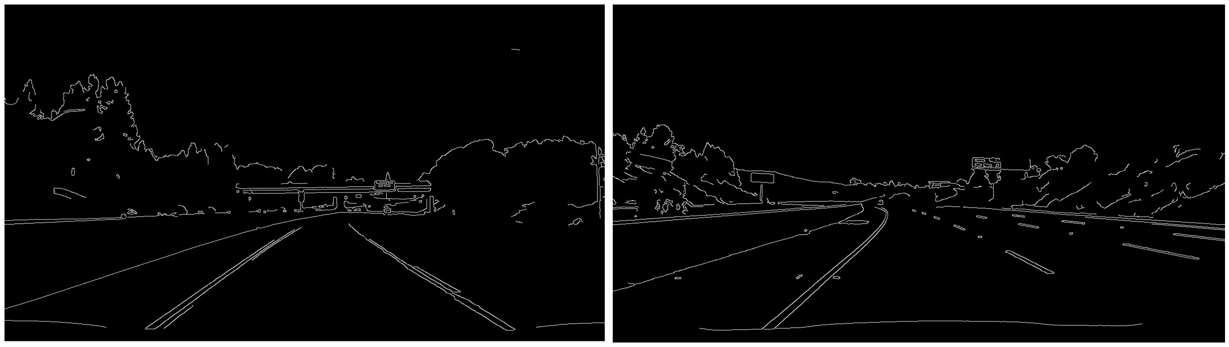
- Similarly we apply ***white mask*** with thresholds to identify white lines. With both the yellow and white masks applied, we can obtain the lanes from the color mask by combining the two lane masks as follows.


RGB White filter. 1) Challenge video 2) Project video

- A combination of color masks provide a stronger indication of the lanes. Since the lanes are expected to be white or yellow markings on dark background we will get large gradients and we will get thick bold lines as a result.

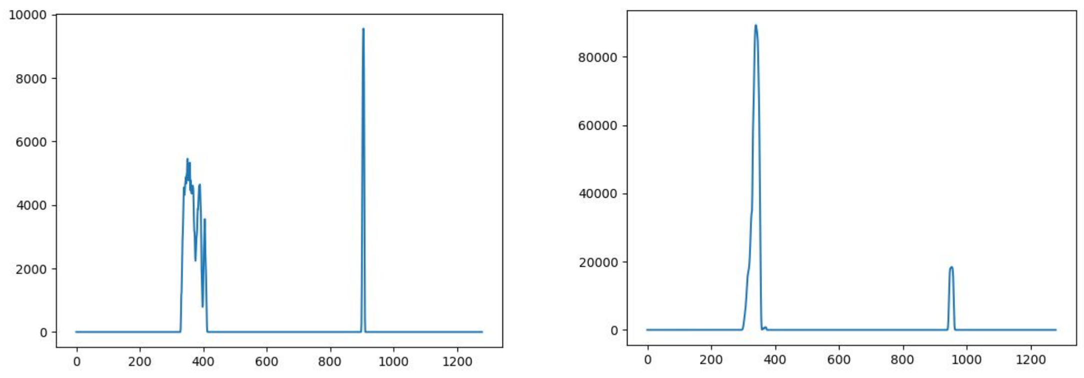
1.3 Refining Lane Candidates

- After Color Segmentation, we explored couple of gradient thresholding methods. Firstly, we computed ***Canny edge detection*** which parses the pixel values with respective to their directional directive, namely, gradient. We supplied low to high threshold ratio to detect the steep directive in at least one direction to compute gradients. Canny edge detection essentially finds the areas of the image that rapidly change over the intensity value and we will have only the single pixels which are indicative of edges. Canny edge detector is used in the project.



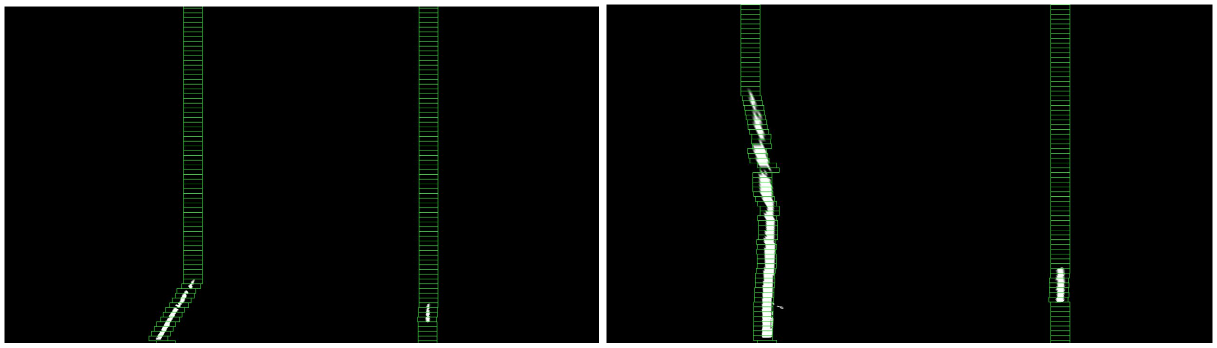
- Using a ***Hough transform***, we transformed all our chosen edge pixels of the lanes to a line or curve in Hough space.

- In Hough space, each line represents a point from Image space and each point represents a line from Hough space. We can then render the detected lines back onto the image itself to make sense of features detected in the image sequence.
- ***Sobel filter*** is also explored to detect the lane markings. Sober filter gives change in image intensity along x and y directions which can further be thresholded to obtain a binary image. However, it was not robust enough to choose just the necessary features of lane.
- Finally through observation, we generate a ***histogram*** of the binary thresholded images. The x axis contains the x position coordinates of the image and the histogram bin, y axis corresponds the sum of pixel intensities in each column for each x. We locate the highest peak and get one of out lane points. We then look for the next peak in the neighbourhood of distance relative to the lanes pixel length (lane size is constant). The location of these peaks are be leveraged as starting points to search for pixels belonging to each lane line. This works for lane change due to the fact that, number of pixels between left and right lane is known and constant and we will be able to locate the peaks there.

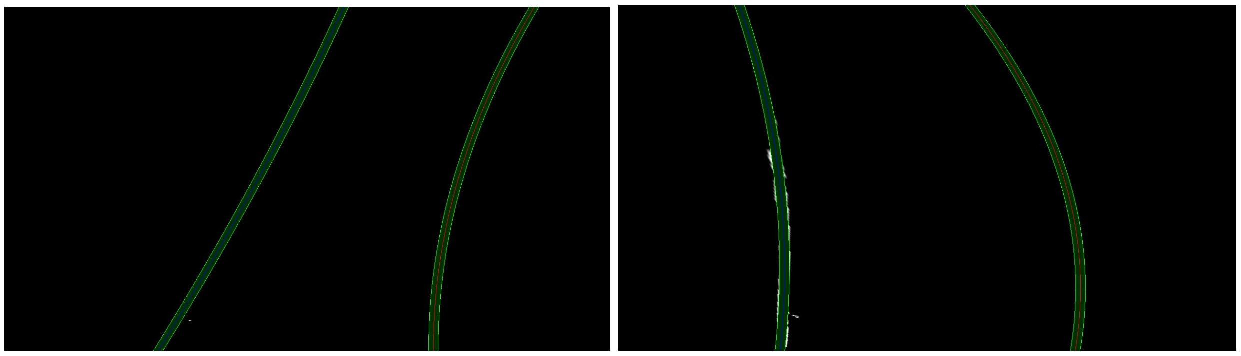


Histogram Plot 1) Challenge Video 2) Project Video

- Now that we know the starting x positions of the pixels from the bottom part of the image to yield a lane line, we can do a sliding window search to capture the pixel coordinates of our lane lines. With ***sliding window*** approach we basically split the lane line pixels into regions/windows say of rectangular shape such that peak of the pixels is in the center and we move it such that the center is maintained at the peak. This approach is used to detect the nonzero pixels within the sliding window boundary and append it to left and right lane pixels array respectively. Then we compute a second degree polynomial to get the coefficients of the curves that best fit the left and right lane line subsequently. One way to fit the pixel to the lane line was to find the mean of the chosen window and try to fit the line. Another efficient way was to do the same was to leverage the coefficients from frame t-1 to find our next lane pixels. But we were not able to find enough lane line pixels and to improve our chances to better fit the curve around our lanes we resorted to least squares.


Sliding window method. 1) Challenge Video 2) Project Video

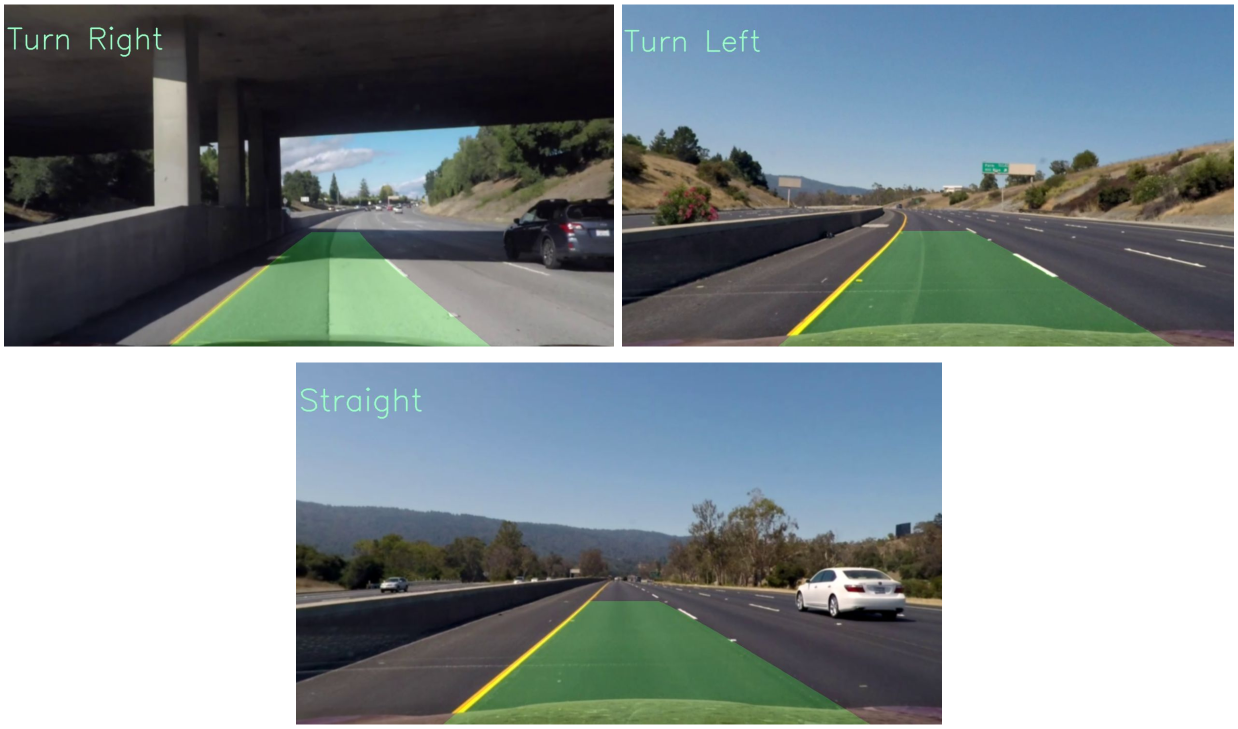
- We leveraged a differentiable ***least-squares fitting*** model that for each set of parameters returns the best fitting line in the weighted least-squares sense.
  Using the histogram peaks of the left and right lanes, to locate the ROI of points needed for fitting. These points are pushed into a list and given as an input the least squares fitting function. However, our custom function was not computationally optimized and it slowed down the processing. So to get a smooth best fit and a fast output we used the numpy's ***polyfit*** function.


Polynomial fit visualization. 1) Challenge Video 2) Project video

- Now to make the lane detection robust to external noise, we use the fact that both the lanes have a constant distance between them and the both lane coefficients are almost the same. We check for these conditions after the coefficient is computed and if there is an exception due to the noise then it uses the coefficient values from the previous frame.
- With these lines as input, we can use ***superimpose*** these lines on the lane lines in the image sequence.

1.4 Turn Detection

- Assuming the camera is fixed and center of the car is same as the middle of the left and right lane lines and also since we already have two lane lines, the mean value coefficient of the polynomial is used to draw observation on threshold for the car to turn left or right.
- For efficiency of turn detection, we averaged the ***polynomial coefficients*** over last 7 iterations which helps us to smoothen the turn detection and in turn more robust.

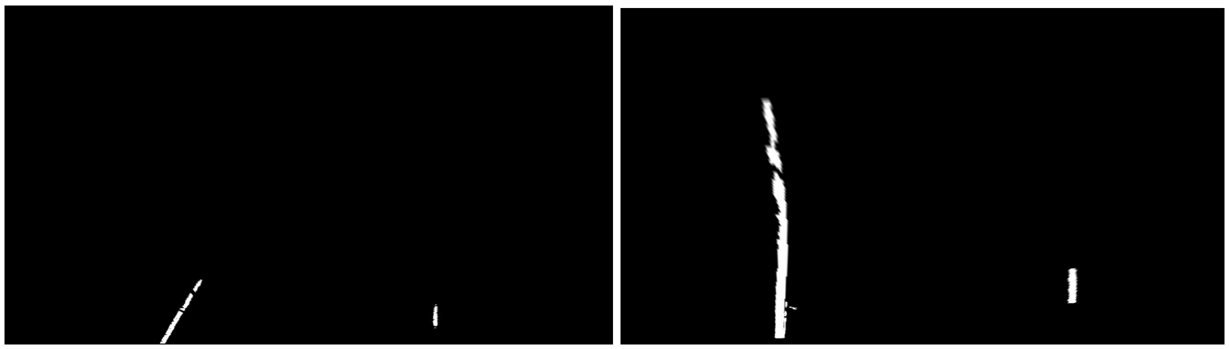


Turn Predictions

2. Report

2.1 Homography

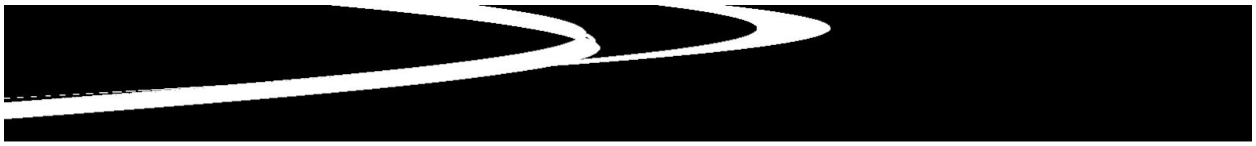
Homography is used to change the lane frame from world coordinates to camera frame for further processing. Then the inverse Homography is used to warp the lane polygon computed in the camera frame to the world frame. The important observation here is that homography is computed once in the start of the processing, with just four arbitrary points, two on each lane. This works because the camera orientation is fixed for lane detection and we don't have to re-compute homography.



Segmented Homography

2.2 Hough Lines

Hough lines use the existing corner points to generate a heat map and the the parameter with highest votes are chosen to draw lines. These parameters are used to construct long parallel lines. But this approach fails when there are turns, since we used a model corresponding to straight lines. So, we didn't go ahead with this method after initial trials. The voting method used is shown below. Points with maximum intersections was used.



Voting Scheme

2.3 Robustness/Generalization

- The above implemented pipeline holds good for the first project video and is little shaky in the challenge video because the lane lines are much lighter.
- The process of selection of lane line pixels can be improved so that even in lighter or irregular lane lines our method is efficient.

2.4 Observations

- After finding the polynomial, the inverse homography to embed the polygon the road had an offset. We initially added an offset to fix move the boundaries of the lane polygon. But that did not yield a stable output like the one before homography transform.
- We assumed it was a mistake from homography computation. But it was because we were superimposing the polygon on the the distorted image rather than undistorted image. After changing the image the output seemed to be flawless.
- In using hough lines to find the lane points and fit the line, we found out that to extract curves using Hough transform is a little harder because higher dimensional hough transforms are resource consuming.

- To precisely detect the lane lines in yellow and to overcome shadows we used HLS color thresholding.
- Use an exponential moving average of line coefficients from previous frames to use it in scene where pixel detection fails.
- The region of interest needs to be flexible in the sense the horizon under consideration needs to dynamic in order to be robust even in case of steep uphill roads, tight turns and so on.
- We observed color thresholding to have larger impact on thresholding compared to gradient and magnitude thresholding.
- The color segmentation process works solid in day light, however, poses serious challenges in limited visibility conditions like driving at night, intense fog, etc.
- The averaging of polynomial coefficients over last couple of iterations is not the ideal

How to run code

1. Unzip the folder which has the code and input sequences
2. Each of the following code parts asks you to select the input sequence you would like to run the code with.
3. Run `python ./Lane_Detection.py` to start Lane detection.

Project Team

1. Nantha Kumar Sunder (UID: 116104777)
2. Nithish Kumar (UID: 116316958)
3. Rama Prashanth (UID: 116428941)

References

1. Computer Vision, A Modern Approach, Forsyth and Ponce (http://cmuems.com/excap/readings/forsyth-ponce-computer-vision-a-modern-approach.pdf)
2. https://www.intmath.com/applications-differentiation/8-radius-curvature.php
3. https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/
4. https://medium.com/@cacheop/advanced-lane-detection-for-autonomous-cars-bff5390a360f
5. https://stackoverflow.com/questions/11270250/what-does-the-python-interface-to-opencv2-fillpoly-want-as-input