Akash Atharv, Rishabh Choudhary, Sanchit Gupta

Perception Project-2
Lane Detection to mimic Lane Departure Warning System

Problem Statement

To detect the lane lines in a video and show the radius of curvature for the lanes for the application of the Lane Departure Warning System. The steps and the pipeline are divided into sections below:

Section – Distortion Correction

After reading a frame from the video, we used the given correction matrices to remove the distortion from the image. We used the undistort function in opencv to remove distortion.

Section- Separating the red and yellow lane lines

As we had to detect the lane lines which are always yellow and white, we convert the color space of the image so that yellow and white can be clearly distinguished. We performed an HSL conversion of the image and then created 2 masks, one for yellow and one for white. We did not hard code the value for the mask but used the function, “in range” to define a range so that any intensity yellow, white can be detected. After creating these 2 masks and separately “bit wise and” with the actual frame image and then “bit wise or” with the each other to get the image similar to the one shown below.

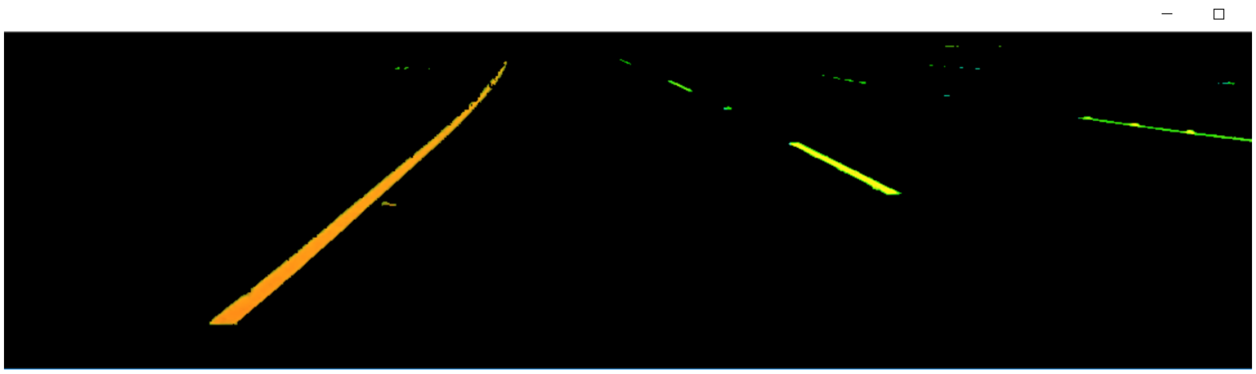


Fig: After putting the masks on the cropped image and separating the lane lines

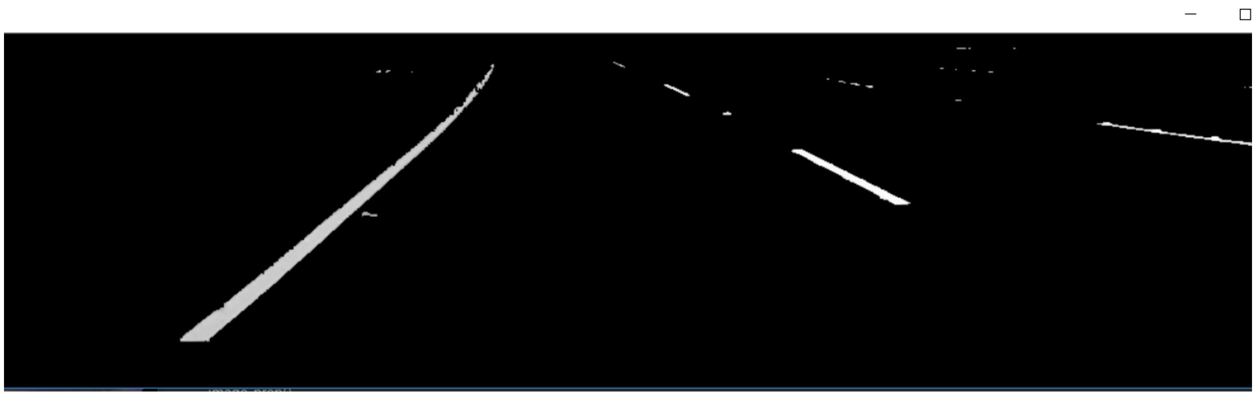


Fig: Showing the gray image of the HSL image

Noise from the image is removed using a bilateral filtering and then canny edge detection is done so that proper lane detection can be done.

Fig: result of Canny

Section- Getting the Bird's Eye view using Homography

As we were experimenting, trying to fit Hough Lines while looking from the perspective of the camera on the lanes, we were not getting good results. We then decided to transform each frame using the same transformation matrix and getting the birds eye view. In this view it will be easier to detect curves and see the lanes more clearly.

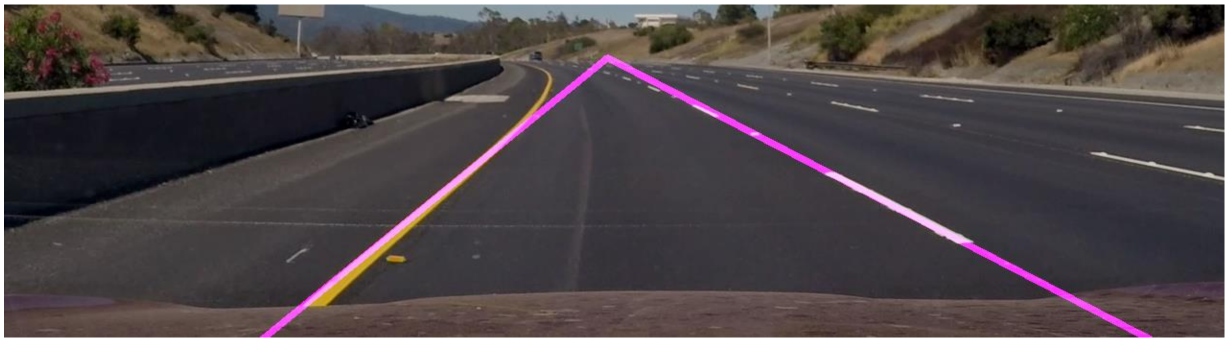


Fig: Image we got from using hough lines projecting on the image

To transform it into the bird's eye view we manually captured a frame from the video where the road is completely straight and took 4 points, 2 on each side of the road. As we know that these points correspond to a parallel road, we took 4 points in the world coordinates which form 2 parallel lines.

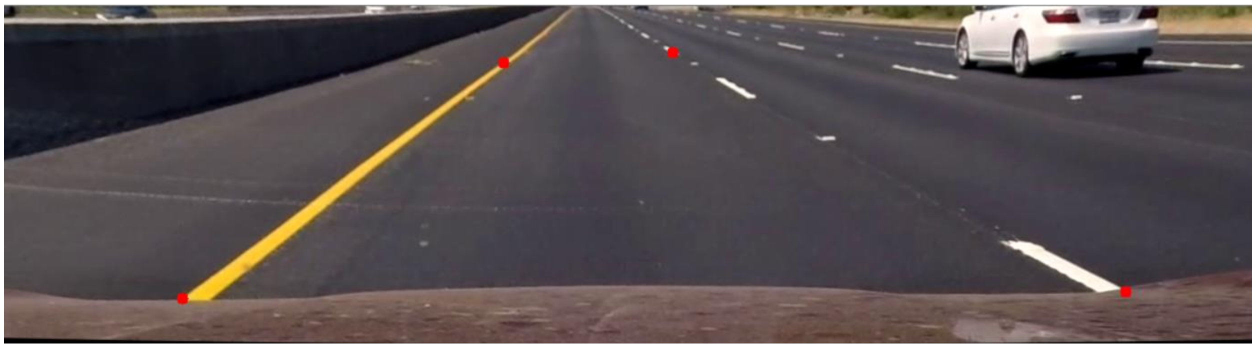
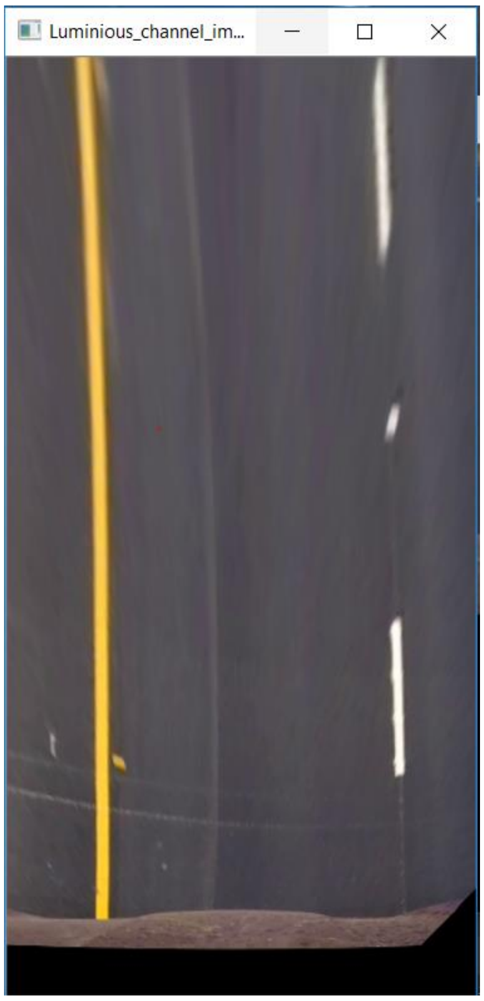
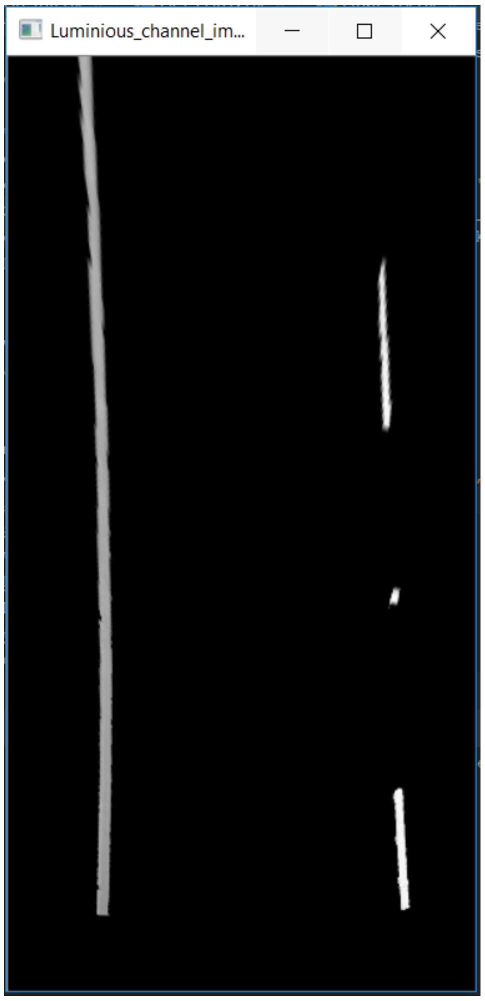


Fig: The red points shows the points taken for homography

Using the source points in the image and the world points as destination points, we used the "Get Persepctive Transform" function to find the homography matrix , this matrix was then used with the wrap perspective function to get the new image from the bird's eye view.





The luminous channel from the HSL is shown in the bird's eye view, here we can see that only the lane lines are seen and can be easily worked on

Fig: Bird's eye view

Section: Finding Lanes in the image

After getting the bird's eye view we have to find the pixel values where the lanes are actually there and then draw a polygon over it.

To find the pixel values, we are using the concept of windowing and the windowing area is defined on the bird's eye image. We divide the image into 2 parts using a vertical axis and then find histogram peaks in these 2 areas, this gives us the location where maximum intensity occurs along the horizontal axis.

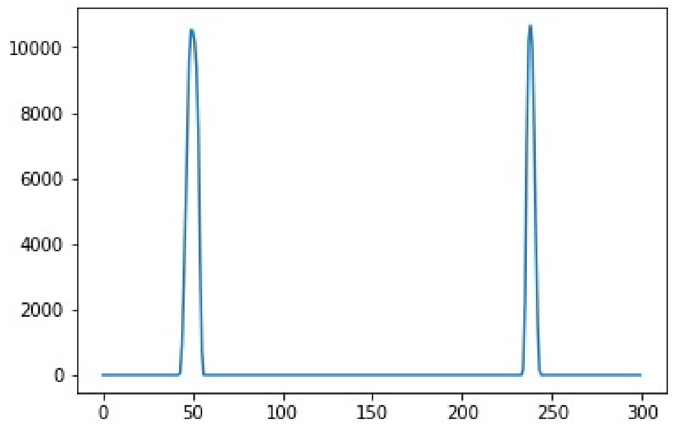


Fig: Histogram representation of a frame

Using these values, we define the window size by choosing an “offset value = 110” , we also determine the number of windows we want in any frame, which we have taken as 10.

Next, we create an empty list to store the pixel location of the right and the left lane. We use the function called nonzero(), on our bird’s eye image, this function returns the indexes of all the non-zero elements in the image. In a for loop we put boundary conditions for each window and append the list for the left and the right lane pixel value. The position of each window is changed according to the mean of the concentration of the pixels, if it is more than the min_pixel value defined, then we shift the position of the new window.

After forming the 2 lists for right and the left lane for both the location of x and y axis, we further subdivide them into right_x, right_y, left_x, left_y and store them in separate lists.

Our next aim is to make an empty black image and then plot these pixel values we have found (in different color) in the previous step on it to see if we are getting the proper results, to do this we make an empty image and plot the left and the right pixel location on it, the result of the same are shown below:

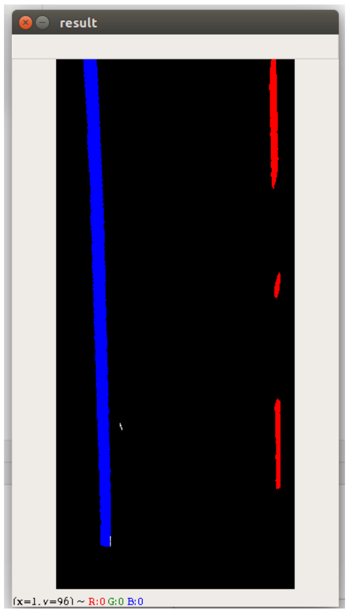


Fig: The pixel values plotted where the lane is detected, these pixel values are stores in a list

Section: Filling the area detected

After finding that the pixel values, do correspond for the lanes we need to find the polynomial which fits the curved lanes, do to this we use the "np.polyfit" function with given input as points and the order of the polynomial as 2. This function returns the value of A, B and C of the second order polynomial, which can be plotted on the image.

The line which is plotted for the right lane after using the linspace and using the equation is shown below.

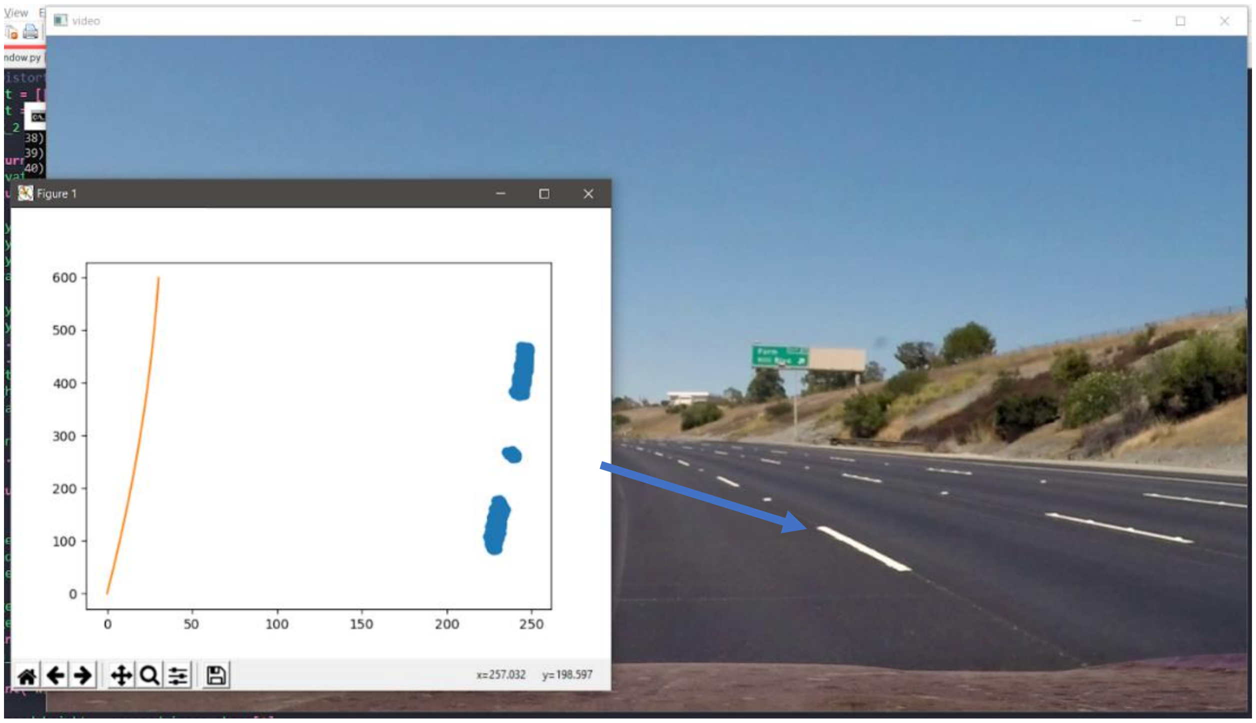


Fig: Polyfit line shown for the right lane, which is a result of the plt.plot

After this the points of each of the lines (left and right) are stored in an array called pts, which is then used in the fillpoly function to fill the area in the center with green color.

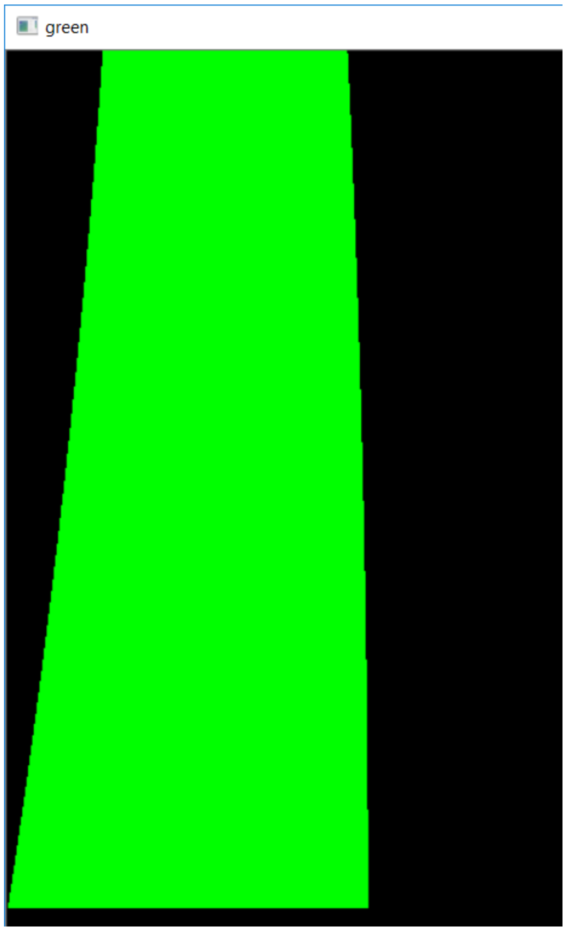


Fig: The predicted lane covered in green shown in the bird's eye view

The next after this is to send this back to the camera view from the bird's eye view to project it on the frame. This is done by again wrap perspective and this time using the inverse of the H matrix we found earlier to find the bird's eye view, the result of the same is shown below:

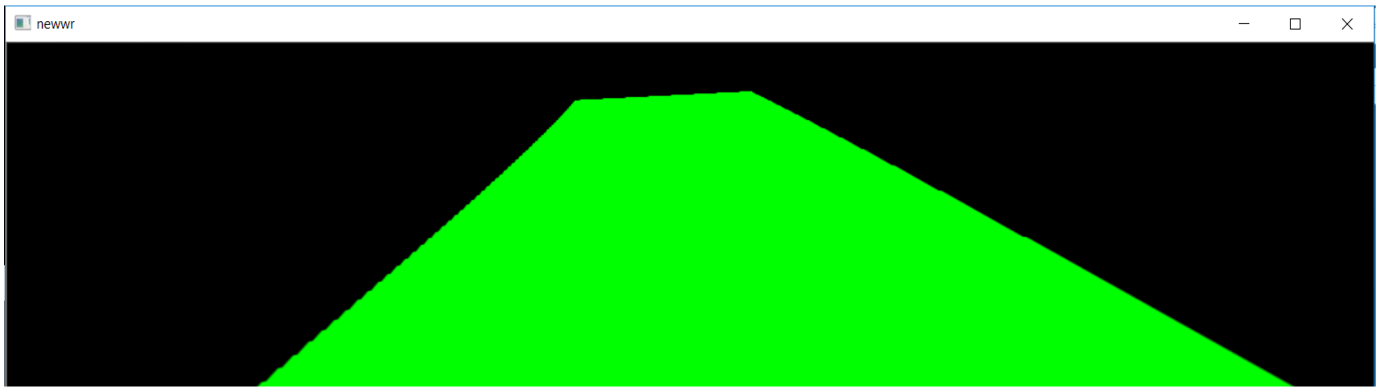


Fig: Detected lane in the actual frame

Section: Final projection on the road

The next part of the project is to project the lane on the frame, this is done by adding this image and the frame together, the adding is done as weighted to show the lane lines and the predicted area together. The same is shown in the image below:

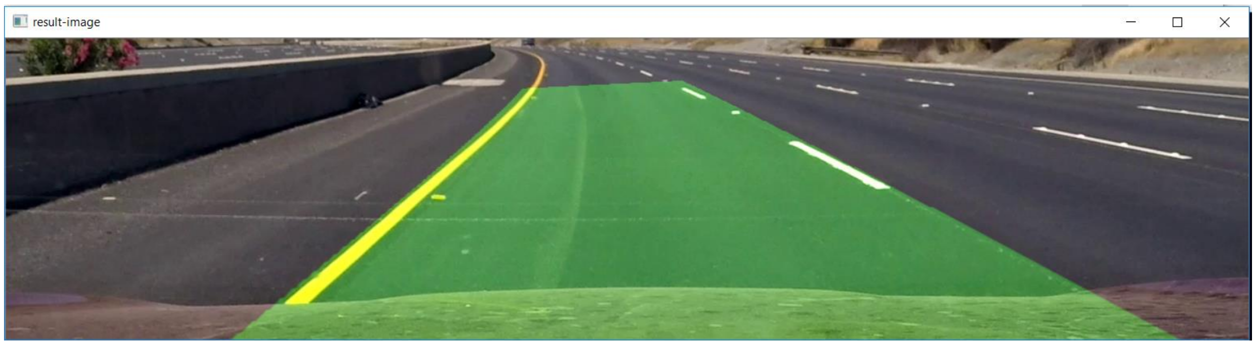


Fig: the predicted area drawn on the frame

Section: Radius of curvature

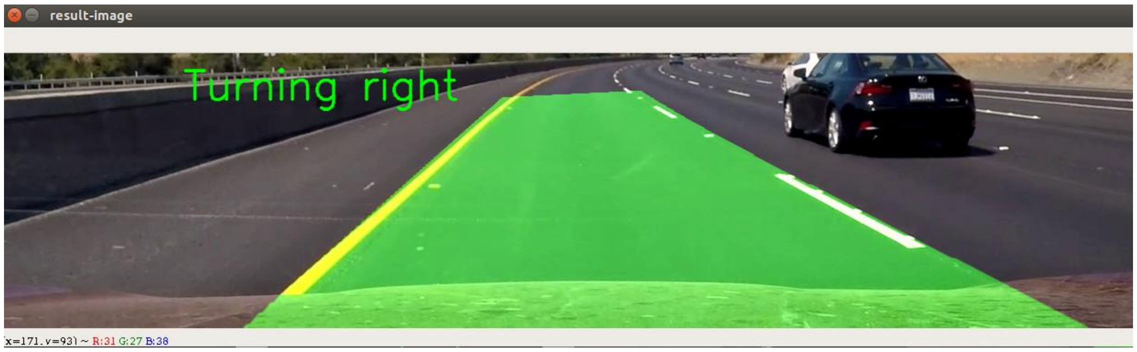
The radius of curvature is calculated using the derivates of the polynomial equation estimated using the polyfit parameters. Both, the first and the second derivatives of the curve are used for calculating the radius of curvature for both the right and the left lanes separately.

The radius of curvature is given by the equation :

$$R_{curve} = \frac{\left[1+\left(\frac{dx}{dy}\right)^2\right]^{3/2}}{\left|\frac{d^2x}{dy^2}\right|}$$

Section: Turn Prediction

The last step in the pipeline is to predict the lane heading for the vehicle. This is basically calculated using the position of the vehicle with respect to lanes at any given point. We first calculate the central point of the lane by using the position of both the left lane and the right lane. Then, we use the position of the center of the vehicle i.e. the center of the image to predict the lane turn. We calculate the difference between the lane center and the image center and if it is positive, then we are able to correctly predict the right turn of the lane and if it is negative, we can identify, the left turn.



Problems faced:

a) **Color selection range:**
To separate out the yellow and the white color using a mask, we had to select the ranges we consider to create the mask, this task specially took us time because we tried different combination at different instances of the video to select the combination which gave us good results, before this the selection of color space was also a task which took us some time after we figured out HLS will be good to perform this.

b) **Homography matrix point selection:**
To convert the lane image from looking at it head on to converting it to look from bird eye view was a challenging part for us because we were not able to find the exact points on the frame image to do the perspective transform from the initial frames in the video. To do this task we saved all the images of the frame and then selected the one where the road was straight, and the points could be easily chosen, we then plotted small red circles on the frame to carefully select these points. And then performed the homography.

c) **Challenge Video Problem:**
When the same code was run on the challenge video the code was crashing because there were times when no lane was detected, we fixed by putting a check on the null condition and continuing the loop when none condition occurs.

Still the lane detection is not as expected because of the presence of white markers in the middle on the lane which creates problems as these are also detected and are stored in the list of the white pixels, this problem can be resolved by more carefully selecting the area of the lanes and using better filtering to remove these shapes detected in the middle of the road.

Section: Understanding of Hough Lines:

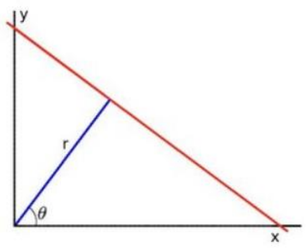
Hough Lines transform is used to detect and generate straight lines in the image. To first apply the Hough transform an edge detection must be done on the image. To test our code for the hough lines, we did canny edge detection.

The equation of a line in polar coordinates requires an estimation of 2 parameters, which are:

Parameters: (r,θ)

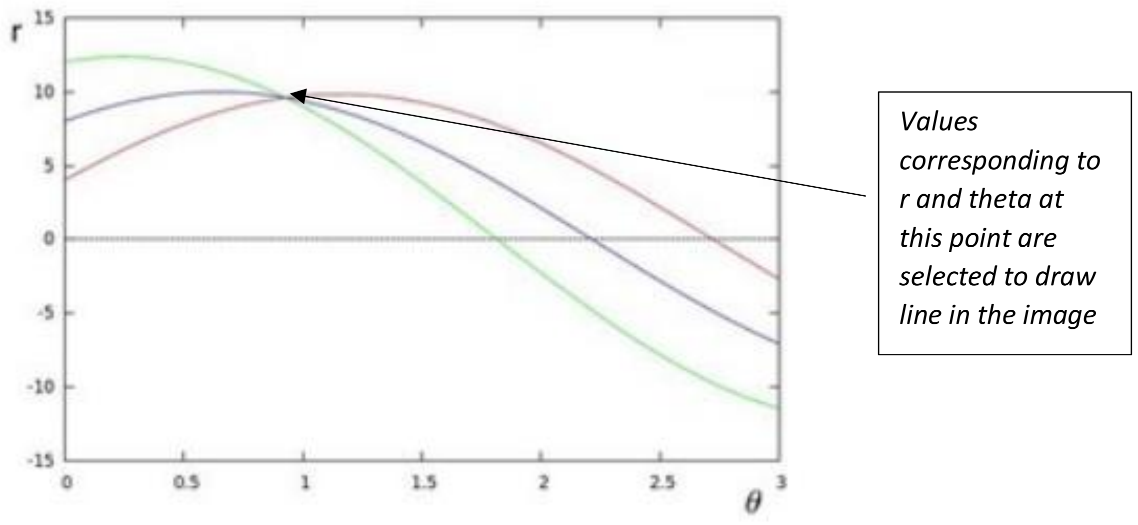
r – Perpendicular distance

θ – Angle made by the line



For each point detected in the image, a family of lines can be plotted that goes through that point. This means that for all values of **(r,θ)** there will be a line passing through the point.

This family of lines is plotted for all the points in the image and a graph is plotted between r and theta. The point where most of the curves in the graph intersect the corresponding r and theta value is selected to plot a line in the image. The same is shown in the image below:



Hough is a voting-based algorithm

References:

- https://medium.com/@tjosh.owoyemi/finding-lane-lines-with-colour-thresholds-beb542e0d839
- https://www.youtube.com/watch?v=VyLihutdsPk&t=1392s
- https://medium.com/@mrhwick/simple-lane-detection-with-opencv-bfeb6ae54ec0
- https://medium.com/@tempflip/lane-detection-with-numpy-2-hough-transform-f4c017f4da39
- https://docs.opencv.org/3.1.0/d1/db7/tutorial_py_histogram_begins.html