Lane and curvature detection

Student: Pradyumna Ingle

Index:

Project Video:
https://youtu.be/bAJ9DVXRrqo
Binary: https://www.youtube.com/watch?v=wjnh1GKhcNc

Challenge Video:
https://youtu.be/y2FGTVUcdPs

1. Introduction:
In this project I have find the lanes of the given video data sets of highways. Those images were converted carefully to binary, and was used to determine the curves of the lane. Algorithm is specifically created considering the dataset provided, and it is not tested against the road scenarios under different weather, daylight and nightlight conditions. The algorithm gives basic idea about the components goes into the pipeline for detecting the lanes, where each component further needs to be tested against different conditions for robustness.

2. Camera Correction:
File: Camera_Calib.ipynb

From the given chessboard images, I manually set the horizontal and vertical corners of the image.

OpenCV's **findChessboardCorners()** function was used to find the internal corners of the chessboard. **drawChessboardCorners()** was used to test the detected corners of the chess board.

Image1: Chessboard corners.

With the obtained corner data, OpenCV's **calibrateCamera()** function was used to calibrate images taken from the particular camera.

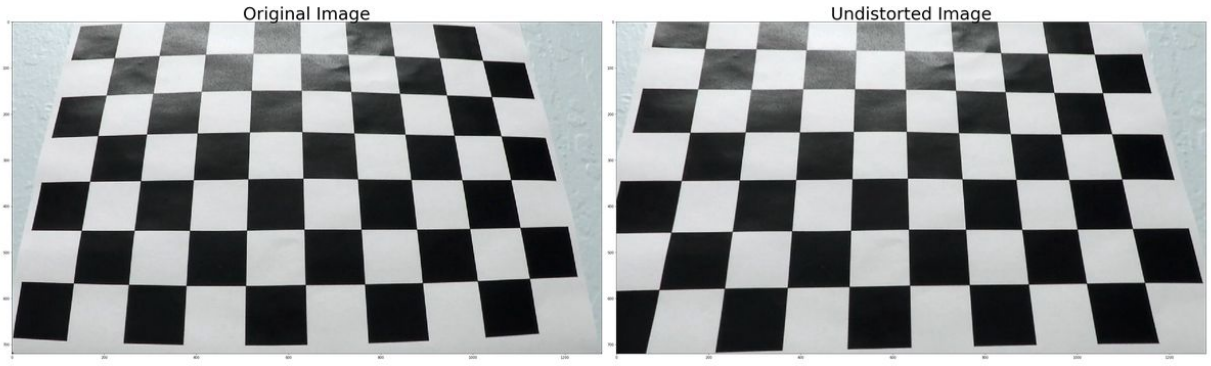

Image 2: Camera undistorted chess board images

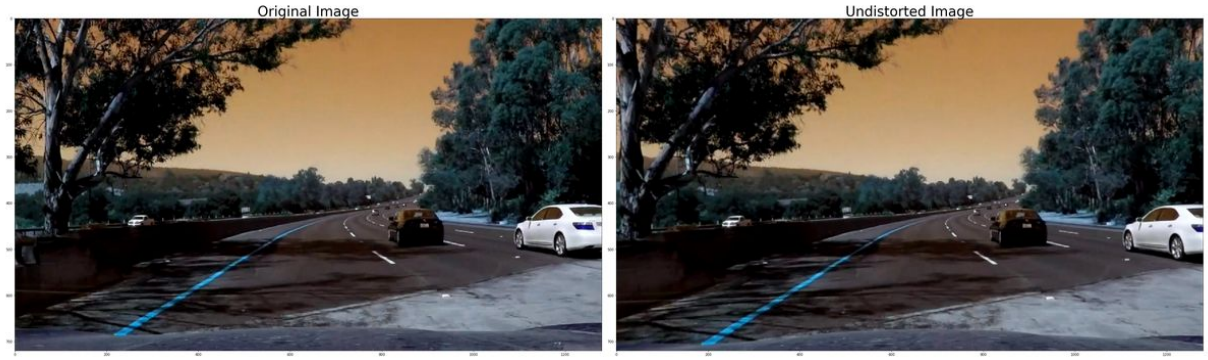

Image 3: Camera undistorted highway images

3. Image wrapping/creating Bird eye view of the lane:

OpenCV's **getPerspectiveTransform()** and **warpPerspective()** functions were used to create the wrapping or the bird eye of the tracks. Here the points from the straight lanes were chosen for 4 corners, which meant to be viewed straight or in bird eye perspective. The output was as following

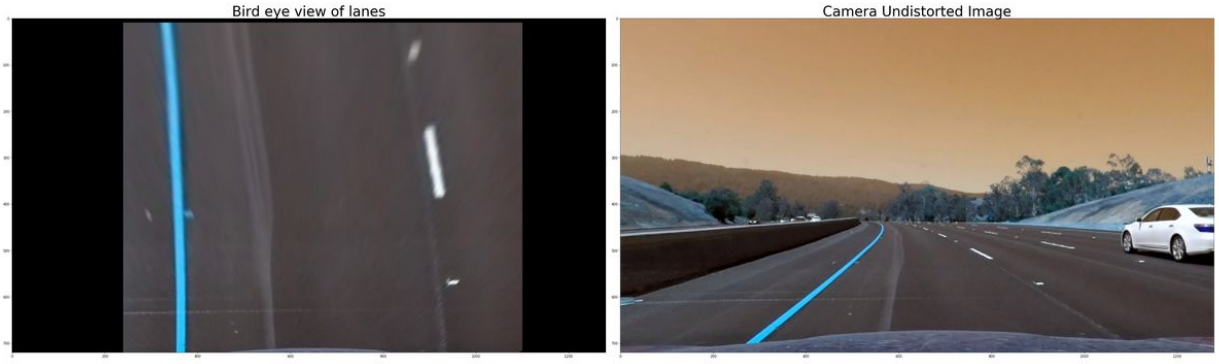

Image 4: Bird eye view of the track

4. Binary images:

One of the main challenge of finding the lanes, is to produce the binary image of the highway track. Given the bright sunlight conditions, cloudy daylight, shadows, asphalt, tyre marks and the old but confusing lane marks makes it extremely difficult to produce the binary where the undesidered marks on the highway do not get noted in the binary.

Lane lines are usually white and yellow. Based upon that, I divided the lane finding task in two groups. One for finding white lane and other for finding Yellow lane.

Track was converted into YUV, RGB, HSV, LAB, YCrCb color spaces to identify which spaces could help finding the Yellow and white lines the best.

Conversions looked something like this:

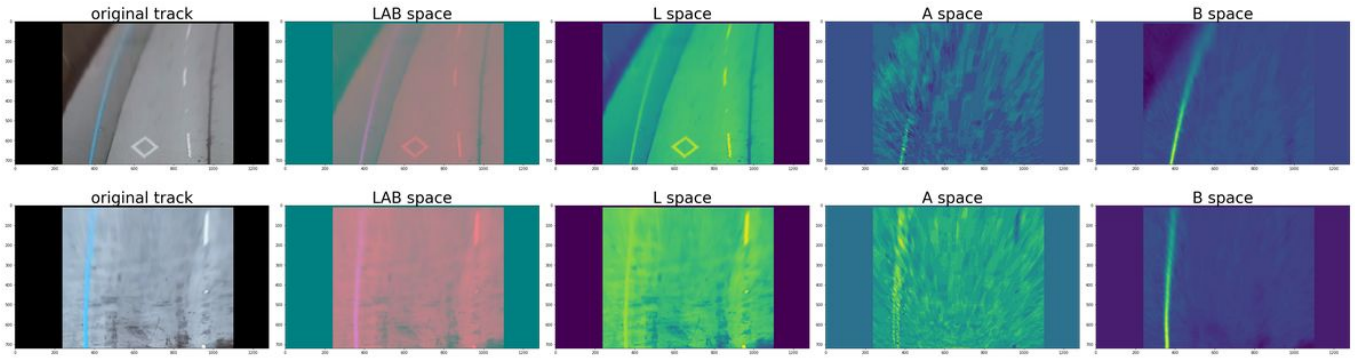

Image 5: Track in LAB space. Yellow track was clearly noticeable in A and B space. L space also showed good sensitivity to white color.

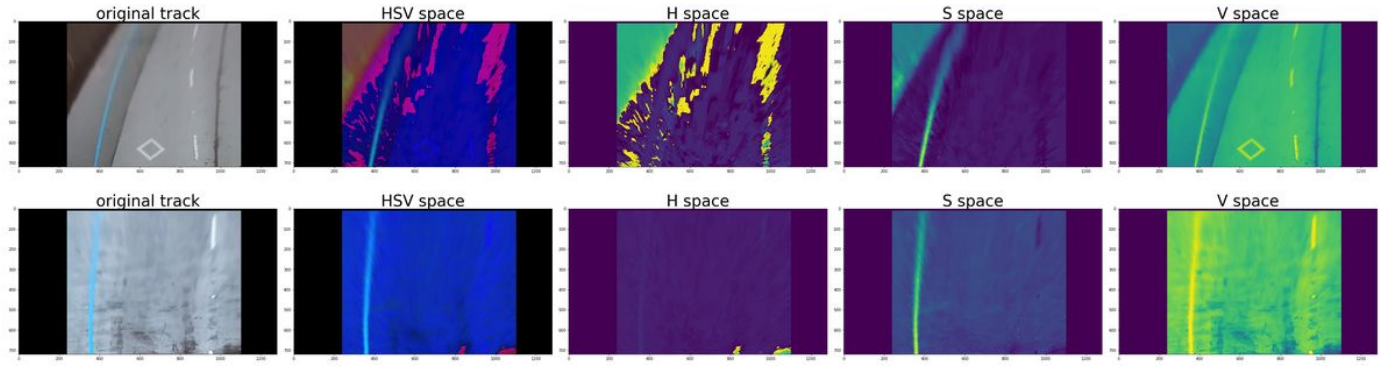

Image 6: Track in HSV space. Yellow track was clearly noticeable in S and V space

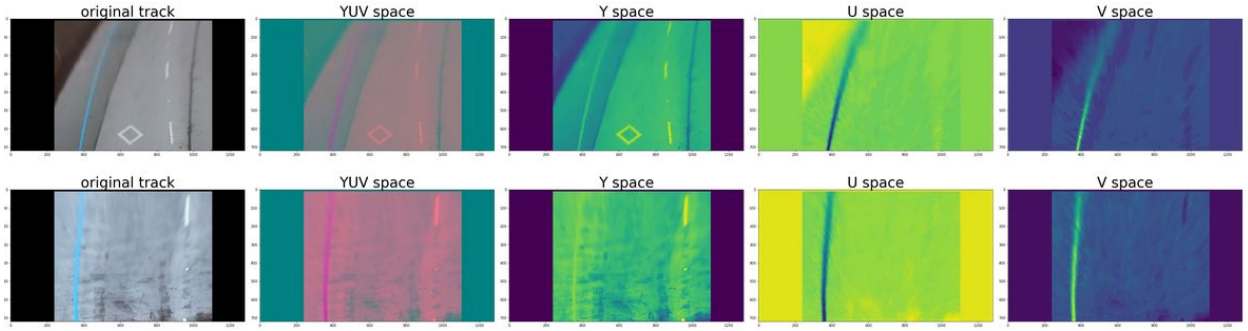

Image 7: Track in YUV space. Yellow track was clearly noticeable in U and V space, while Y space showed good sensitivity to White lane

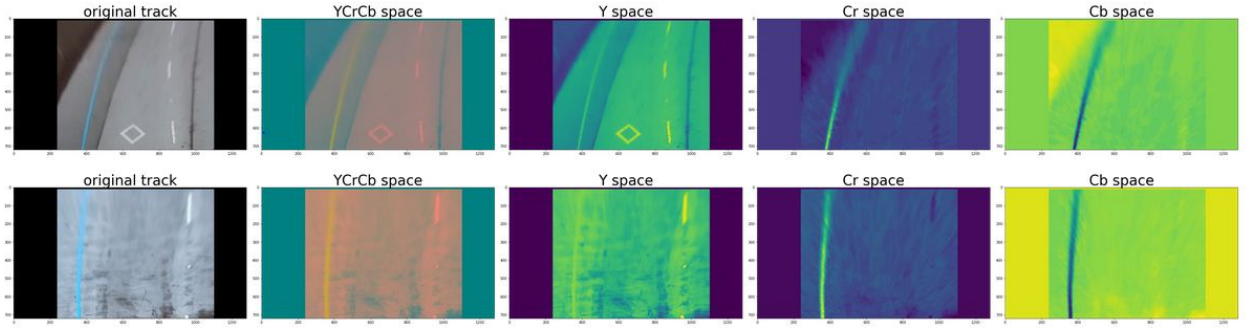

Image 8: Track in YCrCb space. Yellow track was clearly noticeable in Cr and Cb space, while Y space showed good sensitivity to White lane

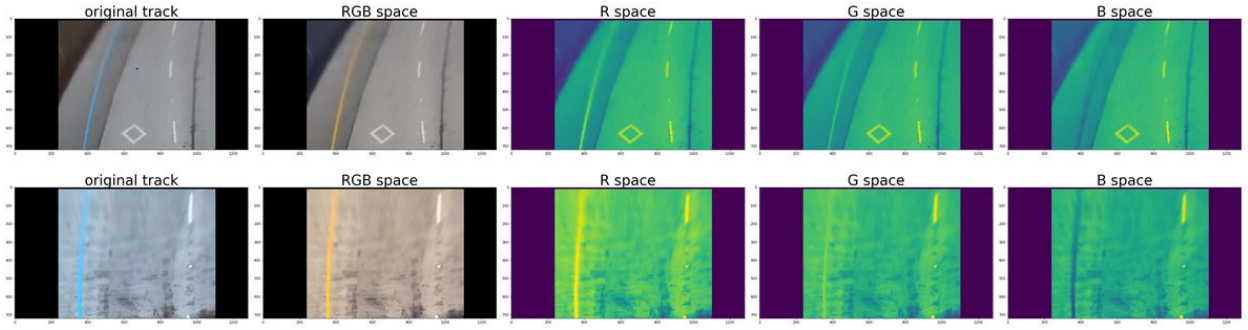

Image 9: Track in RGB space. Yellow track was clearly noticeable in R space, while overall all spaces showed good sensitivity for white lane/color.

I tried the adaptive threshold, but if didn't look much useful.

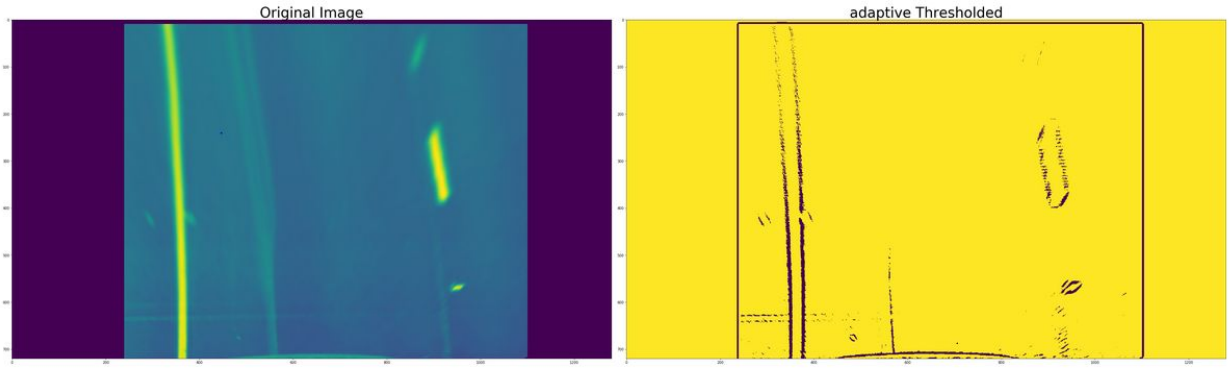

Image 10: adaptive threshold of the image using OpenCV's *adaptiveThreshold()*

Tried locally normalizing the image for 'y' channel. From the normalized image, I thresholded the 'u' channel. It worked fine for the yellow color for me. This technique didn't help much for the white color, specially for the Harder challenge video frames.

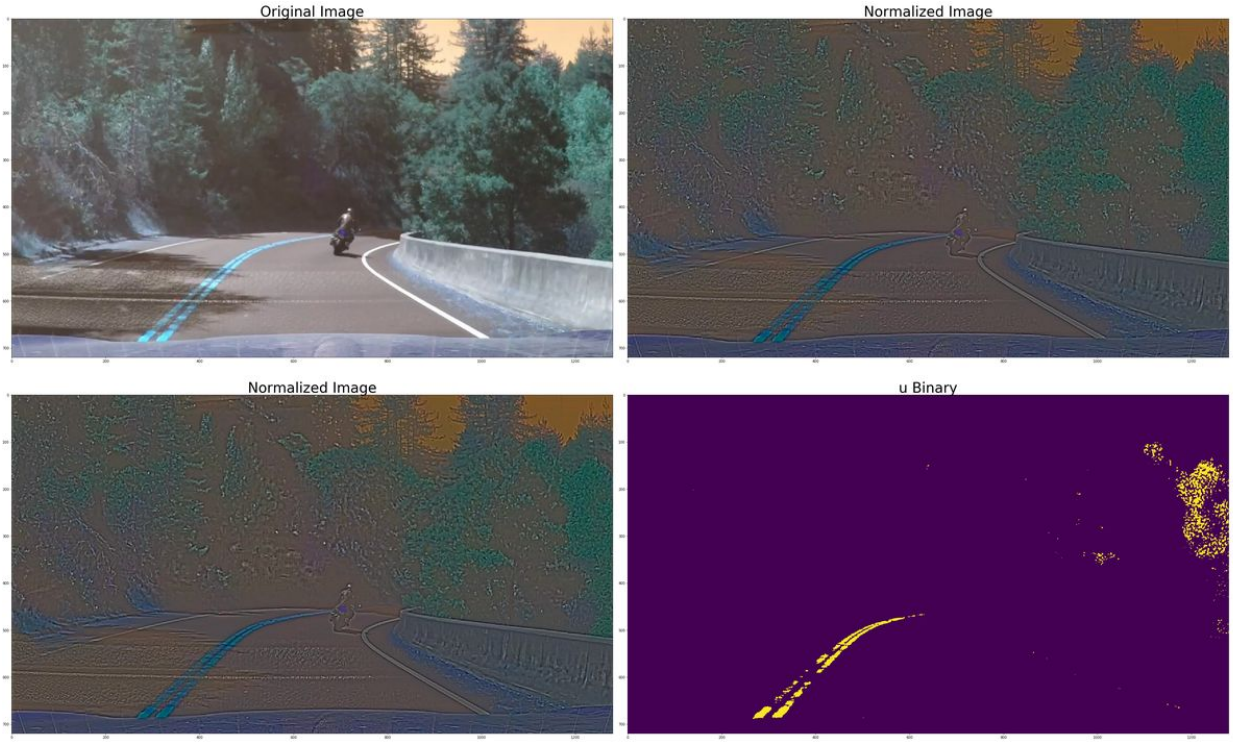

Image 11: locally normalizing the image (y channel) and thresholding the U channel.

Used lower limit of [0,0,220] and upper limit of [255,255,255] in HSV space and it showed considerably good sensitivity to white color:

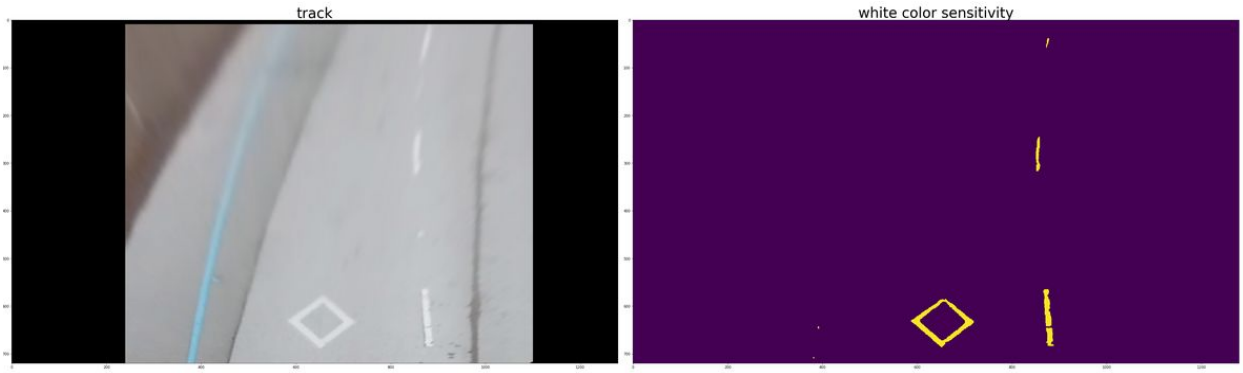

Image 12: HSV color space to find white lines and signs

Used A colorspace from LAB colorspace to detect Y lines.

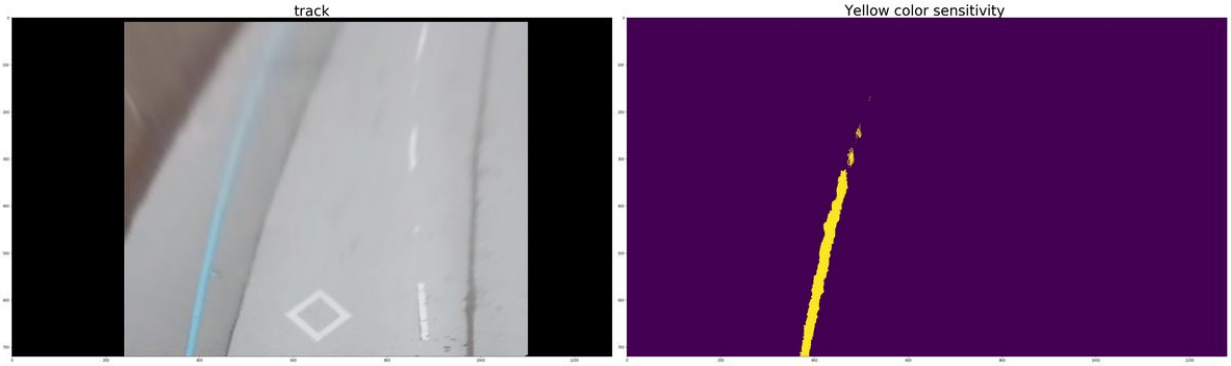

Image 13: LAB color space to find yellow lane

Used threshold of [230, 200, 200] to [255, 255, 255] for RGB space to more solidly find the White lines.

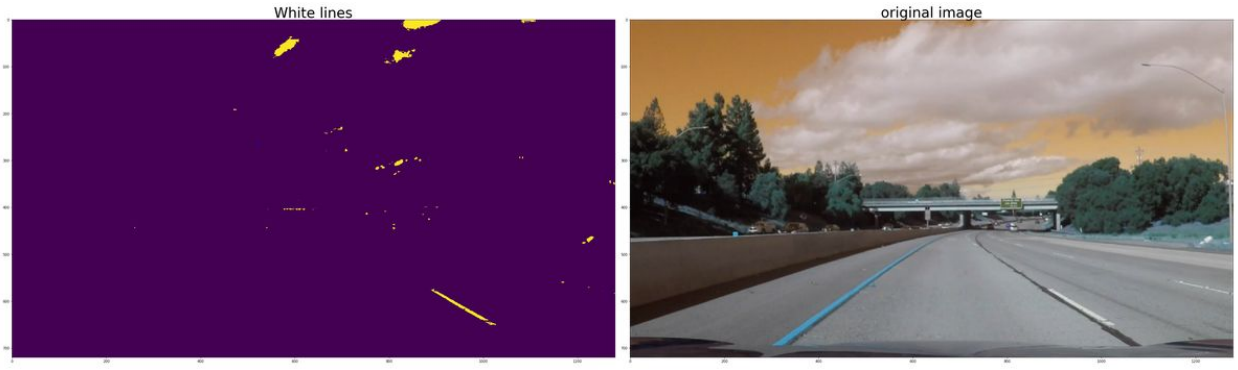

Image 14: RGB space to find white lines.

Combined above (image 12 - 14), the final binary was produced:

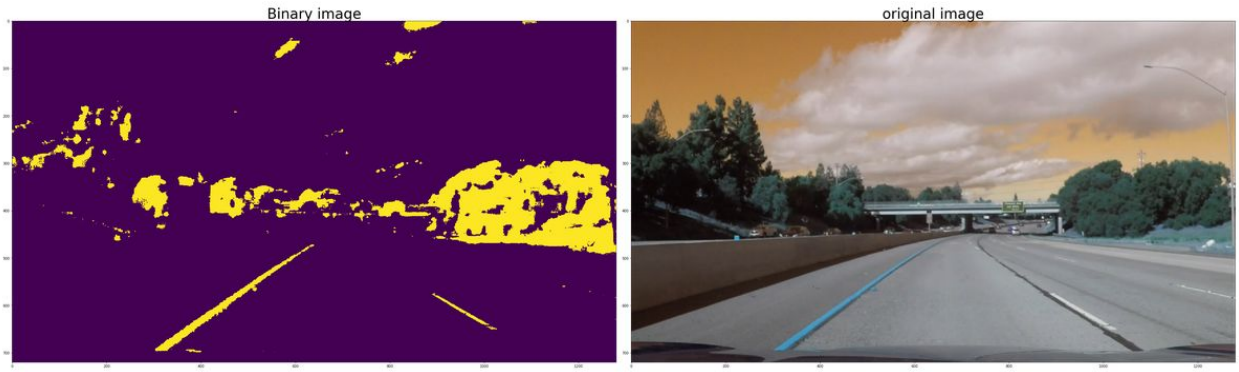

Image 15: Binary of the track.

Video Link for binary video for project video file:
Https://www.youtube.com/watch?v=wjnh1GKhcNc

I did not use sobel x or y, for producing the binary images. Using the sobel added unwanted lines on the binay images, starting from tire marks to the sunlight/shadows. I obtained binary despite using the sobel.

5. Finding the lane lines:

I have created a class names ‘lines’ for the finding the lanes from the binary images. Code is well commended.

As per the course work, I used Sliding window technique to find the lanes for very first image. Once the lanes were found, I fit left and right curves around that. Those fits were used to detect the lane pixels in subsequent frames. The fit was used as central point, and 50 to 100 pixels in left and right of the fits were scanned for finding lane pixels.

For a particular threshold of detected pixels, we update the pixels for creating a fit for the current frame. If the detected pixels were less than threshold, we continued using the fit from the previous frame.

Now, if the pixels detected were less than threshold, ideally I should window search the subsequent frames for lanes, using the sliding windows. I tried that, but my code worked better without doing that. I continue using the last best detected fits, which worked fine, since the highway does not have curves line in ‘hard challenge video’. I am very sure this is not appropriate and one of the reason my code doesn’t appear robust for Harder challenge video.

Besides that, all the curve fits from the past frames are given similar weightage. I believe as per Udacity's suggestion, I should have weighted mean.Also, I have used a fixed sized numpy array to store the curve fits for previous frames, over a list/queue. I intended to make my code run a bit faster. Overall it still shows the frame processing speed to be around 1.3 sec/frame. And didn't help much.

I have a buffer to store left and right line curve fits for past 10 frames. For each frame, I take average of those 10 fits, and search pixels for lanes in the current frames.

Although the code works in finding the lanes in project and challenge video, I am not confident in it. Basically given different scenarios, and even in the harder challenge video, it fails. I tried to work on it, but leaving it in between to try it after the course gets over.
As an attempt to solve all the above issues, i tried considering my left line detection is a confident always. Based upon the left curve, I tried to limit the right curve from wandering. To do that, I tied the right curve detection to left curve, giving it some weightage like this:

```
# If you found > minpix pixels, recenter next window on their mean position
        if len(good_left_inds) > minpix:
                leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
        if len(good_right_inds) > minpix :#and len(good_right_inds) < minpix+30:
                rx = np.int(np.mean(nonzerox[good_right_inds]))
                rightx_current = math.floor(0.75*rx) + math.floor(0.25*(leftx_current+520))
```

Besides, if I do not find curve, I did fresh sliding window search for next frame. Margin was reduced to around 50 pixels to reduce the accomodation of noise.
Wrap-unwrapping in perspective transform was reduce to give a short length visibility of the road. It needs some more sanity check like parallel lines and more.

All my attempts (which works very well for project video) are in the file'experiment_pipeline.ipynb).

6. Finding the curvature of the lane:
For a curve with equation Ax**2 + Bx + C = 0, the radius was found using the equation we derived from the classwork :

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

To find the pixels to meters conversion, I used the average observed lane width. Since the lane width in the US is 3.7 meters, i have the ‘x-axis’ conversions as 3.7 meters/650 pixels, and for y axis conversions, considering I have a visibility of next 15 meters of detected road, it was 15 meters/720 pixels.

These conversions were used to fit the second order polynomial, for the above equation.

Average of Radiuses for both the lanes, were considered to the be lane curve.

Considering the camera is at the center of the car, perfectly focusing front, we assume center of the car is the image center.
We compared that with the center of the lane, to denote deviation of the car from the lane.

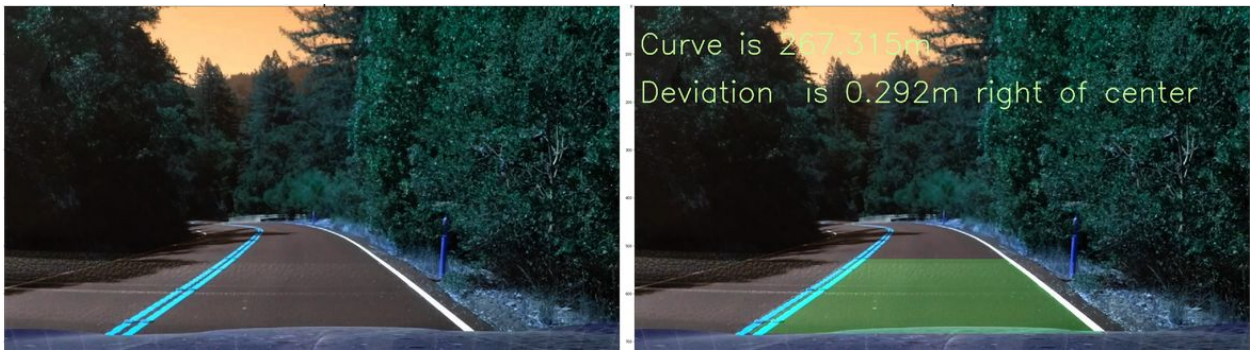

Image 16: Track before and after the lane and curve detection

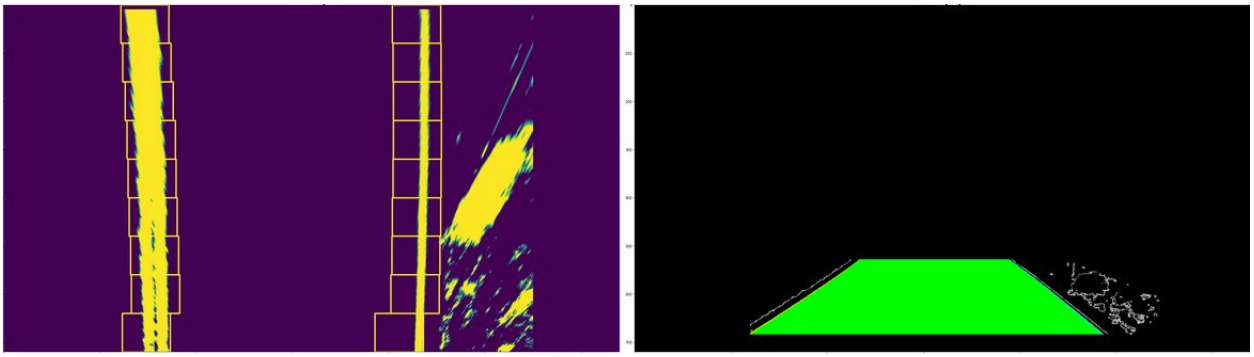

Image 17: Track with windows drawn, and detecting the tracks. On the right, the tracks being changed back to the perspective for the original image.

Project Video Links:
https://youtu.be/bAJ9DVXRrqo
Binary: https://www.youtube.com/watch?v=wjnh1GKhcNc

Challenge Video:
https://youtu.be/y2FGTVUcdPs

Where did it may have gone wrong:

I have tried to explain the same in the section 5. Finding the lane lines.
Although I have tried to rectify this in 'experiment_pipeline.ipynb' file, which is submitted. I do not consider this as a final submission file, as it's not completely functional as expected, and is heavily modified considering the Harder challenge video file.
Some of the things to consider is:

1. When one of the lane is more clearer, use it to find pixels from other undetected or noisey lane.
2. Check if the distance between top and bottom part of the lanes is nearly same (kind of like parallel lanes)