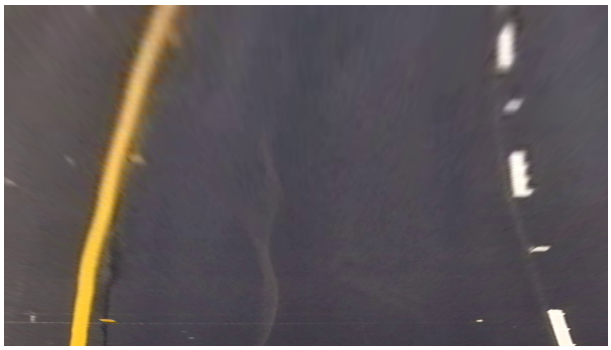
Lane Detection and Turn Prediction

The output video can be found here

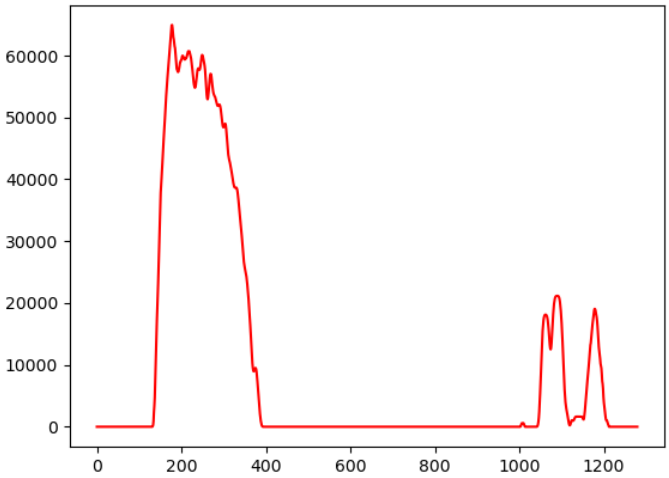
Steps:

1. **ROI and Warping**:
   An ROI is defined which is used for perspective transformation.
   The image is warped using the OpenCV function warp perspective

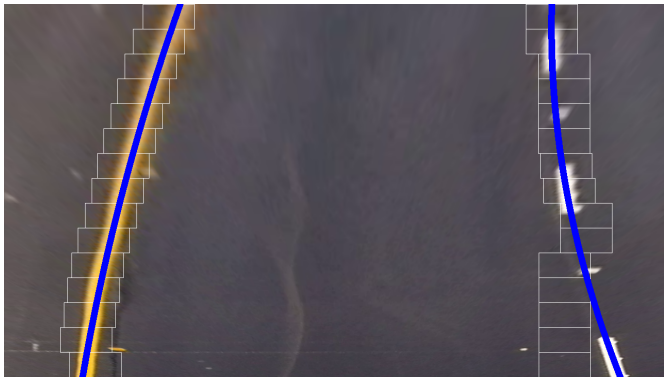
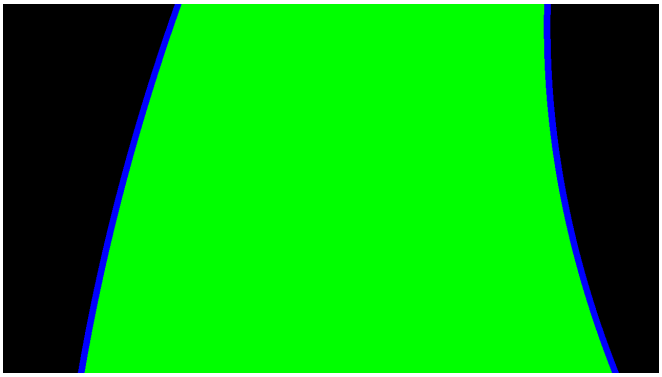


2. **Lane thresholding:**
   This warped image is then used to determine the yellow lane and the white dashed line.
   The yellow lane is detected by using HSV filtering and yellow masking
   The white lane is detected by binary thresholding

3. **Histogram Analysis:**
   The start of both of the lanes are detected using a histogram. A histogram is taken of the image and the two highest peaks in the histogram would be named our two lanes in the image. This will give us the coordinates of the pixels we need to start our sliding windows from.



4. **Sliding window and curve fitting**
   1. Using the pixel locations for the left and right lanes through the histogram. We can use the sliding window technique to scan the lane from the bottom of the image all the way to the top.
   2. Each time we search for active pixels within a sliding window, we add potential lane line pixels to a list.
   3. The mean of the active pixels inside of the window is taken to find the position of the centre of the next window.
   4. After all the last window reaches the top of the image, we used the potential lane line pixel list to find the best fitting polynomial of degree 2 using the function np.polyfit(). This gives us the 3 coefficients of the parabola equation which can be then used to plot the parabola and the lane coverage onto the image.

5. **Inverse warp**
   After getting the best fitting curves and drawing the lane, the image is then warped back onto the original frame for better visualization.
6. **Curvature calculation**
   Since we already have the coefficients(a, b, c) of the parabola, we can easily calculate the radius of curvature by defining pixel to meter ratio in the x and y axis.

$$\text{Radius of Curvature, } R = \frac{(1 + (\frac{dy}{dx})^2)^{3/2}}{|\frac{d^2y}{dx^2}|}$$

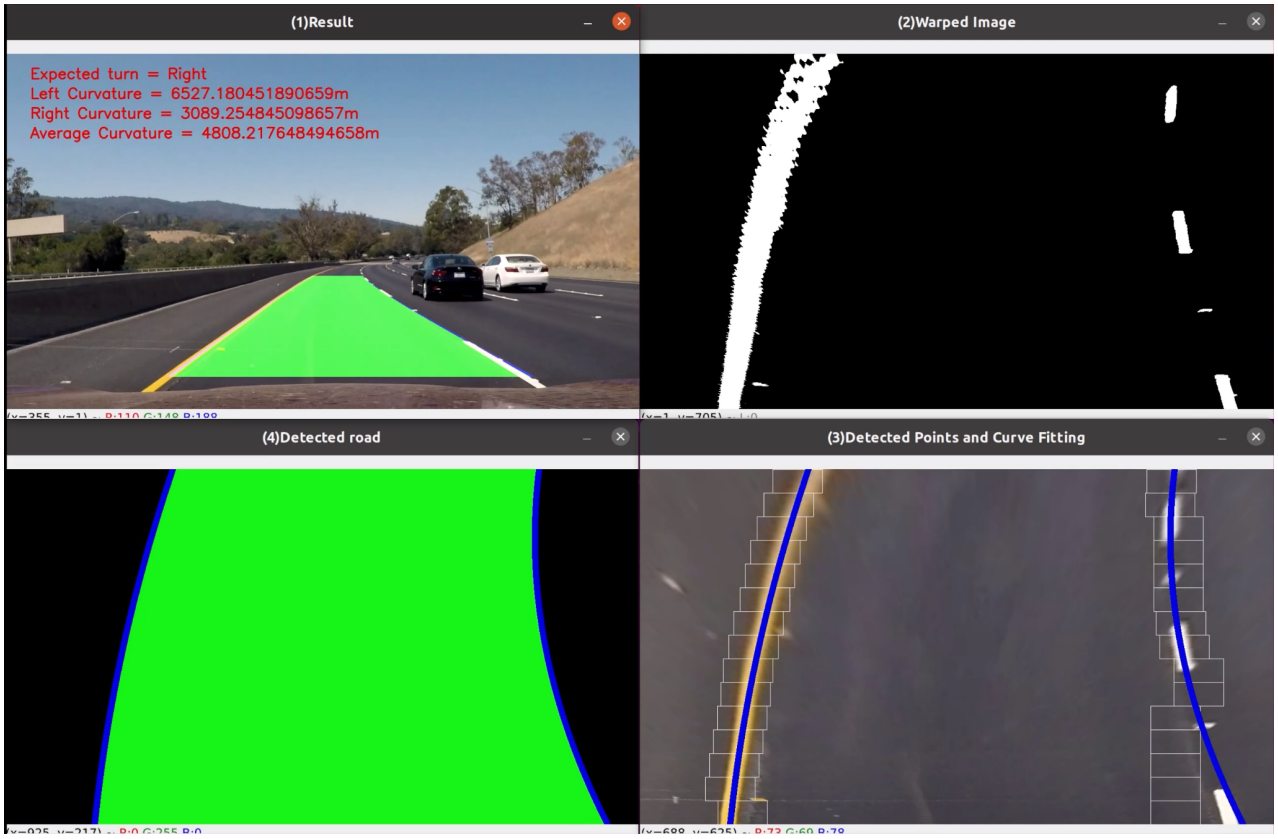
And we know that,

$$y = a * x^2 + b * x + c$$
$$\frac{dy}{dx} = 2ax + b$$
$$\frac{d^2y}{dx^2} = 2a$$

Hence,

$$R = \frac{(1+(2ax+b)^2)^{\frac{3}{2}}}{|2a|}$$

This is how we can calculate the radius of curvature for both the lanes in the image and get it's average.



Problems:

- I tried to use hough lines for this but since the lines are curved, the accuracy was very low and calculating the radius of curvature was impossible.
- Fitting the parabola without the window sliding technique was not accurate at all and would often give obscure parabolas that went out of scope.
- Warping the lanes detected back onto the image directly was not happening. Only the lanes would dewarp but the rest of the image was just filled with zeros. So I had to use a function called cv.addWeighted to add the rest of the pixels from the original image onto the dewarped image to acquire the entire image.
- Took a lot of time to decide the coordinates of the sliding window how the active cells would be recognized.
- In some frames, lanes were not being detected properly, hence I used the previous values ot overcome that issue.

The pipeline is quite generalized and efficient if the position of the camera on the car is known. Due to the sliding window technique, the pipeline becomes quite robust. Even on lanes which are all white or all yellow, the pipeline should work properly.

Homography:

Homography is a method to relate a point in two different images using a pair of points from the same planar surface. Hence it relates to the transformation between two images.
The homography matrix is used as the transformation matrix between two images.

This matrix is created as follows:
1. Get at least four matching pairs between the two images.
2. Use those points to create an A matrix in the following form:

$$\begin{bmatrix} x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

3. Calculate $AT.A$
4. Calculate the eigenvalues and eigenvector of the $AT.A$ matrix and sort the eigenvalues and their corresponding eigenvectors in descending order.
5. The eigenvectors of $AT.A$ give the matrix V. The transpose of V gives us the right singular matrix.
6. Homography matrix was constructed by taking the last column of VT and reshaping the vector in a 3x3 matrix.

Homography can be very useful for changing the perspective of an image or stitching two/multiple images
In this case, I used it for perspective transformation for further image processing to get better data out of it.
We require our lanes in view to be parallel so that we can figure the curvature and in turn predict the turns ahead. We know that in actuality the lanes are parallel but from the perspective of the car, they have a vanishing point and are not parallel to each other in this view and creating a trapezoid kind of figure.

To convert this trapezoidal-like view into a parallel-like view, we need to imagine that the lanes are being viewed from right above.

So we can essentially select the four corners of the trapezoid and apply a warping transformation to achieve the desired view.

Hough Lines:

Hough lines are a method of detecting lines in an image.
Lines can be represented uniquely by two parameters. y=ax+b
But this equation is inadequate when it comes to representing vertical lines, hence hough transform can be used to do this by using the formula r = xcos(theta)+ysin(theta)
The hough space has two parameters r and theta and a line is represented by a single point in the Hough space. Then every single point in the image is transformed to all possible lines that could pass through that point. The point were all there lines intercept is considered to be the true location of the line in the image space.

Now we need to extract the lines that represent the lanes so we will set the resolution of r and theta, the minimum number of intersections to detect a line, minimum number of points that can form a line and the maximum gap between two points to be considered in the same line.