Advance Lane Finding Project.

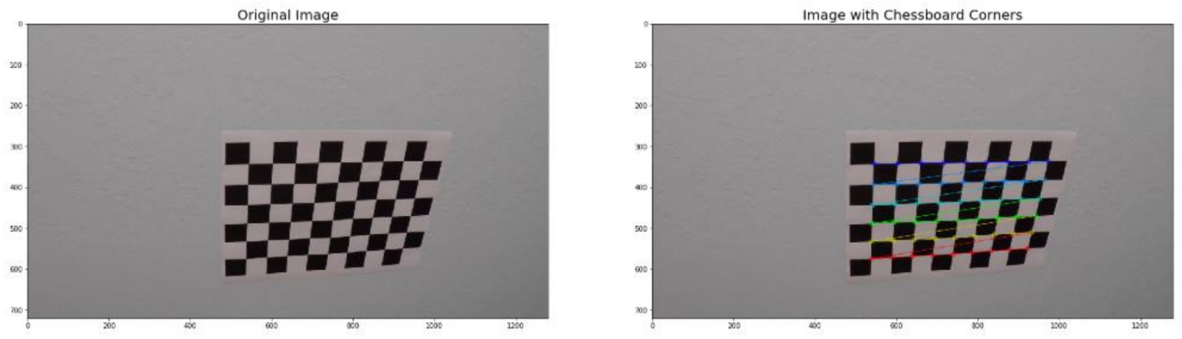
The goals / steps of this project are the following:

- Implementation of camera calibration using chessboard images.
- Distortion coefficients on given set of chessboard images.
- Since we have all the distortion coefficients calculated above, create distortion correction function for raw images.
- Defining a Region of Interest, Perspective Transform, Warping an Image from bird's eye view.
- Selecting the color and returning binary image by comparing channel with threshold.
- Apply Sobel Operator (Heart of Canny Edge Detection) to warped Image.
- Combining Different ColorSpaces(s,l,y channels) and Sobel Variants(sobelx, sobely, sobeldir).
- Detect lane pixels and fit to find the lane boundary using histogram and sliding window search which can be starting point of left/right lanes.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Define the pipeline calling all above mentioned steps one by one inside function
- Running pipeline on video.

✓ Implementation of camera calibration using chessboard images

Steps during camera calibrations:

- Grayscale the image
- Find Chessboard Corners. It returns two values ret, corners. ret stores whether the corners were returned or not.
- If the corners were found, append corners to image points.

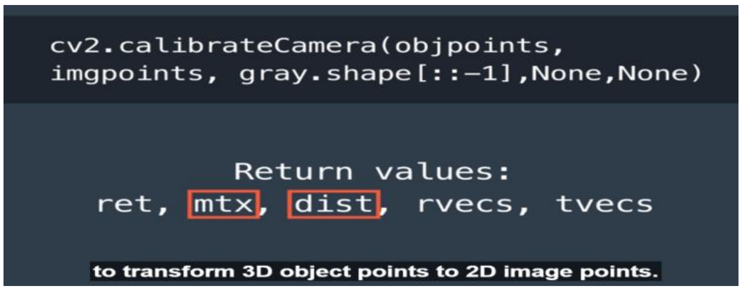


These image points and object points will be required to calculate the camera calibration and distortion coefficients.

✓ Distortion coefficients on given set of chessboard images.

The calibrateCamera function which returns us a different parameters, but we are interested in the camera matrix (mtx) and distortion coefficient (dist). However, below are the parameter details:

- mtx: camera matrix
- dist: distortion coefficient
- rvecs: rotation vectors --> Used to return position of camera in the world.
- tvecs: translation vectors --> Used to return position of camera in the world.



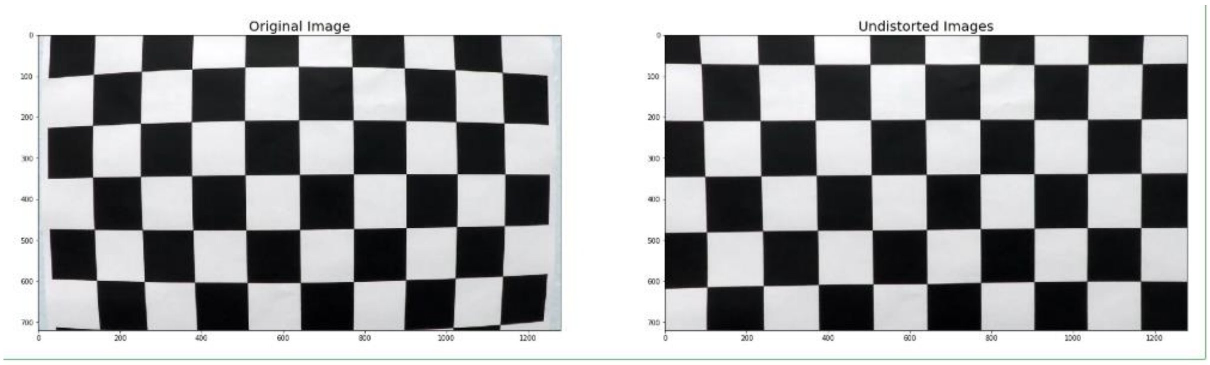
```
cv2.calibrateCamera(objpoints,
imgpoints, gray.shape[::-1],None,None)

Return values:
ret, mtx, dist, rvecs, tvecs
```

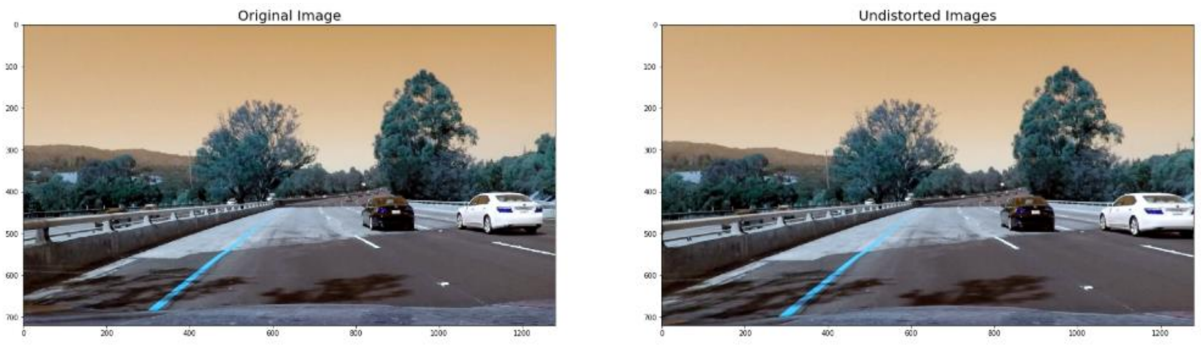
to transform 3D object points to 2D image points.

- ✓ **Since we have all the distortion coefficients calculated above, create distortion correction function for raw images.**

We then use the distortion coefficient to undistort given image.



Implementing the results on test images



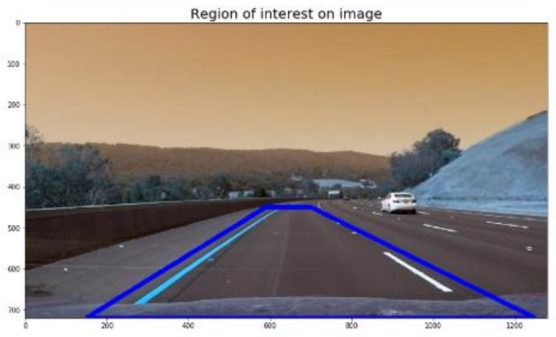
- ✓ **Defining a Region of Interest, Perspective Transform, Warping an Image from bird's eye view.**

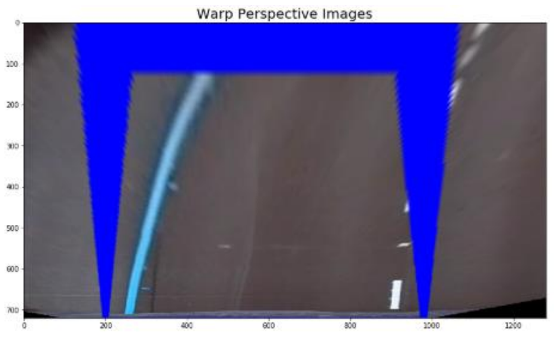
Region Of Interest (ROI) is Trapezoid with four points. (Below are reference from code)

- Left Bottom Corner defined as "left"
- Right Bottom Corner defined as "right"
- Left Upper Corner defined as "apex_left"
- Right Upper Corner defined as "apex_right"

After declaration of ROI, to see image in bird eye perspective, we need to warp the image. Hence calculated the matrix with source and destination points. Destination points are selected randomly as hit and trial method.

Once we will get the matrix, we will use this image and give input to cv2 warpPerspective function to get final warped image from function.

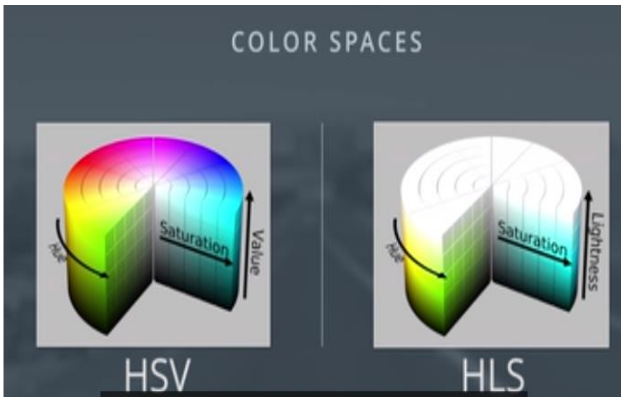




✓ Selecting the color and returning binary image by comparing channel with threshold.

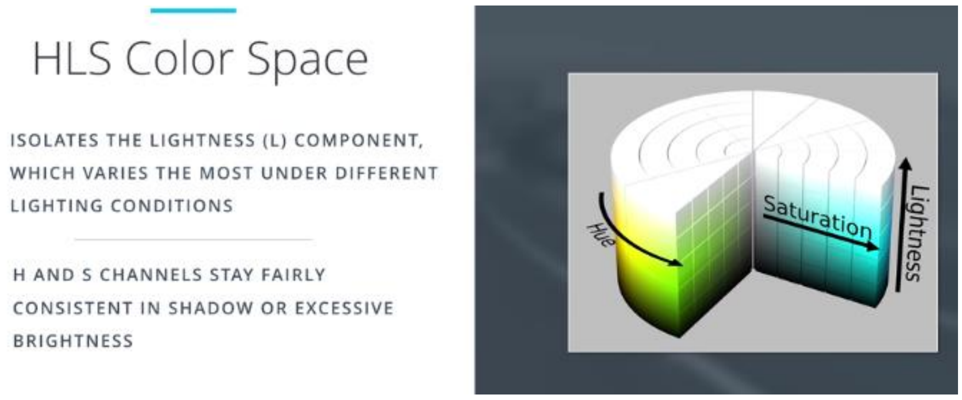
To get good binary image in all lighting conditions, there are various number of colorspaces which can be applied to do so.

- HLS
- HSV
- LAB etc



HSL (**hue, saturation, lightness**)

HSV (**hue, saturation, value**)



HLS Color Space

ISOLATES THE LIGHTNESS (L) COMPONENT, WHICH VARIES THE MOST UNDER DIFFERENT LIGHTING CONDITIONS

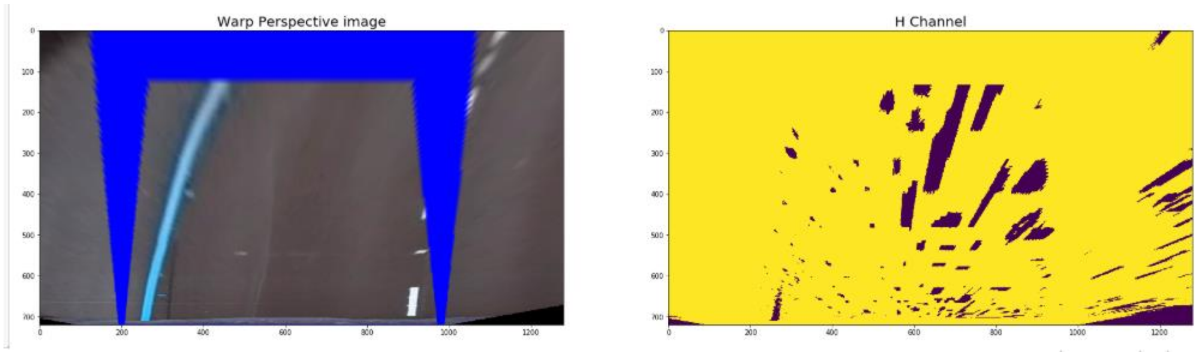
H AND S CHANNELS STAY FAIRLY CONSISTENT IN SHADOW OR EXCESSIVE BRIGHTNESS

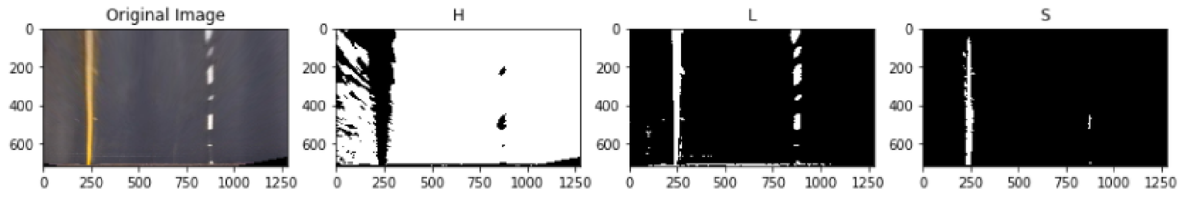
There is a common function to extract a particular channel from a colorspace as “ExtractChannel” in code.

The function has different input parameters such as :

- image - the warped image from which we need to extract
- colorspace- the cv2 colorspace. Ex- cv2.COLOR_RGB2HSV
- threshold- the threshold value of pixels which need to be selected in order to get the binary image. [min_threshold, max_threshold]
- channel- the channel we need to extract from the image

Output of function contains Binary Image with the required channel (s,l,h)and threshold values applied.





✓ Apply Sobel Operator (Heart of Canny Edge Detection) to warped Image.

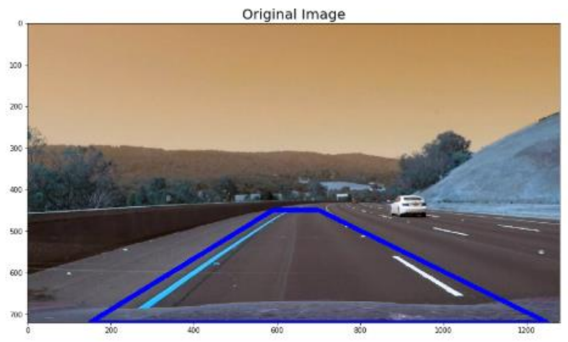
Function created for Sobel operator (to be used in pipeline) consists of :

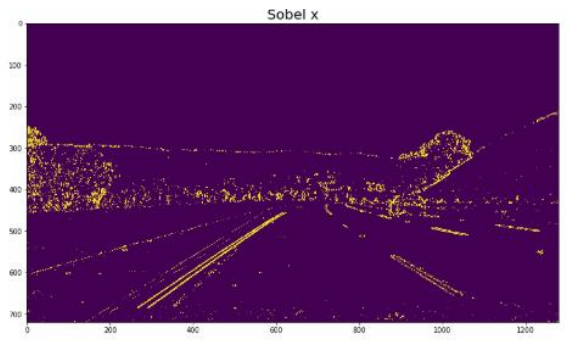
Input-

- warpedimage- the original warped image
- threshold- the threshold that is to be applied to select the pixel values
- sobelType- the direction where we need to take the gradient. values- x- for x gradient , y- for y gradient, xy for absolute and dir for direction
- kernelSize- the size of the kernel

Output- Binary Image with the required thresholds , sobelType and kernelSize.

Also, I have shown step by step details of, how Sobel operator can be applied on the raw original image in code. And below are the results for same.





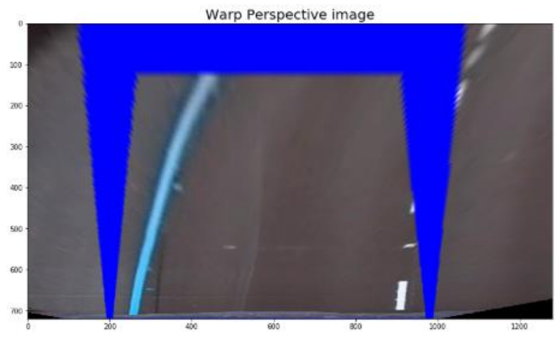
✓ Combining Different ColorSpaces(s,l,y channels) and Sobel Variants(sobelx, sobely, sobeldir).

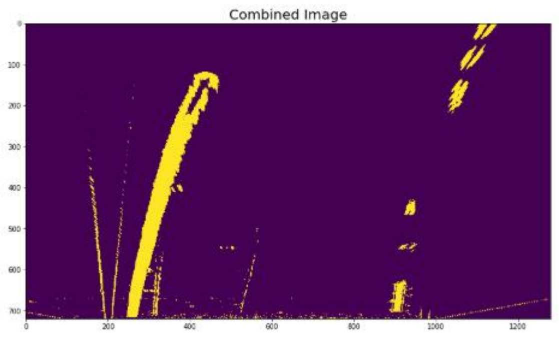
Since one single channel is not able to provide results up to the mark, using three channels as y-channel is luminance channel from YUV colorspace- the Y channel. s-channel is Saturation channel of HLS. l-channel is Lightness channel of HLS which works well in all the conditions except the case when the image is too bright.

Once color space is selected, selected the x gradient. Since we can see clear vertical edges using the x gradient, I decided to use X gradient only.

Final Combination-

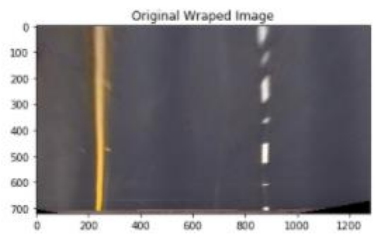
1. Mix Channel 1 = Saturation Channel and Lightness Channel from HLS
2. Mix Channel 2 = Mix Channel 1 and Y channel for YUV
3. Final Combination= Mix Channel 2 or Sobel Gradient in X direction

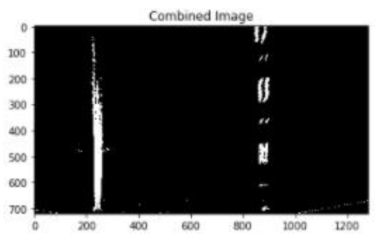


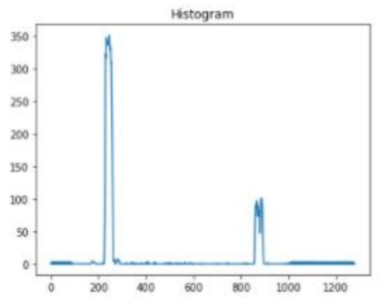


✓ Detect lane pixels and fit to find the lane boundary using histogram and sliding window search which can be starting point of left/right lanes.

The first step is to create a Histogram of lower half of the image. With this way we are able to find out a distinction between the left lane pixels and right lane pixels.



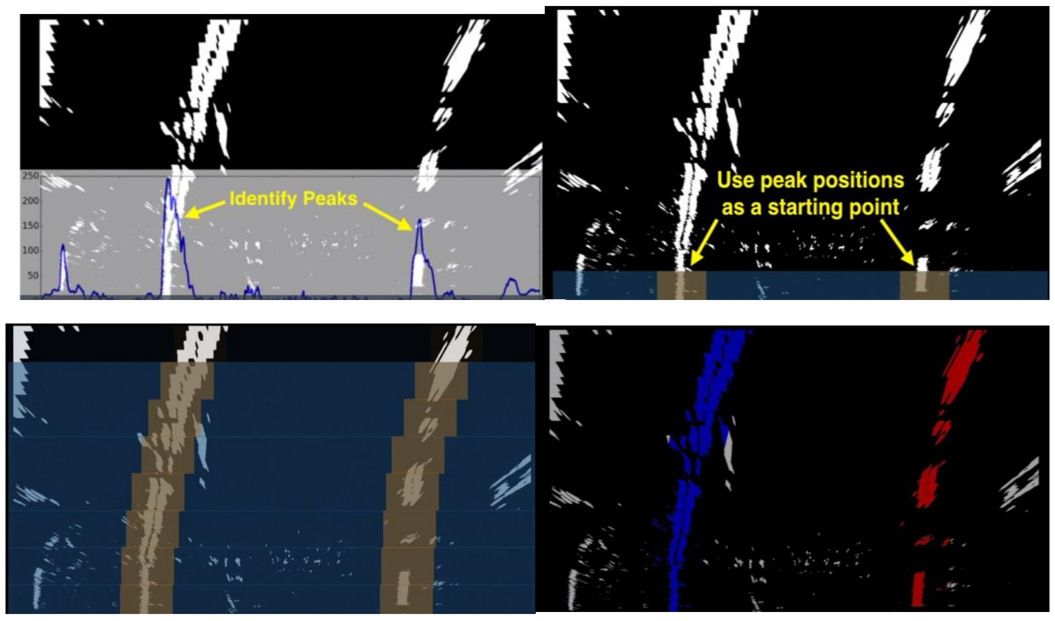


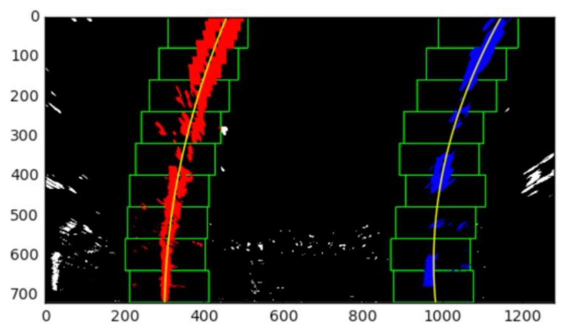
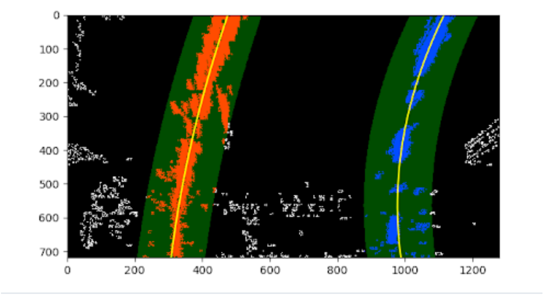


The next step is to initiate a Sliding Window Search in the left and right parts which we got from the histogram. We can use the two highest peaks from our histogram as a starting point for determining where the lane lines are .Then use sliding windows moving upward in the image (further along the road) to determine where the lane lines go

The sliding window is applied in following steps:

1. The left and right base points are calculated from the histogram
2. We then calculate the position of all non-zero x and non-zero y pixels.
3. We then Start iterating over the windows where we start from points calculate in point 1.
4. We then identify the non-zero pixels in the window we just defined
5. We then collect all the indices in the list and decide the center of next window using these points
6. Once we are done, we separate the points to left and right positions
7. We then fit a second degree polynomial using np.polyfit and point calculate in step 6.



✓ Determine the curvature of the lane and vehicle position with respect to center.

To calculate Radius-:

1. First we define values to convert pixels to meters
2. Plot the left and right lines
3. Calculate the curvature from left and right lanes separately
4. Return mean of values calculated in step 3.

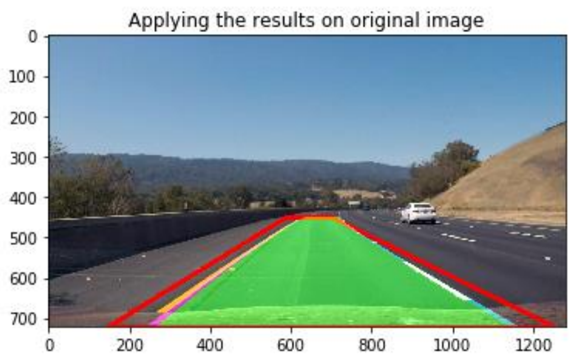
Distance Calculation:

Since the centre of image is the center of the car. To calculate the deviation from the center, we can observe the pixel positions in the left lane and the right lane. We take the mean of the left bottom most point of the left lane and right bottom most point of the right lane and then subtract it from the center of the car to get the deviation from the center.

✓ Warp the detected lane boundaries back onto the original image.

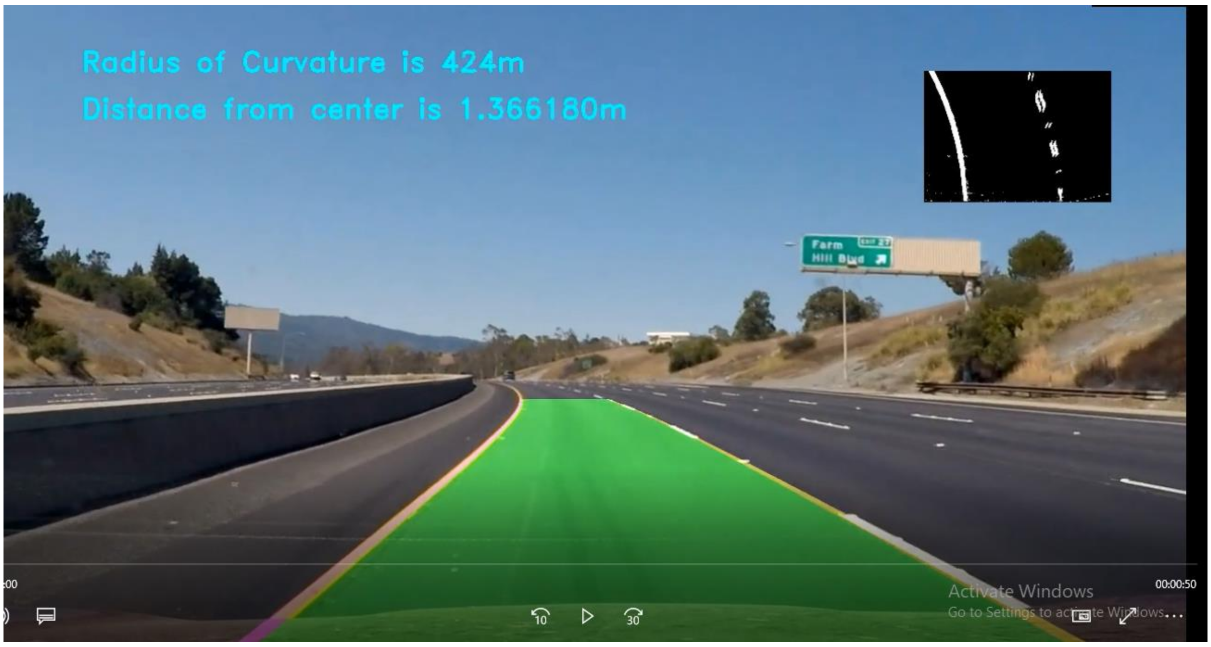
Now it's time to unwarp the image back to original image. Following are the steps :

1. Recast the x and y point to give as input in cv2.fillPoly. These are the same points we got from fitting the lines.
2. Calculate the Minv which is Inverse Matrix. This is done by passing the reverse points this time to getPerspectiveTransform function
3. Draw the sidelines from the points selected in step 1 onto a blank warped image
4. Unwarp the image using cv2.warpPerspective.
5. Combine the original image with the image we got from step 4 to plot the lane lines.



✓ Define the pipeline calling all above mentioned steps one by one inside function and run on video

So far, happy with the results and the output video is attached with all the files.


Radius of Curvature is 424m
Distance from center is 1.366180m
Farm Hill Blvd
00:00:50
Activate Windows
Go to Settings to activate Windows...

Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

1. Since Destination points are selected randomly as hit and trial method, small change in them impacts more on results.
2. Most of the coding part, I am unable to recall the code lines. So used most of the as it is references from available codes which are mapping with concepts.
3. Different combinations of color channels tried but the final combination did not work on all other conditions.
4. There is flickering of images when there is shadow .
5. This pipeline failed on challenge video , since it is getting confused with uneven surfaces within lane boundaries.