# Advanced Lane Finding Project

The goals / steps of this project are the following:

1. Create a software application that can calibrate the camera images for distortion so that the actual images are corrected for distortion.
2. Create a pipeline and test the pipeline to convert a camera image into a binary image that identifies the lanes after undistortion
3. Create additional code that identifies the lane position and apply the same on the original image to identify the lanes
4. Test the application on few test images
5. Finally test the application on videos and create an output video with lanes identified

## Rubric Points

### Here I have considered the rubric points individually and described how I addressed each point in my implementation.

#### 1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

This project has implemented the following steps

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position

I tested the pipeline first on single image to check if the binary output was as expected. Then i tested the application on multiple images including those images where tree shades falls on the lane. Once this is done, i tested my application on full code and then later with the video

### Camera Calibration

#### 1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

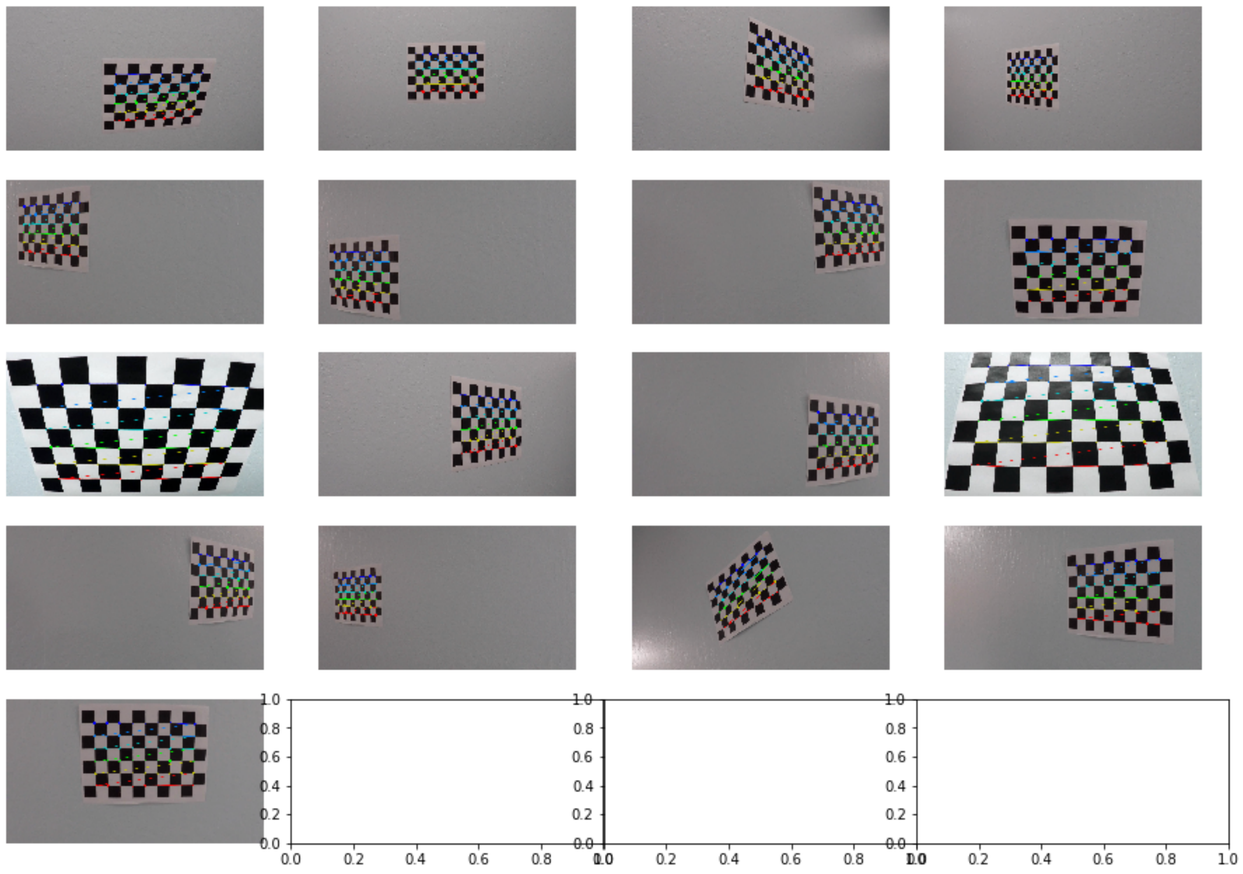
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result. I also realized that the number of corners were completely different from the original example that was present in udacity website. so I introduced a new input parameter imgcorner type that handles for both the scenarios.

Calibrate_camera accepts path of the calibration images and returns - ret,mtx,dist,rvecs,tvecs

### Output of camera calibration

```
calibrating camera for image calibration10.jpg
calibrating camera for image calibration6.jpg
calibrating camera for image calibration9.jpg
calibrating camera for image calibration7.jpg
calibrating camera for image calibration19.jpg
calibrating camera for image calibration20.jpg
calibrating camera for image calibration16.jpg
calibrating camera for image calibration1.jpg
calibrating camera for image calibration17.jpg
calibrating camera for image calibration2.jpg
calibrating camera for image calibration8.jpg
calibrating camera for image calibration4.jpg
calibrating camera for image calibration15.jpg
calibrating camera for image calibration3.jpg
calibrating camera for image calibration14.jpg
calibrating camera for image calibration11.jpg
calibrating camera for image calibration13.jpg
calibrating camera for image calibration12.jpg
calibrating camera for image calibration5.jpg
calibrating camera for image calibration18.jpg
```



## Pipeline (single images)

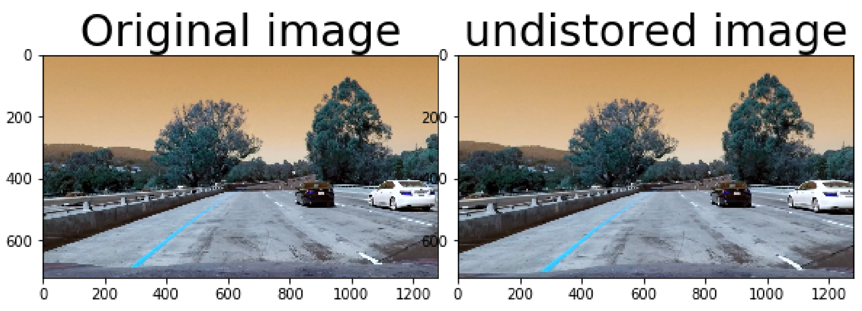
The pipeline consists of the following steps:

1. Undistort function for correction distortion
2. Perspective Transformation
3. Sobel Magnitude
4. Sobel Direction
5. HLS H-channel Threshold
6. HLS S-channel Threshold
7. HLS L-channel Threshold
8. Sobel X and Y

#### 1. Provide an example of a distortion-corrected image.

The distortion correct is done using CV2.Undistort in the function undistort_img

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:



**2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.**

I used a combination of color and gradient thresholds to generate a binary image. Below are the details

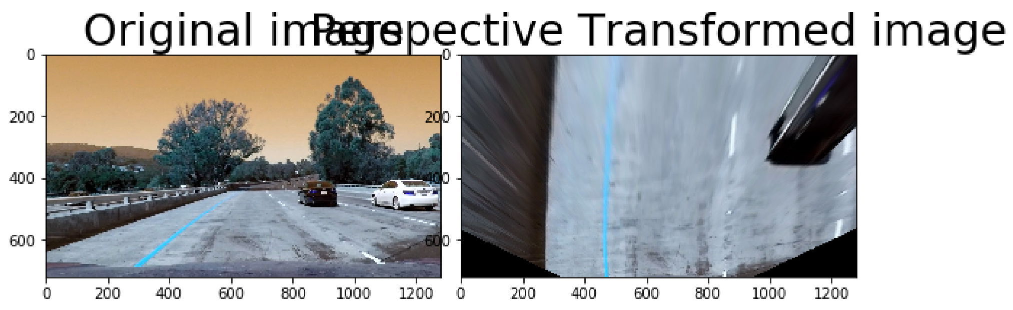
1. HLS Channels
2. Sobel Magnitude
3. Sobel Direction
4. Sobel X and Y

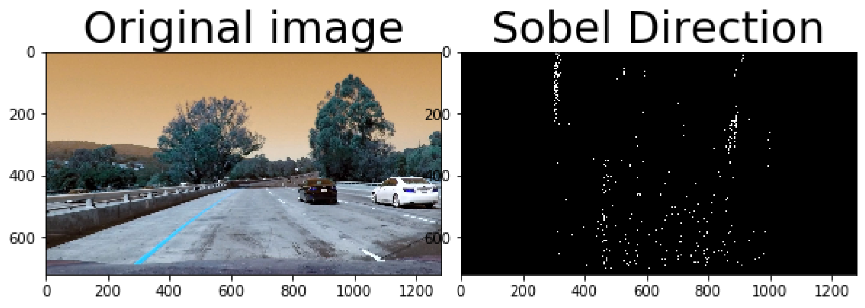
I tested various images for each of these techniques with various thresholds and decided to go with only few methods to get the desired output. However, almost all of these methods were not working for atleast one or two scenarios and most of them like HLS Channels had unwanted pixels along with lanes. So the final binary image had multiple peaks in histogram. so to avoid this, I created a ROI and applied that to the output of these channels to remove these noises.

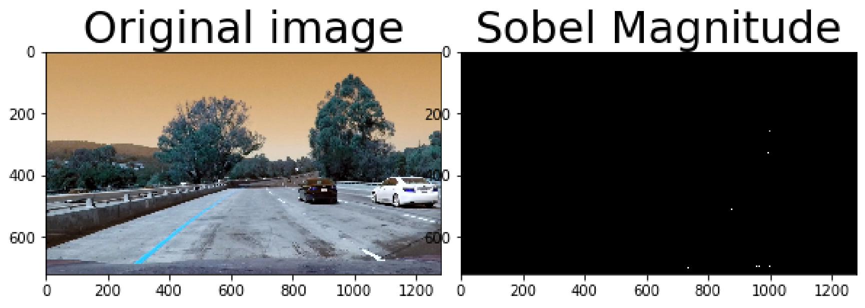
A combination of below thresholds is used to calculate the final binary image. I ignored the other thresholds as they were not producing satisfactory results and with these functions alone, I was able to identify the lanes

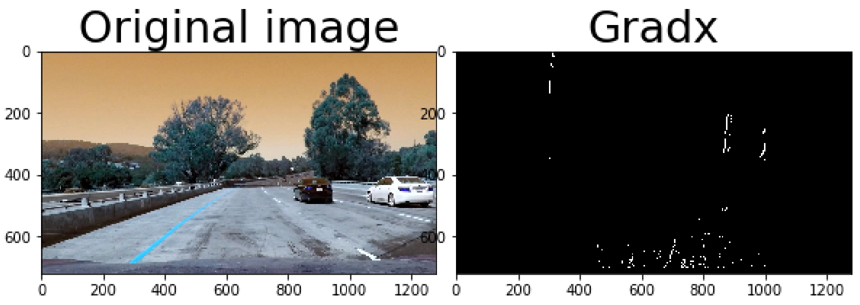
1. Sobel X
2. Sobel Direction
3. HLS S-Channel & L- Channel

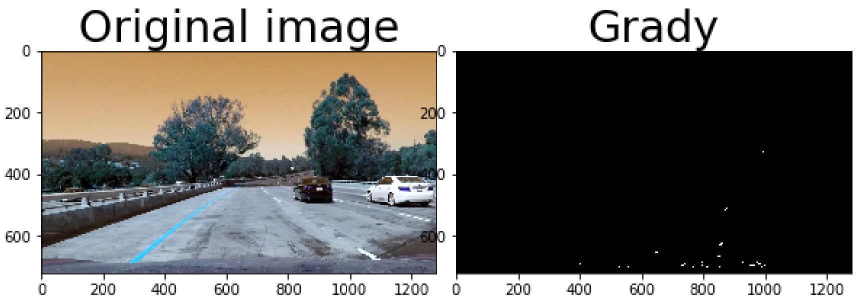
Here's an example of my output for this step. (note: this is not actually from one of the test images)

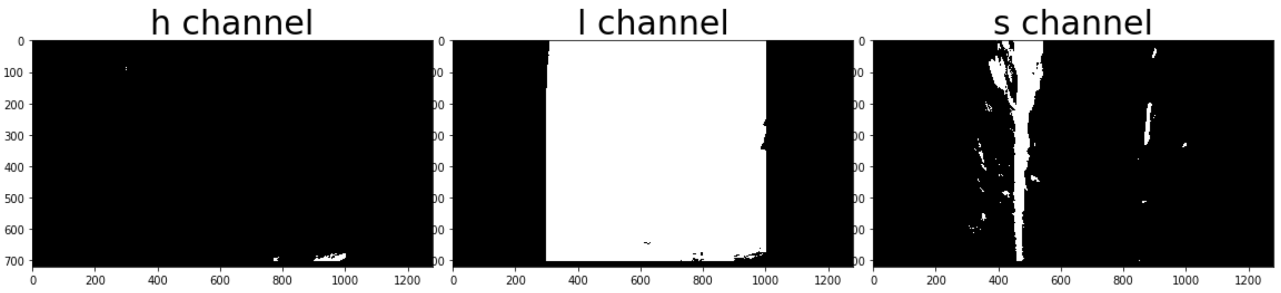


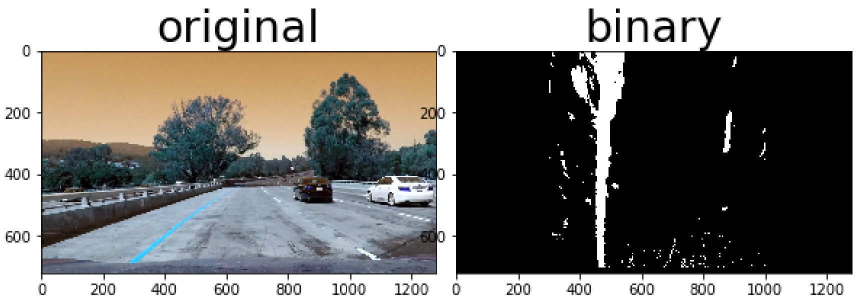






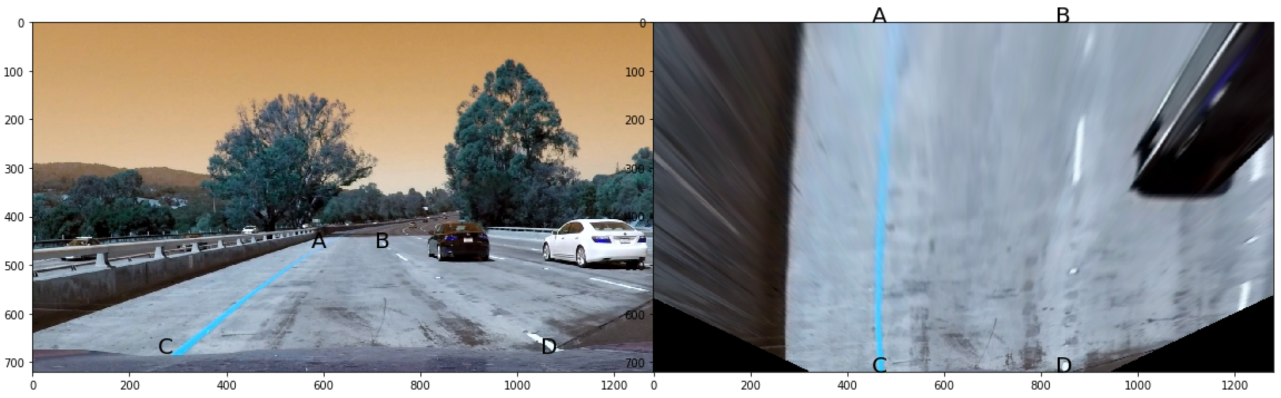






**3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.**

I selected the points by ensuring that the points exactly covered the lane. You can see in the below images, pts A,B,C,D are marked to denote both src and dst



I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image. Here is the code for the perspective transformation. The code returns the warped image, M and Minv (for later applying reverse transformation)

src dst 575,464 450,0 707,464 width-450,0 258,682 450,height 1049,682 width-450,height

The function unwarp_image does the perspective transformation using cv2.getPerspectiveTransform and src and dst. The inverse of this is also returned by shuffling src and dst. This is later used to map the lanes onto the original image

#### 4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Lane fitting is done using the below logic:

1. Histogram of the binary image is calculated to arrive at the two lanes where the heights are at maximum
2. The two base positions of leftx and rightx are identified and the lanes are extrapolated in the y direction using second order polynomial - f(y)=Ay**2 + By + C.
3. Coeffecient A, B and C are arrived as L_fit and R_Fit using Poly function
4. For subsequent images, instead of starting from start, the original positions of the previous image is used to extrapolate
5. To fit the lanes, coefficients are substituted along with various values of y as identified by linspace.

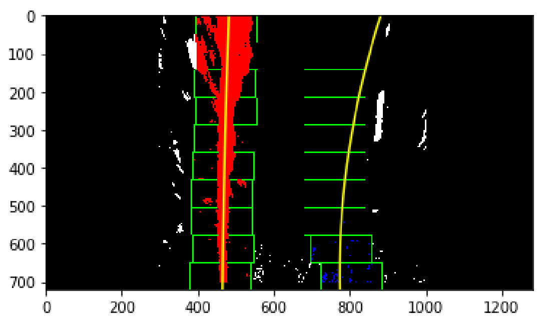
Below are the places in the code where the above logic is implemented:

Fully_process_images: This function invokes the pipeline to get the binary image and calls Sliding_window_polyfit that does the sliding window polyfit. This function returns the left and right fit co-efficients along with the right and left lane indices and the histogram of the binary image. Once the fit is identified, poly_next_fit is identified for the subsequent images. The fits are added to the Left Line and Right Line instance of class Line. Draw_Line function draws the lane based on teh identified fits. Sliding Window polyfit does the below:

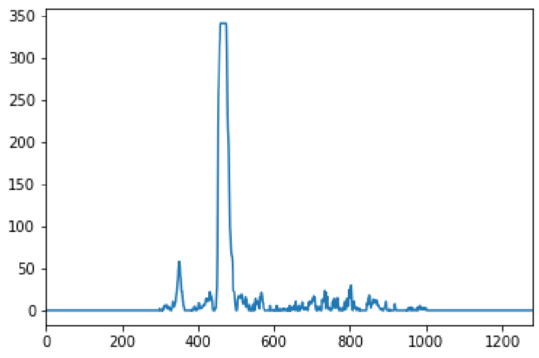
1. creates a histogram of the binary image
2. identify the left and right base for the lanes.
3. Steps through 10 times to identify the window boundaries
4. use Polyfit to arrive at the left and right coeffecients based on the initial position for extrapolation

polyfit_next_fit is used for the subsequent images to identify the fit and indices. Draw_lane function draws the lane based on teh fit into a warped blank image. The warp blank image is of the same size of the original image. CV2.fillpoly and polylines are used to draw the lane onto the warped blank image. Then the warped blank image is transformed into the original image space using the inverse perspective matrix (MINV) and combined with original image to create the final output.

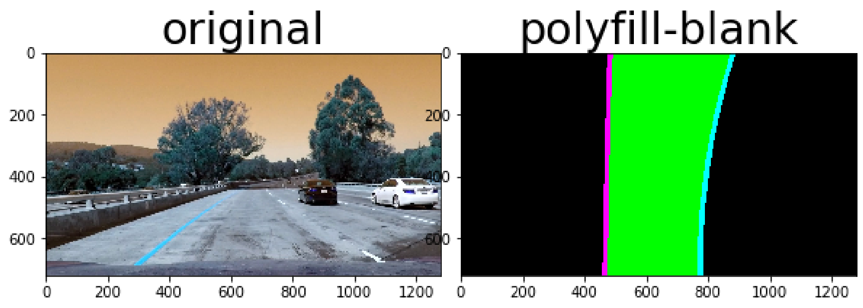
Here is the output

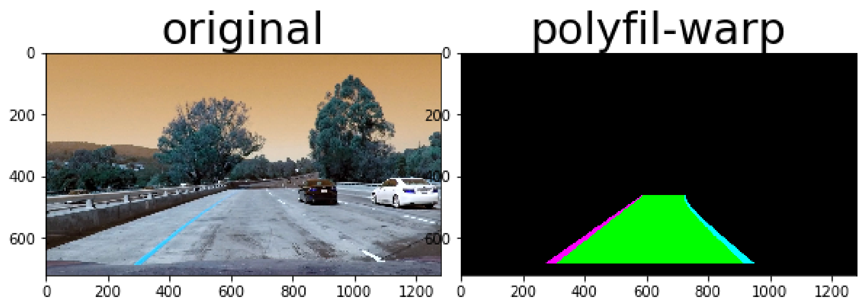


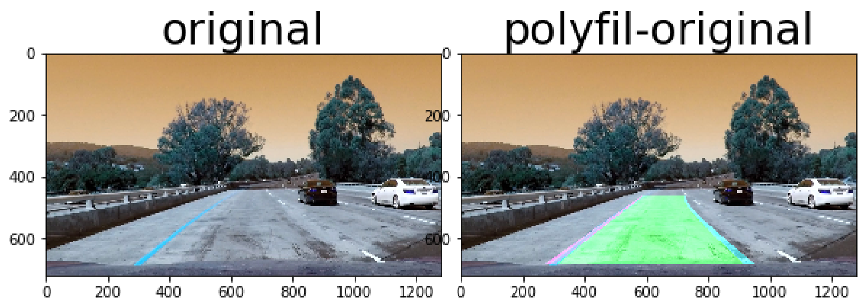
```
histogram for image test1.jpg
```



```
...
```







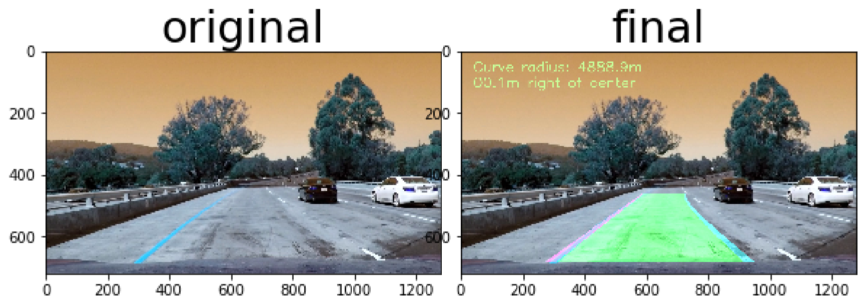
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

```
function to determine radius of curvature and distance from lane center based on binary image, polynomial
fit is defined in calculate_rad and draw_curvature

The radius of curvature is defined by the function

Radius of curve = [((1 + (dx/dy)**2)]**1.5/(f``y)
```

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.



## Pipeline (video)

**1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).**

Here's a link to my video result

## Discussion

**1. 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?**

My implementation fails partially for Challenge and fully for the other challenging video. I believe this is because, fitting needs to be improved in my code. Also i had set ROI to filter noises. Hence this is creating issues when the lane curvature is very high and implementation tries to map the lane within ROI and hence lanes are not correctly mapped. Also i need to look for other color thresholds and other pre processing technique to improve the binary image quality