# Advanced Lane Finding Project

The goals / steps of this project are the following:

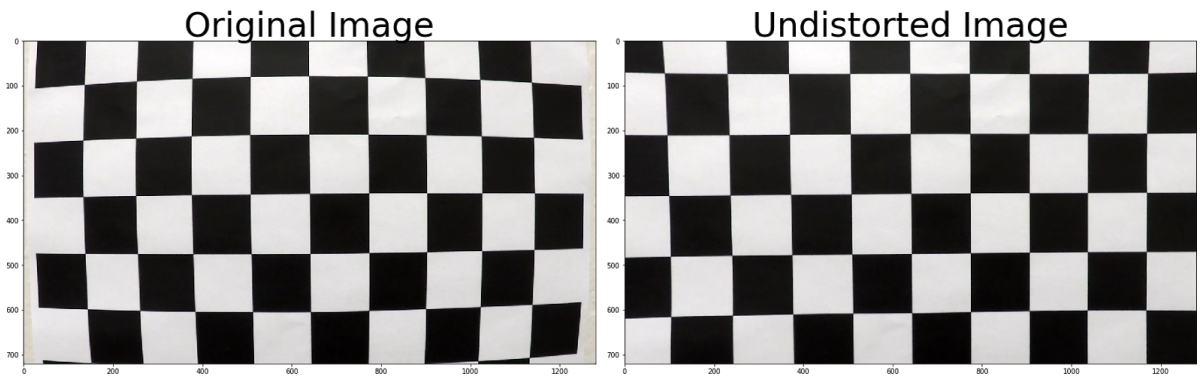
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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.

**Camera Calibration**

*The code for this step is contained in the first code cell of the IPython notebook located in advanced_lane_finder.ipynb" .*

*I followed the lecture to calibrate the camera using the chessboard images .*

*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 saved the distortion coefficients for applying undistortion on video frames.

**Pipeline (single images)**

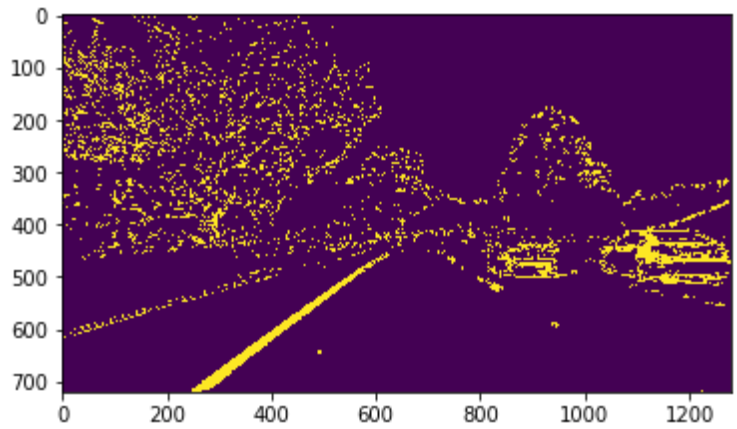
**1. Distortion-correction**


**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 .

```
s_thresh=(170, 255)
sx_thresh=(40, 170)

gradx = abs_sobel_thresh(image, orient='x', sobel_kernel=ksize, thresh=sx_thresh)
grady = abs_sobel_thresh(image, orient='y', sobel_kernel=ksize, thresh=sx_thresh)
mag_binary = mag_thresh(image, sobel_kernel=ksize, thresh=(40, 100))
dir_binary = dir_threshold(image, sobel_kernel=ksize, thresh=(0.7, 1.3))

combined = np.zeros_like(dir_binary)
combined[((gradx == 1) & (grady == 1)) | ((mag_binary == 1) & (dir_binary == 1))] = 1

    # Threshold color channel
 s_binary = np.zeros_like(s_channel)
 s_binary[(s_channel >= s_thresh[0]) & (s_channel <= s_thresh[1])] = 1

 combined_binary = np.zeros_like(s_binary)
 combined_binary[(s_binary == 1) | (combined == 1)] = 1
```

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



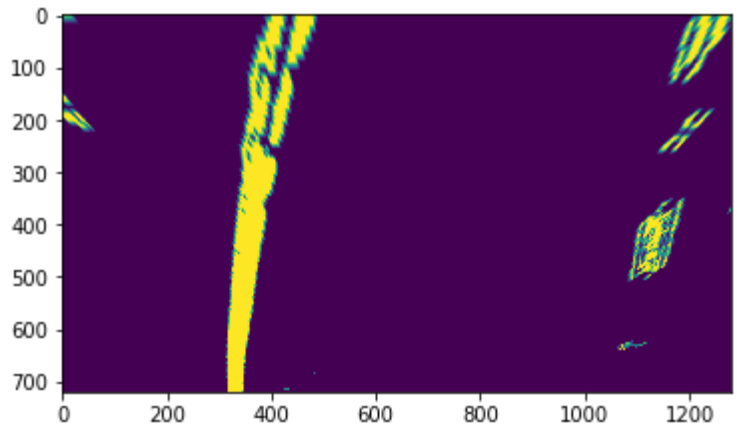

**3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.**

The code for my perspective transform is included in a function  called apply_perspectivetransform and called in get_warped_img(img) .

I choose these points as src and dst points

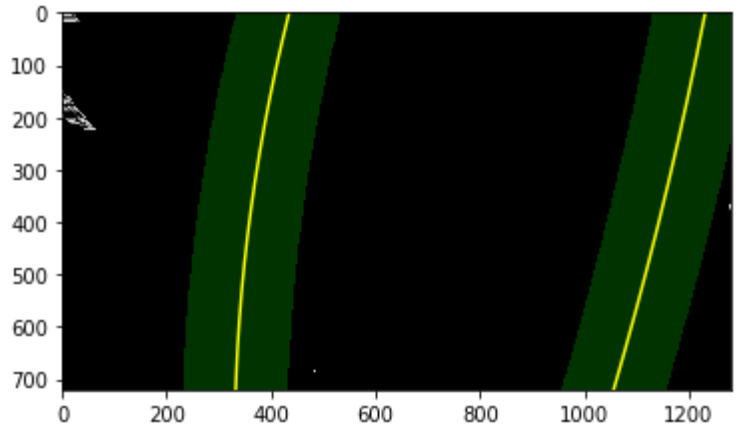
```
src = np.float32([
          [image_width * 0.4475, image_height * 0.65],
          [image_width * 0.5525, image_height * 0.65],
          [image_width * 0.175, image_height * 0.95],
          [image_width * 0.825, image_height * 0.95],
      ])

   # Destination coordinates
   dst = np.float32([
          [image_width * 0.2, image_height * 0.025],
          [image_width * 0.8, image_height * 0.025],
          [image_width * 0.2, image_height * 0.980],
          [image_width * 0.8, image_height * 0.980],
      ])
```

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.




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

Then I apply sliding technique defined in lecture to fit polynomial on lane lines .

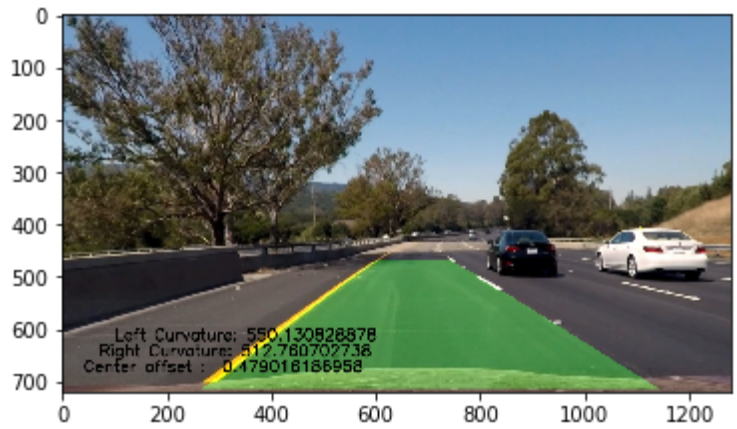



**5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.**

For radius of curvature I followed approach in lecture  and for offset I took guidance from blogs on how to calculate estimated distance from centre

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

I implemented this step in  draw_lane_lines_to_original_image
 Here is an example of my result on a test image:




**Pipeline (video):**

For the video I followed the guidelines given in the lecture .
The video output.mp4 is included along with this file.

## 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?

Currently my approach is not working well in changing of light intensities and also in case of the challenge video it seems to loose track .Will need to implement better sliding window technique and also tweak parameters to get it working on challenge .