# Writeup

# Advanced Lane Finding Project

**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.

# Rubric Points

## Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

### Writeup / README

#### 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.

### Camera Calibration

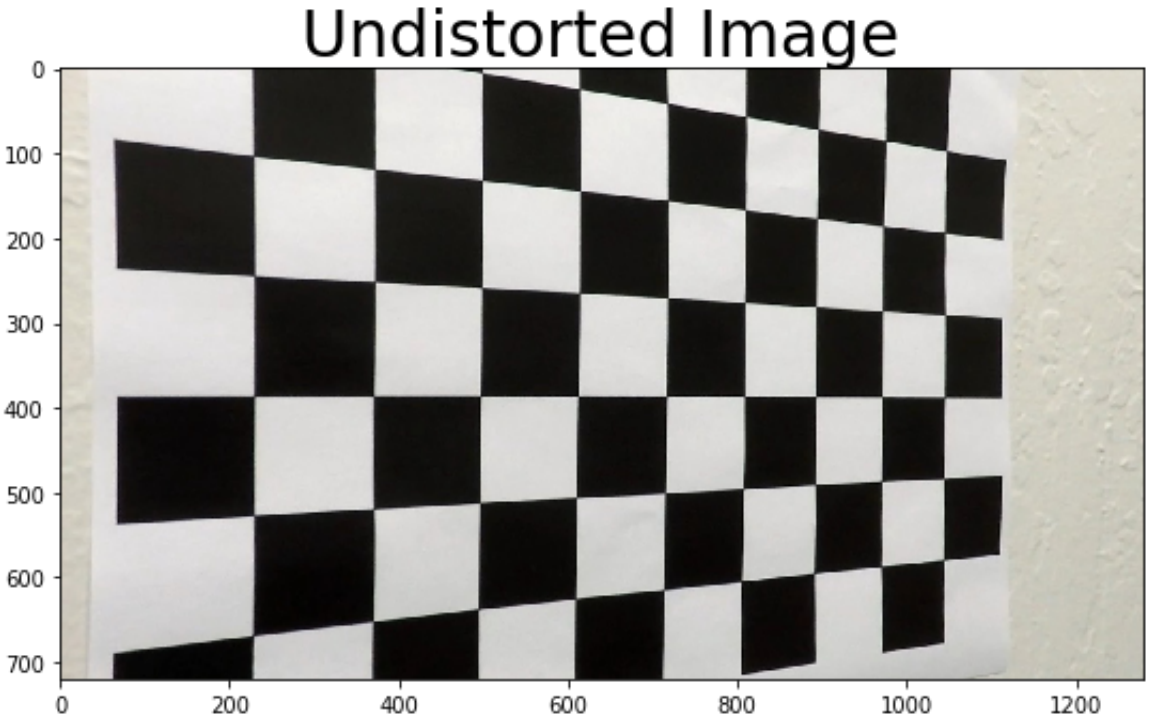
#### 1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in part "1.1 Camera Calibration" of file "/code.ipynb".

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:




## Pipeline (single images)

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

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

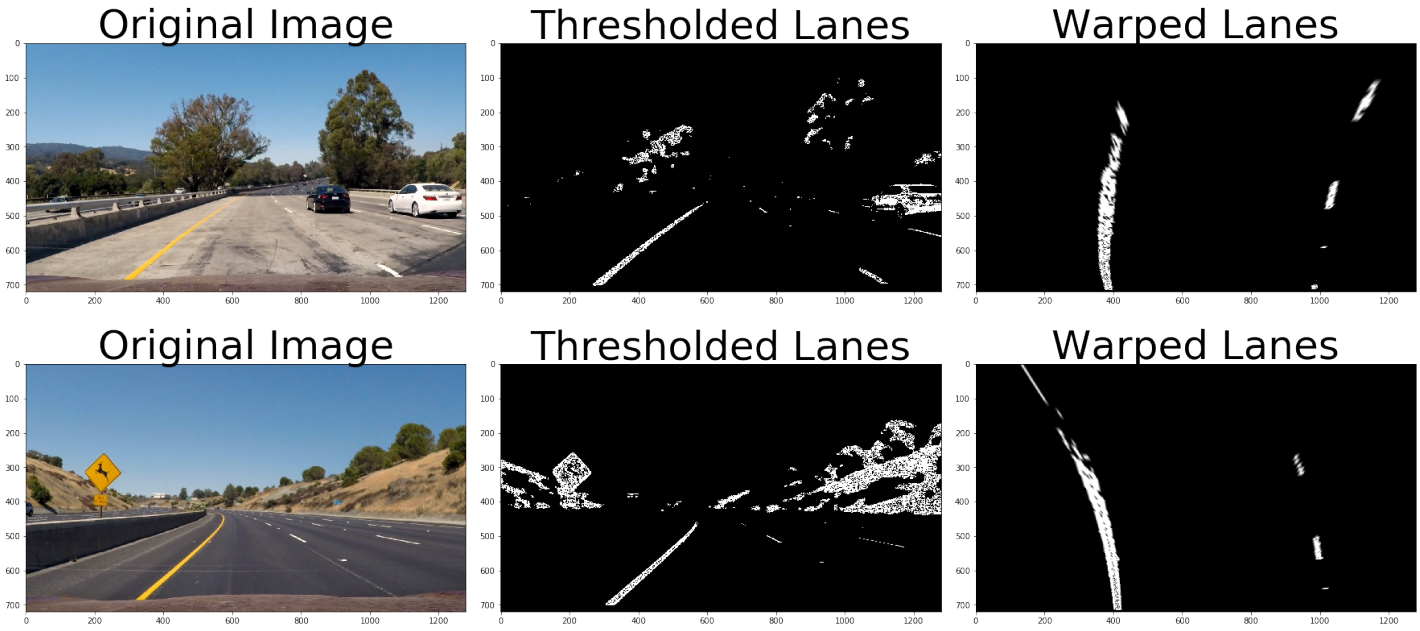

## 2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image (in part "2.1 Find raw lane without fit").
It is noticed that there are only white and yellow lanes in the test vedio. I selected Lab and LUV color spaces for yellow and white lines detecion.
* In Lab, b channel was used to detect yellow lines, with threshold (150,220) which provided acceptable result
* In LUV, L channel was selected to detect white lines, with threshold (220,255).

The gradx thrshold was tried but not selected in this project due to its bad performance.

Additionally, direction threshold (np.pi/6, np.pi/2) was used to remove some noisy pixels. on the b channel of Lab (for yellow lines) and the L channel of LUV (for white lines)

Here's two examples of my output for this step.




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

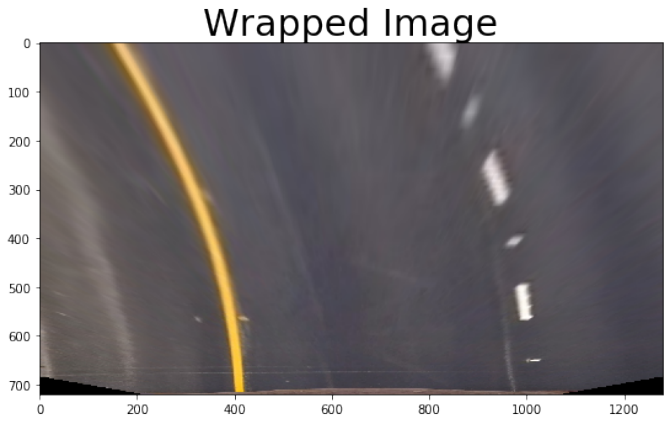
The code for my perspective transform includes a function called `warp()`, which appears in part "0. Helper Functions", the 4th cell of file `code.ipynb`. The `warp()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

```
offest=350
src=np.float32(
        [[600,450],
         [680,450],
         [1080,720],
         [200,720]])
dst = np.float32([[offset, 0], [img_size[0]-offset, 0],
                                    [img_size[0]-offset, img_size[1]],
                                    [offset, img_size[1]]])
```

This resulted in the following source and destination points:

| Source | Destination |
|---|---|
| 600, 450 | 350, 0 |
| 680, 450 | 930, 0 |
| 1080, 720 | 930, 780 |
| 200, 720 | 350, 780 |

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?

The lanes were fitted using following steps:
* Use histogram to identify peaks of the image to determine location of lane
* Use Sliding Window to to search for the lines (window number is set to 9)
* Fitting a second order polynomial to each lane using the numpy function `numpy.polyfit()`
* Fit new polynomials to x,y in world space with parameters(ym*per*pix = 30/720,xm*per*pix = 3.7/700)

### 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.

I did this part "2.3 Curature and offset calculation" in my code in `code.ipynb`

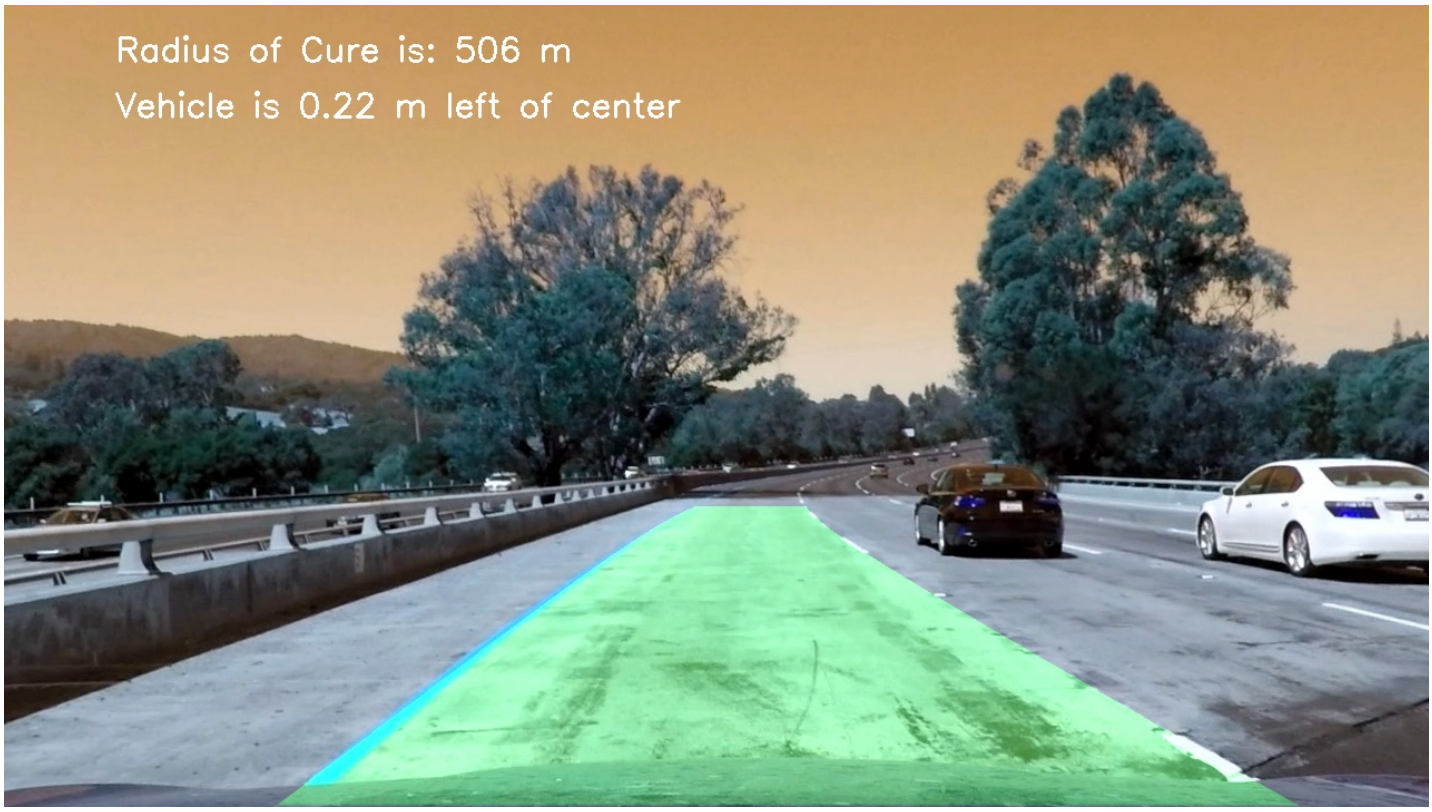
Cure radius calculation:

- Based on the fitted polynomials (world space) in step4.
- The left and right curve radius were calculated based on formular from project material
- 720(pixel value) was chosen as the evalation poit.
  In my final vedio output, only left curve radius was outputed.

Distance to center:
The distance from vehicle center to lane center was calculated by the difference between mid position of two lanes and mid of the picuture.

### 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 function `process_image()` in file `code.ipynb` . Here is an example of my result on a test image:




---

## 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 my video

---

## 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?

The most chanllenge part for me in this project is how to get suitable combination of color and gradient thresholds to generate a binary image.
In some cases, my code was not able to detect lanes in project test vedio. I have to retune the threshold parameters. In my final result, the code worked not robus enough in case of some brightness change areas. The combination shall be fine tuned. It also worth to try other color threshold methods to make the lane detection better.

The code shall be tested and improved in case of driving during night which will bring more chanllenges.