Advanced Lane finding

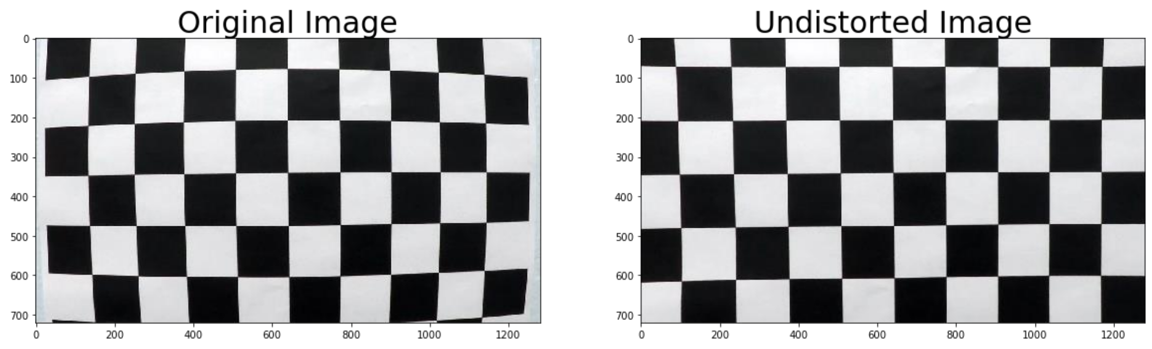
<u>### Camera Calibration</u>

#### 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 the first code cell of the IPython notebook located in "Advanced Lane finding.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:



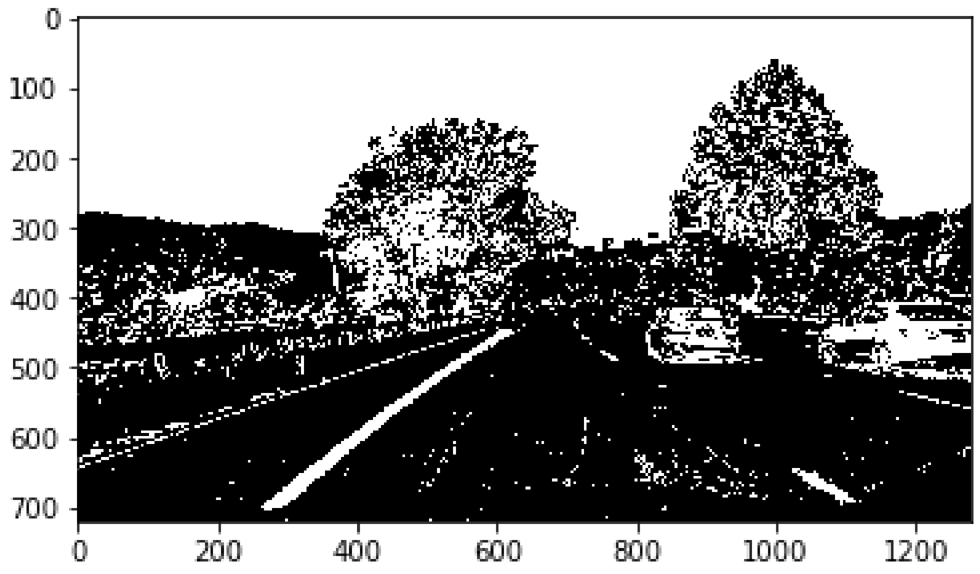
<u>### Pipeline (single images)</u>

#### 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.
The code for this step is contained in the 3rd code cell of the IPython notebook located in "Advanced Lane finding.ipynb" .
Here's an example of my output for this step.



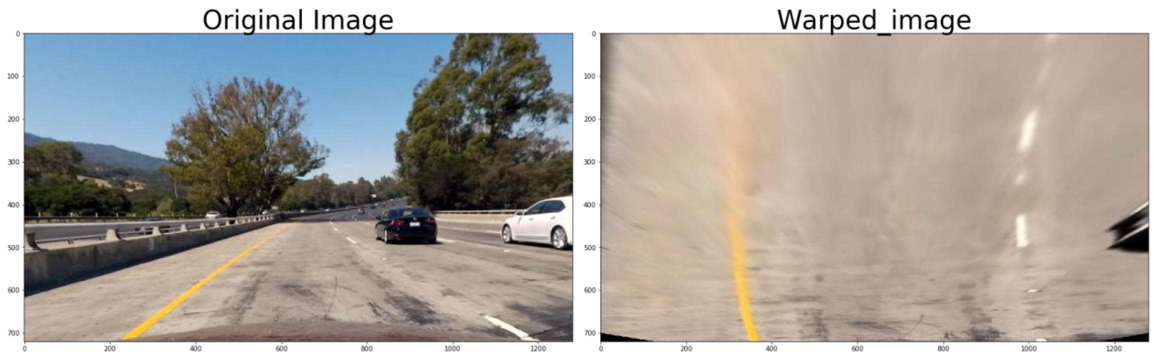
#### 3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for this step is contained in the 4th code cell of the IPython notebook located in "Advanced Lane finding.ipynb" .
This function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points.
I set the following parameter of source and destination points:

Source	Destination
720, 450	1030, 100

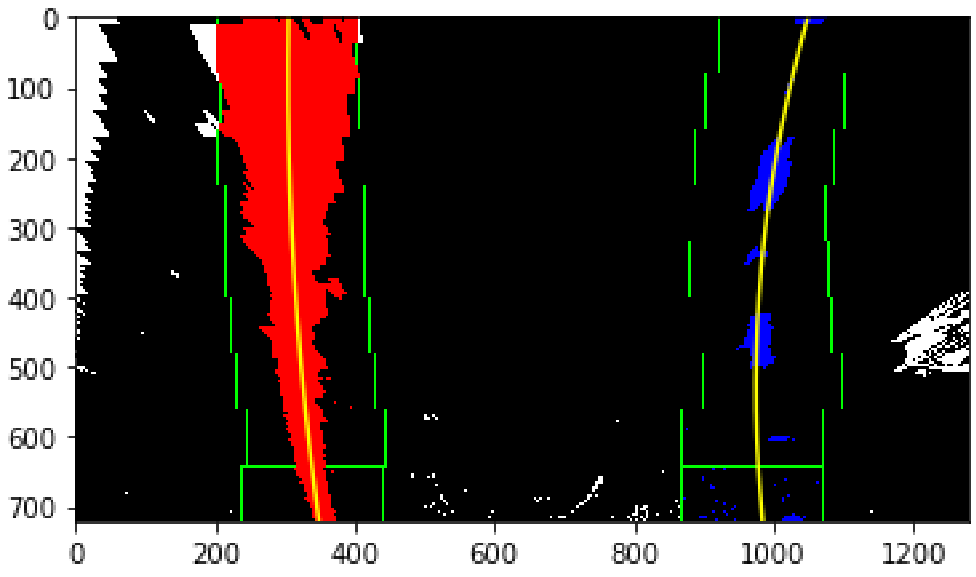
1030, 630	1030, 700
350, 630	350, 700
620, 450	350, 100

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 code for this step is contained in the 5th code cell of the IPython notebook located in "Advanced Lane finding.ipynb" .
I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:



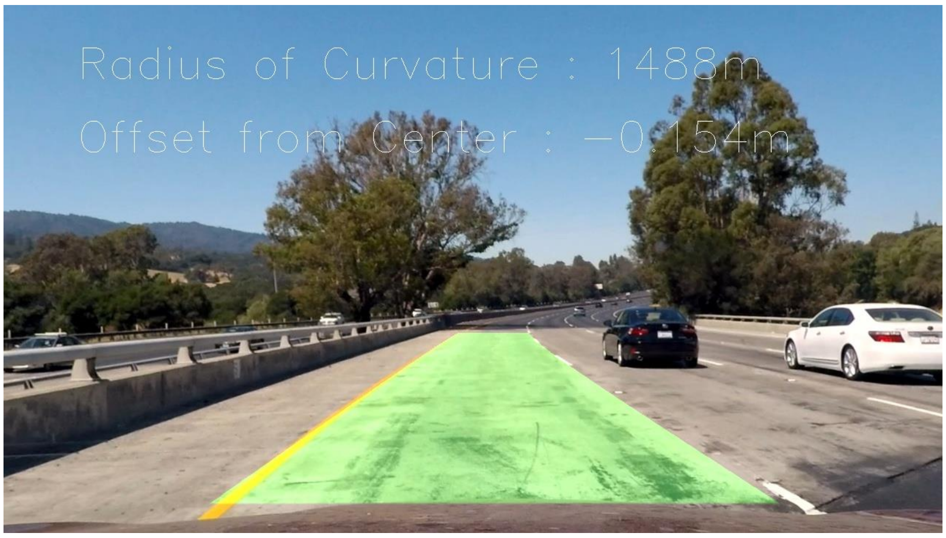
#### 5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

The code for this step is contained in the 7th code cell of the IPython notebook located in "Advanced Lane finding.ipynb" .
I calculated the radius of curvature of the lane and the position of the vehicle with respect to center as following:

Define y-value where we want radius of curvature.
Define conversions in x and y from pixels space to meters.
Fit new polynomials to x,y in world space.
Calculate the new radii of curvature.

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

The code for this step is contained in the 8th code cell of the IPython notebook located in "Advanced Lane finding.ipynb" .
Here is an example of my result on a test image:



<u>### Pipeline (video)</u>

#### 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.
"lane_finding_result.avi"

<u>### Discussion</u>

#### 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?

When a contrast between yellow lane and road serface is small, edge extraction result of yellow lane in far area is not good. Therefore fit result is sometime wrong. For making it more robust, some kind of filter may be necessary.