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](https://review.udacity.com/#!/rubrics/571/view) Points

Here I will consider the rubric points individually and describe 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](https://github.com/udacity/CarND-Advanced-Lane-Lines/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

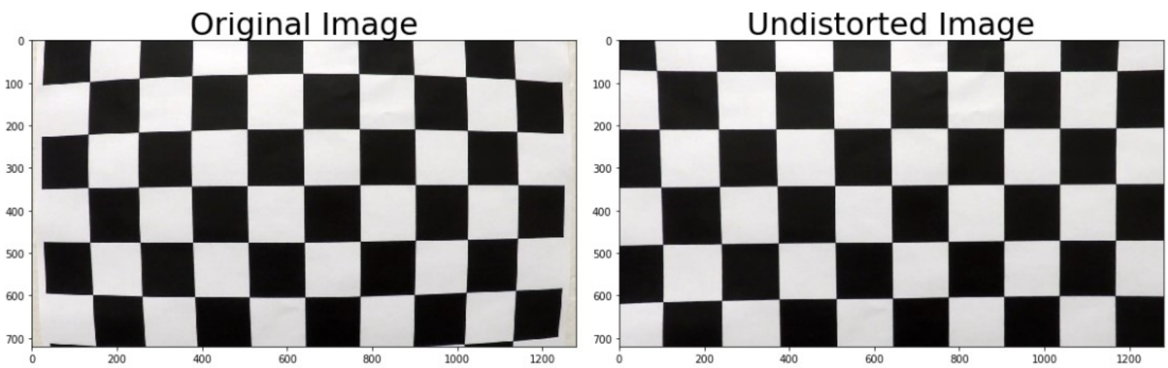
Camera Calibration

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 the first code cell of the IPython notebook located in "./T1P4_AdvancedLaneFinding.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 (thresholding steps at code cell 3 in T1P4_AdvancedLaneFinding.ipynb with function name *convertToBinary()*). 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 my perspective transform includes a function called `*warper()*`, which appears in the 4th code cell of the IPython notebook. The `*warper()*` 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:

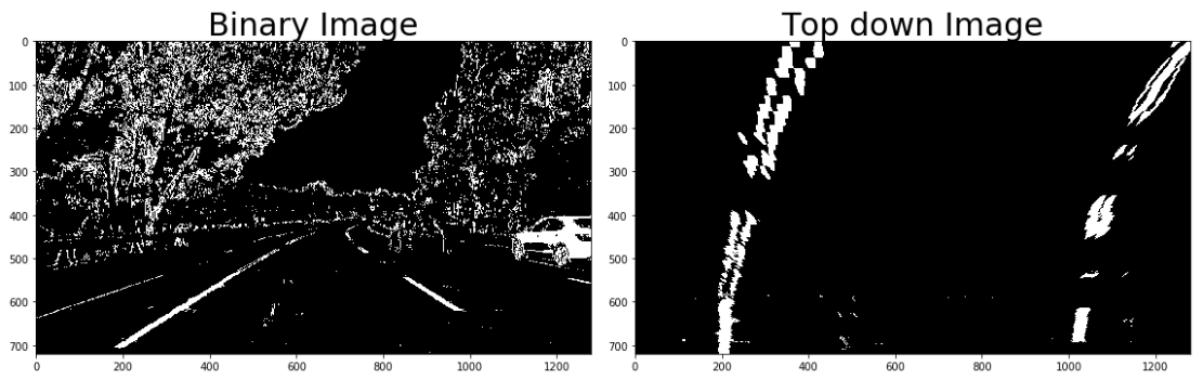
```
offset = 100 # offset for dst points

# Source points
src = np.float32([[[ 610,  450]],
                  [[ 680,  450]],
                  [[ img_size[0]-300,  680]],
                  [[ 380,  680]]])

# Result points
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
610, 450	100, 0
680, 450	1180, 0
980, 680	1180, 720
380, 680	100, 720

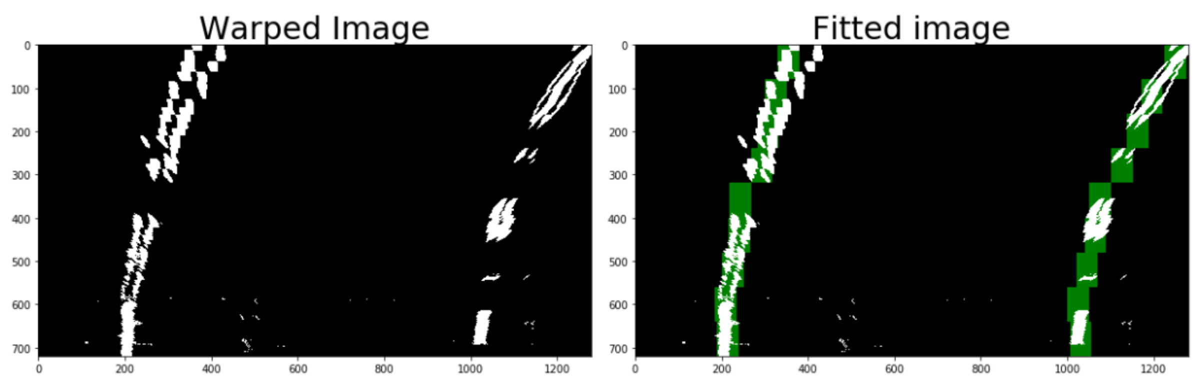
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.



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

I applied a convolution *find_window_centroids()*, which maximized the number of hot pixels in each window.

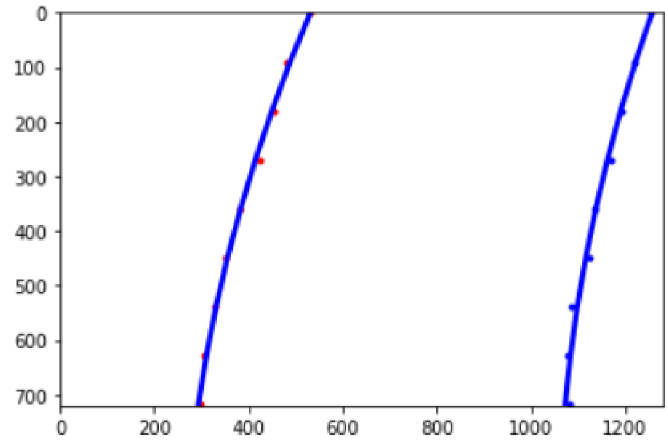
Finally I combined the left and right points over the previous image.



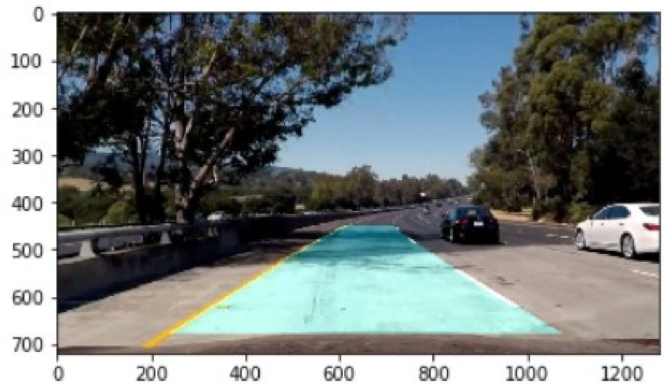
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I used the left and right points found with the *find_window_centroids()* function and was able to define *curvature()* which returned the center of the image as well as the left and right Radius of curvature using the following equation:

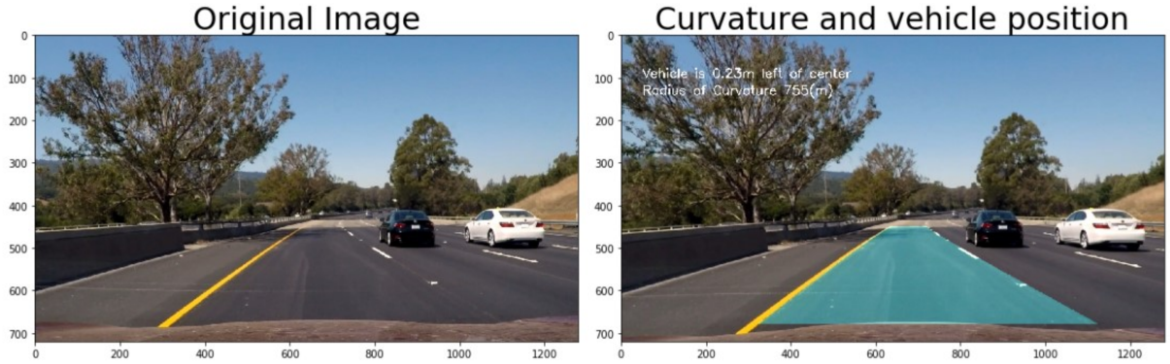
$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$



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



Also, below is output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position



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!).

Attached with the package uploaded.

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?

I faced number of issues at initial stages but things became clearer as and when I debugged the issue and followed step by step approach. I think pipeline is working pretty decently and robust for the video input. There were inconsistencies observed for the challenge video which I hope will work on some time and resolve. The pipeline seems to fail for the harder challenge video. This video has sharper turns and at very short intervals. Some of the things to improve are : better perspective transform and refine pipeline for varied environmental conditions.