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.

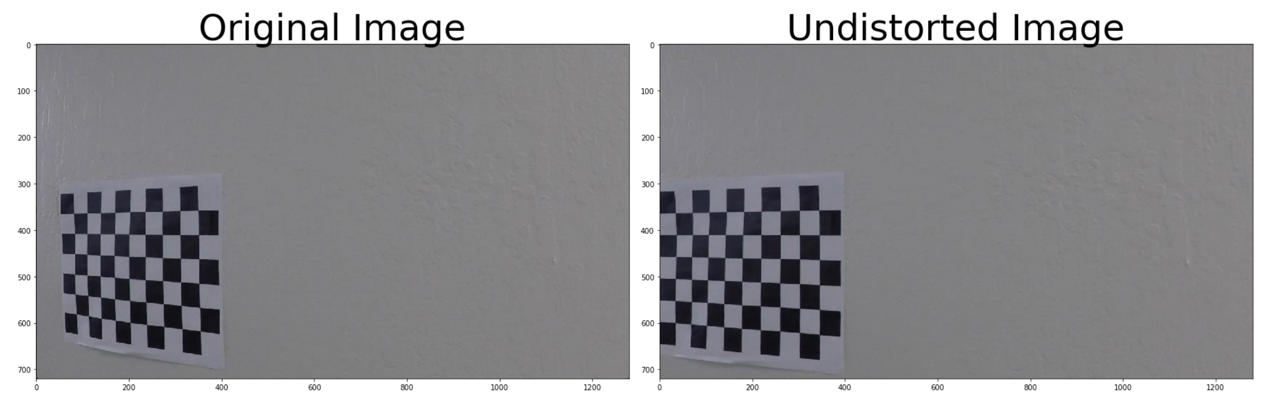
Camera Calibration

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 "adv lane project.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 when 'cv2.findChessboardCorners()' function is used.

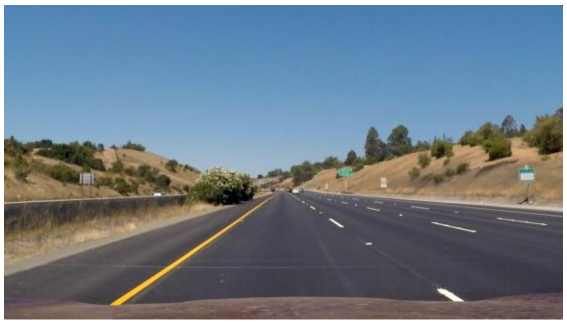
I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



Pipeline (single images)

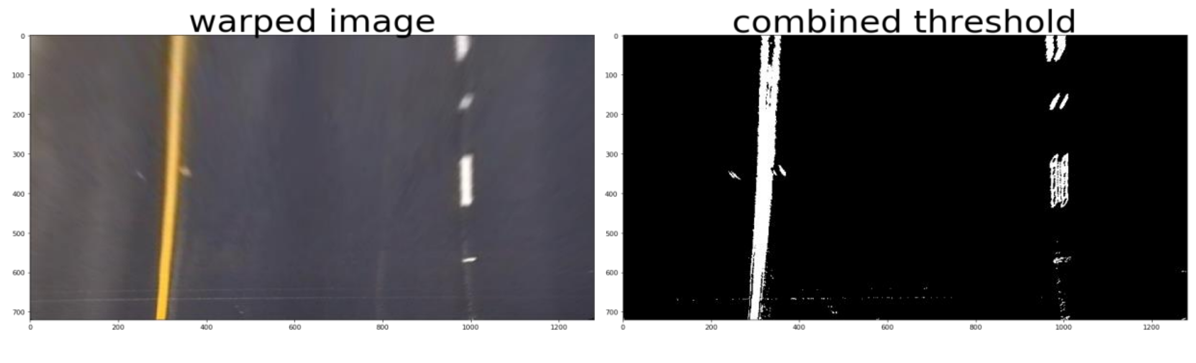
1. Provide an example of a distortion-corrected image.

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 binary image. Provide an example of a binary image result.

I used a combination of color, direction and gradient thresholds to generate a binary image (thresholding steps in cell 11). Here's an example of my output for this step.

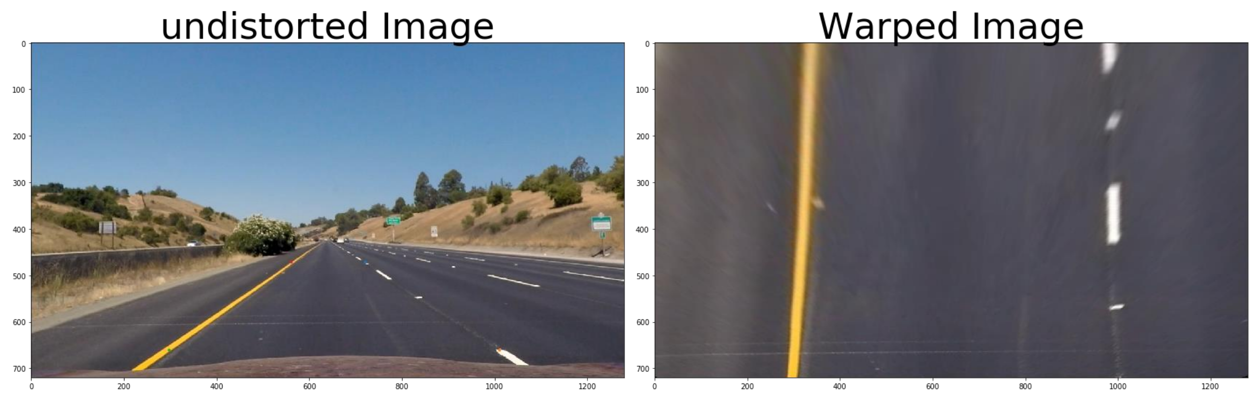


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `warp()`, which appears in 4th cell. The `warp()` function takes as inputs an image (`img`). The source and destination points are defined inside this function. I chose the hardcode the source and destination points in the following manner:

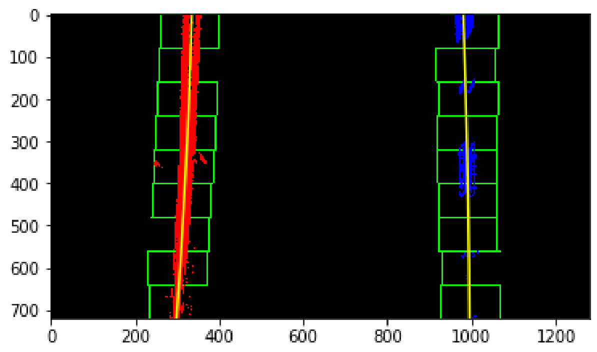
```python
src = np.float32([(720,470),(1010,660),(297,660),(560,470)])

dst = np.float32([(1000,100),(1000,720),(300,720),(300,100)])
```

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?

After the perspective transform and thresholding, a histogram is drawn based on the frequency of lane pixels in the image to find the starting position of the lanes. Then using the sliding window function in cell 13 of ‘adv lane project.ipynb’ in which the lane pixels are detected and a polynomial is fitted using np.polyfit() to obtain a smooth curve. After the first frame, the pixels are searched near the previous frame polynomial for computational advantage.



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.

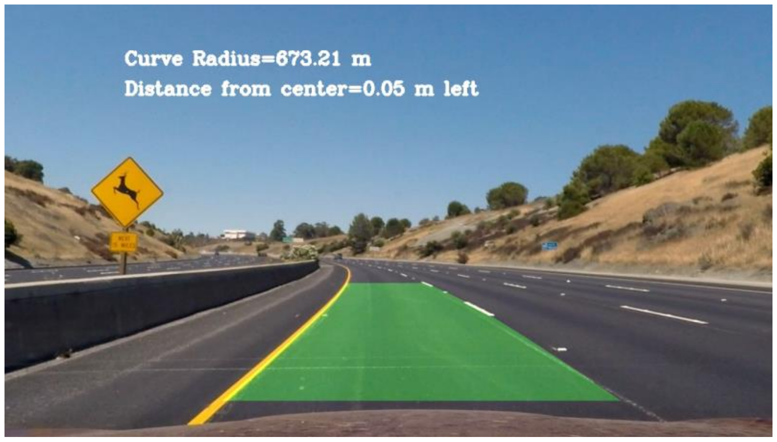
In cell 15, I have clearly mentioned the radius of curvature calculation. The pixel space is converted to real world space using ym_per_pix = 30/720 & xm_per_pix = 3.7/700 and estimated the radius of curvature based on the formula:

$$R_{curve} = \frac{\left[1 + \left(\frac{dx}{dy}\right)^2\right]^{\frac{3}{2}}}{\left|\frac{d^2x}{dy^2}\right|}$$

After obtaining the curvatures of both the lanes, average value is taken.

The lane midpoint is also calculated to obtain the camera position from the lane centre.

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 is the link to my project result video.

https://www.youtube.com/watch?v=i1B5PEJ66r4 (./adv_lane_project.mp4)

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 difficult part is try different color channels, threshold values to obtain a proper binary image. The pipeline may fail if we are following a bright white car among dull white lane lines. A dynamic thresholding based on the lighting in the image can provide best results.