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.

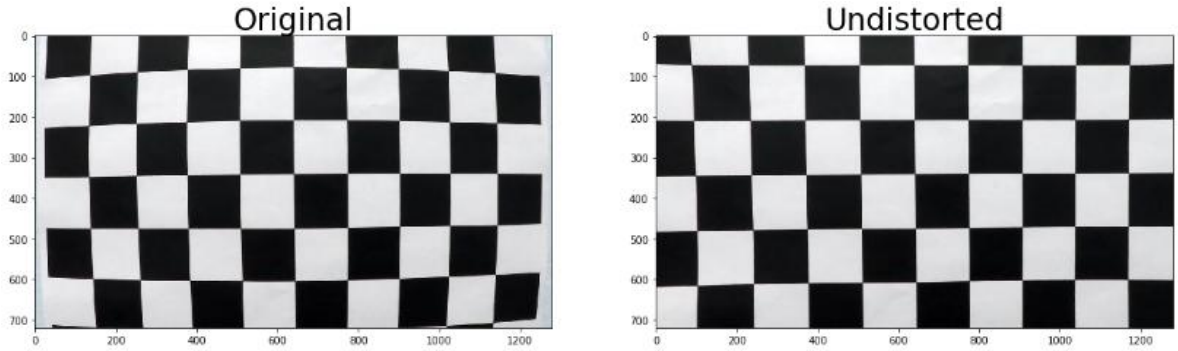
Warp the detected lane boundaries back onto the original image.

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

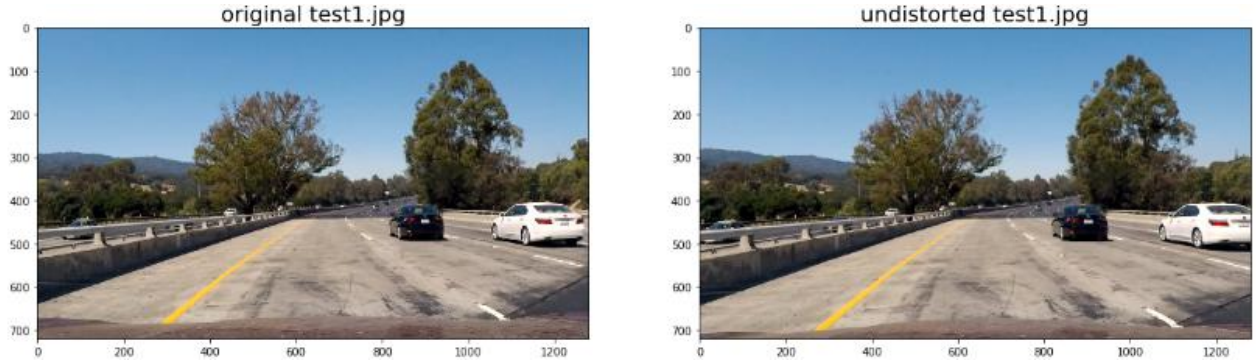
- corrected calibration image.
- Read calibration images
- Generate object points
- Find image points with `cv2.findChessboardCorners`
- Calibrate the camera and obtain distortion coefficients with `cv2.calibrateCamera`



Pipeline (single images)

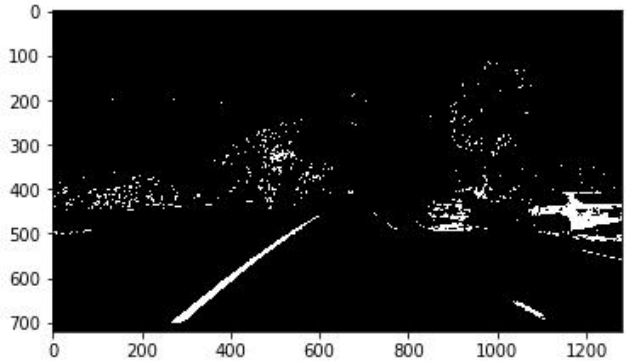
1. Provide an example of a distortion-corrected image.

- Apply `cv2.undistort` with the camera matrix and distortion coefficients obtained.



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 convert image to HLS with `cv2.cvtColor`
- Choose the third channel of HLS image
- Combine the binary threshold [160, 255] to generate a binary image

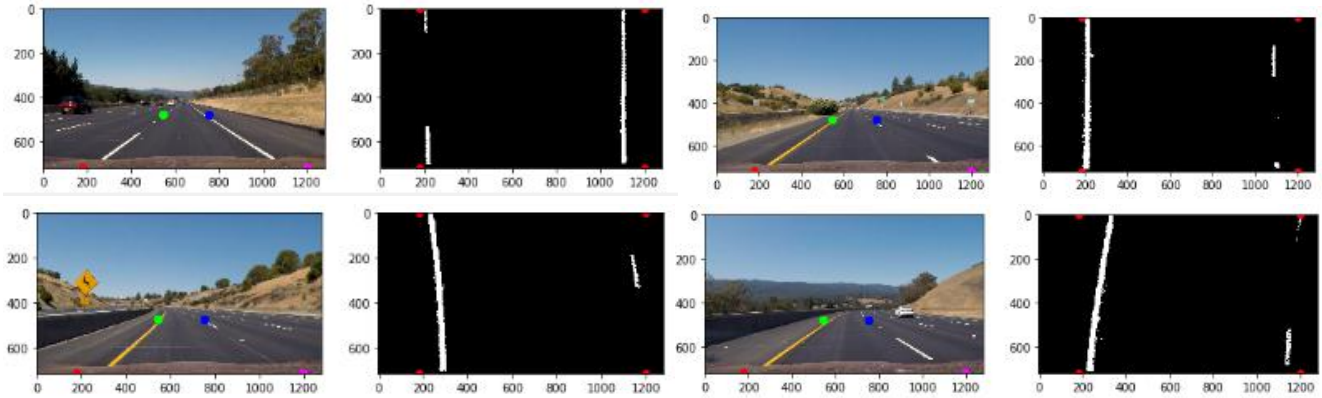


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

- Select the region of interest in source images.
- Set the destination region of transformed images
- Source points and destination points are:

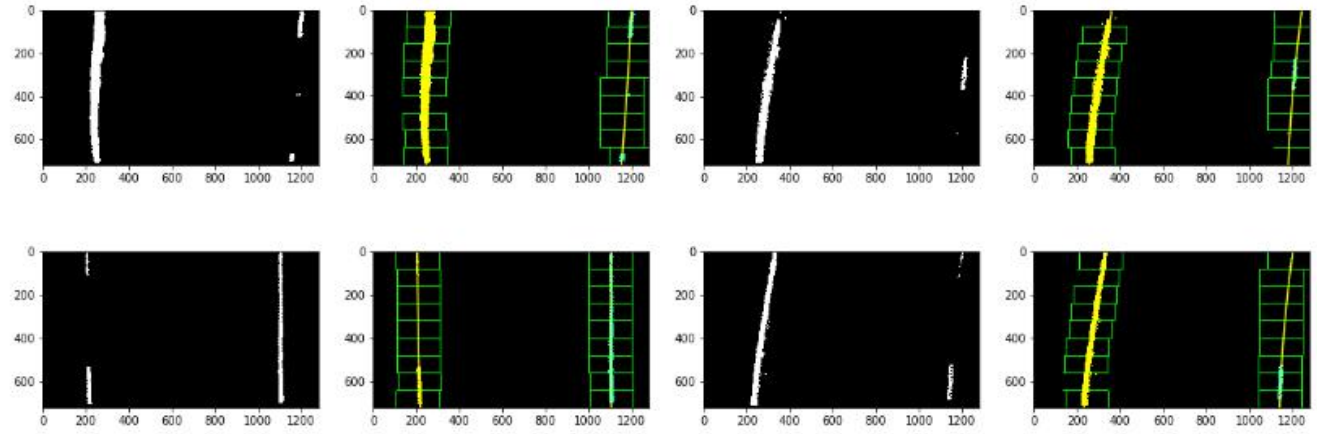
```
corners = []
corners.append([180,720])
corners.append([547,480])
corners.append([755,480])
corners.append([1200,720])
dsts = []
dsts.append(corners[0])
dsts.append([corners[0][0], 0])
dsts.append([corners[3][0], 0])
dsts.append(corners[3])
```

- code the source and destination polygon coordinates and obtain the matrix `M` that maps them onto each other with `cv2.getPerspective`
- Warp the image to the new birds-eye-view perspective with `cv2.warpPerspective` and the perspective transform matrix `M` we just obtained
- Examples of source images and transformed images

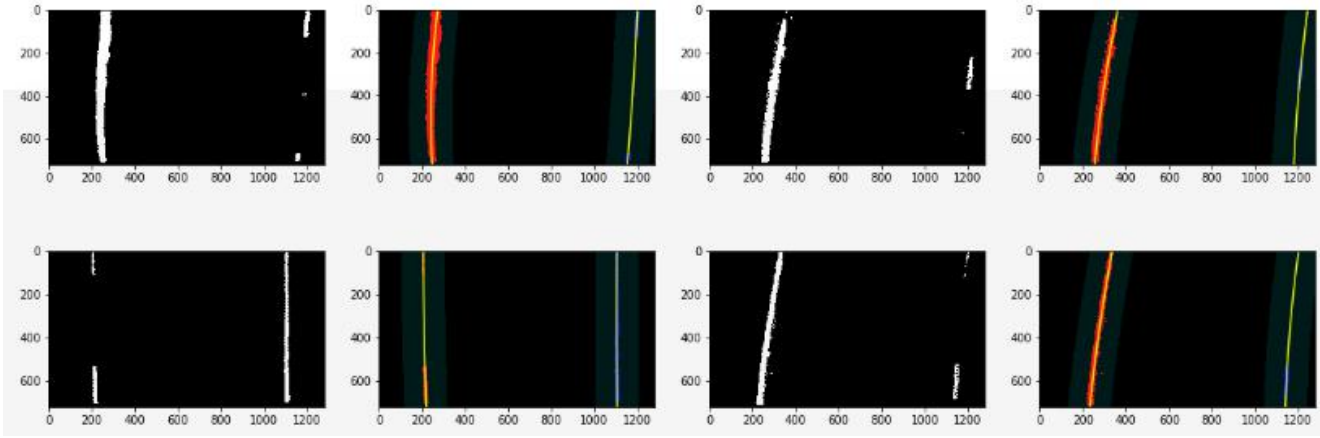


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

- Divide the image into 9 horizontal strips (steps) of equal height.
- For each step, take a count of all the pixels at each x-value within the step window using a histogram generated from np.sum.
- Find the peaks in the left and right halves (one half for each lane line) histogram with `np.argmax`.
- Get (add to our collection of lane line pixels) the pixels in that horizontal strip that have x coordinates close to the two peak x coordinates.



- Fit a second order polynomial to each lane line using np.polyfit.

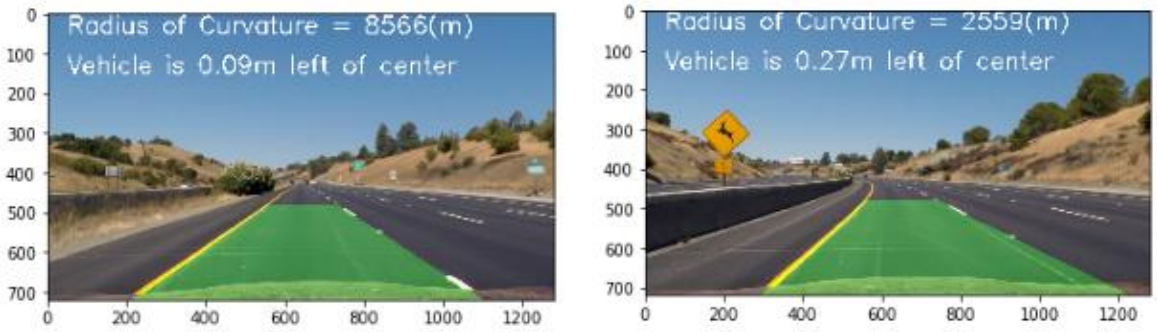


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

- I used the example code in Measuring Curvature section to calculate the left and right curvatures. Then I choose the average of left and right curvatures as my center curvature to control steering angles in the future.
- Code:

```
# Calculate the new radii of curvature
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
# calculate curvature, center and distance to the center
    curvature = (left_curverad + right_curverad) / 2
    center = (leftX[0] + rightX[0]) / 2
    xm_per_pix = 3.7/700 # meters per pixel in x dimension
    CenterDist = (center - 640) * xm_per_pix # 640 is the center pixel of image
```

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!).

- Link: https://github.com/hyo009/CarND-Advanced-Lane-Lines-P4/blob/master/project_output.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?

- Noise interference under trees will result in incorrect lane curves.
  Solution: increase lower threshold bound to decrease noise points in transformed images.
- The radius of curvature changes jumpily and sometimes my pipeline cannot detect right lanes.
  Solution: When I cannot detect the right lane, I can use the previous picture's right lane parameter as the previous one since there are little difference between two continues pictures. For the curvature problem, I can calculate the average radius of curvature of several pictures to make the change of radius more smooth.