Advanced Lane Finding

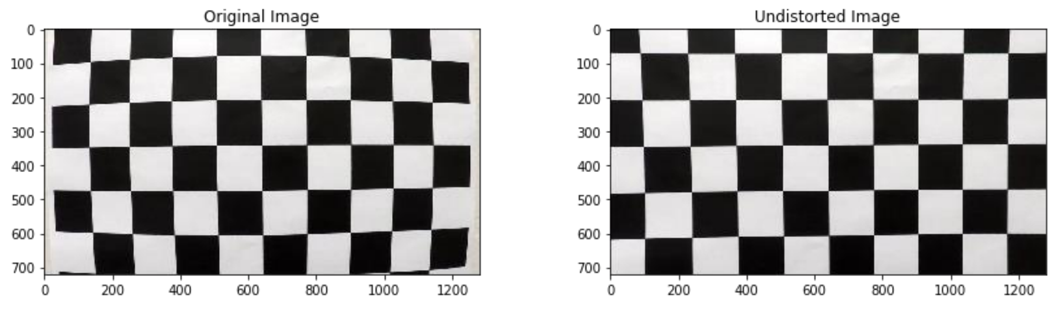
The goals / steps of this project are the following:

- 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.

In this step, I prepared objpoints and imgpoints which represent position of points in real world space and image plane, respectively. Since we are handling with a chessboard, objpoints can be defined as a set of integers, like (0,0,0), (1,0,0), and so on. As for imgpoints, I used an OpenCV function which detects chessboard corners in an image. By finding the relationship between objpoints and imgpoints, I can obtain the camera matrix and distortion coefficients which are unique to the specific camera that I am using. Using these parameters, camera images can be undistorted like below.



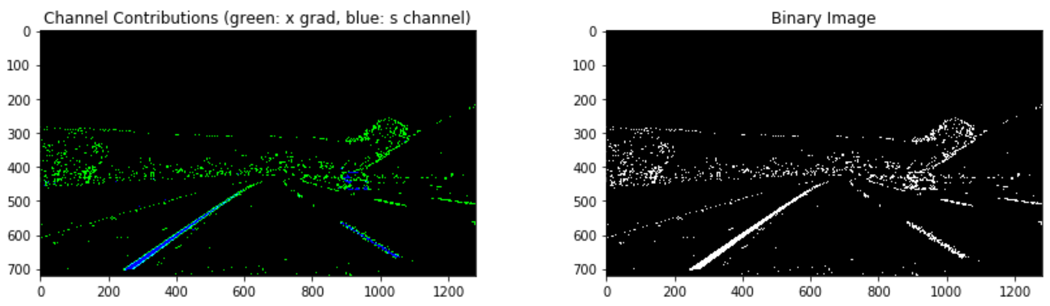
Pipeline (single images)

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

Here is an example of a distortion-corrected image using the parameters obtained by camera calibration.

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.

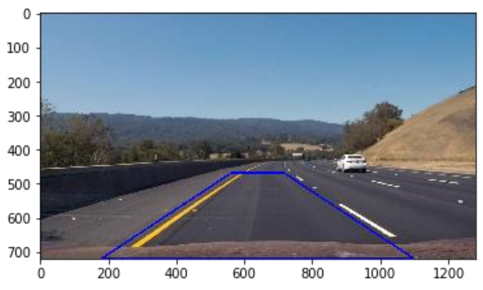
Inside the function "apply_threshold()" in "./P2.ipynb", I applied S channel and x gradient thresholding techniques. Image on left shows each contribution, and image on right shows the binary image result.

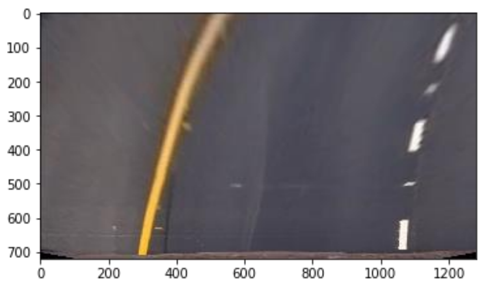


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

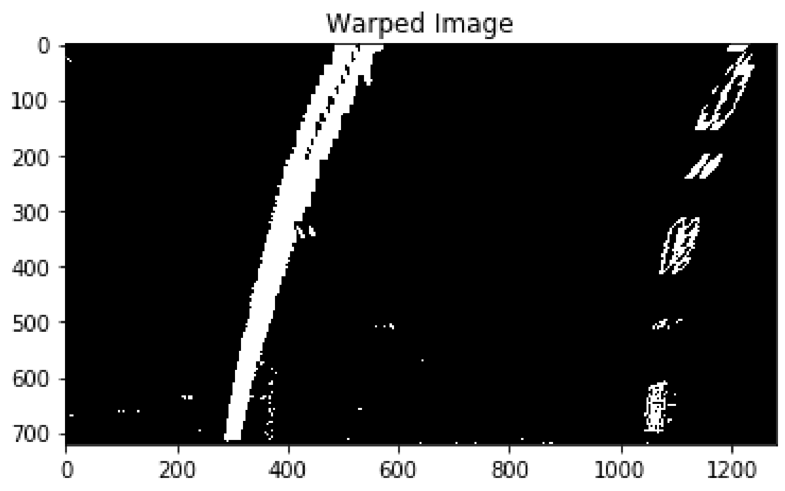
Inside the function "apply_perspective_transform()" in "./P2.ipynb", I performed a perspective transform to an image. In order to define a perspective transform, I first need to think about where straight lane lines appear in the original image, and where I want them to be in the transformed image. These can be described in the form of source points and destination points. I defined them in the function “get_perspective_params()” in “./P2.ipynb”, then double-checked their values in the function “check_perspective_params()”.

Here is an example of a perspective transform. The blue trapezoid on the left is the boundary around source points.





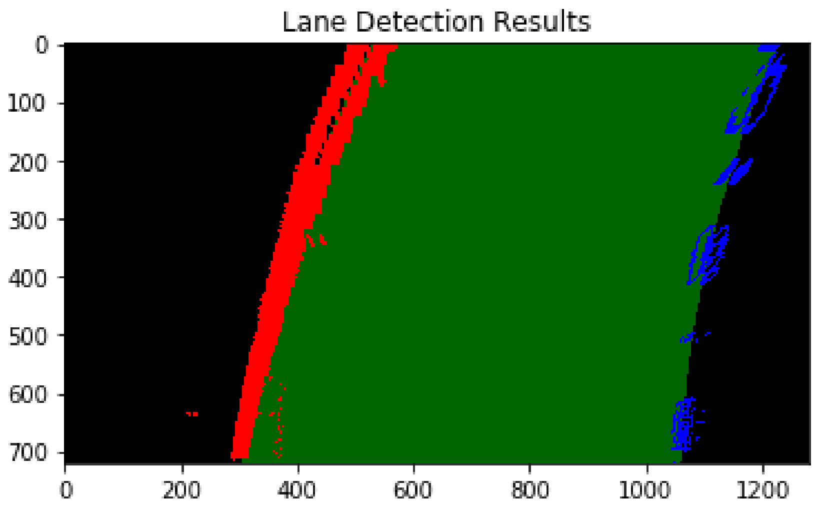
An example of a warped binary image is shown below.



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

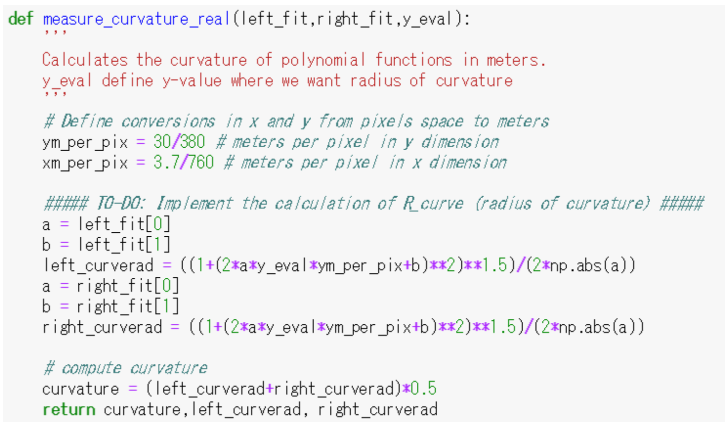
Inside functions “find_lane_pixels()” and “fit_polynomial()” in “./P2.ipynb”, I identified lane-line pixels and fit their positions with a polynomial. Lane lines were detected by first finding peaks in a histogram and then performing a sliding window search. Then each detected lane points were fit to a parabola to give a concrete solution to the lane line position.

Image below shows the lane detection result for left lane line (red) and right lane line (blue). Also, the region in between the left and right polynomials is filled in green.



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.

Inside function “measure_curvature_real ()” in “./P2.ipynb”, I calculated lane curvature for left and right lanes using polynomial fit data and lane curvature formula. I then computed the curvature of the lane by calculating the mean of left and right curvatures.



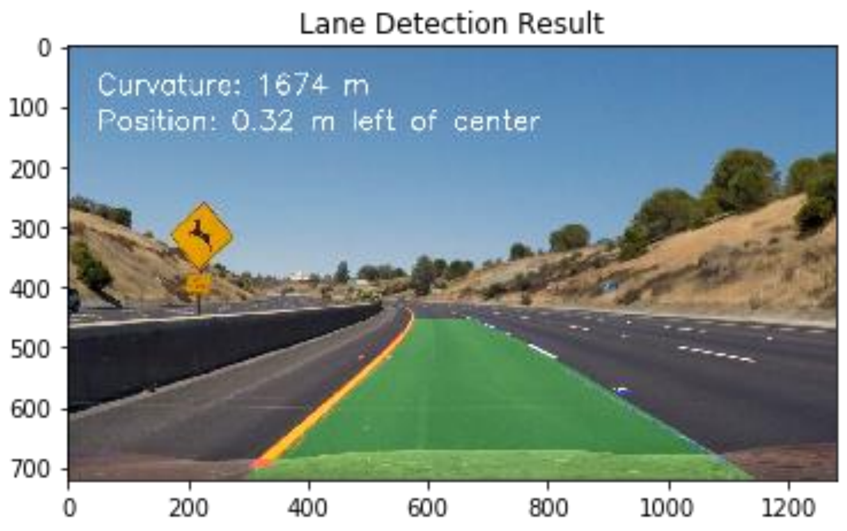
```
def measure_curvature_real(left_fit,right_fit,y_eval):
    '''
    Calculates the curvature of polynomial functions in meters.
    y_eval define y-value where we want radius of curvature
    '''
    # Define conversions in x and y from pixels space to meters
    ym_per_pix = 30/380 # meters per pixel in y dimension
    xm_per_pix = 3.7/760 # meters per pixel in x dimension

    ##### TO-DO: Implement the calculation of R_curve (radius of curvature) #####
    a = left_fit[0]
    b = left_fit[1]
    left_curverad = ((1+(2*a*y_eval*ym_per_pix+b)**2)**1.5)/(2*np.abs(a))
    a = right_fit[0]
    b = right_fit[1]
    right_curverad = ((1+(2*a*y_eval*ym_per_pix+b)**2)**1.5)/(2*np.abs(a))

    # compute curvature
    curvature = (left_curverad+right_curverad)*0.5
    return curvature,left_curverad, right_curverad
```

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

Here is an example of lane lines plotted back onto the original perspective. The lane lines are put into the correct perspective by using the inverse perspective matrix, Minv.



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 link to my video result.

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 time-consuming part of implementing this project was fine-tuning parameters such as color and gradient thresholds. It took a lot of trial and error to find good parameters that work for a wide variety of input images. The pipeline will most likely fail when the road condition is not good and white dashed lines become very unclear.