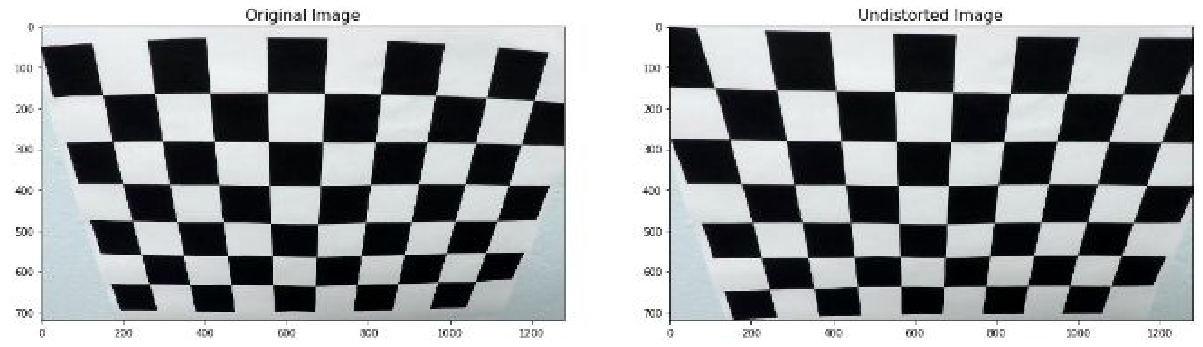
CarND Term 1 Project 4

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 "project4.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)

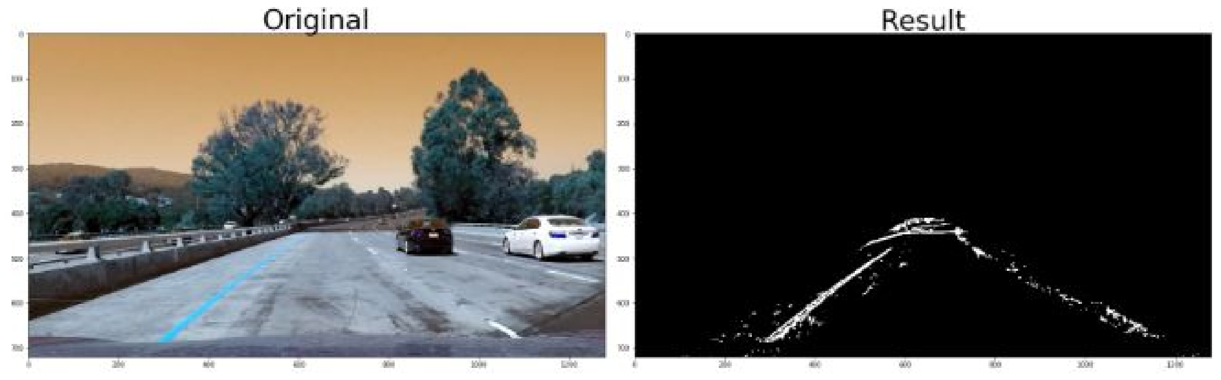
1. Provide an example of a distortion-corrected image.

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 in code cell 3. More specifically, I convolve the image with a gaussian to smooth it, convert to HLS and grayscale, then perform thresholding and apply sobel filters in the x and y directions. I then combine the outputs of these operations and apply a ROI mask.

Here's an example of my output for this step.



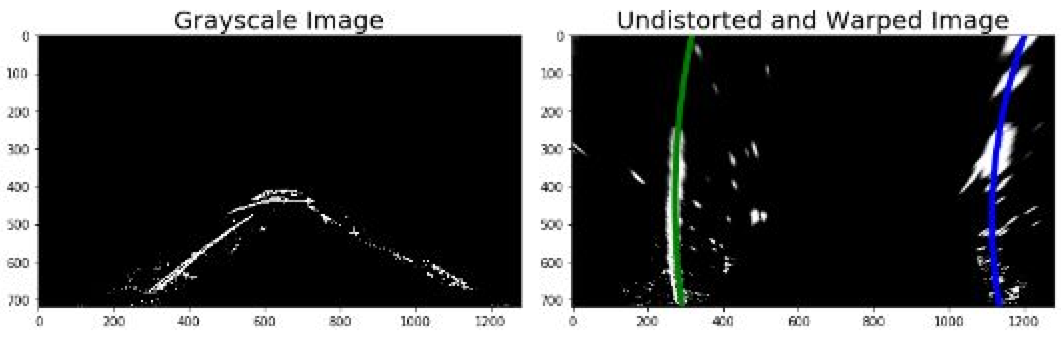
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `warp()`, which appears at the top of cell block 4.

I chose the hardcode the source and destination points in the following manner:

```
    # offset from image corners to plot detected corners

    offset1 = 200 # offset for dst points x value

    offset2 = 0 # offset for dst points bottom y value

    offset3 = 0 # offset for dst points top y value



    # grab outer 4 detected corners in src

    src = np.float32(area_of_interest)



    # choose points that make the warping look nice in dst image

    dst = np.float32([[offset1, offset3],

                      [img_size[0]-offset1, offset3],

                      [img_size[0]-offset1, img_size[1]-offset2],

                      [offset1, img_size[1]-offset2]])
```

Outputs from this operation are:



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 I begin by looking for peaks in the image histogram. This is done using multiple histogram windows. Once points to look for are identified their coordinates are added to an array that is then used to fit lines to. This 2nd order polynomial then defines the lines that are drawn onto the lane lines. The functions i defined for this are called find_peaks(),fit_lanes() and find_lanes().

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.

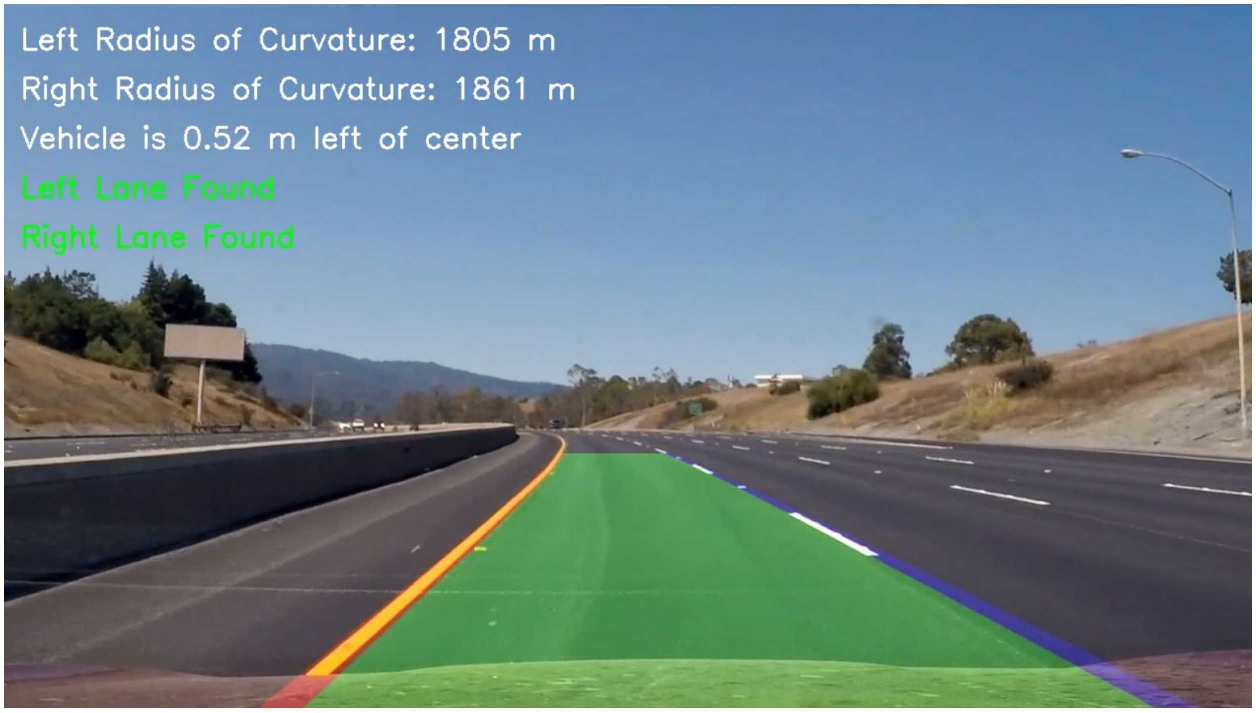
This is again in the 4th cell block of the notebook. It is carried out using the functions find_curvature() and find_position(). The radius of curvature was calculated using the 2nd derivative definition:

```
curverad = ((1 + (2*fit_cr[0]*y_eval + fit_cr[1])**2)**1.5) \

                             /np.absolute(2*fit_cr[0])
```

The position of the vehicle was then determined by taking the min and max position to the left and right of the center of the image where the y pixel value is greater than 700 (near the hood of the car). The position was then taken to be half the distance between these two points, and then converted to world coordinates using the scale factor 3.7/700.

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

This is again in code cell 4,using the draw_poly() function. Here is an example of my result on a test image:


Left Radius of Curvature: 1805 m
Right Radius of Curvature: 1861 m
Vehicle is 0.52 m left of center
Left Lane Found
Right Lane Found

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 the video: https://youtu.be/IFjkeANu790

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 found this still didn't do very great under very shady conditions or with weird lighting. I think if I had done more operations using just the saturation channel things could have improved.