Advanced Lane Finding Project

The goals / steps of this project are the following:

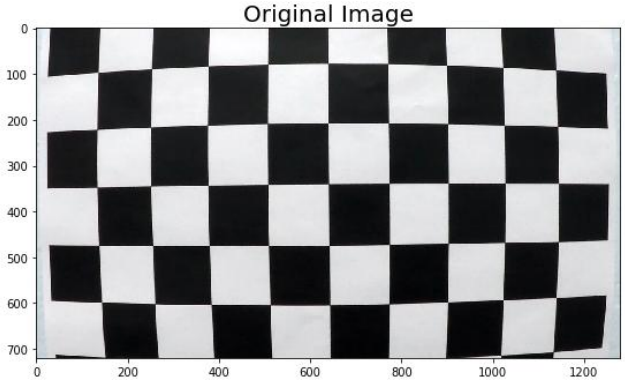
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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.

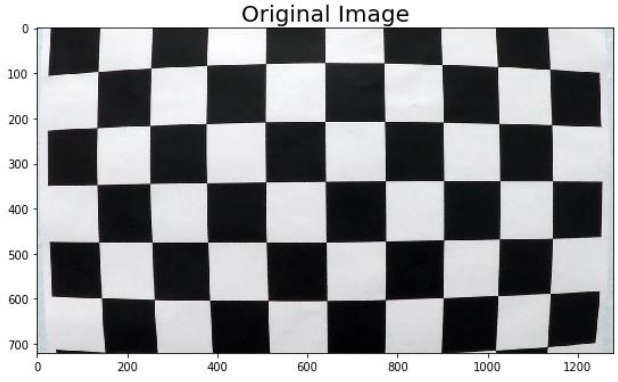
Rubric Points

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 (Camera_calibration.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:

Original | Undistorted | Original-Undistorted

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

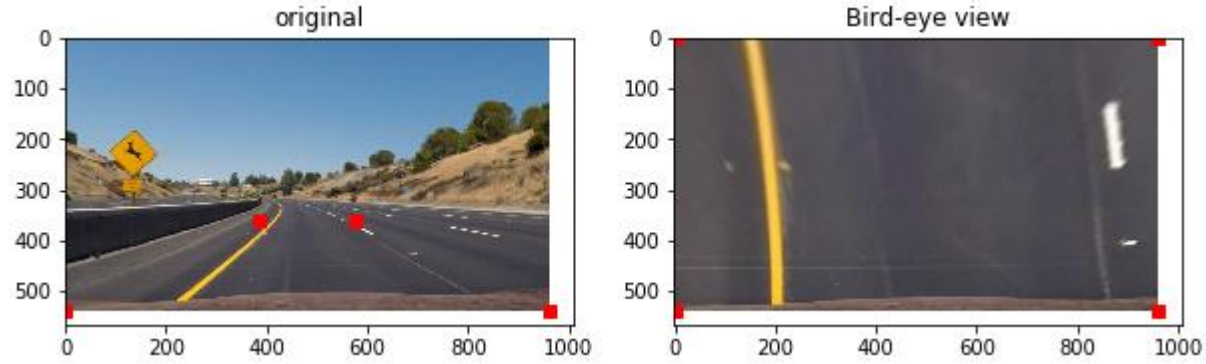
The code for my perspective transform includes a function called `warper()`, which appears in the 9th code cell of the IPython notebook). The `warper()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

```Jupiter notebook ( Section 9 - 20 Advanced_Lane_Lines_EDA.ipynb )
src =np.float32(
[[cbl_window,hb_window],
[cbr_window,hb_window],
[ctr_window,ht_window],
[ctl_window,ht_window]])
dst = np.float32 (
[[0,img_size[0]],
[img_size[1],img_size[0]],
[img_size[1],0],
[0,0]])```

This resulted in the following source and destination points:

| Source | Destination |
|:---|:---|
| 0, 540 | 0, 540 |
| 960, 540 | 960, 540 |
| 576, 360 | 960, 0 |
| 384, 360 | 960, 0 |
```

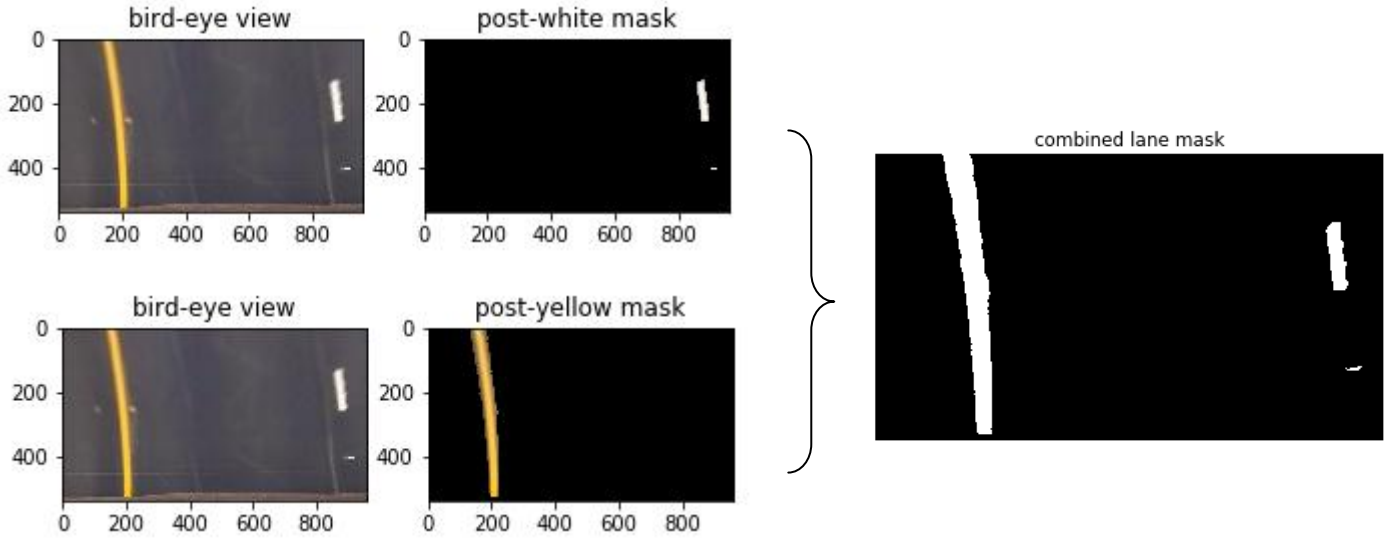
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.



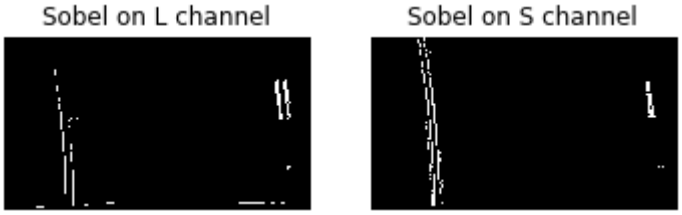
3. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines # through # in `Advanced_Lane_Lines_EDA.ipynb`). Here's an example of my output for this step. (note: this is not actually from one of the test images)

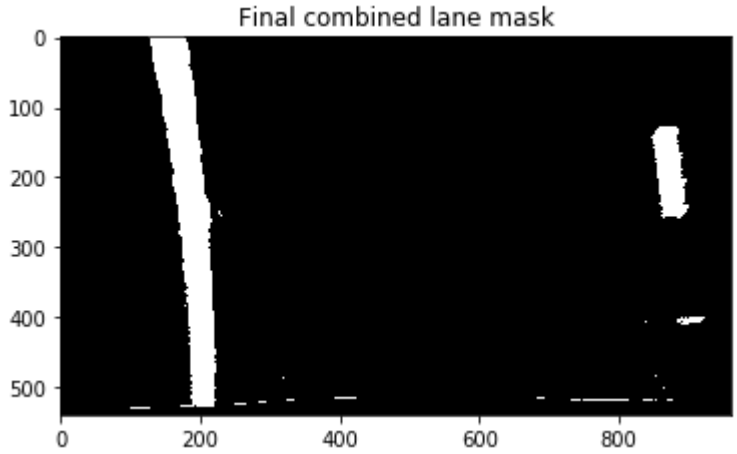
Convert to HSV colorspace and applying color masks:



Apply sobel filters to get potential line/edges:



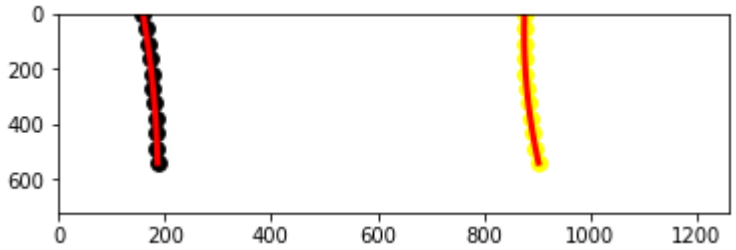
Combine Sobel filters and color masks:



Apply window to remove any markings or features that could be due to other artifacts in the image.

4. 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 did this in lines Section 41 to 42 in my code in `Advanced_Lane_Lines_EDA.ipynb `
Apply polynomial regression to compute the left and right lanes



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

I implemented this step in lines Section 43 in my code in `Advanced_Lane_Lines_EDA.ipynb `. Here is an example of my result on a test image:

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 "project_video.mp4".

Discussion

In this project, I searched lane lanes from the image. The procesure is as follows. First, I calculated camera matrics and distortion coefficients to undistort image. Then I use several techniques to find lane lines. Firstly, I transform the color space from HSV to find white and yellow lines. Then I used sobel operation to detect edges. Finally I put it together and then used gaussian blur to find the lane lines easily.

Project 1 was unable to detect shadows, so it was very good to overcome shadows using this method. After checking the method with the image, we could have pipeline to apply to the actual motion picture.