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.

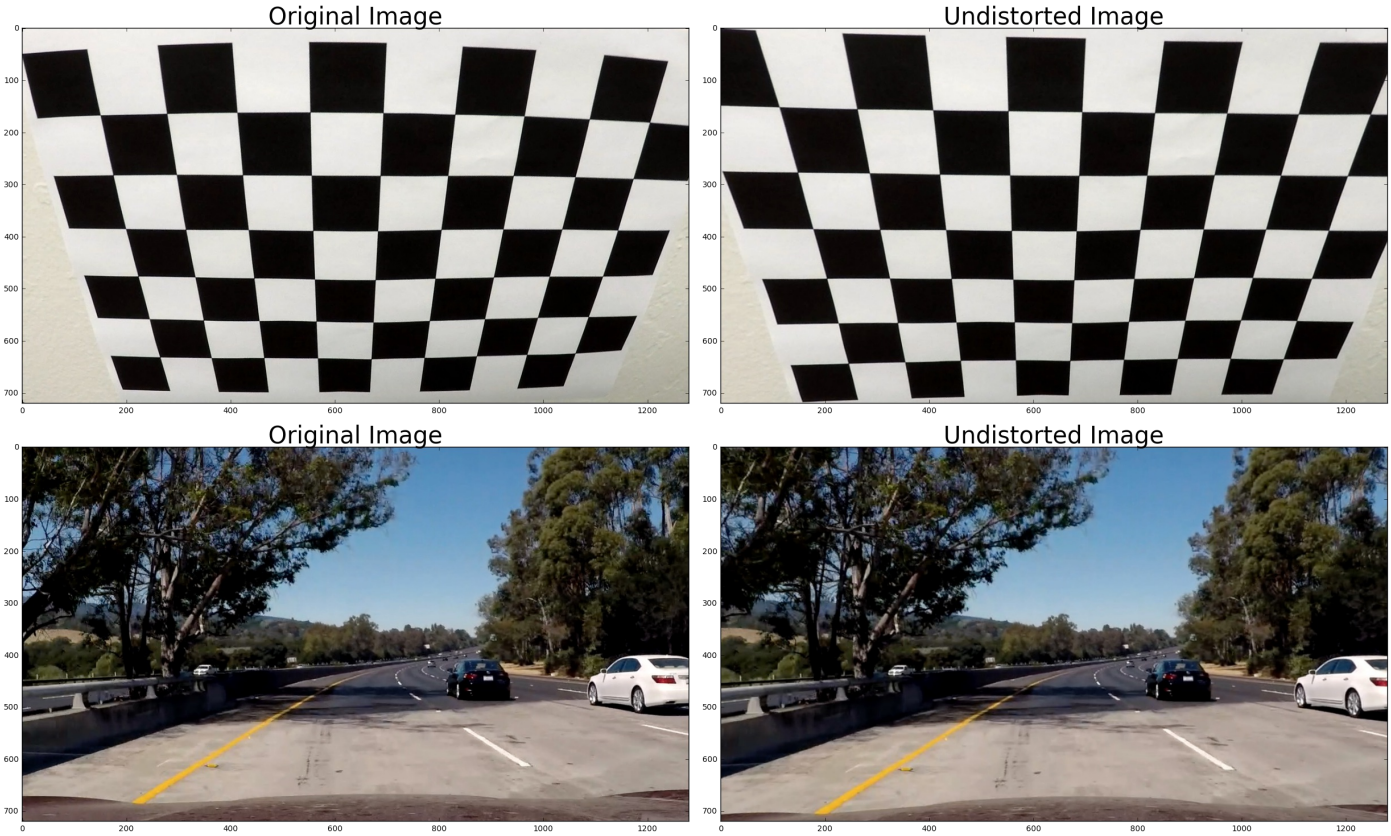
Rubric Points

1. Camera Calibration and Image Undistortion

The code for camera calibration and image undistortion, based on openCV functions `cv2.calibrateCamera` and `cv2.undistort`, is contained in the second and third code cells respectively, of the project notebook file `Advanced_lane_detection.ipynb`. The parameters calculated in the camera calibration step, `mtx` and `dist`, are required for the image undistortion step, and once calculated, are fixed and used throughout the project for image undistortion.

In the following images, the effect of image undistortion is shown on one

calibration checkboard image and one test image, respectively.



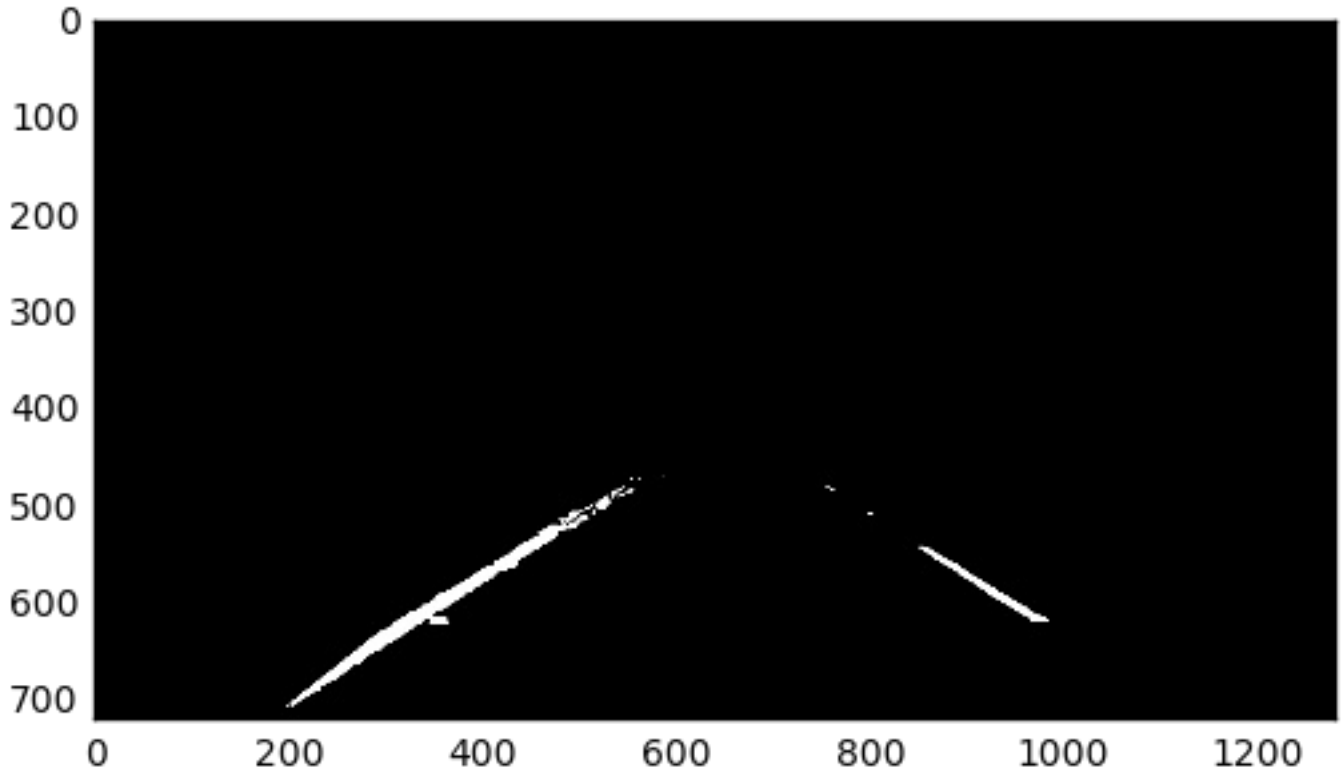
2. Image preprocessing using Mask and Color Thresholds

This step is performed using the function,

```
preprocess(image, mask_vertices, s_thresh, r_thresh, h_thresl
```

the code for which is contained in the eighth code cell of the notebook. First, a mask is applied to the image to isolate the region of interest. The `mask_vertices` selected are the same as those used in the first project of CarND. Next, the color thresholds are applied in the H,S and R channels, and combined - the first two with `and` operation, and the third with `or` operation. S and R thresholds are exactly the same as used in the example in the course lecture, while the H threshold needed to be tuned a little to work well on all the test images.

The resulting binary image obtained after this preprocessing step involving masking and thresholding on the example test image is shown below:

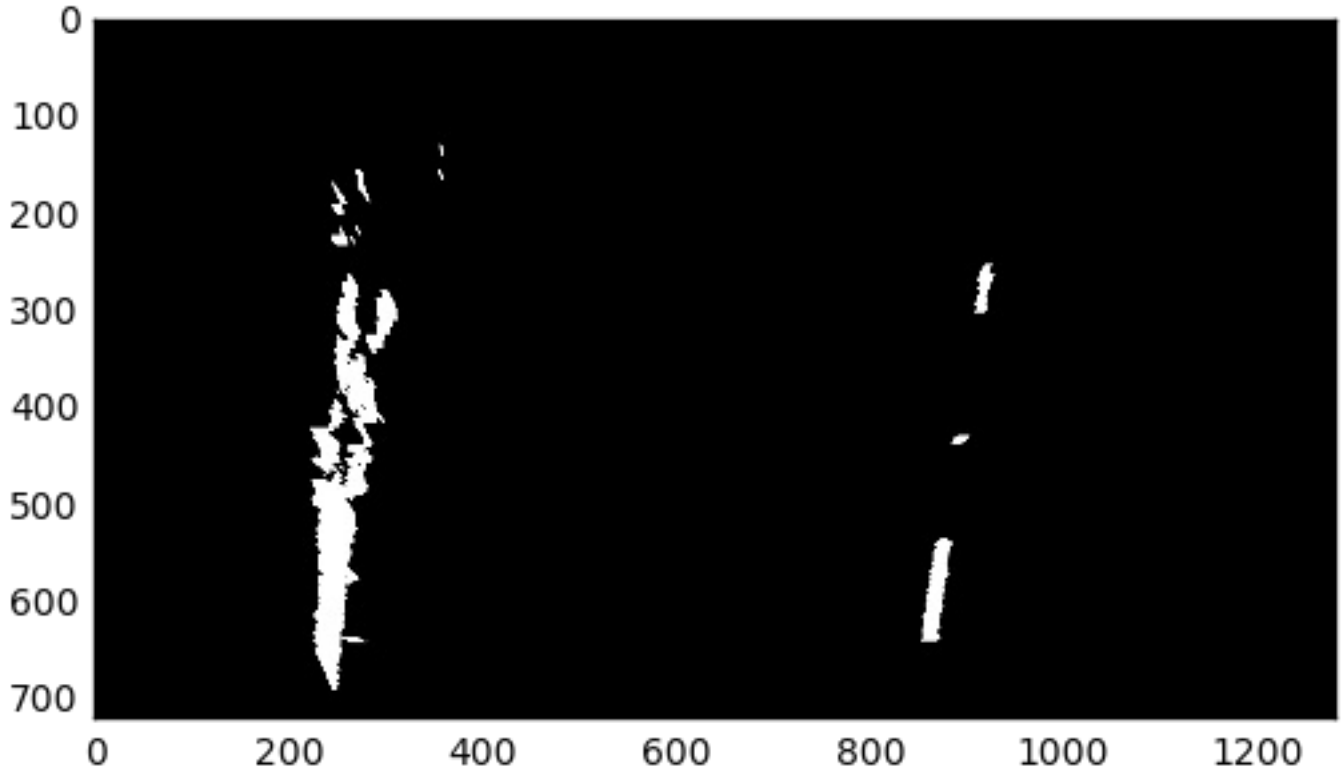


3. Perspective Transformation

The perspective of the preprocessed binary image is changed to that of bird's eye view using the openCV function `cv2.getPerspectiveTransform()` in the eleventh code cell of the project notebook. Apart from the image, this function takes source (`src`) and destination (`dst`) points as arguments, which are hard-coded as follows:

Source	Destination
258, 679	258, 679
446, 549	258, 549
837, 549	837, 549
1045, 679	837, 679

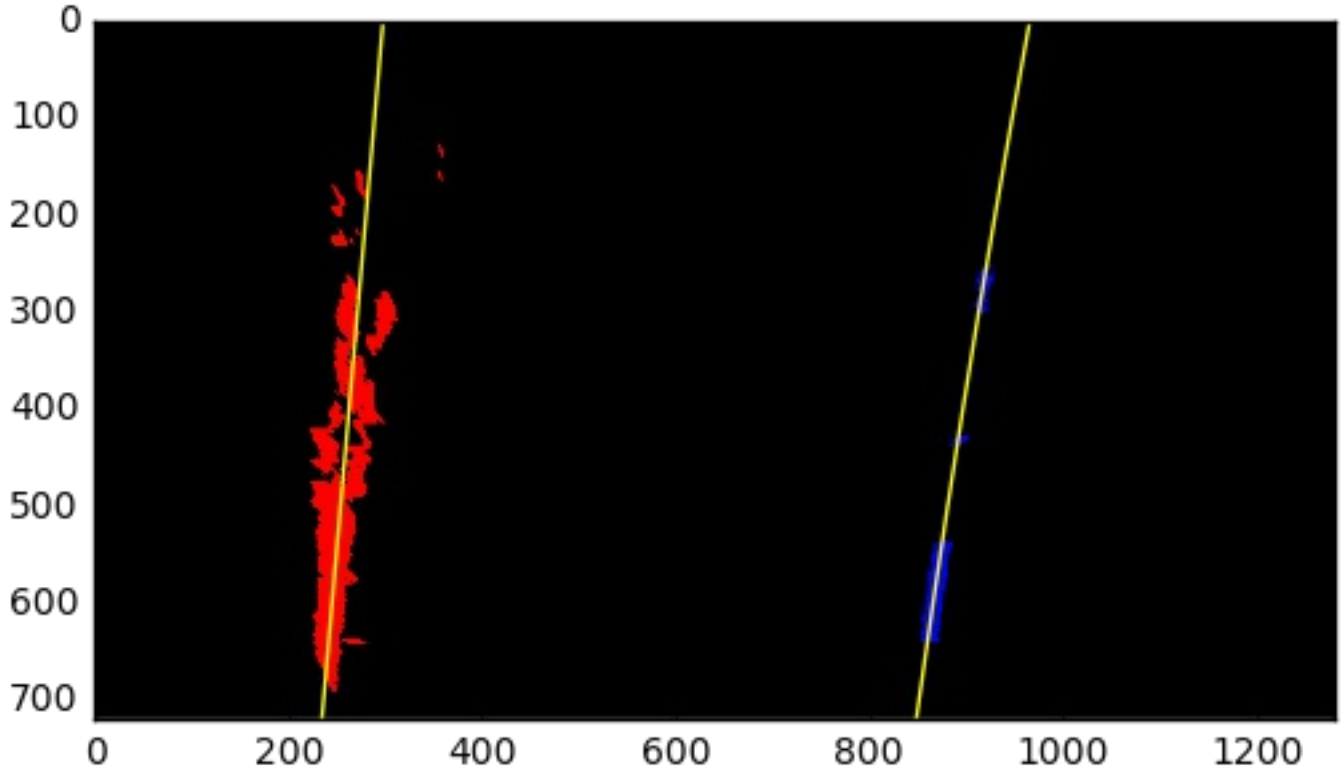
I verified that my perspective transform was working as expected by plotting the warped example test image and observing that the lines appear approximately parallel in the warped image.



4. Lane Detection using Sliding Window Search

The code for this part is in the thirteenth code cell of the project notebook, and consists of three functions. In the first function, `find_lanes()`, a histogram of column pixels in the lower half of the warped image is computed, to find the starting position for the left and right lane lines. From this starting positions, two windows are slid vertically upwards along the putative centre of each lane line, and the pixels detected within the sliding window are concatenated and returned as arrays. The next function, `fit_polynomial()` takes these pixel arrays as arguments, and returns two second order polynomial curves fitted to the corresponding lane pixels using `np.polyfit()` method. The third function, `draw_curves()`, takes these polynomials as arguments, and generates x and y values for plotting.

These functions are applied in series to the warped example test image to get the following visualisation:



5. Measuring Curvature and Deviation from Center

The function `rad_curvature()` in the sixteenth code cell of the notebook calculates both the radius of curvature and the deviation from the center in meters. Since all the preceding analysis has been in pixel space, we had to first compute the scaling factors for converting from the pixels space to meters in real-world space.

6. Highlighting Lane Lines in the Original Image

In the `draw_lane()` function defined in the eighteenth code cell, the detected lane is first highlighted in the warped image, and the resulting image is then inverse transformed from bird's view perspective to the original perspective.

The function is applied to the warped example test image to get the

original image with the lane highlighted. Additionally the information pertaining to the radius of curvature and deviation from center is also printed to get the following output image:

Pipeline (video)

The above steps are combined into a pipeline in the function `lane_detection()` in the twentieth code cell of the notebook, which is then applied to each frame of the project video in the twenty-second code cell.

Once you run this code, you will find here 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?

Most of the code implemented in this project is a tweaked version of the code from the course material. The main challenge was to tune the threshold parameters to obtain pixels sufficiently isolated and prominent to be detected in the subsequent steps. The gradient thresholds seemed to add some noise to the images, so I decided to exclude them completely. A combination of color thresholds in the H, S and R channels with the same thresholds as in the course material worked well in most images, but not all. In some images, the shadow pixels played havoc. I decided to tune the H thresholds, keeping the other two constant. After a little tuning, I got pretty good results on all the test images and the project video as well.

I'm not sure how robust these thresholds would be to the extreme lighting conditions, other than those encountered in the project video. One reason is that I could not test my code on the challenge videos since I have not yet implemented the smoothing of lane across many frames of the video, which would have prevented getting error because of no lane line getting detected. This is definitely my next task, once I'm free of deadline pressure. Perhaps, I will see the utility of gradient thresholds in the challenge videos, that is, once I implement the smoothing and get the code working. For the moment, I'm satisfied with the result I got on the project video.