Advanced Lane Finding

Udacity Self-Driving Cars Project

Objective:

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.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Use color transforms, gradients, etc., to create a thresholded binary image.
- 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.

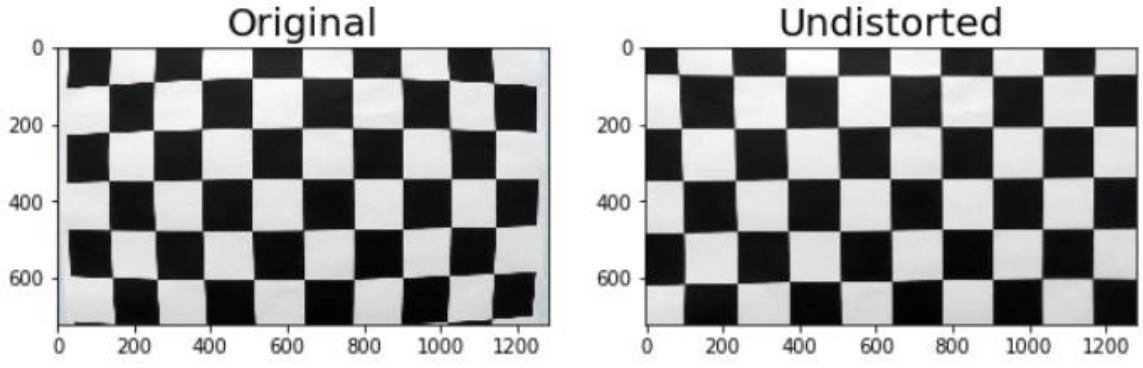
Camera Calibration:

1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

   The code for this step is contained in the cell 2 of the attached IPython notebook.

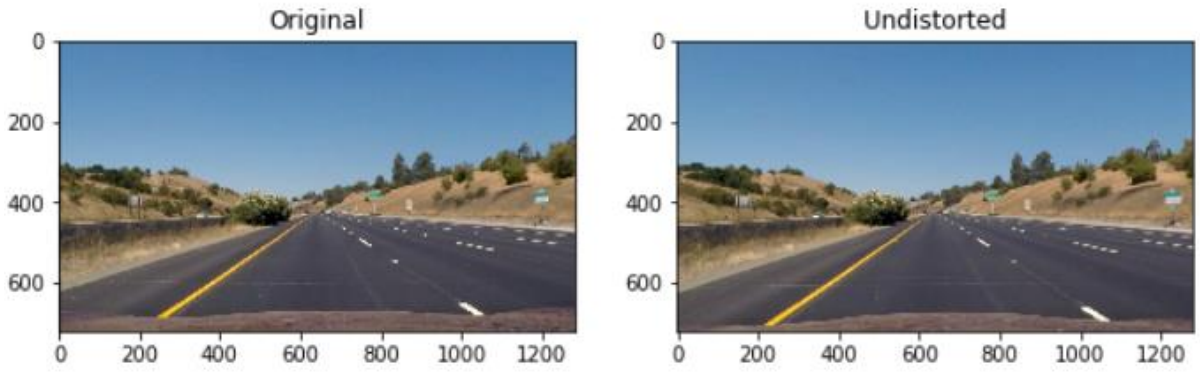
   I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of 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:

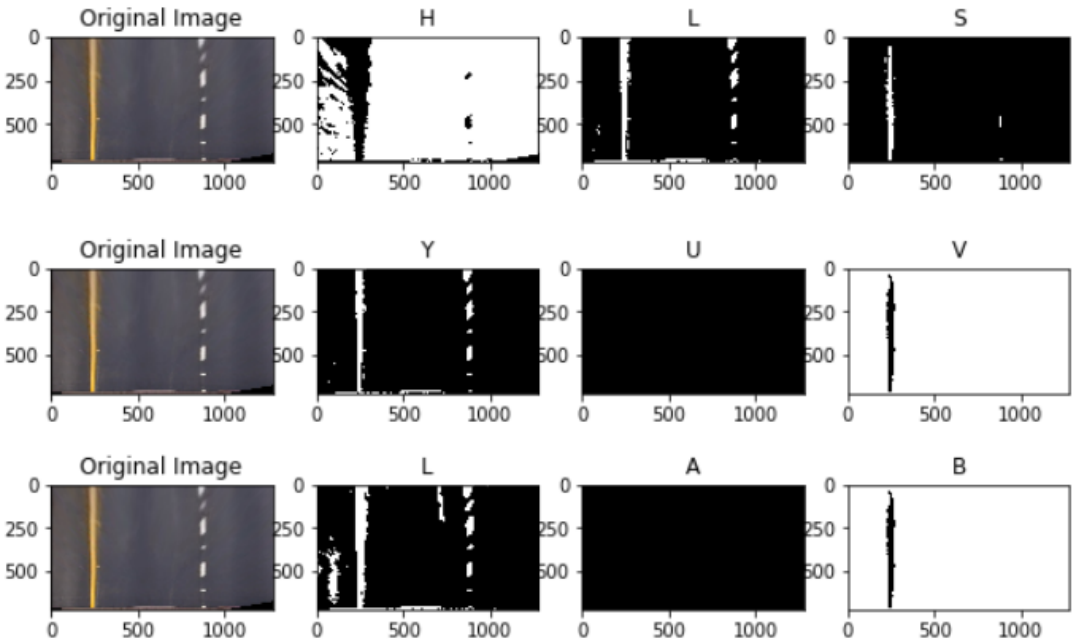
1. Provide an example of a distortion-corrected image

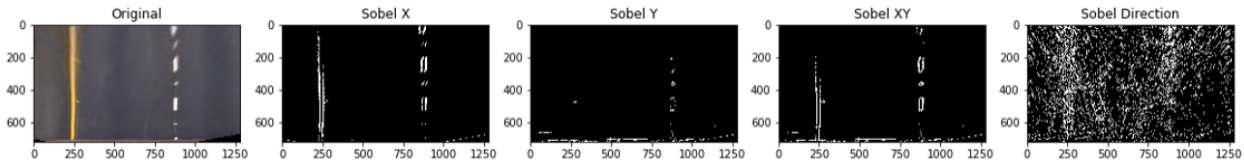


2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

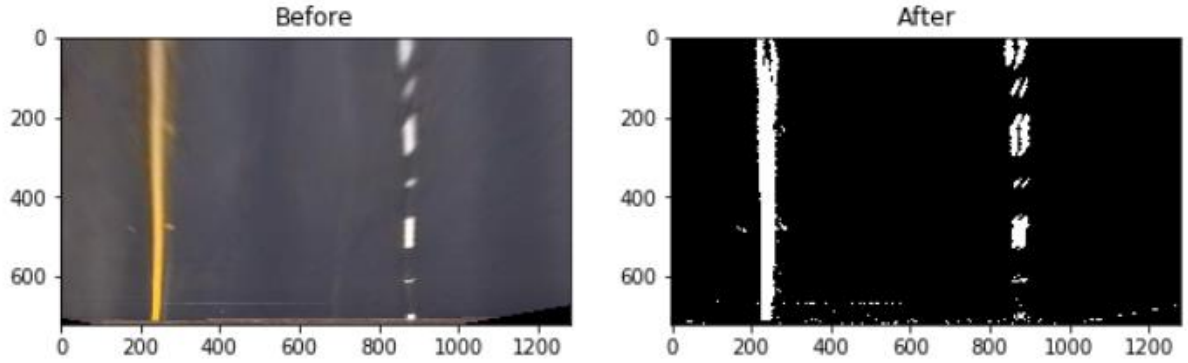
   See code cell 9 onwards of the IPython notebook.
   I used the following three color conversions:
   - cv2.COLOR_BGR2HLS
   - cv2.COLOR_BGR2YUV
   - cv2.COLOR_BGR2Lab

When we visualize individual components, we find that the S channel, Y channel and the L channel in each of the above transformations respectively give a good indication of the lane lines.
Also, I computed the sobel magnitude gradient in x, y and xy directions and also the direction gradient. From the images below, the best suited was the sobel x gradient.
Please see the individual components below,

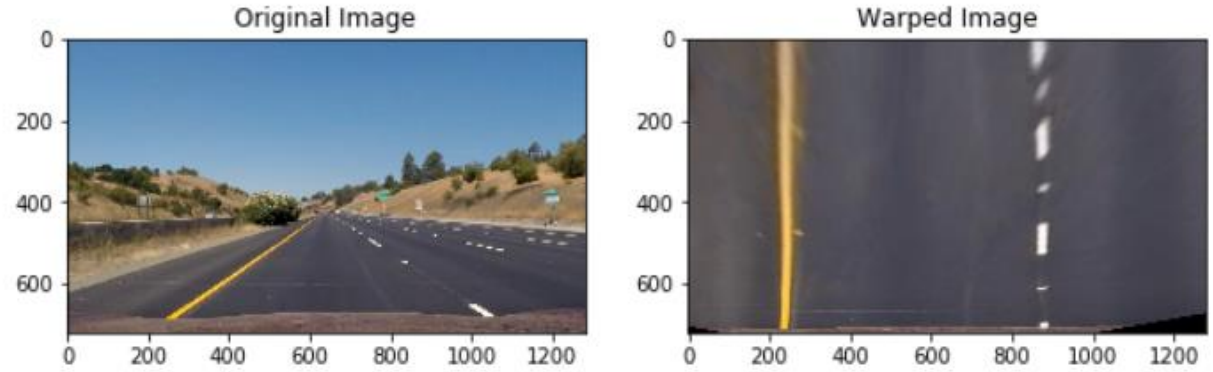




Combining the appropriate thresholds, I got the following result with was fairly OK.



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

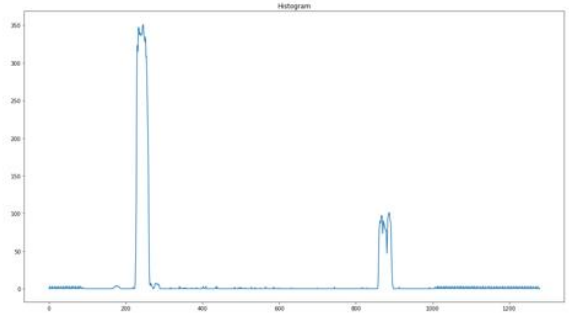


See code cell 6 onwards of the IPython notebook.

I chose the source and the destination points by eyeballing the image plot. Then I obtained the Perspective transform matrix M using the cv2.getPerspectiveTransform. The warped image was then obtained using the cv2.warpPerspective.

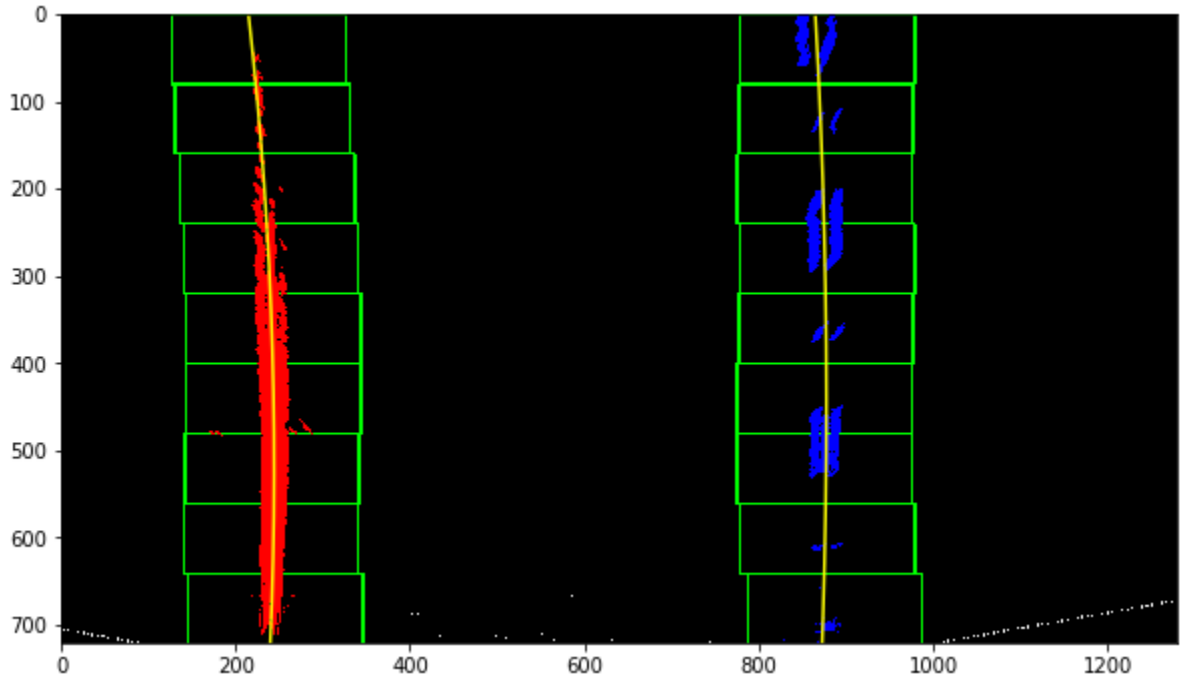
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

   From the below histogram of the warped image, we can see that the two peaks represent the left and right lane lines on the road. I used the sliding window method (Code cell 15) to detect the lane pixels.



To perform the sliding window search, the following steps were followed,

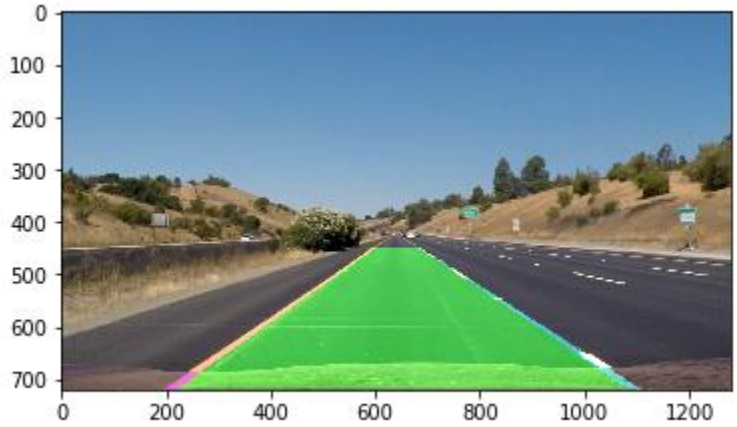
- Find the peak of the left and right halves of the histogram for the left and right lane lines respectively.
- Identifying the position (x,y) of all pixels with non-zero values
- Identifying window boundaries in x and y
- Drawing the windows on the image
- Identifying the non zero pixels within the window and appending those pixels to corresponding lists for left and right lane lines.
- Using polyfit function to fit the identifies points in a second order polynomial.



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

   See code cell 21

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



Pipeline (Video)

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!).

Discussion

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?