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

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

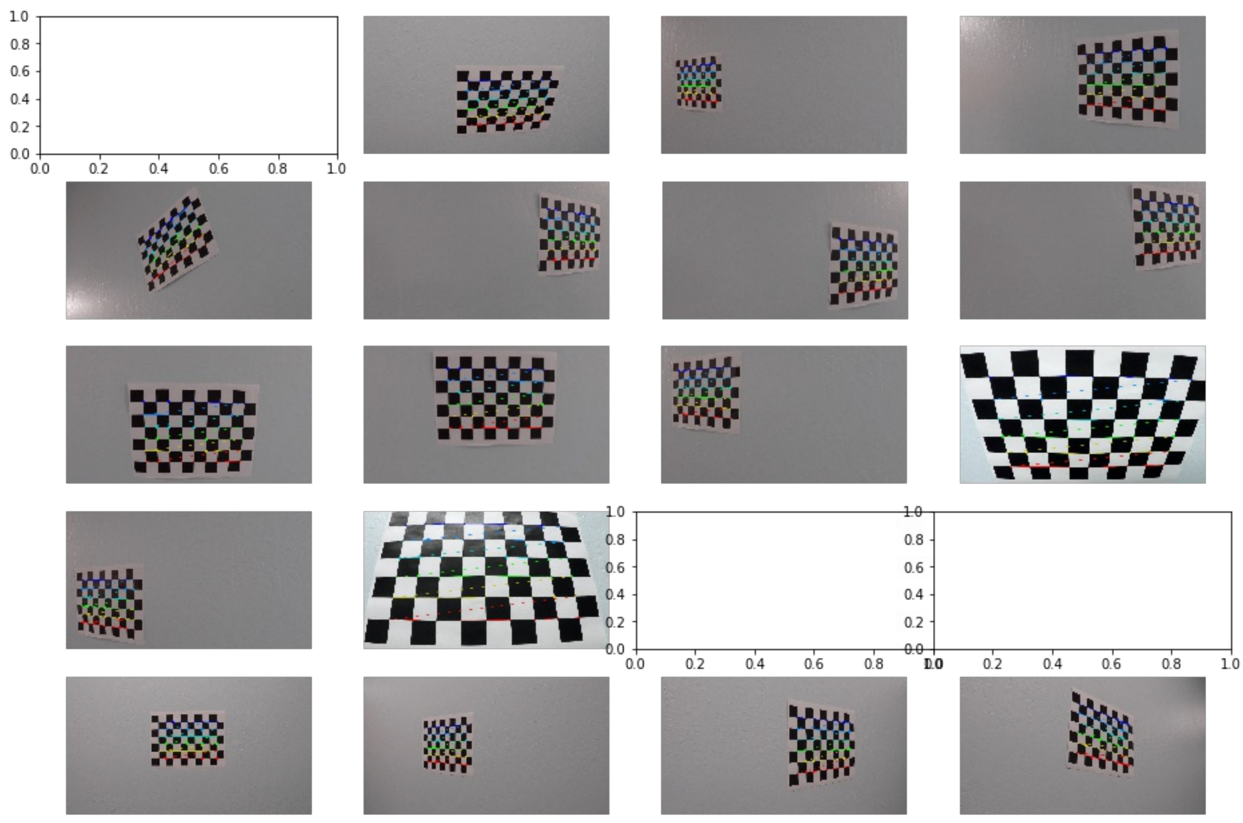
Camera Calibration

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 located in "./P4-Advanced-Lane-Detection.ipynb".

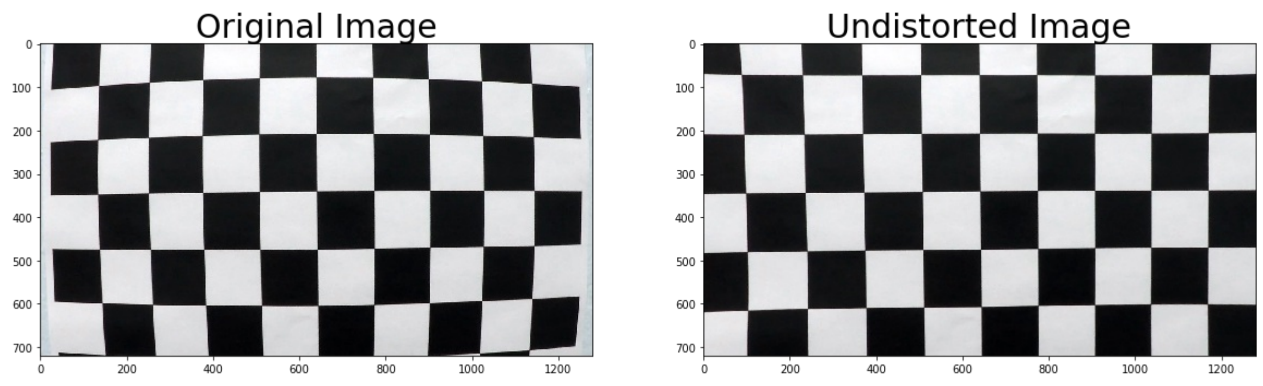
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of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

A sample of the corners that it detected for all the sample training images that were provided:



As you can see, not all the images had the corners detected in them. I am attributing it to the nature of the image and the capabilities of the OpenCV package in finding the edges accurately.

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:

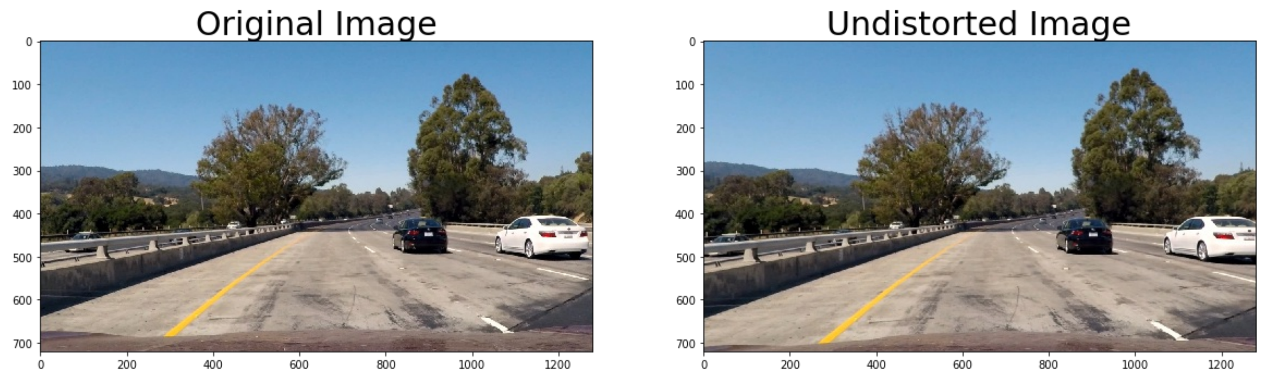


Pipeline (single images)

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:

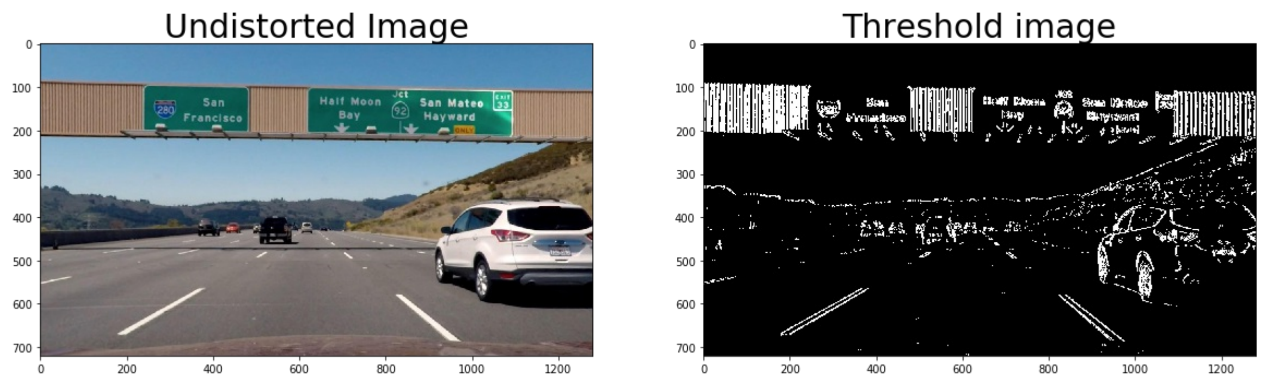
After performing the distortion correction based on the previous set of camera calibration performed, the undistorted image looks like the following:



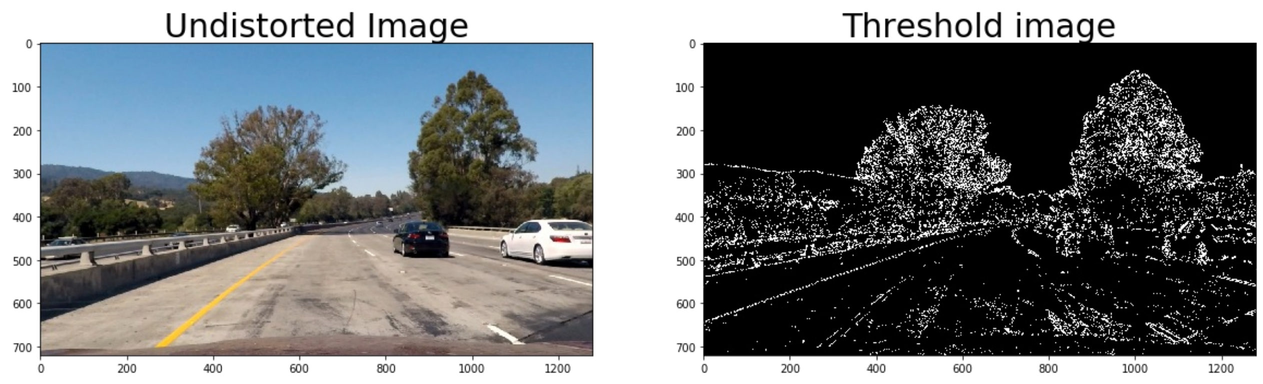
You can see the undistortion on the red car bonnet. In the original image its pretty much straight. In the distortion corrected image, its bent like how a bonnet would look like from the inside of a car.

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.

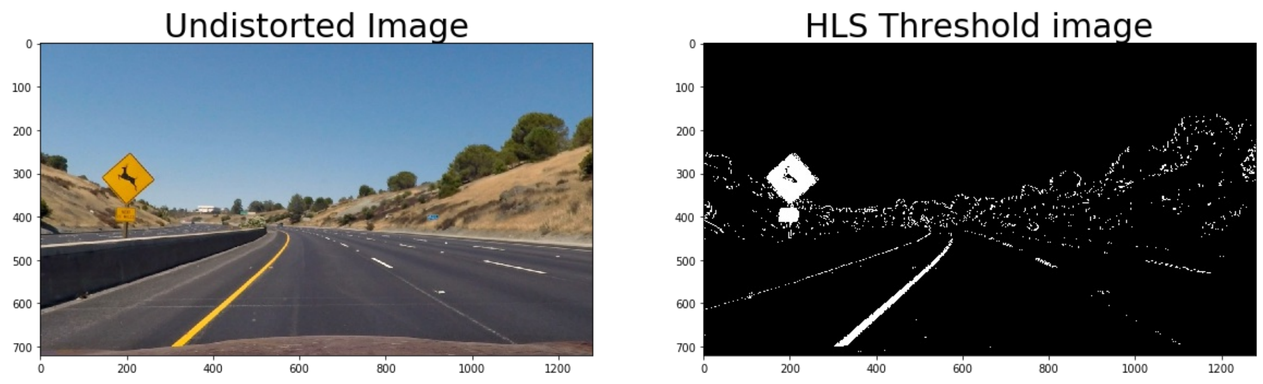
At first I wanted to look at the various types of thresholds that were mentioned in the lectures. Performing each of the Sobel threshold, gradient threshold, direction threshold and combining all together, helped me find the lane lines in the quiz image:



But, w hen I tried it w ith the above test image w hich contains a yellow line on the left side, it w as pretty difficult to find the lane lines:



This is w hen I realized that the reason for using a different color space w as because of how much of data grayscale conversion looses out. Thus converting the image into a HLS color space and performing Sobel threshold on it, helped me get the required lane lines as show n below :

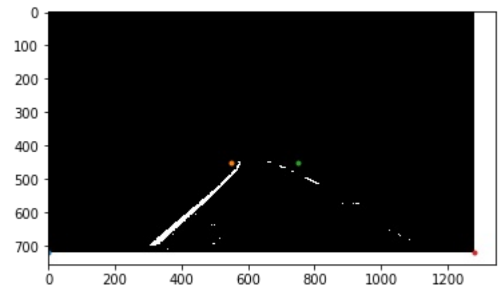


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The next logical step w as to create a mask on the region of interest. This helped me visualize the presence of the lane line better. You can find these in the ipynb file under the method name

```
def apply_region_of_interest(image, vertices):
```

Ex:



I then used the follow ing `src` and `dst` coordinates to perform the perspective transformation. This section of the code can be found in the ipynb file under the section:

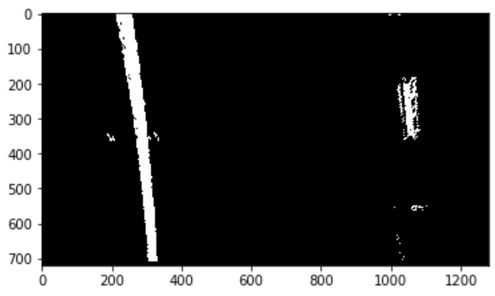
```
def warp(img):
```

Where I used the follow ing map values:

```
src = np.float32(
    [[150, 720],
     [550, 480],
     [750, 480],
     [1200, 720]])

dst = np.float32(
    [[200,720],
     [200,0],
     [1080,0],
     [1080,720]])
```

Once the image w as w arped, it w ould look like the follow ing:



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

Here I used the sliding w indow s approach thought to us in the lectures that is based on the follow ing equation to find the lane lines.

- Divide the image into n 'w indow s'
- Take the histogram for this w indow s
- Based on the peaks, determine the points that w ould be the Lane

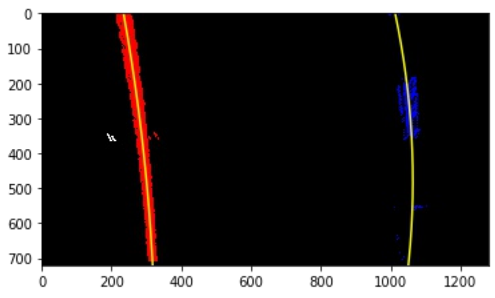
Refer to the code in the file under the method:

```
def sliding_window_polynomial(binary_warped):
```

If w e have the guestimate on w here the lines are currently, w e can reuse them to search around that area to find the lines in the new image frame. Refer to:

```
def skip_windows_polynomial(binary_warped, left_fit, right_fit):
```

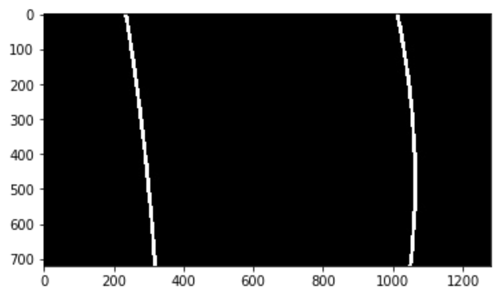
Performing this and visualizing the clusters of data points reveals a sample like this:



Using the follow ing method, I w as able to calculate the polynomial for each lane:

```
def lane_poly(yval, poly_coeffs):
    """Returns x value for poly given a y-value.
    Note here x = Ay^2 + By + C."""
    return poly_coeffs[0]*yval**2 + poly_coeffs[1]*yval + poly_coeffs[2]
```

This results in a sample like:



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.

Here I took some "inspiration" from the lecture codes and ended up w ith the code to find the curvature w ith the follow ing equations once I got the lane polynomial for the given lane fit:

```
left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
center = (((left_fit[0]*720**2+left_fit[1]*720+left_fit[2]) +(right_fit[0]*720**2+right_fit[1]*720+right_fit[2]) ) /2 - 640)*xm_per_pix
```

You can find the entire code in the method

```
def get_curvature_center(binary_warped, left_fit, right_fit):
```

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

Here, I used the `Minv` calculated w hen I performed perspective transformation on the color image block w here I 'guestimate' the line to be present.

Code for it:

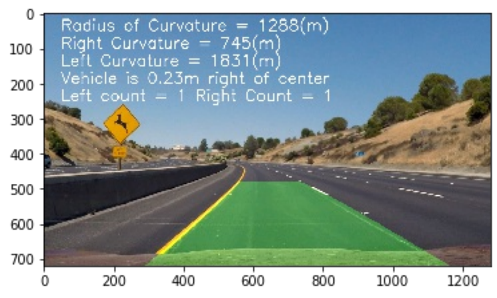
```
warp_zero = np.zeros_like(warp_img).astype(np.uint8)
color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

# Recast the x and y points into usable format for cv2.fillPoly()
pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
pts = np.hstack((pts_left, pts_right))

# Draw the lane onto the warped blank image
cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))

# Warp the blank back to original image space using inverse perspective matrix (Minv)
newwarp = cv2.warpPerspective(color_warp, Minv, (image.shape[1], image.shape[0]))
# Combine the result with the original image
result = cv2.addWeighted(image, 1, newwarp, 0.3, 0)
plt.imshow(result)
```

Once I had all the information, the information w as added to the frame like:


Radius of Curvature = 1288(m)
Right Curvature = 745(m)
Left Curvature = 1831(m)
Vehicle is 0.23m right of center
Left count = 1 Right Count = 1

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 (project_output.mp4)

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 information from the lectures w as quite sufficient for me to get started w ith. But, I soon ran into several problems in trying to understand w hy my lanes w ere not being detected correctly and how to store the best fit and making use of the suggested `class Line`. After some googling around and stumbling across some gem of a Git repositories, I w as able to get some more traction on it and I w as able to complete the video.

Shortfalls:

- Does not w ork on changing lighting conditions:
    - I w ould like to perform a varying threshold for the Sobel threshold and HLS threshold till I get the lines for each frame if the default one does not detect it. This is useful in conditions w here the image's lighting condition varies quite a bit.
- Does not w ork on overly curved roads:
    - This I attribute to the fixes area that I mask to get the lane lines. This needs to be dynamic once again to get the lanes for overly curvy roads. This is something that I w ould like to spend more time in the future to get it done better.
- Maintaining the history of the lane information:
    - This is something that I have done in quite a primitive manner. Some of the checks are good, but I am not currently checking on situations w here I have a good Left lane but the right lane detected has a very small radius and so on. This might help me solve the harder video much better.

There could be many other shortcomings that are present w ith this current implementation. This, I w ill be able to better judge once I have more know ledge on processing these images better.