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.

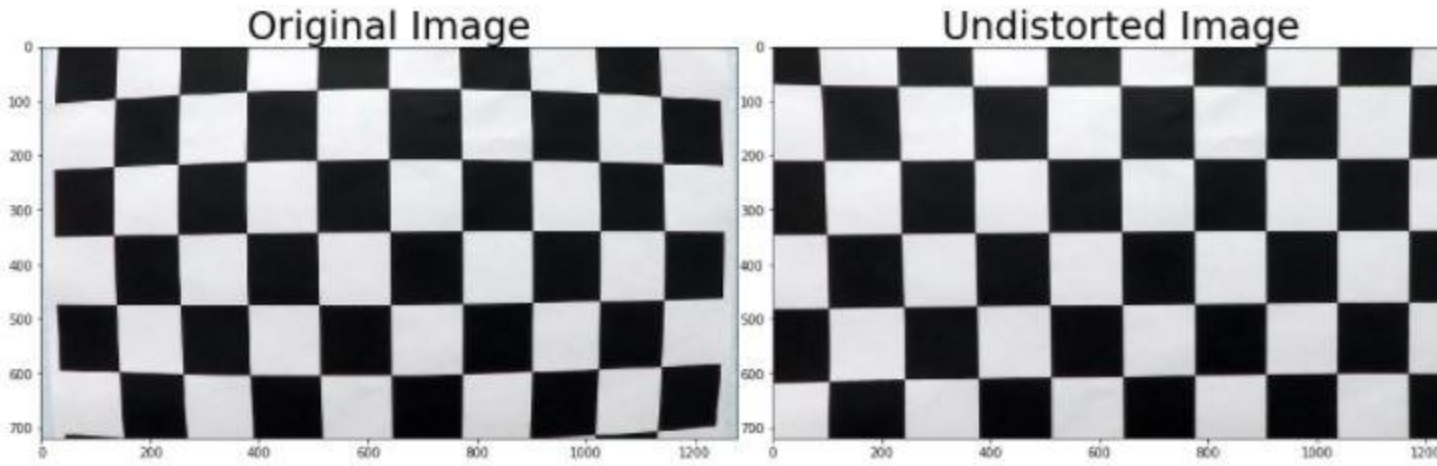
Here I will describe how I addressed each point in my implementation.

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 "./example.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)

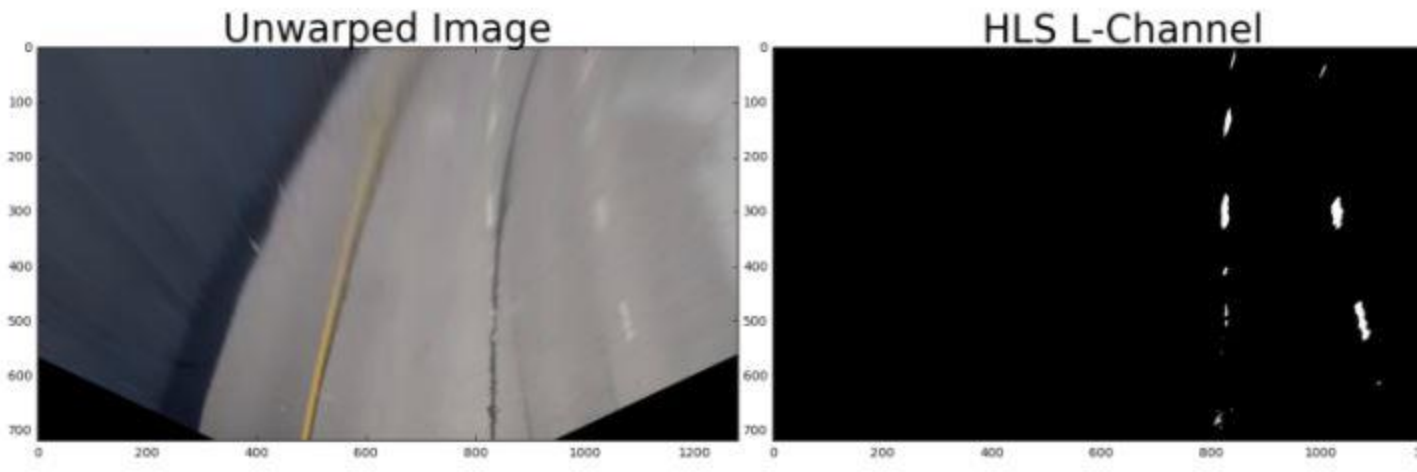
1. An example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this
one:

2. Color transforms, gradients and other methods to create a thresholded binary image.

I tested L-channel and S-channel HLS color transformation and found out the L-channel gives the right amount off information about white lane for me to work on.

Here's an example of my output for this step on one of the test images.



3. Perspective transform.

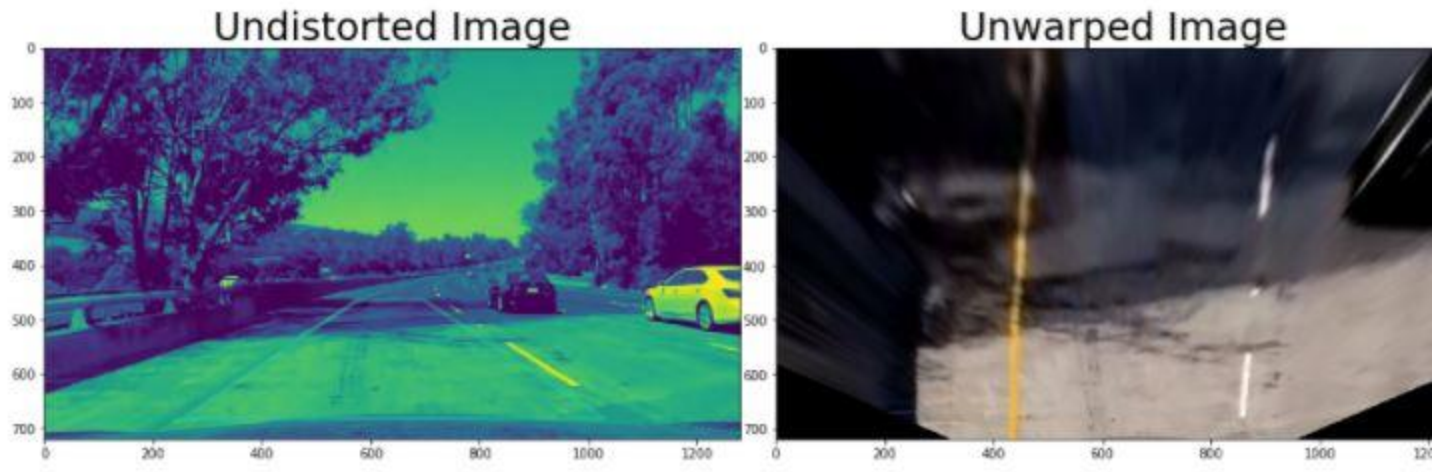
The code for my perspective transform includes a function called `corners_unwrap()`, which appears in the 5th code cell of the IPython notebook).
The `corners_unwrap()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

```
src = np.float32([(575,464),
                  (697,464),
                  (258,682),
                  (1029,682)])
dst = np.float32([(450,0),
                  (w-450,0),
                  (450,h),
                  (w-450,h)])
```

This resulted in the following source and destination points: src: [[575. 464.] [697. 464.] [258. 682.] [1029. 682.]]

dst: [[450. 0.] [830. 0.] [450. 720.]

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.

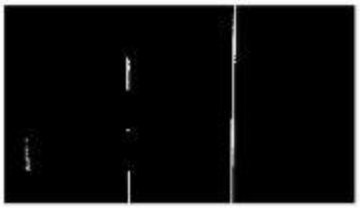


4. Identifiying lane-line pixels and fit their positions with a polynomial

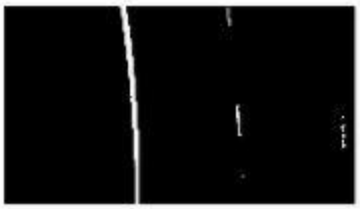
For pipeline, I experiemented with:

1. Combining L-channel with Sobel Mag threshold
2. Combining S-channel with Soble Mag threshold
3. Combining S-channel with Sobel Mag+Dir threshold
4. Combining L-channel with B-channel from LAB colorspace

I got the best identification of right and left lanes using the 4th method and hence I applied it to all my images test set as shown below:

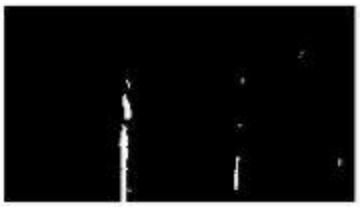





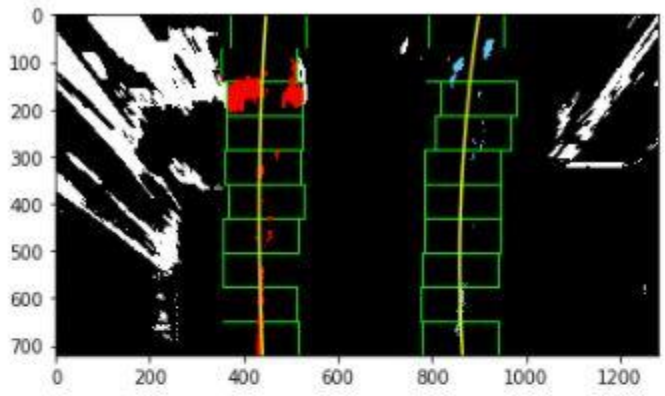






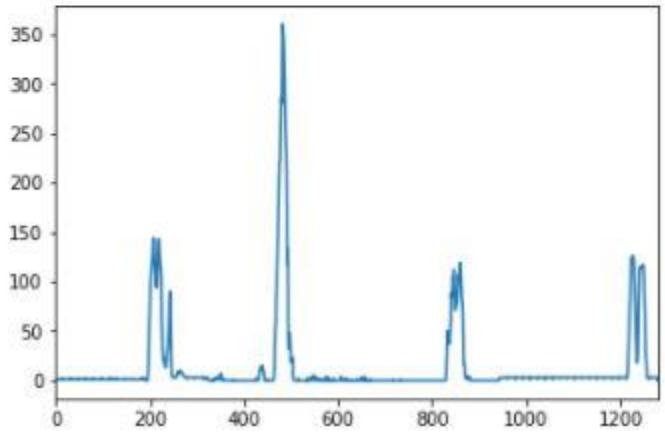


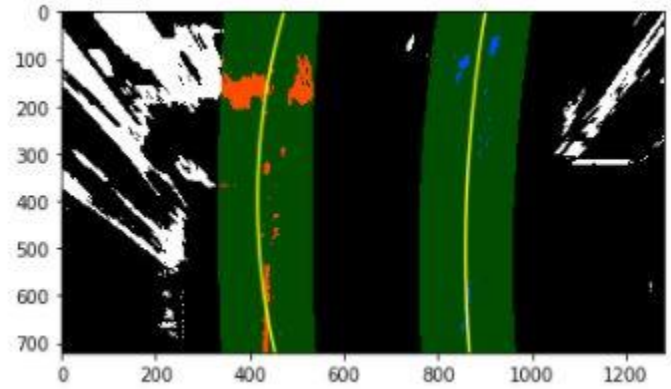
Then, I did the polyfit with and without knowing previous image fit as we assume that the fit will not change significantly from one video frame to the next. Here are is what output looks like after polyfit:




```
plt.plot(histogram)
plt.xlim(0, 1280)
```

```
(0, 1280)
```





5. Radius of curvature of the lane and the position of the vehicle with respect to center.

I have defined calc_radius function in notebook and it results in this for test_image5.jpg in test folder.

6. Result plotted back down onto the road such that the lane area is identified clearly.

I implemented this step in 'draw_lane' function.. 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 Car lane finding Video result
- Here's a Challenge Video

Discussion

1. 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?

- Finding the right src and dst arrays to apply perspective transform was little tricky. The result could easily not fall with in permissible limits or it will turn out total black.
- Selecting right color transform between S-channel and L-channel and combing it with Sobel Mag+Dir threshold required some trial and error

This pipeline needs to more robust for different lighting conditions, for example:

- Shadows: Detection becomes difficult with shadows on lane.
- Brightness of lanes: If the lanes are faded or if they are too bright, that can also cause issues in perception by software logic. Playing with different color spaces and altering the threshold value may be one way to go
- Weather conditions: Like shadow case, SW logic needs to be robust for proper detection, for example detecting in rain on water on the road can impact detection. Utilizing deep learning techniques to detect features automatically by training on different weather conditions can be one possibility to make it robust.