Adnced Lane Finding Project

The goals / steps of this project are the following:

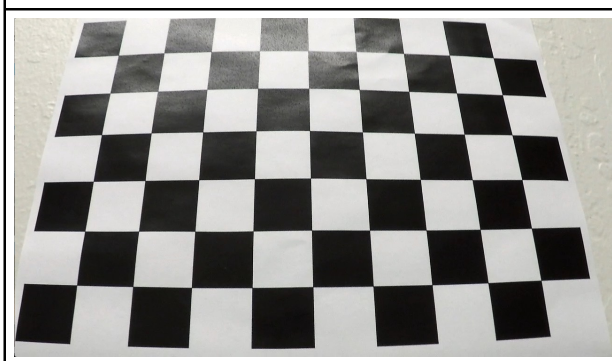
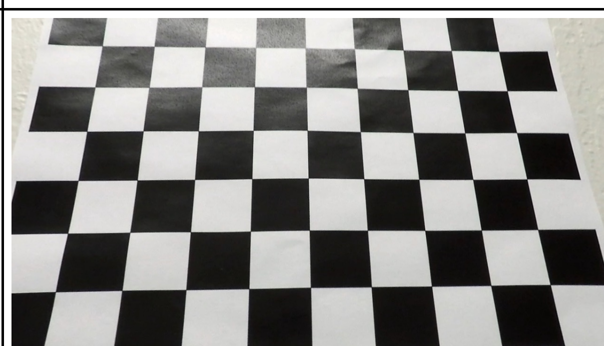
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").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

Camera Calibration

The code for this step is contained in the code cells of the IPython notebook located in `./CameraCal.ipynb` 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:

Original	Undistorted
	

Pipeline (test images)

The code for this step is contained in the code cells of the IPython notebook located in `image_gen.ipynb` notebook.

I have applied the distortion correction using the `cv2.undistort` method to undistort test image and the result looks like this one:

Original	Undistorted
	
Fig: an example of a distortion-corrected image.	

Then I used a combination of color and gradient thresholds to generate a binary image (thresholding steps are in `image_gen.ipynb` notebook method `convert2binary` and `color_threshold` metrhod). The image is converted to HLS and HSV and binarized them separately and then combile them (and operation) to get the final binary image.

Here's an example of my output for this step.

Fig: an example of a binary image result

The code for my perspective transform includes a function called `warp_image()`, in the 3rd code cell of the IPython notebook. The `warp_image()` function takes as inputs an image (`img`), as well as other parameters to calculate source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner (to use in OpenCV `getPerspectiveTransform` method):

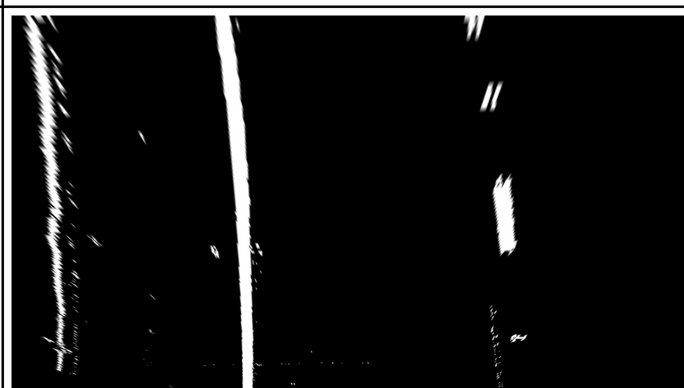
```
    src = np.float32([[img.shape[1]*(.5-mid_width/2),
                        img.shape[0]*height_pct],
                      [img.shape[1]*(.5+mid_width/2),
                        img.shape[0]*height_pct],
                      [img.shape[1]*(.5+bot_width/2),
                        img.shape[0]*bottom_trim],
                      [img.shape[1]*(.5-bot_width/2),
                        img.shape[0]*bottom_trim]])

    offset = img_size[0]*.25
    dst = np.float32([[offset, 0], [img_size[0]-offset, 0],[img_size[0]-offset, img_size[1]],
                      [offset ,img_size[1]]])
```

This resulted in the following source and destination points:

Source	Destination
544, 468	320, 0
736, 468	960, 0
1120, 673	960, 720
160, 673	320, 720

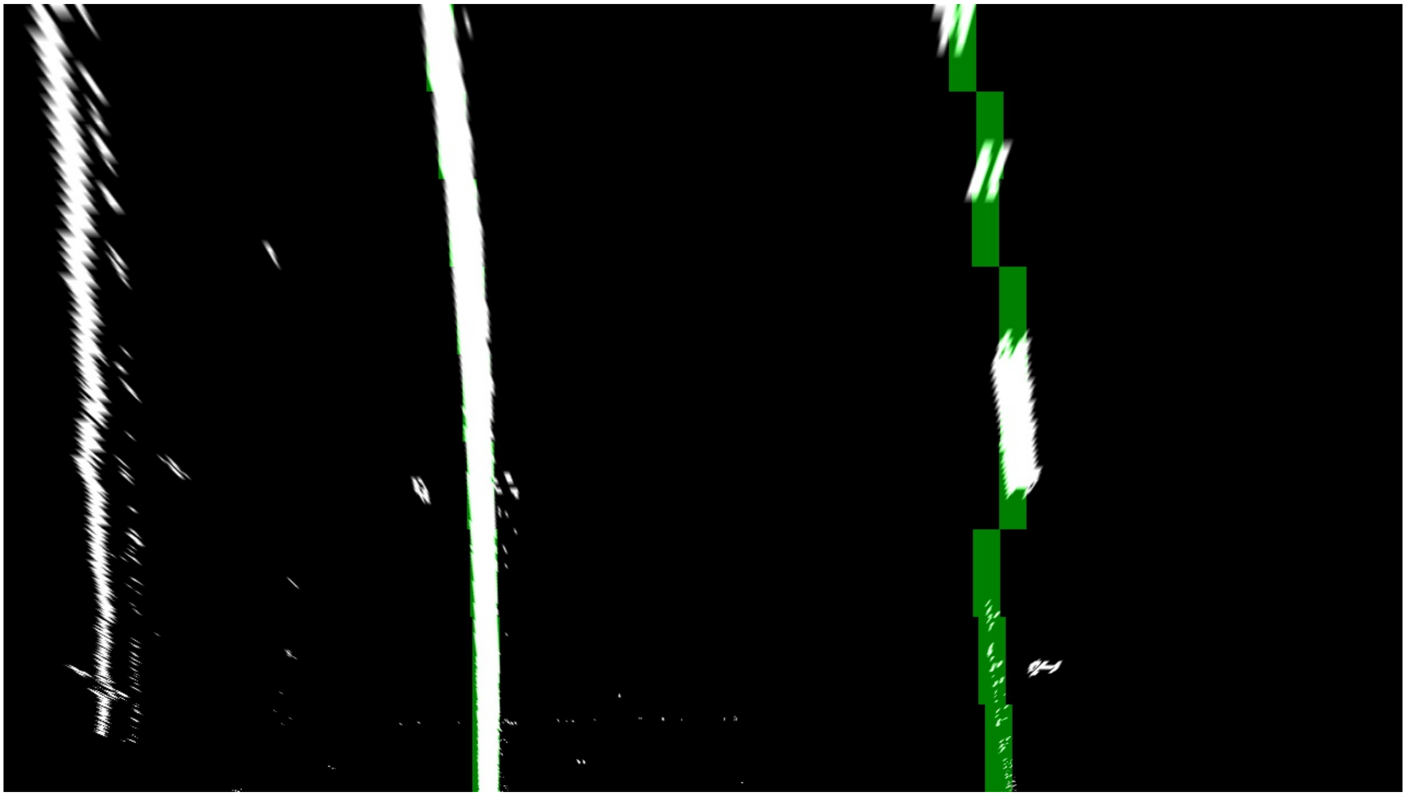
I verified that my perspective transform (done using opencv `warpPerspective` method) was working as expected by looking at the test image and its warped counterpart to verify that the lines appear parallel in the warped image (looks parallel).

Original	Warped
	
Fig: an example of a perspective transformed image.	

Then I did use the `find_window_centroids` method in the 4th cell of the notebook and calculate lane line points. The image is split into 80 pixel high horizontal layers.

First we find the two starting positions for the left and right lane by using np.sum to get the vertical image slice and then np.convolve the vertical image slice and go through each layer looking for max pixel locations to find the best centroid for left and right lane location.

The result looks like this:



Then I used the `fit_curves` method to fit the points with a 2nd order polynomial. And `draw_lane_markers` method draws the lane markers. I have not create an output for this one indivisually- but combined with the lane info and lane plane marker as shown below.

I have calculated the radious of curvature and car displacement from center in `draw_info` method and printed onto the result image.

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

I implemented this step in the `draw_lane_markers` method of the `image_gen.ipynb` notebook.

Pipeline (video)

Here's a link to my video result (https://youtu.be/vjPH_01BZGc) - https://youtu.be/vjPH_01BZGc (https://youtu.be/vjPH_01BZGc)

Discussion

In some cases the lane will be detected to be wider than expected. I have setup an expected width for the lane and will move the lane if it is too wide. Right way to do this would be to track previous frames (8 of them sougld be ideal for the parameeters I have used) and using a moving window in case of unrealistic lane detection. Lanes detected just before the current lane will have more weight and we may use a kalman filter to eliminate such single frame error.

The lane is being detected but the challenge videos shows very poor result. The tracker needs to be more sophisticated than the current one.

The tracker is not keeping track of previous frames lane position and calculating each frame from scratch. This causes problem where the system is not confidently determining the lane position. Without keeping track of previous frame

The lane width is fixed fro sample video and lane is always parallel. The system is not using this information to avoid creating oval shaped lane soimetimes.

I have not extended the lane on top of the car front.

Reference

Code from: Self-Driving Car Project Q&A | Advanced Lane Finding https://www.youtube.com/watch?v=vWY8YUayf9Q (https://www.youtube.com/watch?v=vWY8YUayf9Q)