M ## Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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're reading it!

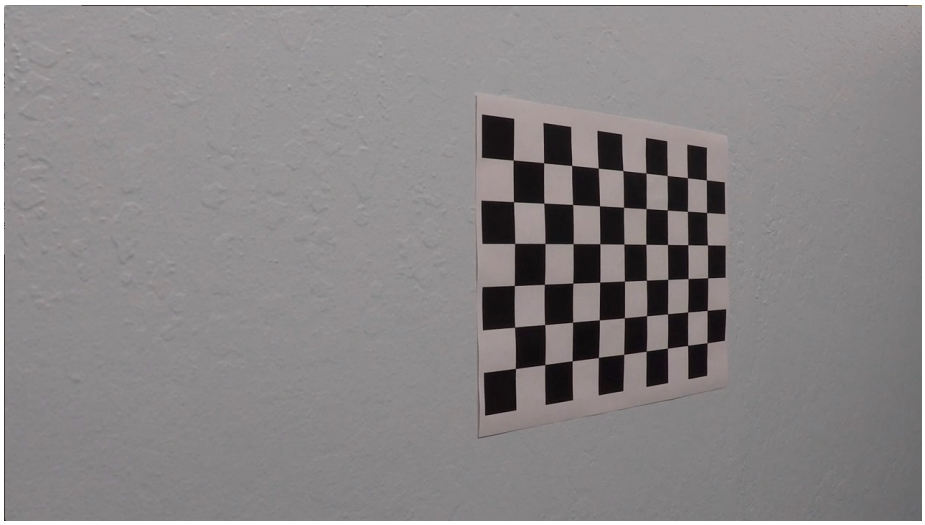
Camera Calibration

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

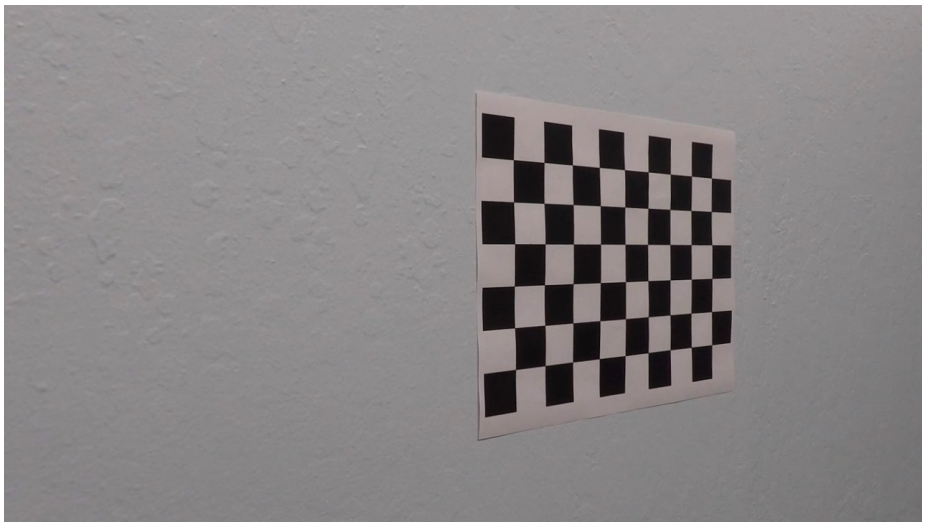
The code for this step is contained in the first code cell of the IPython notebook located in "./P2.ipynb".

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.

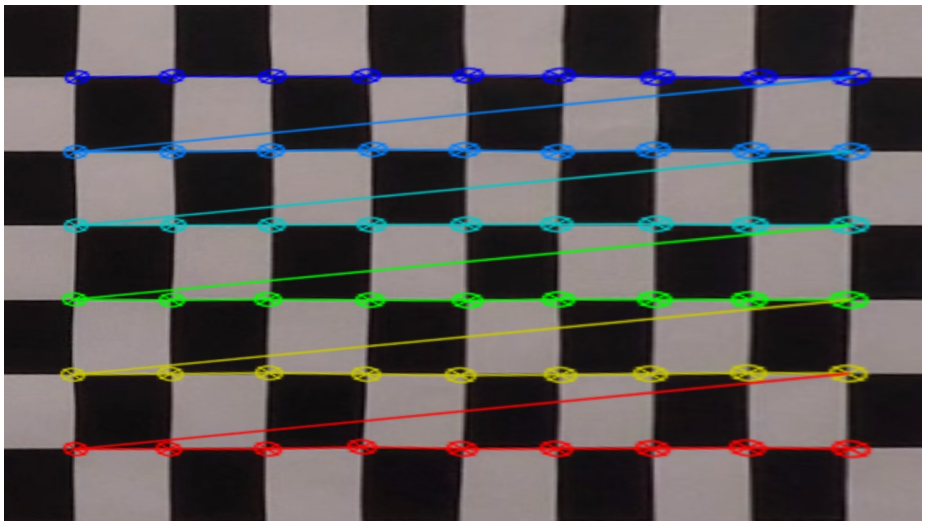
I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



Original Chessboard Image



Undistorted Chessboard Image



Warped Perspective Chessboard Image

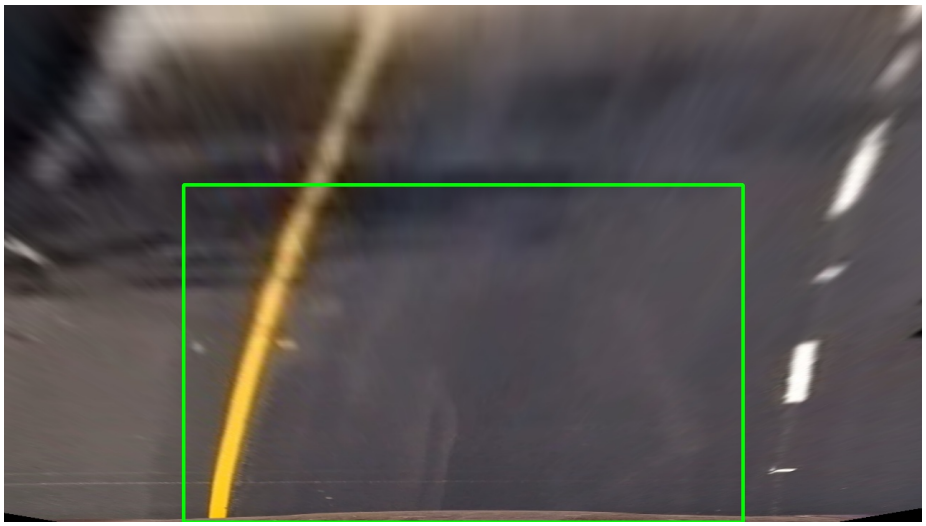
Pipeline (single images)

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:

Original Test 6 Image

Undistorted Test 6 Image



Warped Perspective Test 6 Image

I simply called `undist = cv2.undistort(image, mtx, dist, None, mtx)` using the mtx generated from calibrating for the camera earlier in the notebook.

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.

I used a combination of color and gradient thresholds to generate a binary image in the fuction color_sob_pipeline in cell 14 of the notebook. I anded a x sobel fiter with thresholds 20, 90 and a y sobel filter with thresholds 100, 150. I then or-ed this with a magnitude sobel filter with thresholds 30, 100 or-ed with a dir sobel filter with thresholds 0 and PI/4. This was all then ored with a color HLS filter with s_thresholds 180 and 255 and and h_thresholds of 20 and 30.
Here's an example of my output for this step.

Binary Image Test Image 6

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

I identified a portion of the lane image which should be rectangular from a plan view.

```
SRC_TOP_Y = .65
SRC_TOP_X_LEFT = 0.444
SRC_TOP_X_RIGHT = 0.56
SRC_BOTTOM_X_LEFT = 0.281
SRC_BOTTOM_X_RIGHT = 0.880
offset = 250

# Source src co-ords
src = np.float32([(img_size[1]*SRC_BOTTOM_X_LEFT, img_size[1]), # bottom left
    (img_size[0]*SRC_TOP_X_LEFT,img_size[1]*SRC_TOP_Y), # top left
    (img_size[0]*SRC_TOP_X_RIGHT, img_size[1]*SRC_TOP_Y), # top right
    (img_size[0]*SRC_BOTTOM_X_RIGHT, img_size[1])]) # bottom right


# Destination map coordinates
dst = np.float32([
    [offset, img_size[1]], # bottom left
    [offset, offset], # top left
    [img_size[0]-offset, offset], # Top right
    [img_size[0]-offset, img_size[1]] ]) # Bottom Right
```

I then warped to make these for points rectangular in the function tranform_lane_perspective in the notebook.

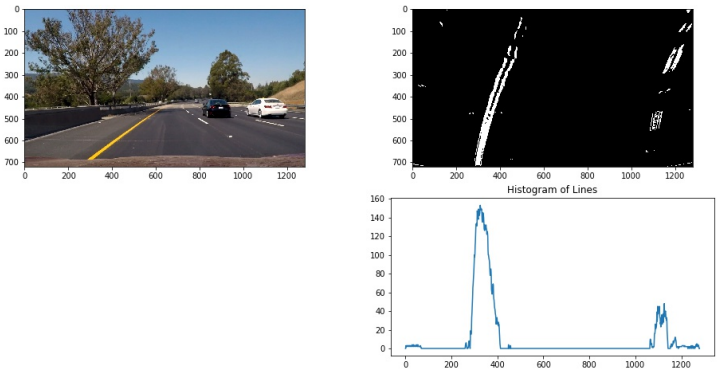
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

Unwarped Lane Rect Test 6 Image

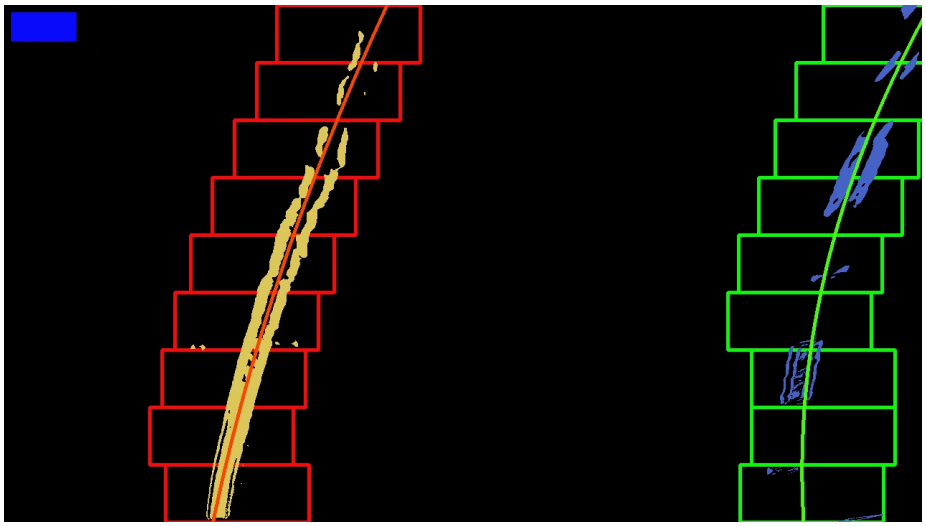
Warped Perspective Lane Rect Test 6 Image

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

I fitted the lane lines in the warped image to a 2nd order polynomial using polyfit. I then displayed the lines to visually check that they made sense.



Histogram used to find the x values to start the curve at the bottom at



The curves generated with a a window and Searching around

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.

I did this in my notebook in the function `measure_curvature()` I calculated the radius of the curve by using the equation:

I scaled the curves using the parmeters

```
    ym_per_pix = (30/720)  * 10# meters per pixel in y dimension
        xm_per_pix = 3.7/750 # meters per pixel in x dimension
```

to transform the scale of the curve up into meters from pixels in warped space.

```
# Get points for warp space fits
left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]

## Implement the calculation of the left line here at the point y_eval
left_curverad = ((1 + (2*left_fit_meters[0]*y_eval*ym_per_pix  +
    left_fit_meters[1])**2)**(3.0/2)) / np.absolute(2*left_fit_meters[0])

right_curverad = ((1 + (2*right_fit_meters[0]*y_eval*ym_per_pix  +
    right_fit_meters[1])**2)**(3.0/2)) / np.absolute(2*right_fit_meters[0])
```

I then averaged the left and right radii and returned this.

For the mid point I simply got the average center point of the bottom 5 points on the right and left lane curves. I then got the difference between them to get the the center point of the lane. I then got the x midpoinit of the image to be the car center. I then got the difference between both of these. I then scalled this up to meters by using xm_per_pix again. The result can be seen below.

The lane overlayed onto the original road image with the curvature radius and center point.

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

I implemented this step in lines in the notebook in the function `project_back()`. Here is an example of my result on a test image:

The lane overlayed onto the original road 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 link to my video [./test_videos_output/output_project_video.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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further. I think my approach is relatively robust in the project video. It does not perform as well in the challenge video as it sees the wall at the left of the road as a road line and then gets confused. Doing a

region of interest block get rid of that part of the image would work but would also make the system less robust to sharp turns and different types of lane lines.