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

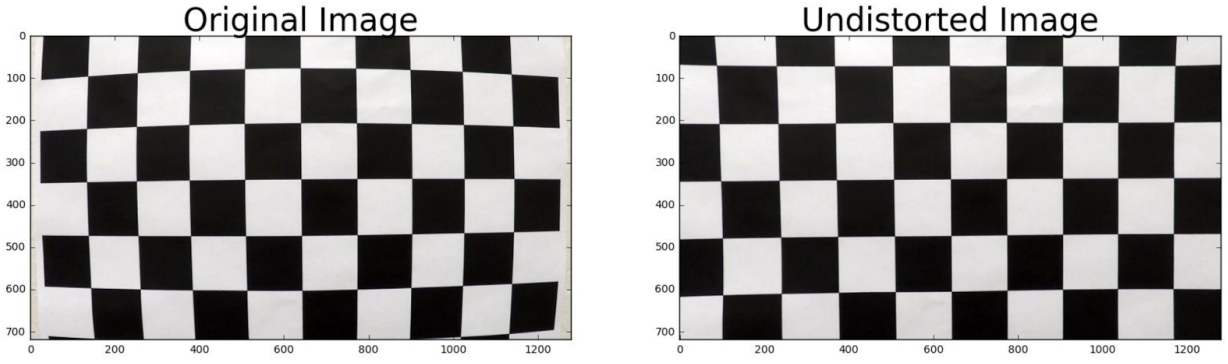
Camera Calibration

1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

IPython notebook **In [2]: def calibration(), return objpoints, imgpoints**

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:



2. Pipeline

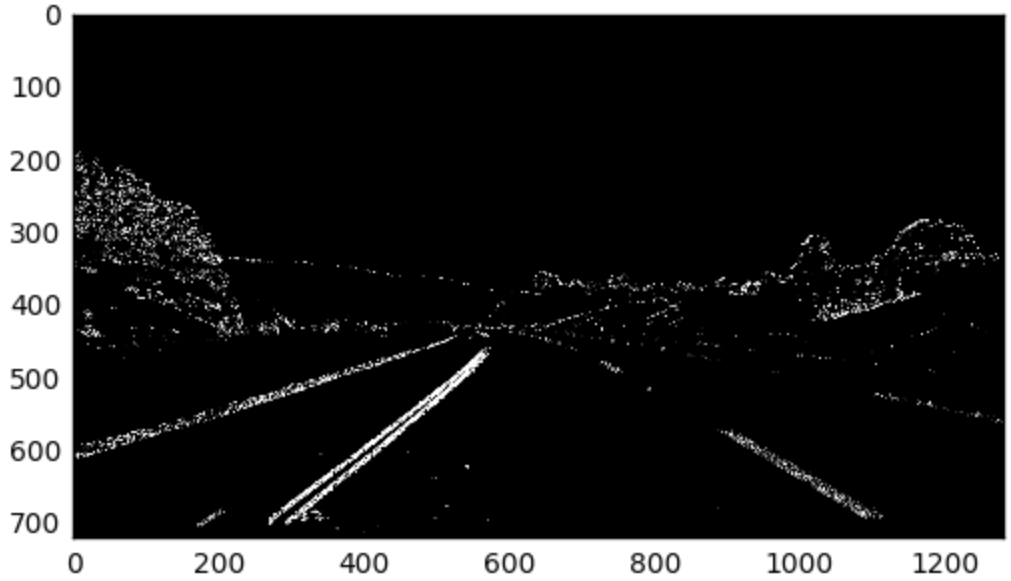
1) Has the distortion correction been correctly applied to each image?
   In [4], def undistort(img, objpoints, imgpoints), return dst
   The undistort function uses opencv cv2.undistort method to undistort images.
2) Has a binary image been created using color transforms, gradients or other methods?
   IPython notebook
   In [9]: Applying gradient threshold
   def abs_sobel_thresh(img, orient='x', sobel_kernel=3, thresh=(0, 255))
   return grad_binary
   In [10]: Applying magnitude threshold
   def mag_thresh(sobelx, sobely, sobel_kernel=3, mag_thresh=(0, 255))
   return mag_binary
   In [11]: Applying direction threshold
   def dir_threshold(sobelx, sobely, sobel_kernel=3, thresh=(0, np.pi/2))
   return dir_binary
   In [12]: The pipeline to create a binary image from distorted image. I created two separate masks: one for s_channel and the other for gray image.
   def pipeline(img, sobel_kernel = 15, s_thresh_min = 20, s_thresh_max = 255,
   sx_thresh_min = 20, sx_thresh_max = 100, sy_thresh_min = 30, sy_thresh_max = 255,
   dir_thresh_min = 0.7, dir_thresh_max = 1.3, mag_thresh_min = 25, mag_thresh_max = 101,
   dir_thresh_gray_min = 0.7, dir_thresh_gray_max = 1.3, mag_thresh_gray_min = 0,
   mag_thresh_gray_max = 101):
   return combined *
   *I used the interact module below to tune my parameter

```
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widget
from IPython import display

interact(pipeline, img = dst, sobel_kernel = (3, 15, 2),h_thresh_min = (0, 255), h_thresh_max =
(0, 255), s_thresh_min = (0, 255), s_thresh_max = (0, 255),
      sx_thresh_min = (0, 255), sx_thresh_max = (0, 255),
      sy_thresh_min = (0, 255), sy_thresh_max = (0, 255), dir_thresh_min = (0, np.pi/2),
dir_thresh_max = (0, np.pi/2),
```

```
    mag_thresh_min = (0,255), mag_thresh_max = (0,255),dir_thresh_gray_min = (0, np.pi/2),
dir_thresh_gray_max = (0, np.pi/2),
    mag_thresh_gray_min = (0,255), mag_thresh_gray_max = (0,255));
```

Here is a binary image example after applying pipeline.

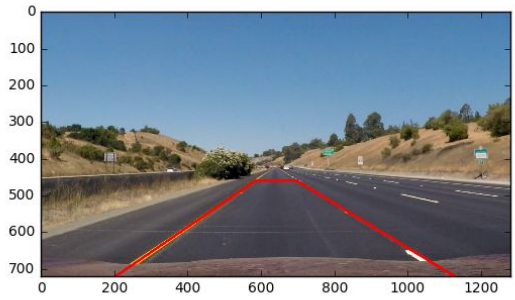


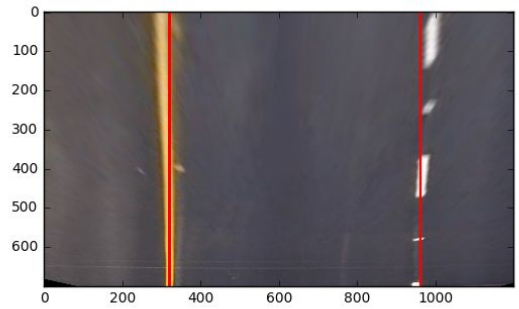
3) Has a perspective transform been applied to rectify the image?

In [13]: Applying a perspective transform to the binary image. I manually coded src and dst, and then applied cv2.getPerspectiveTransform to obtain M, and cv2.warpPerspective to get warped images.

Source	Destination
585, 460	320, 0
203, 720	320, 720
1127, 720	960, 720
695, 460	960, 0

def wraper(img):
return warped, M





4) Have lane line pixels been identified in the rectified image and fit with a polynomial? Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?

I used the sliding window method and ift a polynomial. Curvature is calculated by

$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$ **, curvature_radius = 0.5*(left_curvature_radius + right_curvature_radius). Also filters and smoothing methods are applied in the final pipeline(in [19]) on curvature radius, lane pixels.**

The position of the vehicle is calculated by

center_lane_bottom = 0.5*(left_lane_bottom + right_lane_bottom)

center_of_image = 600

offset = (center_of_image - center_lane_bottom)*xm_per_pix

if the offset is positive, then the vehicle is on the right of the center; if negative, the vehicle is on the left of the center.

In [15]: Identifying lane pixels and calculating curvature and offset

def fit_and_draw(binary_warped):

return lane_detected, left_fit, right_fit, ploty, left_fitx, right_fitx, left_fit_cr, right_fit_cr, leftx, lefty, rightx, righty, left_curverad, right_curverad, offset

In [16]: After several confident searches, skipping sliding widows, and passing left_fit, right_fit to identify lane pixels and calculate curvature and offset

def second_fit(left_fit, right_fit, binary_warped):

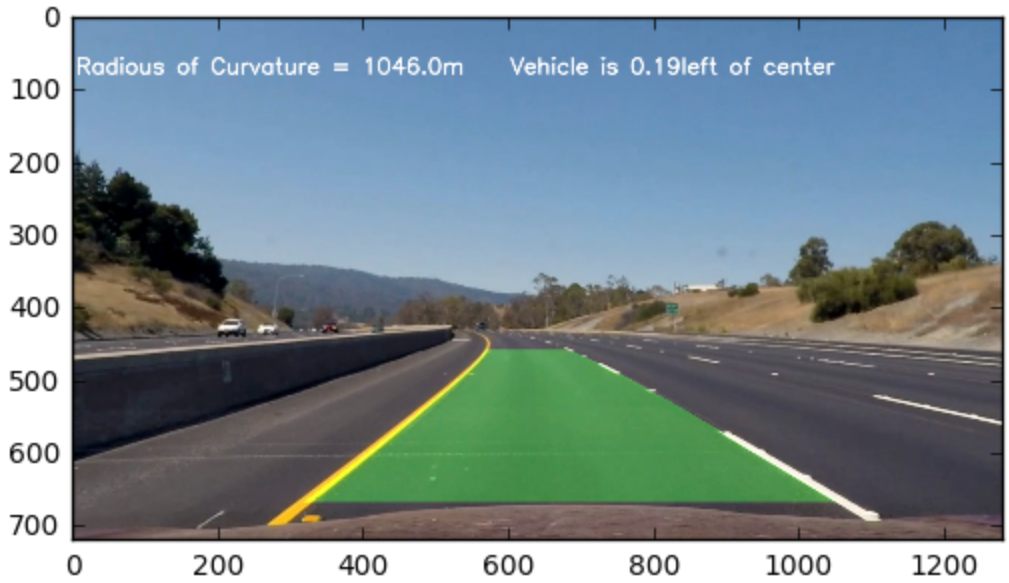
return lane_detected, left_fit, right_fit, ploty, left_fitx, right_fitx, left_fit_cr, right_fit_cr, leftx, lefty, rightx, righty, left_curverad, right_curverad, offset

5) Has the result from lane line detection been warped back to the original image space and displayed?

In [17]: I used the method provided in Udacity draw on the warped back lanes. And also write curvature radius and vehicle position on the output image.

def draw_and_output(ploty, left_fit, right_fit,left_fitx, right_fitx, dst, binary_warped, M, img, left_curverad, right_curverad, offset)

return result



3. Pipeline video

In [19]: Final pipeline

1) Does the pipeline established with the test images work to process the video?
   In the output folder, there are outputs of the test images through the final pipeline.

2) Has some kind of search method been implemented to discover the position of the lines in the first images in the video stream?
   My techniques here are after 60 good searches with sliding windows, the next search will skip sliding window searches and use left_line.best_fit, right_line.best_fit in left line and right_line classes.left_line.best_fit, right_line.best_fit are averages of latest 15 left_fit and right_fit which are recorded in left_line.current_fit, right_line.current_fit.
   Good searches are defined as left and right curvature radius are both more than 200 and less than 8000.
3) Has some form of tracking of the position of the lane lines been implemented?
   Latest 15 picture positions are recorded in left_line.recent_xfitted, and right_line.recent_xfitted. And left_line.bestx, right_line.bestx are averages of eft_line.recent_xfitted, and right_line.recent_xfitted.
4) For smoothing curvature radius, curvature radius are tracked and averaged each time a good search is performed.

Discussion

Besides when the weather condition changes, like strong sunshine, cloudy or rainy weather, parameters of creating binary images should be adjusted or added to make it more robust. For further improvement, I could add h channel to help create binary image.
I could also add checks for lane slops and make sure two detected lines have similar curvature. When curvatures are dramatically changed, new searches should be done instead of the smoothing methodologies.