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 colour 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 centre.
- 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

I started 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:

DISTORTED IMAGE

UNDISTORTED IMAGE

Pipeline

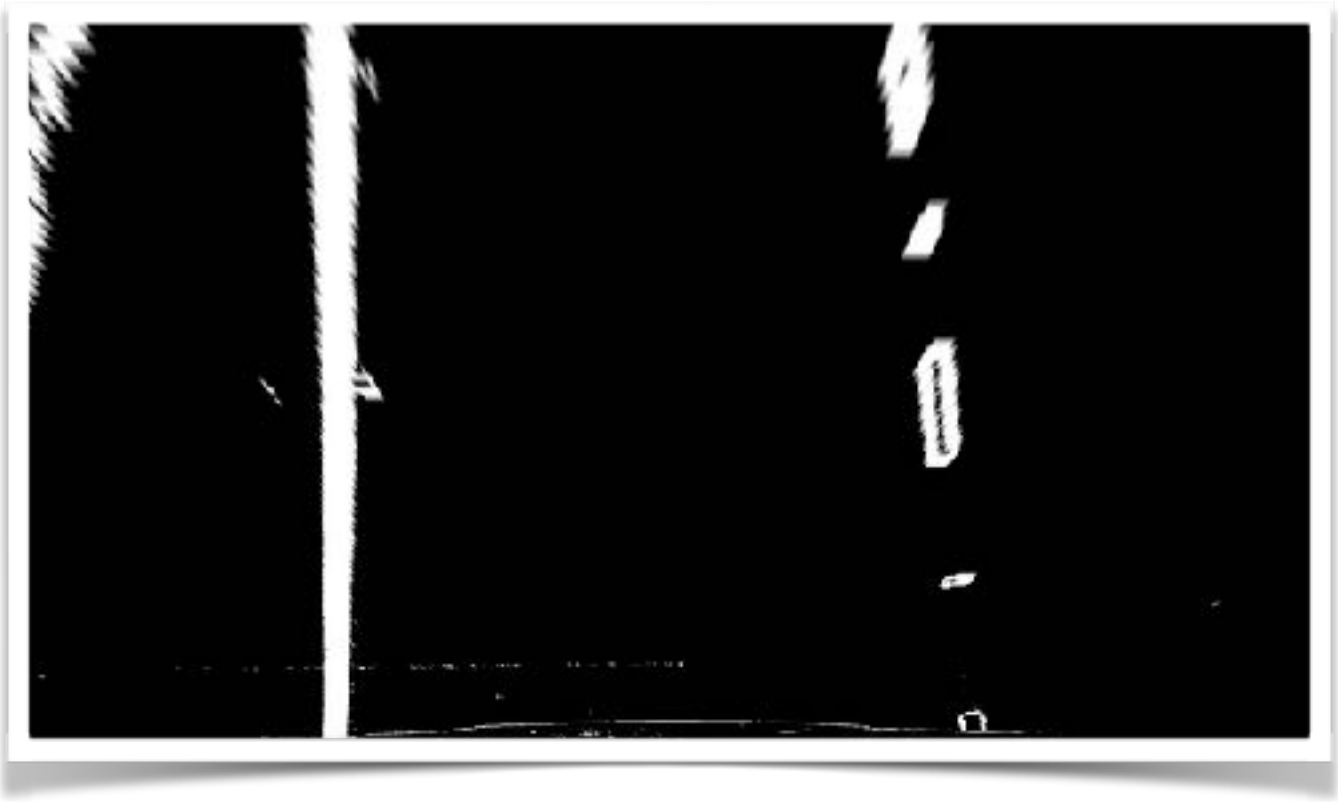
1) I have applied cv2.undistort() function on the image. Result of this function is displayed in the above page.

2) I have used abs_sobel_thresh() function to calculate gradient in x and y direction. Following this is a mag_thresh() function call and color_threshold. The exact parameters I have used can be found in lines [82-85]. I used a combination of colour and gradient thresholds to generate a binary image. I have applied a mag threshold on top of this. These can be found in lines [86-87]. Below is an example of my output for this step.

3) I have applied a perspective transform function using the following as source and destination points.

*src = np.float32([[(img_size[0] / 2) - 55, img_size[1] / 2 + 100], [((img_size[0] / 6) + 10), img_size[1]], [(img_size[0] * 5 / 6) + 60, img_size[1]], [(img_size[0] / 2 + 80), img_size[1] / 2 + 100]])*
*dst = np.float32([[(img_size[0] / 4), 0], [(img_size[0] / 4), img_size[1]], [(img_size[0] * 3 / 4), img_size[1]], [(img_size[0] * 3 / 4), 0]])*

Below is an image after applying warpPerspective function. lines [102-104]: (image_gen.py)



4) I have detected lane line pixels by calling sliding window algorithm on warped images. The tracker class handles finding the lane lines pixels. I have used the following parameters for sliding window algorithm:

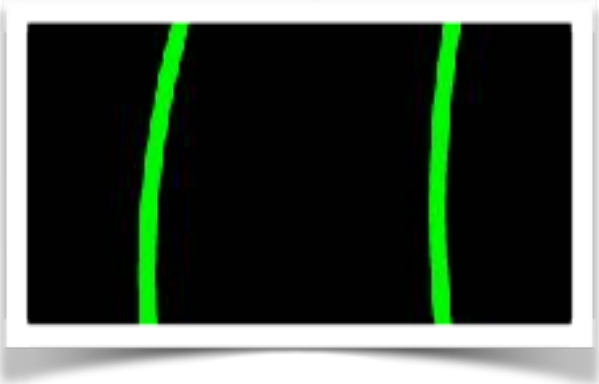
window width = 40
window height = 60
sliding margin = 30

I have used a second order polynomial (quadratic) to fit a curve for the lane lines. This can be found in lines [140-146].

5) I calculated radius of curvature and position of the vehicle with respect to the centre using the following equations:

*curverad = ((1 + (2*curve_fit_cr[0]*yvals[-1]*ym_per_pix + curve_fit_cr[1])**2)**1.5) / np.absolute(2*curve_fit_cr[0])*

Implementation details can be found in lines [169 - 175] (image_gen.py). Below is a picture of how polyfit draws the curves of the road.



6) Below is an image after all the processing steps:

Video

I have attached the video file along with the project files.

Discussion

My pipeline was able to detect and trace the yellow line smoothly but it had tough time maintaining the smoothness for while lane line. This is because I am not averaging the position of each lane line. My pipeline performs poorly on the harder challenge video mainly because the roads are much more curvaceous in that video and there is a sudden shift of lighting conditions. The colour gradient works like a charm in detecting yellow lane lines. This pipeline can be improved by adding in extra preprocessing steps like shadow removal, gaussian blurring, fine-tuning the sliding window algorithm parameters.