***All relevant images are in the ‘output’ folder; Please look at them to verify my results described below.***

*To see the algorithm interactively, set the ‘debug’ flag to on, and run main.py*

Camera Calibration:

- Calibrate camera by taking a 9x6 chess board, taking 20 pictures around it, and using opencv functions (such as cv2.findChessBoardConers) to locate the chess board corners.

Then, pass the coordinates (the ground truth points np.mgrid[0:9,0:6], and the coordinates obtained by cv2.findChessBoardConers) to cv2.calibrateCamera(ground\_truth, camera\_points, ...) function, which outputs the camera matrix and distortion coefficients. Example of unwarped image can be found in output\_images

Pipeline (found in function ‘find\_lane lines’):  
- First undistort the image based on the distortion coeffs and camera matrix found above

- Warp a manually selected trapezoidal region of interest (found in transform\_roi) to a rectangle, to create the effect of looking at the road from a bird’s eye view.

- Then, apply sobel x gradient and extract the points between a certain thershold, and OR (|) it with values between a certain threshold for the ‘S’ channel after converting the image to HLS space. This creates a binary image, where relevant pixels lane finding are 1 and irrelevant pixels are 0 (found in get\_binary\_image function)

- Sum up the binary values found in each column, and take the argmax of left half and right half of image, which represents the starting x coordinate for each lane (found in image\_col\_peaks function)

- Starting from those x positions, and the binary image found above, use a sliding window (starting from the bottom, and sliding upwards) for each the l and r line to collect relevant pixel coordinates within the box. To capture curved lane lines, you must slide the window (the window’s x midpoint) horizontally to the avg coordinate position of (window-1)’s found pixels. Alternatively, if there is an already valid polynomial fitted to each line, collect the relevant pixels (=1) within a reasonable distance from the polynomial line.

- After gathering these pixels, run np.polyfit(y,x,2) for each line to determine a 2nd degree polynomial that takes a y coordinate (in the bird’s eye view image) and outputs the corresponding x coordinate for the lane line

-Generate the curve for the polynomial by plugging in values into the equation for all y btwn 0 and img.shape[0]

- Then, use the source and dst points from the original warp to warp the lines generated from above to fit the original lane lines seen by the car camera. Then, draw a green region in between the lanes

-Calculate offset from center by comparing the x midpoint of the bottom point of the l and r lines, and compare this to image.shape[1]/2 (assuming that the car is exactly the center of the image). Then convert this result to meters, and that is the center offset.

Calculate curvature by first generating polynomial coefficients adjusted from pixels to meters, and plugging in the required values.

***Pipeline for processing video is implemented in the function \_\_call\_\_, and the pipeline for the lane finding algorithm is in the find\_lane\_lines function***

Problems with this method of line detection:

- The sliding window is not always reliable when there is some edge/splotch/gradient on the road that causes a major peak in the sum of column pixels for the warped\_binary\_image.

- ROI coordinates used in the transform of the car camera image into a bird eye’s view image can be wrong, especially when the car is on an incline, on a sharp turn, when there is an occluding object, etc. If you take too big of an ROI, you risk collecting irrelevant pixels.