Udacity Self Driving Car ND
Project 4: Advanced Lane Finding

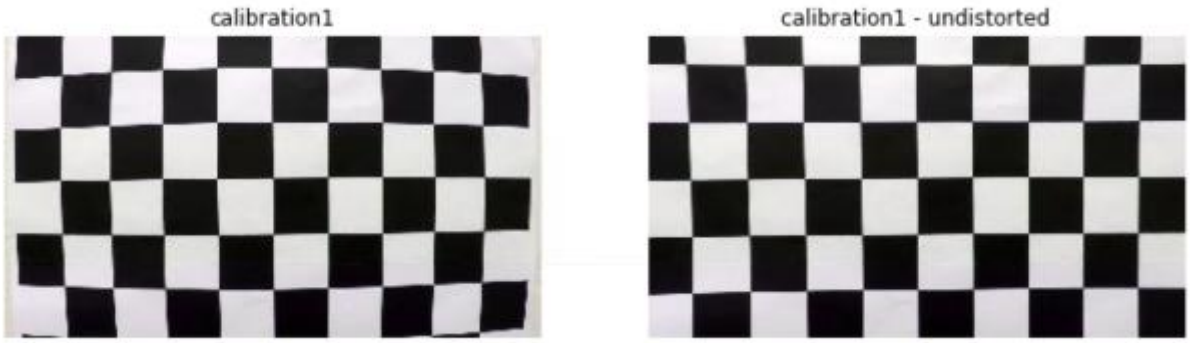
The project implements the following key steps:

1. A one-time only Camera Calibration to compute the calibration matrix and distortion coefficients
2. A one-time only Perspective Transform computation on pre-calculated image points to get a “birds-eye” view of lane lines
3. A one-time only Inverse Perspective Transform computation to warp the found lane line back to the original image
4. An image processing pipeline consisting of the following steps
    a. Apply distortion correction using the earlier computed calibration matrix and distortion coefficients
    b. Use color thresholds to isolate the lane lines and create a binary image
    c. Apply the perspective transform on the binary image to get a “birds-eye” view of the lanes
    d. Detect the left and right lane pixels and fit quadratic polynomial equations to the left and right points
    e. Compute the lane curvature and the vehicle position/offset with respect to the center
    f. Apply the inverse perspective transform on the detect lane lines/polynomial to warp the lane boundaries back to the original image
    g. Display the visual lane boundaries, the lane curvature and the vehicle position information

The code for to test/validate/visualize one-time individual one time steps and pipeline stages on sample images is in the *advanced_lane_lines_pipeline_exploration.ipynb* IPython notebook, while the video processing is in implemented *lane_lines_pipeline.py*

Camera Calibration

To compute the camera calibration matrix and the distortion coefficients, an “object points” array is initialized that contains the (x, y, z) real-world coordinates of the chessboard corners. The chessboard is assumed to be fixed plane i.e. z = 0 and the (x, y) world coordinates are same for all the images. We now iterate through all the chessboard images “camera_cal” folder and for each image we use the OpenCV function cv2.findChessboardCorners to try to find the “image points” of chessboard corners. The found chessboard corner points are further refined (to compute sub-pixel accuracy) using the OpenCV function cv2.cornerSubPix. The pre-determined “object points” and the computed “image points” are appended to the ‘objpoints’ and ‘imgpoints’ arrays respectively for each image.
The final 'objpoints' and 'imgpoints' (computed from all the chessboard images) is used to compute the camera calibration matrix and distortion coefficients using the OpenCV function cv2.calibrateCamera.
The figure below shows the result of the distortion correction by applying cv2.undistort function on one of the input chessboard images.



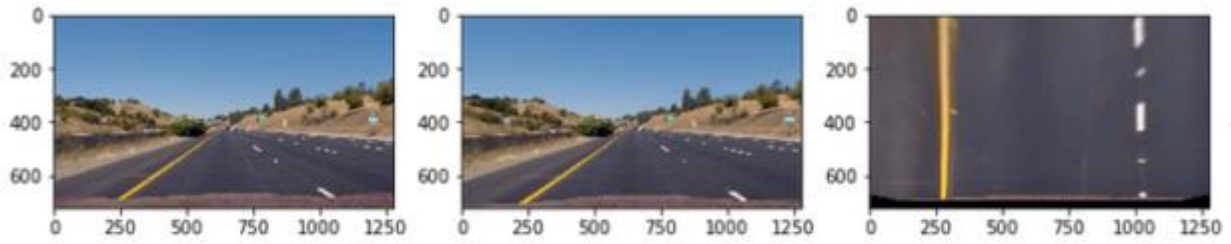
Perspective Transform Computation

To simplify computation of the perspective transform (“birds-eye view”) matrix use a test image with straight lane lines (straight_lines1.jpg), apply distortion correction (using the earlier computed matrix and coefficients) and using the cv2.getPerspectiveTransform function on manually select the source and destination points.

The following source and destination points where chosen for this operation:

SOURCE	DESTINATION
270,670	270,670
580,460	270,0
705,460	1035,0
1035,670	1035,670

The image below shows the result of perspective transform operations by applying the cv2.warpPerspective using the computed transform matrix



An inverse perspective transform matrix is also computed at the same by flipping the source and destination points to the cv2.getPerspectiveTransform function – this matrix is used later to warp the lane line boundaries back to the original image.

Processing Pipeline

Distortion Correction

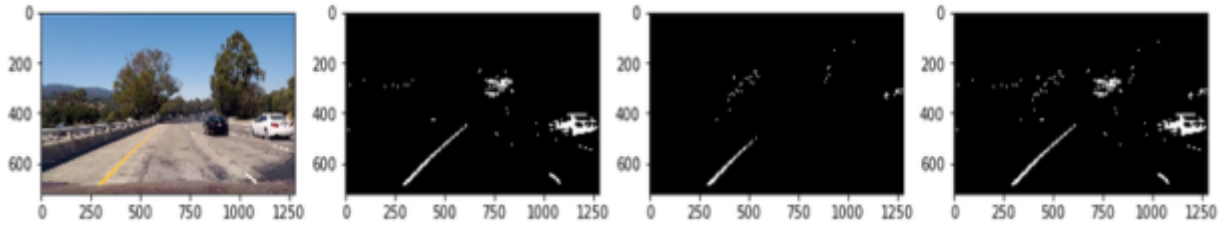
We first start by applying distortion correction on the input frame using the earlier computed camera calibration matrix and the distortion coefficients. The image below show the result of this step in the pipeline

Color Threshold to isolate lane lines

We then isolate the lane lines using image color thresholds. From learnings from Project 1 – the HSV (Hue Saturation Value) image format was found to be very fairly successful to isolate both the yellow and white lane lines using the following thresholds:

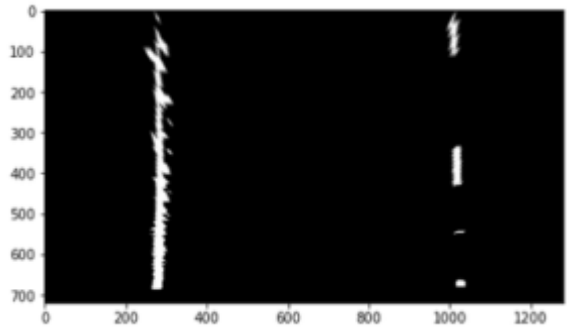
LANE COLOR	THRESHOLD LOW	THRESHOLD HIGH
White	0, 0, 225	180, 255, 255
Yellow	20, 100, 100	80, 255, 255

These thresholds are applied then combined to create a binary image which only contains the filtered/isolated white and yellow pixels. The image below shows the results of this step in the pipeline



Wrap Perspective

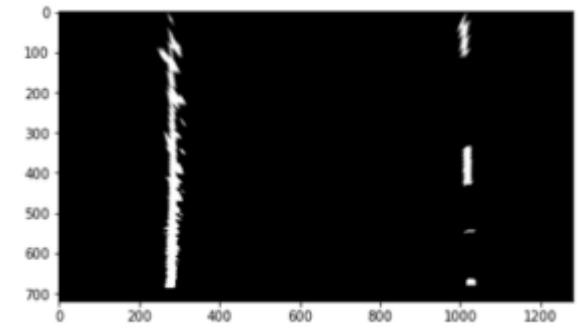
We then apply the wrap perspective transform using the earlier computed perspective transform matrix to get the birds-eye view on the binary image containing the isolated lane lines. The below image shows the result of this step in the pipeline

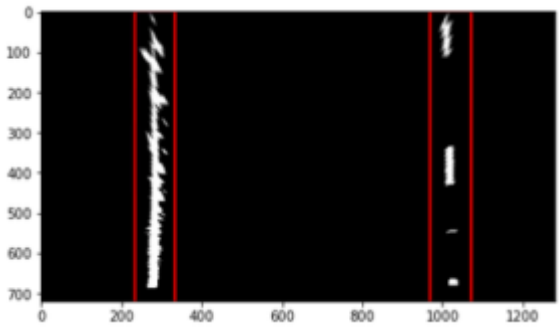


Find Lane Lines

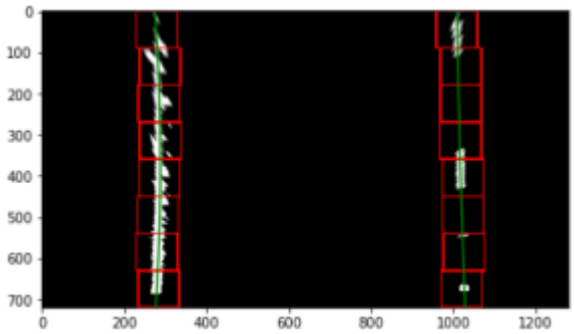
To find lanes - detect lanes in the first frame of the image and then use this location information in the subsequent frames to narrow/reduce the search space.

To detect lanes in the first frame we start with the wrapped lane lines mask from the prior step. We then compute the column level sum on the subset of the image to identify the location where lane lines are present. The below image shows the wrapped binary image and the corresponding initial search area/zone for lane lines

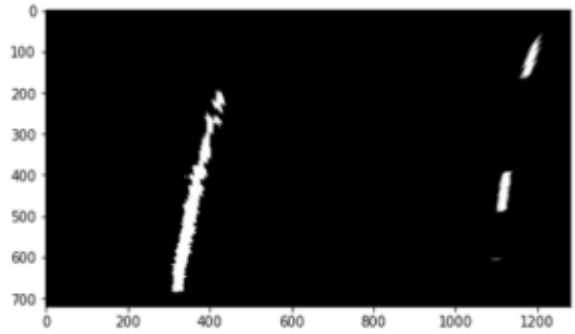


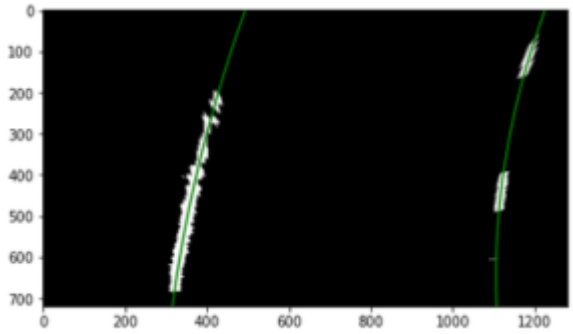


Once we determine the approx. search region for the first frame we segment the frame into 8 regions, and for each region we find the points for left lane and right lane and then update the search region based on the point locations. Once all the left lane and right lanes points are identified fit a polynomial. The below image shows the left and right search region for each slice/segment and the fitted quadratic polynomial for the found lane line points.



Once we have a quadratic polynomial equation for the left and right lane from the first frame, we use the fit equation to find lanes pixels on subsequent frames to reduce the search space and computation time/complexity. The fit equation for left and right lanes are updated based on the updated left and right lanes points from the current image. The below image shows the result of updated fit equation in which the left and right lane pixels where isolated using the polynomial fit from the prior frame.





Compute Curvature and Vehicle Position

We compute the left and the right curvature based on below equation:

$$\text{Radius of curvature} = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}}{\left|\frac{d^2y}{dx^2}\right|}$$

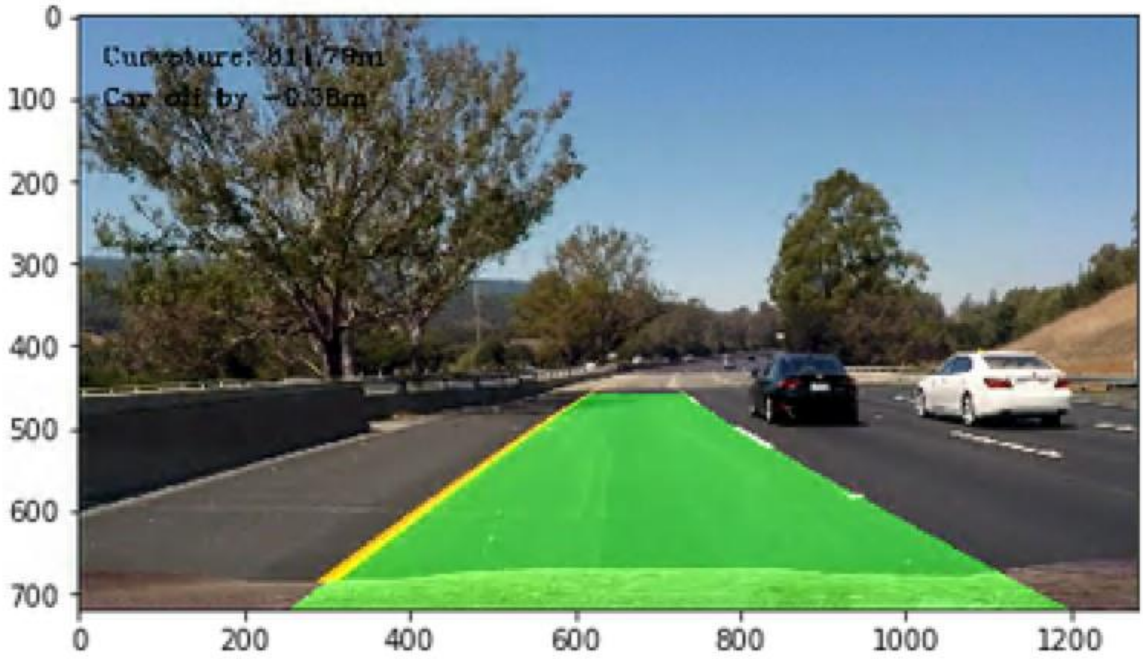
Source: http://www.intmath.com/applications-differentiation/8-radius-curvature.php

The overall curvature is determined by averaging the left and right values
To compute the vehicle position:

- average the X intercept for both the left and right lanes
- find the distance of the avg. x intercept value from center of image
- convert the distance from center from pixels to meters by multiplying with the ratio 3.7m/700px

Output Lane Boundaries, Lane Curvature and Vehicle Position on original image

We now generate the final output image by warping the lane line boundaries back to the original image using the inverse warp perspective matrix computed earlier. The estimated lane curvature and the vehicle position is also displayed for users' reference. The below image shows the result of this step of the pipeline.



Pipeline Result

Please refer to “project_video_solution.mp4” for result of the image processing pipeline on an input video file.

Discussion

The most challenging part of this project was to determine thresholds (safe bounds) to avoid updating the left and right fit equations when the lane detection/points got unstable (no lane detected, noisy points etc.). I finally settled on a minimum number of points/pixels required to be 150 before a previously computed lane polynomial fit equation is updated. As a future enhancement, the following things can be considered:

- use a combination of additional image filtering techniques such as Histogram equalization/Sobel derivative/RGB Image thresholds etc. (along with the HSV thresholds) to ensure that left/right lanes can be isolated prior to fitting
- use a running average of previous 5-10 frames on the left and right polynomial fit to dampen any wild swings in the lane boundaries.

As final observation, the use of computer vision techniques to isolate the lane lines still does not solve the problem of snowy conditions/night conditions (where the lane lines are not visible) and would need reliance on other techniques such as better mapping solutions, use of historical information of detected lane lines, machine learning/deep learning to provide a more general purpose solution.