Advanced Lance Finding Project

By Ahmad Chatha

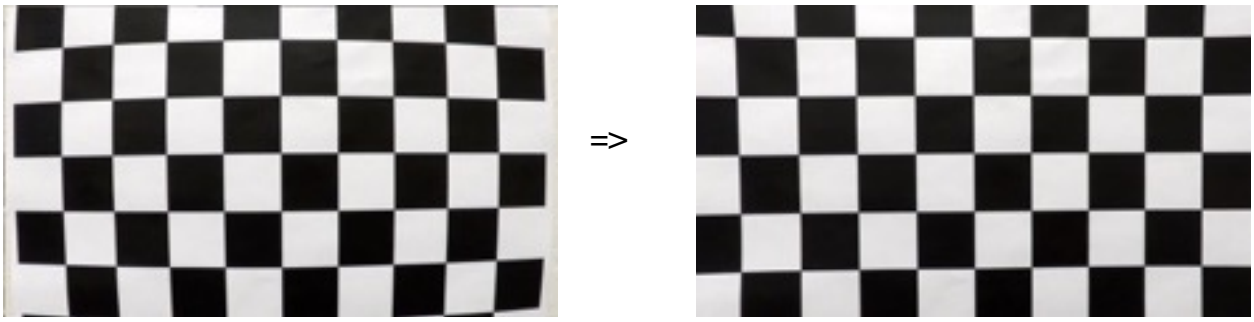
Introduction: The goals / steps of this project are the following:

- 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.

Camera Calibration: The camera calibration was performed following the steps mentioned in the lecture. The code for camera calibration is in *calibration.py* more specifically the function perform_calibration() which uses the objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. Then the cv2.undistort() function was used to undistort the road images.

Here is the result for undistorted chess image:

Original Undistorted



Distortion Correction: Using the cv2.undistort() function the road images were also undistorted as the first step of the image pipeline:

Thresholding:

Various aspects of your gradient measurements (x, y, magnitude, direction) were used to isolate lane-line pixels as mentioned in the lecture. More specifically, the thresholding process involves the following steps (function can be found in *util.py*):

- Thresholds the S-channel of HLS - hls_select() [L-79]
- Calculate directional gradient - abs_sobel_thresh() [L-8]
- Calculate gradient magnitude - mag_thresh() [L-28]
- Calculate gradient direction - dir_threshold() [L-47]

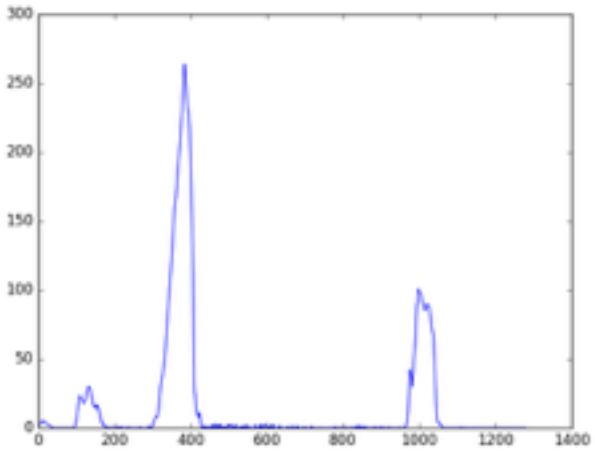
The above functions were combined in one function called *combine_threshold() [L-62]*. Following is an example from undistorted image to binary image:

=>

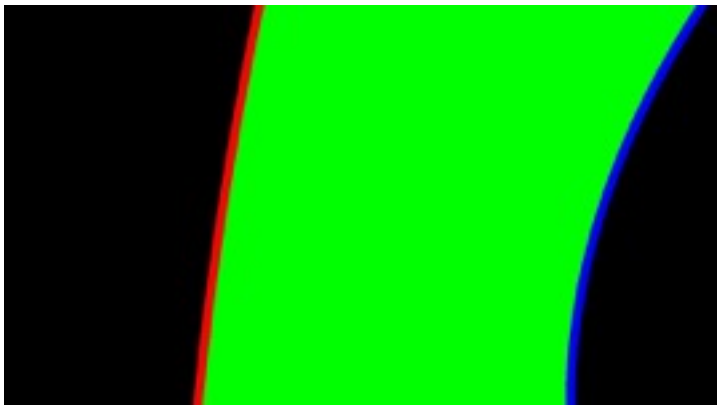
Perspective transform:
Perspective transform was using the cv2.getPerspectiveTransform() function by choosing the source [[599,450], [678,450], [240,715],[1060,715]]
and destination points [[355,0], [940,0], [355,715],[940,715]] which were found manually. The code for all this is in utils.py inside perspective_transform() function (L-86). Following is an example of the image after perspective transform:

Identifying lane-line pixels and fitting polynomial:

In order to identify the lane lines, we first generate a histogram along all the columns in the lower half of the image and then perform a sliding window search placed around the line centers, to find and follow the lines up to the top of the frame.
The code for for this is in the function find_lines() [L-92] in utils.py
Following is an image of the histogram generated where the peaks identify the lanes:

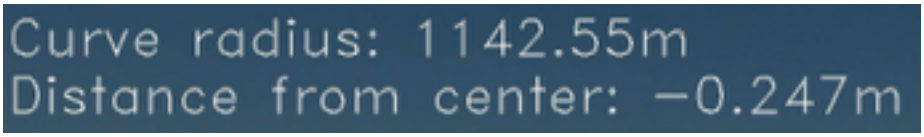


Following is the plot of the image after the lanes have been identified:

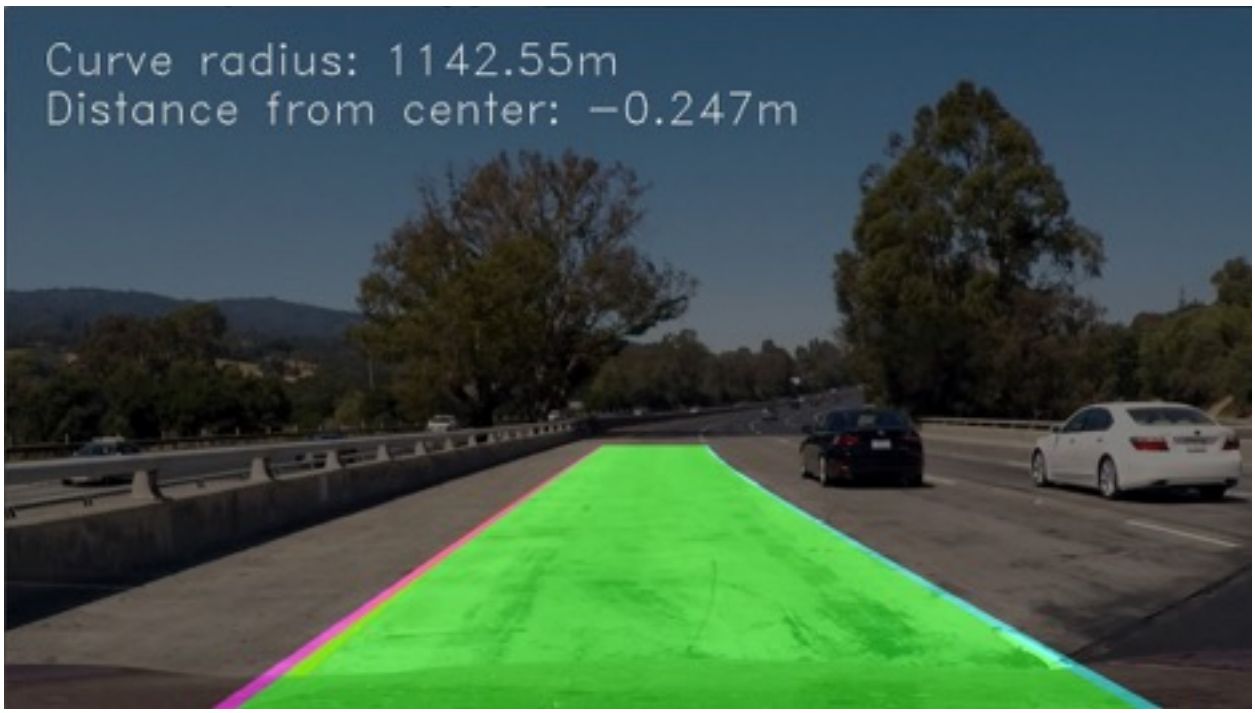


Radius of curvature of the lane and the position of the vehicle:

The radius of the curvature and distance from the center were calculated in calculate_radius_dist() function [L-165] in utils.py as described in the lecture. To find the radius of the curvature, we used the function described in http://www.intmath.com/applications-differentiation/8-radius-curvature.php which returns the radius in pixels and they were converted into meters by following the code described in the lecture.
An example of the measurements are :



Final Result: Finally we warp back the transformed image with identified lanes back onto the original image and write the curvature and distance values on it. Following is an example:



Video:

The same process was repeated on all the frames in the video and the result can be viewed at https://youtu.be/5SBro3f07LI
or can be downloaded from github at https://github.com/ahmadchatha/CarND-Advanced-Lane-Lines

Discussion/Conclusion:
Shadows and the faded lines presented some challenges. For example, consider the following image:

Because part of the white line is missing on the right, we were not able to identify the lanes correctly. Another issue is when there is an additional line due to different road structure (road repairs) as can found in the challenge video which totally throws the algorithm off.

In order to improve the algorithm we can possible make the lane finding more robust by keeping a recent history of best fits and when we are not able to detect lanes we can do some sort of extrapolation to identify the lanes.