Writeup Template

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.

The solution is provided in the Jupyter Notebook Advanced_Lane_Lines.ipynb

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

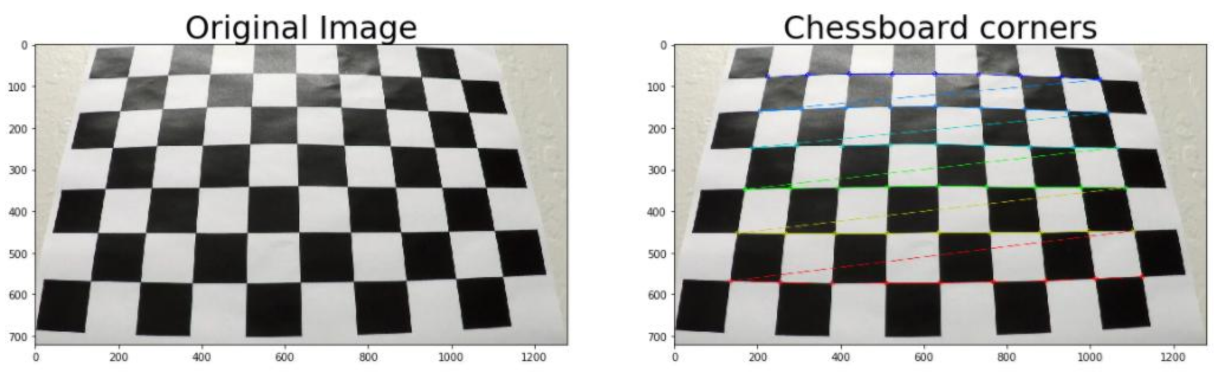
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

You're reading it!

Camera Calibration

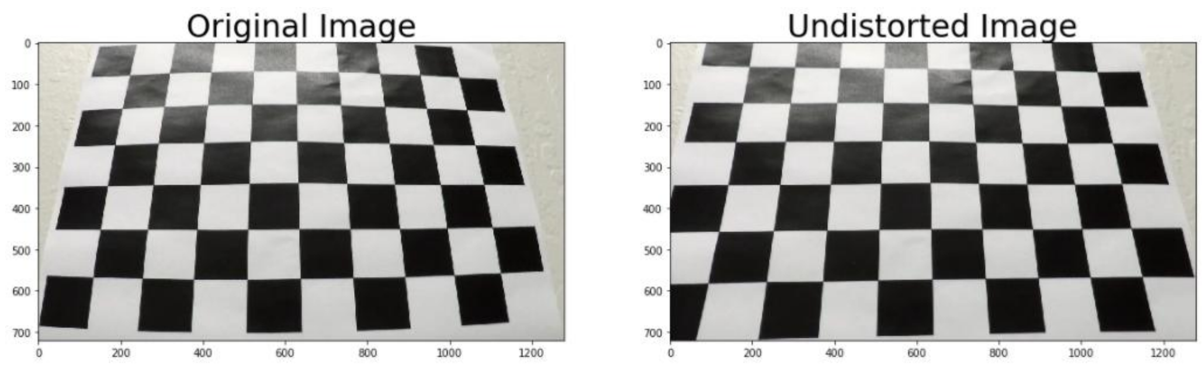
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

I started by testing the OpenCV functions cv2.findChessboardCorners() and cv2.drawChessboardCorners() to detect and plot corners on a provided image showing a chessboard calibration image.



Then I defined a function calibrate_camera() which reads in all 20 provided calibration images and builds up two lists: one list for the (x, y, z) coordinates of the chessboard corners in the world (objpoints) and one list for the detected (x, y) pixel positions of the corners in the image plane found by cv2.findChessboardCorners(). Since the provided calibration images show the same calibration chessboard, all entries in objpoints are the same. Furthermore, I assume that the chessboard is fixed on the (x, y) plane at z=0, so that the z values are all zero.

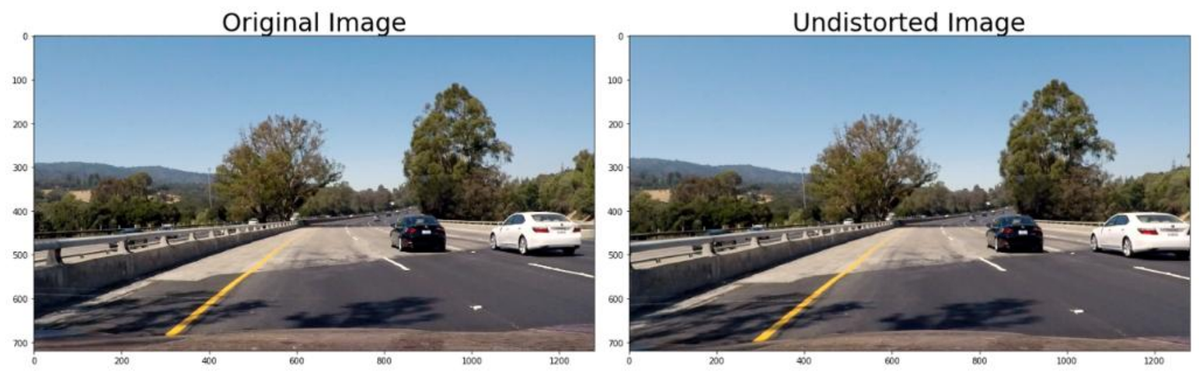
I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:



Pipeline (single images)

1. Provide an example of a distortion-corrected image.

To demonstrate the camera calibration I tested the function calibrate_camera() on an image showing a highway. You can see, that the white car on the right side is shifted a little bit to the right in the undistorted image.

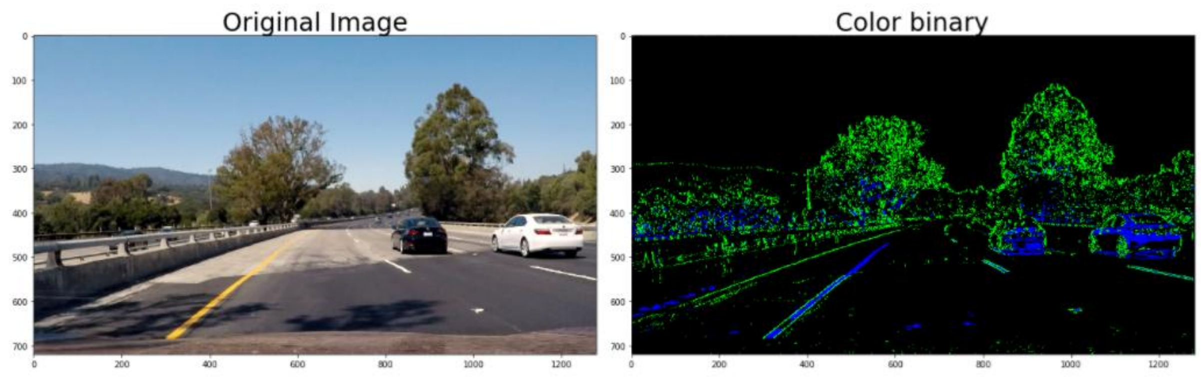


2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

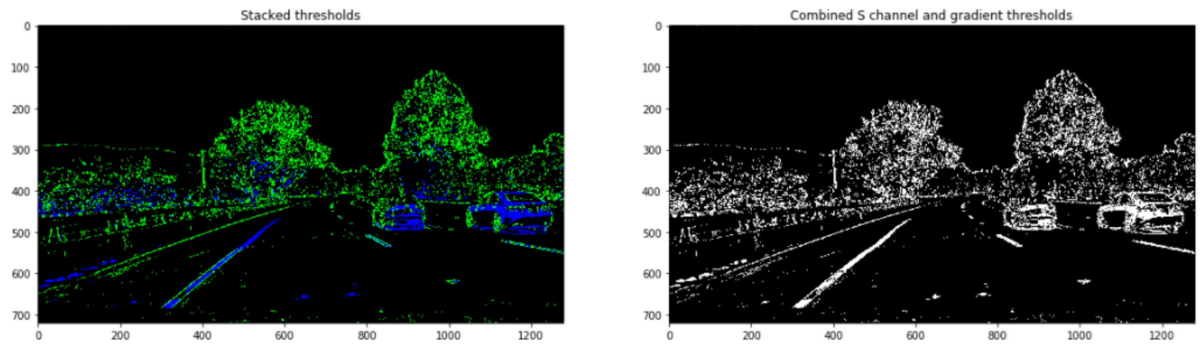
I used a combination of color and gradient thresholds to generate a binary image.

First, the original image is converted to HSV color space. Then cv2.Sobel() is used on the V channel to take the derivative in x direction. A threshold is used to select locations in the image having a high change in brightness. Second, a threshold is used on the S channel.

The results are shown in the following image, where the green color shows the threshold on the x gradient und the blue color shows the threshold on the S channel.



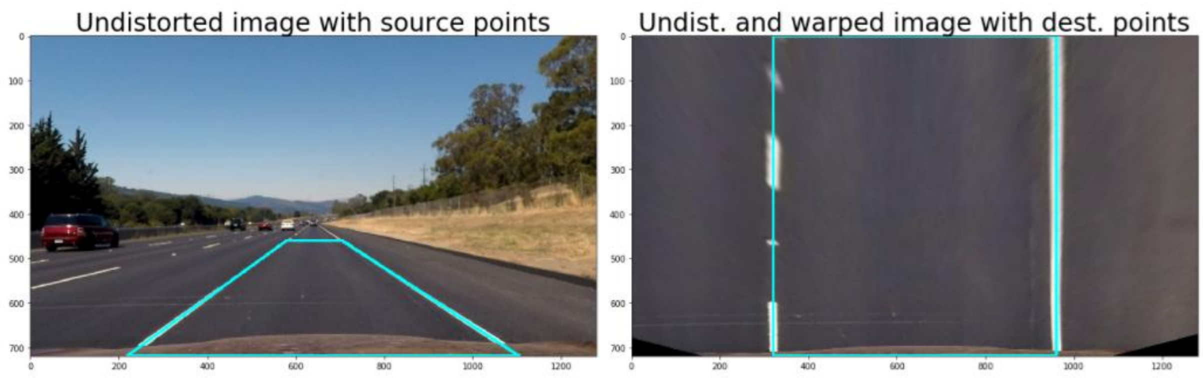
In the next step I combined these two effects to provide a binary image.



3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

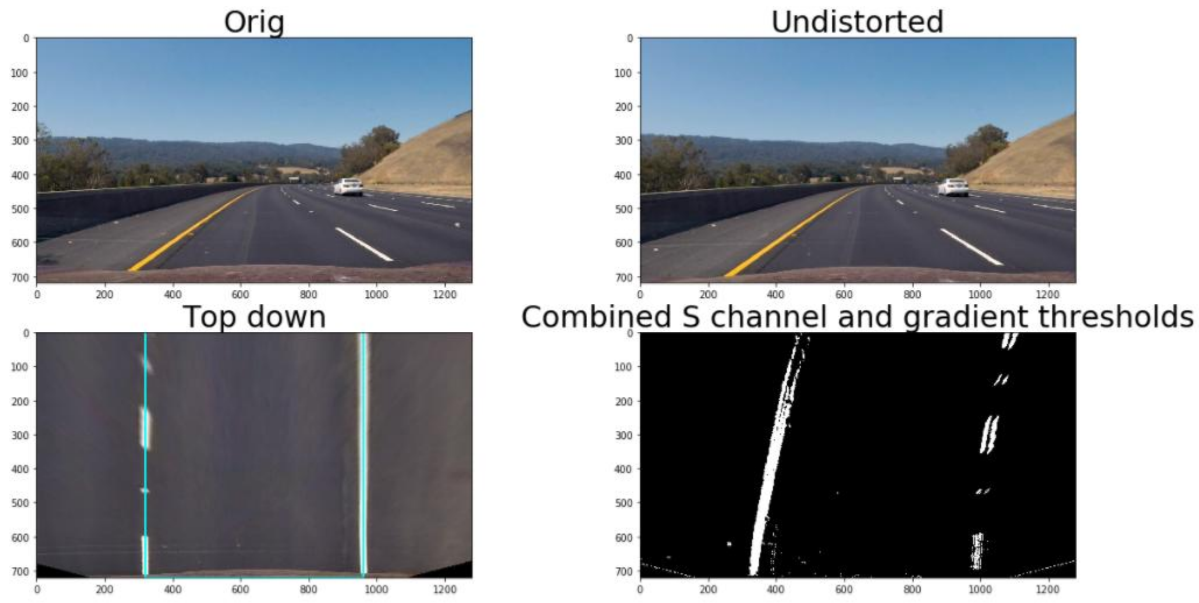
To do the perspective transform, I defined a function warp() that calculates the transformation matrices with the help of 4 hard coded source and destination points and then transforms the image to a top-down view (Bird's eye view).

I verified that my perspective transform was working as expected by drawing the source and destination points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.


Undistorted image with source points
Undist. and warped image with dest. points

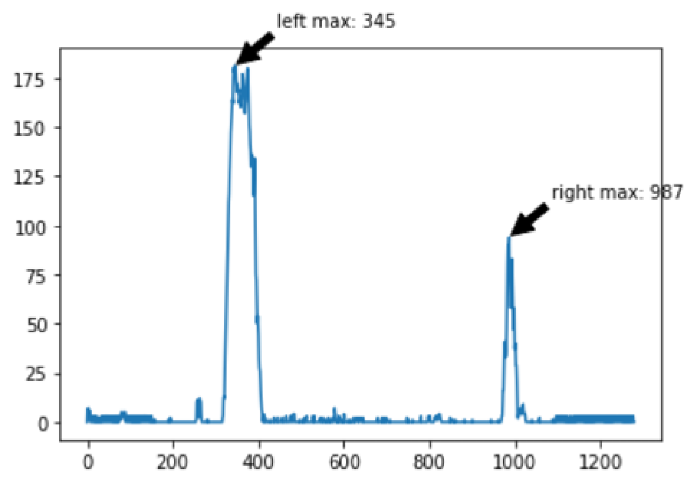
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Then I defined a function get_warped_binary() to warp the binary image to a top-down view.



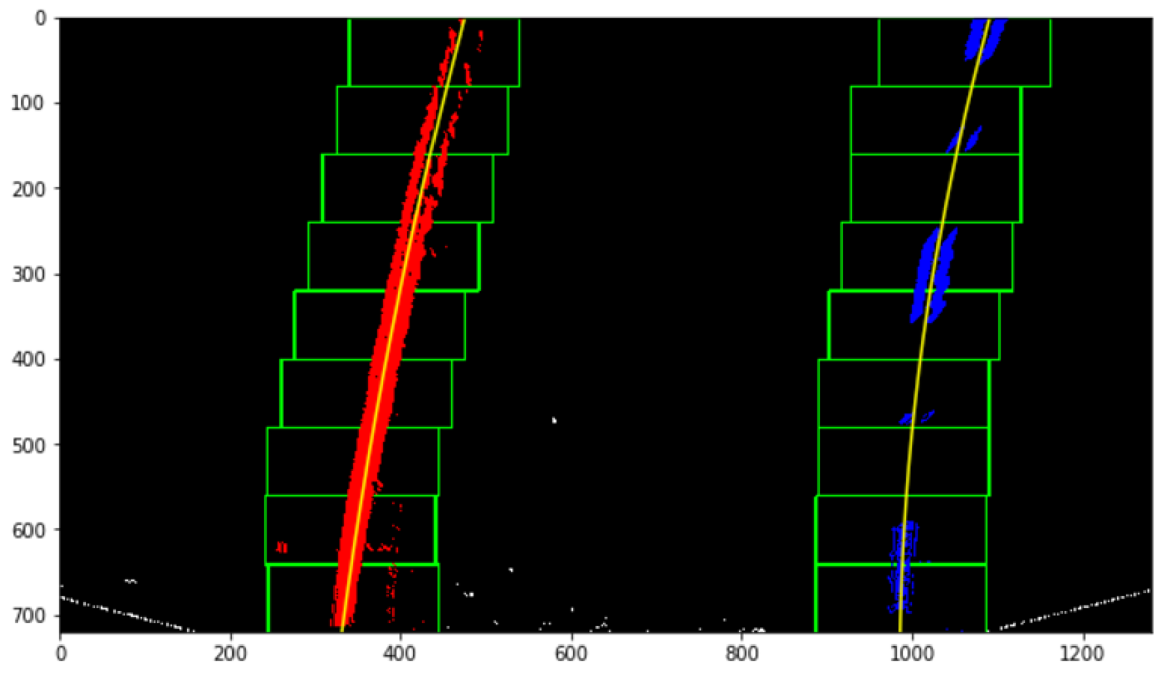
Then I used a sliding window search on the warped binary image.

As a starting point for the left and right lines I used a histogram to detect the peaks of the left and right halves.



The height of the image was divided into 9 parts. For every part a left and right window (margin +- 100 pix) was created. In each window the x position of the line was detected by calculating the mean of the nonzero pixels in the window. The x positions of all windows where used to calculate the coefficients of a second order polynomial.

Here is the visualization of the sliding window search.



5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I defined a function meas_curvature_offset() to measure the curvatures of the lines, the offset of the car to the lane center and the lane width.

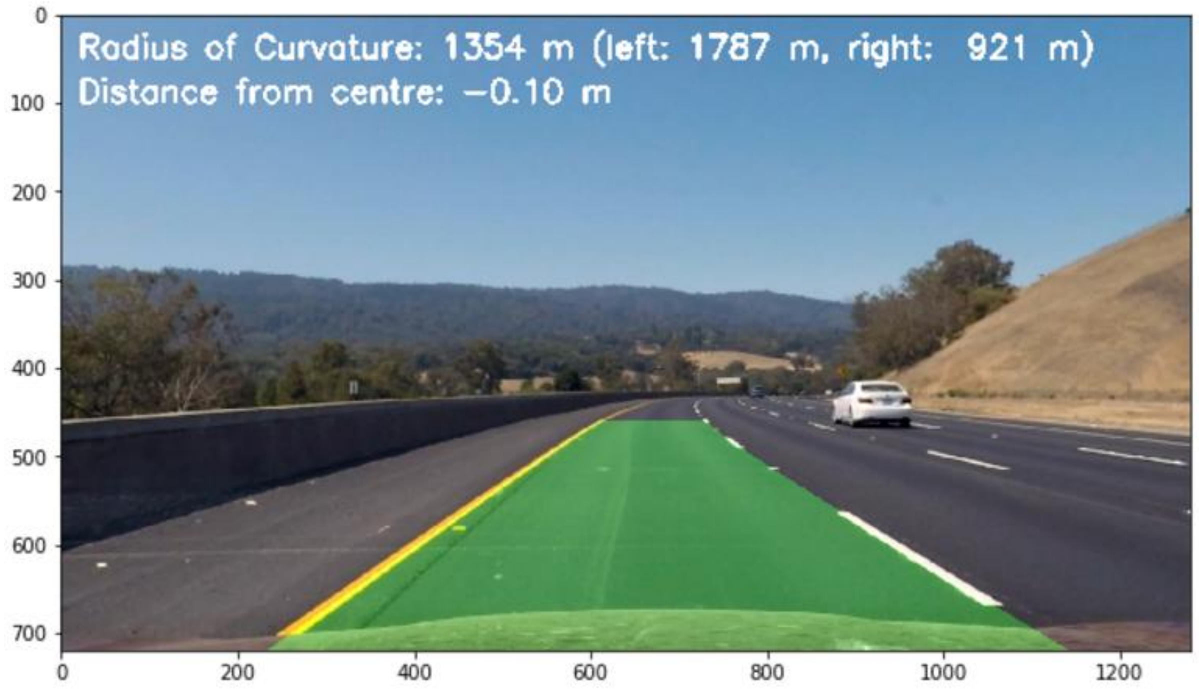
The calculated values for the image above were:

- curvature left: 1787.7 m, right: 921.9 m, mean: 1354.8 m
- vehicle offset: -0.10 m
- lane width: 3.46 m

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I implemented a function draw_lane() that draws the lane on the warped image and then warps the image back to the original object space using inverse perspective transform.

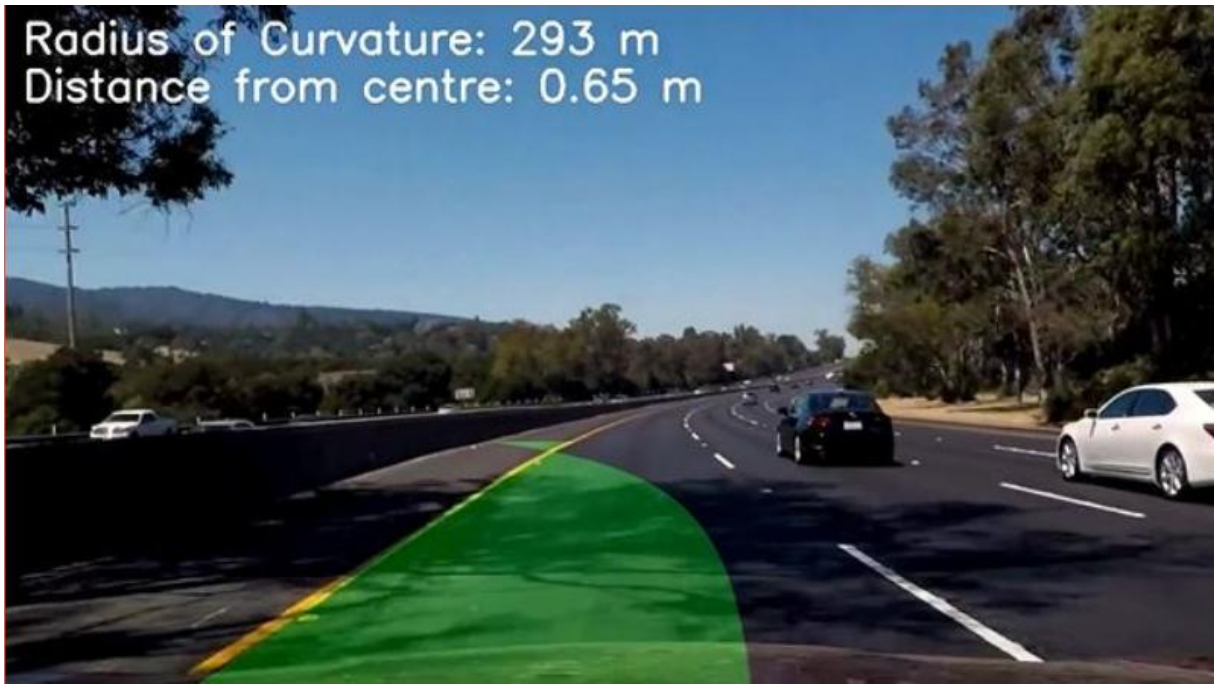
Here is an example of my result on a test image:


Radius of Curvature: 1354 m (left: 1787 m, right: 921 m)
Distance from centre: -0.10 m

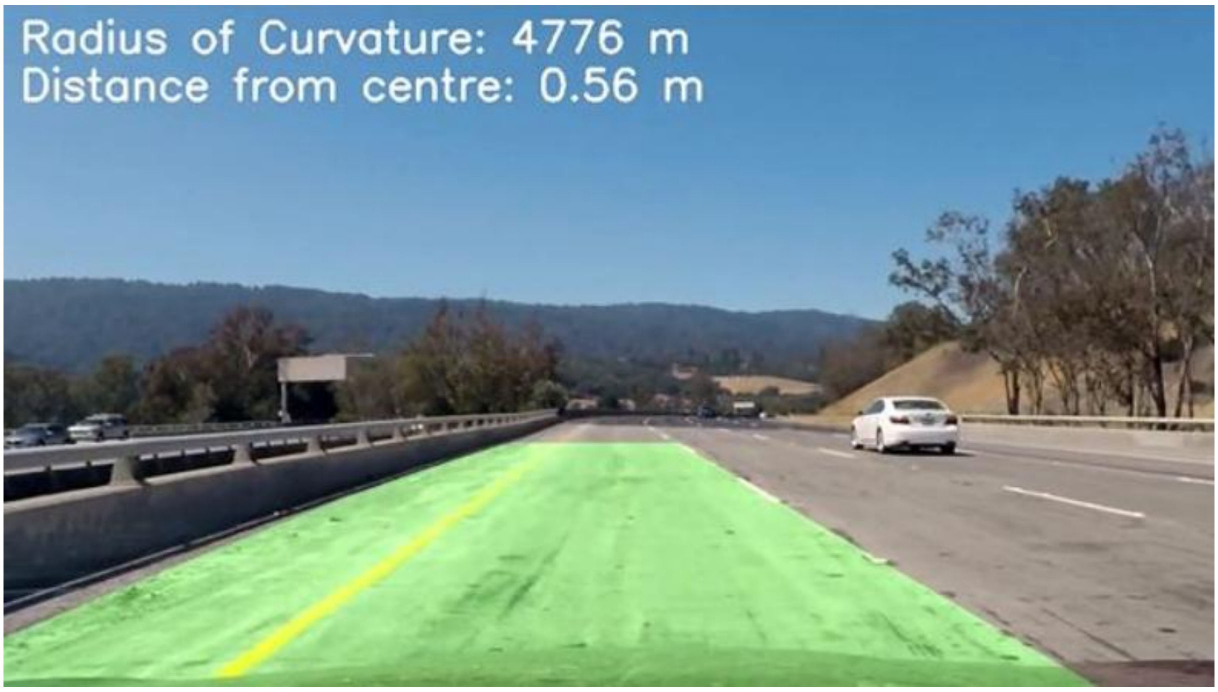
Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

I build a pipeline that made use of the functions described above. I compiled a first video (project_video_processed.mp4) that showed some failures when the light conditions on the road were changing between light and dark due to shadows or when the change in lightness/color due to the yellow line was weak compared to the change at the road boundary.



Failure due to shadows



Failure in presence of strong transition of lightness/color at road boundary

To prevent the failures I introduced a class 'Line' to store values between frames, in order to have fallback values in the case that the lines detected in a frame fail in a sanity test. Furthermore, I calculated mean values of the fitted lines of the last 5 frames.

The second video shows the improved pipeline (project_video_processed_improved.mp4).

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

The pipeline works well if the lane lines have a strong contrast, whereas the detection of the lines gets shaky in presence of shadows or strong variation of brightness at the road boundaries.

The pipeline used fixed gradient and color thresholds. A pipeline with dynamic thresholds could possible perform better.

Furthermore, in the sanity test, I used fixed thresholds for the curvature (curverad < 300) and for the lane width (lane_width < 3.0 or lane_width > 4.0). This is ok for highways, but would probably not be applicable for other roads like country lanes. More knowledge about the road, e.g. from a High-Definition-Map (HD map) could deliver better thresholds.