Advanced Line Finding – P4 – Anna Busquet

Files complementing the submission:

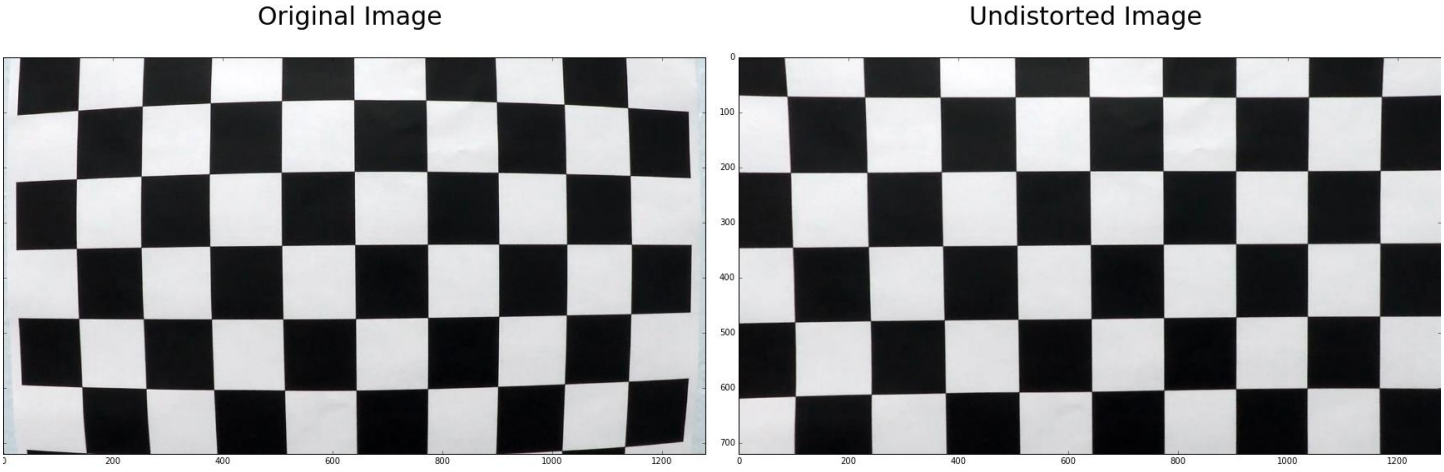
- **advanced_lane_finding.ipynb**: Jupiter notebook with all code
- folder output_images, with all output examples included in that report and some others
- project_video_output.mp4, the video with lane area detected

CAMERA CALIBRATION

In Section 1 of **advanced_lane_finding.ipynb**, I calculate camera matrix and distortion coefficients, using calibration chessboard images provided in the repository. We are given 20 9x6 chessboard images.

I use the function cv2.findChessboardCorners() to get all corners for each chessboard picture. If corners are successfully found, we keep these in the array imgpoints (2D-array), and a set of 3-D points (in the plane z=0), corresponding to each chess corner in a real 3-D world.

Using function cv2.calibrateCamera(), object points and image points, I get the camera calibration and distortion correction. With function cv2.undistort(), we can undistort some of the examples, getting:



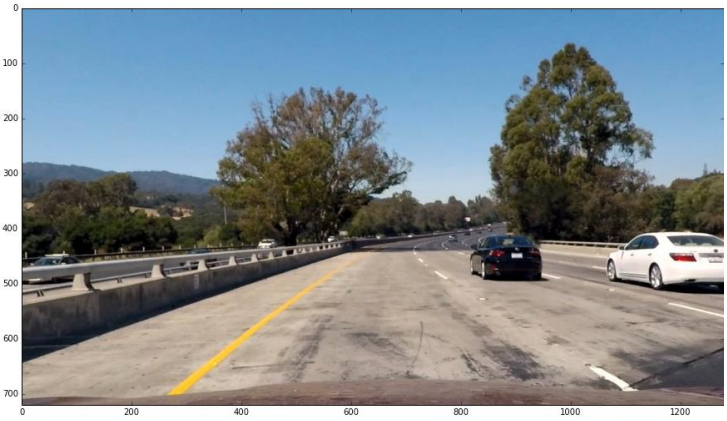
DISTORTION CORRECTION

The final pipeline perform a distortion correction just after reading the image, with function cv2.undistort(img, mtx, dist, None, mtx)

We can find the code for a concrete example of the undistorting process in the Section 2, that gives us plots:

Original Image

Distortion corrected



PERSPECTIVE TRANSFORMATION

(That section has been improved since the first submission, old pictures can be found in the Annex part of the document, pages 12 and further.

Main changes after first submission: move slightly source and destination points to get better parallel lines)

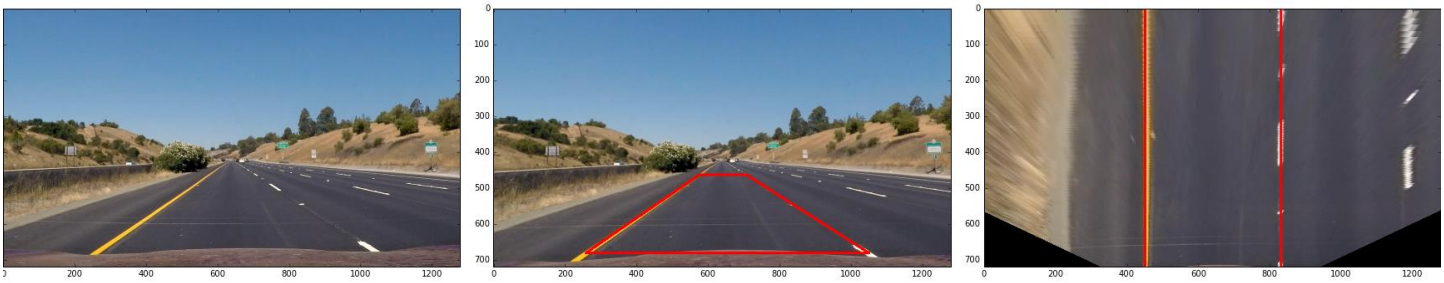
In section 3, I define the function perspective_transformation(img, src, dst), that given 4 points in a picture frame with straight lines (source points) and the corresponding expected projection in a “birds-eye view” (destination points), returns the warped image, the perspective transformation matrix M, and the inverse matrix, Minv.

Chosen points to perform the final transformation are:

source points (src)	destination points (src)
(**575,464**)	(**450,0**)
(**258,682**)	(**450,720**)
(**1049,682**)	(**830,720**)
(**707,464**)	(**830,0**)

Examples of undistorted and warped images, given a pair of original straight lines images, (old examples, can be found in the Annex part):

Original Image Undistorted Image warped image



Original Image Undistorted Image warped image

COLOR TREATMENT AND SOBEL OPERATOR TO GET BINARY IMAGES

(That section has been improved since the first submission, old pictures can be found in the Annex part of the document, pages 12 and further.

Main changes after first submission:

- *Apply perspective transformation before color treatment*
- *Improve color treatment to isolate better yellows and whites, thresholding in the Lab and HLS color spaces)*

In the Section 4 I introduce the process to get a binary image that I will use in the final pipeline.

Advanced Line Finding – P4 – Anna Busquet

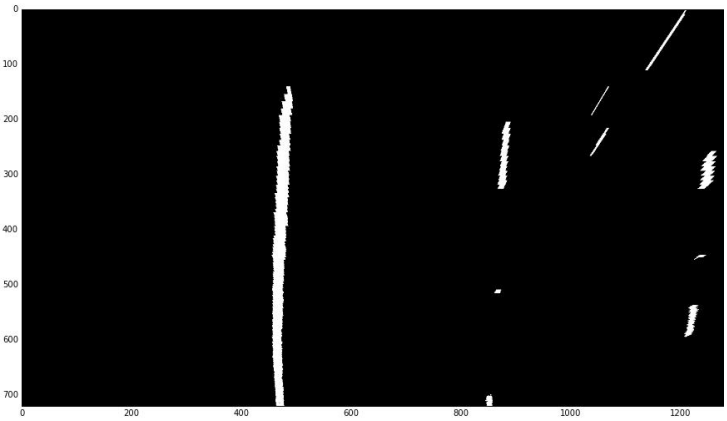
After checking different color spaces (RGB, HSL, LAB…) and Sobel transformations, the process to get better binary output I could get is as follows:

1. Apply perspective transformation
2. Transform the RGB color space to HLS, applying some threshold to L channel. It will be useful to detect white lines.
3. Transform the RGB color space to Lab, applying some threshold to b channel. It works well for yellow lines
4. Combine the two binary thresholds, getting a binary image.

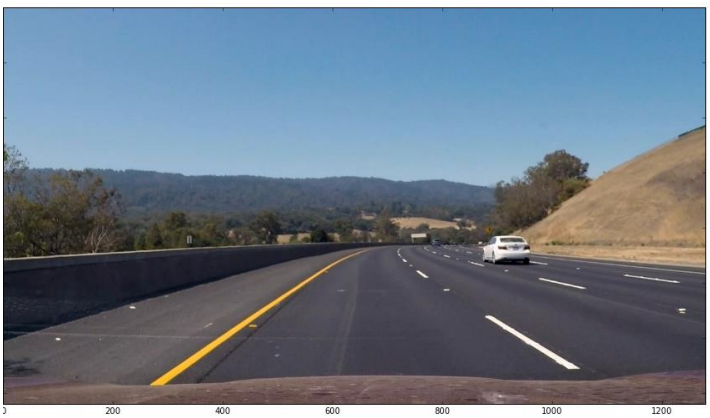
Examples of binary images obtained after thresholding L-channel and b-channel (examples from previous submission could be found in the Annex):

Original Image

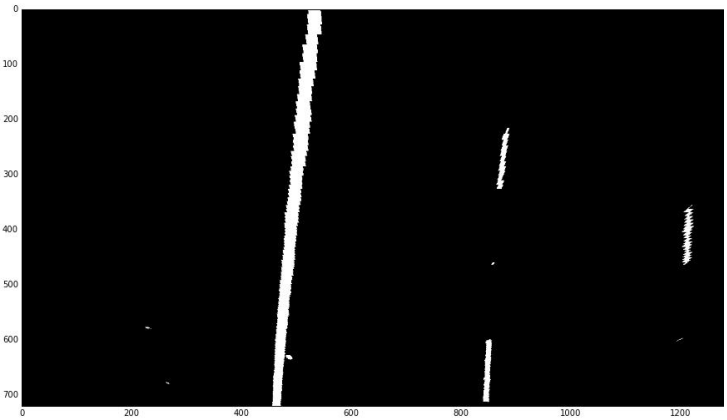
Pipeline Result with perspective transf.



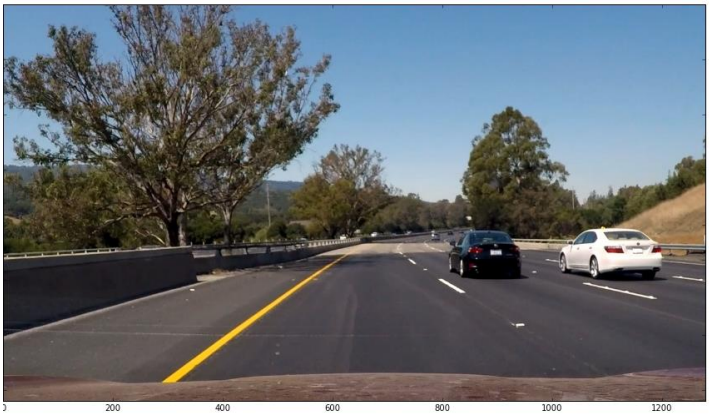
Original Image



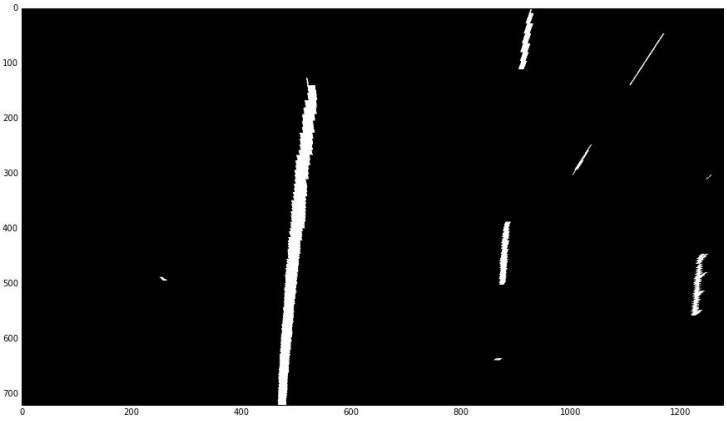
Pipeline Result with perspective transf.



Original Image



Pipeline Result with perspective transf.



Binary images got in before iterations, with other space/sobel combinations in the Annex, pag

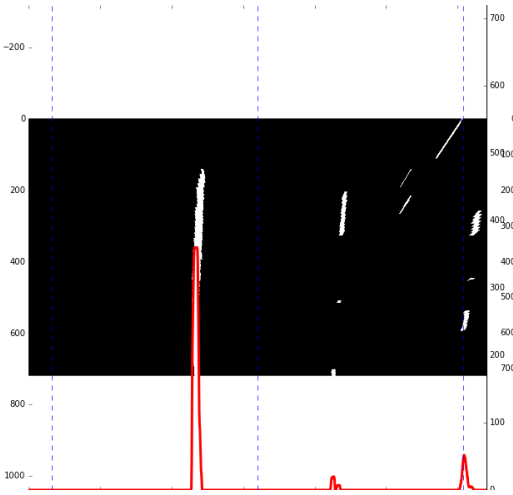
FINDING LANE LINES METHODS

The main method used to fit a polynomial is the one detailed in the Udacity class, and implemented in the function find_lines(binary_warped) in the Section 5.
Here what we do is given a transformed binary image, detect all nonzero points in a certain collection of sliding windows and fit a second order polynomial to each collection of points, for left line and for right line.
The way to detect each sliding boxes is starting from the bottom and recentering each boxes to the mean of nonzero points detected higher than a certain number of pixels threshold of 50.
To center the first box, we calculate a histogram of the ones pixels for the half bottom of the image, discarding any ones detected in the 100px left or right image margin. We detect the x position for the maximum of ones in the left half of the binary image, and the x position for the maximum of the histogram on the right half (removing 100px of margins).

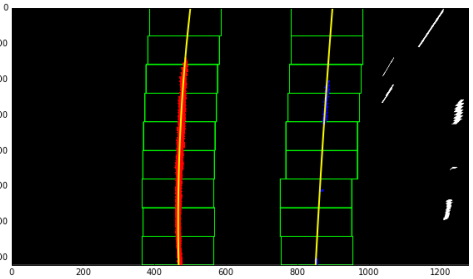
That method gives line lanes like that (in yellow in the third image for each example):
Examples from first submission in the Annex, page 12 and further

Original Image

Pipeline Result with perspective transf.

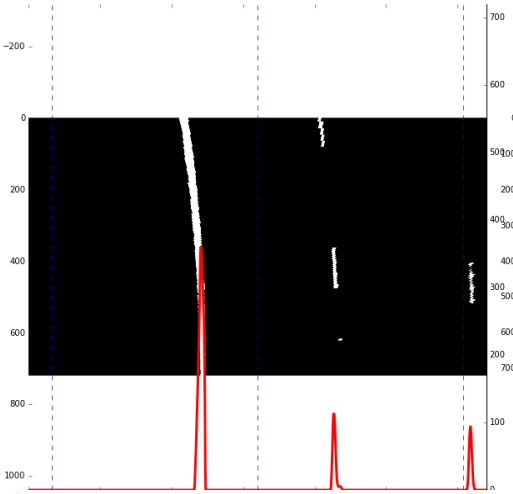


Polynomial interpolation

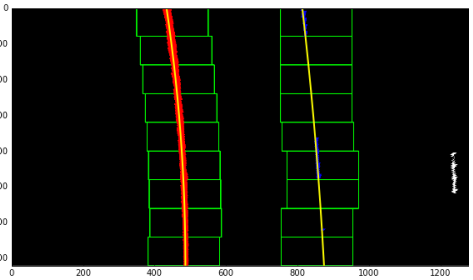


Original Image

Pipeline Result with perspective transf.

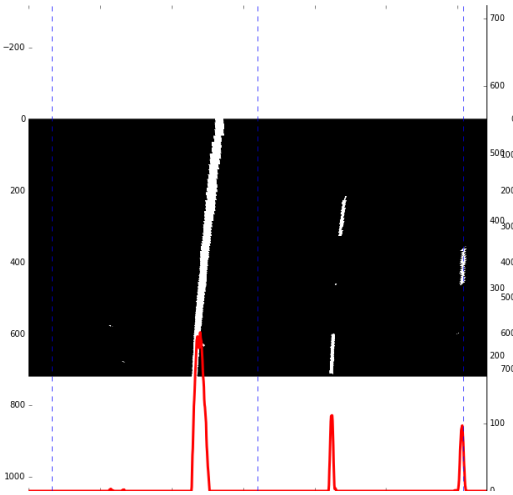


Polynomial interpolation

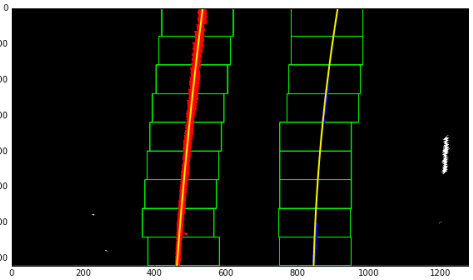


Original Image

Pipeline Result with perspective transf.

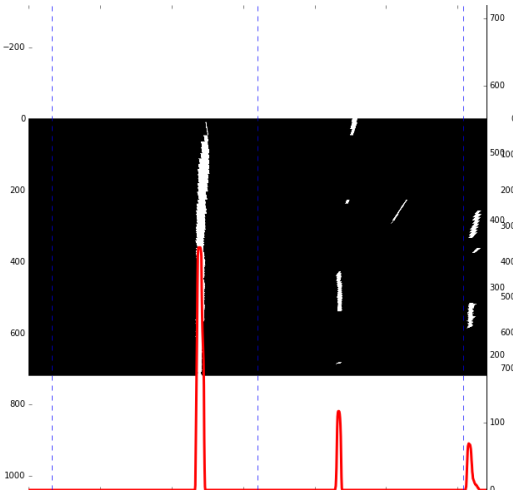


Polynomial interpolation

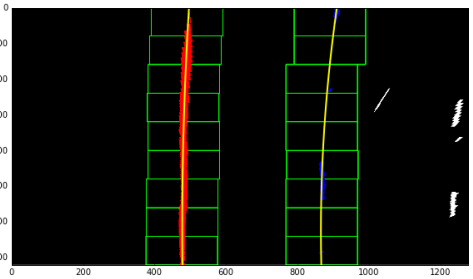


Original Image

Pipeline Result with perspective transf.

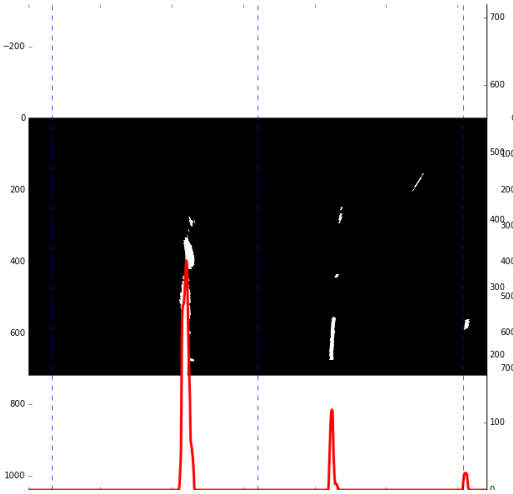


Polynomial interpolation

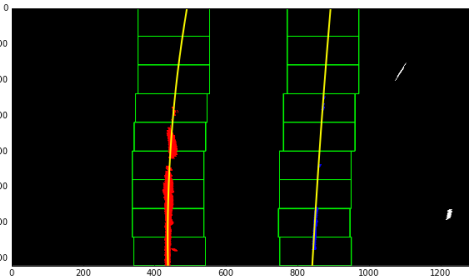


Original Image

Pipeline Result with perspective transf.

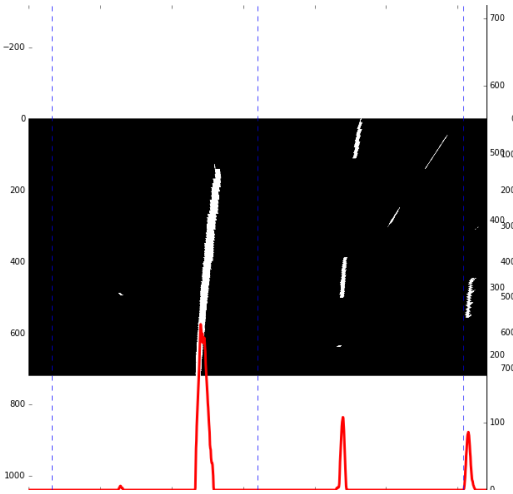


Polynomial interpolation

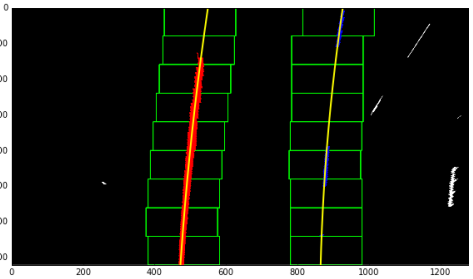


Original Image

Pipeline Result with perspective transf.



Polynomial interpolation



In the final pipeline method, we will combine that line detection method with another that allows us to base the new fitting from the last curve found in the previous frame, within a certain margin, rather than the new calculated one, if we detect that that last is not good enough.
For example, if detected lines are not parallel, or detected lines end crossing each other, or any of the calculated curvature has really low values (very pronounced curve), we will base the new polynomial fit on the obtained in last frame.
The function that performs these sanity checks is detected_lines_sanity_check, and the function that calculates lines in the other way is find_lines_from_prev_frame(binary_warped, left_fit_prev, right_fit_prev)

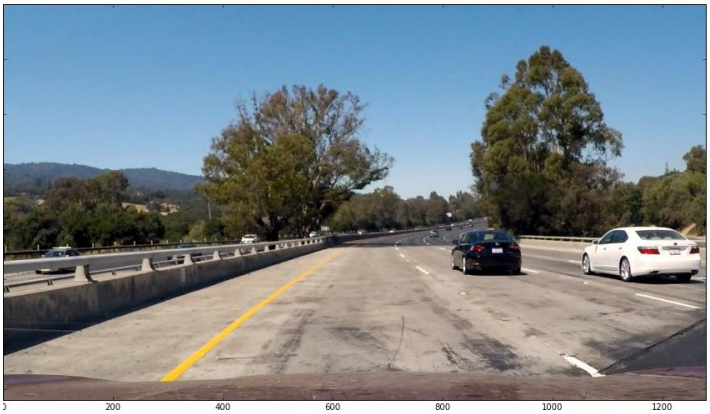
GET RADIUS OF CURVATURE AND SHIFT VALUES FROM THE CENTER OF THE LANE

In section 5 there is a pair of functions, get_curvature(leftx, lefty,rightx, righty,y_eval) and get_offset(left_fit,right_fit,y_eval,w) to calculate radius of curvature (in meters) for each lane line, and the lateral shift of the middle of the car over the center lane.
In both case the transformation between pixels and meters is taken as:
30m. each 720 px. in y dimension and 3.7m. each 700 px. in x dimension

After calculating radius and shift, and found the line lanes, we have to plot the detected area back to the original image. That is done in the function print_onto_road(image, binary_warped, left_fit, right_fit, Minv).
It basically creates a new “birds-eye” image, where the delimited detected lane is drawn, and later warped to the original image space using the inverse transformation perspective matrix, Minv, and the method cv2.warpPerspective().

Some examples with the lane detected, where we can see the average curvature radius and the car lateral offset from the center of the lane:

Original Image

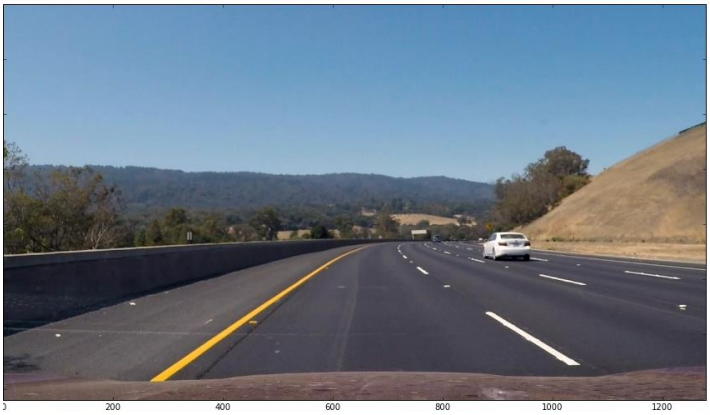


Lane detected

Original Image

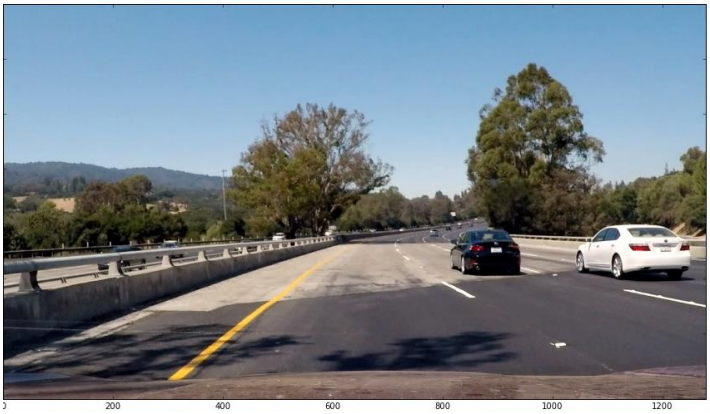
Lane detected

Original Image



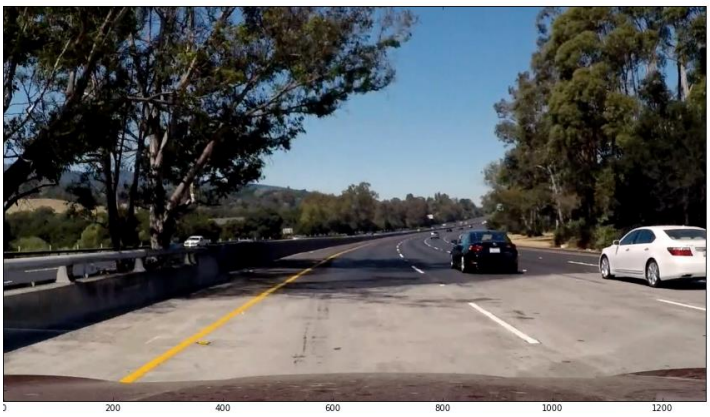
Lane detected

Original Image



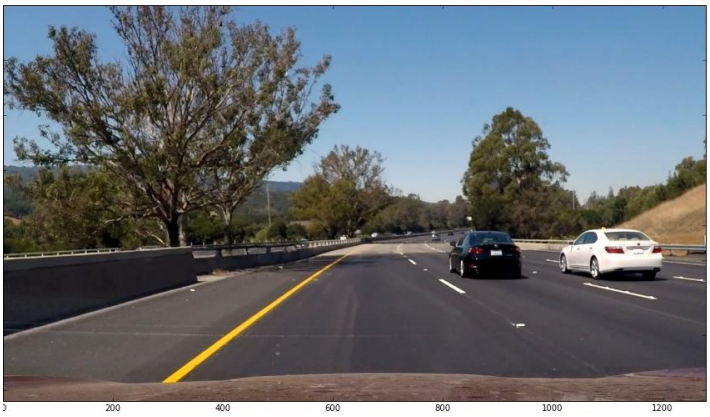
Lane detected

Original Image



Lane detected

Original Image



Lane detected

ALL TOGETHER AND FINAL DISCUSSION

The final pipeline, incorporating all the process discussed before, gets summarized as:

For each image,

- (1) Camera Distortion correction
- (2) Perspective transformation
- (3) Get binary image (applying thresholding in HLS and Lab channels)
- (4) Line finding method from sliding windows, in function find_lines()
- (5) Perform a sanity check to accept or discard that current polynomial fitting, in function detected_lines_sanity_check()
- (6) If the line is not good enough (applying some criteria), we calculate the lane line based in the last frame line lanes, in function find_lines_from_prev_frame()
- (7) Print lane onto road, unwarping the image, in function print_onto_road()

Before implementing (5) and (6), and just with the sliding window method to detect lane lines, I found some frames with bad line detected, such that:

For first submission, there were clear problems with some white lines corresponding to some frames

Most of that problems were solved with better white line detection, using L-channel thresholding from HLS color channe.

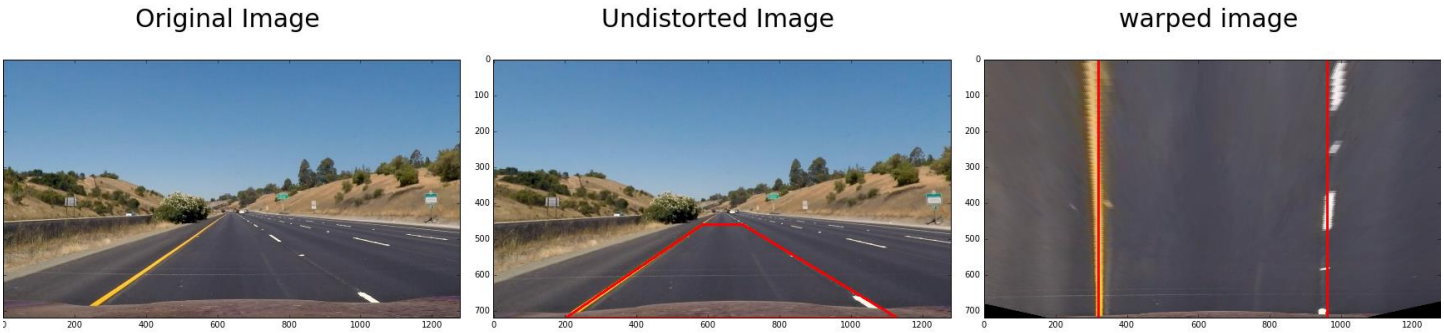
A part from that, I decided to implement some sanity check on the lines found:

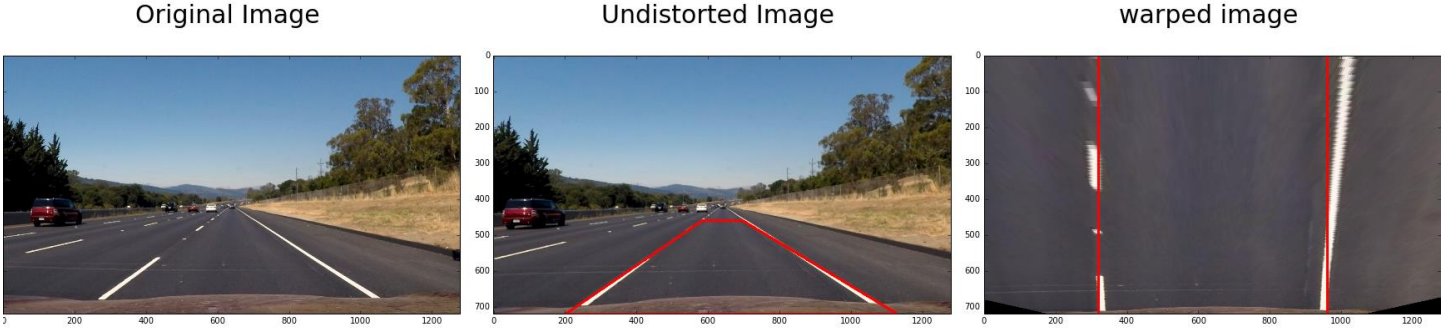
- Discarding all left and right lines that ended crossing each other in the image.
- Discarding lines that were not parallel enough, concretely, where the distance between left and right lines in the bottom of the picture was much different from the distance between these in the top margin, in more than 150 pixels.
- Discarding, if detected extremely curved lines, with radios lower than 450m.

ANNEX.

Here I am showing different set of images obtained for first submission, corresponding to different project parts that have been improved:

<u>Old examples from perspective transformation:</u>



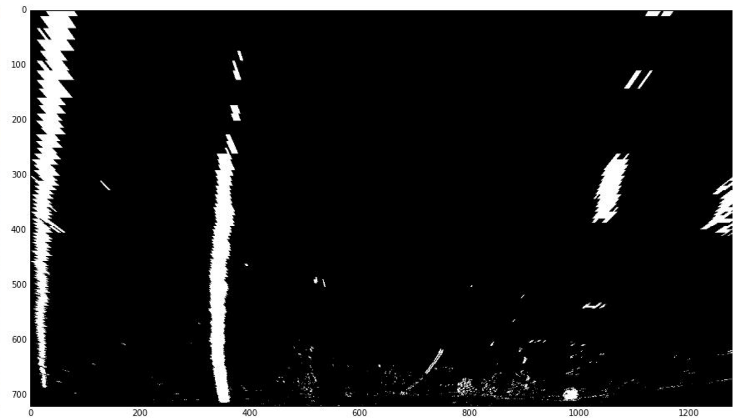


<u>Old examples of binary images obtained after thresholding S-channel and x Sobel:</u>

For first submission, the perspective transformation was applied after color treatment. It seems that works better in the other way, and it has been changed.

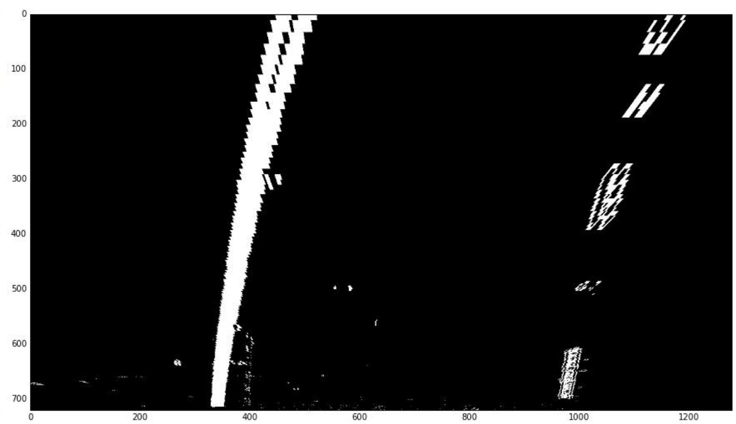
Original Image

Pipeline Result with perspective transf.



Original Image

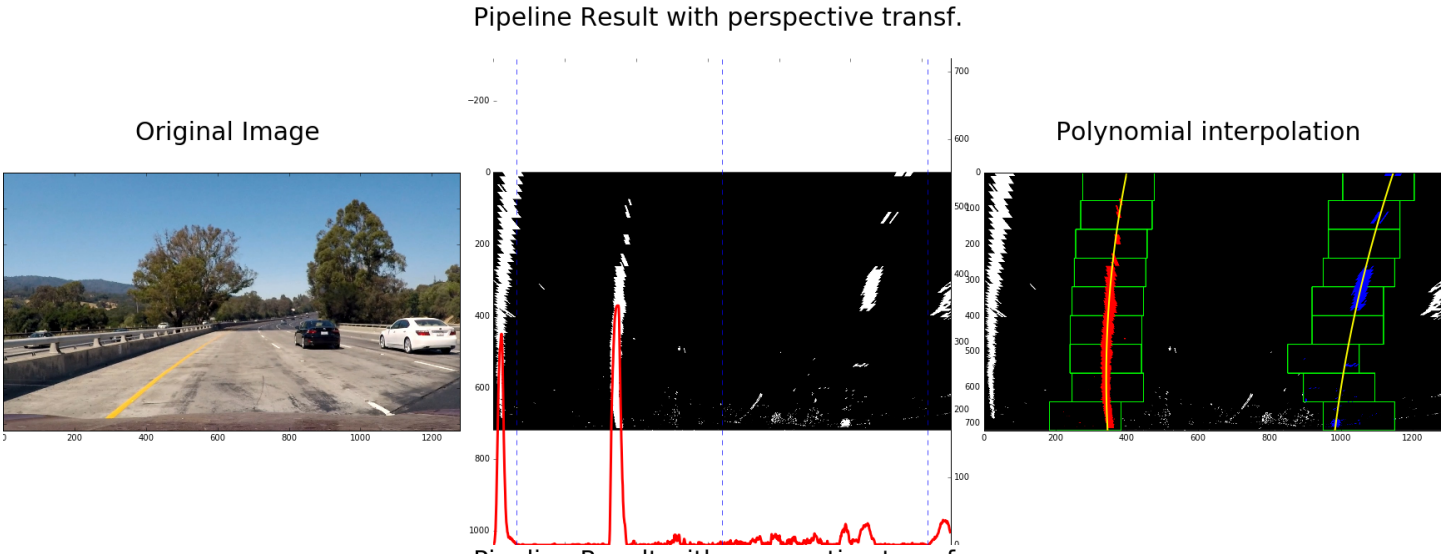
Pipeline Result with perspective transf.

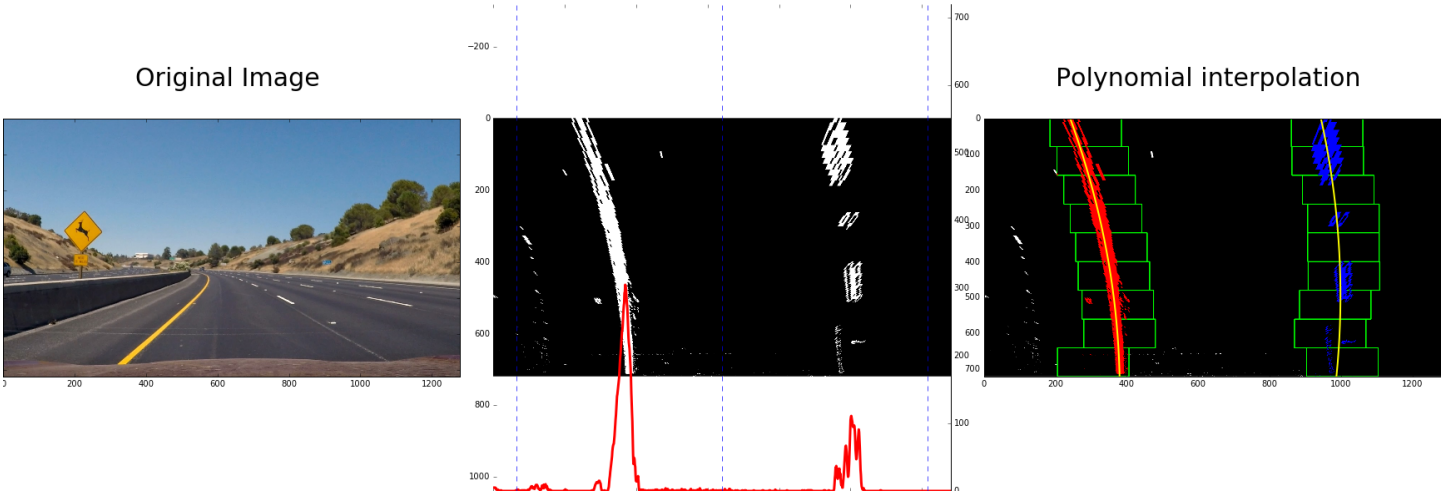


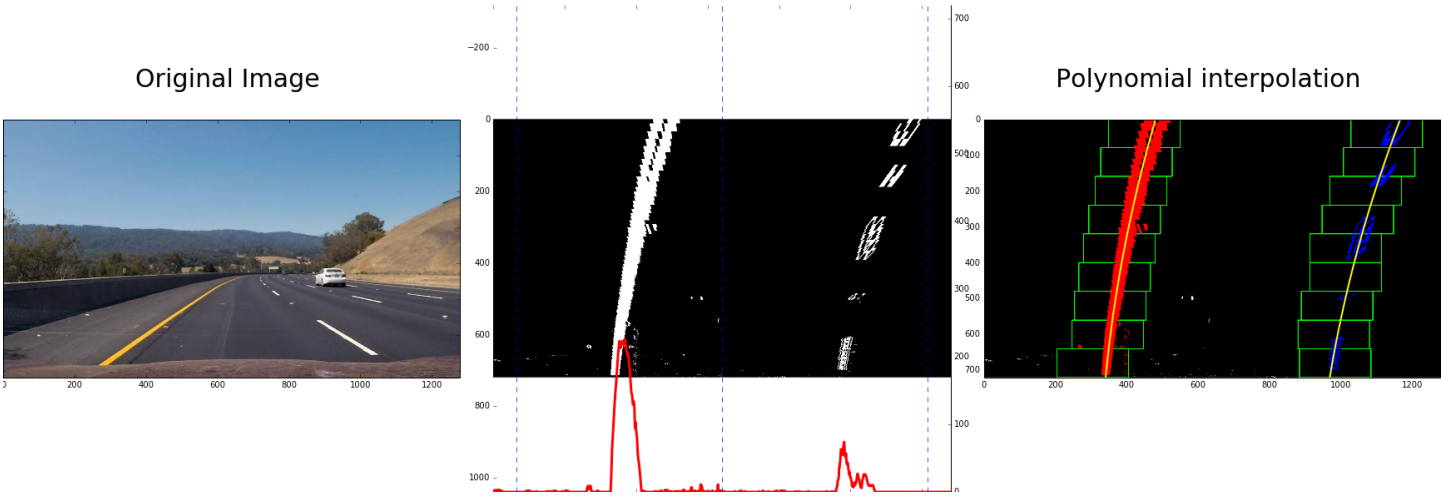
Original Image

Pipeline Result with perspective transf.

Old examples from polynomial interpolation:

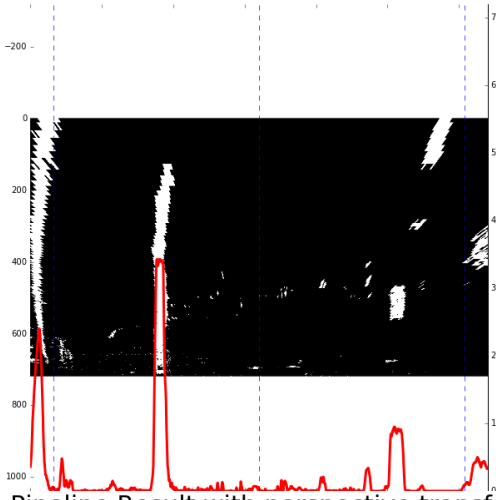




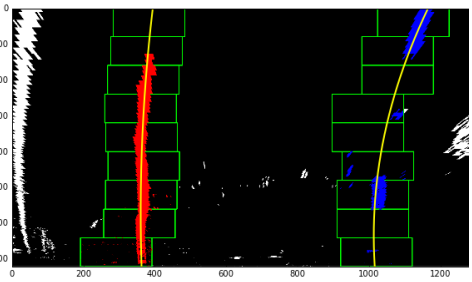


Pipeline Result with perspective transf.

Original Image

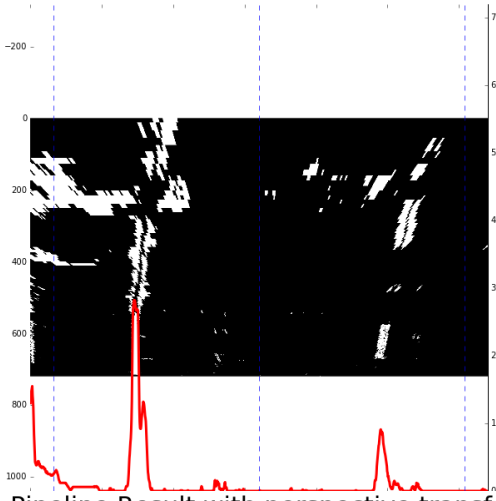


Polynomial interpolation

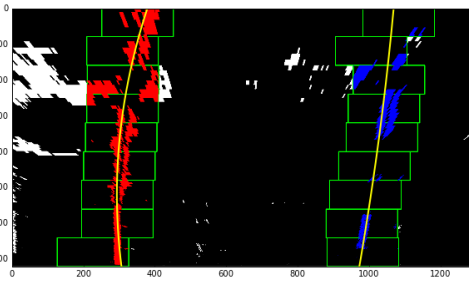


Pipeline Result with perspective transf.

Original Image

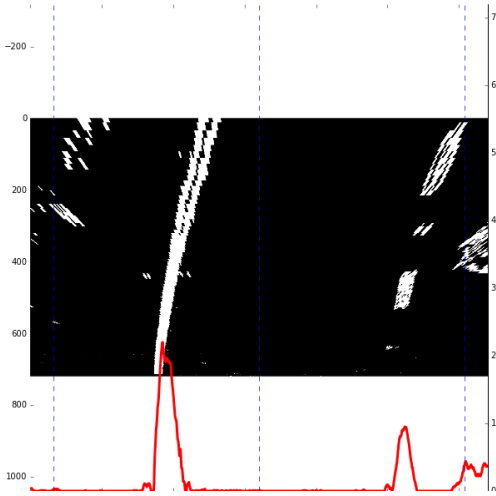


Polynomial interpolation



Pipeline Result with perspective transf.

Original Image



Polynomial interpolation