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.

1. How the camera matrix and distortion coefficients were computed.

The code used for this step was rewritten from the youtube video “Self-Driving Car Project Q&A | Advanced Lane Finding” by Aaron Brown from Udacity youtube channel. It is contained in the file called ‘camera_cal.py’.

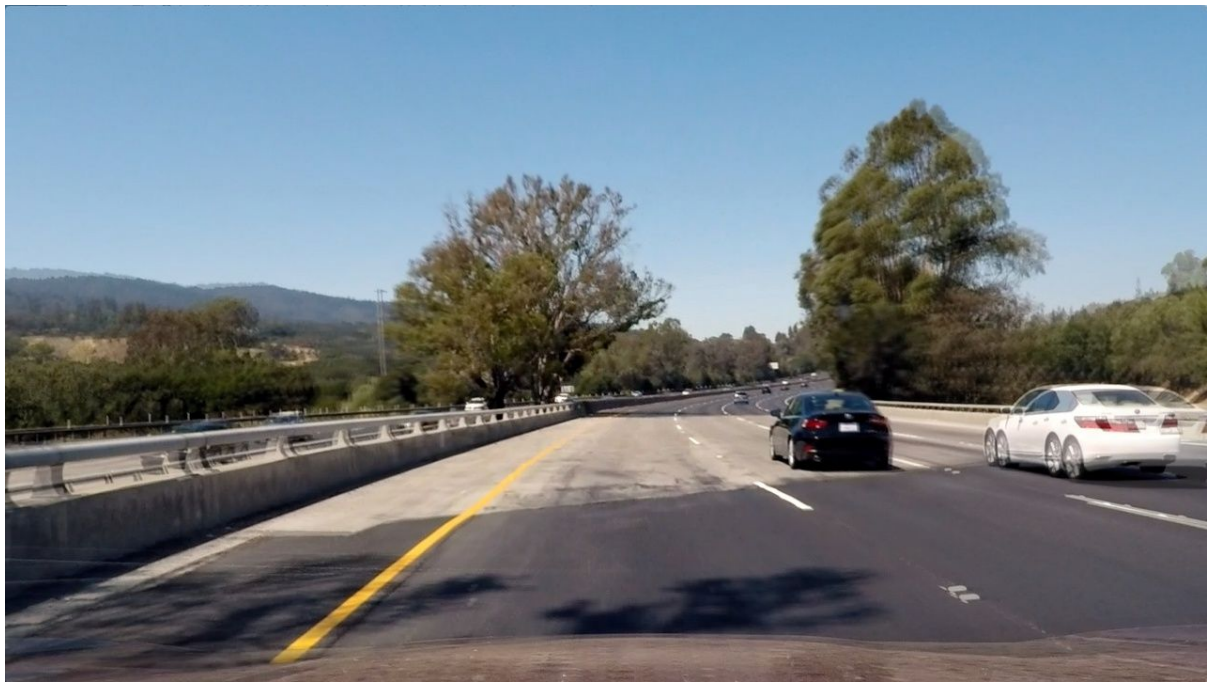


Figure “Nice Center”

First two arrays are prepared to make the coordination of points between the corner coordinates found by function “findChessboardCorners” and proper indexing. This proper indexing is an array of 54 by 3, as the Chessboard has 9 corners along 6 corners across and 3 axis are needed. As the destination will be 2D image plane, the last column will be left with 0 values, the first two columns will have all combinations of 9 values against 6 values and is composed in line #23 with a magical mgrid function.

Then a for loop in line #27 onwards will go through each image in archive “camera_cal” and will try to find all 54 corners in each one, if all 54 are found in an image, the coordinates will be appended to the array “imgpoints” which together with the array objpoints

54 Corners were extracted from 17 images out of the 20 images provided by the github repo. According to the opencv documentation at least 10 images should be provided to function cv2.calibrateCamera in order to make a good calibration. Besides “objpoints”, and “imgpoints”, cv2.calibrateCamera needs to know the shape of the image which in this case is 1280 x 720 which is stracted from calibration1.jpg in lines #47-48. The function returns camera matrix in array “mtx” and distortion coefficients as “dist” (rotation and translation vectors “rvecs” and “tvecs” will not be used).

Figure “Nice Center” is the result of “weight adding” undistorted and distorted test4.ipg image from the github repo. It shows that there is almost no distortion around the center.

2. How color transforms, gradients or other methods were used to create a thresholded binary image.

In this step I could bring an approach to the preprocess that mimics some human behaviour (I am still in P3 mood) in the way that we don’t see every time frame and make decision accordingly. I think that when I am looking at the road I see what I feel is the present road and some of the past road at the same time (a single image versus the thought of a sequence of images).

When doing P1 I spent some time working with cv2.adaptiveThreshold function but could not make use of it (I used simple the simple cv2.threshold), but now in P4 I could. The function zeros pixels under a threshold value in a neighborhood area according to neighborhood intensity.

So, after processing every frame with this magic, the present frame is added to a buffer. This was done by creating a new class called “past” (initiated in line 135 of my_file.py) which instance “remember” will append all present preprocessed images. Then in line 43 of my_file.py Compound Image is created by merging all interesting past frames. The following figure my help my poor explanation:

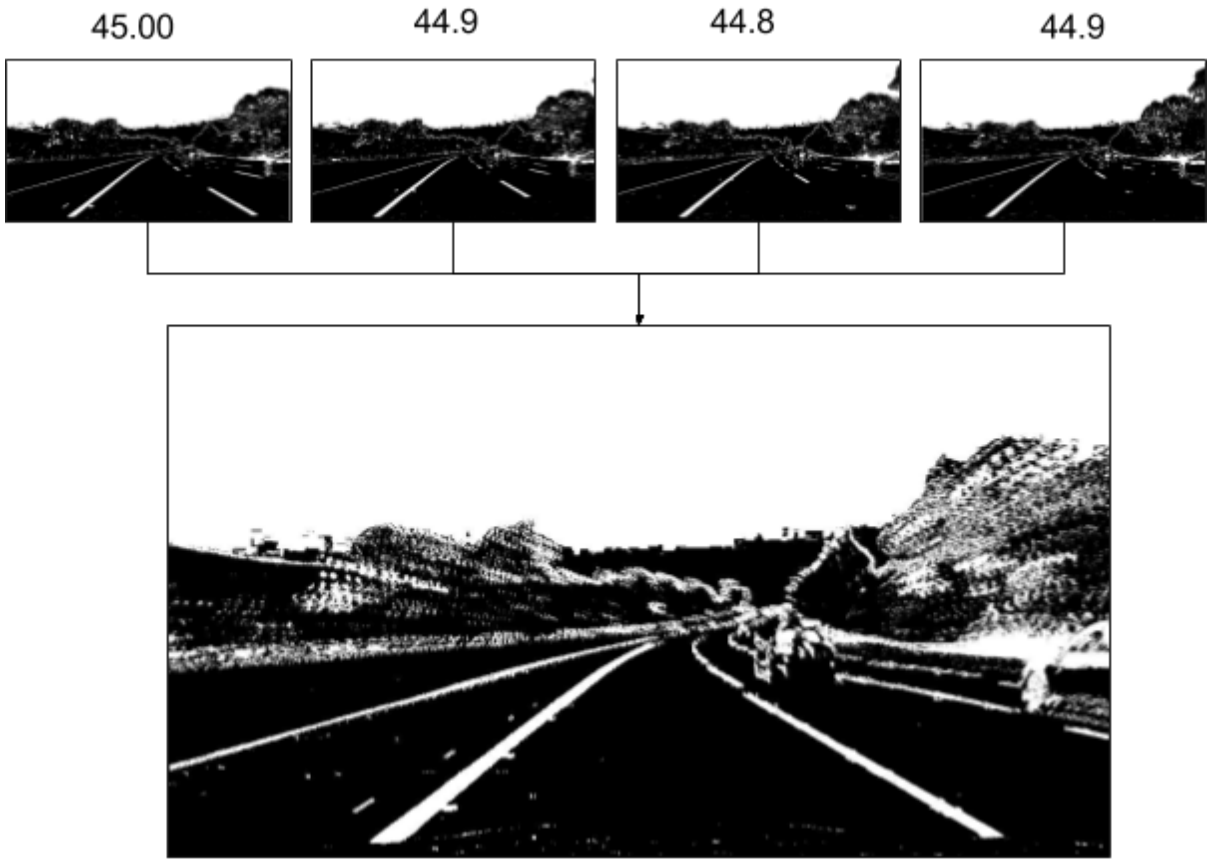


Figure “Nice Thought”

3. How perspective transform was done.

In order to make a perspective transform, cv2.warpPerspective warps image into flat image using transformation matrix M produced by cv2.getPerspectiveTransform which get two sets of coordinates: Source coordinates (in yellow) and destination coordinates (in red). The following image was generated to verify that “scr” and “dst” values were correct. See lines 50-62 of “my_file.py”.

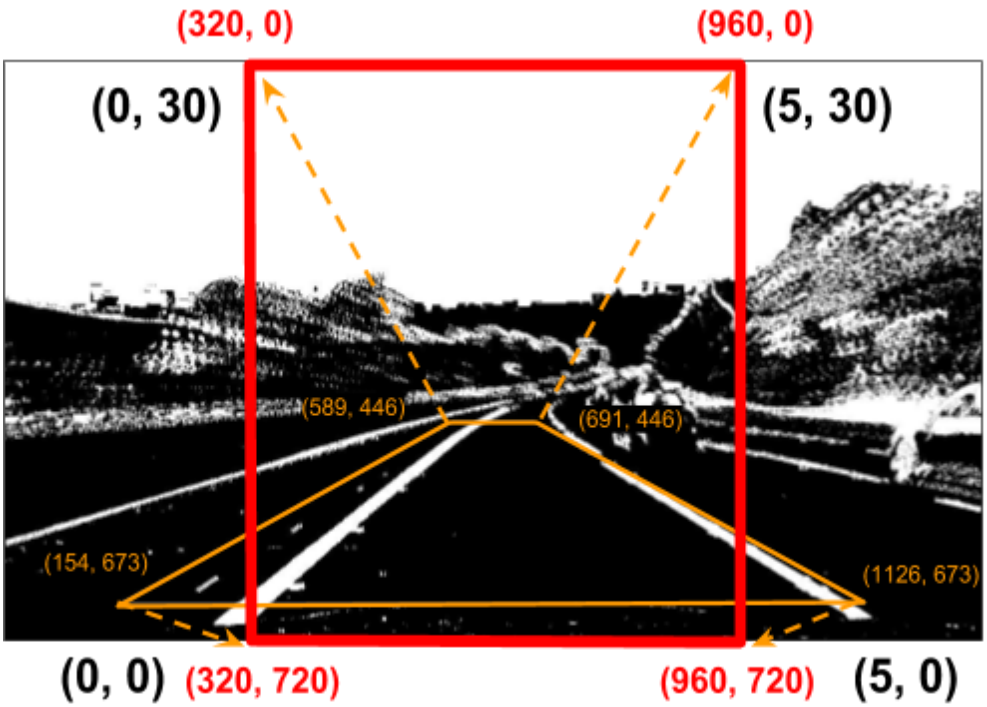
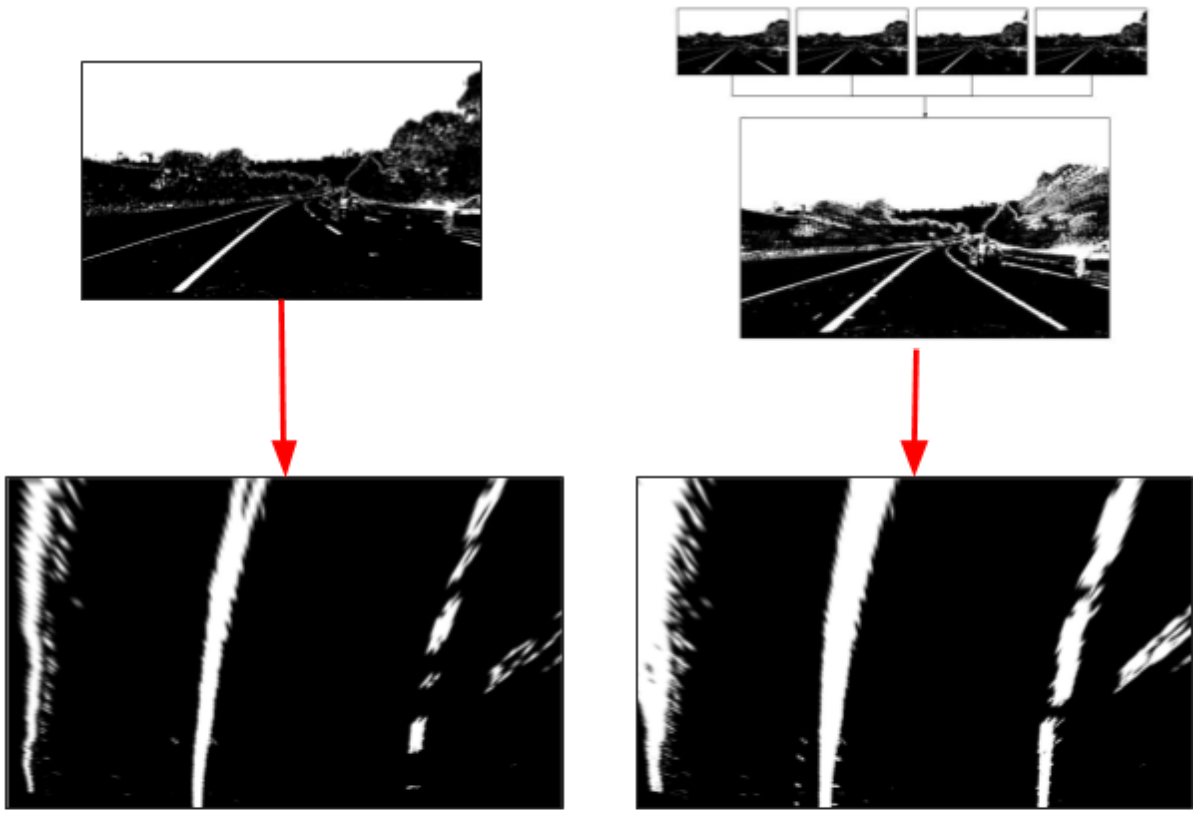


Figure “Three Worlds”

Source	Destination	Real World
(589, 446)	(320, 0)	(0m, 30m)
(691, 446)	(960, 0)	(5m, 30m)
(1126, 673)	(960, 720)	(5m, 0m)
(154, 673)	(320, 720)	(0m, 0m)

It can be seen from the following figure that the transformed image from the Compound Image has more information about lane geometry.



Transformed image from conventional preprocessing

Transformed image from compound image

Figure “Alone vs Family”

4. How lane-line pixels were identified and fit their positions with a polynomial.

Pixels were found with the histogram approach described in the quizzes (I prefered using this instead of the one used in the youtube video as I thought it was simpler and faster). Tracker class was rewritten from the youtube video but I added some lines in order to make it work when one of the two lanes faint. See lines of “tracker_p.py”

Figure “Frodo and Sam”

The instance window_centroids of class tracker_p gets the averaged x position of the last 40 sets of 5 pair of centers of sliding searching windows. The for loop in line 77 to 83 extract the values of each window center and allocates then in one of two lists, one for left and the other one for the right lane.

Then in lines 90 to 98 the x values are fitted to a 2 order polynomial with np.polyfit function that generate a vector with the coefficient of a second order polynomial.

5. How the radius of curvature of the lane and the position of the vehicle with respect to center were calculated.

Using some features from Google Earth it can be seen that the “scr” polygon is 30 meters long and 5 meters wide.

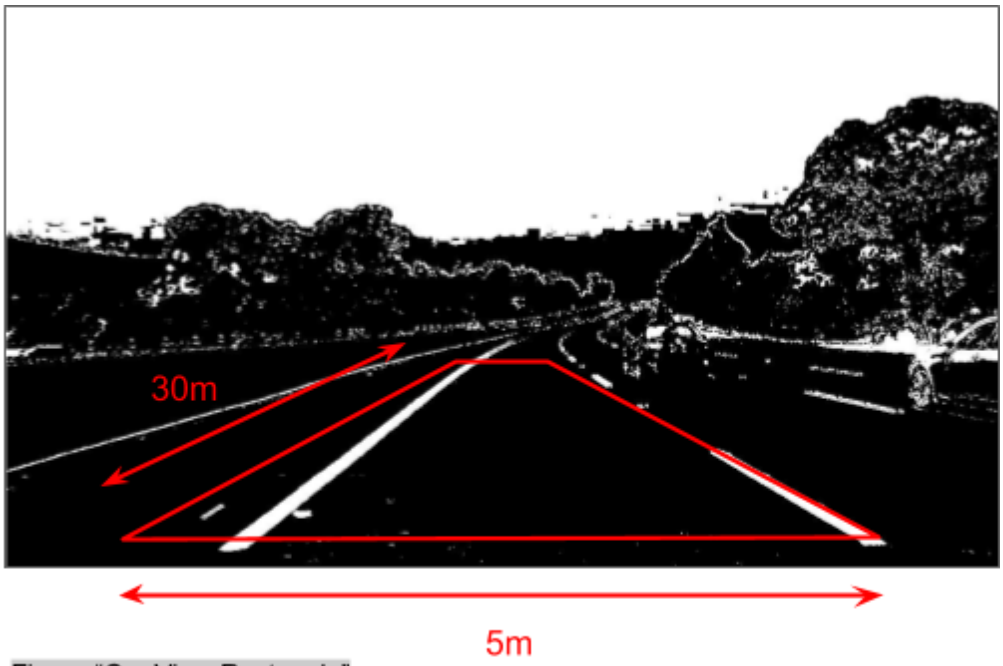


Figure “Car View Rectangle”

I deduced the related real word rectangle from the Google Earth spot on the I - 280 N. Finding a reference point (the same bush on the median at project_video.mp4 and Google Earth) and then counted nine lanes back to second 45 in the video.

This real world dimension are used in line #112 and line #114 to implement la curvature calculations and vehicle position with respect to lane.

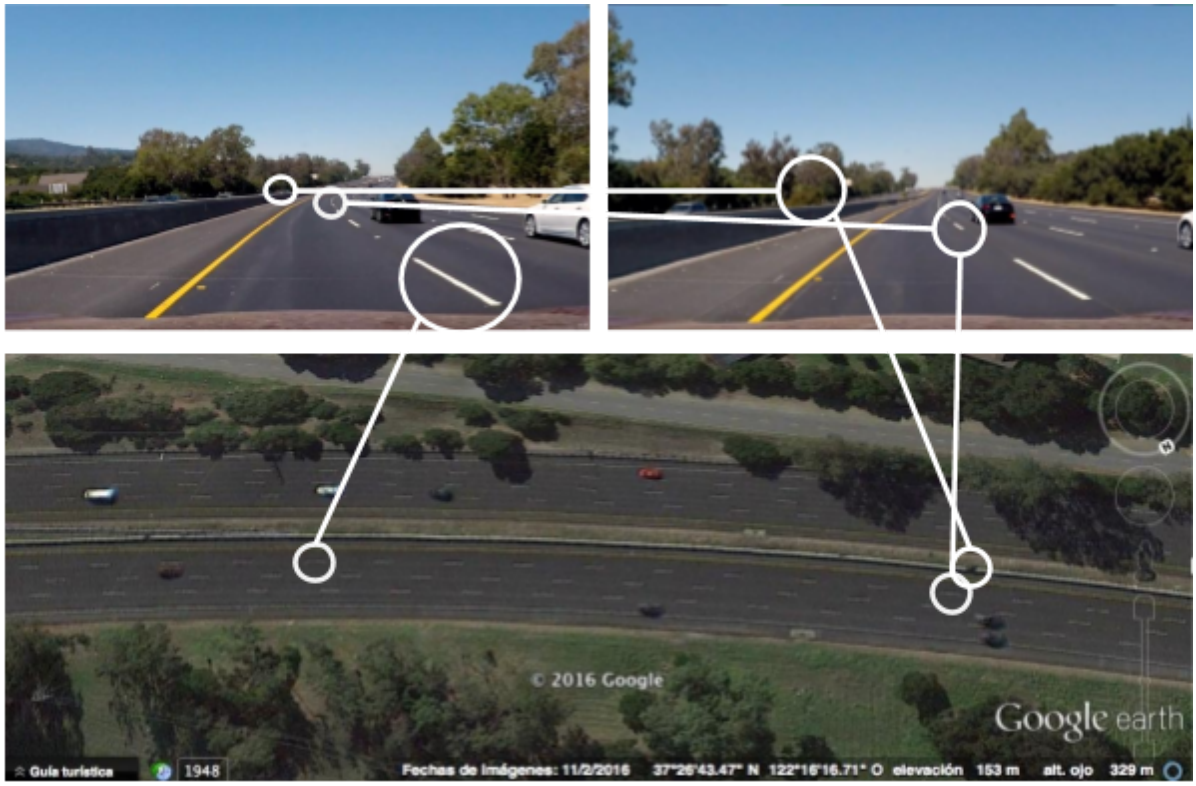


Figure “Sherlock”

Here is the rectangle in real world dimensions:

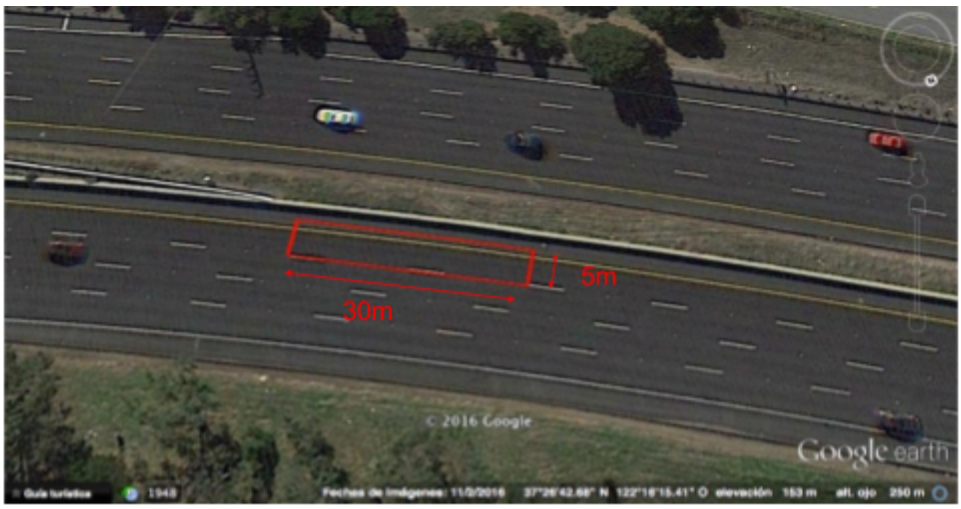


Figure “Drone View Rectangle”

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

Figure "Green Way"

Very short Discussion

I really enjoyed this project. It was great to implement an idea I had been thinking on since P1. I think the next step is to adapt the length of the road according to the road horizon (which depends on read inclination and curvature. Aaron Brown's YouTube video was of great help.

Figure "Next step"