Overview

This document describes the second project submission for the Udacity's nano degree course on self-driving car.

The goal of the project is to establish a pipeline that processes each frame of a video coming from the front camera in a car to detect the lane markings on the road in front. The output is to be produced in such format that it is useful for car as well as can be monitored for sanity by humans. For car the following relevant information is generated

- radius of curvature in meters
- car's offset from the middle of the lane

On basis of these two inputs, other control units in the car can take necessary corrective measures to keep the car in the center of the lane.

For human monitoring, the relevant information is generated in form of colored image showing the detected lane pixels.

I follow following steps to achieve these objectives:

1. Camera calibration
2. Applying distortion correction
3. Use color transforms and gradients to create a thresholded binary image.
4. Applying a perspective transform to rectify binary image ("birds-eye view").
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

These steps are discussed individually in detail in the sections below.

1. Camera Calibration

A camera is sensor that projects 3d outside world onto a 2d pixel image. Owing to the unique sensor/lens assembly and mechanical tolerances, each camera's projection is particular to that camera. Calibration is the step to identify the camera parameters so that the distortions added by camera can be filtered out from the captured image.

Similar to calibration of other instruments, camera's calibration is accomplished by measuring the distortion caused by camera on a standard input ie checkerboard image. For a through calibration a number of these images are shown and the data is accumulated for all the points in 3d world and their 2d projection. This data is then used to compute calibration data for the camera that can be used to un-distort any image captured by the camera.

There are two main components of the calibration data that we use: the intrinsic data of camera (eg focal length, axes skew, coordinates of optical center in the image plane etc) and distortion coefficients. Both these are obtained by the using OpenCV's `calibrateCamera` function.

As an input to **`calibrateCamera`**, we provide a list of object points and cooresponding image points. As mentioned, this is accomplished by checkerboard image sequence. **`findChessboardCorners`** function is used determine the necessary points in the checkerboard image as shown in figure 1 below.

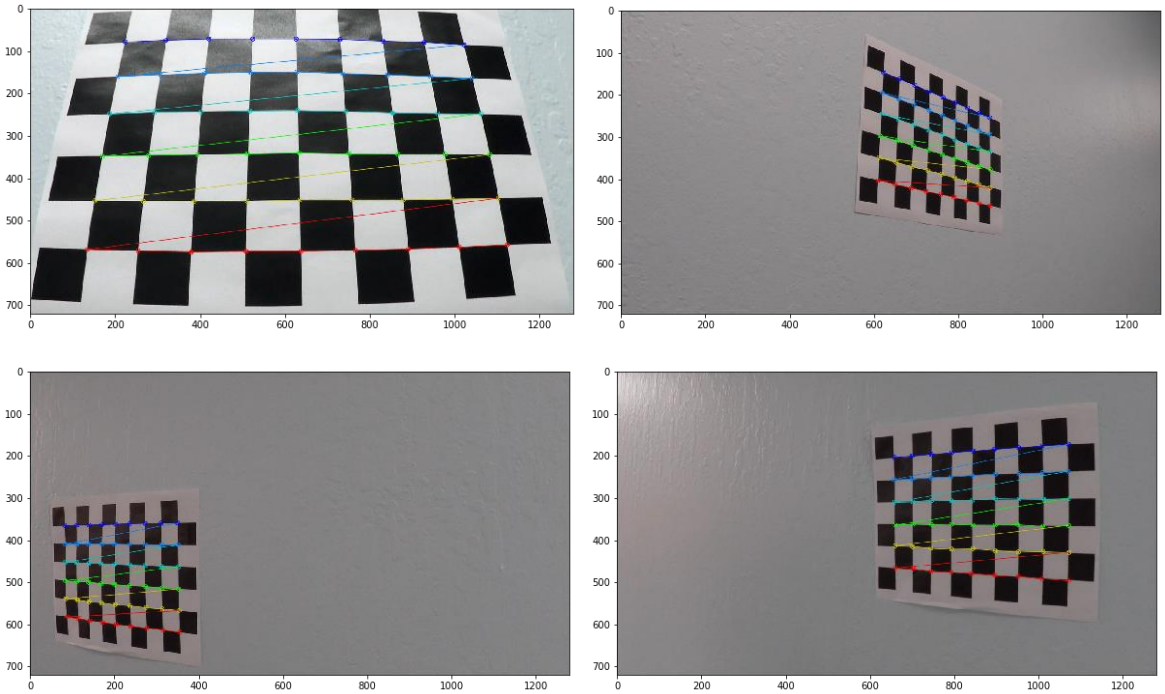

Figure 1. A Part of Checkerboard Sequence used to Calibrate Camera

2. Distortion Correction

Once the calibration parameters are calculated, these can be used to undistort any image captured by the camera. The aim of this operation is to remove the deformities introduced in the image by the image capturing setup. The two main parts of the calibration data are intrinsic matrix of camera (**`mtx`**) and distortion coefficents (**`dist`**). For this operation we use the aptly named **`undistort`** function from OpenCV. Below we show an original captured image in comparison with the undistorted image. For all the input images coming from the camera, undistorting is the first step. After undistorting the image can be used for further processing for feature recognition and evaluation.

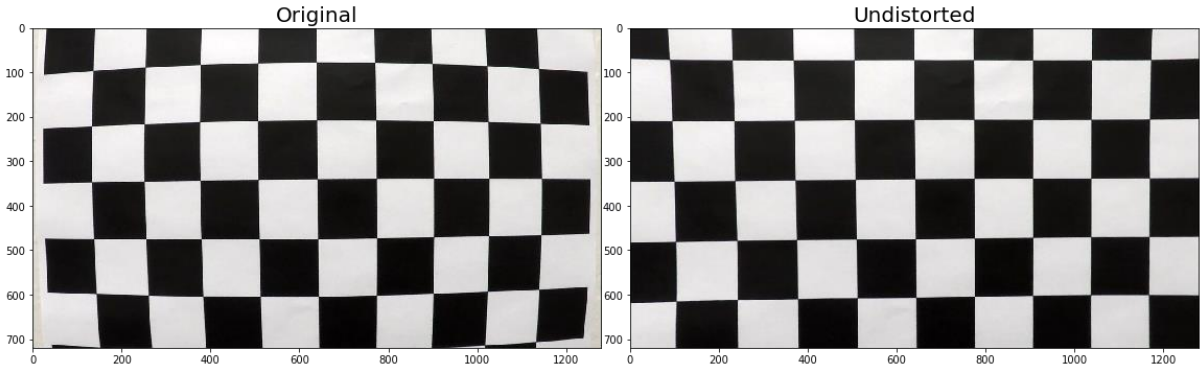

Figure 2. Original vs Undistorted Image

3. Thresholded Binary

The next step in our pipeline is that of extracting the lane lines so that these can be used to decide a future path for the car. The main aim here is to process the captured and undistorted image in such a way that all irrelevant details in the image are ignored leaving behind only lane lines. There are a number of techniques that can be used for this, both independently or in combination with one another. We use a combination of following four techniques:

1. Sobel Gradient Filtering (in x and y)
2. Gradient's Magnitude and Direction Thresholding
3. S channel filtering in HLS color space
4. Region of interest based masking

Each of these is discussed briefly in following sections.

Sobel Gradient Filtering

The core idea of sobel gradient filtering is to take a derivative of the image in either x or y direction and select the parts where the derivative is greater than a certain threshold. This results in edges in the image being highlighted. In this way lane lines are expected to standout in the image which can then be selected for further processing. Naturally, the input for this technique is a grayscale image. This filter can be applied in either x or in y direction. Correspondingly, the edges in x or y direction gets highlighted. Since lane lines are normally originating from the bottom and going upwards, they are more prominently shown in x direction sobel filtering. In our case we create the output of this sub-step by selecting those sections of the resultant image where both x and y sobel filters are higher than the threshold values ie we use AND operation.

Gradient's Magnitude and Direction Thresholding

Another technique of combining both images resulting from x and y directional sobel filtering is to generate the net image on basis of the magnitude of both previously mentioned inputs. Magnitude of the gradient is given by following equation:

$$G = \sqrt{G_x^2 + G_y^2}$$

Where, G is the magnitude of gradient and G_x and G_y are directional gradients in x and y direction respectively.

The same two inputs (G_x and G_y) can be used to compute a directional map (Θ) of gradient ie using the following equation.

$$\theta = atan2(G_y, G_x)$$

This gives the directions of detected edges. Since we know the normal range of angles at which the lane lines appear in our view, we can use this information to filter out the irrelevant lines out of consideration.

In our case, we accept an edge as an "interesting" candidate for a lane line if it satisfies both magnitude and directional thresholds.

S-channel Filtering

In practice examples during the course it was clear that saturation (S) channel form Hue-Lightness-Saturation (HLS) color space was particular helpful in detecting both yellow and white lane lines. We employ this also in out thresholding process. Image is converted into HLS color space from which we separate out S-channel. On basis of S-channel nd then a binary is generated on basis of S-channel satisfying the thresholds.

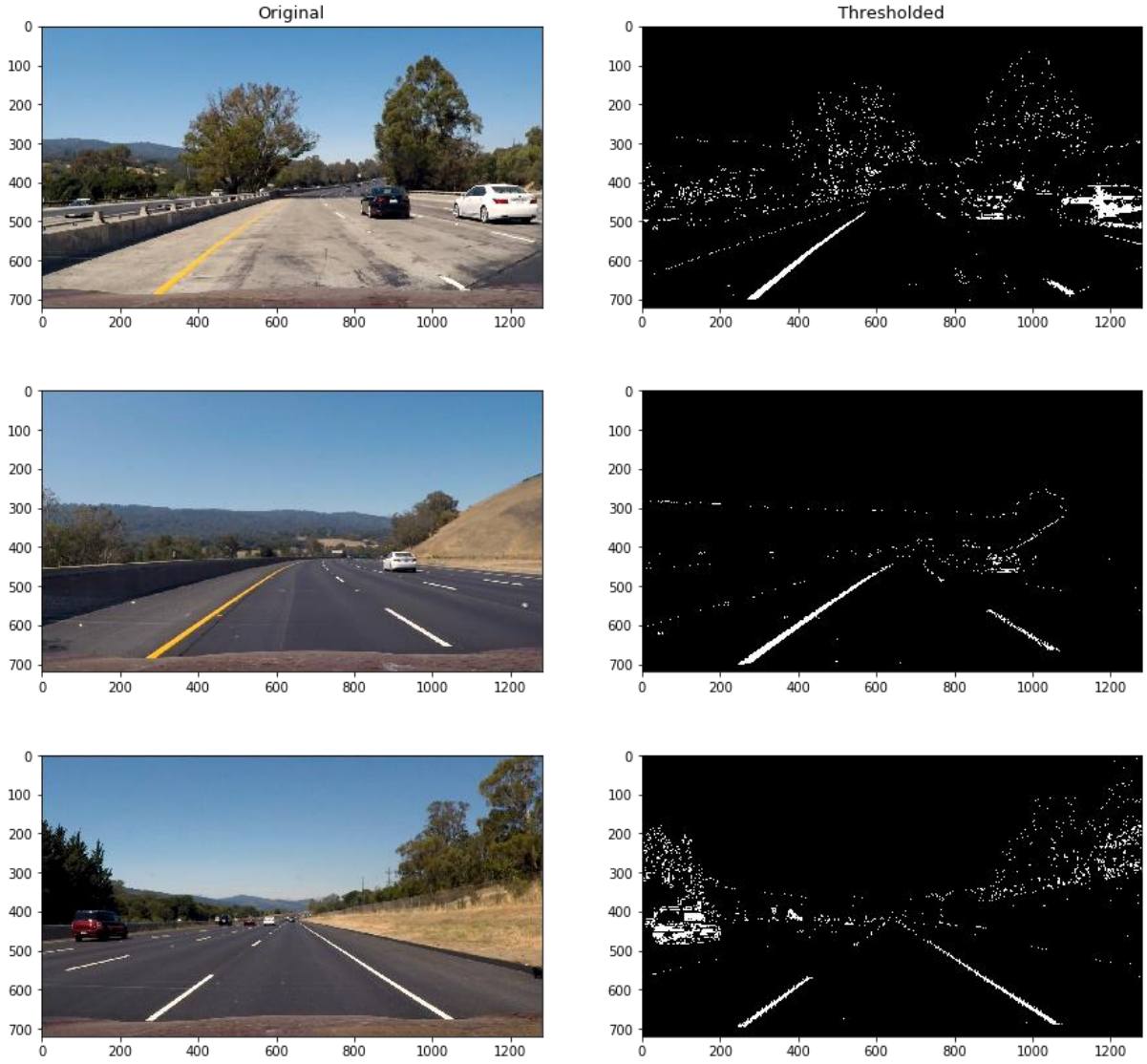

Figure 3: Original image vs edges detected by thresholding procedure. The images on the left shows the original undistorted images and on the right are the resultant images generated by "OR"ing the outputs of the three steps described above.

Region Of interest based Masking

In the final step, outputs from all three above steps are stitched together by OR operator. Ie if an edge is accepted by any of the criteria above it is considered for future steps as a potential candidate for lane lines. Final filter that is applied is based on region of interest. In this step all the parts of the image where we do not expect to find lane lines in the image are discarded. Visually the operation is presented in the following figure. Naturally, in practice the input of the operation is the binary image outputted from the previous step, but here, for the sake of clarity we show the operation on a normal colored image.

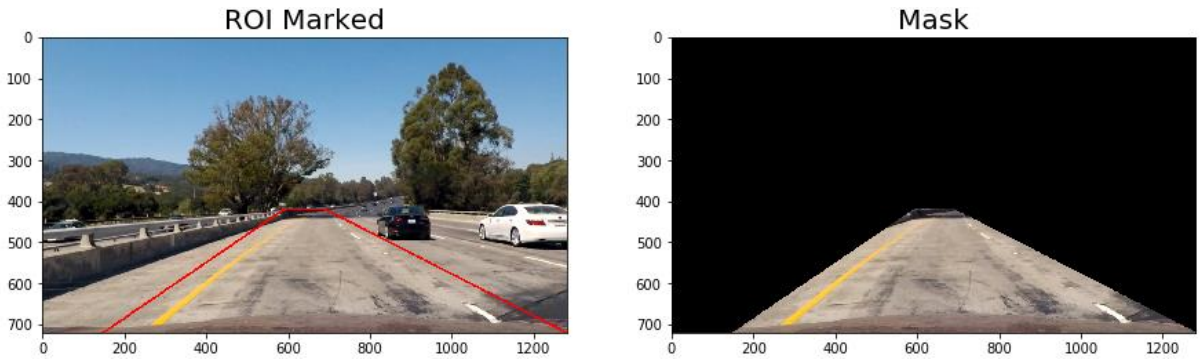

Figure 4. Using Region of interest (ROI) based mask to remove the uninteresting details in the image.

4. Perspective Transform

In this step, we transform the image's perspective from that of a driver to a bird's eye view. This perspective makes it easier to perform mathematical estimations of the road markings such as curve fitting on the shape of lane lines to estimate the radius of curvature. For this step, basic input consists of a set of four points in the original, undistorted image and corresponding four points in the birds eye perspective. With this information we utilize the OpenCV function **`getPerspectiveTransform`** which returns a 3x3 mapping matrix for transforming the provided 4 source points to destination points. With this mapping matrix and the source image as an input **`warpPerspective`** calculates the warped output image. The source image used as input here is the output of previous step ie a binary image with most of the unnecessary parts removed. However to give a better visual explanation we apply the operation on a normal color image in the following figure.

Figure 5. Input vs output of perspective transformation and warping.

5. Detecting Lane Boundaries

This is the first step where actually attempt to identify the pixel belonging to lane lines. All the previous steps were pre-processing steps with the aim to prepare best possible input for this step.

The input of this step is the warped binary image as shown below. We start with summing a histogram across the columns in the bottom half of the image. The peaks in the histogram give a pretty good approximation of the start of lane lines. After the best starting points for the both, left and right lane lines are decided, we use this as the base for our sliding window approach. I.e., the first set of (left and right) windows in the bottom of the image is placed at these points. After this we gradually move up in the image and identify all the non-zero pixel in the next same sized window area and then the window is centered at the mean location of all non-zero identified pixels. For the next step, the x-location of this window is used as a base for placing next window, and so on.

If no new non-zero pixels are found in one of the windows, the next window still takes the same x value as the previous one.

This operation is visually shown in the image below.

Once the all the interesting non-zero pixels are found, we use numpy's `polyfit` to fit a second degree polynomial on these pixels to approximate the location of lane lines. These are shown in the following figure as yellow lines.

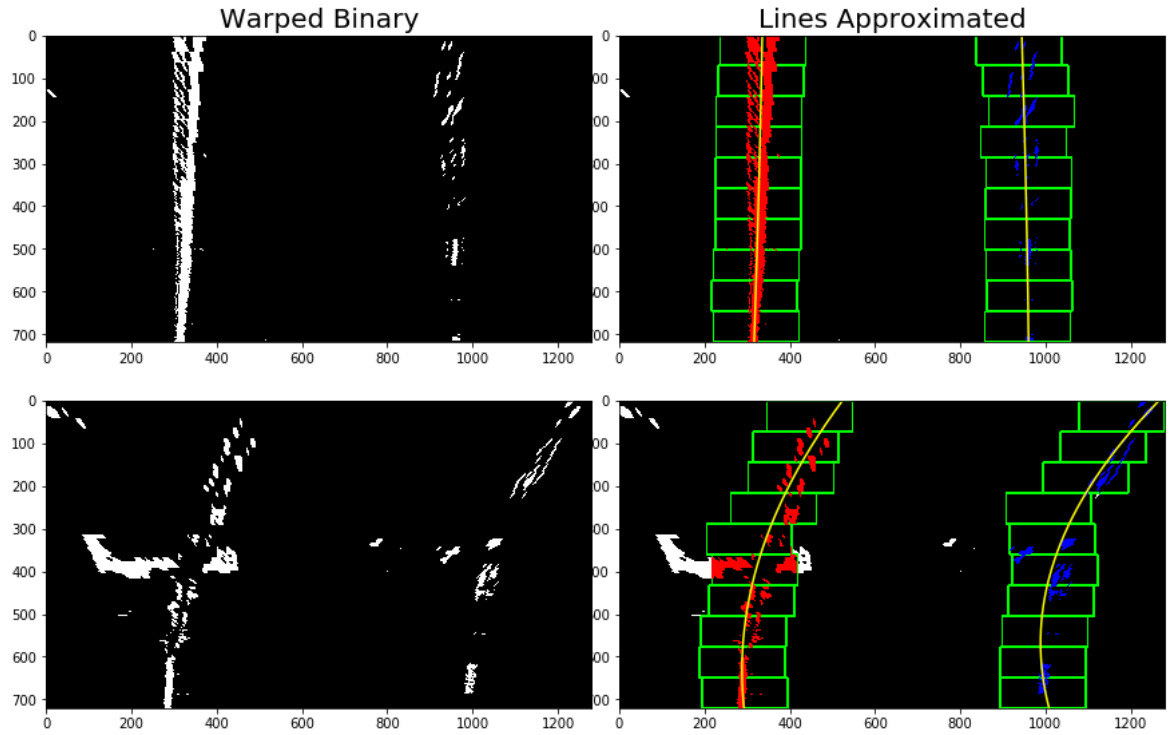

Figure 6. Approximating the location of lane lines by fitting a second-degree polynomial.

6. Determining the Curvature and Offset

Once we have the polynomial that approximates the location of lane lines, we use these to determine the curvature of each line and take the mean to determine the curvature of the lane which is later overlayed to the original image.

7. Warping the Lane boundaries to the Original Image

Once the boundary has been determined, as shown in the figure 6 earlier, it is drawn on a blank image and warped back to the original undistorted image using the same functions as for warping. Then this image is combined with the original image to give the output as shown in Figure 7.

8. Adding Numerical Estimations on a Visual Overlay

In the last step, we overlay the numerical estimations at the top of the image as shown in the sample video in the project intro. The output is shown below.

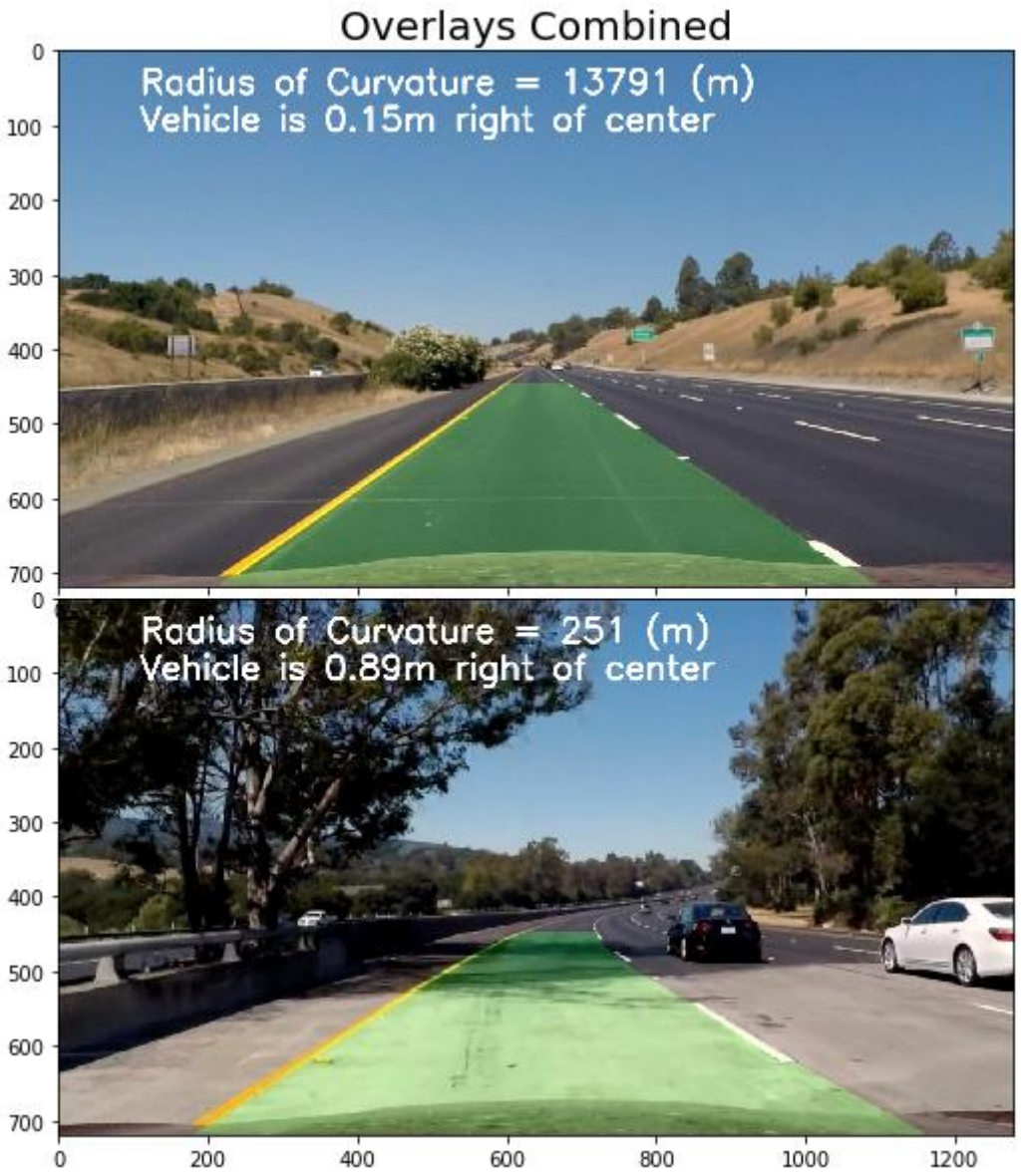

Figure 7. Overlays of unwarped lane boundary and numerical estimations added to the original image

9. Discussion

The techniques learned during the scope of this project already show a major improvement over the previous approach tried in project one. The challenges faced in the first project are trivially solved with these new techniques. For example, earlier we had problems with the shadows in the images also yellow lane lines were not being registered. By converting the image to HLS color space, the yellow lane lines are easily recognized. Also combining the thresholding schemes makes this implementation quite robust in comparison with the previous project.

Also it would be interesting to see how the scheme behaves when unknown input is presented. Some of the examples could be rapidly changing traffic scenario for example other cars and trucks driving by. Specially, it would be interesting to see what happens if the shadows of the other cars are falling on the lane right in front of the camera. Or, another interesting case could be a big truck driving in front which has a picture of a driving lane at its back.

Nevertheless, for given inputs, the current pipeline behaves quite reasonably. However, some easy improvements can still be easily made to enhance robustness. For example,

1. Using iterative subpixel calibration can help undistort the captured images better than current approach. If we look in figure 2 above, we can still see that lines on undistorted chessboard are still not optimally straight. This indicates that undistortion has not achieved its goal totally. Hence there is a room for improvement here which can be exploited by subpixel calibration technique. This technique is supported by OpenCV as well.
2. History can be used to predict the future and smooth out the outliers. The predicted lane lines should be subjected to a sanity test i.e. if they have curved more than a sane value then this prediction should be discarded as not sane and in this case we keep using the old value of the lane lines. Similarly, if the distance between the lane lines changes too much too quickly, or they intersect with each other then it is again a wrong prediction and should be discarded. Hence a number of sanity tests can be added to iron out the wrong predictions.
3. Currently when we identify the line pixels using a sliding window approach where each window is based on the previous window's location. This approach can be refined as follows. After determining the fitted polynomial, the whole procedure can be repeated with one difference: this time the new window location based on the predicted line location (by the polynomial found in previous step) instead of basing it on the location of previous window. In this iterative way, we can refine our line pixel prediction.

In a nutshell, it was quite a rewarding hands-on experience for me and I came across a lot of new areas that I was not aware of.