Advanced Lane finding project*

Shanthan Suresh[1] †

March 8, 2017

1 Overview

In this project we employ advanced computer vision techniques to detect lanes on the road. We run our pipeline through a video where processing occurs frame by frame.

2 Introduction

Lane detection is a challenging problem because of varying brightness, shadow regions, insufficient lighting, varying lane colors along the roads. Choosing the right set of parameters become important since the features are to be selected manually, as compared to deep learning techniques wherein the best features are selected by the network itself.

The advantage of this approach is that we do not need data apriori to perform training. Since the features are handcrafted, we use the features directly on the test images/video to detect the lanes.

This was a tedious exercise since we need to choose the parameters to suit the type of roads. This was done by an iterative method with a series of trial and errors.

**This project was supported by Udacity and Nvidia Graphics Pvt Ltd

†[1]Shanthan Suresh is with Nvidia Graphics Pvt Ltd, Bangalore, India shanthan.n2@gmail.com

3 Processing Pipeline

The pipeline proposed for this project is derived from the lecture videos from Udacity.

- Camera Calibration
- Undistort the image
- Image thresholding
- Region of interest masking
- Perspective transform
- Detect lane points
- Polynomial fit to lane points
- Radius of curvature, lane deviation
- Draw output image

3.1 Camera Calibration

This is the first step of the pipeline, where the camera distortion and projection matrices are determined using a standard image namely chessboard image. Fig. 1 illustrates this step.

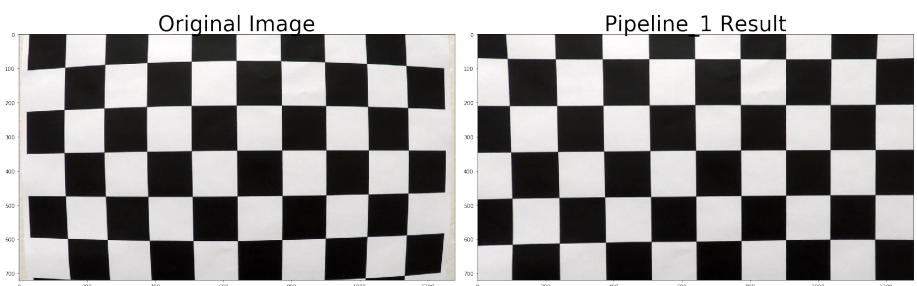

Figure 1: Determination of camera distortion coefficients and projection matrix using a chessboard image.

3.2 Undistort the Image

In the next step we pass the original distorted image through an undistortion block, which rectifies the distortion introduced by the camera.

3.3 Image thresholding

The undistorted image is now passed through an image threshold block which does a selective filtering to retain the lane lines. We use two types of threshold namely color based threshold and gradient based threshold. Fig. 2 illustrate this step.

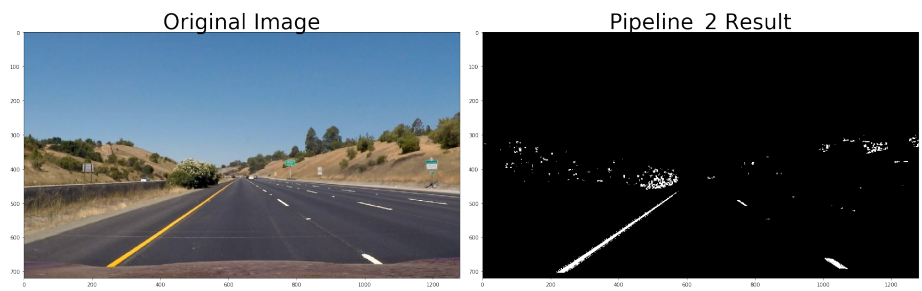

Figure 2: Undistortion followed by thresholding. Note that apart from the lane lines, there are other regions detection as well. These will be removed in the subsequent steps.

3.4 Region of Interest masking

The thresholded image is passed through a region of interest block which is a binary mask. This mask selects the lower portion of the image and eliminates the upper portion of the image. This is exploiting the fact that there is more likelihood of finding the lanes in the bottom half of the image than the top half which would contain the horizon. Fig. 3 illustrates this step.

3.5 Perspective Transform

In this step, the image is passed through the perspective transform block which enables us to obtain a bird's view (top-down view) of the road. Fig. 4 shows the perspective transformation.

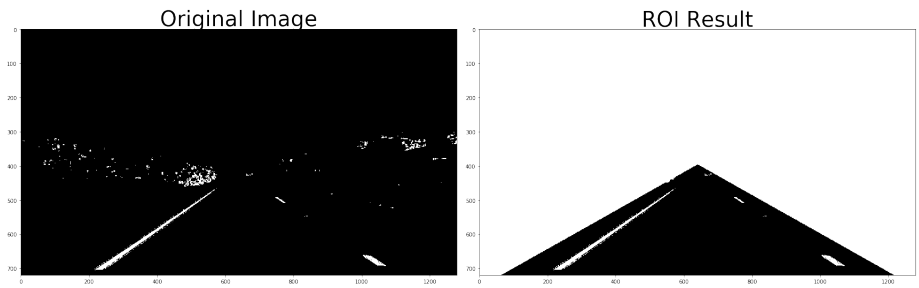

Figure 3: Region of Interest applied on the thresholded image.

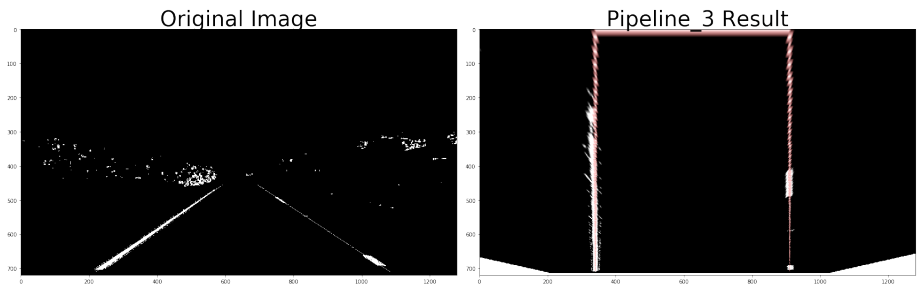

Figure 4: Perspective transform applied on the binary image.

3.6 Detect Lane Points

Once we have a bird's view of the image where the lane lines appear almost parallel, we run the bottom half of the image through a cwt filter that determines the peak points. Once we determine the two peak points corresponding to the left and right lanes, we use it as an initial reference points. We now split the image into 10 horizontal patches and using the initial reference lane points, we have a window of 100 pixels to determine the peak points in each of the 10 horizontal patches. These form the approximate lane points for our next step in the pipeline. Fig. 5 shows the lane points detected.

3.7 Polynomial fit to Lane points

Using the lane points detected in the previous step, we fit two quadratic polynomial curves, one for the left and one for the right lane. The independent variable is the y-cordinate(vertical direction) and the x-cordinate(horizontal direction) is the dependent variable. Fig. 7 shows the polynomial curves that are fit on lane lines.

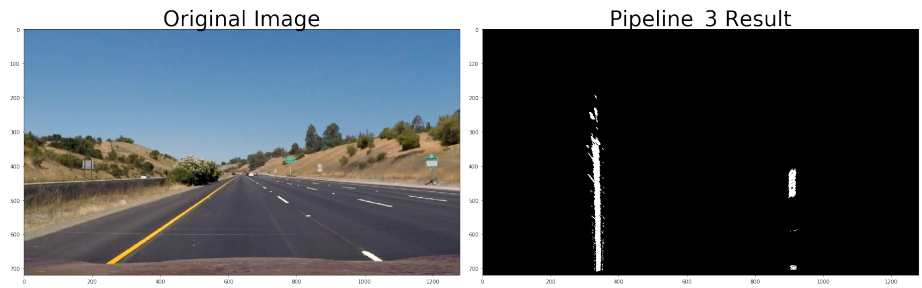

Figure 5: Lane points detected using peak detection and windowed approach.

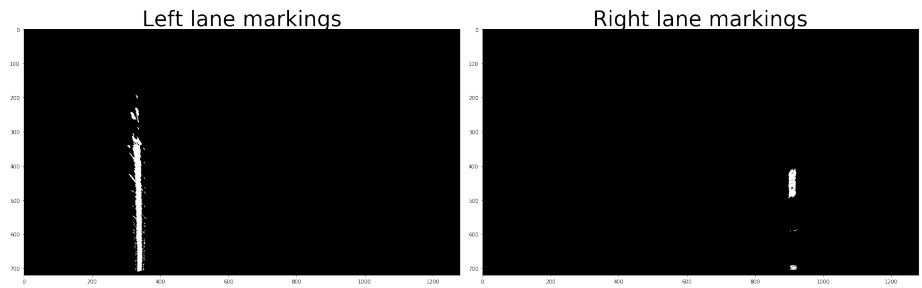

Figure 6: Left and right lane points separated.

3.8 Radius of curvature, lane deviation

After determining the left and right polynomials, we determine the radius of curvature and amount of deviation of the car from the lane center.

3.9 Draw output image

In the final step of the pipeline, we display an output image in which the region between the lanes are highlighted in blue color and also depict the radius of curvature, lane deviation. Fig. 8 shows the output image.

4 Rubric Criteria

In this section we address the points related to rubric criteria, for this project.

Are all required files submitted?

My project includes the following files:

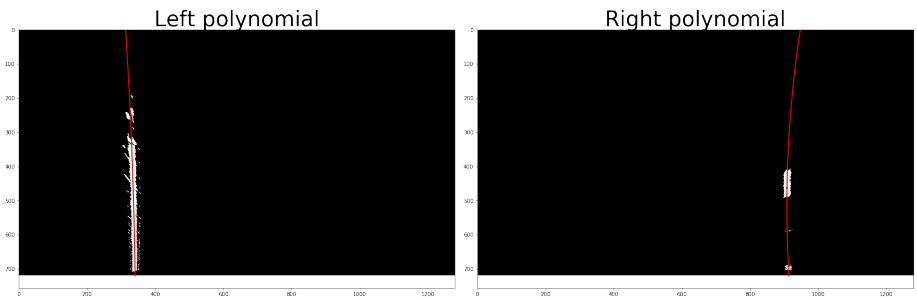

Figure 7: Polynomial curve fits on left and right lane points.

Figure 8: Output image with the lane detected.

* Adv-Lane-Finding.ipynb containing the code for the project.
* output_images folder containing output images of pipeline.
* extras folder containing output of different algorithms in pipeline.
* project_video_output.mp4 containing the output video.
* writeup_report.pdf summarizing the results.

Provide a Writeup / README

You are currently reading the Writeup. This has all the relevant figures and details of the solution.

4.1 Camera Calibration)

Camera Calibration

We used the chessboard calibration images (provided in the repository) to compute the distortion coefficients and projection matrix of

the camera. We used *cv2.calibratecamera()* function to calibrate the camera.

4.2 Pipeline(test images)

Distortion Corrected Image

We then used *cv2.undistort()* function to undistort a test image. The test image was chosen from the calibration set. Fig. 1 illustrates this step. We have also put the undistorted image in *extras* folder by the name *undistort.jpg*

Thresholded Binary Image

As explained in the previous section we used HLS color thresholding and sobel filtering along x axis to obtain the lane points. After this we use a suitable threshold value to create a binary image. Fig. 2 illustrate this step. We have also put the undistorted image in *extras* folder by the name *undistort_thresh.jpg*. The piece of code which does this task in our implementation is called *threshold_image()*.

Perspective Transform

We used opencv functions to obtain a *top-down* view or *birds-eye* view of the image. The functions which did this job for us is *cv2.getPerspectiveTransform*. Please refer to *get_perspective_transform* in our code, to check this part. Fig. 4 shows the perspective transformation. We have also put the perspective transformed image in *extras* folder by the name *perspective.jpg*

Lane Line pixels, Polynomial fit

We used second order polynomials to approximate the left and right lanes. Fig. 7 shows the polynomial curves that are fit on lane lines. We have stored this in *extras* folder as *left_right_poly.jpg*. The function which does this job is called *find_left_right_points*, *find_lane_points* and *fit_poly*.

Radius of Curvature, Lane offset

We have calculated the radius of curvature and lane offset in meters. Please refer to *get_radius_of_curvature()* and *calc_lane_offset* for this task.

Output Image

Fig. 8 shows the output image after all these operations. It shows the lane areas clearly in between the lane lines. It also depicts the radius of curvature and lane offset in units of meters. We have stored this in *extras* folder as *output_img.jpg*

4.3 Pipeline(video)

Output Video

We have generated the output video by running the pipeline on a video. The processing is happening frame by frame. This output video is named *project_video_output.mp4* and contains all the details that we described earlier for output image. The details include lanes identified, radius of curvature and lane offset. Our pipeline is able to map the lane lines even when they are not straight, works well in shadows, and works on colored lanes as well.

5 Discussion

This project was a very tedious one as it involved careful choice of parameters to obtain the best features for lane detection. In the end, we felt that this was worth an effort after seeing the video pipeline work.

We started work on this project by developing each module and doing an unit test, to make sure that the module functions as expected. Then we tied all the modules together to form the processing pipeline.

We faced a challenge in choosing the suitable filters for thresholding. After some experimentation we choose HLS filter and sobel filter, with suitable values for the threshold. We were also getting some spurious elements in the image after thresholding. These elements were the trees and some cars being detected. This was when we introduced the Region of Interest filtering to just look out for the lanes in the bottom half of the image.

After some effort and struggle we were able to get the pipeline work on single images given in test-images folder. We then switched to video processing and new set of problem emerged. We observed that the performance was deteriorating when there is shadow on the roads. We then revisited the pipeline and had to make some changes. Initially we were detecting peaks

for all the frames, which was an unoptimized solution and not exploiting the temporal dependence of the frames to look for lane points in the neighbourhood of previously detected lane points in earlier frames. We then introduced *fill_masks_next_frame()* function which calculates the mask based on previous frame. This helped us to improve the performance significantly.

Calculating the radius of curvature was another issue. We were getting very huge values of radius of curvature in meters. This turned out to be a bug in our code, where we had exchanges x and y values. We fixed this and started getting reasonable values for the radius of curvature and lane ofsset.

Overall this turned out to be a good project to give us a feel of the challenges involved in computer vision based approaches.

5.1 Scope for further improvement

We see in our implementation that there are some sharp changes in the detected lane points towards the horizon. We feel that this could be improved further by smoothing out the filter coefficients. We so have a smoothing of the coefficients in the current code, but this could be improved.

5.2 Results

Fig. 9 to Fig. 16 depict the results of applying the pipeline on the road images. These images are placed in output_images directory of our submission.

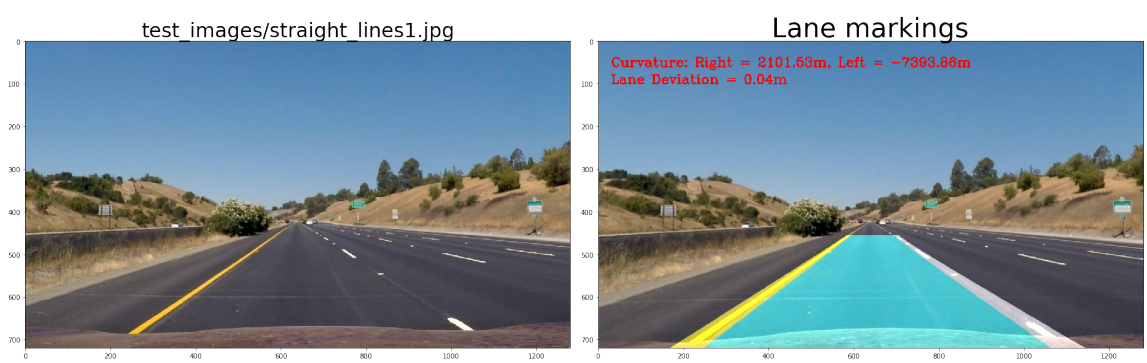

Figure 9: Output image-1 with the lane detected.

Figure 10: Output image-2 with the lane detected.

Figure 11: Output image-3 with the lane detected.

6 Conclusions

In this project we adopt advanced computer vision techniques to find the lanes along the road. We observe that these techniques enable us to determine the lanes even when they are curved. We also see that it is possible to determine the position of the lane lines when they are not white colored, and when there are shadows of objects on the roads. The major challenge here is to handcraft the features for best possible results, which necessitates the usage of neural networks to let it decide for the best features.

References

[1] M. Bourne, "http://www.intmath.com/applications-differentiation/8-radius-curvature.php"

Figure 12: Output image-4 with the lane detected.

Figure 13: Output image-5 with the lane detected.

Figure 14: Output image-6 with the lane detected.

Figure 15: Output image-7 with the lane detected.

Figure 16: Output image with the lane detected.