ENPM673

Project – 2: Lane Detection

Submitted by –

| Chayan Kumar Patodi |

I Prasanna Marudhu Balasubramanian |

| Saket Seshadri Gudimetla |

INTRODUCTION

In this project, we were assigned to do a simple Lane Detection, like that used in Self Driving Cars. We were provided with 2 video sequences, Project Video and Challenge Video to perform the task. We were given a few steps in the project description, which gave us an initial start and helped us a lot throughout the project.

The project was great and helped us learn various concepts such as Homography, Warping, Masking and Thresholding.

WORK DONE & RESULTS

To begin with, as suggested, we wanted to extract four coordinates of four points on the lanes. To do this we created a code, to select four points on the lane and print those coordinates, so that we can use them as source points for Homography later in the project. The whole idea of selecting the 4 points randomly is that, the lanes are parallel to each other and almost always vertical, which makes any set of points on the lane a generalized set, that can be used for the entire code.

Our code, when executed for the first time, will capture the video and read all the frames. The frames were saved at first, though it was not needed, so we modified the code, we only selected a common frame in which the car was moving straight in both the videos. We have selected **frame 348th** for our task. Once the frame is saved, we read that frame and we ask the user to select the four points on that image. Those selected four points will be used as source points for the final code. We selected [632,470], [698,470], [990,696], [302,696].

The above code can be found in the file named **Calculate_Points.py** attached with this report.

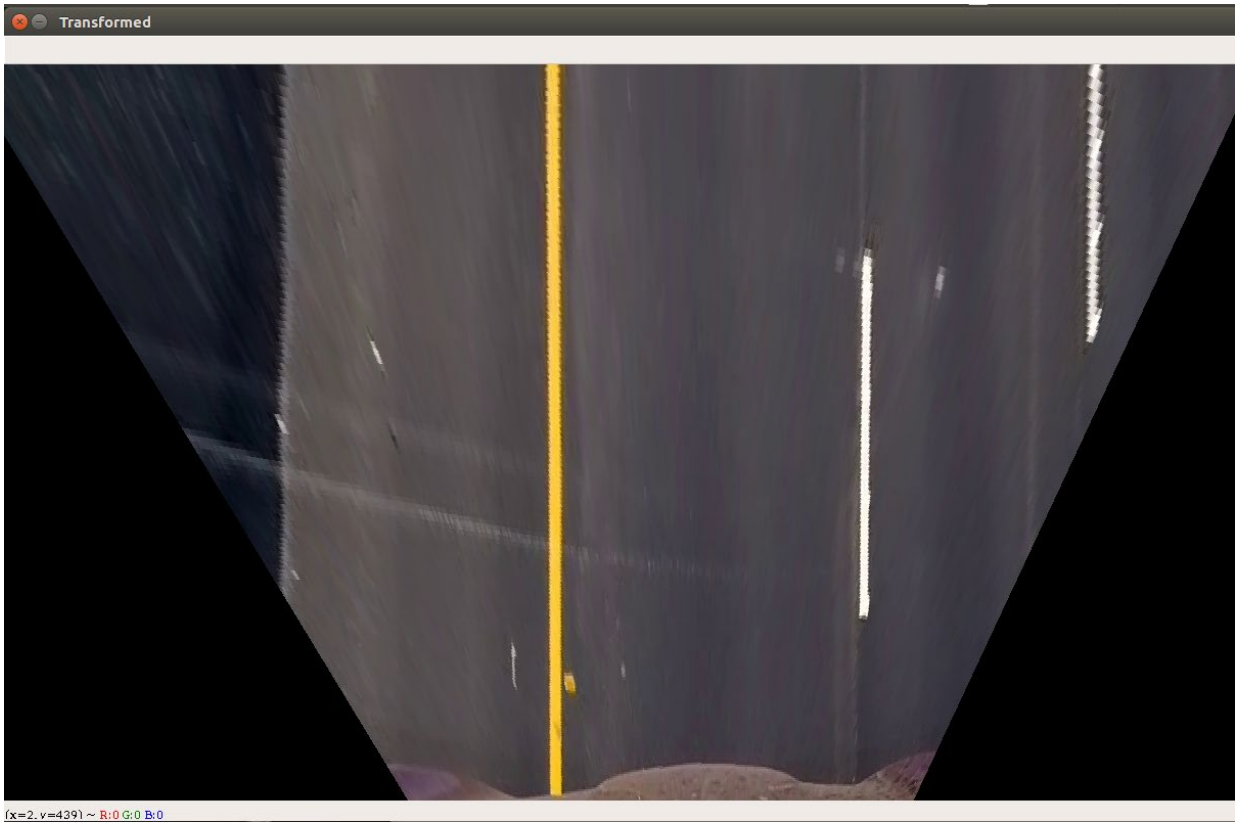
Following the next step, we started preparing the input. We applied pre-processing and in the while loop, each frame is processed independently.

We used the function **cv2.undistort** to undistort the frames, as we were provided with the camera parameters already which included the camera matrix and distortion matrix. We call the **function image binary** on the frames. Within this function, we are changing the color space of the image. In this function we are amplifying the intensity of the yellow color, using a mask, which makes it easy for the algorithm to detect lanes. This is an important step for the challenge video, because of a middle extra edge in the video. We are also using a median filter here, for smoothening of the image. We have also used sobel transform for emphasizing the edges. We

convert the image from HSV to grayscale to black and white, which means we are generating a binary image as the output of this function.

We are also selecting a ROI at this point on the code, using the **function region_of_interest**, which helps us eliminate sky and the bridge. We are eliminating top half of the screen and not creating a ROI in the lanes, so as our code can be generalized were all the lanes, that is if the car wants to change the lanes, it should work fine in that condition as well.

We are calling in the **warp function** here, which uses cv2.getPerspectiveTransform and cv2.warpPerspective which helps us calculate the Homography. We used the **function fit_from_lines** after warping the image, to calculate the left and the right points and then we used the **sliding window function**, to create the sliding windows. This sliding function uses the histogram and we are calculating the peak of the left and the right halves of the histogram, which would become the starting point of the lanes. We have been poly-fitting the polynomial to the detected lane candidates for better results, as suggested by the description. The other suggested method suggested was Hough lines, which we didn't use in this project.



Finally, we are calling the **draw lines function**, which concludes our code and merges every other output we found in the code, using inverse perspective matrix and other methods.

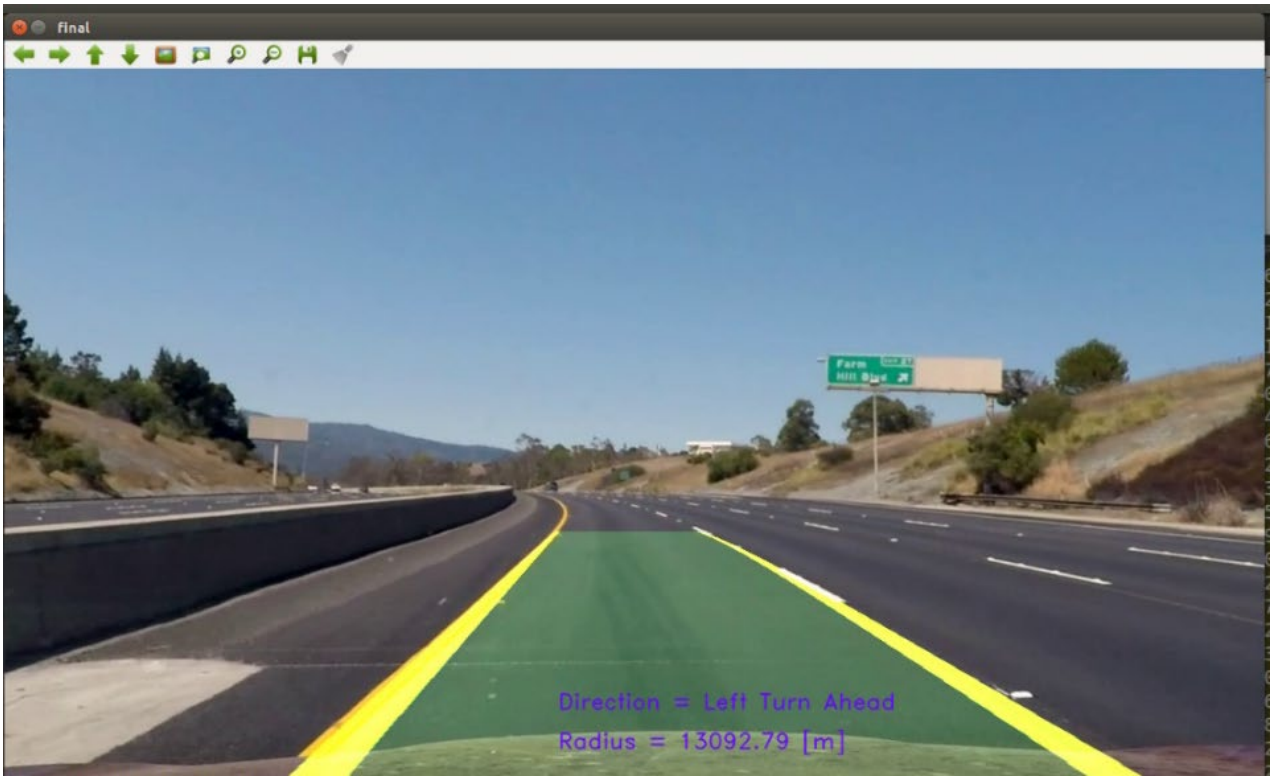
The inbuilt functions that have helped us throughout the code, are **cv2.addWeighted, cv2.polylines, cv2.fillPoly**, which have helped us throughout the masking.

Turn Prediction:

To determine the turns while the car is moving, what we did was, we monitored the left lane points/co-ordinates. If the car is moving left, the value of the points decrease (becomes less than 200 in our case), if the car is moving straight, we get the points in the range of 200-475 and if the car moves right, the left lane coordinates increases beyond 475 & thus predicting the turns for the car.

Whenever there is a turn ahead, the code predicts the turn and prints Left Turn ahead or Right turn ahead, depending on what turn is coming next.

The code for the above-mentioned steps could be found in the file named **LaneDetection.py,** attached with this report. We have created this code in **Python 2.7** and it works fine in it.



Questions specifically asked for the report:

1. Homography: In laymen terms, Homography is nothing but projection of one plane to another. It relates the pixel coordinates of one image to another and when applied to every pixel, the new image is nothing both a warped version of itself. We have used Homography here, using the inbuilt functions. It was necessary because the actual camera and the perception camera will view the road from different angles and will have different views and thus warping is important.

2. We have not used Hough Lines in this project, though we read about its basics, while we were deciding on which method to work with and we then choose the Histogram method for lane detection. The Hough transform is a way of finding the most likely values which represent a line or a shape.

3. Our detection pipeline is a generalized code and would work for other similar videos. We also tested this code on other videos, apart from the given input sequences and it was a success.

Problems faced:

1. The major problem we faced was in the challenge video. When the car went under the bridge, the lanes got into darker region, because of the shadow of the bridge. We couldn't detect the lanes in that part, and the lanes would become unstable. To solve this, we tried different methods such as using B channel of the LAB color-space, tried the HLS color-space and tried manipulating the brightness, but the solution that worked for us was the Gamma Correction. The reason we apply gamma correction is because our eyes perceive color and luminance differently than the sensors in a digital camera.

2. The 2nd problem we faced was, the intensity of yellow lane was a bit less and was causing problems while detecting the lanes. What we did to solve that problem, by applying a yellow mask on the yellow lane, which helped us identify the lanes, while working on the problem and giving us the accurate results.

References:

1. Advances in Visual Computing: 4th International Symposium, ISVC 2008, Las Vegas edited by George Bebis, Bahram Parvin, Darko Koracin, Richard Boyle, Fatih Porikli, Jörg Peters, James Klosowski, Theresa-Marie Rhyne, Laura Arns, Yu Ka Chun, Laura Monroe.
2. M. Foedisch and A. Takeuchi, "Adaptive real-time road detection using neural networks," in Proceedings. The 7thInternational IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749), pp. 167–172, Oct 2004.
3. 2) A. S. Huang, D. Moore, M. Antone, E. Olson, and S. Teller, "Finding multiple lanes in urban road networks with vision andlidar," Autonomous Robots, vol. 26, no. 2, pp. 103–122, 2009.
4. J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," IEEE Transactions on Intelligent Transportation Systems, vol. 7, pp. 20–37, March 2006.