Project 2 - ENPM 673
Spring 2022

Nishad Kulkarni
A. James Clark, School of Engineering
University of Maryland, College Park
UID - 117555431 Email: nkulkar2@umd.edu

***Abstract*—The following document is the report for Project 2 for the class ENPM 673, held in the Spring 2022 Semester. There were three questions in the Project, thus the sections are divided according to these questions.**

I. Introduction

Readers must first note that we will first see the directory structure of the submission in this section. The submission is titled `nkulkar2_prj2.zip`. The assignment codes were written in Python(3.8.10) and used the packages numpy(1.22.3), matplotlib(3.5.1), and opencv(4.5.5, opencv-contrib-python).

```
nkulkar2_prj2
├── code
│   ├── p1.py
│   ├── p2.py
│   └── p3.py
├── Output
│   ├── 01.png
│   ├── ...
│   ├── 25.png
│   ├── ada_hist.png
│   ├── basic_hist.png
│   ├── clahe_hist.png
│   ├── Hist_11.png
│   ├── op.png
│   ├── original.png
│   └── trapezium_clipped_p2.png
├── nkulkar2_prj2_report.pdf
└── README.md
```

II. Question 1 - Histogram Equalization

Histogram equalization is basically the evening out of intensities over the entire image. A histogram is a plot of intensity values and how many of those are present in an image. Histogram is defined by the equation:

$$h(i) = \sum_{x}\sum_{y}[I(x,y)] = i \tag{1}$$

Further, we find the Cumulative Frequency Distribution (CFD) of this histogram, using the equation:

$$C(i) = \sum_{j<=i}^{N} h(j)/N \tag{2}$$

Using this CFD, we reassign every pixel as follows:

$$x_i^{new}, y_i^{new} = (x_i^{original}, y_i^{original}) * C((x_i^{original}, y_i^{original})) \tag{3}$$

This is the basic histogram equalization. The output of this can be seen in the following figure:

Fig. 1. Original image

Fig. 2. Histogram Equalized Image

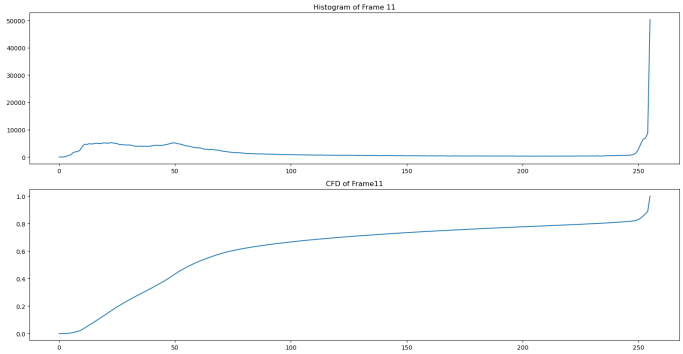

Fig. 3. Histogram and CFD

In some cases, the histogram must not be equalized over an entire image. For this there exist another method called Adaptive Histogram Equalization. In this method the entire image is divided into a grid of tiles- usually of size (8 x 8), and then histogram equalization is performed individually on these tiles. The output for this type of histogram equalization can be seen below.

Fig. 4. Adaptive Histogram Equalized Image

Another variant of Adaptive Histogram equalization is the aptly named Contrast Limited Adaptive Histogram Equalization (CLAHE). This method 'clips' all the intensities above a certain threshold and divides them among all the intensities. The output of this tyow of histogram equalization can be seen below.

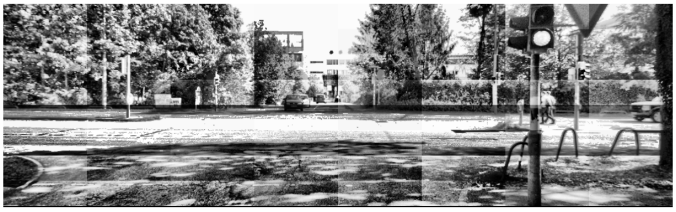

Fig. 5. CLAHE Image

Readers may notice that although Adaptive and CLAHE histogram equalizations are better in preserving features, they end up add phantom edges in the resulting image. This is due to the tile division. The code for this question is titled `p1.py`. The ouput of all the frames can be found in the directory titled `Output`. Also, a looped video of all the 25 frames stitched together has been uploaded, the link for which can be found in the Video Links Appendix. A final comparitive output can also be seen below.

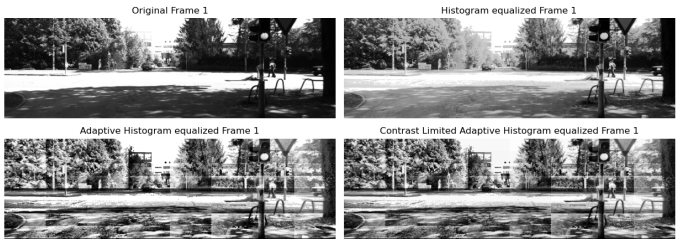

Fig. 6. Final output of this section compared

III. Question 2 - Lane Edge Detection

For knowing that a car is in a lane, it is necessary to detect the edges of a lane. This is acheived programatically by following a specific pipeline for every frame:

1) Convert frame to grayscale.
2) Consider lower half of the image only(Optional).
3) Apply a mask that only selects pixels in probable zone where lane might be. Usually, this is a trapezium like shape.
4) Threshold masked image to get brightest points, as lane edges are usually white.
5) Perform Canny edge detection (Optional)
6) Perform Hough Line detection to get lines of two separate types - long and short.
7) Mark long lines with green and short lines with red.

As step 1 and 2 are fairly self-explanatory, we will start understanding the pipeline from step 3.

The reason that a trapezoidal mask isapplied is due to the fact that there are multiple elements in the background environment that may introduce noise. For instance, the edge of the road is also detected as an edge in both Canny and Hough line detector. Also, as the reflection from the spokes of the wheels of other cars in the neighbouring lanes can add to this noise as well. Finally, a minor, but signifiant addition comes in the form of garbase and paint spots on the road itself. The output of the application of the trapezoidal mask can be seen below. Note that it is a trapezium and not a triangle as the upper edge is very small.

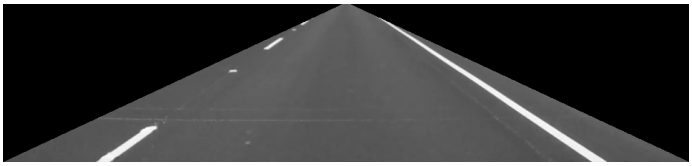

Fig. 7. The lane after application of a trapezoidal mask

Canny edge detection is again an optional step to detect hard edges. This is followed with Hough line detection. In the accompanied code titled `p2.py`, this is acheived using the `cv2.HoughLinesP()` detector. This is used as the function `cv2.HoughLines()` gives continous lines instead of line segments. Before proceeding any furhter we will first understand the concept of Hough Line Detection.

Hough Line Detection - Hough Transforms are a voting scheme for feature detection/extraction. This can also be used to detect edges when there is a break in pixels.

The idea of Hough transforms is to represent a point in Equalidean space into the Hough space such that, if there exists line in the euclidan space, it will be represented in the Hough space by it's perpendicular distance from the Euclidean origin denoted by ***'p'*** and the angle it makes with the positive x - axis θ. A line, thus becomes a point in Hough Transform.

For this question we used `cv2.HoughLinesP()` which gives the extremes of a detected line.

Further we get two separate sets of lines with 2 distinct mimimum length thresholds. The set with lower threshold includes all the lines above that threshold, meaning that it also includes the longer lane edge. We simply select the lanes with a mmaximum length and mark them with red. Further, we use the second set directly to mark lines with larger threshold with the color green. The output of this program can be seen in the image below.

Fig. 8. The Output of line detection

This program may supposedly be generalized for any other video, assuming that the lanes are within the detectable trapezoidal zone.

IV. Question III - Lane Detection

The final question in this project was focussed on detecting lanes on a road. For this, the main players are homography/warping, thesholding, and polyfitting. The pipeline that is foolowe in the accompanied code titled `p3.py`, is as follows:

1) Convert frame to grayscale.
2) Apply a mask that only selects pixels in probable zone where lane might be. Usually, this is a trapezium like shape.
3) Apply bilateral filter (optional)
4) Find Homography, Inverse Homography, and warped image.
5) Threshold warped image to get brightest points, as lane edges are usually white.
6) Apply sliding window algorithm on left half and right half within a certain zone.
7) Polyfit if line is detected or use previous data.
8) Find curvature of fitted curves.
9) Use inverse homography to backproject detected curves onto original image and fill the region between them as the detected lane.
10) Depending on the average curvatures of left and right pixels, decide direction of turning.
11) Pass current fits to next frame and repeat whole process.

Homography - Homography can simply be understood as the projective transformation between a set of points between two projections. This is usually between a set of matching features in 2 projective spaces. It is also very useful to visualise an image from a different perspective and in undistortion of images. Mathematically, it is the rotation, translation, scaling, distorting/undistorting of an image.

Just like earlier, we use a trapezoidal mask to isolate probable lane zone. Now, we warp the zone to see the lane from a top-down perspective as seen in the figure below.

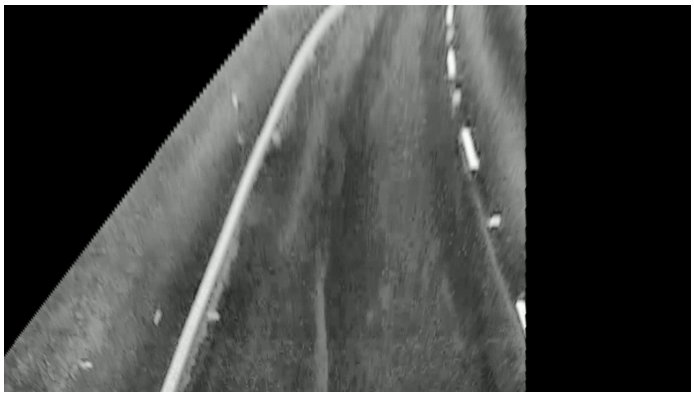

Fig. 9. Warped road from a top-down perspective

Please note that the view is not perfect as the parameters had to be varied to get a good output and were found by hit or miss. Although, the perspective can be found parametrically, it is not perfect as far as lane detection is concerned and gives a very small patch. Further, this image is thresholded. Post this, the left and right pixels are marked. Finally, curve fitting is performed as seen in the image below.

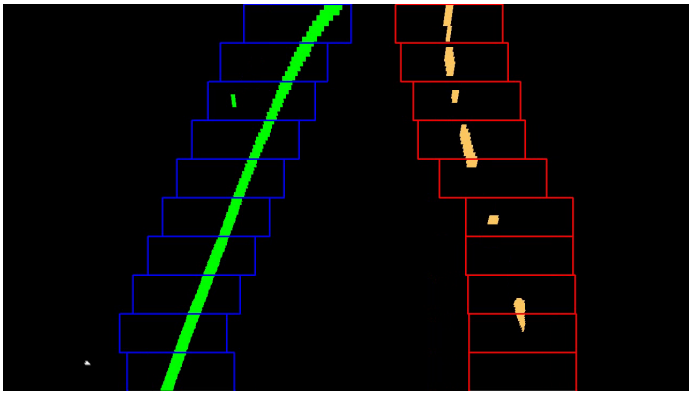

Fig. 10. The Output of line detection on warped-thresholded image

Finally, the curvature of these lanes are found. We backproject the image onto the original color frame to get the detected lane edges and the inlying lane. The curvature of left and right lanes are displayed, along with the average curvature of the lane itself and a suggest direction to turn the wheel depending on a certain threshold.

Fig. 11. The Final output of lane detection

This pipeline makes multiple assumption:

1) The road is almost purely planar and there are negligible inclination variation
2) The lane is of standard width 12ft and dashed edges are of standard height 10ft.
3) Intensity vriation is not of very high frequency and doesn't occur often. This is noticeable when we have the concrete patch which ha very bright pixels, and also during the appearance of the tree's shadow, The latter was dealt by using CLAHE on the frame to enhance features.
4) Vehicle is travelling continously on a single lane and doesn't switch.
5) Homography is assumed to be correct. The parameters were hardcoded.
6) Camera has zero motion with respect to the vehicle itself.
7) Many of the parameters were hardcoded. Best case, this pipeline will work for a flipped version of itself, or another video that satisfies the above consraints.

Thus the project was completed successfully....more or less.

V. Appendix - Video links

Following are the links to the videos for this project.

1) Combined Histogram output - Link1.
2) Lane edge detection - Link2.
3) Warped Lane - Link3.
4) Warped Thresholded Lane edge detection - Link4.
5) Final Lane detection - Link5.