Perception for Autonomous Robots

ENPM673

Project 2

Author: Cheng Liu (117694802)

Date: 04/04/2022

Contents

Problem 1 – Histogram equalizations

This problem aims to improve the lighting conditions of the image. There are 25 input images given. To achieve this goal, histogram equalizations can distribute the intensities of the image and make them evenly spread.

Fig 1. Original image

The main idea of histogram equalizations is cumulative distribution function (cdf), which is

$$cdf_x(i) = \sum_{j \le i} h(j)/N$$

Indicates the fraction of pixels with intensity less than or equal to i, and assumes the image has N pixels.

Fig 2. Image after implementing histogram equalizations

In Fig 2, it is easy to notice the upper region of the image can show more details than original image. The darker region is also brighter than original image.

For adaptive histogram equalizations, it is based on histogram equalizations. This method divide image into several pieces and utilize histogram equalizations on each piece. In the code, the height and the width of the image are divided by 100 in order to shorter the output time.

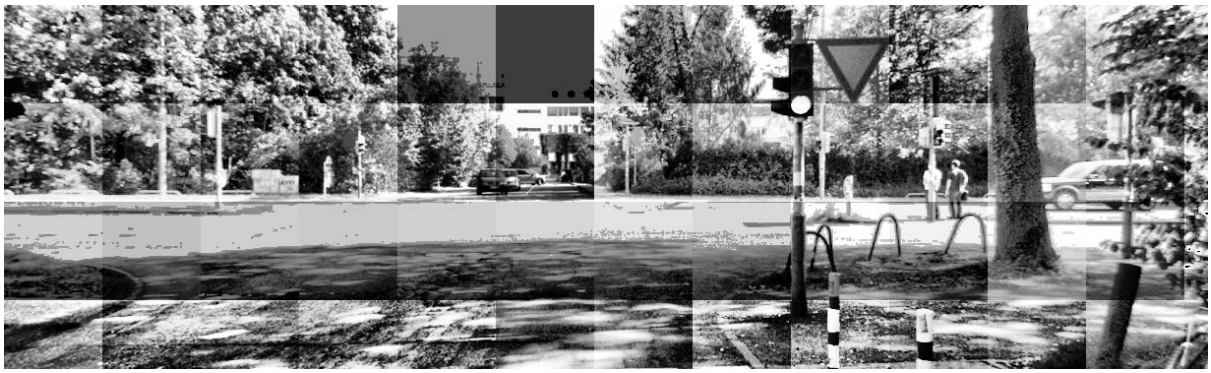

Fig 3. Image after implementing adaptive histogram equalizations

In Fig 3, though the grids that separates the image can be clearly observed, the details of the image are actually better than histogram equalizations. We can view more details on the trees, people, and the car in the background. The reason why adaptive histogram equalizations perform better is that the difference of the grayscale grids isn't larger than the whole image. Therefore, the grids can distribute the intensities better than the whole image.

Output video link:

https://drive.google.com/drive/folders/10EhxkAJQ7HGrJ3pTXs_1FpJkgcG29SnA?usp=sharing

Problem 2 – Straight Lane Detection

In this task, we are given a short video contains lanes on the road. The goal is to classify the dashed and solid lines on the road. Green is for solid line and red is for dashed line.
To achieve this goal, the following steps should be implemented

1. Leave yellow and white pixels to remove unnecessary edges
2. Convert image into grayscale
3. Implement CLAHE to adjust light conditions
4. Utilize Gaussian blur to smooth the image
5. Use Canny to find edges
6. Select a smaller region which contains the lanes we want
   (40% height, 7% top width, 85% bottom width)
7. Use HoughLinesP function to detect the lines in above region
   (rho: 2, theta: np.pi/180, threshold: 15, minLineLength: 10, maxLineGap: 20)
8. Separate the lines to right and left lines and use polyfit function to get the best fit
9. Use Bresenham algorithms to find the pixels in lines and check if they are dashed or solid
10. Draw the lines green for solid and red for dashed

In the second step, CLAHE is implemented. Contrast Limited Adaptive Histogram Equalization is to set the specified contrast limit (40), and those pixels are clipped and distributed uniformly. Then remove artifacts in tile borders after applying histogram equalizations.

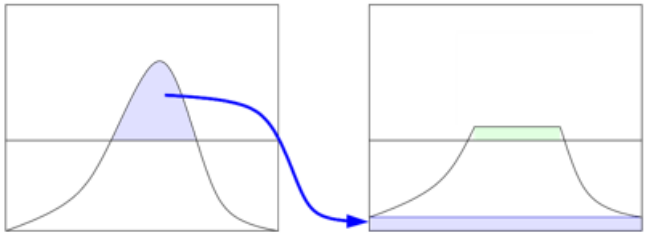

Fig 4. Clip the intensities of image and distribute them evenly

In the seventh step, HoughLinesP function is implemented. Hough Transform can detect the lines even they are broken or distorted. The theory behind it is that using (ρ, θ) represents all the lines in the image. ρ is the length of the line and θ is the angle of the line. By voting the cell, we can assume that there is a line in the image with maximum votes.

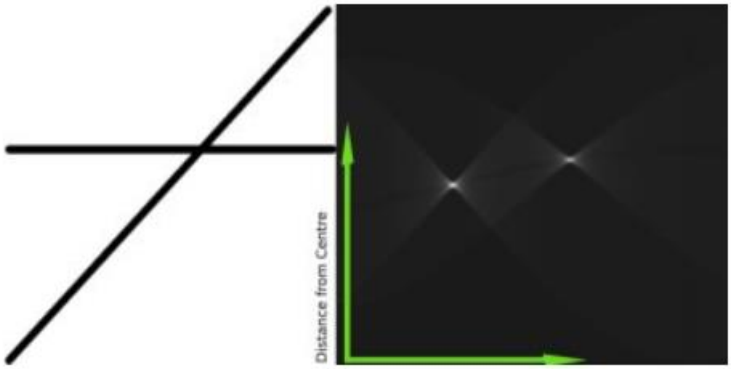

Fig 5. The two bright point shows that they are possible lines in this image

In the ninth step, Bresenham algorithms are implemented. This algorithm is used to find the pixels which the line passes through. In order to classify the dashed and solid lanes, we can utilize Bresenham algorithms to get the pixels and check the number of white pixels. Normally, dashed line will have fewer white pixels than solid lanes. For only testing those pixels on one image, the results of solid and dashed line are 105/338 and 32/313 separately in the image. However, when applying this method on video, it shows bad result. Apparently, the pixels are not enough for classifying. Therefore, when assessing the pixels, we consider 5 pixels along x axis to get more dataset, the difference of the number of white pixels will increase. Finally, the output video could show most of the frames correctly.

Fig 6. Original image for testing

Fig 7. Output shows green for solid and red for dashed

Output video link:

https://drive.google.com/drive/folders/1E7EMC8r4tsXkI-V69VN0A80vicfQKiH7?usp=sharing

Problem 3 – Predict Turn

This task is to detect the curved lanes and predict the turn. A short video is given. There are a yellow and white line in video for predicting turn. To achieve this goal, the following steps are necessary.

1. Filter yellow and white colors to ignore unnecessary edges
2. Convert the image to grayscale
3. Use CLAHE to get more details in image
4. Use Gaussian Blur to smooth the image
5. Threshold the image from 100 to 255 and select the interested region
6. Warp the image to get the view from the top
7. Slide 9 windows to get better line detection
8. Find the radius of curvature to estimate the turn
9. Draw the lanes, the area between two lanes, and the arrows in the middle
10. Use inverse matrix to warp the drawn image combining with original image

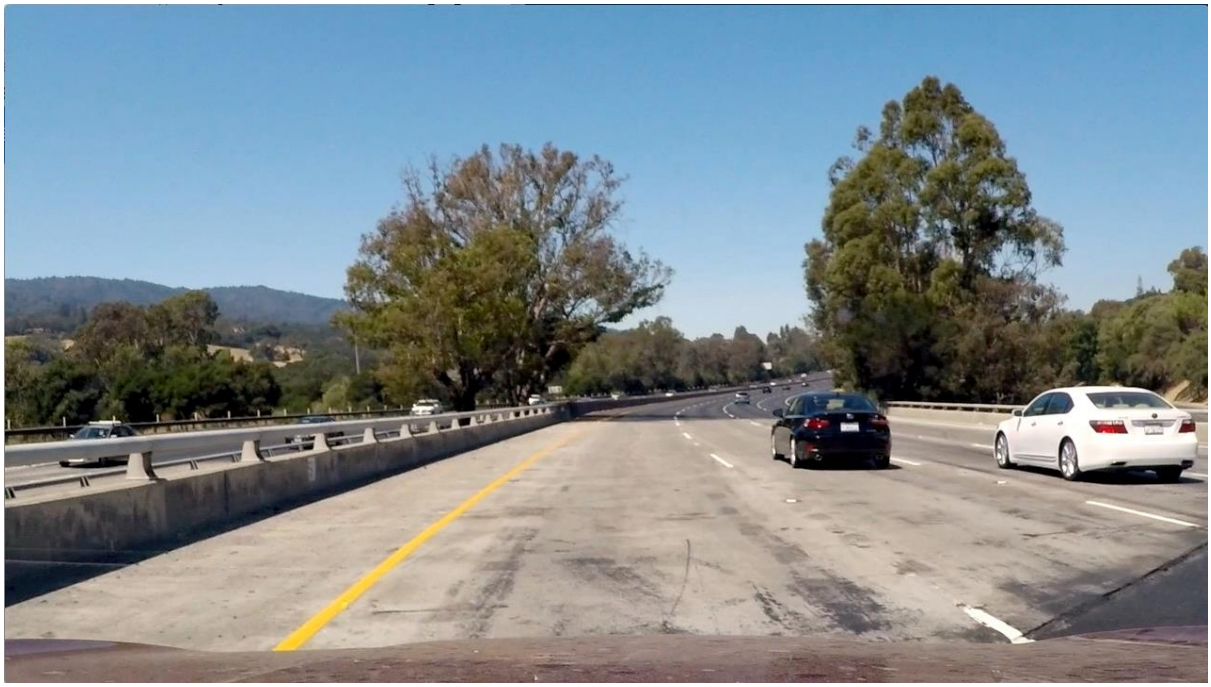

Fig 8. Original image

Fig 9. Filter colors image

Fig 10. Grayscale image

Fig 11. CLAHE and GaussianBlur image

Fig 12. Threshold image

Fig 13. Warped image

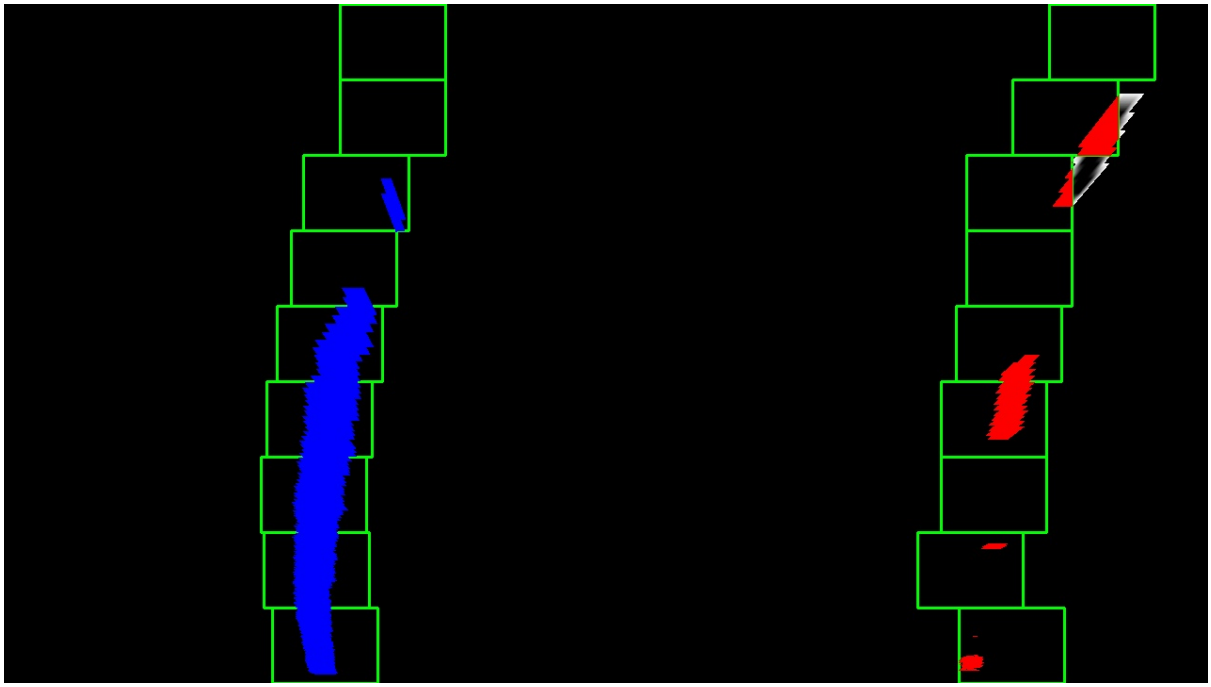

Fig 14. Sliding windows to detect the lines

In step 7, it is hard to detect lane lines all the time. Therefore, we set global variables to store the left line and right line. Once sliding windows cannot find the lane, it will access the history in Line class and utilize previous lines to fit current image.

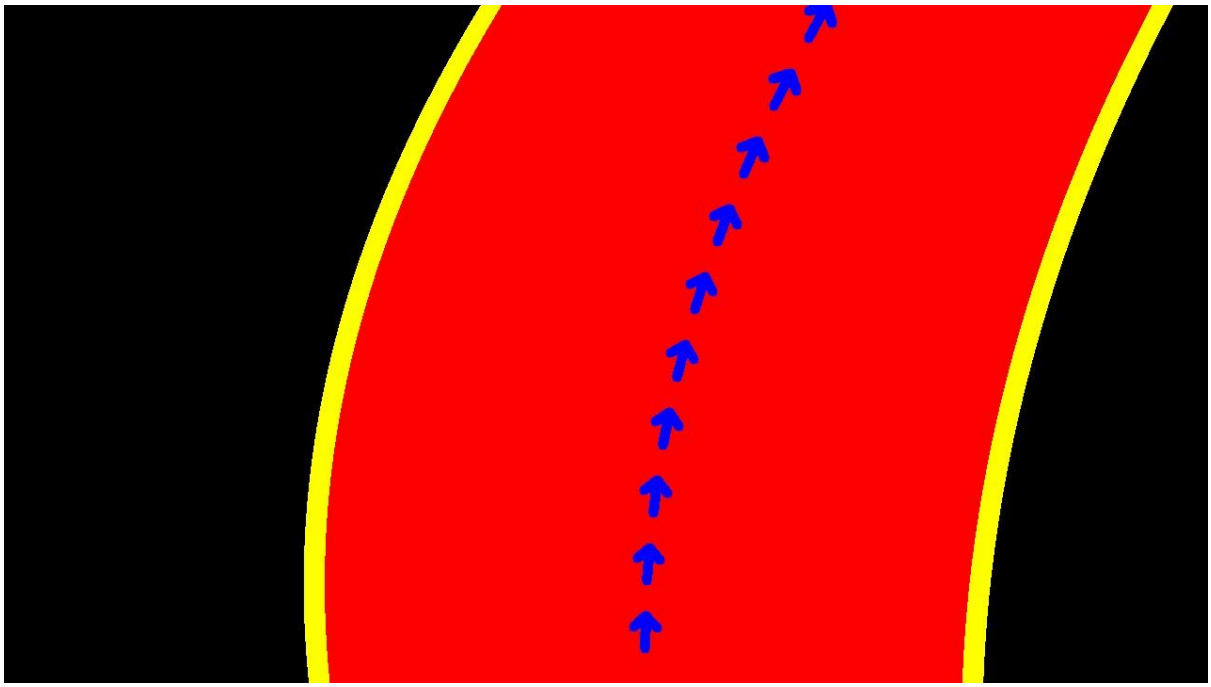

Fig 15. Draw lane lines, road, and arrows in the warped image

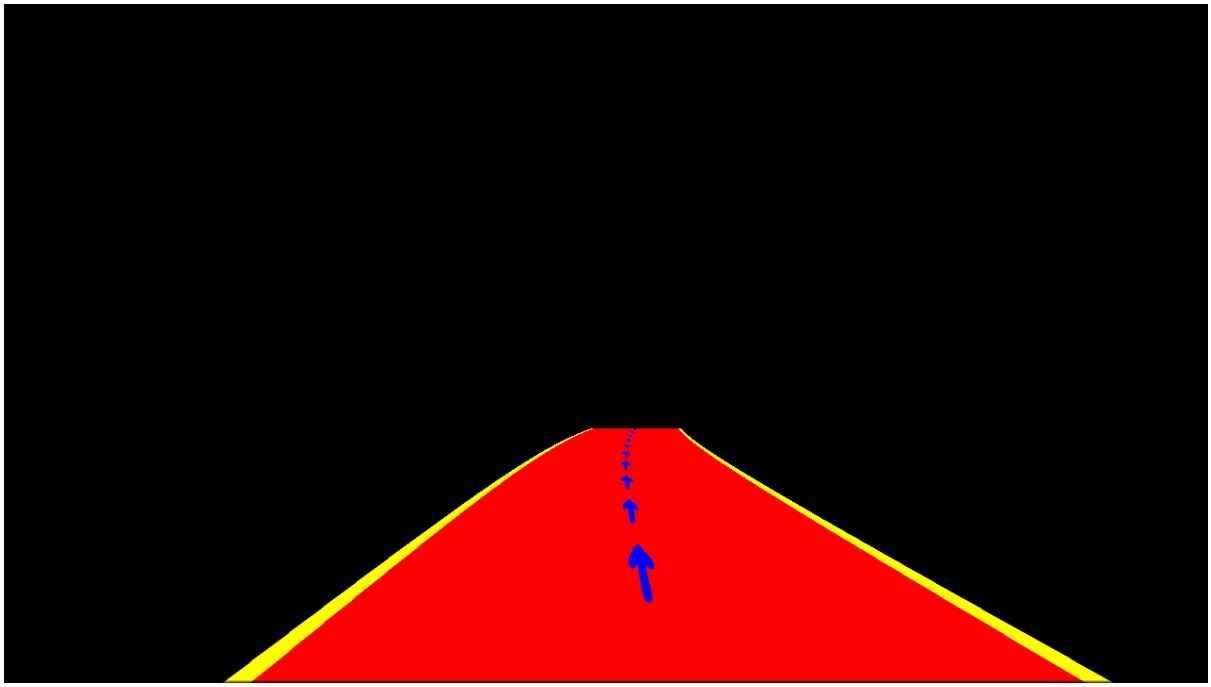

Fig 16. Use inverse matrix to warp the image again.

Fig 17. Result image

Output video link:
https://drive.google.com/drive/folders/1C9QMBQ1SKsuy7qaNk_fca4nwTZSKATgJ?usp=sharing

Whole project link:
https://drive.google.com/drive/folders/1JSFxF9J0mrzsxrWXbACJ10IsT8oBbDOv?usp=sharing

Reference

- Farag, Wael. (2020). A Comprehensive Real-Time Road-Lanes Tracking Technique for Autonomous Driving. International Journal of Computing and Digital Systems. 9. 349-362. 10.12785/ijcds/090302.