Homework #4

B05902120 / Yu-Ting, TSENG

May 13, 2019

Basic Execution

- Platform: Unix (MacBook)
- Language: C++

Problem1: Hough Transform for Line Detection

(a) Please perform edge detection on I_1 and output the resultant edge map as E.

There are several methods being taught for edge detection, including First Order Gradient and Second Order Gradient. To avoid messy output image, I choose "Canny Order". We can find that the top-right part of the image disappeared, this is because of the different chosen threshold. The bigger threshold we choose, the more details disappear. However, if we chose threshold too small, lots of details would be included, which would make the image messy. Hence, I set the variable `T_h` to be 48 and `T_l` to be 45.

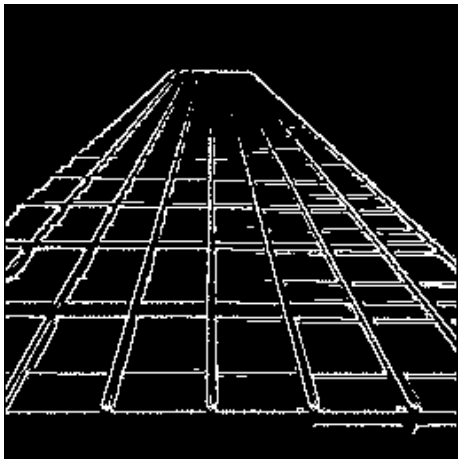

Figure 1: E.png

(b) Perform Hough transform on E and output the accumulator array as a new image, H_1, where the horizontal axis and vertical axis represent theta and rho values, respectively.

The concept is that transform x-y coordinate to r-ρ coordinate and mark each point in the new coordinate system.

We make use of the function $\rho = x\cos\theta + y\sin\theta$ and count the appearance times of each value. At last, we plot it on the graph by ρ as i and θ as j.

```
for (int i = 0; i < size; i ++)
    for (int j = 0; j < size; j ++){
        if (img[i][j] == 0) continue;

        for (int k = 0; k < 180; k ++){
            int value = int(i * sink[k] + j * cosk[k]);
            tmp[value + size * 2][k] ++;
        }
    }
```

Worth mention is that to prevent negative numbers we add `size * 2` on ρ. In this case, we will be able to record the occurrences of each value. As a result, we can get an image. To display in the report, I firstly crop the upper and lower part which is all black. Besides, I execute some distortion and the below image is the results after these steps.

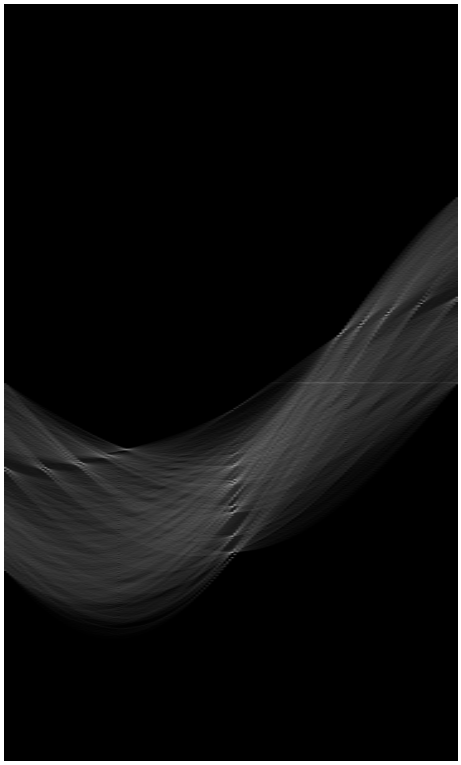

Figure 2: H_1.png

(c) Please perform contrast adjustment on H_1 and output the result as H_2 for better visualization.

Observing the above resultant image, we might feel it hard to see the situation clearly, therefore image enhancement is needed. Same as (a), there are also a great variety of methods we have learned from previous class, I take use of "Histogram Equalization" to execute enhancement. First, count the occurrences of each intensity values and rearrange them to distribute more uniformly. The image displayed below is the resultant image.

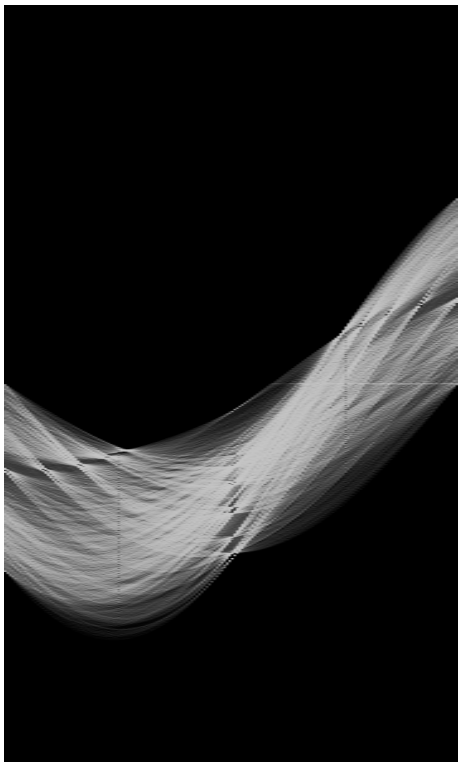

Figure 3: H_2.png

(d) By utilizing the accumulator array, draw the top 10 and top 20 significant lines with different colors on the edge map E and output the resultant images as D_1 and D_2, respectively. Please provide discussions about the pros and cons of Hough transform.

Based on these top largest intensity values, we can compute several formulas, then transform these formulas from r-ρ coordinate to x-y coordinate. I first get the binary images. Eventually label them with different color.

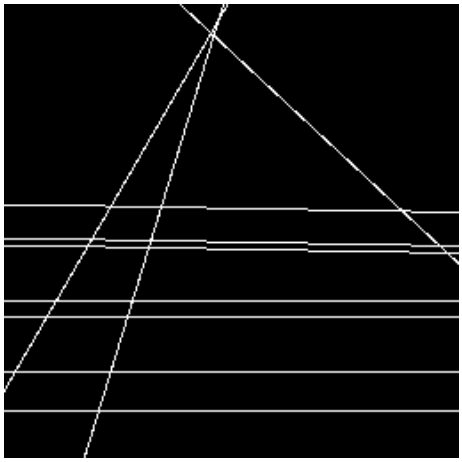

Figure 4: D_1

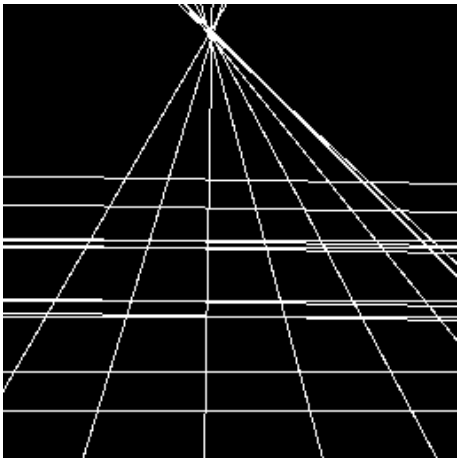

Figure 5: D_2

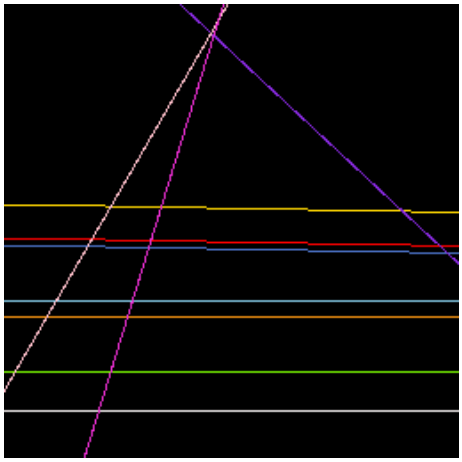

Figure 6: D_1

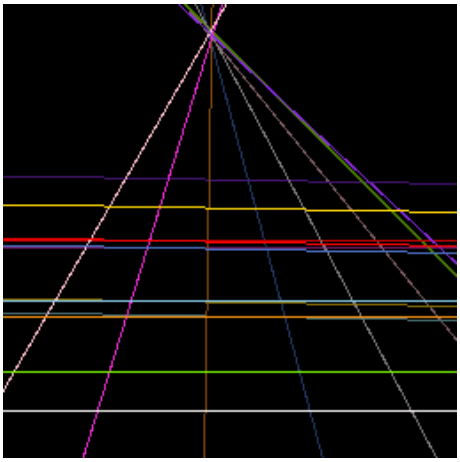

Figure 7: D_2

There are plenty of advantages of using Hough Transform. We might be able to detect lines with short breaks owing to noise. On the other hand, we can only get the infinite lines rather than finite lines with defined end points. Moreover, something might goes wrong if the objects are aligned by chance.

Appendix

- `Problem1_a.cpp` to execute edge detection.
  Compile by `g++ Problem1_a.cpp -o Problem1_a`;
  execute by `./Problem1_a` and get the output image `E.raw`.

- `Problem1_b.cpp` to execute hough transformation.
  Compile by `g++ Problem1_b.cpp -o Problem1_b`;
  execute by `./Problem1_b` get the output image `H_1.raw`.

- `Problem1_c.cpp` to execute image enhancement.
  Compile by `g++ Problem1_c.cpp -o Problem1_c`;
  execute by `./Problem1_c` get the output image `H_2.raw`.

- `Problem1_d.cpp` to execute hough detailed content.
  Compile by `g++ Problem1_d.cpp -o Problem1_d`;
  execute by `./Problem1_d` and get the output images `D_1.raw` and `D_2.raw`.