EECS 332 Introduction to Computer Vision

Machine Problem 6: Hough Transform

Name: Runlong Lin ID:3075255

1. Introduction

Hough Transform can be regarded as a feature extracted techniques in computer vision. It's mainly used for detecting lines, circles and other shapes. The core of Hough Transform is the voting procedure. This procedure takes place in a parameter space, from which candidates are obtained as local maxima in a accumulator space. In Hough Transform, the parameter space is (rho,theta) space. The relation between x-y space and rho-theta space is:

- A point (x0,y0) in x-y space is mapped to a sinusoid curve in rho-theta space.
- A set of co-linear points in x-y space is mapped to a set of lines which intersect at a particular point (rho,theta).
- The transform between (x,y) and (rho,theta) can be simply written as: rho=x*cos(theta)+y*sin(theta)

The purpose of MP6 is to implement Hough Transform from scratch and use it to detect the lines in the test images.

2. Algorithm Description

There are 4 main steps for Hough Transform. We use MATLAB to implement it and built up 4 functions:

- **Create a Hough Space**

First we should create the parameter space. This is simple as we have already known the transformation formula between x-y space and Hough space.

```
function [h] = hough(img_edge, num_r, num_theta)
```

img_edge is the edge image of the original image. In mp6, we use canny edge detection to find the edge image. num_r and num_theta define the size of the Hough space. h is the matrix of Hough Space.

- **Find the local maximum**

The second step is to find the local maximum in Hough Space. As the intersect point in Hough space is correspondent to a line in x-y space, the point with larger "votes" is likely to be mapped to the lines that we need to detect in x-y space. Here the function is :

```
function[h_local_max]=find_local_max(h,radius)
```

The method we find the local maximum is: we loop the neighbors of a pixel and see if this pixel is smaller than its neighbors. If it's true, we set this pixel as zero and check next pixel. Here the "radius" is the size of the neighbors region. Specially, we use the padding form of the Hough matrix instead of the original Hough matrix:

```
padding_h=zeros(len_h(1)+radius*2,len_h(2)+radius*2);
padding_h(1+radius:len_h(1)+radius,1+radius:len_h(2)+radius)=h
;
```

By doing so, we can guarantee that the border pixels' neighbors are still in the matrix.

- **Find Threshold**

  Then we should set up a threshold to filter some local maximum points in Hough space.

  ```
  function[th] = Find_Threshold(h_local_max, line_pre)
  ```

  Here the “line_pre” is the number of lines that we predict in the original image. In this function we use histogram methods and the special value “line_pre” to figure out the threshold. Actually, we can adjust line_pre to have more or less lines detected.

- **Draw the line in the image**

  ```
  function draw_lines(img_input, rho, theta,h)
  ```

  Finally, we obtain the voted (rho, theta). We can go back to x-y space and draw the lines using parameters.

3. Result Analysis

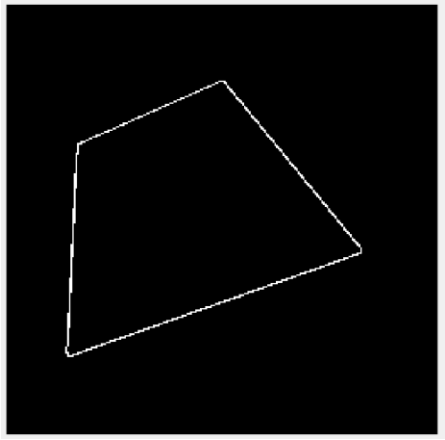

Fig.1 edge image

Fig.2 Hough Space

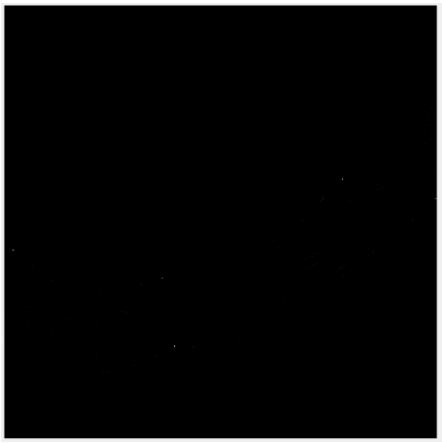

Fig.3 local maximum

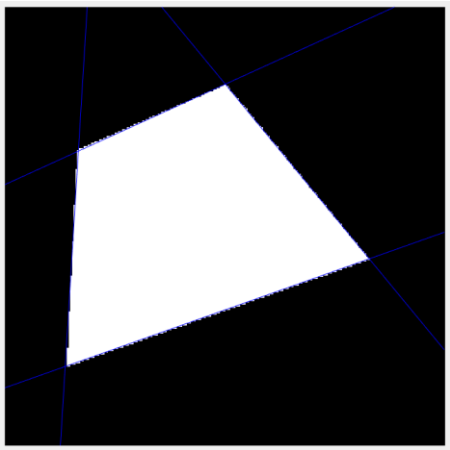

Fig.4 line detection

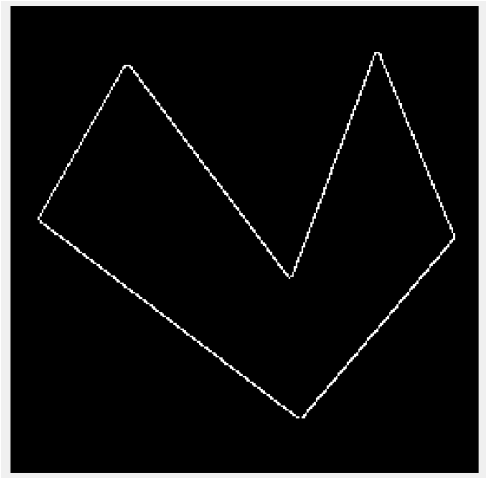

Fig.5 edge image

Fig.6 Hough Space

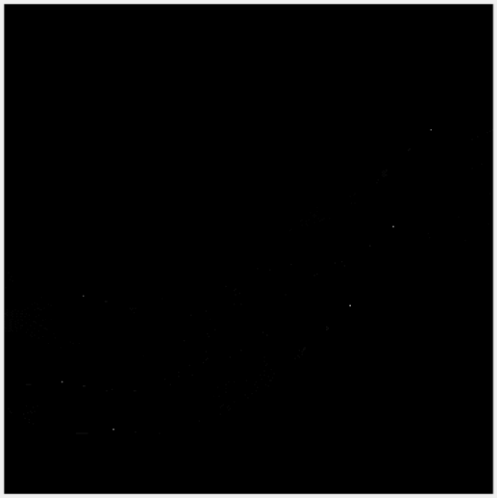

Fig.7 local maximum

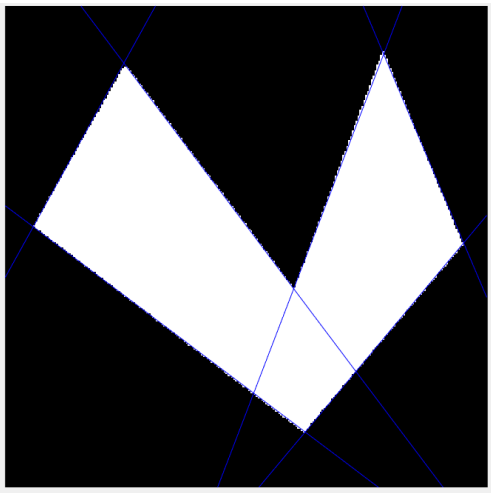

Fig.8 line detection

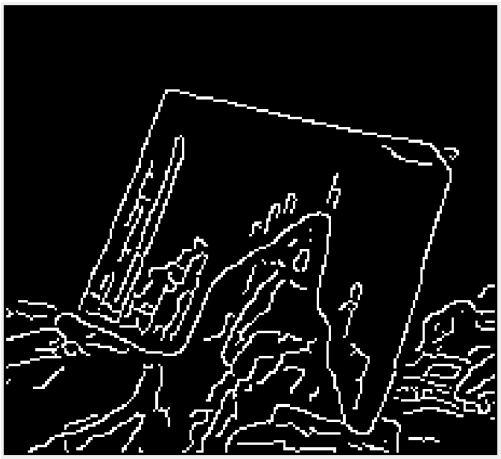

Fig.9 edge image

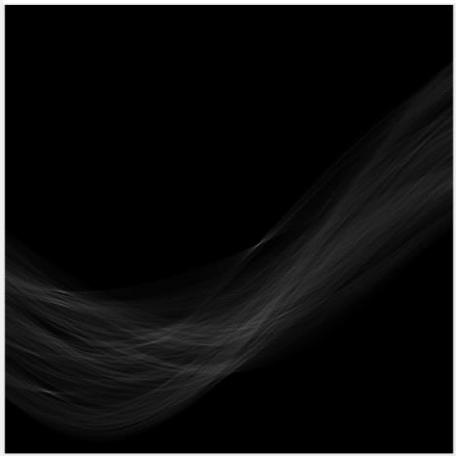

Fig.10 Hough Space

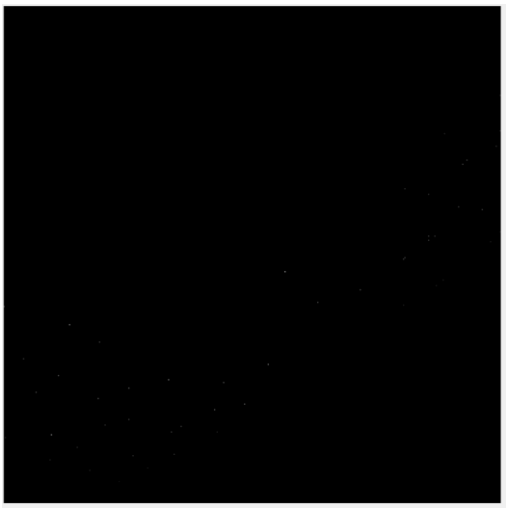

Fig.11 local maximum

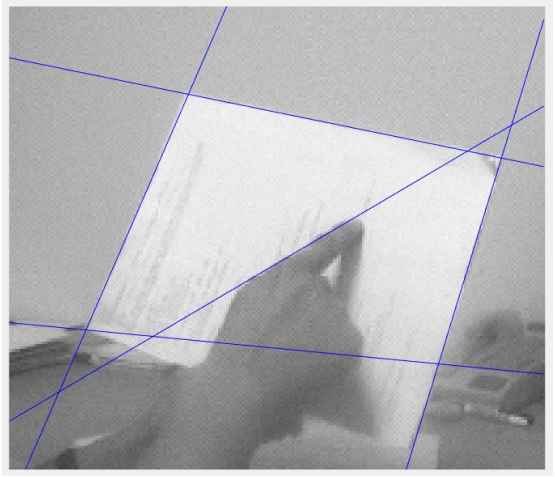

Fig.12 line detection

Analysis: As we can see in these figures, we successfully detect the lines in the image. But we still need some improvements. For example, in Fig.12, the algorithm can not detect the bottom line of the white paper accurately. The reason is that this line's correspond point in Hough space is so "weak" that it's under the threshold.