100296766 04/11/2022

CSE 107

Lab#4 Assignment

Edge detection

Andre Martin

Abstract: In this lab, you will implement a simple edge detection scheme. You will use the magnitude of the gradient to determine whether a pixel belongs to an edge or not. The gradient will be implemented using linear spatial filtering.

Technical Discussion: The first thing to do in creating the edge detector is to find out how the filtering works. For this problem we need a filter that tells us where in the image the pixel values change very differently. The 0 row ignores the outcomes that are in-between places that are more preferable in the 1's and 2's area. So in an example if the value hasn't changed often then the values would cancel out, thus giving you a filter value of 0. The equation underneath the technical discussion basically shows how the mapping of the index should be done for the filter. For the equation the filter is marked as "w" and the image is "f". The iterations both show "s" and "t" as indexes that check all eight areas from the current pixel that the filter has been applied to. Furthermore, theory zero padding shouldn't affect the image a lot since the values cancel out.

Spatial Filtering

$$\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

Sobel Filters

+1	+2	+1
0	0	0
-1	-2	-1

Horizontal

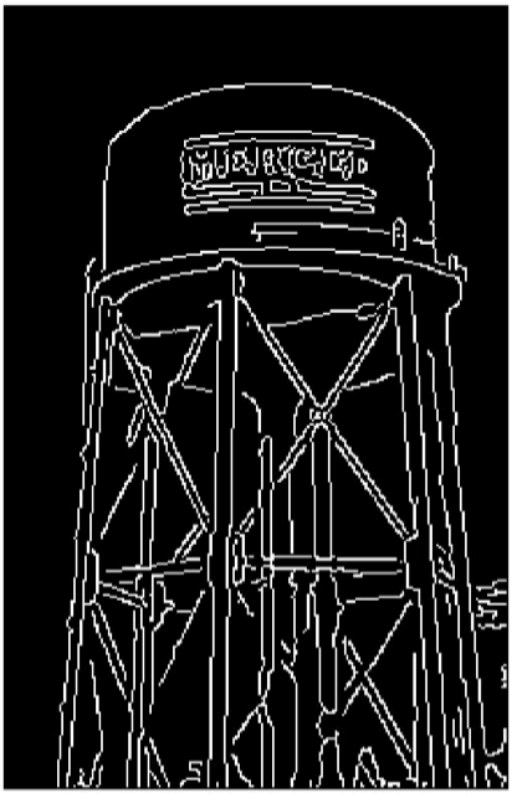
-1	0	+1
-2	0	+2
-1	0	+1

Vertical

Discussion of Results: When applying the edge detection while using the sobel filters turns out to be successful when the noise detected being the only problem. Since the sobel filter checks for changes in the pixel values it sometimes creates noise by objects or light that can only come up as an edge. When having a max sort of fixes, this problem is by reducing the amount of outcomes but it also gets rid of the details that are on the edges. The higher the threshold the thinner the edges become, thus getting rid of the the noise to a certain extent. The problem can be improved by implementing the gaussian filter that makes an image smooth and thus giving rid of un-needed edges. Furthermore, this is the the other type of filtering called Canny edge detection. In retrospect, the Canny edge detection surpasses the quality of detecting due to the ability and complexity in order to improve the image before putting the filter.

Original Greyscale Image

Sobel Edge Detector with Threshold of 200



Canny Edge Detector (MATLAB)

Code:

```
function [output] = spatial_filter(image,filter)
%spatial_filter
%
%INPUT: Grayscale Image (type double) and a filter
%
%OUTPUT: Matrix of gradient magnitude
%
%filters an image with a specific filter. Edges are avoided by using zero
%padding.
%

image = double(image);

imageSize = size(image);
imageRow = imageSize(1);
imageColumn = imageSize(2);

filterSize = size(filter);
filterRow = filterSize(1);
filterColumn = filterSize(2);

paddedImage = padarray(image, [filterRow, filterColumn] ,'both');

filteredImage = zeros(size(image));

filterCalculation = 0;

for x = 1: imageRow
    for y = 1: imageColumn
        for i = -1:1
            for j = -1:1
                filterCalculation =  filterCalculation +
(paddedImage(x+i+filterRow,y+j+filterColumn) * filter(i+2,j+2));
            end
        end
        filteredImage(x,y) = filterCalculation;
        filterCalculation = 0;
    end
end

output = double(filteredImage);
end


function [gradientImage] = gradient_magnitude(image)
image = double(image);

%Sobel Filters

%Horizontal Filter

filterY = [-1,-2,-1;
           0,0,0;
           1,2,1];
```

```
%Vertical Filter

filterX = [-1,0,1;
           -2,0,2;
           -1,0,1];

imageSize = size(image);
imageRow = imageSize(1);
imageColumn = imageSize(2);

gradientImage = zeros(imageRow,imageColumn);

gradientImageX = spatial_filter(image,filterX);
gradientImageY = spatial_filter(image,filterY);

for s = 1 : imageRow
    for c = 1 : imageColumn
        gradientImage(s,c) = sqrt((gradientImageX(s,c)^2) +
(gradientImageY(s,c)^2));
    end
end

gradientImage = double(gradientImage);
end


function [finalEdges] = findEdges(image,threshold)
edgeImage = gradient_magnitude(image);

finalEdges = uint8(zeros(size(edgeImage)));

imageSize = size(image);
imageRow = imageSize(1);
imageColumn = imageSize(2);

for s = 1 : imageRow
    for c = 1 : imageColumn

        if(edgeImage(s,c) < threshold)
            finalEdges(s,c) = 0;
        else
            finalEdges(s,c) = 255;
        end

    end
end

imshow(finalEdges);
end
```