Edge Detection

Sigi Lopez

11/13/18

For this lab we needed to implement a simple algorithm to solve edge detection. To start detecting edges we implemented a function for spatial filtering to find the differences in the image. The specific filters that we used were called Sobel masks two for each orientation. After finding those differences or "gradients" we calculated their magnitude which in result showed us how the image changed values drastically. The results from using this method for finding edges was not perfect. While most edges were detected on the image the filter also picked up some noise thus creating false edges.

Technical Discussion

The first step for creating the edge detector was to find out how filtering worked. For our problem we needed a filter that told us were in the image pixel values changed drastically. This is the reason we picked the Sobel filters. The 0 row ignores the values that are in between while giving more preference to the places where 1's or 2's. Therefore, in a simple example if the value doesn't change frequently the values would cancel out giving you a filter value of 0. In the equation below it essentially shows how the index mapping should be implemented for the filter. In this case our filter is marked as "w" and our image is "f" in the function below. The both iterations "s" and "t" are indexes that check for all eight adjacent blocks from the current pixel you are trying to apply the filter to. Another thing to point out is that in this lab we dealt with edge problems by implementing zero padding which essentially means that you add extra rows of zeros on the sides to avoid negative indexes. In theory zero padding should not affect the image that much since the values should cancel out.

Equations

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

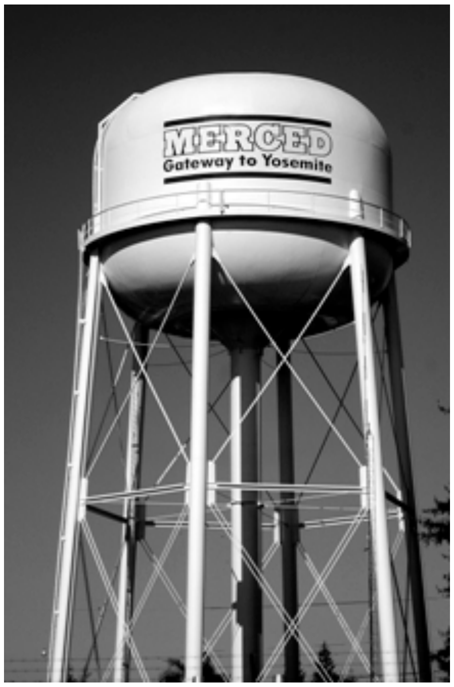
-1	0	+1
-2	0	+2
-1	0	+1

Vertical

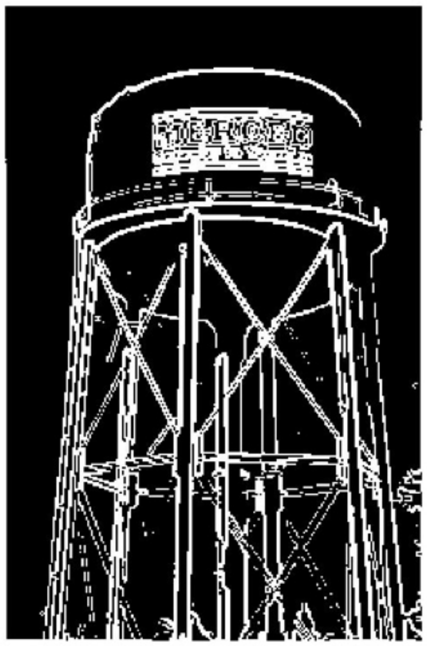
Discussion of Results

The results for applying edge detection using the Sobel filters turned out to be mostly successful with the only problem being the noise detected. Since Sobel filters check for changes in pixel values sometimes noise created by objects or light could come up as an edge. Applying a threshold sort of fixes this problem by reducing the amount of values but it also gets rid of details on edges. I noticed that the higher the threshold the thinner the edges became therefore eliminating noise to a certain extend. The problem can be further improved by implementing a gaussian filter that smoothens the image thus gets rid of unnecessary edges. This is something that another type of filter does which is called the Canny edge detection. I have displayed both results of the Sobel filter and Canny edge detection. Overall Canny edge detection surpasses the quality of detection due to its complexity and ability to improve the image before applying the filter.

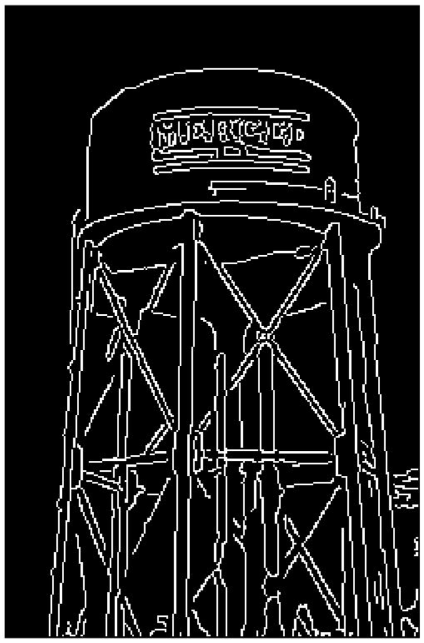
Results



Original Greyscale Image



Sobel Edge Detector with Threshold of 200



Canny Edge Detector (MATLAB Function)

Code

```matlab
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)
%gradient_magnitude
%
%INPUT: Grayscale Image (type double)
%
%OUTPUT: Matrix of gradient magnitude
%
%Calculates the gradient filters for both horizontal and vertical edges.
%Calculates the gradient magnitude of both directions.
```

```
%

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)
%findEdges
%
%INPUT: Image and a threshold
%
%OUTPUT: Matrix of 0's and 255's.
%
%255 or white essentialy tell you that
%theres an edge at that point. The threshold is a number that helps reduce
%noise depending on the image.

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
```