Edge Detection

Lab Assignment #4

Ralphilou Tatoy

April 14, 2022

Abstract:

In this lab, using spatial_filter, gradient_magnitude, and find_edges functions collectively, we can find the edges present in a grayscale image. Using spatial filter mask calculations, the use of Sobel Kernel matrix implements a gradient double matrix that identifies edges in an image. The find edges function will use this gradient matrix to selectively choose which pixel location are either 0 or 255 (white or black) by using a threshold value. The use of Sobel Kernel matrix identifies the rate of change of pixel values given a direction of a vector. The threshold helps identify which rate of change value from the gradient matrix is plausible to be identifies as an edge. For example, if the gradient value is higher than threshold value 255, assign it the pixel location to white, otherwise it will be black. The white pixels will show the edges of an image while other pixel locations are black. Once this process is performed for each pixel value of the grayscale image, the output image should present the edges that can be detected in the input image.

Technical Directions:

The spatial filter function takes in a grayscale image and a filter for neighborhood operation. The function must go through every pixel location of the input image to do the spatial linear operation. Checking the boundaries when using the filter operations must be considered as the required pixel intensity that is given from the input image may be not available for extraction therefore, we must use the zero-padding solution to mitigate execution error. Zero-padding method is instead of using any pixel intensity found in the input image, the function uses zero to represent the out of bounded pixel coordinate location. Each pixel location is mapped into a specific filter value for operations purposes. The function adds each operation and will use the result of the additions to represent the pixel in the input image in the double matrix vector. The output outputs a double with the same dimension as the input image.

The gradient magnitude takes an input of grayscale image and output a matrix of type double with the same dimensions as the input image. Using the matrix provided by the method of Sobel kernel in the textbook, the spatial filter function can compute the gradient x and gradient y of each individual pixel. Once we find the magnitude of the gradient vector by adding absolute of both gradient x and gradient y. The result is the gradient of the pixel coordinate. This will be used for edge detection.

The find edge function uses the gradient magnitude function to get the gradient vector to be used to apply the edge detection. The function acquires this by using threshold and compare the value of the gradient vector matrix and applying which pixel value will be either 0 or 255 (white or black). Using this method, the function will show which edge are present in an image. For my function, I used threshold 255 and it showed explementary success. If the gradient value is larger than the threshold, then the pixel intensity will be white therefore will be considered part of an edge. The function assigns the appropriate pixel value to the respective pixel location.

Results

Using different threshold for my function, higher threshold will increase the edge detected but it will diminish the detail such as lettering as that the rate of change between letters for example, are little due to spatial value being minimal. The built-in function in MATLAB called ‘edge’ using Sobel method outputted an image that shows the edge of an image, but it may be minimal in default compared to my implementation. I believe the built-in function would be the same as mine if the threshold used were below 155 using the implemented functions. The best threshold that output the best edge detection of the water tower image is 255. It seems to be the closest the lab instruction image but also a little better in the lettering of Merced on the front of the tank. Using the edge built-in function using canny method is much cleaner for the edges. The edges seem to be one line in width while my implementation is rather sporadic in conveying the edges. My implementation seem to work best for the lettering as that part is more legible in my implementation.

Figure 1: Original Image

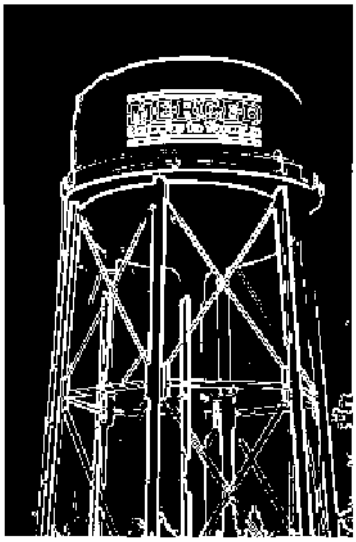

Figure 2: Edge Detected Image using 255 thresholds

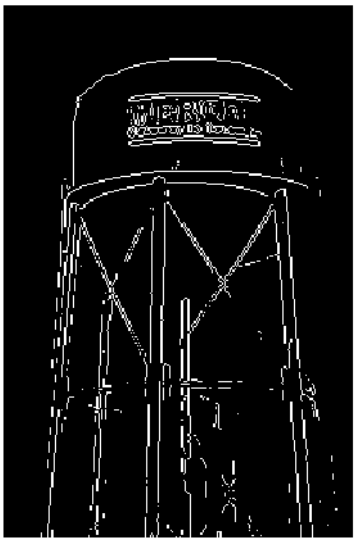

Figure 3: Built-In edge detection function image without using threshold using sobel

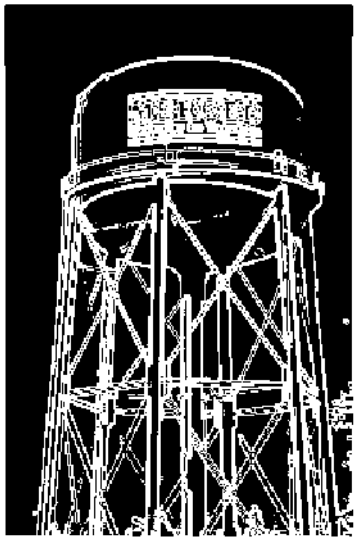

Figure 4: Using edge detection function with 155 thresholds

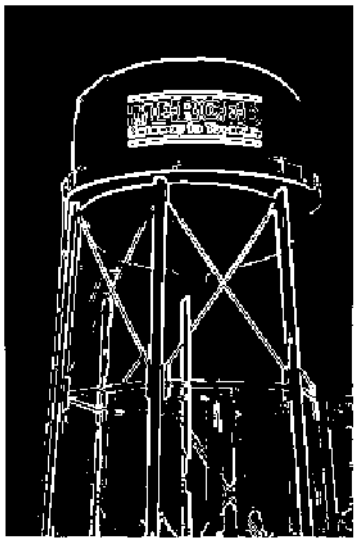

Figure 5: Using edge detection using 355 thresholds

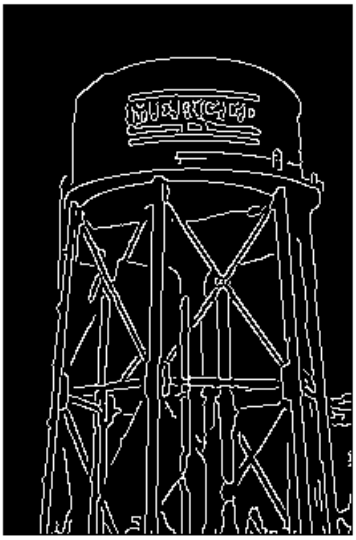

Figure 6: Built-In edge detection function image without using threshold using canny

```
fprintf("Starting\n");
in_image = imread("watertower.tif");
% filter = [0,0,0;0,1,0;0,0,0];
%out_image = spatial_filter(in_image,filter);
%out_image = gradient_magnitude(in_image);
out_image = find_edges(in_image,255);
imshow(in_image);
figure;
imshow(out_image);
figure
[sizeX,sizeY] = size(in_image);
test_build_in_sobel = edge(in_image,"sobel");
test_build_in_canny = edge(in_image,"canny");
imshow(test_build_in_sobel);
figure;
imshow(test_build_in_canny);
```

```
function out_image = find_edges(in_image,threshhold)
% This function identifies the edges by calling the gradient_magnitude
% function and using the threshold to identify the edges found in the image
%
% Syntax:
% out_image = find_edges(in_image,threshhold)
%
% Input:
% in_image - grayscale image
% threshold - threshhold to identify which pixel location is an edge
%
% Output:
% out_image - Uint8 image that has undergone edge detection
%
% History:
% Ralphilou Tatoy - 4/14/22 - Created function
% Ralphilou Tatoy - 4/14/22 - Completed function implementation
% Ralphilou Tatoy - 4/14/22 - Completed commenting
%Size of input image
[sizeX,sizeY] = size(in_image);

%Getting gradient matrix to use for edge detection
gradient = gradient_magnitude(in_image);

%Intializing output image
out_image = uint8(zeros(sizeX,sizeY));
for i = 1:sizeX
    for j = 1:sizeY
        %Using treshhold to identify the edges in the image
        if(gradient(i,j)>threshhold)
            out_image(i,j) = 255;
        else
            out_image(i,j) = 0;
        end
    end
end

end
```

```
function out_image= gradient_magnitude(in_image)
% This function creates a gradient of an image using the Sobel kernel
%
%
% Syntax:
% out_image= gradient_magnitude(in_image)
%
% Input:
% in_image - grayscale image
%
% Output:
% out_image - Type double matrix gradient of the gradient of the input
% image
%
% History:
% Ralphilou Tatoy - 4/14/22 - Created function
% Ralphilou Tatoy - 4/14/22 - Completed function implementation
% Ralphilou Tatoy - 4/14/22 - Completed commenting

[sizeX,sizeY] = size(in_image);
out_image = double(zeros(sizeX,sizeY));
sobel1 = [1,2,1;0,0,0;-1,-2,-1];
sobel2 = [-1,0,1;-2,0,2;-1,0,1];
gy = spatial_filter(in_image,sobel1);
gx = spatial_filter(in_image,sobel2);
for i = 1:sizeX
    for j = 1:sizeY
        out_image(i,j) = abs(gx(i,j))+abs(gy(i,j));
        %out_image(i,j) = sqrt((gx(i,j))^2+(gy(i,j)^2));
    end

end
end
```

```
function outImage = spatial_filter(inImage,sFilter)
% This function outputs takes in a grayscale image and performs the linear
% spatial process to output a neighborhood operations effect in the output
% image
%
%
% Syntax:
% outImage = spatial_filter(inImage, sFilter)
%
% Input:
% inImage - grayscale image
% sFilter - Filter
%
% Output:
% outImage - A double vector with the same dimensions as the input image and have
% undergone a neighborhood operations using a filter.
%
% History:
% Ralphilou Tatoy - 4/13/22 - Created function
% Ralphilou Tatoy - 4/13/22 - Implemented function
% Ralphilou Tatoy - 4/14/22 - Completed commenting

%Size of input image
[sizeX, sizeY] = size(inImage);

%Initializing output image with the same size as the input image
outImage = double(zeros(sizeX,sizeY));
%outImage = uint8(zeros(sizeX,sizeY));


for i = 1:sizeX
    for j = 1:sizeY
        %The calculation of spatial filtering process
        added = 0;
        for ks = -1:1
            for kt = -1:1
                %Initialize for spatial filter access
                s = 2;
                t = 2;

                %Boundary checking
                if(i+ks < 1 || i+ks > sizeX || j+kt < 1 || j+kt >sizeY)
                    picked = 0;
                else
                    picked = double(inImage(i+ks,j+kt));
                    %picked = inImage(i+ks,j+kt);
                end
%                 added = double(added + picked*sFilter(s+ks,t+kt));
                added = added + picked*sFilter(s+ks,t+kt);
```

```
            end
        end
        %Assign pixel intensity
        outImage(i,j) = added;
    end

end
return;
end
```