CSE 107 Lab 4

Edge Detection

Darren Correa

4/25/2022

Abstract

For this lab our goal is to generate an image that finds edge locations from a grayscale image. The actions to be taken include using spatial filtering and gathering the gradient magnitude of the image. Spatial filtering will apply a Sobel mask across a limited window that will move across the image. That filter generates the gradient magnitude. With the gradient magnitude and applying a scalar to magnify the intensity of the gradient will generate an image highlighting the edges from the original image. The tasks will consist of three functions that will call each other to perform each task.

Technical Discussion

Spatial filters apply some transformative property to a certain area of space. Derivative filters can sharpen images while low pass filters can blur them. It all depends on the filter itself and how applying it will affect the original image. As the name suggests, the filter is applied over a certain space. For instance, if we have a 3x3 filter that filter will be applied over a three-by-three window in the original image. Equation 1 shown below, can be explained simply as apply a spatial filter to a point and the pixels surrounding that point that span the size of the spatial filter.

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

With this equation we run into the issue of edge cases. When applying a spatial filter to the pixels along the edges of images, there are not pixel values before edges on the right or after on edges on the left. With MATLAB, this will cause an out of bound error. For this reason, image padding was essential in this lab. This is a process of padding an image matrix with zeroes, so we will not run into any boundary issues later.

To apply the filter, it was easiest to find the beginning row and column value and ending row and column value of the window. Rather than looping pixel to pixel, I move the window and apply the filter to the pixel in the center of the window. With the zero padding, our first pixel is a padded zero and not the first intensity value of the image. So, finding the center pixel according to the size of the window took care of boundary issues or applying the filter to a padded zero value. After the window moves around the entire image, we are then left with the filtered image. I tested the spatial filter function using a low pass filter. Compare Images 1 and 2 to see the results of only the spatial filter function.

The gradient magnitude function calls the spatial filter function but the filter being applied is a Sobel filter. A Sobel filter will measure the gradient intensity of each pixel which makes it a great filter for detecting edges. Applying this filter to our grayscale image with the spatial filter function generates an output image of the pixels with a high gradient intensity being shown.

The find edges function applies a scaler to the intensity values from our gradient magnitude image. The scalar or threshold will adjust what pixels will be seen as edges and which will not. Our results section will show how changing the value of the threshold will affect the final output image.

Discussion of Results

Comparing my results with the resulting image from the lab handout, I have found some differences. My resulting image (Image 4) has less edges detected. The most noticeable difference being the legs of the water tower. I am still able to discern there is a leg there, but the outline of the leg is very faint to nothing at all, while the professor's image has a very distinct edge. My threshold value does not change the outline of that leg. Increasing or decreasing the value will increase of decrease the intensity of which other edges are detected. When my threshold is zero, no edges have been detected. When it is 500 you can clearly see the edges show a water tower. The "Merced" text is also visible and readable in my resulting image so I was satisfied with that. Comparing my results with the built in Canny edge detection, while the built-in function provided a traced outline of the water tower showing all the edges, it looks rough. The generated image shows more detail and smoother edges. This leads me to believe the function I wrote is more precise when it comes to the position of an edge when one is located, and Canny edge detection is more precise at finding an edge. Images 4 and 5 show the two processed images.

Results

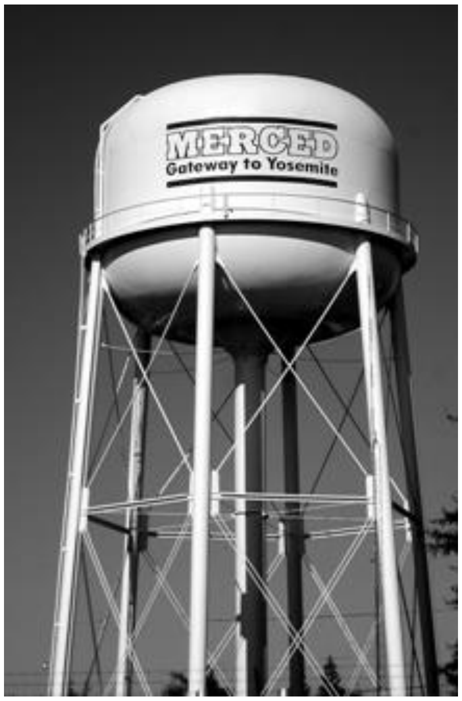

Image 1: Original Water tower Image

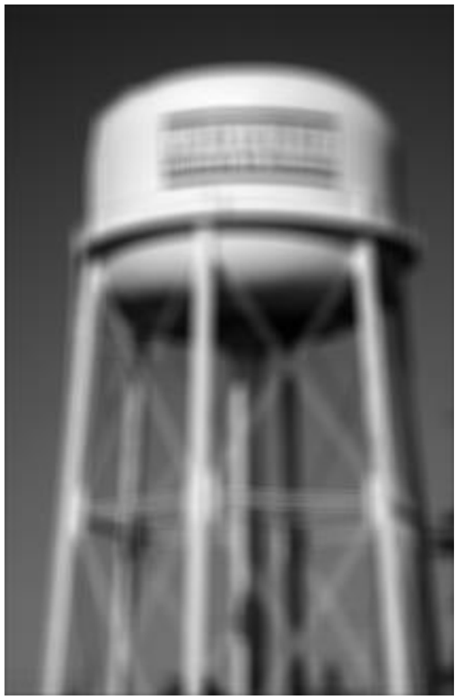

Image 2: Low Pass Filtered Image

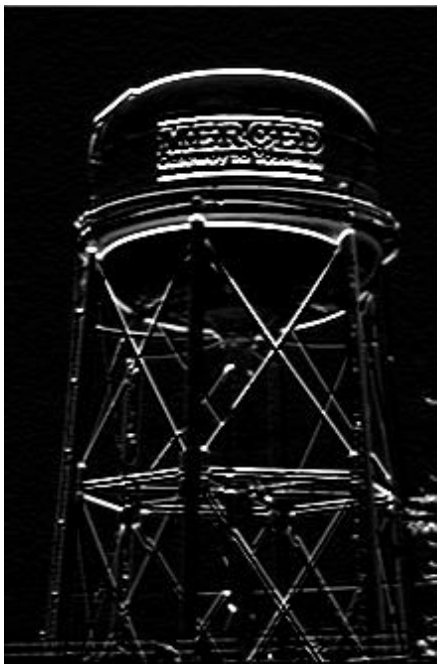

Image 3: Gradient Magnitude Image

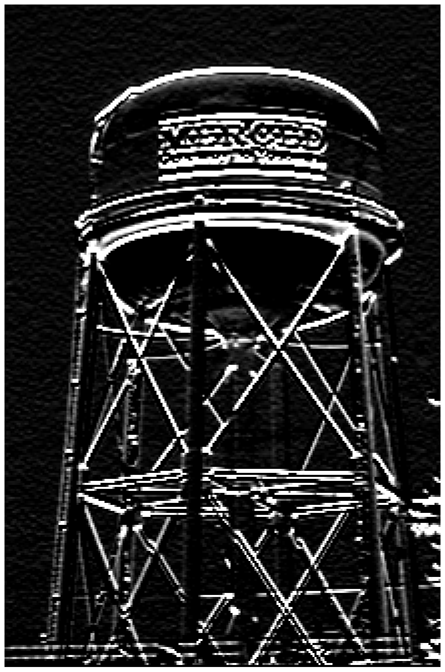

Image 4: Find_Edges Image. Threshold = 800

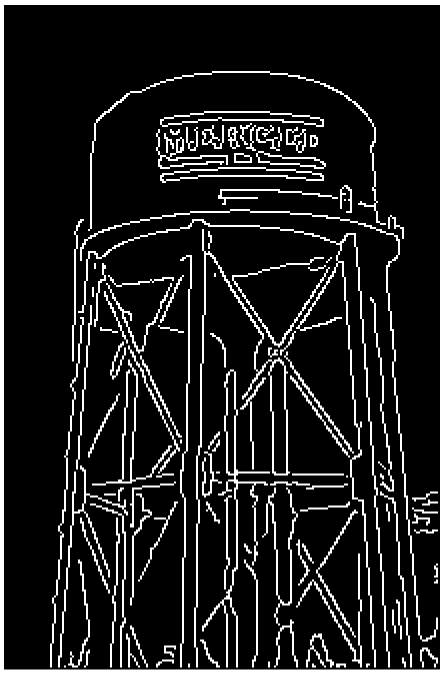

Image 5: Canny Edge Detection Image

Code

Spatial_filter.m

```
function output_img = spatial_filter(img, filter)
% spatial_filter    Applies a spatial filter to an image matrix by
%                   examening the image and filter size, then creating
%                   a window to apply the filter over a certain area
%                   of pixels in the image.
%
% Syntax:
%   output_img = spatial_filter(img, filter)
%
% Input:
%   img = Image matrix of type double taken from grayscale image.
%
%   filter = Filter to be used. Must be a matrix.
%
% Output:
%   output_img = Image matrix generated after applying spatial filter
%
% History:
%   D. Correa   Created                 4/20/2022
%   D. Correa   Tested and completed    4/25/2022



% Pad image matrix with zeros for boundry issues
padded_img = padarray(img, [1,1], 0, 'both');

% Get dimensions from filter and image to be used to compute position of
% filter window applied onto image.
filter_size = size(filter);
img_size = size(padded_img);

% Define size of output image
output_img = zeros(img_size(1) - (filter_size(2) - 1), img_size(2) -
filter_size(2) - 1);

% Define filter window size
start_row = (filter_size(2) + 1) / 2;
start_col = (filter_size(2) + 1) / 2;
end_row = img_size(1) - ( (filter_size(2)-1) / 2);
end_col = img_size(2) - ( (filter_size(2)-1) / 2);

% Apply filter window to padded image
for i = start_row : end_row
    for j = start_col : end_col

        % Set window to be the center pixel, [i,j], and its surrounding
        % pixels. Output image is the sum of applying the filter to each
        % pixel in the window.
        window = padded_img(i - ((filter_size(2)-1)/2): i + ((filter_size(2)-
1)/2), j - ((filter_size(2)-1)/2): j + ((filter_size(2)-1)/2));
        output_img(i - ((filter_size(2)-1)/2), j - ((filter_size(2)-1)/2)) =
sum( window .* filter, 'all');

    end
end
end
```

gradient_magnitude.m

```
function gradient_output = gradient_magnitude(img)
% gradient_magnitude    Takes an image matrix of type double and calls
%                       spatial_filter function to apply Sobel filter.
%
% Syntax:
%   gradient_output = gradient_magnitude(img)
%
% Input:
%   img = Image matrix of type double from grayscale image.
%
% Output:
%   gradient_output = Gradient output image matrix after applying Sobel
%                     filter.
%
% History:
%   D. Correa   Created, tested and completed   4/25/2022

%Sobel mask from figure 10.14 in the textbook
sobel_filter = [-1,-2,-1; 0,0,0; 1,2,1];

%Call spatial filter to generate gradient magnitude image using sobel mask
%as filter.
gradient_output = spatial_filter(img, sobel_filter);

end
```

find_edges.m

```
function edge_img = find_edges(img, scalar)
% find_edges    Generate image matrix showing edges detected from gradient
%               image enhanced by threshold scalar.
%
% Syntax:
%   edge_img = find_edges(img, scalar)
%
% Input:
%   img = Image matrix of type double taken from grayscale image.
%
%   scalar = Threshold scalar value to determine sensitivity of which edges
%            are detected.
%
% Output:
%   edge_image = Output image showing detected edges
%
% History:
%   D. Correa   Created, tested and completed.  4/25/2022
%

%Get gradient image from gradient magnitude
gradient_output = gradient_magnitude(img);

% Apply threshold to each value in gradient image
edge_img = gradient_output .* scalar;

% Convert double edge_img to uint8.
edge_img = uint8(edge_img);

end
```

Script

```
%Script to call and get all images from all functions

%Get image matrix frim image
img = imread('watertower.tif');

%Convert uint8 matrix to double
img = im2double(img);

%Set filter
filter = ones(9)/81;

%Call spatial filter function
output_img = spatial_filter(img, filter);

%Show  and save image from spatial filter function
imshow(output_img);
imwrite(output_img, 'LowPassFilterImg.png')

%Call gradient magnitude funtion
gradient_output = gradient_magnitude(img);

%Show  and save image from gradient magnitude function
imshow(gradient_output);
imwrite(gradient_output, 'GradientImg.png');

%Set threshold for find_edges and call function
scalar = 800;
edge_img = find_edges(img, scalar);

%Show and save edge images
imshow(edge_img);
imwrite(edge_img,'watertower_edges.png');

% Use built in Canny edge detection to compare with written function
canny_img = edge(img, 'Canny');
imshow(canny_img);
imwrite(canny_img, 'CannyImg.png');
```