Nick Neisen October 27, 2015
CENG 414 - Introduction to Computer Vision
Homework 7

Introduction

The purpose of this lab was to write our own fspecial and gradient functions and use them for edge detection. To test our code we were given a Matlab example using the standard functions that we could replace with ours. The fspecial function creates difference sized kernels that follow common curves in 3D such as Gaussian. The gradient takes an image and finds the gradient of that image.

Procedure

To test the edge detection of our program we were asked to find an image to use for the lab. My original image can be seen in **Figure 1**. The figures following compare the output of the Matlab functions on the right to the output created using my functions on the left. In the first set of images we can make out the edges but there is some blurring. By using both of the functions created together we can see that there are more edges and they are more distinct.

The first step of the lab was to create the gradient function. This function allows the user to pass in an image and the distance between each step. The gradient function creates two arrays the same size as the image for holding the results. It then iterates over the image and finds the derivative with respect to x and y. These values values are placed into their respective array at the corresponding position.

Three techniques were used to calculate the derivative depending on the current position in the array. The majority of the values can be calculated using the central difference. This can be expressed as:

$$\delta_h[f](x) = f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)$$

This is the most accurate way to find the difference but it runs in to some difficulties. When finding the first and final values of the array this would cause us to go out of bounds. For this reason we also need to use the forward difference:

$$\Delta_h[f](x) = f(x + h) - f(x)$$

and the backward difference:

$$\Delta_h[f](x) = f(x) - f(x - h)$$

With these equations are are able to successfully find the gradient of the image. The results of the horizontal and vertical gradients can be seen in **Figures 2-7**.

The second function we need to write is one to create different types of kernels for filtering. These filters are used to generate difference effects when convolving with an image. Our function must be able to create different types of kernels depending on the first argument passed into it. The types of kernels required are the Gaussian, Prewitt, Sobel, and Roberts. The Gaussian filter can be created using:

$$G(x,y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The result of this can be seen in **Figure 12 & 13**. The remaining filters are always used with specific matrix sizes. Each of these can be seen below:

$$G(x,y) = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Prewitt

$$G(x,y) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel

$$G_x(x,y) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$G_y(x,y) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Roberts

Conclusion

Having these functions available in Matlab makes it easy to get going and start coding, however, taking the time to write these ourselves helps us to gain a better understanding of what these are doing and the effect they have on our images. Finding the gradients of an image gives us the edges of the image because the change in pixel magnitude is greatest at these points. By convolving the images with a filter before finding the gradient, we can make the lines of the edges more distinct.

Figures

Figure 1: Original image

Figure 2: My X Gradient

Figure 3: Original X Gradient

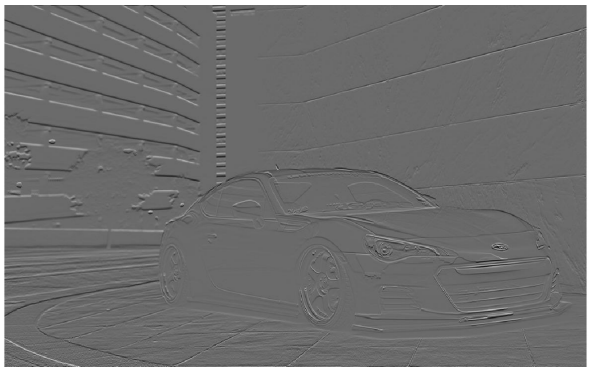

Figure 4: My Y Gradient

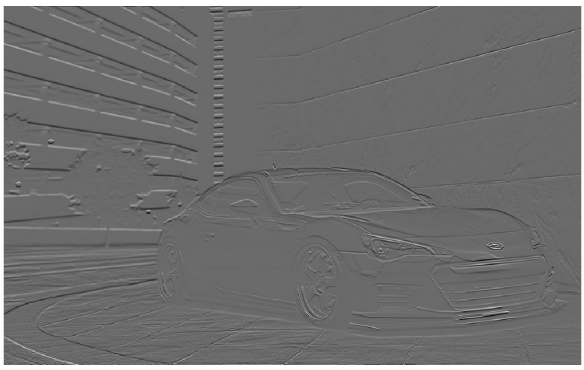

Figure 5: Original Y Gradient

Figure 6: My Magnitude Gradient

Figure 7: Original Magnitude Gradient

Figure 8: My Filtered X Edges

Figure 9: Original Filtered X Edges

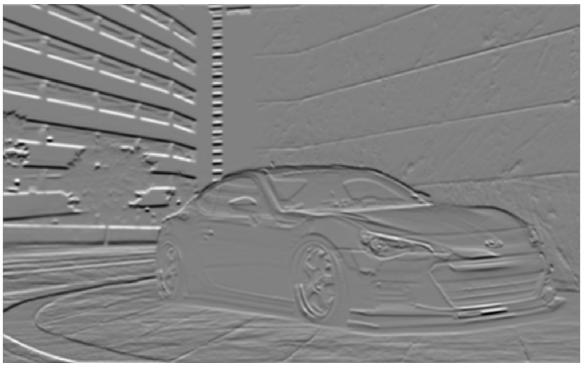

Figure 10: My Filtered Y Edges

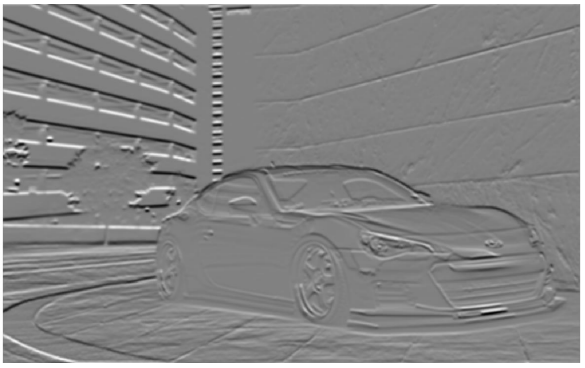

Figure 11: Original Filtered Y Edges

Figure 12: My Filtered Edges

Figure 13: Original Filtered Edges

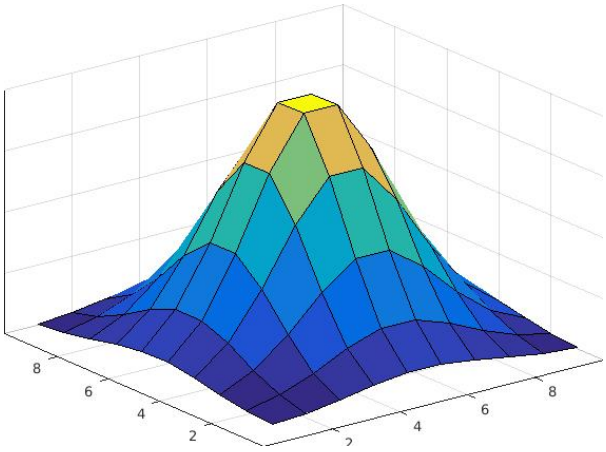

Figure 14: My Gaussian Filter

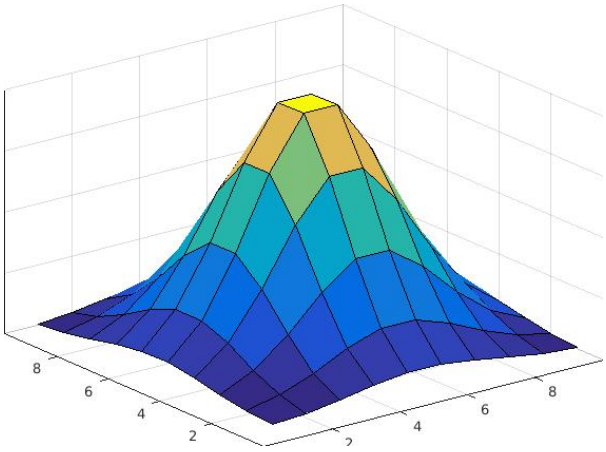

Figure 15: Original Gaussian Filter

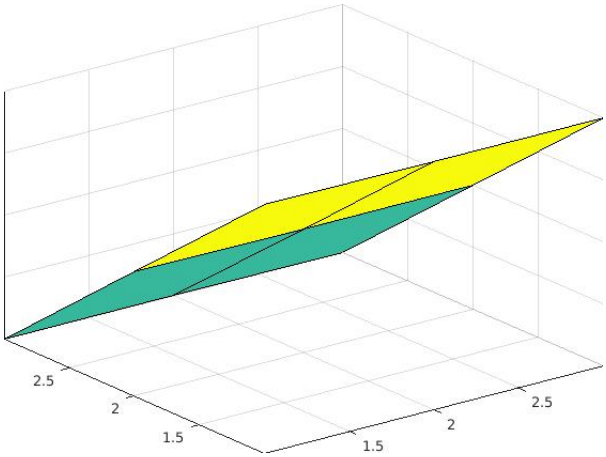

Figure 16: My Prewitt Filter

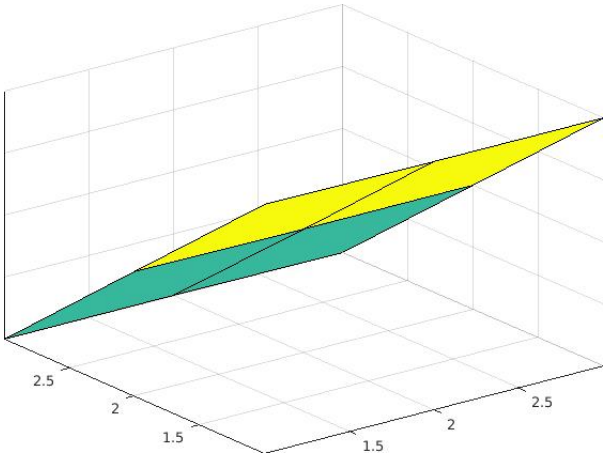

Figure 17: Original Prewitt Filter

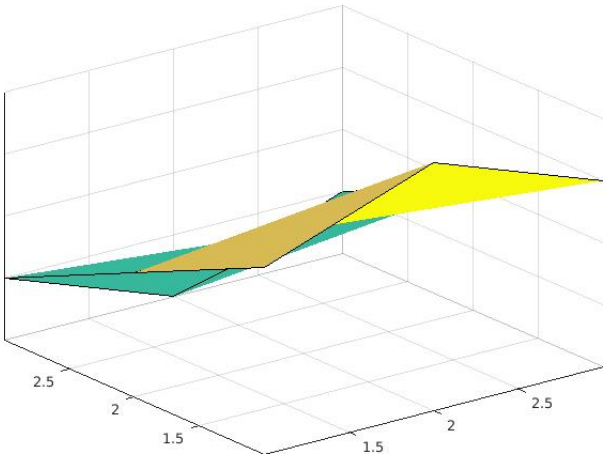

Figure 18: My Sobel Filter

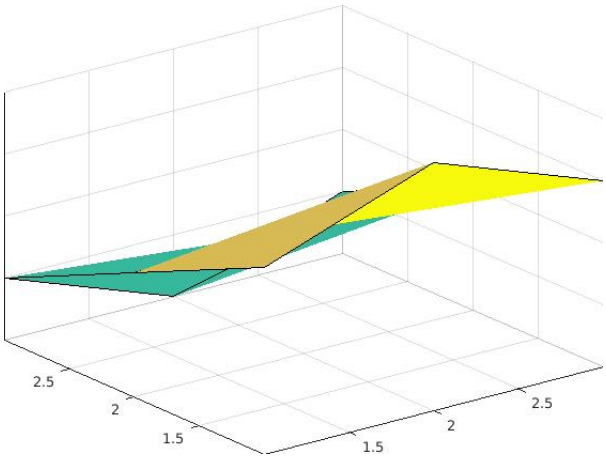

Figure 19: Original Sobel Filter

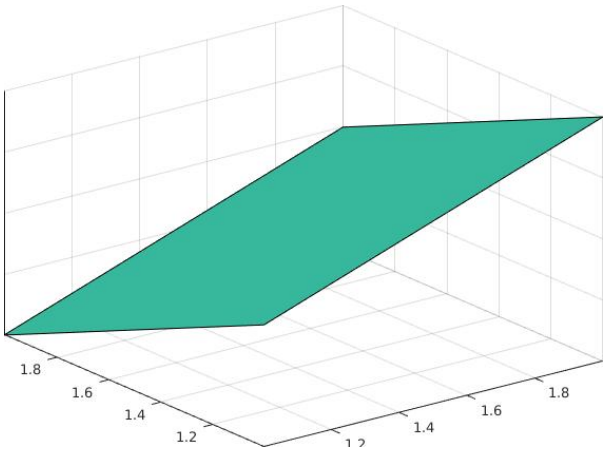

Figure 20: Roberts horizontal filter

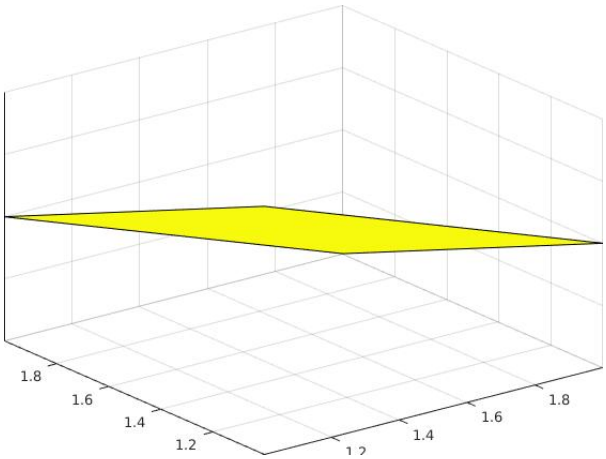

Figure 21: Roberts vertical filter

Appendix

Main Matlab

```
clc;
clear all;
close all;

% Load in the image
I = imread('subaru-brz.jpg');
% Convert it to BW
I = rgb2gray(I);
% Convert to doubles
I = im2double(I);

%% Creating our own gradient function
% Find the gradient of the image
[Iy,Ix] = gradient(I);
[myIx,myIy] = myGradient(I);

% Combine the horizontal and vertical edges
Iedge = sqrt((Ix.^2) + (Iy.^2));
myIedge = sqrt((myIx.^2) + (myIy.^2));

% Show the original edges
figure();
imshow(Ix,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(Iy,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(Iedge,[]);
set(gca,'position',[0 0 1 1],'units','normalized');

% Show my edges
figure();
imshow(myIx,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(myIy,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(myIedge,[]);
set(gca,'position',[0 0 1 1],'units','normalized');

%% Creating our own fspecial function
% Make a Gaussian kernel
H = fspecial('gaussian',10);
myH = myFSpecial('gaussian',10);

% Find the gradients of the Gaussian
[Hx, Hy] = myGradient(H);
[myHx, myHy] = myGradient(myH);
```

```
% Conolve the kernel with the image to find edges
Ix = myConvolution(I, Hx);
Iy = myConvolution(I, Hy);
myIx = myConvolution(I, myHx);
myIy = myConvolution(I, myHy);

% Combine the horizontal and vertical edges
Iedge = sqrt((Ix.^2) + (Iy.^2));
myIedge = sqrt((myIx.^2) + (myIy.^2));

figure();
imshow(Ix,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(Iy,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(Iedge,[]);
set(gca,'position',[0 0 1 1],'units','normalized');

figure();
imshow(myIx,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(myIy,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
figure();
imshow(myIedge,[]);
set(gca,'position',[0 0 1 1],'units','normalized');
```

My gradient Matlab

```
function [gradientY,gradientX] = myGradient(image, distance)
    % Default distance of 1
    if nargin < 2
        distance = 1;
    end

    % Get the dimensions of the matrix
    [height,width] = size(image);

    % Create two 0 matrices for the derivatives
    gradientX = zeros([height,width]);
    gradientY = zeros([height,width]);

    % Go through each pixel in the image
    for x = 1:width
        for y = 1:height

            % Find the derivative with respect to X
            if x == 1
                % Forward difference on the first value
                gradientX(y,x) = (image(y,x+1)-image(y,x))/distance;
            elseif x == width
                % Backwards difference on the final value
                gradientX(y,x) = (image(y,x)-image(y,x-1))/distance;
            else
                % Central difference on the other values
                gradientX(y,x) = 0.5*(image(y,x+1)-image(y,x-1))/distance;
            end

            % Find the derivate with respect to Y
            if y == 1
                % Forward difference on the first value
                gradientY(y,x) = (image(y+1,x)-image(y,x))/distance;
            elseif y == height
                % Backwards difference on the final value
                gradientY(y,x) = (image(y,x)-image(y-1,x))/distance;
            else
                % Central difference on the other values
                gradientY(y,x) = 0.5*(image(y+1,x)-image(y-1,x))/distance;
            end
        end
    end
end
```

My FSpecial Matlab

```
function kernel = myFSpecial(type,size,mean)

    % Check if the user just wants a square kernel
    if nargin > 1 && length(size) == 1
        size = [size,size];
        mean = 0.5;
    end

    if strcmp(type,'average')
        kernel = (1/(size(1)*size(2))).*ones(size(1),size(2));
    end

    if strcmp(type,'disk')
        % Initialize our kernel to all zeros
        kernel = zeros((size(1)*2)+1,(size(1)*2)+1);

        % Cut the sizes in half
        size = length(kernel(:,1))/2;

        % Create a mesh grid
        for x = -size:size-1
            for y = -size:size-1
                kernel(y+size+1,x+size+1) = 1/abs(x*y);
            end
        end
    end

    if strcmp(type,'gaussian')
        % Initialize our kernel to all zeros
        kernel = zeros(size(1),size(2),2);

        % Cut the sizes in half
        xSize = (size(1)-1)/2;
        ySize = (size(2)-1)/2;

        % Create a mesh grid
        for x = -xSize:xSize
            for y = -ySize:ySize
                kernel(y+ySize+1,x+xSize+1,1) = x;
                kernel(y+ySize+1,x+xSize+1,2) = y;
            end
        end

        % Find Hg
        hg = exp(-((kernel(:,:,1).^2)+(kernel(:,:,2).^2))/(2*mean^2));

        % Get the final kernel values
        kernel = hg./sum(hg(:));
    end

    if strcmp(type,'laplacian')
        kernel = ones(3,3);
```

```
    for x = 1:3
        for y = 1:3
            if x==2 && y==2
                kernel(y,x) = -1;
            elseif x==2 || y==2
                kernel(y,x) = (1-size(1))/4;
            elseif x==1 || x==3 || y==1 || y==3
                kernel(y,x) = size(1)/4;
            end
        end
    end

    % Once last multiplier
    kernel = (4/(size(1)+1)).*kernel;
end

if strcmp(type,'log')
    % Initialize our kernel to all zeros
    kernel = zeros(size(1),size(2),2);

    % Cut the sizes in half
    xSize = (size(1)-1)/2;
    ySize = (size(2)-1)/2;

    % Create a mesh grid
    for x = -xSize:xSize
        for y = -ySize:ySize
            kernel(y+ySize+1,x+xSize+1,1) = x;
            kernel(y+ySize+1,x+xSize+1,2) = y;
        end
    end

    % Find Hg
    hg = exp(-((kernel(:,:,1).^2)+(kernel(:,:,2).^2))/(2*mean^2));

    % Get the final kernel values
    kernel = hg./sum(hg(:));

    for x = 1:size(1)
        for y = 1:size(2)
            k = ((x^2)+(y^2)-(2*(mean^2)))/(2*pi*(mean^6));
            kernel(y,x) = k*kernel(y,x);
        end
    end
end

if strcmp(type,'motion')
    % Check if the size is odd
    if mod(size(1),2) == 0
        for x = 1:size(1)+1
            if x==1 || x==size(1)+1
                kernel(x) = 1/(2*size(1));
            else
                kernel(x) = 1/size(1);
            end
        end
    else
```

```
            for x = 1:size(1)
                kernel(x) = 1/size(1);
            end
        end
    end

    if strcmp(type,'prewitt')
        kernel = [1 1 1;0 0 0;-1 -1 -1];
    end

    if strcmp(type,'sobel')
        kernel = [1 2 1;0 0 0;-1 -2 -1];
    end

    if strcmp(type,'roberts')
        kernel(:,:,1) = [0 1;-1 0];
        kernel(:,:,2) = [1 0;0 -1];
    end
end
```