CSC 514: Computer Vision: Homework #7

Due on Tuesday, October 27, 2015

Hoover 11:00am

Julian A. Brackins

Problem 1

Using your functions developed in Homework 6 to computer convolution and correlation of images with a specified kernel, in this homework you will develop a function to compute the edges of an input image. Your function will take as input an image and a kernel and output an edge map of the input image. You may want to create a second function to return the kernel based on use input (i.e., Roberts, Sobel, Prewet, and Gaussian). In addition, you will want your output image to be the same size as the input image, therefore, you'll need to deal with the edges accordingly and document in your write-up.

Preface

In the previous program, we examined how using correlation could be utilized to find similarities between an image and a kernel. In this assignment, convolution is used to find differences between images and a kernel. These differences in images can be used to generate what is known as an edge map. Edges arise from various reasons in images. Edges can be seen in changes of reflectance or texture, changes in surface orientation, or even in cast shadows. In a perfect world, edges are used to determine depth discontinuity and identify object boundaries.

Equations

This assignment once again utilizes the Convolution equation.

Convolution:

$$G\Big[i,j\Big] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} H\Big[u,v\Big]F\Big[i-u,j-v\Big]$$

Where F is the image and H is the kernel.
Convolution can actually be performed using Correlation. Flip the kernel in both directions (or rotate the kernel 180°), then apply correlation.

In this assignment, finite difference kernels are also utilized. The following family of gradient kernels, also known as a mask, will be used for this program:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ 2-p & 0 & p-2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & 2-p & -1 \\ 0 & 0 & 0 \\ 1 & p-2 & 1 \end{bmatrix}$$

Where p varies based on the type of kernel being used:

	Prewitt	Sobel	Isotropic
p	3	4	$2+\sqrt{2}$

Finally, in order to generate the edge map, the gradient magnitude formula will be used.

Problem 1 continued on next page...

$$\| \bigtriangledown f \| = \sqrt{(\frac{\delta f}{\delta x})^2 + (\frac{\delta f}{\delta y})^2}$$

Where $\frac{\delta f}{\delta x})$ is the x derivative map generated from the G_x gradient, and $\frac{\delta f}{\delta y})$ is the y derivative map generated from the G_y gradient.

Process

The following is the process used to find the edges.

- Read in the image.
- Convert image to grayscale double.
- Determine kernel used for edge detection (Roberts, Sobel, Prewitt, or Gaussian)
- Calculate x derivative and y derivative from kernels using Convolution:
  - Rotate kernel by 180 degrees.
  - Pad image edges to retain appropriate dimensions (see Image Padding subsection)
  - Perform Correlation
- Generate Edge Map by using gradient magnitude.

Image Padding

Image padding is required for applying the kernel to the input image. In order to do this, we'll use the functions *vertcat*() and *horzcat*(). The *vertcat*() method will put horizontal bars on the top and bottom of the image, and *horzcat*() will put vertical bars on the left and the right of the image. The kernel size is used to determine the size of the horizontal and vertical bars that are added to the image. Refer to Line 15 in Listing 5 (the Convolution function) for a representation of how *horzcat*() and *vertcat*() are utilized to add the padding.

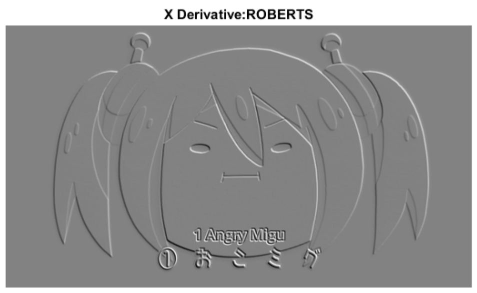
Matlab

The Matlab Code for the program attached after the conclusion of this Homework Document.

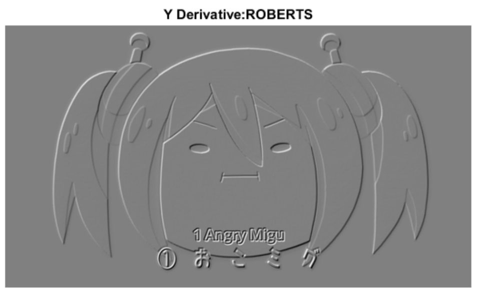
Conclusion

This assignment covers the utility of convolution in detecting edges in images. The real-world applications of this skill are wide spread. In computer vision, the ability to discern edges in a scene can allow a computer to have a better spacial understanding of a given area. The different kernels used to create the edge maps are shown on the following pages, as well as the x and y derivative images.

Problem 1 continued on next page. . .

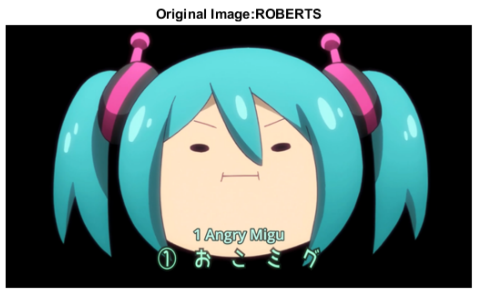


(a) X Derivative

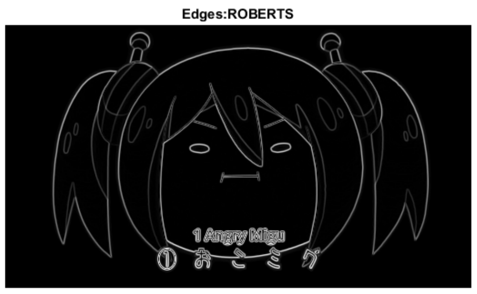


(b) Y Derivative

Figure 1: Derivative Maps Using Roberts



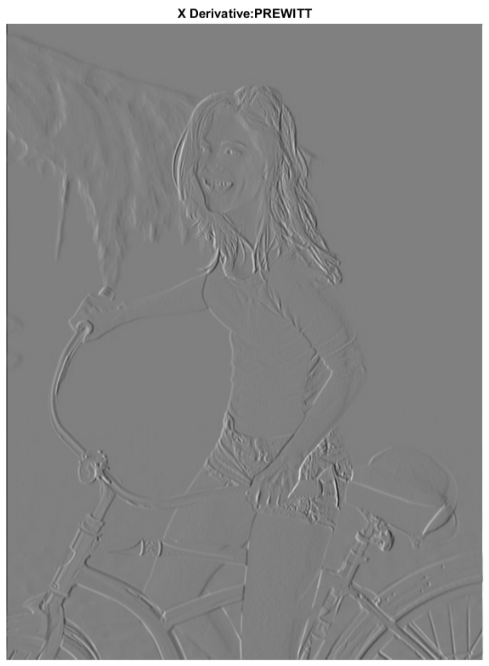
(a) Original Image



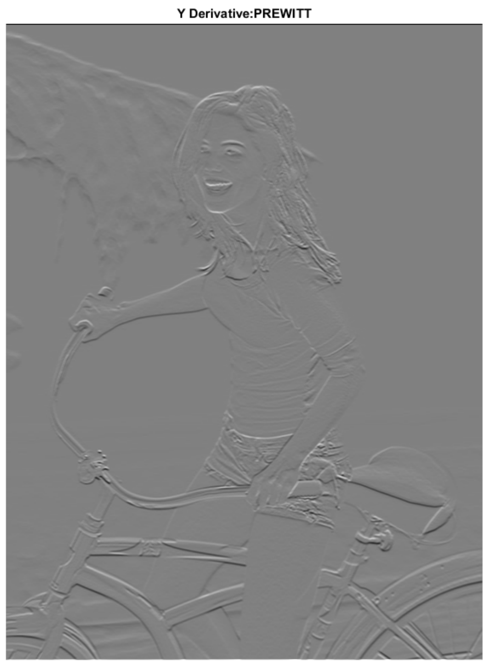
(b) Edge Map

Figure 2: Edge Detection Using Roberts

Problem 1 continued on next page. . .

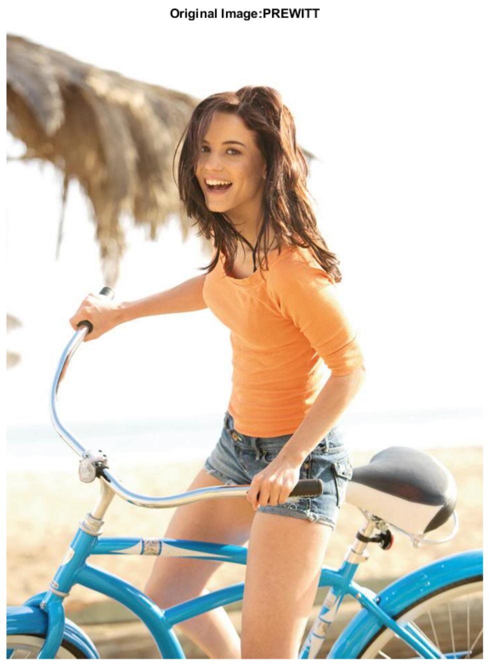


(a) X Derivative

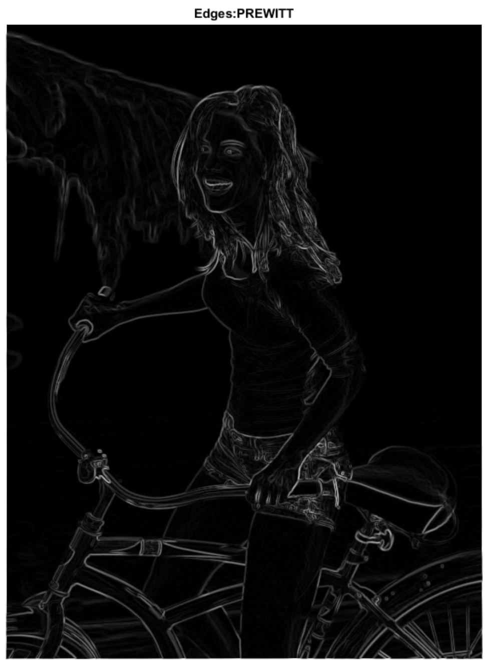


(b) Y Derivative

Figure 3: Derivative Maps Using Prewitt



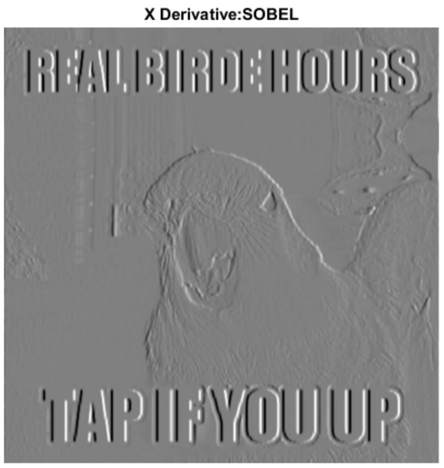
(a) Original Image



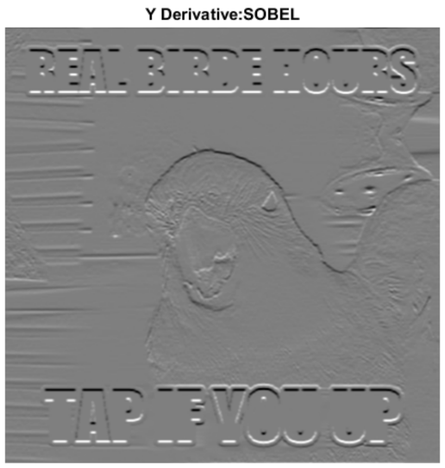
(b) Edge Map

Figure 4: Edge Detection Using Prewitt

Problem 1 continued on next page. . .

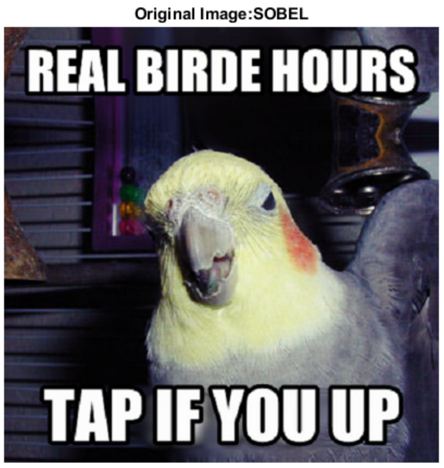


(a) X Derivative

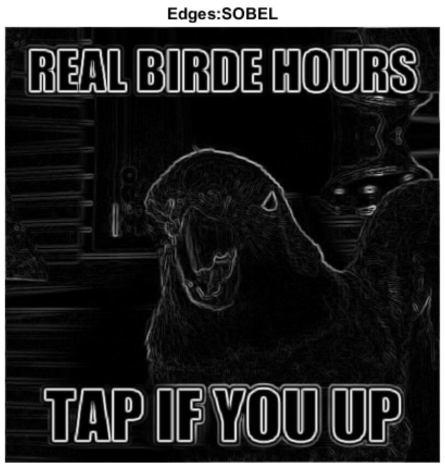


(b) Y Derivative

Figure 5: Derivative Maps Using Sobel



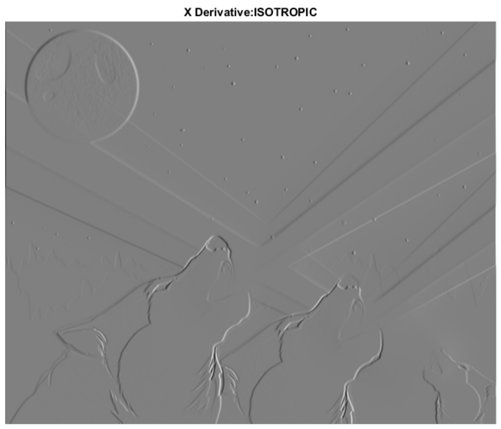
(a) Original Image



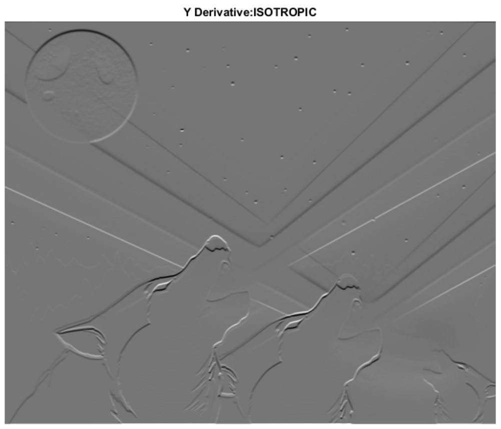
(b) Edge Map

Figure 6: Edge Detection Using Sobel

Problem 1 continued on next page. . .

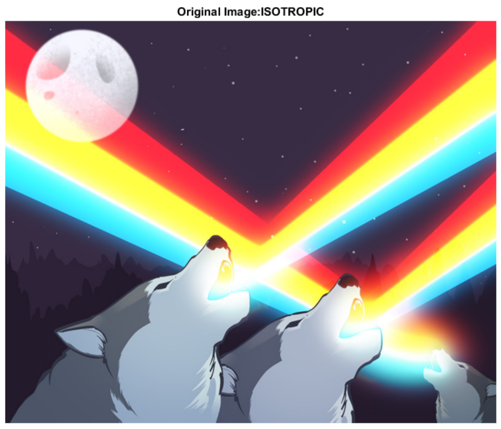


(a) X Derivative

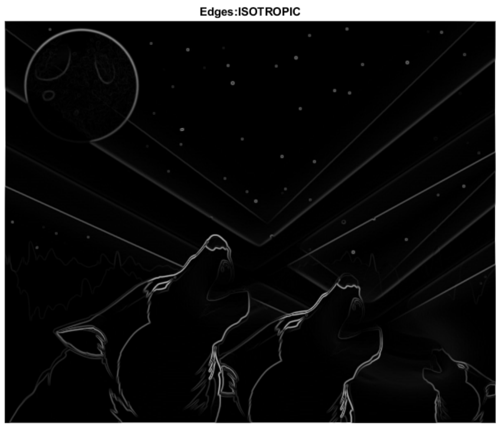


(b) Y Derivative

Figure 7: Derivative Maps Using Isotropic



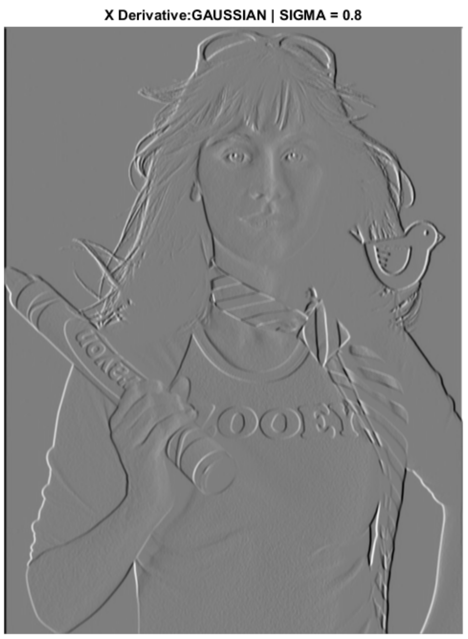
(a) Original Image

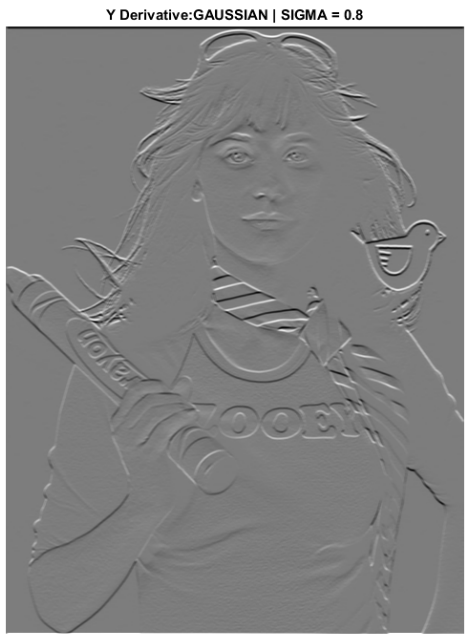


(b) Edge Map

Figure 8: Edge Detection Using Isotropic

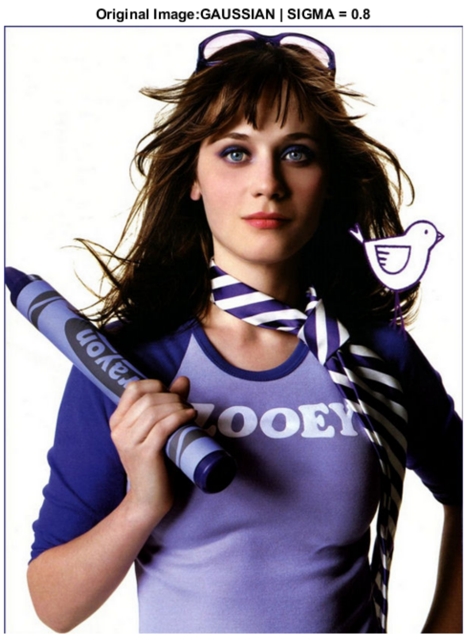
Problem 1 continued on next page...

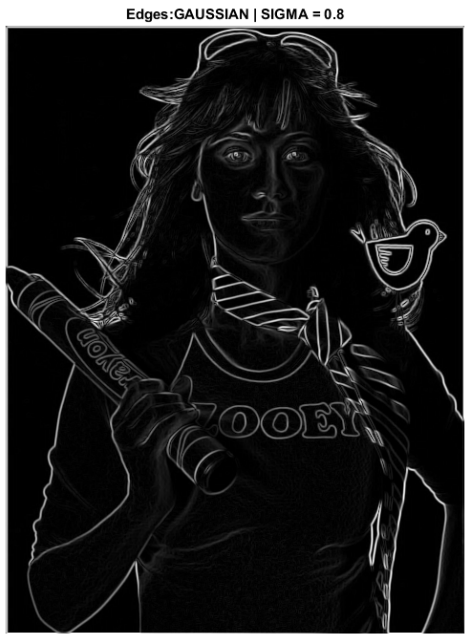




(a) X Derivative (b) Y Derivative

Figure 9: Derivative Maps Using Gaussian





(a) Original Image (b) Edge Map

Figure 10: Edge Detection Using Gaussian

Problem 1 continued on next page...

Listing 1: Main Script

```
function edge()

    clc
    close all

    %Image Files
    f1 = 'angry.png';
    f2 = 'bike.jpg';
    f3 = 'real.png';
    f4 = 'wolf.jpg';
    f5 = 'zooey.jpg';

    %KERNEL methods
    rob = 'ROBERTS';
    pre = 'PREWITT';
    sob = 'SOBEL';
    iso = 'ISOTROPIC';
    gau = 'GAUSSIAN';

    %Convert each image to grayscale doubles
    %See prepimage() function below
    [ img01, img01g ] = prepimage( f1 );
    [ img02, img02g ] = prepimage( f2 );
    [ img03, img03g ] = prepimage( f3 );
    [ img04, img04g ] = prepimage( f4 );
    [ img05, img05g ] = prepimage( f5 );

    %Generate Kernels
    %See get_kernel() function below
    [ k01x, k01y ] = get_kernel(rob);
    [ k02x, k02y ] = get_kernel(pre);
    [ k03x, k03y ] = get_kernel(sob);
    [ k04x, k04y ] = get_kernel(iso);
    [ k05x, k05y ] = get_kernel(gau, 0.8 );

    %Generate Images (x derivative, y derivative, edge map)
    %See get_edge() function below
    [ map01x, map01y, edge01 ] = get_edge( img01g, k01x, k01y );
    [ map02x, map02y, edge02 ] = get_edge( img02g, k02x, k02y );
    [ map03x, map03y, edge03 ] = get_edge( img03g, k03x, k03y );
    [ map04x, map04y, edge04 ] = get_edge( img04g, k04x, k04y );
    [ map05x, map05y, edge05 ] = get_edge( img05g, k05x, k05y );

    %Display Images (original image, x derivative, y derivative, edge map)
    %see genfigure() function below
    genfigure( img01, map01x, map01y, edge01, rob);
    genfigure( img02, map02x, map02y, edge02, pre);
    genfigure( img03, map03x, map03y, edge03, sob);
    genfigure( img04, map04x, map04y, edge04, iso);
    genfigure( img05, map05x, map05y, edge05, strcat(gau, ' | SIGMA = 0.8'));

end
```

Listing 2: Preparing Images

```
function [ img, img_g ] = prepimage( file )
    %Read in image, scale it down a bit, return colored img and grayscale
    img = imread( file );
    img = imresize( img, 0.8 );
    img_g = rgb2gray( img );
    img_g = double( img_g );
end
```

Listing 3: Generate Figures

```

function genfigure( i, ix, iy, iedge, descr )

    %Generate Images (original image, x derivative, y derivative, edge map)
    set(gca,'LooseInset',get(gca,'TightInset'))

    %Show images with appropriate descriptions
    figure, imshow(i,[]);
    title(strcat('Original Image: ', descr));

    figure, imshow(ix,[]);
    title(strcat('X Derivative: ', descr));

    figure, imshow(iy,[]);
    title(strcat('Y Derivative: ', descr));

    figure, imshow(iedge,[]);
    title(strcat('Edges: ', descr));

end
```

Listing 4: Generate Edge Map

```
function [ derX, derY, edge ] = get_edge( image, kernX, kernY )
    %Generate X Derivative Map through convolution
    derX = convolution( image, kernX );
    %Generate Y Derivative Map through convolution
    derY = convolution( image, kernY );
    %Generate Edge Map through gradient magnitude
    edge = sqrt(derX.^2 + derY.^2);
end
```

Problem 1 continued on next page. . .

Listing 5: Convolution

```
function [ G ] = convolution( scene, kernel )
    %%For Convolution, just Flip the filter in both directions
    %%Then apply Correlation as before...
    %%BTW rot90(x,2) is faster than flipud() + fliplr() which also works...
    kernel = rot90(kernel,2);

    %%Find the dimensions for the scene and kernel
    [kH, kW] = size(kernel);
    [sH, sW] = size(scene);

    %Get Half-Dimensions of Kernel to determine padding on each side
    kH2 = floor(kH / 2);
    kW2 = floor(kW / 2);

    %GENERATE HORIZONTAL PADDING
    pad = zeros(kH2,sW);
    %PAD TOP OF IMAGE
    scene = vertcat(pad,scene);
    %PAD BOTTOM OF IMAGE
    scene = vertcat(scene,pad);

    %GENERATE VERTICAL PADDING
    pad = zeros(sH + 2 * kH2,kW2);
    %PAD LEFT SIDE OF IMAGE
    scene = horzcat(pad,scene);
    %PAD RIGHT SIDE OF IMAGE
    scene = horzcat(scene,pad);


    %%set F, G, H matrices so that they match what's in the book.
    G = double(zeros(sH, sW));
    F = scene;
    H = kernel;

    %Calculate Correlation. Fixed so the summation appears closer to slides
    for i = 1:sH
        for j = 1:sW
        total = 0;
            for u = 1:kH
                for v = 1:kW
                    total = total + H(u,v) * F(i + u - 1,j + v - 1);
                end
            end
            G(i,j) = total / (sH * sW * kH * kW);
        end
    end

end
```

Problem 1 continued on next page...

Listing 6: Generate Kernel

```
function [ Gx, Gy ] = get_kernel( kType, kSigma )
    %Ensure first arg is always caps
    kType = upper(kType);
    %Only Gaussian has kSigma
    if nargin < 2
       kSigma = 0;
    end
    if strcmp( kType, 'ROBERTS')
        %ROBERTS:
        Gx = [   0  1   ;
                -1  0 ];
        Gy = [   1  0   ;
                 0 -1 ];
    else
        %HANDLE THE OTHERS
        if strcmp( kType, 'PREWITT')
            %PREWITT:
            p = 3;
        elseif strcmp( kType, 'SOBEL')
            %SOBEL:
            p = 4;
        elseif strcmp( kType, 'ISOTROPIC')
            %ISOTROPIC:
            p = 2 + sqrt(2);
        elseif strcmp( kType, 'GAUSSIAN')
            %GAUSSIAN:
            p =  2 * pi * ( kSigma * kSigma ) ;
        end

        if strcmp( kType, 'GAUSSIAN')
        %Gradient Kernel Family used for Gaussian:
        GradKernX = [   -1    0    1   ;
                        -1    0    1   ;
                        -1    0    1  ];
        GradKernY = [   -1   -1   -1   ;
                         0    0    0   ;
                         1    1    1  ];
        else
        %Gradient Kernel Family used for Prewitt, Sobel, Iso:
        GradKernX = [   -1    0    1   ;
                        2-p   0   p-2  ;
                        -1    0    1  ];
        GradKernY = [   -1   2-p  -1   ;
                         0    0    0   ;
                         1   p-2   1  ];
        end

        %Apply finite difference kernels
        Gx = (1/p) * GradKernX;
        Gy = (1/p) * GradKernY;
    end
end
```

Problem 1 continued on next page. . .