Classwork6

Min Han

```
%%Prob1

function [eq_img] = my_histeq(input_img)
    % Convert image to double for calculations
    img = double(input_img);

    % Get the histogram of the original image
    [counts, ~] = imhist(input_img);

    % Compute the cumulative distribution function (CDF)
    cdf = cumsum(counts) / numel(input_img);

    % Create the transformation function (mapping)
    transform_map = uint8(255 * cdf);

    % Apply the transformation to the input image
    eq_img = transform_map(img + 1);

    % Convert back to original image type
    eq_img = uint8(eq_img);
end



%%Prob2


function roberts_img = my_roberts(input_img)
    % Convert image to grayscale if it is not already
    if size(input_img, 3) == 3
        img = rgb2gray(input_img);
    else
        img = input_img;
    end

    % Define the Roberts cross operator kernels
    Gx = [1 0; 0 -1];
    Gy = [0 1; -1 0];

    % Apply the Roberts operator
    gx = imfilter(double(img), Gx, 'replicate');
    gy = imfilter(double(img), Gy, 'replicate');

    % Calculate the magnitude of the gradients
    roberts_img = sqrt(gx.^2 + gy.^2);

    % Normalize the output image
    roberts_img = uint8((roberts_img / max(roberts_img(:))) * 255);
end
```

```
function prewitt_img = my_prewitt(input_img)
    % Convert image to grayscale if it is not already
    if size(input_img, 3) == 3
        img = rgb2gray(input_img);
    else
        img = input_img;
    end

    % Define the Prewitt operator kernels
    Gx = [-1 0 1; -1 0 1; -1 0 1];
    Gy = Gx';

    % Apply the Prewitt operator
    gx = imfilter(double(img), Gx, 'replicate');
    gy = imfilter(double(img), Gy, 'replicate');

    % Calculate the magnitude of the gradients
    prewitt_img = sqrt(gx.^2 + gy.^2);

    % Normalize the output image
    prewitt_img = uint8((prewitt_img / max(prewitt_img(:))) * 255);
end


function sobel_img = my_sobel(input_img)
    % Convert image to grayscale if it is not already
    if size(input_img, 3) == 3
        img = rgb2gray(input_img);
    else
        img = input_img;
    end

    % Define the Sobel operator kernels
    Gx = [-1 0 1; -2 0 2; -1 0 1];
    Gy = Gx';

    % Apply the Sobel operator
    gx = imfilter(double(img), Gx, 'replicate');
    gy = imfilter(double(img), Gy, 'replicate');

    % Calculate the magnitude of the gradients
    sobel_img = sqrt(gx.^2 + gy.^2);

    % Normalize the output image
    sobel_img = uint8((sobel_img / max(sobel_img(:))) * 255);
end


%%Prob3

function sharpened_img = my_laplacian_sharpening(input_img)
    % Convert the input image to grayscale if it's not already
    if size(input_img, 3) == 3
        img = rgb2gray(input_img);
```

```
    else
        img = input_img;
    end

    % Define Laplacian filter
    laplacian_filter = [0 -1 0; -1 4 -1; 0 -1 0];

    % Apply Laplacian filter to the grayscale image
    laplacian_img = imfilter(double(img), laplacian_filter, 'replicate');

    % Sharpen the image by adding the Laplacian image to the original image
    sharpened_img = double(img) + laplacian_img;

    % Normalize the sharpened image to be in the range 0 to 255
    sharpened_img = uint8(255 * (sharpened_img - min(sharpened_img(:))) /
(max(sharpened_img(:)) - min(sharpened_img(:))));
end
```