Digital Image Processing
Lab 3

Group 25:
Henry Maathuis (S2589540)
Sebastian Wehkamp (S2589907)

December 17, 2017

Contents

1 Exercise 1

1.1 a

For this exercise we create a function called $IPpyr\_decomp$ which implements the Laplacian pyramid decomposition. This pyramid is created using Equation 1. We first create the Gaussian pyramid f_j by applying the reduce operation multiple times. Then for every level we perform the expand operation and calculate the difference. The last part of the code takes care of positioning the image in such a way that it is a pyramid and scales the pixel values to be in the interval [0, 1]. This is necessary since the Laplacian pyramid decomposition allows for computations that causes (some of) the pixel values to end up below 0 or above 1. The final version of the code can be seen in Listing 1

$$\begin{aligned} f_1 &= f \\ f_j &= REDUCE(f_{j-1}), j = 2, ...J \\ d_j &= f_j - EXPAND(f_{j+1}) j = 1, ...J - 1 \end{aligned} \tag{1}$$

```
% INPUT: image f, level J, sigma s
% OUTPUT: resulting laplacian pyramid image g,
% gaussian pyramid images G,
% laplacian images L

function [g, G, L] = IPpyr_decomp(f, J, s)
    f = im2double(f);
    M = size(f, 1);

    sum = 0;

    % Determine the height of the resulting image
    for idx = 1:J
        sum = sum + (0.5^(idx - 1));
    end

    P = M * sum;

    % Construct the matrix
    g = ones(P, M);

    % Gaussian pyramid
    G{1} = f;
    for idx = 1:J - 1
        f = IPreduce(f, s);
        G{idx + 1} = f;
    end

    % Use the size of the output image to determine where
    % each L_n image should be placed
    [g_h, g_w] = size(g);
```

```
    offsetY = 0;

    % Expand the guassian images and subtract it from the original G_n
    for idx = 1:J
        if (idx == 1)
            f =  G{J - idx + 1};
        else
            G{J - idx + 1}
            IPexpand(G{J - idx + 2}, s)
            f =  G{J - idx + 1} - IPexpand(G{J - idx + 2}, s);
        end

        % Store Laplacian images
        L{idx} = f;

        [p, q] = size(f);
        offsetY = offsetY + p;

        % Compute where the image should be placed in the resulting image
        height =  g_h - offsetY + 1 : g_h - offsetY + p;
        width = g_w / 2 - 0.5 * p + 1: g_w / 2 + 0.5 * p;
        g(height, width) = f;
    end

    % Scale values between 0 and 1
    minV = min(g(:));
    maxV = max(g(:));

    g = g - minV;
    g = g / (maxV - minV);
end
```

Listing 1: Function that implements Laplacian pyramid decomposition

1.2 b

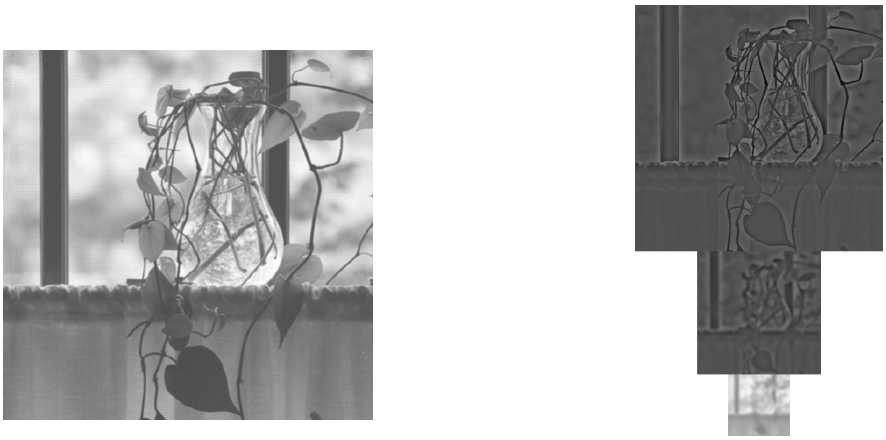
This exercise was meant to test our function listed in Listing 1. The script performs decomposition using 3 levels and a sigma of 5. The test script is listed in Listing 2.

```
img = imread('plant.tif');

figure;
imshow(img);

res = IPpyr_decomp(img, 3, 5);

figure;
imshow(res);
```

Listing 2: Test script that applies pyramid decomposition to the image 'plant.tif'

1.3 c

The input image and the output image of Listing 2 is shown in Figure 1. You can clearly see all of the levels in Figure 1b. The top two images in the pyramid (Figure 1b) are the residuals, the bottom image in the pyramid is the image obtained after reducing the original image J - 1 times. The residual images contain a lot of important textural features of the original image at different resolution scales. Having these features at different resolution scales allows us to develop algorithms that are scale-invariant.



(a) Original image plant.tif

(b) Laplacian decomposition pyramid of image 'plant.tif'

Figure 1: Input and output of the test script Listing 2

2 Exercise 2

2.1 a

In this exercise we created a function that implements the Laplacian pyramid reconstruction. This is the opposite of what we did for exercise 1 where we had to perform pyramid deconstruction. This function takes as input arguments a decomposition array g, the number of levels J, and the

standard deviation sigma. The reconstruction is performed by using Equation 2. First we have to split the input image in multiple levels, creating a new image for every level. After this is done the reconstruction is performed by using Equation 2. Similarly to the decomposition algorithm, we have to scale the pixels again to be in the interval [0, 1]. The final version of the function is found in Listing 3.

$$f_j = EXPAND(f_{j+1} + d_j, j = J_1, ..., 1 \tag{2}$$

```
% INPUT: laplacian pyramid g, level J, sigma s
% OUTPUT: reconstructed image im

function [im] = IPpyr_recon(g, J, s, or)
    g = im2double(g);
    [M, N] = size(g);

    current_height = 1;
    sz = N;

    center = N / 2;
    G = {};

    % Convert the pyramid to a list of images G
    for idx = 1:J
        height_pos = current_height:current_height + sz - 1;
        width_pos = center - (0.5 * sz) + 1 : center + (0.5 * sz);
        G{idx} = g(height_pos, width_pos);
        current_height = current_height + sz;
        sz = sz / 2;
    end

    % Set starting image to bottom level of pyramid
    im = G{J};

    % For every level expand the image and add the differences
    for idx = 1:J-1
        im = IPexpand(im, s) + G{J-idx};
    end

    % Scale values between 0 and 1
    minV = min(im(:));
    maxV = max(im(:));

    im = im - minV;
    im = im / (maxV - minV);

    imshow(im);

end
```

Listing 3: Function that implements Laplacian

2.2 b

We created a test script which applies the reconstruction formula to the pyramid decomposition of plant.tif. The input of the reconstruction is J=3 and sigma = 1.0. The output of the test script Listing 4 can be seen in Figure 2. The left image is the original while the right image is a reconstructed version of it. Note that the difference between the images is mainly due to the scaling we apply at the end of our reconstruction. When we apply scaling to the original image as well the result looks like Figure 3 in which both images are very similar.

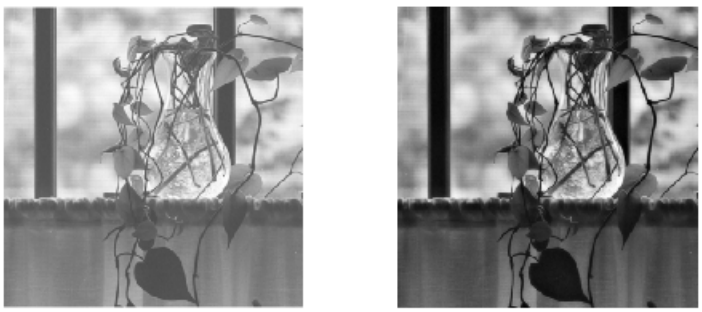


Figure 2: Comparison between the original image and the reconstructed image.

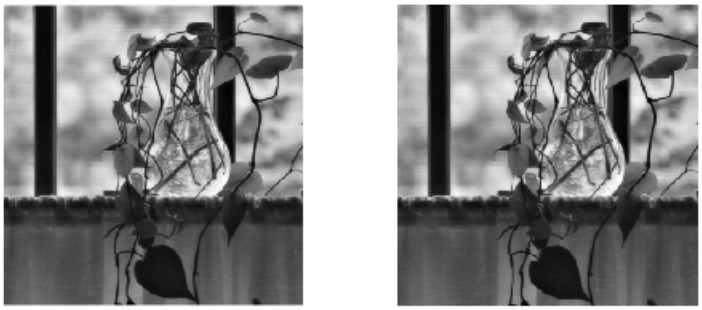


Figure 3: Comparison between the original image and the reconstructed image with scaling applied to the original

```
res = load('pyramid.mat');
res = res.res;

original_image = imread('plant.tif');
original_image = im2double(original_image);

g2 = IPpyr_recon(res, 3, 1.0);

figure;
subplot(1,2,1), imshow(original_image);
subplot(1,2,2), imshow(g2);

[M, N] = size(g2);

sum_diff = 0;

for i = 1:N
    for j = 1:N
        diff = abs(g2(i, j) - original_image(i, j));
        sum_diff = sum_diff + diff;
    end
end

```

```
error = sum_diff / (M * N)
```

Listing 4: Script that applies the reconstruction function to the image plant.tif

2.3 c

This assignment asked us to expand the Listing 4 function by adding an error calculation to it. This error calculation is already at the bottom of Listing 4. The error calculation is done by using the formula listed in Equation 3. The error calculated for the images shown in Figure 2 is 0.1672 while the error for the images shown in Figure 3 is 0.0317. As expected, the error is very small for the images where we applied scaling to. While the error is quite large when we do not apply scaling to the original image. Also note that the error is due to the fact that we use a different sigma in the reconstruction phase.

$$error(f, f_k) = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} |f(i,j) - f_k(i,j)| \tag{3}$$

We created a comparison between the original image, the reconstructed image, and the difference image as can be seen in Figure 4. Since the differences are very small we used a scaled version of the difference image to display it. As said before, these differences are mostly due to using a different sigma for reconstruction.

Figure 4: On the left is the original image, in the middle is reconstructed, and on the right is a difference image.