Image Processing
lab 3
Group 1

Alessandro Pianese S4106431
Viljami Linna S4169921

March 25, 2020

Individual contributions

Viljami Linna mainly planned and implemented a code and wrote the report related to Exercise 2. The code and report related to Exercise 1 was mainly planned and implemented by Alessando Pianese. Both members of the group are familiar with all exercise solutions. Structure and language of the report has been checked by both members of the group.

Exercise 1 – Laplacian pyramid: decomposition

a. For this assignment, it was required to write a function `IPpyr_decomp(f, J, sigma)` that produced as an output the Laplacian pyramid of input image `f` with `J` levels. The last input parameter `sigma` is used to generate the lowpass kernel for filtering the images. The process for generating a pyramid are as follows:

- start with input image f which will be labelled as f_1 to show that it's the starting image
- Obtain the next image f_2 by performing the reduce operation on the previous image $f_2 = REDUCE(f_1)$. The reduce operation is displayed in equation 4. It basically apply a smoothing filter (in this specific assignment it performs Gaussian filtering) to the input image before downsampling the image by a factor of two. The general form for this step is

$$f_j = REDUCE(f_{j-1}) \quad j = 2, 3, ..., J \tag{1}$$

- Obtain the difference image d_1 by, first, performing a expand operation on f_2 and then subtracting $EXPAND(f_2)$ to f_1. The expand operation is shown in equation 3. It consists of upsampling the input image by a factor of 2 (in this specific case the zoom operation was performed) and then apply the Gaussian filter again. The generic formula for this step is

$$d_j = f_j - EXPAND(f_{j+1}) \quad j = 1, 2, ..., J-1 \tag{2}$$

  The detail image d_j is what is actually going to be stored in the pyramid except for the last level J. At top of the pyramid (or at the bottom, depending from the point of view) is going to be placed the $J-1$ reduced image.

- Repeat the two steps right above $J-1$ times to obtain the Laplacian pyramid of J^{th} order.

$$EXPAND(f) = h_0 * (\uparrow_2 (f)) \tag{3}$$

$$REDUCE(f) = \downarrow_2 (h_0 * f) \tag{4}$$

To implement a solution, we started by implementing the support functions $EXPAND(f)$ and $REDUCE(f)$. Their implementation is displayed below in listing 1 and 2. They straightforwardly implement the steps described above [1]. However, It is observable how these two functions presents two way of performing gaussian blur. One with the matlab built-in function `imgaussfilt` (which is commented) and the other one with our own implementation of a gaussian filter `gaussBlur` which is presented in listing 3.

```
1 function g = expand(f, sigma)
2 %function that perform the expansion and gaussian blur
3 factor=2;
4 g=IPzoom(factor, f);
5 %g=imgaussfilt(g, sigma);
6 g=gaussBlur(g, sigma);
7 end
```

Listing 1: Expand

```
1 function g = reduce(f, sigma)
2 %function that performs gaussian blur and reduction
3 factor=2;
4 %g=imgaussfilt(f, sigma);
5 g=gaussBlur(f, sigma);
6 g=IPdownsample(g, factor);
7 end
```

Listing 2: Reduce

The `gaussBlur` implementation is fairly simple. After defining a bidimensional grid with the desired interval, which is dependent on the kernel size, we calculate the kernel as a product of two gaussians. The normalization factor $\frac{1}{\sqrt{2\pi}\sigma}$ is replaced with the normalization on line 13 which divide each element of the kernel for the sum of all elements. In the end, the filter is applied by calling the function `IPfilter`[2].

```
1 function g = gaussBlur(f, sigma)
2
3 kernel_size = 3;
4 %Define interval to use for generating the bidimensional grid
5 interval = -floor(kernel_size/2) : floor(kernel_size/2);
6 %Generate bidimensional grid
7 [X, Y] = meshgrid(interval, interval);
```

[1] These two function make also use of previously discussed function `IPzoom` and `IPdownsample` which implementation was given to us before the lab. Since these two functions were fairly well discussed in the report for the first lab, nothing will be added on top of that.

[2] The implementation of this function is re-used from the second lab assignment. Since, as previously mentioned, this function has already been fairly discussed, nothing will be added on top of what is already been said.

```
8
9  %Define kernel as product of two gaussian
10 kernel=exp( -(X.^2 + Y.^2) / (2*sigma*sigma) );
11
12 %normalization factor
13 kernel=kernel/sum(kernel(:));
14
15 %apply gaussian kernel to input image
16 g=IPfilter(f, kernel);
17
18 end
```

Listing 3: Our own implementation of Gaussian filtering

After having laid out these function, the implementation proceded smoothly. The code is presented in listing 8. The idea behind the function is presented intro three bits of code. Firstly, from line 7 to line 11 we get the value of P which will be the number of rows that the final pyramid image is going to have. To obtain this, we simply implemented the formula

$$P = M \times (1 + \sum_{i=1}^{J-1} (\frac{1}{2})^i)$$

and with this value, we initialize the output image g on line 14.

```
6  %Compute the series to calculate the size of P
7  sum=1;
8  for k=1:J-1
9      sum=sum+(0.5^k);
10 end
11 P = M * sum;
12
13 %Init output image with ones
14 g=ones(P,M);
```

Listing 4: Definition of number of rows P.

Secondly, in the core loop of the function the pyramid is actually built. Since the function only translate to code the theory discussed at the start of the paragraph, the only element that may cause confusion is the naming scheme of variable and the indexing of images. For the name, we have four variables that represents images:

- **curr** is the current image being handled and represents f_{j-1} in equation 1
- **next** is the reduced image and represents f_j in equation 1
- **zoomed** is the expanded image next and represents $EXPAND(f_j)$ in equation 2
- **d** is the difference image and represents d_j in equation 2

The other two variables that may be a bit harder to read are `rStart` and `cStart` which respectively represents the row index and the column index of the starting position of the current detail image of the loop. The behaviour of `rStart` is simple, it starts at one (the first index) and the gets incremented by the size of the image that has been inserted into the pyramid on line 32. `cStart`, instead, subtracts from M (which is the columns number of the output image) the number of columns of the current detail image, to get the sum of the lateral "padding", and then divide it by two to obtain the starting column value.

On line 30 we can see how these indexes are used to position the detail image d inside the output image g.

```
16 %Core loop that create the pyramid layers
17 %curr image for firt iteration is input image while next is the
       reduced
18 %image
19 %zoomed is expanded next
20 curr=f;
21 %variable for keeping track of images starting row index
22 rStart=1;
23 for i=1:J-1
24     next=reduce(curr, sigma);
25     zoomed=expand(next, sigma);
26     %get details
27     d=curr-zoomed;
28     %get the starting column index
29     cStart=((M-size(d,1))/2)+1;
30     g(rStart:rStart+size(d,1)-1, cStart:M-cStart+1)=d;
31     %update variables for next cicle
32     rStart=rStart+size(d,1);
33     curr=next;
34 end
```

Listing 5: Core loop of function that builds the pyramid.

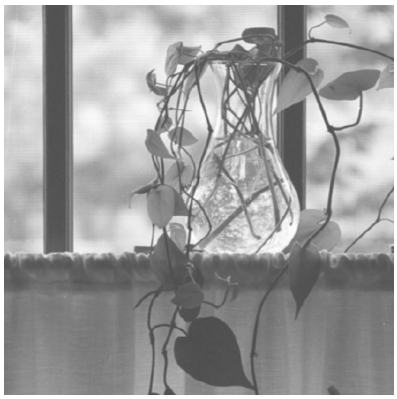
Lastly, the function places the last reduced image `next` at the bottom of the pyramid on line 38 after calculating the starting column index on line 37 in a similar fashion to what has been showed previously.

```
36 %put top pyramid layer at the bottom of the image
37 cStart=((M-size(next,1))/2)+1;
38 g(P-size(next,1)+1:P,cStart:M-cStart+1)=next;
```

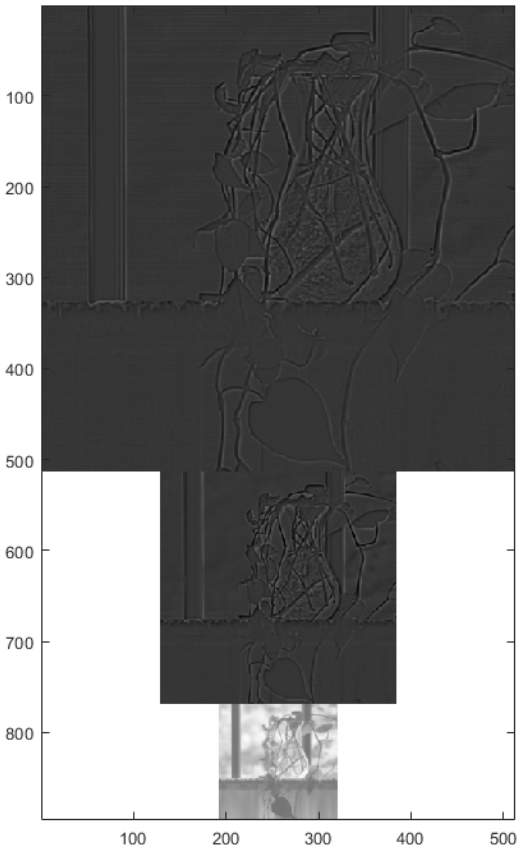
Listing 6: Add last image to pyramid.

b. This part of the assignment required us to write a script that would call the above descripted function. The parameter that we needed to call the function with were $J = 3$ and $\sigma = 1.0$. We called the function twice, one with the matlab built-in function `imgaussfilt` which is displayed in figure 1b and the other one with our own `gaussBlur` function which is displayed in figure 2b. We can observe how, overall, there are no big differences. The only one notable difference that can be observed are the top and left borders of the middle detail image in figure 2b, where there are two very bright line of pixels. This, in our opinion, is very likely due to how the `IPfilter` function applies the gaussian kernel. Since the difference in detail in figures 1b and 2b are almost none, We can also add that the `imgaussfilt` function uses a kernel of size similar to what we have been using for our function. As both figures 1b and 2b show, we can confirm that the function does what it's meant to do and create a pyramid of detail images from a starting image.

c. This last point asked us to save the resulting images (and raw data) to a filename that showed the parameter with which the image and data were generated. These files will be used to implement exercise 2 of the assignment where we reconstruct the image from the pyramid.

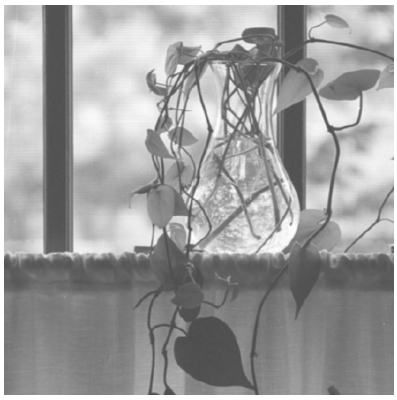


(a) The image: '`plant.tif`'

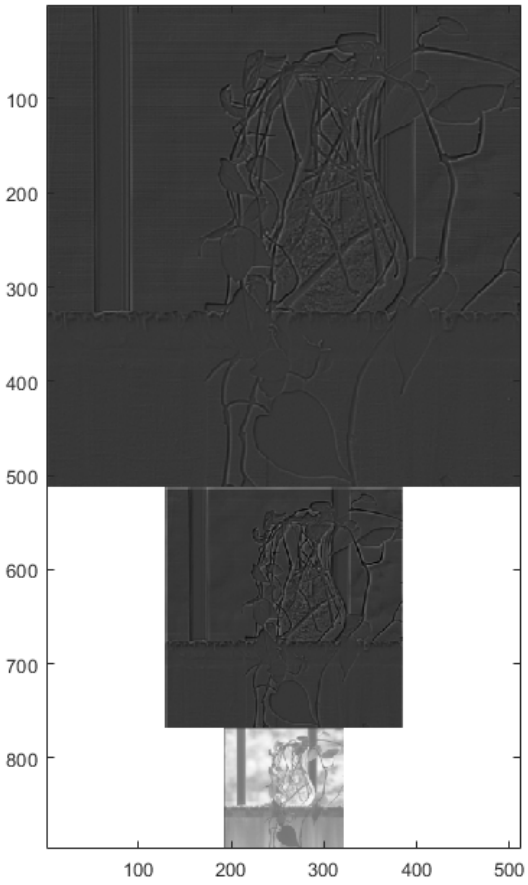


(b) Laplacian pyramid of '`plant.tif`'

Figure 1: Comparison between original image and pyramid built using `imgaussfilt`



(a) The image: '`plant.tif`'



(b) Laplacian pyramid of '`plant.tif`'

Figure 2: Comparison between original image and pyramid built using `gaussBlur`

Exercise 2 – Laplacian pyramid: reconstruction

a. The assignment was to implement a function which takes input parameters g(pyramid decomposition array), J(number of levels) and $sigma$(the standard deviation of the Gaussian filter). The function had to construct the Laplacian pyramid reconstruction to the given Laplacien pyramid decomposition. In this exercise we could assume original image to be square image of size MxM where M is power of 2.

Laplacian pyramid decomposition array contains J layers of images. On the bottom of the pyramid we have the coarsest approximation of the input image f_J. On top of it we have difference images $d_1, d_2, ..., d_{J-1}$ so that the size of d_{J-1} is same as the original image.

Reconstructing the original image we repeatedly reconstructed f_J until we reached f_1 which is the reconstruction of the original image.

Reconstruction operation is defined as follows:

$$f_j = EXPAND(f_{j+1}) + d_j, j = J-1, ..., 1. \tag{5}$$

The $EXPAND()$ operation was defined earlier in Listing 3.

Our implementation including helper functions is represented in appendices in Listing 9. At first, we extracted the coarsest approximation of the image from the pyramid image:

```
5     g2 = extract_layer(g,J);
```

After that, we repeated reconstruction operation until we reached the reconstruction of the original image f_1:

```
6     for iteration = (J-1):-1:1
7         g2 = expand(g2, sigma);
8         difference = extract_layer(g,iteration);
9         g2 = g2 + difference;
10    end
```

We defined three helper functions `expand`, `geosum` and `extract_layer` which both the main function uses.

The `expand` function preforms the $EXPAND()$ operation to given image f using given input parameter $sigma$ as a standard deviation of the Gaussian blurring operation.

```
1     function g = expand(f, sigma)
2         %function that perform the expansion and gaussian blur
3         factor=2;
4         g=IPzoom(factor, f);
5         %g=imgaussfilt(g, sigma);
6         g=gaussBlur(g, sigma);
7     end
```

The `geosum` function returns the sum of the geometric series with given constant ratio between two terms and the power of the last term of the sum.

```
1     function S = geosum(factor,power)
2         S = sum(factor.^(0:power));
3     end
```

The `extract_layer` function extracts the j:th layer of the Laplacian pyramid as a separate image. This function takes $pyramid$ and $current\_j$ as an input parameters.

```
    function layer = extract_layer(pyramid, current_j)
        width = size(pyramid,2);
        %current layer placement dimensions on pyramid
        y_end = width*geosum(1/2,(current_j-1));
        y_start = 1 + width*geosum(1/2,(current_j-2));
        x_start = width*(geosum(1/2,(current_j-1))-1)/2 + 1;
        x_end = width - x_start  + 1;
        layer = pyramid(y_start:y_end,x_start:x_end);
    end
```

b. In this part of the exercise we had to write a test script for our reconstruction function `IPpyr_recon` represented in Listing 9. In the script we had to define g as the decomposition array of the image `plant.tif`, $J = 3$ and $sigma = 1.0$. Our implementation is represented in Listing 10.

At first, we defined the input parameters for the `IPpyr_recon`. Parameter g was defined by loading the decomposition array made in Exercise 1.

```
function IPpyr_recon_test()

    filename = 'plant.tif';
    J = 3;
    sigma = 1.0;
    originalImage = imread(filename);
    load 'J=3-sigma=1-g.mat';
```

Next, we run our reconstruction function using the defined parameters

```
    reconstructedImage = IPpyr_recon(g, J, sigma);
```

Reconstructed image compared to original image is represented in Figure 3

c. In this part of the exercise we extended the test script implemented in (b) computing the error between the reconstructed image $g2$ and the original image f. Error is defined as follows:

$$error(f, g) = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} |f(i, j) - g(i, j)|. \tag{6}$$

Our implementation is represented in Listing 7.

```
function error_value = calc_error(org_img, recon_img)
    if size(org_img,1) ~= size(recon_img,1) || size(org_img,2) ~=
        size(recon_img,2)
        error('Image dimensions not match');
    end
    error_value = 1/(size(org_img,1)*size(org_img,2))*sum(abs(org_img
        -recon_img),'all');
end
```

Listing 7: Error calculation

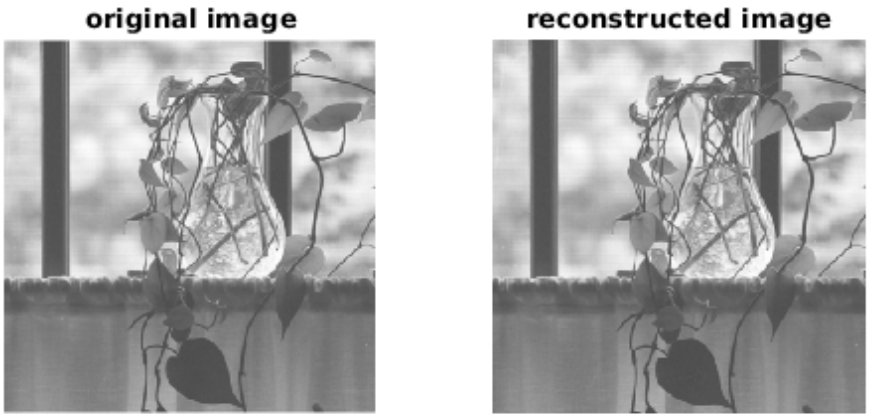


Figure 3: Reconstructed image compared to the original image.

We also add line in our implementation computing the difference image between reconstructed and original image. The difference image can be calculated by taking an absolute value from the difference of every corresponding pixel value of both images.

Following code represent the difference image computation in our implementation:

```
9     difference_img = abs(im2uint8(reconstructedImage)-originalImage);
```

d. In this part of the exercise we had to apply our extended script represented in Listing 10 to the image `plant.tif`.

The original image, reconstructed image and difference image is represented in Figure 4. We got error value 0 which means that there is no difference between original and reconstructed image.

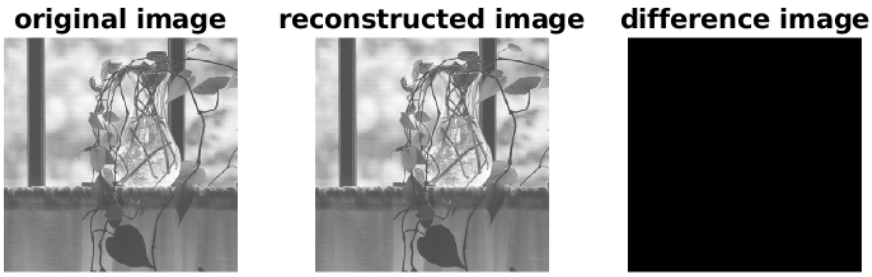


Figure 4: Original, reconstructed and difference image.

Appendix A The MATLAB code

```
function g = IPpyr_decomp(f, J, sigma)

%get input image size
M = size(f, 1);

%Compute the series to calculate the size of P
sum=1;
for k=1:J-1
    sum=sum+(0.5^k);
end
P = M * sum;

%Init output image with ones
g=ones(P,M);

%Core loop that create the pyramid layers
%curr image for firt iteration is input image while next is the reduced
%image
%zoomed is expanded next
curr=f;
%variable for keeping track of images starting row index
rStart=1;
for i=1:J-1
    next=reduce(curr, sigma);
    zoomed=expand(next, sigma);
    %get details
    d=curr-zoomed;
    %get the starting column index
    cStart=((M-size(d,1))/2)+1;
    g(rStart:rStart+size(d,1)-1, cStart:M-cStart+1)=d;
    %update variables for next cicle
    rStart=rStart+size(d,1);
    curr=next;
end

%put top pyramid layer at the bottom of the image
cStart=((M-size(next,1))/2)+1;
g(P-size(next,1)+1:P,cStart:M-cStart+1)=next;

colormap(gray(256));
imagesc(g);

end

function g = expand(f, sigma)
%function that perform the expansion and gaussian blur
factor=2;
g=IPzoom(factor, f);
%g=imgaussfilt(g, sigma);
g=gaussBlur(g, sigma);
end
```

```
52
53 function g = reduce(f, sigma)
54 %function that performs gaussian blur and reduction
55 factor=2;
56 %g=imgaussfilt(f, sigma);
57 g=gaussBlur(f, sigma);
58 g=IPdownsample(g, factor);
59 end
```

Listing 8: IPpyr_decomp

```
1  function g2 = IPpyr_recon(g,J,sigma)
2      % pyramid dimensions
3      P = size(g,2);
4      M = size(g,1);
5      g2 = extract_layer(g,J);
6      for iteration = (J-1):-1:1
7          g2 = expand(g2, sigma);
8          difference = extract_layer(g,iteration);
9          g2 = g2 + difference;
10     end
11
12
13
14
15     function g = expand(f, sigma)
16         %function that perform the expansion and gaussian blur
17         factor=2;
18         g=IPzoom(factor, f);
19         %g=imgaussfilt(g, sigma);
20         g=gaussBlur(g, sigma);
21     end
22
23     function layer = extract_layer(pyramid, current_j)
24         width = size(pyramid,2);
25         %current layer placement dimensions on pyramid
26         y_end = width*geosum(1/2,(current_j-1));
27         y_start = 1 + width*geosum(1/2,(current_j-2));
28         x_start = width*(geosum(1/2,(current_j-1))-1)/2 + 1;
29         x_end = width - x_start  + 1;
30         layer = pyramid(y_start:y_end,x_start:x_end);
31     end
32
33     function S = geosum(factor,power)
34         S = sum(factor.^(0:power));
35     end
36 end
```

Listing 9: IPpyr_recon

```
1  function IPpyr_recon_test()
2
3      filename = 'plant.tif';
4      J = 3;
```

```
    sigma = 1.0;
    originalImage = imread(filename);
    load 'J=3-sigma=1-g.mat';
    reconstructedImage = IPpyr_recon(g, J, sigma);
    difference_img = abs(im2uint8(reconstructedImage)-originalImage);

    error = calc_error(im2double(originalImage), reconstructedImage);
    figure_title = ['Error: ' num2str(error)];
    f1 = figure(1);
    subplot(1,3,1);
    imshow(originalImage);

    title('original image');

    subplot(1,3,2);
    imshow(reconstructedImage);
    title('reconstructed image');


    subplot(1,3,3);
    imshow(difference_img);
    title('difference image');
    sgtitle(figure_title);
    saveas(f1,'IPpyr_recon_test_results.png')
end
```

Listing 10: IPpyr_recon$_test$