Image Processing
lab 1
Group 1

Alessandro Pianese S4106431
Viljami Linna S4169921

March 25, 2020

Individual contributions

Viljami Linna mainly planned and implemented a code and wrote a report related to the Exercise 1 and 2. The code, and report related to Exercise 3 was mainly planned and implemented by Alessando Pianese. Both members of the group are familiar with solutions of every exercise. Structure and language of the report has been checked by both members of the group.

Exercise 1 – Downsampling, upsampling, and zooming

a. The assignment was to write a function IPdownsample capable of downsampling (shrinking) an image with desired downsampling factor type of integer.

Downsampling can be done in many different ways. In this exercise, we used the desired integer factor to reduce image size dividing it by the value of the given factor.

Precisely we constructed a matrix from the image by representing every pixel as an unsigned 8-bit integer depending on the grayscale value of the pixel. From that matrix, our algorithm picks rows to downsampled image so that only first of every factor defined amount of rows is represented in it. After the right rows is picked we repeat the same operation to columns. Listing 1 shows our implementation of above-mentioned algorithm.

```
function outputImage = IPdownsample(shrinkFactor,fileName)
    file = imread(fileName);
    outputImage = file(1:shrinkFactor:end,:);
    outputImage = outputImage(:,1:shrinkFactor:end);
end
```

Listing 1: IPdownsample

b. The goal of the exercise was to construct a downsampled image by using the algorithm presented in (a) with given downsampling factor 4. Figure 1 was used as an input file to the IPreduce.

The output image is shown in Figure 2. As the results show, in the downsampled image most of the characters of the board are still recognizable but not so clearly as in the original image.

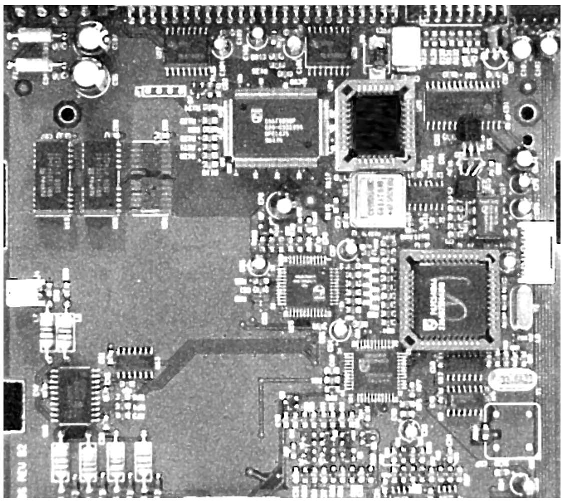


Figure 1: The image: ‘cktboard.tif’.

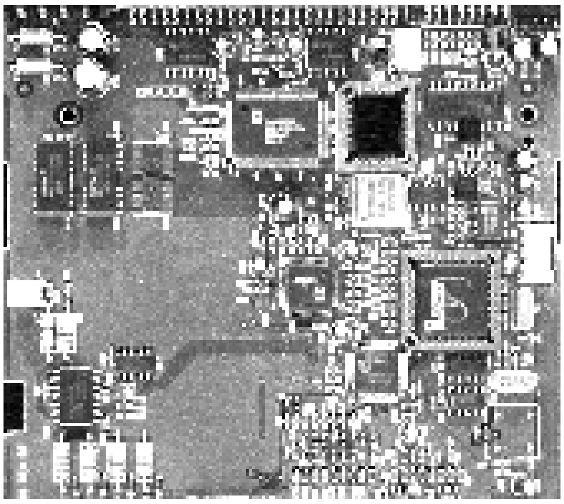


Figure 2: The image: ‘DScktboard.tif’.

c. The assignment was to write a function IPupsample capable of upsampling an image by introducing zeroes depending on desired integer upsampling factor.

Our algorithm first constructs a zero matrix which dimensions are defined by multiplying original image dimensions with the desired factor. Next, the algorithm replaces zeros at (factor*n,factor*m) with corresponding original image pixels value (n,m). Listing 2 shows our implementation of the algorithm.

```
function outputImage = IPupsample(expandFactor,filename)
file = imread(filename);

outputImage = zeros(expandFactor*size(file), 'uint8');
outputImage(1:expandFactor:end, 1:expandFactor:end) = file;

end
```

Listing 2: IPupsample

d. The goal of the exercise was to construct an upsampled image by using the algorithm represented in (c) with given upsampling factor 4. Figure 3 was used as an input to the IPupsample.

The result of the upsampled image is shown in Figure 4. As the results show, the original image is not anymore recognizable in the upsampled image if we not zoom it. The reason why the upsampled image looks black is that we used ”introducing zeroes”-method making all pixels without given information from the original image representing black. In this case, the amount of these black pixels is four times higher than non-black pixels. If we use factor 2 instead of 4 the characters of the original image can still be seen more clearly behind the black pixels as shown in Figure 5.

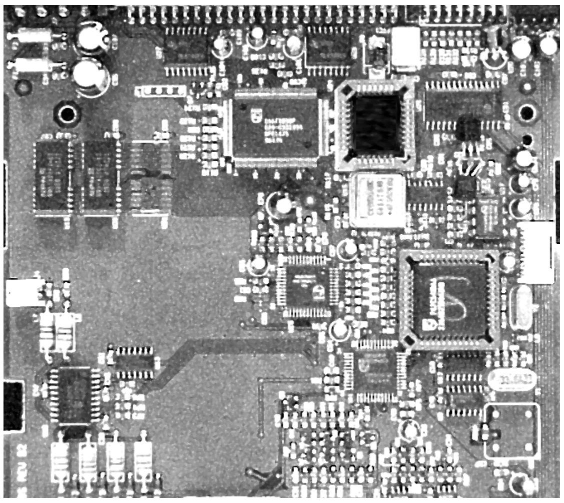


Figure 3: The image: ‘cktboard.tif’.

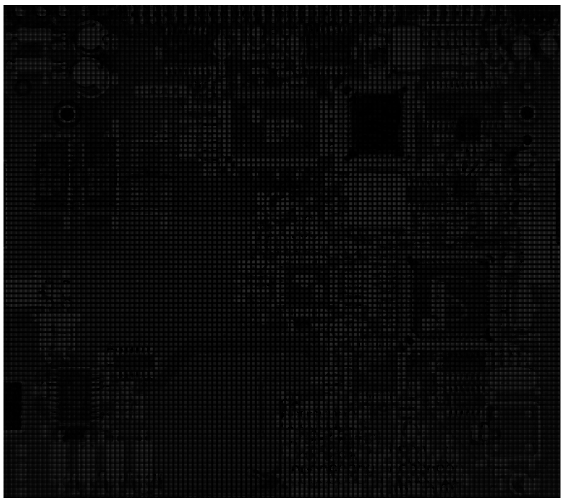


Figure 4: The image: ‘US4cktboard.tif’.

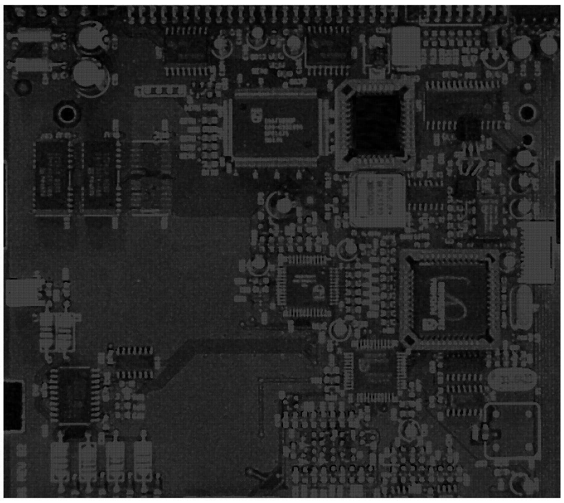


Figure 5: The image: ‘US2cktboard.tif’.

e. The assignment was to write a function called IPzoom capable of zooming an image by pixel replication, assuming that the desired zoom factor is an integer. In our implementation displayed in Listing 3, a new image O is constructed by using the original image $I(n \times m)$ and the given factor f so that

$$O_{f \cdot n - a, f \cdot m - a} = I_{n,m} \forall a \in \{0, 1, 2, .., f - 1\}. \tag{1}$$

```
function outputImage = IPzoom(zoomingFactor, filename)
file = imread(filename);
if zoomingFactor <= 1
    outputImage = file;
else
    outputImage = repelem(file, zoomingFactor, zoomingFactor);
end
end
```

Listing 3: IPzoom

f. The goal of the exercise was to construct an image using the function displayed in (e) using Figure 6 as an input file with given zooming factor 4. As we can see in Figure 7, the result looks similar comparing to the input image. Precisely the output image is four times bigger than the original file.

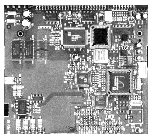


Figure 6: The image: 'cktboard.tif'.

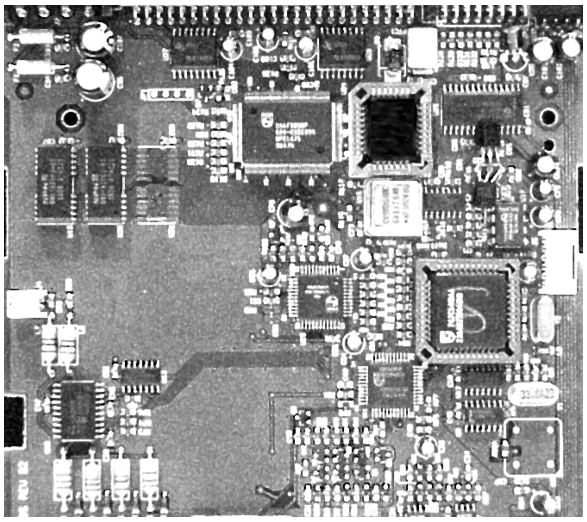


Figure 7: The image: 'Zoomedcktboard.tif'.

g. The assignment was to zoom the downsampled image (Figure 2) using function represented in Listing 3 back to the resolution of the original and then compare the result with the original image. Input file (a) and output file (b) is represented in Figure 8.

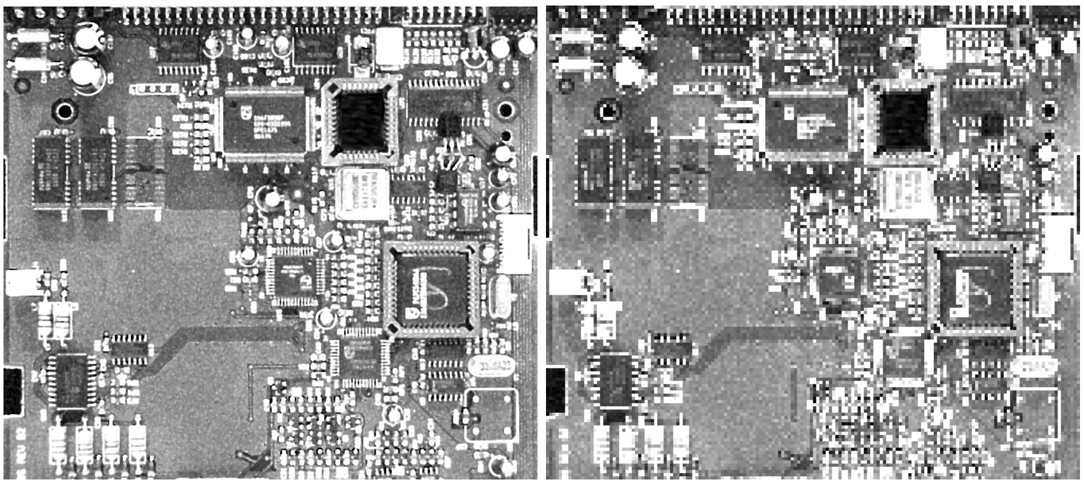

(a) Original file (b) Re-zoomed file

Figure 8: Comparison result after using IPdownsample and IPzoom to 'cktboard.tif.'

As seen in Figure 8 the output file is kind of a grainy version of the original file. Main characteristics still appear in the downsampled image but the very little details as text in the board are not readable anymore. The reason behind these effects is downsampling operation, where 3/4 of the information from the original image is lost. When zooming image after that, every preserved bit must be repeated four times.

Exercise 2 – Histogram equalization

a. The assignment of this part was to write a function IPhistogram for computing the histogram of an 8-bit image. Our implementation takes a file name as an input and returns struct called 'histogram' which contains fields 'values' and 'counts.' Values is defined as a vector of integers including every possible unsigned 8-bit integer value. For each value(i) exist count(i) which contains total amount of detected pixels which grayscale value equals value(i). Our implementation is displayed in Listing 4. At first, our algorithm constructs a vector from the matrix of image pixels. After that, it iterates through all pixels and every time adds one to the value in depending index of 'counts.'

```
function histogram = IPhistogram(filename)
    file = imread(filename);
    fileBytes = reshape(file, 1,[]);
    histogram.counts = zeros(1, (2^8));
    for i = 1:size(fileBytes,2)
        histogram.counts(fileBytes(i) +1) = histogram.counts(
            fileBytes(i) + 1) + 1;
    end
    histogram.values = (0:1:(2^8-1));

```

```
end
```

Listing 4: IPhistogram

b. In this exercise, we had to implement the histogram equalization function IPhisteq. To achieve the equalization we used a transform function

$$O(f(x,y)) = (L-1)P(f(x,y)) \tag{2}$$

where O is output pixel grayscale value and

$$P = \frac{1}{N}\sum_{m=0}^{l} h(m). \tag{3}$$

P can be seen as a normalized grayscale value which is scaled to fit the grayscale in the actual implementation using Equation 2. Our implementation takes a file name as an input parameter and returns transformed file which histogram has been equalised.

At first we read original file and construct its histogram using function IPhist represented in Listing 4.

```
file = imread(filename);
fileHistogram = IPhistogram(filename);
```

Next, we constructed output file with copying original file in it. We also calculated amount of pixels at this point.

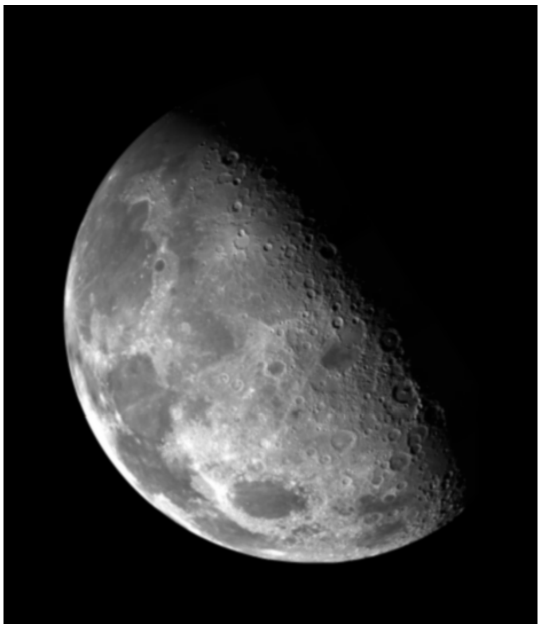
```
transformedFile = file;

numberOfBytes = numel(file);
```

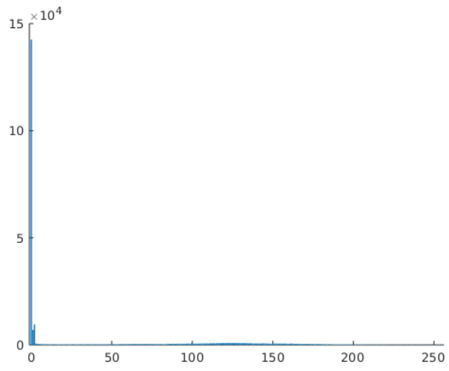
The actual equalization was made with using the expression represented in Equation 2.

```
for i = 1:numberOfBytes
    currentByteValue = file(i);
    transformedValue = 255*(1/numberOfBytes)*sum(fileHistogram.counts
        (1:(currentByteValue+1)));
    transformedFile(i) = transformedValue;
end
```

The whole implementation of our algorithm is displayed in Listing 7.

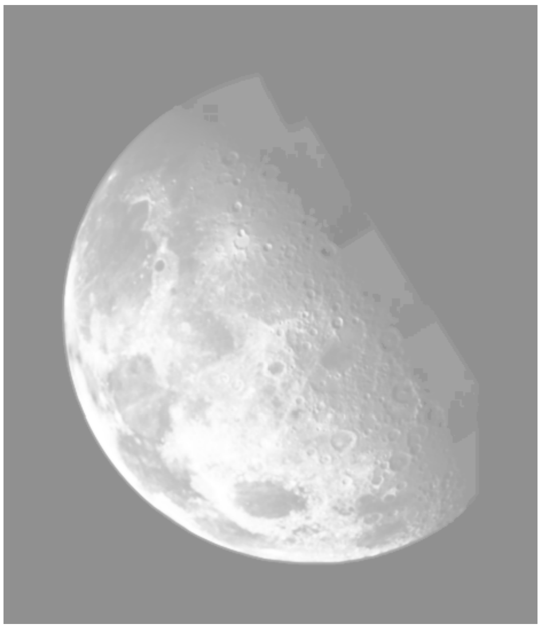
c. The assignment was to perform histogram equalization to given image using the function IPHisteq represented in the b part of this exercise. Input image and its histogram is presented in Figure 9. Output image and its histogram is presented in Figure 10.

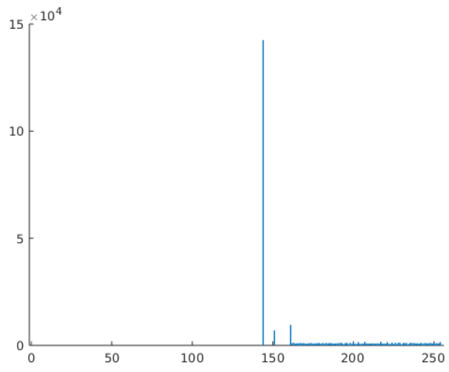

(a) The image: 'blurrymoon.tif'



(b) Histogram of 'blurrymoon.tif'

Figure 9: IPHisteq input image and its histogram.


(a) Equalized 'blurrymoon.png'



(b) Histogram of equalized 'blurrymoon.tif'

Figure 10: IPHisteq output image and its histogram.

In Figure 9 we see that most of the pixels (over 50%) in input image have a grayscale value zero. When we use Equation 3 with this kind of image, all black pixels become displayed pretty bright in the output image. Dominant rate of smallest possible grayscale value in the input image causes output image contrast being reduced. As we can see in Figure 10 only grayscale values over 143 used in output image.

Exercise 3 – Spatial filtering

a. This part of the assignment required a function `IPfilter` to perform spatial filtering on a 8 bit image using a 3x3 filter. To perform spatial filtering on an input image f means to produce an output image g such that

$$g(x,y) = \sum_{-a}^{a} \sum_{-b}^{b} w(s,t) f(x+s, y+t)$$

where $w(s,t)$ is a set of coefficients and both s,t span over a neighbourhood N of the origin:

$$N = \{(s,t) : -a \leq s \leq a, -b \leq t \leq b\}$$

To achieve this, the sharpening filter displayed in table 1 has been used. First of all, we obtained the sizes of the input image `img` in lines 8-9, where `R` stands for rows and `C` stands for columns.

```
8  R = size(img,1);
9  C = size(img,2);
```

The same goes for the filter on line 12-13, after it has been declared on line 11.

```
11 filter = [0 -1 0; -1 5 -1; 0 -1 0];
12 FR = size(filter, 1);
13 FC = size(filter, 2);
```

On line 15, we initialized the matrix used to apply the filter with all zeros. This matrix has one more row and column for each side in this particular case where the filter is a 3 by 3 matrix.

```
15 imgp = zeros(R + FR -1, C + FC -1);
```

On line 17, the input image has been put in the middle of the just initialized matrix.

```
17 imgp(1+floor(FR/2):R+floor(FR/2), 1+floor(FC/2):C+floor(FC/2)) = img;
```

0	-1	0
-1	5	-1
0	-1	0

Table 1: Sharpening filter

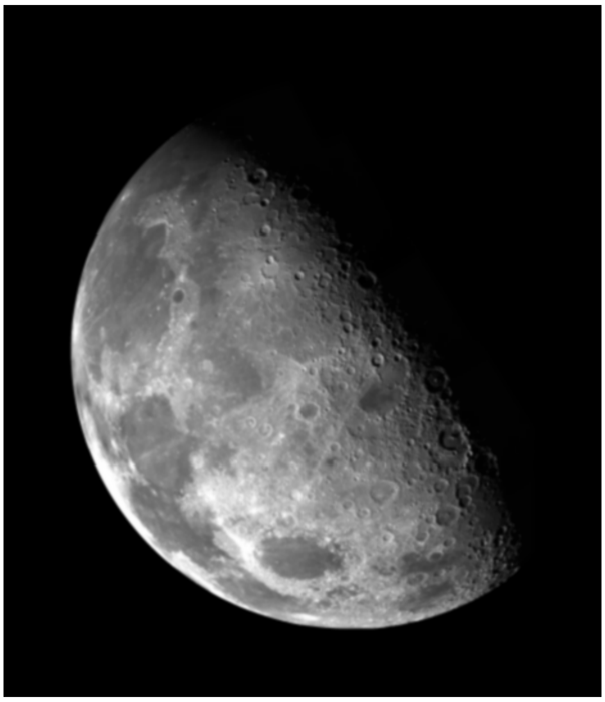


Figure 11: blurrymoon.tif

on lines 19-23, for each pixel of the input image, we first select the 8 neighbours matrix of the current pixel, then we multiply all the elements in the same position in this matrix by the filter matrix and we then sum the results.

```
19 for i=1:R
20     for j=1:C
21         img(i,j)=sum(sum(filter.*imgp(i:i+FR-1, j:j+FC-1)));
22     end
23 end
```

For this function we used the `blurrymoon.tif` image displayed in figure 11 and the algorithm produced the figure 12 as a result. We can easily observe how the resulting image has more sharp borders.

The complete code for this function is displayed in listing 8.

b. To implement Laplacian filters, we just replaced the filter used in `IPfilter` with the Laplacian filters displayed in table 2.

$$H(u,v) = -4 + 2cos(\frac{2\pi u}{M}) + 2cos(\frac{2\pi v}{N}) \tag{4}$$

The code, although it's mostly the same, is displayed in listing 9. The only notable differences are the filter declaration and, therefore, the output image generation:

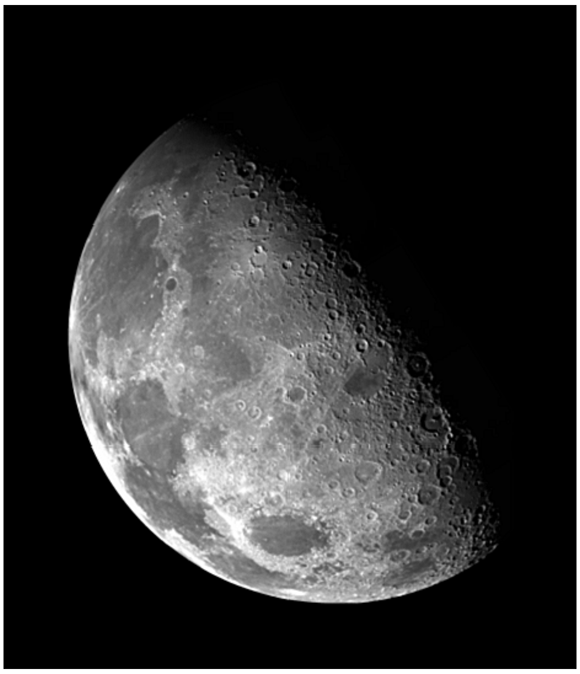


Figure 12: Applying IPfilter to figure 11

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Table 2: Laplacian filter

```
filter1 = [0 -1 0; -1 4 -1; 0 -1 0];
filter2 = [-1 -1 -1; -1 8 -1; -1 -1 -1];
```

Listing 5: IPlaplacian

```
for i=1:R
    for j=1:C
        img1(i,j)=sum(sum(filter1.*imgp(i:i+FR-1, j:j+FC-1)));
        img2(i,j)=sum(sum(filter2.*imgp(i:i+FR-1, j:j+FC-1)));
    end
end
```

Listing 6: IPlaplacian

c. We applied the `IPlaplacian` function to the image `bubble.tif` displayed in figure 13. As a results, we got two images, one obtained by applying the 4-neighbours Laplacian filter (figure 14) and the other one by applying the 8-neighbours Laplacian filter (figure 15). We can observe how in both figures the edges of the items portrayed in the input image are the only data left in it. However, in figure 14, the edges are weaker than the ones in figure 15. This is due to the number of neighbours considered for the edge detection. More neighbours imply more data to add to the central pixel, boosting its value.

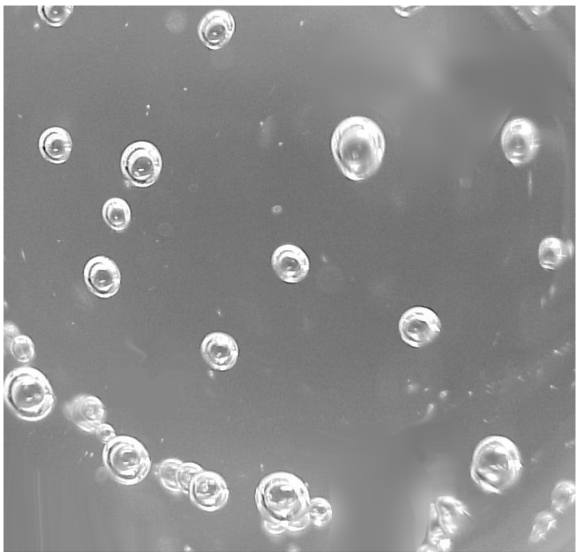


Figure 13: The input image for the laplacian filtering

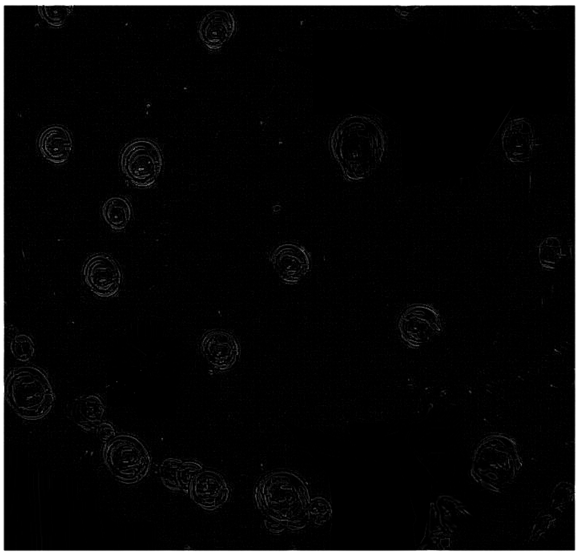


Figure 14: Figure 13 after 4-neighbours Laplacian filter

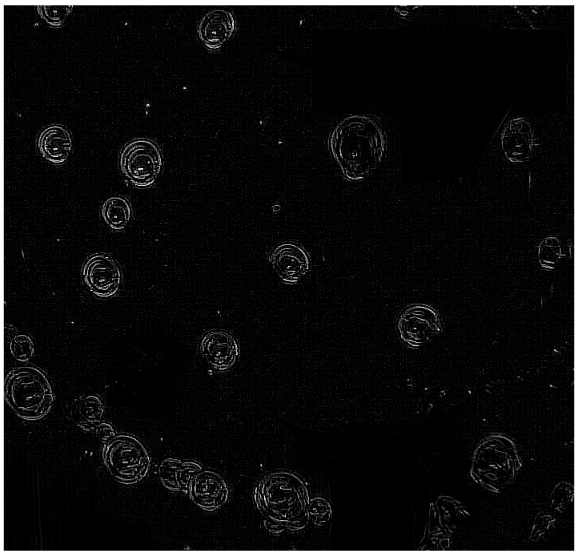


Figure 15: Figure 13 after 8-neighbours Laplacian filter

Appendix A The MATLAB code

```
function outputFile = IPhisteq(filename)
file = imread(filename);
fileHistogram = IPhistogram(filename);
transformedFile = file;

numberOfBytes = numel(file);
for i = 1:numberOfBytes
    currentByteValue = file(i);
    transformedValue = 255*(1/numberOfBytes)*sum(fileHistogram.counts(1:(
        currentByteValue+1)));
    transformedFile(i) = transformedValue;
end
outputFile = transformedFile;
end
```

Listing 7: IPhisteq

```
function out = IPfilter(inImg)
%IPFILTER Summary of this function goes here
%   Detailed explanation goes here

inImg = ['../images/', inImg,'.tif'];
img = imread(inImg);
img= im2double(img);
R = size(img,1);
C = size(img,2);

filter = [0 -1 0; -1 5 -1; 0 -1 0];
FR = size(filter, 1);
FC = size(filter, 2);

imgp = zeros(R + FR -1, C + FC -1);

imgp(1+floor(FR/2):R+floor(FR/2), 1+floor(FC/2):C+floor(FC/2)) = img;

for i=1:R
    for j=1:C
        img(i,j)=sum(sum(filter.*imgp(i:i+FR-1, j:j+FC-1)));
    end
end

imshow(img);

end
```

Listing 8: IPfilter

```matlab
function IPlaplacian(inImg)
inImg = ['../images/', inImg,'.tif'];
img = imread(inImg);
img= im2double(img);
R = size(img,1);
C = size(img,2);

filter1 = [0 -1 0; -1 4 -1; 0 -1 0];
filter2 = [-1 -1 -1; -1 8 -1; -1 -1 -1];
FR = size(filter1, 1);
FC = size(filter1, 2);

imgp = zeros(R + FR -1, C + FC -1);

imgp(1+floor(FR/2):R+floor(FR/2), 1+floor(FC/2):C+floor(FC/2)) = img;

img1 = zeros(size(img, 1), size(img,2));
img2 = zeros(size(img, 1), size(img,2));

for i=1:R
    for j=1:C
        img1(i,j)=sum(sum(filter1.*imgp(i:i+FR-1, j:j+FC-1)));
        img2(i,j)=sum(sum(filter2.*imgp(i:i+FR-1, j:j+FC-1)));
    end
end

%imshow(img1);
outputfile = ['./','bubblesLap1.png'];
imwrite(img1, outputfile);
%imshow(img2);
outputfile = ['./','bubblesLap2.png'];
imwrite(img2, outputfile);

end
```

Listing 9: IPlaplacian