1 Image enhancement using intensity transformations

1. By inspecting the given figures we can see that the transformation is simply a polynomial transformation of degree $\gamma$ for both the positive and negative intensity transformations. Let us call $s_k^{(p)}$ the positive transformation and $s_k^{(n)}$ for the negative transformation. Therefore, the class of functions, parameterized by $\gamma, \gamma > 0$, that maps the normalized intensity levels $r_k \in \{0, \frac{1}{L-1}, \ldots, 1\}$ into the normalized intensity levels $s_k = r_k$ are:

\[
\begin{align*}
  s_k^{(p)} &= r_k^\gamma \\
  s_k^{(n)} &= (1 - r_k)^\gamma.
\end{align*}
\]

2. Below we present a MATLAB function that implements both the intensity transformation as well as the contrast stretching for positive and negative images. In Figs.1 and 2 we show some results obtained after applying the transformations to an image. Note that: i) the gamma correction enhances the white intensities when $\gamma < 1$ and it enhances the black intensities when $\gamma > 1$ despite of considering the
Figure 1: Intensity transformations: positive and negative gamma corrections: a) Original image and its histogram. Positive, b), and negative, c), images corrected with $\gamma = 0.5$. Positive, d), and negative, e), images corrected with $\gamma = 2$.

positive or the negative image; ii) the histogram stretching produces a binary image when the slope $\alpha$ is infinite.

```matlab
function ImgT=IntensityTransformation(Img,Type,Gamma,Inv)
% IntensityTransformation: function to modify the intensity of the image
% Img according to the polynomial function: ImgT(x,y)=Img(x,y)^Gamma
% Inputs:
Figure 2: Intensity transformations: positive and negative contrast stretching: a) Original image and its histogram. Positive, b), and negative, c), images transformed with $\alpha = 2$. Positive, d), and negative, e), images transformed with $\alpha = 2$. Binary images: positive, f), and negative, g), images transformed with $\alpha = \infty$. (In all cases $m = 0.5$.)

% Img: the image in normalized values ([0,1])
% Type: ‘Gamma’ polynomial adjustment, ‘CStretch’ contrast stretch,
Gamma: For type the 'Gamma' is a scalar with the transformation factor. For type 'CStretch', Gamma=[m Alpha] with the threshold value m and the slope Alpha of the correction. Alpha can be equal to inf. For the type 'BinTrans' is a 2x2 matrix with [rmin rmax; sL sH], i.e., the range and the binary values. For the type 'LevBoost' is a vector of 1x3: [rmin rmax sH], i.e., the range and the value at the boosted range
Inv: create the negative of the image (optional). 0: positive image. 1: negative image.

Output:

```
switch Type
    case 'Gamma'
        % If inversion parameter is not used
        if (nargin==3)
            ImgT=Img.^Gamma;
        end
        % If inversion parameter is used
        if (nargin==4)
            ImgT=(1*Inv+(-1)^Inv*Img).^Gamma;
        end
    case 'CStretch'
        m=Gamma(1); Alpha=Gamma(2);
        % If inversion parameter is not used
        if (nargin==3)
            if isnfinite(Alpha)
                ImgT=1./(1+(m./(Img+eps)).^Alpha);
            else % infinity slope
                ImgT=double(Img>=m);
            end
        end
        % If inversion parameter is used
        if (nargin==4)
            if isnfinite(Alpha)
                ImgT=1./(1+(m./(1*Inv+(-1)^Inv*Img+eps)).^Alpha);
            else % infinity slope
                if Inv==0
                    %
                end
            end
end
```

4
图像分割

2 Gray-level slicing

The function **IntensityTransformation** presented above implements also the gray-level slicing transformations. In Fig. 3 we show the results obtained after applying the required transformations to a sample image.

3 Bit-level slicing

Bit-level slicing is a technique used to partition the intensities of an image into $n$ levels that are not gray-levels. The bit-level slicing technique highlights the contribution made by specific bits to the total image intensity. It can be said that only the five most significant bits (MSB) contain visually significant data while the three less significant bit (LSB) planes contribute the more subtle details.

Recall from binary arithmetics that shifting $i$ bits to the left (right) and padding
with zeros is equivalent to multiply (divide) by $2^i$. If we set to zero the lower order bit planes then we are setting to zero the LSB of an 8-bit binary number. Therefore, we are generating integer numbers that are multiples of $2^i$. So, we are representing all the integer values in the range $[n \times 2^i, (n + 1) \times 2^i - 1]$, $n = 0, 1, \ldots, 2^{8-i} - 1$ by the number $n \times 2^i$. You can think of the situation as quantizing the intensity values using $2^{8-i}$ bits because in terms of the gray-levels we reduce the number of intensities from $2^8$ to $2^{8-i}$.

Therefore, the histogram will reduce the number of intensity levels and since the size of the image does not change, then the histogram will show an increment in the number of pixels at the intensities $n \times 2^i$, $n = 0, 1, \ldots, 2^{8-i} - 1$. Note that the proposed change in the bits will produce, in general, an image with less contrast.

Setting to zero the higher order bit planes will reduce also the number of gray-levels from $2^8$ to $2^{8-i}$ but at the same time the change will clip to $2^{8-i} - 1$ all those pixels values that are bigger of equal to $2^{8-i} - 1$. In terms of the histogram, it will remain the same as in the original image for all those values below $2^{8-i} - 1$ and the number of pixels with intensity $2^{8-i} - 1$ will, in general, increase because the size of the image remains the
Figure 4: Bit-level slicing transformations. a) Effect of setting the \( i = 3 \) LSB of the intensity values of the image to zero. b) Effect of setting the \( i = 3 \) MSB of the intensity values of the image to zero. c) Effect of inverting the \( i = 3 \) MSB of the intensity values of the image.

Finally, if we invert the higher order bit planes then the result is that we are interchanging all the pixels values between one bit plane and its complement. If we invert the first \( i \) MSB, then all the intensity values that belong to the interval \( [n \cdot 2^{8-i}, (n+1) \cdot 2^{8-i} - 1] \), \( n = 0, 1, \ldots, 2^i - 1 \) will be mapped to the interval \( [k \cdot 2^{8-i}, (k+1) \cdot 2^{8-i} - 1] \), \( k = 2^i - 1 - n \) and vice versa. Note that when we maintain the \( 2^{8-i} \) LSB unchanged then all the intensity values keep the same positions within the mapped interval. Hence, the number of
intensities the number will not change, however the histogram will look reflected if we look at the level of the intervals determined by high-order bit planes, but it will remain unchanged within the intervals.

The following code implements the three bit-plane operations commented above. In Fig. 4 the results of the processing are shown for the case of modifying $i = 3$ bits. It is easy to see that the mask $\text{uint8(bin2dec('00001000')*ones(M,N))}$ in conjunction with the and operation will activate the third order bit-plane of an image.

```matlab
clear all; clc;
Img=imread('IRImage2.gif');
N=size(Img,2); M=size(Img,1);

% Set the 3 LSB to zero
Mask=uint8(bin2dec('11111000')*ones(M,N));
ImgT=bitand(Img,Mask);

% Set the 3 MSB to zero
Mask=uint8(bin2dec('00011111')*ones(M,N));
ImgT=bitand(Img,Mask);

% Invert the 3 MSB
ImgT=Img;
for i=1:M
    for j=1:N
        for k=6:8
            % Get the 3 MSB and negate them
            ImgT(i,j)=bitset(ImgT(i,j),k,not(bitget(ImgT(i,j),k)));
        end
    end
end
```

4 Histogram equalization and histogram specification

1. Histogram equalization is a mapping or redistribution of the histogram components on a new intensity scale. To obtain an uniform histogram it is required that pixel intensities mapped form, say, $L$ groups of $N/L$ pixels per group, where $L$ is the number of allowed discrete gray-levels and $N$ is the total number of pixels of the
image. However, the histogram equalization method does not change the number of pixels per gray-level, it simply reallocate the intensity values and eliminate the gaps between the intensities. Therefore, the resulting image will not have an uniform histogram unless the original image had one.

2. The histogram-equalization transformation is defined as:

\[
T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k-1} \frac{n_j}{n} + \frac{n_k}{n} = s_{k-1} + \frac{n_k}{n}, \quad k = 1, \ldots, L-1, \tag{1}
\]

where, \(s_0 = \frac{n_0}{n}\), \(L\) is the number of gray-levels, \(n\) is the total number of pixels in the image, and \(n_k\) is the number of pixels having the gray level \(r_k\).

Condition (i): Clearly, \(T(r_k)\) (see the expression at the RHS of (1)) is monotonically increasing in the discrete interval \([0, 1]\). To see that \(T(r_k)\) is single-valued consider the following two cases: (a) \(n_k > 0 \ \forall k\), then clearly from the expression on the RHS of (1) \(T\) is single-valued; (b) if \(n_k = 0\) for some \(k\)'s then consider \(T^*(r_k) = T(r_k)|_{n_k \neq 0}\), i.e., the restriction of \(T(r_k)\) to all those intensity values in \([0, 1]\) that actually appear in the image. Hence, for the restriction \(T^*(r_k)\) we have \(n_k > 0 \ \forall k\) so \(T^*(r_k)\) is single-valued.

Condition (ii): it is easy to see from the definition of \(T(r_k)\) and from its domain that \(0 \leq T(r_k)\). Recalling that \(\sum_{j=0}^{L-1} n_j = n\) we have that \(0 \leq T(r_k) \leq 1\). So, \(T(r_k)\) satisfies conditions (i) and (ii).

From the previous analysis we conclude the if all the gray-levels are present in the image then the direct mapping is strictly increasing. Therefore, the inverse mapping is single-valued and is monotonically increasing as well. Now, given that the original domain considers \(L\) gray-levels in the range \([0, 1]\) and the mapped intensities belong also to \([0, 1]\) then \(T^{-1}(r_k)\) inverse maps its domain into the range \([0, 1]\) as well. So, \(T(r_k)\) satisfies conditions (i) and (ii).

Histogram equalization is not a point processing operation because the mapping of every pixel intensity into another pixel intensity involves the cummulated sum of pixel intensities, therefore, there is in general some interaction between the pixels.
3. The code given in the assignment implements equation (1) in the following steps: (i) it creates the normalized histogram of the image (hist1); (ii) it produces the cumulated sum (CDF1) to generate the summation in (1), note that in this step if some of the intensities is missing in the image, then they are not mapped into the range; and (iii) it assigns the new intensities.

4. The following code is an example on how to perform the histogram specification.

```matlab
clear all; close all; 
img1 = imread('phobos.jpg');

%Transform to Uniform Distribution 
[hist1, bins1] = hist(double(img1(:)),0:255); 
hist1 = hist1./length(img1(:)); 
T = cumsum(hist1);
img1eq = zeros(size(img1)); 
for i=0:255 
    img1eq(find(img1==i)) = T(i+1);
end 
[hist1eq, bins1eq] = hist(double(255*img1eq(:)),0:255); 
hist1eq = hist1eq./length(img1(:)); S = cumsum(hist1eq);

%Specify New Histogram 
load HistogramV7.mat

%Compute New CDF’s from Specified Histogram (Iterative) 
G = (cumsum(z)/length(img1(:))); 
Ginv = zeros(size(G)); 
for k=1:256 
    dff = -1; m = 0; 
    while(dff < 0) 
        m = m+1; dff = G(m) - S(k); 
    end 
    Ginv(k) = m-1; 
end 

img1mt = zeros(size(img1eq)); ieq = floor(255*img1eq); 
for i=0:255 
```

10
\[ \text{img1mt(find(ieq==i)) = Ginv(i+1);} \]

\[ \text{end} \]

\[ [\text{hist1mt, bins1mt}] = \text{hist(double(img1mt(:)),0:255);} \]

\[ \text{img1mt = img1mt/255;} \]

The main difference between histogram specification and histogram equalization is that in histogram specification one can freely specify the shape of the histogram for the processed image, while the histogram equalization seeks to produce a processed image with an uniform histogram.

5 Spatial filtering

Table 1: Summary of the performance metrics achieved by the LF and the MF for different mask sizes. The original image in Fig. 5a) was corrupted with Gaussian noise (GN) of zero mean and variance 0.01. In addition the image was corrupted with Salt & Pepper noise (SPN) affecting the 5% of the pixels.

<table>
<thead>
<tr>
<th>Metric</th>
<th>3 × 3 GN</th>
<th>9 × 9 GN</th>
<th>15 × 15 GN</th>
<th>3 × 3 SPN</th>
<th>9 × 9 SPN</th>
<th>15 × 15 SPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>( CPU^{LF} )</td>
<td>0.0700</td>
<td>0.0200</td>
<td>0.0200</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0.0200</td>
</tr>
<tr>
<td>( CPU^{MF} )</td>
<td>0.0800</td>
<td>0.2500</td>
<td>0.6400</td>
<td>0.0300</td>
<td>0.2200</td>
<td>0.5500</td>
</tr>
<tr>
<td>( MSE^{LF} )</td>
<td>0.0019</td>
<td>0.0033</td>
<td>0.0056</td>
<td>0.0025</td>
<td>0.0035</td>
<td>0.0060</td>
</tr>
<tr>
<td>( MSE^{MF} )</td>
<td>0.0019</td>
<td>0.0014</td>
<td>0.0028</td>
<td>0.0002</td>
<td>0.0008</td>
<td>0.0020</td>
</tr>
<tr>
<td>( \rho^{LF} )</td>
<td>0.1020</td>
<td>0.0399</td>
<td>0.0314</td>
<td>0.0909</td>
<td>0.0403</td>
<td>0.0312</td>
</tr>
<tr>
<td>( \rho^{MF} )</td>
<td>0.1137</td>
<td>0.0407</td>
<td>0.0311</td>
<td>0.0391</td>
<td>0.0300</td>
<td>0.0267</td>
</tr>
<tr>
<td>( Q^{LF} )</td>
<td>0.3931</td>
<td>0.5527</td>
<td>0.4902</td>
<td>0.3629</td>
<td>0.5256</td>
<td>0.4788</td>
</tr>
<tr>
<td>( Q^{MF} )</td>
<td>0.3413</td>
<td>0.5276</td>
<td>0.4994</td>
<td>0.8997</td>
<td>0.7484</td>
<td>0.6059</td>
</tr>
</tbody>
</table>

To prove that a median filter (MF) is nonlinear consider to images \( I_1(x, y) \) and \( I_2(x, y) \). If we apply a median filter to each image we obtain: \( O_1(x, y)\{I_1\} \) and \( O_2(x, y)\{I_2\} \). Now, consider the combined image \( I(x, y) = I_1(x, y) + I_2(x, y) \) and apply a median filter to it: \( O(x, y)\{I\} = O(x, y)\{I_1 + I_2\} \neq O(x, y)\{I_1\} + O_2(x, y)\{I_2\} \) because given an \( N \times N \) mask we have to sort (rank) the intensities of the pixels within the mask and pick as the output of the \( (x, y) \) pixels the intensity value located in the middle of the rank. So, if
Figure 5: Evaluation of the performance of the LF and the MF for different mask sizes. a) Image corrupted with Gaussian noise (GN) of zero mean and variance 0.01. a) Image corrupted with Salt & Pepper noise (SPN) affecting the 5% of the pixels.
we add the images first then the ranking operation will not produce necessarily the same intensity value in the middle of the rank as the sum of the intensity values obtained after ranking individually the masks. So, superposition does not hold and the MF is nonlinear.

We tested and compared the performance of the MF and an average spatial linear filter (LF) under two scenarios. In Fig. 5 we show the filtered images obtained by each filter, and in Table 1 we list the summary of the performance metrics achieved by them. Clearly, the MF takes a larger time than the LF to produce the result. Such situation is expected due to the complexity of the ranking operation. In addition, the results obtained show that for the evaluated image the MF outperforms the LF in almost every performance metric but the computing time. Finally, a visual evaluation also coincides with the objective results.

The following MATLAB code is an example of the required implementation.

clear all; clc; kf=1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Read image and calculate its FFT-2D
% The images available are: IRImage.gif, IRImage2.gif, and IRImage3.gif
Img2Use=2; % Select the image to use

switch Img2Use
    case 1
        Img=double(imread('IRImage1.gif'))/255;
    case 2
        Img=double(imread('IRImage2.gif'))/255;
    case 3
        Img=double(imread('IRImage3.gif'))/255;
end

N=size(Img,2); M=size(Img,1);

for i=1:2
    switch i
        case 1
            ImgN=imnoise(Img,'Gaussian',0,0.01);
        case 2
            ImgN=imnoise(Img,'salt & pepper');
    end
Nv=[3 9 15];
for k=1:length(Nv)
    % Compute the cputime
    t=cputime;
    ImgLF=filter2(fspecial('average',Nv(k)),ImgN);
    DtLF(i,k)=cputime-t;
    % Compute performance metrics
    MSELF(i,k)=mean2((Img-ImgLF).^2);
    RhoLF(i,k)=Roughness(ImgLF);
    QLF(i,k)=Qindex(Img,ImgLF);
    % Compute the cputime
    t=cputime;
    ImgMF=medfilt2(ImgN,[Nv(k) Nv(k)])
    DtMF(i,k)=cputime-t;
    % Compute performance metrics
    MSEMF(i,k)=mean2((Img-ImgMF).^2);
    RhoMF(i,k)=Roughness(ImgMF);
    QMF(i,k)=Qindex(Img,ImgMF);
end
end
[ DtLF(1,:) DtLF(2,:); DtMF(1,:) DtMF(2,:); ... 
 MSELF(1,:) MSELF(2,:); MSEMF(1,:) MSEMF(2,:); ... 
 RhoLF(1,:) RhoLF(2,:); RhoMF(1,:) RhoMF(2,:); ... 
 QLF(1,:) QLF(2,:); QMF(1,:) QMF(2,:)]

6 Laplacian spatial filtering

The following MATLAB code is an example of the required implementation.

clear all; close all;

% Laplacian Filtering
img1 = im2double(imread('moon.jpg'));
lap = [1 1 1; 1 -8 1; 1 1 1];
img2 = conv2(img1, lap, 'same');
img3 = [img2 - min(img2(:))]./max(img2(:) - min(img2(:)));
img4 = img1 - img2;

14
% Laplacian Filtering Comparison
img1 = im2double(imread('SEM.jpg'));
lapa = [0 -1 0; -1 5 -1; 0 -1 0];
lapb = [-1 -1 -1; -1 9 -1; -1 -1 -1];
img2 = conv2(img1, lapa, 'same');
img3 = conv2(img1, lapb, 'same');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Laplacian Enhancement
img1 = im2double(imread('SEM_dark.jpg'));
lapa = [-1 -1 -1; -1 8 -1; -1 -1 -1];
lapb = [-1 -1 -1; -1 9 -1; -1 -1 -1];
lapc = [-1 -1 -1; -1 9.7 -1; -1 -1 -1];
img2 = conv2(img1, lapa, 'same');
img2 = [img2 - min(img2(:))]./ max(img2(:) - min(img2(:)));
img3 = conv2(img1, lapb, 'same');
img4 = conv2(img1, lapc, 'same');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sobel Gradient Filtering
img1 = im2double(imread('contact.jpg'));
soba = [-1 -2 -1; 0 0 0; 1 2 1];
sobb = [-1 0 1; -2 0 2; -1 0 1];
imga = abs(conv2(img1, soba, 'same'));
imgb = abs(conv2(img1, sobb, 'same'));
img2 = imga + imgb;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Combining Spatial Enhancements
lap = [-1 -1 -1; -1 8 -1; -1 -1 -1];
soba = [-1 -2 -1; 0 0 0; 1 2 1];
sobb = [-1 0 1; -2 0 2; -1 0 1];
h = ones(5,5)/25;
img1 = im2double(imread('bonescan.jpg'));
img2 = conv2(img1, lap, 'same');
img3 = img1 + img2;
img4 = abs(conv2(img1, soba, 'same')) + abs(conv2(img1, sobb, 'same'));
img5 = conv2(img4, h, 'same');
img6 = img3.*img5;
img7 = img1 + img6;
img8 = img7.^0.5;
7 Change detection using image subtraction

![Image Subtraction Example](image.png)

Figure 6: Detection of movement by subtracting images.

The following MATLAB code is an example of the required implementation. In Fig. 6 we show the resulting images.

```matlab
img2 = (img2 - min(img2(:))) ./ max(img2(:) - min(img2(:)));

clear all; close all;
ima = im2double(imread('scene1.jpg'));
imb = im2double(imread('scene2.jpg'));
img1 = ima(10:size(ima,1)-9, 10:size(imb,2)-9);
img2 = imb(10:size(ima,1)-9, 10:size(imb,2)-9);

dff = abs(img1 - img2); idx = find(dff>.2);
mask = zeros(size(dff)); mask(idx) = 1;
red = img1; green = img1; blue = img1;
red(idx) = 1; green(idx) = 0; blue(idx) = 0;
img3 = cat(3, cat(3,red,green), blue);

figure;
```
subplot(2,2,1); imshow(img1); title('Scene 1');
subplot(2,2,2); imshow(img2); title('Scene 2');
subplot(2,2,3); imshow(mask); title('Difference Mask');
subplot(2,2,4); imshow(img3); title('Detected Motion');