C4 TĂNG CƯỜNG ẢNH TRONG KHÔNG GIAN

Spatial domain Image enhancement

Ý NGHĨA

- Tăng cường ảnh là làm cho ảnh thích hợp hơn cho việc xử lý tiếp theo, ví dụ như thay dổi dộ tuong phản, thay đổi màu sắc, cường dộ sáng, lọc nhiễu, nội suy, làm trơn ảnh.
- Có hai pp tăng cường ảnh: trong không gian và trong miền tần số
- Pp không gian xử lý các pixel, xử lý cường độ sáng tại mỗi điểm (x,y)

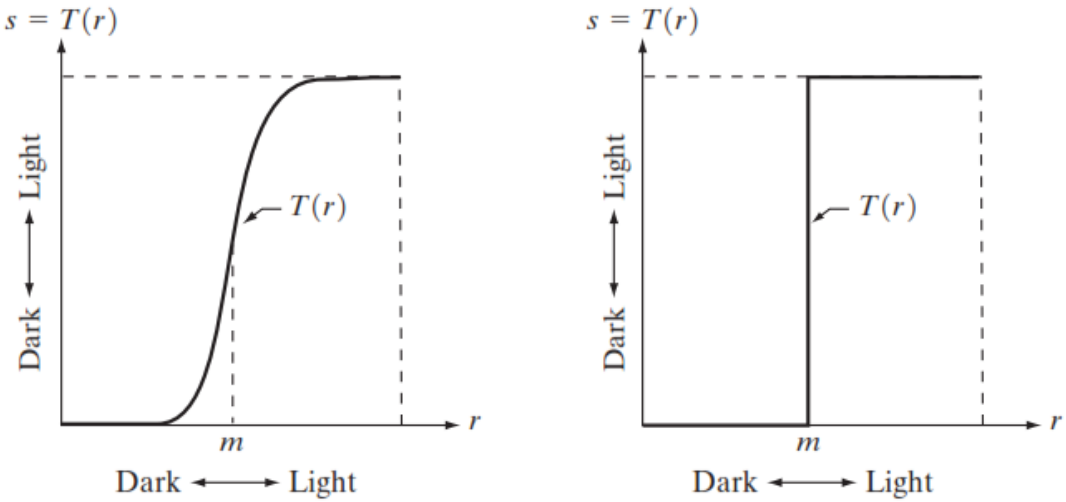
$$g(x,y)=T(f(x,y))$$

f là cường độ sáng điểm gốc, g là cường độ sáng sau phép biến đổi T

T có thể là tuyến tính, phi tuyến hay tuyến tính từng đoạn

ĐiỀU CHỈNH CƯỜNG ĐỘ SÁNG

- Đơn giản nhất là chỉnh độ tương phản (CONTRAST), giảm cường độ sáng khi dưới mức m và tăng khi trên mức m



• Nếu gọi r cường độ sáng ảnh vào và s cường độ sáng ảnh ra, ta có biểu thức

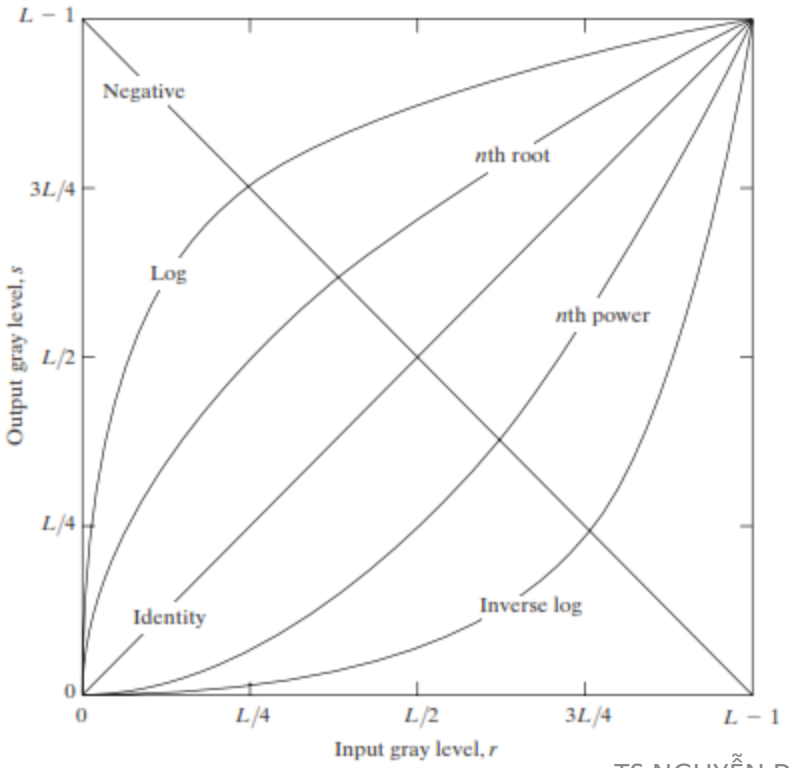
$$s = \frac{1}{1+\left(\frac{m}{r}\right)^{E}}$$

ĐIỀU CHỈNH CƯỜNG ĐỘ SÁNG

- Trong trường hợp phân ngưỡng nhị phân:

  S=1 nếu r>=m; s=0 nếu r<m

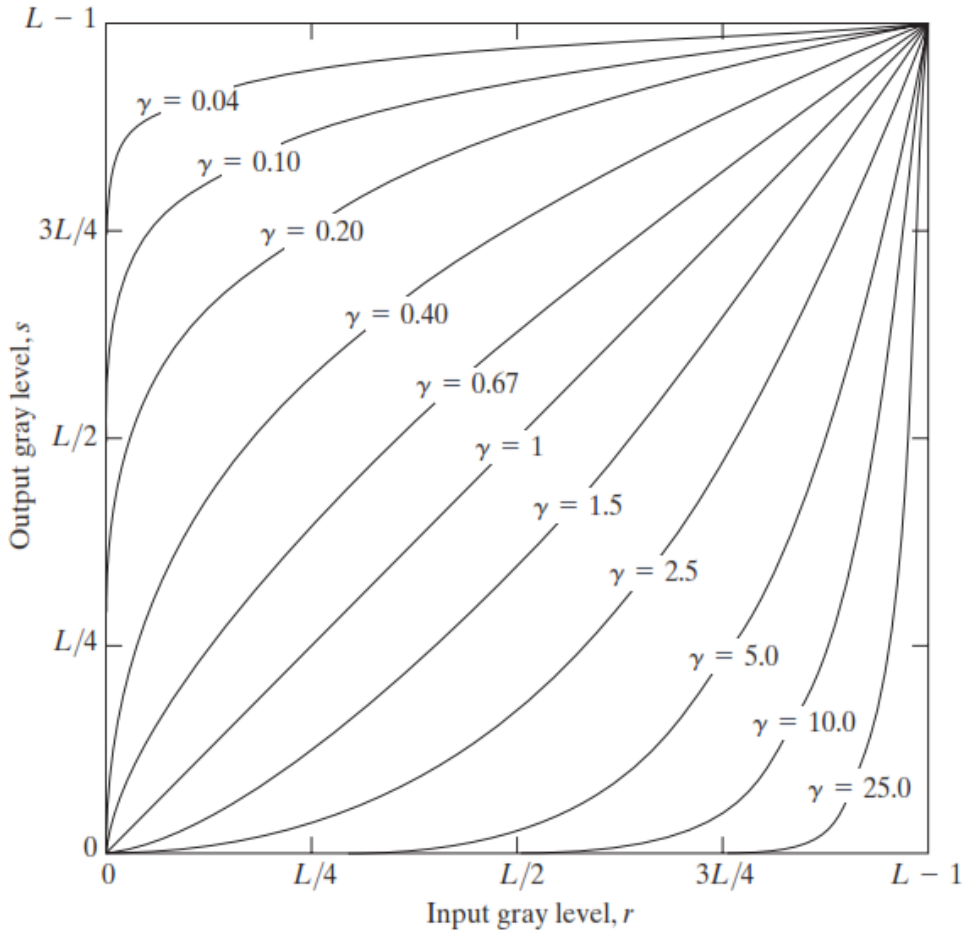
- Lấy âm ảnh: ảnh có vùng tối rộng thường được lấy âm bản để dễ xử lý, giả sử cường độ từ 0 đến L, ta có s=L-r
- Phép biến đổi log : s=c log(1+r) tăng mức tối và giảm mức sáng



- Phép biến đổi lũy thừa còn gọi là hiệu chỉnh gamma

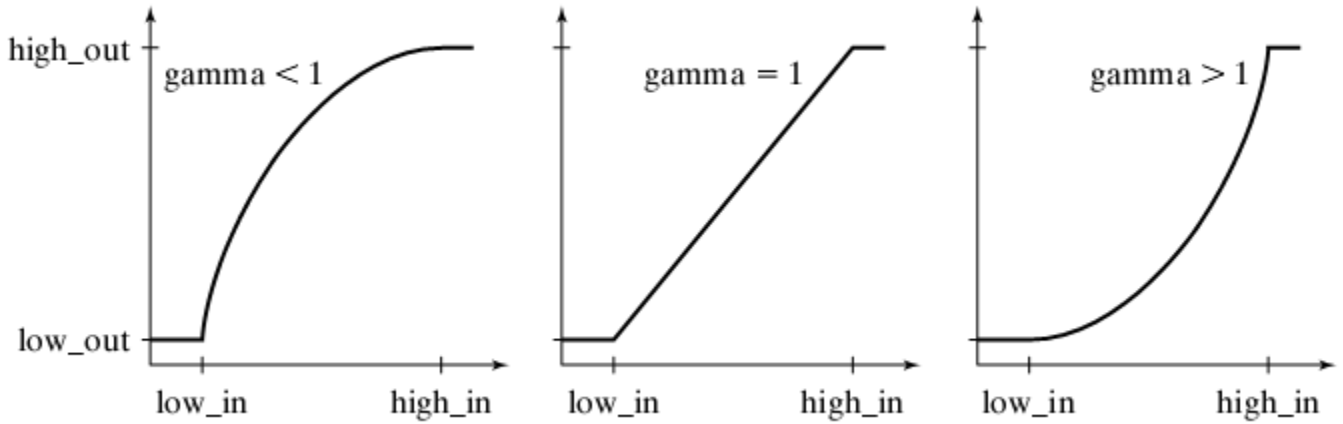
  $s=cr^{\gamma}$

ĐIỀU CHỈNH CƯỜNG ĐỘ SÁNG



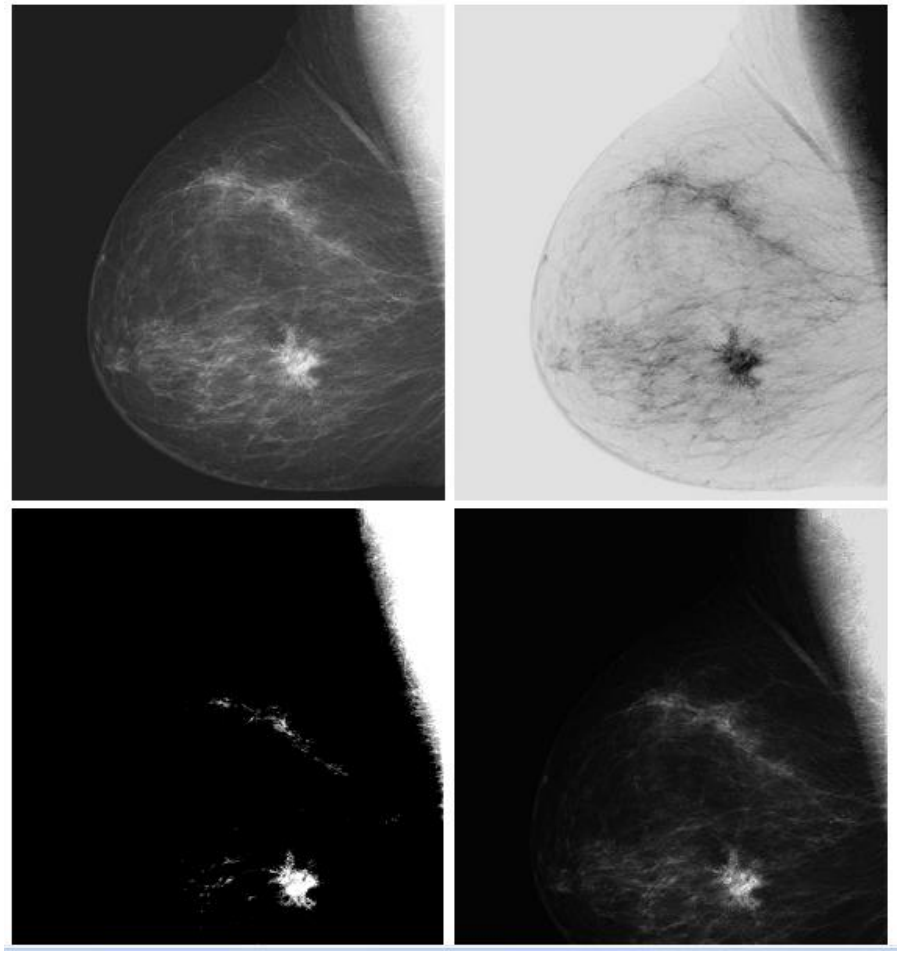
HÀM MATLAB

- g = imadjust(f, [low_in high_in], [low_out high_out], gamma)



g = imcomplement(f); // lấy âm bản

g1 = imadjust(f, [0 1], [1 0]);// lấy âm bản



a: ảnh gốc, b: âm bản, c: khuếch đại vùng 0.5 đến 0.75, d: gamma=2

```
I = imread('pout.tif');
J = imadjust(I);
figure, imshow(I), figure, imshow(J)

K = imadjust(I,[0.3 0.7],[]);
figure, imshow(K)

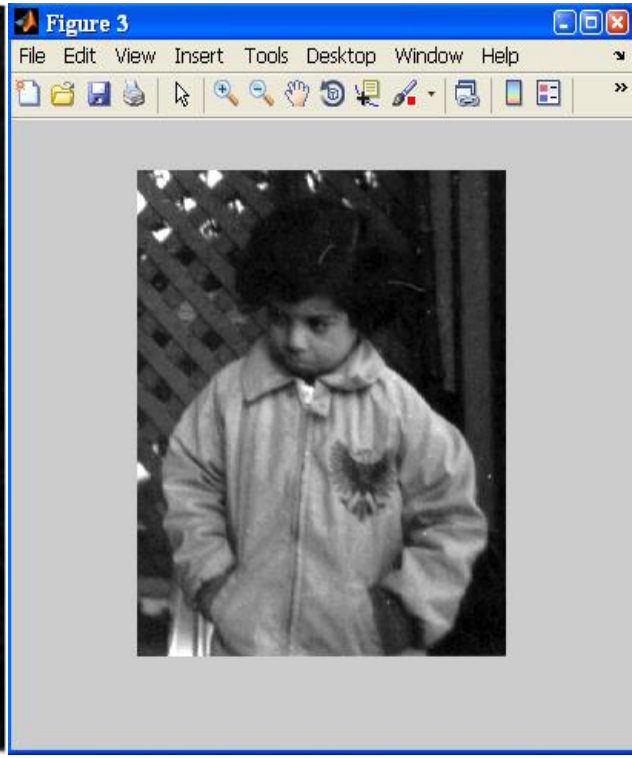
RGB1 = imread('football.jpg');
RGB2 = imadjust(RGB1,[.2 .3 0; .6 .7 1],[]);
figure, imshow(RGB1), figure, imshow(RGB2)
```

- J = imadjust(I) maps the values in intensity image I to new values in J such that 1% of data is saturated at low and high intensities of I. This increases the contrast of the output image J.
- J = imadjust(I,[LOW_IN; HIGH_IN],[LOW_OUT; HIGH_OUT],GAMMA) maps the values of I to new values in J as described in the previous syntax.

GAMMA specifies the shape of the curve describing the relationship between the values in I and J. If GAMMA is less than 1, the mapping is weighted toward higher (brighter) output values. If GAMMA is greater than 1, the mapping is weighted toward lower (darker) output values. If you omit the argument, GAMMA defaults to 1 (linear mapping).

- RGB2 = imadjust(RGB1,...) performs the adjustment on each image plane (red, green, and blue) of the RGB image RGB1. As with the colormap adjustment, you can apply unique mappings to each plane.

ĐIỀU CHỈNH CƯỜNG ĐỘ SÁNG



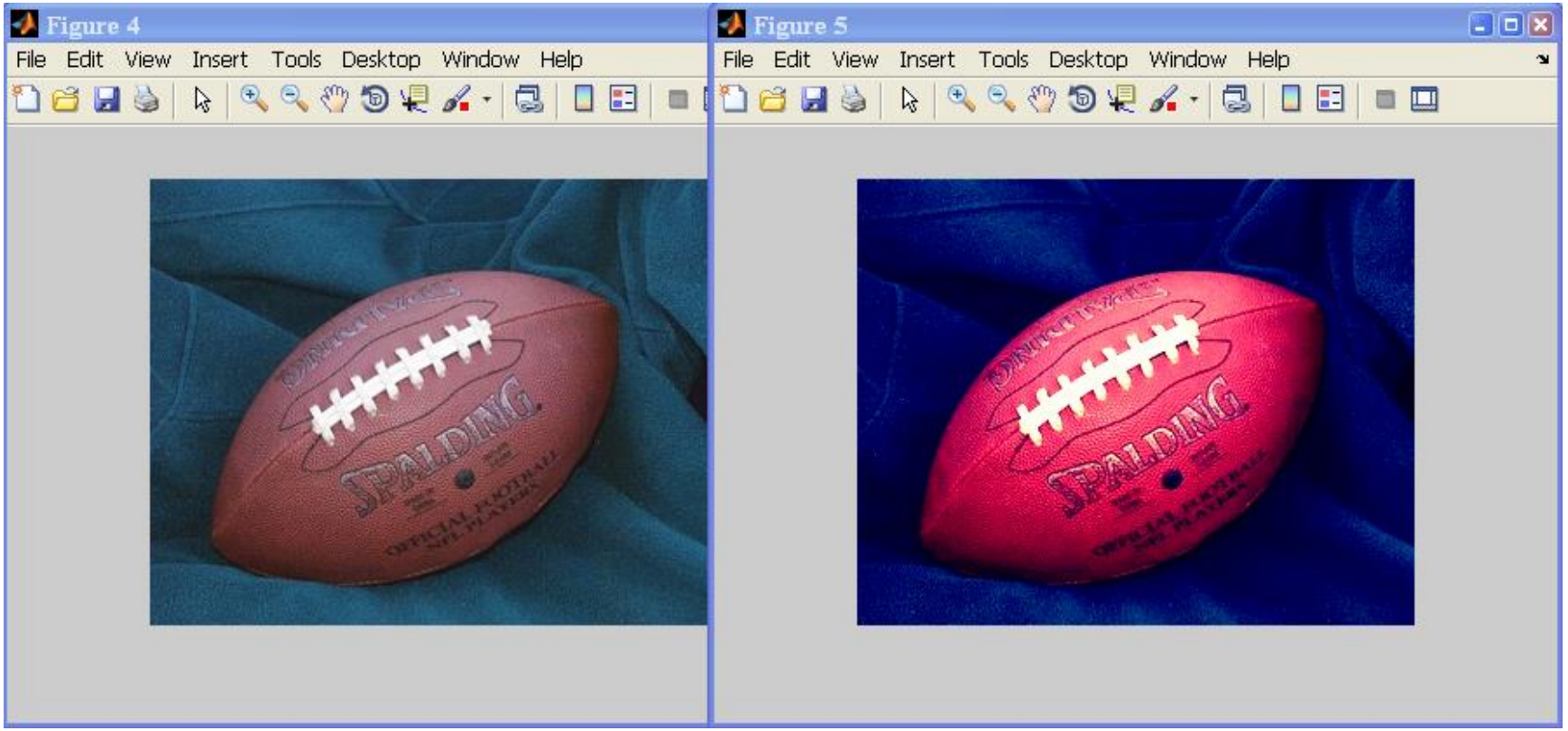
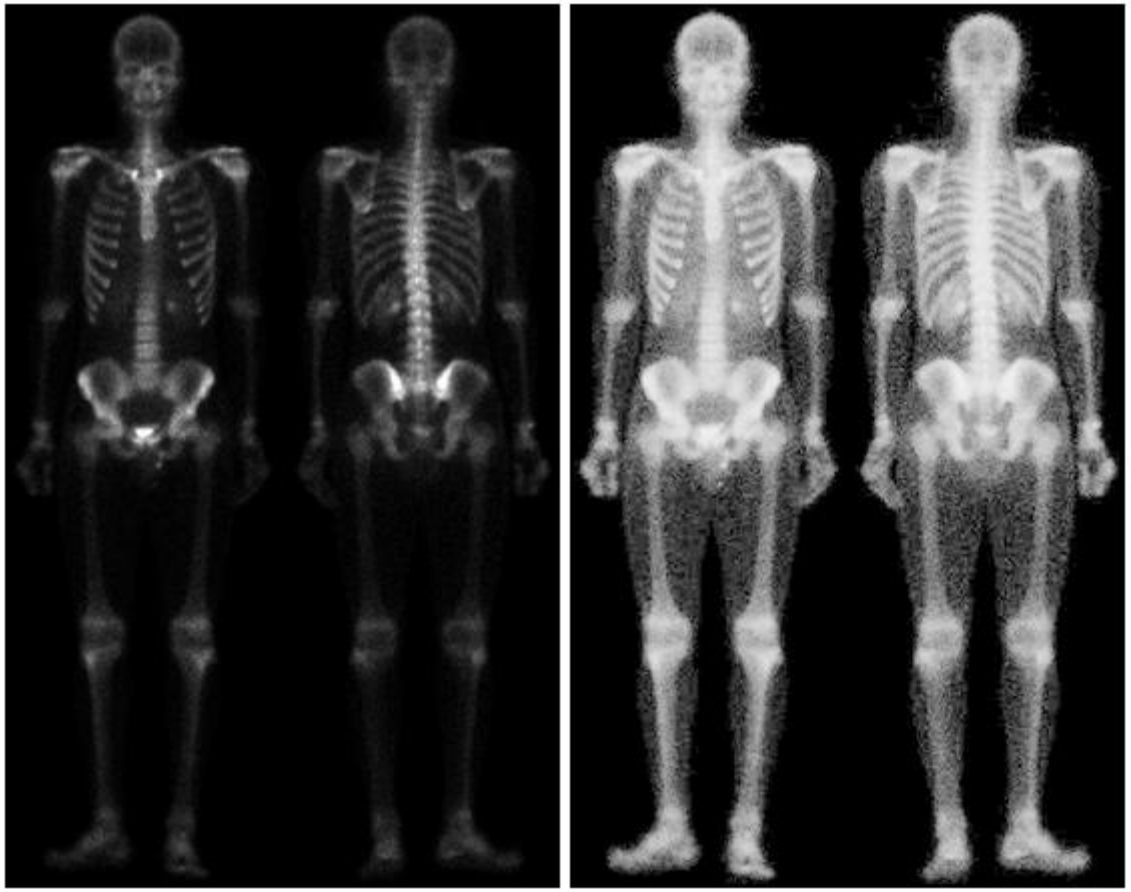
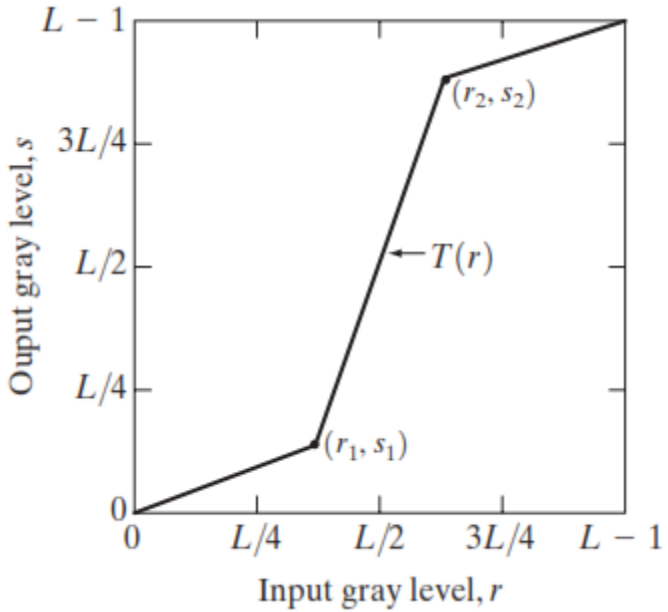


Figure 4
File Edit View Insert Tools Desktop Window Help
Figure 5
File Edit View Insert Tools Desktop Window Help



ĐIỀU CHỈNH CƯỜNG ĐỘ SÁNG

- Điều chỉnh tuyến tính từng đoạn



ĐIỀU CHỈNH ĐỘ SÁNG VÀ TƯƠNG PHẢN C

- Điều chỉnh cường độ sáng là cộng giá trị V của ảnh với hằng số b
- Điều chỉnh tương phản là nhân giá trị V với số c
- new_image(i,j)=c*org_image (I,j)+b
- Dùng hàm

Mat org_image; Mat new_image;

org_image.convertTo(new_image, -1, c, b);

C++: void Mat::convertTo(OutputArray m, int rtype=-1, double c=1, double b=0) const

- Nên đổi ảnh màu sang HSV và điều chỉnh kênh V rồi ghép trở lại

ĐIỀU CHỈNH ĐỘ SÁNG VÀ TƯƠNG PHẢN C

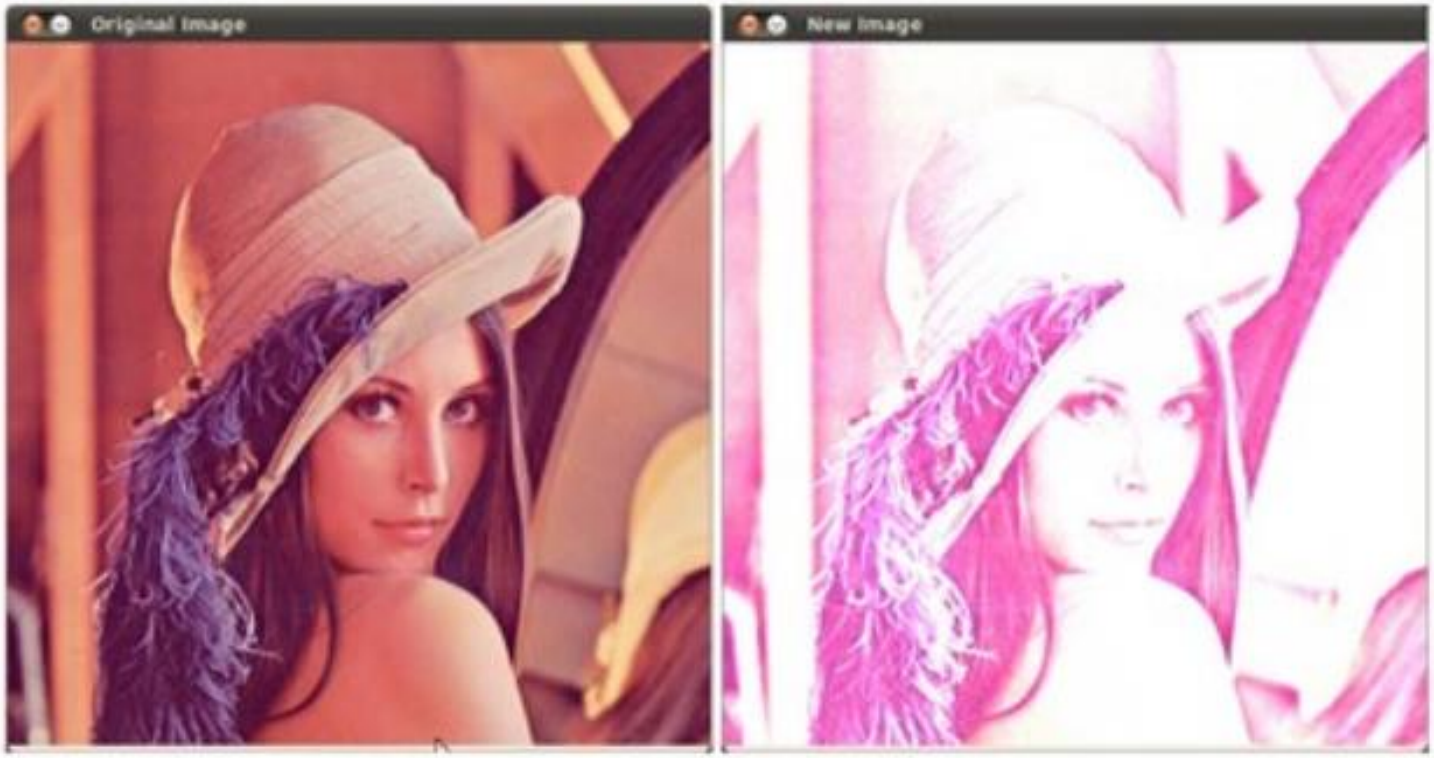
```
using namespace cv;
double alpha; /**< Simple contrast control */
int beta;  /**< Simple brightness control */
int main( int argc, char** argv )
{
Mat image = imread( argv[1] );
 Mat new_image = Mat::zeros( image.size(), image.type() );
 /// Initialize values
std::cout<<"* Enter the alpha value [1.0-3.0]:
";std::cin>>alpha;
 std::cout<<"* Enter the beta value [0-100]: ";
std::cin>>beta;
```

ĐIỀU CHỈNH ĐỘ SÁNG VÀ TƯƠNG PHẢN C

```
for( int y = 0; y < image.rows; y++ )
    { for( int x = 0; x < image.cols; x++ )
        { for( int c = 0; c < 3; c++ )
            {      new_image.at<Vec3b>(y,x)[c] =
        saturate_cast<uchar>( alpha*(
image.at<Vec3b>(y,x)[c] ) + beta );           }    }    }
 imshow("Original Image", image);
 imshow("New Image", new_image);
waitKey();
 return 0;
}
```

ĐIỀU CHỈNH ĐỘ SÁNG VÀ TƯƠNG PHẢN C

- c=2.2, b=50



HIỆU CHỈNH GAMMA C

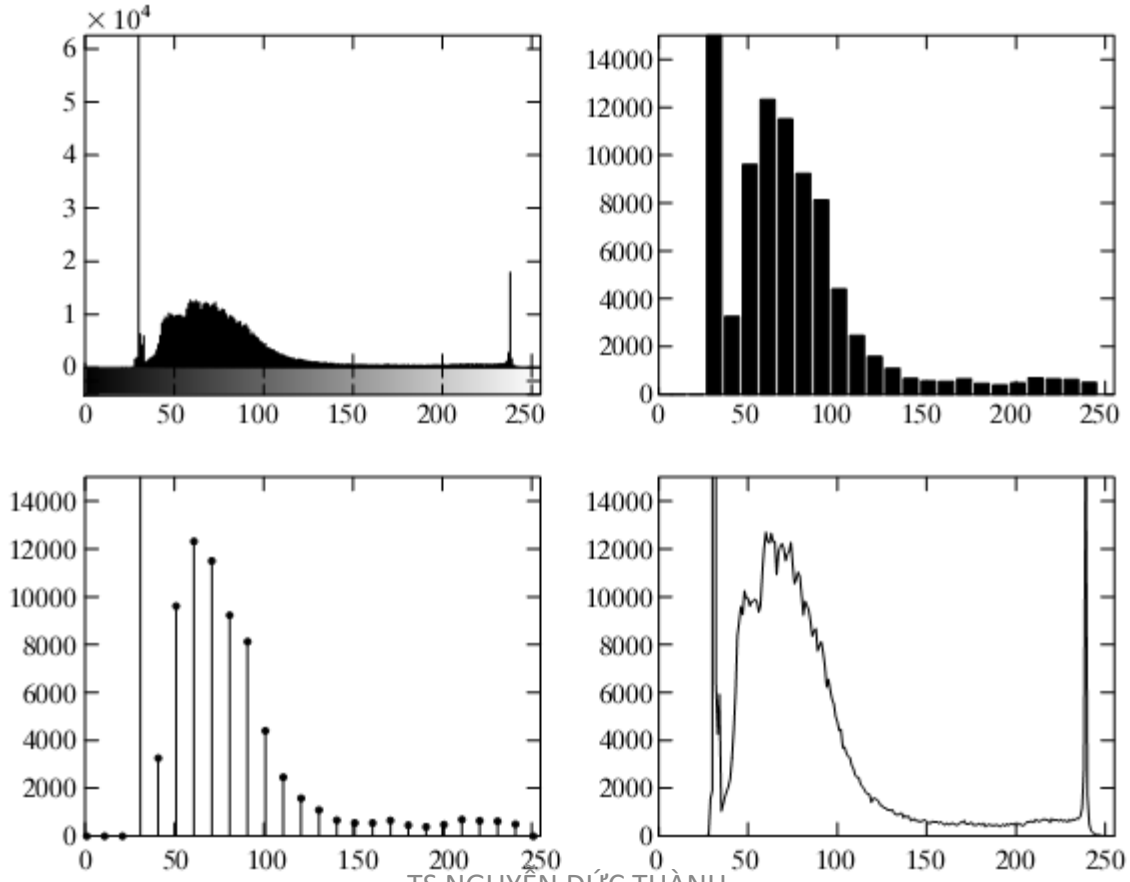
$$O = \left(\frac{I}{255}\right)^{\gamma} \times 255$$

```
Mat correctGamma( Mat& img, double gamma ) {
Mat lut_matrix(1, 256, CV_8UC1 );
 uchar * ptr = lut_matrix.ptr();
 for( int i = 0; i < 256; i++ )
 ptr[i] = p[i] = saturate_cast<uchar>(pow(i / 255.0, gamma_)
* 255.0);
  Mat result=img.clone();
 LUT( img, lut_matrix, result );
  return result;
}
```

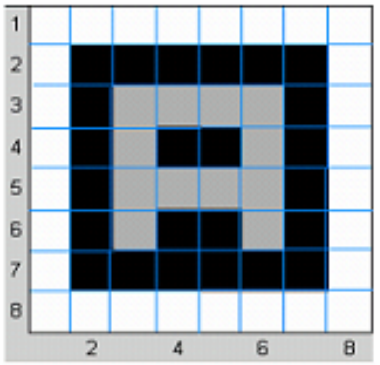
HIỆU CHỈNH GAMMA C

HISTOGRAM (LƯỢC ĐỒ XÁM, QUANG ĐỒ)

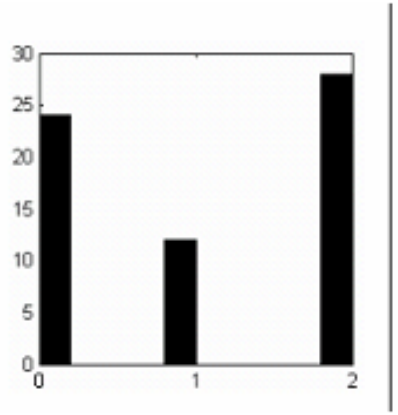
- Histogram biểu thị số lượng pixel có cường độ sáng trong một vùng nào đó, với 8 bit ta có 256 mức
- Histogram vẽ dưới dạng biểu đồ, trục ngang là cường độ, trục dọc là số pixel

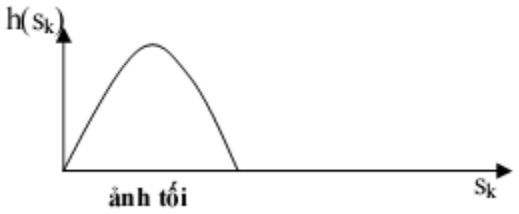


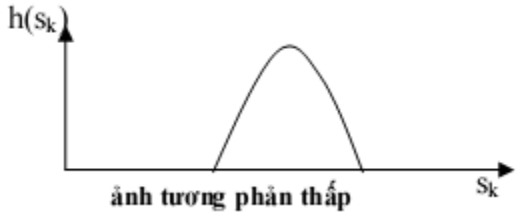
HISTOGRAM (LƯỢC ĐỒ XÁM)

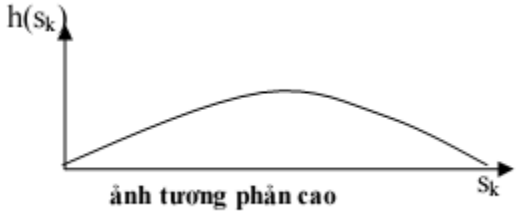


2	2	2	2	2	2	2	2
2	0	0	0	0	0	0	2
2	0	1	1	1	1	0	2
3	0	1	0	0	1	0	2
2	0	1	1	1	1	0	2
2	0	1	0	0	1	0	2
2	0	0	0	0	0	0	2
2	2	2	2	2	2	2	2









- Histogram cho biết chất lượng ảnh, nếu ảnh tối biểu đồ dời về bên trái, ảnh sáng biểu đồ dời về bên phải
- $h(r_k)=n_k$; n_k là số pixel của ảnh vào r ở múc cường độ k
- $p(r_k) = h(r_k)/n$ gọi là histogram chuẩn hóa hay xác suất
- h = imhist(f, b); //b: số vùng
- p = imhist(f, b)/numel(f); histogram chuẩn hóa

CÂN BẰNG HISTOGRAM (HISTOGRAM EQUALIZATION)

- Để cải thiện chất lượng ảnh ta cân bằng histogram, sao cho số điểm ảnh ở mỗi mức sáng tương đương nhau

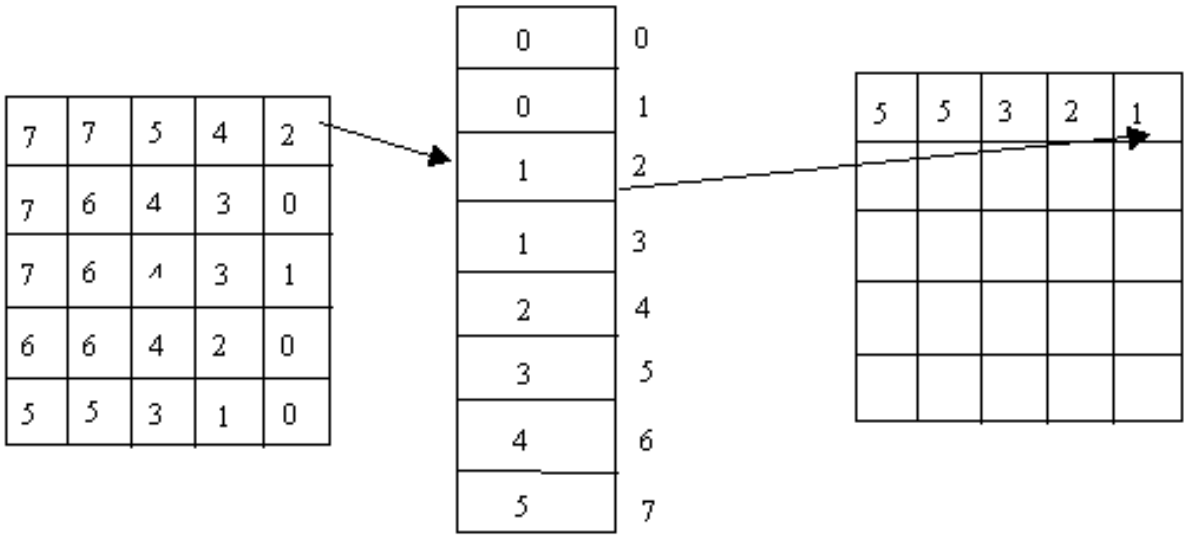
$$
\begin{aligned}
s_k &= T(r_k) \\
&= \sum_{j=1}^{k} p_r(r_j) \\
&= \sum_{j=1}^{k} \frac{n_j}{n}
\end{aligned}
$$

- Matlab: g = histeq(f, nlev); nlevel là số mức xám, thường là 256

THUẬT TOÁN

- Để đơn giản ta dùng phương pháp tra bảng Look up Table, Giả sử mức xám biểu thị bằng ba bit, ta dùng một bảng 8 phần tử, chỉ số là cường độ ảnh vào, giá trị phần tử là cường độ ảnh ra, ta đọc từng điểm ảnh, cường độ sáng là k (từ 0 đến 7) truy xuất đến phần tử thứ k của bảng, trong đó ta lấy ra giá trị cường độ của ảnh mới
- Để tính histogram cho mức xám 8 bit, ta lập bảng 256 phần tử, chỉ số từ 0 đến 255, giá trị ban đầu là 0
- Duyệt qua các pixel, cứ mỗi pixel cường độ k ta tăng thêm 1 giá trị phần tử chỉ số k
- Từ bảng trên ta lập bảng mới, giá trị phần tử k là tổng các giá trị của các phần tử bảng cũ từ 0 đến k
- Chuẩn hóa bảng mới bằng cách nhân cho 255/số điểm ảnh rồi đổ ra số nguyên từ 0 đến 255

TRA BẢNG



Ví dụ

Cân bằng histogram của ảnh S

$$S = \begin{vmatrix} 10 & 20 & 30 & 40 & 50 \\ 20 & 40 & 70 & 30 & 30 \\ 40 & 60 & 50 & 50 & 70 \\ 70 & 70 & 60 & 60 & 30 \\ 20 & 10 & 10 & 20 & 30 \end{vmatrix}$$

Xác định tần số mức xám

mức xám	10	20	30	40	50	60	70
tần số	3	4	5	3	3	3	4

Cân bằng

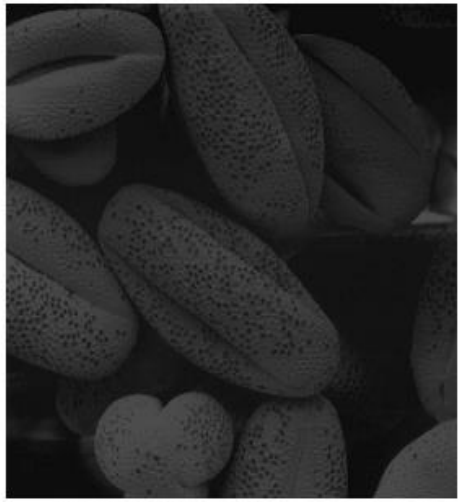
10	**20**	**30**	**40**	**50**	**60**	**70**
3	**7**	**12**	**15**	**18**	**21**	**25**

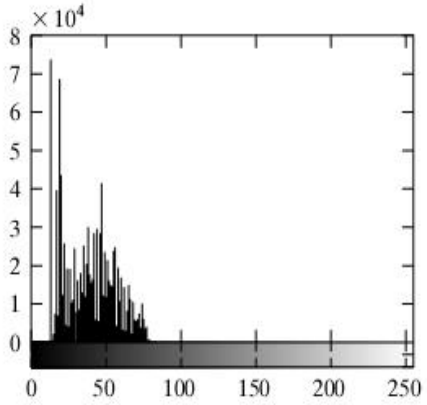
Chuẩn hóa: Nhân bảng với 255/25

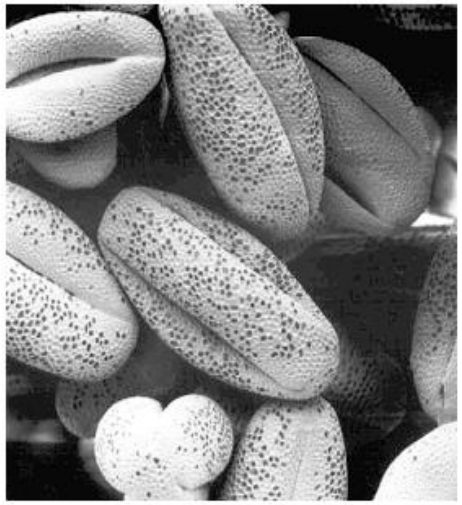
Mức xám s_{in}	10	20	30	40	50	60	70
Thay thế bởi s_{out}	31	72	122	153	184	214	255

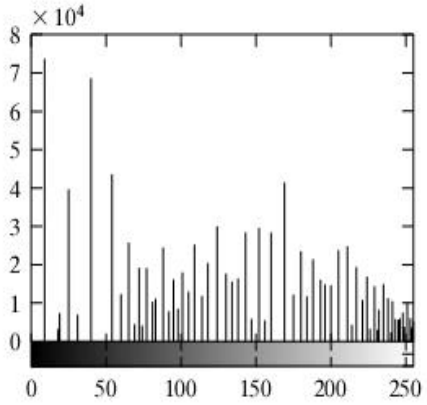
CÂN BẰNG HISTOGRAM

```
>>imshow(f)
>> figure, imhist(f)
>> ylim('auto')
>> g = histeq(f, 256);
>> figure, imshow(g)
>> figure, imhist(g)
>> ylim('auto')>
```





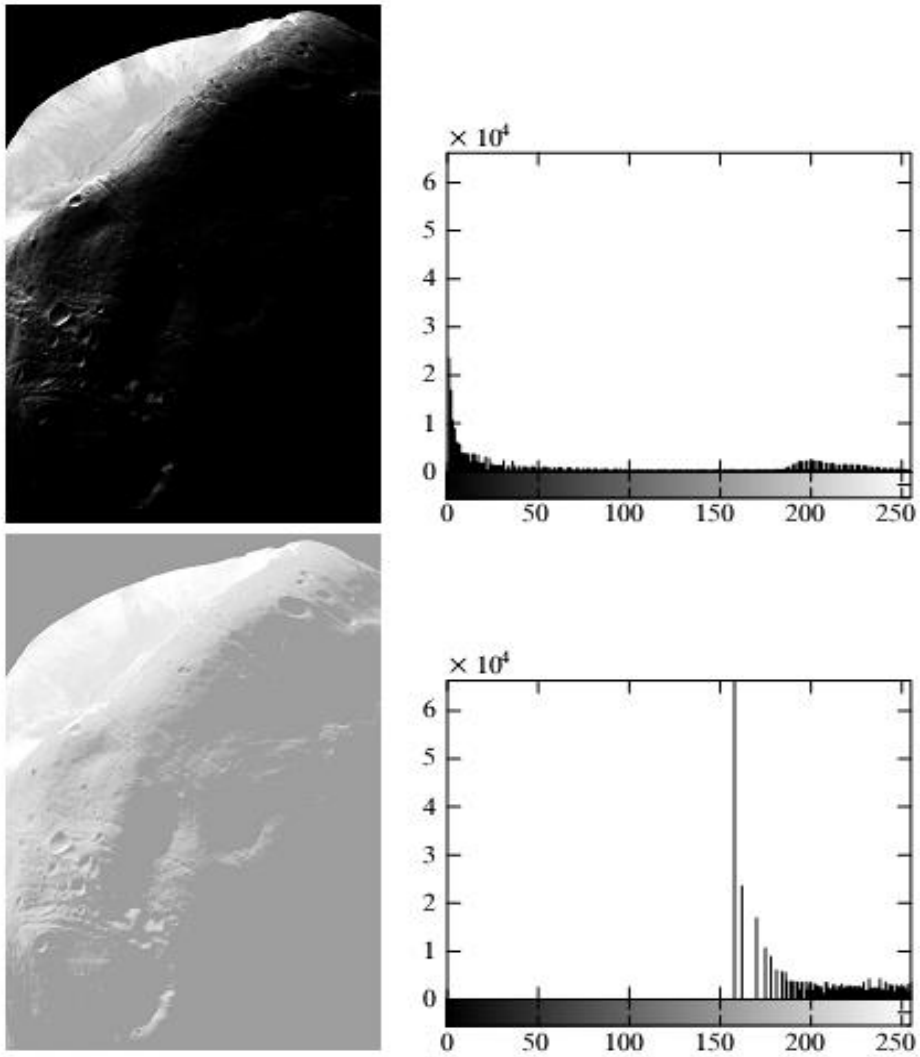




CÂN BẰNG HISTOGRAM

a b
c d

(a) Image of the Mars moon Phobos.
(b) Histogram.
(c) Histogram-equalized image.
(d) Histogram of (c).
(Original image courtesy of NASA).



HISTOGRAM MATCHING

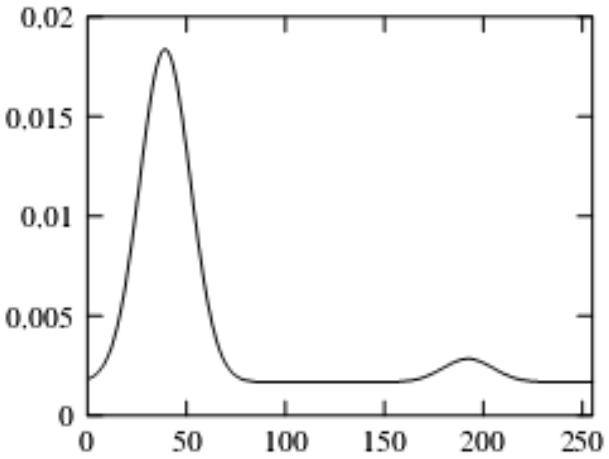
- Tạo một ảnh có histogram cho trước từ ảnh gốc
  - Gọi r là ảnh vào, s là ảnh cân bằng histogram và z là ảnh có được sau histogram matching,
  - Ta có s=T(r), s=G(z), suy ra G^{-1} [T(r)]
- Matlab: g = histeq(f, hspec), f là ảnh vào, hspec là histogram mong muốn, là vector hàng
- OpenCV: void cvEqualizeHist(

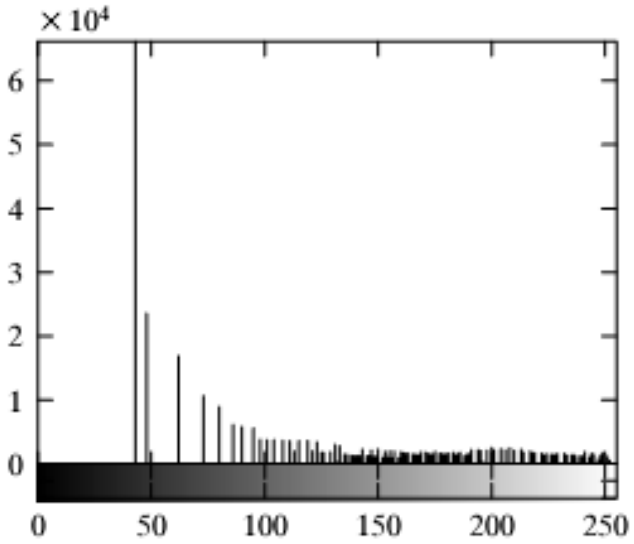
```
const CvArr* src,
CvArr*       dst
);
```

HISTOGRAM MATCHING

a b
c

(a) Specified histogram.
(b) Result of enhancement by histogram matching.
(c) Histogram of (b).





CÁC HÀM MATLAB TĂNG CƯỜNG ẢNH

Imadjust	Adjust image intensity values or colormap
Imcontrast	Adjust Contrast tool
Imsharpen	Sharpen image using unsharp masking
Histeq	Enhance contrast using histogram equalization
Adapthisteq	Contrast-limited adaptive histogram equalization (CLAHE)
Imhistmatch	Adjust histogram of image to match N-bin histogram of reference image
Decorrstretch	Apply decorrelation stretch to multichannel image
stretchlim	Find limits to contrast stretch Image
intlut	Convert integer values using lookup table
Imnoise	Add noise to image

HISTOGRAM GRAY IMAGE C

```
Mat src, dst;
  /// Convert to grayscale
  cvtColor( src, src, CV_BGR2GRAY );
  /// Apply Histogram Equalization
  equalizeHist( src, dst );
```

HISTOGRAM COLOR IMAGE C

```
Mat image; Mat histeq_image;
cvtColor(image, histeq_image, COLOR_BGR2YCrCb);
vector<Mat> vec_channels;
split(histeq_image, vec_channels);
//Equalize the histogram of only the Y channel
equalizeHist(vec_channels[0], vec_channels[0]);
merge(vec_channels, histeq_image);
cvtColor(histeq_image, histeq_image, COLOR_YCrCb2BGR);
```

Chương trình C smooth

```
#include "cv.h"
#include "highgui.h"
#include <iostream>
#include "cxcore.h"
Int main()
{
    CvCapture* capture = cvCaptureFromCAM(0);
    IplImage* src = NULL;
  cvNamedWindow("Webcam",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("WebcamHistogram",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("WebcamSmooth",CV_WINDOW_AUTOSIZE);
    IplImage* frame = cvQueryFrame(capture);
    IplImage* hsv = cvCreateImage( cvGetSize(frame), 8, 3 );
    IplImage* smooth = cvCreateImage( cvGetSize(frame), 8, 3 );
    IplImage* h_plane  = cvCreateImage( cvGetSize(frame), 8, 1 );
    IplImage* s_plane  = cvCreateImage( cvGetSize(frame), 8, 1 );
```

Chương trình C smooth

```
IplImage* v_plane  = cvCreateImage( cvGetSize(frame), 8, 1 );
    IplImage* planes[] = { h_plane, s_plane };
 int h_bins = 30, s_bins = 32, scale = 10;
    IplImage* hist_img = cvCreateImage(
            cvSize( h_bins * scale, s_bins * scale ),
            8,
            3
            );
   cvZero( hist_img );
    while(1)
    {
            src = cvQueryFrame(capture);
            if(!src) break;
            cvSmooth(src,smooth,CV_BLUR,5,3);
            cvCvtColor( src, hsv, CV_BGR2HSV );
            cvCvtPixToPlane( hsv, h_plane, s_plane, v_plane, 0 );
```

Chương trình C smooth

```
CvHistogram* hist;
    {
                int    hist_size[] = { h_bins, s_bins };
                float  h_ranges[]  = { 0, 100 };        // day la nguong hue, thay doi
    theo mong muon
                float  s_ranges[]  = { 0, 255 };
                float* ranges[]    = { h_ranges, s_ranges };
                hist = cvCreateHist(
                        2,
                        hist_size,
                        CV_HIST_ARRAY,
                        ranges,
                        1
                        );
    }
    cvCalcHist( planes, hist, 0, 0 );
```

Chương trình C smooth

```
float max_value = 0;
 cvGetMinMaxHistValue( hist, 0, &max_value, 0, 0 );
for( int h = 0; h < h_bins; h++ ) {
    for( int s = 0; s < s_bins; s++ ) {
        float bin_val = cvQueryHistValue_2D( hist, h, s );
        int intensity = cvRound( bin_val * 255 / max_value );
        cvRectangle(
                                        hist_img,
                                        cvPoint( h*scale, s*scale ),
                                        cvPoint( (h+1)*scale - 1, (s+1)*scale - 1),
                                        CV_RGB(intensity,intensity,intensity),
                                        CV_FILLED
                                        );
    }
 }
```

Chương trình C smooth

```
//cvFlip(smooth,smooth,0);
            cvShowImage("Webcam",src);
            cvShowImage("WebcamSmooth",smooth);
            cvShowImage("WebcamHistogram",hist_img);
            char c = cvWaitKey(5);
            if(c==27) break;
    }

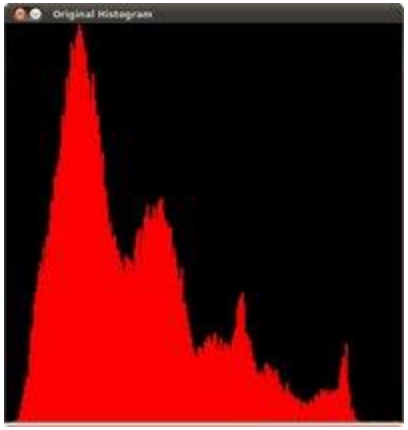
     cvReleaseCapture(&capture);
    cvReleaseCapture(&capture1);
         cvDestroyAllWindow();
    return 0;
 }
```

HISTOGRAM EQUALZATION C++

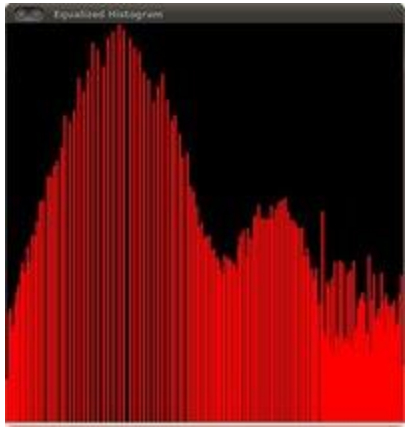
```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream> #include <stdio.h>
using namespace cv;
using namespace std;
/** @function main */
int main( int argc, char** argv )
{ Mat src, dst;
char* source_window = "Source image";
char* equalized_window = "Equalized Image";
/// Load image
src = imread( argv[1], 1 );
if( !src.data )
 { cout<<"Usage: ./Histogram_Demo <path_to_image>"<<endl; return -1;}
```

```
/// Convert to grayscale
cvtColor( src, src, CV_BGR2GRAY );
/// Apply Histogram Equalization
equalizeHist( src, dst );
 /// Display results
namedWindow( source_window, CV_WINDOW_AUTOSIZE );
namedWindow( equalized_window, CV_WINDOW_AUTOSIZE );
imshow( source_window, src );
imshow( equalized_window, dst );
/// Wait until user exits the program
waitKey(0);
return 0; }
```


Source image


Original Histogram


Equalized Image


Equalized Histogram

COLOR IMAGE HISEQ

- Đối với ảnh là ảnh xám, việc cân bằng histogram là đơn giản, ta chỉ vần gọi hàm cvEqualizeHist(src, dst), trong đó src là ảnh xám nguồn và dst là ảnh xám đích có cùng kích thước.
  Tuy nhiên đối với trường hợp ảnh là ảnh mầu, đầu tiên ta phải chuyển đổi sang không gian màu hsv, sau đó cân bằng mức xám ở kênh màu V (value, tức là kênh về giá trị cường độ sáng), cuối cùng là hợp các kênh này lại dùng hàm cvMerge để được kết quả cuối cùng. Sau đây là một số hình ảnh ví dụ về cân bằng mức xám trong chương trình trên, với ảnh đầu vào là ảnh hoa sen trong thư mục ảnh của Windows, ta giả sử ảnh nằm ở ổ D.

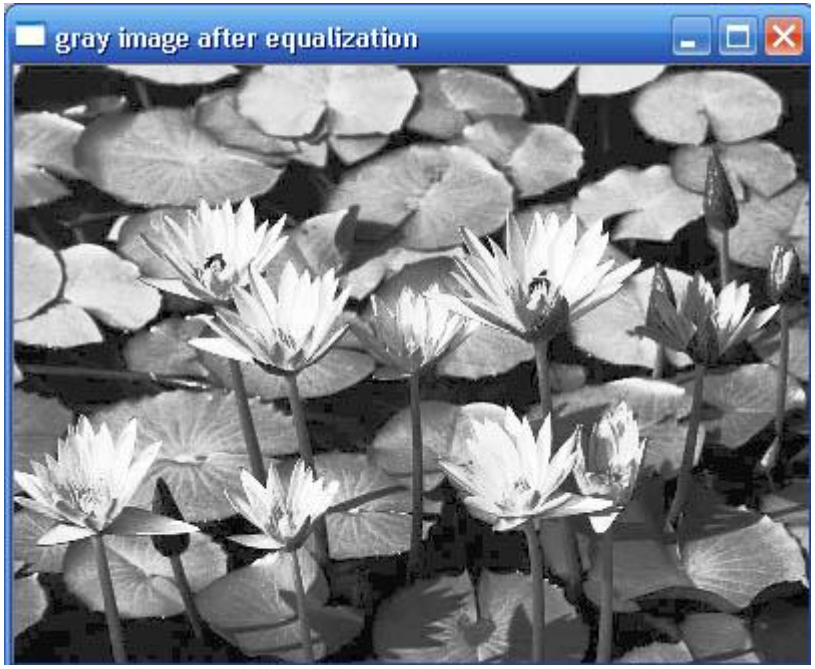
COLOR IMAGE HISEQ

- ```
  #include <opencv2/opencv.hpp>
  int main() {
      IplImage *src = cvLoadImage("D:\\flower.jpg",
  CV_LOAD_IMAGE_COLOR);
      IplImage *gray = cvCreateImage(cvSize(src->width, src->height),
  8, 1);
      IplImage *hsv = cvCloneImage(src);
      cvCvtColor(src, gray, CV_BGR2GRAY);
      cvCvtColor(src, hsv, CV_RGB2HSV);
      IplImage *h, *s, *v;
      h   = cvCreateImage(cvGetSize(src), 8, 1);
      s   = cvCreateImage(cvGetSize(src), 8, 1);
      v   = cvCreateImage(cvGetSize(src), 8, 1);
      cvSplit(hsv, h, s, v, NULL);
  ```
  ```

COLOR IMAGE HISEQ

- cvNamedWindow("source color image", *CV_WINDOW_NORMAL*);
  cvNamedWindow("source gray image", *CV_WINDOW_NORMAL*);
  cvShowImage("source color image", src);
  cvShowImage("source gray image", gray);
  cvEqualizeHist(gray, gray);
  cvEqualizeHist(v, v);
  cvMerge(h, s, v, NULL, hsv);
  cvCvtColor(hsv, src, CV_HSV2RGB);
  cvNamedWindow("color image after equalization", *CV_WINDOW_NORMAL*);
  cvNamedWindow("gray image after equalization", *CV_WINDOW_NORMAL*);
  cvShowImage("color image after equalization", src);
  cvShowImage("gray image after equalization", gray);
  cvWaitKey(0);
  return 0;
  }


source gray image


gray image after equalization


source color image


color image after equalization

HIST EQU VIDEO

```
#include "stdafx.h"
#include "highgui.h"
#include "cv.h“
int main()
{
	 CvCapture* capture = cvCaptureFromCAM(0);
	IplImage* src = NULL;
	cvNamedWindow("Webcam",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("Webcam_contract",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("WebcamSmooth",CV_WINDOW_AUTOSIZE);
	IplImage* frame = cvQueryFrame(capture);
	IplImage* hsv = cvCreateImage( cvGetSize(frame), 8, 3 );
	IplImage* smooth = cvCreateImage( cvGetSize(frame), 8, 3 );
```

HIST EQU VIDEO CONSOLE

```
IplImage* h  = cvCreateImage( cvGetSize(frame), 8, 1 );
IplImage* s  = cvCreateImage( cvGetSize(frame), 8, 1 );
IplImage* v  = cvCreateImage( cvGetSize(frame), 8, 1 );
while(1)
    {
    src = cvQueryFrame(capture);
    if(!src) break; // khong thu duoc capture => break.
    cvSmooth(src, smooth, CV_BLUR, 5, 3);/* Smoothes array (removes
    noise) */ // lam tron mang, loai bo nhieu
    //CV_BLUR : linear convolution with size1xsize2  box kernel (all 1's)
    with subsequent scaling by  1/[size1xsize2]
      cvCvtColor( src, hsv, CV_RGB2HSV); // chuyen doi mau
      //The function converts an input image from one color space to
      another
      cvSplit(hsv,h,s,v, 0);
```

HIST EQU VIDEO console

```
            /* Equalize the histogram */
            cvEqualizeHist(v,v);
            cvMerge(h, s, v,0, hsv);
            cvCvtColor(hsv, hsv, CV_HSV2RGB);
            //Show results
            cvShowImage("Webcam",src);
            cvShowImage("WebcamSmooth",smooth);
            cvShowImage("Webcam_contrast", hsv);
            char c = cvWaitKey(33);
            if(c==27) break;
            }
cvReleaseCapture(&capture);
cvReleaseImage(&smooth);
cvReleaseImage(&src);
cvReleaseImage(&frame);
cvReleaseImage(&hsv);

}
```

TẠO VIDEO GRAY WINFORM OPENCV2.2.0

- Tạo form có hai picturebox 1,2. Hai nút nhấn Start Exit, Timer 30ms
- Các đoạn code quan trọng

```
#pragma once
#include <cxcore.h>
#include <highgui.h>
#include <opencv2/imgproc/imgproc_c.h>  // cvCvtColor
#ifdef _DEBUG
#pragma comment(lib,"opencv_core220d.lib")
    #pragma comment(lib,"opencv_highgui220d.lib")
    #pragma comment(lib,"opencv_imgproc220d.lib")
#else
#pragma comment(lib,"opencv_core220.lib")
```

TẠO VIDEO GRAY WINFORM OPENCV2.2.0

```
 #pragma comment(lib,"opencv_highgui220.lib")
      #pragma comment(lib,"opencv_imgproc220d.lib")
#endif
CvCapture* capture;
IplImage* frame;
IplImage* gray;
IplImage* gray2;
```

TẠO VIDEO GRAY WINFORM OPENCV2.2.0

```
private: System::Void button1_Click(System::Object^
    sender, System::EventArgs^  e) {
cvReleaseCapture(&capture);
 this->Close(); }//exit
private: System::Void button2_Click (System::Object^
    sender, System::EventArgs^  e) {
capture = cvCaptureFromCAM(0);
          timer1->Start(); }//start
```

TẠO VIDEO GRAY WINFORM OPENCV2.2.0

```
private: System::Void timer1_Tick (System::Object^
    sender, System::EventArgs^  e) {
frame = cvQueryFrame(capture);
 IplImage* gray = cvCreateImage( cvGetSize(frame), 8, 1
    );
 IplImage* gray2 = cvCreateImage( cvGetSize(frame), 8, 3
    );
cvCvtColor(frame,gray, CV_BGR2GRAY );//1 plane
cvMerge(gray , gray, gray, NULL, gray2);  //3 planes
pictureBox1->Image  = gcnew    //replacement of
    cvShowImage
```

TẠO VIDEO GRAY WINFORM OPENCV2.2.0

- System::Drawing::Bitmap(frame->width,frame->height,frame->widthStep,
- System::Drawing::Imaging::PixelFormat::Format24bppRgb,(System::IntPtr) frame->imageData);
- pictureBox1->Refresh();//Color video
- pictureBox2->Image = gcnew
- System::Drawing::Bitmap(gray2->width,gray2->height,gray2->widthStep,
- System::Drawing::Imaging::PixelFormat::Format24bppRgb,(System::IntPtr) gray2->imageData);
- pictureBox2->Refresh(); }//gray video

TẠO VIDEO GRAY WINFORM OPENCV2.2.0

CÂN BẰNG CAMERA MÀU

```
#include "cv.h"
#include "stdafx.h"
#include "highgui.h"
int main()
{
CvCapture* capture = cvCaptureFromCAM(0);
IplImage* src = NULL;
IplImage* hist_img =NULL;
IplImage* a_src = NULL;
while(1)
{
IplImage *src = cvQueryFrame(capture);
IplImage *hsv = cvCloneImage(src);//coppy
```

```
IplImage *a_src = cvCloneImage(src);//coppy
cvCvtColor(src, hsv, CV_RGB2HSV);
IplImage *h, *s, *v;
h   = cvCreateImage(cvGetSize(src), 8, 1);
s   = cvCreateImage(cvGetSize(src), 8, 1);
v   = cvCreateImage(cvGetSize(src), 8, 1);
cvSplit(hsv, h, s, v, NULL);
cvEqualizeHist(v, v);
cvMerge(h, s, v, NULL, hsv);
cvCvtColor(hsv, a_src, CV_HSV2RGB);
cvShowImage("Webcam", src);
cvShowImage("Webcam after equalization", a_src);
char c = cvWaitKey(5);
if(c==27) break;
}
}
```

LÀM ẢNH NÉT

- The blurring, or degradation, of an image can be caused by many factors:
  - Movement during the image capture process, by the camera or, when long exposure times are used, by the subject
  - Out-of-focus optics, use of a wide-angle lens, atmospheric turbulence, or a short exposure time, which reduces the number of photons captured
  - Scattered light distortion in confocal microscopy
- A blurred or degraded image can be approximately described by this equation
- $$g = Hf + N, \text{ where}$$

- g The blurred image
- H The distortion operator, also called the point spread function (PSF). In the spatial domain, the PSF describes the degree to which an optical system blurs (spreads) a point of light. The PSF is the inverse Fourier transform of the optical transfer function (OTF). In the frequency domain, the OTF describes the response of a linear, position-invariant system to an impulse. The distortion operator, when convolved with the image, creates the distortion.
- f The original true image
- N Additive noise, introduced during image acquisition, that corrupts the image

Ảnh nét và ảnh không nét

TĂNG CƯỜNG ẢNH DÙNG PHÉP TOÁN SỐ HỌC

- Trừ ảnh làm nổi bật sự khác biệt giữa hai ảnh

  $g(x, y) = f(x, y) - h(x, y)$

- Trung bình nhiều ảnh làm giảm nhiễu

  $g_i(x, y) = f(x, y) + \eta_i(x, y)$

  $\eta_i(x, y)$ là nhiễu

$$\bar{g}(x, y) = \frac{1}{K} \sum_{i=1}^{K} g_i(x, y)$$

$$E\{\bar{g}(x, y)\} = f(x, y)$$

$$\sigma^2_{\bar{g}(x,y)} = \frac{1}{K} \sigma^2_{\eta(x,y)}$$

LỌC MIỀN KHÔNG GIAN

- Nhiễu gây cho ta những khó khăn khi phân tích tín hiệu. Vì vậy, kỹ thuật lọc số miền không gian được ứng dụng.
- Trong kỹ thuật này, người ta sử dụng một mặt nạ (kernel) và di chuyển khắp ảnh gốc.
- Tùy theo cách tổ hợp điểm đang xét với các điểm lân cận mà ta có kỹ thuật lọc không gian tuyến tính hay phi tuyến. Điểm ảnh chịu tác động của biến đổi là điểm tâm mặt nạ.

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

LỌC TUYẾN TÍNH LINEAR SPATIAL FILTERING

- Dùng mặt nạ 3x3, biểu thị bằng ma trận 3x3, nhân ma trận này với cường độ điểm ảnh láng giềng rồi cộng kết quả để được cường độ điểm giữa, còn gọi là tích chập convolution, thường dùng các mặt nạ sau

$$H_1 = \frac{1}{9}\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \quad H_2 = \frac{1}{10}\begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{vmatrix} \quad H_3 = \frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

- Mặt nạ H1 là lấy trung bình, H2 ưu tiên cho điểm giữa, H3 ưu tiên cho hướng x,y
- Matlab: g = imfilter(f, w, 'replicate')

LỌC TUYẾN TÍNH LINEAR SPATIAL FILTERING

Nếu H là bộ lọc kích thước (n+1) x (n+1), n chẵn và tổng các hệ số là K, I_f sẽ được tính bởi:

$$I_f = \frac{1}{K} \sum_{i=-n/2}^{n/2} \sum_{j=-n/2}^{n/2} H_1(i+n/2, j+n/2)\, I_i(x+i, y+j)$$

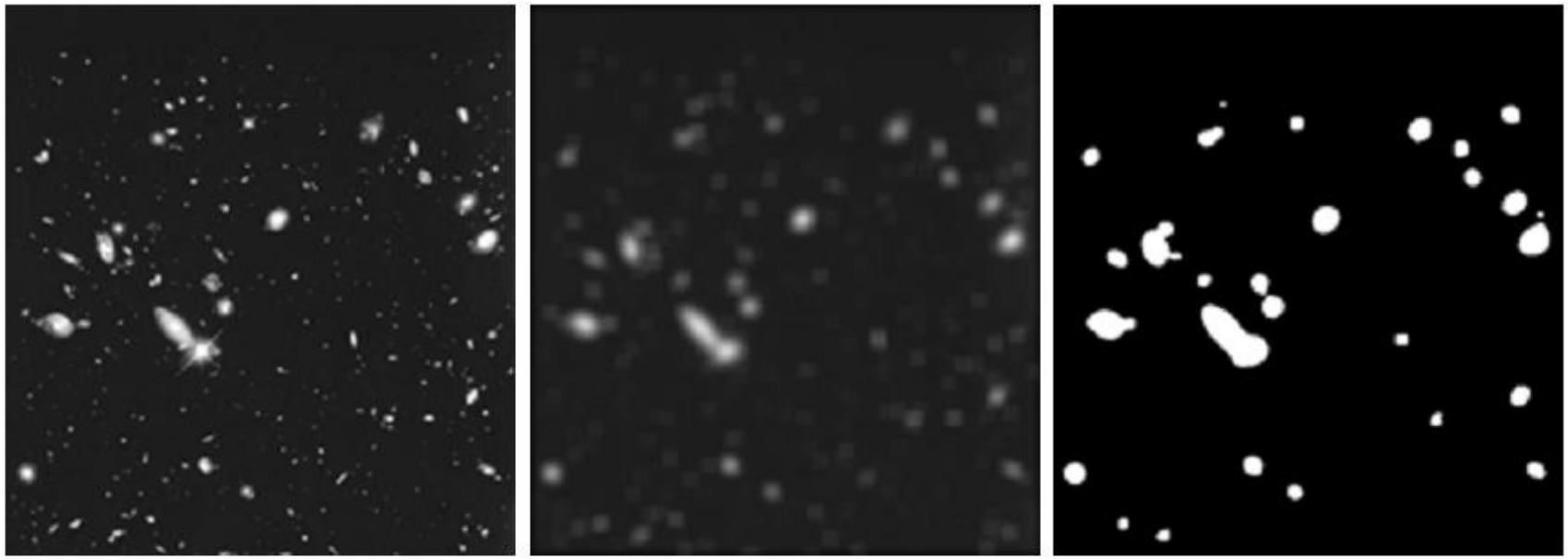
Ví dụ: Dùng mặt nạ H_1

$$S = \begin{matrix} 20 & 20 & 30 & 40 \\ 30 & 50 & 70 & 60 \\ 20 & 200 & 30 & 10 \\ 40 & 100 & 30 & 70 \end{matrix}$$

$$S(3,2) = \left\lfloor \frac{30+50+70+20+200+30+40+100+30}{9} \right\rfloor = 63$$

Với các điểm ảnh ở biên ta bổ sung thêm các điểm ảnh mới bằng cách sao chép hay cho một giá trị nào đó cho cường độ, sau khi lọc ta sẽ bỏ các điểm ảnh này

LỌC TUYẾN TÍNH LINEAR SPATIAL FILTERING



a b c

(a) Image from the Hubble Space Telescope. (b) Image processed by a 15 × 15 averaging mask. (c) Result of thresholding (b). (Original image courtesy of NASA.)

LỌC KHÔNG GIAN PHI TUYẾN

- Lọc trung vị (vị trí giữa), median filter: cho cửa sổ quét qua các điểm ảnh, cường độ sáng các pixel trong cửa sổ được xếp theo thứ tự tăng dần tạo thành chuỗi có số hạng lẻ, cường độ sáng điểm giữa cửa số là giá tri số hạng giữa của chuỗi

Ví dụ:

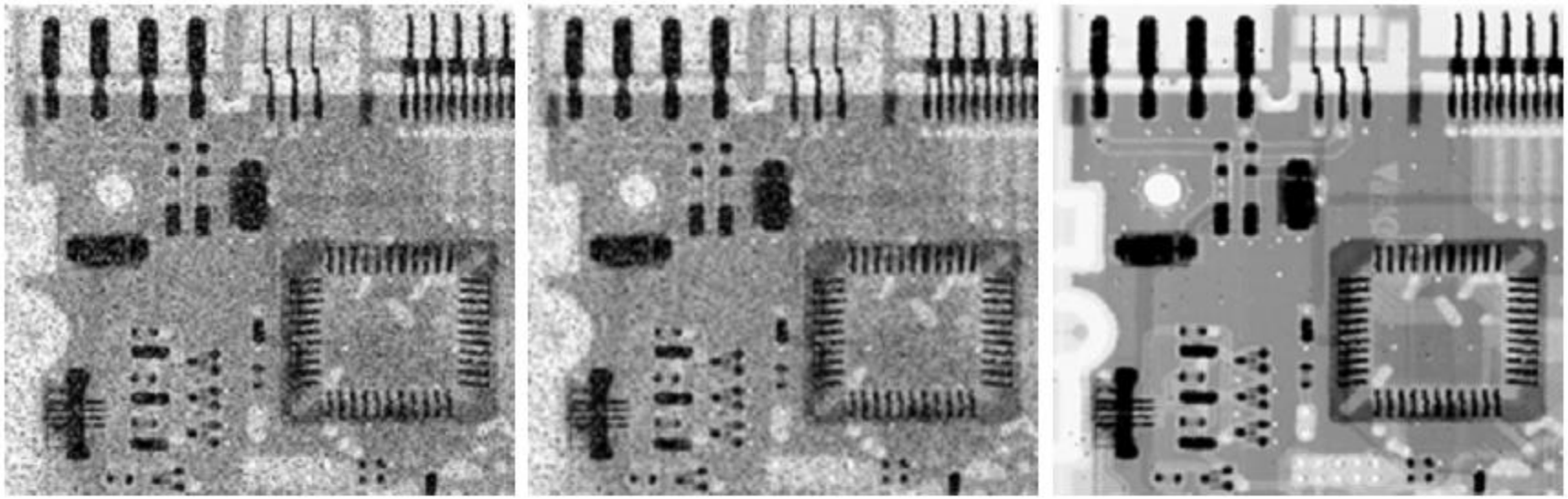
$$S = \begin{matrix} 20 & 20 & 30 & 40 \\ 30 & 50 & 70 & 60 \\ 20 & 200 & 30 & 10 \\ 40 & 100 & 30 & 70 \end{matrix}$$

Dùng cửa sổ 3x3 ta có.

20 30 30 30 **40** 50 70 100 200

LỌC KHÔNG GIAN PHI TUYẾN

- Lọc trung vị không làm mờ ảnh và giúp lọc nhiễu xung muối tiêu (salt and pepper)



a b c

(a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a 3×3 averaging mask. (c) Noise reduction with a 3×3 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

(a)

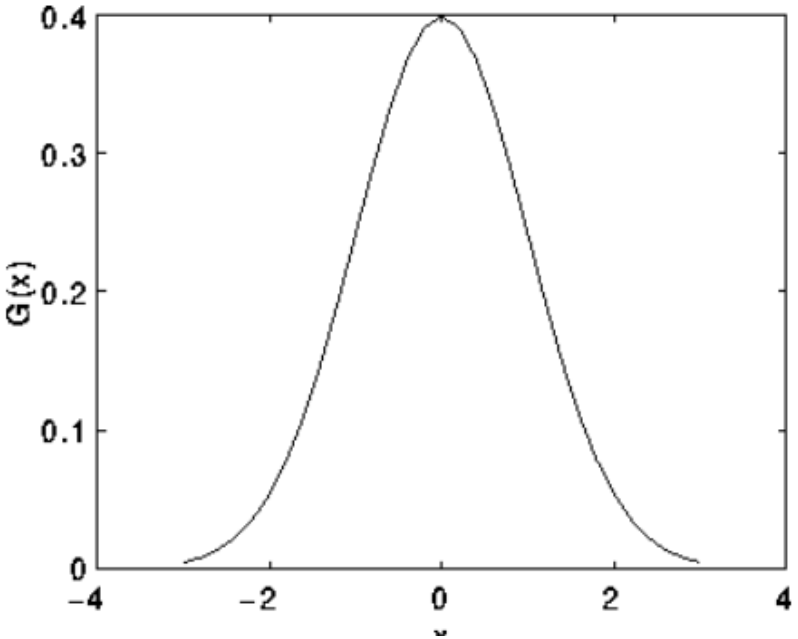
(b)

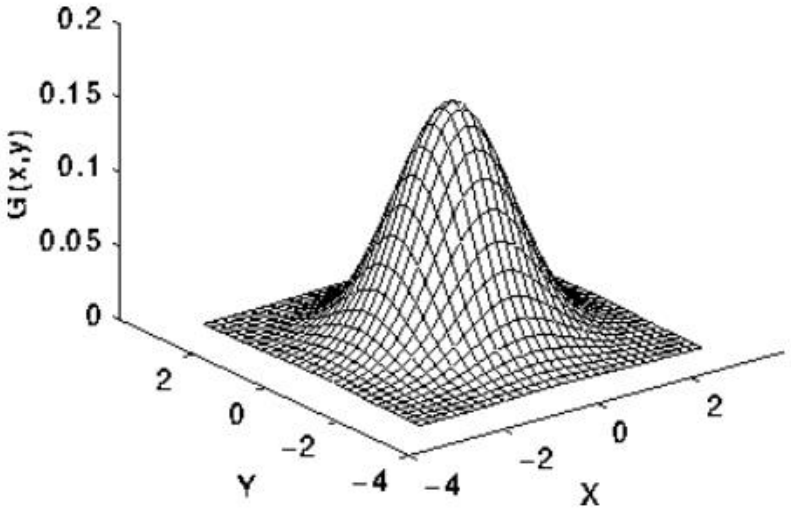
Ảnh lọc trung vị 3*3

LỌC GAUSS

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$





Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$

LỌC ẢNH MATLAB

Basic Image Filtering in the Spatial Domain

imfilter	N-D filtering of multidimensional images
fspecial	Create predefined 2-D filter
roifilt2	Filter region of interest (ROI) in image
nlfilter	General sliding-neighborhood operations
imgaussfilt	2-D Gaussian filtering of images
imgaussfilt3	3-D Gaussian filtering of 3-D images
wiener2	2-D adaptive noise-removal filtering
medfilt2	2-D median filtering
medfilt3	3-D median filtering
ordfilt2	2-D order-statistic filtering
stdfilt	Local standard deviation of image

IMFILTER, FSPECIAL

- B = imfilter(A,h) filters the multidimensional array A with the multidimensional filter h. The array A can be logical or a nonsparse numeric array of any class and dimension. The result B has the same size and class as A.
- h = fspecial(type) creates a two-dimensional filter h of the specified type. Some of the filter types have optional additional parameters, shown in the following syntaxes. fspecial returns h as a correlation kernel, which is the appropriate form to use with imfilter.
- h = fspecial('average',hsize) returns an averaging filter h of size hsize.
- h = fspecial('disk',radius) returns a circular averaging filter (pillbox) within the square matrix of size 2*radius+1.

FSPECIAL

- h = fspecial('gaussian',hsize,sigma) returns a rotationally symmetric Gaussian lowpass filter of size hsize with standard deviation sigma (positive). Not recommended. Use imgaussfilt or imgaussfilt3 instead.
- h = fspecial('laplacian',alpha) returns a 3-by-3 filter approximating the shape of the two-dimensional Laplacian operator, alpha controls the shape of the Laplacian.
- h = fspecial('log',hsize,sigma) returns a rotationally symmetric Laplacian of Gaussian filter of size hsize with standard deviation sigma (positive).
- h = fspecial('motion',len,theta) returns a filter to approximate, once convolved with an image, the linear motion of a camera. len specifies the length of the motion and theta specifies the angle of motion in degrees in a counter-clockwise direction.

```
I =
imread('cameraman.tif');
imshow(I);
H =
fspecial('disk',10);
blurred =
imfilter(I,H,'replicate');
imshow(blurred);
```

LỌC TRUNG VỊ MATLAB

```
I = imread('eight.tif');
figure, imshow(I)
%Add salt %pepper noise
J = imnoise(I,'salt & pepper',0.02);
%Use a median filter to filter out the noise.
K = medfilt2(J);
%Display results, side-by-side.
imshowpair(J,K,'montage')
```

ROI Region of Interest

- J = roifilt2(h,I,BW) filters regions of interest (ROIs) in the 2-D image I using the 2-D linear filter h. BW is a binary mask, the same size as I, that defines the ROIs in I. roifilt2 returns an image that consists of filtered values for pixels in locations where BW contains 1s, and unfiltered values for pixels in locations where BW contains 0s.
- J = roifilt2(I,BW,fun) processes the data in ROIs of I using the function fun. The value fun must be a function handle.

ROI Region of Interest

- BW = roipoly(I,c,r) returns a polygonal ROI, BW with vertices defined by pixel column and row indices, c and r.
- BW = roipoly(x,y,I,xi,yi) returns a polygonal ROI with vertices defined in a nondefault spatial coordinate system. x and y specify the image limits in the world coordinate system. xi and yi specify coordinates of polygon vertices as locations in this coordinate system.

ROI Region of Interest

```
%Read an image into the workspace.
I = imread('eight.tif');
%Define the vertices of the mask polygon.
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
%Create the binary mask image.
BW = roipoly(I,c,r); imshow(BW)
%Filter the region of the image I specified by the mask BW.
H = fspecial('unsharp');
J = roifilt2(H,I,BW);
imshow(I)
figure
imshow(J)
```

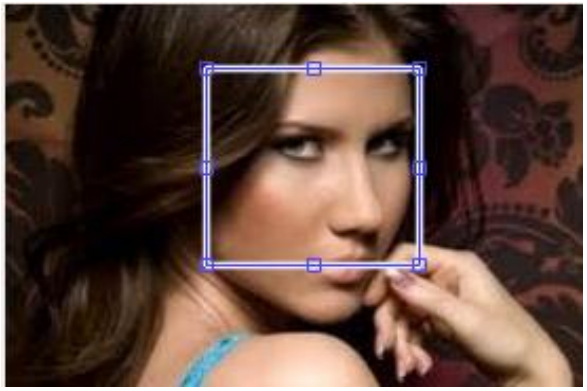
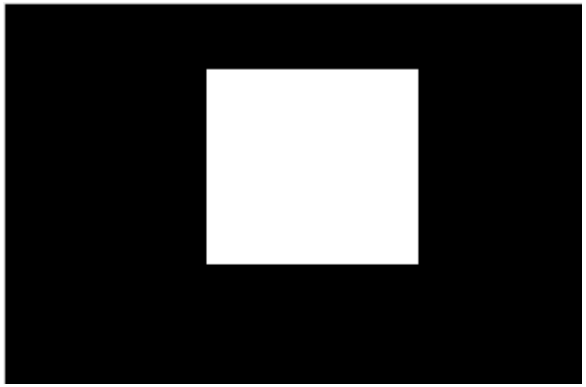
Mouse Select ROI Gray Matlab

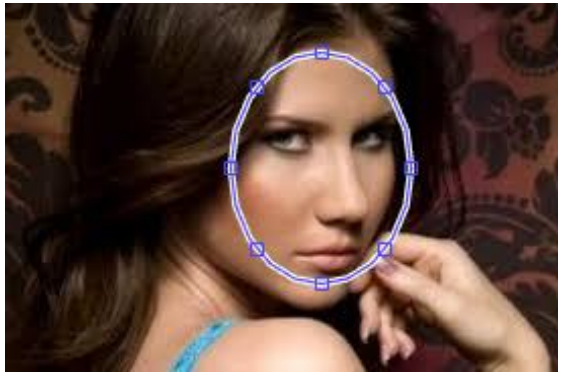
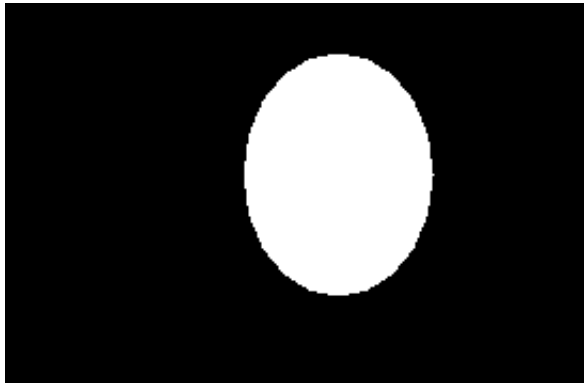
```
I=imread('eight.tif');
figure
I1=imshow(I)
h = imrect;%Use Mouse to select Rectangle
BW = createMask(h,I1);
figure
imshow(BW)
H = fspecial('unsharp');
I = roifilt2(H,I,BW);
figure
imshow(I)
```

Mouse Select ROI Color Matlab

```
I=imread('c:/annachapman.jpg');
figure
I1=imshow(I)
h = imrect; %imellipse;
%Use wait to block the MATLAB® command line. Double-click on the rectangle to resume execution of the MATLAB command line.
Wait(h);
BW = createMask(h,I1);
figure
imshow(BW)
H = fspecial('disk',10);
r=I(:,:,1);g=I(:,:,2);b=I(:,:,3);
rf = roifilt2(H,r,BW);gf = roifilt2(H,g,BW);bf = roifilt2(H,b,BW);
```

Mouse Select ROI Color Matlab

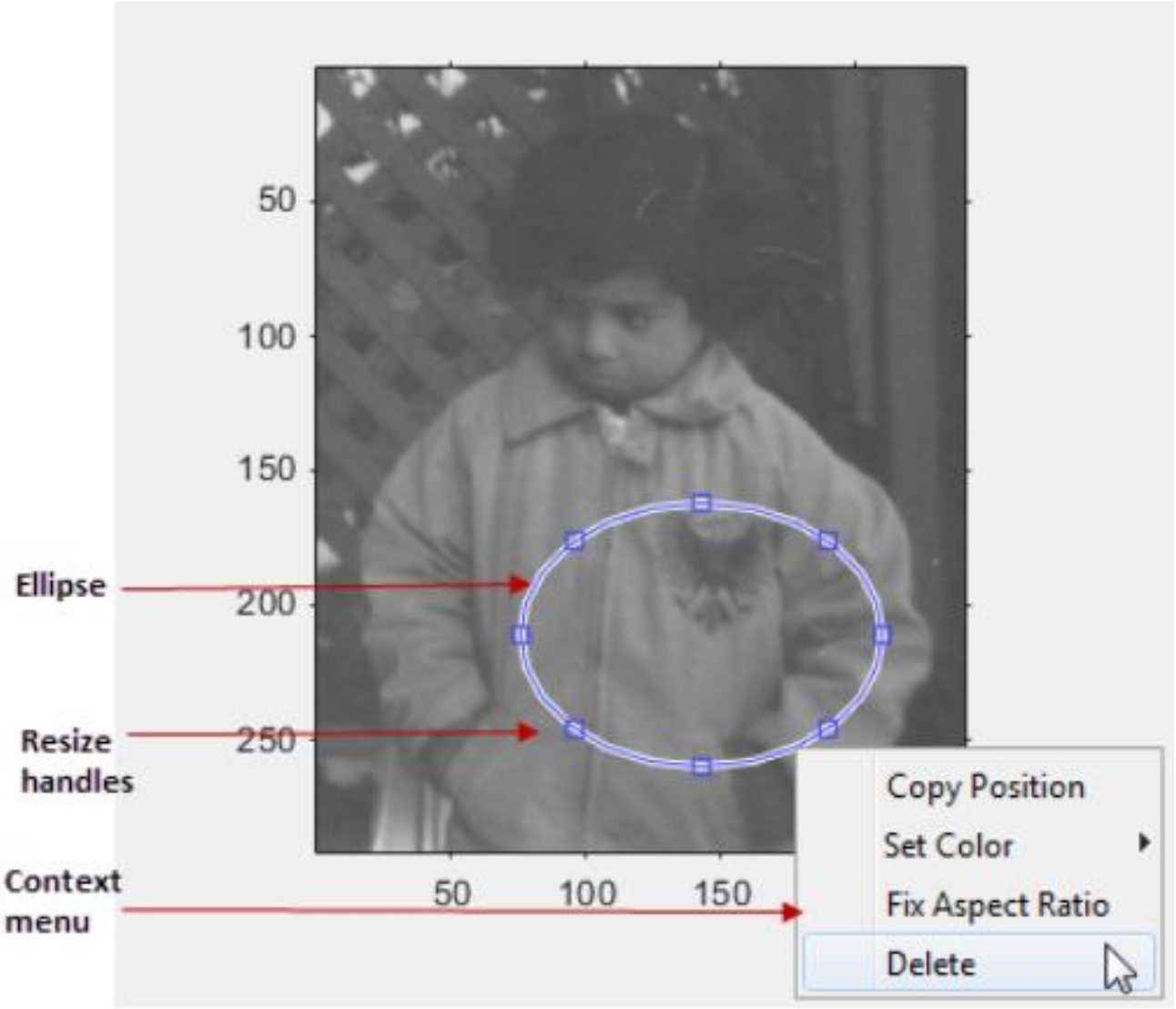




Các hàm liên quan ROI Matlab

- imellipse Create draggable ellipse. An imellipse object encapsulates an interactive ellipse over an image. You can adjust the size and position of the ellipse by using the mouse. The ellipse also has a context menu that controls aspects of its appearance and behavior.
- imfreehand Create draggable ellipse
- imrect Create draggable rectangle
- impoly Create draggable, resizable polygon
- When you call imellipse with an interactive syntax, the pointer changes to a cross hairs when over an image. Click and drag the mouse to specify the size and position of the ellipse. The ellipse also supports a context menu that you can use to control aspects of its appearance and behavior. Right-click on the ellipse to access this context menu.

Các hàm liên quan ROI Matlab



Các hàm liên quan ROI Matlab

addNewPositionCallback	Add new-position callback to ROI object
createMask	Create mask within image
delete	Delete handle object
getColor	Get color used to draw ROI object
getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object
resume	Resume execution of MATLAB command line
setClosed	Set closure behavior of ROI object
setColor	Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
wait	Block MATLAB command line until ROI creation is finished

LỌC GAUSS C

GaussianBlur ¶

Blurs an image using a Gaussian filter.

C++: void **GaussianBlur**(InputArray **src**, OutputArray **dst**, Size **ksize**, double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT)

Python: `cv2.GaussianBlur`(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst

Parameters:

- **src** – input image; the image can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – Gaussian kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from `sigma*` .

- **sigmaX** – Gaussian kernel standard deviation in X direction.
- **sigmaY** – Gaussian kernel standard deviation in Y direction; if `sigmaY` is zero, it is set to be equal to `sigmaX`, if both sigmas are zeros, they are computed from `ksize.width` and `ksize.height` , respectively (see **`getGaussianKernel()`** for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of `ksize`, `sigmaX`, and `sigmaY`.
- **borderType** – pixel extrapolation method (see **`borderInterpolate()`** for details).

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

bilateralFilter

Applies the bilateral filter to an image.

C++: void **bilateralFilter**(InputArray **src**, OutputArray **dst**, int **d**, double **sigmaColor**, double **sigmaSpace**, int **borderType**=BORDER_DEFAULT)

Python: `cv2.bilateralFilter`(src, d, sigmaColor, sigmaSpace[, dst[, borderType]]) → dst

Parameters:

- **src** – Source 8-bit or floating-point, 1-channel or 3-channel image.
- **dst** – Destination image of the same size and type as `src` .
- **d** – Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from `sigmaSpace` .
- **sigmaColor** – Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see `sigmaSpace`) will be mixed together, resulting in larger areas of semi-equal color.

- **sigmaSpace** – Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see `sigmaColor`). When `d>0` , it specifies the neighborhood size regardless of `sigmaSpace` . Otherwise, `d` is proportional to `sigmaSpace` .

Sigma values: For simplicity, you can set the 2 sigma values to be the same. If they are small (< 10), the filter will not have much effect, whereas if they are large (> 150), they will have a very strong effect, making the image look “cartoonish”.

Filter size: Large filters (d > 5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for offline applications that need heavy noise filtering.

LỌC TRUNG VỊ C

```
Mat src;  Mat dst;
    //Apply median filter
   medianBlur ( src, dst, 15 );
   imshow("source", src);
   imshow("result", dst);
```

LỌC TRUNG VỊ C KHÔNG DÙNG LỆNH CÓ SẴN

```
void insertionSort(int window[])
{
    int temp, i , j;
    for(i = 0; i < 9; i++){
        temp = window[i];
        for(j = i-1; j >= 0 && temp < window[j]; j--){
            window[j+1] = window[j];
        }
        window[j+1] = temp;
    }
}
```

LỌC TRUNG VỊ C

```
int main()
{     Mat src, dst;
src = imread("book.png",
CV_LOAD_IMAGE_GRAYSCALE);
//create a sliding window of size 9
      int window[9];
        dst = src.clone();
        for(int y = 0; y < src.rows; y++)
            for(int x = 0; x < src.cols; x++)
                dst.at<uchar>(y,x) = 0.0;
        for(int y = 1; y < src.rows - 1; y++){
            for(int x = 1; x < src.cols - 1; x++){
```

LỌC TRUNG VỊ C

```
window[0] = src.at<uchar>(y - 1 ,x - 1);
window[1] = src.at<uchar>(y, x - 1);
window[2] = src.at<uchar>(y + 1, x - 1);
window[3] = src.at<uchar>(y - 1, x);
window[4] = src.at<uchar>(y, x);
window[5] = src.at<uchar>(y + 1, x);
window[6] = src.at<uchar>(y - 1, x + 1);
window[7] = src.at<uchar>(y, x + 1);
window[8] = src.at<uchar>(y + 1, x + 1);
insertionSort(window);
dst.at<uchar>(y,x) = window[4]; } } imshow("final", dst);
imshow("initial", src); waitKey(); return 0;}
```

LỌC TRUNG VỊ C

medianBlur

Blurs an image using the median filter.

C++: void **medianBlur**(InputArray **src**, OutputArray **dst**, int **ksize**)

Python: `cv2.`**medianBlur**(src, ksize[, dst]) → dst

Parameters:
- **src** – input 1-, 3-, or 4-channel image; when `ksize` is 3 or 5, the image depth should be `CV_8U`, `CV_16U`, or `CV_32F`, for larger aperture sizes, it can only be `CV_8U`.
- **dst** – destination array of the same size and type as `src`.
- **ksize** – aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

The function smoothes an image using the median filter with the **ksize** × **ksize** aperture. Each channel of a multi-channel image is processed independently. In-place operation is supported.

Bài tập

Cho ma trận [20, 20, 30, 40, 30, 50, 70, 60, 20, 200, 30, 10, 40, 100, 30, 70]

Viết chương trình Matlab và C lọc trung vị với kernel 3*3

- Matlab

```
org=[20 20 30 40;30 50 70 60;20 200 30 10;40 100 30 70];
dest=medfilt2(org,[3 3])
```

```
dest =
     0    20    30     0
    20    30    40    30
    30    40    60    30
     0    30    30     0
```

Bài tập

- Opencv

```
medianBlur(org, dest, 3);
cout << "dest = " << endl << " " << dest << endl << endl;
waitKey(0);
return(0);
```

```
dest =
 [ 20,  30,  40,  40;
  20,  30,  40,  40;
  40,  40,  60,  60;
  40,  40,  70,  30]
```

OPENCV ROI

- Khai báo hình chữ nhật đỉnh top left x,y có bề rộng w và bề cao h làm ROI

```
Mat image = imread("");
Rec_Roi= Rect(x, y, w, h);
Mat image_roi = image(Rec_Roi);
```

- Cách khác

```
// image roi = image(Range(y,y+h), Range(x,x+w));
//Mat mask = ( input_mat != 0);
```

OPENCV ROI

```
#include <opencv2\opencv.hpp>
using namespace cv;
int main(void)
{
    Mat img = imread("path_to_image");
    imshow("Original", img);
    Rect r(100,100,200,200);
    Mat3b roi3b(img(r));
GaussianBlur(roi3b, roi3b, Size(), 10);
    imshow("After Blur", img);
```

OPENCV ROI

```
Mat1b roiGray;
cvtColor(roi3b, roiGray, COLOR_BGR2GRAY);
threshold(roiGray, roiGray, 200, 255, THRESH_BINARY);
Mat3b roiGray3b;
cvtColor(roiGray, roiGray3b, COLOR_GRAY2BGR);
roiGray3b.copyTo(roi3b);
imshow("After Threshold", img);
waitKey();
return 0;
}
```

ROI CIRCLE

```
Mat image = cv::imread("img.jpg");
Mat mask = cv::Mat::zeros(image.size(), CV_8UC1);
Point circleCenter(mask.cols / 2, mask.rows / 2);
int radius = min(mask.cols, mask.rows);
circle(mask, circleCenter, radius, CV_RGB(255, 255, 255));
Mat imagePart = Mat::zeros(image.size(), image.type());
image.copyTo(imagePart, mask);
Rect roi(circleCenter.x - radius, circleCenter.y - radius,
2*radius, 2*radius);
roi = roi & Rect(0, 0, image.cols, image.rows)).
```

OPENCV ROI

SELECT ROI MOUSE GAUSSIAN BLURRING

```
#include "stdafx.h"
#include <opencv2/opencv.hpp>
#include <opencv2/video/tracking.hpp>
 using namespace std;
using namespace cv;
 int main ()
{Mat im = imread("image.jpg");
Rect2d r = selectROI(im);
/*Mat imCrop = im(r);
imshow("Image", imCrop);*/
GaussianBlur(im(r), im(r), Size(), 20);
imshow("Image", im);waitKey(0); return 0;
}
```

LỌC CAO QUA
SHARPENING , HIGHPASS FILTER

- Tăng cường các chi tiết nhỏ
- Dùng đạo hàm cấp 1 và cấp 2

$$\frac{\partial f}{\partial x} = f(x+1) - f(x).$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x).$$

Đạo hàm cấp 2 Laplace

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial^2 x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial^2 y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

LỌC CAO QUA ĐẠO HÀM CẤP 1, GRADIENT

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}.$$

$$\begin{aligned} \nabla f &= \text{mag}(\nabla \mathbf{f}) \\ &= [G_x^2 + G_y^2]^{1/2} \\ &= \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2}. \end{aligned}$$

Để đơn giản, dùng công thức sau:

$$\nabla f = [(z_9 - z_5)^2 + (z_8 - z_6)^2]^{1/2}$$

$$\nabla f \approx |z_9 - z_5| + |z_8 - z_6|.$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

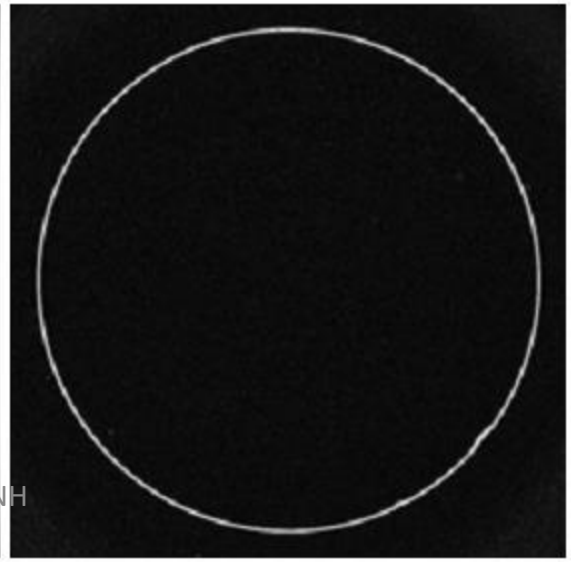
ĐẠO HÀM CẤP 1, GRADIENT

Toán tử Sobel

$$\nabla f \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|.$$

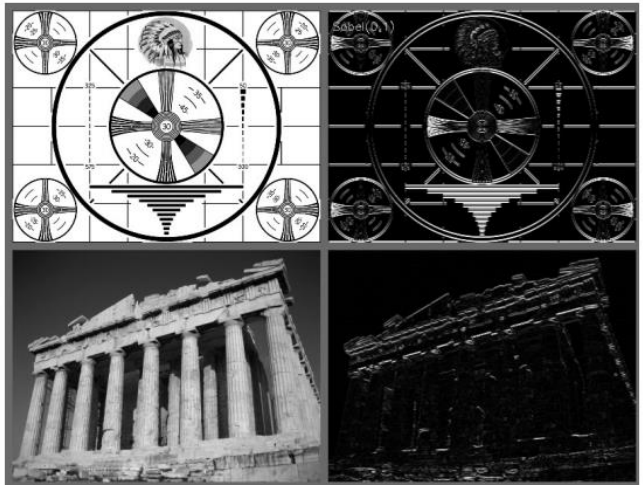
−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1



SOBEL

```
void cv::Sobel(
  cv::InputArray  src,                    // Input image
  cv::OutputArray dst,                    // Result image
  int             ddepth,                 // Pixel depth of output (e.g., CV_8U)
  int             xorder,                 // order of corresponding derivative in x
  int             yorder,                 // order of corresponding derivative in y
  cv::Size        ksize      = 3,         // Kernel size
  double          scale      = 1,         // Scale (applied before assignment)
  double          delta      = 0,         // Offset (applied before assignment)
  int             borderType = cv::BORDER_DEFAULT  // Border extrapolation
);
```

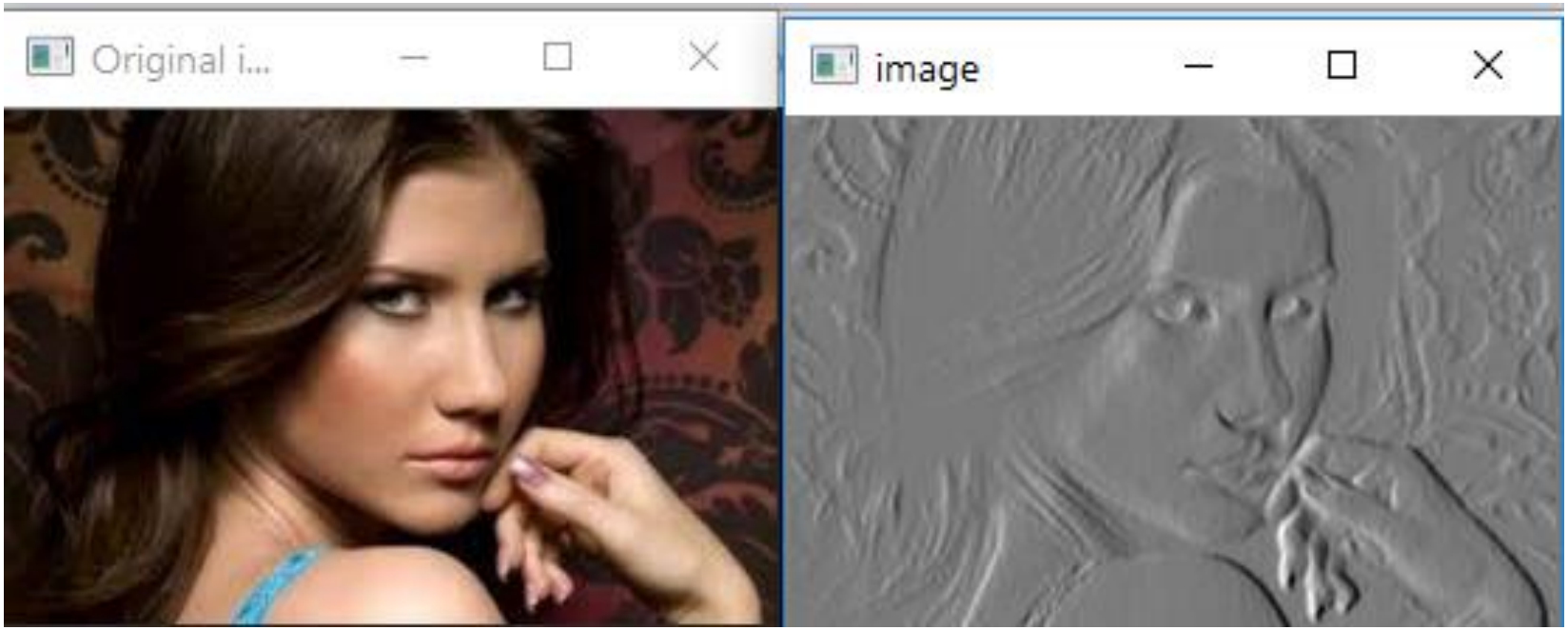


The effect of the Sobel operator when used to approximate a first derivative in the y-dimension

SOBEL x

```
Mat src,grey,sobelx,draw; double minVal, maxVal;
    src = imread(“c:/annachapman.jpg", 1);
    namedWindow( "Original image", 1 );
    imshow( "Original image", src1 );
cvtColor(src1, grey, CV_BGR2GRAY);
Sobel(grey, sobelx, CV_32F, 1, 0);
minMaxLoc(sobelx, &minVal, &maxVal); //find minimum and maximum intensities
    cout << "minVal : " << minVal << endl << "maxVal : " << maxVal << endl;
sobelx.convertTo(draw, CV_8U, 255.0/(maxVal - minVal), -minVal * 255.0/(maxVal - minVal));
     namedWindow("image", CV_WINDOW_AUTOSIZE);
    imshow("image", draw);
```

SOBEL x



SOBEL xy

```
Mat src, src_gray;   Mat grad;int scale = 1;
  int delta = 0;  int ddepth = CV_16S;  int c;
  /// Load an image
  src = imread( “c:/annachapman.jpg”);
namedWindow(“Source”,0); namedWindow(“Sobel”,0);
GaussianBlur( src, src, Size(3,3), 0, 0, BORDER_DEFAULT );
  /// Convert it to gray
  cvtColor( src, src_gray, CV_BGR2GRAY );
/// Generate grad_x and grad_y
  Mat grad_x, grad_y;   Mat abs_grad_x, abs_grad_y;
//Scharr( src_gray, grad_x, ddepth, 1, 0, scale, delta,
BORDER_DEFAULT );
```

SOBEL xy

```
Sobel( src_gray, grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT );
  convertScaleAbs( grad_x, abs_grad_x );
//Scharr( src_gray, grad_y, ddepth, 0, 1, scale, delta, BORDER_DEFAULT );
  Sobel( src_gray, grad_y, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT );
  convertScaleAbs( grad_y, abs_grad_y );
addWeighted( abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad );
 imshow( “Source”, src );
 imshow( “Sobel”, grad );
```


Source
Sobel

LỌC CAO QUA MẶT NẠ LAPLACE

- Thuật toán Laplace được thể hiện bởi các mặt nạ sau:

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

a b
c d

(a) Filter mask used to implement the digital Laplacian, as defined in Eq. (3.7-4). (b) Mask used to implement an extension of this equation that includes the diagonal neighbors. (c) and (d) Two other implementations of the Laplacian.

MẶT NẠ LAPLACE

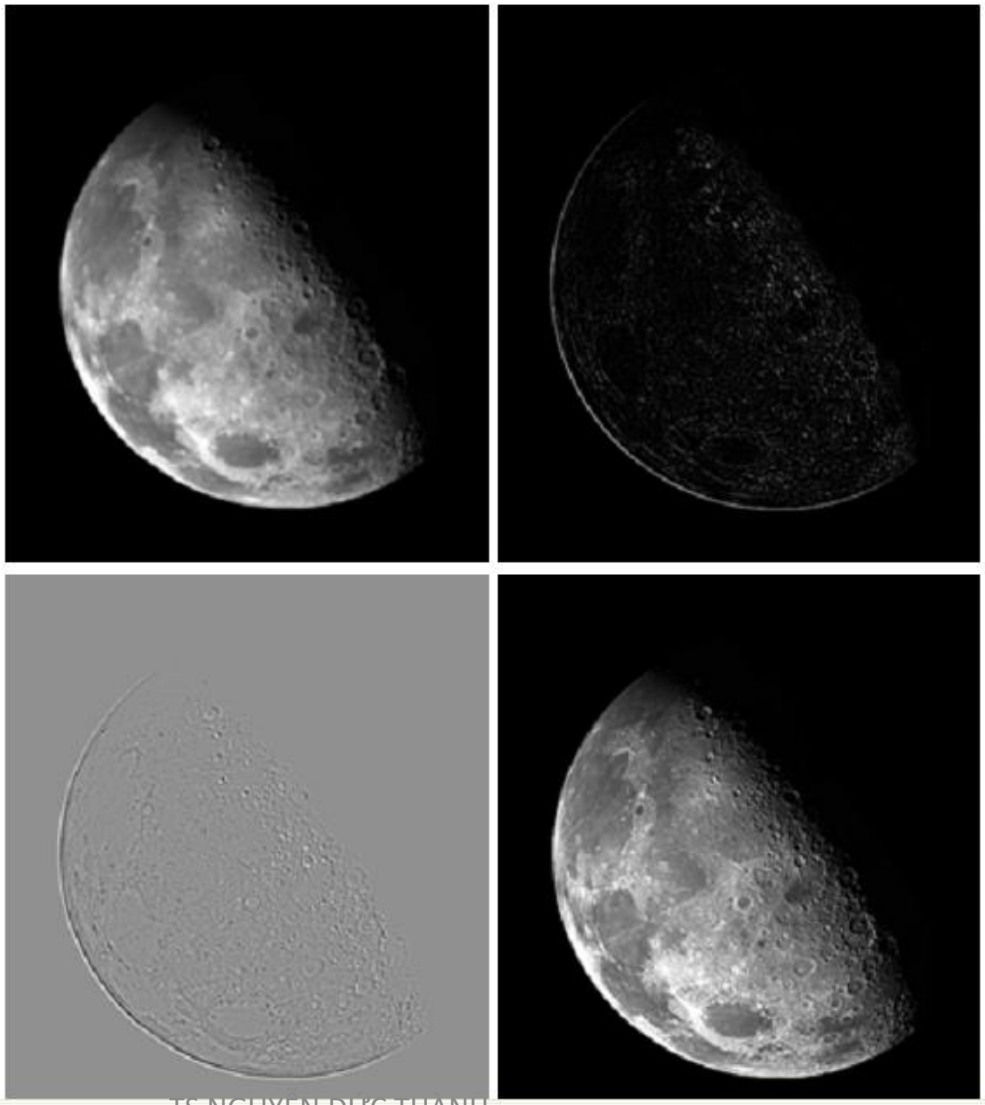
- Lọc Laplace làm chi tiết nền có mức xám đều bị ảnh hưởng, để khắc phục ta dùng thuật toán sau:

$$g(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y) & \text{if the center coefficient of the Laplacian mask is negative} \\ f(x,y) + \nabla^2 f(x,y) & \text{if the center coefficient of the Laplacian mask is positive.} \end{cases}$$

MẶT NẠ LAPLACE

a b
c d

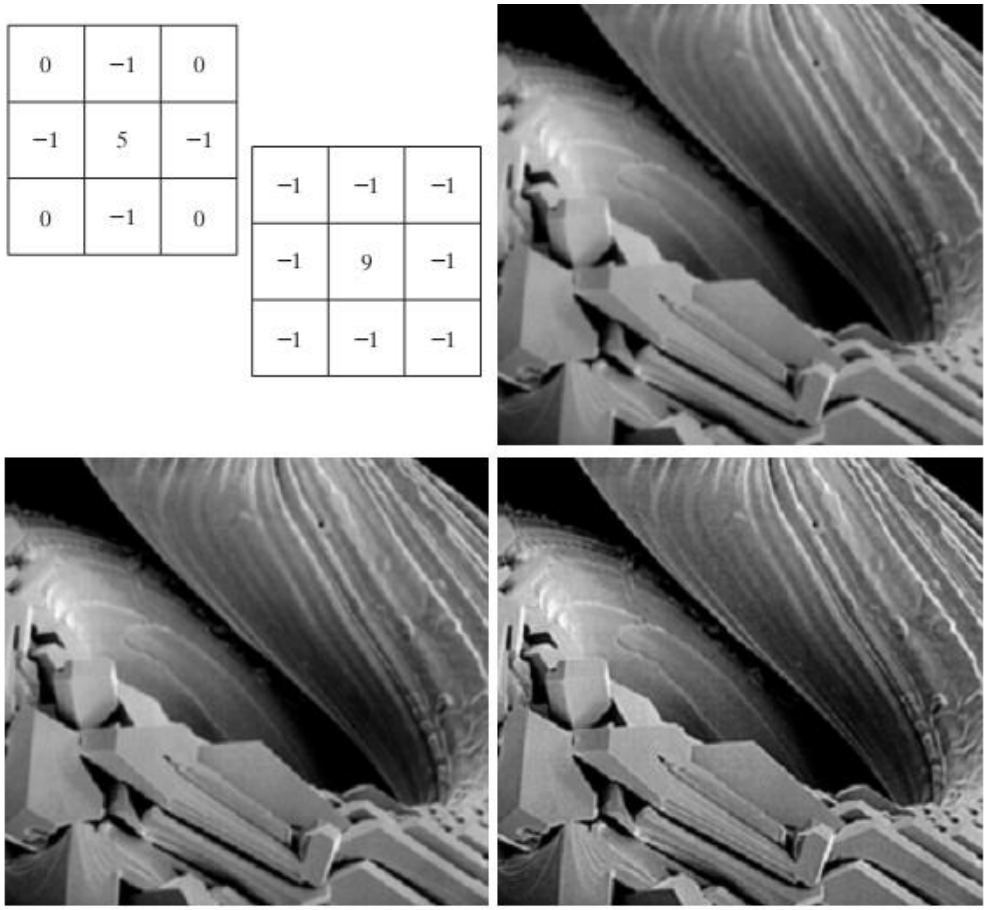
(a) Image of the North Pole of the moon.
(b) Laplacian-filtered image.
(c) Laplacian image scaled for display purposes.
(d) Image enhanced by using Eq. (3.7-5).
(Original image courtesy of NASA.)



MẶT NẠ LAPLACE

0	−1	0
−1	5	−1
0	−1	0

−1	−1	−1
−1	9	−1
−1	−1	−1



a b c
d e

FIGURE 3.41 (a) Composite Laplacian mask. (b) A second composite mask. (c) Scanning electron microscope image. (d) and (e) Results of filtering with the masks in (a) and (b), respectively. Note how much sharper (e) is than (d). (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene.)

Laplacian

```
void cv::Laplacian(
  cv::InputArray  src,                    // Input image
  cv::OutputArray dst,                    // Result image
  int             ddepth,                 // Depth of output image (e.g., CV_8U)
  cv::Size        ksize      = 3,         // Kernel size
  double          scale      = 1,         // Scale applied before assignment to dst
  double          delta      = 0,         // Offset applied before assignment to dst
  int             borderType = cv::BORDER_DEFAULT  // Border extrapolation to use
);
```

```
Mat src, gray, dst, abs_dst;
    src = imread( "lena.jpg" );
     /// Remove noise by blurring with a Gaussian filter
    GaussianBlur( src, src, Size(3,3), 0, 0,
BORDER_DEFAULT );
    cvtColor( src, gray, CV_RGB2GRAY );
     /// Apply Laplace function
    Laplacian( gray, dst, CV_16S, 3, 1, 0,
BORDER_DEFAULT );
    convertScaleAbs( dst, abs_dst );
    imshow( "result", abs_dst );
```

UNSHARP MASKING, HIGHBOOST FILTERING

- Unsharp masking: lấy ảnh gốc trừ ảnh bị mờ

$$f_s(x, y) = f(x, y) - \bar{f}(x, y)$$

High boost filter:

$$f_{hb}(x, y) = Af(x, y) - \bar{f}(x, y)$$

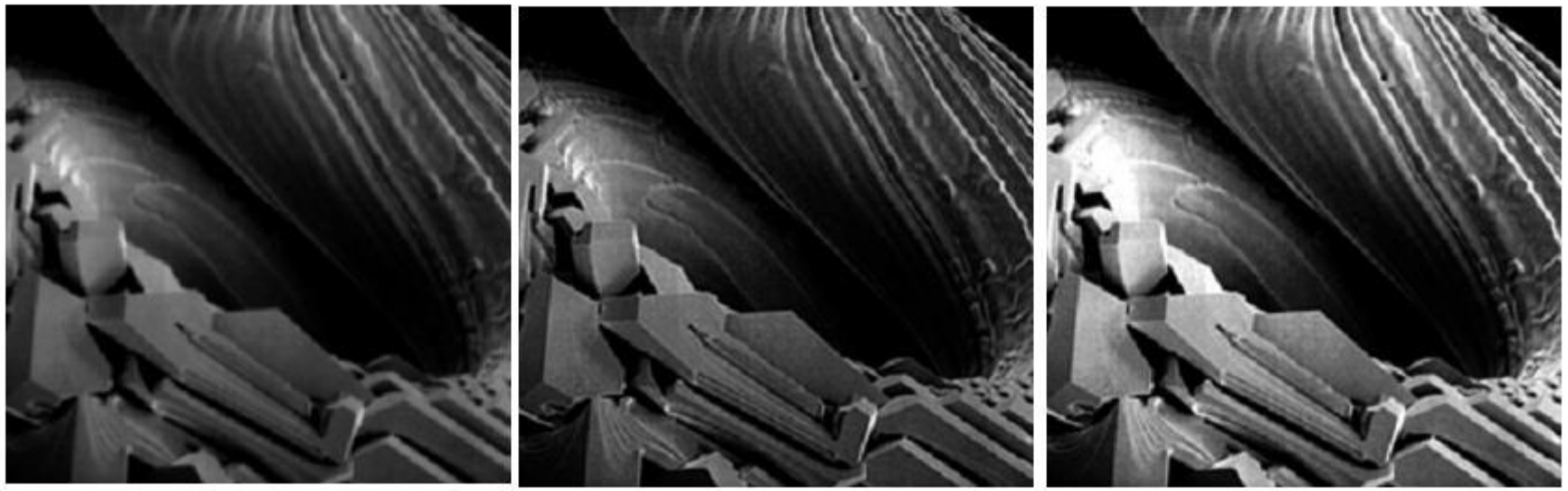
0	−1	0
−1	$A + 4$	−1
0	−1	0

−1	−1	−1
−1	$A + 8$	−1
−1	−1	−1

a b

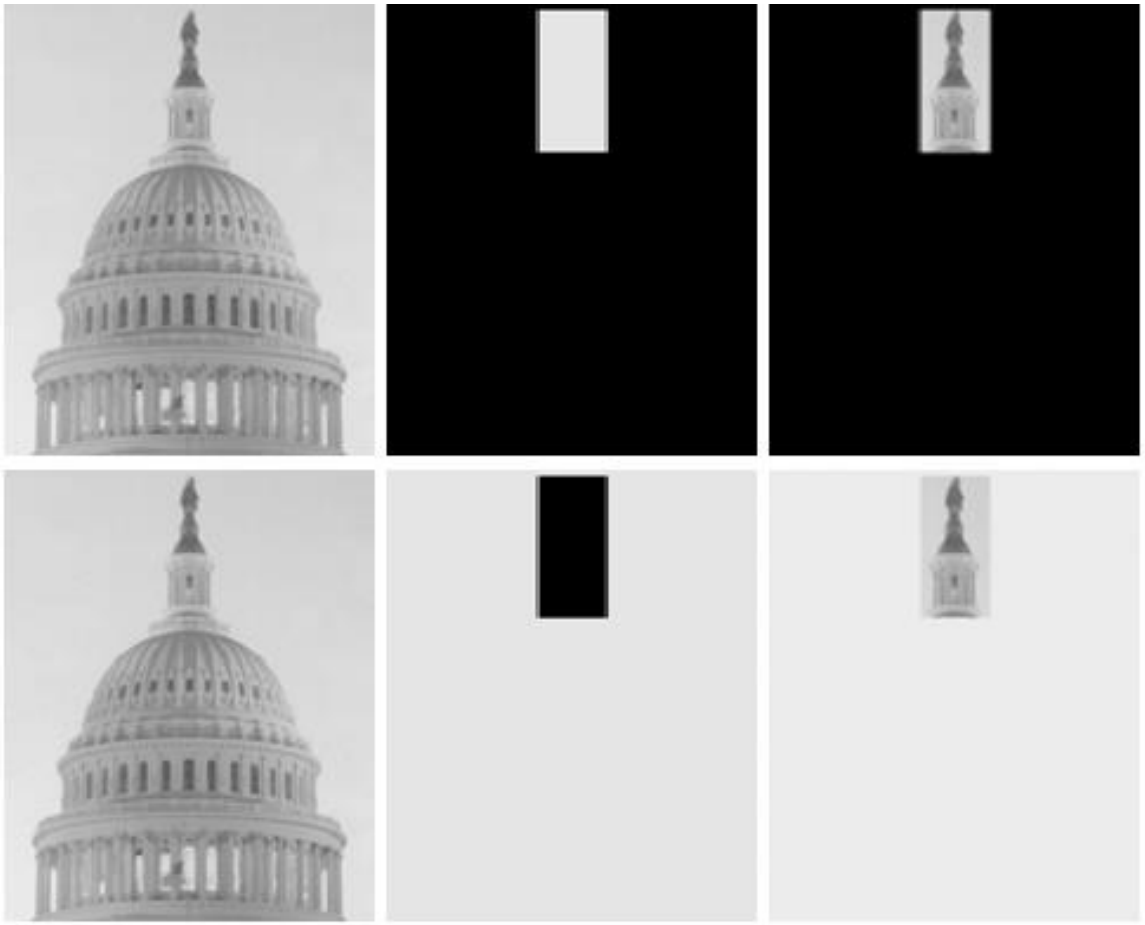
The high-boost filtering technique can be implemented with either one of these masks, with $A \geq 1$.

HIGHBOOST FILTERING



Ảnh gốc A=1 A=1,7

TĂNG CƯỜNG ẢNH DÙNG PHÉP TOÁN LOGIC



a b c
d e f

(a) Original image. (b) AND image mask. (c) Result of the AND operation on images (a) and (b). (d) Original image. (e) OR image mask. (f) Result of operation OR on images (d) and (e).