

**Laboratory Objective**

The primary goal of this laboratory is to introduce some basic concepts in image processing. The laboratory will focus on how to read and display an image in MATLAB, perform simple edge detection on an image that this laboratory gave as well as on our image. Also, understanding and implementing of scaling an image to create its thumbnail version is required.

This lab is also to prepare you for the higher-level class related to Image Processing that is EE440

\*Resources used for this laboratory consist of MATLAB software (version 2021b), the image ‘DailyShow.jpg’ in the JPEG format

**Table of Contents**

[**1. Background 3**](#_heading=h.akm679i4xvid)

[**2. Assignment 4**](#_heading=h.9afdko2b40r2)

[2.1. Edge Detection (25%) 4](#_heading=h.h5cjz6dea64i)

[2.2. Scaling (25%) 7](#_heading=h.fjjxwxdkxmjt)

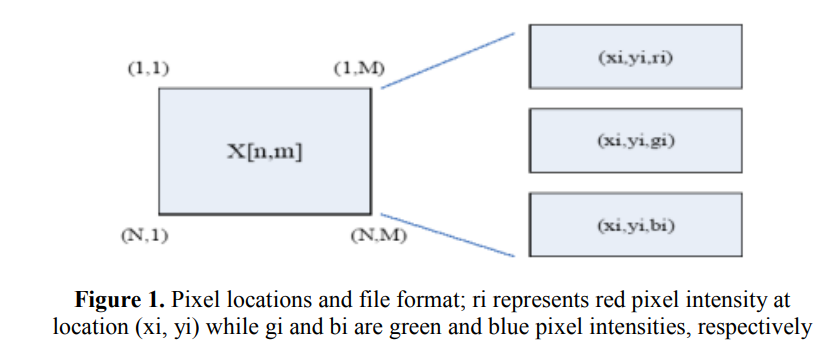
[2.3. Image Flipping (25%) 10](#_heading=h.jf3yy161zoep)

[2.4. Image Expanding (25%) 12](#_heading=h.8t5jo0divwan)

[**3. Conclusions 15**](#_heading=h.14gyzl6zy0fv)

# **Background**

Digital images consist of pixels (picture elements). When the pixels are placed close to each other, an image viewed on a computer display or printed on a paper appears to be continuous. The number of pixels per inch (ppi) varies with an application. Some monitors can only display 72 ppi. For publishing, 200–1200 ppi is often required. Laser printers are usually capable of 300-600 ppi. The brightness and color information for each pixel is represented by a number in a two dimensional array (matrix). The location in the matrix corresponds to the location of a pixel in the image. For example, x(1,1) (usually) identifies the pixel located in the upper left corner, as shown in Figure 1.



The pixel values in an 8-bit gray scale image can take any value from 0 to 255. Usually black is encoded by a value of 0 and white by a value of 255. A color image is stored in a three-dimensional array, where the first plane in the 3rd dimension represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities. True color has 24 bits of resolution (8 bits for each of the red, green, and blue planes).

In this lab, we will use the **Image Processing Toolbox**.

# **Assignment**

## **2.1. Edge Detection (25%)**

The ambition of this assignment is to introduce the edge detection technique by using MATLAB.

First, the image was loaded into MATLAB using the ‘***imread(‘DailyShow’, ‘jpeg’)’*** command. The `***imshow()***` function was employed to display the image. Since we will be working with gray scale images in this lab, our next step is to convert the input image to an 8-bit gray scale format using the ‘***rgb2gray()***’ command. Fig.2 displayed the image before and after converting to 8-bit grayscale. The dimensions of the grayscale image were checked using ***‘size()***’ command and showed below

**Original image size:**

**400 468 3**

**Gray image size:**

**400 468**

As we can see, by using function ‘***rgb2gray()***’, we can convert a RGB image (which consists of three array(:,:,1:3)) into a grayscale version (which consists of single array (:,:,1))



**Figure 2**: Original image and Gray scale image

Second, we will perform edge detection on your image. (Edge detection is often a first step into later, more complicated image processing operations.) Two dimension-

al convolution, appropriate for images, is implemented in MATLAB in the function conv2(). Convolution can be used to implement edge detection.

Create the following Sobel vertical edge detection convolution kernel. This mask is designed to respond maximally to edges running vertically relative to the pixel grid. It is a two-dimensional matrix h1[n,m], in Matlab notation:

h1 = [-1 0 1; -2 0 2; -1 0 1]

Next create the following Sobel horizontal edge detection convolution kernel. This mask is designed to respond maximally to edges running horizontally relative to the pixel grid. It is a two-dimensional matrix h2[n,m], in Matlab notation:

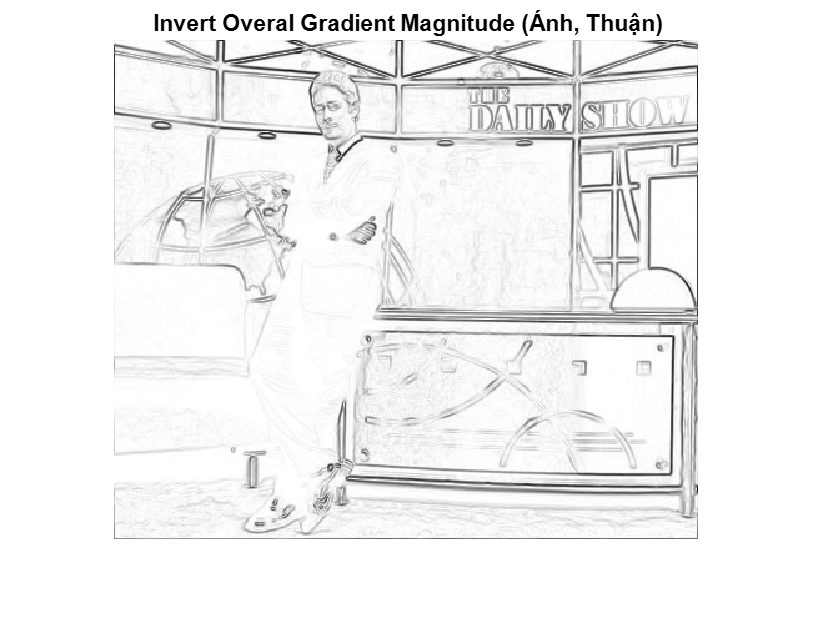
h2 = [1 2 1; 0 0 0; -1 –2 -1]

Now, convolve grayscale DailyShow image with the two edge detection kernels described as follows. Assume M1 is the result of convolving the grayscale DailyShow image with h1 (*i.e. M1 is the row gradient of the grayscale DailyShow image*), and M2 is the result of convolving the grayscale DailyShow image with h2 (*i.e. M2 is the column gradient of the grayscale DailyShow image*). MATLAB is used to display the row gradient magnitude (|M1|), the column gradient magnitude (|M2|), and the overall gradient magnitude (i.e. (M1^2 + M2^2)^0.5). All edge image of this part is presented in Fig.3

To save toner usage due to dark areas in edge images, making (1 - overall gradient magnitude) image was devised to invert the grayscale of the images, transforming darker areas to lighter ones and vice versa. This image also add in Fig.3

The code for this part is located in the file ‘EdgeDetection.m’ that we attach with this report.





**Figure 3**: Edge detection

Results:

The grayscale image of the "*DailyShow.jpg*" resulted in a clear, detailed image, revealing the image structure without color distractions. The dimensions of the grayscale image are [400;468;1].

The application of the Sobel edge detection kernels effectively highlighted the edges within the image (Fig.3). The gradient magnitude image provided a complete representation of the edges in the image, showing both vertical and horizontal edges with clarity.

Inverting the gradient images for printing purposes resulted in images that were lighter in black areas previously, which reduced the toner required for printing these images.

## **2.2. Scaling (25%)**

The objective of this section was to explore image scaling techniques in the spatial domain. By applying different scaling factors, we aimed to create thumbnail-sized versions of the original "*DailyShow.jpg*" image. This process involves both simple and advanced scaling techniques, allowing for a comparative analysis of their impacts on image quality. We are going to scale the original image with scaling factors S = 2 and S = 5.

*Simple scaling*: keep one out of S2 pixels. Since this is a 2D scaling, we keep the center pixel in each square of S2 pixels, when S is an odd number and one of the 4 center pixels when S is even. Below is a MATLAB function perform a simple scaling

* **function simpleScaledImage = SimpleScaling(S)**
* **% Read image and convert to 8-bit gray format**
* **img = imread('DailyShow.jpg','jpeg');**
* **img = rgb2gray(img);**
* **% Calculate the size of new image**
* **[rows, cols] = size(img);**
* **newRows = ceil(rows/S); % ceil: bounded upward rows/S (integers)**
* **newCols = ceil(cols/S);**
* **simpleScaledImage = zeros(newRows, newCols, 'uint8'); % create new image matrix = 0**
* **for i = 1:newRows**
* **for j = 1:newCols**
* **if mod(S, 2) == 1**
* **% S odd: keeping the center pixel**
* **rowIdx = (i-1)\*S + floor(S/2) + 1;**
* **colIdx = (j-1)\*S + floor(S/2) + 1;**
* **else**
* **% S even: one of the 4 center pixels (left)**
* **rowIdx = (i-1)\*S + (S/2);**
* **colIdx = (j-1)\*S + (S/2);**
* **end**
* **simpleScaledImage(i, j) = img(rowIdx, colIdx);**
* **end**
* **end**
* **% Plot image**
* **figure;imshow(simpleScaledImage);**
* **title(['Simple Scaling with S = ', num2str(S), ' (Ánh,Thuận)']);**
* **end**

Next, we are going to perform a more advanced scaling operation. Instead of keeping the center pixel in each square of S2 pixels, keep the average of all of the pixels in this square. The function for advanced scaling is shown below.

* **function advancedScaledImage = AdvancedScaling(S)**
* **% Read image and convert to 8-bit gray format**
* **img = imread('DailyShow.jpg','jpeg');**
* **img = rgb2gray(img);**
* **% Calculte the size of new image**
* **[rows, cols] = size(img);**
* **newRows = ceil(rows/S); % ceil: bounded upward rows/S (integers)**
* **newCols = ceil(cols/S);**
* **advancedScaledImage = zeros(newRows, newCols, 'uint8'); % create new image matrix = 0, uint8: 0 -255**
* **for i = 1:newRows**
* **for j = 1:newCols**
* **% Calculate S\*S block**
* **rowStart = (i-1)\*S + 1;**
* **colStart = (j-1)\*S + 1;**
* **rowEnd = min(rowStart + S - 1, rows); % using min to avoid un-boundary edge**
* **colEnd = min(colStart + S - 1, cols);**
* **child\_block = img(rowStart:rowEnd, colStart:colEnd);**
* **advancedScaledImage(i, j) = mean(child\_block(:));**
* **end**
* **end**
* **% Plot image**
* **figure;imshow(advancedScaledImage);**
* **title(['Advanced Scaling with S = ', num2str(S), ' (Ánh,Thuận)']);**
* **end**

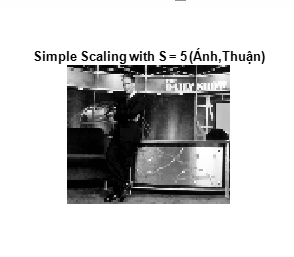
The code for this part is located in the file ‘Scaling.m’ that we attach with this report and scaling images are displayed in Fig.4.

Results:

The scaling operations successfully produced thumbnail images of the original “DailyShow” image for specified scaling factors (S = 2 and S = 5) (Fig.4).

From Fig.4 (page 10), we can clearly see that the advanced scaling method resulted in thumbnails that are noticeably smoother and retain more detail (offering a balance between reduction and detail preservation). This was particularly evident in lager factor (S = 5), where the simple scaling method is limited.





**Figure 4:** Scaling images

## **2.3. Image Flipping (25%)**

The objective of this experiment is to understand and apply image flipping operations to the “DailyShow” image (vertically, horizontally, and both vertically and horizontally)

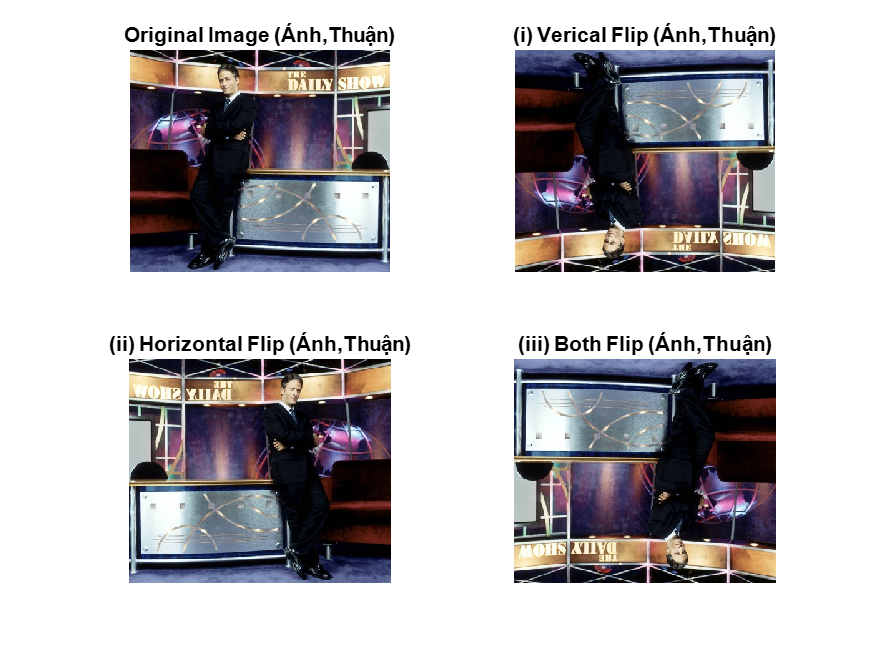
First, we consider the original image x[n,m] and guess how the following images look like in three cases. (where 1≤ n≤ , 1≤ m≤ M)

(i) x[N-n+1, m] : the change take action in row (N-n+1) which inverted the pixel in row (first-last, second-(last-1),..), so we guess the image will flip vertical

(ii) x[n, M-m+1] : the change take action in column (M-m+1) which inverted the pixel in column (first-last, second-(last-1),..), so we guess the image will flip horizontal

(iii) x[N-n+1, M-m+1] : the change take action both in row (N-n+1) and in column (M-m+1), so we guess the image will flip horizontal

Now, by applying ‘*fliplr()’* and ‘*flipud()’* commands in MATLAB, we can verify our guess and display images (i)(ii)(iii). The code for this part is located in the file ‘Flipping.m’ that we attach with this report.



**Figure 5**: Flipping images

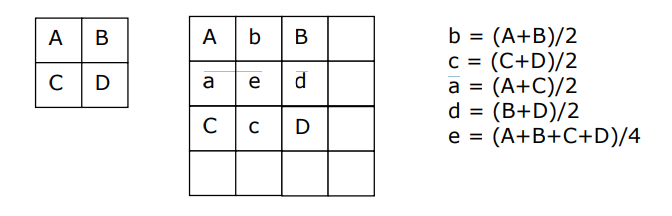
Results:

The image verifies that our guesses are true. The (i) resulted in an image that appears to be mirrored along the horizontal axis, with the top of the original image becoming the bottom and vice versa. The (ii) produced an image mirrored along the vertical axis, making the left side of the original image appear on the right and vice versa. And the (iii) operation effectively rotated the image 180 degrees, creating an upside-down version of the original.

## **2.4. Image Expanding (25%)**

The aim of this section is to explore and implement bilinear interpolation for image expansion. By applying this method to the "DailyShow" image, the goal is to upscale the image by a factor of 2 in both dimensions.

*Bilinear interpolation* is a technique used for resampling to achieve a smooth transition between pixels. It calculates the value of new pixels based on a weighted average of the 4 nearest original pixels, providing a more natural-looking result in the expanded image (example of “*Bilinear interpolation”* can see in Fig.6).



**Figure 6**: Enlarging the image by a factor of 2 using bilinear interpolation

Here is a MATLAB function that can expand the input image “DailyShow.jpg” by using “*Bilinear interpolation”*

* **function Expanding()**
* **% Read the original image**
* **originalImage = imread('DailyShow.jpg');**
* **% Get the size of image**
* **[rows, cols, ~] = size(originalImage);**
* **expandedImage = zeros(2\*rows, 2\*cols, size(originalImage, 3));**
* **% Bilinear interpolation**
* **for row = 1:rows**
* **for col = 1:cols**
* **A = double(originalImage(row, col, :));**
* **B = double(originalImage(row, min(col+1,cols), :));**
* **C = double(originalImage(min(row+1,rows), col, :));**
* **D = double(originalImage(min(row+1,rows),min(col+1,cols), :));**
* **b = (A + B) / 2;**
* **c = (C + D) / 2;**
* **a = (A + C) / 2;**
* **d = (B + D) / 2;**
* **e = (A + B + C + D) / 4;**
* **expandedImage(2\*row-1,2\*col-1, :) = A;**
* **expandedImage(2\*row-1,2\*col+1, :) = B;**
* **expandedImage(2\*row-1, 2\*col, :) = b;**
* **expandedImage(2\*row, 2\*col-1, :) = a;**
* **expandedImage(2\*row+1,2\*col-1, :) = C;**
* **expandedImage(2\*row+1, 2\*col, :) = c;**
* **expandedImage(2\*row+1, 2\*col+1, :) = D;**
* **expandedImage(2\*row, 2\*col+1, :) = d;**
* **expandedImage(2\*row, 2\*col, :) = e;**
* **end**
* **end**
* **% Display the original and expanded images**
* **figure, imshow(originalImage), title('Original Image (Ánh,Thuận)');**
* **figure, imshow(uint8(expandedImage)), title('Expanded Image by Bilinear Interpolation (Ánh,Thuận)');**
* **end**

We also display the expanding image in Fig.7 in order to compare the size of this technique. The code for this part is located in the file ‘Expanding.m’ that we attach with this report.



**Figure 7**: Expanding 2Nx2M image

Results:

The expanded image, twice the size of the original in both dimensions, demonstrated the effectiveness of “bilinear interpolation” in upscaling technique. The transition between pixels appears smooth, with minimal visual artifacts or pixelation, especially in areas of gradual intensity change.

Comparing the original and expanded images side by side, it is evident that the bilinear interpolation maintained the integrity and clarity of the image's features.

# **Conclusions**

In laboratory 2, we focus on presenting some basic concepts in image processing. We explore various fundamental techniques by using MATLAB. These experiments, encompassing edge detection, image scaling, flipping, and expanding through bilinear interpolation, provided knowledge of image processing.