# Image Project

**Haiyan Cheng**
Department of Computer Science
Willamette University
Salem, OR 97301
hcheng@willamette.edu

**CS-435 Computational Science and Applications**

*Fall 2016*


## 1 Introduction

In this project you will learn the following:

1. How images are stored and represented in MATLAB.
2. How to use a 3-dimensional array.
3. How to load, resize, and save an image.
4. How to convert a color image to black-and-white image.
5. Two image processing techniques: Image repair and edge detection.
6. MATLAB image processing toolbox functions for different edge detection filters.

### 1.1 Image Representation

#### 1.1.1 Black-and-white Images

An image can be represented by individual dots. The dots are called picture elements (**pixels**). In MATLAB, an image is represented as a matrix. Each element in the matrix corresponds to a pixel in the image. Each pixel of a black-and-white image stores the appropriate amount of "grayness."

The Willamette clock tower shown in Figure 1 is a 681-by-1024 array of pixels. In MATLAB, it is represented by a matrix of size $681 \times 1024$, where the matrix element $(i, j)$ shows the grayness of the pixel $(i, j)$. Notice that there is a difference between the regular coordinate system and the MATLAB matrix representation. In the MATLAB representation, the rows and columns start from index $(1, 1)$ using top-down and left-right directions. The level of grayness is represented by integers from 0 to 255, with 0 corresponding to black and 255 corresponding to white.

MATLAB has a data type `uint8` to store pixel values. The uint8 is a 1-byte, unsigned integer format which is more economical than the 8-byte double format which supports floating-point arithmetic.

**Figure 1.** A Black-and-White Picture is a Matrix in MATLAB

### 1.1.2 Color Images

There are two different methods to represent color images in MATLAB : **truecolor** and **colormap indexed**. The truecolor images are generated by digital cameras and are widely used in computer graphics. The indexed images are often used to display scientific and engineering data with an associated color scale representing the data units.

In a truecolor image, a particular pixel stores the RGB color for that pixel. So instead of one entry for the "grayness" in the black-and-white image, three entries are needed to store the "redness", "greenness", and "blueness". In a sense, a color image is a three-dimensional array of size M-by-N-by-3, or three matrices of the same size M-by-N, representing its RGB colors, as shown in Figure 2.

**Figure 2.** A Truecolor Image is Three Matrices in MATLAB

The other method to represent the color is to index into a **colormap**: the index is an integer that maps to a row in a color map matrix. The color map itself stores the red, green, and blue components in three separate columns. We have used this approach in the physics project for simulating a multiple-particle Brownian motion.

### 1.2 Examples of Color Indexing

The following example shows the current colormap (default name is jet), its size (how many colors are there in the colormap), and the first five colors of various shades of blue.

```
map = colormap
[r, c] = size(map)
map(1:5, :)
```

The following MATLAB code loads an image, and displays the image using the index to a specified colormap. Change "map" to different colormaps to see the effect.

```
load mandrill
figure('color','k')
image(X)
colormap(map)
axis off          % Remove axis ticks and numbers
axis image        % Set aspect ratio to obtain square pixels
```

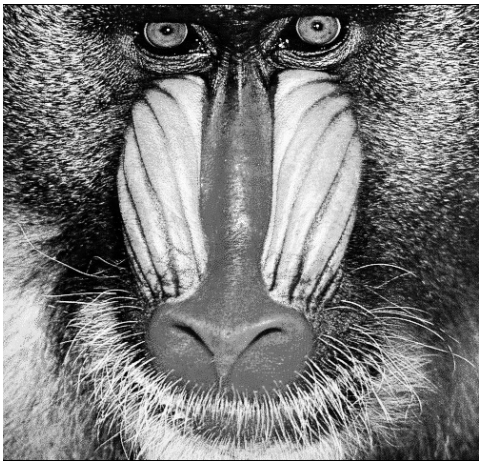
```
% Try changing the colormap
map = colormap(hsv);  % other colormap incude: gray, hot, cool, bone, copper,
                      % pink, jet, prism, flag, etc.
```

**Figure 3.** Color indexing in Matlab with different color maps (original truecolor, hsv, bone, gray)

Original truecolor

hsv

bone



gray

### 1.3 Image Repairing and Edge Detection

Digital image processing is to use computer algorithms to perform image processing on digital images. In image processing, both input and output are images. There are also techniques called image analysis and image understanding. In image analysis, the output can be analysis data or statistics. In image understanding, the output can be high-level descriptions about the image. Other related image techniques include image classification, image compression, feature extraction, pattern recognition etc. Each technique has a specific goal or desired outcome of the image. In this project, we will learn algorithms and techniques to repair a contaminated image by cleaning the image and detect the edge of the objects in a given image.

Because a picture is represented in MATLAB as a matrix, image processing involves numerical computing with the underlying pixel values. For example, if a black-and-white picture is contami-

nated with specks of dirt, then it will have random pixels with grayness that is very different from its neighbors. In that case, the image processing procedure involves "repairing the damage" by replacing the errant pixel value with a number that is more consistent with its pixel neighbors.

Another image processing procedure concerns the problem of edge detection. Pixel values change rapidly near the edge of an object in the picture. For example, along the "bright side" of the object, the RGB triplet may be $(215, 232, 186)$, while on the "dark side" the color may be $(51, 29, 37)$. By computing differences between nearby pixels and setting an appropriate threshold, it is possible to locate the object edges.

## 2 In-Class Project

### 2.1 Basic Image Manipulations

The basic image manipulations in MATLAB involve reading the image into MATLAB, checking image size, choosing a smaller area to create a smaller image, and extracting the red, green and blue components of the original image.

In the in-class project part 1, we will learn MATLAB built-in functions for basic image processing. Using the given sample image: Willamette.jpg, perform the following tasks:

1. Read the image into MATLAB and save it to a variable;
2. Show the image;
3. Check the size of the image (should be M-by-N-by-3 for a color image);
4. Crop the image to extract the desired part, and save it to a different image;
5. Extract the red, blue and green components and save them to individual images.
6. Convert the image to black-and-white;
7. (Advanced) Use a smaller image to create a large image that contains the original image, the red, green and blue components stacked up as 2-by-2 blocks, for example in figure 4.

**Figure 4.** Stacked Images

In the following, you will learn two image processing techniques: image repair through "median filter" and image edge detection. Both image processing tasks involve looking at local pixel variations.

### 2.2 Image Repair

**Write a MATLAB program to clean up the dirty specks in the provided image.**

The idea of the cleaning process is to replace the "dirty" pixel values with values that are more typical in its surrounding neighborhood. We say that pixels $(i_1, j_1)$ and $(i_2, j_2)$ are neighbors if $|i_1 - i_2| \leq 1$ and $|j_1 - j_2| \leq 1$. With this definition, we see that an interior pixel has nine neighbors (including itself) while an edge pixel has either four or six neighbors depending on if it is in the corner. The idea of the filtering process is to visit each pixel in the image and replace its value by the median value of its neighbors. This process is called "median filtering."

The following shows an example of a contaminated 6-by-6 patch before and after a median filtering process:

$$\begin{bmatrix} 224 & 200 & 219 & 225 \\ 228 & 0 & 7 & 217 \\ 231 & 1 & 10 & 216 \\ 226 & 203 & 221 & 225 \end{bmatrix} \rightarrow \begin{bmatrix} 215 & 219 & 217 & 217 \\ 214 & 200 & 200 & 217 \\ 214 & 203 & 203 & 217 \\ 216 & 221 & 216 & 220 \end{bmatrix} \tag{1}$$

Notice how the contaminated values (which are much smaller than surrounding values) get updated to become similar to the surrounding values, while the original values that are similar to their neighbor values do not change much.

**Note:** The median is a much better choice than the mean value in the image repair example. If the mean value update were used, the contaminated patch would become the following:

$$\begin{bmatrix} 224 & 200 & 219 & 225 \\ 228 & 0 & 7 & 217 \\ 231 & 1 & 10 & 216 \\ 226 & 203 & 221 & 225 \end{bmatrix} \rightarrow \begin{bmatrix} 195 & 173 & 172 & 196 \\ 169 & 124 & 122 & 174 \\ 170 & 125 & 122 & 174 \\ 197 & 175 & 173 & 197 \end{bmatrix} \tag{2}$$

Notice that the uncontaminated region is affected significantly due to the averaging effect, which can significantly affect the basic feature of the original image, and this is not what we wanted. Figure 5 shows the effect of a median filter for cleaning the contaminated image. Matlab has a built-in function for the median filter that does similar job: medfilt2.

**Figure 5.** Before and After Image Repairing

In the in-class project part 2, please finish the following tasks:

1. Read in and display the contaminated image: Willamette_bw_dirty_small.jpg, which can be downloaded from the WISE site under project 5, image folder.
2. Perform a median filter for each of the pixel value.
3. Display the repaired image.
4. Try to use the mean instead of median to generate a new filtered image for comparison.
5. Use MATLAB built-in function medfilt2 for similar image repair result.
6. (Advanced) Write the median filter into a function: medianfilter, which takes a contaminated image as input and produces a repaired image as output.

### 2.3 Image Edge Detection

**Write a MATLAB program to detect the edges of objects in the given image.**

Image edge detection involves identifying object edges based on pixel value changes. Different from the image repair problem, the goal of edge detection is to use density discontinuities rather than clean them up. Using a 3-by-3 patch, the pixel value rate of change $\rho_{ij}$ at location $(i, j)$ is

defined as the *maximum discrepancy* between $A(i,j)$ and its neighbors: $A(i-1,j-1)$, $A(i-1,j)$, $A(i-1,j+1)$, $A(i,j-1)$, $A(i,j+1)$, $A(i+1,j-1)$, $A(i+1,j)$, and $A(i+1,j+1)$ (with the assumption that the pixel $(i,j)$ is not at the edge). For example, if

$$A(8:10,20:22) = \begin{bmatrix} 240 & 26 & 18 \\ 237 & 242 & 31 \\ 236 & 241 & 241 \end{bmatrix} \tag{3}$$

then $\rho_{9,21} = |242 - 18| = 224$.

The edge detection algorithm involves a threshold value $\tau$, and defines pixel $(i,j)$ as a "pixel of interest" if $\rho \geq \tau$. It is possible that a "dirt speck" has a high rate of change and can be identified mistakenly as a pixel of interest. However, edges have structures, if all the curves and edges are highlighted, we expect to see curves/lines near the true edges. By setting an appropriate threshold value $\tau$, we can reconstruct an image such that the pixel $(i,j)$ is black (0) if $\rho_{ij} < \tau$ and white (255) if $\rho_{ij} \geq \tau$. Figure 6 shows an edge detection result with $\tau = 50$. Matlab provides a built-in function **edge** to implement different edge detection algorithms that include sobel, prewitt, canny, roberts, log, and zerocross. You can try different edge detection algorithms in the project.

In the in-class project part 3, please finish the following tasks:

1. Implement the edge detection algorithm described above. Basically, for each pixel in the original image, if the maximum discrepancy of the pixel with respect to its neighbor is greater than a predefined threshold $\tau$, color it to black, otherwise color it to white.
2. Display the pattern created from the original image. This is the image with the edge detected.
3. Use the MATLAB built-in function for edge detection. Read the help document for edge command and try different algorithms to see the results. For each result, display the image title with corresponding algorithm name.

**Figure 6.** Before and After Edge Detection

## 3 Lab Projects

In this lab project, you will practice image manipulation using MATLAB matrix format and learn how to use MATLAB image processing toolbox with different edge detection algorithms.

### 3.1 Image Manipulation

1. Write a function **ShowNeg(A)** that converts a given image in .jpg format into its negative. Your program should be able to handle black-and-white input images directly, and convert color images into black-and-white images before producing the negative.
2. Write a function **DisplayMosaic(A, m, n)** that displays a m-by-n mosaic of the color picture A.
3. Write a function **Rotate90L(A)** to rotate an input image A 90 degrees to the left.
4. Write a function**FlipLR(A)** that flips the picture A left to right.
5. Write a function **Rotate90R(A)** to rotate an input image A 90 degrees to the right.
6. Use the provided test code **Lab5_test.m** (downloadable from WISE) to test your functions.

### 3.2 Edge Detection with Image Processing Toolbox

Using the provided **Crocus.jpg** image, do the following:

1. Resize to your desired dimension
2. Convert it to black-and-white
3. Apply the algorithm we used in class to detect the edge and display the result.
4. Use the MATLAB image processing toolbox command edge to perform edge detection (Read the help document first.) Display the result with proper title corresponding to the algorithm used. Use all the available filters and compare the results. Additionally, you may play with different threshold (level) to obtain the result that you think is the best.
5. Perform the same operations with your favorite picture.

### 3.3 Background Change

1. Remove the background of a given picture while preserving the foreground image. Use the **cat.jpg** as an example to convert the light yellow background to a white background, so that it looks like the image on the top right of this document. (Extra: test your algorithm on the **moo_goat.png** file, if your original algorithm does not work, design a better one. )
2. Frequently, you want to include the university logo in your presentation. In case you want to use a specific colored background, you want the logo to have the same background to match the surroundings. Write a function to change a given image with white background to a new background color with RGB color provided by the user. You may use the **Willamette_logo_255_255_255.png** and change its background to light blue which has corresponding RGB color: [235, 241, 246].

### 3.4 Lab Project Submission

Please create folders for each problem. If there are source data required to run your program, you need to put the data under the same folder, so that as soon as I unzip your code, I can test your code. Please zip all the folders that contain programs you want to submit and name it `YourFirstname_P5.zip`, submit the zipped file to the WISE dropbox before 11:59PM on Nov 9. You need to demonstrate your program on or before Nov 10's lab and answer questions in order to get credits.