Machine Vision

Fundamentals of Image Representation and I/O in MATLAB

Lab Activity Sheet 1

Author: Dr. Mostapha Kalami Heris

Introduction

This activity sheet introduces the fundamental concepts of image processing in MATLAB. Reading, displaying, and writing images are the most basic yet essential tasks in machine vision. Before performing advanced analysis, it is important to understand how MATLAB represents and handles image data.
In MATLAB, an image is stored as a numerical array:

- **Grayscale images**: two-dimensional arrays ($M \times N$).
- **Truecolor (RGB) images**: three-dimensional arrays ($M \times N \times 3$), with red, green, and blue components.
- **Binary images**: logical values (0 or 1) for black and white pixels.
- **Indexed images**: an index matrix with an associated colormap.

Different data types may be used, such as `uint8` (0–255), `uint16` (0–65,535), or `double` (scaled between 0 and 1). Choosing the correct type is essential for correct display and processing.
Displaying an image in MATLAB depends on the function used. The function `imshow` shows raw data values, whereas `imagesc` rescales the data to the display range. This difference becomes important when working with floating-point or scaled images.
Finally, images can be stored in different **file formats** such as JPEG, PNG, and TIFF. JPEG uses lossy compression, reducing file size but losing detail, whereas PNG and TIFF are lossless formats that preserve image quality.
This lab session builds the foundation for later topics in the module. By completing the activities, students will gain practical experience with MATLAB image functions, explore image formats and channels, and develop skills to manipulate and analyze images confidently.

Available Images in MATLAB

For this lab session, you are encouraged to experiment with a variety of predefined images that are included with MATLAB. The following images are available in the `imdata` folder:

- `ngc6543a.jpg` (Cat's Eye Nebula, RGB)
- `corn.tif` (Corn, RGB)
- `peppers.png` (Peppers, RGB)
- `coloredChips.png` (Coloured chips, RGB)
- `rice.png` (Rice grains, grayscale)
- `pout.tif` (Portrait image, grayscale)
- `llama.jpg` (Llama photo, RGB)
- `pears.png` (Pears, RGB)
- `cameraman.tif` (Classic test image, grayscale)
- `greens.jpg` (Green vegetables, RGB)
- `hallway.jpg` (Hallway photo, RGB)
- `saturn.png` (Saturn planet image, RGB)
- `trailer.jpg` (Trailer photo, RGB)

Instruction: Apply each of the activities in this sheet not only to the default image (`cameraman.tif`) but also to at least **two other images** of your choice from the list above. Compare the results across different images and comment on any differences (for example, behaviour of grayscale vs RGB images, or the effect of compression).

Activity 1: Reading, Displaying, and Writing Images

Objective: Build core literacy for reading, displaying, inspecting, and writing images in MATLAB.

Description

This activity introduces how MATLAB stores and visualises images. You will read images from disk, inspect array size and data type, display them correctly using `imshow` and `imagesc`, and write images to common file formats. You will also examine metadata using `imfinfo`. Mastering these basics is essential for pre-processing and analysis tasks later in the module.

Function notes

- `imread(filename)`: load an image into an array ($M \times N$ grayscale or $M \times N \times 3$ RGB).
- `imshow(A)`: display using native interpretation. For `double`, MATLAB expects values in $[0, 1]$.
- `imagesc(A)`: scale data to the current colormap range; use `colorbar` and `colormap gray`.
- `size(A)`, `class(A)`: report array dimensions and data type.
- `imwrite(A, filename)`: write an image to disk in the format implied by the extension.
- `imfinfo(filename)`: file metadata (format, bit depth, resolution, file size).

Tasks

1. Read one image from the provided list, then report its `size` and `class`.
2. Display the image with `imshow` and with `imagesc`. Add `axis image off`, `colormap gray`, and `colorbar` where appropriate. Describe the differences you observe.
3. Save the image in at least two formats (for example, PNG and JPEG) using `imwrite`, then use `imfinfo` to compare file size and bit depth. Explain the differences.
4. Repeat steps 1–3 for at least two more images, including one grayscale and one RGB image.

Starter code

```
% Read and inspect:
filename = 'cameraman.tif';
I  = imread(filename);
sz = size(I)
tp = class(I)

% Display with imshow (native):
figure('Name','imshow (native)');
imshow(I);
title('imshow');

% Display with imagesc (scaled):
figure('Name','imagesc (scaled)');
imagesc(I);
axis image off;
colormap gray;
colorbar;
title('imagesc');

% Save in different formats:
imwrite(I, 'my_image.png');
imwrite(I, 'my_image.jpg');
imwrite(I, 'my_image.tif');

% Compare metadata:
info_in  = imfinfo(filename)
info_png = imfinfo('my_image.png')
info_jpg = imfinfo('my_image.jpg')
info_tif = imfinfo('my_image.tif')
```

Reflection

- How do the displays produced by `imshow` and `imagesc` differ for your images?
- How does the image `class` influence the display outcome?
- Which format preserves detail best for your chosen images and why?

Activity 2: Grayscale and Color Channels

Objective: Explore grayscale conversion and RGB channel manipulation in MATLAB.

Description

This activity focuses on converting color images to grayscale and inspecting individual RGB channels. By separating channels, you will see how red, green, and blue contribute to the overall image. You are also encouraged to experiment with swapping or removing channels to understand how color composition works.

Function notes

- `rgb2gray(A)`: converts an RGB image to grayscale.
- `im2gray(A)`: a safer alternative that works for RGB or grayscale input.
- `I(:,:,k)`: extracts the kth channel of an RGB image (1 = red, 2 = green, 3 = blue).

Tasks

1. Convert an RGB image (e.g., `peppers.png`) to grayscale using `rgb2gray` and `im2gray`.
2. Extract and display the red, green, and blue channels separately.
3. Swap two channels (for example, red and blue) and display the modified image.
4. Remove one channel (e.g., set it to zeros) and observe the effect.
5. **(Optional)** Create your own grayscale image by blending RGB channels manually:
   - Start with a simple average: $(R + G + B)/3$.
   - Try a weighted average, for example: $0.3R + 0.6G + 0.1B$.
   - Compare with the result of `rgb2gray`.

Starter code

```
% Read an RGB image
I = imread('peppers.png');

% Convert to grayscale
G1 = rgb2gray(I);
G2 = im2gray(I);
figure, imshow(G1), title('Grayscale (rgb2gray)');
figure, imshow(G2), title('Grayscale (im2gray)');

% Extract channels
R = I(:,:,1); G = I(:,:,2); B = I(:,:,3);
figure, imshow(R), title('Red channel');
figure, imshow(G), title('Green channel');
figure, imshow(B), title('Blue channel');

% Swap channels (red <-> blue)
I_swap = I(:,:,[3 2 1]);
figure, imshow(I_swap), title('Channels swapped (R <-> B)');

% Remove green channel
I_noG = I;
I_noG(:,:,2) = 0;
figure, imshow(I_noG), title('Green channel removed');
```

Reflection

- How do the grayscale images from `rgb2gray` and `im2gray` compare?
- What visual changes occur when you remove or swap channels?

Activity 3: Color Space Conversion

Objective: Explore color spaces and learn how to convert between RGB and HSV.

Description

Images are commonly represented in the RGB color space, where each pixel is described by its red, green, and blue components. While RGB is natural for display, it is not always the most convenient for analysis. The HSV (Hue–Saturation–Value) color space separates color information (hue) from intensity (value) and purity (saturation).

- **RGB**: well-suited for display and storage, but difficult for tasks such as color-based segmentation.
- **HSV**: often used for detecting or isolating colors, adjusting brightness, or image editing because hue is separated from intensity.

HSV channel definitions (from MATLAB documentation)

- **Hue**: Value from 0 to 1 that corresponds to the color's position on a color wheel. As hue increases from 0 to 1, the color transitions from red to orange, yellow, green, cyan, blue, magenta, and finally back to red.
- **Saturation**: Amount of hue or departure from neutral. 0 indicates a neutral shade, whereas 1 indicates maximum saturation.
- **Value**: Maximum value among the red, green, and blue components of a color.

Function notes

- `rgb2hsv(A)`: converts an RGB image to HSV.
- `hsv2rgb(A)`: converts an HSV image back to RGB.

Tasks

1. Read a color image (e.g., `peppers.png`) and convert it to HSV using `rgb2hsv`.
2. Display the three HSV channels separately (Hue, Saturation, Value).
3. Modify the saturation channel (e.g., reduce it by half) and convert back to RGB with `hsv2rgb`.

4. Compare the modified image with the original. What visual differences do you observe?
5. (Optional) Try isolating a specific color by thresholding the hue channel.

Starter code

```
% Read an RGB image and convert to HSV
I = imread('peppers.png');
HSV = rgb2hsv(I);

% Separate channels
H = HSV(:,:,1);  % Hue
S = HSV(:,:,2);  % Saturation
V = HSV(:,:,3);  % Value

% Display each channel
figure, imshow(H), title('Hue channel');
figure, imshow(S), title('Saturation channel');
figure, imshow(V), title('Value channel');

% Modify saturation (reduce by half)
HSV_mod = HSV;
HSV_mod(:,:,2) = S * 0.5;

% Convert back to RGB
I_mod = hsv2rgb(HSV_mod);

% Display comparison
figure, imshow(I), title('Original RGB');
figure, imshow(I_mod), title('Reduced Saturation (RGB)');
```

Reflection

- Why is HSV sometimes more useful than RGB for image analysis?
- How does reducing the saturation affect the appearance of the image?
- What applications can you think of where HSV would be preferable to RGB?