Faculty of Mechanical Engineering and Robotics

Department of Robotics and Mechatronics

Basics of AI and Deep Learning

Course for Mechatronic Engineering with English as instruction language

Instruction 6:

Computer Vision: An Introduction
and
Image Preprocessing Methods

You will learn: How images are represented in MATLAB, what are RGB greyscale and binary images, how to load, view and save images, how to manipulate image matrices and single pixels, what is the image histogram and how to understand and manipulate it, what is image filtering, what are linear filters and how they are realized mathematically, what are non-linear filters, what is image restoration – a practical applications of both types of filters, what are edges in the image and how they are detected, what are morphological filters applied to greyscale images, what is image segmentation and thresholding as the simplest segmentation method

Additional materials:

- Course lecture
- Exemplary images

Course supervisor:
Ziemowit Dworakowski, zdw@agh.edu.pl

Instruction author:
Krzysztof Holak, holak@agh.edu.pl

Introductory materials – working with images in MATLAB

The instructions will use materials available in the *VisionDatabase* archive – in our webpage.

Useful commands:

`cls` – clearing the MATLAB command window of commands and calculation results
`clear all` – clearing all variables from the MATLAB's workspace
`close all` – closing all opened figures
`ls` – list all files in the working directory

To load an image into MATLAB workspace we have to specify a variable in which we store image data and use command

```
A = imread('path');
```

We provide the full path to the image or the image file name with an extension if the image has been copied to the directory in which we are working. The image will be saved in a matrix variable A. Depending on the image we loaded it may be an RGB or greyscale image representation. To check the number of color channels and image size (image resolution), use command

```
im_size = size(A);
```

The vector im_size contains three numbers: M,N – number of pixels horizontally and vertically (image resolution) and C - the number of color channels.
For example, for color image you may get, for example, `im_size = [700 700 3];`
An image data is stored in a matrix of a size MxNxC for color images or MxN for greyscale ones. By default, three color channels represent color images in RGB color space, but color space representation may be changed to the one more appropriate for a given task.

To display an image, use command

```
figure,
imshow(A);
```

There are other display functions in MATLAB that show matrices as images using pseudocoloring. They are useful for viewing some image manipulation results and will be discussed in later laboratory exercises.

In MATLAB, you can work both in color as well as greyscale images. However, there are functions that work only with greyscale images. Therefore, all image operations will be done on greyscale images, unless stated otherwise. To change the color space from RGB to greyscale, use command

```
A_gray = rgb2gray(A);
```

Remember NOT to use this command on an image which is already in greyscale. You will get an error.

You can save your images after processing as image files on the disk. To do this, use

command (an example)
```
imwrite(A_gray, 'path');
```

Again, you can use the entire path to the new file or just the name of the new file with and extension – the image file will be saved in working directory.
Remember to use this command with caution, and not overwrite any existing image files!

Image matrix manipulations

You can easily access pixel data in the image matrix variable. For example, if you to crop a smaller image out of the original image, just specify the range of rows and pixels, as follows (an example)

```
B = A(20:100, 50:150);
```

In MATLAB, the origin of the image coordinate system is located in **the upper left corner** of the image. Therefore, **row indices** increase from top to bottom the image and **column indices** increase from left to right .

Now, see two small examples of cropping images:

1) To crop an image manually, select the upper left corner of the part of the image to be cropped, next choose the crop rectangle range, click RPM on its edge and select crop image from the menu

   ```
   [A_small rectangle] = imcrop(A);
   ```

   As output you get two variables: **`A_small`** – a matrix with cropped image pixel data and **`rectangle`** – a vector containing the upper left corner of the crop rectangle and its size (width and height)

1) For automatic cropping an image, you know the position and size of the part of the image to be cropped (e.g. when processing a set of the images), use the same command but with **`rectangle`** as the second parameter,

   ```
   rectangle = [x y width height];
   A_small = imcrop(A,rectangle);
   ```

In order to reach particular pixel data (its x and y coordinates and greyscale or RGB value) use the following command

```
[x y value] = impixel(A);
```

You can choose as many pixel as you want. Just click LMB on the pixels you choose one by one. To finish selection of pixels, press Enter.

Task 1.1: Please try image matrix manipulation in MATLAB. Load color image from the Image Set provided by the lecturer, display it as color image and as grayscale image. Using impixel tool, for both cases, read values of pixels from three different regions of the image. Using imcrop tool, cut small detail from the big image and save it as a new image variable. Show the cutting rectangle info – its point of origin and width and length.

Image histograms

For grayscale images, **a histogram** is a graph representing how many pixels with a given intensity (brightness) level are in the image. It should be mentioned here that if the images are 8-bit, the intensity of a pixel can be represented by 256 integer values (**uint8**) from 0 (black) to 255 (white). All numbers in between represent different degrees of gray.
double variable in the range 0-1 will also be interpreted as brightness levels.
In the case of a color image, each color channel has 8 bits, so each pixel of the image is represented by 24 bits. This allows you to quantify over 16 million colors.
The histogram of an image (and any other type of data) can also be understood as a probability mass function, informing about the probability of various intensity values occurring in the image.

In MATLAB, you can generate grayscale image histogram using the commands

`imhist(A);` – to quickly compute and visualize the histogram or
`[value x] = imhist(A);` – if you want to save histogram data to variables. You can visualize the histogram obtained in this way, for example using **`bar(x, value)`** command.

Now, see a simple command that loads an image, computes and shows its histogram

```
clc
clear all
close all

imRGB = imread('Mountains.jpg');

figure(1)
subplot(1,2,1);
imshow(imRGB);

imGray = rgb2gray(imRGB);

figure(1)
subplot(1,2,2);
imshow(imGray);

imDataRGB = size(imRGB);
imDataGray = size(imGray);

[value coordinate] = imhist(imGray);

figure(2)
subplot(1,2,1);
imshow(imGray);
subplot(1,2,2);
bar(coordinate, value);
```

Now, please do practical tasks to better understand the idea of image histogram.

Histogram equalization

One of the basic operations on histograms is **histogram equalization**. It is usually used when the image has low contrast. The introduced change in the distribution of brightness levels in the image " flattens " the histogram, which in practice increases its visual contrast.

Such an operation may, for example, reveal some structures that were previously invisible to the human eye due to low contrast. To equalize the image histogram, use the **`histeq(A)`** function.

```
[A_enh T] = histeq(A);
```

The function's argument is a grayscale image, but this operation may be applied to RGB images as well. The output is enhanced image and the intensity value transformation used.

Task 1.2: Please compute and plot histograms for images (in grayscale) chosen from an Image Set provided by the instructor. Plot an image and its histogram in the same Matlab figure. Compute histogram equalization. Plot the resulting image and its histogram in the second figure.

Task 1.3: Please create two random noise images using MATLAB 's random number generators. In the first case, the noise should be uniform (with a flat histogram), and in the second case, the noise should be normal (its histogram is a Gaussian bell curve, centered on the middle grey intensity). Write appropriate code using the rand() and randn() functions.

Remember what are the range of values that a grayscale pixel can take (0-255 for uint8 type or 0-1 for double)! If the random number generator returns numbers that are outside this range, you need to rescale the result.

Use larger images if you want your resulting histogram to resemble the theoretical histogram for a uniform and normal distribution.

An application of histograms – a simple thresholding

Image segmentation involves dividing the image into objects and the background. This is a very broad issue, covering both the simplest algorithms in which a classification of each pixel to a given object or background is determined by its intensity, as well as very complex semantic segmentation methods using deep neural networks.

The simplest algorithm, but very important in practice, is **thresholding (binarization)**. It involves converting a grayscale image into a black and white image. Thresholding is performed based on the **binarization threshold**. For example, pixels with a value less than the threshold are given a value of 0, and pixels with a value greater than the threshold are given a value of 1. Binarization is often a first step in many image analysis algorithms. It allows you to divide the scene into objects and background if the brightness of their pixels contrast with each other and can be separated by one or a number of thresholds. The value of the threshold can be find manually by a histogram inspection. For example, if the histogram of an image is bimodal – with two peaks corresponding to objects and a background, respectively, a middle intensity level between them may be chosen as a threshold of binarization. You can also use automatic thresholding, using **Otsu method**, for example.

See the following code that performs a simple binarization.

```
clc
clear all
close all
```

```
imRGB = imread('Mountains.jpg');

figure(1)
subplot(1,3,1);
imshow(imRGB);

imGray = rgb2gray(imRGB);

figure(1)
subplot(1,3,2);
imshow(imGray);


threshold = % fill in your threshold value!
% write here the value of the threshold you found by insepcting the histogram of your
image or apply automatic thresholding threshold = graythresh(imGray); in MATLAB, the
function im2bw needs threshold in double 0, it is a number between 0 and 1!

imBW = im2bw(imGray, threshold);
% you can use also logical statements, for example imBW = imGray < threshold


figure(1)
subplot(1,3,3);
imshow(imBW);
```

If you are using **`im2bw()`** function, please remember that it requires threshold value to be a number between 0 and 1. If you read the threshold value from a histogram created for integer valued image, you need to normalize it to the range 0-1 first.

Task 1.4: Using the code above, please perform a binarization of the image from the Image Set according to the instructor's request. In your resulting image, the one object chosen by your instructor must be white, the rest of the pixels must be set to zero.

Basics of Image Filtering

First, a short theoretical background. The image can be understood as a 2D signal in which the independent variables are the position coordinates (x, y) of the pixel in the coordinate system of the image, and the dependent variable is the brightness of the pixel (for grayscale images) or a vector of 3 values describing the color of the pixel (for RGB images).

Filtering is a basic contextual operation performed on images. Linear filters are implemented by convolving the image with masks with different numerical coefficients depending on their functions. Linear filters are divided into low-pass and high-pass filters. As with one-dimensional signals, low-pass filters remove spatial frequencies higher than a given cutoff frequency from the image, and high-pass filters remove low frequencies while preserving high frequencies in the image.

In the example below, you will figure out what a spatial frequency is and what low and high spatial frequencies are in the image. What image elements are these frequencies responsible for and what will happen if they are removed (or reduced) from the image by the filtering process?

There is a **`conv2()`** function in MATLAB that calculates the convolution of 2 dimensional signals. This function takes an image and a user-prepared filter mask as arguments. The result is a new image, after filtering. The result is stored in a matrix of double numbers , so you must convert the numbers to uint8 before displaying. In this exercise, you will see

examples of a low-pass and a high-pass filter.

As an example of a low-pass filter, we take the simplest **rectangular averaging filter** (box). Its mask for size 3x3 is

```
mask1 = (1/9)*[1 1 1; 1 1 1; 1 1 1];
```

Filtering using the previously mentioned function is performed as follows

```
imFilt1 = conv2(image, mask1);
```

As an example of a high-pass filter, let's see **Sobel operator** (named after Irwin Sobel, born 1940, a filter developed in 1968 at the Stanford AI Project as part of a doctoral thesis).

```
mask2 = [1 0 -1; 2 0 -2; 1 0 -1];
```

A simple code to test these filters

```
clc
clear all
close all

imRGB = imread('Mountains.jpg');
imGray = rgb2gray(imRGB);

figure(1)
subplot(1,3,1);
imshow(imGray);

mask1 = (1/9)*[1 1 1; 1 1 1; 1 1 1]; % LP box filter
mask2 = [1 0 -1; 2 0 -2; 1 0 -1];    % HP Sobel mask (directional)

im_filt1 = conv2(imGray), mask1);
im_filt2 = conv2(imGray), mask2);

figure(1)
subplot(1,2,3);
imshow(uint8(im_filt1));

figure(1)
subplot(1,3,3);
imshow (uint8(im_filt2));
```

Let's do a simple exercises using the provided code.

Task 1.5: Run the code provided in the lab instruction. Choose an image from an Image Set provided by the lecturer, display the results of filtering using both linear filters. Change the size of the mask1 from 3x3 to 5x5 and 7x7. Use a sum of ones in the mask as normalization constant (1/9 in the above example). Display the results. Transpose the mask 2, and display filtering results.

Low pass filtering – linear filters

Due to the great practical importance of image filtering, MATLAB has a specialized **`imfilter()`** function dedicated to image filtering. If you want to use standard filters, their masks can be obtained easily using the **`fspecial()`** helper function. Provide the name of the filter and the basic parameters related to each of them.

Fore example, to have our box filter's application from the previous exercise, use

```
h = fspecial('average',[3 3]);
im_filt = imfilter(imGray,h,'replicate');
```

Let's do our previous exercise using these code

```
clc
clear all
close all

imRGB = imread('Mountains.jpg');
imGray = rgb2gray(imRGB);

figure(1)
subplot(1,3,1);
imshow(imGray);

mask1 = fspecial('average',3); % LP box filter
mask2 = fspecial('sobel');     % HP Sobel mask (directional)

im_filt1 = imfilter(imGray,mask1);
im_filt2 = imfilter(imGray,mask2');

figure(1)
subplot(1,2,3);
imshow(im_filt1);

figure(1)
subplot(1,3,3);
imshow (im_filt2);
```

First, we will deal with low-pass filters. Low-pass filtering is performed by the following types of filters available in the fspecial function :

1) A rectangular averaging filter with the window size given by the hsize variable (odd integer)
   `h = fspecial('average',hsize);`

2) A circular averaging filter of the size given by the radius variable
   `h = fspecial('disk',radius);`

3) A Gaussian filter requiring the mask size and the standard deviation of a bell curve that is approximated by the mask coefficients
   `h = fspecial('gaussian',hsize,sigma)`

Now consider the following exercise

Task 1.6: Please perform low pass filtering of an image from the Image Set. Display the results of filtering by average and disk filters. Try three different sizes of masks.
After consulting Matlab's help, choose two different values of hsize and sigma parameter for Gaussian filter, display the results.

Non-Linear Filtering

The most common nonlinear filter in image processing is the **median filter**. This filter determines the median for each pixel, i.e. the central value of the pixel brightness in the window whose center is located at this pixel. In MATLAB, median filtering can be

performed using the **medfilt2()** function.

An example of a median filter use:

```
im_noisy = imnoise(imGray,'salt & pepper',0.02);
im_filtered = medfilt2(im_noisy);
```

The example uses the MATLAB **imnoise()** function, which adds noise to the image with a given statistical distribution (in the example 'salt & pepper ' or an impulse noise) of a given intensity (in the example 2% of modified pixels).

Task 1.7: Improve the quality of an image by removing noise or scratches. Apply appropriate filters to remove noise present in the image provided by the instructor.

High pass filtering and Edge Detection

There are many edge detection methods, of which the laboratory class will discuss only those based on linear high-pass filtering. These filters numerically calculate the first or second derivative of the image. To determine the masks of this type of filters you may use the already known **imfilter()** and **fspecial()** functions. The second one allows you to generate the following filters:

Operators that numerically calculate the first derivative of an image. They are directional, i.e. they give maximum response to vertical or horizontal edges. You can change their directionality by taking a transpose of the mask

1) **Sobel operator**
   ```
   h = fspecial('sobel');
   ```

2) **Operator Prewitt** (named after Judith Prewitt , work from 1970)
   ```
   h = fspecial('prewitt');
   ```

Operators that numerically calculate the sum of the second partial derivatives with respect to the variables x and y. There are two available in MATLAB - **the Laplace operator** and the **Laplacian of Gaussian** (**LoG**). They are non-directional operators (to some extent, they operate on the space of discrete points in a regular rectangular grid).

3) **Laplace operator** - the alpha variable determines the shape of the Laplacian (see MATLAB help, for alpha = 0 we get the standard Laplacian)
   ```
   h = fspecial('laplacian',alpha);
   ```

4) **LoG operator** – before calculating the derivatives, the image is Gaussian-blurred to reduce the impact of noise on edge detection. The meaning of the parameters is the same as for the Gaussian filter.
   ```
   h = fspecial('log', hsize,sigma);
   ```

An alternative to these approach is the **edge()** function, which is exclusively dedicated to detecting edges in images. These functions use the operators described above. The edges are found at the pixels for which the first derivative has a maximum or at the zero-crossing points of the second derivative.
For example, to apply a Sobel filter using this function, use the following code

```
im_edge = edge(imGray, 'sobel', thresh, direction);
```

thresh variable specifies the filter response above which a pixel is considered to belong to an edge, and direction specifies the direction in which the image first derivative is computed (' horizontal ' or ' vertical '). This function allows you to use the Sobel , Prewitt , log operators as before.
In addition, it offers new types of edge detectors:

5) **Roberts' Cross** – name after Lawrence Roberts (1937-2018), work from 1963, L. Roberts was one of the creators of ARPANET and is called one of the founding fathers of the Internet. His filter is the simplest edge detector.

6) **Canny edge detector** – one of the most famous vision algorithms used in practice. It has two thresholds, the correct setting of which allows you to determine uninterrupted edges while reducing noise. John Canny (born 1958) is a professor at the University of Berkeley, California. Paper on the topic of the detector is from 1986.

Now, performs series of tasks concerning edge detection

Task 1.8: Apply edge detection to an image provided by the instructor. Apply high pass filters: Sobel, Prewitt and Laplacian using imfilter() function and display the results, apply edge() function for the same type of filters as previously and display the result.

Task 1.9: Detect edges using Canny filtering method. Select parameters of the filter to obtain the edges of objects in the image provided by the instructor.

Morphological Filtering

In this part, you will learn how to use morphological filters on grayscale images. In the next lab, these filters will be discussed in much more detail for binary images case. Broadly speaking, for grayscale images, these filters work like max or min filters. They replace the pixel value with the value that is highest/lowest in the mask centered on that pixel.
The shape of the filter mask can be freely selected by the user.
To use this filter, first you must create a structural element, as in an example

```
se1 = strel('type',size);
```

Chose a mask shape according to the task you are going to do. Then, you can apply the following four operations to the input image - erosion, dilation, opening and closing.

```
bw_erode = imerode(im_bw,se1);
bw_dilate = imdilate(im_bw,se1);
bw_open = imopen(im_bw,se1);
bw_close = imclose(im_bw,se1);
```

Now, do the final set of tasks in the instruction.

Task 1.10: Apply the four basic operation on an image from the Image Set. Plot the results of four basic morphological filtering in one Matlab figure. Show the results for two different

shapes of structuring elements and 2 different size of the mask.

Task 1.11: Improve the quality of images by removing noise or scratches using only morphological operations. Apply appropriate filters to remove noise present in the images provided.

Additional Tasks

Task 1.12: Please do the exercise 1.2, but use RGB color images instead of grayscale ones. Remember that you have to process each color channel separately and then make RGB histograms yourself. Try to convert the RGB color space to HSV. Using different color images try to understand what HSV channels represent. Create histograms for images in this representation and compare them with the ones made using RGB images.

Task 1.13: Generate a series of noisy images with impulse and Gaussian noise by an application of the imnoise() function by successively increasing the noise parameters. Then try the filters you've learned to remove noise as effectively as possible without significantly changing the image content. Try different sizes of filter masks and different values of other parameters.
You may show the level of the noise present in the by cropping a part of the image of constant intensity and then generating a histogram for it. A histogram of the image without noise should be lumped around similar values of intensity. As a numerical measure of noise you can use standard deviation of the image part of constant intensity.

Task 1.14: Generate a series of noisy images with impulse and Gaussian noise by an application of the imnoise() function by successively increasing the noise parameters. Then try using the morphological filters to remove noise as effectively as possible without significantly changing the image content. Try different sizes and shapes of filter masks.
You may show the level of the noise present in the by cropping a part of the image of constant intensity and then generating a histogram for it. A histogram of the image without noise should be lumped around similar values of intensity. As a numerical measure of noise you can use standard deviation of the image part of constant intensity.