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 8:

Image Features and Object Classification

You will learn: what are objects in binary image, what is labelling, how to find basic features of objects using regionprops MATLAB function, what are geometrical features: invariant moments, Hu moments, how to build feature vector, how to evaluate feature sensitivity to scaling and rotation, how to build a simple object classifier, what is the measure of similarity and how to find a known pattern in the image, what is normalize cross-correlation coefficient, what are local features and their applications, how to implement and test Harris corner detector

Additional materials:

- Course lecture
- Image Database provided by the lecturer

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

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

Image correlation coefficient for finding objects

The image correlation coefficient can be used to find a given pattern in an image. The correlation function takes a pattern in the form of a smaller image and looks for it in an image or series of images. The variable created by the correlation function is a matrix, a new image, that illustrates the degree of similarity between the pattern and the area in the image placed below the pattern for each position of the pattern in the image. Then, by searching the maximum value in this matrix, the location of the pattern in the image can be found. To calculate the similarity map between the pattern and the image, use the function

```
c = normxcorr2(image_pattern,image);
```

The variable c can be displayed as an image (**`image()`** or **`imagesc()`** functions) or as a surface plot using the command **`surf(c)`**. Then, given the similarity values as a function of position in the image, one can find the maximum value of the function and the location of this maximum. In MATLAB this can be done with code:

```
[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax(1));
```

Remember that the **`normxcorr2()`** function changes the size of the analyzed image. Therefore, to display the center of the found pattern, we recalculate the indexes as follows:

```
peak_offset = [(xpeak - size (image_pattern,2))
               (ypeak - size (image_pattern,1))];
```

After using **`surf()`**, use the **`shading flat`** command to display the correlation function response as a surface, because otherwise too many edges will be drawn in the figure and the entire plot will be black.

Here is the simple code that searches for a pattern in the image

```
clc;
clear all;
close all;
% Correlation for pattern detection examples
im = imread('Wagtail.JPG');
pattern = imread('Wagtail_P2.JPG');

imGray = rgb2gray(im);
patternGray = rgb2gray(pattern);
imGrayTemp = imGray;

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

% image correlation computation - find one pattern in the image
c = normxcorr2(patternGray,imGray);

[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax(1));

peak_offset = [(xpeak - size (patternGray,2))
(ypeak - size (patternGray,1))];

figure,
subplot(1,2,1)
```

```
imagesc(c);
subplot(1,2,2)
surf(c);
shading flat

figure,
imshow(im); hold on
% plot(peak_offset(2),peak_offset(2),'*'); hold on
rectangle('Position',[peak_offset',size(patternGray)],'Edgecolor','red');
```

Image correlation has become an important tool in non-contact measurements of mechanical quantities such as displacements, deflections, deformations and strains. It is the basis of a measurement technique called DIC (Digital Image Correlation).
Read about it on the Internet and try to find solutions available on the market. If this topic interests you, you can read more, learn about the algorithm and download and test a free version of the program developed in the MATLAB computing environment at http://www.ncorr.com/

Task 3.1. Run the MATLAB code for finding a pattern in the image. Mark the found pattern by enclosing it by a red rectangle or dot representing its center. See how the function finds different patterns. Observe the correlation response and discuss which of the patterns are the best for detection and why. Which of them would be more robust to noise in the image?

Image Analysis – Objects detection and classification

Let's start with the review of the previous lab exercise. Recall that the first step of object analysis is a binarization. You got to know the basic as well as more advanced methods. After binarization, the objects in the image must consists of white pixels (of value 1) while the background becomes black (pixels of value 0). Next, all objects have to be labeled using **`bwlabel()`**. This function simply gives numbers (labels) to all disconnected objects in the image. In the next step, the geometric parameters describing shapes are computed using **`regionprops()`** function. In this exercise, use that function to obtain all available features – use property 'all'.

Here's there is a simple program from the previous class that computes all the features of objects.

```
clc;
clear all;
close all;

imRGB = imread('Objects.jpeg');

figure(1)
subplot(1,2,1)
imshow(imRGB);
title('Original Image')

imGray = rgb2gray(imRGB);

thresh = graythresh(imGray);
imBW = im2bw(imGray,thresh);

figure(1)
subplot(1,2,2)
imshow(imBW);
title('Binary Image')

imLB = bwlabel(imBW,4);
```

```
features = regionprops(imLB,'all');
```

Let's modify the program in the following exercises:

Task 3.1 Change the program in such a way that all features are computed instead of just basic ones. Use an image in which there are several different shapes present. Perform a successful binarization of that image – obtaining a binary image with all objects. Write a code that prints a centroid of each object in the image and its number next to it. Find how different features change with the shape of the object.

The features which you should examine in this exercise: *Area, BoundingBox, Centroid, ConvexArea, Circularity, Eccentricity, EquivDiameter, EulerNumber, Extent, Orientation, MaxFeretProperties, MinFeretProperties, MajorAxisLength, MinorAxisLength, Solidity.*

Task 3.2 Try to find the minimum set of features that is necessary to classify all shapes in the image. The features should not be redundant, e.g. you should discard the feature that carries the same information as the other ones. Next, write a simple code for shape classification based on feature values using a nested if-else programming structure.

Moments and moment invariants

In the binary image processing, the moments and moment invariants are mathematical concepts that describe geometrical features of objects e.g. its shape. They are important in the field of object recognition and classification and shape analysis.

Moments – The moments are numbers associated with objects that quantify the spatial distribution of image pixels and may be used to describe a shape. There are several types of moments which will be described in the following lab instruction.

1. **Ordinary Moments** – they describe general geometric properties (spatial distribution of pixels) of objects based on their shape (pixel distribution).

   $$M_{pq} = \sum_x \sum_y x^p y^q I(x,y)$$

   Where I(x,y) – is pixel intensity at (x, y) – in the case of the binary images it takes a value 1 for object's pixels and p, q – are integer s indicating the order of the moments.

2. **Central Moments** – they describe spatial distribution of object's pixels relative to its centroid. They are **robust** to the shift in **the position** of the object in the image. In order to compute the central moments, you first have to compute the position of the centroid of the object, using first order ordinary moment. Then the central moments are computed as follows

   $$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q I(x,y)$$

The formula is the same as before, except, you have to subtract the centroid x and y coordinate as indicated in the equation.

3. **Normalized Moments** – these moments are **robust** to **scale change** of the objects. They are computed based on the central moments and area of the objects. They are central moments divided by a scaling factor.

$$\varphi_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{1+\frac{p+q}{2}}}$$

4. **Moment invariants** – these are types of moments that are invariant under transformations of the image, such as translation, scaling and rotation. They help to recognize the same object type in the image, even if it is changed by these transformations.

   One of the most commonly used types of moment invariants are **Hu moments**. They were reported in the scientific literature by M.K. Hu in 1962. For each object, a set of 7 Hu moments is computed based on the normalized central moments. Hu moments are invariant to translation, rotation and scaling transformations.

In the following set of exercises, you will be applying MATLAB functions provided by the instructor to carry out object recognition and classification.

Here is the code of the function that computes moments and moment invariants

First, after image thresholding and labelling, use **`regionprops()`** function to compute centroids and obtain a list of pixels that belongs to each of the objects.

```
features = regionprops(imLB, 'Centroid', 'PixelList');
```

Next you can loop through the objects to compute the moments, for example in the case of central moments:

```
order = [3 3];

for k = 1:length(features)

    % find centroids and pixel list for each of the objects
    centroid(k).coordinates = features(k).Centroid;
    pixelList(k).List = features(k).PixelList;

    % Compute central moments for the object
    moments(k).CentralMoments = computeCentralMoments(pixelList(k).List,
centroid(k).coordinates,order);

end
```

Next, the functions that computes all moments and moments invariants.

```
function moments = computeCentralMoments(pixelList, centroid,order)

moments = zeros(order(1)+1, order(2)+1);  % Initialize a matrix for central moments
```

```
(p=0..2, q=0..2)
    c_x = centroid(1);
    c_y = centroid(2);

    % Calculate central moments for p, q = 0, 1, 2, 3, etc.

for i = 1:size(pixelList, 1)
        x = pixelList(i, 1);
        y = pixelList(i, 2);

        for p = 0:order(1)
            for q = 0:order(2)
                moments(p+1, q+1) = moments(p+1, q+1) + (x - c_x)^p * (y - c_y)^q ;
            end
        end
    end
end
```

To compute Hu moments, your first have to normalize central moments obtained in the previous step.

```
function normalizedMomentsList = computeNormalizedMoments(momentsList)

for k = 1:length(momentsList)

    moments = momentsList(k).CentralMoments;
    normalized_moments = zeros(size(moments,1), size(moments,2));

    m00 = moments(1, 1);

    for p = 0:size(moments,1)-1
        for q = 0:size(moments,2)-1

            normalized_moments(p+1, q+1) = moments(p+1, q+1) / (m00^(1 + (p + q)/2));

        end
    end

    normalizedMomentsList(k).NormalizedMoments = normalized_moments;
end

end
```

Now, you can compute a set of seven Hu moments:

```
function hu_moments = computeHuMoments(normalizedMomentsList)

for k = 1:length(normalizedMomentsList)

normalized_moments = normalizedMomentsList(k).NormalizedMoments;

    eta20 = normalized_moments(3, 1);
    eta02 = normalized_moments(1, 3);
    eta11 = normalized_moments(2, 2);
    eta30 = normalized_moments(4, 1);
    eta03 = normalized_moments(1, 4);
    eta21 = normalized_moments(3, 2);
    eta12 = normalized_moments(2, 3);

    % Compute the 7 Hu moments
    phi1 = eta20 + eta02;
    phi2 = (eta20 - eta02)^2 + 4 * eta11^2;
    phi3 = (eta30 - 3 * eta12)^2 + (3 * eta21 - eta03)^2;
    phi4 = (eta30 + eta12)^2 + (eta21 + eta03)^2;
    phi5 = (eta30 - 3 * eta12) * (eta30 + eta12) * ((eta30 + eta12)^2 - 3 * (eta21 +
eta03)^2) + (3*eta12 - eta03)*(eta12 + eta03)*(3*(eta30 + eta12)^2 - (eta21 +
eta30)^2);
    phi6 = (eta20 - eta02) * ((eta30 + eta12)^2 - (eta21 + eta03)^2) + 4 * eta11 *
(eta30 + eta12) * (eta21 + eta03);
    phi7 = (3 * eta21 - eta03) *(eta30 + eta12)* ((eta30 + eta12)^2  - 3 *(eta12 -
eta30))-(eta30-3*eta12)*(eta21+eta03)*(3*(eta30 + eta12)^2 - (eta21 + eta30)^2);

    % Store Hu moments in a vector
    hu_moments(k).Hu = [phi1, phi2, phi3, phi4, phi5, phi6, phi7];

end

end
```

Task 3.3 Analyze the images with the same geometric objects, but with introduced changes – translation, rotation and scaling. Compute central moments and normalized moments. Compute moments of different and increasing order for each of the objects and each of the images. Discuss how the value of the moments change with shapes and how they change with introduced transformations. Which of the moments are invariant to translation, scaling and rotation?

Task 3.4 Repeat exercise 3.3, including the Hu moments in the feature vector. Check the invariance of Hu moments for transformations. Write a classification program that uses all moments.

Task 3.5 Apply the knowledge acquired in the previous exercise to write a program for simple objects' classification based on the values of moments. The program should classify all shapes present in the image. Draw centroids of each objects and mark each class by a bounding box of a distinct color.

Local features and feature detectors

In many practical cases, the region-based image segmentation in not enough to properly describe objects present in the image. Even the best algorithms may not segment the image into objects with the desired level of accuracy. In such cases, instead of describing the objects as regions, we may treat them as a set of points. The points may represent the characteristic parts in objects like its corners. By detecting these points and computing their positions we may model the entire object. For example, we may use the set of points to identify the same object on a series of images by analyzing spatial relations between the points. It is possible even if the object deforms as it moves in a video, because each of the point may be tracked individually. In the computer vision literature, these characteristic points are often called **local image features** (point features).

There are many different feature detector and descriptors available in the programming packages. They differ by the types of features they detect the best (**corner features vs blob features**), time and complexity of computation and type of **invariance** (e.g. rotation and scale invariance).

Here are some detectors that are implemented in MATLAB:

1. Harris detector (1988) – the most known and widely used in practice corner detector that detects corner points,
2. Minimum Eigenvalue algorithm (Shi-Tomasi algorithm, 1994) – corner detector using the same eigenvalue approach as Harris, but different scoring function,

3. FAST (Features From Accelerated Segment Test, 2006) – corner detector that is computationally fast compared to other detectors that use DoG approach (e.g. SIFT),
4. SURF (Speed-Up Robust Features, 2006) – multiscale feature descriptor, faster than more known SIFT descriptor, applicable for detection of blob-like features at various scales, the scale-space is generated using image pyramids,
5. ORB (Oriented FAST and Rotated BRIEF, 2011) – detector which is a combination of FAST corner detector to localize the feature and BRIEF descriptor, but is rotation-invariant, free to use response to SIFT and SURF (patented ones),
6. BRISK (Binary Invariant Robust Scalable Keypoints, 2011) – another scale-invariant feature descriptor that is a free response to a patented SIFT, it is in general a multiscale FAST corner detector, not rotationally invariant,
7. KAZE (2012) – multiscale feature descriptor, uses Nonlinear Diffusion Filtering to create scale-space (instead of pyramid approach and Gaussian blurring), it has also an open source code

In this exercise you will learn how to compute and use image local features using feature detectors and descriptors.

You may apply detectors using a series of built-in function as in the following example (for an application of Harris corner detector)

```
points =
detectHarrisFeatures(im,'MinQuality',0.55,'FilterSize',15);
```

As you can see, there are several parameters which values may be changed. These values may be tuned to get a desired results e.g. all corner points present in the image detected, but no double or triple corners at the same place and no corners due to image noise are found. Read MATLAB documentation to learn about the meaning of all parameters and then try to find the values for which you obtain the best result for your image.

Also you may get only N corners for which the detector's response is the largest.

```
points = points.selectStrongest(N);
```

Now you can show the corners found by the algorithm using the following code:

```
points.Location = points.Location + [cropRect(1) cropRect(2)];

figure,

  imshow(im);
  hold on;
  x = points.Location(:,1);
  y = points.Location(:,2);

  plot(x,y,'*g');
  hold on
```

In the code, the cropRect variable is added if you first cropped a subimage. If you are using full image, set cropRect to zero vector. To see how the Harris corner detector operates, please implement the detector in a function yourself and check how it performs comparing to the Harris detector implemented in the MATLAB. The exercise will be interesting since it involves many image processing that you have learned so far.

First of all, your function must read and image and convert it into grayscale representation. Next, convert you image to double.

```
I = double(I);
```

Next, you need to create a measure of ‘cornerness’. First step is to obtain the gradient image. You can also filters that compute first derivative, like Sobel filter, instead.

```
[Ix, Iy] = gradient(I); % Gradient in x and y direction
```

Now, compute product of derivatives.

```
Ixx = Ix.^2;
Ixy = Ix .* Iy;
Iyy = Iy.^2;
```

As we discussed in the previous exercises, it is useful to blur image before further computation. Therefore, apply Gaussian filtering.

```
sigma = 1; % Standard deviation for Gaussian kernel
windowSize = 3; % Size of the window for Gaussian filter

G = fspecial('gaussian', windowSize, sigma);
Ixx = conv2(Ixx, G, 'same');
Ixy = conv2(Ixy, G, 'same');
Iyy = conv2(Iyy, G, 'same');
```

Now let’s compute the Harris corner response function, using k constant according to the literature.

```
k = 0.04; % Harris corner constant
R = (Ixx .* Iyy - Ixy.^2) - k * (Ixx + Iyy).^2;
```

Now, using a threshold, find only the points for which Harris corner response function was above the threshold

```
threshold = 0.01 * max(R(:)); % Threshold is a fraction of the
maximum value of R
corners = R > threshold;
```

Now you can visualize the corners by drawing them on the image.

Task 3.6 Write a program that detects corners in an image. Try to choose the parameters of the Harris corner detector to detect the highest number of true edges. Detect the corners as indicated by the instructor. Try to minimize false detection e.g. places where there are no corners, edges, avoid double corners at the same locations. Discuss the results.

Task 3.7 Repeat the Task 3.6 to test other types of feature detector and descriptors available in the MATLAB. Note, that some of them detect corner features, like Harris corner detector, other ones are better suited to detect blob-like features. You will be given different images containing both of these types of features. Test which detectors are better to detect which types, try to find the best set of parameters for a given image like in the previous task.

There are the following functions available in MALAB:

```
detectMinEigenFeatures();
detectFASTFeatures();
detectORBFeatures();
detectBRISKFeatures();
detectKAZEFeatures();
detectSURFFeatures();
```

An application of local features – feature matching

One of the most common application of image local features is feature matching, in which the program matches the same feature across many images with the same scene. In can be applied for a particular object detection in the image with multiple objects and object tracking in a video sequence. Also, feature matching in one of the major steps in all 3D structure and motion reconstruction algorithms. It is used to find the corresponding points e.g. the projections of the same 3D world point on two or more images captured by a camera system. The mutual relationship of the corresponding points is necessary to compute the depth of the scene. In the case of image stitching, the feature matching is needed for panorama image generation.

In order to match features in MATLAB, it is not enough to know the positions of the points in image, but we must first generate **feature vector** for each of these points. Feature vector contains information on the mathematical description of spatial distribution of intensity levels and geometry of an image patch in the neighborhood of the points. Let us assume that we have two image containing the same scene seen from two viewpoints, or the same objects on two different scenes, etc. and we want to find the set of corresponding feature point pairs.

The feature matching can be carried out in the following way:

First, read two images:

```
I1 = rgb2gray(imread('ImageLeft.png'));
I2 = rgb2gray(imread('ImageRight.png'));
```

We extract features using one of the previously analyzed feature detectors, for example Harris corner detector

```
points1 = detectHarrisFeatures(I1);
```

```
points2 = detectHarrisFeatures(I2);
```

We can use plot detected feature points to see the results of our detection. Next, we have to build feature representation for all detected feature points.

```
[features1,valid_points1] = extractFeatures(I1,points1);
[features2,valid_points2] = extractFeatures(I2,points2);
```

We can match the features based on their mathematical representation

```
indexPairs = matchFeatures(features1,features2);
```

The function returns a Nx2 matrix containing pairs of indices of corresponding points. Next we use these indices to get image coordinates of these pairs.

```
matchedPoints1 = valid_points1(indexPairs(:,1),:);
matchedPoints2 = valid_points2(indexPairs(:,2),:);
```

MATLAB provides a function that quickly visualized the correspondence between points.

```
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,'montage
');
```

Task 3.8 Chose a feature detector that gave you the best results in the previous exercise. Find the highest number of correct corresponding point pairs in the sets of two images provided by the instructor. Try to repeat the exercise for pairs of images captured by yourself.

Additional Tasks

Task 3.9. Create an image that contains several patterns of the same type (the patterns may be slightly different from each other, you can modify them by adding disturbances). Then, write a MATLAB program that will detect all patterns of a given type and mark them in the image. The program has to count the number of patterns present in the image. Test the program on the images provided by the instructor.

Task 3.10. In this exercise, you'll learn how robust is the correlation to changes in the image.
See how the pattern detection algorithm response with the following disturbances:
1) the image is brightened,
2) the image is darkened,
2) the image is blurred by a low-pass filter,
3) the image is noisy (e.g. added salt and pepper noise, Gaussian noise, etc.),
4) the image is rotated (e.g. using the imrotate() MATLAB function),
Test the robustness of the correlation function by gradually increasing the strength of each type disturbance and find the value of the disturbance for which the function fails to find the pattern. Check whether the correlation method's effectiveness depends on the type of pattern to be detected. You may show your results by drawing a graph showing a position of a center of the detected pattern as a function of disturbance strength.

Task 3.11 Please, test the invariance of the features to different changes in the image. Prepare an object which has visible set of local features. For at least three of available feature descriptors, find the best parameters to detect the highest number of true local features in the reference image of the object. Then, change the following taking photos of the objects after each change: distance of the camera to the object, the focus on the object (slight blurring), the angle of view of the camera with respect to the object, the lighting conditions. After introducing each of the change, observe if the features are still detected and if they are detected in the correct place. Discuss the results.