TNM087 – Image Processing and Analysis
Lab 4 – Hough transform, Morphological operations and Image features

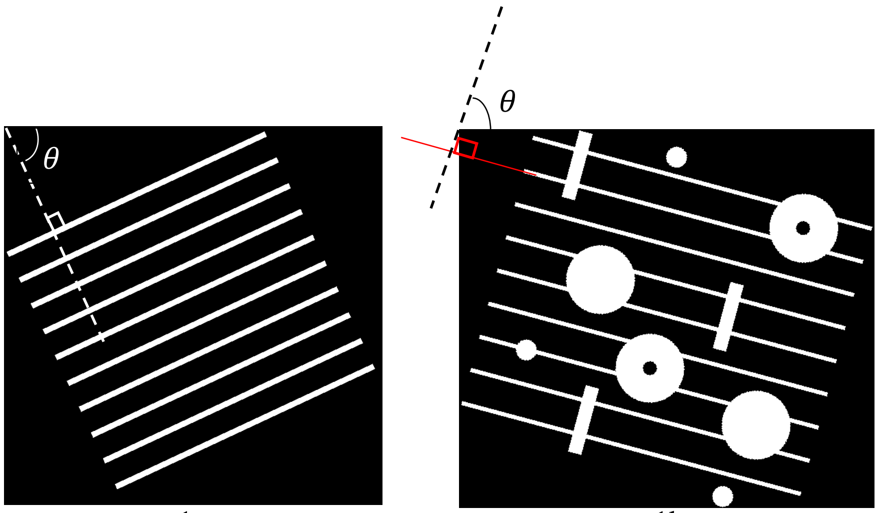
TASK 1 Preparation

The preparation task consists of a number of problems that should be solved using Matlab. Your answers (in Swedish or English) should be written in the document *Lab_4.1_Preparation_Answers.docx*, where you also insert the required images. To save the images you can use the MATLAB functions `imwrite` or `imsave`. Make sure to save the images in an uncompressed format, such as **.tif** or **.png**. Before submitting the answer document on Lisam, first save the document as **.pdf**!

For the preparation tasks you do not need to submit your m-file. However, it is strongly recommended that you save your experiments in an m-file, in case you need to go back and correct anything later. Sometimes, you can also re-use your code in later tasks.

1) Hough transform

In this part of the task, you are going to study the Hough transform to detect principal lines in an image and rotate the rotated image to the horizontal level, meaning that the principal lines become horizontal after appropriate rotation. Before you continue, read the book or the lecture notes for Chapter 10 (**FÖ 7 in the course**), **pages 31-38** for the theory**,** and **pages 39-41** for MATLAB instructions regarding Hough transform. As discussed in the lecture notes, the angle of the principal lines is interpreted a bit differently in the course book (and lectures) compared to MATLAB. Here, we specify the angles as they are defined in MATLAB. Recall that the angle θ is supposed to vary between -90° and 90°. The following two images are the ones you are going to work with in this part of the task, called $Image1a.tif$ and $Image1b.tif$, respectively. As seen in the below figure, the angle corresponding to the principal lines in *Image 1a* is positive while the one for *Image 1b* is negative. In order to rotate these two images to horizontal level, $Image\ 1a$ needs to be rotated clockwise, and $Image1b$ counterclockwise.



1a 1b

Read the image $Image1a.tif$ and call it $Image1a$ in MATLAB. **Notice** that this image is binary (of logical type) and doesn't need to be scaled on $[0,1]$. This image is the same as *Image 1a* in the above figure. Find the Hough transform of this image by (for example):

$$[H, teta, ro] = hough(Image1a, 'Rhoresolution', 5, 'Theta', -90{:}\,0.5{:}\,89.5);$$

In the above command, we have specified the angle θ to vary from -90° to 89.5°, in step of 0.5°. H is the Hough transform of the image and $teta$ is the vector $[-90{:}\,0.5{:}\,89.5]$ and ro is the vector $[-940{:}\,5{:}\,940]$. Notice that ρ for an $M \times M$ image is varying from $-\sqrt{2}M$ to $\sqrt{2}M$. Since $Image1a$ is 665×665, ρ varies between -940 and 940. Since we have chosen $'Rhoresolution' = 5$, the ro vector becomes $[-940{:}\,5{:}\,940]$.

Problem 1) Find the Hough transform of $Image1a$ and use **contrast stretching** to scale it on $[0,1]$. Call the result $H1$. Insert your result in the answer document.

As you notice, there are 10 dominant peaks in $H1$, which correspond to the 10 lines in $Image1a$. Since the lines are parallel, they are all appearing at the same angle. We know that the horizontal axis of this plot is θ and it varies on $[-90{:}\,0.5{:}\,89.5]$.

Problem 2) Make a guess on the angle corresponding to these parallel lines from the Hough transform image, i.e. $H1$.

In order to find the exact angle corresponding to these lines, we just need to find where the maximum (or maxima) in the Hough transform occurs (occur). The following MATLAB command finds the position of the maximum (or maxima) in the Hough transform image,

$$[r, t] = find(H == max(H(:)));$$

The angle at which the maximum (maxima) occurs can easily be found by $teta(t)$. Notice that, you obtained the vector $teta$ where you called the function $hough$ above.
If you are interested to know the distance from this line (these lines) to the origin, you can use $ro(r)$.

Problem 3) What is the exact angle corresponding to the lines in $Image1a$?

By viewing $Image1a$, you can see that in order to rotate this image to horizontal level, we need a **clockwise** rotation.

Problem 4) What is the angle of **clockwise** rotation to rotate $Image1a$ to the horizontal level? Use your answer from problem 3.

Now, rotate $Image1a$ about its center to the horizontal level, and call the resulting image $Image1a\_rotated$. This can be done by using the MATLAB function $imrotate(image, angle, 'bicubic', 'crop')$, where $image$ is the image to be rotated, $angle$ is the rotation angle in degrees, $bicubic$ is the chosen interpolation and $crop$ is used to crop the rotated image to be the same size as the image being rotated. If you want the rotated image to have its full size, remove $crop$ from the above command.
Important: If *angle* is positive, then the rotation is counterclockwise, and if it is negative, the rotation is clockwise.

Problem 5) Insert the rotated image, i.e. $Image1a\_rotated$, in the answer document.

Read the image $Image1b.tif$ and call it $Image1b$ in MATLAB. **Notice** again that this image is binary (of logical type) and doesn't need to be scaled on [0,1]. This image is the same as *Image 1b* in the above figure. As seen in the Hough transform of this image, there are 9 parallel lines in this image.

Problem 6) What is the exact angle corresponding to the straight lines in $Image1b$? (Use Hough transform as you did for $Image1a$)

Problem 7) What is the angle of **counterclockwise** rotation to rotate $Image1b$ to horizontal level? Use your answer from problem 6.

Now, rotate $Image1b$ to the horizontal level, and call the resulting image $Image1b\_rotated$.

Problem 8) Insert the rotated image, i.e. $Image1b\_rotated$, in the answer document.

2) Morphology

In this part of the task, you are going to experiment with morphological operations on binary images. The theoretical background is found in Chapter 9 (FÖ 8 in the course).

Read the image $Image1c.tif$ in MATLAB. Notice that this image is binary (of logical type) and doesn't need to be scaled on [0,1]. As you can see, the image contains some simple geometrical objects, vertical lines, and also some noise. In the image, the width of the vertical lines is 5 pixels. The radius of the small disks is 14 and the radius for larger disks is 100. The rectangles are 20 x 100 pixels.

First you need do remove the noise from the image (both the white noise in the background and the black noise within the objects). One possible way to clean up binary images from noise is to perform a morphological opening, followed by a morphological closing, using the same structuring element, SE. In Matlab, this can be done, as:

```
IM2 = imopen(IM, SE)
IM3 = imclose(IM2, SE)
```

Where `IM` is the original (binary) image and `SE` is the structuring element. To define the structuring element, you can use the function:

```
SE =strel('disk',r)
```

This creates a disk-SE with radius = r. Besides 'disk' the `strel` function can create SEs in many different forms, e.g. 'line', 'square', 'rectangle', etc. To view a structuring element, SE, created by the strel-function, you can write:

```
disp(SE.Neighborhood)
```

Problem 9) Use morphological opening, followed by closing on the noisy binary image. The proper size and shape of the SE depends on the objects and the noise. In this image, use a disk-SE of radius 3. Insert the image with the noise removed in the answer document.

Problem 10) The vertical lines are a problem since they connect the objects, and thus needs to me removed. One possible way to remove vertical lines is to use `imopen` with a horizontal line-SE.

```
SE = strel('line',len,deg)
```

Where `len` is the length of the line and the `deg` is the angle in degrees. For a horizontal SE, `deg` =0. The length of the line must be greater than the linewidth to remove the lines. Now remove the vertical lines by opening the image with a proper line-SE. Inert the image with the lines removed in the answer document. Save it as *Image1c_clean* for later use.

Problem 11a) Remove the rectangles from the image, leaving only the discs, using `imopen`. Use a disk-shaped SE of proper size. The radius of the SE should to be smaller than the radius of the small disks, otherwise they will be removed. The diameter of the SE must be larger than the smallest side of the rectangle. Save the image with only the disks for later use.

Problem 11b) Remove also the small disks from the image in 11a, i.e. everything but the large disks. Use a disk-shaped SE of proper size. Save the image with only large disks for later use.

Problem 11c) Remove everything but the rectangles from the image *Image1c_clean*. A rectangular SE should be a good choice. Save the image with only rectangles for later use.

Problem 11d) Now create an RGB-image, displaying rectangles, large circles and small disks in different colors. You can create an empty RGB-image as:

```
RGB=zeros(r,c,3);
```

Where `r` and `c` are dimensions of the image, given by;

```
[r,c]=size(Image1c);
```

Now simply insert the images of the different objects in different channels in the RGB-image, e.g. as:

```
RGB(:,:,1)=Im_Rectangles;
```

Notice that you need to combine two images to get an image with only small disks. Insert the image, showing the three types of objects in different colors, in the answer document.

3) Labelling and object features

In a binary image (such as *Image1c_clean*, where you removed the noise and lines), you typically have a number of objects (connected regions of 1s) on the background (0). If you want to extract different features for the objects in the image, you must first label them, assigning each object in the image a unique number. In Matlab, labelling can be done, as:

```
    L = bwlabel(BW)
```

Where `BW` is the input binary image (only 0s and 1s) and `L` is a matrix (same size as `BW`) where each object is assigned a number (integers, from 1 up to number of objects).

Problem 12) Perform labelling on your image *Image1c_clean* (without the noise and lines). If you show the result using `imshow` it will be identical to *Image1c_clean* because the labeled image, `L`, is in double format with integers > 1. If you use

```
    imshow(L,[])
```

the image will be scaled by the max-value, and the objects will be shown in different gray levels. For example, if the image has 10 objects, they will be labelled 1…10, and shown in 10 different shades of gray (0.1, 0.2, …1). Insert the labelled image in the answer document.

Problem 13) In a labelled image, where the objects are identified by numbers, different object features can be computed. A useful Matlab function is:

```
    stats = regionprops(L,properties)
```

which returns a set of properties for a labelled image, L. There is a long list of object features (properties) that can be extracted, e.g. 'Area', 'BoundingBox', 'Centroid', 'ConvexHull', 'Perimeter', 'EulerNumber', etc. The object features specified by 'properties' are returned as `stats`, which is a `struct.`

If you, for example, want to compute the perimeter, the area, and the Euler number for the objects in your labelled image, L, you can write:

```
S = regionprops(L,'Perimeter','Area','EulerNumber');
```

To get for example the perimeter from the object labelled as '1' from the struct `S`, you can write:

```
Perimeter = S(1).Perimeter;
```

Sometimes it is easier to extract the data and use matrices instead of using the datatype struct. To collect the data for all objects, you unfortunately need to use for-loops:

```
for n=1:length(S)
    Perimeter(n)=S(n).Perimeter;
    Area(n)=S(n).Area;
    Euler(n)=S(n).EulerNumber;
end
```

will create:

- The vector `Perimeter`, with the perimeter for all objects
- The vector `Area`, with the area for each object, and
- The vector `Euler`, with Euler number for each object (i.e. the number of objects in the region minus the number of holes in those objects).

Now compute these features for you labelled image. If you for example want to find all objects with area larger than 3000 pixels, you can write;

```
LargeO=find(Area>3000)
```

This will return a vector with the labels of all objects that fulfill the condition. If you want to find for example the perimeter for all the large objects, simply write

```
Perimeter(LargeO)
```

If you want to create an image containing only the objects with area > 3000, you can write:

```
LargeO_Im=zeros(r,c);           % creates an empty matrix


for n=1:length(LargeO)          % the number of large objects
    LargeO_Im(L==LargeO(n))=1;  % Large object area is set to 1
end
```

Insert the image with only the large objects in the answer document, and write the perimeters for the objects.

Problem 14) Look at the histogram for the perimeter of your objects.

```
hist(Perimeter);
```

From the histogram, select a threshold value to separate the class of objects with the smallest perimeter. Write the value of your selected threshold and the labels for all the objects in this class in the answer document.

Problem 15) From the histogram for the perimeter, select a threshold value to separate the class of objects with the largest perimeter. In this class of objects, remove the ones that has holes (can be determined by EulerNumber). Create an image with this class of objects (largest perimeter & no holes) and insert it in the answer document. Write the labels of the selected objects.