# Lab 6:
# Automatical counting of blod cells.
# Experiments with the structure tensor.

Maria Magnusson, 2018, 2019, 2021, 2023
Computer Vision Laboratory, Department of Electrical Engineering,
Linköping University, Sweden

*Based on an older lab developed at the Department of Electrical Engineering.*

## 1 Introduction

☞ Read all instructions before the lab-exercise. Problem marked with a pointing hand are supposed to be solved as preparation for the lab-exercise.

💻 A computer symbol indicates that a MATLAB -script has to be written and demonstrated during the lab-assignment.

Start by extracting the zip-archive containing the files `blod256.tif`, `chess.png`, `cmanEx.m`, `cmanmod.png`, `corr.m`, `corrc.m`, `corrdc.m`, `corrn.m`, `gopimage.m`, `goptab.m`, `nuf0c.tif` into a suitable folder.

## 2 Pattern Recognition with template matching

In this computer section, you will perform experiments with different correlation methods.

### 2.1 Common correlation

The common correlation (or actually cross-correlation) can be used to detect the presence of a pattern $m(x, y)$ in an image $b(x, y)$. The result of the correlation is given by

$$(m \Box b)(x, y) = \sum_{\alpha} \sum_{\beta} m(\alpha, \beta) \cdot b(x + \alpha, y + \beta).$$

Here, the square is used to denote correlation. We will use this method to locate a text-patch in the image `clic`.

Create a file `corrExp.m` with the subsequent contents and execute it:

```
im = double(imread('clic.tif'));
pattern = im(33:33+19, 18:18+19);
fact = 0.99;
rescorr = corr(im, pattern);

figure(1)
colormap(gray(256))
subplot(2,2,1), imagesc(im, [0 255]);
axis image; title('original image'); colorbar;
subplot(2,2,2), imagesc(pattern, [0 255]);
axis image; title('pattern'); colorbar;
subplot(2,2,3), imagesc(rescorr);
axis image; title('result corr'); colorbar;
subplot(2,2,4), imagesc(rescorr>(max(rescorr(:))*fact));
axis image; title('thresh corr'); colorbar;
```

An image named `clic` is loaded and a patch is of size $25 \times 25$ is cut out. They are send to the function `corr`. The image is threshholded at 95 percent of the maximum value, but this can be adjusted. Also the colortable can be chnged to e.g. `jet` if desired.

**QUESTION 1**: Why does this template matching method not work here?

---

## 2.2 Normalized correlation

Normalized correlation, with the MATLAB function `corrn`, is given by

$$\begin{aligned}(m\Box_n b)(x,y) &= \frac{(m\Box b)(x,y)}{\|m(x,y)\| \cdot \|b(x,y)\|_{\text{neighborhood}}} \\ &= \frac{\sum_\alpha \sum_\beta m(\alpha,\beta) \cdot b(x+\alpha, y+\beta)}{\sqrt{\sum\sum_{\alpha,\beta} m^2(\alpha,\beta) \cdot \sum\sum_{\alpha,\beta\in\text{neighborhood}} b^2(x+\alpha,y+\beta)}}.\end{aligned}$$

Here we divide with the norm of the pattern and the image in the actual environment. In this way we avoid high values of the correlation result because of high image values. The maximum value of $(m\Box b)(x,y)$ is 1 and it is obtained only when there is a perfect match between pattern and image. Extend the MATLAB script `corrExp.m` with normalized correlation and show the results in `figure(2)`.

**QUESTION 2**: Did you get a good result? What is the correlation result at the pattern position? Click in the image and check.

---

### 2.3 Correlation without local DC-level. Covariance.

Correlation without local average, with the MATLAB function `corrdc`, is given by

$$(m\square b)_{DC}(x,y) = \sum_{\alpha} \sum_{\beta} [m(\alpha,\beta) - \bar{m}] \cdot [b(x+\alpha, y+\beta) - \bar{b}].$$

The local mean in the image, $\bar{b}$, is calculated in a neighborhood, whose size is determined by the size of the pattern. This prevents high correlation values due to high image values. Extend the MATLAB script `corrExp.m` with correlation withot DC-level and show the results in `figure(3)`.

**QUESTION 3**: Did you get a good result? In which aspect is this result better than the result obtained with normalized correlation?

---

### 2.4 Normalized correlation without local DC-level. Correlation coefficient

Normalized correlation without local DC-level i.e. the correlation coefficient, with the MATLAB function `corrc`, is given by

$$(m\square_c b)(x,y) = \frac{\sum_{\alpha} \sum_{\beta} [m(\alpha,\beta) - \bar{m}] \cdot [b(x+\alpha, y+\beta) - \bar{b}]}{\sqrt{\sum\sum_{\alpha,\beta} [m(\alpha,\beta) - \bar{m}]^2 \cdot \sum\sum_{\alpha,\beta\in \text{neighborhood}} [b(x+\alpha, y+\beta) - \bar{b}]^2}}$$

Extend the MATLAB script `corrExp.m` with normalized correlation withot DC-level and show the results in `figure(4)`.

**QUESTION 4**: What about your result?

---

### 2.5 Correlation using the blood cells image

In the previous examples, the pattern matched perfectly with the image. In a real case, the consistency is usually not perfect.

**DEMO A**: Load the image `blod256.tif`, cut out one representative blood cell and test `corr`, `corrn`, `corrdc` and `corrc` again. You will need to change the factor `fact` for the thresholding. Show the results after the four correlation methods, before and after thresholding, in one figure!

**QUESTION 5**: Comment on your your results!

---

## 3 Thresholding with hysteresis

In the lecture notes, thresholding with hysteresis was described. Create a file `hysteresis.m` with the subsequent contents and execute it.

```
nuf = double(imread('nuf0c.tif'));
binvect = [0:1:255];
histo = hist(nuf(:), binvect);
figure(1)
colormap(gray(256))
subplot(2,2,1), imagesc(nuf, [0 255]);
axis image; title('image nuf0c'); colorbar
subplot(2,2,2), plot(binvect, histo, '.-b');
axis tight; title('histogram')
```

You should see the nuf0c image (see figure below, left) and its histogram.

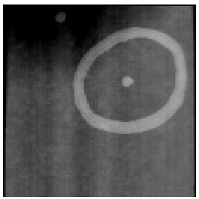
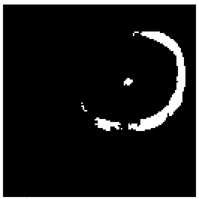
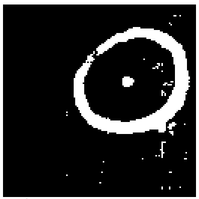
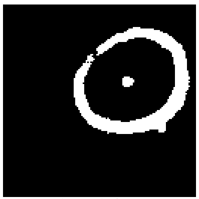


image nuf0c


after threshold 1


after threshold 2


resulted image

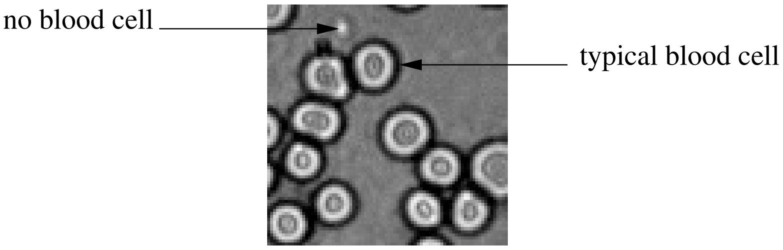
**DEMO B**: Complete the `hysteresis.m` MATLAB script. Try to find thresholds so that you get the two central images. Then implement thresholding with hysteresis so that you get the resulting right image. Please include a `pause()` command within the loop so that you can show the teacher how the object grows during the hysteresis.

## 4 Blood cells counting

Microscopy images are very well suited for two-dimensional image processing since these images are in principle two-dimensional. You will here design two MATLAB program that counts the number of red blood cells in an image.

### 4.1 Blood cells counting using thresholding with hysteresis

A section of a typical blood cell image is shown below.



**It is recommended to focus on the bright rings instead of the dark rings when calculating the number of blood cells!** A problem that may arise is that one blood cell may be split into two and therefore be counted as two. Another possible problem is that small objects that are not blood cells may be counted as well. These are some of the problems that you should be able to handle.

**DEMO C**: Load the image `blod256.tif` and count the number of blood cells. Solve the problem by using thresholding with hysteresis!
Compare the original image (`blod`) and the binary detected blodcells image (`detectim`) by overlay (`detectim`) on (`blod`). Suggested code:

```
resultim = zeros(256,256,3);
resultim(:,:,1) = (detectim==1).*255+(detectim==0).*blod;
resultim(:,:,2) = blod;
resultim(:,:,3) = blod;
figure(99), imshow(resultim/255);
```

*Tip:* Labelling and `bwareaopen.m` are also useful.

**QUESTION 6**: Discuss which blood cells the program found, which blood cells the program missed, and why.

---

## 4.2 Blood cells counting using correlation

**DEMO D**: Write a MATLAB script which count the number of blood cells by using correlation!

**QUESTION 7**: Discuss which blood cells the program found, which blood cells the program missed, and why.

---

# 5 The structure tensor

The theory for this lab moment can be found in the lecture slides about the structure tensor. The program `cmanEx.m` is shown below.

```
% Read original image;
im = double(imread('cmanmod.png'));

figure(1)
colormap(gray(256))
subplot(1,1,1); imagesc(im, [0 256]); colorbar;
axis image; axis off;

% Compute derivative images
dx = [1 0 -1; 2 0 -2; 1 0 -1]/8; % sobelx
fx=conv2(im,dx, 'valid'); % With valid you get rid of the dark frame
maxv = max(max(abs(fx)))/2;

figure(2)
colormap(gray(256))
subplot(1,2,1); imagesc(fx, [-maxv maxv]); colorbar('horizontal');
axis image; axis off;
title('f_x')
```

Execute the program `cmanEx.m` in MATLAB . It starts by loading a slightly modified verison of the well-known test-image "camera man". An artificial pattern, a bare code, has been added in the upper right part of the image. The derivative in the x-direction $f_x = \partial f(x, y)/\partial x$ is shown in an image.

## 5.1 Implementation of the structure tensor

You will now inplement the structure tensor step by step. You will produce similar images as in the lecture slides. The derivative in the x-direction $f_x = \partial f(x, y)/\partial x$ is already implemented. Also implement the derivative in the y-direction $f_y = \partial f(x, y)/\partial y$ and show it in an image besides $f_x$.

**QUESTION 8**: How can you see that the $f_y$-image looks reasonable?

---

Now implement all the components of the structure tensor, i.e. $T_{11}$, $T_{12}$, and $T_{22}$.

**QUESTION 9**: Do your $T_{11}$ and $T_{22}$ images of the camera man look similar as in slide **"T11 and T22 before LP-filtering"**? Some line-like structures have resulted in lines with gaps inbetween. Point out a few such places!

The gaps can be filled by LP-filtering. Use a separable $19 \times 19$ Gaussian:

```
sigma=0.25;
lpH=exp(-0.5*([-9:9]/sigma).^2);
lpH=lpH/sum(lpH); % Horizontal filter
lpV=lpH';         % Vertical filter
```

Perform convolution, first with `lpH` and then with `lpV`. The given `sigma=0.25` gives almost no effect. Check this! Then adjust it so that your $T_{11}$ and $T_{12}$ images look similar as in slide **"T11 and T22 after LP-filtering"**?

**QUESTION 10**: Which `sigma` did you use?

A complex representation of a vector can be constructed from the tensor elements: $\mathbf{z} = T_{11} - T_{22} + j2T_{12}$.
The angle of this vector is **doubled** compared to the angle of the gradient vector: $\mathbf{z_g} = f_x + jf_y$.
Compute the magnitude image $\mathrm{abs}(\mathbf{z})$ and the angle image $\arg(\mathbf{z})$. For the angle-image, use the `goptab` color table, constructed to show angles in the range: $0 \leq \arg(\mathbf{z}) \leq 2\pi$.
The argument can be calculated using the `atan2` function, for which
`-pi <= atan2(Y,X) <= pi`.
Therefore, to fit `goptab`, you must add $2\pi$ to negative angles. Check so that your images look similar as in slide **"abs(z) and arg(z), magnitude and angle"**. It is possible to have different colortables for different subplots, by e.g. using the commands

```
gca = subplot(1,2,1);
colormap(gca, gray(256));
```

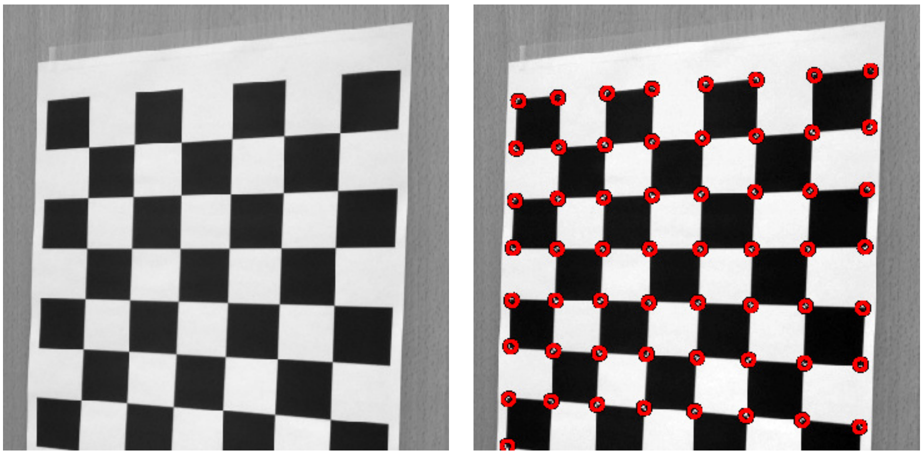
and similarly for the `subplot(1,2,2)` area.
You can also send **z** to the gopimage function, i.e. `gopimage(z)`. This shows a nice image where $\mathrm{abs}(\mathbf{z})$ and $\arg(\mathbf{z})$ has been combined into one color image. Check so that your image look similar as in slide **"Left: abs(z) and arg(z) combined in one image."**.

**DEMO E**: Show all the images that you have produced in this section to the teacher!

## 5.2 Finding points of interest in a calibration pattern

For computer vision applications, a calibrated camera is necessary. A calibration pattern used at the Computer Vision Lab (CVL) is shown in the figure below, left. The pattern is shown to the camera in several different orientations. By establishing the transformation between the points in the world and points in the images, the inner parametes (e.g. zoom, optical center) of the camera can be established. Also, the outer parameters (camera translation and rotation) in relation to the world coordinate system can be determined. The corners of the chess pattern are used as calibration points. The world points of interest are measured carefully in advance. (The distance between them is 40 mm.) The position of the image points, however, must be determined with image analysis. This is an important step in the calibration procedure. Load the calibration pattern with:

```
im = double(rgb2gray(imread('chess.png')));
```



**DEMO F**: Implement your own Harris corner detector based on the structure tensor to solve the problem! Produce an image with circles at the positions of the calibration points, see the image below, right, where we used the following command to draw the circles:

```
viscircles([y x], r,'EdgeColor','r');
```

**QUESTION 11**: You will get a better result if you restrict the original image to `im(2:end-1,2:end-1)`. Why?

☞ **QUESTION 12**: Regard the corner response function:
$C_{Harris} = \det\mathbf{T} - k(tr\mathbf{T})^2$. Rewrite it by inserting derivatives $f_x$ and $f_y$:
$C_{Harris} = f_x^2 f_y^2 - (f_x f_y)^2 - k(f_x^2 + f_y^2)^2 = 0 - k(f_x^2 + f_y^2)^2$.
It seems that the first term of $C_{Harris} = 0$?!
Explain why this is not the case in practice.