Lab 7:
Image restauration. Edge detection with Canny and Hough transform.

Maria Magnusson, 2018 - 2023
Computer Vision Laboratory, Department of Electrical Engineering,
Linköping University, Sweden
Based on an older lab developed at the Department of Electrical Engineering.

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:
`baboon.tif, foppa.tif, skylt.tif, hus.tif, polyg.tif`
`addnoise.m, circconv.m, wiener.m, MagnGradSobel.m`
into a suitable folder.

1 Wiener filtering

In the general case, where both noise suppression and inverse filtering has to be done, the Wiener filter is given by

$$W(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + S_N(u,v)/S_F(u,v)} \ ,$$

where S_N and S_F denote the noise and the image power spectrum, respectively. Instead of calling the Wiener filter function with the parameters $S_N(u,v)$ and $S_F(u,v)$, we use the term

$$\frac{S_N(u,v)}{S_F(u,v)},$$

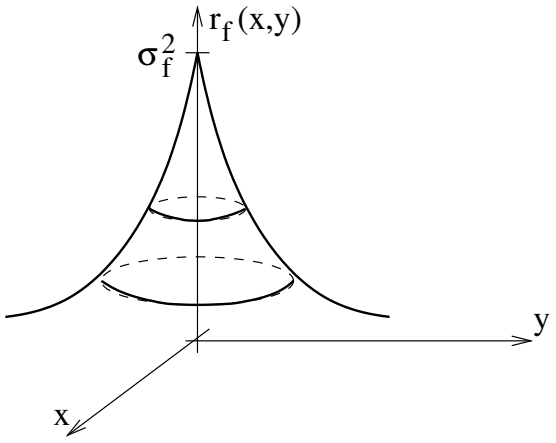
which is the inverse signal-to-noise ratio for a given frequency (u,v). We assume that the noise is white with variance σ_n^2, so that we can write the autocorrelation function of the noise as

$$r_n(x,y) = \sigma_n^2 \cdot \delta(x,y). \tag{1}$$

The model of the autocorrelation function of the image is

$$r_f(x,y) = \sigma_f^2 \cdot \rho^{\sqrt{x^2+y^2}}, \tag{2}$$

where the variance is σ_f^2, and the normalized correlation between two adjacent pixels are given by the parameter ρ, which is a number between 0 and 1, see figure below.



QUESTION 1: Calculate the power spectra $S_N(u,v)$ and $S_F(u,v)$ if the autocorrelation functions $r_n(x,y)$ and $r_f(x,y)$ are given by equation (1) and (2). Use the formula collection. Unfortunately, $S_F(u,v)$ is difficult to calculate. Obtain an approximate value by calculating in one dimension only, i.e. start with $r_f(x,0) = \sigma_f^2 \cdot \rho^{|x|} = \sigma_f^2 \cdot e^{-(-ln\rho \cdot |x|)} = ...$

The program **wiener** utilizes the fact that the autocorrelation function and the power spectrum is a Fourier transform pair. We therefore need only two parameters, ρ, and the signal-to-noise ratio,

$$SNR = 10 \cdot \log \frac{\sigma_f^2}{\sigma_n^2}.$$

The value of ρ varies between different images, **baboon**, for example has $\rho = 0.83$. However, the value is quite constant for similar images, such as images of faces or images of license plates. Finally we introduce the parameter r. By decreasing r from 1 towards 0, the Wiener filter will turn into an inverse filter. Thus, the appearance of the *parametric Wiener filter* will be

$$W(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + r \cdot S_N(u,v)/S_F(u,v)}.$$

QUESTION 2: Show that $W(u, v)$ becomes an inverse filter if $r = 0$.

Here follows a short MATLAB program. At first, the image is convolved with an average filter. Then noise is added and finally the image is Wiener filtered. Create a file `wienExp.m` with the subsequent contents and execute it:

```
f = double(imread('skylt.tif')); % load image
h0 = ones(5,5)/25;               % a suitable filter
g = circconv(f,h0);              % convolve
snr = 20;
h = addnoise(g,snr);             % add noise with SNR=20dB
rho = 0.80;
r = 1.0;
fhat=wiener(h,h0,snr,rho,r);     % Wiener filter

figure(1), colormap(gray)
subplot(2,2,1), imagesc(f,[0 255]), axis image
title('original image'); colorbar
subplot(2,2,2), imagesc(h,[0 255]), axis image
title('degraded image'); colorbar
subplot(2,2,3), imagesc(fhat,[0 255]), axis image
title('restored image'); colorbar
```

Wiener filtering is consequently performed by calling the MATLAB function **wiener** with

- the degraded image `h`
- the point spread function `h0` (= a convolution filter)
- the signal-to-noise ratio `snr`
- the parameter `rho` in the model of the image autocorrelation function
- the parameter `r`

QUESTION 3: Does the Wiener filter manage to restore the degraded image?

Yes, partly but there are still artifacts left

1.1 Only noise suppression

Use the image **skylt** in this and the following exercises. Study the case with only noise suppression, i.e. $h(x, y) = \delta(x, y)$. This gives no degrading convolution - right?

```
h1 = 1;                          % a very small filter
```

QUESTION 4: Try two different signal-to-noise conditions, such as 10 and 5 dB. What is the result of the noise suppression?

The degraded images look better than before. The Wiener filter has a harder time to restore the image with lower SNR

1.2 Only inverse filtering

Study the case with very little noise, i.e. almost only inverse filtering. Set the signal-to-noise ratio to a high value, for example SNR = 40dB. Choose a big filter, either $h2$ or $h3$ below.

```
h2 = ones (7,7)/49;              % a big filter
h3 = ones (11,11)/121;           % a very big filter
```

QUESTION 5: What is the result of the inverse filtering?

The degraded image becomes very blurry, but the W-filter restores it pretty accurately

QUESTION 6: Also, try to set a small value of the parameter r. Why will the result be bad if r is set to 0?

Because we try to do inverse filtering without accounting for the added noise

1.3 Both noise suppression and inverse filtering

DEMO A: Write a program `wienExp2.m`, where you study the general case, both inverse filtering and noise suppression. Use the filter $h3$.

QUESTION 7: For which noise level is the text difficult to read?

For lower signal to noise ratios (SNR < 30)

QUESTION 8: Is it possible to improve the image quality by adjusting the parameter r? Experiment until you get a good value of r. The mathematically correct value is $r = 1$. Consider why a different value might be better for a human.

According to our observations, varying the value of r does not yield very different results except for r = 0

1.4 Motion blur

QUESTION 9: Construct a filter, which provides strong horizontal motion blur in the picture. What does the filter look like?

h_rect = [1 1 1 1 1 1 1 1 1] / 9

DEMO B: Write a program `wienExp3.m`, where you study the motion blur case.

1.5 The Autocorrelation function

Suppose that we want to calculate ρ for an image. The Wiener filter requires that the mean value of the image is 0. If the mean value of the image is not 0, it may be subtracted before the Wiener filtration and added afterwards. This is performed in the function **wiener**. Consequently, the parameter ρ should be calculated on an image f with subtracted mean value $\bar{f}$. Then the autocorrelation becomes

$$r_f(x,y) = [(f-\bar{f}) \square (f-\bar{f})](x,y).$$

The parameter ρ can now be estimated as

$$\rho_x = r_f(1,0)/r_f(0,0), \quad \rho_y = r_f(0,1)/r_f(0,0), \quad or \quad \rho = \frac{\rho_x + \rho_y}{2}.$$

QUESTION 10: Check that the three estimates are reasonable by using equation (2).

Yes they are reasonable

DEMO C: Write a program `autocorr.m`, where you calculate the autocorrelation function $r_f(x,y)$ for some images, such as **skylt**, **baboon** and **foppa**. The calculation should be performed in the Fourier domain.

QUESTION 11: Look at the image of the autocorrelation function! Is it similar to the sketch of $r_f(x,y)$ below equation (2)? Mention why the result is not perfect, especially close to the borders of the image.

Yes, it's pretty similar, but at the edges the correlation is the most inaccurate due to the offset being high

QUESTION 12: Calculate ρ from the autocorrelation function!

p_skylt = 0.7957

p_baboon = 0.8247

p_foppa = 0.9628

2 Edge detection with Canny

The most obvious way to perform edge detection so that edges are marked with a one-pixel wide line is the following procedure;

- Apply the filters $sobel_x$ and $sobel_y$ on the image $f(x, y)$, so that the approximative derivative images, $f_x(x, y)$ and $f_y(x, y)$, are received. Then calculate the magnitude of gradient, $\sqrt{f_x(x, y)^2 + f_y(x, y)^2}$.
- Threshold the magnitude gradient image.
- Perform thinning to an 8-connective skeleton.

This procedure is performed on the image **hus** in the MATLAB -script `MagnGradSobel.m`. Execute `MagnGradSobel.m`. Remember to zoom the figure(2) window over almost the full screen, so that you can see every pixel.

QUESTION 13: Vary the threshold `T`. Is it possible to find a threshold that preserves the chimney and leaves the roof ridge unbroken, but do not mark the edges of the roofing tiles?

It is not possible :3

QUESTION 14: Choose a threshold that gives a rather "nice looking" image!

16

Canny's edge detection includes the following procedure;

- Compute the magnitude and direction of the gradient.
- Apply non-maxima suppression of the magnitude of the gradient.
- Apply double thresholding. It is recommended that the lower threshold is 0.4 times the higher threshold.
- Expand the 1:s (edges) in the high-threshold image to neighboring pixels that are marked as edges in the low-threshold image (thresholding with hysteresis).

QUESTION 15: Canny's edge detection detects the *maximum of the magnitude of the gradient* in the *direction of the edge normal*. What is the advantage of this compared to thresholding the magnitude of the gradient followed by thinning?

Canny's surpress everything that is not a sharp change => less noise sensitive & more precise

QUESTION 16: Canny's edge detection performs edge thresholding with hysteresis. What is the advantage of this?

weak line segments with supporting strong line segments are maintained => produce sharper edges

---

Extend the MATLAB script `MagnGradSobel.m` with code for Canny's edge detection:

```
[cannyim1,T] = edge(im,'canny');
T
```

and look at the resulting image. Note that edge function seems to normalize the magnitude of the gradient.

QUESTION 17: Which two thresholds do the operation **edge** automatically choose?

0.0188 0.0469

Unfortunately, our first attempt with Canny gives an image that contains an excessive number of small short edges. However, the threshold values in the Canny method can be varied by passing an appropriate value of `T` as

```
cannyim2 = edge (im, 'canny', [0.4*T T]);
```

QUESTION 18: Find a value of `T` that gives a good Canny-image. Preserve the chimney and leave the roof ridge unbroken, but do not mark the edges of the roofing tiles!

T = 0.2

QUESTION 19: Drag in your MATLAB figure to make it smaller. Why do more and more lines seem to be broken then?

Because the edges are only 1 px wide which are too small to get rendered at high zoom out

---

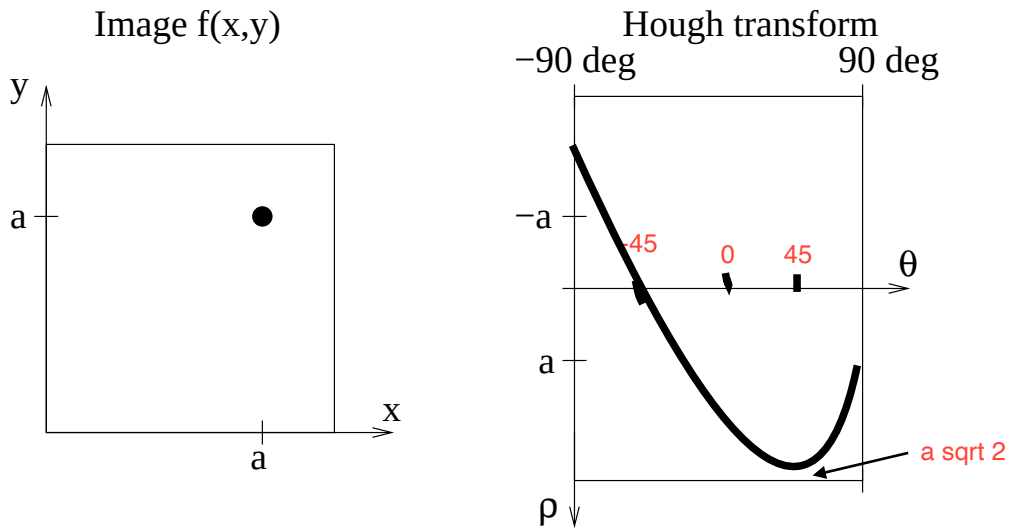
3 The Hough transform

The Hough transform is a global method to detect lines, circles and other patterns. The Hough transform was first introduced by Hough who patented the method in the sixties. We use the variant that transforms a point in the image to a sinusoid in the Hough transform. This special case of the Hough transform is also called the Radon transform. Look at the following equation:

$$\rho = x\cos\theta + y\sin\theta$$

Each point in the image $f(x, y)$ is transformed to a sinusoid in the Hough transform $H(\rho, \theta)$. The sinusoid satisfies the above equation.

QUESTION 20: Sketch the Hough transform $H(\rho, \theta)$ of the image $f(x, y)$ below!



DEMO D: Create a file `HoughTest1.m` with the subsequent contents and execute it. The result should confirm the sketch that you made above.

```
% Make an image with a point
% ===========================
im = zeros(64,64);
im(48,48) = 1;

figure(1)
subplot(2,2,1), imagesc(im);
axis image; axis xy; colorbar;
title('Image'),

% Call the Hough transform
% ========================
[H,T,R] = hough(im, 'Theta', -90:89);

subplot(2,2,2), imagesc(T,R,H);
xlabel('\theta'), ylabel('\rho');
title('Hough transform'), colorbar;
```

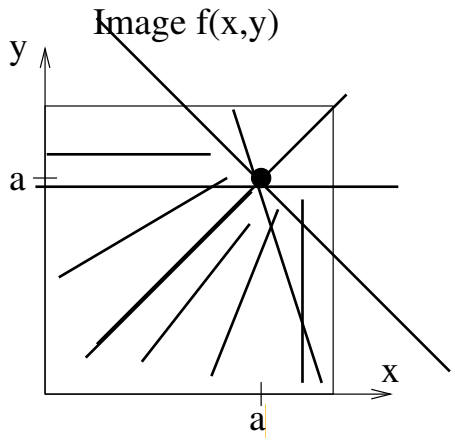
Extend the file `HoughTest1.m` with the subsequent contents and execute it. By `houghpeaks`, up to 200 values above the threshhold 0 are detected in the Hough transform. In our case it means that all values >0 are detected. The function `houghlines` then computes the inverse Hough transform of the detected points.

```
% Detect peaks
% ============
P  = houghpeaks(H,200,'threshold', 0);
x = T(P(:,2)); y = R(P(:,1));
hold on
plot(x,y,'s','color','red'), hold off

% Inverse Hough transform give Hough lines
% ========================================
lines = houghlines(ones(size(im)),T,R,P);

% Overlay Hough lines on image
% ============================
subplot(2,2,3), imagesc(im), hold on
title('Result'),
axis image; axis xy; colorbar;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
end
hold off
```

QUESTION 21: Sketch the resulted image and explain the look of it!



The image consists of multiple lines that all pass through our original point

DEMO E: Copy `HoughTest1.m` to `HoughTest2.m`. Add four more points:

```
im(1,1) = 1;
im(16,16) = 1;
im(32,32) = 1;
im(48,48) = 1;
im(64,64) = 1;
```

Change the houghpeaks command to:

```
P  = houghpeaks(H,10,'threshold',ceil(0.5*max(H(:))))
```

Now `houghpeaks` detects up to 10 values above the threshhold `ceil(0.5*max(H(:)))` in the Hough transform.

QUESTION 22: Describe what happens!

The resulting image is a line through all of the points

DEMO F: Try the Hough transform on the image **polyg**. Such an image may be obtained during industrial inspection aimed for checking the dimensions of a machine detail. First apply Canny to get edge lines. Then apply the Hough transform to locate the four strongest lines. Overlay these lines on the original image. The result will look like as shown to the left below.

DEMO G: Finally, you will try edge linking, so that the lines stops at the corner of the polygon. Then the parameters `'FillGap'` and `'MinLength'` must be send to `'houghlines'`. As the first parameter to `'houghlines'`, the slightly dilated Canny result is recommended. The result will look like as shown to the right below.