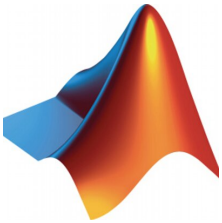
INM460 / IN3060 Computer Vision

In this lab we learn how to apply filters to blur and sharpen images, detect edges, corners and textures in images.



Resources:

The lab will make use of Matlab, and the following data files in the Lab materials 03 on Moodle in the Session 3 area:

1. Hawkes_Bay_NZ.jpg
2. LondonEye.jpg
3. Westminster.jpg
4. Class.jpg

Image Filtering

Low pass filtering (aka smoothing) is employed to remove high spatial frequency noise from a digital image. The low-pass filters usually employ moving window operator that affects one pixel of the image at a time, changing its value by some function of a local region (window) of pixels. The operator moves over the image to affect all the pixels in the image.

A high-pass filter can be used to make an image appear sharper. These filters emphasize fine details in the image - the opposite of the low-pass filter. High-pass filtering works in the same way as low-pass filtering; it just uses a different convolution kernel.

When filtering an image, each pixel is affected by its neighbours, and the net effect of filtering is moving information around the image.

Task 1: Smoothing

Mean filtering is easy to implement. It is used as a method of smoothing images, reducing the amount of intensity variation between one pixel and the next resulting in reducing noise in images.

The idea of mean filtering is simply to replace each pixel value in an image with the mean (`average') value of its neighbours, including itself. This has the effect of

eliminating pixel values that are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean. Often a 3×3 square kernel is used:

mf = ones(3,3)/9

mf =

0.1111	0.1111	0.1111
0.1111	0.1111	0.1111
0.1111	0.1111	0.1111

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Y = imfilter(X,h) filters the multidimensional array A with the multidimensional filter h. (you can also use filter2() for 2-d arrays and filters as a substitute)

Now we want to apply the kernel defined in the previous section using **imfilter()**:

```
img = londonEye;
img = imread('LondonEye.jpg');
imgG = rgb2gray(img);
imgd = im2double(imgG); % imgd in [0,1]
f = ones(3,3)/9;
img1 = imfilter(imgd,f);
figure;
subplot(121); imshow(imgd);
subplot(122); imshow(img1);
linkaxes;
```

We can see the filtered image (right) has been blurred a little bit compared to the original input (left).

Try to experiment with different sizes for the kernel f.

As mentioned earlier, the low pass filter can be used denoising. Let's test it. First, to make the input a little bit dirty, we spray some pepper and salt on the image, and then apply the mean filter:

```
img = imread('cameraman.tif');
imgd = im2double(img);   % imgd in [0,1]
imgd = imnoise(imgd,'salt & pepper',0.02);
f = ones(3,3)/9;
img1 = imfilter(imgd,f);
subplot(121);imshow(imgd);
subplot(122);imshow(img1);
linkaxes;
```

It has some effect on the salt and pepper noise but not much. It just made them blurred.

How about trying the Matlab's built-in median filter?

```
I = imread('cameraman.tif');
J = imnoise(I,'salt & pepper',0.02);
K = medfilt2(J);
subplot(121);imshow(J);
subplot(122);imshow(K);
linkaxes;
```

Much better. Unlike the previous filter which is just using mean value, this time we used **median**. Median filtering is a nonlinear operation often used in image processing to reduce "salt and pepper" noise. The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighboring entries.

Also note that the **medfilt2()** is **2-D** filter, so it only works for grayscale image but you can use imfilter to work on RGB images as well.

In Task 2, you will learn how to separate channels and apply a 2-D filter to each channel and combine the colour channels.

h = fspecial(type) creates a two-dimensional filter **h** of the specified type. It returns **h** as a correlation kernel, which is the appropriate form to use with **imfilter()**. The **type** is a string having one of these values:

Value	Description
average	Averaging filter
disk	Circular averaging filter (pillbox)
gaussian	Gaussian lowpass filter
laplacian	Laplacian of Gaussian filter
motion	Approximates the linear motion of a camera

prewitt	Prewitt horizontal edge-emphasizing filter
sobel	Sobel horizontal edge-emphasizing filter

Here is an example of using **disk** filter:

```
I = imread('cameraman.tif');
radius = 1;
J1 = fspecial('disk', radius);
K1 = imfilter(I,J1,'replicate');
radius = 10;
J10 = fspecial('disk', radius);
K10 = imfilter(I,J10,'replicate');
subplot(131);imshow(I);title('original');
subplot(132);imshow(K1);title('disk: radius=1');
subplot(133);imshow(K10);title('disk: radius=10');
linkaxes;
```

Task 2: Removing noise in RGB images

The filter we used to remove the "salt & pepper" type noise was **medfilt2()**. However, as the "2" in the name indicates it's for 2-D array, it won't work for RGB image unless we decomposed each RGB channel and concatenate after the filtering each channel. That's exactly the following script does:

```
I = imread('Westminster.jpg');
J = imnoise(I,'salt & pepper',0.2);

% filter each channel separately
r = medfilt2(J(:, :, 1), [3 3]);
g = medfilt2(J(:, :, 2), [3 3]);
b = medfilt2(J(:, :, 3), [3 3]);
```

```
% reconstruct the image from r,g,b channels
K = cat(3, r, g, b);
figure
subplot(121);imshow(J);
subplot(122);imshow(K);
linkaxes;
```

Try and experiment with different window sizes.

Exercise 1: Load one of your images (or use ‘class.jpg’ provided on Moodle) and blur only the faces. The result should look like the image below.

Hint: you can manually find the coordinates of the faces and apply the filter to it and put it back in the big image.

Task 3: Sharpening

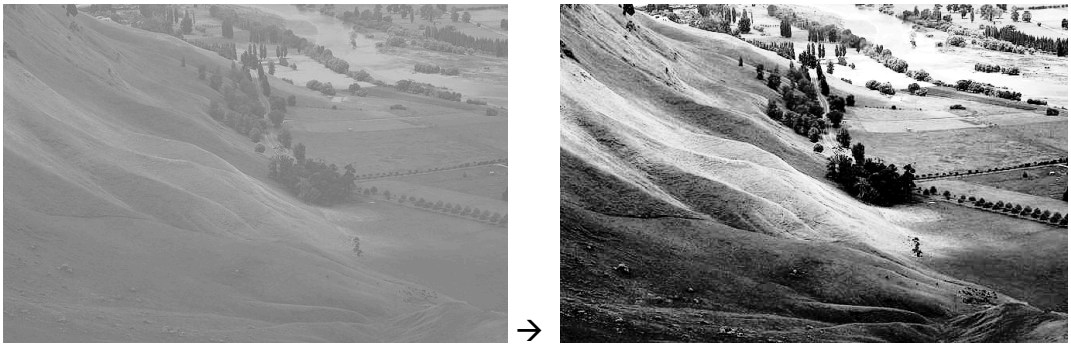
A high-pass filter can be used to make an image appear sharper. These filters emphasize fine details in the image - the opposite of the low-pass filter. High-pass filtering works in the same way as low-pass filtering; it just uses a different convolution kernel:

```
K = [0, -1, 0; -1, 5, -1; 0, -1, 0];
img = imread('Westminster.jpg');
imgG = rgb2gray(img);
imgd = im2double(imgG);
img1 = imfilter(imgd,K);
subplot(121);imshow(imgG);
subplot(122);imshow(img1);
linkaxes;
```

Task 4: Histogram Equalization

The Histogram Equalization algorithm enhances the contrast of images by transforming the values in an intensity image so that the histogram of the output image is approximately flat.

```
img = imread('Hawkes_Bay_NZ.jpg');
figure, img_eq = histeq(img); imshow(img_eq);
```



Task 5: Edge Detection

There are several methods for edge detection in images. In this section we look into detection of vertical and horizontal edges in an image, and also implement Sobel, and LoG methods.

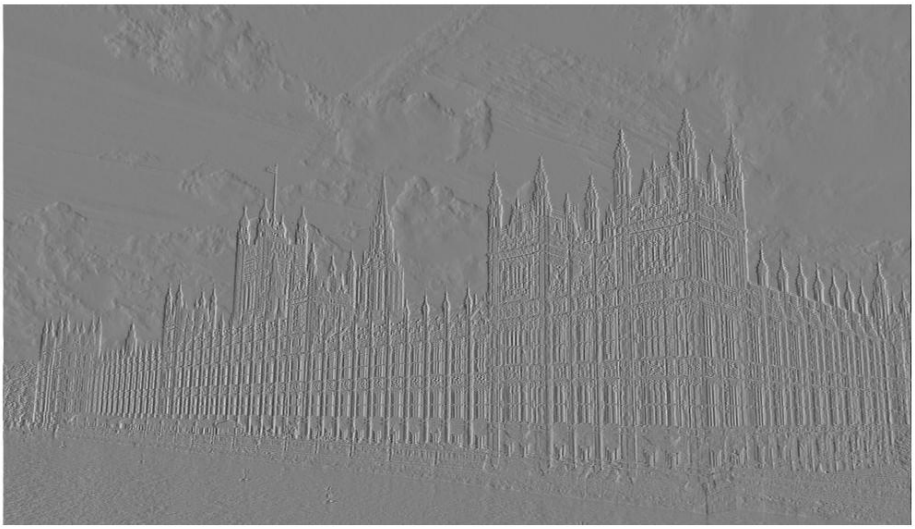
First load and convert the image:

```
img = imread('Westminster.jpg');
imgG = rgb2gray(img);
imgd = im2double(imgG);

I = imgd;
```

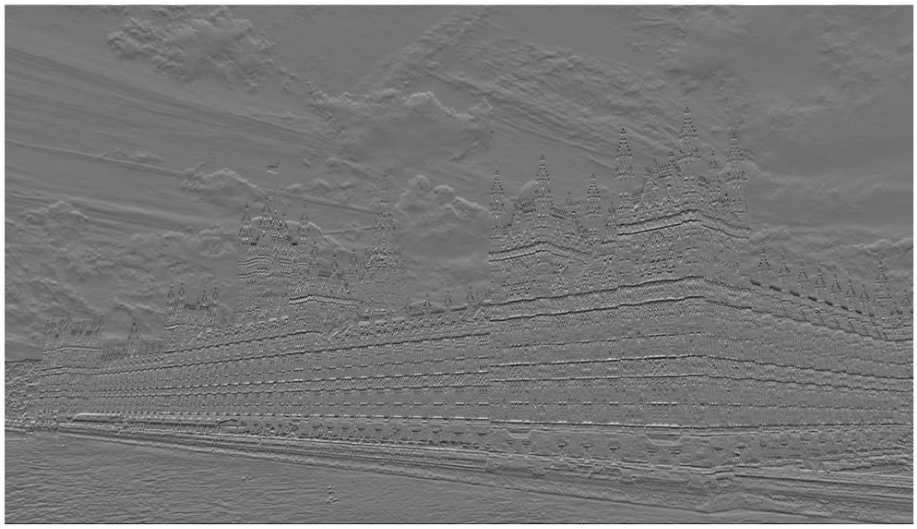
Vertical Edges:

```
K = 0.5*[-1 0 1];
Ix = imfilter(I, K);
figure;
imshow(Ix,[]); % show full dynamic range with []
```



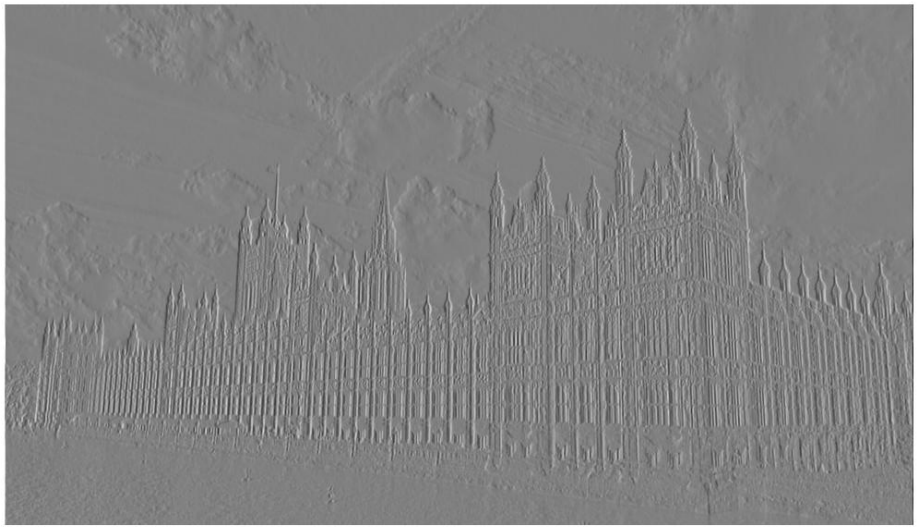
Horizontal Edges:

```
K = 0.5*[-1 0 1]';
Iy = imfilter(I, K);
figure;
imshow(Iy,[]);
```



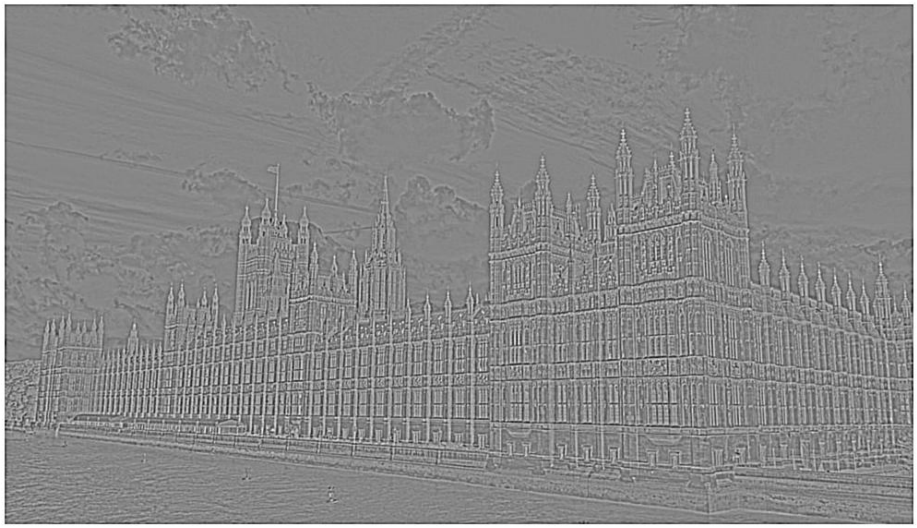
Sobel Edge detection for vertical edges:

```
S = [-1, 0, 1 ; -2, 0, 2; -1,0,1];
IS = imfilter(I, S);
figure; imshow(IS,[]);
```



LoG filter:

```
K = fspecial('log', 11, 1);
J = imfilter(I, K);
figure; imshow(J,[]);
title('LoG');
```



Exercise 2:

Use peppers image (I = imread('peppers.png');) and apply Sobel and LOG edge detection methods and compare the results.

(hint: you can use function “edge” to find edges in images and `imshowpair(BW1,BW2,'montage')` to show two images next to each other)