Smoothing and template matching

Outline

- Linear filtering of images with Matlab
- Non-linear filtering
- Design some linear and nonlinear smoothing operators and check their performance on one test image corrupted by noise

- Template matching

Linear filtering

- In Matlab, convolution of an image I(i,j) with a kernel mask(i,j) is accomplished either through the **imfilter** of **conv2** operators:

```
img = imread('disks.png');
imgPoisson = imnoise(img, 'poisson');
imgSpeckle = imnoise(img, 'speckle');
imshow(imgPoisson); figure; imshow(imgSpeckle);

mask = ones([3 3]) ./ 9 ;

imgOut = imfilter(imgPoisson, mask, 'conv');
figure; imshow(imgOut)
imgOut = conv2(double(imgPoisson), double(mask), 'same');
figure; imshow(imgOut ./ max(imgOut(:)))
```

conv2 is slower than imfilter as this latter uses the Intel Performance Primitives Library

Linear filtering

- Accuracy of the smoothing operator can be estimated by computing the mean squared error:

```
img = imread('disks.png');
imgPoisson = imnoise(img, 'poisson');
imgSpeckle = imnoise(img, 'speckle');
imshow(imgPoisson); figure; imshow(imgSpeckle);

mask = ones([3 3]) ./ 9 ;

imgOut = imfilter(imgPoisson, mask, 'conv');
figure; imshow(imgOut)
imgOut = conv2(double(imgPoisson), double(mask), 'same');
figure; imshow(imgOut ./ max(imgOut(:)));

d = mse( double(imgOut(:) ) - double(img(:)) )
```

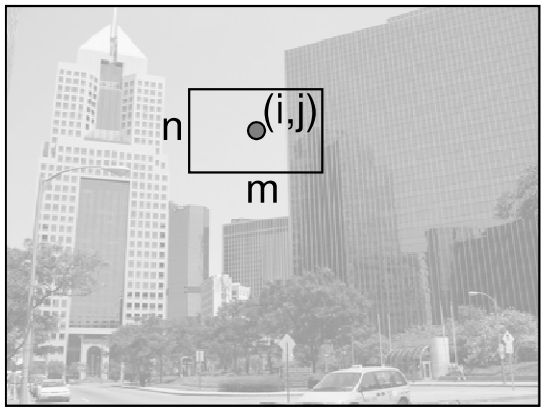
Linear filtering

- The makeGauss.m function returns a Gaussian kernel mask based on the σ value specified in input:

```
img = imread('disks.png');
imgPoisson = imnoise(img, 'poisson');
imgSpeckle = imnoise(img, 'speckle');

mask = makeGauss(1.2);

imgOut = imfilter(imgPoisson, mask, 'conv');
figure; imshow(imgOut);

d = mse( double(imgOut(:)) - double(img(:)) )
```

Non-linear filtering

- In Matlab, application of non-linear filters is managed through the **nlfilter** operator
- For each pixel (i,j) of the input image nlfilter passes the [n m] neighborhood of the pixel to a suitable function that performs computation of the output value




```
img = imread('disks.png');
imgPoisson = imnoise(img, 'poisson');
imgOut = nlfilter(imgPoisson, [n m], @myMedian)
```

myMedian.m

```
function out = myMedian(inputData)
out=median(inputData(:));
```

Median filtering

- Actually, application of the median filter can be managed more conveniently through the **MEDFILT2** operator

```
img = imread('disks.png');
imgPoisson = imnoise(img, 'poisson');
imgOut = MEDFILT2(imgPoisson, [n m])
```

Assignment

- Design the bilater filter and use the nlfilter operator to filter the 'disks.png' image corrupted by noise (Poisson, Speckle, Gaussian)
  - Define the m-file *bilateral(inputData)* to compute coefficients of the [n n] kernel mask based on inputData values
  - Apply the nonlinear filter through the operator *nlfilter(img, [n n], @bilateral)*
- Measure the accuracy (mean squared error) of the Gaussian, Bilateral and Median filters on the 'disks.png' image corrupted by noise