# Introduction to Medical Imaging (5XSA0)

## Image restoration and color image processing

**Sveta Zinger**

( s.zinger@tue.nl )





## What is image restoration?

- **Reconstructing or recovering an image that has been degraded by using a priori knowledge of the degradation phenomenon**
  - Improving a given image in some predefined sense
  - Modeling the degradation and applying the inverse process in order to recover the original image

(The slides are based on "Digital Image Processing Using Matlab", R. C. Gonzalez, R. E. Woods, S. L. Eddins)





## What is the difference between image restoration and image enhancement?

- **Image enhancement is largely a subjective process, while image restoration is mostly an objective process**
- **Enhancement** – manipulates an image in order to take advantage of the psychophysical aspects of the human visual system;
- **Restoration** – formulates a criterion of goodness that yields an optimal estimate of the desired result





## Difference between image restoration and image enhancement: example

- **Enhancement: contrast stretching**
  - Based primarily on the pleasing aspects it might present to the viewer
- **Restoration: removal of image blur**
  - Applying a deblurring function is a restoration technique





## Model of image degradation process

- **Model of a degradation process**

$$g(x, y) = H[f(x, y)] + \eta(x, y)$$

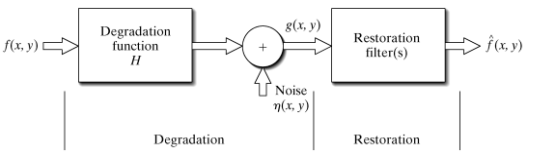
where $g(x,y)$ – degraded image, $f(x,y)$ – input image, $H$ – degradation function, $\eta(x, y)$ – additive noise

- **Objective of restoration**
  - Obtain an estimate of the original image





## Model of image degradation / restoration process



## Noise simulation – (1)

* **Noise models**
  - Behavior and effects of noise are central to image restoration
  - We assume that noise is independent from image coordinates
* **Adding noise: function `imnoise`**
  - Adds Gaussian, Poisson, speckle, "salt and pepper" noise





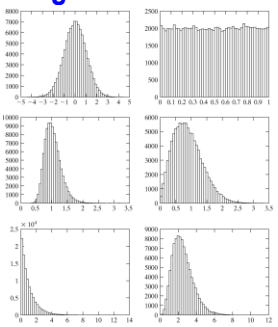
## Noise simulation – (2)

* **Matlab functions used for noise generation**
  - Uniformly distributed pseudo-random numbers – `rand`
  - Normally distributed random numbers (Gaussian distribution) – `randn`
  - Find indices of nonzero elements – `find`





## Histograms of random numbers



a b
c d
e f

**FIGURE 5.2** Histograms of random numbers: (a) Gaussian, (b) uniform, (c) lognormal, (d) Rayleigh, (e) exponential, and (f) Erlang. In each case the default parameters listed in the explanation of function `imnoise2` were used.

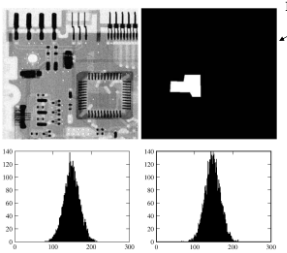




## Estimating noise parameters

`B=roipoly(f)`

`imhist(f(B))`



a b
c d

**FIGURE 5.4** (a) Noisy image. (b) ROI generated interactively. (c) Histogram of ROI. (d) Histogram of Gaussian data generated using function `imnoise2`. (Original image courtesy of Lixi, Inc.)





## Restoration in presence of noise only: spatial filtering

* **When the only degradation present is noise, then**

$$g(x,y) = f(x,y) + \eta(x,y)$$

where $g(x,y)$ – degraded image, $f(x,y)$ – input image, $\eta(x,y)$ – additive noise

* **The method of choice for reducing noise in this case – spatial filtering**



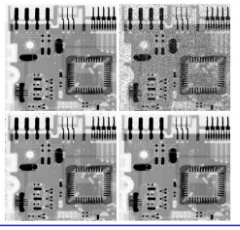


## Linear spatial filtering

* **`imfilter` – linear filtering with a user-defined mask**
  - options – boundary (symmetric, replicate, circular), output size (same or full), correlation or convolution
* **`fspecial` – create predefined 2D filters**
  - types of filters – average, gaussian, laplacian, prewitt, sobel, etc.

# Nonlinear spatial filtering

* **ordfilt2** – 2D order-statistic filter
  - g=ordfilt2(f,1,ones(m,n)) – min filter
* **medfilt2** – 2D median filter



a b
c d

**FIGURE 3.18** Median filtering. (a) X-ray image. (b) Image corrupted by salt-and-pepper noise. (c) Result of median filtering with `medfilt2` using the default settings. (d) Result of median filtering using the `'symmetric'` image extension option. Note the improvement in border behavior between (d) and (c). (Original image courtesy of Lixi, Inc.)





# Spatial filters

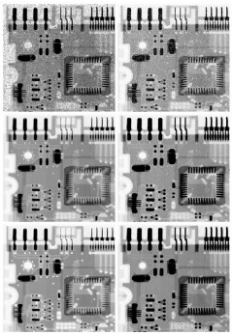
**TABLE 5.2** Spatial filters. The variables $m$ and $n$ denote respectively the number of rows and columns of the filter neighborhood.

| Filter Name | Equation | Comments |
|---|---|---|
| Arithmetic mean | $\hat{f}(x,y) = \frac{1}{mn}\sum_{(s,t)\in S_{xy}} g(s,t)$ | Implemented using IPT functions `w = fspecial('average', [m, n])` and `f = imfilter(g, w)`. |
| Geometric mean | $\hat{f}(x,y) = \left[\prod_{(s,t)\in S_{xy}} g(s,t)\right]^{\frac{1}{mn}}$ | This nonlinear filter is implemented using function `gmean` (see custom function `spfilt` in this section). |
| Harmonic mean | $\hat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}$ | This nonlinear filter is implemented using function `harmean` (see custom function `spfilt` in this section). |
| Contraharmonic mean | $\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$ | This nonlinear filter is implemented using function `charmean` (see custom function `spfilt` in this section). |
| Median | $\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\}$ | Implemented using IPT function `medfilt2`: `f = medfilt2(g, [m n])`. |
| Max | $\hat{f}(x,y) = \max_{(s,t)\in S_{xy}}\{g(s,t)\}$ | Implemented using IPT function `ordfilt2`: `f = ordfilt2(g, m*n, ones(m, n))`. |
| Min | $\hat{f}(x,y) = \min_{(s,t)\in S_{xy}}\{g(s,t)\}$ | Implemented using IPT function `ordfilt2`: `f = ordfilt2(g, 1, ones(m, n))`. |
| Midpoint | $\hat{f}(x,y) = \frac{1}{2}\left[\max_{(s,t)\in S_{xy}}\{g(s,t)\} + \min_{(s,t)\in S_{xy}}\{g(s,t)\}\right]$ | Implemented as 0.5 times the sum of the max and min filtering operations. |
| Alpha-trimmed mean | $\hat{f}(x,y) = \frac{1}{mn-d}\sum_{(s,t)\in S_{xy}} g_r(s,t)$ | The $d/2$ lowest and $d/2$ highest intensity levels of $g(s,t)$ in $S_{xy}$ are deleted, $g_r(s,t)$ denotes the remaining $mn-d$ pixels in the neighborhood. Implemented using function `alphatrim` (see custom function `spfilt` in this section). |





# Spatial filters: example



a b
c d
e f

**FIGURE 5.5**
(a) Image corrupted by pepper noise with probability 0.1. (b) Image corrupted by salt noise with the same probability. (c) Result of filtering (a) with a $3 \times 3$ contraharmonic filter of order $Q = 1.5$. (d) Result of filtering (b) with $Q = -1.5$. (e) Result of filtering (a) with a $3 \times 3$ max filter. (f) Result of filtering (b) with a $3 \times 3$ min filter.





# Adaptive median filter

*Adaptive Median Filter* : For each pixel location $(i, j)$, do

1. Initialize $w = 3$.
2. Compute $s_{i,j}^{\min,w}$, $s_{i,j}^{\text{med},w}$ and $s_{i,j}^{\max,w}$, which are the minimum, median and maximum of the pixel values in $S_{i,j}^{w}$ respectively.
3. If $s_{i,j}^{\min,w} < s_{i,j}^{\text{med},w} < s_{i,j}^{\max,w}$, then go to Step 5. Otherwise, set $w = w + 2$.
4. If $w \leq w_{\max}$ go to Step 2. Otherwise, we replace $y_{i,j}$ by $s_{i,j}^{\text{med},w_{\max}}$.
5. If $s_{i,j}^{\min,w} < y_{i,j} < s_{i,j}^{\max,w}$, then $y_{i,j}$ is not a noise candidate, else we replace $y_{i,j}$ by $s_{i,j}^{\text{med},w}$.





# Adaptive median filter: example

a b c

**FIGURE 5.6** (a) Image corrupted by salt-and-pepper noise with density 0.25. (b) Result obtained using a median filter of size $7 \times 7$. (c) Result obtained using adaptive median filtering with $S_{\max} = 7$.





# RGB images – (1)

* **RGB color image**
  - *M* x *N* x 3 array of color pixels, where each color pixel is a triplet corresponding to the red, green and blue components of an RGB image at a specific spatial location
  - It is a "stack" of three gray-scale images that, when fed into the red, green and blue inputs of a color monitor, produce a color image on the screen

## RGB images – (2)

* **Component images**
  - three images forming an RGB color image
* **Bit depth**
  - number of bits used to represent the pixel values of the component images
  - For example, if each component image is an 8-bit image, the corresponding RGB image is said to be 24 bits deep





## RGB images – (3)

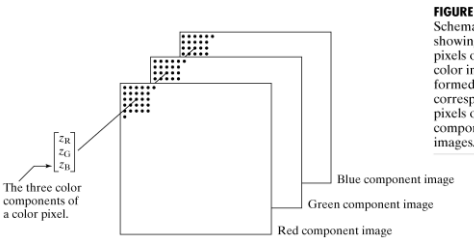



**FIGURE 6.1** Schematic showing how pixels of an RGB color image are formed from the corresponding pixels of the three component images.





## Indexed images – (1)

* **Data matrix of integers**
* **Colormap matrix, map**
  - *m* x 3 array, where *m* – number of colors the map defines
  - each row of map specifies the red, green and blue components of a single color
  - Color of each pixel is determined by using the corresponding value of integer data matrix as a pointer into map





## Indexed images – (2)

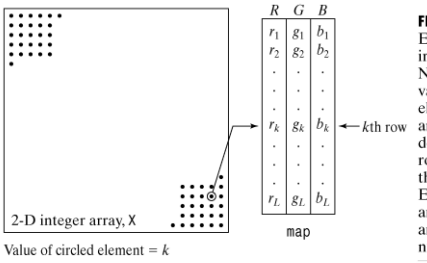



**FIGURE 6.3** Elements of an indexed image. Note that the value of an element of integer array X determines the row number in the colormap. Each row contains an RGB triplet, and *L* is the total number of rows.





## Indexed images – (3)

* **Some of Matlab predefined colormaps**
  - autumn, cool, gray, hot, pink, spring, summer, winter, etc.
* **Some Image Processing toolbox functions for converting between RGB, indexed and gray-scale intensity images**
  - `gray2ind`, `ind2gray`, `rgb2gray`, `ind2rgb`, etc.
* **Convertion to other color spaces**
  - `rgb2ycbcr`, `ycbcr2rgb`, `rgb2hsv`, `hsv2rgb`, etc.



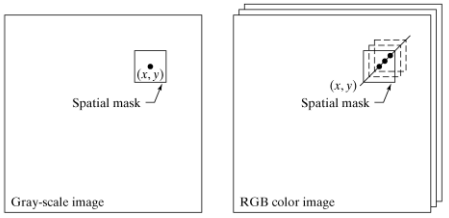


## Basics of color image processing – (1)

* **Color pixels are vectors**
  - For example, in the RGB system, each color point can be interpreted as a vector extending from the origin to that point in the RGB coordinate system:

$$c(x,y) = \begin{bmatrix} c_R(x,y) \\ c_G(x,y) \\ c_B(x,y) \end{bmatrix} = \begin{bmatrix} R(x,y) \\ G(x,y) \\ B(x,y) \end{bmatrix}$$

## Basics of color image processing – (2)




a b

**FIGURE 6.10** Spatial masks for gray-scale and RGB color images.





## Spatial filtering of color images: color image smoothing

* Instead of single pixels we now deal with vector values
* Average of *K* RGB vectors in neighborhood $S_{xy}$

$$\bar{c}(x, y) = \frac{1}{K} \sum_{(s,t) \in S_{xy}} c(s, t); \qquad \bar{c}(x, y) = \begin{bmatrix} \frac{1}{K} \sum_{(s,t) \in S_{xy}} R(s, t) \\ \frac{1}{K} \sum_{(s,t) \in S_{xy}} G(s, t) \\ \frac{1}{K} \sum_{(s,t) \in S_{xy}} B(s, t) \end{bmatrix}$$





## Linear spatial filtering of color image

* **Steps for smoothing an RGB image f**
  - Extract the three component images :

```
fR=f(:,:,1); fG=f(:,:,2); fB=f(:,:,3);
```

  - Filter each component image individually:

```
fR_smooth=imfilter(fR,w);
```

similarly find fG, fB

  - Reconstruct the filtered RGB image:

```
f_filtered=cat(3, fR_smooth, fG_smooth, fB_smooth);
```

* **Or use Matlab function**

```
F_filtered=imfilter(f,w)
```





## RGB image: example

a b
c d

**FIGURE 6.19** (a) RGB image; (b) through (d) are the red, green and blue component images, respectively.





## HSI color model: example

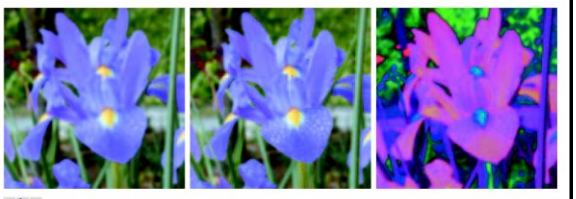
a b c

**FIGURE 6.20** From left to right: hue, saturation, and intensity components of Fig. 6.19(a).





## Smoothing result



a b c

**FIGURE 6.21** (a) Smoothed RGB image obtained by smoothing the *R*, *G*, and *B* image planes separately. (b) Result of smoothing only the intensity component of the HSI equivalent image. (c) Result of smoothing all three HSI components equally.

# Reference


– Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, "Digital Image Processing Using Matlab", Pearson Education, 2004
  – Chapter 5
  – Chapter 6