(5) Image Restoration

- Image restoration
  → Recover an image that has been degraded using a prior model of the degradation process
  • Restoration: model the degradation and apply an inverse process to recover the original image
  • Objective process

- Image enhancement
  → Emphasize features of an image making it more visually pleasing
  • Enhancement: enhancement techniques (e.g., contrast stretching) are used without a prior model of the process that created the image
  • Subjective process

(a) Degradation/restoration process model

- Degradation model
A degradation function and additive noise that operate on an input image \( f(x, y) \) to produce a degraded image \( g(x, y) \):

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

- Restoration model
  - Given \( g(x, y) \) and some knowledge about the degradation function \( H \) and the noise \( \eta \), obtain an estimate \( \hat{f}(x, y) \) of the original image.
  - If \( H \) is a linear spatially invariant process:
    - In spatial domain:
\[ g(x, y) = h(x, y) \ast f(x, y) + \eta(x, y) \]

* \( h(x, y) \): spatial representation of the degradation function

→ **Point spread function** (PSF): let \( h(x, y) \) operate on a point of light to obtain the characteristics of degradation

* Convolution:

\[
h(x, y) \ast f(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x-m, y-n)
\]

• In frequency (Fourier) domain:

\[ G(u, v) = H(u, v)F(u, v) + N(u, v) \]

* \( G, H, F, \) and \( N \) are the Fourier transforms of \( g, h, f, \) and \( \eta, \) respectively

* \( H(u, v) \): **optical transfer function** (OTF)

• \( h(x, y) \) may be estimated by experimentation

→ If the equipment used to acquire the degraded image is available, it is possible to obtain an accurate estimate of the degradation

* E.g., obtain the impulse response of the degradation by imaging an impulse (small dot of light)

→ Fourier transform of an impulse is a constant \((A)\), hence
\[ H(u, v) = \frac{G(u, v)}{A} \]

- Hence,
  * The degradation process is sometimes referred to as “convolving the image with a PSF or OTF”
  * The restoration process is sometimes referred to as \textit{deconvolution}
    \rightarrow \textit{Inverse filtering}

\[
\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)} \quad \text{or} \quad \hat{F}(u, v) = F(u, v) + \frac{N(u, v)}{H(u, v)}
\]

**(b) Noise model**

- Sources of noise
  \rightarrow Arise during image acquisition (digitization) and/or transmission
  * Environmental conditions: light, temperature, humidity, atmospheric disturbance...
  * Quality of sensing elements and transmission media
• Human interference

- Assumptions of noise
  • Independent of spatial coordinates
  • Uncorrelated with respect to the image

(c) Noise probability density functions

- Spatial noise
  • Considered as random variables, characterized by a probability density function (PDF)

- Noise models
  → Simulate the behavior and effect of noise
  • Gaussian (normal) noise, PDF:

    \[ p(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \]

  * \( z \): gray level, \( \mu \): mean value, \( \sigma \): standard deviation, \( \sigma^2 \): variance
• Rayleigh noise

\[ p(z) = \begin{cases} \frac{2}{b} (z - a) e^{-(z-a)^2/b} & \text{for } z \geq a \\ 0 & \text{for } z < a. \end{cases} \]

* Mean: \( \mu = a + (\pi b/4)^{1/2} \), variance: \( \sigma^2 = b(4-\pi)/4 \)

• Erlang (Gamma) noise

\[ p(z) = \begin{cases} \frac{a^b z^{b-1} e^{-az}}{(b-1)!} & \text{for } z \geq 0 \\ 0 & \text{for } z < 0. \end{cases} \]

* \( a > 0, b: \) positive integer, mean: \( \mu = b/a \), variance: \( \sigma^2 = b/a^2 \)

• Exponential noise

\[ p(z) = \begin{cases} ae^{-az} & \text{for } z \geq 0 \\ 0 & \text{for } z < 0. \end{cases} \]

* \( a > 0, \) mean: \( \mu = 1/a \), variance: \( \sigma^2 = 1/a^2 \)

• Uniform noise
\[
p(z) = \begin{cases} 
\frac{1}{b-a} & \text{if } a \leq z \leq b \\
0 & \text{otherwise}
\end{cases}
\]

* Mean: \( \mu = (a+b)/2 \), variance: \( \sigma^2 = (b-a)^2/12 \)

- Impulse (salt-and-pepper) noise

\[
p(z) = \begin{cases} 
P_a & \text{for } z = a \\
P_b & \text{for } z = b \\
0 & \text{otherwise}
\end{cases}
\]

* Either \( P_a \) or \( P_b \) is zero: unipolar noise
* If \( P_a \approx P_b \): bipolar (salt-and-pepper) noise

MATLAB \texttt{rand}, \texttt{randn} functions

- E.g.: \( N = \texttt{rand}(m); N = \texttt{rand}(m, n); \) \( \Rightarrow m \times m, m \times n \) uniformly distributed random numbers with range \([0, 1]\)
- \( N = \texttt{randn}(m); N = \texttt{randn}(m, n); \) \( \Rightarrow \) Normally distributed random number with mean=0 and variance=1 (Gaussian noise): \( \sim N(0, 1) \)
- MATLAB `imnoise` function: add noise to the image
  - E.g.: `g = imnoise(f, 'gaussian', 0, 1) \Rightarrow \sim N(0, 1)`
  - Noise type: 'gaussian', 'localvar', 'poisson', 'salt & pepper', 'speckle'

(d) Restoration in the presence of noise only—spatial filtering

- Degradation model:
  \[ g(x, y) = f(x, y) + \eta(x, y) \]

- Spatial noise filters:
  - Mean filters
    - Arithmetic mean filter: reduce noise as a result of blurring
      \[
      \hat{f}(x, y) = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s,t)
      \]
      \# $S_{xy}$: the $m \times n$ area for computing $\hat{f}(x, y)$
# MATLAB: \( f = \text{imfilter}(g, \text{fspecial('average', [m n]))} \) \quad \rightarrow \text{Padded with 0}

* Geometric mean filter: reduce noise as a result of blurring

\[
\hat{f}(x, y) = \left[ \prod_{(s,t) \in S_{xy}} g(s,t) \right]^{\frac{1}{mn}}
\]

# MATLAB: \( f = \exp(\text{imfilter}(\log(g), \text{ones}(m, n), 'replicate')) .^ (1/(m*n)) \)

\( \rightarrow \log(a*b*c\ldots) = \log(a) + \log(b) + \log(c) + \ldots \)

* Harmonic mean filter: for pepper and Gaussian noise

\[
\hat{f}(x, y) = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s,t)}}
\]

# MATLAB: \( f = m*n ./ \text{imfilter}(1 ./ (g + \text{eps}), \text{ones}(m, n), 'replicate') \)

• Order-statistics filters

* Median filter: for bipolar or unipolar impulse noise

\[
\hat{f}(x, y) = \text{median}\{g(s,t)\}
\]
# MATLAB: \( f = \text{medfilt2}(g, [m, n]) \)

* Max and min filters: for salt or pepper noise

\[
\hat{f}(x, y) = \max_{(s,t)\in S_{xy}} \{g(s,t)\}, \quad \hat{f}(x, y) = \min_{(s,t)\in S_{xy}} \{g(s,t)\}
\]

# MATLAB: \( f = \text{ordfilt2}(g, m\times n, \text{ones}(m, n)), \quad f = \text{ordfilt2}(g, 1, \text{ones}(m, n)) \)

* Midpoint filter: combine order statistics and averaging filtering; work well for randomly distributed noise

\[
\hat{f}(x, y) = \frac{1}{2} \left[ \max_{(s,t)\in S_{xy}} \{g(s,t)\} + \min_{(s,t)\in S_{xy}} \{g(s,t)\} \right]
\]

# MATLAB: \( f = 0.5 \times (\text{ordfilt2}(g, m\times n, \text{ones}(m, n)) + \text{ordfilt2}(g, 1, \text{ones}(m, n))) \)

* Alpha-trimmed mean filter

\[
\hat{f}(x, y) = \frac{1}{mn-d} \sum_{(s,t)\in S_{xy}} g_r(s,t)
\]

# Remove \( d/2 \) lowest and highest gray-level values of \( g(s, t) \)

# \( g_r(s, t) \): the remaining \( mn-d \) pixels

# For multiple types of noise, e.g., combination of salt-and-pepper and Gaussian noise
# MATLAB:

```matlab
f = imfilter(g, ones(m, n), 'replicate');  % Sum of the pixels in S_{xy}
for k = 1:d/2
    f = f - ordfilt2(g, k, ones(m, n), 'replicate');  % Subtract first \( d/2 \) pixels
end
for k = (m*n - d/2 + 1):m*n
    f = f - ordfilt2(g, k, ones(m, n), 'replicate');  % Subtract last \( d/2 \) pixels
end
f = f/(m*n - d);
```

- Adaptive filters:
  
  Filters capable of adapting their behavior depending on the characteristics of the image area being filtered
  
  - E.g., adaptive, local noise reduction filter

\[
\hat{f}(x, y) = g(x, y) - \frac{\sigma_n^2}{\sigma_L^2} \left[ g(x, y) - m_L \right]
\]

where \( \sigma_n^2 \) (a scalar) is estimated variance of the noise corrupting \( f(x, y); m_L \)
and $\sigma_L^2$ are the local mean and variance of pixels in $S_{xy}$, respectively

* Behavior:
1. If $\sigma_\eta^2 = 0$, return $g(x, y)$
2. If $\sigma_L^2$ is high relative to $\sigma_\eta^2$ (i.e., edges), return a value close to $g(x, y)$
3. If $\sigma_\eta^2 = \sigma_L^2$ (i.e., local area has same property as the overall image), return local arithmetic mean

* MATLAB \texttt{nlfilter} (nonlinear filtering) function:

\[
\text{mean} = \text{imfilter}(g, \text{ones}(m, n), \text{'replicate'})/(m*n); \quad \rightarrow \text{mean}: \text{local mean}
\]

\[
\text{var} = \text{nlfilter}(g, [m \ n], \text{'std2(x).^2'}); \quad \rightarrow \text{var}: \text{local variance}
\]

\[
f = g - (\text{sn} ./ (\text{var}+\text{eps})) .* (g - \text{mean}); \quad \rightarrow \text{sn}: \text{variance of noise}
\]

# Compute var using an inline function:

\[
\text{fun1} = \text{inline}('\text{std2(x).^2'});
\]

\[
\text{var} = \text{nlfilter}(g, [m \ n], \text{fun1});
\]

# Compute var using a function handle @:

\[
\text{var} = \text{nlfilter}(g, [m \ n], @\text{fun2});
\]

\[
\rightarrow \text{M-file } \text{fun2.m}:
\]

\[
\text{function v = fun2(x)}
\]
\[ v = \text{std2}(x)^2; \]

# Compute var using colfilt (column filtering: much faster):

\[
\text{fun3} = \text{inline}('\text{std}(x).^2')
\]

\[
\text{var} = \text{colfilt}(g, [m \ n], 'sliding', \text{fun3});
\]

• E.g., adaptive median filter

* Denote:

\[
\begin{align*}
    z_{\text{min}} &= \text{minimum intensity value in } S_{xy} \\
    z_{\text{max}} &= \text{maximum intensity value in } S_{xy} \\
    z_{\text{med}} &= \text{median of the intensity values in } S_{xy} \\
    z_{xy} &= \text{intensity value at coordinates } (x, y)
\end{align*}
\]

* Adaptive median filter works at two levels, A and B \((S_{\text{max}}: \text{max allowed window size})\)

Level A: If \(z_{\text{min}} < z_{\text{med}} < z_{\text{max}}\), go to level B

Else increase the window size

If window size \(\leq S_{\text{max}}\), repeat level A

Else output \(z_{\text{med}}\)

Level B: If \(z_{\text{min}} < z_{xy} < z_{\text{max}}\), output \(z_{xy}\) (\(\Rightarrow\) do not filter)

Else output \(z_{\text{med}}\) (\(\Rightarrow\) filter by replacing the pixel with \(z_{\text{med}}\))
* Advantage over **medfilt2**: less blurring and distortion

* MATLAB: supplement *adpmedian.m*

(e) **Motion blur modeling and restoration**

- Motion blur modelling:
  - Approximate the linear motion of the camera
  
  • MATLAB:

```matlab
f = checkerboard(8);  % Create “checkerboard” image
PSF = fspecial('motion', 7, 45);  % Move 7 pixels along the 45 degrees line
```

![Checkerboard](image1)
![Motion blur](image2)
![Motion blur with noise](image3)
\begin{verbatim}
gb = imfilter(f, PSF, 'circular');  \rightarrow \text{Degradation ('circular' reduces border effect)}
noise = imnoise(zeros(size(f)), 'gaussian', 0, 0.01);  \rightarrow \text{Gaussian noise}
g = gb + noise;  \rightarrow \text{Add noise}
\end{verbatim}

- Restoration: \textit{Wiener} filter
  
  \begin{itemize}
  \item Degradation model:
    \[ g(x, y) = h(x, y) * f(x, y) + \eta(x, y) \]
  \item Wiener filter: a statistical approach to seek an estimate \( \hat{f} \) that
    minimizes the statistical function (mean square error):
    \[ e^2 = E\{(f - \hat{f})^2\} \]
  \end{itemize}

  * Assumptions:
    # Image and noise are uncorrelated
    # Image and/or noise has zero mean
    # Gray levels in the estimate are linear function of the levels in the
      degraded image
In frequency domain:

\[ \hat{F}(u, v) = \frac{H^*(u, v)S_f(u, v)}{S_f(u, v)|H(u, v)|^2 + S_\eta(u, v)} G(u, v) \]

\[ = \left[ \frac{H^*(u, v)}{|H(u, v)|^2 + S_\eta(u, v)/S_f(u, v)} \right] G(u, v) \]

\[ = \left[ \frac{1}{H(u, v)} \cdot \frac{|H(u, v)|^2}{|H(u, v)|^2 + S_\eta(u, v)/S_f(u, v)} \right] G(u, v) \]

* \( H(u, v) \): degradation function  
* \( |H(u, v)|^2 = H^*(u, v)H(u, v) \)  \( \Rightarrow \) \( H^*(u, v) \): complex conjugate  
* \( S_\eta(u, v) = |N(u, v)|^2 \): the power spectrum of the noise
* $S_f(u, v) = |F(u, v)|^2$: the power spectrum of the undegraded image

* $S_\eta(u, v)/S_f(u, v)$: noise-to-signal power ratio

  $\rightarrow$ If $S_\eta(u, v) = 0$: inverse filter

• Approximation of the noise-to-signal power ratio

  $S_\eta(u, v)/S_f(u, v) \cong R (= \eta_A/f_A$: constant)

  $\Rightarrow \eta_A = \frac{1}{MN} \sum_u \sum_v S_\eta(u, v), \quad f_A = \frac{1}{MN} \sum_u \sum_v S_f(u, v)$

* Hence, $\hat{F}(u, v) = \left| \frac{1}{H(u, v)} \frac{|H(u, v)|^2}{|H(u, v)|^2 + R} \right| G(u, v)$

• MATLAB

  * Wiener filter:
    
    
    ```matlab
    PSF = fspecial('motion', 7, 45);
    fr1 = deconvwnr(g, PSF);
    ```

  * Parameterized Wiener filter (1):
    
    ```matlab
    Sn = abs(fft2(noise)) .^2;
    ```
nA = sum(Sn(:))/prod(size(noise));
Sf = abs(fft2(f)).^2;
fA = sum(Sf(:))/prod(size(f));
R = nA/fA;
fr2 = deconvwnr(g, PSF, R);

* Parameterized Wiener filter (2):
NCORR = fftshift(real(ifft2(Sn)));
ICORR = fftshift(real(ifft2(Sf)));
fr3 = deconvwnr(g, PSF, NCORR, ICORR);

# NCORR: autocorrelation function of noise
# ICORR: autocorrelation function of image

(f) Geometric transformations
- Geometric spatial transformation
→ An image \( f \), defined over a \((w, z)\) coordinate system, undergoes geometric distortion to produce image \( g \), defined over a \((x, y)\) coordinate system: \((x, y) = T\{(w, z)\}\)

- Affine transform: scaling, rotation, translation, and shearing

→ Note: in MATLAB the transformation matrix is on the right

\[
\begin{bmatrix}
  x \\ y \\ 1
\end{bmatrix} = \begin{bmatrix}
w \\ z \\ 1
\end{bmatrix} T = \begin{bmatrix}
w \\ z \\ 1
\end{bmatrix} \begin{bmatrix}
t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1
\end{bmatrix}
\]

• Identity:
  \[
  x = w \\
  y = z
  \]
  \[
  I = \begin{bmatrix}
  1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1
  \end{bmatrix}
  \]

• Rotation (clockwise):
  \[
  x = w\cos \theta - z\sin \theta \\
  y = w\sin \theta + z\cos \theta
  \]
  \[
  R = \begin{bmatrix}
  \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1
  \end{bmatrix}
  \]
• Scaling: \[ S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
  \[ x = S_x w \]
  \[ y = S_y z \]

• Horizontal shearing (x direction): \[ S_x = \begin{bmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
  \[ x = w + \alpha z \]
  \[ y = z \]

• Vertical shearing (y direction): \[ S_y = \begin{bmatrix} 1 & \beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
  \[ x = w \]
  \[ y = \beta w + z \]

• Translation: \[ T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix} \]
  \[ x = w + \delta x \]
  \[ y = z + \delta y \]

- Graylevel interpolation
  • Transformation: scan each pixel of the output image, and compute the corresponding location in the input image using \( T^{-1}\{(x, y)\} \)
• Geometric transformation results in noninteger coordinates: need to infer the graylevel

\[ g(x, y) = f(w_1, z_0) \]

• Nearest neighbor: \( g(x, y) = f(w_1, z_0) \)

• Bilinear interpolation
  * First, linear interpolation in the \( w \) (or \( x \)) direction

\[ p = \frac{w_1 - w}{w_1 - w_0} f(w_0, z_0) + \frac{w - w_0}{w_1 - w_0} f(w_1, z_0) \]

\[ = (w_1 - w) f(w_0, z_0) + (w - w_0) f(w_1, z_0) \]
\[ q = \frac{w_1 - w}{w_1 - w_0} f(w_0, z_1) + \frac{w - w_0}{w_1 - w_0} f(w_1, z_1) \]
\[ = (w_1 - w) f(w_0, z_1) + (w - w_0) f(w_1, z_1) \]

* And then, linear interpolation in the \( z \) (or \( y \)) direction

\[ g = \frac{z_1 - z}{z_1 - z_0} p + \frac{z - z_0}{z_1 - z_0} q = (z_1 - z) p + (z - z_0) q \]

- Other interpolation methods: bicubic, bicubic smoother, stairstep …

- MATLAB point transform
  - Create spatial transformation structure: \( \text{tform} = \text{maketform}(\text{type}, \quad \text{Transform_matrix}) \)
  - Type: 'affine', 'projective', 'custom', 'box', 'composit'
  - E.g.: \( WZ \) and \( XY \) are \( P \times 2 \) matrices of points, each row: \( (w, z) \) and \( (x, y) \)
    \( WZ = [1 \ 1; 2 \ 3; 5 \ 7]; \)
    \( \text{tform} = \text{maketform}('\text{affine}', [2 \ 0 \ 0; 0 \ 3 \ 0; 0 \ 0 \ 1]); \quad \rightarrow \text{Scaling matrix} \)
    \( XY = \text{tformfwd}(WZ, \text{tform}); \quad \rightarrow \text{Forward transform: } XY = [2 \ 3; 4 \ 9; 10 \ 21] \)
    \( WZ = \text{tforminv}(XY, \text{tform}); \quad \rightarrow \text{Inverse transform: } WZ = [1 \ 1; 2 \ 3; 5 \ 7] \)
- MATLAB image transform
  \[ x = \pi/6; \quad \% \text{Rotate 30 degrees} \]
  \[ R = [\cos(x) \sin(x) \ 0; -\sin(x) \cos(x) \ 0; \ 0 \ 0 \ 1]; \quad \% \text{Rotation matrix} \]
  \[ \text{tform} = \text{maketform('affine', R);} \]
  \[ [g \ xdata \ ydata] = \text{imtransform}(f, \text{tform}); \]
  \[ \text{figure, imshow(uint8(g), [ ]}, \ 'XData', xdata, 'YData', ydata), \text{title('Image g');} \]
  \[ \text{impixelinfo, axis on, axis([xdata ydata])}; \]
- Supplement: *transform.m*

**(g) Image registration**

- Image registration
  \[ \rightarrow \text{Seek to align two images of the same scene} \]
  - Images are taken at different times, places, or using different acquisition instruments
  - Applications: align images acquired by different equipments, restoration of a transformed image, panorama (360° view: an
unobstructed view extending in all directions, especially of a landscape

- MATLAB: `cp2tform`
  ➔ Infer geometric transformation from control point pairs
  • Select two sets of corresponding points in *input* and *base* images
  • Establish the geometric transform between them
    
    ```matlab
    base_points = [83 81; 450 56; 43 293; 249 392; 436 442];
    input_points = [68 66; 375 47; 42 286; 275 434; 523 532];
    tform = cp2tform(input_points, base_points, 'affine');
    register = imtransform(input, tform);
    ```
  
  • GUI to select control points: `cpselect(input, base)`
    * ⊕: add control points in both images
    * ⊕⊕: add a control point in one image and predict matches in another
    * Save control points: File ➔ Save Points to Workspace

- Supplement: *panorama.ppt*