Digital Image Processing

Mathematical tools used in DIP

By
Dr. K. M. Bhurchandi

Array versus Matrix Operations

- Consider two 2 x 2 images

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ and } \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Array Product is:

$$\begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

- Matrix Product is:

$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

- *Array* operation involving one or more images is carried out on a *pixel-by-pixel* basis.

- Ex. i) Raising an image to a power.

  Individual pixel is raised to that power.

  ii) Dividing an image by another:

  Division is between corresponding pixel pairs.

Linear versus Nonlinear Operations

- General operator, *H*, that produces an output image, *g*(*x*, *y*), for a given input image, *f* (*x*, *y*):

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

- *H* is said to be a *linear operator* if

  $$H[a_i f_i(x,y) + a_j f_j(x,y)] = a_i H[f_i(x,y)] + a_j H[f_j(x,y)]$$
  $$= a_i g_i(x,y) + a_j g_j(x,y) \text{—Eq.(i)}$$

  where, a_i, a_j – arbitrary constants

  $f_i(x,y)$, $f_j(x,y)$ – images of same size.

- Suppose H is the sum operator, $\sum$

- $\sum[aifi(x,y) + ajfj(x,y)] = \sum a_i f_i(x,y) + \sum a_j f_j(x,y)$

$$= a_i \sum f_i(x,y) + aj \sum f_j(x,y)$$

$$= a_i g_i(x,y) + ajgj(x,y)$$

Thus, $\sum$ operator is linear.

- Consider ***max*** operation,
- Let f1 = $\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}$, f2 = $\begin{bmatrix} 6 & 5 \\ 4 & 7 \end{bmatrix}$, a1 = 1, a2 = -1.

- To Test Linearity,
- LHS of eq(i): $max\left\{(1)\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix} + (-1)\begin{bmatrix} 6 & 5 \\ 4 & 7 \end{bmatrix}\right\} = -2$
- RHS of eq(i): $(1)max\left\{\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}\right\} + (-1)max\left\{\begin{bmatrix} 6 & 5 \\ 4 & 7 \end{bmatrix}\right\}$ = -4
- LHS ≠ RHS
- So, ***max*** is non-linear operation.

Arithmetic Operations

- Arithmetic operations are array operations that are carried out between corresponding pixel pairs.

- Four arithmetic operations:

$$s(x, y) = f(x, y) + g(x, y)$$
$$d(x, y) = f(x, y) - g(x, y)$$
$$p(x, y) = f(x, y) * g(x, y)$$
$$v(x, y) = f(x, y) / g(x, y)$$

Where, $x = 0,1,2,\ldots,M-1$, $y = 0,1,2,\ldots.N-1$.

All images are of size M (rows) x N (columns).

Set and Logical Operations

- Basic Set operation
- Let *A* - set composed of *ordered pairs* of real numbers.
- If pixel a = (x,y), is an element of A

$$a \in A$$

- If a is not an element of A

$$a \notin A$$

- Set with no elements is called the *null* or *empty set*

$$\varnothing.$$

- If every element of a set *A* is also an element of a set *B*, then *A* is said to be a *subset* of *B*

$$A \subseteq B$$

- *Union* of two sets *A* and *B*

$$C = A \cup B$$

- *Intersection* of two sets *A* and *B*

$$D = A \cap B$$

- Two sets *A* and *B* are *disjoint* or *mutually exclusive* if they have no common elements

$$A \cap B = \varnothing$$

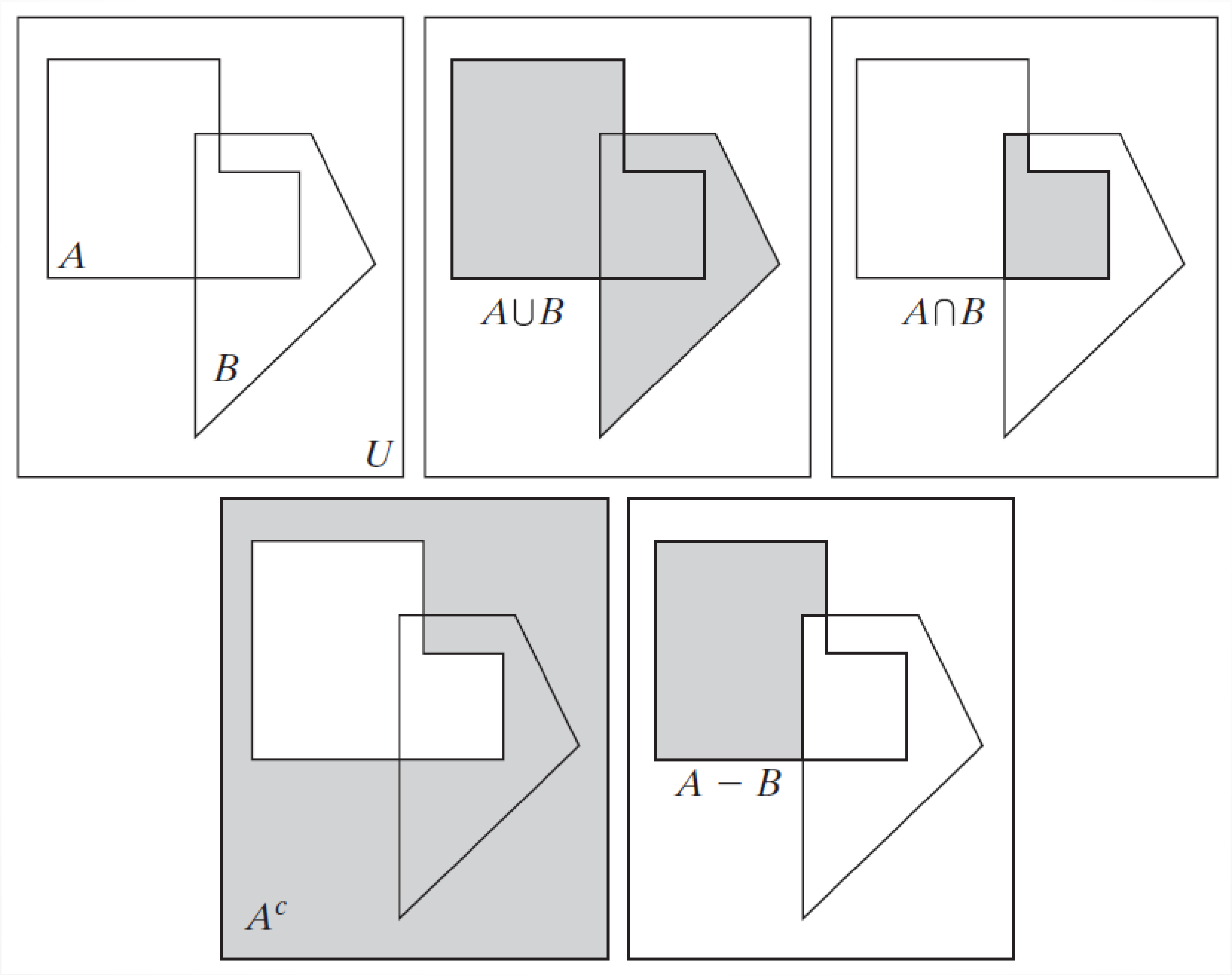
- The *set universe, U,* is the set of all elements in a given application.
- *complement* of a set *A* is the set of elements that are not in *A*

$$A^c = \{w | w \notin A\}$$

- *difference* of two sets *A* and *B*,

$$A - B = \{w | w \in A, w \notin B\} = A \cap B^c$$

Illustration of Set Concept



Set operations on gray-scale images

- a) Original image b) Image Negative c) Union of (a) & constant image

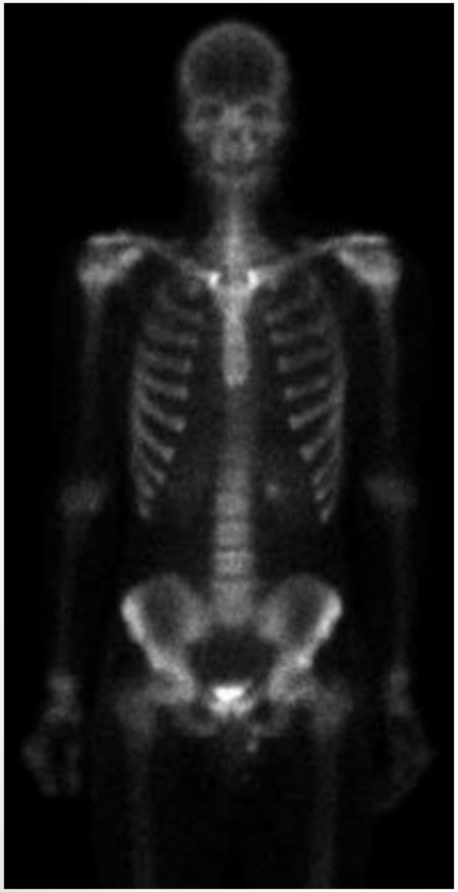
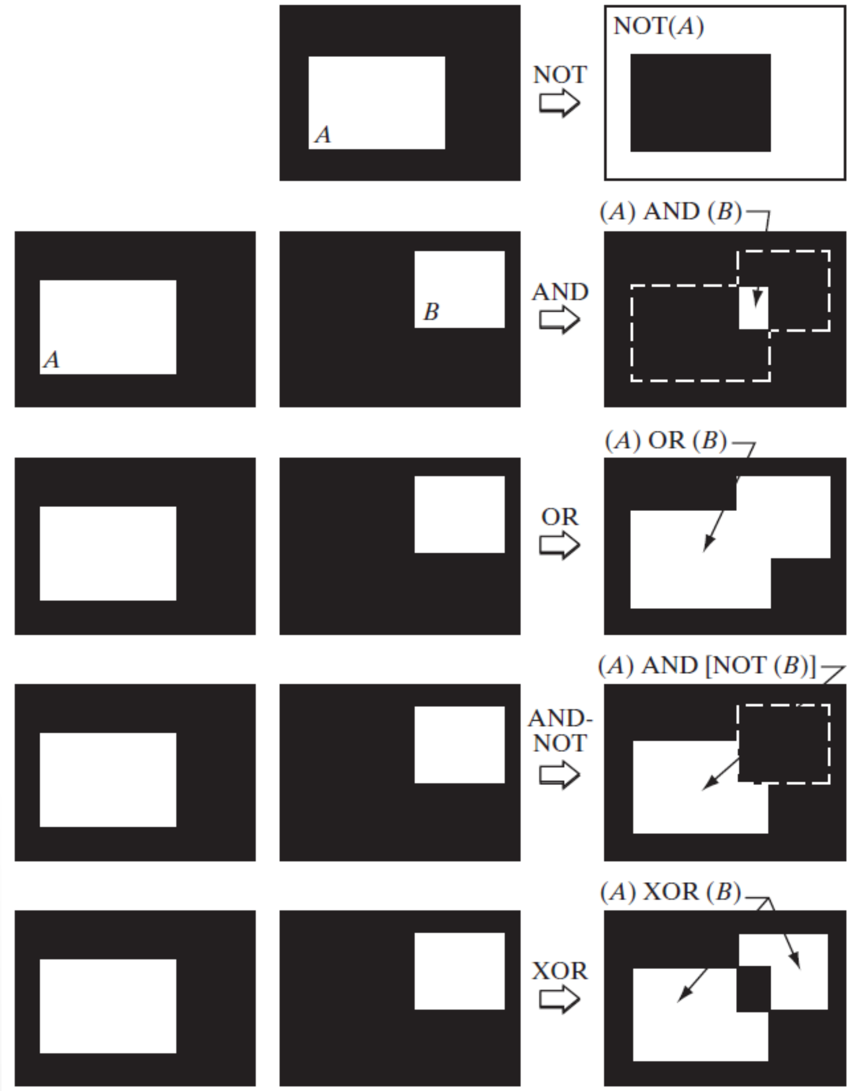


Illustration of Logical Operators



Spatial Operations

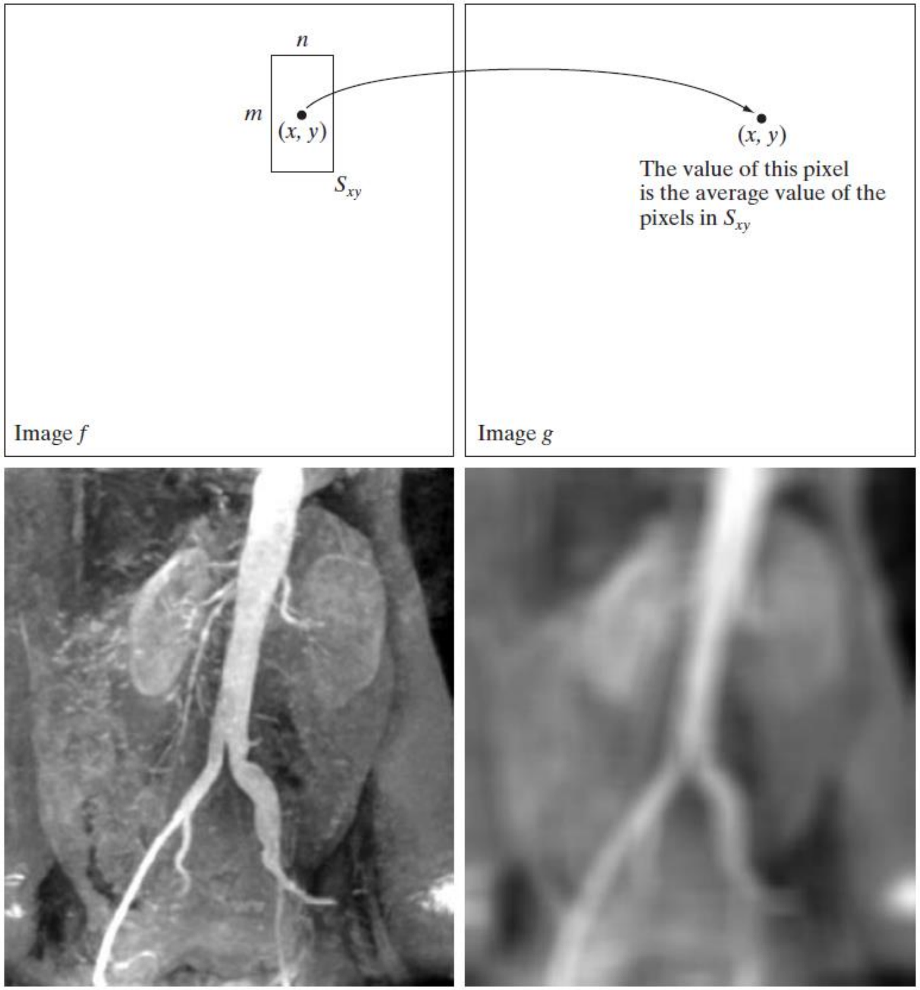
- Spatial operations are performed directly on the pixels of a given image.
    - (1) single-pixel operations,
    - (2) neighborhood operations, &
    - (3) geometric spatial transformations.

- **Single-pixel operations**

$$s = T(z)$$

- *z* - intensity of a pixel in the original image
- *s* - (mapped) intensity of the corresponding pixel in the processed image.

- **Neighborhood operations**



- We can express the operation in equation form as

$$g(x, y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} f(r, c)$$

- where *r* and *c* are the row and column coordinates of the pixels whose coordinates are members of the set S_{xy}.

- **Geometric spatial transformations**
- They modify the spatial relationship between pixels in an image.
- a.k.a. *rubber-sheet* transformations.
- They consists of two basic operations:
  - (1) spatial transformation of coordinates and
  - (2) intensity interpolation that assigns intensity values to the spatially transformed pixels.
- The transformation of coordinates may be expressed as

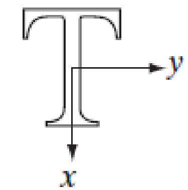
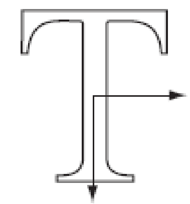
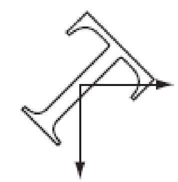
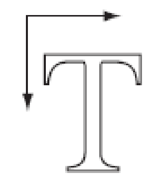
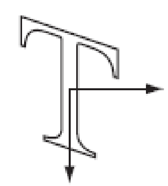
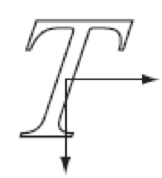
$$(x, y) = T\{(v, w)\}$$

  - (v, w) - pixel coordinates in the original input image
  - (x, y) - the corresponding pixel coordinates in the transformed output image.

- One of the most commonly used spatial coordinate transformations is the ***affine transform***
- Its General Form

$$[x \;\; y \;\; 1] = [v \;\; w \;\; 1]\,\mathbf{T} = [v \;\; w \;\; 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

- This transformation can scale, rotate, translate, or sheer a set of coordinate points, depending on the value chosen for the elements of matrix **T**.

Transformation Name	Affine Matrix, T	Coordinate Equations	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = w$	
Scaling	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = c_x v$ $y = c_y w$	
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v\cos\theta - w\sin\theta$ $y = v\cos\theta + w\sin\theta$	
Translation	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
Shear (vertical)	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v + s_v w$ $y = w$	
Shear (horizontal)	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = s_h v + w$	

- ***forward mapping***, consists of scanning the pixels of the input image and, at each location, (v, w), computing the spatial location, (x, y), of the corresponding pixel in the output image.

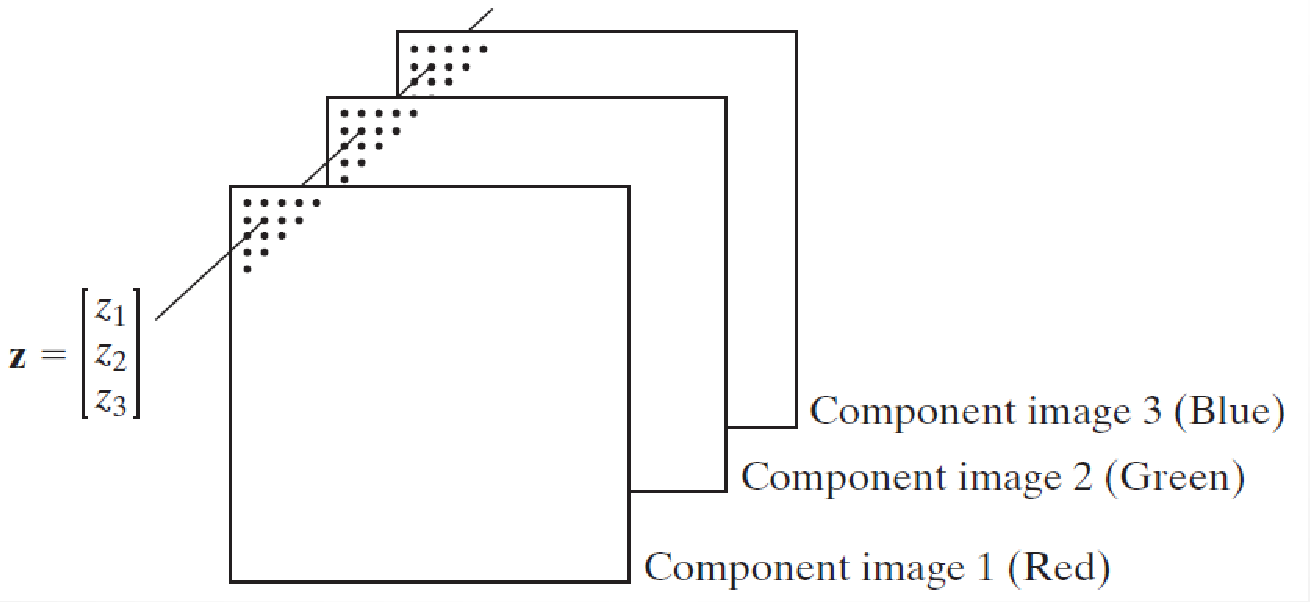
- ***inverse mapping***, scans the output pixel locations and, at each location, (x, *y*), computes the corresponding location in the input image using

$$(v, w, 1) = T^{-1}(x, y, 1)$$

- It then interpolates using either of nearest neighbor, bilinear, and bicubic interpolation techniques.

Vector and Matrix Operations

- Color images are formed in RGB color space by using red, green, and blue component images.
- *Each* pixel of an RGB image has 3 components, which can be organized in the form of a *column vector.*



- *Euclidean distance*, *D*, between a pixel vector **z** and an arbitrary point **a** in *n*-dimensional space is defined as the vector product

$$D(\mathbf{z}, \mathbf{a}) = \left[(\mathbf{z} - \mathbf{a})^T(\mathbf{z} - \mathbf{a})\right]^{\frac{1}{2}}$$

$$= \left[(z_1 - a_1)^2 + (z_2 - a_2)^2 + \cdots + (z_n - a_n)^2\right]^{\frac{1}{2}}$$

- This is a generalization of the 2-D Euclidean distance
- Sometimes is referred to as a *vector norm, denoted by* $||z - a||$.

- An image of size M X N can be represented as a vector of dimension MN X 1.
- A broad range of linear processes can be applied to such images by using notation

$$g = Hf + n$$

  - f – MN X 1 vector representing Input image
  - n - MN X 1 vector representing M X N noise pattern
  - g - MN X 1 vector representing affected image
  - H – MN X MN matrix representing linear process applied to input image

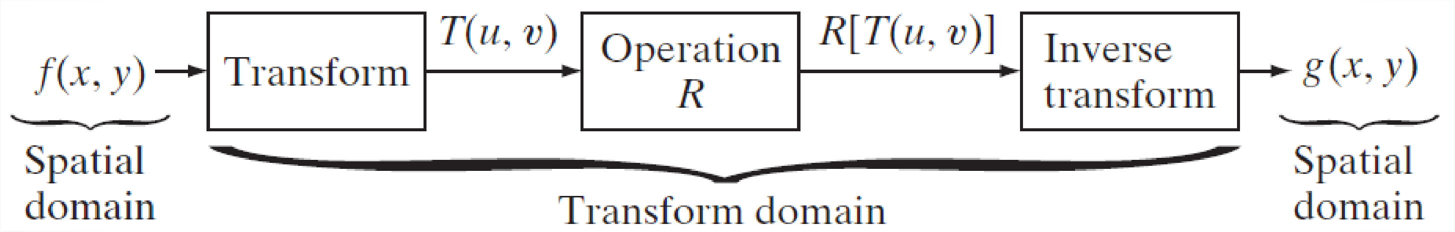
Image Transforms

- Approaches discussed till now work directly on *spatial domain*.
- Some tasks are best formulated by transforming the input images, carrying the specified task in a *transform domain*, and applying the inverse transform to return to the *spatial domain*.
- General form of 2-D linear transforms is given by:

$$T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) r(x, y, u, v) \quad \text{.....eq.(1)}$$

  - f(x, y) - is the input image
  - r(x, y, u, v) - is called the *forward transformation kernel*
  - *u – 0, 1, 2, ...,M-1*
  - *v – 0, 1, 2, ...,N-1*

- General approach for operating in the linear transform domain.



  - x, y - spatial variables
  - u, v – transform variables
  - M, N - row and column dimensions of *f*.
  - T(u, v) - is called the *forward transform* of f(x, y).

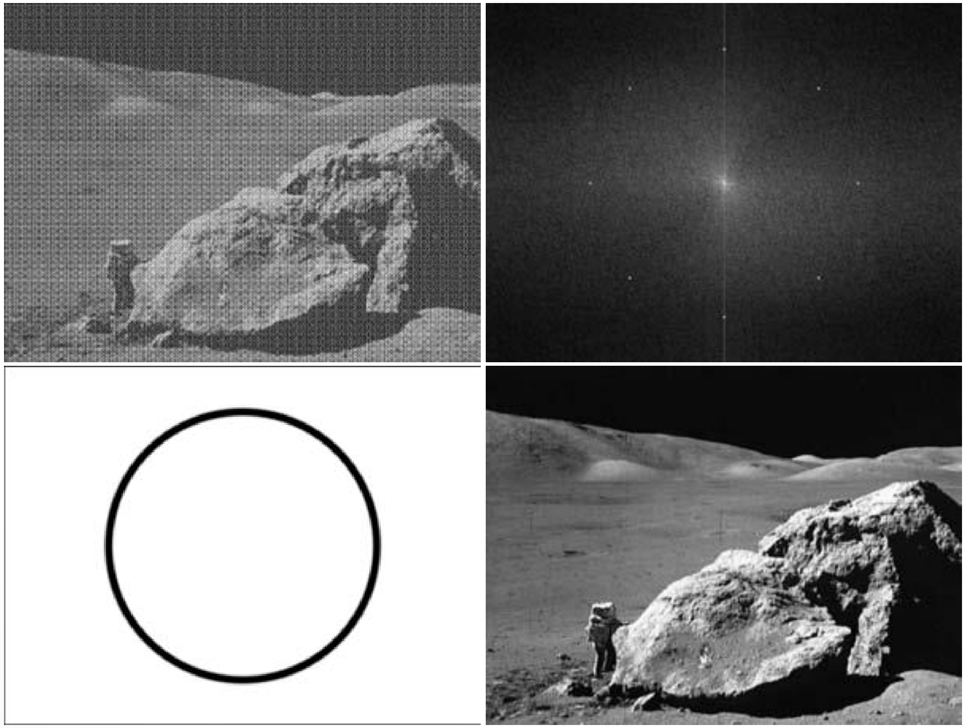
- Given T(u, v) , we can recover f(x, y) using the *inverse transform* of T(u, v),

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u, v) s(x, y, u, v) \quad \text{....eq.(2)}$$

  - x – 0,1,2,…..,M-1
  - y – 0,1,2,…..,N-1
  - s(x, y, u, v) - is called the *inverse transformation kernel*.

- (a) Image corrupted by sinusoidal interference.
- (b) Magnitude of the Fourier transform.

(a) (b)



(c) (d)

- (c) Mask (Filter) used to eliminate the energy bursts.
- (d) Result of computing the inverse of the modified Fourier transform.

- The forward transformation kernel is said to be *separable* if

$$r(x, y, u, v) = r_1(x, u) r_2(y, v)$$

- Also the kernel is said to be *symmetric* if $r_1(x, y)$ is functionally equal to $r_2(x, y)$.

$$r(x, y, u, v) = r_1(x, u) r_1(y, v)$$

- Identical comments apply to the inverse kernel by replacing *r* with *s* in the preceding equations.
- Thus, forward & inverse kernels are given by:

$$r(x, y, u, v) = e^{-j2\pi(ux/M + vy/N)}$$

$$s(x, y, u, v) = \frac{1}{MN} e^{j2\pi(ux/M + vy/N)}$$

- Substituting these kernels into the general transform formulations, we get the *Discrete Fourier transform pair*:

$$T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u, v) e^{j2\pi(ux/M + vy/N)}$$

- Fourier (forward and inverse) kernels are
  - separable and symmetric
  - allow 2-D transforms to be computed using 1-D transforms

    &
- f(x, y) is a square image of size M x M
- Then, eq.(1) & eq.(2) can be expressed in matrix form as

$$\mathbf{T} = \mathbf{AFA}$$

  - F – M x M matrix with element of input image f(x, y).
  - A – M x M matrix with elements $a_{ij} = r_1(x, y)$
  - T – resulting M x M matrix with values T(u, v), u, v = 0,1,2….., M-1

- To obtain the inverse transform, we pre- and post-multiply above equation by an inverse transformation matrix **B**.

$$\mathbf{BTB} = \mathbf{BAFAB}$$

- If $\mathbf{B} = \mathbf{A}^{-1}$

$$\mathbf{F} = \mathbf{BTB}$$

- **F** can be recovered completely from its forward transform.
- If $\mathbf{B} \neq \mathbf{A}^{-1}$
- Approximation is $\hat{\mathbf{F}} = \mathbf{BAFAB}$