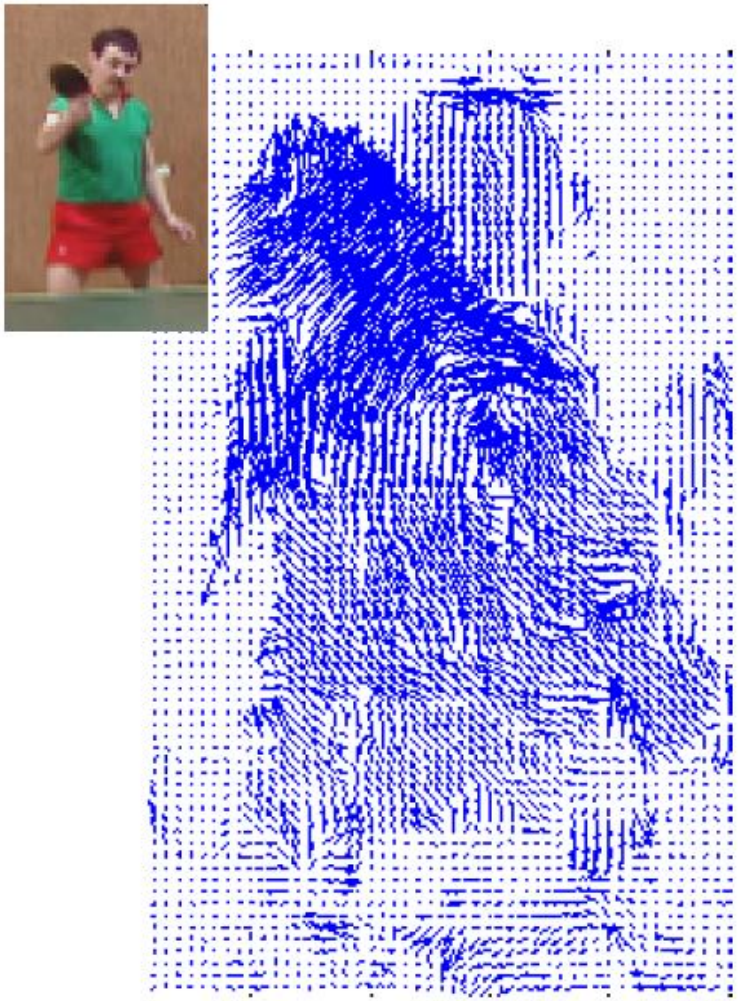
Optical Flow
CS-477 Computer Vision

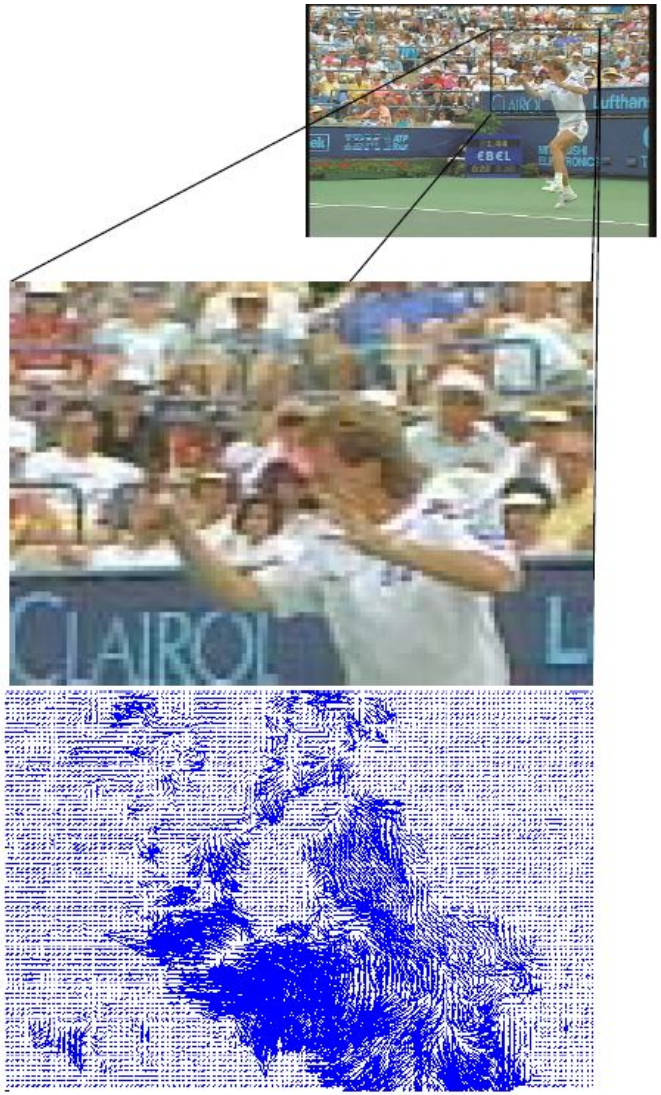
Dr. Mohsin Kamal
Associate Professor
dr.mohsinkamal@seecs.edu.pk

School of Electrical Engineering and Computer Science (SEECS)
National University of Sciences and Technology (NUST), Pakistan

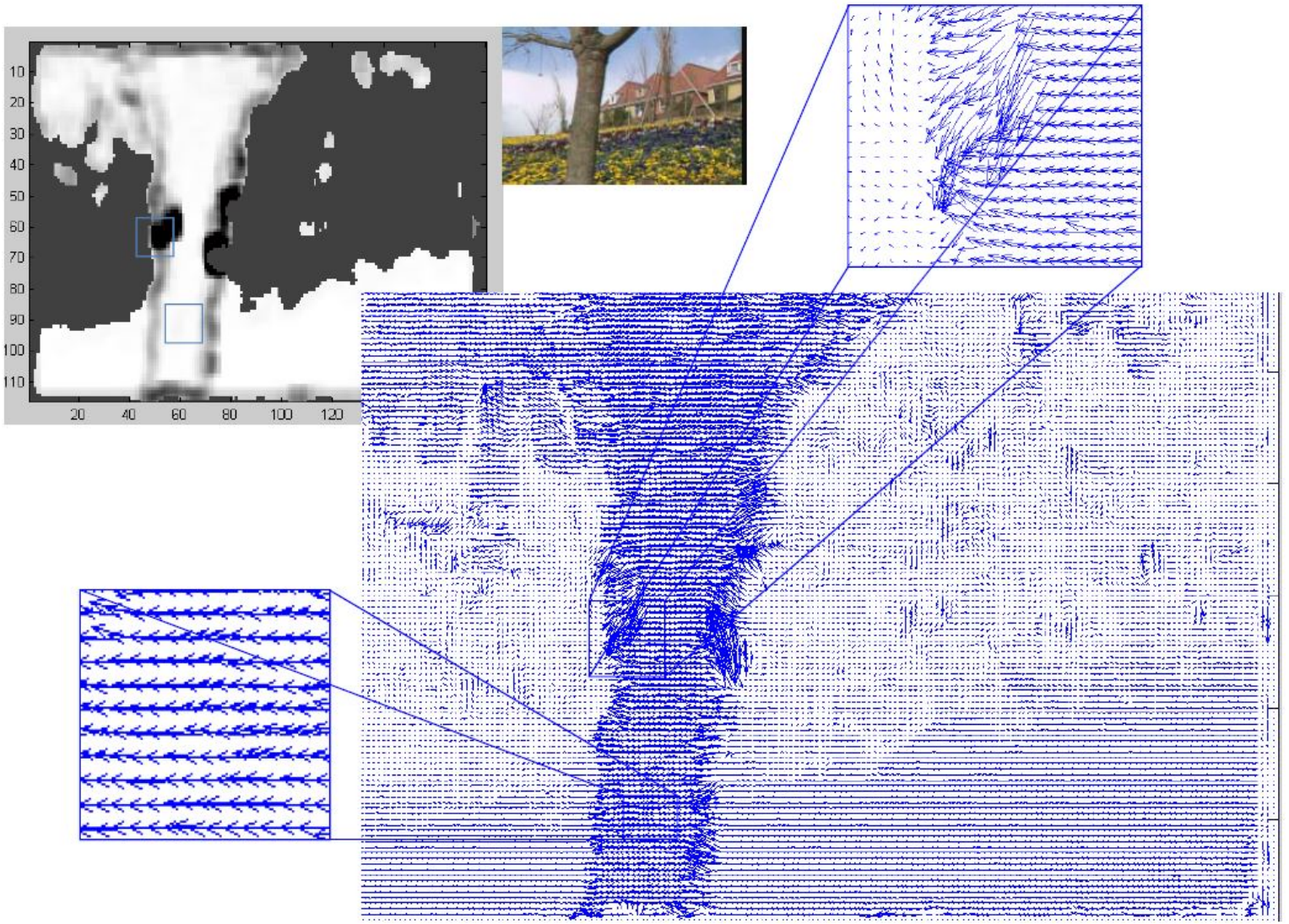
- Optical flow is a technique used to describe image motion.
- It is usually applied to a series of images that have a small time step between them, for example, video frames.
- Optical flow calculates a velocity for points within the images, and provides an estimation of where points could be in the next image sequence.
- Applications:
  - object tracking,
  - video stabilization, and
  - motion analysis.

Motion of the brightness patterns in an image is referred to as the "Optical Flow".





Optical flow equation



Optical flow equation

A1: Brightness constancy assumption

$$f(x, y, t) = f(x + dx, y + dy, t + dt)$$

A2: Displacements and time step are small

Taylor series approximation gives

$$f(x + dx, y + dy, t + dt) = f(x, y, t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt$$

let $\frac{\partial f}{\partial x}$ be denoted as f_x and so on. Above equation can then be written as:

$$f(x + dx, y + dy, t + dt) = f(x, y, t) + f_x dx + f_y dy + f_t dt$$

Subtracting **A1** from **A2**, we get

$$f_x dx + f_y dy + f_t dt = 0$$

Divide both sides by dt, we get

$$f_x u + f_y v + f_t = 0$$

Interpretation of optical flow equation

As we know

$$f_x u + f_y v + f_t = 0$$

This optical flow equation can further be modified as,

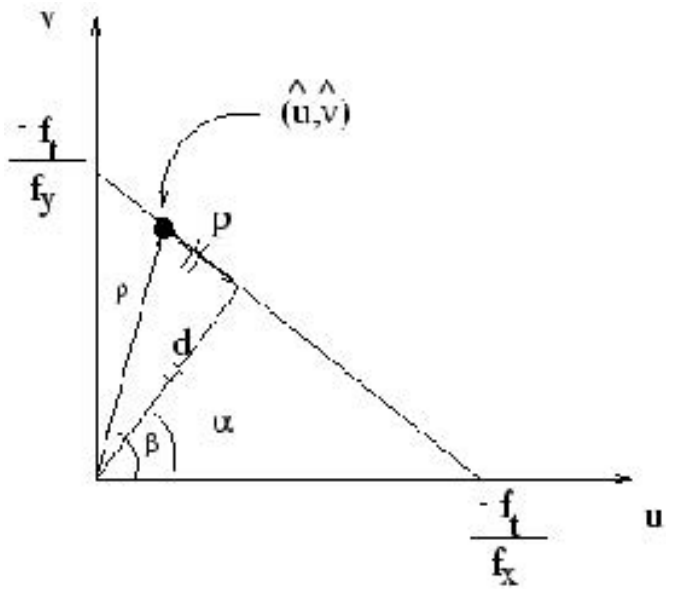
$$v = -\frac{f_x}{f_y} u - \frac{f_t}{f_y} \tag{1}$$

which is representing the equation of a straight line

d = normal flow

p = parallel flow

where $d = \frac{f_t}{\sqrt{f_x^2 + f_y^2}}$



Interpretation of optical flow equation

Derivative Masks

$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \text{first image}$$
$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \text{second image}$$
$$f_x$$

$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \text{first image}$$
$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \text{second image}$$
$$f_y$$

$$\begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} \text{first image}$$
$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \text{second image}$$
$$f_t$$

- Apply first mask to 1st image
- Apply second mask to 2nd image
- Add the response to get f_x, f_y and f_t

Lucas & Kanade

The Lucas Kanade method uses the assumption that the optical flow in a very small neighborhood in a scene is the same for all the points within that neighborhood.

Lucas & Kanade

Method 1:

- Optical flow equation

$$f_x u + f_y v = -f_t$$

- Consider a 3 by 3 window

$$f_{x1} u + f_{y1} v = -f_{t1}$$
$$\vdots$$
$$f_{x9} u + f_{y9} v = -f_{t9}$$

- This can be written as:

$$\begin{bmatrix} f_{x1} & f_{y1} \\ \vdots & \vdots \\ f_{x9} & f_{y9} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -f_{t1} \\ \vdots \\ -f_{t9} \end{bmatrix}$$

$$Au = f_t$$

Lucas & Kanade

$$Au = f_t$$

To make it a squared matrix, multiply A^T on both sides

$$A^T Au = A^T f_t$$

$$u = (A^T A)^{-1} A^T f_t$$

Method 2:

Apply least square fit as:

$$\min \sum_i (f_{xi} u + f_{yi} v + f_{ti})^2$$

$$\sum (f_{xi} u + f_{yi} v + f_{ti}) f_{xi} = 0$$

$$\sum (f_{xi} u + f_{yi} v + f_{ti}) f_{yi} = 0$$

$$\sum f_{xi}^2 u + \sum f_{xi} f_{yi} v = -\sum f_{xi} f_{ti}$$

$$\sum f_{xi} f_{yi} u + \sum f_{yi}^2 v = -\sum f_{yi} f_{ti}$$

$$\begin{bmatrix} \sum f_{xi}^2 & \sum f_{xi} f_{yi} \\ \sum f_{xi} f_{yi} & \sum f_{yi}^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum f_{xi} f_{ti} \\ -\sum f_{yi} f_{ti} \end{bmatrix}$$

Lucas & Kanade

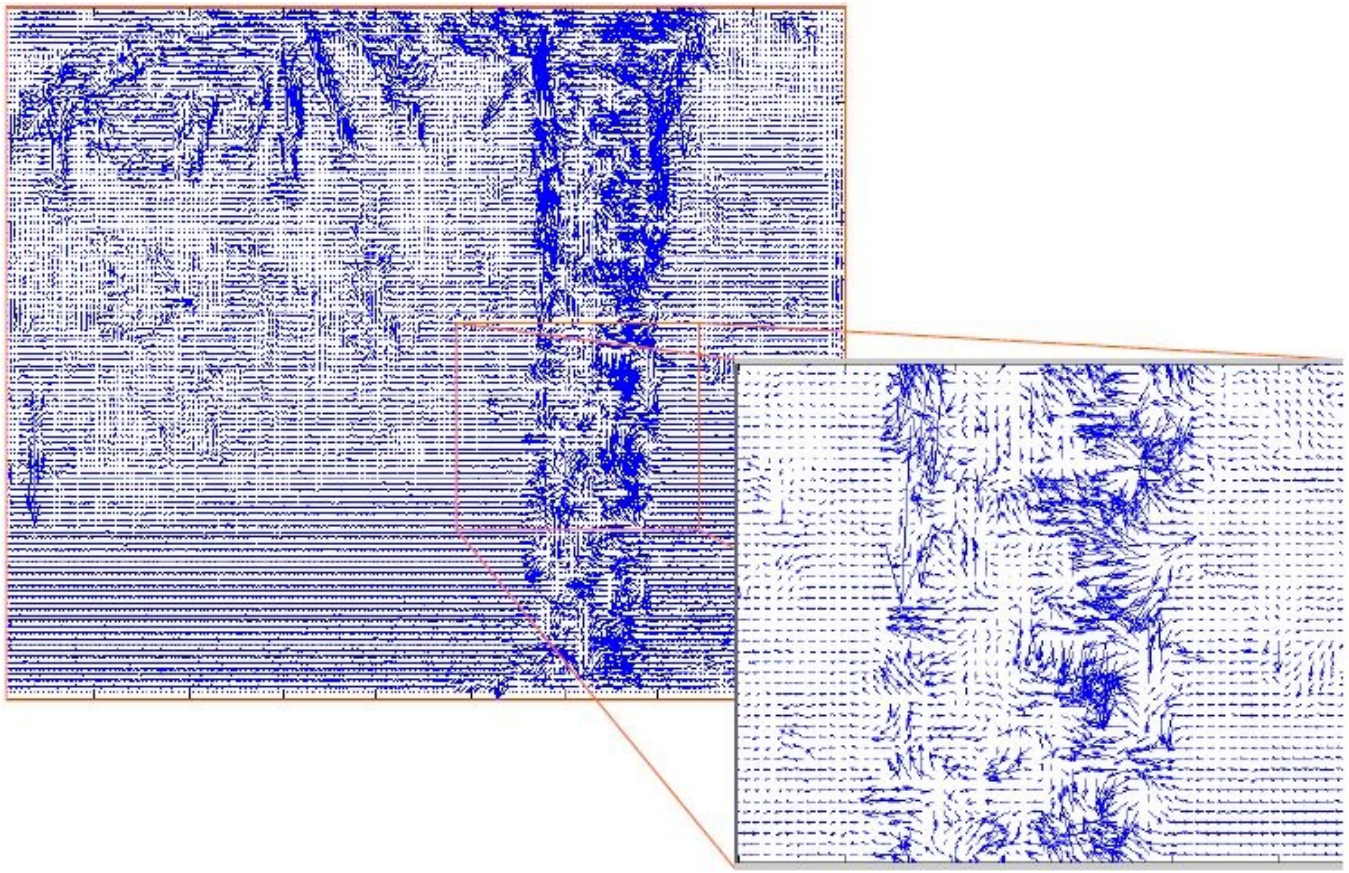
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum f_{xi}^2 & \sum f_{xi} f_{yi} \\ \sum f_{xi} f_{yi} & \sum f_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum f_{xi} f_{ti} \\ -\sum f_{yi} f_{ti} \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\sum f_{xi}^2 \sum f_{yi}^2 - (\sum f_{xi} f_{yi})^2} \begin{bmatrix} \sum f_{yi}^2 & -\sum f_{xi} f_{yi} \\ -\sum f_{xi} f_{yi} & \sum f_{xi}^2 \end{bmatrix} \begin{bmatrix} -\sum f_{xi} f_{ti} \\ -\sum f_{yi} f_{ti} \end{bmatrix}$$

$$u = \frac{-\sum f_{yi}^2 \sum f_{xi} f_{ti} + \sum f_{xi} f_{yi} \sum f_{yi} f_{ti}}{\sum f_{xi}^2 \sum f_{yi}^2 - (\sum f_{xi} f_{yi})^2}$$

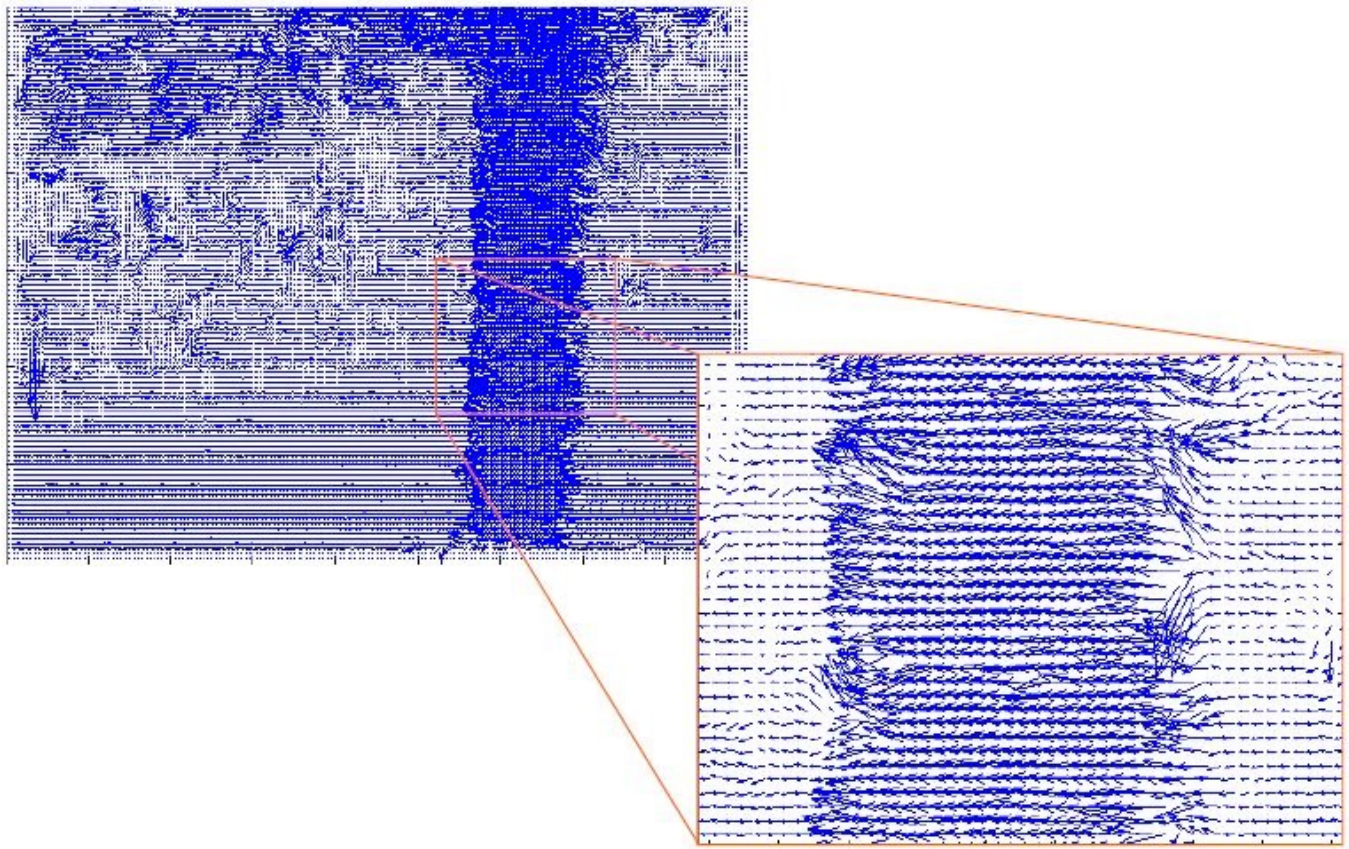
$$v = \frac{\sum f_{xi} f_{ti} \sum f_{xi} f_{yi} - \sum f_{xi}^2 \sum f_{yi} f_{ti}}{\sum f_{xi}^2 \sum f_{yi}^2 - (\sum f_{xi} f_{yi})^2}$$

Without pyramids



Fails in areas of large motion

With pyramids



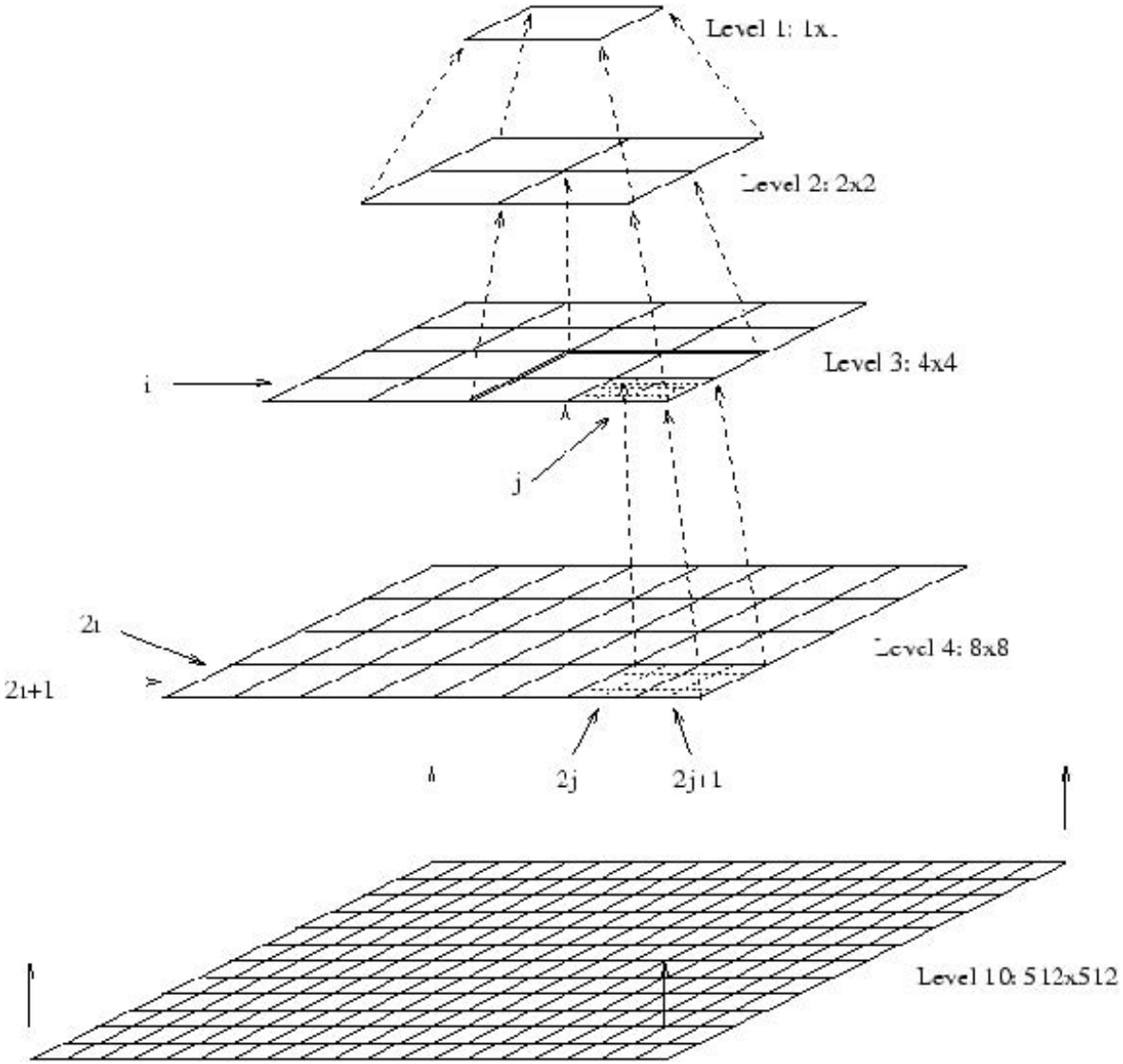
Comments

- Lucas-Kanade optical method works only for small motion.
- If object moves faster, the brightness changes rapidly,
  - 2×2 or 3×3 masks fail to estimate spatiotemporal derivatives.
- Pyramids can be used to compute large optical flow vectors.

Pyramids

- Very useful for representing images.
- Pyramid is built by using multiple copies of image.
- Each level in the pyramid is 1/4 of the size of previous level.
- The lowest level is of the highest resolution.
- The highest level is of the lowest resolution.

Pyramids


Level 1: 1x1
Level 2: 2x2
Level 3: 4x4
i
j
2i
2i+1
Level 4: 8x8
2j
2j+1
Level 10: 512x512

Reduce

$$g_l(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g_{l-1}(2i+m, 2j+n) \qquad (2)$$

- where l represents the level.
- $g_l = Reduce[g_{l-1}]$
- w represents the mask
- g is the image

Reduce (1D)

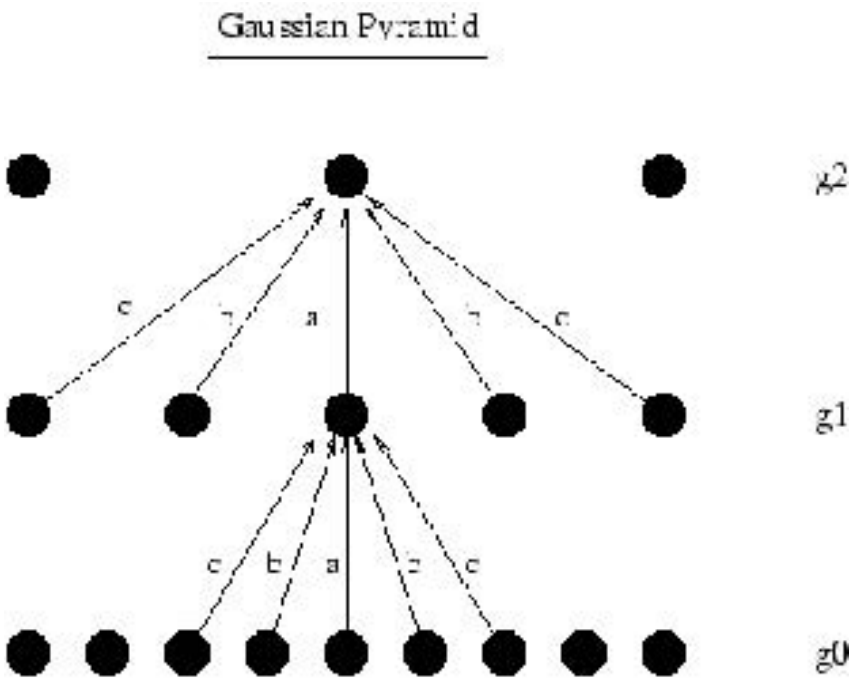
$$g_l(i) = \sum_{m=-2}^{2} w(m) g_{l-1}(2i + m) \tag{3}$$

$$g_l(2) = w(-2)g_{l-1}(4-2) + w(-1)g_{l-1}(4-1) + w(0)g_{l-1}(4) + w(1)g_{l-1}(4+1) + w(2)g_{l-1}(4+2)$$

$$g_l(2) = w(-2)g_{l-1}(2) + w(-1)g_{l-1}(3) + w(0)g_{l-1}(4) + w(1)g_{l-1}(5) + w(2)g_{l-1}(6)$$

Gaussian Pyramids

Reduce



$g0 = \text{IMAGE}$

$g1 = \text{REDUCE}[g_{l-1}]$

Expand

$$g_{l,n}(i,j) = \sum_{p=-2}^{2} \sum_{q=-2}^{2} w(p,q) g_{l,n-1}(\frac{i-p}{2}, \frac{j-q}{2}) \quad (4)$$

- where l represents the level.
- $g_l = Expand[g_{l,n-1}]$
- w represents the mask
- g is the image

Expand (1D)

$$g_{l,n}(i) = \sum_{p=-2}^{2} w(p) g_{l,n-1}(\frac{i-p}{2}) \tag{5}$$

$$g_{l,n}(4) = w(-2)g_{l,n-1}(\tfrac{4+2}{2}) + w(-1)g_{l,n-1}(\tfrac{4+1}{2}) + w(0)g_{l,n-1}(\tfrac{4}{2}) + w(1)g_{l,n-1}(\tfrac{4-1}{2}) + w(2)g_{l,n-1}(\tfrac{4-2}{2})$$

$$g_{l,n}(4) = w(-2)g_{l,n-1}(3) + w(0)g_{l,n-1}(2) + w(2)g_{l,n-1}(1)$$

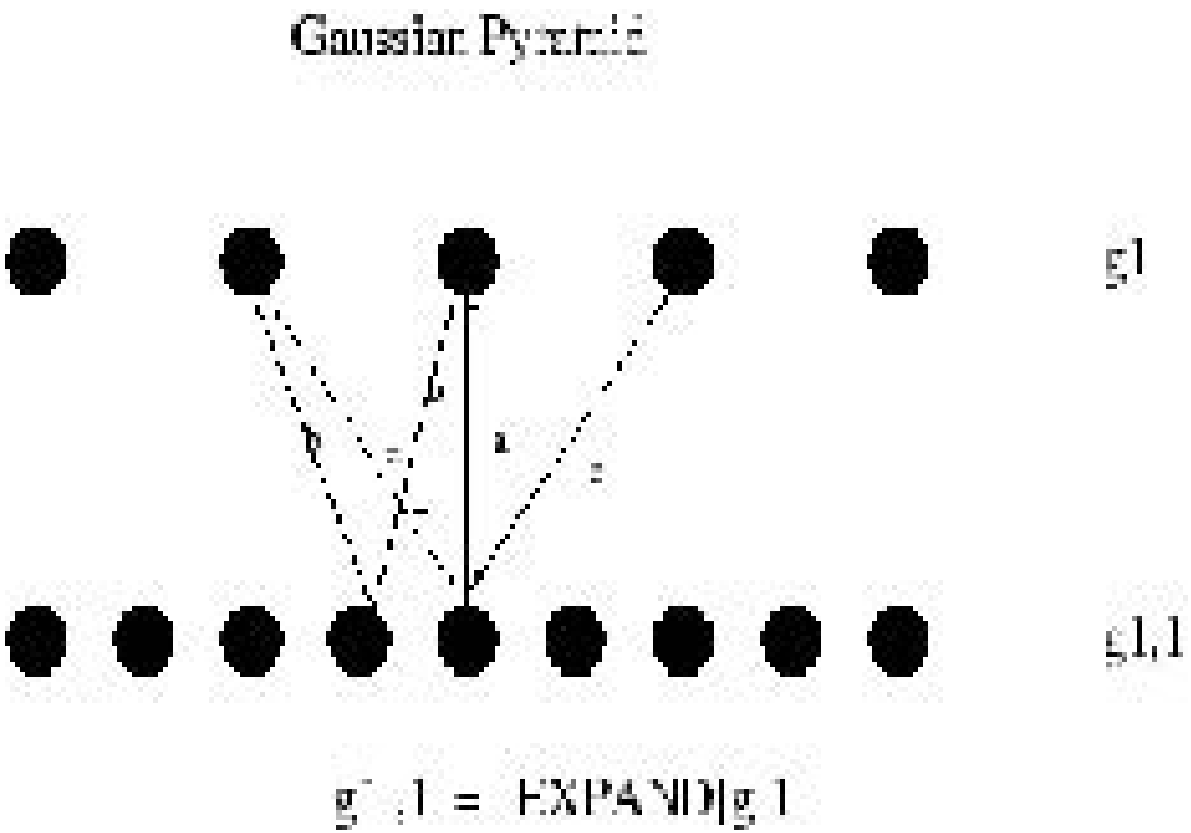
Expand (1D) - Another example

$$g_{l,n}(i) = \sum_{p=-2}^{2} w(p) g_{l,n-1}(\frac{i-p}{2}) \tag{6}$$

$$g_{l,n}(3) = w(-2)g_{l,n-1}(\tfrac{3+2}{2}) + w(-1)g_{l,n-1}(\tfrac{3+1}{2}) + w(0)g_{l,n-1}(\tfrac{3}{2}) + w(1)g_{l,n-1}(\tfrac{3-1}{2}) + w(2)g_{l,n-1}(\tfrac{3-2}{2})$$

$$g_{l,n}(3) = w(-1)g_{l,n-1}(2) + w(1)g_{l,n-1}(1)$$

Expand



Convolution mask

We have: $[w(-2), w(-1), w(0), w(1), w(2)]$

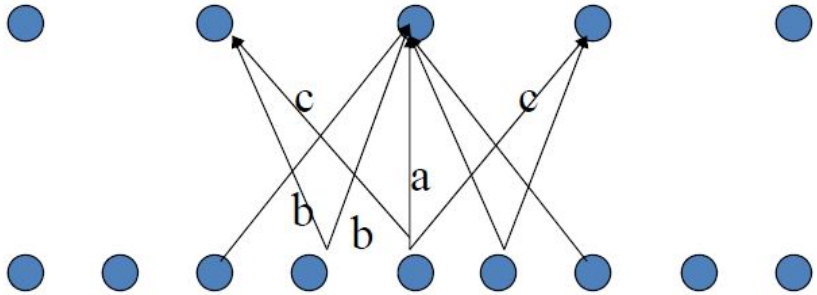
Properties of convolution mask

1. Separable, i.e., $w(m, n) = w(m)w(n)$
2. Symmetric, i.e., $w(i) = w(-i)$

Convolution mask

- The sum of mask should be equal to 1.
$$a + 2b + 2c = 1$$
- All nodes at a given level must contribute the same total weight to the nodes at the next higher level.
$$a + 2c = 2b$$

Convolution mask



From the above, the following can be observed

$$a + 2c = 2b$$
$$a + 2b + 2c = 1$$

Solving the above two equations, we get

$$b = \frac{1}{4}$$
$$c = \frac{1}{2}(2b - a)$$
$$c = \frac{1}{4} - \frac{a}{2}$$

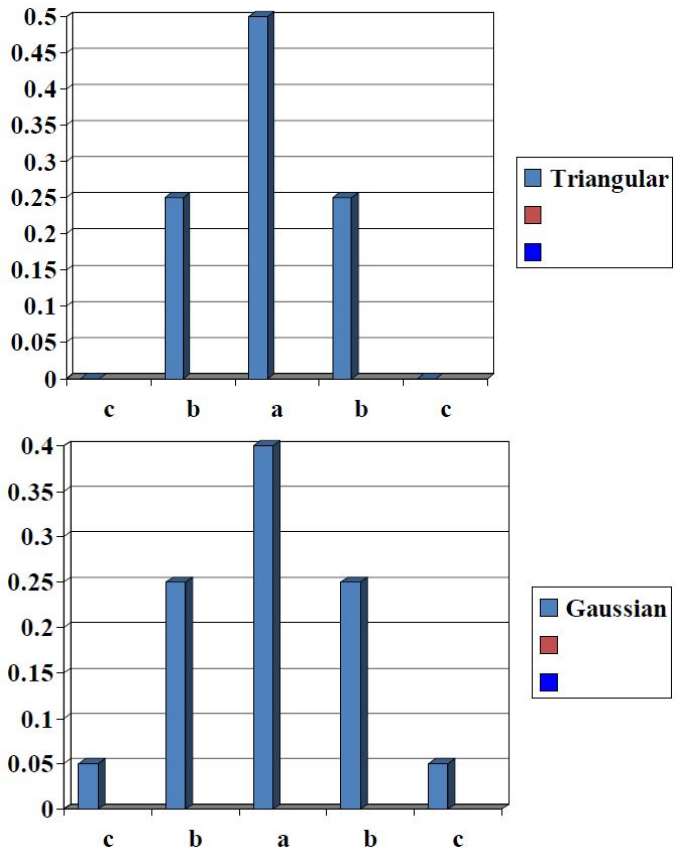
Convolution mask

So,

$$w(0) = a$$
$$w(-1) = w(1) = \frac{1}{4}$$
$$w(-2) = w(2) = \frac{1}{4} - \frac{a}{2}$$

We can get approximation of gaussian Gaussian and Triangular by using $a = 0.4$ and $a = 0.5$, respectively.

Convolution mask



Algorithm

- Apply 1-D mask to alternate pixels along each row of image.
- Apply 1-D mask to each pixel along alternate columns of resultant image from previous step.

Gaussian Pyramids

Laplacian Pyramids

- Similar to edge detected images.
- Most pixels are zero.
- Can be used for image compression

$$L_1 = g_1 - Expand[g_2]$$
$$L_2 = g_2 - Expand[g_3]$$
$$L_3 = g_3 - Expand[g_4]$$

Laplacian Pyramids

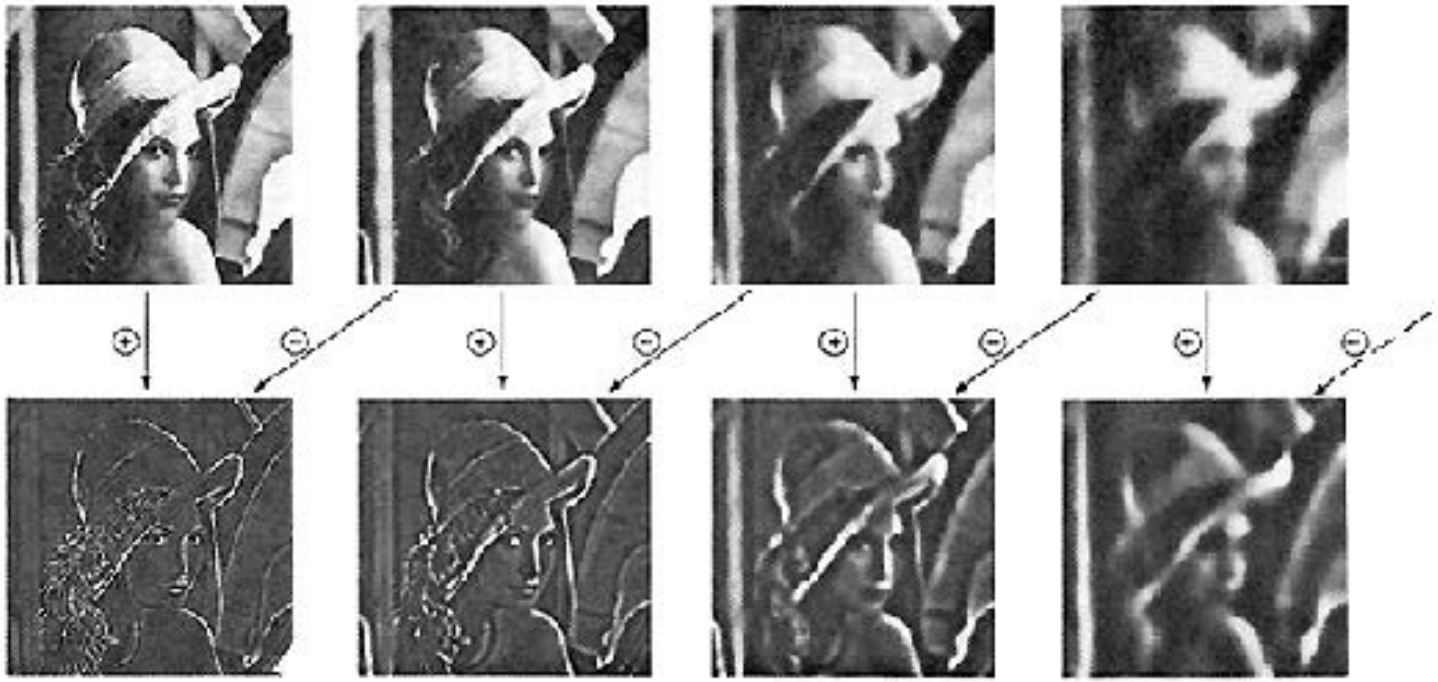


Fig 5. Four levels of the Gaussian and Laplacian pyramid. Gaussian images, appearing [illegible] (Fig. 1) through Gaussian interpolation. Each level of the Laplacian pyramid is the difference between corresponding levels of the Gaussian pyramid.

Steps of implementing Laplacian Pyramid

- Compute Gaussian Pyramid

$$g_1, g_2, g_3, g_4$$

- Compute Laplacian pyramid

$$L_1 = g_1 - Expand[g_2]$$
$$L_2 = g_2 - Expand[g_3]$$
$$L_3 = g_3 - Expand[g_4]$$
$$L_4 = g_4$$

- Code Laplacian pyramid

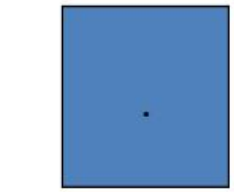
Decoding using Laplacian Pyramid

- Decode Laplacian pyramid
- Compute Gaussian pyramid from Laplacian pyramid

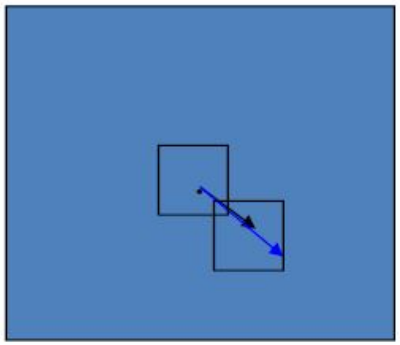
$$g_4 = L_4$$
$$g_3 = Expand[g_4] + L_3$$
$$g_2 = Expand[g_3] + L_2$$
$$g_1 = Expand[g_2] + L_1$$

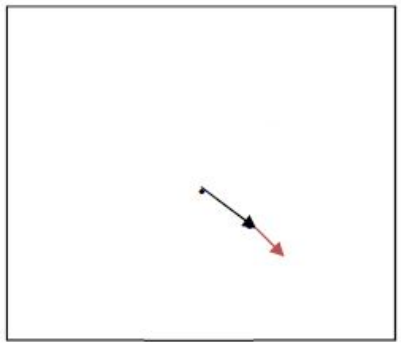
- g_1 is the reconstructed image

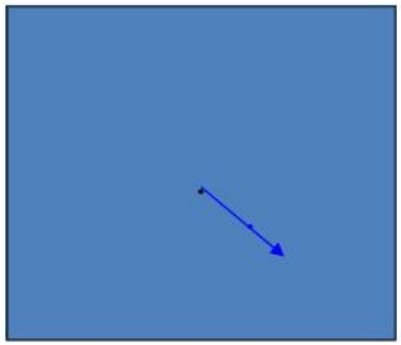
- Compute the pyramids of frames i.e., frame 1, frame 2, ...
- Compute "simple" LK optical flow at highest level (because we assume that the motion will be small)
- At level i
  - Take flow u_{i-1}, v_{i-1} from level $i-1$
  - bilinear interpolate it to create u_i^*, v_i^* matrices of twice resolution for level i
  - multiply u_i^*, v_i^* by 2
  - compute f_t from a block displaced by $u_i^*(x,y), v_i^*(x,y)$
  - Apply LK to get $u_i^{'}(x,y), v_i^{'}(x,y)$ (the correction in flow)
  - Add corrections $u_i^{'} v_i^{'}$, i.e., $u_i = u_i^* + u_i^{'}, v_i = v_i^* + v_i^{'}$.



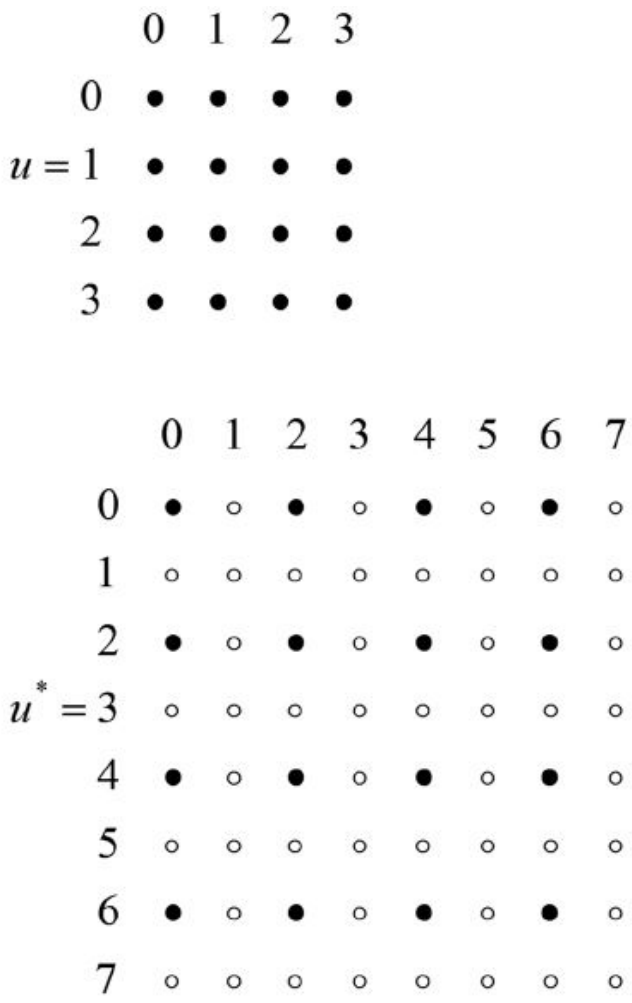
$$u_i = u_i^* + u_i', v_i = v_i^* + v_i'$$

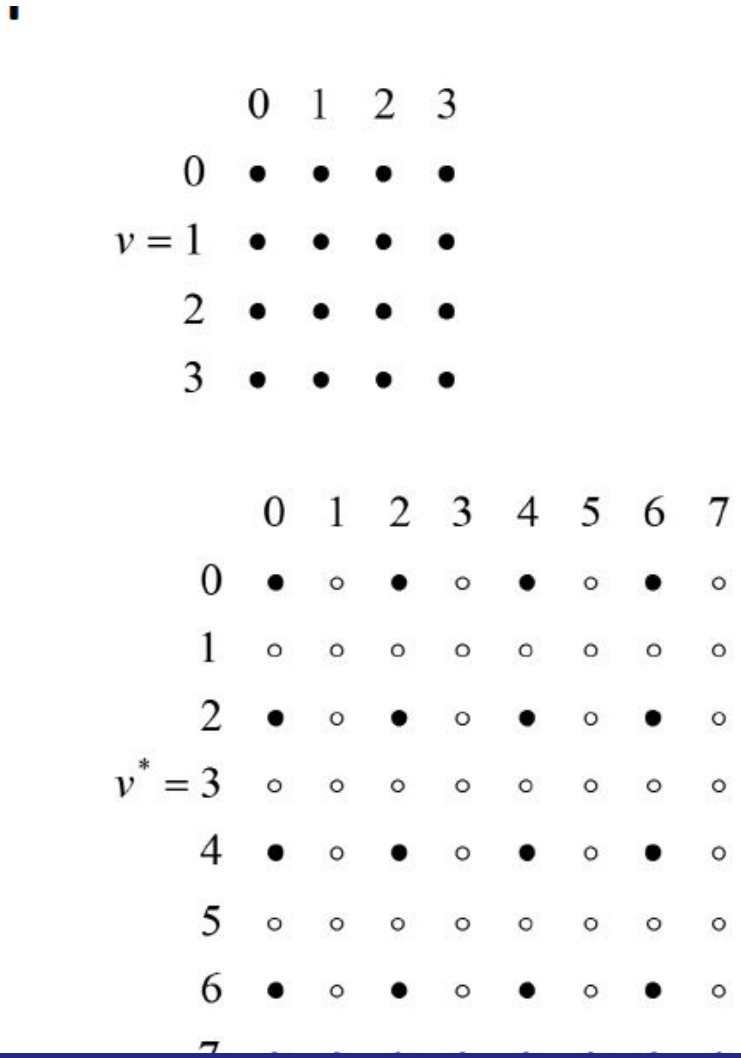




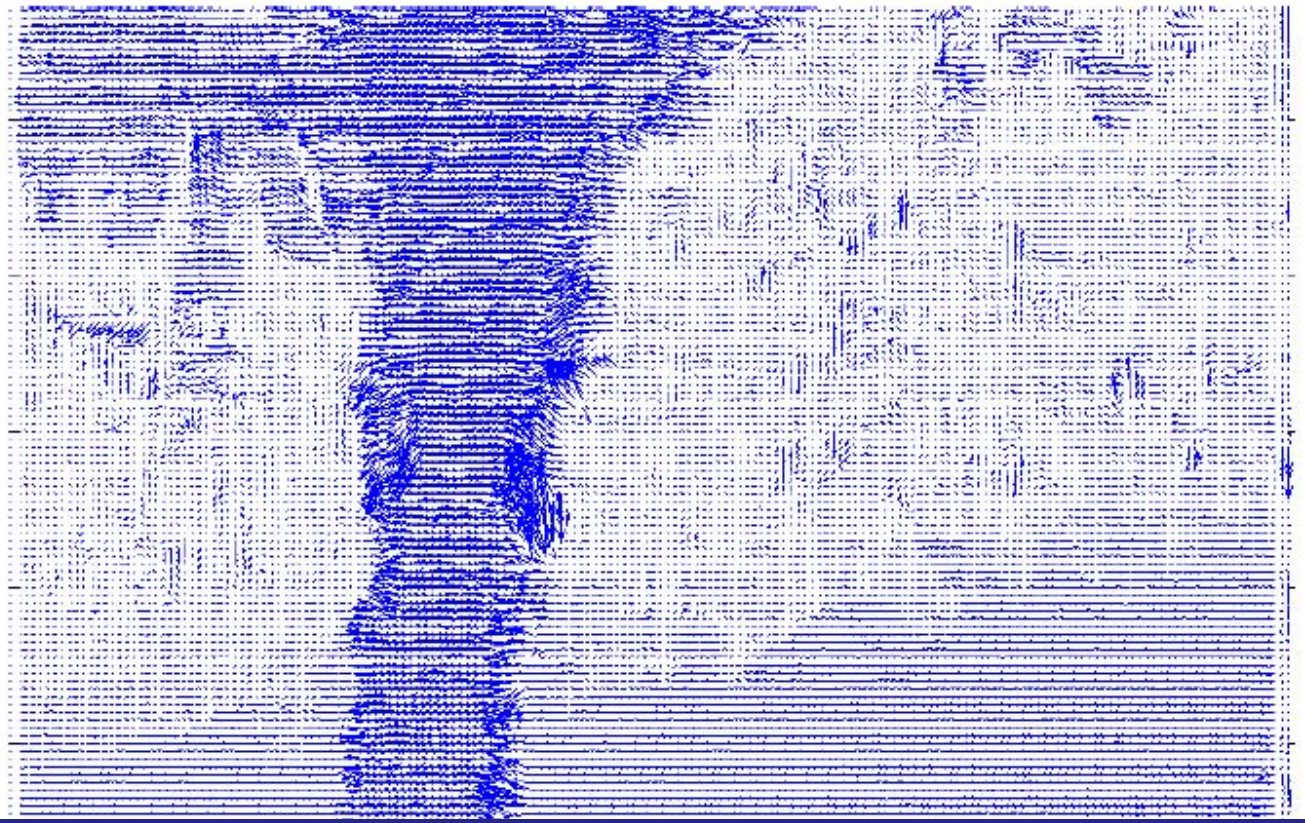


Interpolation





LK



LK