Matlab Tutorial. Optical Flow

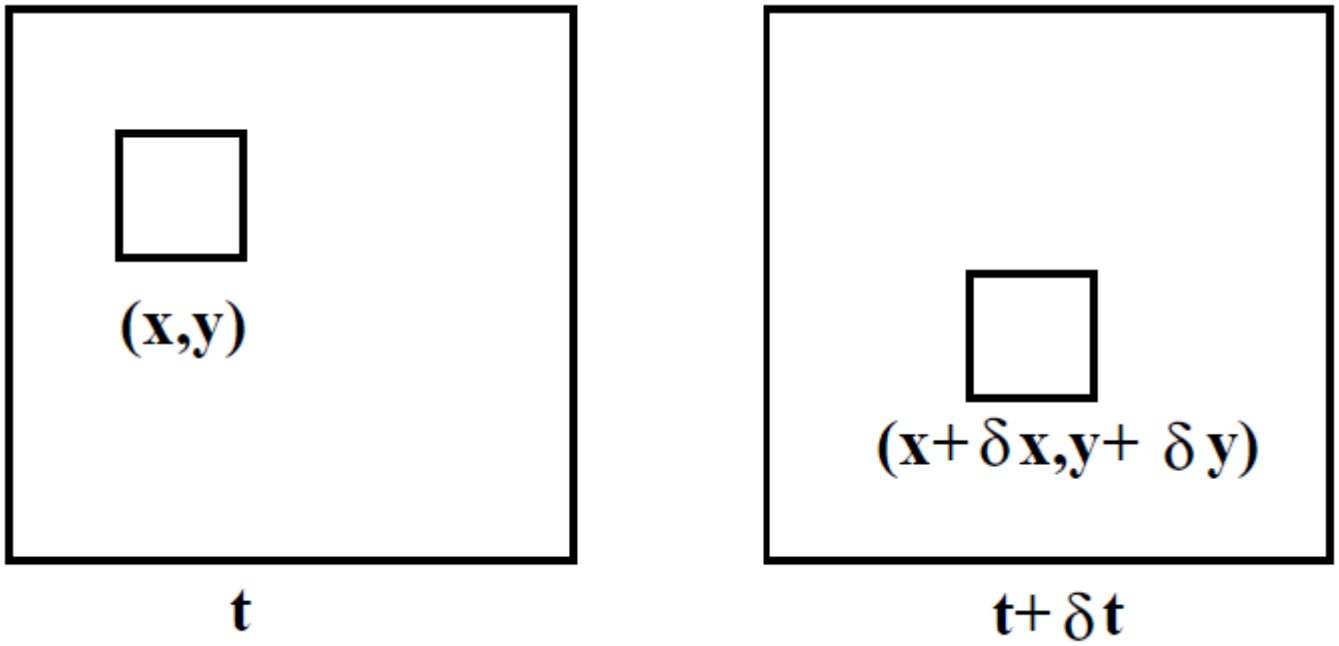
Gonzalo Vaca-Castano

REU 2013

Optical flow

- **Definition**
  - **Optical flow** or **optic flow** is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene

2D motion constraint



$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t).$$

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + H.O.T.,$$

2D motion constraint

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + H.O.T.,$$

$$\begin{aligned} \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t &= 0 \text{ or} \\ \frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\underbrace{\frac{\delta t}{\delta t}}_{=1} &= 0 \text{ and finally :} \\ \frac{\partial I}{\partial x}v_x + \frac{\partial I}{\partial y}v_y + \frac{\partial I}{\partial t} &= 0. \end{aligned}$$

$v_x = \frac{\delta x}{\delta t}$ and $v_y = \frac{\delta y}{\delta t}$ Are the components of Optical Flow

$\nabla I \cdot \vec{v} = -I_t \longrightarrow$ 2D Motion constraint

Lucas & Kanade (Least Squares)

- Optical flow eq

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

- Consider 3 by 3 window

$$f_{x1} u + f_{y1} v = -f_{t1}$$

$$\vdots$$

$$f_{x9} u + f_{y9} v = -f_{t9}$$

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

$$\mathbf{A}\mathbf{u} = \mathbf{f}_t$$

Lucas & Kanade

$$\mathbf{Au} = \mathbf{f}_t$$

$$\mathbf{A}^T\mathbf{Au} = \mathbf{A}^T\mathbf{f}_t$$

$$\mathbf{u} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{f}_t$$

Pseudo Inverse

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

Least Squares Fit

Lucas-Kanade implementation

```
function [u, v] = LucasKanade(im1, im2, windowSize);
%LucasKanade  lucas kanade algorithm, without pyramids (only 1 level);
%REVISION: NaN vals are replaced by zeros
[fx, fy, ft] = ComputeDerivatives(im1, im2);
u = zeros(size(im1));
v = zeros(size(im2));
halfWindow = floor(windowSize/2);
for i = halfWindow+1:size(fx,1)-halfWindow
   for j = halfWindow+1:size(fx,2)-halfWindow
      curFx = fx(i-halfWindow:i+halfWindow, j-halfWindow:j+halfWindow);
      curFy = fy(i-halfWindow:i+halfWindow, j-halfWindow:j+halfWindow);
      curFt = ft(i-halfWindow:i+halfWindow, j-halfWindow:j+halfWindow);

      curFx = curFx';
      curFy = curFy';
      curFt = curFt';


      curFx = curFx(:);
      curFy = curFy(:);
      curFt = -curFt(:);

      A = [curFx curFy];

      U = pinv(A'*A)*A'*curFt;

      u(i,j)=U(1);
      v(i,j)=U(2);
   end;
end;
u(isnan(u))=0;
v(isnan(v))=0;
```

Lucas-Kanade implementation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fx, fy, ft] = ComputeDerivatives(im1, im2);
%ComputeDerivatives Compute horizontal, vertical and time derivative
%                           between two gray-level images.

if (size(im1,1) ~= size(im2,1)) | (size(im1,2) ~= size(im2,2))
   error('input images are not the same size');
end;

if (size(im1,3)~=1) | (size(im2,3)~=1)
   error('method only works for gray-level images');
end;


fx = conv2(im1,0.25* [-1 1; -1 1]) + conv2(im2, 0.25*[-1 1; -1 1]);
fy = conv2(im1, 0.25*[-1 -1; 1 1]) + conv2(im2, 0.25*[-1 -1; 1 1]);
ft = conv2(im1, 0.25*ones(2)) + conv2(im2, -0.25*ones(2));

% make same size as input
fx=fx(1:size(fx,1)-1, 1:size(fx,2)-1);
fy=fy(1:size(fy,1)-1, 1:size(fy,2)-1);
ft=ft(1:size(ft,1)-1, 1:size(ft,2)-1);
```

Optical Flow code (Download it from webpage)

```
addpath('LucasKanade');
i1=imread(‘car1.jpg');
i2=imread(‘car2.jpg');
[u,v] =LucasKanade(rgb2gray(i1),rgb2gray(i2),20);
f(:,:,1)=u;
f(:,:,2)=v;
imshow(flowtocolor(f));
```

Optical Flow (input)

Optical Flow (input)

Optical Flow (Output)

Lucas Kanade with Pyramids

- Compute 'simple' LK optical flow at highest level
- 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.* $u_i = u_i^* + u_i'$, $v_i = v_i^* + v_i'$.

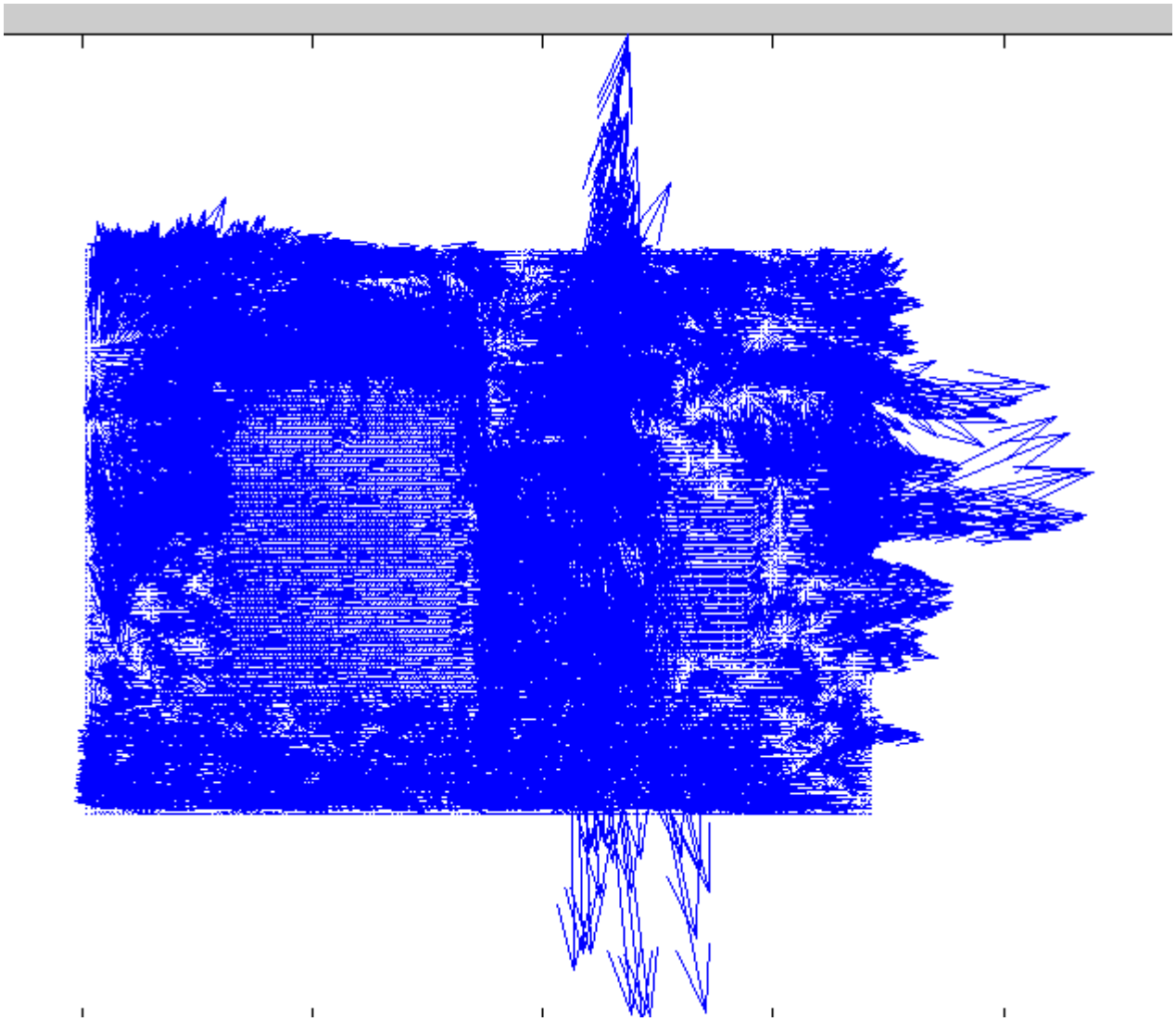
Optical Flow code (Download it from webpage)

```
addpath('LucasKanade');
i1=imread('table1.jpg');
i2=imread('table2.jpg');
[u,v,cert]
   =HierarchicalLK(rgb2gray(i1),rgb2gray(i2),3,2,2,1)
f(:,:,1)=u;
f(:,:,2)=v;
flowtocolor(f)
```

Optical Flow with pyramids (input)

Optical Flow with pyramids (input)

Optical Flow with pyramids (output)



Optical Flow with Pyramids