Motion

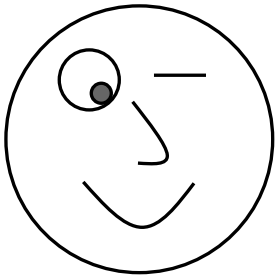
1

Outline

- Motion basics
- Lucas-Kanade algorithm
- Horn-Schunck algorithm

2

What if you only had one eye?



Depth perception is possible by moving eye

3

Head movements for depth perception

Some animals move their heads to generate parallax

pigeon

http://pinknpurplelizard.files.wordpress.com/2008/06/pigeon1.jpg

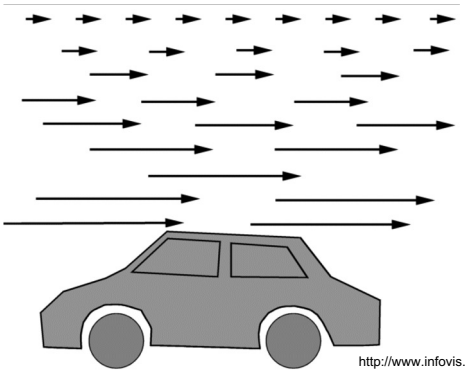
praying mantis

http://www.animalwebguide.com/Praying-Mantis.htm

4

Parallax

Motion *parallax* – apparent displacement of object viewed along two different lines of sight



http://www.infovis.net/imagenes/T1_N144_A6_DifMotion.gif

5

Difference between stereo and 2-frame motion

- **Stereo:**
  - **Images taken at same time**
  - **Scene is guaranteed to be static**
  - **Epipolar constraint restricts search to 1D**
- **2-frame motion:**
  - **Images taken at different times**
  - **Objects in scene may have independently moved**
  - **Epipolar constraint no longer guaranteed**

Of course, motion involves a video sequence of images

6

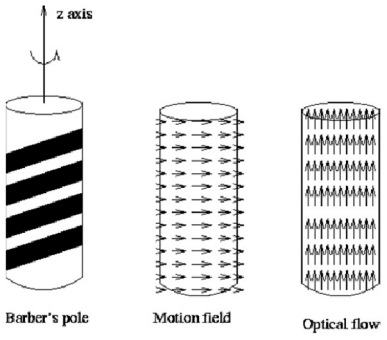
Motion field and optical flow

- ***Motion field*** **– The actual 3D motion projected onto image plane**
- ***Optical flow*** **– The "apparent" motion of the brightness pattern in an image**
  (sometimes called *optic flow*)

7

When are the two different?

Barber pole illusion



http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT12/node4.html

Television / movie

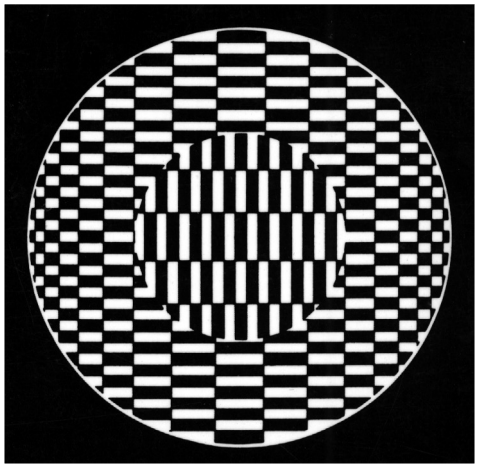
(no motion field)

Rotating ping-pong ball

(no optical flow)

8

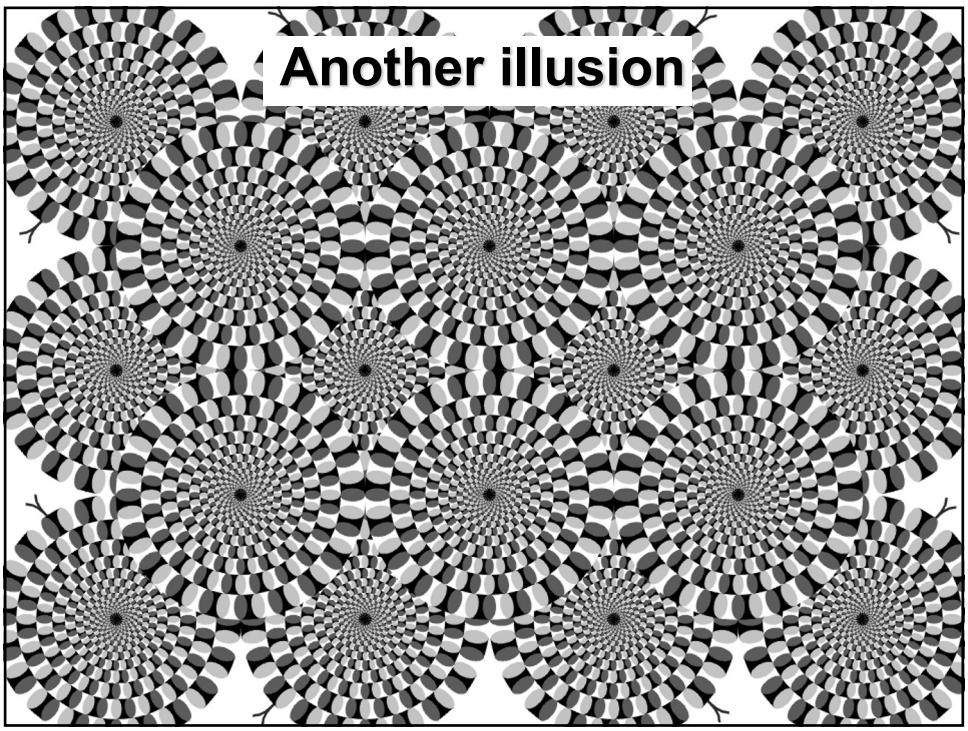
Optical flow breakdown



⟷ Perhaps an aperture problem discussed later.

➔

9



10

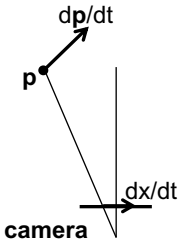
Motion field

Point in world:

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Projection onto image:

$$(x, y) = \left(f\frac{p_x}{p_z}, f\frac{p_y}{p_z} \right)$$



Motion of point in world:

$$\frac{d\mathbf{p}}{dt} = \mathbf{t} - \omega \times \mathbf{p} = \begin{bmatrix} -t_x - \omega_y p_z + \omega_z p_y \\ -t_y - \omega_z p_x + \omega_x p_z \\ -t_z - \omega_x p_y + \omega_y p_x \end{bmatrix}$$

translation — rotation

where $\omega \times \mathbf{p} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \omega_x & \omega_y & \omega_z \\ p_x & p_y & p_z \end{vmatrix} = \begin{bmatrix} \omega_y p_z - \omega_z p_y \\ \omega_z p_x - \omega_x p_z \\ \omega_x p_y - \omega_y p_x \end{bmatrix}$

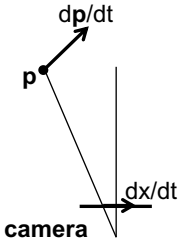
11

Motion field (cont.)

Image velocity of projection:

$$\begin{aligned} u &\equiv \frac{dx}{dt} = \frac{f}{p_z^2}\left(p_z \frac{dp_x}{dt} - p_x \frac{dp_z}{dt} \right) \\ v &\equiv \frac{dy}{dt} = \frac{f}{p_z^2}\left(p_z \frac{dp_y}{dt} - p_y \frac{dp_z}{dt} \right) \end{aligned}$$

$$\mathbf{u} \equiv \begin{bmatrix} u \\ v \end{bmatrix}$$



Expanding yields:

$$\begin{aligned} u &= \frac{f}{p_z^2}\left(p_z \frac{dp_x}{dt} - p_x \frac{dp_z}{dt} \right) \\ &= \frac{f}{p_z^2}\left((-t_x - \omega_y p_z + \omega_z p_y) p_z - p_x(-t_z - \omega_x p_y + \omega_y p_x) \right) \\ &= f\frac{t_z p_x - t_x p_z}{p_z^2} - \omega_y f + \frac{\omega_z f p_y}{p_z} + \frac{\omega_x f p_x p_y}{p_z^2} - \frac{\omega_y f p_x^2}{p_z^2} \\ &= \frac{t_z x - t_x f}{p_z} - \omega_y f + \omega_z y + \frac{\omega_x xy}{f} - \frac{\omega_y x^2}{f} \\ v &= \frac{t_z y - t_y f}{p_z} + \omega_x f - \omega_z x - \frac{\omega_y xy}{f} + \frac{\omega_x y^2}{f} \end{aligned}$$

depth

Note that rotation gives no depth information

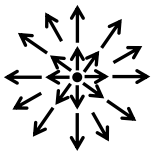
12

Motion field (cont.)

Two special cases:

Special case: pure translation. $\omega = 0$.

$$u = \frac{t_z x - t_x f}{p_z} = \frac{t_z}{p_z}\left(x - \frac{t_x f}{t_z}\right)$$
$$v = \frac{t_z y - t_y f}{p_z} = \frac{t_z}{p_z}\left(y - \frac{t_y f}{t_z}\right)$$

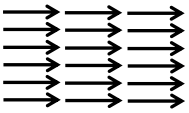


Note: This is a radial field emanating from the *instantaneous epipole*

$$\left(f\frac{t_x}{t_z}, f\frac{t_y}{t_z}\right)$$

Special case: rectified stereo. $\omega = 0$ and $t_z = t_y = 0$.

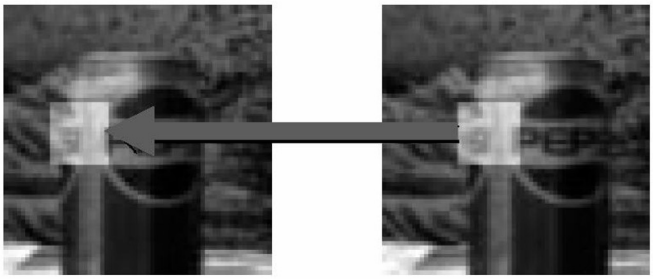
$$u = -\frac{t_x f}{p_z}$$
$$v = 0$$



Note: Here the *instantaneous epipole* is at infinity

13

Optical Flow Assumptions: Brightness Constancy



Assumption

Image measurements (e.g. brightness) in a small region remain the same although their location may change.

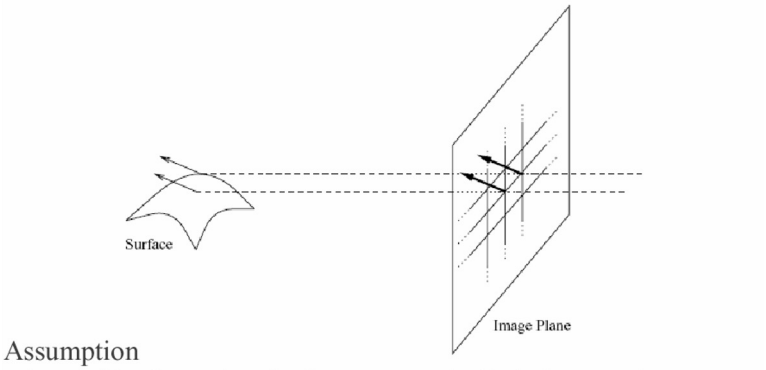
$$I(x+u, y+v, t+1) = I(x, y, t)$$

(assumption)

* Slide from Michael Black, CS143 2003

14

Optical Flow Assumptions:
Spatial Coherence



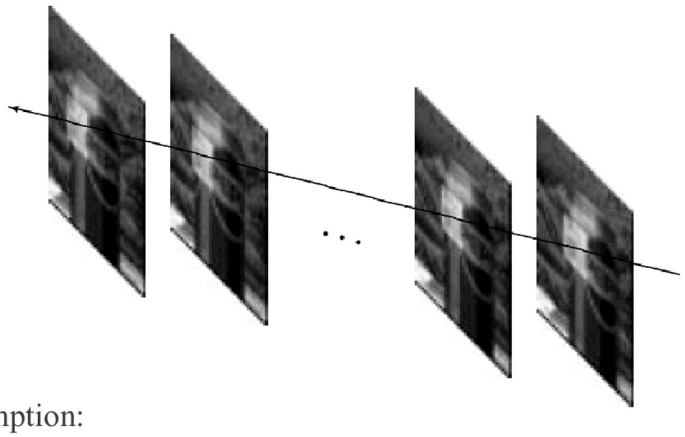
Assumption

* Neighboring points in the scene typically belong to the same surface and hence typically have similar motions.
* Since they also project to nearby points in the image, we expect spatial coherence in image flow.

* Slide from Michael Black, CS143 2003

15

Optical Flow Assumptions:
Temporal Persistence



Assumption:

The image motion of a surface patch changes gradually over time.

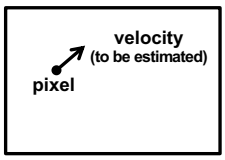
* Slide from Michael Black, CS143 2003

16

Optical flow constraint equation

$$I(x(t+\Delta t), y(t+\Delta t), t+\Delta t) = I(x(t), y(t), t)$$

Brightness constancy assumption



$$\frac{dx}{dt}\frac{\partial I}{\partial x} + \frac{dy}{dt}\frac{\partial I}{\partial y} \approx -\frac{\partial I}{\partial t}$$

Optical flow constraint equation

image velocity (unknown)

image gradient (known)

temporal derivative (known)

$$\frac{dx}{dt} \approx x(t+1) - x(t) \qquad \frac{dy}{dt} \approx y(t+1) - y(t) \qquad I_t \approx I(t+1) - I(t)$$

17

Recall: Taylor series

A function *f(x)* can be approximated near $x=x_0$ by

$$f(x) = f(x_0) + \frac{(x-x_0)}{1!}\frac{df(x_0)}{dx} + \frac{(x-x_0)^2}{2!}\frac{d^2f(x_0)}{dx^2} + \frac{(x-x_0)^3}{3!}\frac{d^3f(x_0)}{dx^3} + \cdots$$

Linear approximation is

$$f(x) \approx f(x_0) + (x-x_0)\frac{df(x_0)}{dx}$$

If *f* is a function of 2 variables:

$$f(x,y) \approx f(x_0,y_0) + (x-x_0)\frac{\partial f(x_0,y_0)}{\partial x} + (y-y_0)\frac{\partial f(x_0,y_0)}{\partial y}$$

If *f* is a function of 3 variables:

$$f(x,y,t) \approx f(x_0,y_0,t_0) + (x-x_0)\frac{\partial f(x_0,y_0,t_0)}{\partial x} + (y-y_0)\frac{\partial f(x_0,y_0,t_0)}{\partial y} + (t-t_0)\frac{\partial f(x_0,y_0,t_0)}{\partial t}$$

18

Deriving the optical flow constraint equation

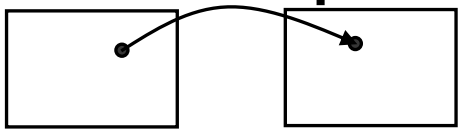


Image at time t Image at time t + Δt

Brightness constancy assumption:

$$I(x(t+\Delta t), y(t+\Delta t), t+\Delta t) = I(x(t), y(t), t)$$

Taylor series expansion:

$$I(x(t+\Delta t), y(t+\Delta t), t+\Delta t) \approx$$

$$I(x(t), y(t), t) + (x(t+\Delta t) - x(t))\frac{\partial I}{\partial x} + (y(t+\Delta t) - y(t))\frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t}$$

Putting together yields:

$$(x(t+\Delta t) - x(t))\frac{\partial I}{\partial x} + (y(t+\Delta t) - y(t))\frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t} \approx 0$$

19

Deriving the optical flow constraint equation

From previous slide:

$$(x(t+\Delta t) - x(t))\frac{\partial I}{\partial x} + (y(t+\Delta t) - y(t))\frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t} \approx 0$$

Divide both sides by Δt and take the limit:

$$\lim_{\Delta t \to 0} \frac{(x(t+\Delta t) - x(t))}{\Delta t}\frac{\partial I}{\partial x} + \lim_{\Delta t \to 0} \frac{(y(t+\Delta t) - y(t))}{\Delta t}\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} \approx 0$$

or

$$\boxed{\frac{dx}{dt}\frac{\partial I}{\partial x} + \frac{dy}{dt}\frac{\partial I}{\partial y} \approx -\frac{\partial I}{\partial t}}$$

optical flow constraint equation

More compactly,

$$(\nabla I)^T \mathbf{u} \approx -I_t$$

$$\nabla I \equiv \begin{bmatrix} I_x \\ I_y \end{bmatrix} \equiv \begin{bmatrix} \partial I/\partial x \\ \partial I/\partial y \end{bmatrix} \qquad \mathbf{u} \equiv \begin{bmatrix} u \\ v \end{bmatrix} \equiv \begin{bmatrix} dx/dt \\ dy/dt \end{bmatrix} \qquad I_t \equiv \frac{\partial I}{\partial t}$$

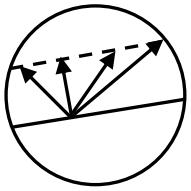
20

Aperture problem

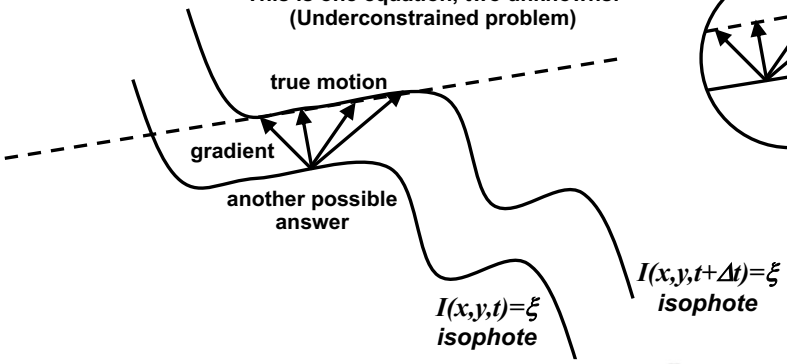
From previous slide:

$$I_x(\mathbf{x})u + I_y(\mathbf{x})v \approx -I_t(\mathbf{x})$$

Key idea:
Any function looks linear through small `aperture'



This is one equation, two unknowns!
(Underconstrained problem)

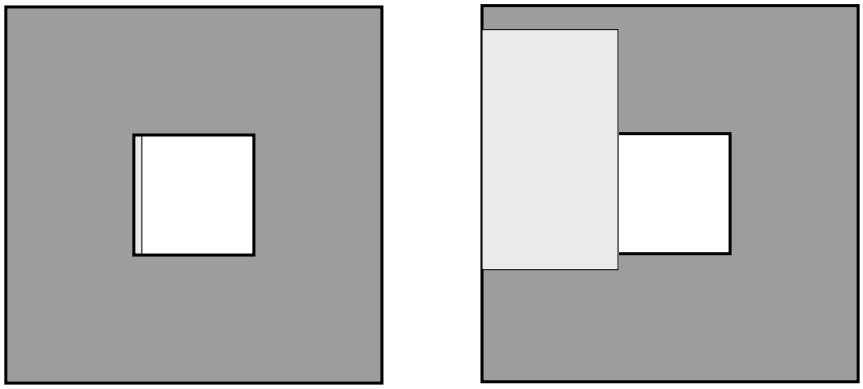


We can only compute component of motion in direction of gradient:

$$u_n = \frac{-I_t}{||\nabla I||} = \frac{(\nabla I)^T \mathbf{u}}{||\nabla I||} = \left(\frac{\nabla I}{||\nabla I||}\right)^T \mathbf{u}$$

21

Aperture Problem Exposed



Motion along just an edge is ambiguous

from G. Bradski, CS223B

22

Two approaches to overcome aperture problem

- Horn-Schunck (1980)
  - Assume neighboring pixels are *similar*
  - Add regularization term to enforce smoothness
  - Compute (u,v) for every pixel in image
  - → Dense optical flow
- Lucas-Kanade (1981)
  - Assume neighboring pixels are *same*
  - Use additional equations to solve for motion of pixel
  - Compute (u,v) for a small number of pixels (features); each feature treated independently of other features
  - → Sparse optical flow

23

Outline

- Motion basics
- Lucas-Kanade algorithm
- Horn-Schunck algorithm

24

Lucas-Kanade

Recall scalar "equation" was derived by linearizing function about current estimate:

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \approx -I_t$$ **+ higher order terms**

Total displacement that we want (u)

Reinterpret:

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = -I_t$$

Incremental displacement that we can compute (u_Δ)

This is one iteration of Newton's method. So we iterate and accumulate:

$$\mathbf{u} = \mathbf{u}_0 + \mathbf{u}_\Delta^{(1)} + \mathbf{u}_\Delta^{(2)} + \mathbf{u}_\Delta^{(3)} + \cdots$$

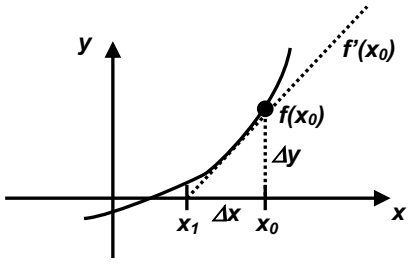
Initial estimate (could be zero)

25

Recall: Newton's method

Goal: Find root (or zero) of function *f(x)*

- **Use first-order approximation (tangent line)**
- **Start with initial guess**
- **Iterate:**
  - **Use derivative to find root, under assumption that function is linear**
  - **Result used as estimate for next iteration**



Solution: Iterate

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$f'(x_n) = \frac{\text{rise}}{\text{run}} = \frac{\Delta \text{y}}{\Delta \text{x}} = \frac{f(x_n) - 0}{x_n - x_{n+1}}$$

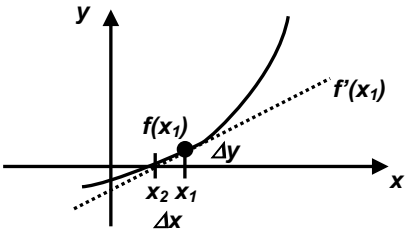
x = x₁ = x₀ + Δx

26

Recall: Newton's method

Goal: Find root (or zero) of function *f(x)*

- **Use first-order approximation (tangent line)**
- **Start with initial guess**
- **Iterate:**
  - **Use derivative to find root, under assumption that function is linear**
  - **Result used as estimate for next iteration**



Solution: Iterate

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$f'(x_n) = \frac{\text{rise}}{\text{run}} = \frac{\Delta y}{\Delta x} = \frac{f(x_n) - 0}{x_n - x_{n+1}}$$

$x = x_2 = x_1 + \Delta x$

27

Recall: Newton's method

Goal: Find extremum (min or max) of function *f(x)*

Key idea: Extremum occurs when *f'(x)=0*
→ Replace *f* with *f'*

Solution: Iterate

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

Note: Now we need 2nd derivative

28

Recall: Gauss-Newton method

Goal: Find extremum (min or max) of function *f(x)=(e(x))²*

Key idea: When *f(x)* has this special form, derivatives of *f* are easy to compute:

f'(x) = 2 e(x) e'(x)

f''(x) = 2 (e'(x))² + 2 e(x) e''(x)

Ignoring the 2nd derivative yields

f''(x) ≈ 2 (e'(x))²

Solution: Iterate

$$x_{n+1} = x_n - \frac{e(x_n)}{e'(x_n)}$$

Note: Now we only need 1st derivative

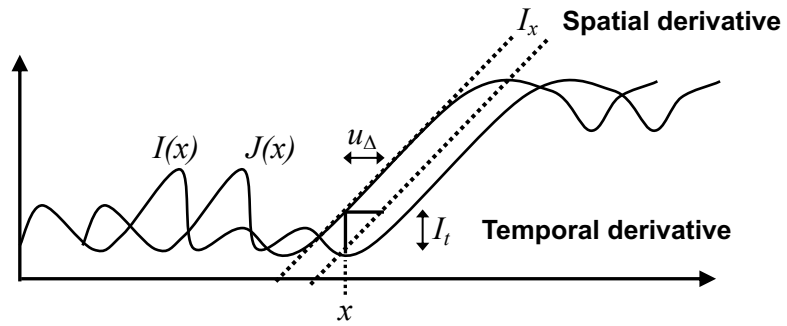
29

1D Lucas-Kanade tracking

2D: $\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = -I_t$

1D: $I_x u_\Delta = -I_t$

$$I_x = \frac{\text{rise}}{\text{run}} = \frac{-I_t}{u_\Delta}$$

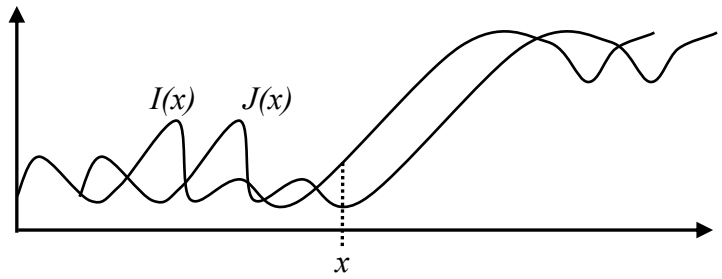


Incremental displacement: $u_\Delta = -I_t / I_x$

30

1D Lucas-Kanade tracking

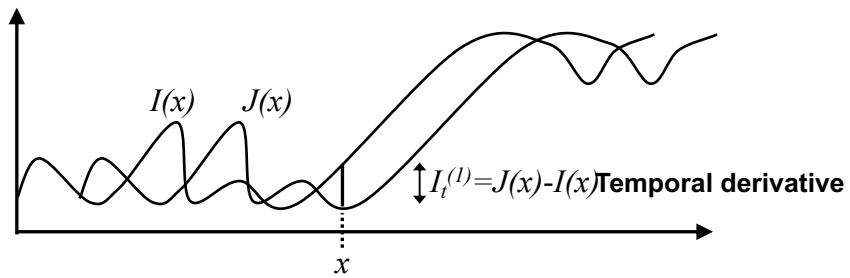
First iteration: Given two images, and position x



31

1D Lucas-Kanade tracking

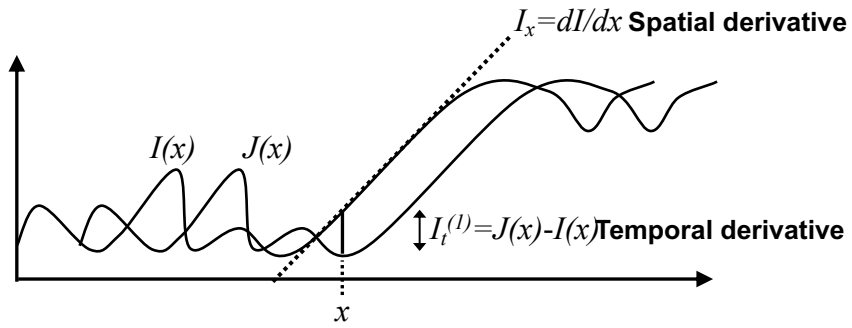
First iteration: Compute temporal derivative



32

1D Lucas-Kanade tracking

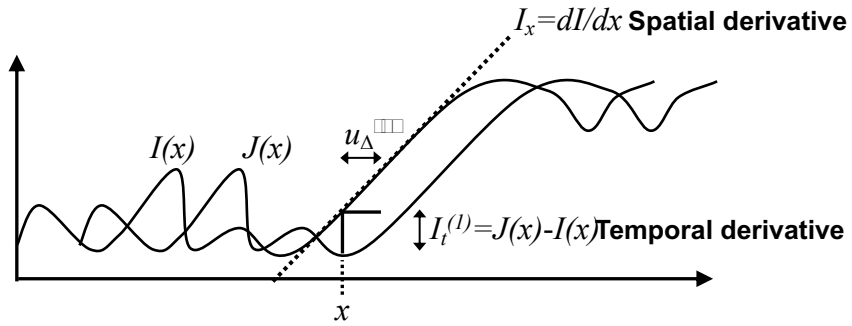
First iteration: Compute spatial derivative



33

1D Lucas-Kanade tracking

First iteration: Compute incremental displacement



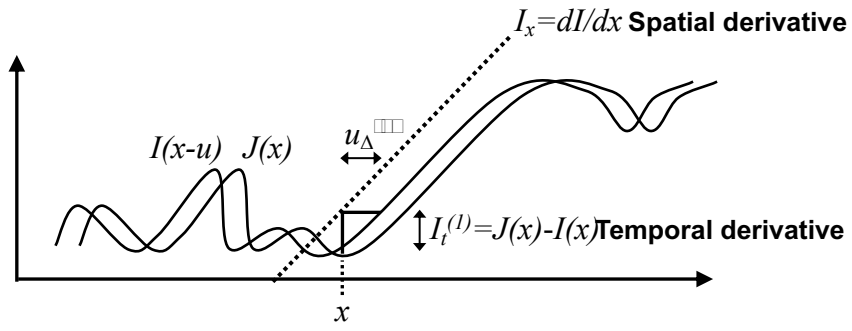
Compute incremental displacement: $u_\Delta^{(1)} = -I_t^{(1)}/I_x$

Total displacement: $u = u_\Delta^{(1)}$

34

1D Lucas-Kanade tracking

First iteration: Shift *I(x)* to the right by u_Δ … or

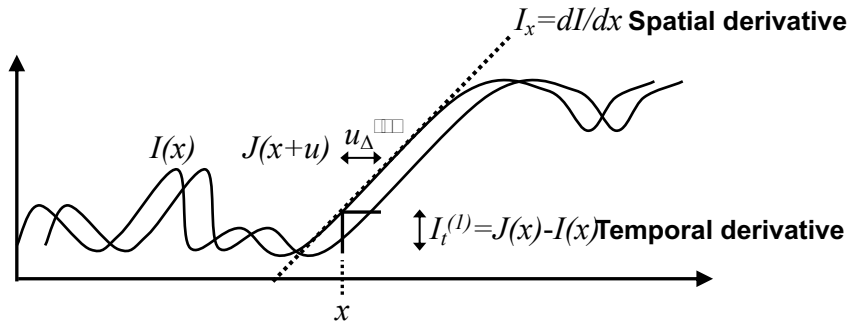


Compute incremental displacement: $u_\Delta^{(1)} = -I_t^{(1)}/I_x$ **Total displacement:** $u = u_\Delta^{(1)}$

35

1D Lucas-Kanade tracking

First iteration: Shift *J(x)* to the left by u_Δ (equivalent)

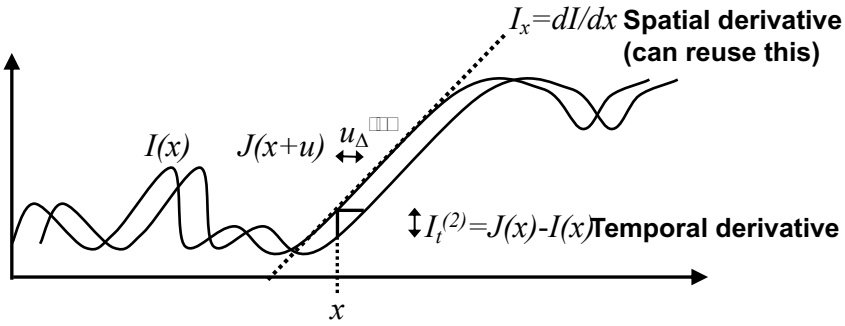


Compute incremental displacement: $u_\Delta^{(1)} = -I_t^{(1)}/I_x$ **Total displacement:** $u = u_\Delta^{(1)}$

36

1D Lucas-Kanade tracking

Second iteration: Repeat same steps



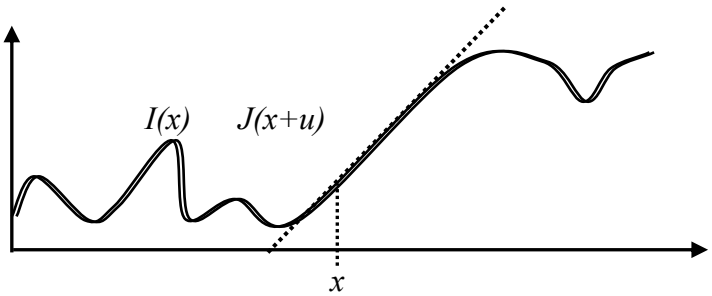
Compute incremental displacement: $u_\Delta^{(2)} = -I_t^{(2)}/I_x$

Total displacement. $u = u_\Delta^{(1)} + u_\Delta^{(2)}$

37

1D Lucas-Kanade tracking

… until convergence



Total displacement: $u = u_\Delta^{(1)} + u_\Delta^{(2)} + \cdots$

38

2D Lucas-Kanade in a nutshell

2D: $\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = -I_t$ **1D:** $I_x u_\Delta = -I_t$

Instead of solving for u_Δ we need to solve for u_Δ□(u_Δ,v_Δ□

Given two consecutive images I and J from a sequence,

- **Take pixel-wise difference between them → I_t**
- **Compute gradient of one image → I_x and I_y**
- **Sum over window to get Z and e**
- **Solve Zu_Δ=e for u_Δ**
- **Update u= u_Δ**
- **Shift image**
- **Repeat**

(2x2 matrix) (2x1 vector)

What are Z and e?
Why do we need to sum over window?

39

Lucas-Kanade derivation

Scalar equation with two unknowns is an underconstrained system:

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = -I_t$$

Introduce constraint. Assume neighboring pixels have *same* motion:

window around pixel

$$\underbrace{\begin{bmatrix} I_x(\mathbf{x}_1) & I_y(\mathbf{x}_1) \\ I_x(\mathbf{x}_2) & I_y(\mathbf{x}_2) \\ \vdots & \vdots \\ I_x(\mathbf{x}_n) & I_y(\mathbf{x}_n) \end{bmatrix}}_{A} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = -\underbrace{\begin{bmatrix} I_t(\mathbf{x}_1) \\ I_t(\mathbf{x}_2) \\ \vdots \\ I_t(\mathbf{x}_n) \end{bmatrix}}_{\mathbf{b}} \qquad \mathbf{x} \equiv \begin{bmatrix} x \\ y \end{bmatrix}$$

n = number of pixels in window

or

$$A\mathbf{u}_\Delta = \mathbf{b}$$

Can solve this directly using least squares, or (better yet) …

40

Lucas-Kanade derivation

Multiply by A^T :

$$\underbrace{\begin{bmatrix} I_x(\mathbf{x}_1) & \cdots & I_x(\mathbf{x}_n) \\ I_y(\mathbf{x}_1) & \cdots & I_y(\mathbf{x}_n) \end{bmatrix}}_{A^T} \underbrace{\begin{bmatrix} I_x(\mathbf{x}_1) & I_y(\mathbf{x}_1) \\ \vdots & \vdots \\ I_x(\mathbf{x}_n) & I_y(\mathbf{x}_n) \end{bmatrix}}_{A} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = - \underbrace{\begin{bmatrix} I_x(\mathbf{x}_1) & \cdots & I_x(\mathbf{x}_n) \\ I_y(\mathbf{x}_1) & \cdots & I_y(\mathbf{x}_n) \end{bmatrix}}_{A^T} \begin{bmatrix} I_t(\mathbf{x}_1) \\ I_t(\mathbf{x}_2) \\ \vdots \\ I_t(\mathbf{x}_n) \end{bmatrix}$$

or

$$\underbrace{\sum_{\mathbf{x}\in\mathcal{R}} \begin{bmatrix} I_x^2(\mathbf{x}) & I_x(\mathbf{x})I_y(\mathbf{x}) \\ I_x(\mathbf{x})I_y(\mathbf{x}) & I_y^2(\mathbf{x}) \end{bmatrix}}_{Z} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = - \underbrace{\sum_{\mathbf{x}\in\mathcal{R}} \begin{bmatrix} I_x(\mathbf{x})I_t(\mathbf{x}) \\ I_y(\mathbf{x})I_t(\mathbf{x}) \end{bmatrix}}_{\mathbf{e}}$$

(2x2 matrix) **(2x1 vector)**

$$\mathcal{R} = \{\mathbf{x}_i\}_{i=1}^n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

(region of neighboring pixels)

$$A^T A \mathbf{u}_\Delta = A^T \mathbf{b}$$

or

$$Z\mathbf{u}_\Delta = \mathbf{e}$$

41

RGB version of Lucas-Kanade

- **For color images, we can get more equations by using all color channels**
- **E.g., for 7x7 window we have 49*3=147 equations!**

$$\sum_{c\in\{R,G,B\}} \sum_{i=1}^{n} \begin{bmatrix} I_x^{(c)}(\mathbf{x}_i)I_x^{(c)}(\mathbf{x}_i) & I_x^{(c)}(\mathbf{x}_i)I_y^{(c)}(\mathbf{x}_i) \\ I_x^{(c)}(\mathbf{x}_i)I_y^{(c)}(\mathbf{x}_i) & I_y^{(c)}(\mathbf{x}_i)I_y^{(c)}(\mathbf{x}_i) \end{bmatrix} \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix}$$
$$= - \sum_{c\in\{R,G,B\}} \sum_{i=1}^{n} \begin{bmatrix} I_x^{(c)}(\mathbf{x}_i)I_t^{(c)}(\mathbf{x}_i) \\ I_y^{(c)}(\mathbf{x}_i)I_t^{(c)}(\mathbf{x}_i) \end{bmatrix}$$

- **But color channels are highly correlated, so rarely helps much**

42

Lucas-Kanade pseudocode

Typically *R* is a pixel x=(*x,y*) and the width of the square window

LucasKanade($I, J, \mathcal{R}$)

Input: 2D images I and J, and region $\mathcal{R}$
Output: the 2D displacement $\mathbf{u}$ of the pixel represented by $\mathcal{R}$

```
1  u ← 0
2  iter ← 0
3  G_x, G_y ← Gradient(I)
4  repeat
5        Z ← Compute2x2GradientMatrix(G_x, G_y, R)
6        e ← Compute2x1ErrorVector(I, J, G_x, G_y, R, u)
7        u_Δ ← Solve2x2LinearSystem(Z, e)
8        u ← u + u_Δ
9        iter ← iter +1
10 until ||u_Δ|| < th OR iter ≥ max-iter
11 return u
```

OK to move this out of loop

Compute2x2GradientMatrix($G_x, G_y, \mathcal{R}$)

```
1  for (x, y) ∈ R do
2        Z_xx ← Z_xx + G_x(x, y) * G_x(x, y)
3        Z_xy ← Z_xy + G_x(x, y) * G_y(x, y)
4        Z_yy ← Z_yy + G_y(x, y) * G_y(x, y)
5  return Z_xx, Z_xy, Z_yy
```

$$Z = \sum_{\mathbf{x} \in \mathcal{R}} (\nabla I(\mathbf{x})) (\nabla I(\mathbf{x}))^T$$

can *x* and *y* be floats?

Compute2x1ErrorVector($I, J, G_x, G_y, \mathcal{R}, \mathbf{u}$)

```
1  for (x, y) ∈ R do
2        e_x ← e_x + G_x(x, y) * (I(x, y) − J(x + u, y + v))
3        e_y ← e_y + G_y(x, y) * (I(x, y) − J(x + u, y + v))
4  return e_x, e_y
```

$$\mathbf{e} = - \sum_{\mathbf{x} \in \mathcal{R}} (\nabla I(\mathbf{x})) I_t(\mathbf{x})$$

can *x* and *y* be floats?

Solve2x2LinearSystem($Z, \mathbf{e}$)

```
1  det ← Z_xx * Z_yy − Z_xy * Z_xy
2  u_Δ ← (1/det) * (Z_yy * e_x − Z_xy * e_y)
3  v_Δ ← (1/det) * (Z_xx * e_y − Z_xy * e_x)
4  return u_Δ, v_Δ
```

$$\mathbf{u}_\Delta = \begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix} = \frac{1}{\det(Z)} \begin{bmatrix} Z_{yy} e_x - Z_{xy} e_y \\ Z_{xx} e_y - Z_{xy} e_x \end{bmatrix}$$

(solving 2x2 equation is easy: invert Z by hand)

Lucas-Kanade details

Need to

1. loop over frames and features
2. interpolate
3. declare features lost
4. detect good features
5. smooth the image to handle large motions

45

1. Loop over frames and features

- We have only described the procedure for
  - a single pixel (using its surrounding window)
  - two consecutive image frames
- Need to wrap this in 2 for loops
  - over frames in the sequence
  - over features (sparse pixels) in the image

46

Loop over frames and features

features is 2D array over features and frames

LucasKanadeSequence($I_{0:m}$, *window-width*)

Input: sequence of 2D images $I_0, I_1, \ldots, I_m$ and width of square feature window
Output: 2D positions of all features in all frames

```
1  features(:, 0) ← SelectGoodFeatures(I_0)
2  for i ← 1 to m do
3      for j ← 1 to Length(features) do
4          u ← LucasKanade(I_{i-1}, I_i, SquareRegion(features(j, i − 1), window-width))
5          features(j, i) ← features(j, i − 1) + u
6  return features
```

float

colon selects entire row (like Matlab)

float

float

(Conceptually) constructs a region from pixel and *window-width*
(In reality) does nothing; just pass both parameters to LucasKanade

47

2. Interpolate

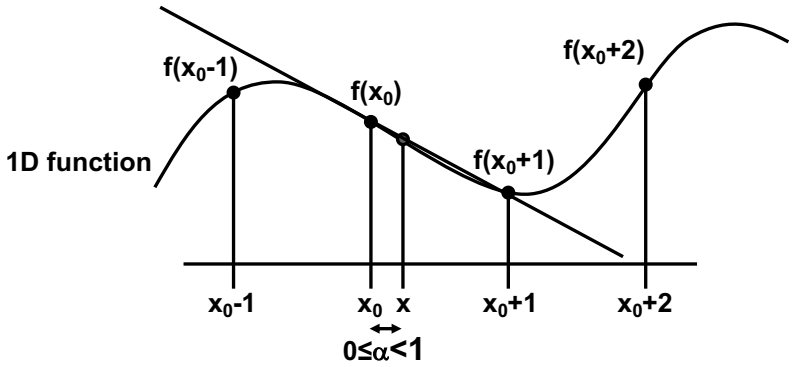
- After very first iteration, x and y can be floats; same for u and v
- Could simply round to nearest integer before computing

$$I(x, y) \quad J(x + u, y + v) \quad G_x(x, y) \quad G_y(x, y)$$

- But results are more accurate if we interpolate

48

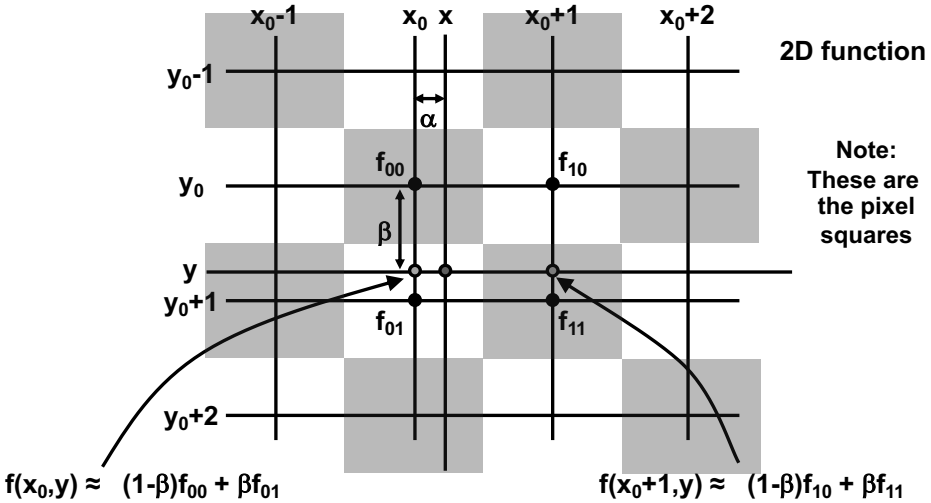
Linear interpolation



$$f(x) \approx f(x_0) + (x-x_0)\,(f(x_0+1) - f(x_0))$$
$$= f(x_0) + \alpha\,(f(x_0+1) - f(x_0))$$
$$= (1-\alpha)\,f(x_0) + \alpha\,f(x_0+1)$$

49

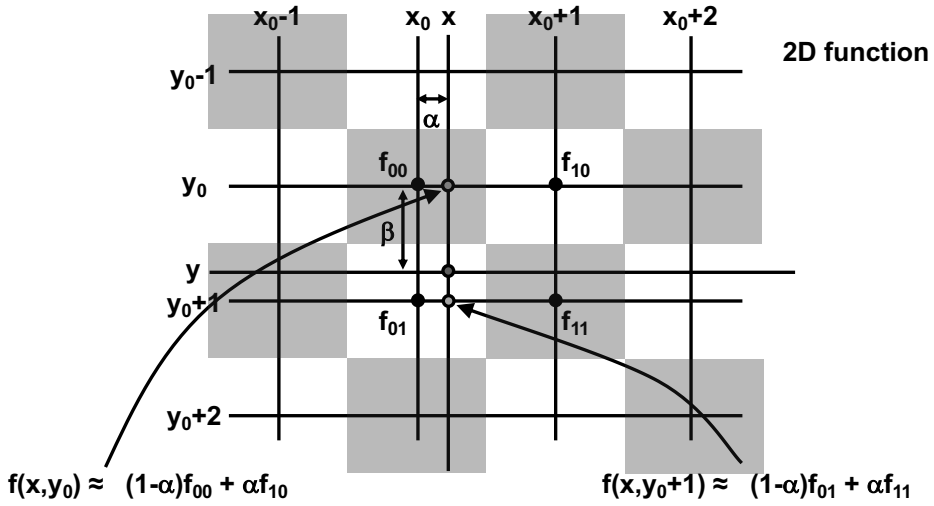
Bilinear interpolation



$$f(x,y) \approx (1-\alpha)\,[(1-\beta)f_{00} + \beta f_{01}] + \alpha[(1-\beta)f_{10} + \beta f_{11}]$$
$$= (1-\alpha)(1-\beta)f_{00} + (1-\alpha)\beta f_{01} + \alpha(1-\beta)f_{10} + \alpha\beta f_{11}$$

50

Bilinear interpolation



51

Bilinear interpolation

- **Simply compute double weighted average of four nearest pixels**
- **Be careful to handle boundaries correctly (details omitted here)**

INTERPOLATEBILINEAR(I, x, y)

Input: graylevel image I, floating-point pixel coordinates (x, y)
Output: bilinearly interpolated pixel value

$x_0 \leftarrow \lfloor x \rfloor$
$y_0 \leftarrow \lfloor y \rfloor$
$\alpha_x \leftarrow x - x_0$
$\alpha_y \leftarrow y - y_0$
return $(1-\alpha_x)*(1-\alpha_y) * I(x_0, y_0) + \alpha_x*(1-\alpha_y) * I(x_0+1, y_0) +$
$(1-\alpha_x)*\alpha_y * I(x_0, y_0+1) + \alpha_x*\alpha_y * I(x_0+1, y_0+1)$

S. Birchfield, lemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

52

3. Declare features lost

- **Problem: If feature changes appearance too much, it has probably drifted onto another (occluding) surface in the scene**
- **Solution: Check the residue by computing SSD of feature window in both I and J**
- **If residue is too high, then declare feature lost, and do not continue to track**

53

4. Detect good features

- **Lucas-Kanade could be used to compute motion for every pixel in the image**
- **But**
  - **Motion of nearby pixels are similar**
  - **Pixel motion cannot be computed in areas of little (or no) texture**
- **Solution: First detect good features (pixel windows), then only track those features**

54

What is a good feature?

- To solve this equation, Z= A^TA must be invertible:

$$\underbrace{\sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}}_{Z} \underbrace{\begin{bmatrix} u_\Delta \\ v_\Delta \end{bmatrix}}_{\mathbf{u}_\Delta} = \underbrace{-\sum \begin{bmatrix} I_xI_t \\ I_yI_t \end{bmatrix}}_{\mathbf{e}}$$

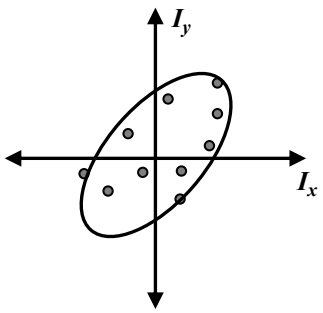
- In real world, all matrices are invertible
- Instead, we must ensure that Z=A^TA is *well-conditioned* (not close to singular)

55

Eigenvalues of Z

$$Z = \sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

- Z is *gradient covariance matrix*
  - Same matrix as Harris operator
  - Also called Hessian
- Recall from PCA:
  - (Square root of) eigenvalues give length of best fitting ellipse
  - Large eigenvalues means large gradient vectors
  - Small ratio means information in all directions
- We want:
  - Both eigenvalues λ_1 and λ_2 are large
  - Ratio of eigenvalues λ_1 / λ_2 is not too large



Note: $\lambda_1 \geq \lambda_2$

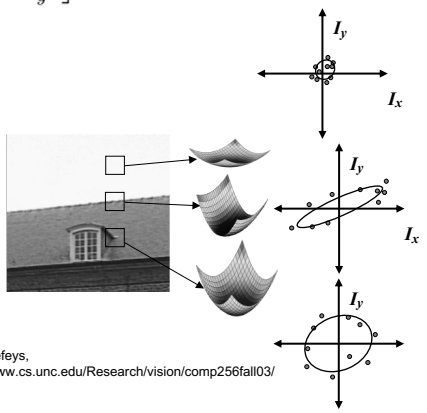
56

Finding good features

$$Z = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Three cases:

- **λ_1 and λ_2 small**
  Not enough texture for tracking
- **λ_1 large, λ_2 small**
  On intensity edge
  Aperture problem: Only motion perpendicular to edge can be found
- **λ_1 and λ_2 large**
  Good feature to track ("corner")



M. Pollefeys, http://www.cs.unc.edu/Research/vision/comp256fall03/

Note: Even though tracking is a two-frame problem, we can determine good features from only one frame

57

Three "cornerness" measures

1. **Threshold minimum eigenvalue (We will use this) (Tomasi-Kanade 1991)**
2. **Use trace and determinant (Harris and Stephens 1988)**

   known as Harris corner detector or Plessey operator
3. **Treat like resistors in parallel**
   $\det(Z) / \text{trace}(Z) = 1 / (1/\lambda_1 + 1/\lambda_2)$

All 3 perform about the same

58

Tomasi-Kanade cornerness

- **In practice, eigenvalues cannot be arbitrarily large, b/c image values range from [0,255]**
- **Therefore, we can ignore maximum eigenvalue**
- **This is the simplest approach**

$$\text{cornerness} = \min\{\lambda_1, \lambda_2\} = \lambda_2$$

Recall:

$$\lambda_{1,2} = \frac{1}{2}\left[(g_{xx} + g_{yy}) \pm \sqrt{(g_{xx} - g_{yy})^2 + 4g_{xy}^2}\right] \qquad Z = \begin{bmatrix} g_{xx} & g_{xy} \\ g_{xy} & g_{yy} \end{bmatrix}$$

59

Harris corner detector

- **Easy to show that**

$$\det(Z) = \lambda_1\lambda_2 \qquad \text{trace}(Z) = \lambda_1 + \lambda_2$$

- **Use**

$$\begin{aligned} \text{cornerness} &= \det(Z) - k\,(\text{trace}(Z))^2 \\ &= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \end{aligned}$$

recommended: ***k = 0.04***

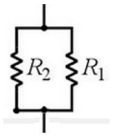
- **Advantage: No need to actually compute eigenvalues (no square root)**
- **The second term reduces effect of having a small eigenvalue with a large dominant eigenvalue**

60

Resistors in parallel

- **Alternative is to treat eigenvalues like resistors in parallel**

$$\frac{1}{\text{cornerness}} = \frac{\text{trace}(Z)}{\det(Z)} = \frac{\lambda_1 + \lambda_2}{\lambda_1 \lambda_2} = \frac{1}{\lambda_1} + \frac{1}{\lambda_2}$$

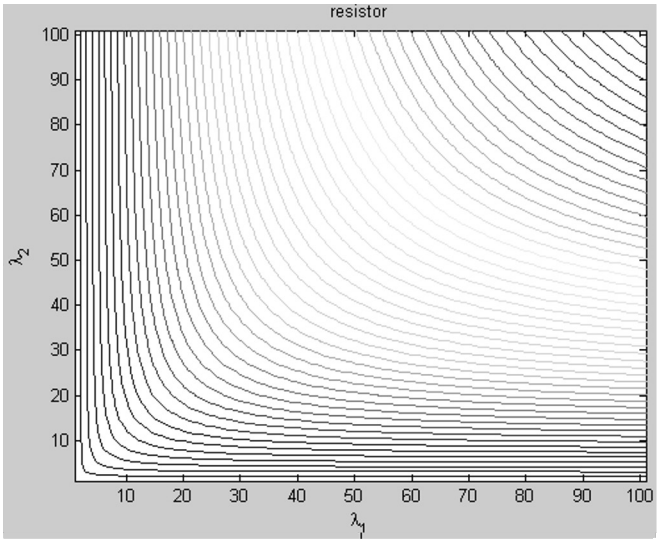


- **Combined resistance is bound by individual resistances**

$$\frac{\lambda_{\min}}{2} \leq \frac{\det(Z)}{\text{trace}(Z)} \leq \lambda_{\min}$$

61

Comparison of "cornerness"



Conclusion: They are all very similar

62

Finding good features

To find good features,

- **scan image**
- **evaluate "cornerness" measure at each pixel**
- **threshold**
- **perform non-maximal suppression**

3x3 window is all you need

```
DetectGoodFeatures(I, threshold)
 Input:  graylevel image I
 Output: 2D coordinates of all pixels with locally maximal texture above threshold
1  G_x, G_y ← Gradient(I)
2  for x ∈ I do
3        Z ← Compute2x2GradientMatrix(G_x, G_y, SquareRegion(x, 3))
4        λ_1, λ_2 ← Eigen(Z)
5        cornerness(x) ← if λ_2 ≥ threshold then λ_2 otherwise 0
6  cornerness ← NonMaxSuppression(cornerness)
7  return {x : cornerness(x) ≠ 0}
```

Tomasi-Kanade

(can also enforce minimum distance between features)

63

Flashback: Moravec interest operator

- **Moravec (1970s) proposed to measure "cornerness" by measuring SSD of window after shifting right, left, down, and up:**

$$\text{cornerness} = \min\{\epsilon_{\mathcal{R}}(1,0), \epsilon_{\mathcal{R}}(-1,0), \epsilon_{\mathcal{R}}(0,1), \epsilon_{\mathcal{R}}(0,-1)\}$$

$$\epsilon_{\mathcal{R}}(\Delta\mathbf{x}) = \sum_{\mathbf{x}\in\mathcal{R}} [I(\mathbf{x}) - I(\mathbf{x}+\Delta\mathbf{x})]^2$$

… but this is not rotationally symmetric

- **Expand using Taylor series and substitute**

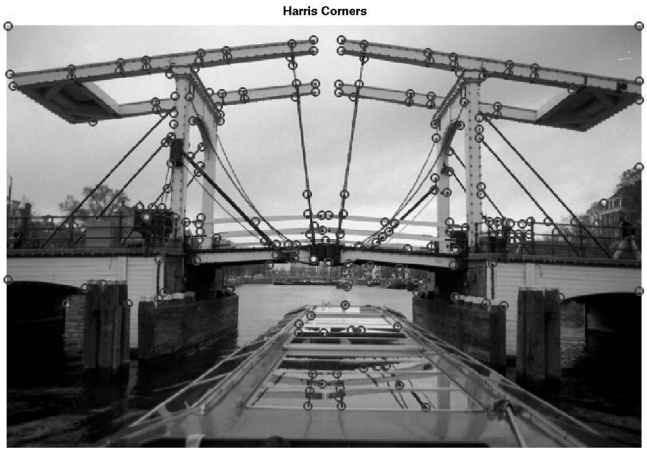
$$I(\mathbf{x}+\Delta\mathbf{x}) \approx I(\mathbf{x}) + (\Delta\mathbf{x})^T \frac{\partial I}{\partial \mathbf{x}}$$

$$\begin{aligned}\epsilon_{\mathcal{R}}(\Delta\mathbf{x}) &= \sum_{\mathbf{x}\in\mathcal{R}} \left[(\Delta\mathbf{x})^T \frac{\partial I}{\partial \mathbf{x}}\right]^2 \\ &= \sum_{\mathbf{x}\in\mathcal{R}} (\Delta\mathbf{x})^T \left(\frac{\partial I}{\partial \mathbf{x}}\right)\left(\frac{\partial I}{\partial \mathbf{x}}\right)^T (\Delta\mathbf{x}) \\ &= \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} Z \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}\end{aligned}$$

our old friend again

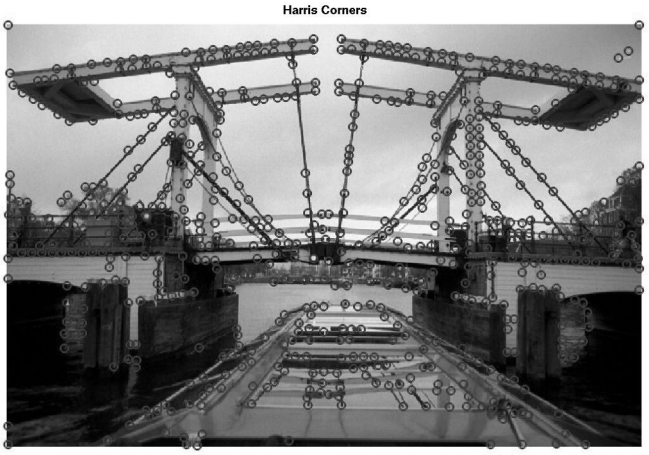
64

Example (σ=0.1)



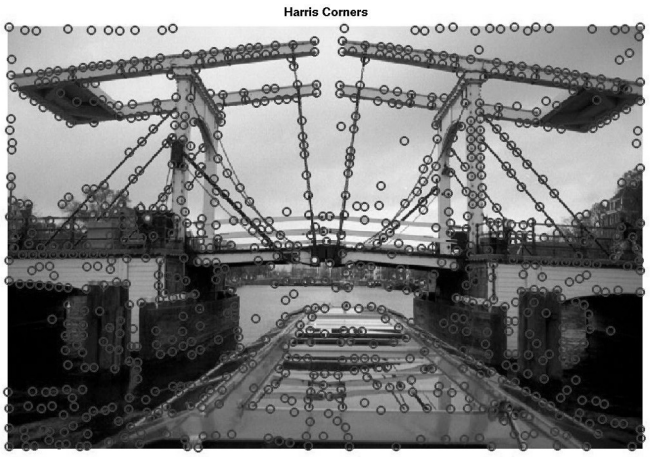
65

Example (σ=0.01)



66

Example (σ=0.001)



67

5. Smooth the image

- Motion between consecutive image frames can be large
- Solution: Smooth the image first
- Note: Smoothing is already in gradient computation, so just use large σ
- But large σ leads to less accuracy
- Solution: Sequentially decrease σ
- Pyramid makes this computationally efficient

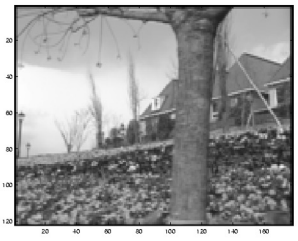
68

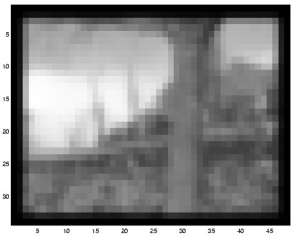
Revisiting the small motion assumption

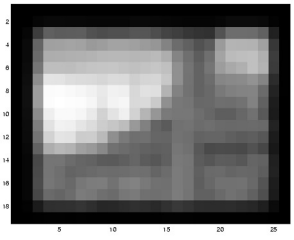
- **Is this motion small enough?**
  - **Probably not—it's much larger than one pixel (2^{nd} order terms dominate)**
  - **How might we solve this problem?**

69

Reduce the resolution!

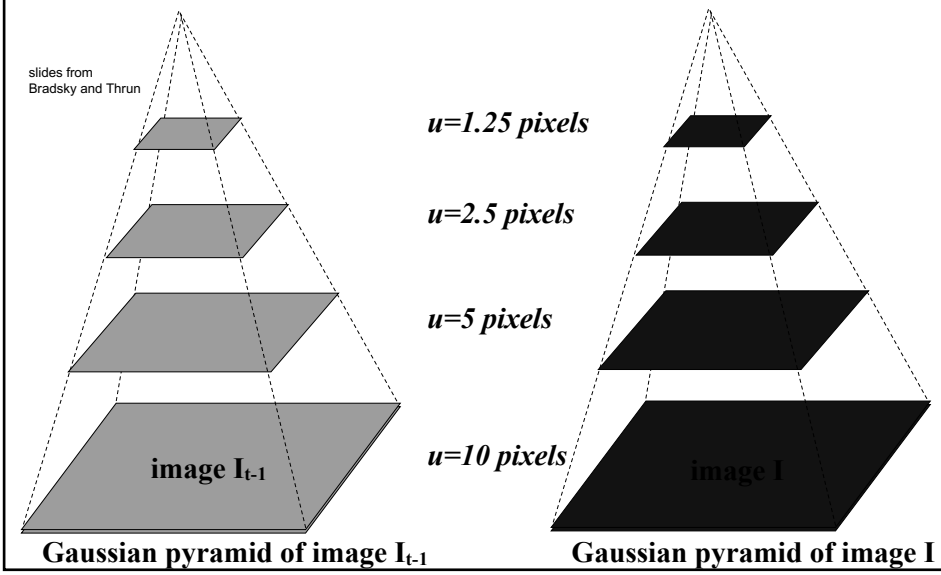






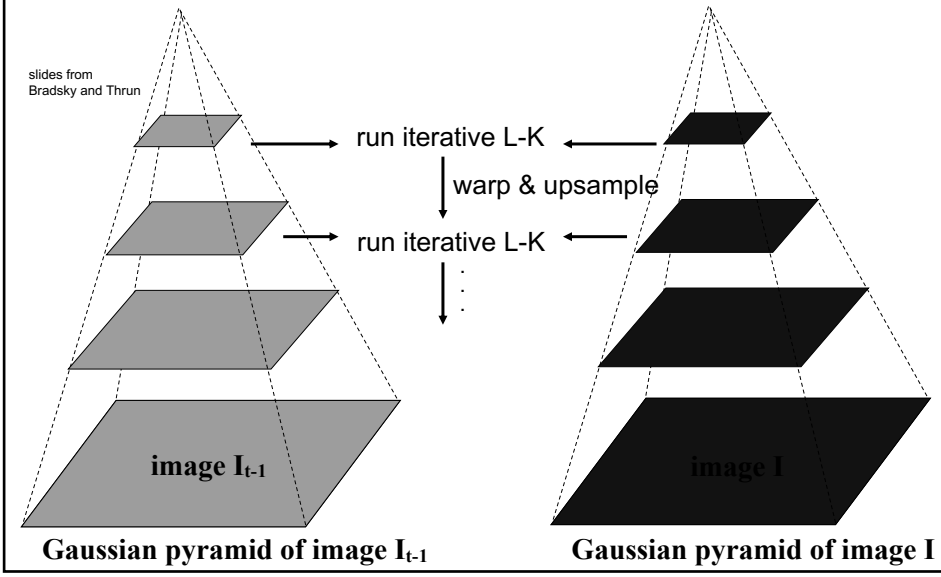
70

Coarse-to-fine optical flow estimation



71

Coarse-to-fine optical flow estimation



72

One final problem

- **Not always easy to tell whether to discard feature from frame-to-frame tracking because distortions may occur gradually over several frames (frog in pot)**
- **Therefore, need to compare feature appearance over longer periods of time**
- **Translation assumption is fine between consecutive frames, but long periods of time introduce other types of deformations**
- **Solution: Use affine model of motion**
- **Lucas-Kanade easily generalizes to other motion models, but requires a different derivation**

73

Alternative derivation

- **Compute translation assuming it is small**

$$\text{argmin}_{\Delta} \iint_W (I + \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix} \Delta - J)^2 w(x,y) \mathrm{d}x\mathrm{d}y$$

differentiate:

$$\iint_W 2 \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} (I + \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix} \Delta - J) w(x,y) \mathrm{d}x\mathrm{d}y = 0$$

$$\left(\iint_W \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix} w(x,y) \mathrm{d}x\mathrm{d}y \right) \Delta = \iint_W \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} (J - I) w(x,y) \mathrm{d}x\mathrm{d}y$$

Affine is also possible, but a bit harder (6x6 instead of 2x2)

$$\text{argmin}_{\Delta, \mathbf{A}} \iint_W (I + \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix} \left(\Delta + \mathbf{A} \begin{bmatrix} x \\ y \end{bmatrix} \right) - J)^2 w(x,y) \mathrm{d}x\mathrm{d}y$$

74

Example

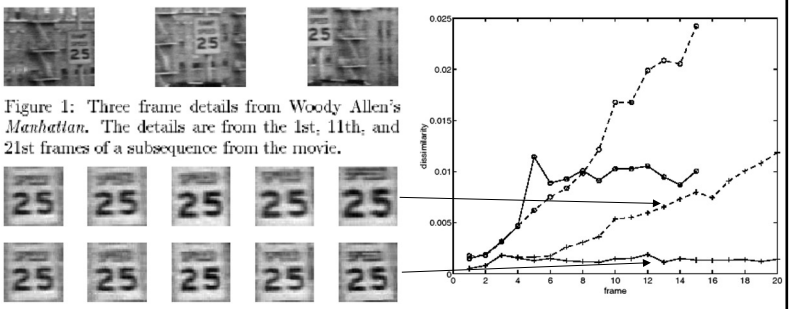


Figure 1: Three frame details from Woody Allen's *Manhattan*. The details are from the 1st, 11th, and 21st frames of a subsequence from the movie.

Simple displacement is sufficient between consecutive frames, but not to compare to reference template

75

Example

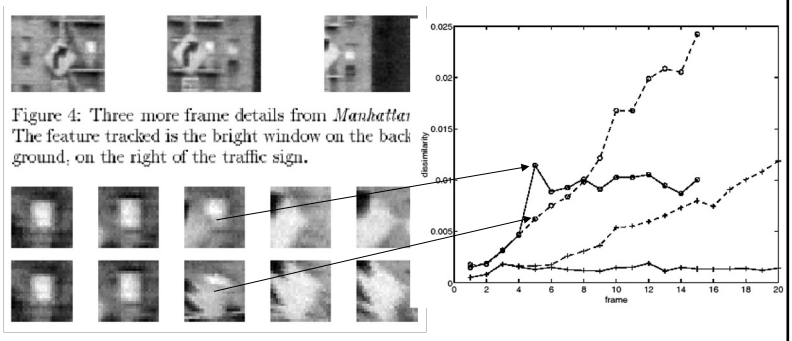
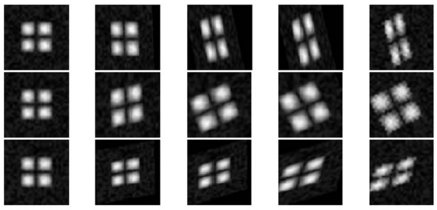
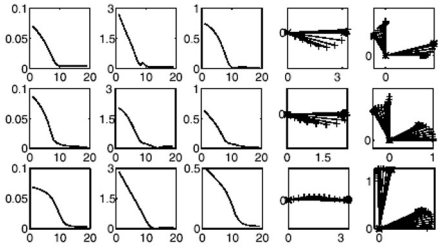


Figure 4: Three more frame details from *Manhattan* The feature tracked is the bright window on the background, on the right of the traffic sign.

76

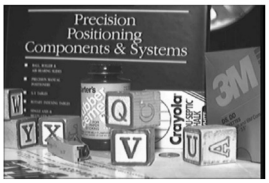
Synthetic example

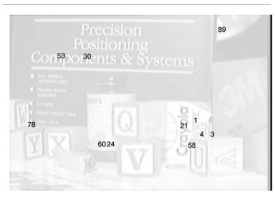




77

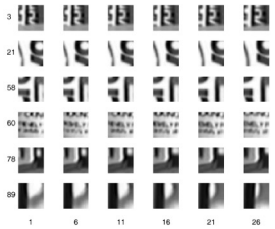
Good features to keep tracking

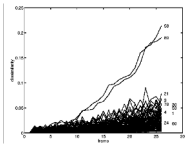


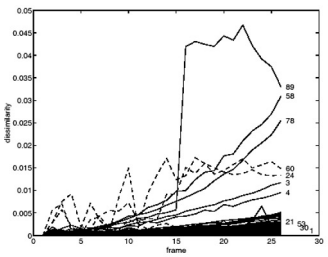


Perform affine alignment between first and last frame

Stop tracking features with too large errors



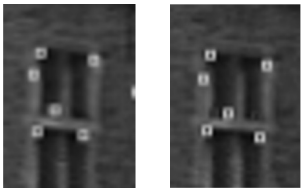




78

Feature matching vs. tracking

Image-to-image correspondences are key to passive triangulation-based 3D reconstruction



Extract features independently and then match by comparing descriptors

Extract features in first images and then try to find same feature back in next view

79

Outline

- **Motion basics**
- **Lucas-Kanade algorithm**
- **Horn-Schunck algorithm**

80

Horn-Schunck

- **Overcome aperture problem by assuming that neighboring pixels have *similar* displacement**
- **Minimize** $\int\int (E_d^2 + \lambda E_s^2) dx dy$

  where

$$E_d = I_x u + I_y v + I_t \quad \textbf{(data)}$$

$$E_s^2 = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \quad \textbf{(smoothness)}$$

(Global approach to compute dense optical flow)

81

Problem

- **Goal is to minimize to find *u* and *v*** $\int\int (E_d^2 + \lambda E_s^2) dx dy$
- **But now *u* and *v* are themselves functions of *x* and *y*:** $u(x,y) \quad v(x,y)$
- **So in reality, we have a function of functions**

$$E_d(u(x,y), v(x,y), x, y) = I_x(x,y)u(x,y) + I_y(x,y)v(x,y) + I_t(x,y)$$

$$E_s^2(u(x,y), v(x,y), x, y) = \left(\frac{\partial u(x,y)}{\partial x}\right)^2 + \left(\frac{\partial u(x,y)}{\partial y}\right)^2 + \left(\frac{\partial v(x,y)}{\partial x}\right)^2 + \left(\frac{\partial v(x,y)}{\partial y}\right)^2$$

- **How to do this?**

82

Calculus of variations

- A *function* maps values to real numbers
  - To find the value that minimizes a function, use *calculus*
- A *functional* maps functions to real numbers
  - To find the function that minimizes a functional, use *calculus of variations*

83

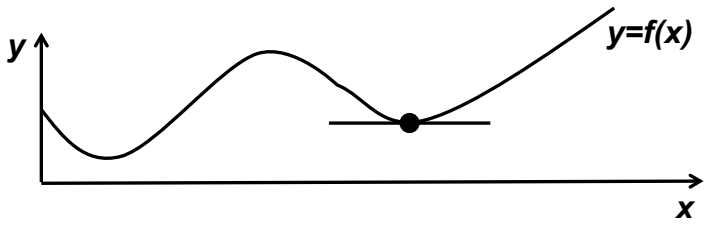
"The calculus"

The *function*

$y = f(x)$

is *stationary* where

$df(x)/dx = 0$



84

Calculus of variations

The integral of the *functional*

$$\int_a^b F(q, q', x)\,dx$$

explicit function ("Lagrangian") $F(q, q', x)$

dependent variable (position, voltage, …) $q = f(x)$

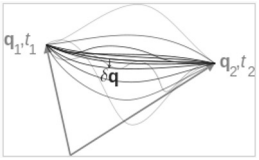
$q' \equiv \frac{dq}{dx}$

independent variable (space or time) x

is *stationary* where

$$\frac{\partial F}{\partial q} - \frac{d}{dx}\left(\frac{\partial F}{\partial q'}\right) = 0$$

Euler-Lagrange equations



"principle of least action"
F = KE – PE
(brachistochrone problem)

85

Calculus of variations

General case

$$F\left(q_1, \ldots, q_n, \frac{\partial q_1}{\partial x_1}, \ldots, \frac{\partial q_n}{\partial x_1}, \ldots, \frac{\partial q_1}{\partial x_m}, \frac{\partial q_n}{\partial x_m}, x_1, \ldots, x_m\right)$$

***n* dependent variables** ***m* independent variables**

$$\frac{\partial F}{\partial q_k} - \sum_{i=1}^{m} \frac{\partial}{\partial x_i}\left(\frac{\partial F}{\partial\left(\frac{\partial q_k}{\partial x_i}\right)}\right) = 0 \qquad k = 1, \ldots, n$$

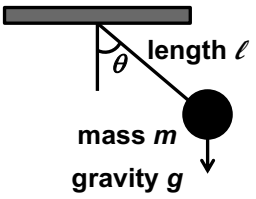
Euler-Lagrange equations

independent variables → multiple terms
dependent variables → multiple equations

86

COV applied to pendulum

Newtonian approach



Use Newton's 2nd Law:

F = ma
-mg sin θ = mℓθ"
-g sin θ = ℓθ"
g sin θ + ℓθ" = 0

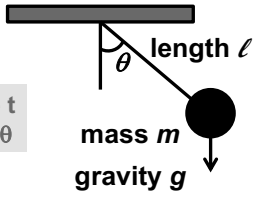
arclength = ℓθ
velocity = ℓθ'
acceleration = ℓθ"

θ=0 → θ"=0
θ=π/2 → ℓθ"=-g
θ=-π/2 → ℓθ"=g

87

COV applied to pendulum

Euler-Lagrange approach



1) Formulate Lagrangian:

indep. variable: time t
dep. variable: angle θ

$$\begin{aligned} F(\theta,\theta',t) &= \mathrm{KE}-\mathrm{PE} \\ &= \frac{1}{2}mv^2 - mgh \\ &= \frac{1}{2}m(\ell\theta')^2 - mg(\ell-\ell\cos\theta) \end{aligned}$$

2) Take derivatives:

$$\frac{\partial F}{\partial \theta} = -mg\ell\sin\theta \qquad \frac{\partial F}{\partial \theta'} = m\ell^2\theta' \qquad \frac{d}{dt}\left(\frac{\partial F}{\partial \theta'}\right) = m\ell^2\theta''$$

3) Plug into Euler-Lagrange equations:

$$\begin{aligned} \frac{\partial F}{\partial \theta} - \frac{d}{dt}\left(\frac{\partial F}{\partial \theta'}\right) &= 0 \\ -mg\ell\sin\theta - m\ell^2\theta'' &= 0 \\ g\sin\theta + \ell\theta'' &= 0 \end{aligned}$$

← **Same result!**

arclength = ℓθ
velocity = ℓθ'
acceleration = ℓθ"

θ=0 → θ"=0
θ=π/2 → ℓθ"=-g
θ=-π/2 → ℓθ"=g

88

COV applied to optical flow

$$F = E_d^2 + \lambda E_s^2$$

explicit function

$$x_1 = x \qquad x_2 = y$$

independent variables

$$q_1 = u = \partial x / \partial t \qquad q_2 = v = \partial y / \partial t$$

dependent variables

$$\frac{\partial F}{\partial u} - \frac{\partial}{\partial x}\left[\frac{\partial F}{\partial\left(\frac{\partial u}{\partial x}\right)}\right] - \frac{\partial}{\partial y}\left[\frac{\partial F}{\partial\left(\frac{\partial u}{\partial y}\right)}\right] = 0$$

$$\frac{\partial F}{\partial v} - \frac{\partial}{\partial x}\left[\frac{\partial F}{\partial\left(\frac{\partial v}{\partial x}\right)}\right] - \frac{\partial}{\partial y}\left[\frac{\partial F}{\partial\left(\frac{\partial v}{\partial y}\right)}\right] = 0$$

Euler-Lagrange equations

89

COV applied to optical flow

Expand:

$$\begin{aligned} F &= E_d^2 + \lambda E_s^2 \\ &= I_x^2 u^2 + I_y^2 v^2 + I_t^2 + 2I_x I_y uv + 2I_x I_t u + 2I_y I_t v \\ &\quad + \lambda\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2\right] \end{aligned}$$

Take derivatives:

$$\frac{\partial F}{\partial u} = 2I_x^2 u + 2I_x I_y v + 2I_x I_t$$

$$\frac{\partial F}{\partial v} = 2I_y^2 v + 2I_x I_y u + 2I_y I_t$$

$$\frac{\partial}{\partial x}\left[\frac{\partial F}{\partial\left(\frac{\partial u}{\partial x}\right)}\right] = \frac{\partial}{\partial x}\left[2\lambda\left(\frac{\partial u}{\partial x}\right)\right] = 2\lambda\left(\frac{\partial^2 u}{\partial x^2}\right)$$

$$\frac{\partial}{\partial y}\left[\frac{\partial F}{\partial\left(\frac{\partial u}{\partial y}\right)}\right] = \frac{\partial}{\partial y}\left[2\lambda\left(\frac{\partial u}{\partial y}\right)\right] = 2\lambda\left(\frac{\partial^2 u}{\partial y^2}\right)$$

$$\frac{\partial}{\partial x}\left[\frac{\partial F}{\partial\left(\frac{\partial v}{\partial x}\right)}\right] = \frac{\partial}{\partial x}\left[2\lambda\left(\frac{\partial v}{\partial x}\right)\right] = 2\lambda\left(\frac{\partial^2 v}{\partial x^2}\right)$$

$$\frac{\partial}{\partial y}\left[\frac{\partial F}{\partial\left(\frac{\partial v}{\partial y}\right)}\right] = \frac{\partial}{\partial y}\left[2\lambda\left(\frac{\partial v}{\partial y}\right)\right] = 2\lambda\left(\frac{\partial^2 v}{\partial y^2}\right)$$

90

COV applied to optical flow

Plug back in:

$$\begin{aligned} I_x^2 u + I_x I_y v + I_x I_t - \lambda \nabla^2 u &= 0 \\ I_y^2 v + I_x I_y u + I_y I_t - \lambda \nabla^2 v &= 0 \end{aligned}$$

where

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$
$$\nabla^2 v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}$$

Laplacian

Rearrange:

$$\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \lambda \nabla^2 u - I_x I_t \\ \lambda \nabla^2 v - I_y I_t \end{bmatrix}$$

Approximate:

$$\begin{bmatrix} I_x^2 + \lambda & I_x I_y \\ I_x I_y & I_y^2 + \lambda \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \lambda \bar{u} - I_x I_t \\ \lambda \bar{v} - I_y I_t \end{bmatrix}$$

where $\nabla^2 u \approx h(\bar{u} - u)$ (difference of Gaussians)

constant mean

What if λ=0?

91

COV applied to optical flow

Lucas-Kanade solves 2N x 2N system as N 2x2 systems

$$\begin{bmatrix} Z_1 & 0 & 0 & \cdots & 0 \\ 0 & Z_2 & 0 & \cdots & 0 \\ 0 & 0 & Z_3 & \cdots & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & \cdots & Z_N \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_N \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \\ \vdots \\ \mathbf{e}_N \end{bmatrix}$$

$$\mathbf{u}_i = Z_i^{-1} \mathbf{e}_i$$

Horn-Schunck solves 2N x 2N system as sparse linear system

$$\begin{bmatrix} Z_1' & \lambda' & 0 & \cdots & 0 \\ \lambda' & Z_2' & \lambda' & \cdots & 0 \\ 0 & \lambda' & Z_3' & \cdots & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & \cdots & Z_N' \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_N \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1' \\ \mathbf{e}_2' \\ \mathbf{e}_3' \\ \vdots \\ \mathbf{e}_N' \end{bmatrix}$$

$$\mathbf{u}_i = (Z_i')^{-1} \left(\mathbf{e}_i' - \lambda' \sum_{\mathbf{u} \in \mathcal{N}(\mathbf{u}_i)} \mathbf{u} \right)$$

$$Z' \equiv Z + \lambda I_2$$
$$\lambda' \equiv \frac{\lambda}{4} I_2$$
$$\mathbf{e}' \equiv \mathbf{e} + \lambda \bar{\mathbf{u}}$$

92

COV applied to optical flow

Solve equation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \bar{u} \\ \bar{v} \end{bmatrix} - \frac{I_x \bar{u} + I_y \bar{v} + I_t}{\lambda + I_x^2 + I_y^2} \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

This is a pair of equations for each pixel. The combined system of equations is 2N x 2N, where N is the number of pixels in the image.

Because it is a sparse matrix, iterative methods (Jacobi iterations, Gauss-Seidel, successive over-relaxation) are more appropriate.

Iterate:

$$\begin{aligned} u_{ij}^{(k+1)} &= \bar{u}_{ij}^{(k)} - \gamma I_x \\ v_{ij}^{(k+1)} &= \bar{v}_{ij}^{(k)} - \gamma I_y \end{aligned}$$

where $\gamma = \dfrac{I_x \bar{u}_{ij}^k + I_y \bar{v}_{ij}^k + I_t}{\lambda + I_x^2 + I_y^2}$

93

Stationary iterative methods

Problem is to solve sparse linear system:

$$Ax = b \qquad \sum_{j=1}^{n} a_{ij} x_j = b_i \qquad A = D - L - U$$

matrix = (diagonal) – (lower) – (upper)

Jacobi method (simply solve for element, using existing estimates):

$$x_i^k = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{k-1} \right) \qquad x^k = D^{-1}(L+U)x^{k-1} + D^{-1}b$$

Gauss-Seidel is “sloppy Jacobi” (use updates as soon as they are available):

$$x_i^k = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^k - \sum_{j>i} a_{ij} x_j^{k-1} \right) \qquad x^k = (D-L)^{-1}(Ux^{k-1} + b)$$

Successive over relaxation (SOR) speeds convergence:

$$x_i^k = \omega \bar{x}_i^k + (1-\omega) x_i^{k-1}$$

G-S iterate

$$x^k = (D - \omega L)^{-1}(\omega U + (1-\omega)D)x^{k-1} + \omega(D - \omega L)^{-1}b$$

$0 < \omega < 2$ ω=1 means Gauss-Seidel

94

Horn-Schunck Algorithm

HORN-SCHUNCK(I, λ)

1 $I_x, I_y \leftarrow$ GRADIENT(I)
2 **for** $(x, y) \in I$ **do**
3 $\quad u(x, y) \leftarrow 0$
4 $\quad v(x, y) \leftarrow 0$
5 **repeat**
6 $\quad$ **for** $(x, y) \in I$ **do**
7 $\quad\quad \bar{u} \leftarrow \frac{1}{4}(u(x-1, y) + u(x+1, y) + u(x, y-1) + u(x, y+1))$
8 $\quad\quad \bar{v} \leftarrow \frac{1}{4}(v(x-1, y) + v(x+1, y) + v(x, y-1) + v(x, y+1))$
9 $\quad\quad \gamma \leftarrow \frac{I_x(x,y)\bar{u} + I_y(x,y)\bar{v} + I_t(x,y)}{\lambda + I_x(x,y)^2 + I_y(x,y)^2}$
10 $\quad\quad u(x, y) \leftarrow \bar{u} - \gamma I_x(x, y)$
11 $\quad\quad v(x, y) \leftarrow \bar{v} - \gamma I_y(x, y)$
12 **until** convergence

95

Horn-Schunck Algorithm

```
begin
    for j:= 1 to N do  for i := 1 to M do  begin
        calculate the values E_x(i,j,t), E_y(i,j,t), and E_t(i,j,t) using
            a selected approximation formula;
                { special cases for image points at the image border
                                    have to be taken into account }
        initialize the values u(i,j) and v(i,j) with zero
    end {for};
    choose a suitable weighting value λ;                    { e.g λ = 10 }
    choose a suitable number n0 ≥ 1 of iterations;          { n0 = 8 }
    n := 1;                                                 { iteration counter }
    while n ≤ n0 do begin
        for j:= 1 to N do  for i := 1 to M do  begin
            ū := 1/4 (u(i-1,j) + u(i+1,j) + u(i,j-1) + u(i,j+1));
            v̄ := 1/4 (v(i-1,j) + v(i+1,j) + v(i,j-1) + v(i,j+1));
                { treat image points at the image border separately }
            α := (E_x(i,j,t) ū + E_y(i,j,t) v̄ + E_t(i,j,t)) / (1 + λ (E_x²(i,j,t) + E_y²(i,j,t))) · λ ;
            u(i,j) := ū − α · E_x(i,j,t) ;  v(i,j) := v̄ − α · E_y(i,j,t)
        end {for};
        n := n + 1
    end {while}
end;
```

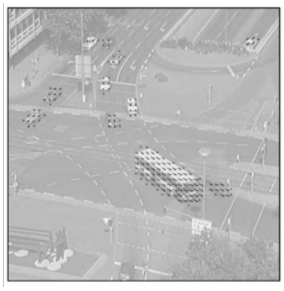
note that this λ is defined differently

96

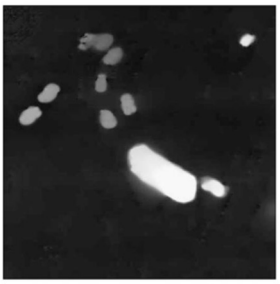
Lucas-Kanade meets Horn-Schunck

- IJCV 2005 paper by Bruhn et al.
- Shows that
  - Local Methods (Lucas-Kanade) and
  - Global Methods (Horn-Schunck)

  complement one another



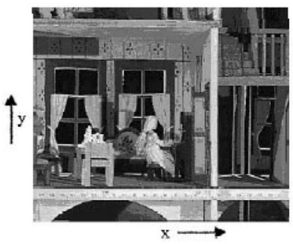
flow field on image

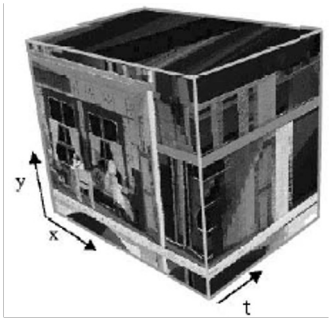


magnitude of flow field

97

Spatiotemporal volumes





98