Least squares and motion

Nuno Vasconcelos

ECE Department, UCSD
Plan for today

- today we will discuss motion estimation
- this is interesting in two ways
  - motion is very useful as a cue for recognition, segmentation, compression, etc.
  - is a great example of least squares problem
- we will also wrap up discussion on least squares
- introduce two types of motion estimation
  - block matching
  - differential methods
- will talk about motion ambiguities, and local vs global motion
Least squares

- **a least squares problem** is one where we have
  - two variables \((X, Y)\) related by an unknown function \(Y = g(X)\)
  - a training set \(D = \{(x_1, y_1), \ldots, (x_n, y_n)\}\)
  - a model \(Y = f(x; \Phi)\) where \(\Phi = (\phi_1, \ldots, \phi_k)\) is a vector of parameters

- **the goal is:**
  - to find the model parameters that lead to the best approximation to the observed data, i.e. to determine

\[
\mathcal{E}^* = \min_{\Phi} \sum_i \left[ y_i - f(x_i, \Phi) \right]^2
\]

- the canonical example is the problem of fitting a line to a set of points
- here \(\Phi = (a, b)\), and \(f(x; a, b) = ax + b\)
Two main cases

- **non-linear least squares**
  - $f(x, \Phi)$ not linear on $\Phi$, e.g.
    $$f(x; \Phi) = \sum_k \sin(\phi_k x)$$

- **linear least squares**
  - $f(x, \Phi)$ linear on $\Phi$, e.g.
    $$f(x; \Phi) = \sum_k \phi_k \sin(x)$$

- **note**: all that matters is linearity on $\Phi$, both nonlinear on $x$

- **other linear models**: polynomials, splines, neural networks, Fourier decompositions, etc.
Non-linear least squares

- most difficult case
- optimal solution if and only if:
  - gradient of $\varepsilon$ is zero
  - Hessian of $\varepsilon$ negative definite

$$\forall z, \quad z^T \left( \nabla^2_{\Phi} \varepsilon \right) z < 0$$

- in general this has no closed form
- numerical solution, e.g. gradient descent
  - pick initial estimate $\Phi^{(0)}$
  - iterate $\Phi^{(n+1)} = \Phi^{(n)} - \alpha \nabla_{\Phi} \varepsilon(\Phi^{(n)})$
Linear least squares

- closed form solution
  - write
    \[
    \begin{bmatrix}
    f(x_1, \Phi) \\
    \vdots \\
    f(x_n, \Phi)
    \end{bmatrix} = \Gamma(x_1, \cdots, x_n)\Phi,
    \quad
    y = \begin{bmatrix}
    y_1 \\
    \vdots \\
    y_n
    \end{bmatrix}
    \]
  - solution is given by normal equations
    \[
    \Phi = (\Gamma^T\Gamma)^{-1}\Gamma^T y
    \]
- e.g. for a line \( f(x; \phi_0, \phi_1) = \phi_0 + \phi_1x \)
  \[
  \Gamma(x_1, \cdots, x_n) = \begin{bmatrix}
  1 & x_1 \\
  \vdots & \vdots \\
  1 & x_n
  \end{bmatrix},
  \quad
  [\begin{bmatrix}
  \phi_0 \\
  \phi_1
  \end{bmatrix}] = \begin{bmatrix}
  \sum_{i} 1 & \sum_{i} x_i \\
  \sum_{i} x_i & \sum_{i} x_i^2
  \end{bmatrix}^{-1}\begin{bmatrix}
  \sum_{i} y_i \\
  \sum_{i} y_i x_i
  \end{bmatrix}
  \]
Very powerful

Q: what is the best linear approximation of a N point sequence by M DFT style exponentials?

\[ x_n = \sum_{k=1}^{M} \phi_k e^{j \frac{2\pi}{N} kn} \]

to get least squares solution, we need \( \Gamma(1, \ldots, N) \)

\[
\begin{bmatrix}
    x_0 \\
    x_1 \\
    \vdots \\
    x_{N-1}
\end{bmatrix}
= 
\begin{bmatrix}
    \sum_{k=0}^{M-1} \phi_k e^{j \frac{2\pi 0}{N} k} \\
    \sum_{k=0}^{M-1} \phi_k e^{j \frac{2\pi}{N} k} \\
    \vdots \\
    \sum_{k=0}^{M-1} \phi_k e^{j \frac{2\pi (N-1)}{N} k}
\end{bmatrix}
= 
\begin{bmatrix}
    1 \\
    e^{j \frac{2\pi}{N}} \\
    \vdots \\
    e^{j \frac{2\pi (N-1)}{N}}
\end{bmatrix}
\begin{bmatrix}
    \phi_0 \\
    \phi_1 \\
    \vdots \\
    \phi_{M-1}
\end{bmatrix}
\]
Best Fourier approximation

this means that

\[ \Gamma = \begin{bmatrix}
1 & e^{j\frac{2\pi}{N}} & e^{j\frac{4\pi}{N}} & \cdots & e^{j\frac{2(M-1)\pi}{N}} \\
e^{-j\frac{2\pi}{N}} & e^{-j\frac{2\pi}{N}} & e^{-j\frac{2\pi}{N}} & \cdots & e^{-j\frac{2\pi}{N}} \\
e^{-j\frac{4\pi}{N}} & e^{-j\frac{4\pi}{N}} & e^{-j\frac{4\pi}{N}} & \cdots & e^{-j\frac{4\pi}{N}} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
e^{-j\frac{2\pi(N-1)}{N}} & e^{-j\frac{2\pi(N-1)}{N}} & e^{-j\frac{2\pi(N-1)}{N}} & \cdots & e^{-j\frac{2\pi(N-1)}{N}}
\end{bmatrix} \]

this is orthonormal, i.e. \( \Gamma^T \Gamma = I \), and

\[ \Phi = \left( \Gamma^T \Gamma \right)^{-1} \Gamma^T x = \Gamma^T x \iff \phi_k = \sum_{n=0}^{N-1} x_n e^{j\frac{2\pi}{N}kn}, \quad k = 0, \ldots, M-1 \]

i.e. the best approximation are the \( M \) DFT coefficients associated with the exponentials
Signal approximation

Q: what is the band-pass filter $h(n)$ whose output $y(n)$ best approximates a signal $x(n)$ in the frequency range $\Omega$?

we have seen that $y(n)$ must have DFT

$$Y(k) = \begin{cases} X(k), & k \in \Omega \\ 0, & \text{otherwise} \end{cases}$$

hence optimal filter has DFT

$$H(k) = \begin{cases} 1, & k \in \Omega \\ 0, & \text{otherwise} \end{cases}$$

i.e. it is the ideal band-pass filter of band $\Omega$

intuitive: ideal = best approximation in LS sense!
Motion estimation

is an important practical example of LS problems

many applications:

- **recognition**: many events are characterized by the type of motion (e.g. walking vs running)
- strong **clues about scene structure** (e.g. when we rotate a 3D object, motion of a pixel determined by how far the 3D point is from camera)
- **segmentation** (things that move “together” belong to the same object)
- **alignment** (once we know the motion we can align images in a sequence, e.g. the NASA panoramas)
- **compression** (estimate motion, align images, transmit only error)
- etc
Motion estimation

consider the following two images

time t
Motion estimation

- consider the following two images
Motion estimation

- **goal:** given images $I(x,y,t)$ and $I(x,y,t+1)$, for each pixel find $(u,v)$ which minimizes difference

\[
D(x,y) = \left[ I(x-u, y-v, t) - I(x, y, t+1) \right]^2
\]

- **problem:** impossible to solve from one pixel alone
  - two unknowns $(u,v)$, one equation
Fundamental law

- motion can only be solved over a neighborhood
  - need at least two pixels
  - makes sense to consider more and minimize the average error
- this is least squares

\[ \varepsilon = \sum_{x,y \in R} [l(x-u, y-v, t) - l(x, y, t+1)]^2 \]
Block matching

- in fact, it is a non-linear least squares problem, since $I(x-u,y-v,t)$ is a non-linear function of $(u,v)$

- solution I: block matching
  - for each block in $I(x,y,t+1)$ do an exhaustive search in $I(x,y,t)$ for the closest match
  - very common in compression, e.g. MPEG
Block matching

- is computationally intensive
  - need to compute the squared error between the block and a collection of blocks in the previous image

- does not always produce good motion estimates
  - e.g. many matches can be equally good

- this is a problem for all motion estimation methods:
  - motion can be ambiguous when measured locally (e.g. by matching windows)
Motion ambiguities

- clearly we cannot determine the motion of a flat neighborhood
- for an edge neighborhood, we can only determine one of the two components
- the two components are uniquely defined only when the neighborhood contains 2D image structure
- this is called the “aperture problem”
Differential methods

we can at least eliminate the complexity problem, by looking for a closed-form solution to

$$\varepsilon^* = \min_{d_x,d_y \in \mathbb{R}} \sum_{x,y} \left[ l(x-u, y-v, t) - l(x, y, t+1) \right]^2$$

problem: this is a non-linear function of \((u,v)\)

solution: clearly, the problem is due to

$$\Delta = l(x, y, t+1) - l(x-u, y-v, t)$$

this equation can be made linear on \((u,v)\) by a Taylor series approximation

$$l(x-u, y-v, t) = l(x, y, t) - u \frac{\partial l(x, y, t)}{\partial x} - v \frac{\partial l(x, y, t)}{\partial y}$$
Differential methods

which leads to

\[
\Delta = I(x, y, t+1) - I(x, y, t) + A \left( u \frac{\partial I(x, y, t)}{\partial x} + v \frac{\partial I(x, y, t)}{\partial y} \right)
\]

note: we know how to compute these terms

- A is the difference between consecutive frames
  \[
  A = l_t(x, y, t) = I(x, y, t+1) - I(x, y, t)
  \]
- B is
  \[
  B = \nabla I(x, y, t)^T \begin{bmatrix} u \\ v \end{bmatrix}
  \]
  i.e. a function of the image gradient
  \[
  \nabla I(x, y, t) = \begin{bmatrix} l_x(x, y, t) \\ l_y(x, y, t) \end{bmatrix}^T
  \]
  \[
  = \begin{bmatrix} \frac{\partial I(x, y, t)}{\partial x} \\ \frac{\partial I(x, y, t)}{\partial y} \end{bmatrix}^T
  \]
Differential methods

we thus have

\[ \Delta = l_t(x, y, t) + ul_x(x, y, t) + vl_y(x, y, t) \]

and the least squares problem is

\[ \varepsilon^* = \sum_{x, y \in R} \left[ l_t(x, y) + ul_x(x, y) + vl_y(x, y) \right]^2 \]

(note: since t is constant, we omit it)

this is now linear least squares, we can just use our formula

recall that
Linear least squares

if

\[ \varepsilon^* = \min_{\Phi} \sum_i [y_i - f(x_i, \Phi)]^2 \]

then the LS solution is:

- write

\[
\begin{bmatrix}
  f(x_1, \Phi) \\
  \vdots \\
  f(x_n, \Phi)
\end{bmatrix}
= \Gamma(x_1, \cdots, x_n) \Phi,
\]

\[
y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}
\]

- solution is given by normal equations

\[
\Phi = (\Gamma^T \Gamma)^{-1} \Gamma^T y
\]
Least squares solution

- for motion, instead of
  \[ \epsilon^* = \sum_i [y_i - f(x_i, \Phi)]^2 \]
- we have
  \[ \epsilon^* = \sum_{x,y \in R} \left[ l_t(x, y) + ul_x(x, y) + vl_y(x, y) \right]^2 \]
- and write

\[
\begin{bmatrix}
  f(x_1, y_1, \Phi) \\
  \vdots \\
  f(x_n, y_n, \Phi)
\end{bmatrix} =
\begin{bmatrix}
  l_x(x_1, y_1) & l_y(x_1, y_1) \\
  \vdots & \vdots \\
  l_x(x_n, y_n) & l_y(x_n, y_n)
\end{bmatrix}
\begin{bmatrix}
  u \\
  v
\end{bmatrix},
\begin{bmatrix}
  y
\end{bmatrix} =
\begin{bmatrix}
  l_t(x_1, y_1) \\
  \vdots \\
  l_t(x_n, y_n)
\end{bmatrix}
\]
Least squares solution

the normal equations are

\[
\begin{align*}
\begin{bmatrix}
u \\ v
\end{bmatrix} &= - \left( \begin{bmatrix}
l_x(x_1, y_1) & \cdots & l_x(x_n, y_n) \\ l_y(x_1, y_1) & \cdots & l_y(x_n, y_n)
\end{bmatrix} \begin{bmatrix}
l_x(x_1, y_1) \\ l_y(x_1, y_1)
\end{bmatrix} & & \cdots & & \begin{bmatrix}
l_x(x_n, y_n) \\ l_y(x_n, y_n)
\end{bmatrix} \right)^{-1} \begin{bmatrix}
l_x(x_1, y_1) \\ l_y(x_1, y_1)
\end{bmatrix} \\
\begin{bmatrix}
x \\ y \\
x \\ y
\end{bmatrix} &= \begin{bmatrix}
l_x(x_1, y_1) & \cdots & l_x(x_n, y_n) \\ l_y(x_1, y_1) & \cdots & l_y(x_n, y_n)
\end{bmatrix} \begin{bmatrix}
l_t(x_1, y_1) \\ \vdots \\ l_t(x_n, y_n)
\end{bmatrix}
\end{align*}
\]

leading to the solution

\[
\begin{align*}
\begin{bmatrix}
u \\ v
\end{bmatrix} &= - \left( \begin{bmatrix}
\sum_{x,y \in R} l_x^2(x, y) & \sum_{x,y \in R} l_x(x, y)l_y(x, y) \\ \sum_{x,y \in R} l_x(x, y)l_y(x, y) & \sum_{x,y \in R} l_y^2(x, y)
\end{bmatrix} \right)^{-1} \begin{bmatrix}
\sum_{x,y \in R} l_x(x, y)l_t(x, y) \\ \sum_{x,y \in R} l_y(x, y)l_t(x, y)
\end{bmatrix}
\end{align*}
\]
Least squares solution

when is this well defined?

note that

$$\begin{bmatrix}
\sum_{x,y \in R} I_x^2(x, y) & \sum_{x,y \in R} I_x(x, y)I_y(x, y) \\
\sum_{x,y \in R} I_x(x, y)I_y(x, y) & \sum_{x,y \in R} I_y^2(x, y)
\end{bmatrix}$$

has to be invertible

it turns out that this is a function of the image structure within the window $R$
Orientation representations

more general question: what sorts of structure are there?

It is common to describe image patches by the variation of the gradient orientation

important types:

• constant window
  • small gradient mags

• edge window
  • few large gradient mags in one direction

• flow window
  • many large gradient mags in one direction (e.g. hair)

• corner window
  • large gradient mags that swing (e.g. corner)
Representing Windows

- How can we detect these types of windows?
- The key is the matrix

\[ H = \sum_{\text{window}} (\nabla I)(\nabla I)^T \]

- How does it relate to edges?
- The answer is in the rank

\[ H = \sum_{\text{edge pts}} \begin{pmatrix} -k \\ 0 \end{pmatrix}(\begin{pmatrix} -k \\ 0 \end{pmatrix})^T = \begin{pmatrix} nk^2 & 0 \\ 0 & 0 \end{pmatrix} \]

\[ H = \#\{\text{edges}\} x \begin{pmatrix} nk^2 & 0 \\ 0 & 0 \end{pmatrix} \]

\[ H = \sum_{\text{vert edge}} \begin{pmatrix} -k \\ 0 \end{pmatrix}(\begin{pmatrix} -k \\ 0 \end{pmatrix}) + \sum_{\text{horiz edge}} \begin{pmatrix} 0 \\ -k \end{pmatrix}(0 \ -k) \]

\[ = \begin{pmatrix} \frac{nk^2}{2} & 0 \\ 0 & \frac{nk^2}{2} \end{pmatrix} \]
Representing Windows

- recall: the eigenvalues of a diagonal matrix are the diagonal entries
- hence:
  - constant window
    - small eigenvalues
  - edge window
    - one medium, one small
  - flow window
    - one large, one small
  - corner window
    - two large eigenvalues

\[
H = \sum_{\text{window}} (\nabla I)(\nabla I)^T
\]
Representing Windows

- what about other orientations?
- useful property
  - if $A$ is a $2 \times 2$ matrix
  - then
    \[ \lambda_1 \lambda_2 = \det(A) \]
    \[ \lambda_1 + \lambda_2 = a_{11} + a_{22} = \text{trace}(A) \]
- to have full rank we need diversity in the component matrices
- i.e. need edges of different orientation

\[
H = \sum_{\text{window}} (\nabla I)(\nabla I)^T
\]

\[
H = \sum_{\text{edge}} \begin{pmatrix} -a \\ -b \end{pmatrix} \begin{pmatrix} -a & -b \end{pmatrix} = n \begin{pmatrix} a^2 & ab \\ ab & b^2 \end{pmatrix}
\]

\[ \lambda_1 = n(a^2 + b^2); \lambda_2 = 0 \]

\[
H = n_1 \begin{pmatrix} a^2 & ab \\ ab & b^2 \end{pmatrix} + n_2 \begin{pmatrix} c^2 & 0 \\ 0 & 0 \end{pmatrix}
\]

\[ = \begin{pmatrix} n_1 a^2 + n_2 c^2 & ab \\ ab & b^2 \end{pmatrix} \]

\[ \lambda_1, \lambda_2 > 0 \]
Representing Windows

- **in summary:**
  - **constant window**
    - small eigenvalues
  - **edge window**
    - one medium, one small
  - **flow window**
    - one large, one small
  - **corner window**
    - two large eigenvalues

- **this confirms what we had already seen:**
  - motion can only be computed unambiguously when the neighborhood contains 2D information (e.g. corners)

\[ H = 0 \]
\[ H \propto \begin{pmatrix} k^2 & 0 \\ 0 & 0 \end{pmatrix} \]
\[ H \propto \begin{pmatrix} k^2 & 0 \\ 0 & k^2 \end{pmatrix} \]
In summary

$[U, V] = \text{lsmc}(I, I', w)$

- compute gradients $I_x, I_y, I_t = I' - I$
- for each pixel $(x, y)$
  - let window $R = \{(x_i, y_i) | x - w \leq x_i \leq x + w, \ y - w \leq y_i \leq y + w\}$
  - compute

$$
\begin{bmatrix}
U \\
V
\end{bmatrix} = 
\begin{bmatrix}
\sum_{x, y \in R} I_x^2(x, y) & \sum_{x, y \in R} I_x(x, y)I_y(x, y) \\
\sum_{x, y \in R} I_x(x, y)I_y(x, y) & \sum_{x, y \in R} I_y^2(x, y)
\end{bmatrix}^{-1}
\begin{bmatrix}
\sum_{x, y \in R} I_x(x, y)I_t(x, y) \\
\sum_{x, y \in R} I_y(x, y)I_t(x, y)
\end{bmatrix}
$$

- make $U(x, y) = u, \ V(x, y) = v$
- return $U, V$
Problems

- recall we used the Taylor series approximation

\[
I_{(x-u, y-v, t)} = I_{(x, y, t)} - u \frac{\partial I_{(x, y, t)}}{\partial x} - v \frac{\partial I_{(x, y, t)}}{\partial y}
\]

- this is a good approximation only for small \((u, v)\)

- to avoid this problem we need to use pyramids
Hierarchical estimation

algorithm:

- do motion estimation using \( I_0 \) and \( I'_0 \) to obtain \((u_0, v_0)\)
- warp \( I_0 \) with \((u_0, v_0)\):
  \[
  wpd_0(x, y) = I_0(x-u_0, y-v_0)
  \]
- up-sample by 2 to get \( I_1 \)
- do motion estimation using \( I_1 \) and \( I'_1 \) to obtain \((u_1, v_1)\)
- warp \( I_1 \) with \((u_1, v_1)\)
- etc.
Hierarchical estimation

- each stage improves the match

solution:
- upsample all \((u_i, v_i)\) to full resolution
- add to obtain \((u, v)\)

note that
- small displacements at low resolution
- are large displacements at full resolution

combines linearity with ability to estimate large displacements
Motion models

- so far we have dealt
  - local motion (each pixel moves by itself)
  - translation

\[ l(x, y, t + 1) = l(x - u, y - v, t) \]

- local motion is the most generic (e.g. tree leaves blowing in the wind)

- one important alternative case is that of global motion
  - motion of all pixels satisfies one common equation
  - usually due to camera motion: panning, rotation, zooming
Important cases

- Point \((x, y)\) at time \(t\) warped into point \((x', y')\) at time \(t+1\)

- Important global motions are:
  - Translation by \((u, v)\)
    \[
    \begin{bmatrix}
    x' \\
    y'
    \end{bmatrix} = \begin{bmatrix}
    x \\
    y
    \end{bmatrix} + \begin{bmatrix}
    u \\
    v
    \end{bmatrix}
    \]
  - Rotation by \(\theta\)
    \[
    \begin{bmatrix}
    x' \\
    y'
    \end{bmatrix} = \begin{bmatrix}
    \cos \theta & \sin \theta \\
    -\sin \theta & \cos \theta
    \end{bmatrix} \begin{bmatrix}
    x \\
    y
    \end{bmatrix}
    \]
  - Scaling by \((s_x, s_y)\)
    \[
    \begin{bmatrix}
    x' \\
    y'
    \end{bmatrix} = \begin{bmatrix}
    s_x & 0 \\
    0 & s_y
    \end{bmatrix} \begin{bmatrix}
    x \\
    y
    \end{bmatrix}
    \]
Affine transformations

- these are all special cases of the affine transformation

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}
\]

- motion of entire image described by \( \Phi = (a,b,c,d,e,f)^T \)

- can account for translation, rotation, scaling, and shear

![Affine transformations](image)
Any questions?