

CS9840
Learning and Computer Vision
Prof. Olga Veksler

Lecture 2

Linear Machines, Optical Flow

Some Slides are from Cornelia, Fermüller,
[Mubarak Shah](#),
Gary Bradski,
Sebastian Thrun

Outline

- Linear Machines
- Start preparation for the first paper
 - “*Recognizing Action at a Distance*” by A. Efros, A.Berg, G. Mori, Jitendra Malik
 - there should be a link to PDF file on our web site
- Next time:
 - Discuss the paper and watch video
 - Prepare for the second paper

Last Time: Supervised Learning

- Training samples (or examples) $X^1, X^2, \dots, X^n$
- Each example is typically multi-dimensional
 - $X^i_1, X^i_2, \dots, X^i_d$ are typically called *features*, X^i is sometimes called a *feature vector*
 - **How many features and which features do we take?**
- Know desired output for each example (labeled samples) $Y^1, Y^2, \dots, Y^n$
 - This learning is supervised (“teacher” gives desired outputs).
 - Y^i are often one-dimensional, but can be multidimensional

Last Time: Supervised Learning

- Wish to design a *machine* $f(X, W)$ s.t.
 $f(X, W) = \text{true output value at } X$
 - In classification want $f(X, W) = \text{label of } X$
 - **How do we choose f ?**
 - when we choose a particular f , we are making implicit assumptions about our problem
 - W is typically multidimensional vector of weights (also called *parameters*) which enable the machine to “learn”
 - $W = [w_1, w_2, \dots, w_k]$

Training and Testing

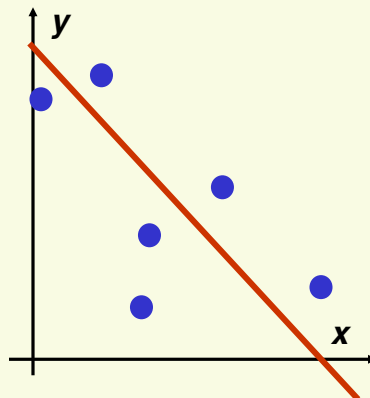
- There are 2 phases, training and testing
 - Divide all labeled samples $X^1, X^2, \dots, X^n$ into 2 sets, *training* set and *testing* set
 - Training phase is for “teaching” our machine (finding optimal weights W)
 - Testing phase is for evaluating how well our machine works on unseen examples
- Training phase
 - Find the weights W s.t. $f(X^i, W) = Y^i$ “as much as possible” for the *training* samples X^i
 - “as much as possible” needs to be defined
 - Training can be quite complex and time-consuming

Loss Function

- How do we quantify what it means for the machine $f(X, W)$ do well in the training and testing phases?
- $f(X, W)$ has to be “close” to the true output on X
- Define Loss (or Error) function L
 - This is up to the designer (that is you)
- Typically first define per-sample loss $L(X^i, Y^i, W)$
 - Some examples:
 - for classification, $L(X^i, Y^i, W) = \mathbf{I}[f(X^i, W) \neq Y^i]$,
where $\mathbf{I}[\text{true}] = 1$, $\mathbf{I}[\text{false}] = 0$
 - we just care if the sample has been classified correctly
 - For continuous Y , $L(X^i, Y^i, W) = \|f(X^i, W) - Y^i\|^2$,
 - how far is the estimated output from the correct one?
- Then loss function $L = \sum_i L(X^i, Y^i, W)$
 - Number of missclassified example for classification
 - Sum of distances from the estimated output to the correct output

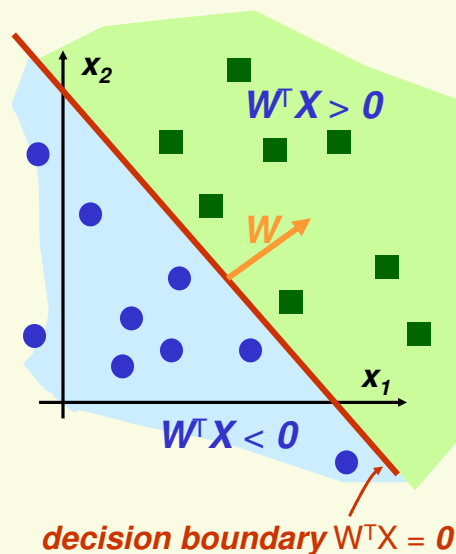
Linear Machine, Continuous Y

- $f(X, W) = w_0 + \sum_{i=1,2,\dots,d} w_i x_i$
 - w_0 is called bias
- In vector form, if we let $X = (1, x_1, x_2, \dots, x_d)$, then $f(X, W) = W^T X$
 - notice abuse of notation, I made $X = [1 \ X]$
- This is standard linear regression (line fitting)
 - assume $L(X^i, Y^i, W) = ||f(X^i, W) - Y^i||^2$
 - optimal W can be found by solving linear system of equations $W^* = [\sum X^i (X^i)^T]^{-1} \sum Y^i X^i$



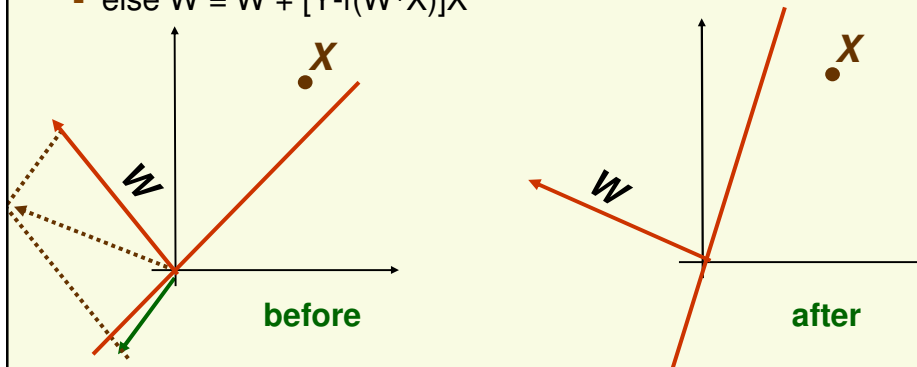
Linear Machine: binary Y

- $f(X, W) = \text{sign}(w_0 + \sum_{i=1,2,\dots,d} w_i x_i)$
 - $\text{sign}(\text{positive}) = 1$, $\text{sign}(\text{negative}) = -1$
 - w_0 is called bias
- In vector form, if we let $X = (1, x_1, x_2, \dots, x_d)$ then $f(X, W) = \text{sign}(W^T X)$



Perceptron Learning Procedure (Rosenblatt 1957)

- $f(X, W) = \text{sign}(w_0 + \sum_{i=1,2,\dots,d} w_i x_i)$
- Let $L(X^i, Y^i, W) = \mathbf{I}[f(X^i, W) \neq Y^i]$. How do we learn W ?
- A solution:
- Iterate over all training samples
 - if $f(X, W) = Y$ (correct label), do nothing
 - else $W = W + [Y - f(W^T X)]X$



Perceptron Learning Procedure (Rosenblatt 1957)

- Amazing fact: If the samples are linearly separable, the perceptron learning procedure will converge to a solution (separating hyperplane) in a finite amount of time
- Bad news: If the samples are not linearly separable, the perceptron procedure will not terminate, it will go on looking for a solution which does not exist!
- For most interesting problems the samples are not linearly separable
- Is there a way to learn W in non-separable case?
 - Remember, it's ok to have training error, so we don't have to have "perfect" classification

Optimization

- Need to minimize a function of many variables

$$J(\mathbf{x}) = J(x_1, \dots, x_d)$$

- We know how to minimize $J(\mathbf{x})$
 - Take partial derivatives and set them to zero

$$\begin{bmatrix} \frac{\partial}{\partial x_1} J(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_d} J(\mathbf{x}) \end{bmatrix} = \nabla J(\mathbf{x}) = 0$$

gradient

- However solving analytically is not always easy

- Would you like to solve this system of nonlinear equations?

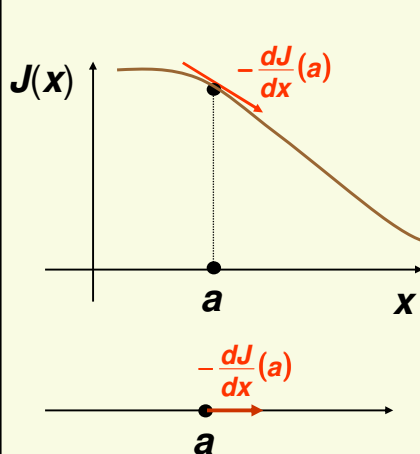
$$\begin{cases} \sin(x_1^2 + x_2^3) + e^{x_4^2} = 0 \\ \cos(x_1^2 + x_2^3) + \log(x_3^3)^{x_4^2} = 0 \end{cases}$$

- Sometimes it is not even possible to write down an analytical expression for the derivative, we will see an example later today

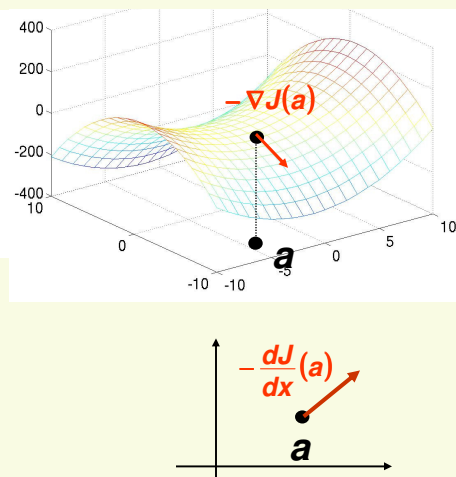
Optimization: Gradient Descent

- Gradient $\nabla J(\mathbf{x})$ points in direction of steepest increase of $J(\mathbf{x})$, and $-\nabla J(\mathbf{x})$ in direction of steepest decrease

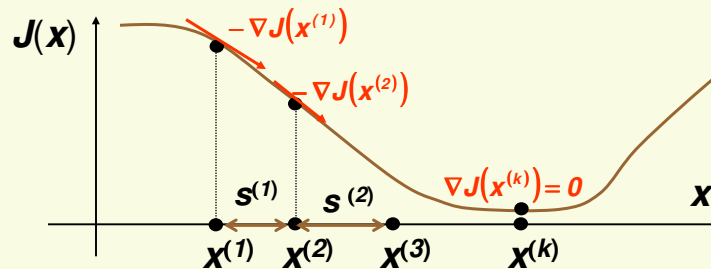
one dimension



two dimensions



Optimization: Gradient Descent



Gradient Descent for minimizing any function $J(x)$

set $k = 1$ and $x^{(1)}$ to some initial guess for the weight vector

while $\eta^{(k)} |\nabla J(x^{(k)})| > \varepsilon$

choose **learning rate** $\eta^{(k)}$

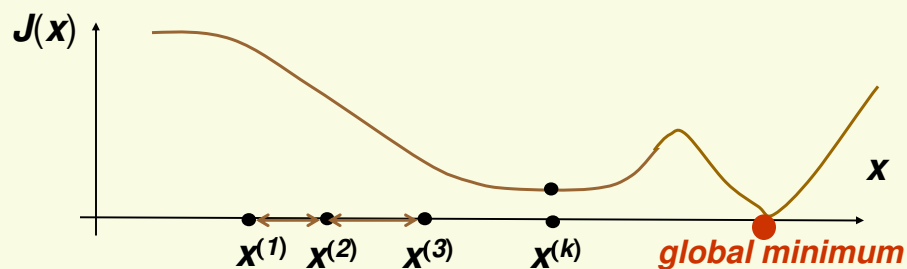
$$x^{(k+1)} = x^{(k)} - \eta^{(k)} \nabla J(x)$$

(update rule)

$k = k + 1$

Optimization: Gradient Descent

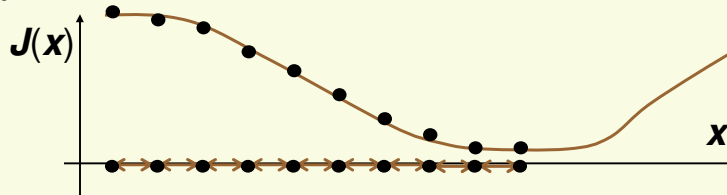
- Gradient descent is guaranteed to find only a local minimum



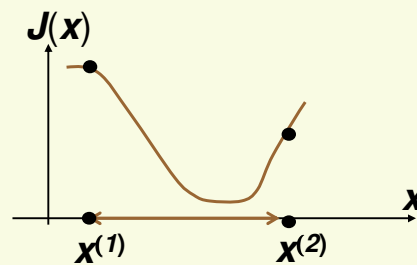
- Nevertheless gradient descent is very popular because it is simple and applicable to any differentiable function

Optimization: Gradient Descent

- Main issue: how to set parameter η (*learning rate*)
- If η is too small, need too many iterations



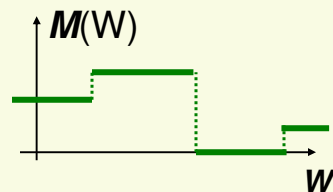
- If η is too large may overshoot the minimum and possibly never find it (if we keep overshooting)



“Optimal” W with Gradient Descent

- $f(X, W) = \text{sign}(w_0 + \sum_{i=1,2,\dots,d} w_i x_i)$
- If we let $L(X^i, Y^i, W) = \mathbb{I}[f(X^i, W) \neq Y^i]$, then $L(W)$ is the number of misclassified examples
- Let M be the set of examples misclassified by W

$$M(W) = \{ \text{sample } X^i \text{ s.t. } W^T X^i \neq Y^i \}$$
- Then $L(W) = |M(W)|$, the size of $M(W)$
- $L(W)$ is piecewise constant, gradient descent is useless

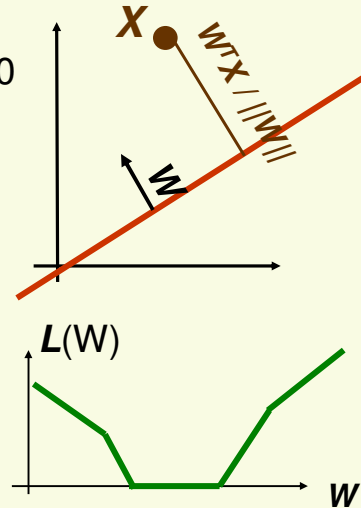


“Optimal” W with Gradient Descent

- Better choice:

$$L(W) = \sum_{X^i \in M} (-W^T X^i) Y^i$$

- If X^i is misclassified, $(W^T X^i) Y^i \leq 0$
- Thus $L(W, X^i, Y^i) \geq 0$
- $L(W, X^i, Y^i)$ is proportional to the distance of misclassified example to the decision boundary
- $L(W) = \sum L(W, X^i, Y^i)$ is piecewise linear and thus suitable for gradient descent



Batch Rule

$$L(W, X^i, Y^i) = \sum_{X \in M} (-W^T X) Y$$

- Gradient of L is $\nabla L(W) = \sum_{X \in M} (-X) Y$
 - M are samples misclassified by W
 - It is not possible to solve $\nabla L(W) = 0$ analytically
- Update rule for gradient descent: $X^{(k+1)} = X^{(k)} - \eta^{(k)} \nabla J(X)$
- Thus *gradient decent batch update rule* for $L(W)$ is:

$$W^{(k+1)} = W^{(k)} + \eta^{(k)} \sum_{Y \in M} X Y$$

- It is called *batch* rule because it is based on all misclassified examples

Single Sample Rule

- Thus *gradient decent single sample rule* for $L(W)$ is:

$$W^{(k+1)} = W^{(k)} + \eta^{(k)}(XY)$$

- apply for any sample X misclassified by $W^{(k)}$
- must have a consistent way of visiting samples

Convergence

- If classes are linearly separable, and $\eta^{(k)}$ is fixed to a constant, i.e. $\eta^{(1)} = \eta^{(2)} = \dots = \eta^{(k)} = c$ (*fixed learning rate*)
 - *both single sample and batch rules converge to a correct solution* (could be any W in the solution space)
- If classes are not linearly separable:
 - Single sample algorithm does not stop, it keeps looking for solution which does not exist
 - However by choosing appropriate learning rate, heuristically stop algorithm at hopefully good stopping point

$$\eta^{(k)} \rightarrow 0 \text{ as } k \rightarrow \infty$$

- for example, $\eta^{(k)} = \frac{\eta^{(1)}}{k}$
- for this learning rate convergence in the linearly separable case can also be proven

Learning by Gradient Descent

- Suppose we suspect that the machine has to have functional form $f(X,W)$, not necessarily linear
- Pick differentiable per-sample loss function $L(X^i, Y^i, W)$
- We need to find W that minimizes $L = \sum_i L(X^i, Y^i, W)$
- Use gradient-based minimization:
 - Batch rule: $W = W - \eta \nabla L(W)$
 - Or single sample rule: $W = W - \eta \nabla L(X^i, Y^i, W)$

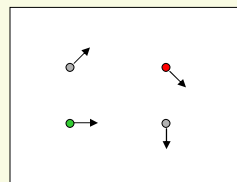
Important Questions

- How do we choose the feature vector X ?
- How do we split labeled samples into training/testing sets?
- How do we choose the machine $f(X,W)$?
- How do we choose the loss function $L(X^i, Y^i, W)$?
- How do we find the optimal weights W ?

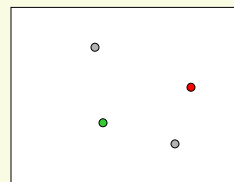
Background Preparation for Paper

- Paper: “*Recognizing Action at a Distance*” by A. Efros, A. Berg, G. Mori, Jitendra Malik
 - Optical Flow Field (related to motion field)
 - Correlation

Optical flow



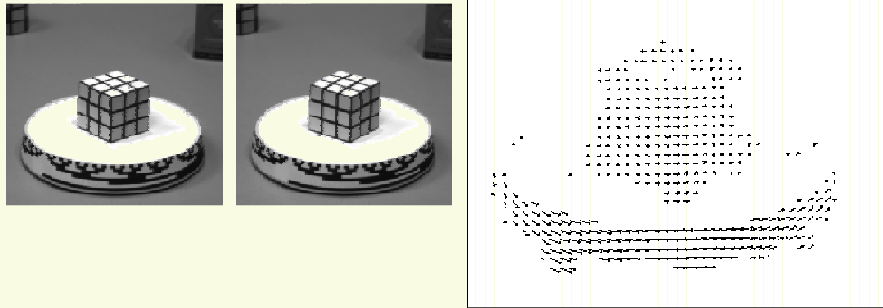
first image I_1



second image I_2

- How to estimate pixel motion from image I_1 to image I_2 ?
 - Solve pixel correspondence problem
 - given a pixel in I_1 , look for **nearby** pixels of the **same** color in I_2
- Key assumptions
 - **color constancy**: a point in I_1 looks the same in I_2
 - For grayscale images, this is **brightness constancy**
 - **small motion**: points do not move very far
- This is called the **optical flow** problem

Optical Flow Field



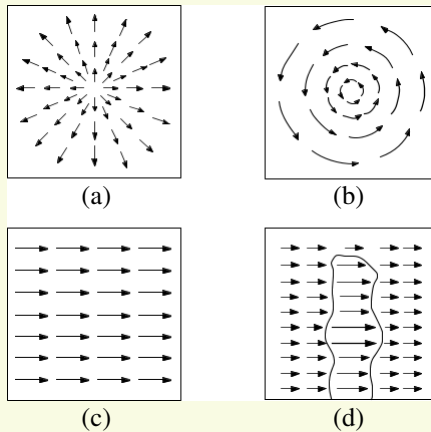
Optical Flow and Motion Field

- Optical flow field is the apparent motion of brightness patterns between 2 (or several) frames in an image sequence
- Why does brightness change between frames?
- Assuming that illumination does not change:
 - changes are due to the **RELATIVE MOTION** between the scene and the camera
 - There are 3 possibilities:
 - Camera still, moving scene
 - Moving camera, still scene
 - Moving camera, moving scene

Motion Field (MF)

- The **MF** assigns a velocity vector to each pixel in the image
- These velocities are INDUCED by the RELATIVE MOTION between the camera and the 3D scene
- The **MF** is the projection of the 3D velocities on the image plane

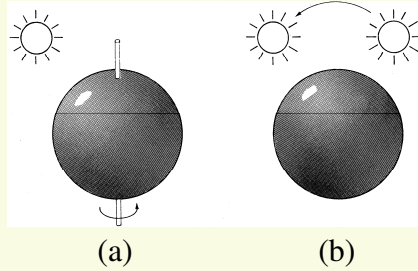
Examples of Motion Fields



(a) Translation perpendicular to a surface. (b) Rotation about axis perpendicular to image plane. (c) Translation parallel to a surface at a constant distance. (d) Translation parallel to an obstacle in front of a more distant background.

Optical Flow vs. Motion Field

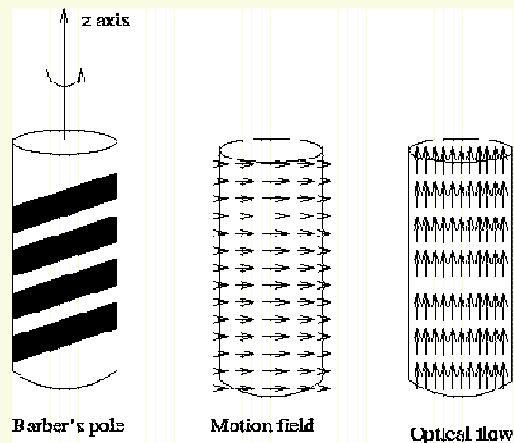
- Recall that Optical Flow is the apparent motion of brightness patterns
- We equate Optical Flow Field with Motion Field
- Frequently works, but not always:



- (a) A smooth sphere is rotating under constant illumination. Thus the optical flow field is zero, but the motion field is not
- (b) A fixed sphere is illuminated by a moving source—the shading of the image changes. Thus the motion field is zero, but the optical flow field is not

Optical Flow vs. Motion Field

- Often (but not always) optical flow corresponds to the true motion of the scene



Computing Optical Flow: Brightness Constancy Equation

- Let ***P*** be a moving point in 3D:
 - At time ***t***, ***P*** has coordinates (***X(t)***, ***Y(t)***, ***Z(t)***)
 - Let ***p***=(***x(t)***, ***y(t)***) be the coordinates of its image at time ***t***
 - Let ***E(x(t), y(t), t)*** be the brightness at ***p*** at time ***t***.
- Brightness Constancy Assumption:
 - As ***P*** moves over time, ***E(x(t), y(t), t)*** remains constant

Computing Optical Flow: Brightness Constancy Equation

$$E(x(t), y(t), t) = Constant$$

Taking derivative wrt time:

$$\frac{dE(x(t), y(t), t)}{dt} = 0$$

$$\frac{\partial E}{\partial x} \frac{dx}{dt} + \frac{\partial E}{\partial y} \frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Computing Optical Flow: Brightness Constancy Equation

1 equation with 2 unknowns

$$\frac{\partial E}{\partial x} \frac{dx}{dt} + \frac{\partial E}{\partial y} \frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Let

$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \end{bmatrix} \quad (\text{Frame spatial gradient})$$

$$v = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} \quad (\text{optical flow})$$

and $E_t = \frac{\partial E}{\partial t} \quad (\text{derivative across frames})$

Computing Optical Flow: Brightness Constancy Equation

- How to get more equations for a pixel?
 - Basic idea: impose additional constraints
 - most common is to assume that the flow field is smooth locally
 - one method: pretend the pixel's neighbors have the same (u,v)
 - If we use a 5x5 window, that gives us 25 equations per pixel!

$$E_t(p_i) + \nabla E(p_i) \cdot [u \ v] = 0$$

$$\begin{bmatrix} E_x(p_1) & E_y(p_1) \\ E_x(p_2) & E_y(p_2) \\ \vdots & \vdots \\ E_x(p_{25}) & E_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} E_t(p_1) \\ E_t(p_2) \\ \vdots \\ E_t(p_{25}) \end{bmatrix}$$

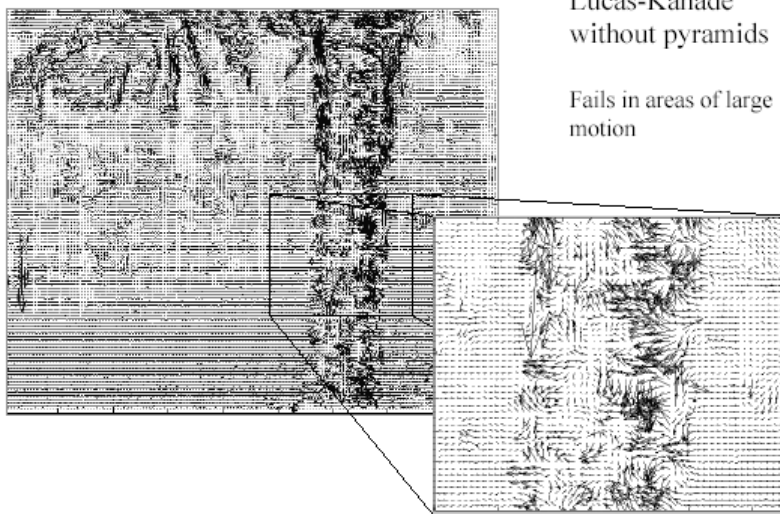
matrix **E** vector **d** vector **b**
 25x2 2x1 25x1

Video Sequence



* Picture from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Optical Flow Results



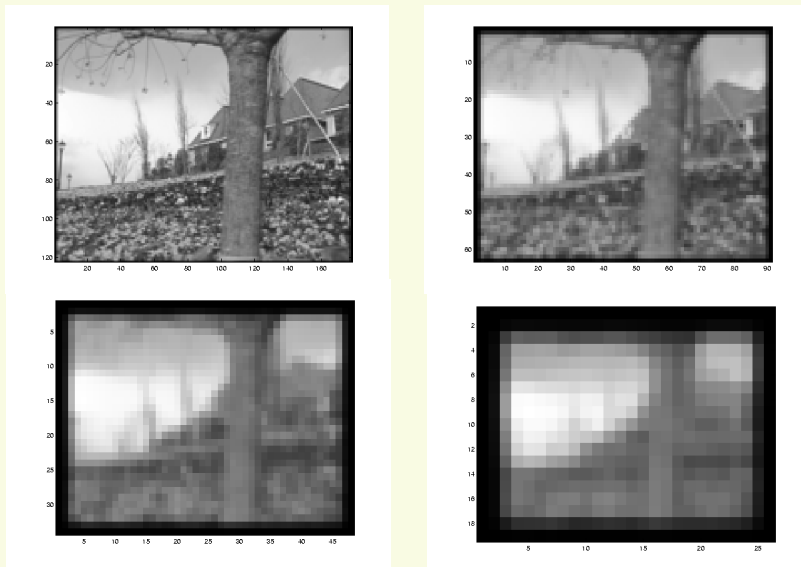
* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Revisiting the small motion assumption

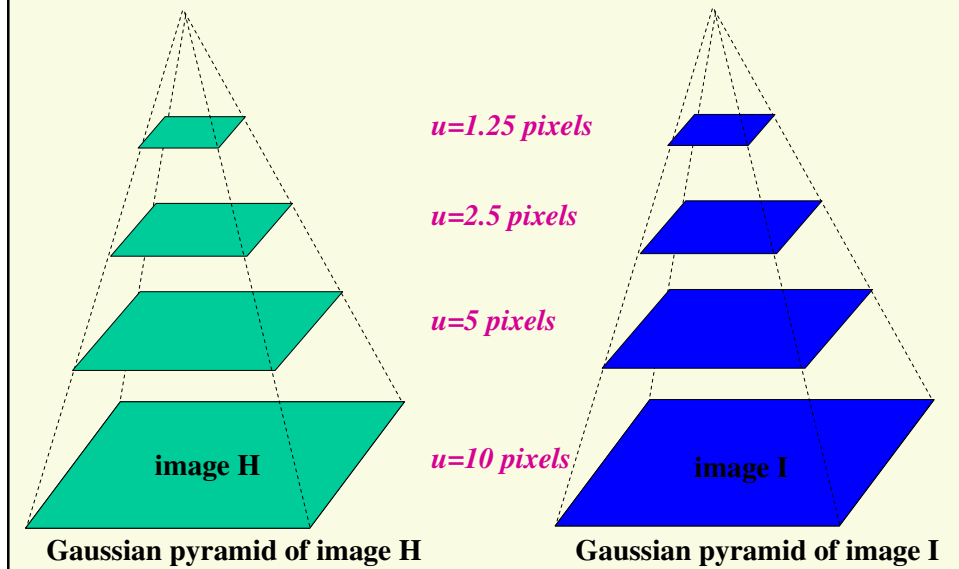


- Is this motion small enough?
 - Probably not—it's much larger than one pixel (2nd order terms dominate)
 - How might we solve this problem?

Reduce the resolution!



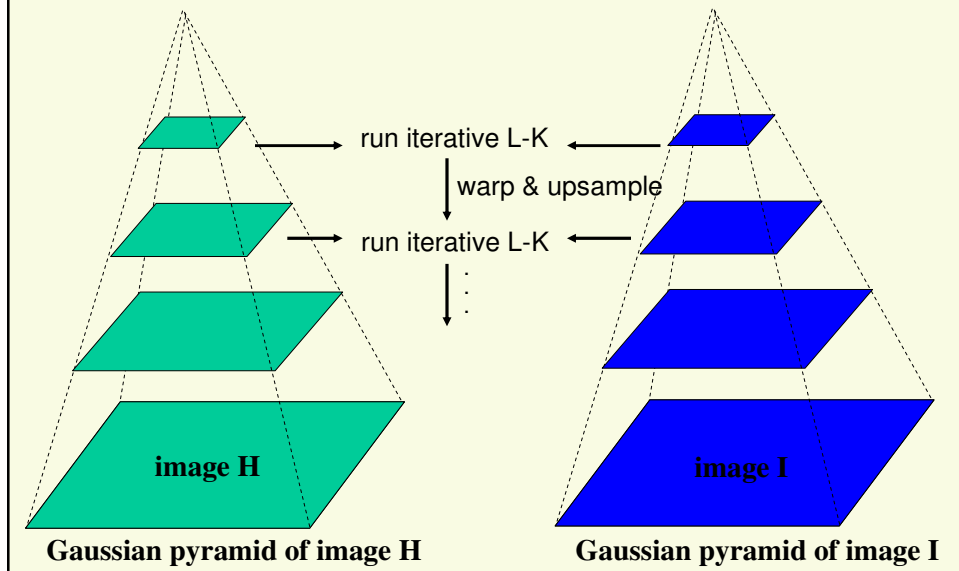
Coarse-to-fine optical flow estimation



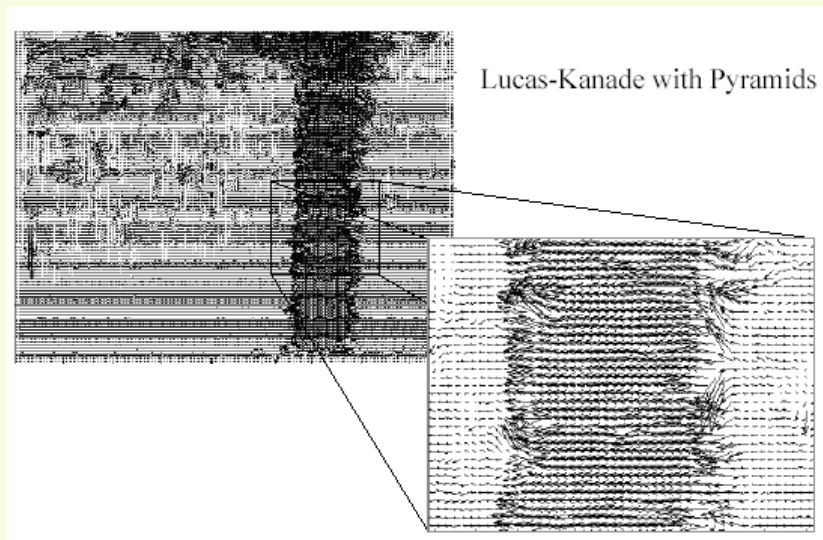
Iterative Refinement

- Iterative Lukas-Kanade Algorithm
 1. Estimate velocity at each pixel by solving Lucas-Kanade equations
 2. Warp H towards I using the estimated flow field
 - use image warping techniques
 3. Repeat until convergence

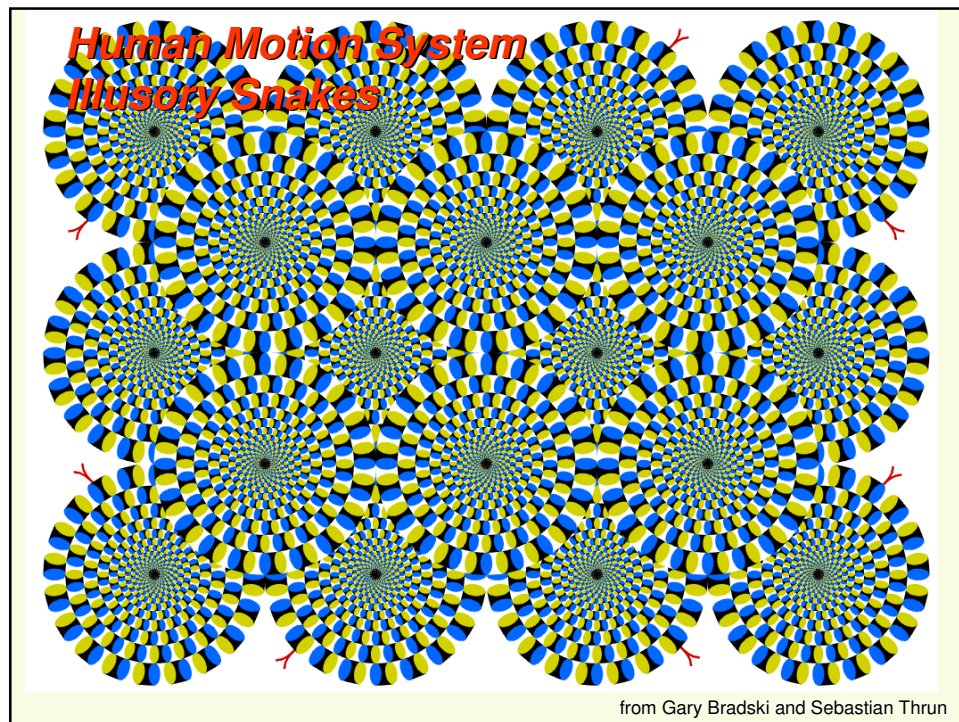
Coarse-to-fine optical flow estimation



Optical Flow Results



* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003



Other Concepts to Review

- Convolution is the operation of applying a “kernel” to each pixel of an image

image											
I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}			
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}			
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}			
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}			
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}			
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}			

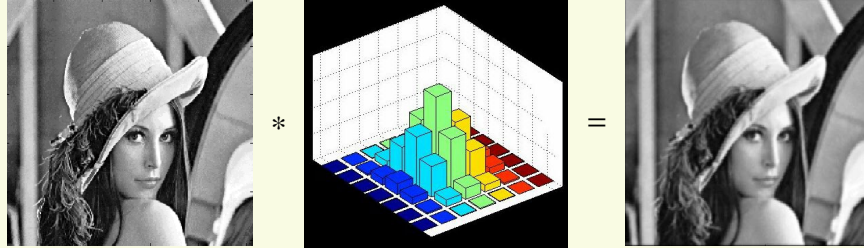
kernel		
K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

- Result of convolution has the same dimension as the image
- For example:

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23}$$
- Convolution is frequently denoted by *, for example $I * K$

Other Concepts to Review

- Gaussian smoothing (blurring): convolution operator that is used to 'blur' images and removes small detail and noise from an image



$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Gaussian vs. Smoothing



Gaussian Smoothing

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1



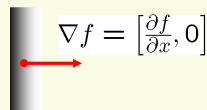
Smoothing by Averaging

$\frac{1}{25}$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Other Concepts to Review

- Image gradient: points in the direction of the most rapid increase in intensity of image f



$$\nabla f = [0, \frac{\partial f}{\partial y}]$$



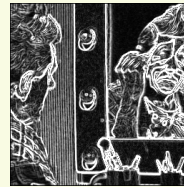
$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

- Sobel operator to compute gradient:

$$\frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \frac{\partial f}{\partial x}$$

$$\frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \frac{\partial f}{\partial y}$$

- Results:



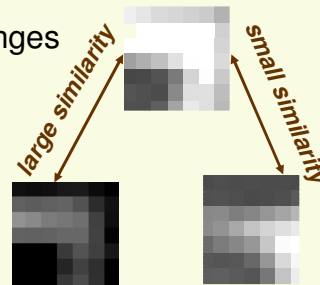
Other Concepts to Review

- Cross-correlation

$$c(f, g) = \sum_{i=1}^d f(i)g(i)$$

- measures similarity between images (or image regions) f and g
- works OK if there is no change in intensity
- Normalized cross correlation, more popular in image processing
 - Insensitive to linear intensity changes between image patches f and g

$$NCC(f, g) = \frac{\sum_{i=1}^d (f(i) - \bar{f})(g(i) - \bar{g})}{\left[\sum_{i=1}^d (f(i) - \bar{f})^2 \sum_{k=1}^d (g(i) - \bar{g})^2 \right]^{1/2}}$$



Next Time

- Paper: “*Recognizing Action at a Distance*” by A. Efros, A. Berg, G. Mori, Jitendra Malik
- When reading the paper, think about following:
 - What is the problem paper tries to solve
 - What makes this problem difficult?
 - What is the method used in the paper to solve the problem
 - What is the contribution of the paper (what new does it do)?
 - Do the experimental results look “good” to you?