Automatic Image Alignment + Optical Flow

© Mike Nese

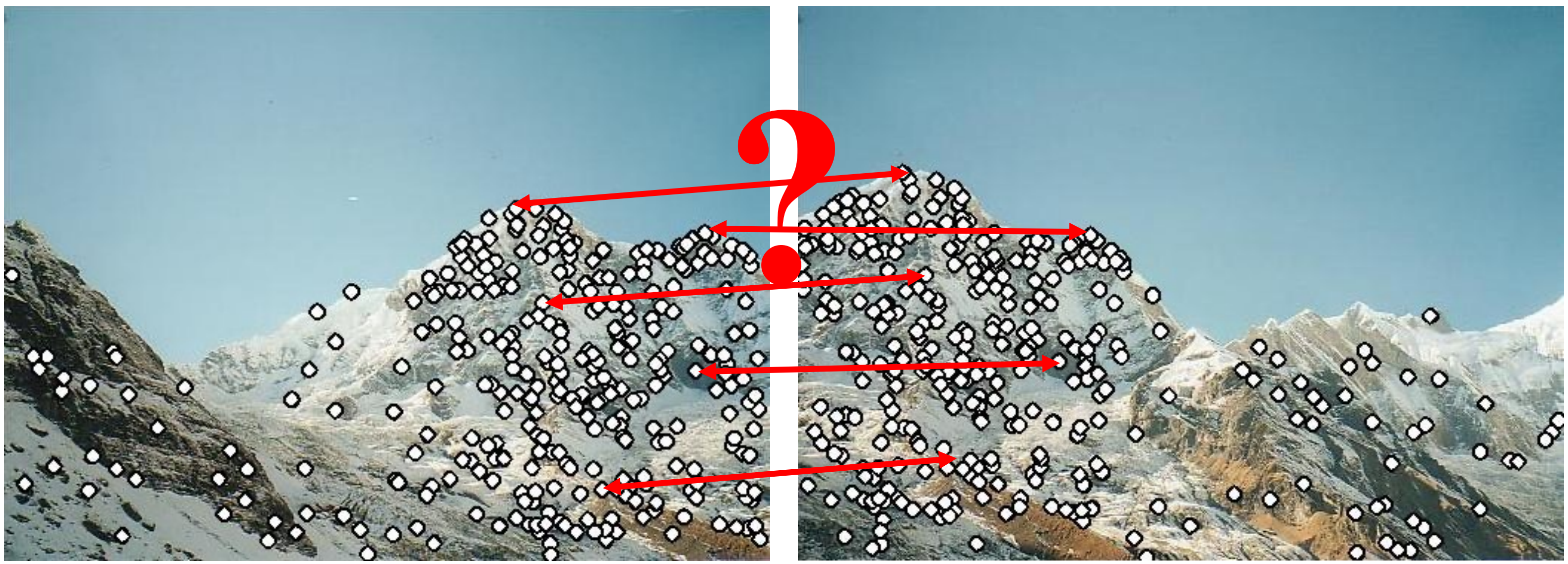
CS194: Intro to Comp. Vision and Comp. Photo
Alexei Efros, UC Berkeley, Fall 2022

with a lot of slides stolen from Steve Seitz and Rick Szeliski

Feature descriptors

We know how to detect points

Next question: **How to match them?**



Point descriptor should be:

1. Invariant
2. Distinctive

Feature Descriptor – MOPS

Multi-Scale Oriented Patches

Interest points

- Multi-scale Harris corners
- Orientation from blurred gradient
- Geometrically invariant to rotation

Descriptor vector

- Bias/gain normalized sampling of local patch (8x8)
- Photometrically invariant to affine changes in intensity

[Brown, Szeliski, Winder, CVPR’2005]

Detect Features, setup Frame

Orientation = blurred gradient

Rotation Invariant Frame

- Scale-space position (x, y, s) + orientation (θ)

Detections at multiple scales

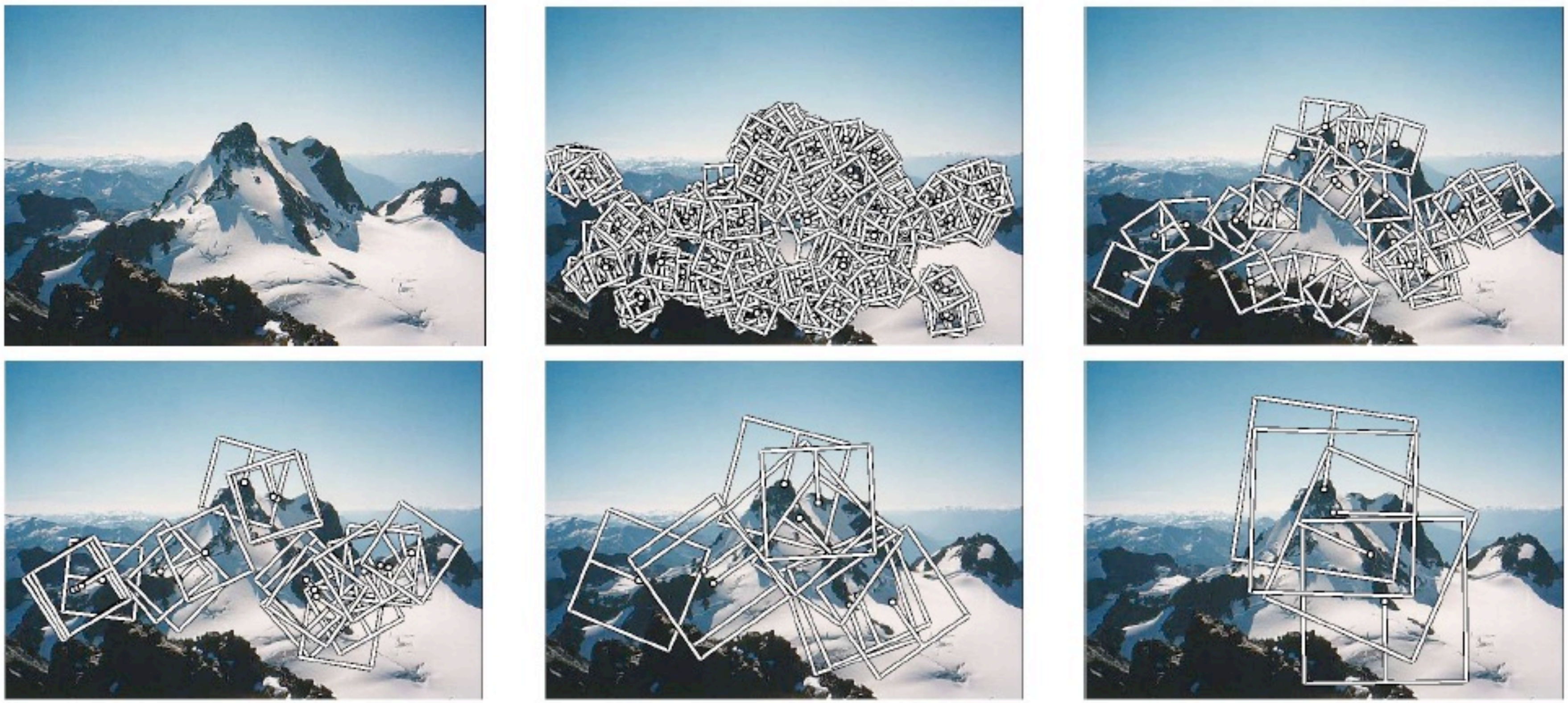


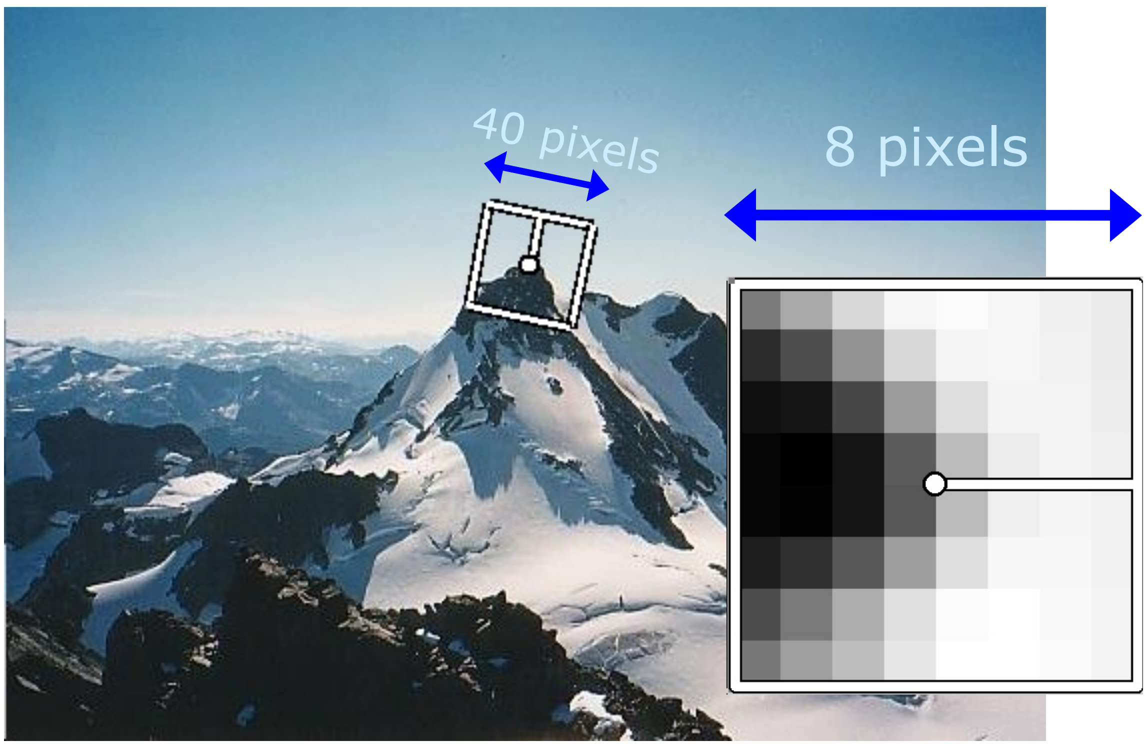
Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

MOPS descriptor vector

8x8 oriented patch

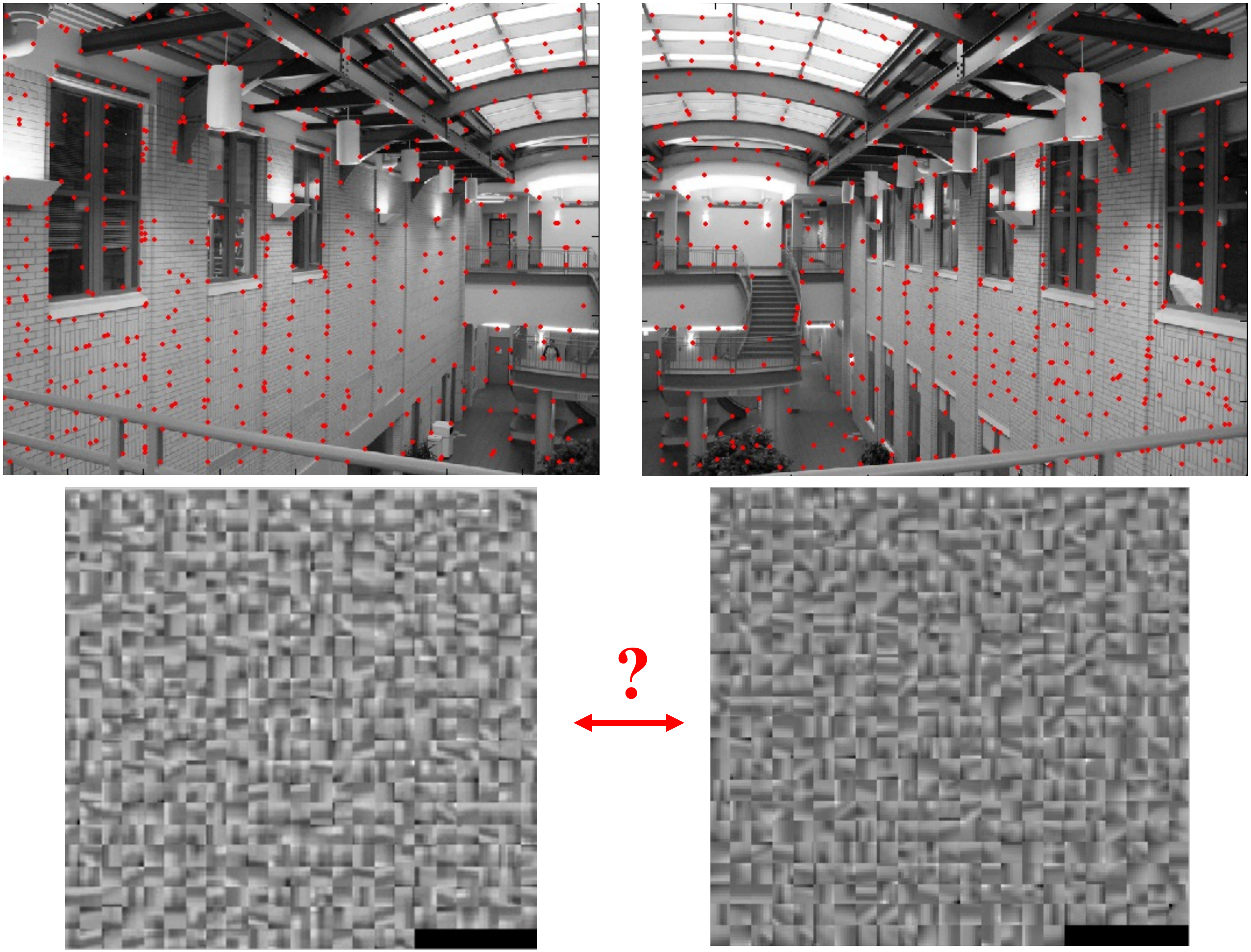
- Sampled at 5 x scale

Bias/gain normalisation: $I' = (I - \mu)/\sigma$



Automatic Feature Matching

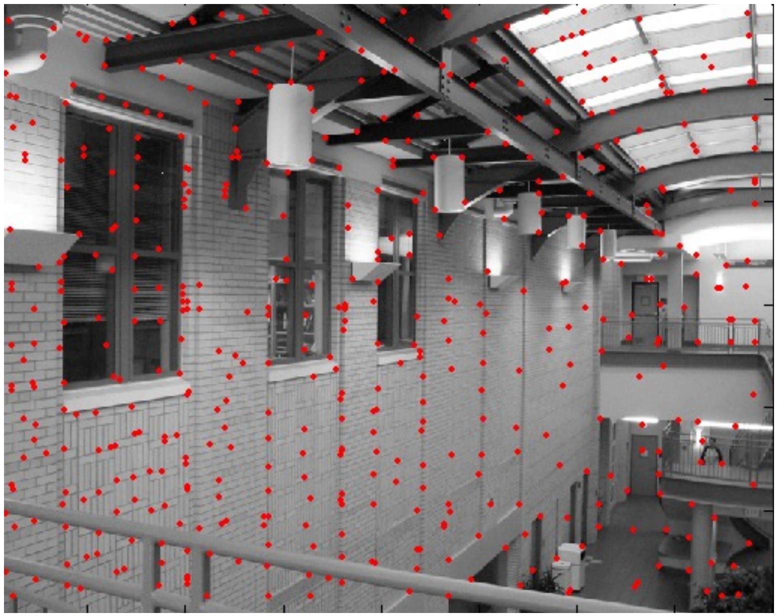
Feature matching

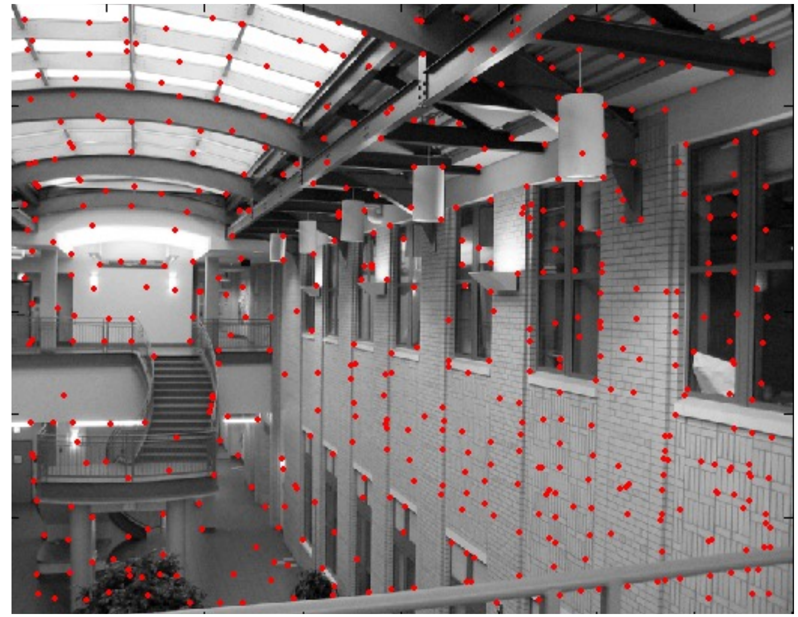


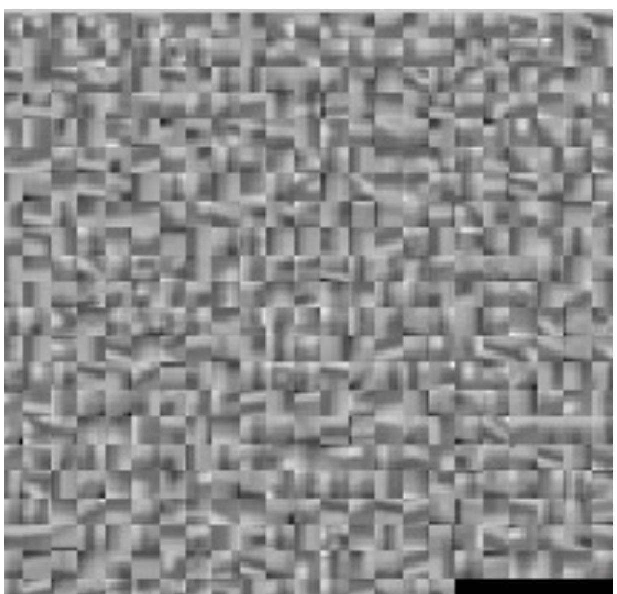
Feature matching

- Pick best match!
  - For every patch in image 1, find the most similar patch (e.g. by SSD).
  - Called “nearest neighbor” in machine learning
- Can do various speed ups:
  - Hashing
    - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)
  - Fast Nearest neighbor techniques
    - *kd*-trees and their variants
  - Clustering / Vector quantization
    - So called “visual words”

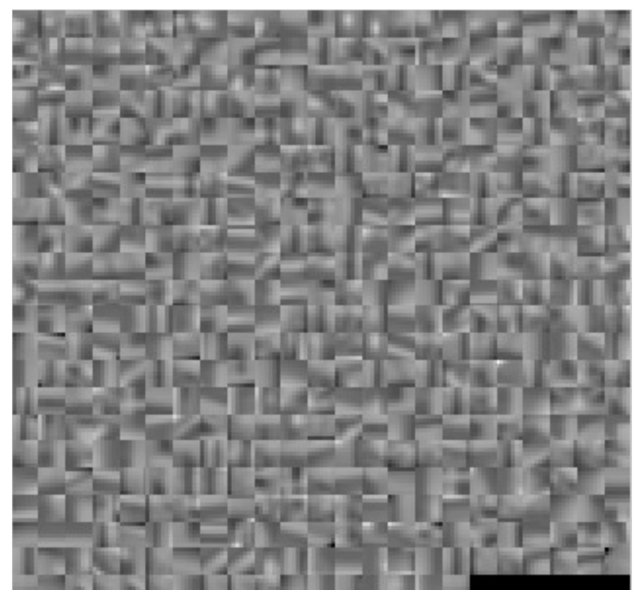
What about outliers?







?

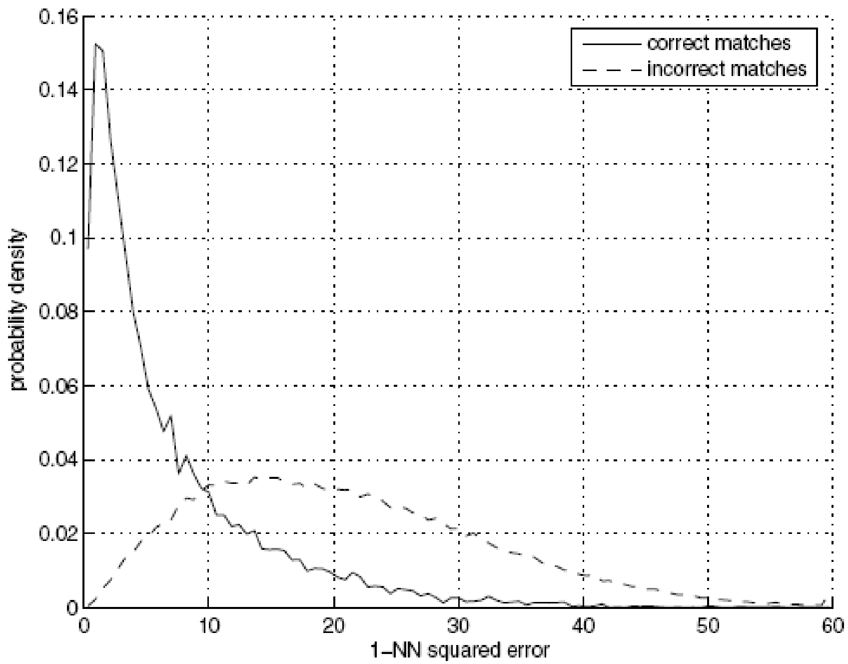


Feature-space outlier rejection

Let's not match all features, but only these that have "similar enough" matches?

How can we do it?

- SSD(patch1,patch2) < threshold
- How to set threshold?



Feature-space outlier rejection: symmetry

Let's not match all features, but only these that have "similar enough" matches?

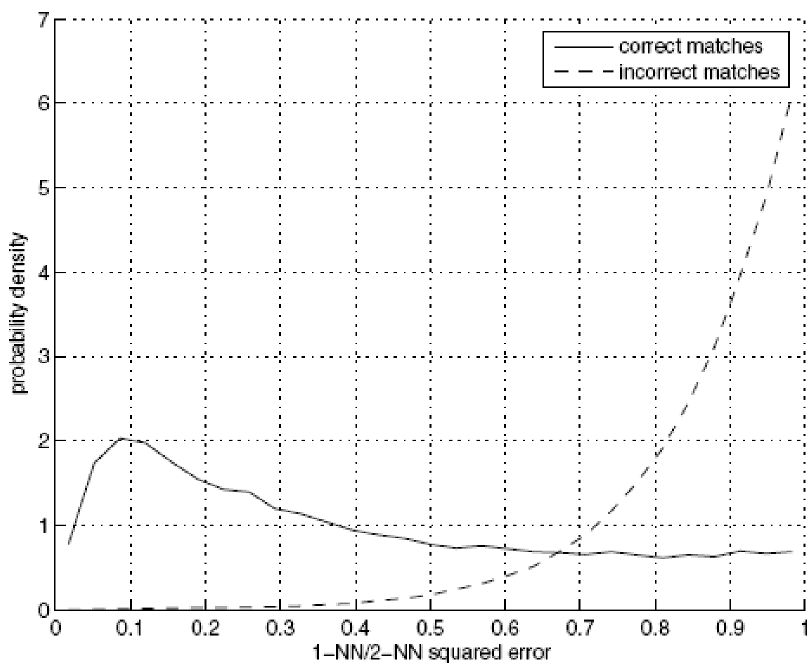
How can we do it?

- Symmetry: x's NN is y, and y's NN is x

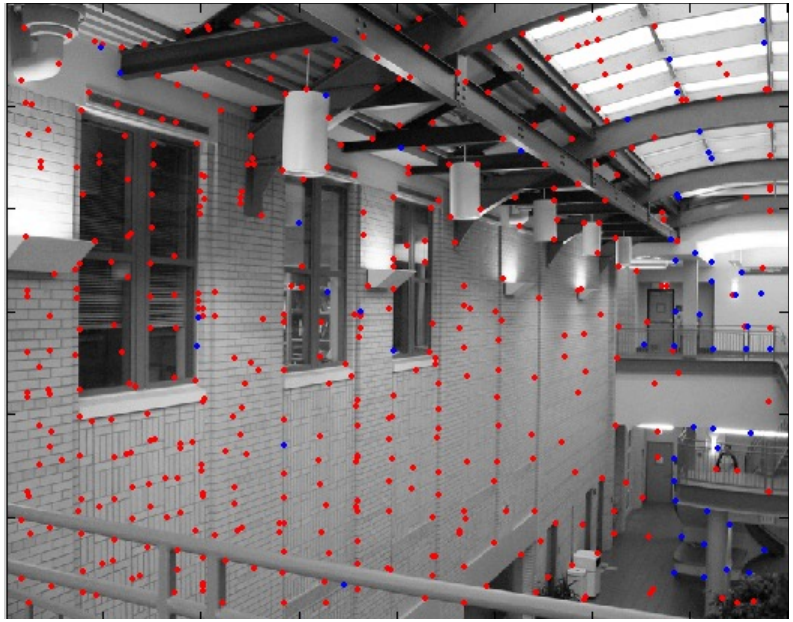
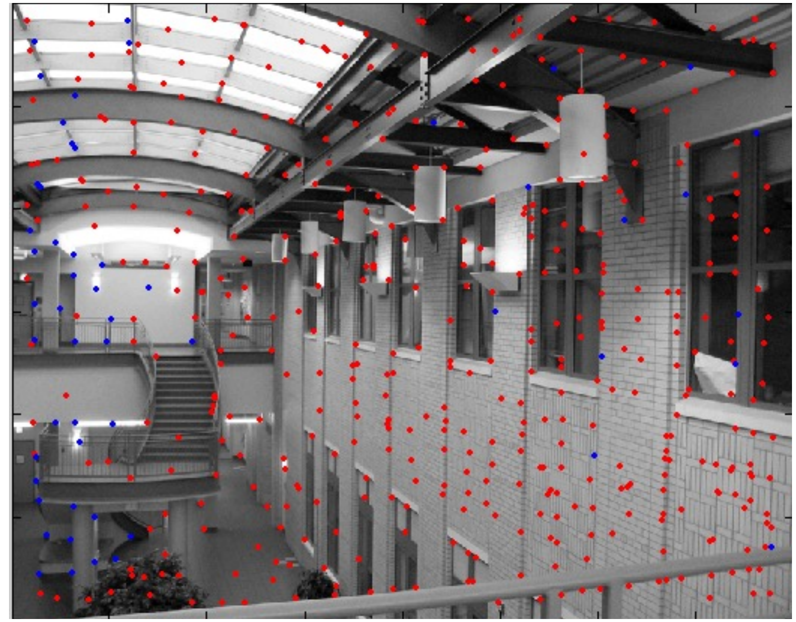
Feature-space outlier rejection: Lowe's trick

A better way [Lowe, 1999]:

- 1-NN: SSD of the closest match
- 2-NN: SSD of the second-closest match
- Look at how much better 1-NN is than 2-NN, e.g. 1-NN/2-NN
- That is, is our best match so much better than the rest?



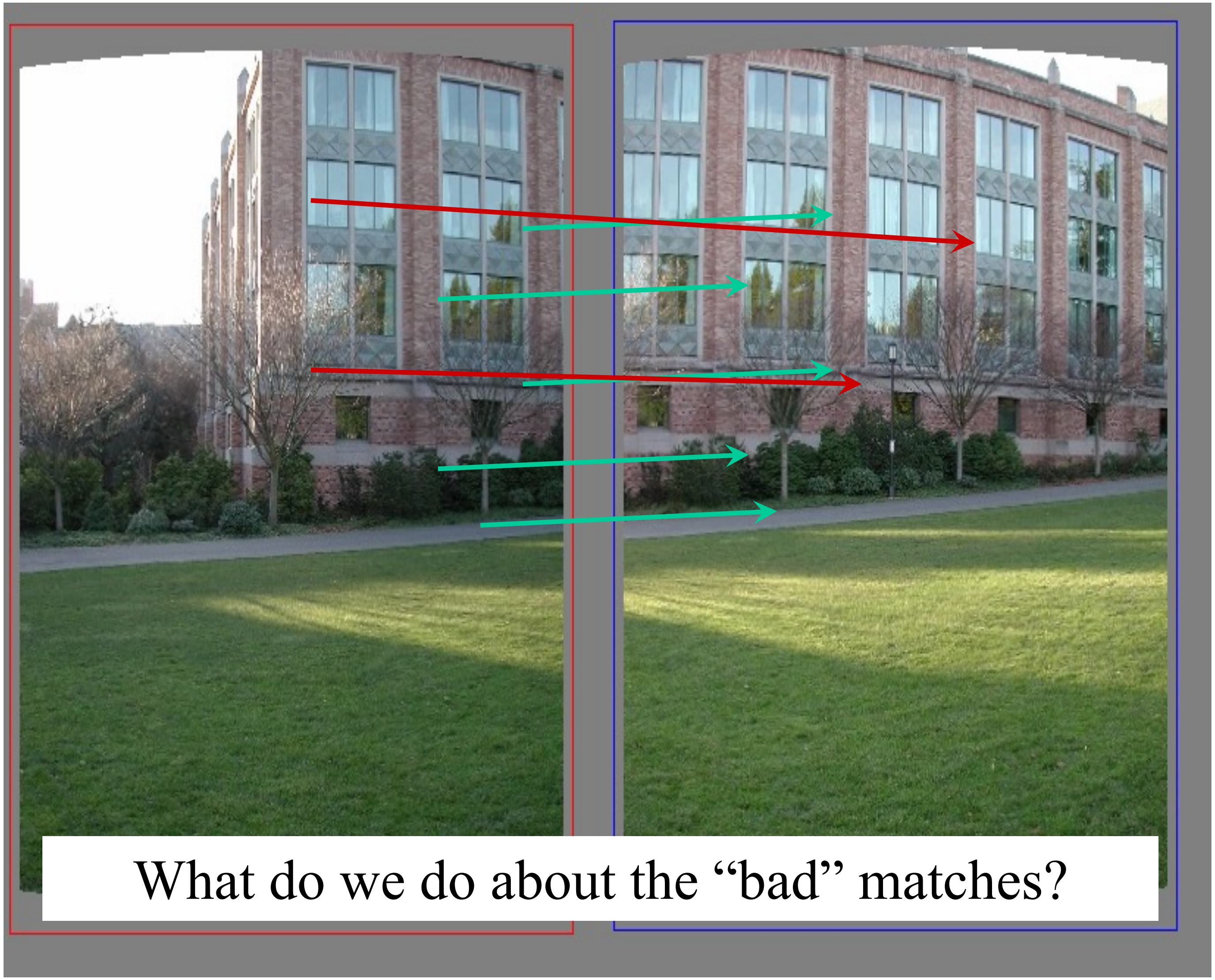
Feature-space outliner rejection

Can we now compute H from the blue points?

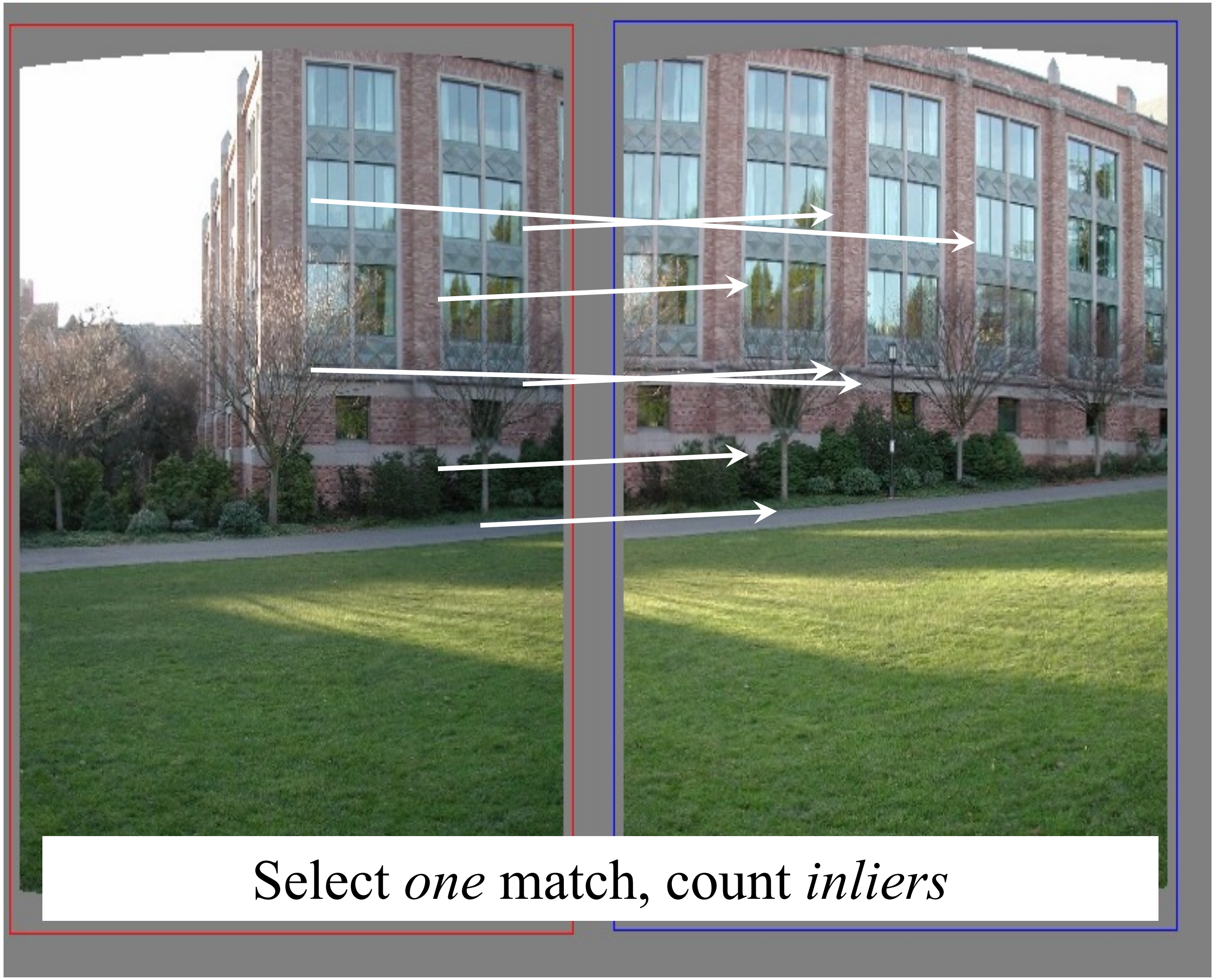
- No! Still too many outliers…
- What can we do?

Matching features



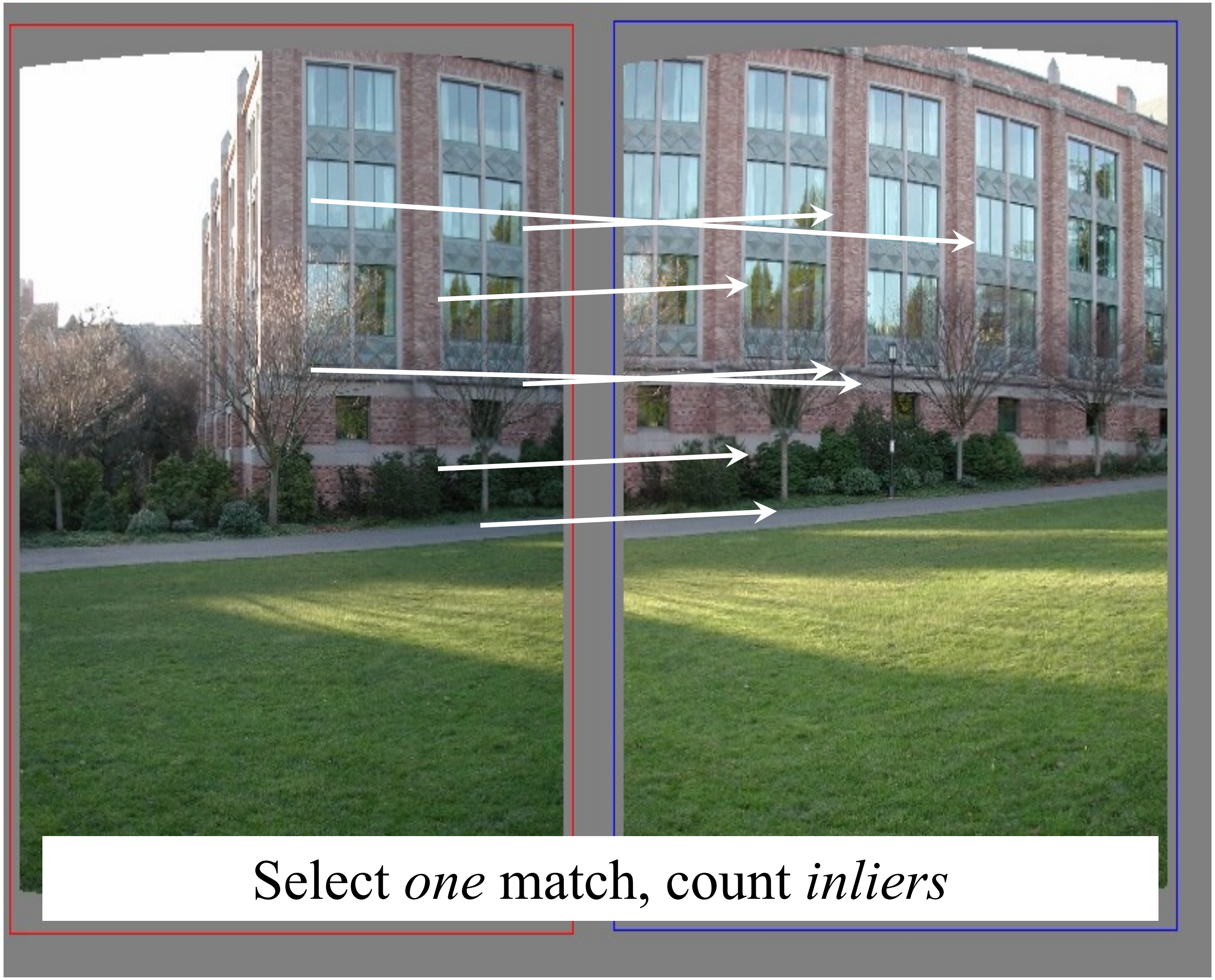
What do we do about the “bad” matches?

RAndom SAmple Consensus



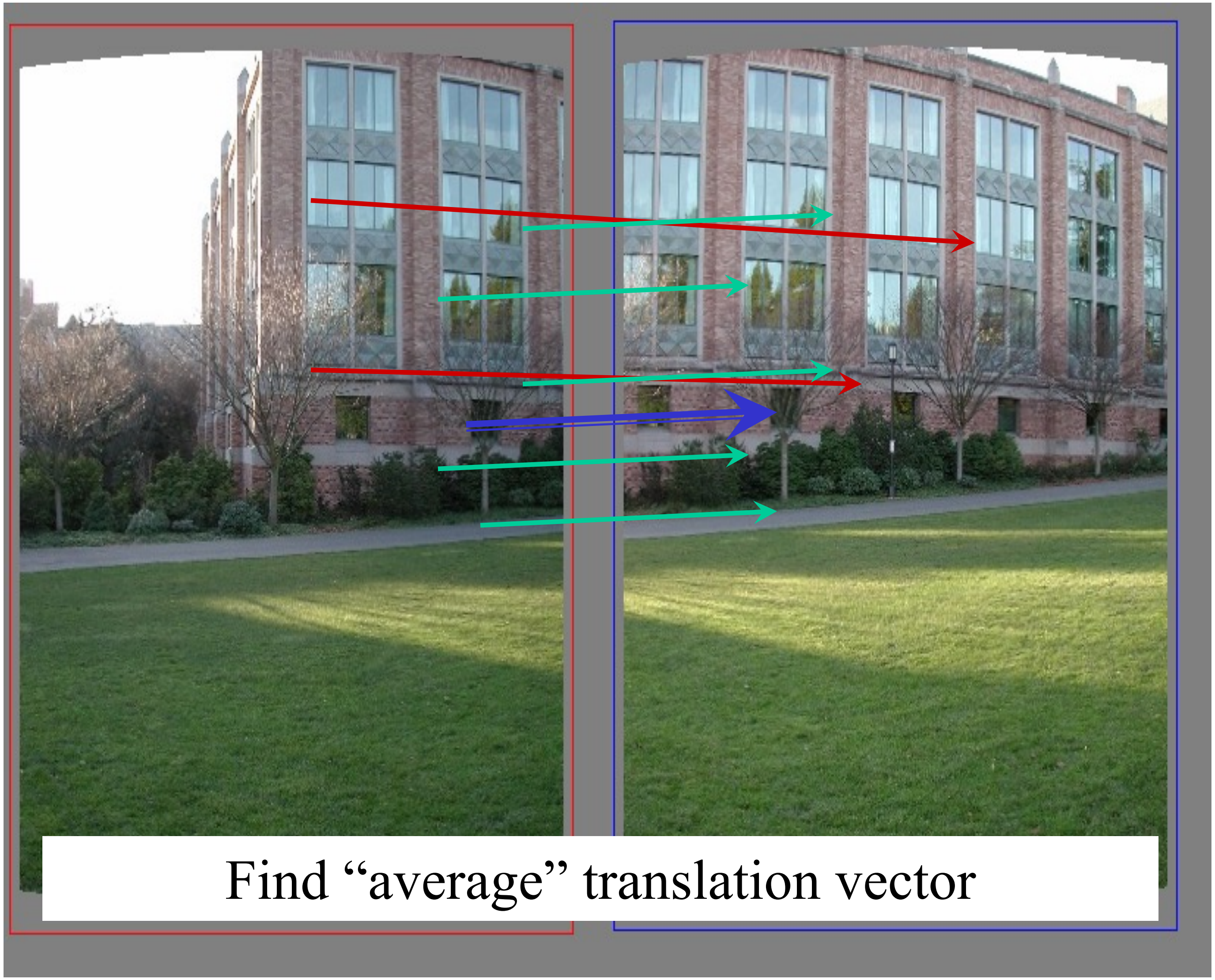
Select *one* match, count *inliers*

RAndom SAmple Consensus



Select *one* match, count *inliers*

Least squares fit



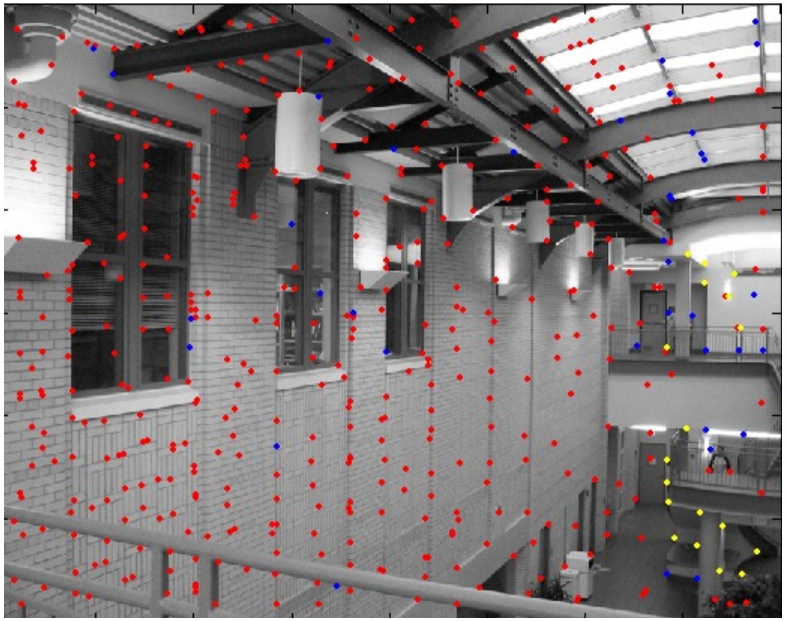
Find “average” translation vector

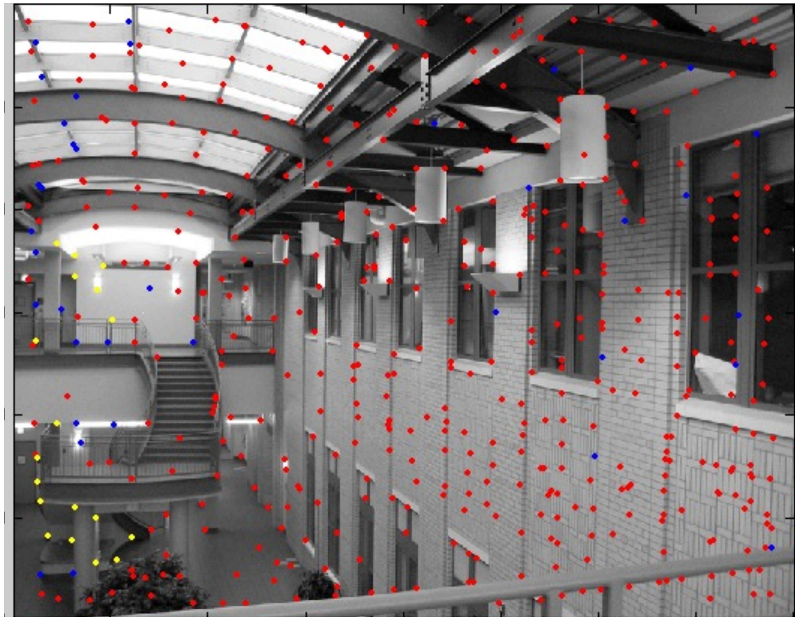
RANSAC for estimating homography

RANSAC loop:

1. Select four feature pairs (at random)
2. Compute homography H (exact)
3. Compute *inliers* where $dist(p_i', \boldsymbol{H}\, p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers

RANSAC

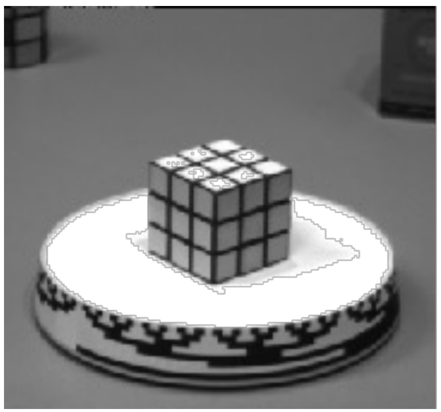


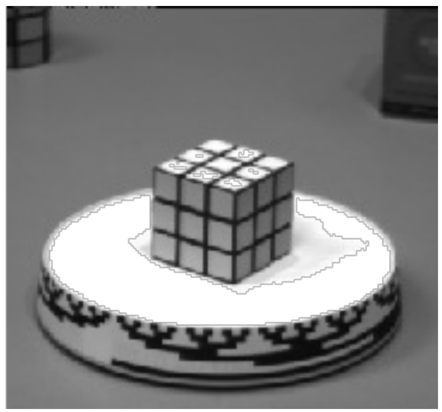


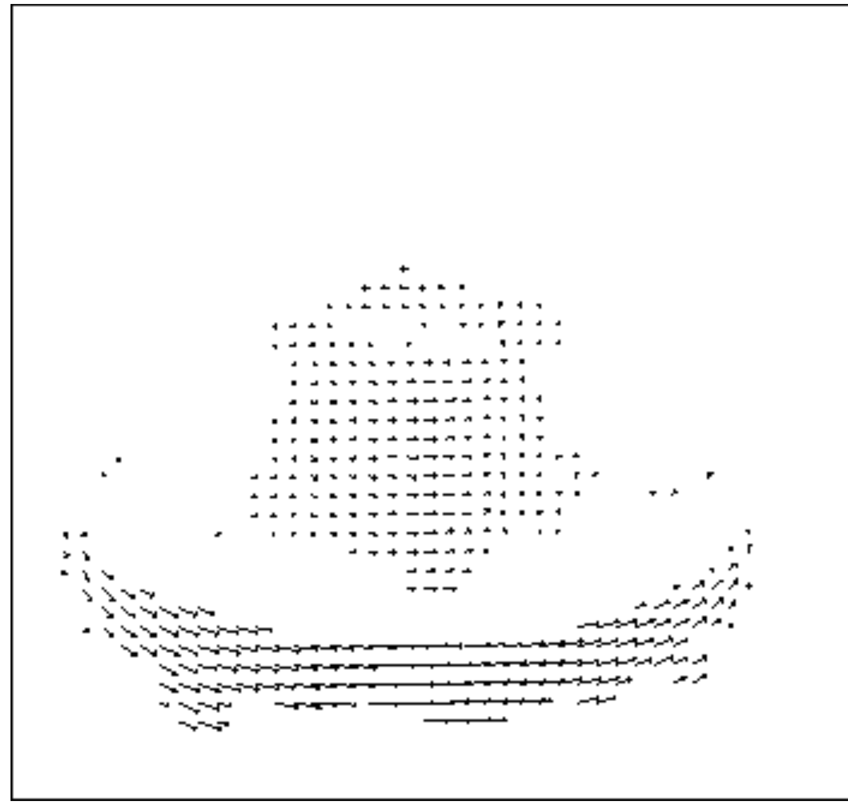
Limitations of Alignment

We need to know the global transform (e.g. affine, homography, etc)

Optical flow







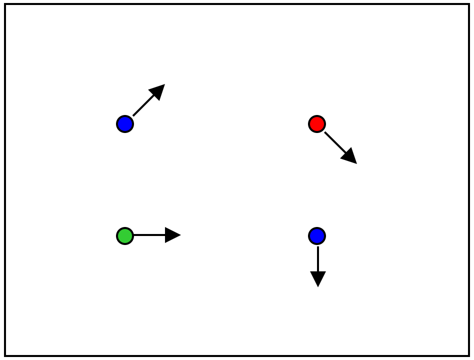
Will start by estimating motion of each pixel separately
Then will consider motion of entire image

Why estimate motion?

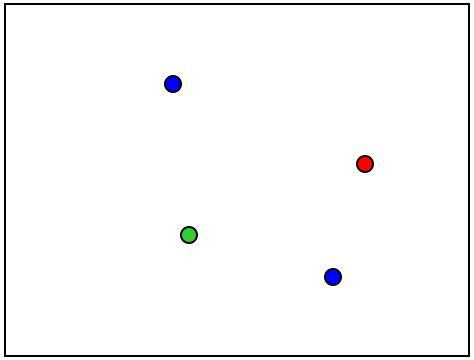
Lots of uses

- Track object behavior
- Correct for camera jitter (stabilization)
- Align images (even if no global transform)
- 3D shape reconstruction
- Special effects

Problem definition: optical flow



$H(x, y)$



$I(x, y)$

How to estimate pixel motion from image H to image I?

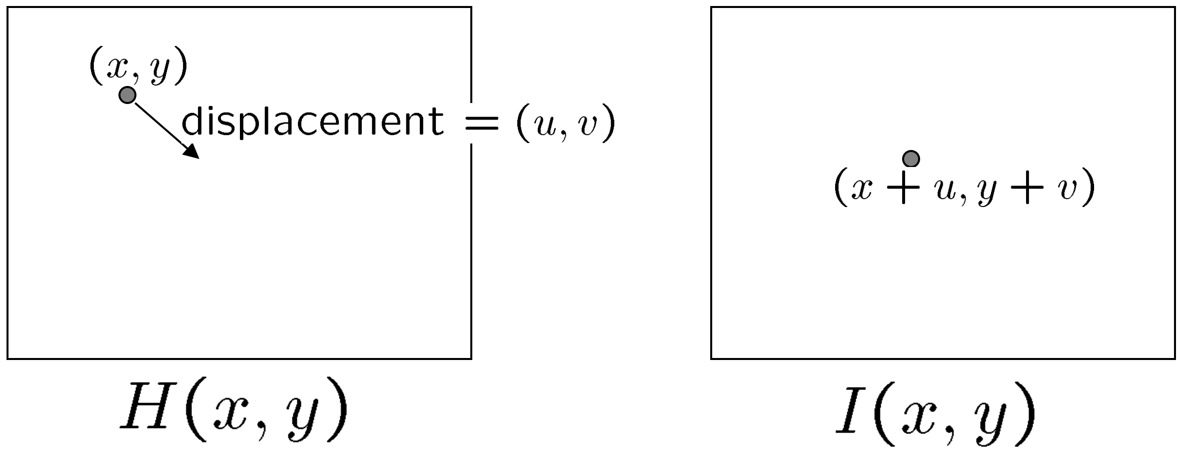
- Solve pixel correspondence problem
  - given a pixel in H, look for nearby pixels of the same color in I

Key assumptions

- **color constancy**: a point in H looks the same in I
  - For grayscale images, this is **brightness constancy**
- **small motion**: points do not move very far

This is called the **optical flow** problem

Optical flow constraints (grayscale images)



Let's look at these constraints more closely

- brightness constancy: Q: what's the equation?

$$0 = I(x+u, y+v) - H(x,y)$$

- small motion: (u and v are less than 1 pixel)
  - suppose we take the Taylor series expansion of I:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

$$\approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

Optical flow equation

Combining these two equations

shorthand: $I_x = \frac{\partial I}{\partial x}$

$$0 = I(x+u, y+v) - H(x, y)$$

$$\approx I(x, y) + I_x u + I_y v - H(x, y)$$

$$\approx (I(x, y) - H(x, y)) + I_x u + I_y v$$

$$\approx I_t + I_x u + I_y v$$

$$\approx I_t + \nabla I \cdot [u \;\; v]$$

In the limit as u and v go to zero, this becomes exact

$$0 = I_t + \nabla I \cdot [\tfrac{\partial x}{\partial t} \;\; \tfrac{\partial y}{\partial t}]$$

Optical flow equation

$$0 = I_t + \nabla I \cdot [u \;\; v]$$

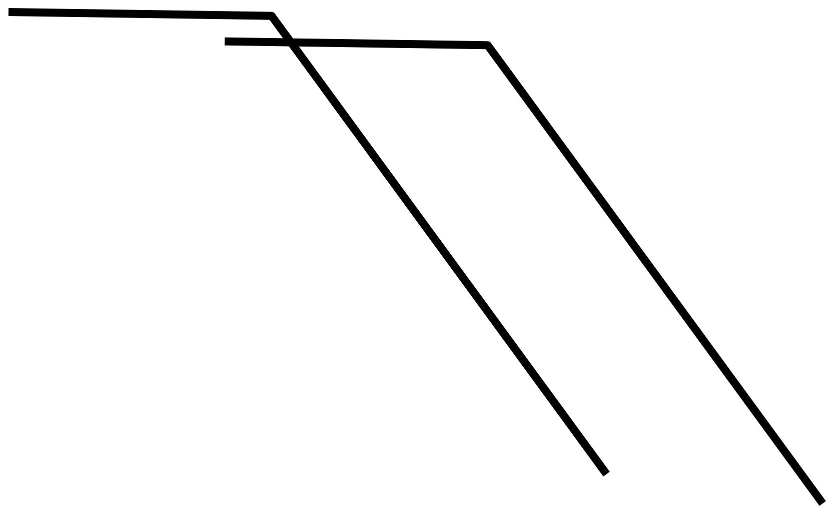
Q: how many unknowns and equations per pixel?

Intuitively, what does this constraint mean?

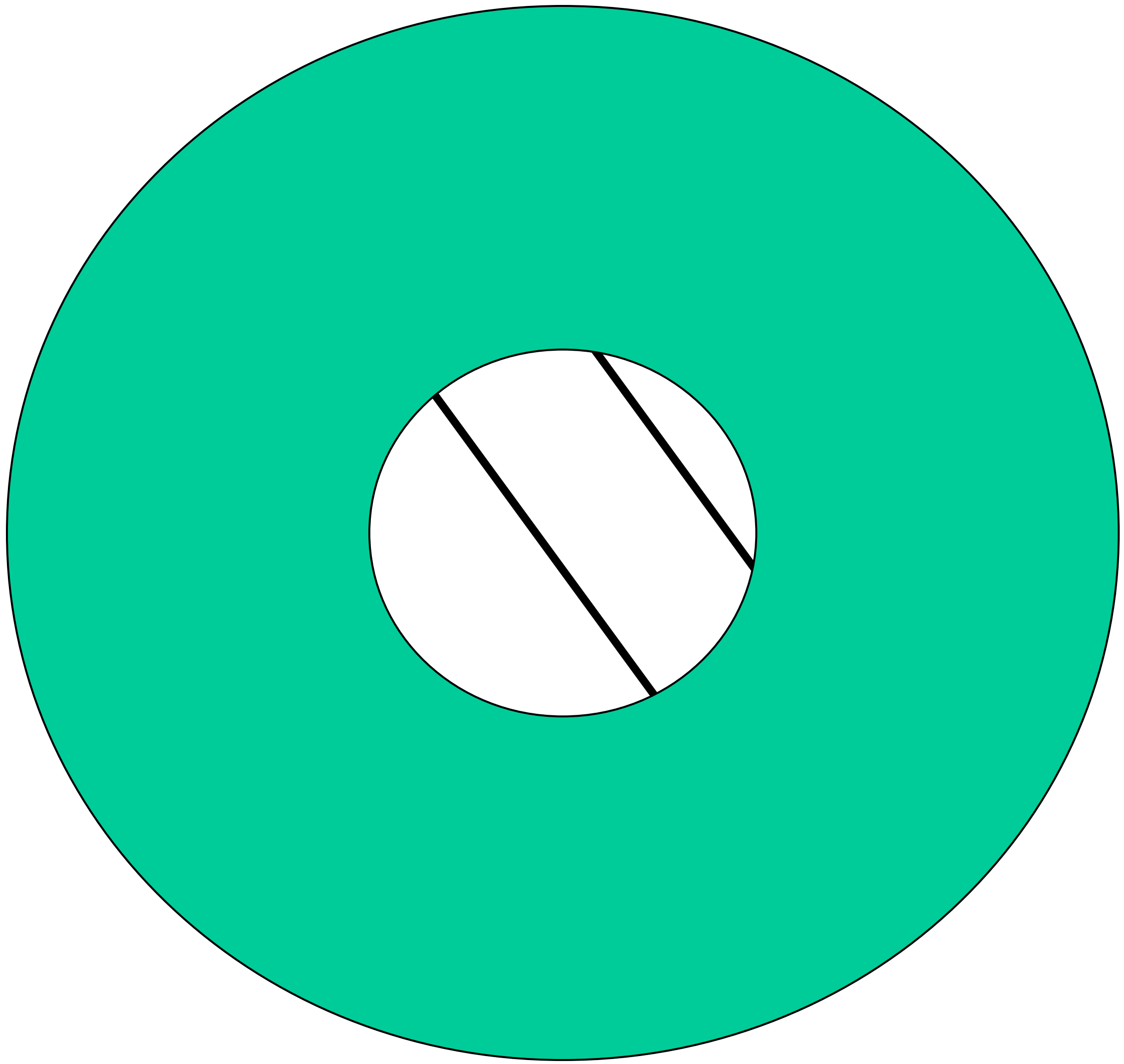
- The component of the flow in the gradient direction is determined
- The component of the flow parallel to an edge is unknown

This explains the Barber Pole illusion
http://www.sandlotscience.com/Ambiguous/barberpole.htm

Aperture problem



Aperture problem



Solving the aperture problem

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!

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u \ \ v]$$

$$\underset{\substack{A \\ 25\times2}}{\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix}} \underset{\substack{d \\ 2\times1}}{\begin{bmatrix} u \\ v \end{bmatrix}} = - \underset{\substack{b \\ 25\times1}}{\begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}}$$

RGB version

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*3 equations per pixel!

$$0 = I_t(\mathbf{p_i})[0,1,2] + \nabla I(\mathbf{p_i})[0,1,2] \cdot [u \;\; v]$$

$$\underbrace{\begin{bmatrix} I_x(\mathbf{p_1})[0] & I_y(\mathbf{p_1})[0] \\ I_x(\mathbf{p_1})[1] & I_y(\mathbf{p_1})[1] \\ I_x(\mathbf{p_1})[2] & I_y(\mathbf{p_1})[2] \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}})[0] & I_y(\mathbf{p_{25}})[0] \\ I_x(\mathbf{p_{25}})[1] & I_y(\mathbf{p_{25}})[1] \\ I_x(\mathbf{p_{25}})[2] & I_y(\mathbf{p_{25}})[2] \end{bmatrix}}_{\substack{A \\ 75\times 2}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\substack{d \\ 2\times 1}} = - \underbrace{\begin{bmatrix} I_t(\mathbf{p_1})[0] \\ I_t(\mathbf{p_1})[1] \\ I_t(\mathbf{p_1})[2] \\ \vdots \\ I_t(\mathbf{p_{25}})[0] \\ I_t(\mathbf{p_{25}})[1] \\ I_t(\mathbf{p_{25}})[2] \end{bmatrix}}_{\substack{b \\ 75\times 1}}$$

Lukas-Kanade flow

Prob: we have more equations than unknowns

$$\underset{25\times 2}{A}\ \underset{2\times 1}{d} = \underset{25\times 1}{b} \longrightarrow \text{minimize } \|Ad - b\|^2$$

Solution: solve least squares problem

- minimum least squares solution given by solution (in d) of:

$$\underset{2\times 2}{(A^T A)}\ \underset{2\times 1}{d} = \underset{2\times 1}{A^T b}$$

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

- The summations are over all pixels in the K x K window
- This technique was first proposed by Lukas & Kanade (1981)

Conditions for solvability

- Optimal (u, v) satisfies Lucas-Kanade equation

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

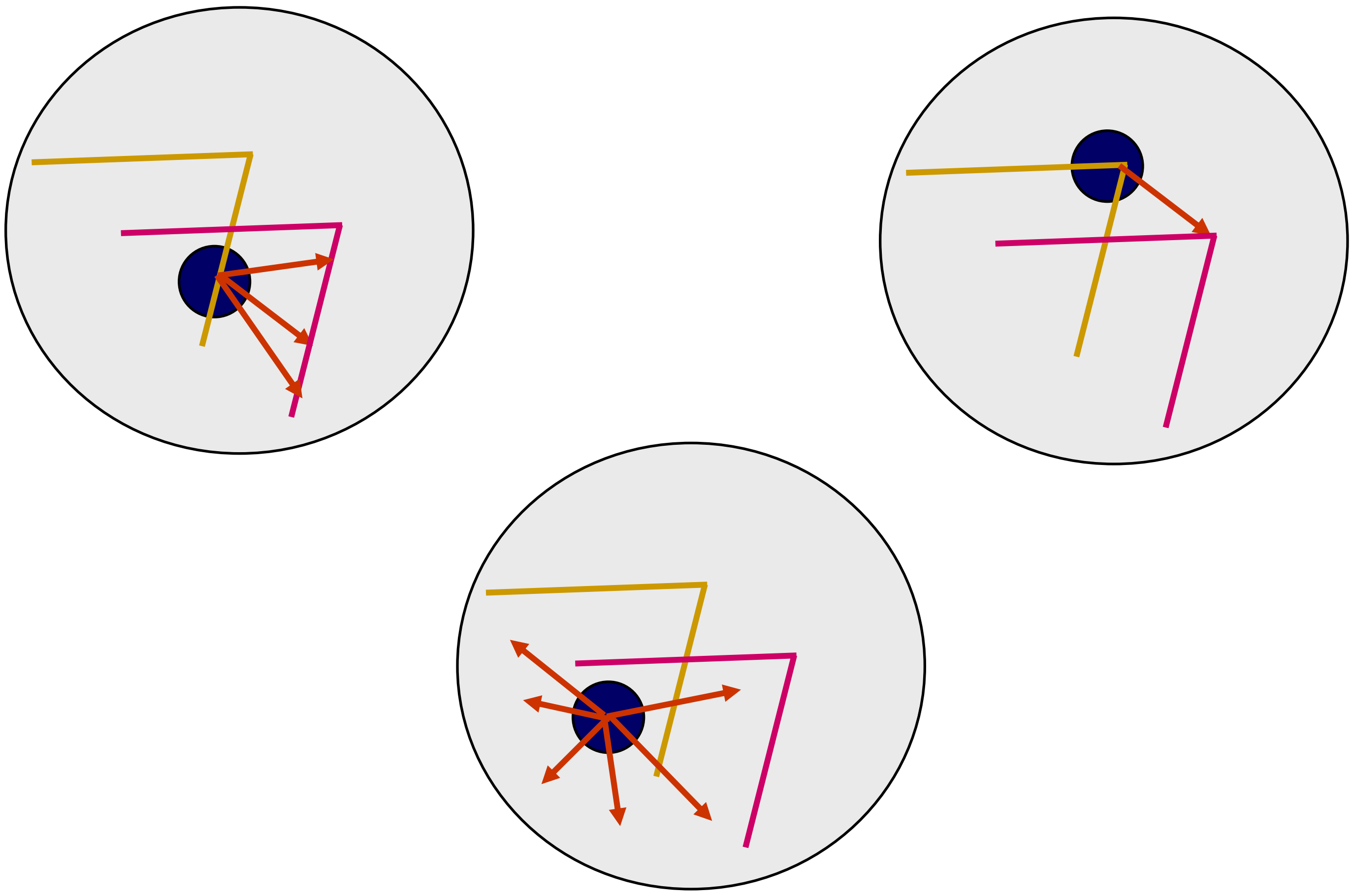
When is This Solvable?

- **A^T^A** should be invertible
- **A^T^A** should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of $\mathbf{A^TA}$ should not be too small
- $\mathbf{A^TA}$ should be well-conditioned
  - λ_1/λ_2 should not be too large (λ_1 = larger eigenvalue)

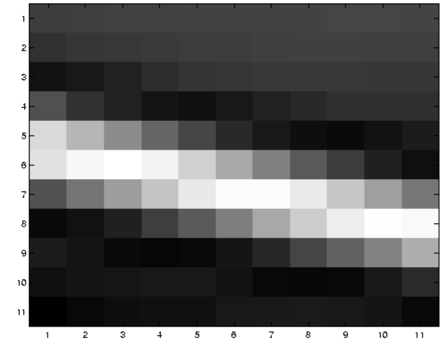
$\mathbf{A^TA}$ is solvable when there is no aperture problem

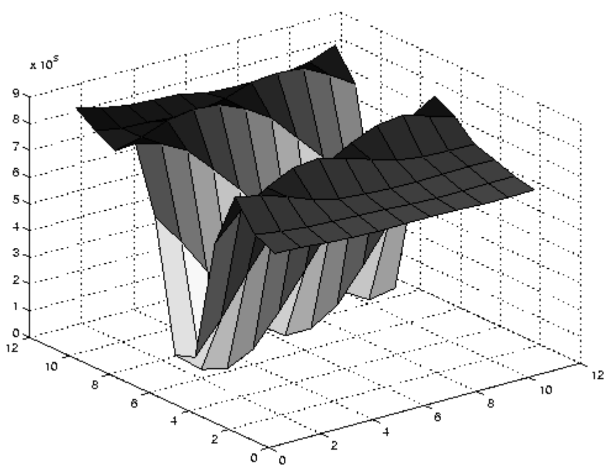
$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] = \sum \nabla I (\nabla I)^T$$

Local Patch Analysis



Edge

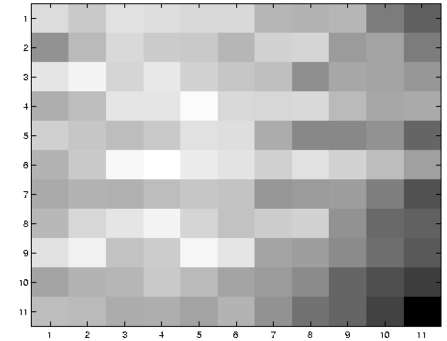


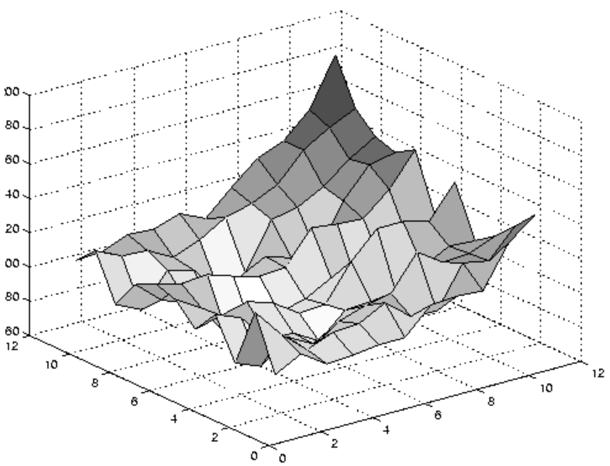


$$\sum \nabla I (\nabla I)^T$$

- – large gradients, all the same
- – large λ_1, small λ_2

Low texture region



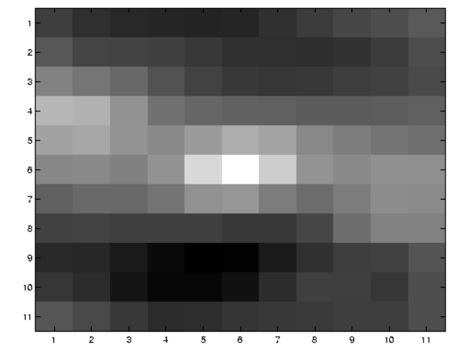


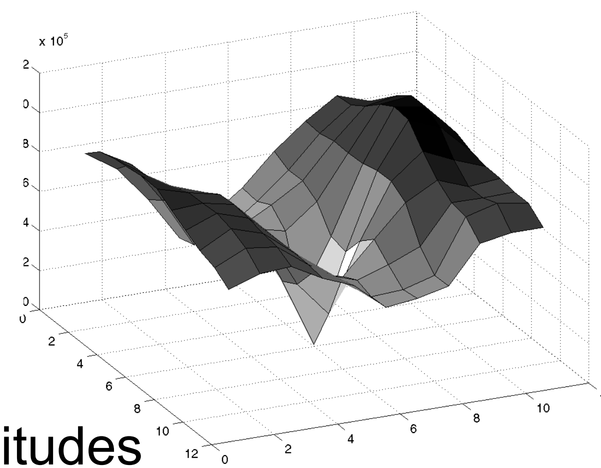
$$\sum \nabla I (\nabla I)^T$$

– gradients have small magnitude

– small λ_1, small λ_2

High textured region





$$\sum \nabla I (\nabla I)^T$$

– gradients are different, large magnitudes

– large λ_1, large λ_2

Observation

This is a two image problem BUT

- Can measure sensitivity by just looking at one of the images!
- This tells us which pixels are easy to track, which are hard
  - very useful later on when we do feature tracking...

Errors in Lukas-Kanade

What are the potential causes of errors in this procedure?

- Suppose A^TA is easily invertible
- Suppose there is not much noise in the image

When our assumptions are violated

- Brightness constancy is **not** satisfied
- The motion is **not** small
- A point does **not** move like its neighbors
  - window size is too large
  - what is the ideal window size?

Iterative Refinement

Iterative Lukas-Kanade Algorithm

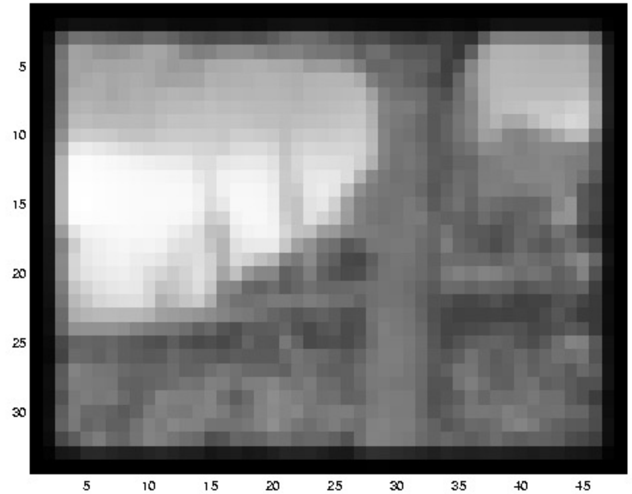
1. Estimate velocity at each pixel by solving Lucas-Kanade equations
2. Warp H towards I using the estimated flow field
   - *use image warping techniques*
3. Repeat until convergence

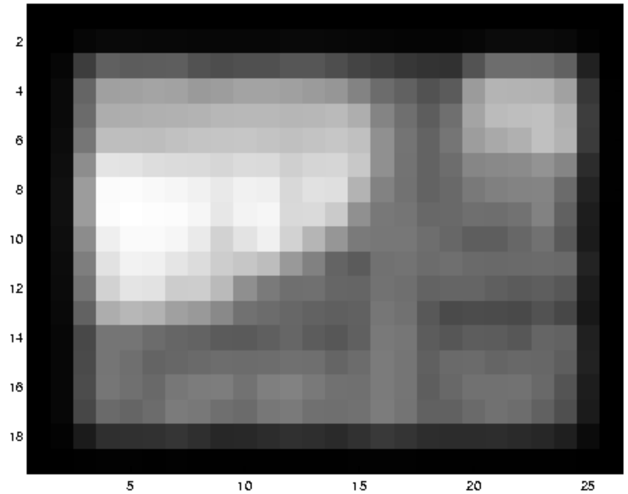
Revisiting the small motion assumption

Is this motion small enough?

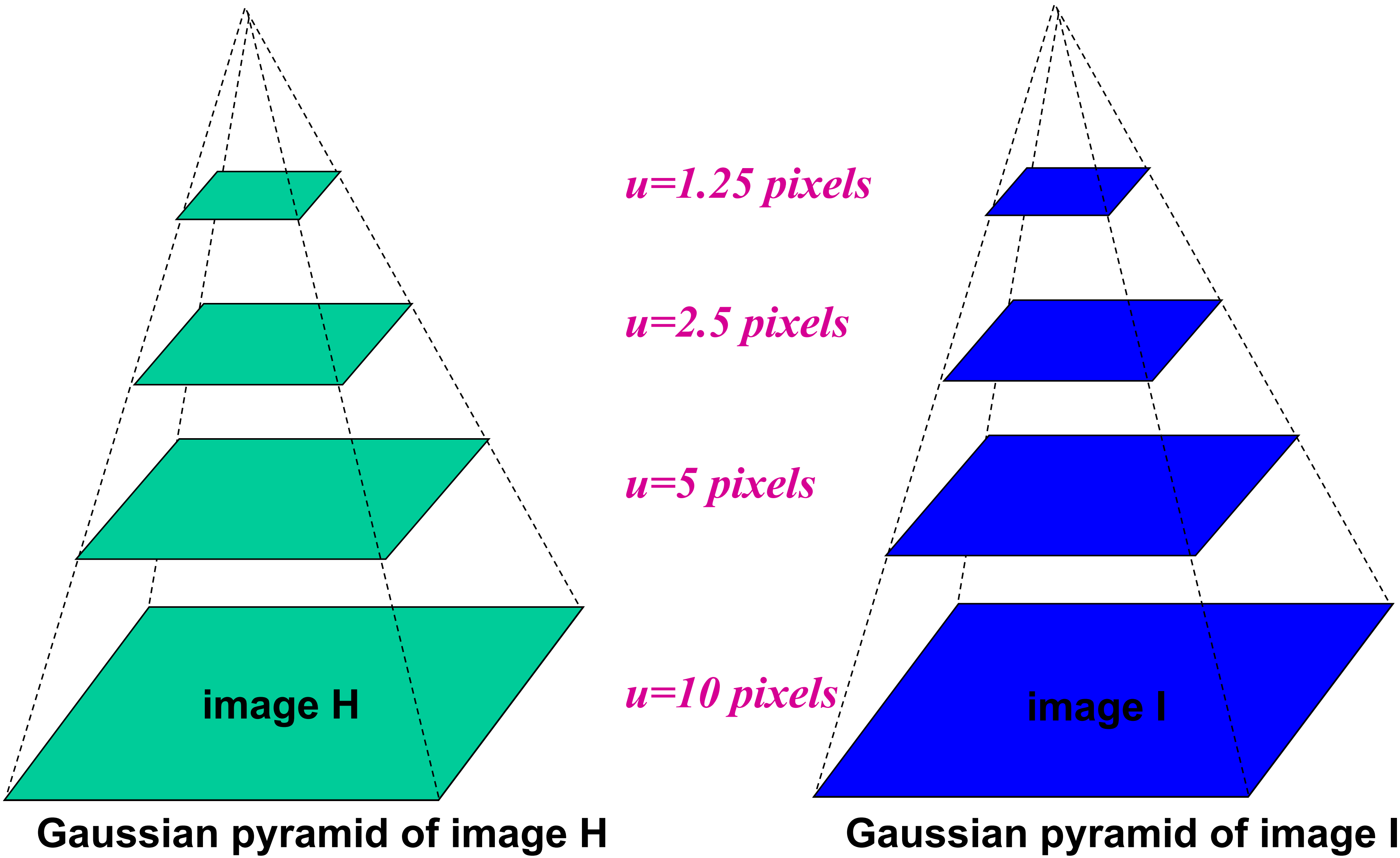
- Probably not—it’s much larger than one pixel (2^{nd} order terms dominate)
- How might we solve this problem?

Reduce the resolution!





Coarse-to-fine optical flow estimation



Coarse-to-fine optical flow estimation