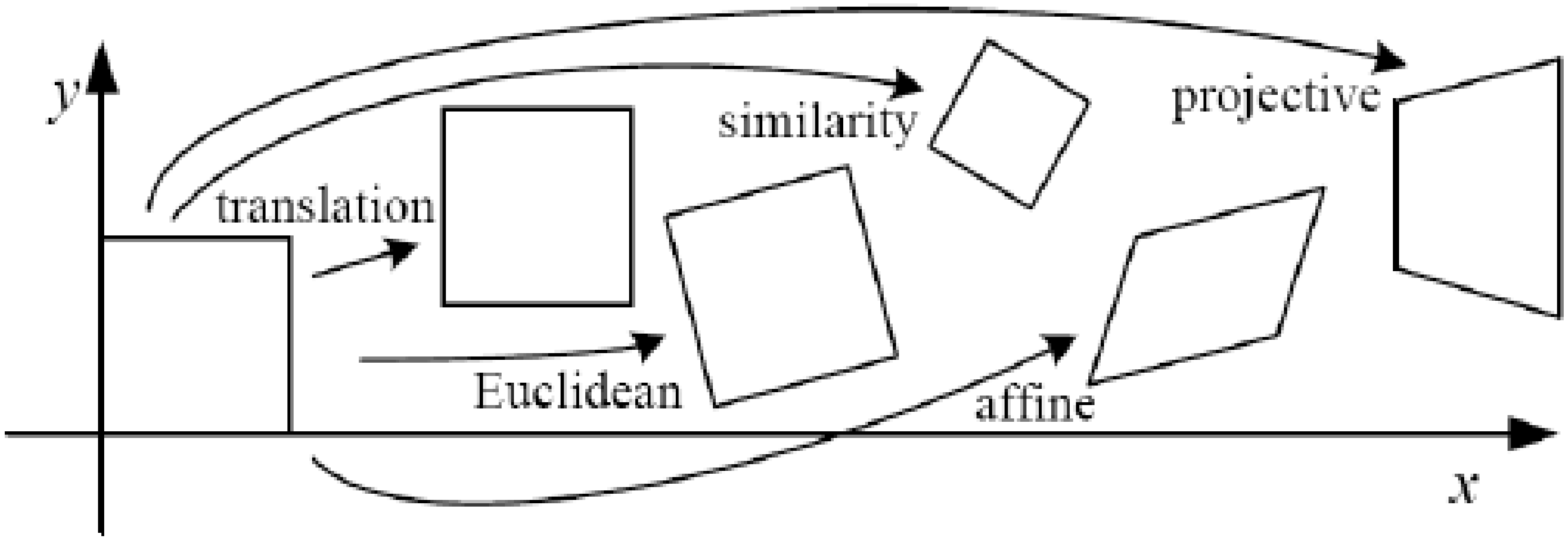
Video Signals

ROBUST FEATURES

Taxonomy of 2D correspondence maps



Correspondence maps

- Notation:
  - Homogeneous coordinates; reference image $\underline{\mathbf{x}} = (x \quad y \quad 1)^T$
  - Inhomogeneous coordinates; input image $\mathbf{x}' = (x' \quad y')^T$
- Translation

$$\mathbf{x}' = \mathbf{x} + \mathbf{t} \quad \text{or} \quad \mathbf{x}' = [\mathbf{I} \quad \mathbf{t}]\underline{\mathbf{x}}$$

- Euclidean transformation (rotation and translation)

$$\mathbf{x}' = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \end{bmatrix} \underline{\mathbf{x}}$$

- Scaled rotation (similarity transform)

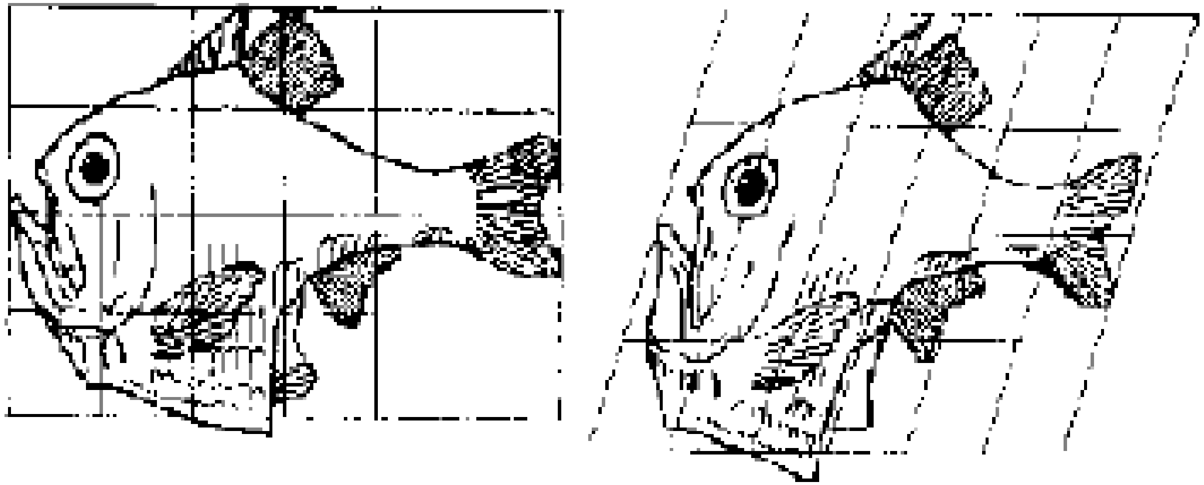
$$\mathbf{x}' = \begin{bmatrix} s\cdot\cos\theta & -s\cdot\sin\theta & t_x \\ s\cdot\sin\theta & s\cdot\cos\theta & t_y \end{bmatrix} \underline{\mathbf{x}}$$

Correspondence maps

- Affine transformation

$$\mathbf{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \underline{\mathbf{x}}$$

- Motion of planar surface in 3d under orthographic projection
- Parallel lines are preserved



Argyropelecus olfersi. *Sternoptyx diaphana.*

Correspondence maps

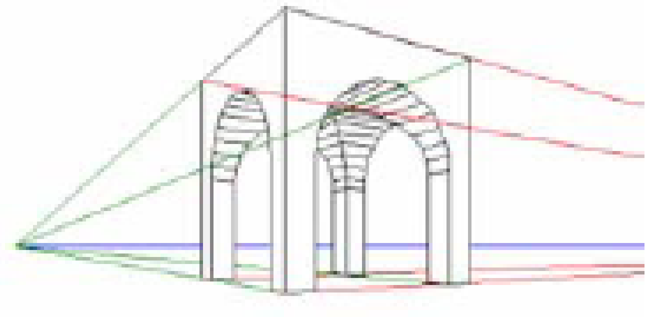
- Perspective transformation (homography); homogeneous coordinates

$$\underline{\mathbf{x}}' \sim \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \underline{\mathbf{x}}$$

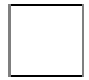
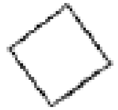
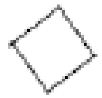
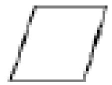
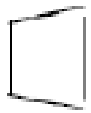
- Inhomogeneous coordinates (after normalization)

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \qquad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}$$

- Motion of planar surface in 3d under perspective projection
- Straight lines are preserved



2d correspondence maps - summary

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times 3}$	2	orientation $+ \cdots$	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times 3}$	3	lengths $+ \cdots$	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times 3}$	4	angles $+ \cdots$	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times 3}$	6	parallelism $+ \cdots$	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times 3}$	8	straight lines	

How to find the correspondence map?

- Direct methods
  - Calculate an error metric to determine similarity between input image $f(x,y)$ and warped reference image $g(x+\Delta x, y+\Delta y)$
  - Based on pixel values, uses all information in the image
  - Search or gradient-based method in model parameter space
- Feature-based methods
  - Identify location of robust feature points
  - Establish feature correspondences based on a feature characteristics
  - Obtain model parameters from feature correspondences

Displacement Estimation by Block Matching

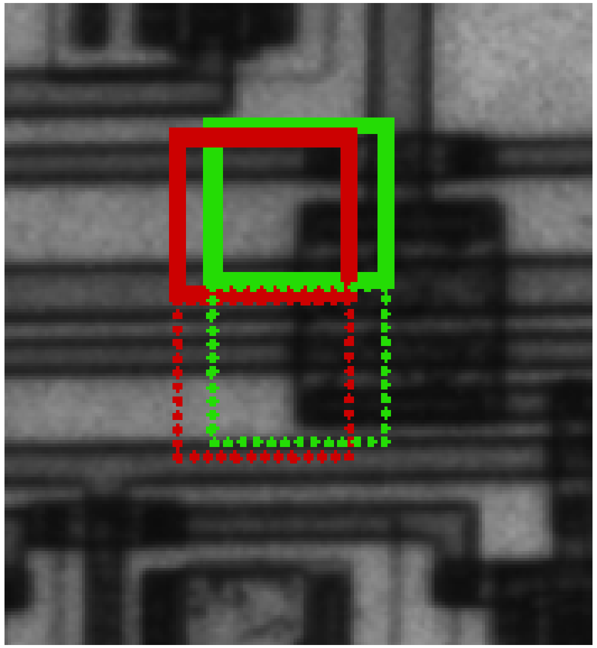


Image $g(x,y)$

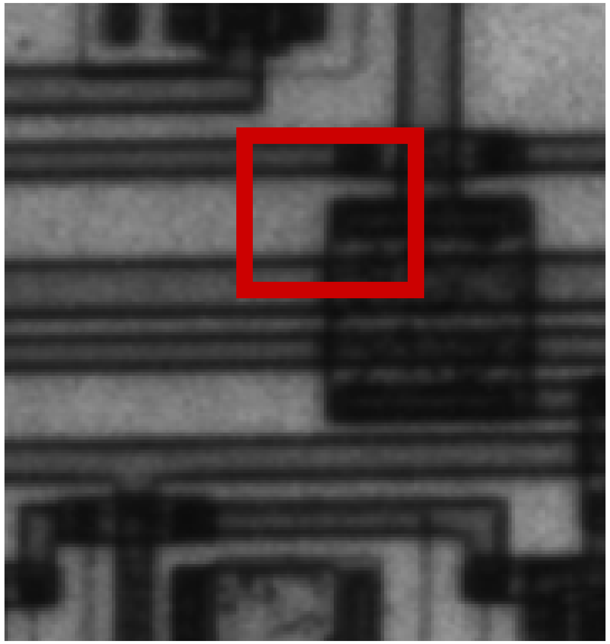


Image $f(x,y)$

. . . process repeated for another measurement window position.

Integer Pixel Shifts

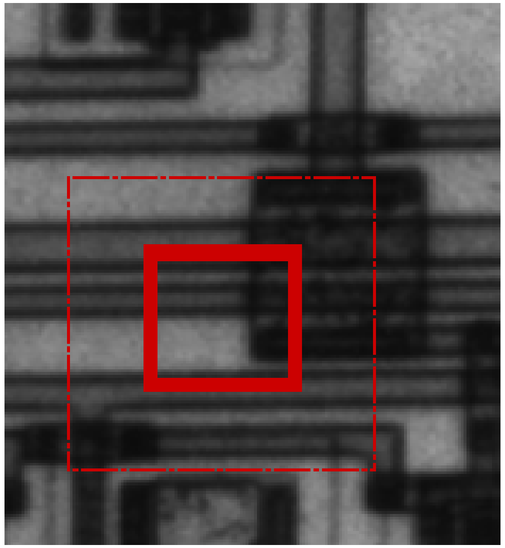


Image $g(x,y)$

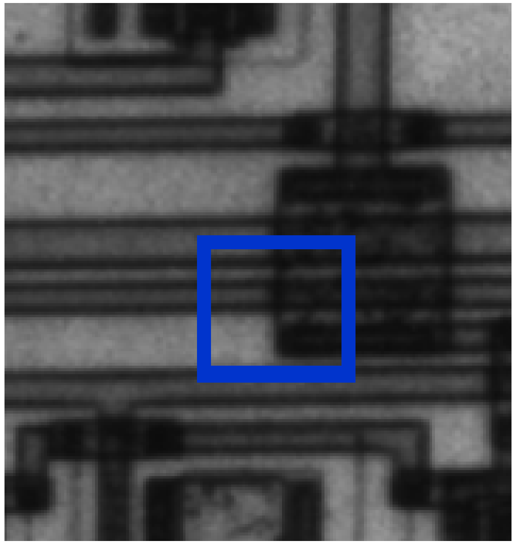
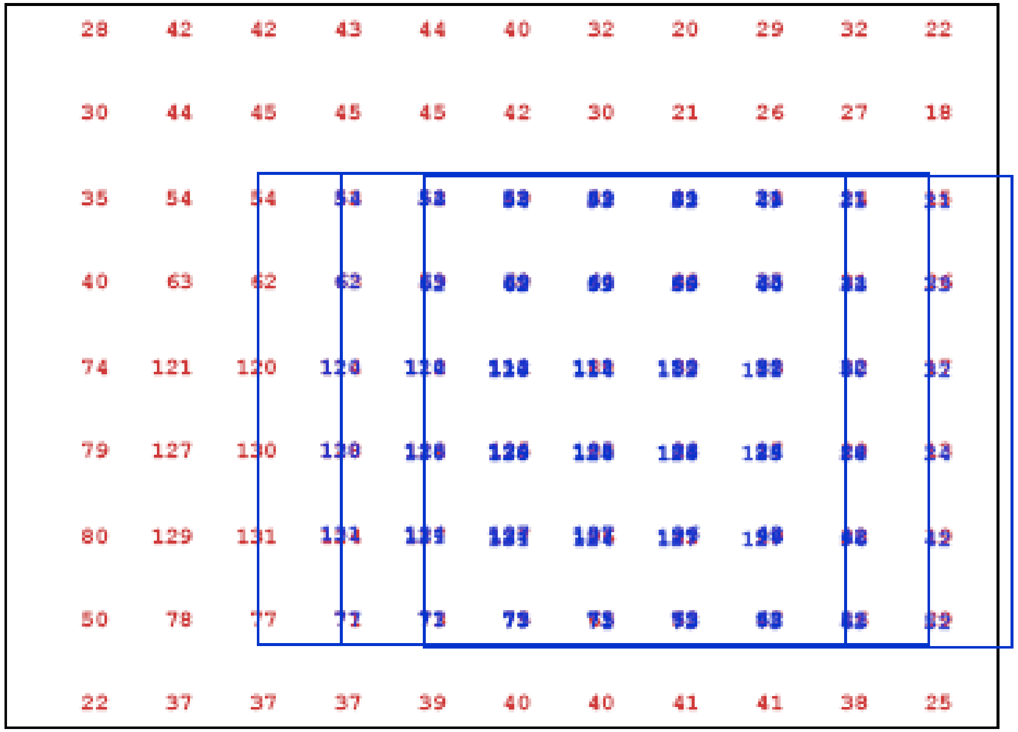


Image $f(x,y)$

Measurement window is compared with a shifted array of pixels in the other image, to determine the best match

Rectangular array of pixels is selected as a measurement window

Integer Pixel Shifts

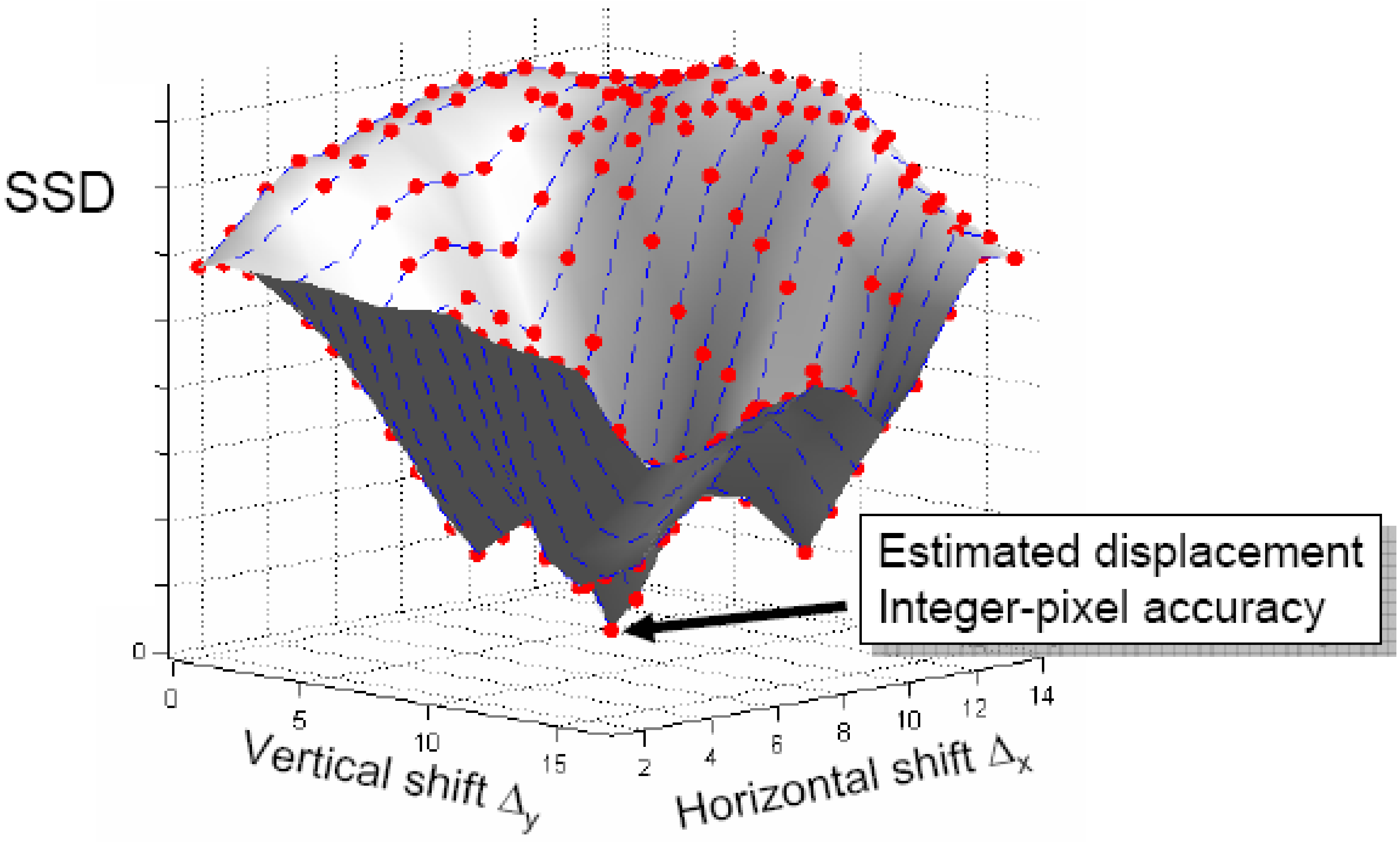


Measurement window is compared with a shifted array of pixels in the other image, to determine the best match

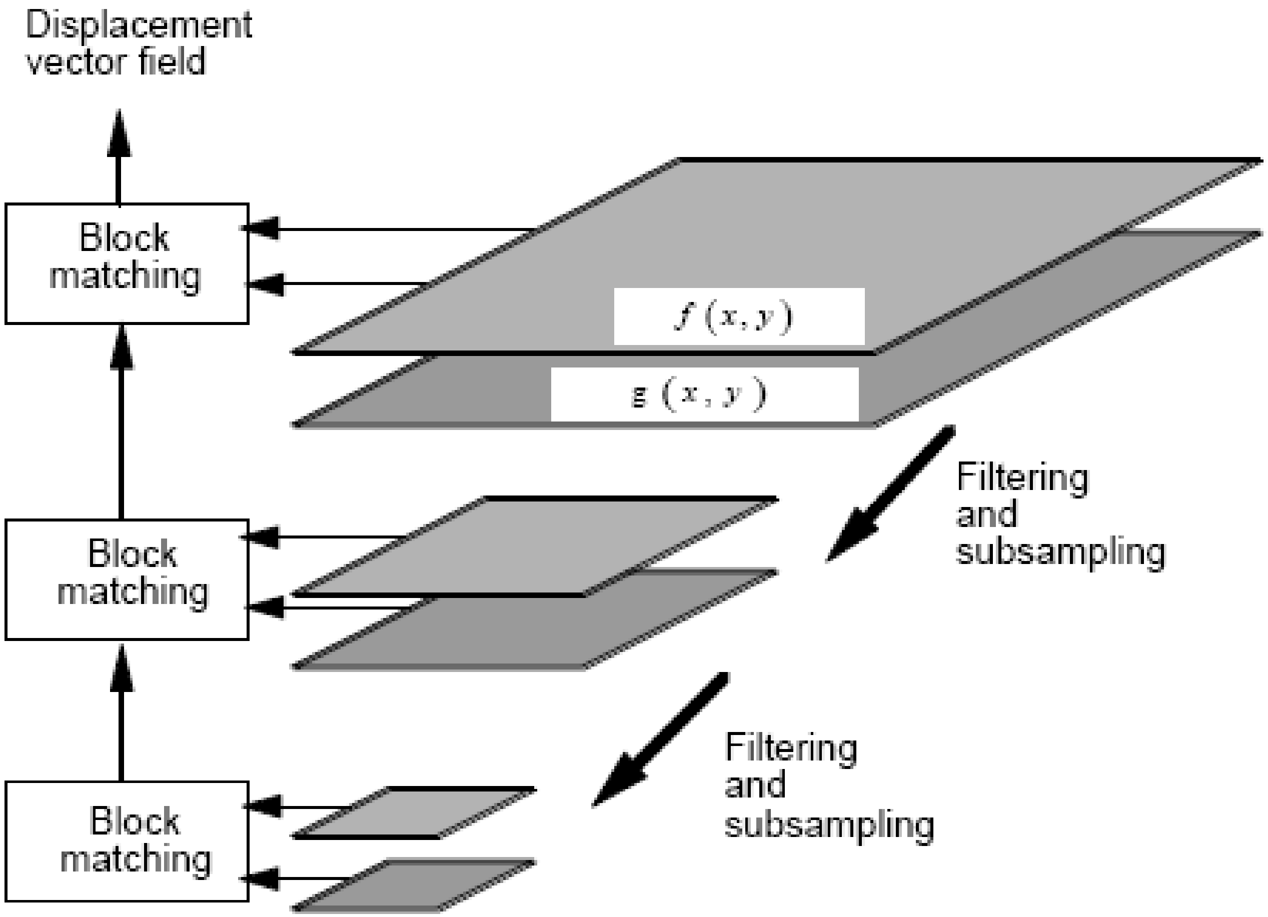
54	53	52	49	31	21
62	63	59	60	44	33
120	114	112	111	80	32
130	128	124	125	88	24
131	124	127	127	96	42
77	71	73	75	63	52

Rectangular array of pixels is selected as a measurement window

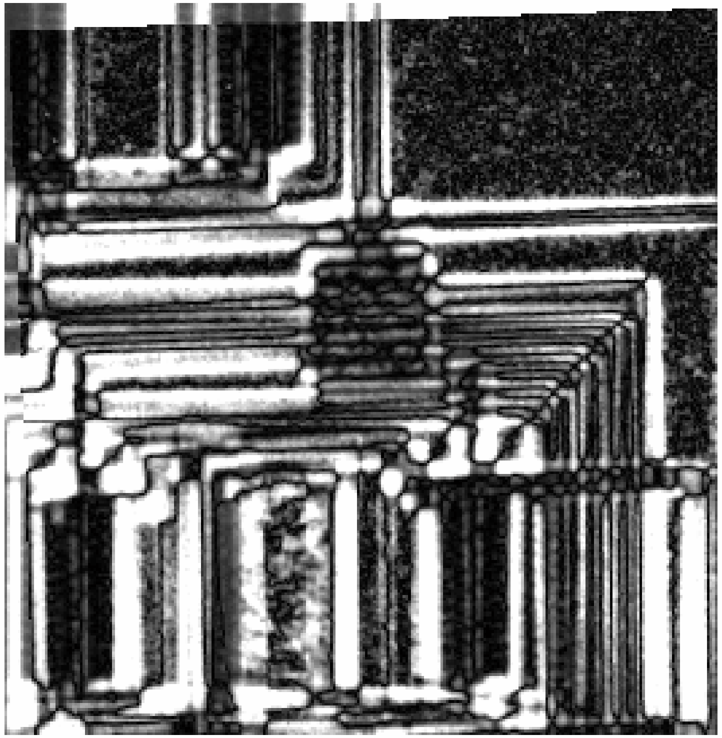
SSD Values Resulting from Block Matching



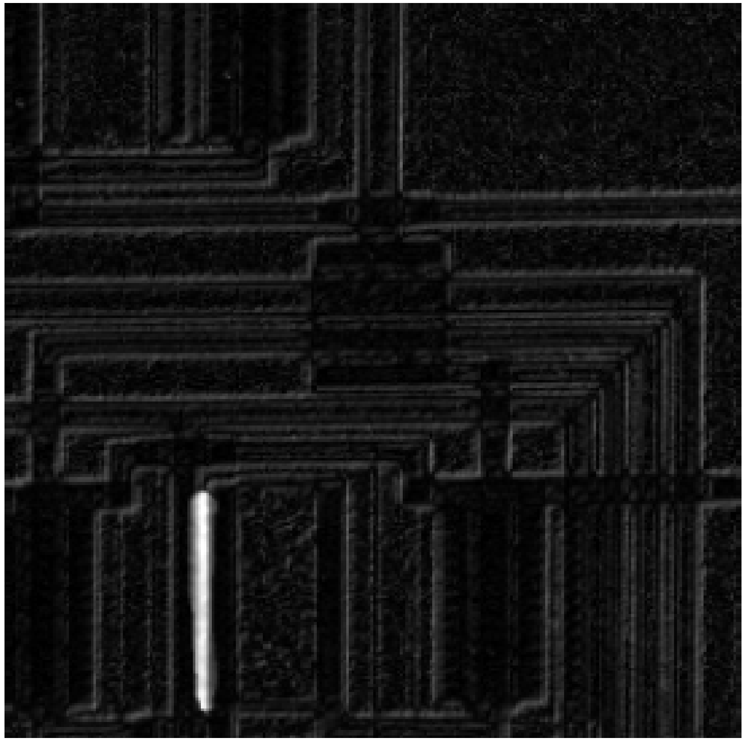
Multi-scale block matching



Absolute difference between images

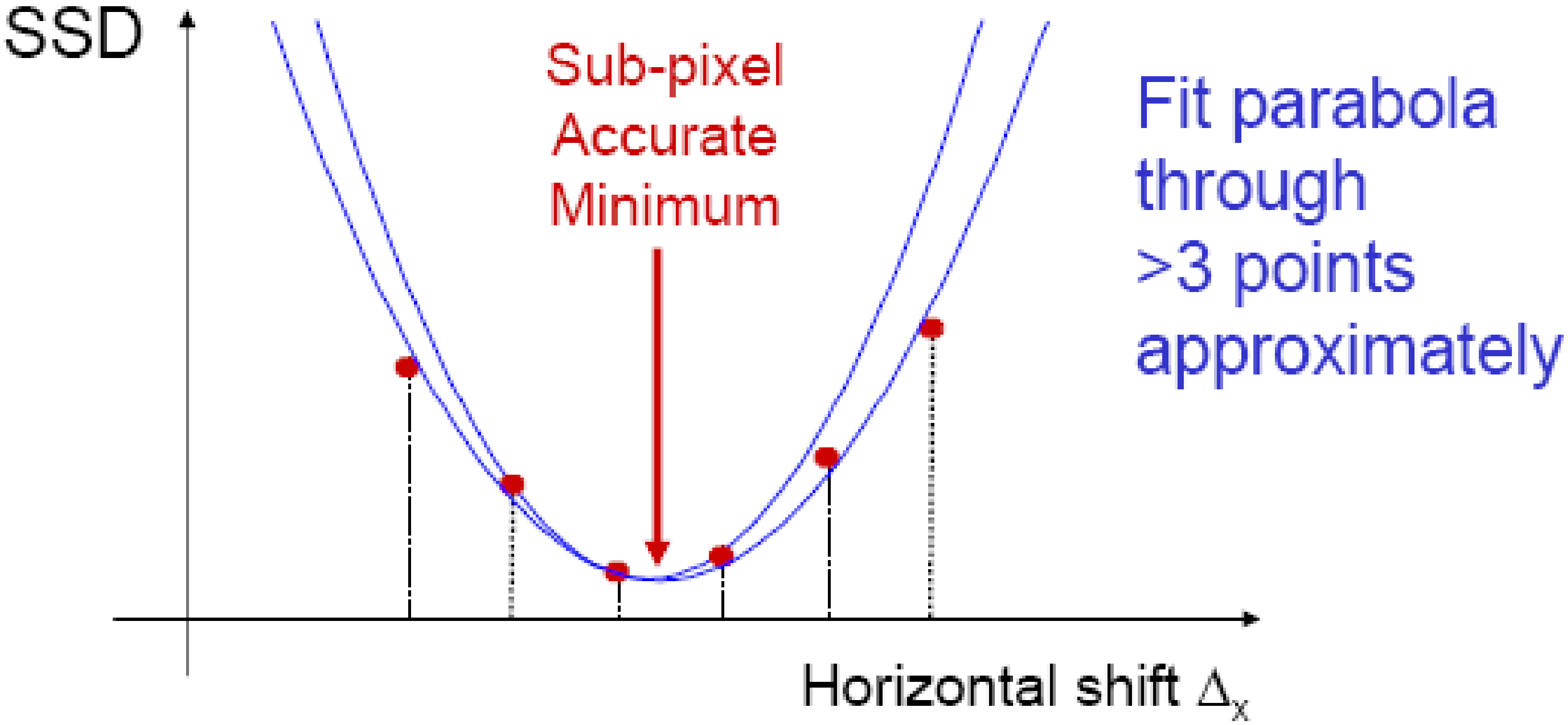


w/o alignment



w/ integer-pixel alignment

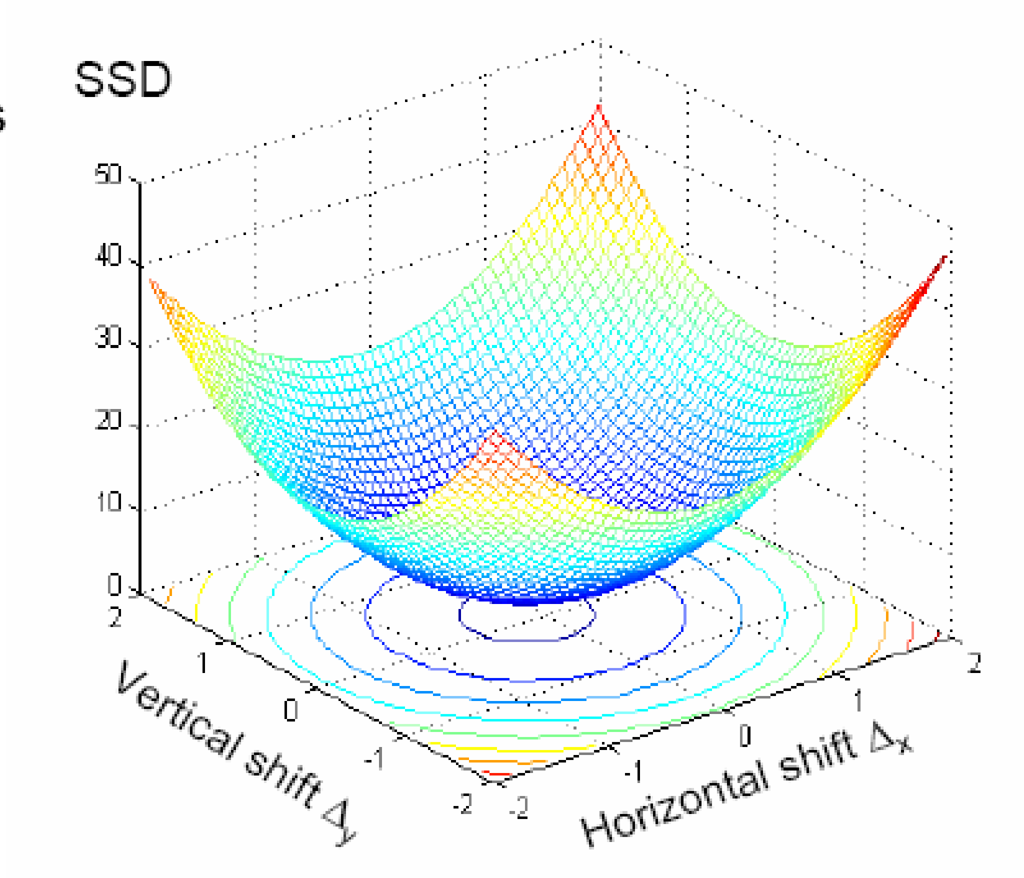
Interpolation of the SSD Minimum



2-d Interpolation of SSD Minimum

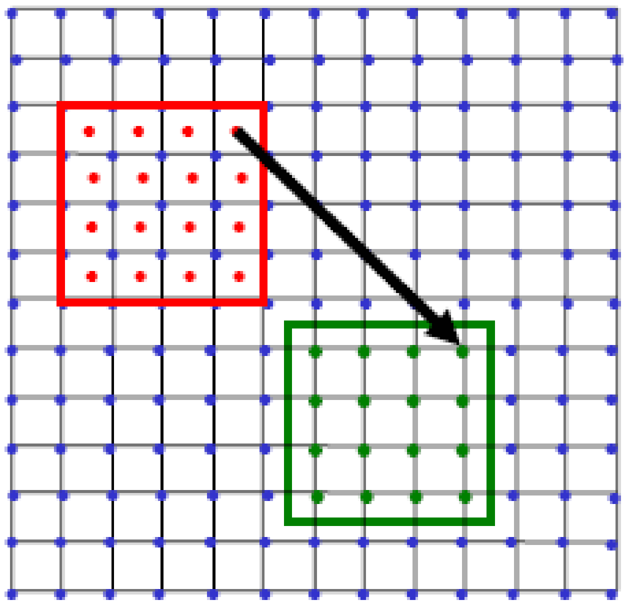
Paraboloid

- Perfect fit through 6 points
- Approximate fit through >6 points



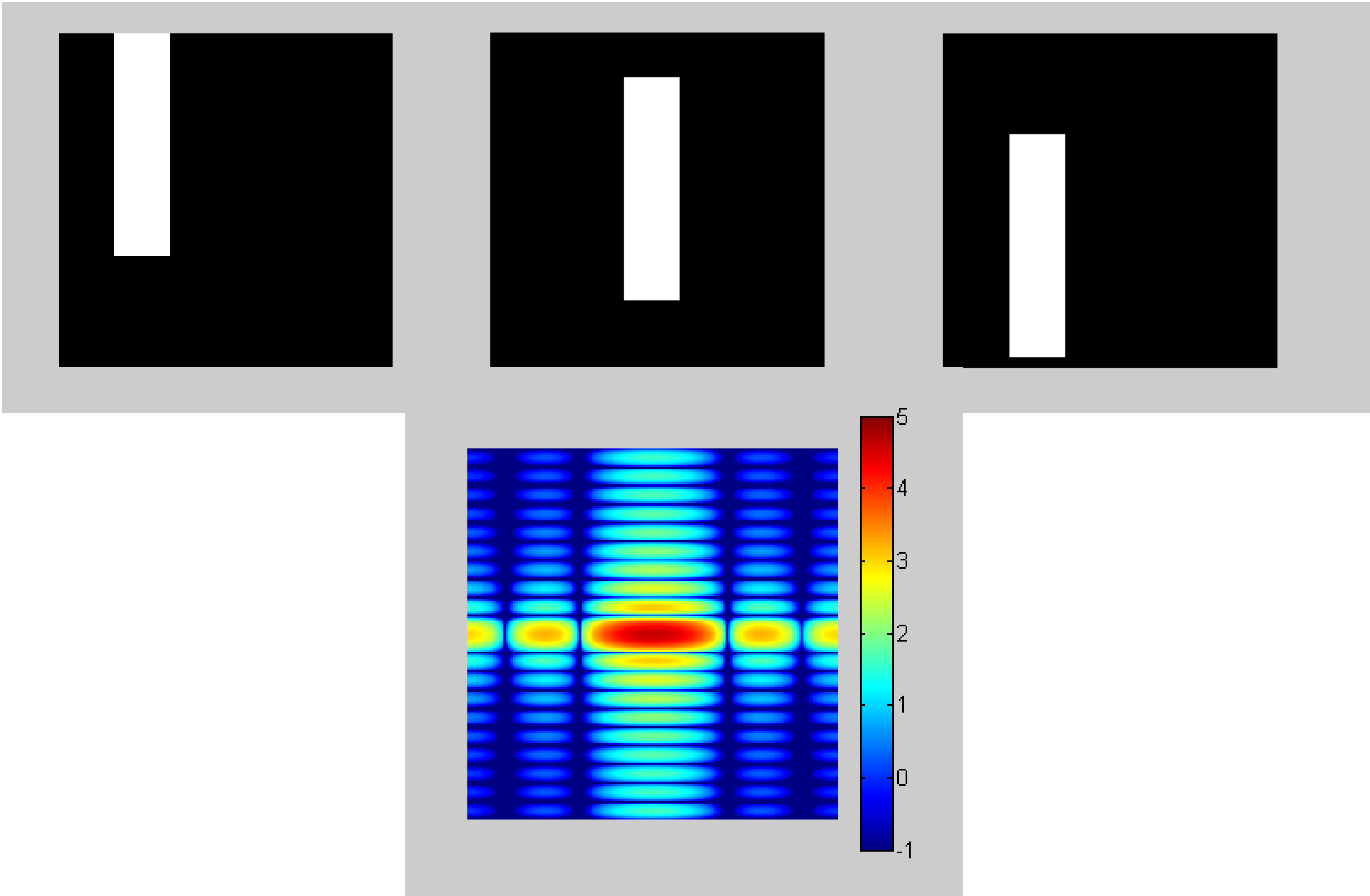
Sub-pixel accuracy

- Interpolate pixel raster of the reference image to desired sub-pixel accuracy (typically by bi-linear interpolation)
- Straightforward extension of displacement vector search to fractional accuracy
- Example: half-pixel accurate displacements

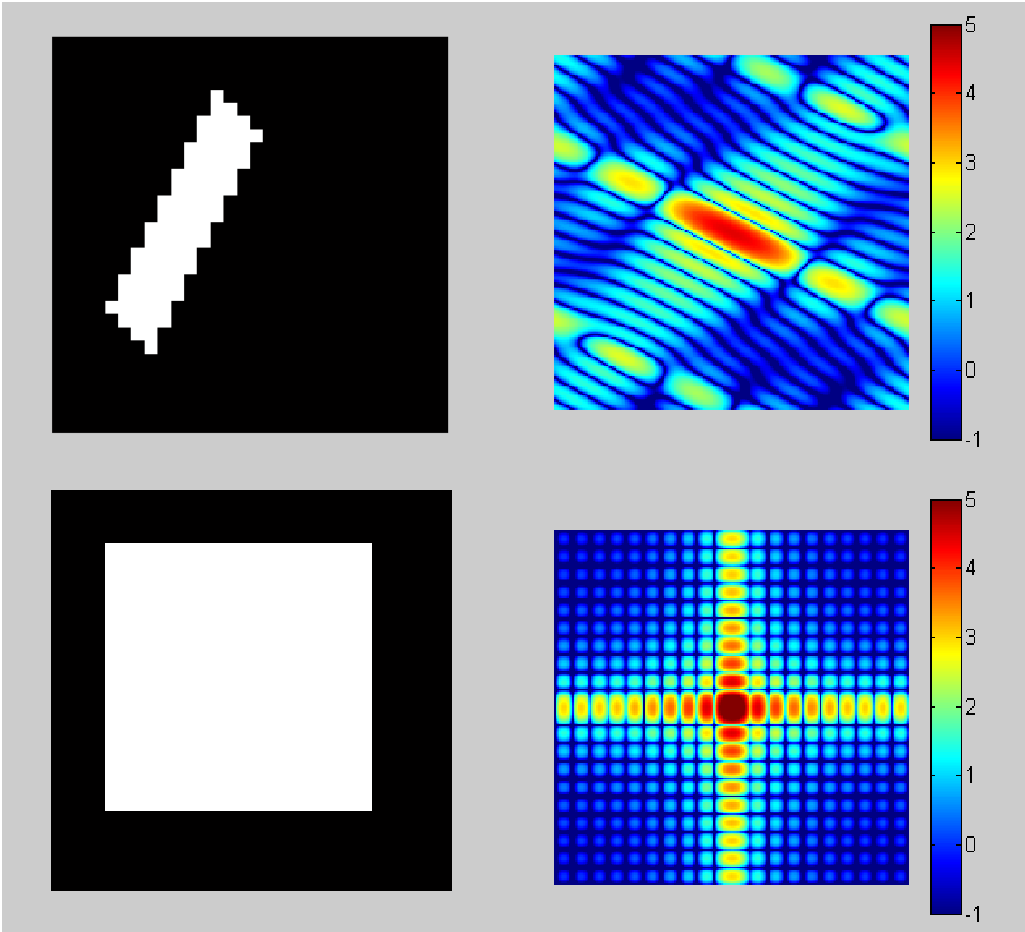


$$\begin{pmatrix} \Delta_x \\ \Delta_y \end{pmatrix} = \begin{pmatrix} 4.5 \\ 4.5 \end{pmatrix}$$

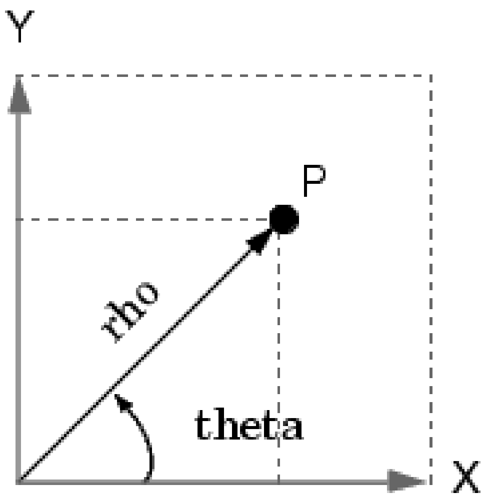
Fourier Transform

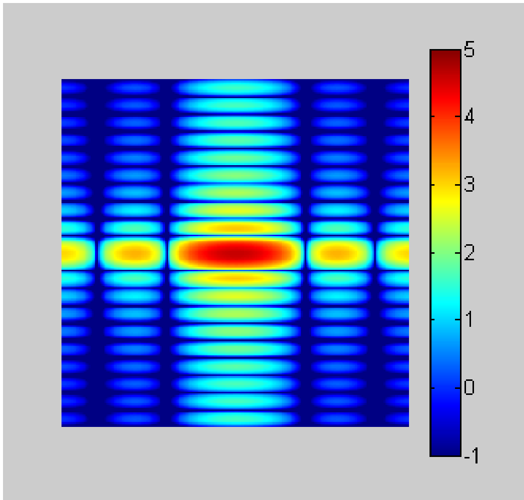


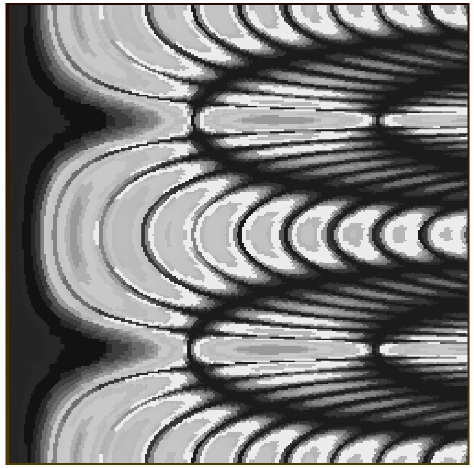
Rotation and Scale effects

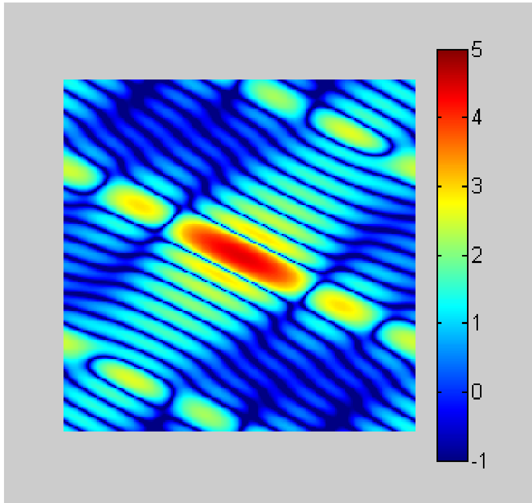


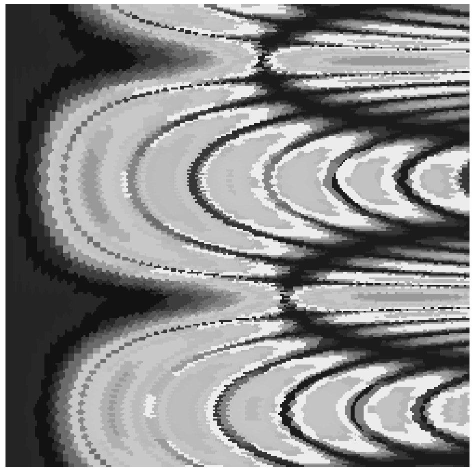
Polar transform











A rotation is a vertical translation!

Log-Polar Transform

Expansion: $f(t) \rightarrow f(at)$

Using logarithmic scale:

$$f(\log(t)) \rightarrow f(\log(a) + \log(t))$$

A scale variation corresponds to a horizontal translation

Translation, rotation and scaling invariance

Translation invariance can then be obtained through a first 2D Fourier transform and disregarding the phase.

A further transform of the previous Fourier Amplitude in log-polar coordinates, where rotations and scaling become translations, is then applied.

A second 2D Fourier Transform is then applied and phase is again disregarded.

The final amplitude of the second 2D Fourier Transform is then invariable to translation, scaling and rotation.

Moments for pattern recognition

Geometric Moments

Let $I(x, y)$ be a continuous image function. Its *geometric moment* of order $p + q$ is defined as

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q I(x, y)\, dx\, dy$$

Moments

combinations of normalized versions of the moments. Specifically, our goal will be to define moments that are invariant to:

Translations:

$$x' = x + a, \quad y' = y + b$$

Scaling:

$$x' = \alpha x, \quad y' = \alpha y$$

Rotations:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Central Moments

Central moments:

$$\mu_{pq} = \int\int I(x, y)(x - \bar{x})^p (y - \bar{y})^q \, dx \, dy$$

where

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

Central moments are invariant to translations.

Normalized central moments:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \quad \gamma = \frac{p + q + 2}{2}$$

These are easily shown to be invariant to both translation and scaling

Hu’s seven moments

The seven moments of Hu: Hu [Hu 62] has defined a set of seven moments that are invariant under the actions of translation, scaling, and rotation. These are

$$p + q = 2$$

$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

$$p + q = 3$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (\eta_{03} - 3\eta_{21})^2$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{03} + \eta_{21})^2$$

$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{03} - 3\eta_{21})(\eta_{03} + \eta_{21})[(\eta_{03} + \eta_{21})^2 - 3(\eta_{12} + \eta_{30})^2]$$

$$\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21})$$

$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[(\eta_{03} + \eta_{21})^2 - 3(\eta_{30} + \eta_{12})^2]$$

Discretization

For a digital image $I(i, j)$, with $i = 0, 1, \ldots, N_x - 1$, $j = 0, 1, \ldots, N_y - 1$, the preceding moments can be *approximated* by replacing integrals by summations,

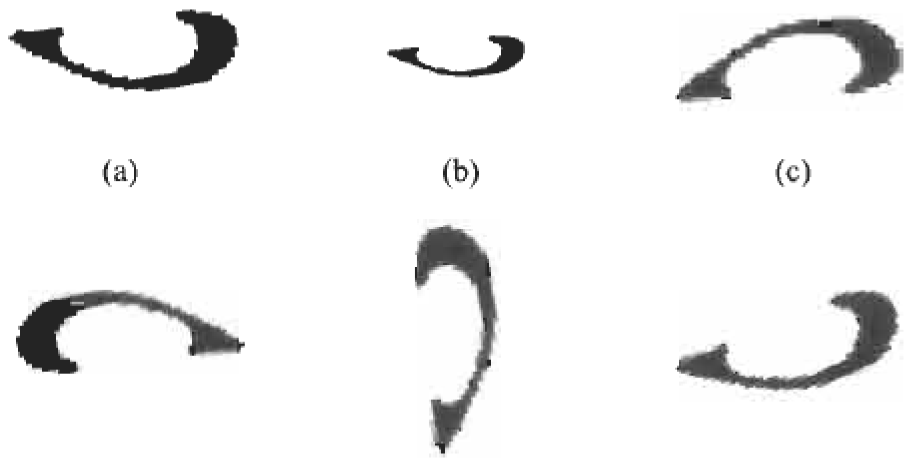
$$m_{pq} = \sum_i \sum_j I(i, j) i^p j^q \qquad (7.24)$$

In order to keep the dynamic range of the moment values consistent for different-sized images, a normalization of the $x - y$ axis can be performed, prior to the computation of the moments. The moments are then approximated by

$$m_{pq} = \sum_i I(x_i, y_i) x_i^p y_i^q$$

where the sum is over all image pixels. Then x_i, y_i are the coordinates of the center point of the ith pixel and are no longer integers but real numbers in the interval $x_i \in [-1, +1]$, $y_i \in [-1, +1]$. *For digital images, the invariance properties of the moments we have defined are only approximately true.*

The Byzantine symbol “petasti“



Moments	0°	Scaled	180°	15°	Mirror	90°
ϕ_1	93.13	91.76	93.13	94.28	93.13	93.13
ϕ_2	58.13	56.60	58.13	58.59	58.13	58.13
ϕ_3	26.70	25.06	26.70	27.00	26.70	26.70
ϕ_4	15.92	14.78	15.92	15.83	15.92	15.92
ϕ_5	3.24	2.80	3.24	3.22	3.24	3.24
ϕ_6	10.70	9.71	10.70	10.57	10.70	10.70
ϕ_7	0.53	0.46	0.53	0.56	−0.53	0.53

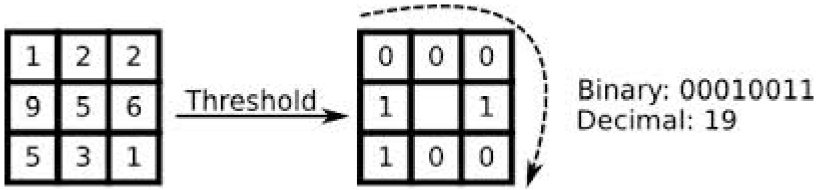
Local Binary Pattern (LBP)

Compare 8-connected neighborhood with center pixel

If pixel > center, replace with '1' else '0'

Construct a binary number by going clockwise

Replace the center pixel with the decimal value of the binary number



Binary number is sensitive to starting point – LBP is not rotation invariant

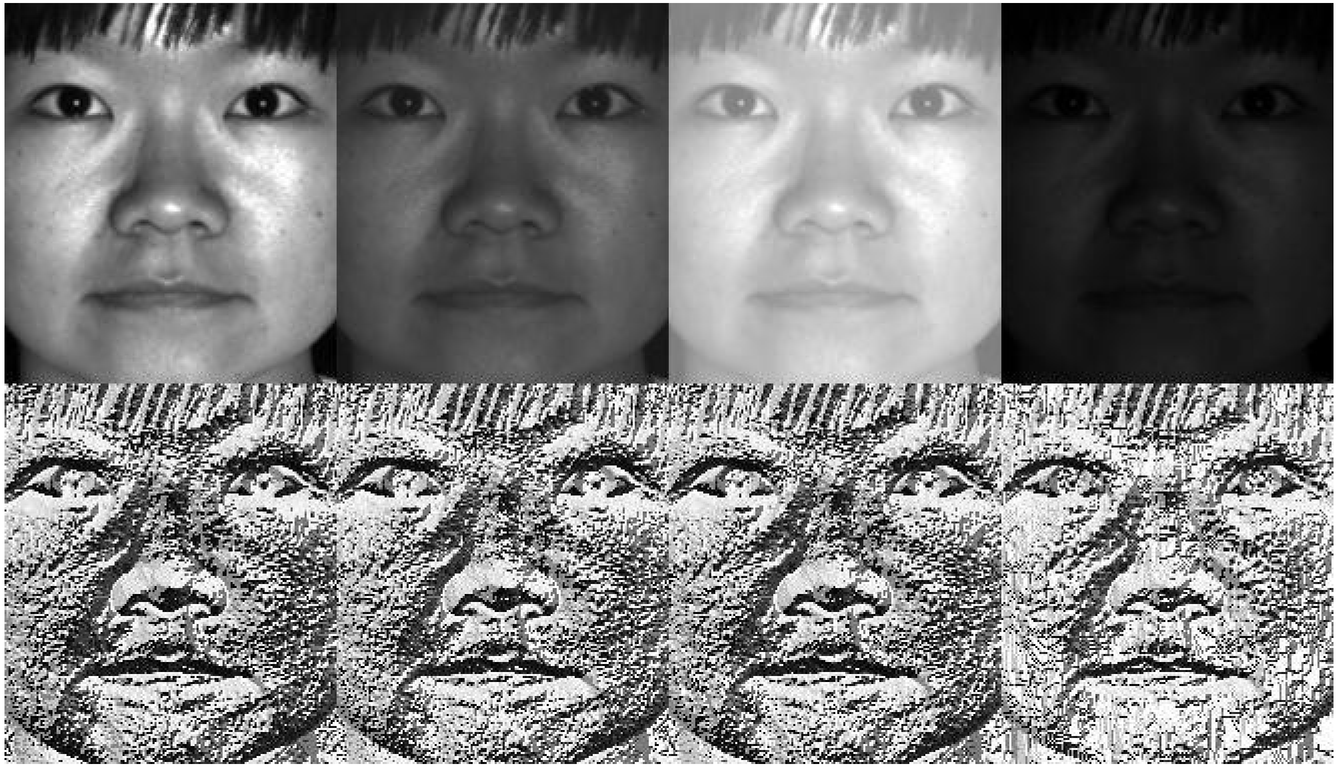
Rotate binary string to minimize decimal value for rotation invariance

Minor changes in illumination can change the decimal value

Partition image into cells and construct LBP histograms in each cell

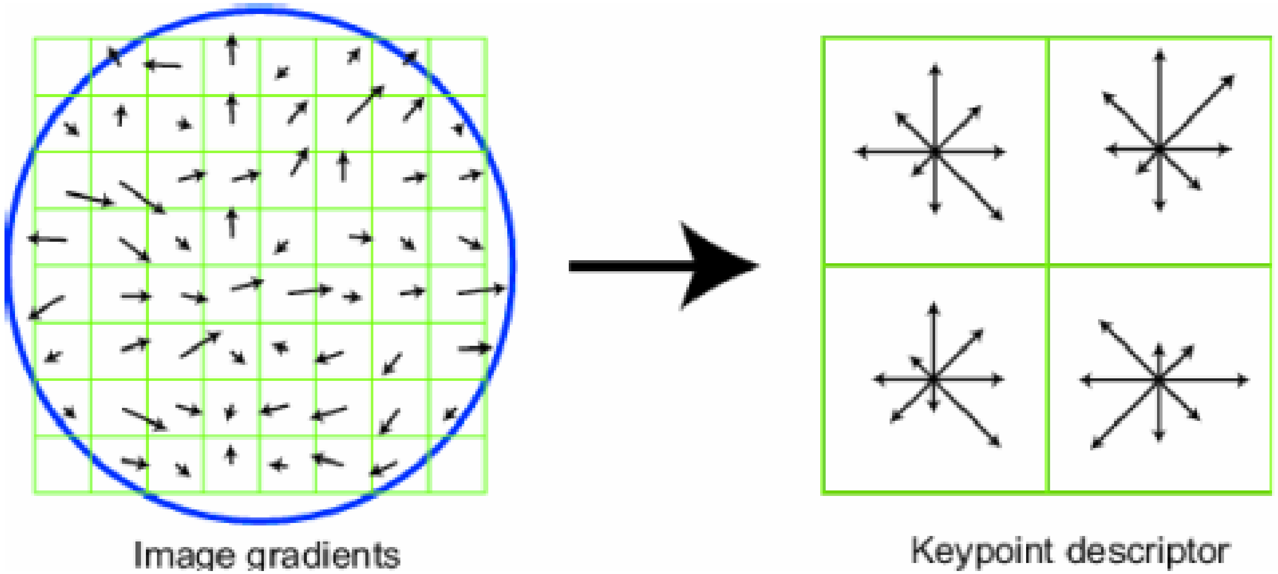
LBP Example from OpenCV

Notice how LBP feature are illumination invariant



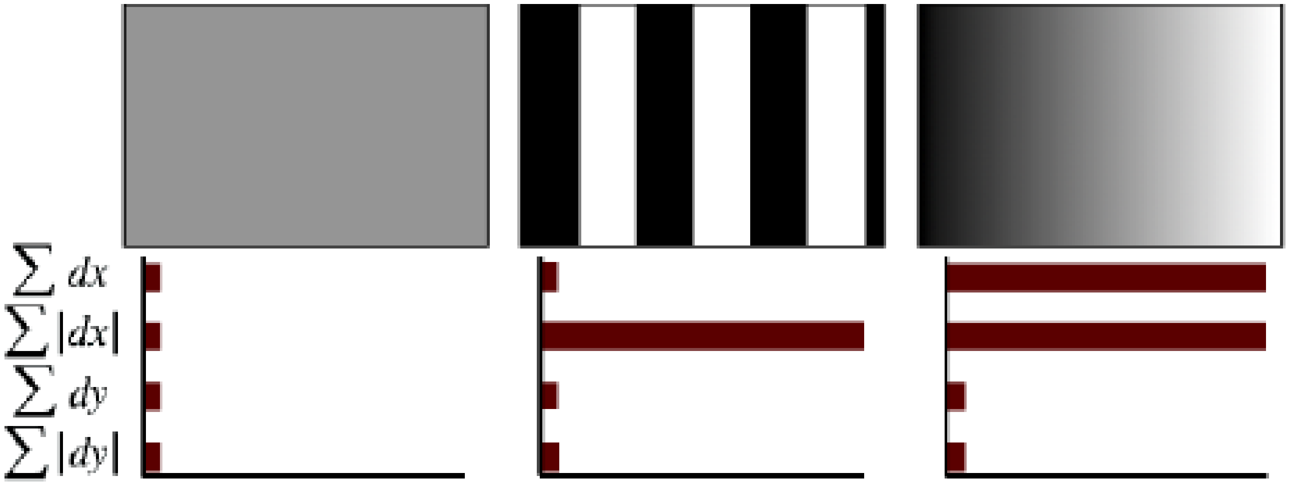
SIFT descriptors

- SIFT - Scale-Invariant Feature Transform *[Brown, Lowe, 2002]*
- Sample thresholded image gradients at 16x16 locations in scale space
(in local coordinate system for rotation and scale invariance)
- Generate 4x4 orientation histograms with 8 directions each;
each observation weighted with magnitude of image gradient and window function
- 128-dimensional feature vector



SURF descriptors

- SURF – Speeded Up Robust Features *[Bay et al. 2006]*
- Compute horizontal and vertical pixel differences, dx, dy (in local coordinate system for rotation and scale invariance, window size $20\sigma \times 20\sigma$, where σ^2 is feature scale)
- Accumulate dx, dy, and $|dx|, |dy|$ over 4x4 subregions (SURF-64) or 3x3 subregions (SURF-36)
- Normalize vector for gain invariance, but distinguish bright blobs and dark blobs based on sign of Laplacian (trace of Hessian)



Affine parameters from feature correspondences

- Given: feature correspondences

$$(x_i', y_i') \leftrightarrow (x_i, y_i) \quad i = 1, \ldots, N$$

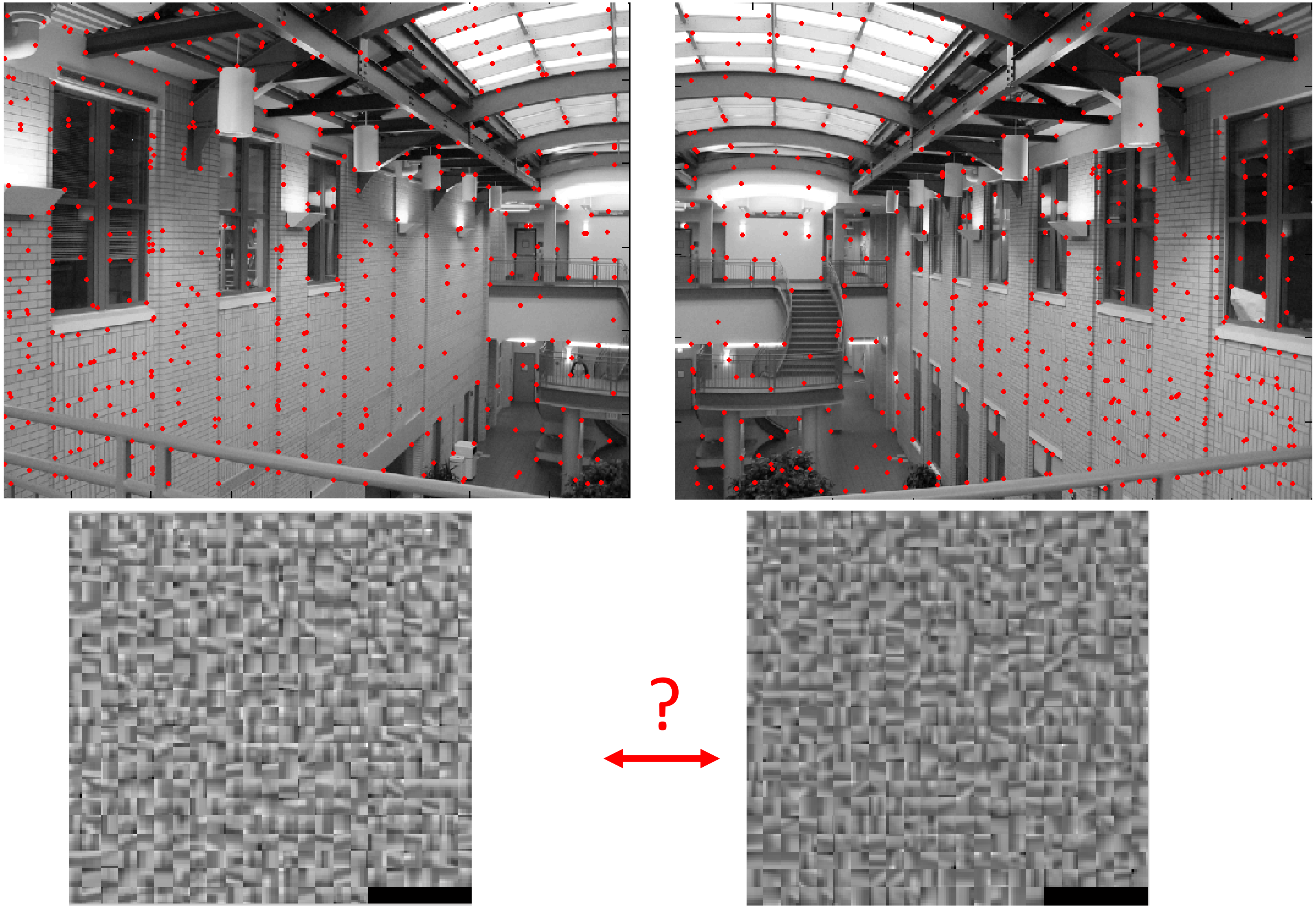
- Set up $2N$ linear equations with 6 unknowns $a_{00}, a_{01}, a_{02}, a_{10}, a_{11}, a_{12}$

$$\mathbf{x}_i' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \underline{\mathbf{x}}_i$$

- Solve by least mean squares (or least median of squares)
- Easily extended to higher-order linear warping model, e.g.,

$$x_i' = a_1 + a_2 x_i + a_3 y_i + a_4 x_i^2 + a_5 y_i^2 + a_6 x_i y_i$$
$$y_i' = b_1 + b_2 x_i + b_3 y_i + b_4 x_i^2 + b_5 y_i^2 + b_6 x_i y_i$$

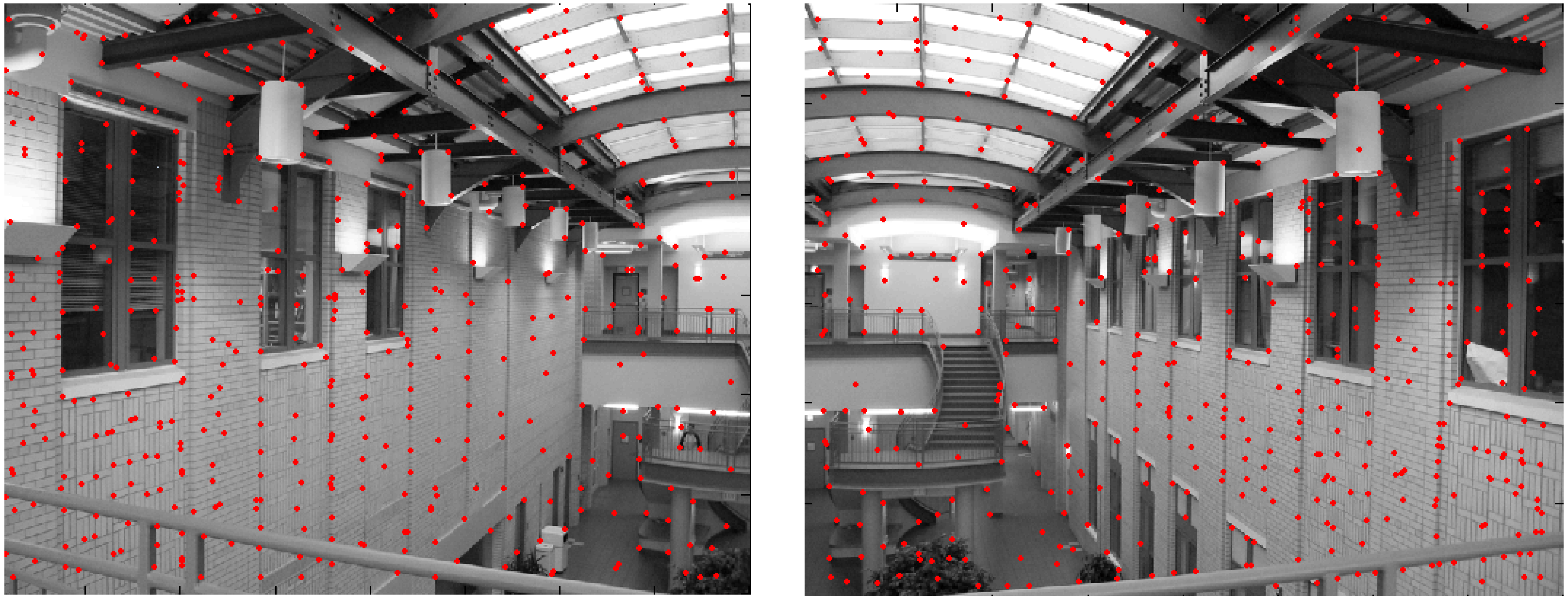
Feature matching

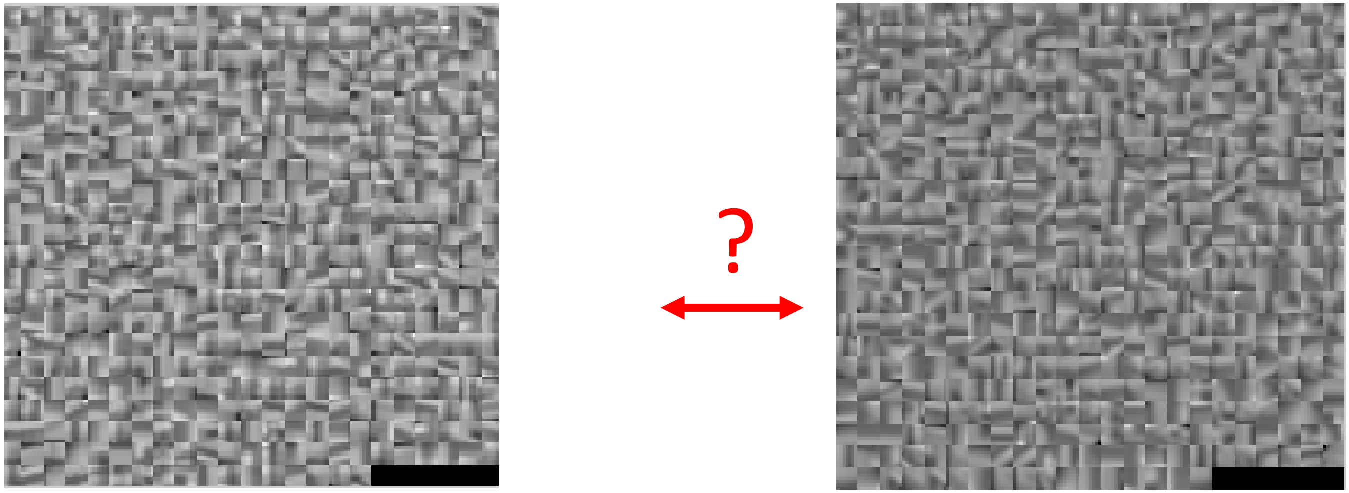


Feature matching

- Exhaustive search
  - for each feature in one image, look at *all* the other features in the other image(s)
- Hashing
  - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)
- Nearest neighbor techniques
  - *k*-trees and their variants

What about outliers?



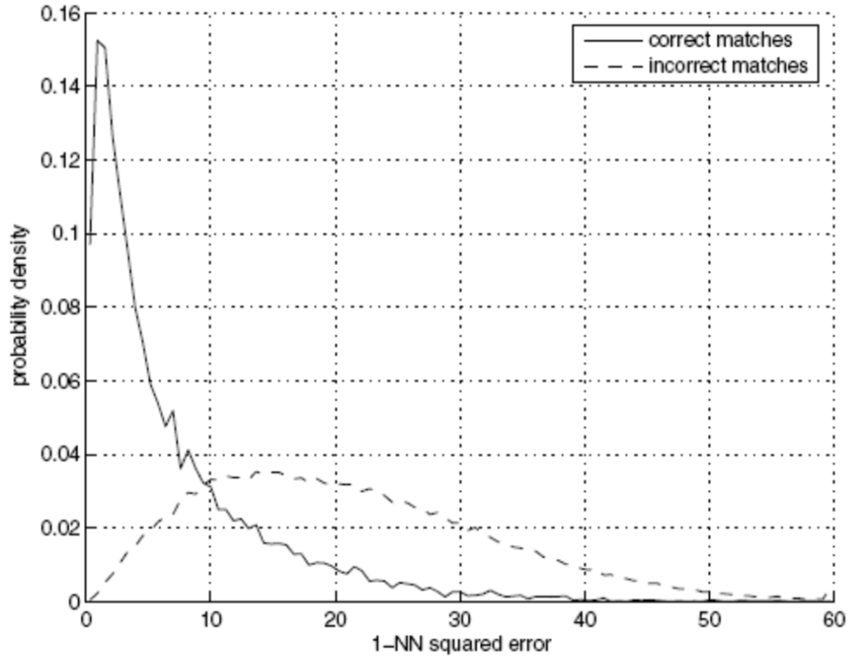


Feature-space outlier rejection

Let’s not match all features, but only these that have “similar enough” matches?

How can we do it?

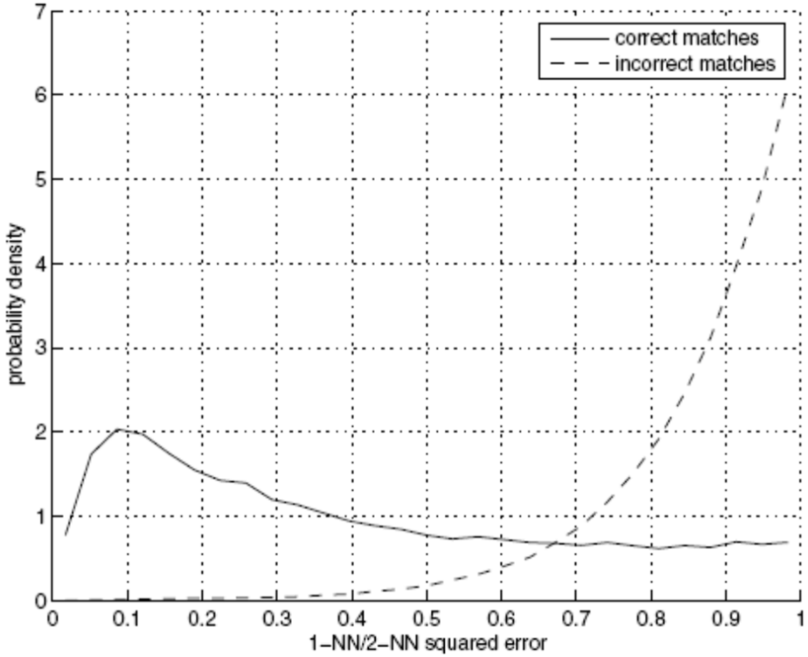
- SSD(patch1,patch2) < threshold
- How to set threshold?



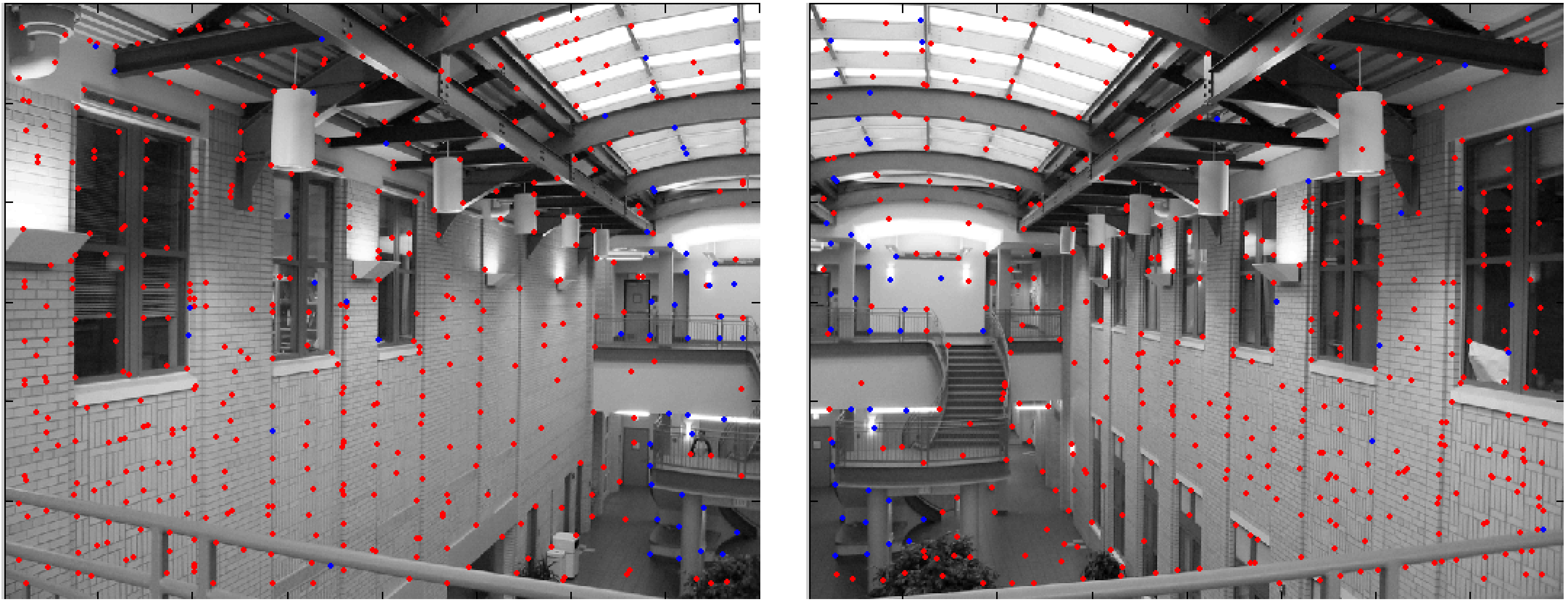
Feature-space outlier rejection

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?



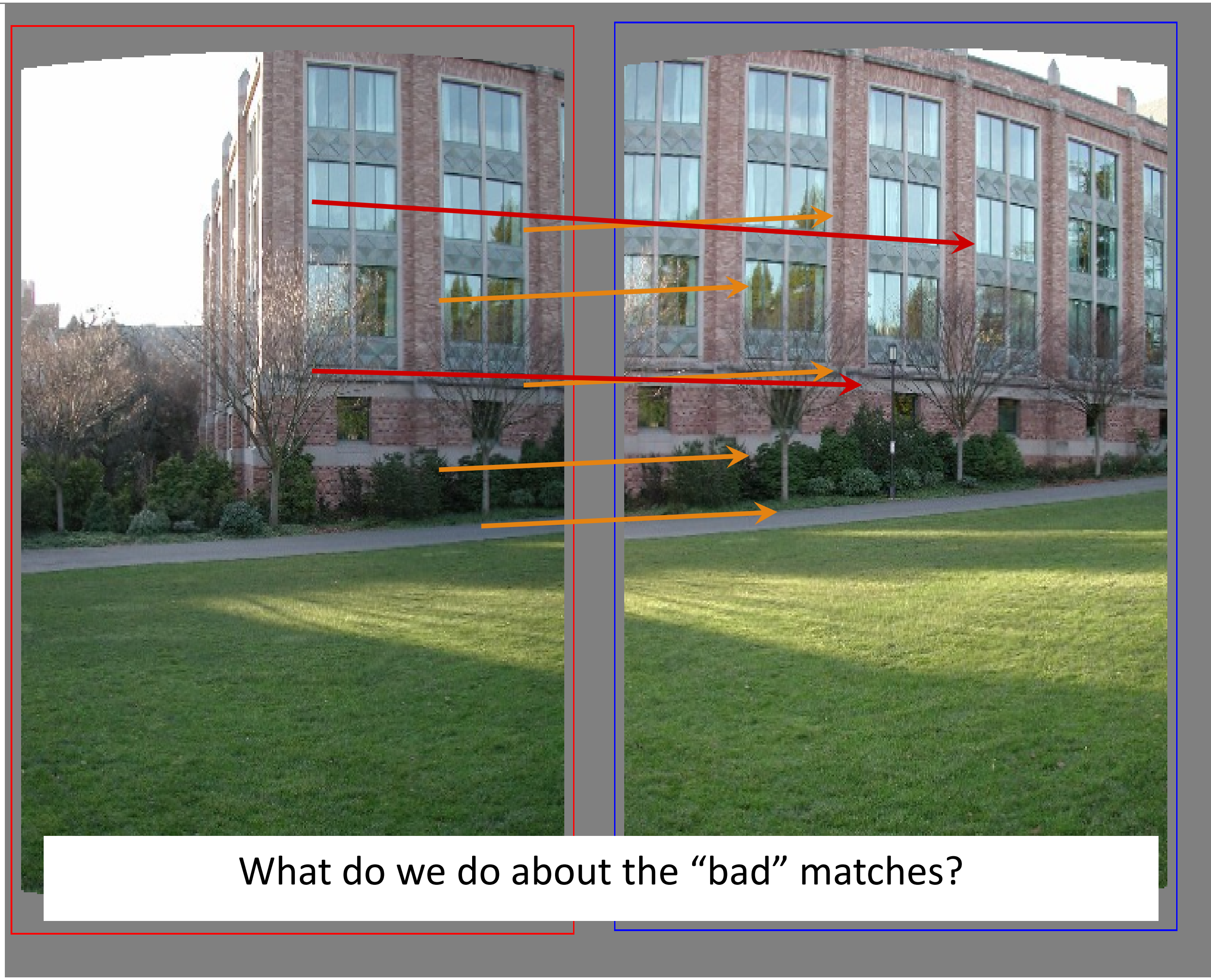
Feature-space outliner rejection



Can we now compute H from the blue points?

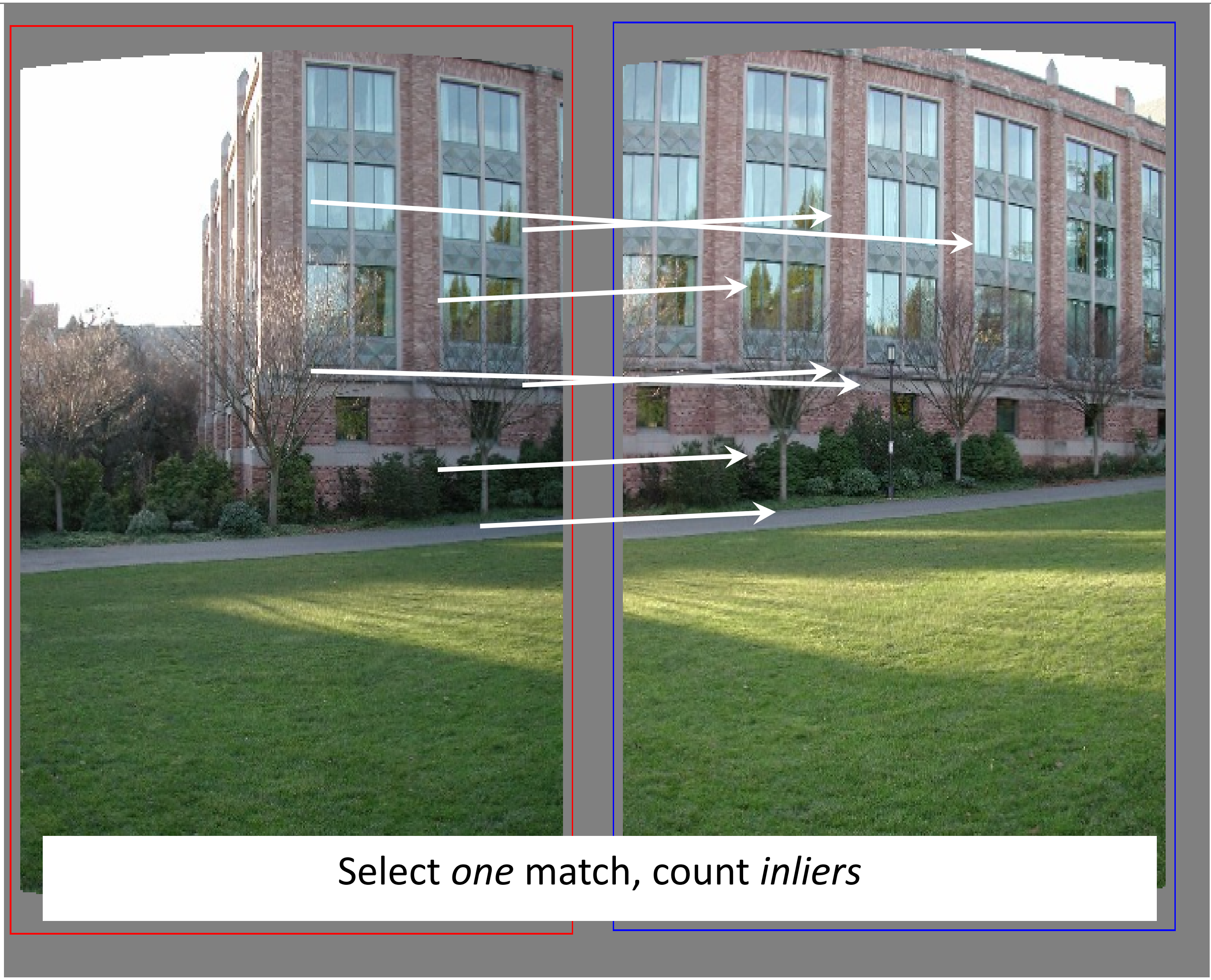
- No! Still too many outliers…
- What can we do?

Matching features



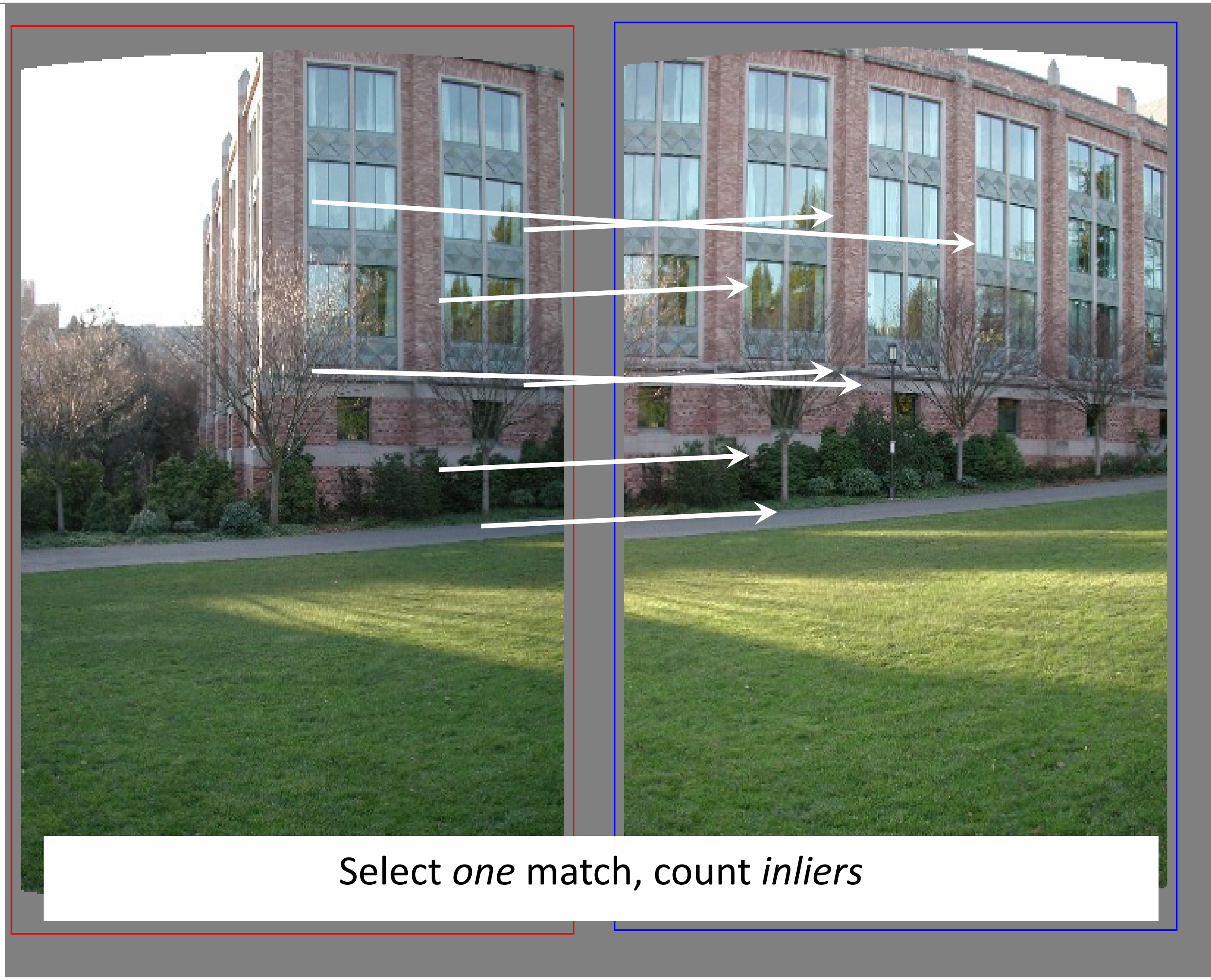
What do we do about the “bad” matches?

RAndom SAmple Consensus



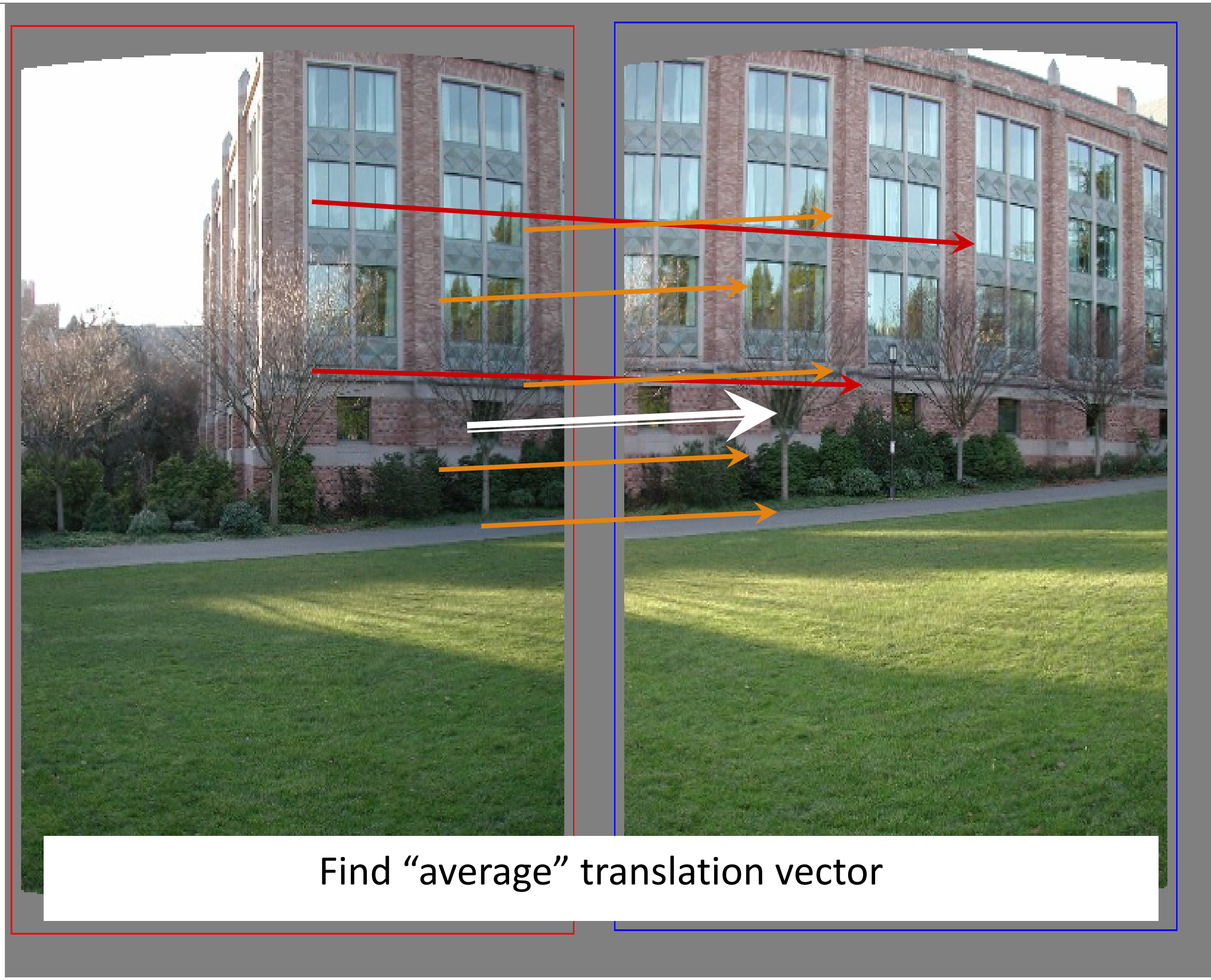
Select *one* match, count *inliers*

RAndom SAmple Consensus



Select *one* match, count *inliers*

Least squares fit

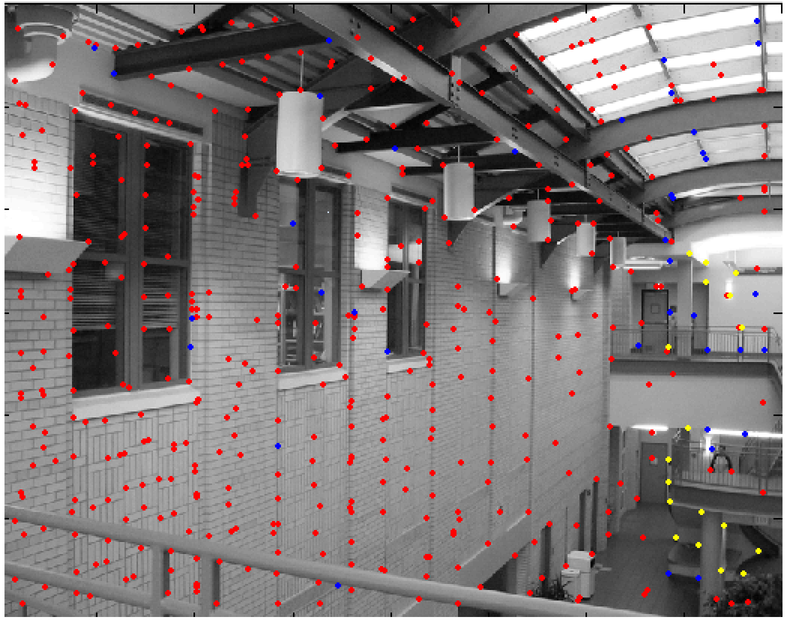


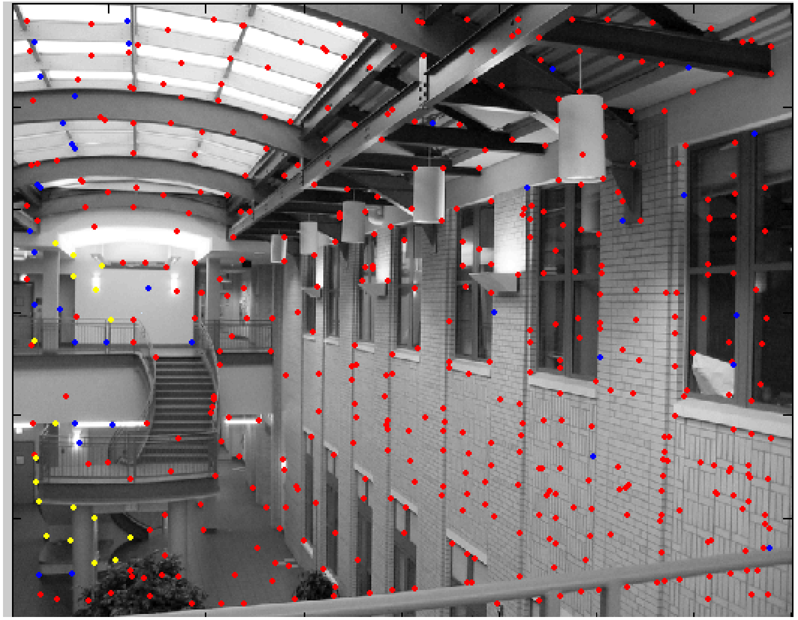
RANSAC for estimating homography

RANSAC loop:

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

RANSAC





Example: Recognising Panoramas

M. BROWN AND D. LOWE,

UNIVERSITY OF BRITISH COLUMBIA

Why "Recognising Panoramas"?

Why “Recognising Panoramas”?

1D Rotations (θ)

- Ordering $\Rightarrow$ matching images

Why "Recognising Panoramas"?

1D Rotations (θ)

- Ordering $\Rightarrow$ matching images

Why "Recognising Panoramas"?

1D Rotations (θ)

- Ordering $\Rightarrow$ matching images

Why “Recognising Panoramas”?

1D Rotations (θ)

- Ordering $\Rightarrow$ matching images

- 2D Rotations (θ, ϕ)
  - Ordering $\nRightarrow$ matching images

Why "Recognising Panoramas"?

1D Rotations (θ)

- Ordering $\Rightarrow$ matching images

- 2D Rotations (θ, ϕ)
  - Ordering $\nRightarrow$ matching images

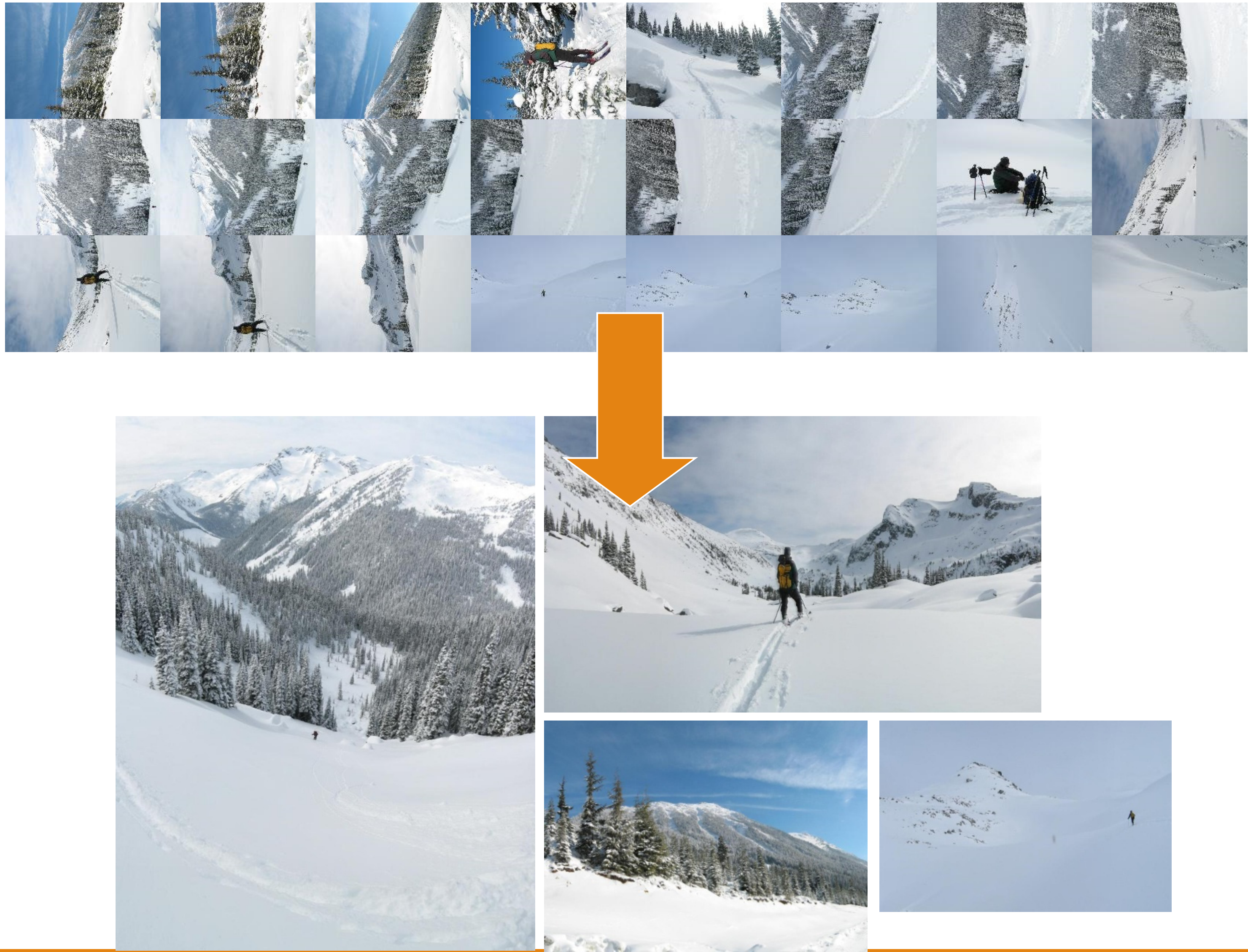
Why “Recognising Panoramas”?

1D Rotations (θ)

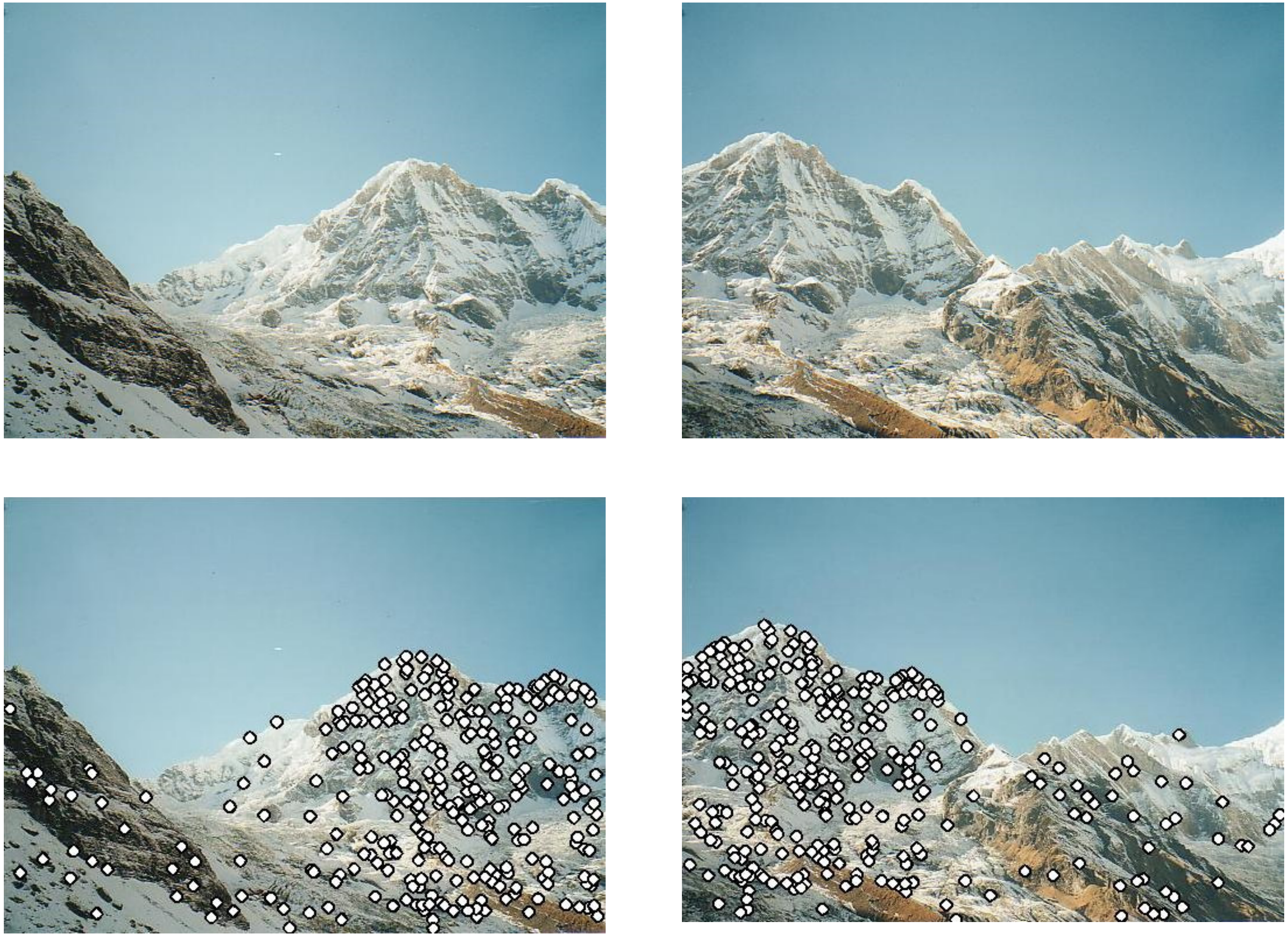
- Ordering ⇒ matching images

- 2D Rotations (θ, ϕ)
  - Ordering ⇏ matching images

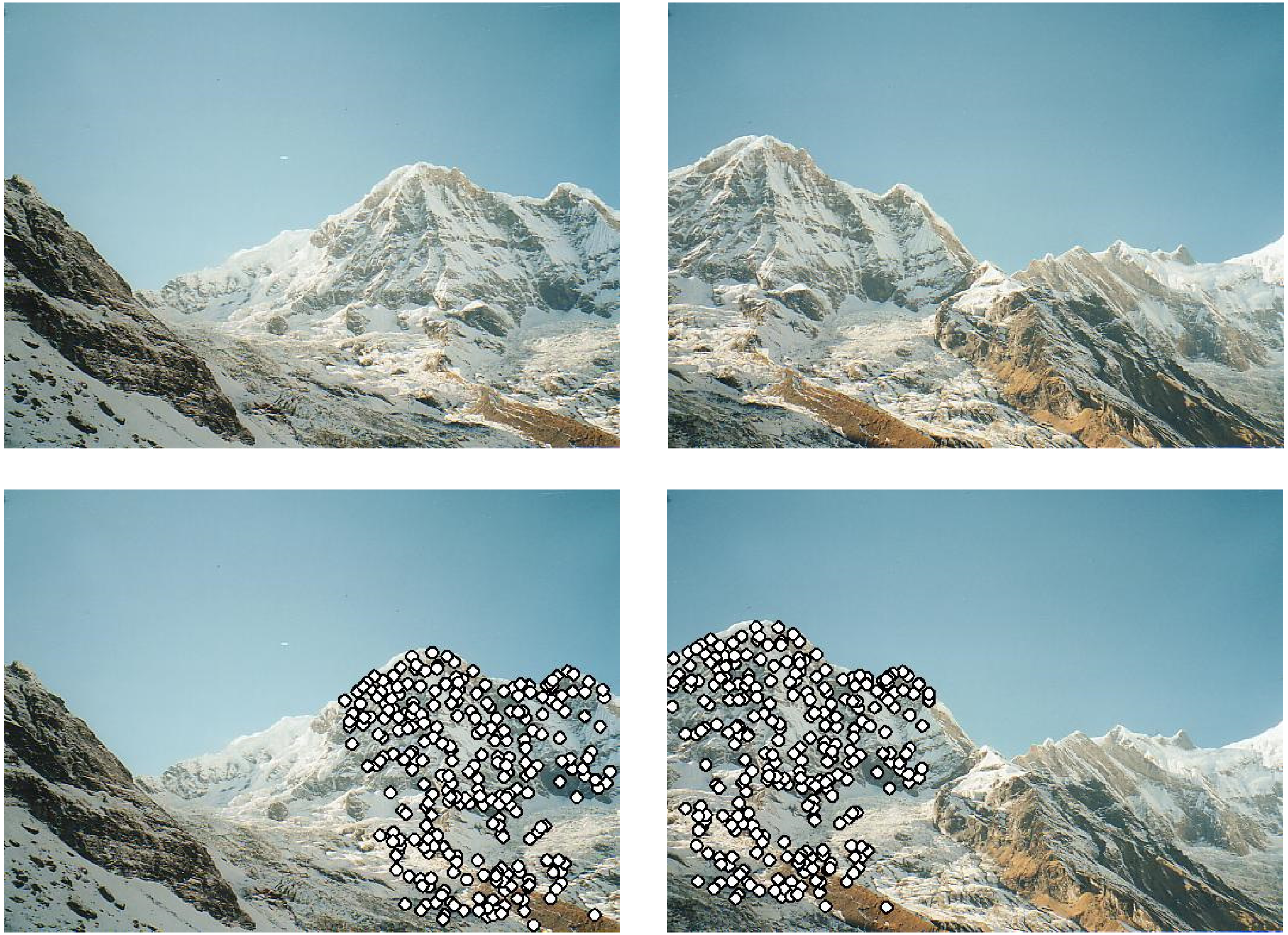
Why “Recognising Panoramas”?



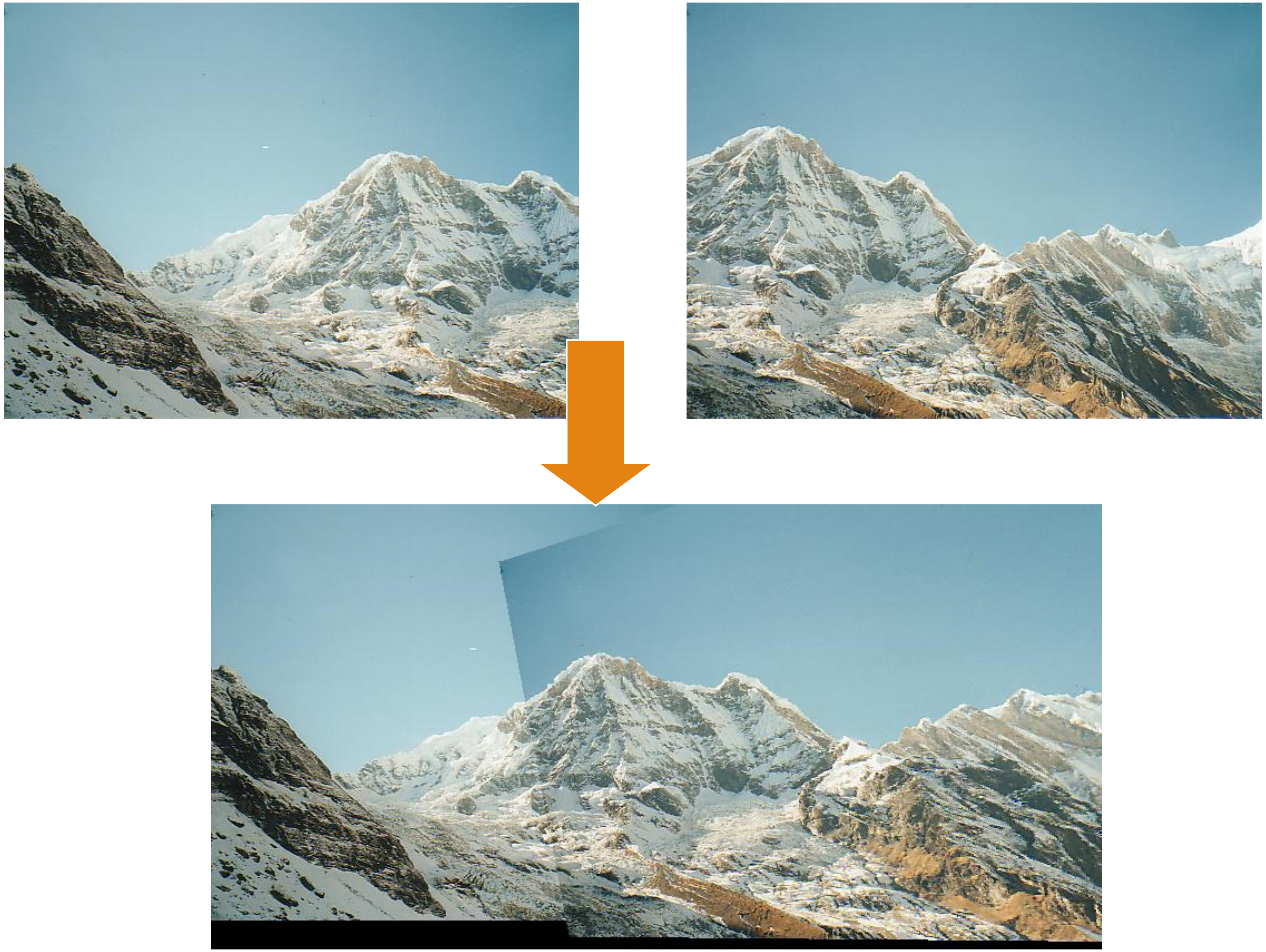
RANSAC for Homography



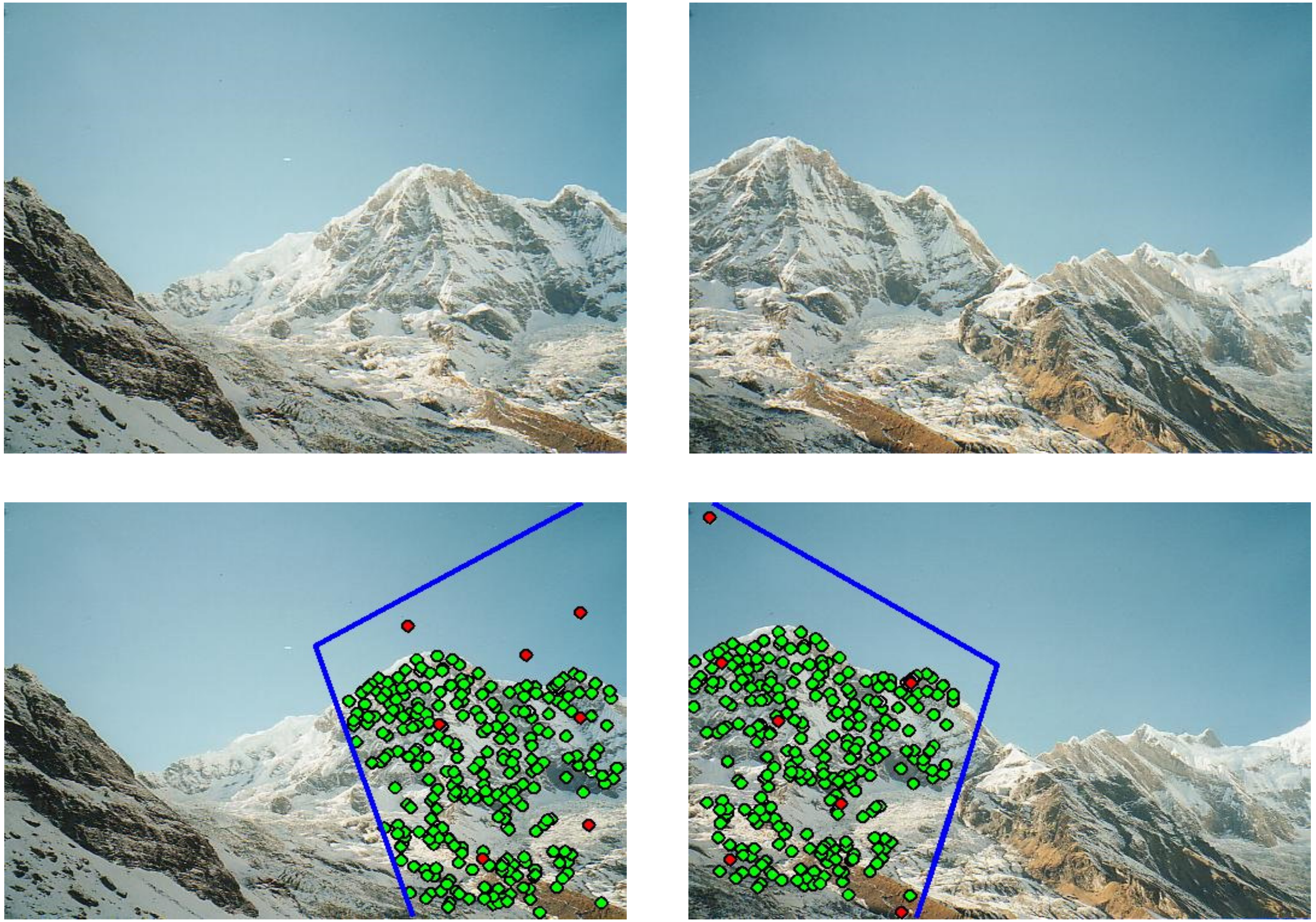
RANSAC for Homography



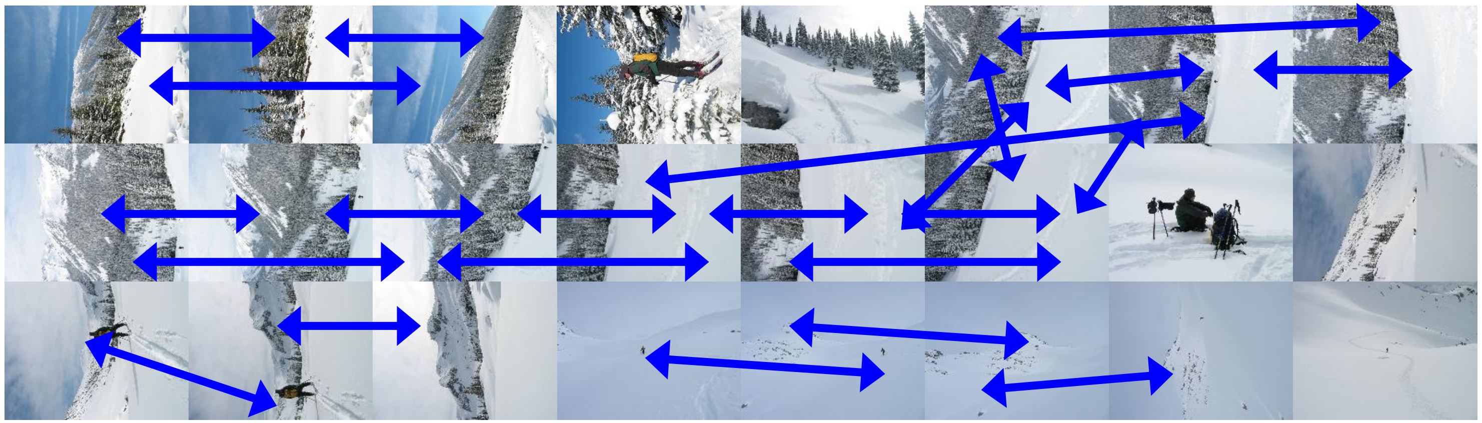
RANSAC for Homography



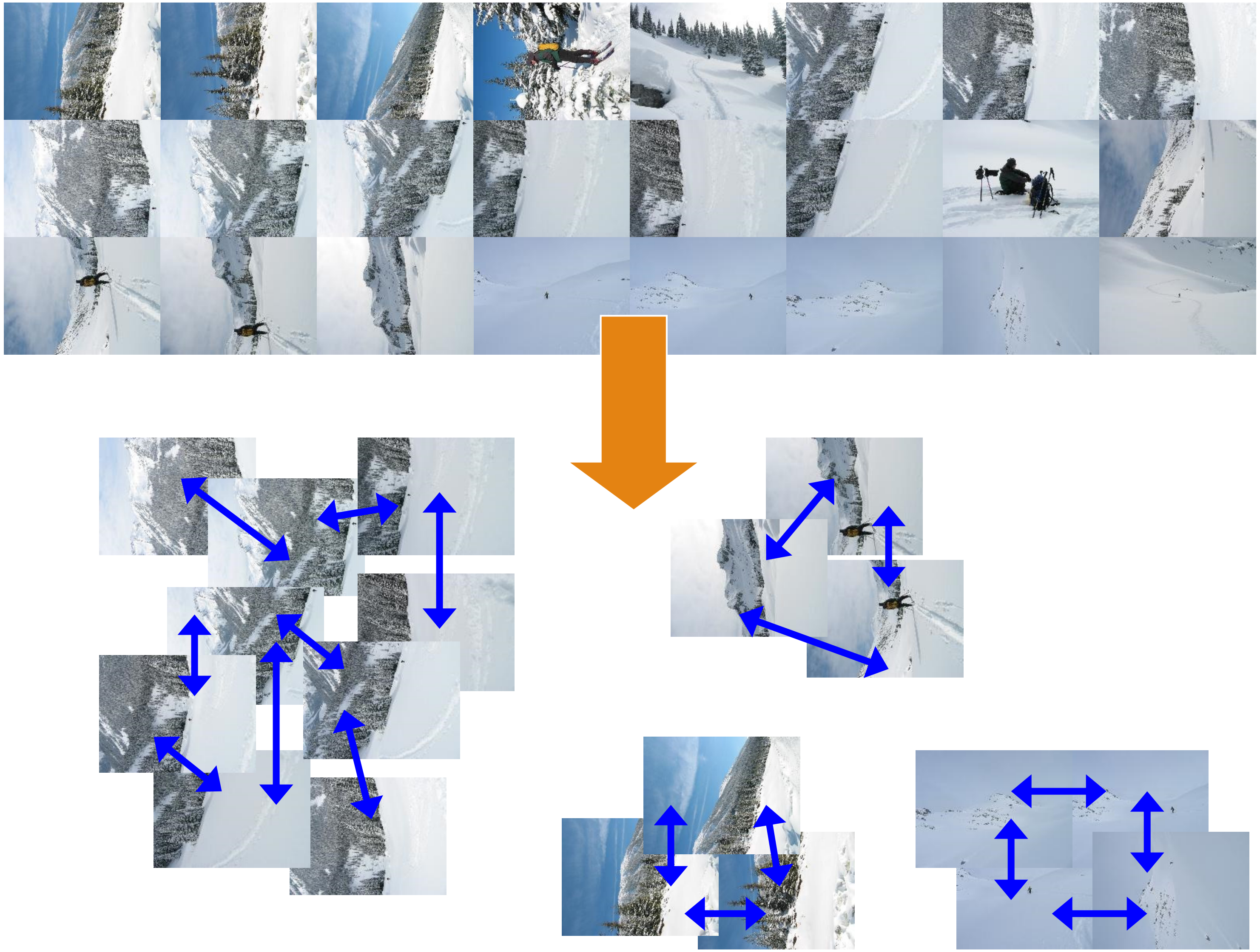
Probabilistic model for verification



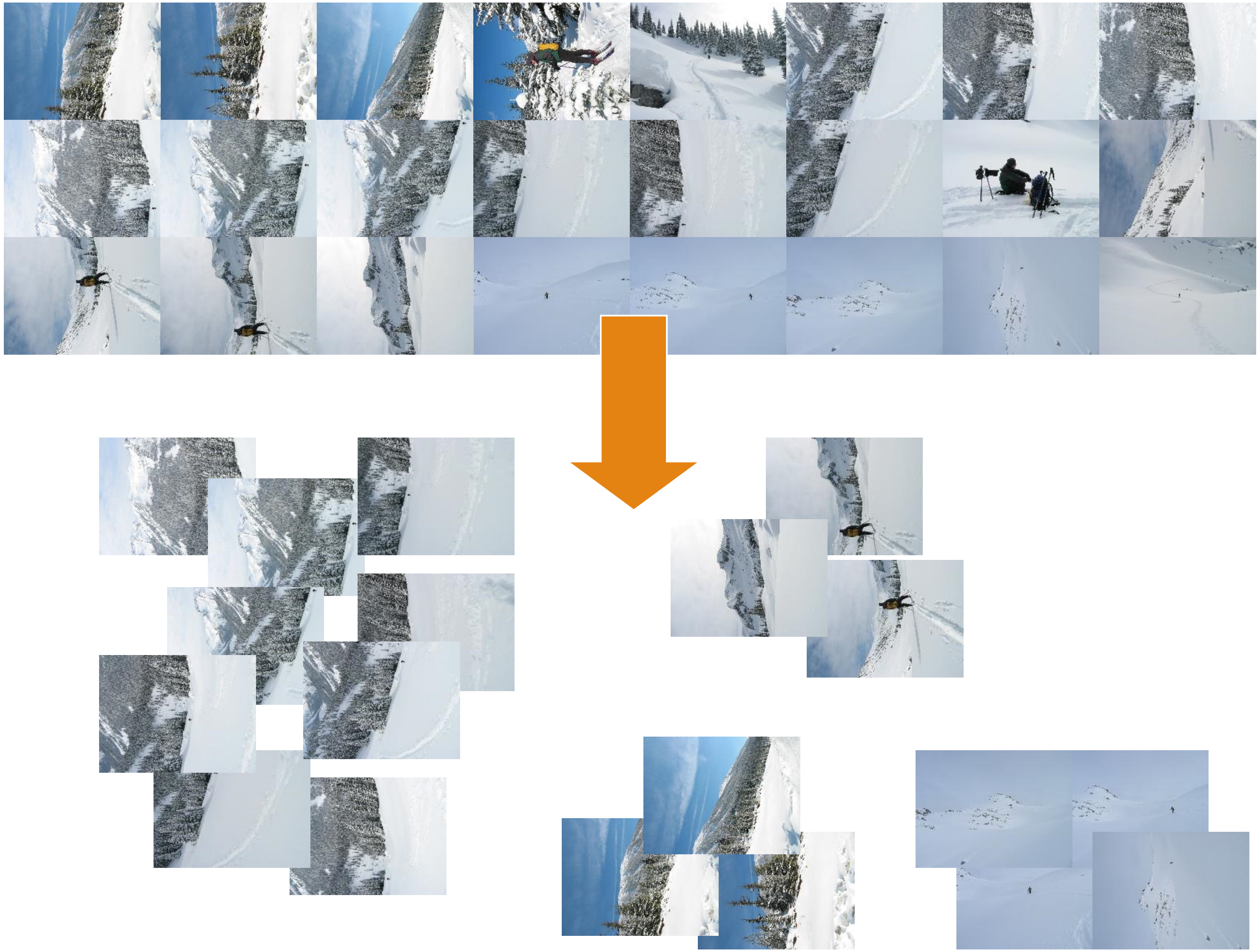
Finding the panoramas



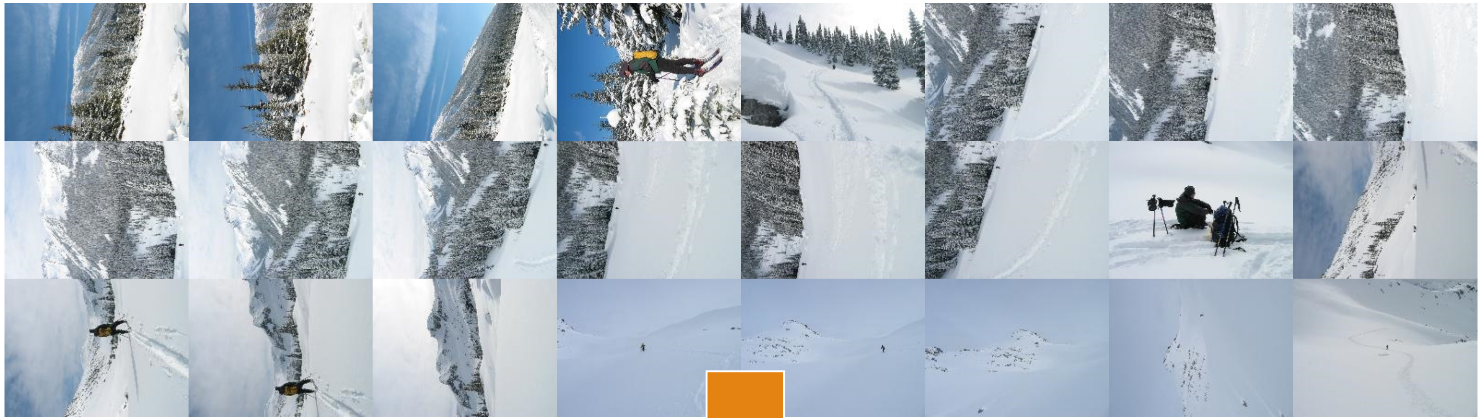
Finding the panoramas

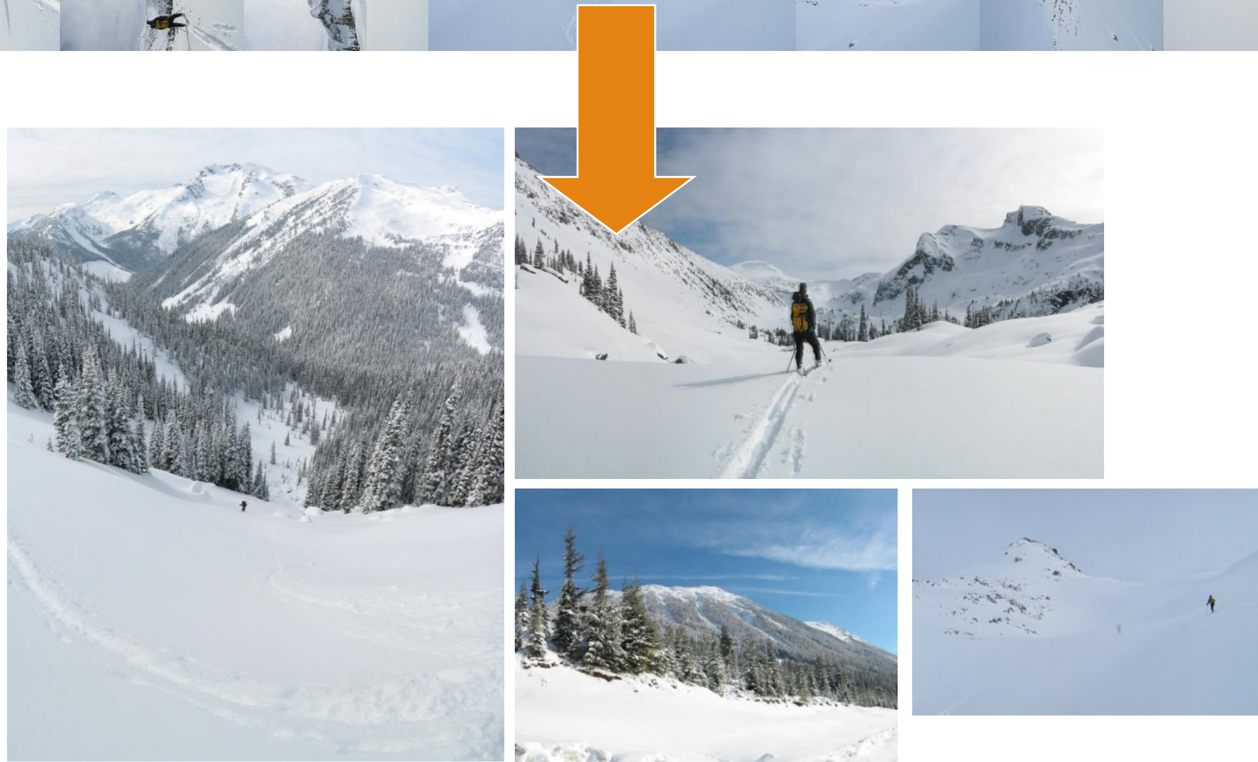


Finding the panoramas



Finding the panoramas





Homography for Rotation

Parameterise each camera by rotation and focal length

$$\mathbf{R}_i = e^{[\boldsymbol{\theta}_i]_\times}, \;\; [\boldsymbol{\theta}_i]_\times = \begin{bmatrix} 0 & -\theta_{i3} & \theta_{i2} \\ \theta_{i3} & 0 & -\theta_{i1} \\ -\theta_{i2} & \theta_{i1} & 0 \end{bmatrix}$$

$$\mathbf{K}_i = \begin{bmatrix} f_i & 0 & 0 \\ 0 & f_i & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This gives pairwise homographies

$$\tilde{\mathbf{u}}_i = \mathbf{H}_{ij}\tilde{\mathbf{u}}_j, \;\; \mathbf{H}_{ij} = \mathbf{K}_i\mathbf{R}_i\mathbf{R}_j^T\mathbf{K}_j^{-1}$$

Bundle Adjustment

New images initialized with rotation, focal length of best matching image

Bundle Adjustment

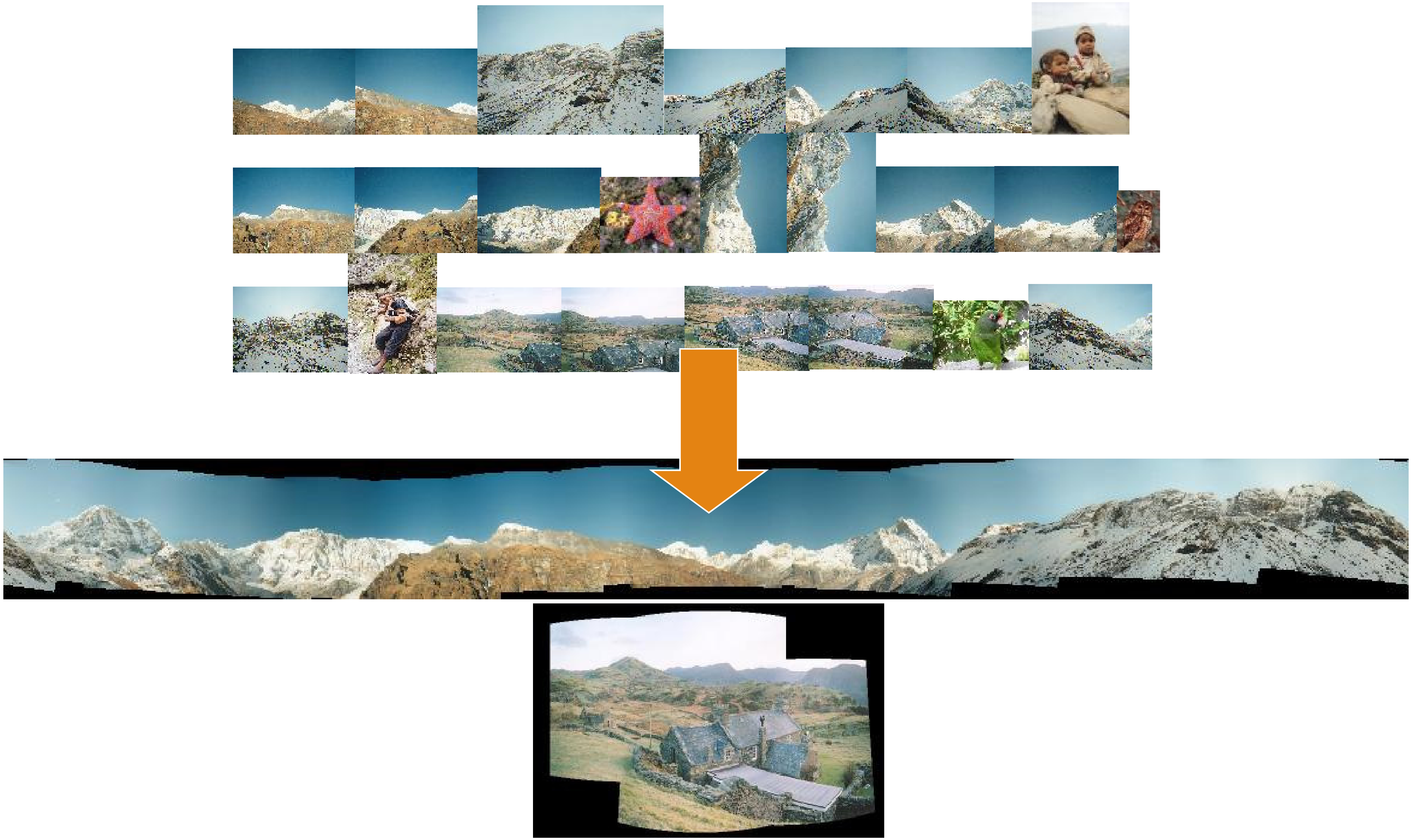
New images initialized with rotation, focal length of best matching image

Multi-band Blending

Burt & Adelson 1983

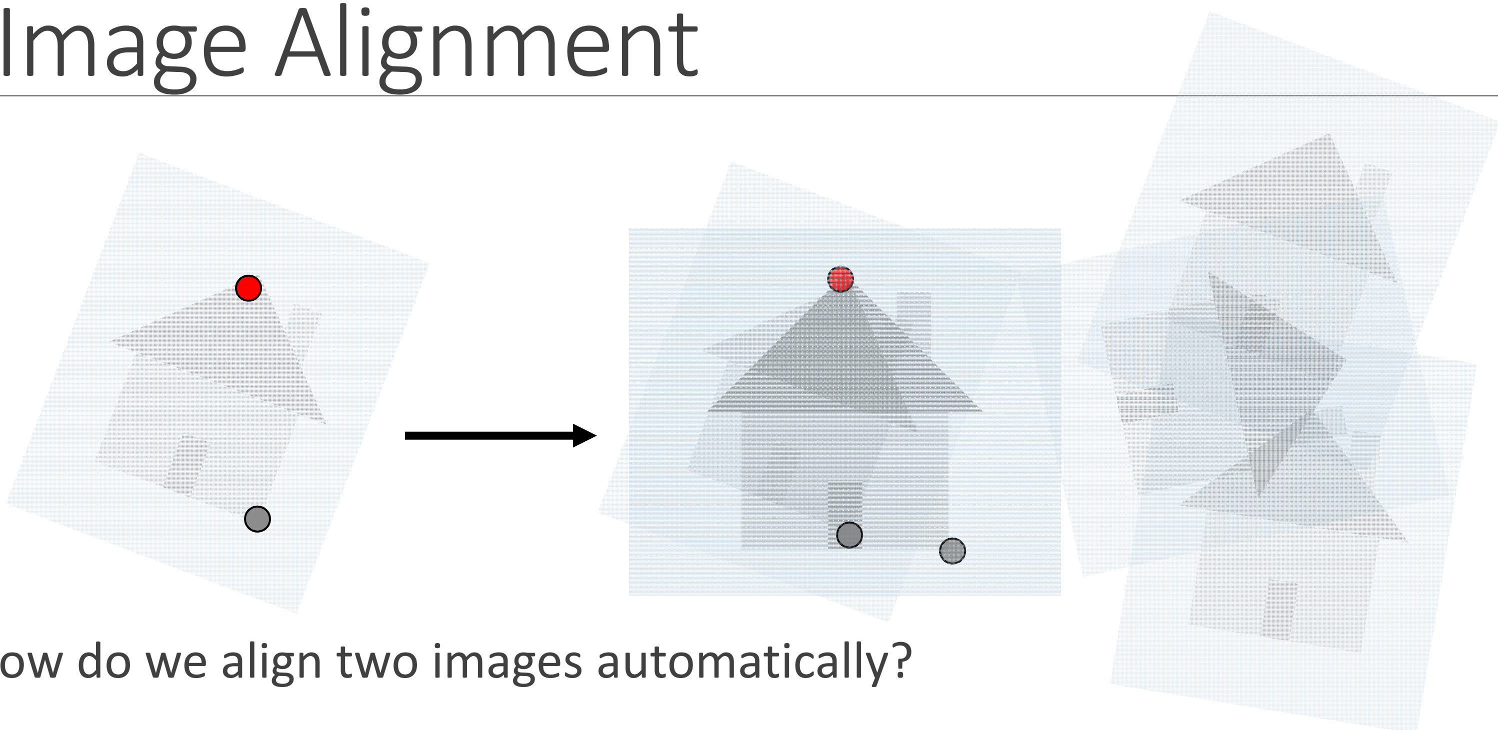
- Blend frequency bands over range $\propto \lambda$

Results



OPTICAL FLOW

Image Alignment



How do we align two images automatically?

Two broad approaches:

- Feature-based alignment
  - Find a few matching features in both images
  - compute alignment
- Direct (pixel-based) alignment
  - Search for alignment where most pixels agree

Direct Alignment

The simplest approach is a brute force search (hw1)

- Need to define image matching function
  - SSD, Normalized Correlation, edge matching, etc.
- Search over all parameters within a reasonable range:

e.g. for translation:

```
for tx=x0:step:x1,
  for ty=y0:step:y1,
     compare image1(x,y) to image2(x+tx,y+ty)
  end;
end;
```

Need to pick correct `x0,x1` and `step`

- What happens if `step` is too large?

Direct Alignment (brute force)

What if we want to search for more complicated transformation, e.g. homography?

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

```
for a=a0:astep:a1,
  for b=b0:bstep:b1,
    for c=c0:cstep:c1,
      for d=d0:dstep:d1,
        for e=e0:estep:e1,
          for f=f0:fstep:f1,
             for g=g0:gstep:g1,
                for h=h0:hstep:h1,
     compare image1 to H(image2)
end; end; end; end; end; end; end; end;
```

Problems with brute force

Not realistic

- Search in $O(N^8)$ is problematic
- Not clear how to set starting/stopping value and step

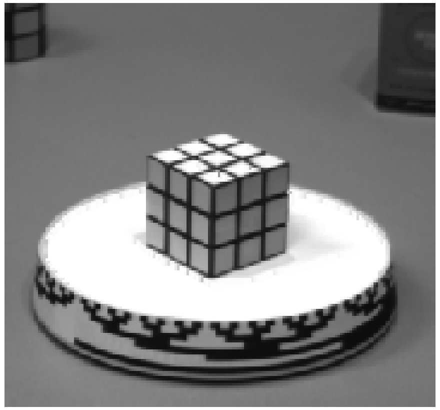
What can we do?

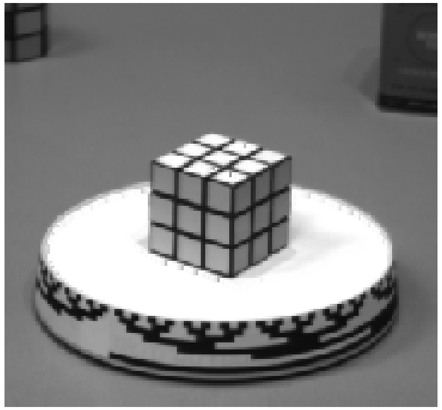
- Use pyramid search to limit starting/stopping/step values
- For special cases (rotational panoramas), can reduce search slightly to $O(N^4)$:
  - $H = K_1 R_1 R_2^{-1} K_2^{-1}$ (4 DOF: f and rotation)

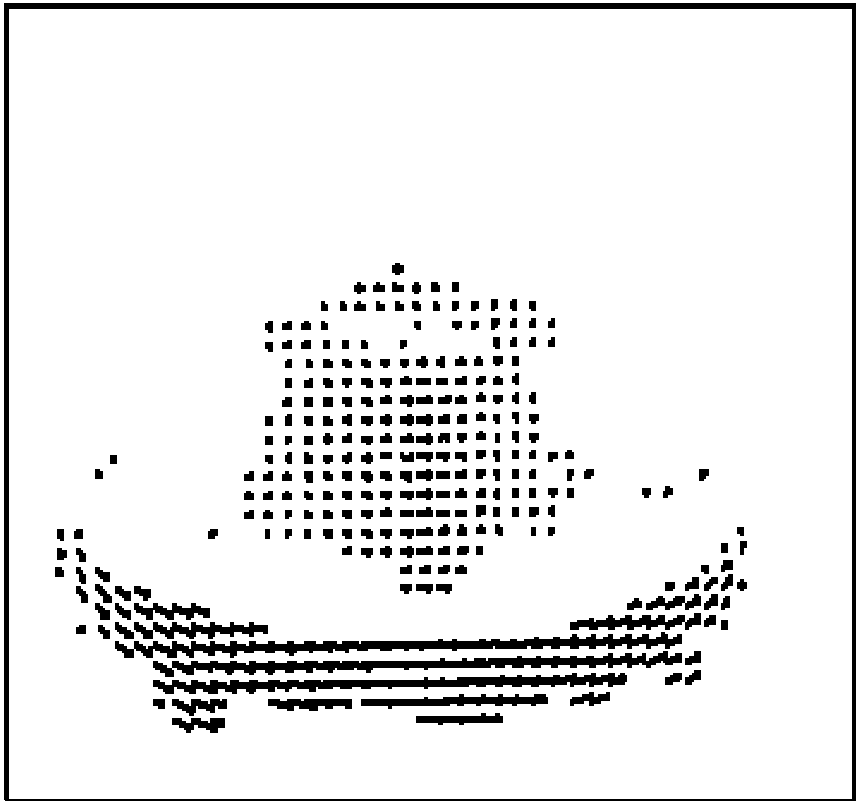
Alternative: gradient decent on the error function

- i.e. how do I tweak my current estimate to make the SSD error go down?
- Can do sub-pixel accuracy
- BIG assumption?
  - Images are already almost aligned (<2 pixels difference!)
  - Can improve with pyramid
- Same tool as in **motion estimation**

Motion estimation: Optical flow







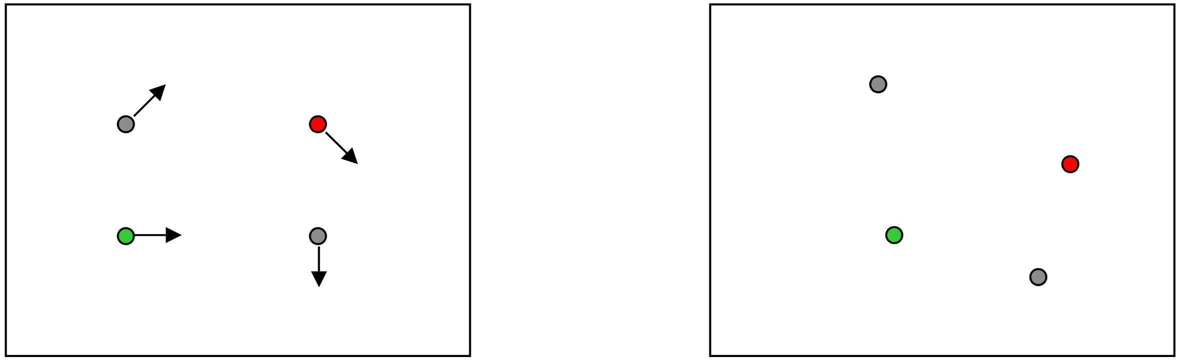
Will start by estimating motion of each pixel separately
Then will consider motion of entire image

Why estimate motion?

Lots of uses

- Track object behavior
- Correct for camera jitter (stabilization)
- Align images (mosaics)
- 3D shape reconstruction
- Special effects

Problem definition: optical flow



$H(x, y)$ $\qquad$ $I(x, y)$

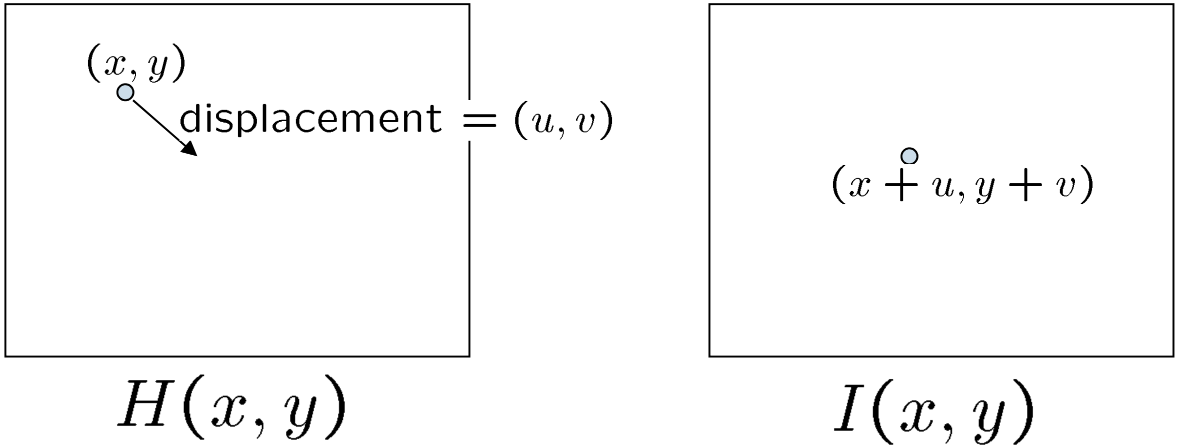
How to estimate pixel motion from image H to image I?

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?

  H(x,y)=I(x+u, y+v)

- 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

$$0 = I(x+u, y+v) - H(x,y) \qquad \text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

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

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

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

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

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?

2 unknowns, one equation

Intuitively, what does this constraint mean?

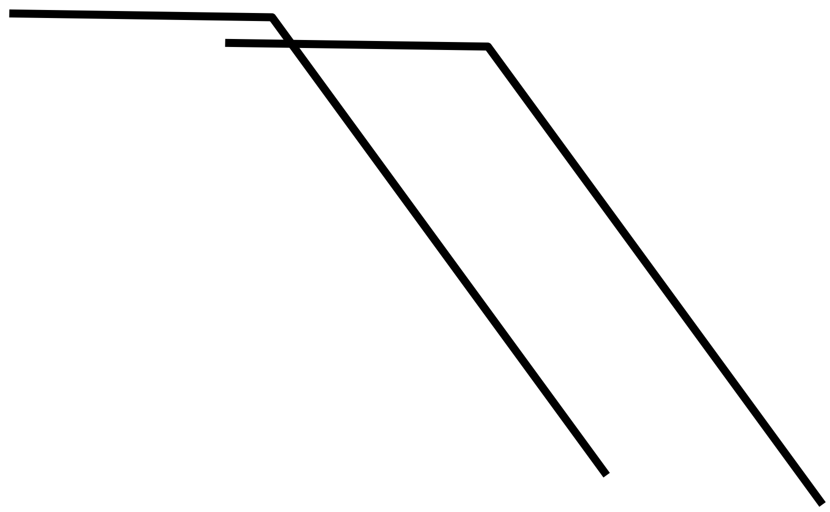
- The component of the flow in the gradient direction is determined
- The component of the flow parallel to an edge is unknown

This explains the Barber Pole illusion

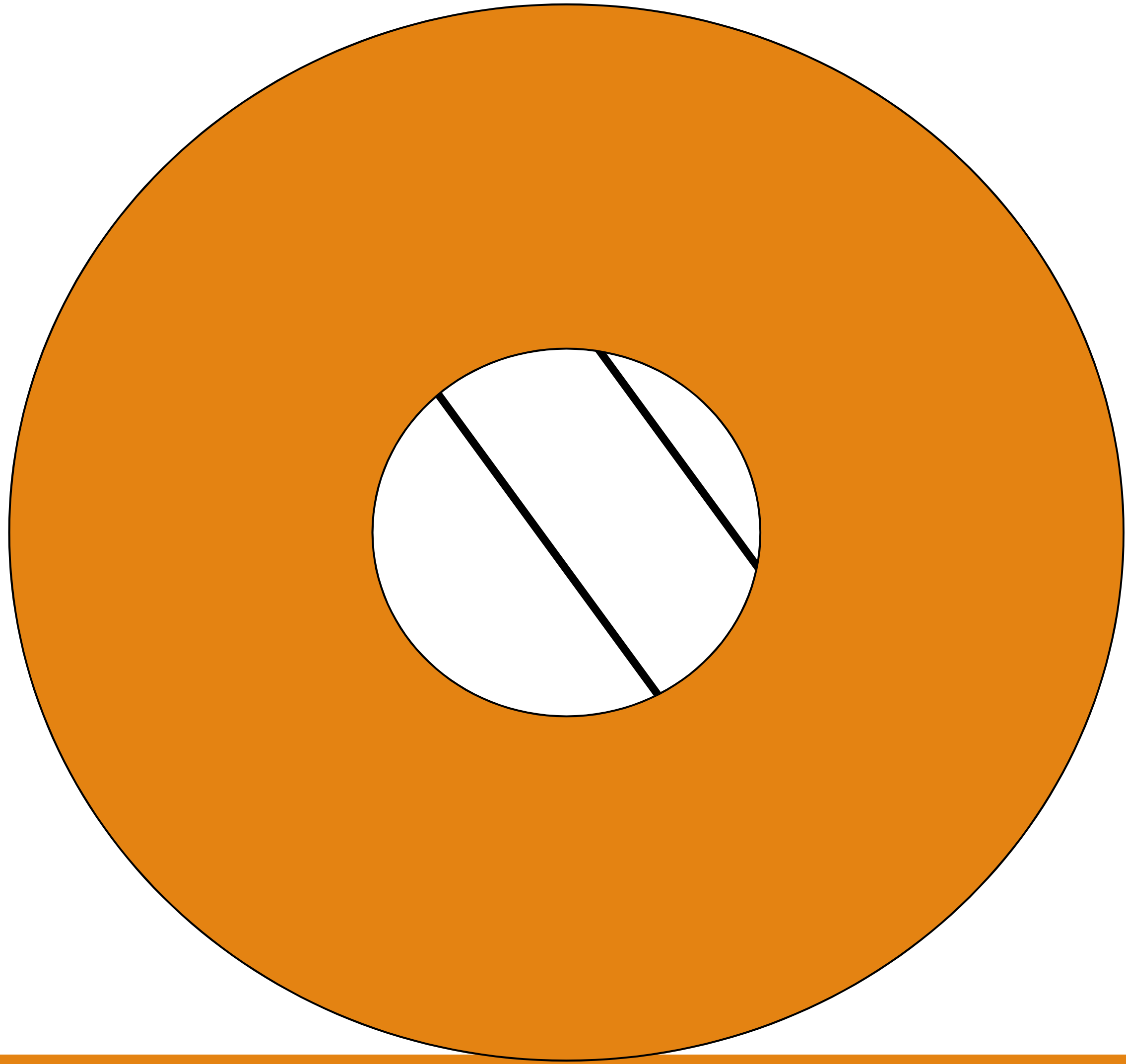
http://www.sandlotscience.com/Ambiguous/Barberpole_Illusion.htm

http://en.wikipedia.org/wiki/Barber's_pole

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!

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

$$\underset{25\times 2}{\underbrace{\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix}}_{A}} \underset{2\times 1}{\underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d}} = - \underset{25\times 1}{\underbrace{\begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}}_{b}}$$

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!

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

$$\underset{75\times2}{\underbrace{\begin{bmatrix} I_x(\mathbf{p}_1)[0] & I_y(\mathbf{p}_1)[0] \\ I_x(\mathbf{p}_1)[1] & I_y(\mathbf{p}_1)[1] \\ I_x(\mathbf{p}_1)[2] & I_y(\mathbf{p}_1)[2] \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25})[0] & I_y(\mathbf{p}_{25})[0] \\ I_x(\mathbf{p}_{25})[1] & I_y(\mathbf{p}_{25})[1] \\ I_x(\mathbf{p}_{25})[2] & I_y(\mathbf{p}_{25})[2] \end{bmatrix}}_{A}} \underset{2\times1}{\underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d}} = - \underset{75\times1}{\underbrace{\begin{bmatrix} I_t(\mathbf{p}_1)[0] \\ I_t(\mathbf{p}_1)[1] \\ I_t(\mathbf{p}_1)[2] \\ \vdots \\ I_t(\mathbf{p}_{25})[0] \\ I_t(\mathbf{p}_{25})[1] \\ I_t(\mathbf{p}_{25})[2] \end{bmatrix}}_{b}}$$

Note that RGB is not enough to disambiguate because R, G & B are correlated
Just provides better gradient

Lukas-Kanade flow

Prob: we have more equations than unknowns

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

Solution: solve least squares problem

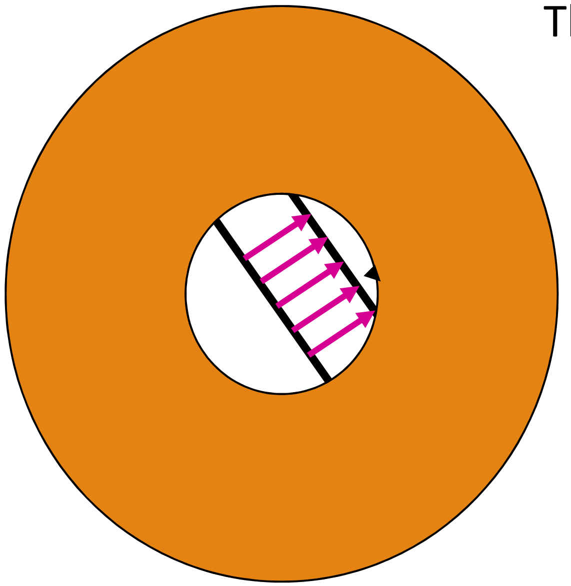
- minimum least squares solution given by solution (in d) of:

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

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

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

Aperture Problem and Normal Flow

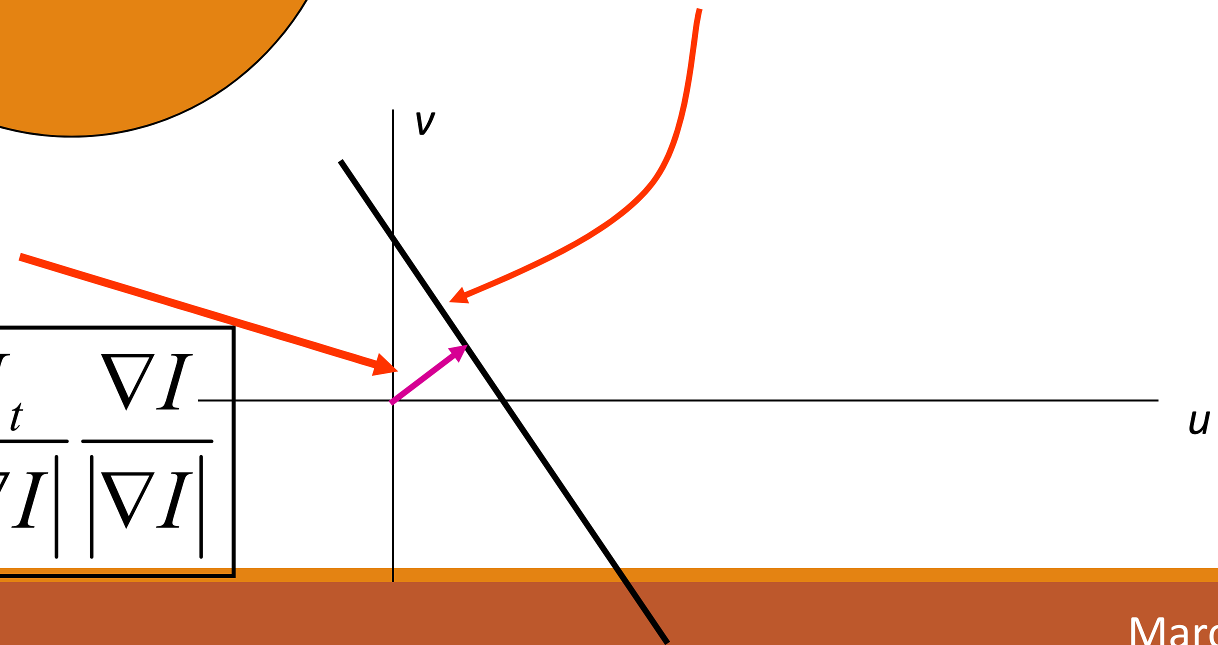


The gradient constraint:

$$I_x u + I_y v + I_t = 0$$

$$\nabla I \bullet \vec{U} = 0$$

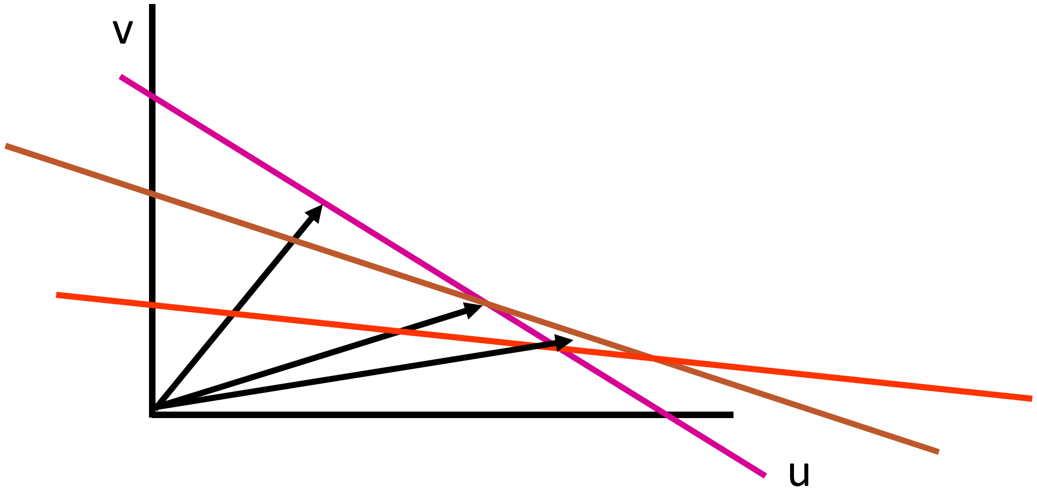
Defines a line in the *(u,v)* space



Normal Flow:

$$u_{\perp} = -\frac{I_t}{|\nabla I|}\frac{\nabla I}{|\nabla I|}$$

Combining Local Constraints



$$\nabla I^1 \bullet U = -I_t^1$$

$$\nabla I^2 \bullet U = -I_t^2$$

$$\nabla I^3 \bullet U = -I_t^3$$

etc.

Conditions for solvability

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

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

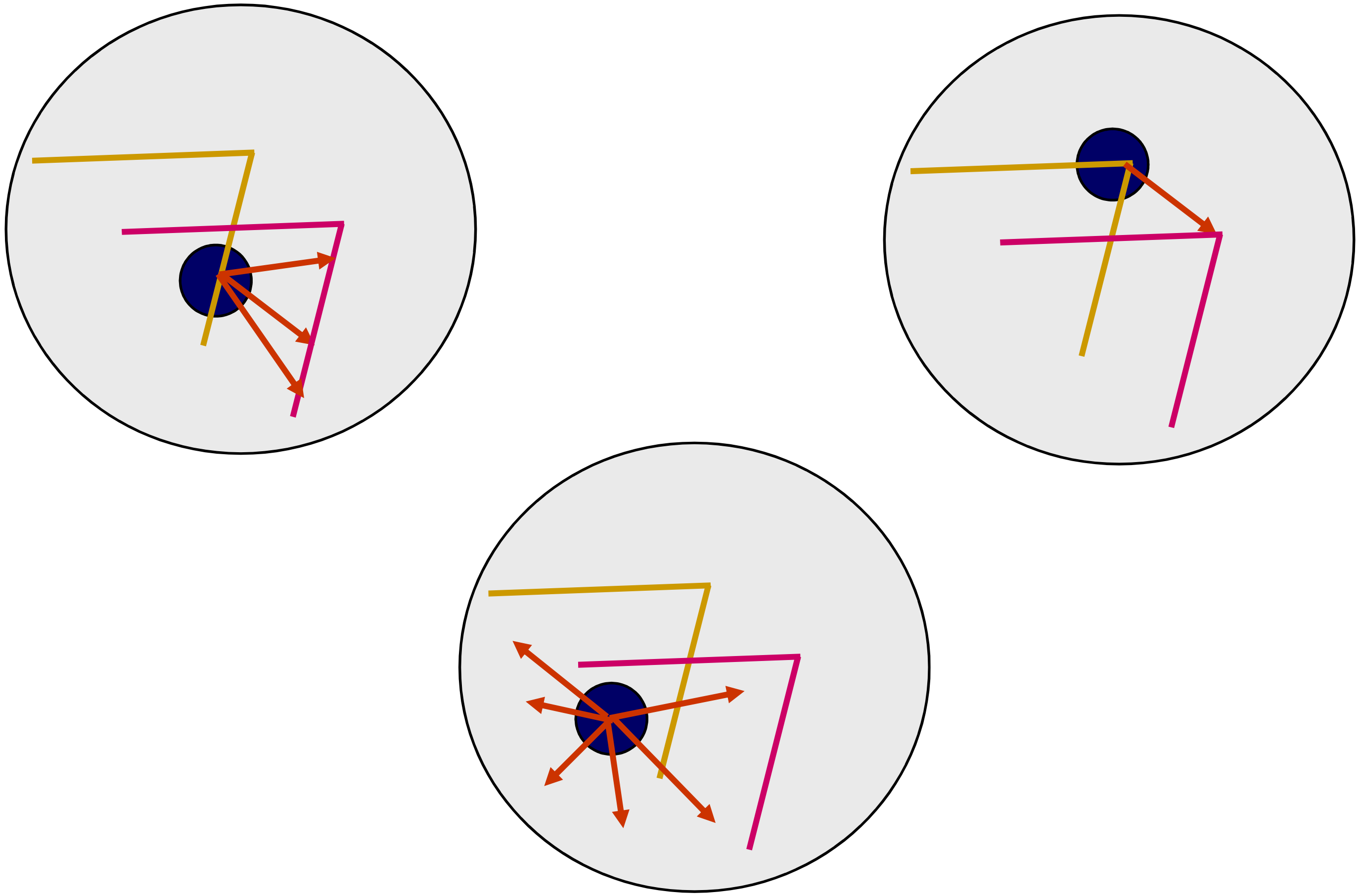
When is This Solvable?

- A^TA should be invertible
- A^TA should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of A^TA should not be too small
- A^TA should be well-conditioned
  - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

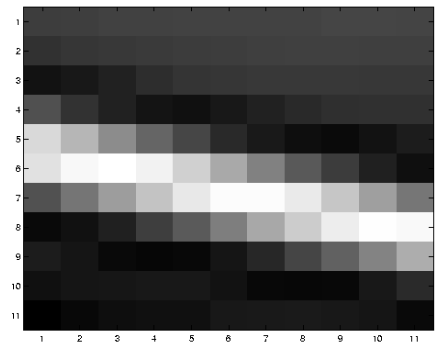
A^TA is solvable when there is no aperture problem

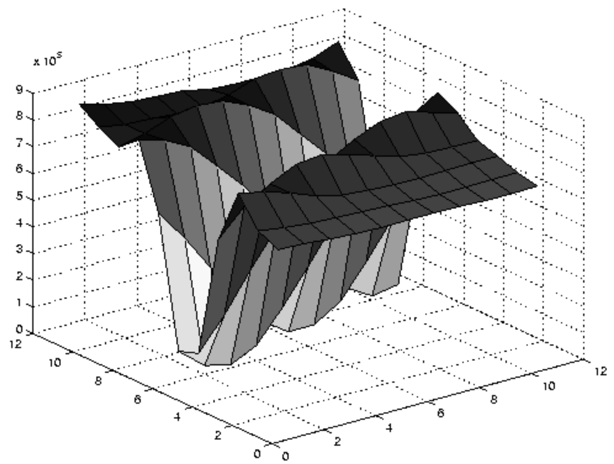
$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \ I_y] = \sum \nabla I (\nabla I)^T$$

Local Patch Analysis



Edge



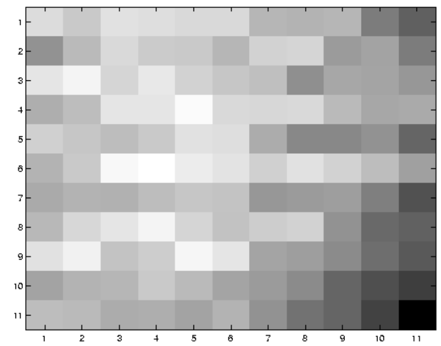


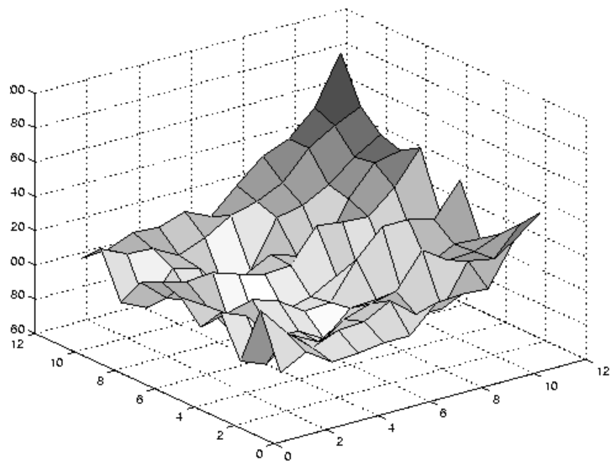
$$\sum \nabla I (\nabla I)^T$$

– large gradients, all the same

– large λ_1, small λ_2

Low texture region



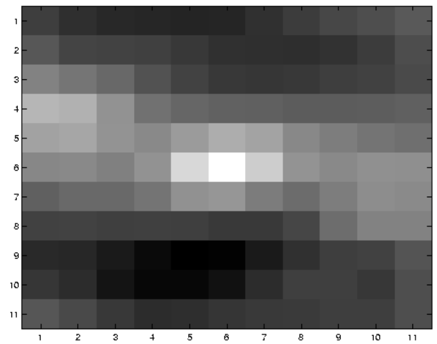


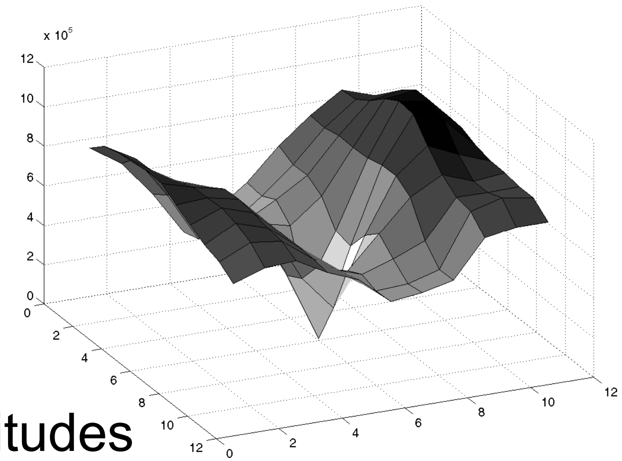
$$\sum \nabla I (\nabla I)^T$$

– 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 $A^T A$ 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

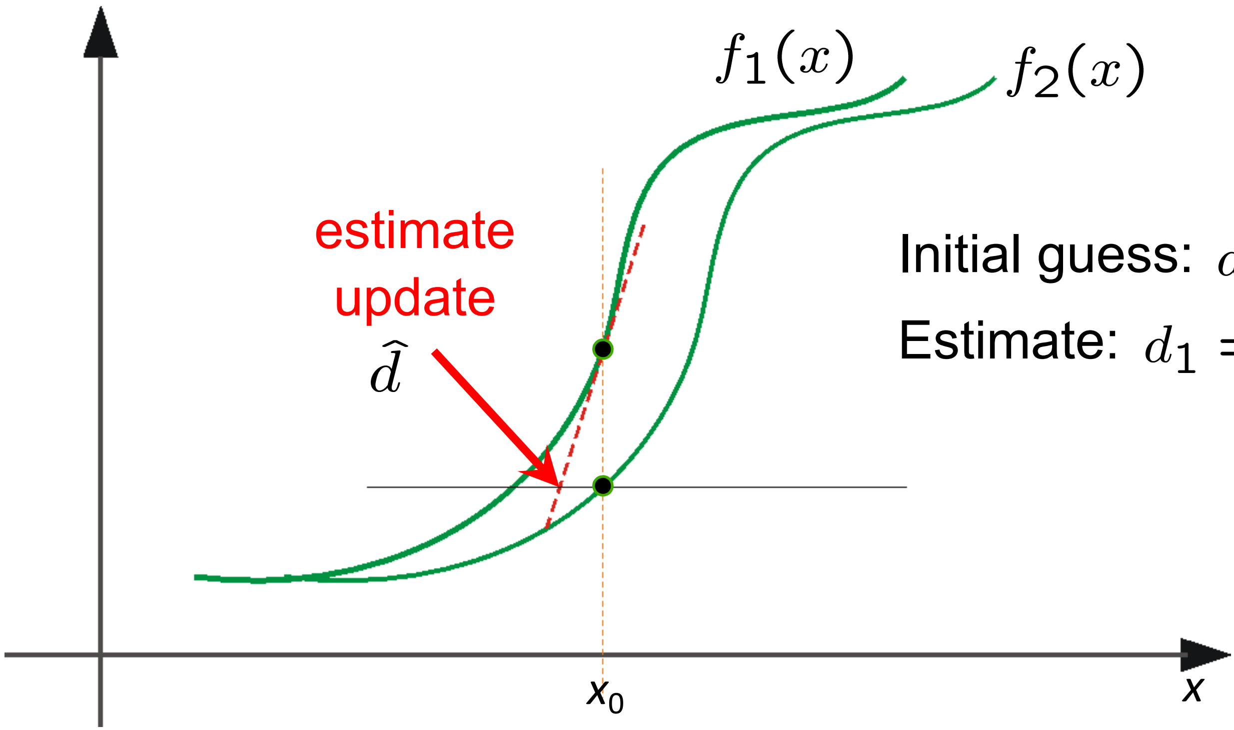
Iterative Lukas-Kanade Algorithm

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

Optical Flow: Iterative Estimation

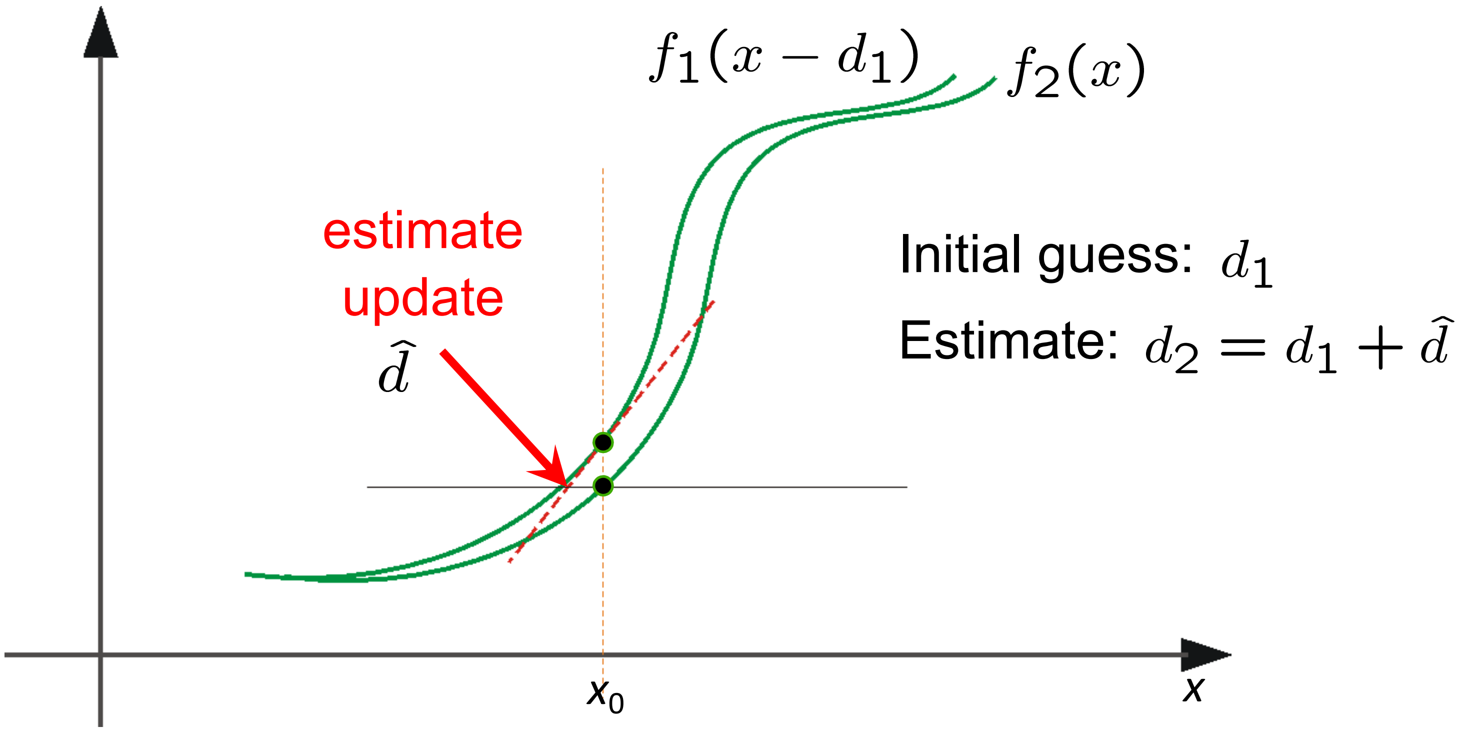


Initial guess: $d_0 = 0$

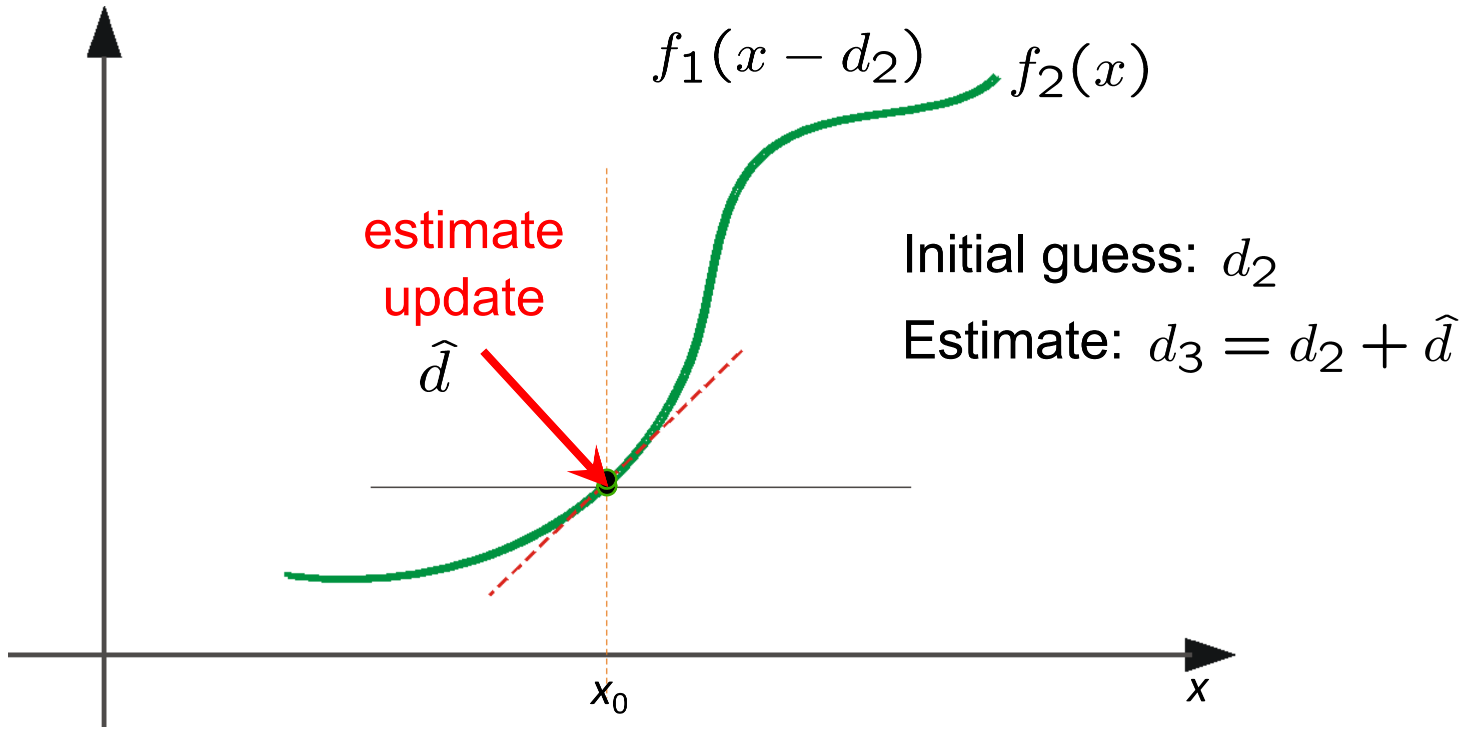
Estimate: $d_1 = d_0 + \hat{d}$

(using *d* for *displacement* here instead of *u*)

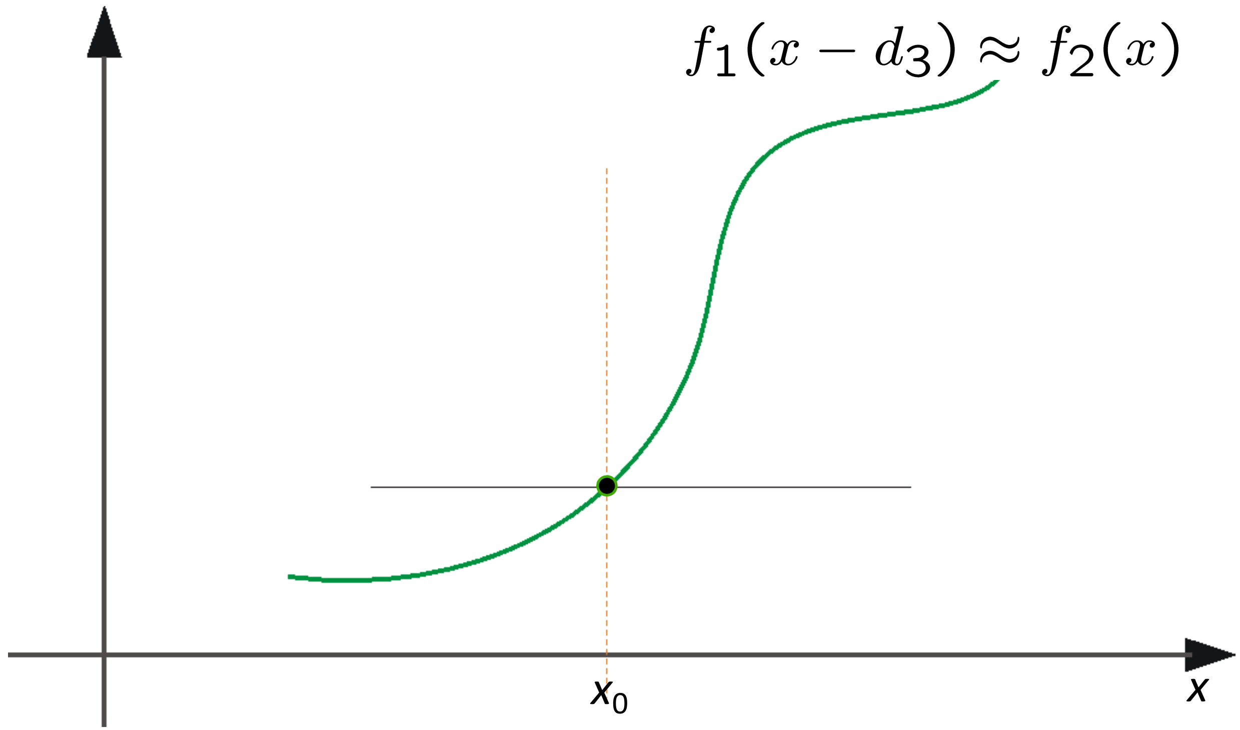
Optical Flow: Iterative Estimation



Optical Flow: Iterative Estimation



Optical Flow: Iterative Estimation



Optical Flow: Iterative Estimation

Some Implementation Issues:

- Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)
- Warp one image, take derivatives of the other so you don't need to re-compute the gradient after each iteration.
- Often useful to low-pass filter the images before motion estimation (for better derivative estimation, and linear approximations to image intensity)

Revisiting the small motion assumption

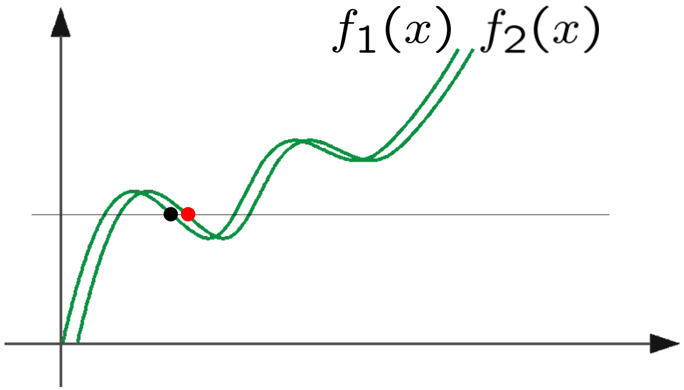
Is this motion small enough?

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

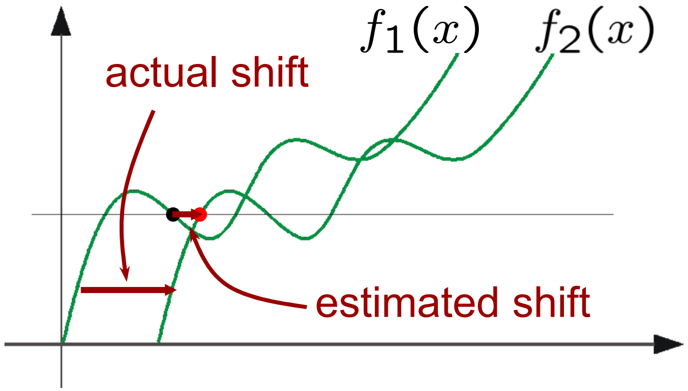
Optical Flow: Aliasing

Temporal aliasing causes ambiguities in optical flow because images can have many pixels with the same intensity.

I.e., how do we know which 'correspondence' is correct?



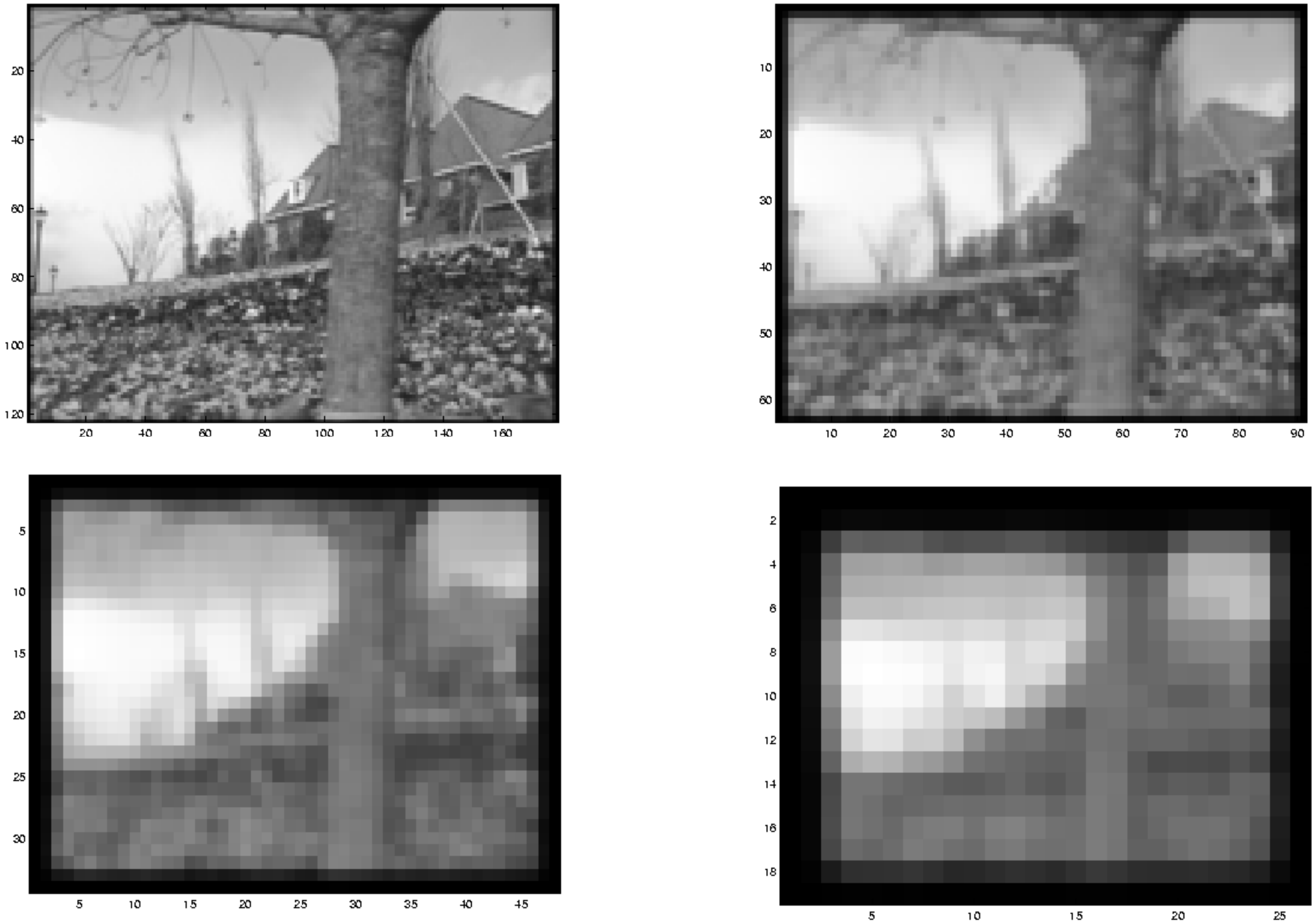
nearest match is correct (no aliasing)



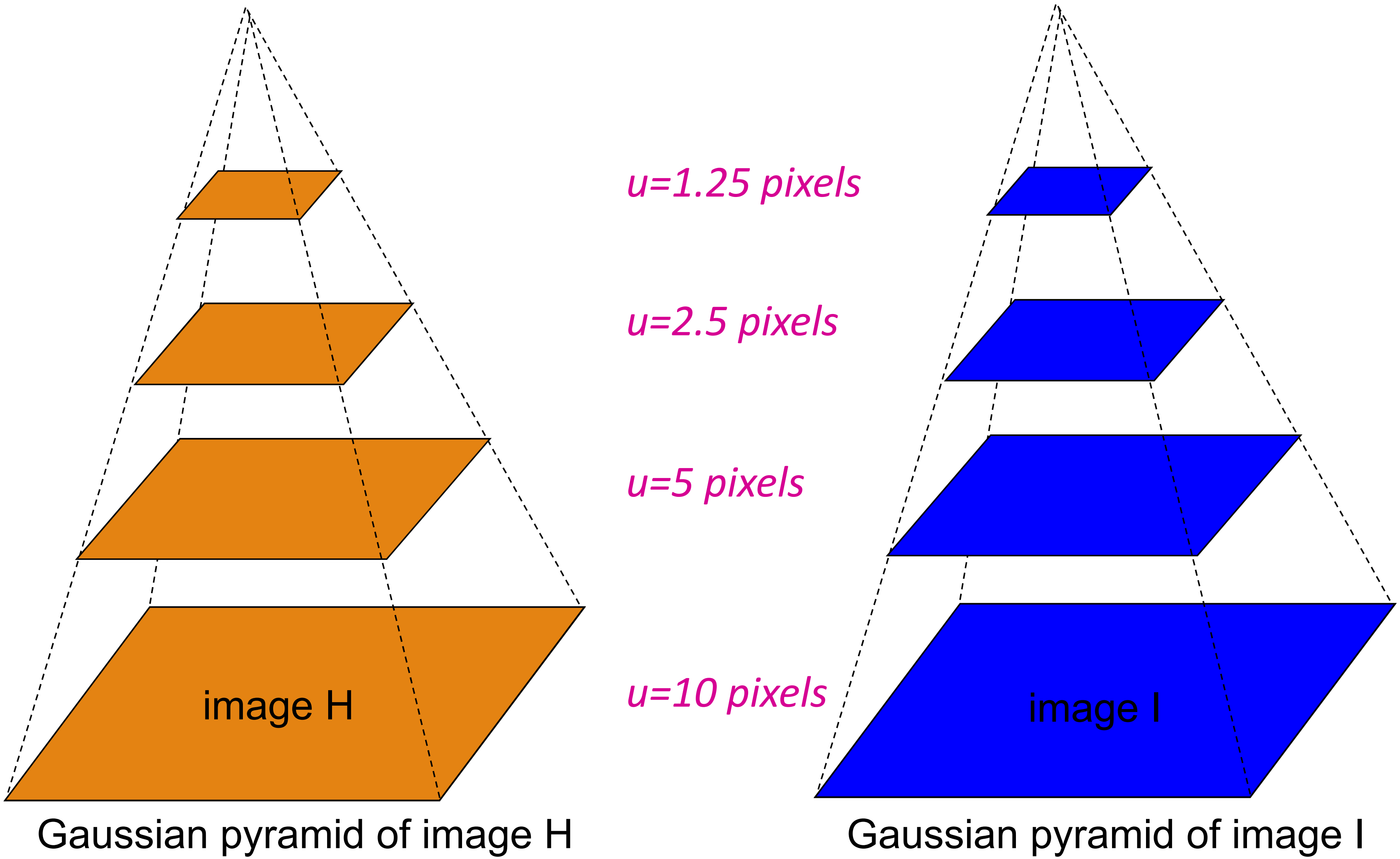
nearest match is incorrect (aliasing)

To overcome aliasing: coarse-to-fine estimation.

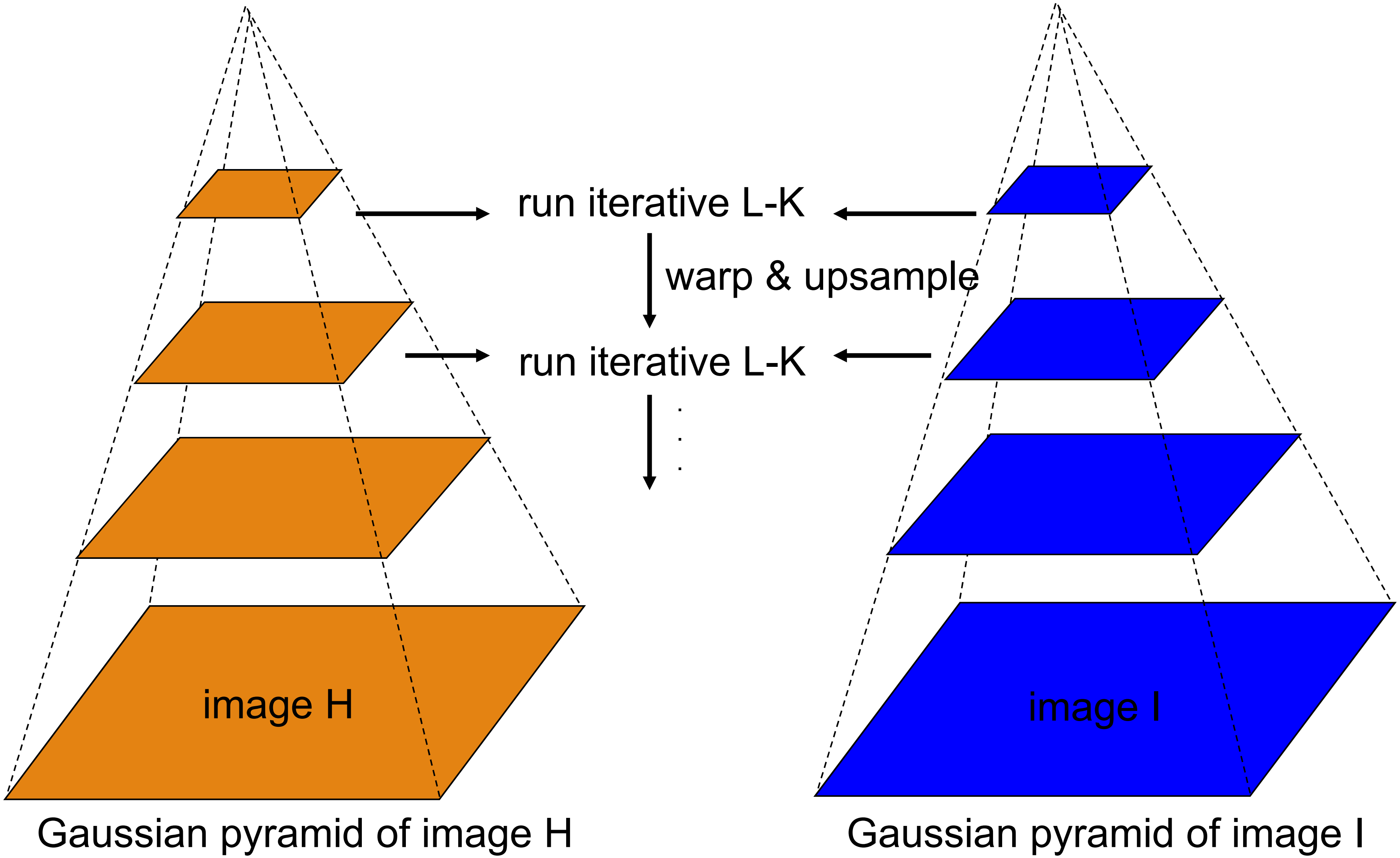
Reduce the resolution!



Coarse-to-fine optical flow estimation



Coarse-to-fine optical flow estimation



Beyond Translation

So far, our patch can only translate in (u,v)

What about other motion models?

- rotation, affine, perspective

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \sum_i \mathbf{J}\nabla\mathbf{I}(\nabla\mathbf{I})^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}$$

$$\mathbf{A}^{\mathrm{T}}\mathbf{b} = -\sum_i \mathbf{J}^{\mathrm{T}} I_t (\nabla\mathbf{I})^{\mathrm{T}}$$

Same thing but need to add an appropriate Jacobian

See Szeliski's survey of Panorama stitching

Recap: Classes of Techniques

Feature-based methods (e.g. SIFT+Ransac+regression)

- Extract visual features (corners, textured areas) and track them over multiple frames
- Sparse motion fields, but possibly robust tracking
- Suitable especially when image motion is large (10-s of pixels)

Direct-methods (e.g. optical flow)

- Directly recover image motion from spatio-temporal image brightness variations
- Global motion parameters directly recovered without an intermediate feature motion calculation
- Dense motion fields, but more sensitive to appearance variations
- Suitable for video and when image motion is small (< 10 pixels)

Block-based motion prediction

Break image up into square blocks

Estimate translation for each block

Use this to predict next frame, code difference (MPEG-2)