# Local Features, Image Alignment and Matching

Lecture 12

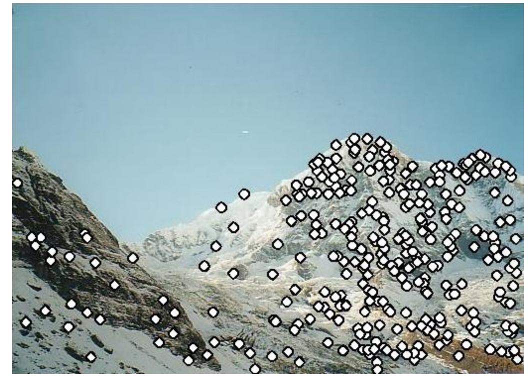
Slides from: N. Snavely, S. Lazebnik, S. Seitz, M. Pollefeys, A. Efros.

# Motivating Problem

- How do we build panorama?
- Detection & Matching of local features in the two images

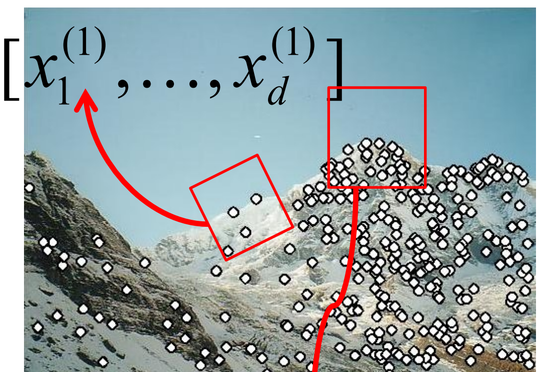
# Local features: main components

1) **Detection:** Identify the interest points



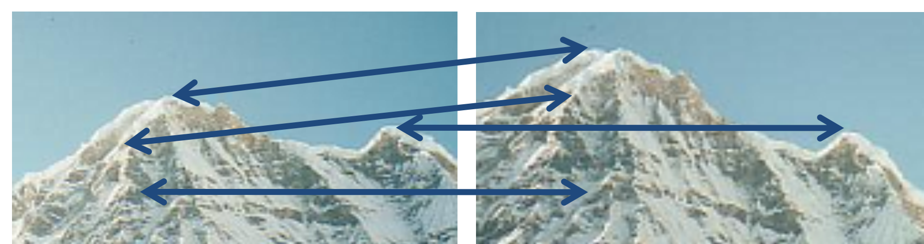
2) **Description**: Extract vector feature descriptor surrounding each interest point.

$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$



$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

3) **Matching:** Determine correspondence between descriptors in two views

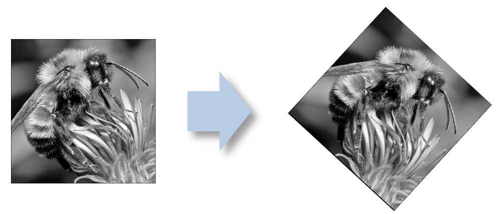


Kristen Grauman

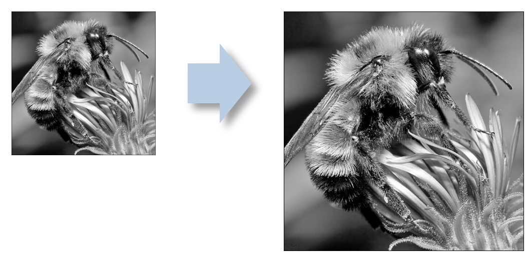
# Image transformations

- Geometric

**Rotation**

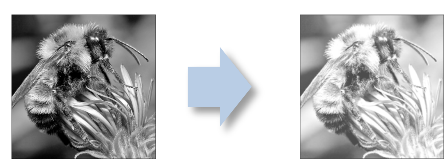


**Scale**



- Photometric

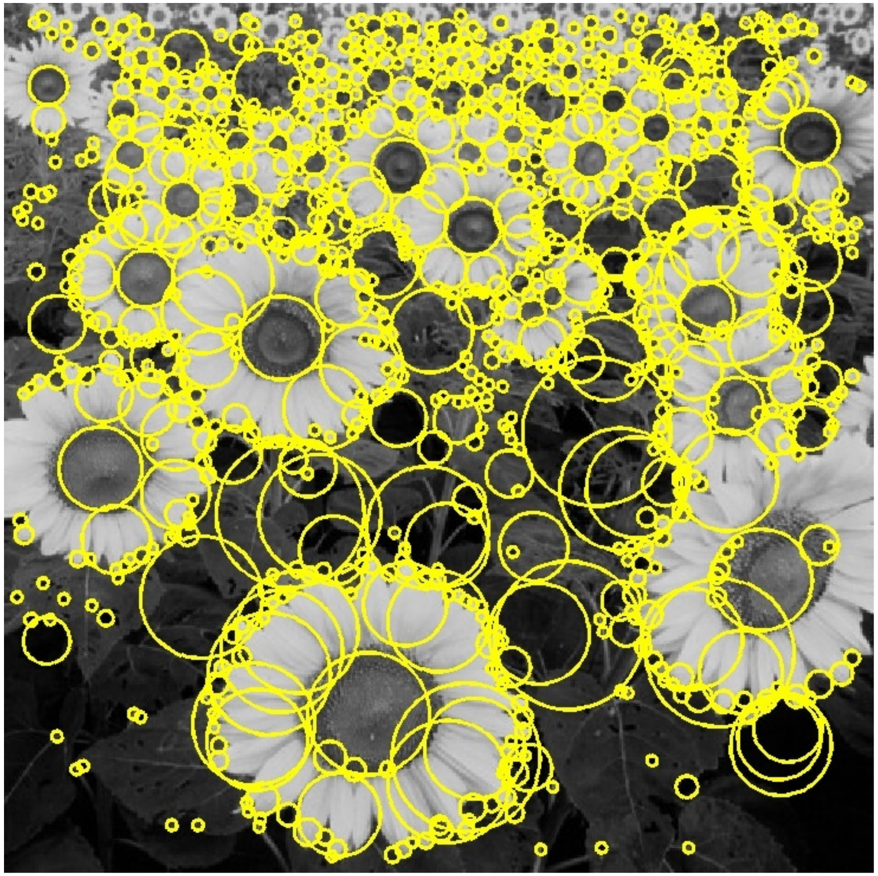
**Intensity change**



# Invariance and covariance

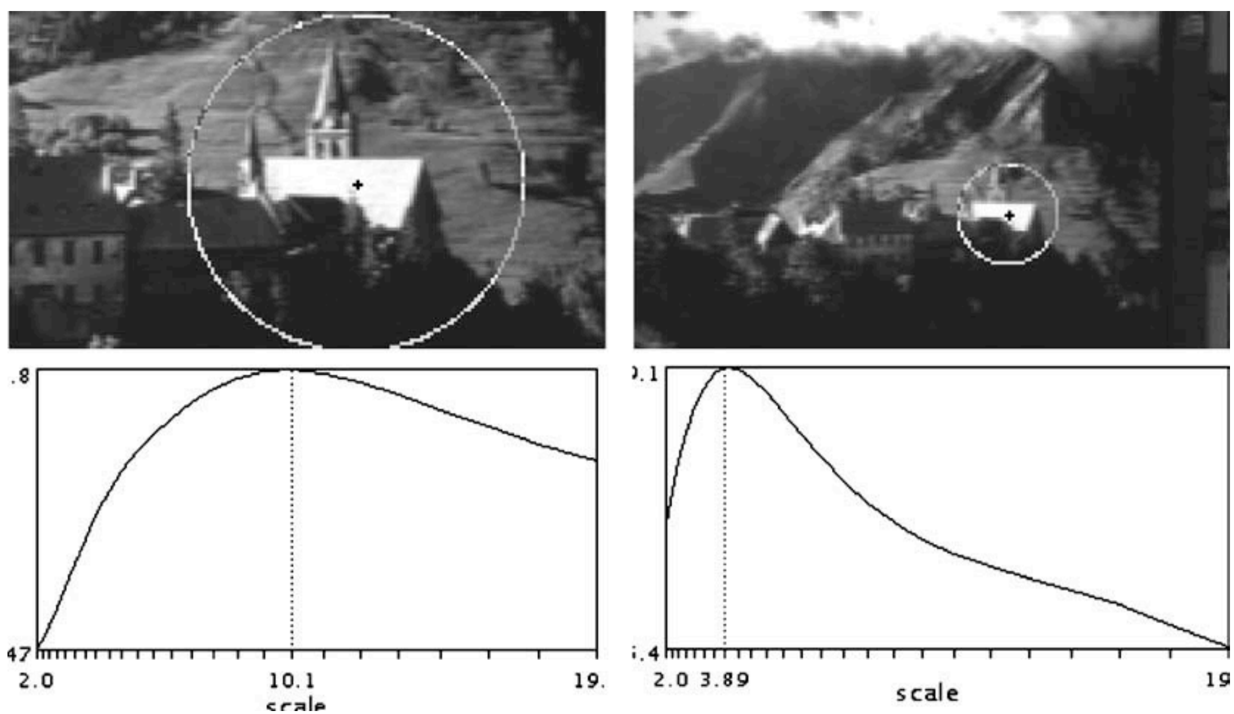
- We want interest points to be *invariant* to photometric transformations and *covariant* to geometric transformations
  - **Invariance:** image is transformed and corner locations do not change
  - **Covariance:** if we have two transformed versions of the same image, features should be detected in corresponding locations

# Blob detection with scale selection

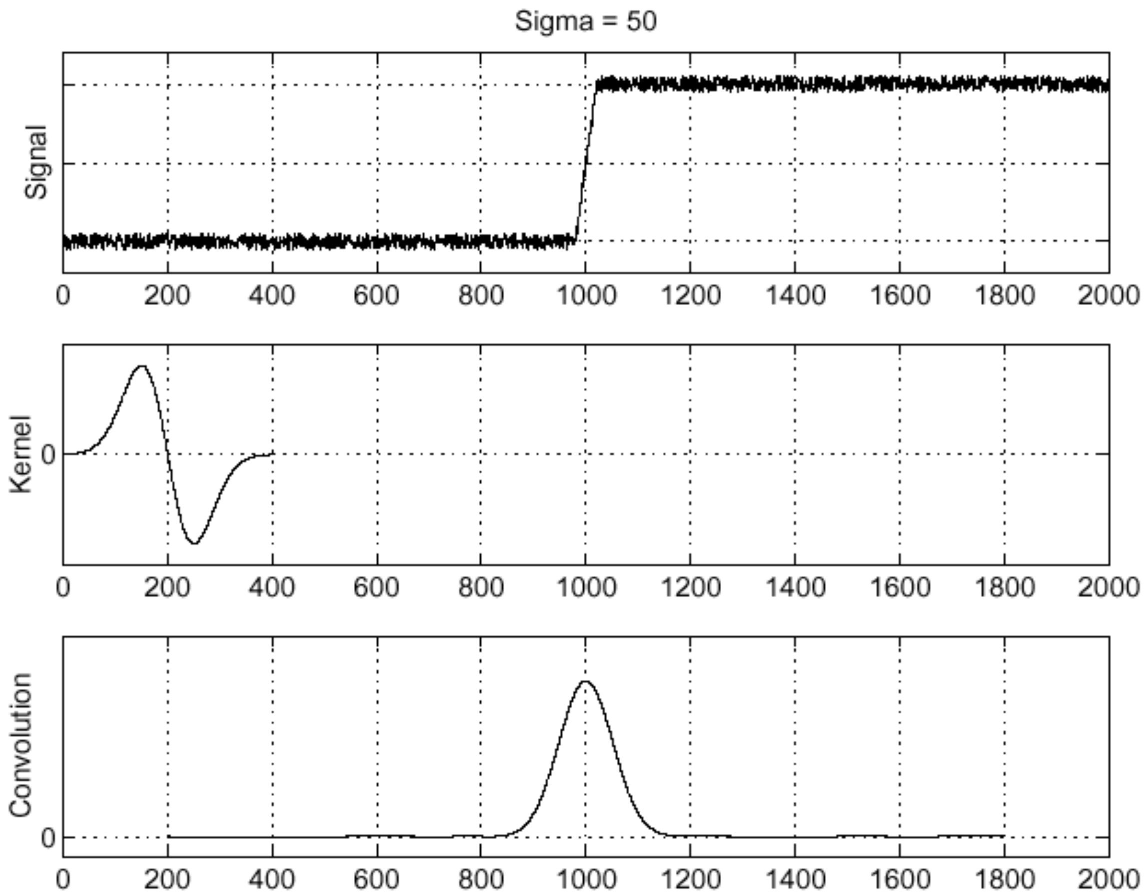


# Achieving scale covariance

- Goal: independently detect corresponding regions in scaled versions of the same image
- Need *scale selection* mechanism for finding characteristic region size that is *covariant* with the image transformation



# Edge detection



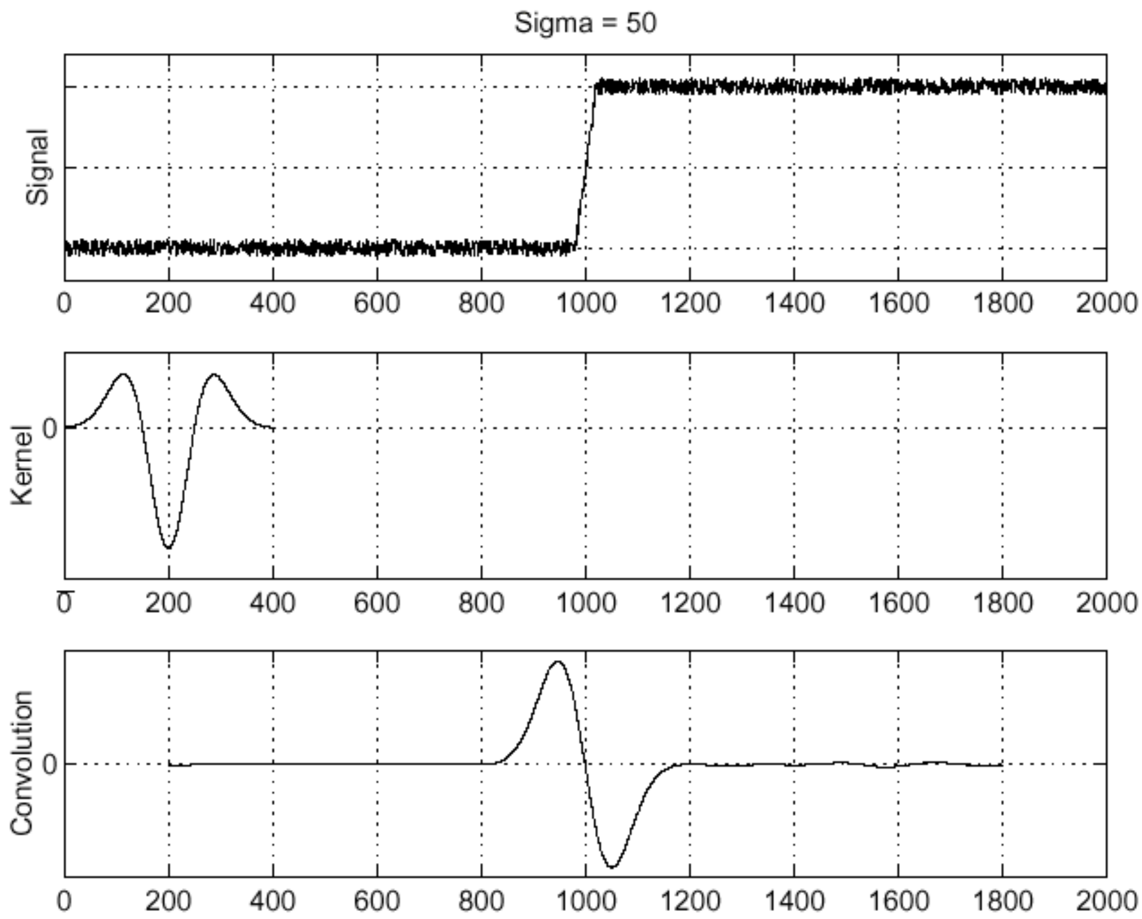

$f$ Edge

$\frac{d}{dx} g$ Derivative of Gaussian

$f * \frac{d}{dx} g$ Edge = maximum of derivative

Source: S. Seitz

# Edge detection, Take 2




$f$ Edge

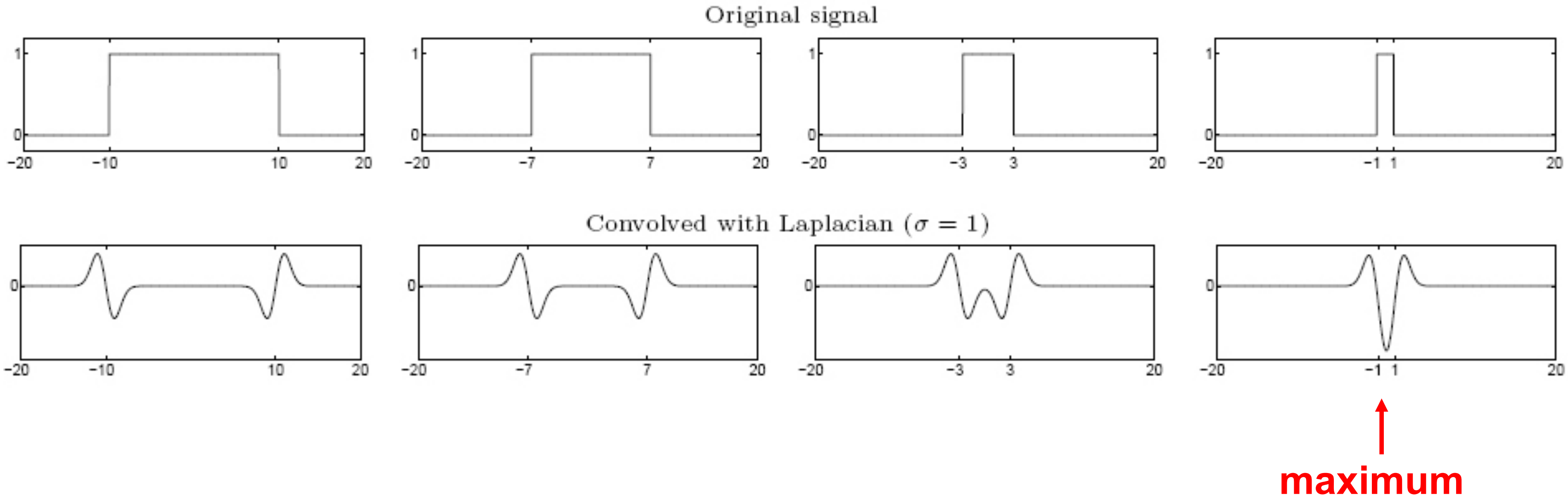
$\frac{d^2}{dx^2} g$ Second derivative of Gaussian (Laplacian)

$f * \frac{d^2}{dx^2} g$ Edge = zero crossing of second derivative

Source: S. Seitz

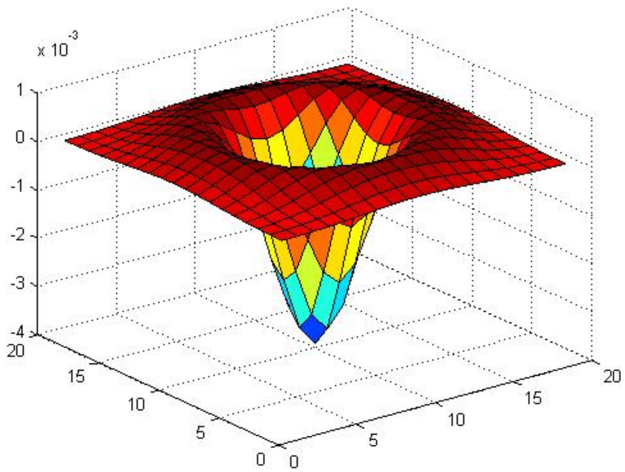
# From Edges to Blobs

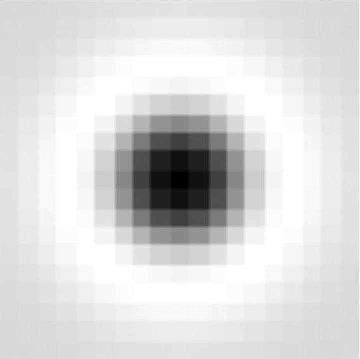
- Edge = ripple
- Blob = superposition of two ripples




**Spatial selection**: the magnitude of the Laplacian response will achieve a maximum at the center of the blob, provided the scale of the Laplacian is "matched" to the scale of the blob

# *Laplacian of Gaussian* (*LoG*)

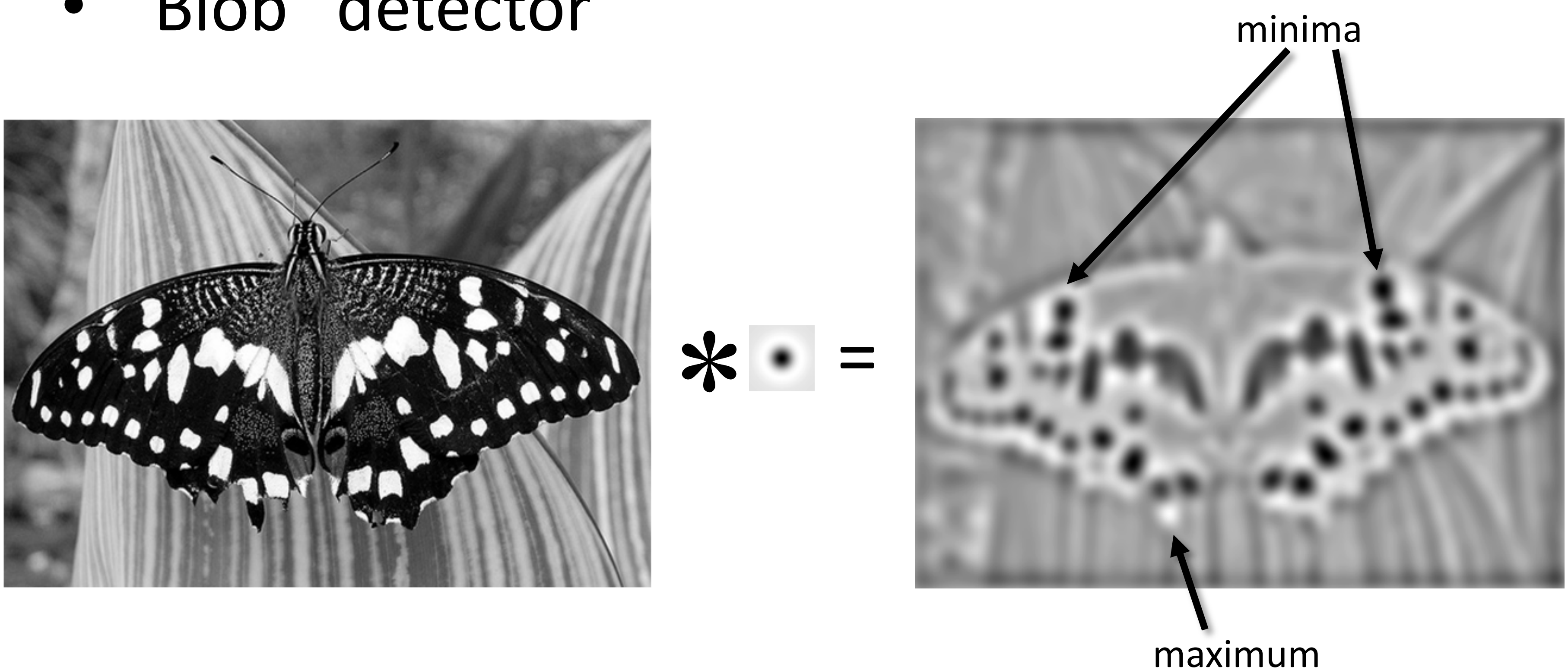


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

# Laplacian of Gaussian

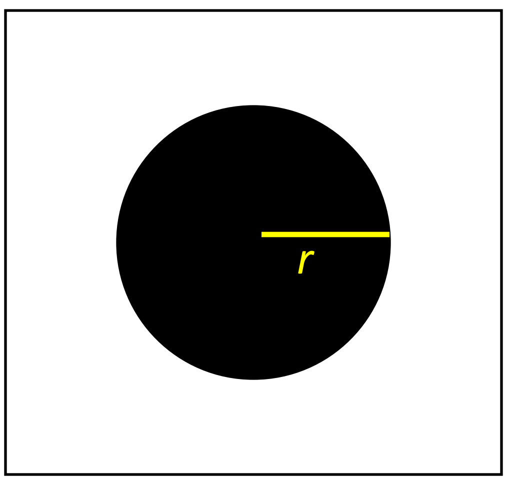
- “Blob” detector



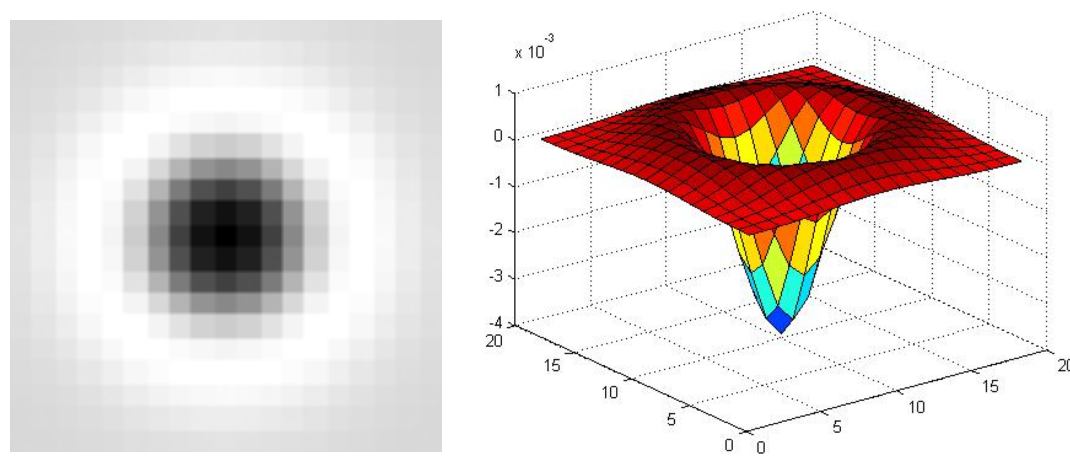

- Find maxima *and minima* of LoG operator in space and scale

# Scale selection

- At what scale does the Laplacian achieve a maximum response for a binary circle of radius r?



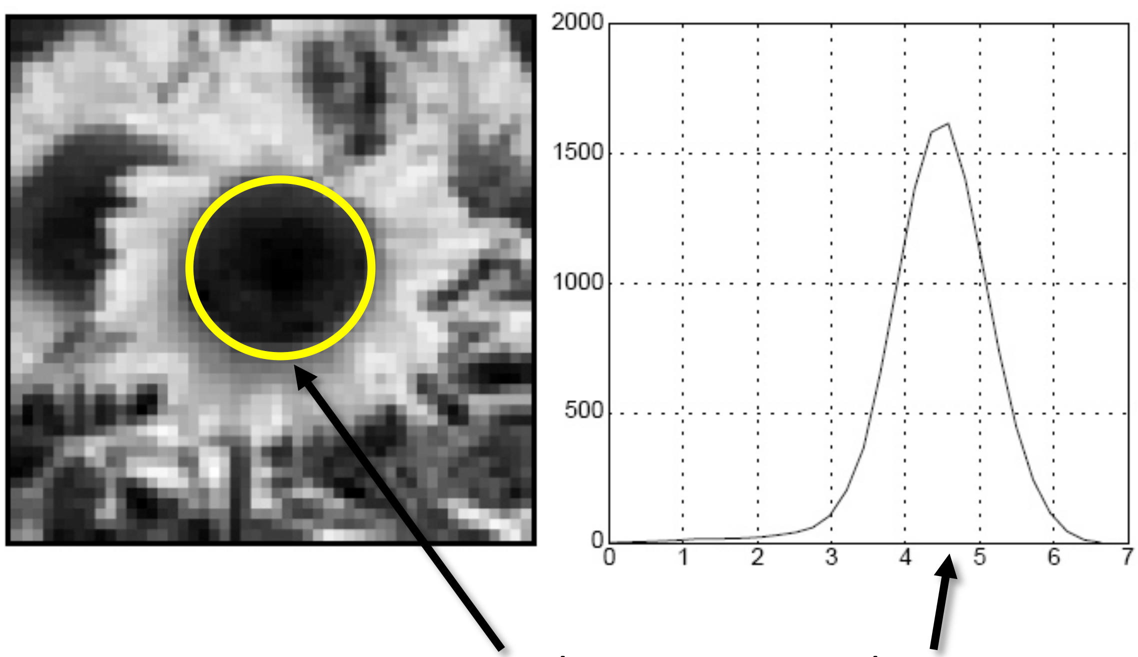

image




Laplacian

# Characteristic scale

- We define the characteristic scale as the scale that produces peak of Laplacian response

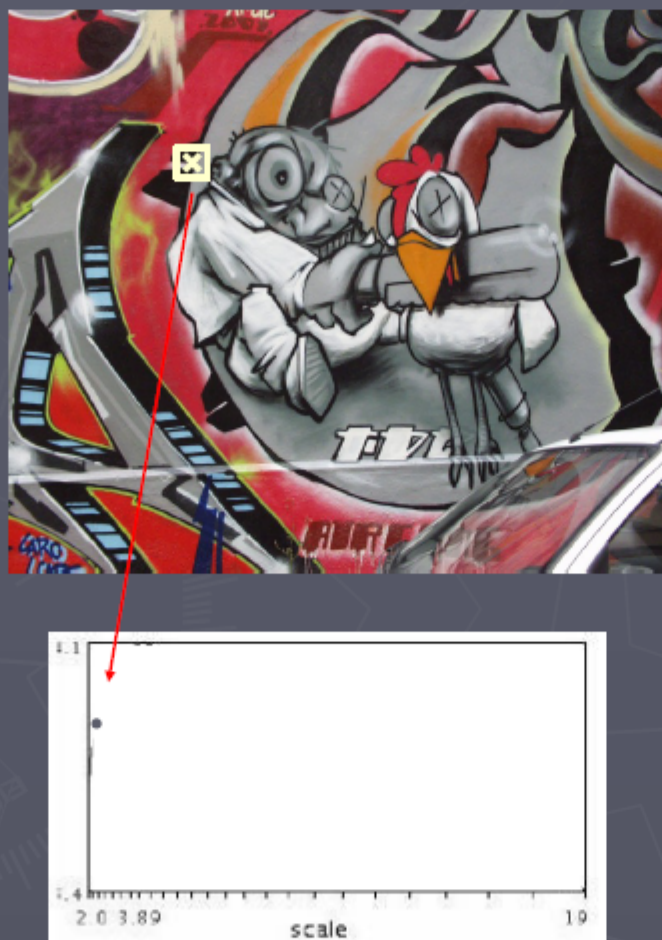



characteristic scale

T. Lindeberg (1998). ["Feature detection with automatic scale selection."]() *International Journal of Computer Vision* **30** (2): pp 77--116.

# Automatic scale selection

Lindeberg et al., 1996



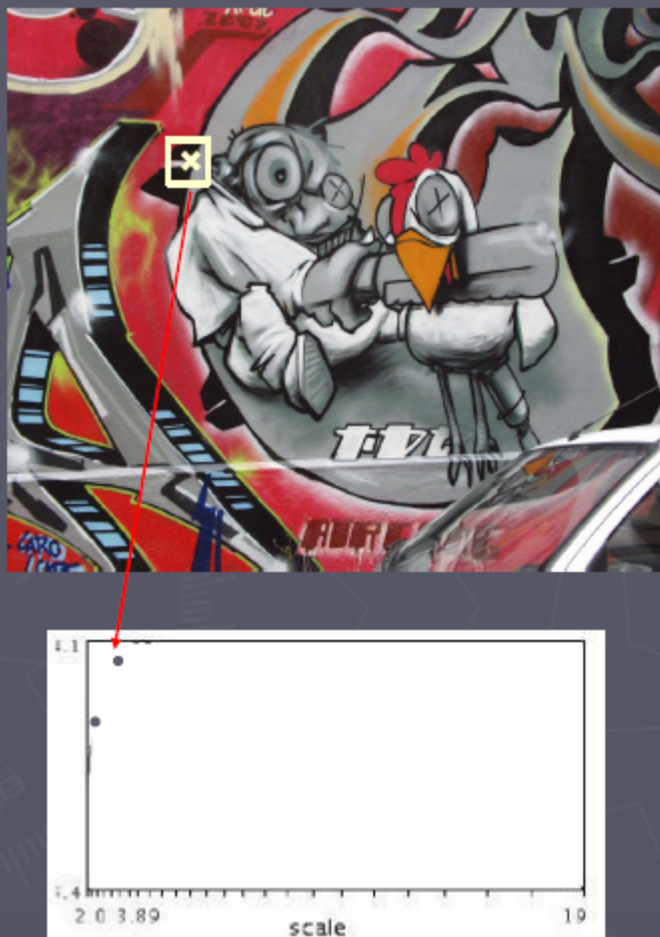

$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Slide from Tinne Tuytelaars

# Automatic scale selection

Function responses for increasing scale

Scale trace (signature)




$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

# Automatic scale selection

Function responses for increasing scale

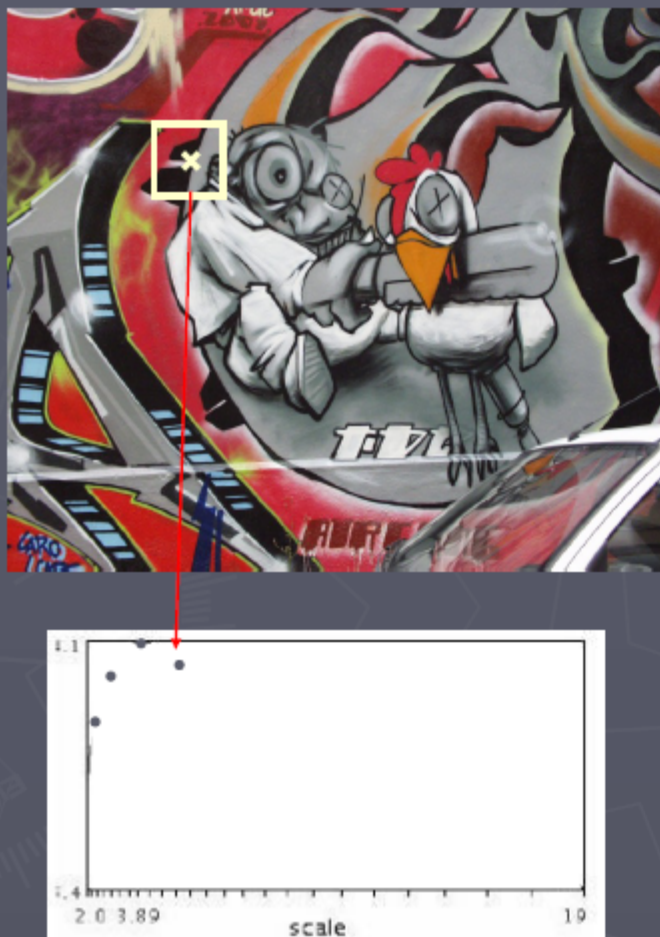
Scale trace (signature)


$f(I_{i_1 \ldots i_m}(x,\sigma))$

# Automatic scale selection

Function responses for increasing scale

Scale trace (signature)

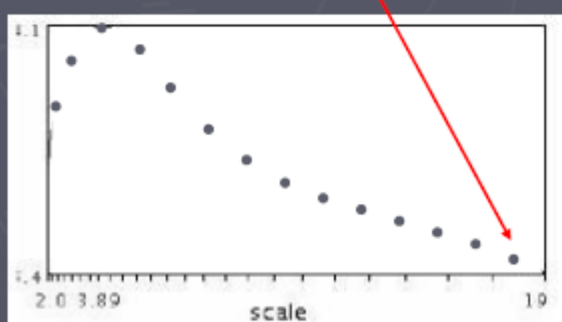



$f(I_{i_1 \ldots i_m}(x,\sigma))$

# Automatic scale selection

Function responses for increasing scale

Scale trace (signature)

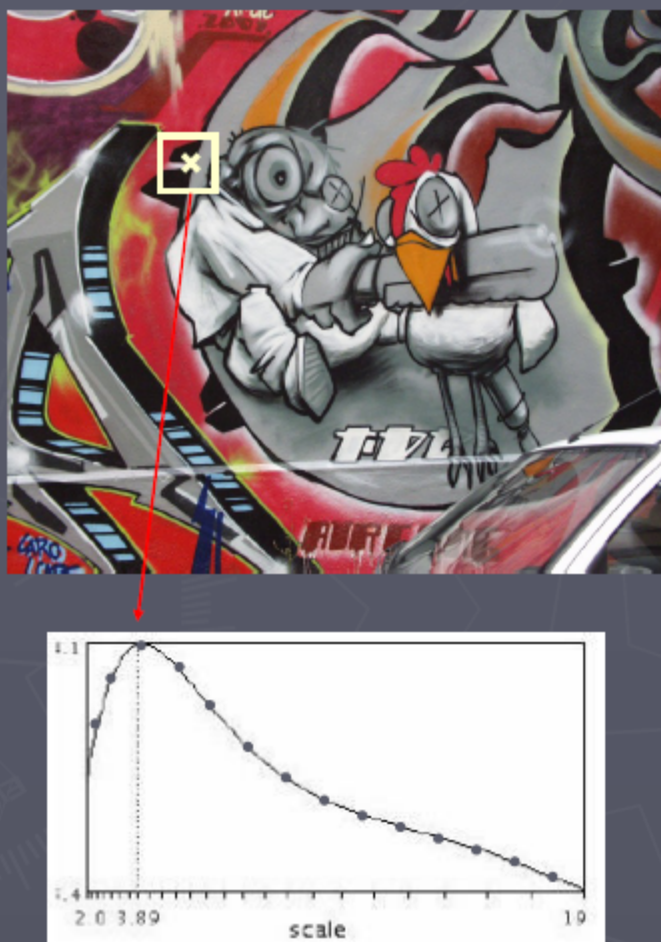



$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

# Automatic scale selection

Function responses for increasing scale

Scale trace (signature)

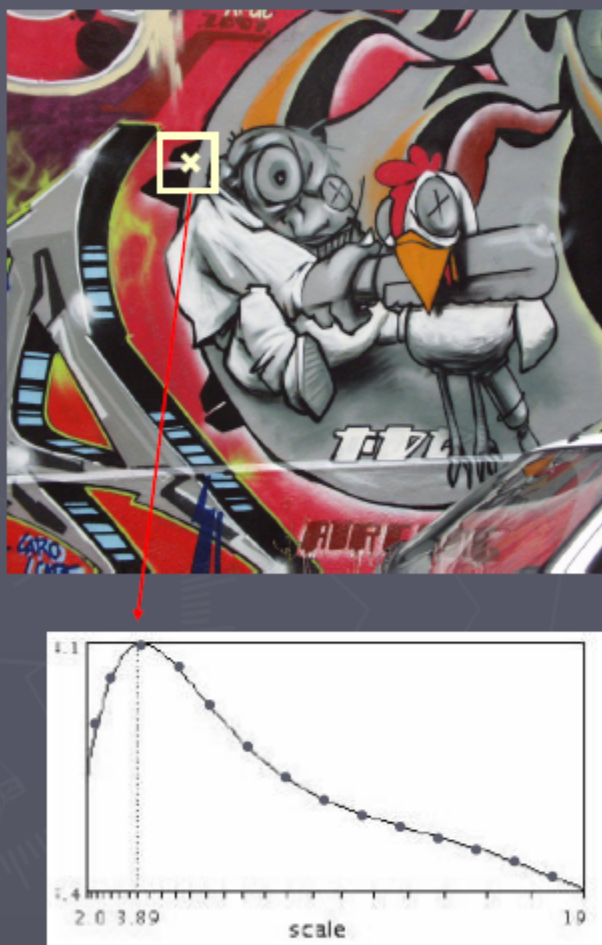



$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

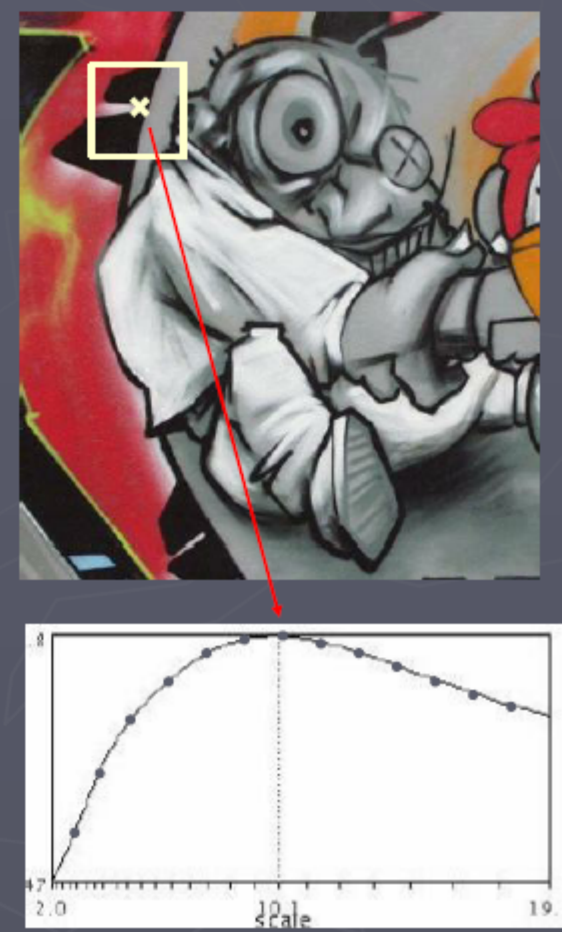
# Automatic scale selection

Function responses for increasing scale

Scale trace (signature)



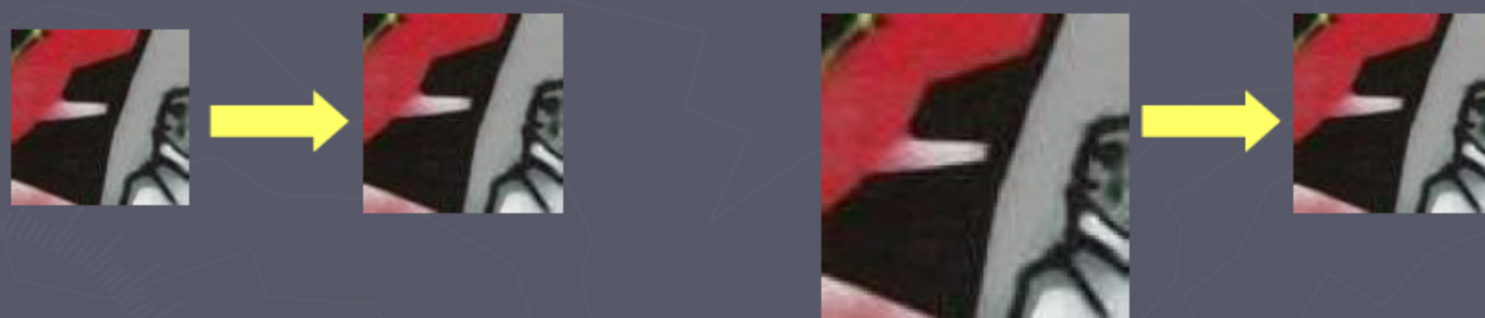

$$f(I_{i_1 \ldots i_m}(x,\sigma))$$



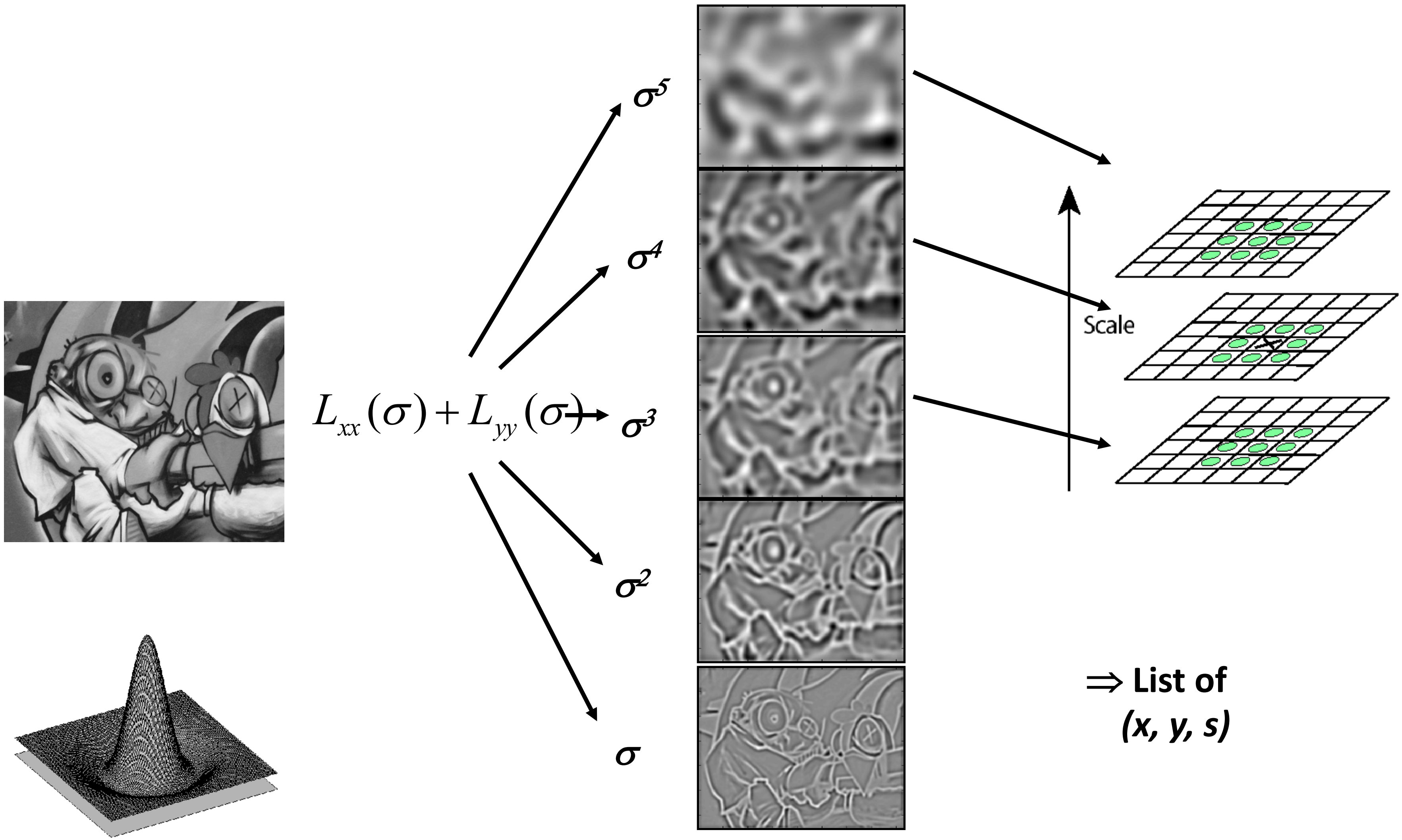

$$f(I_{i_1 \ldots i_m}(x',\sigma'))$$

# Automatic scale selection

Normalize: rescale to fixed size



# Find local maxima in position-scale space



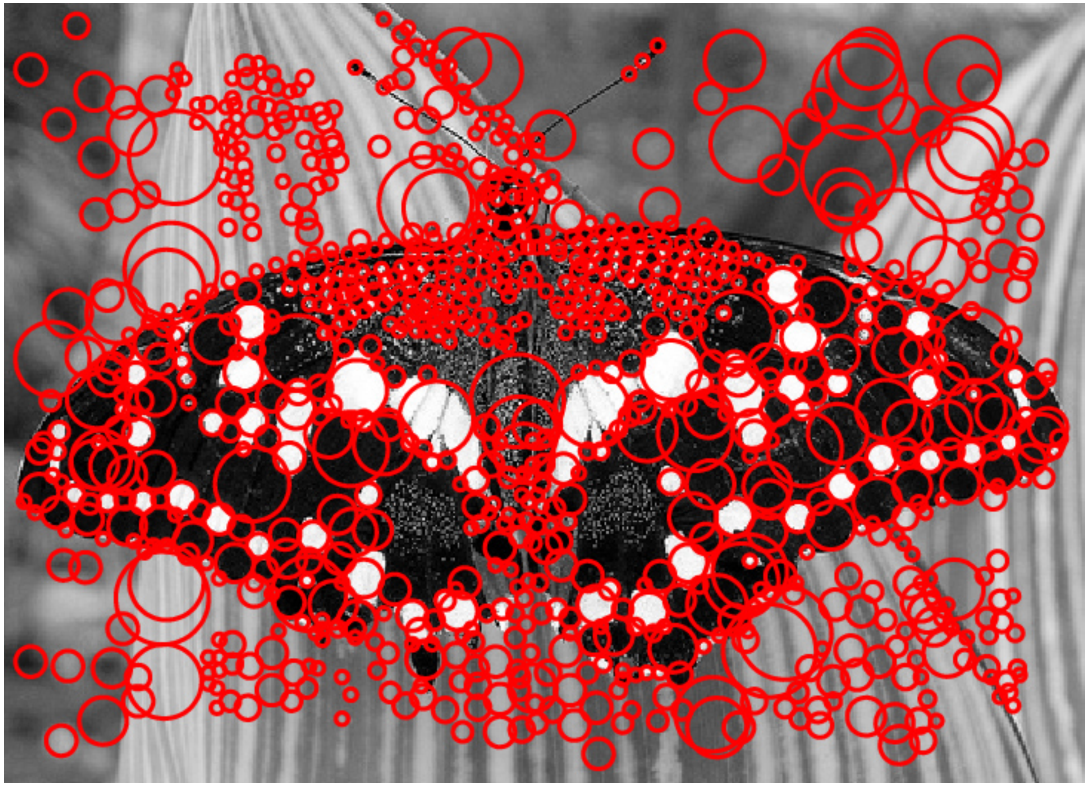

$\Rightarrow$ **List of *(x, y, s)***

# Scale-space blob detector: Example

# Scale-space blob detector: Example

sigma = 11.9912

# Scale-space blob detector: Example



# Scale Invariant Detection

- Functions for determining scale

$$f = \mathrm{Kernel} * \mathrm{Image}$$

Kernels:

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

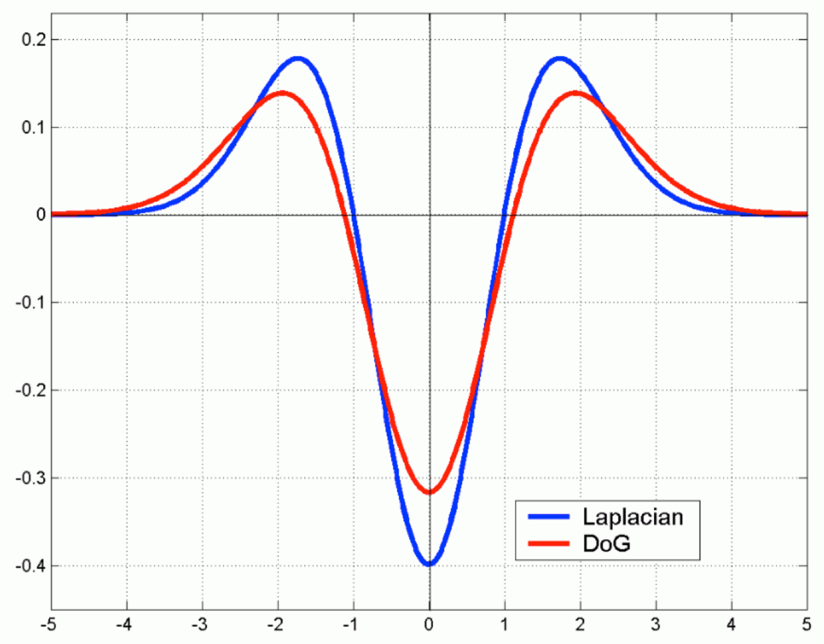
(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)

where Gaussian

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

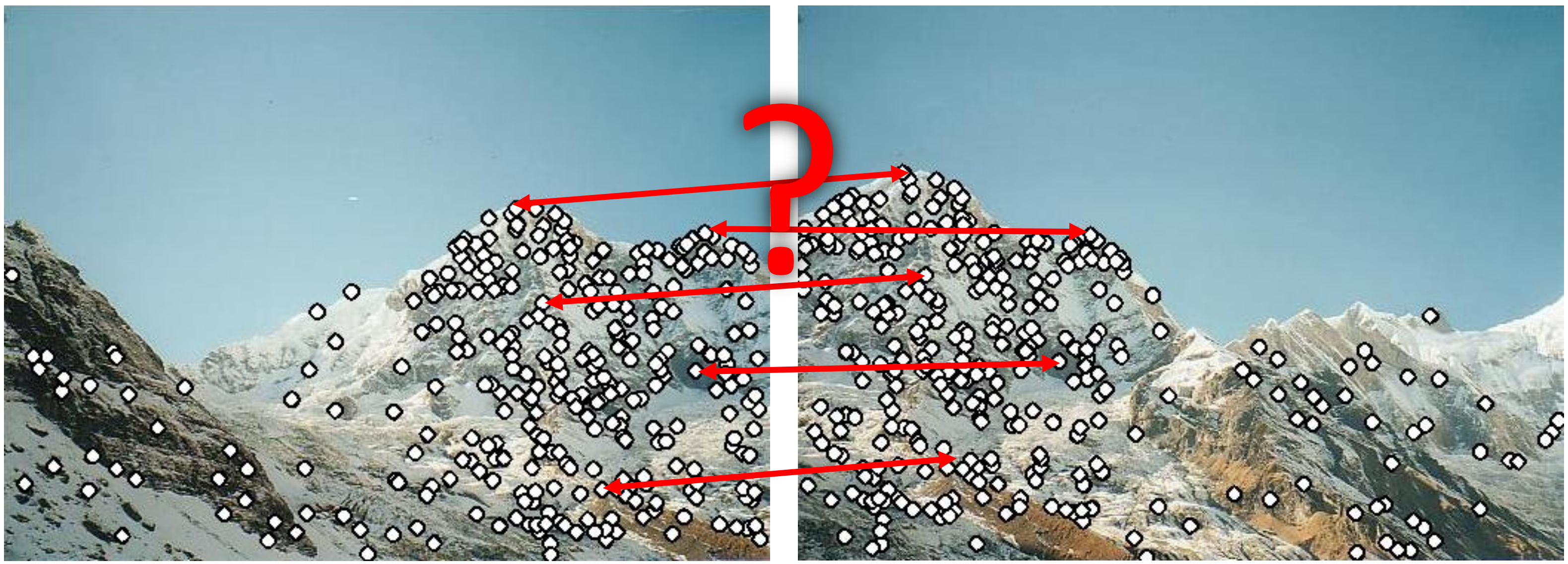



Note: both kernels are invariant to *scale* and *rotation*

# Feature descriptors

We know how to detect good points

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

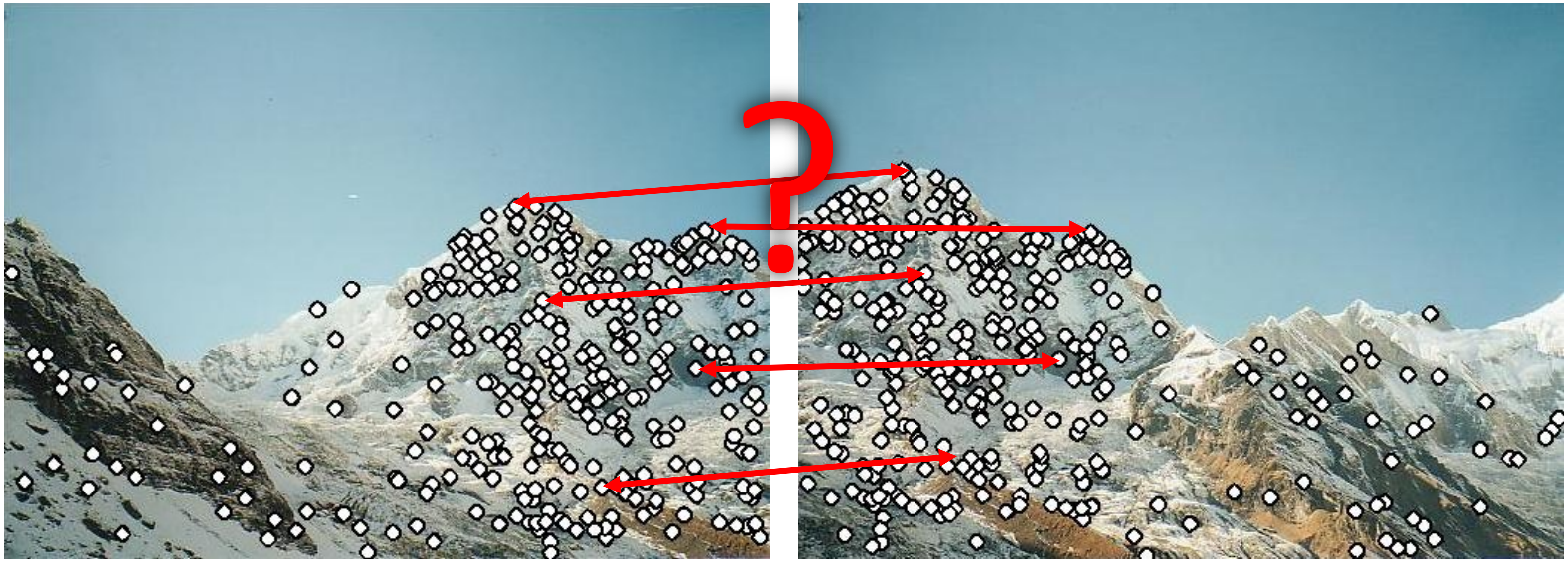



**Answer:** Come up with a *descriptor* for each point, find similar descriptors between the two images

# Feature descriptors

We know how to detect good points

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




Lots of possibilities (this is a popular research area)

- Simple option: match square windows around the point
- State of the art approach: SIFT
  - David Lowe, UBC http://www.cs.ubc.ca/~lowe/keypoints/

# Invariance vs. discriminability

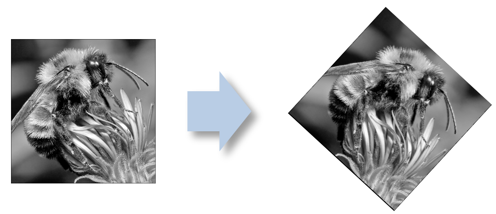
- Invariance:
  – Descriptor shouldn’t change even if image is transformed

- Discriminability:
  – Descriptor should be highly unique for each point

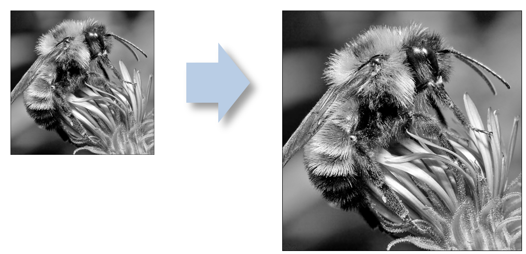
# Image transformations

- Geometric

**Rotation**

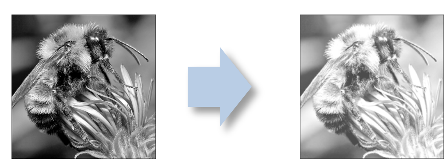


**Scale**



- Photometric

**Intensity change**



# Invariance

- Most feature descriptors are designed to be invariant to
  - Translation, 2D rotation, scale

- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes

# Rotation invariance for feature descriptors

- Find dominant orientation of the image patch
  - This is given by $\mathbf{x}_{\mathbf{max}}$, the eigenvector of **H** ($2^{nd}$ moment matrix) corresponding to $\lambda_{max}$ (the *larger* eigenvalue)
  - Rotate the patch according to this angle

Figure by Matthew Brown

# Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

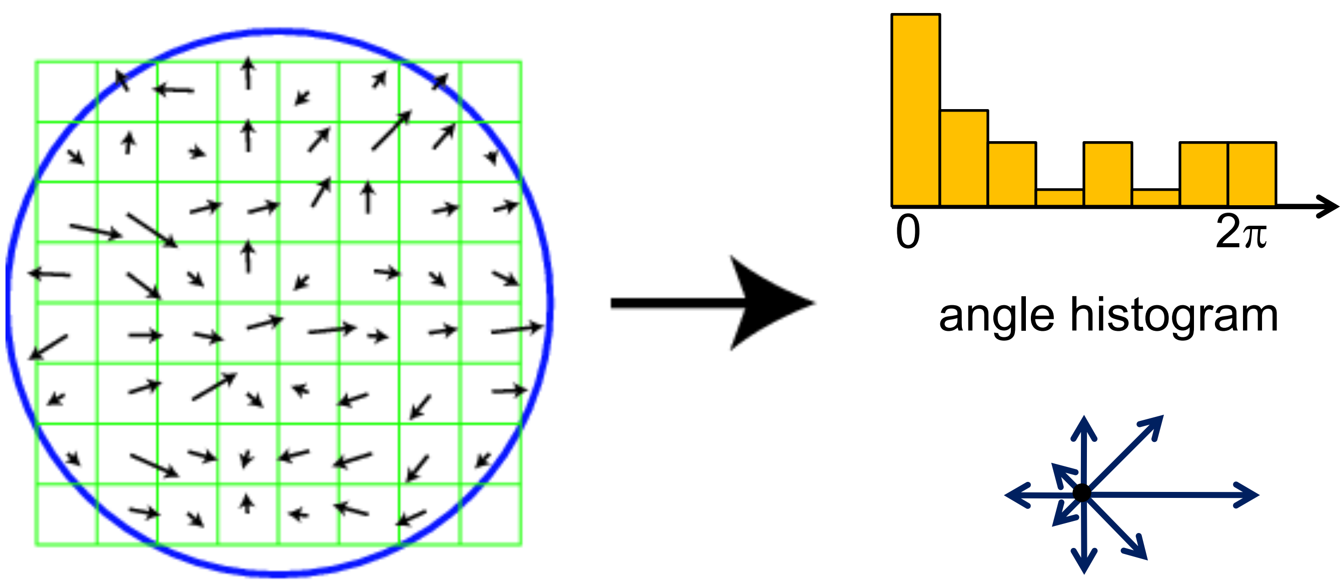


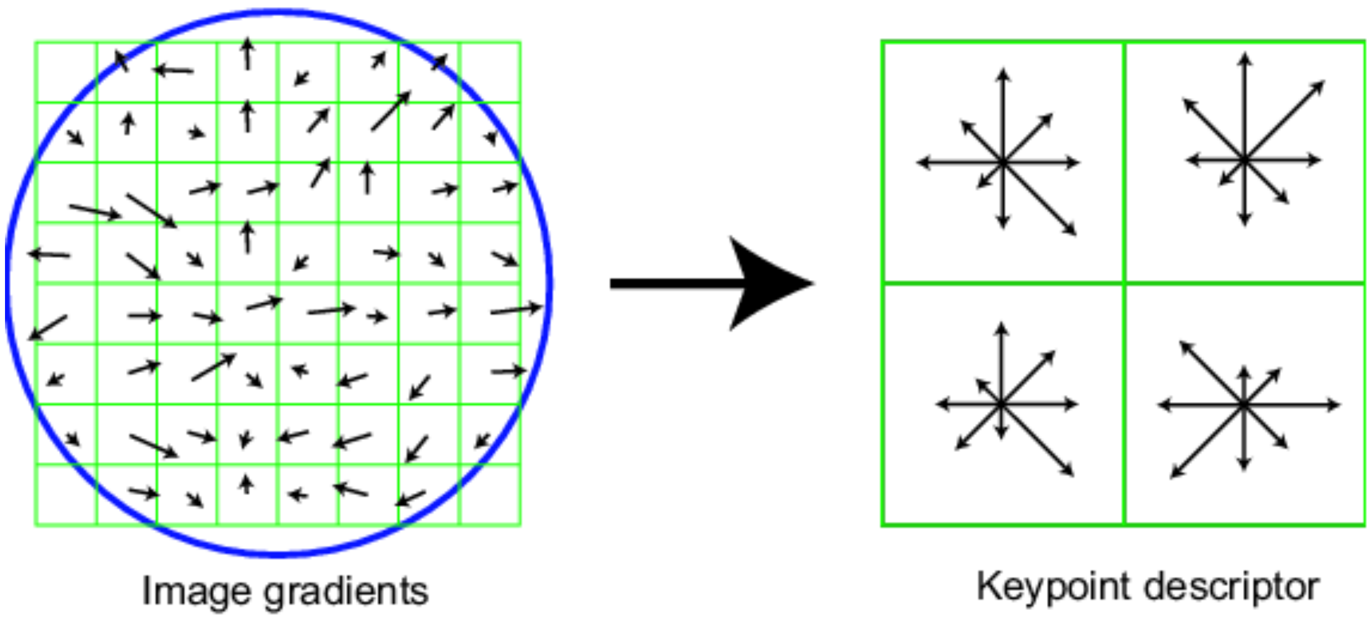

Image gradients

Adapted from slide by David Lowe

# SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor




Adapted from slide by David Lowe

# Properties of SIFT

Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
  - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

# Other descriptors

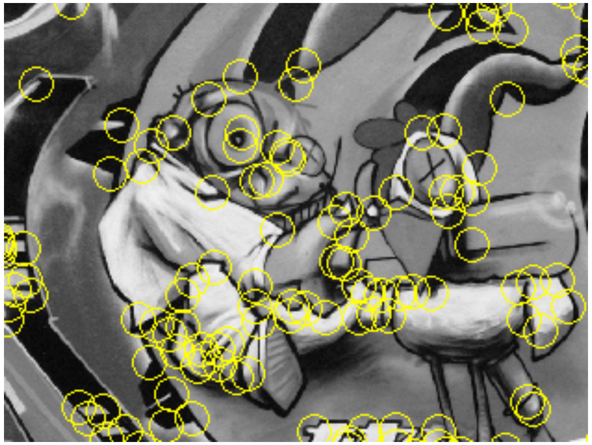
- HOG: Histogram of Gradients (HOG)
  - Dalal/Triggs
  - Sliding window, pedestrian detection

- FREAK: Fast Retina Keypoint
  - Perceptually motivated
- LIFT: Learned Invariant Feature Transform
  - Learned via deep learning

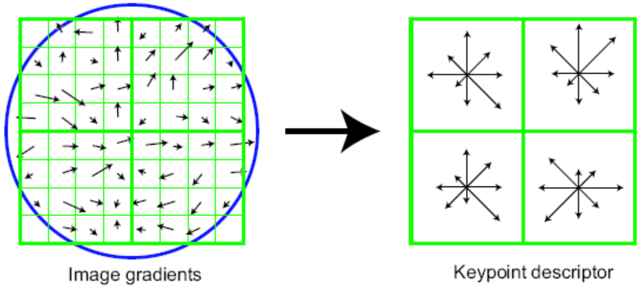
    https://arxiv.org/abs/1603.09114

# Summary

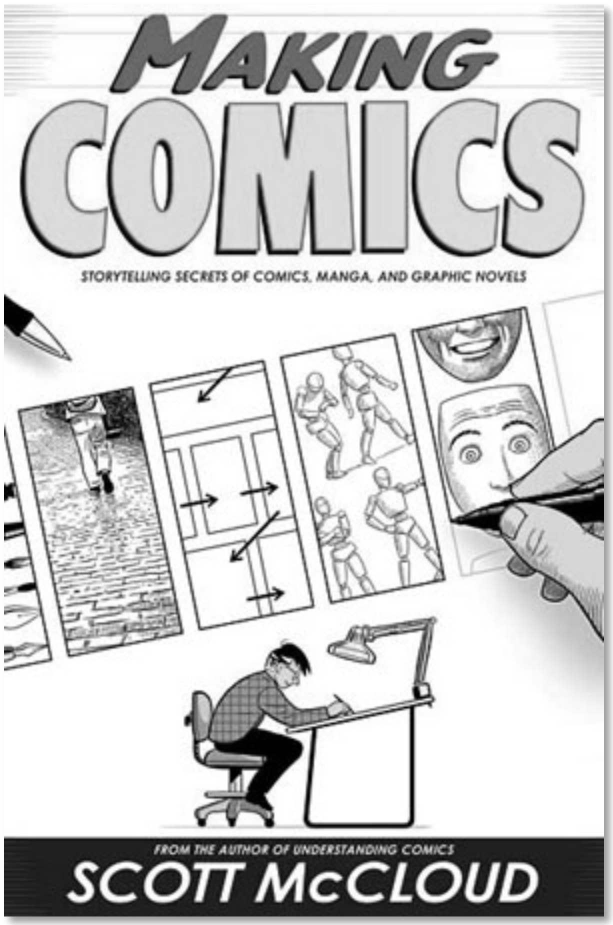
- Keypoint detection: repeatable and distinctive
  - Blobs via Difference-of-Gaussians



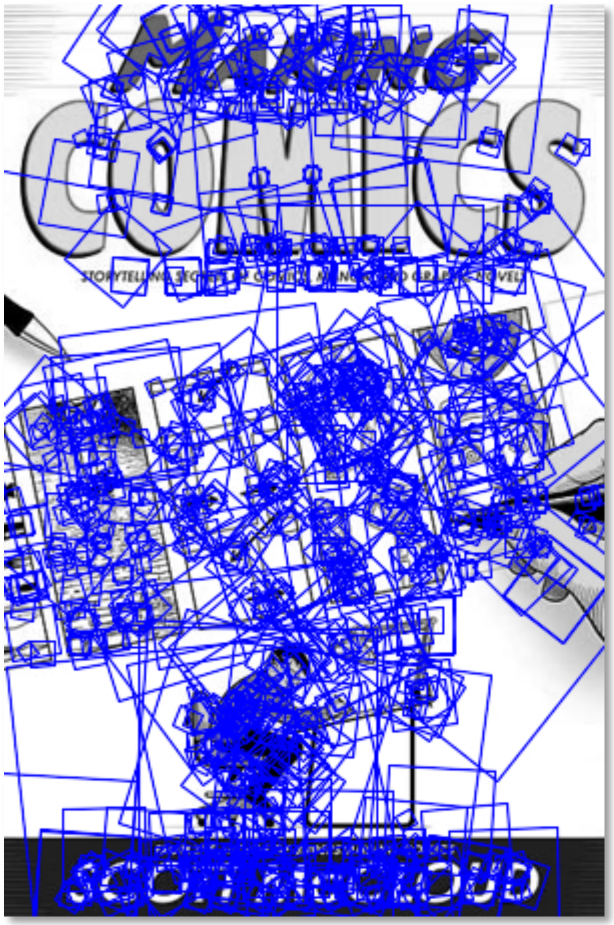
- Descriptors: robust and selective
  - spatial histograms of orientation
  - SIFT and variants are typically good for stitching and recognition
  - But, need not stick to one



# SIFT Example

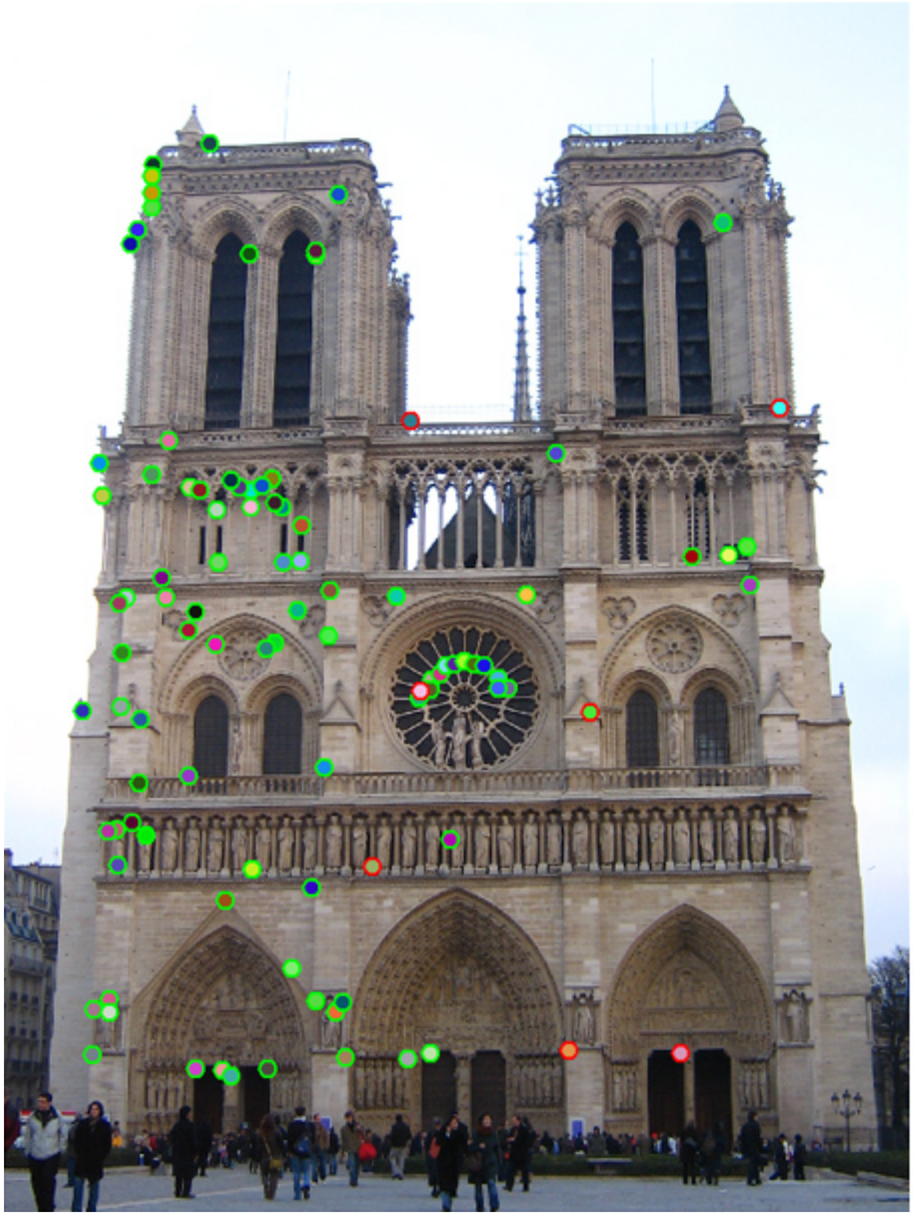


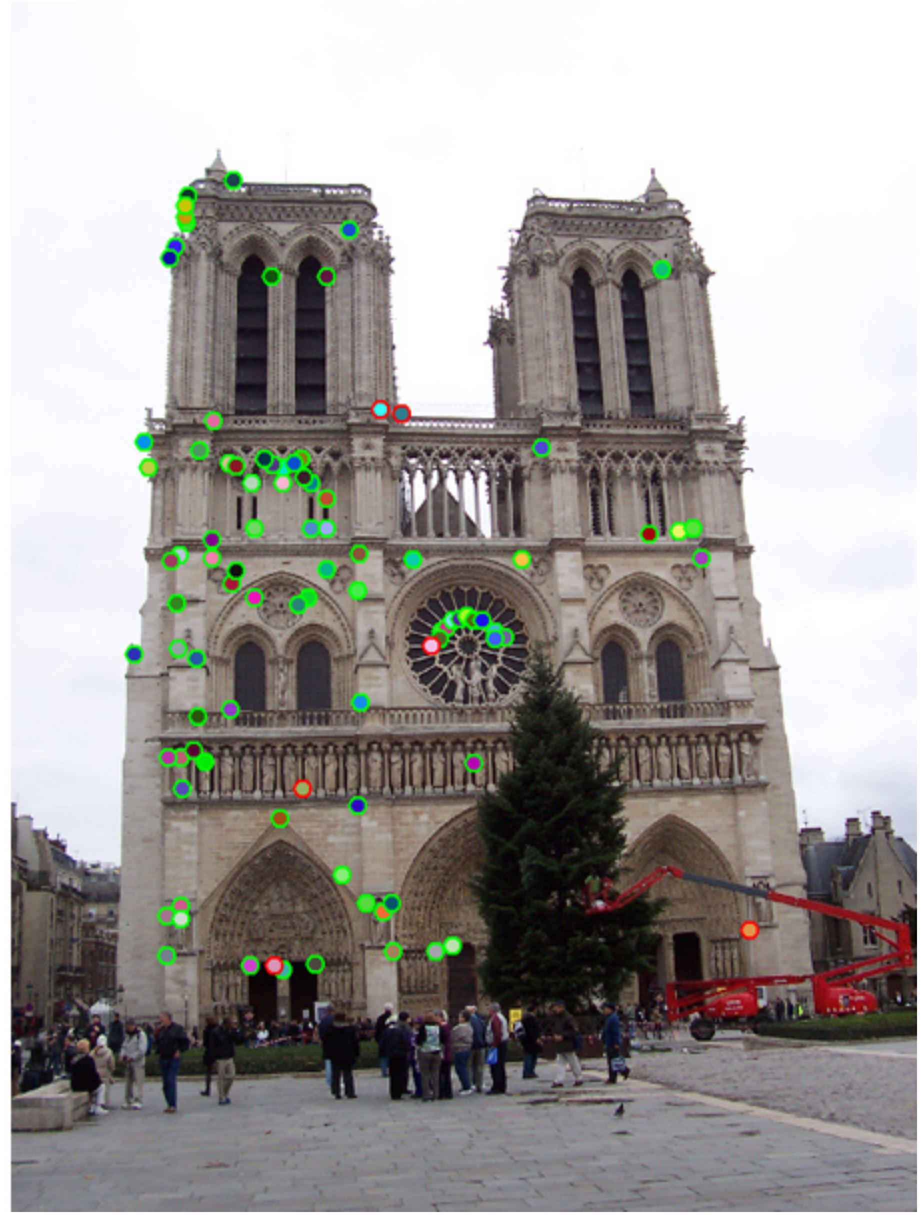

sift



**868 SIFT features**

# Which features match?





# Feature matching

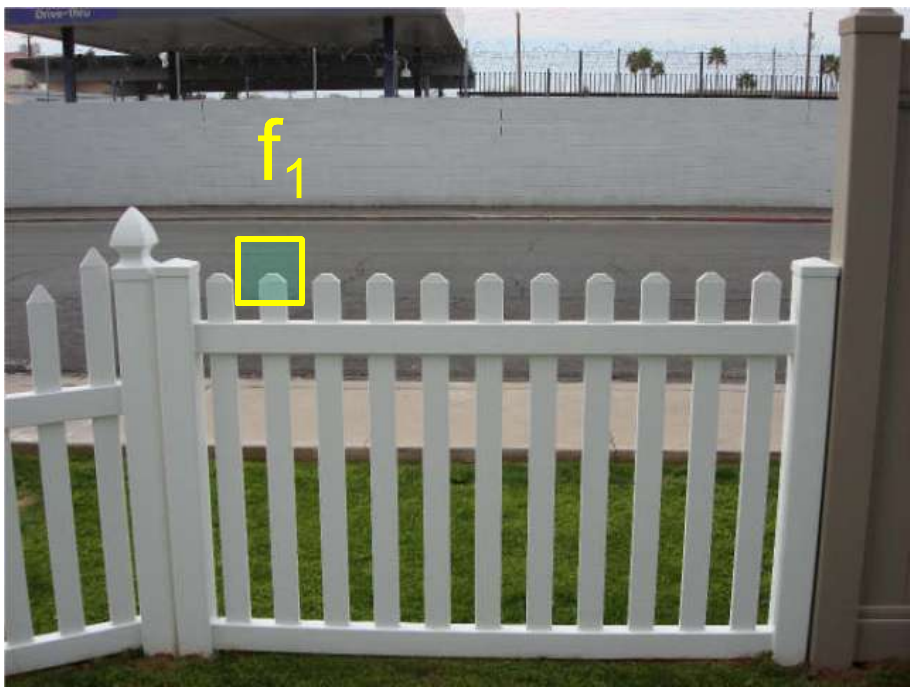
Given a feature in $I_1$, how to find the best match in $I_2$?

1. Define distance function that compares two descriptors
2. Test all the features in $I_2$, find the one with min distance

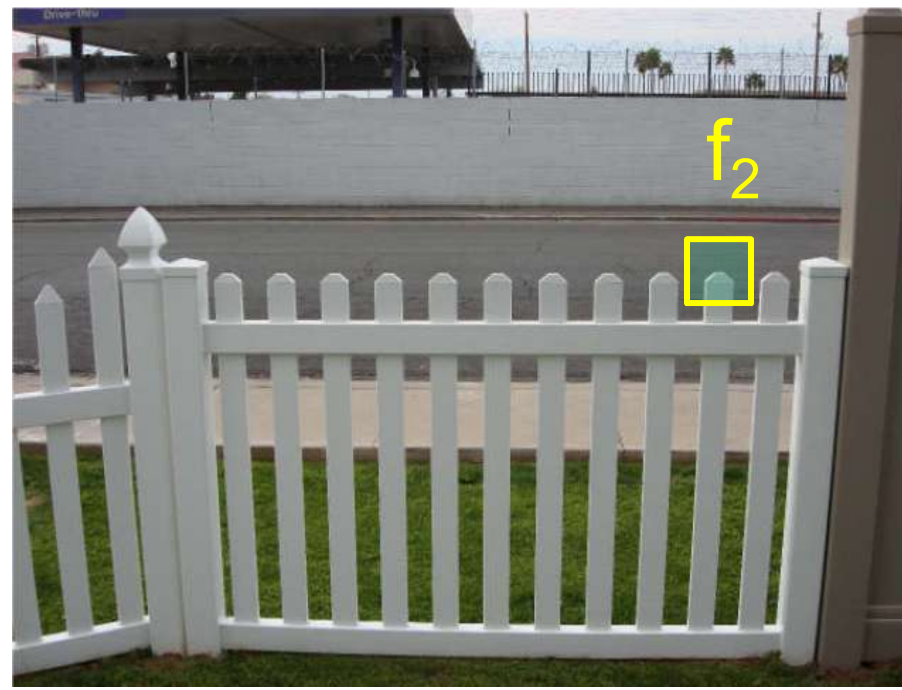
# Feature distance

How to define the difference between two features $f_1$, $f_2$?

- Simple approach: $L_2$ distance, $||f_1 - f_2||$ (aka SSD)
- can give good scores to ambiguous (incorrect) matches




$I_1$

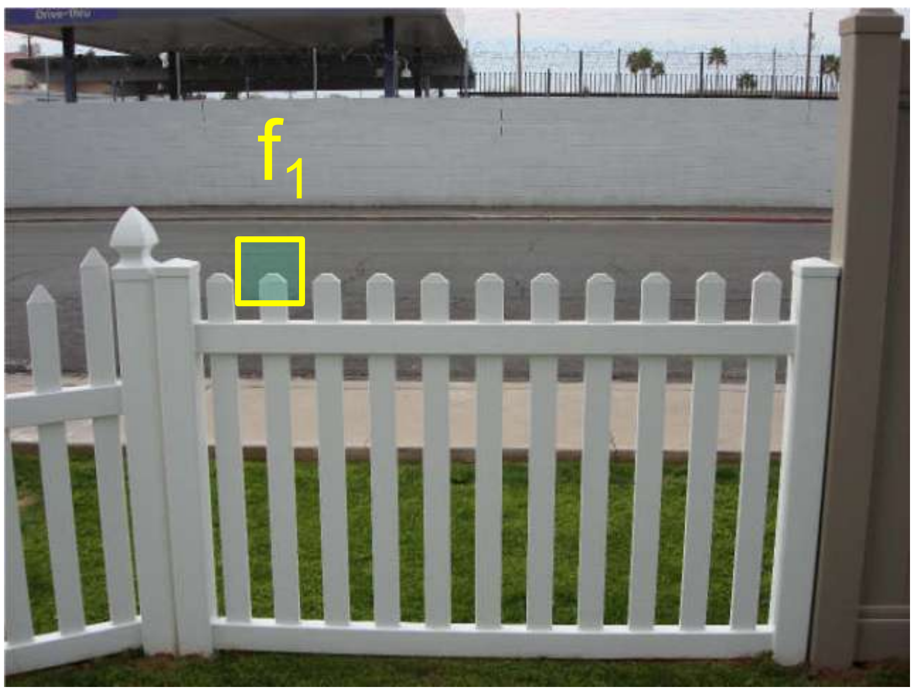



$I_2$

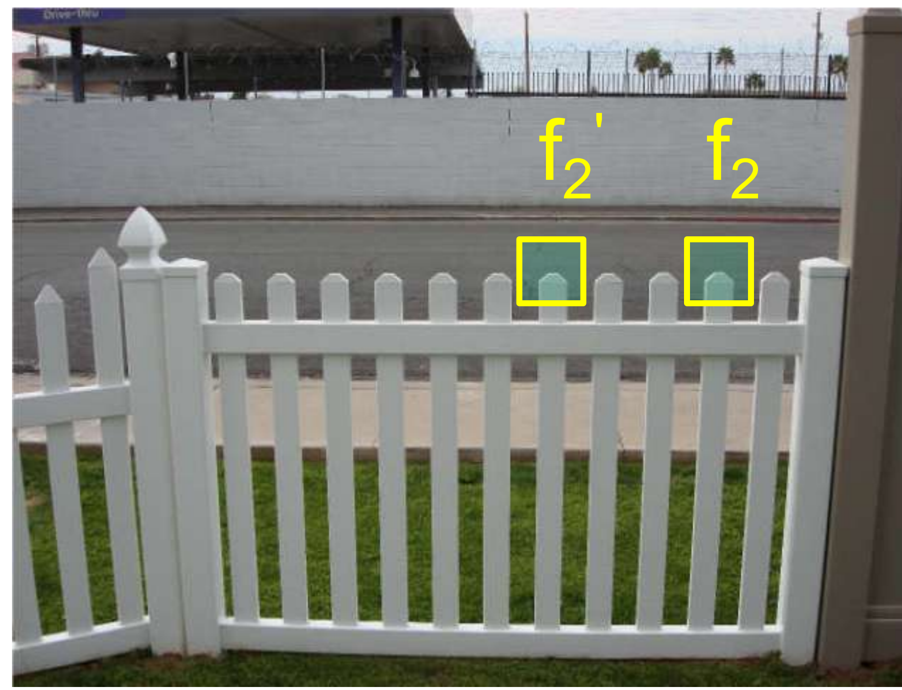
# Feature distance

## How to define the difference between two features $f_1, f_2$?

- Better approach: ratio distance = $||f_1 - f_2 || / || f_1 - f_2' ||$
  - $f_2$ is best SSD match to $f_1$ in $I_2$
  - $f_2'$ is $2^{nd}$ best SSD match to $f_1$ in $I_2$
  - gives large values for ambiguous matches




$I_1$

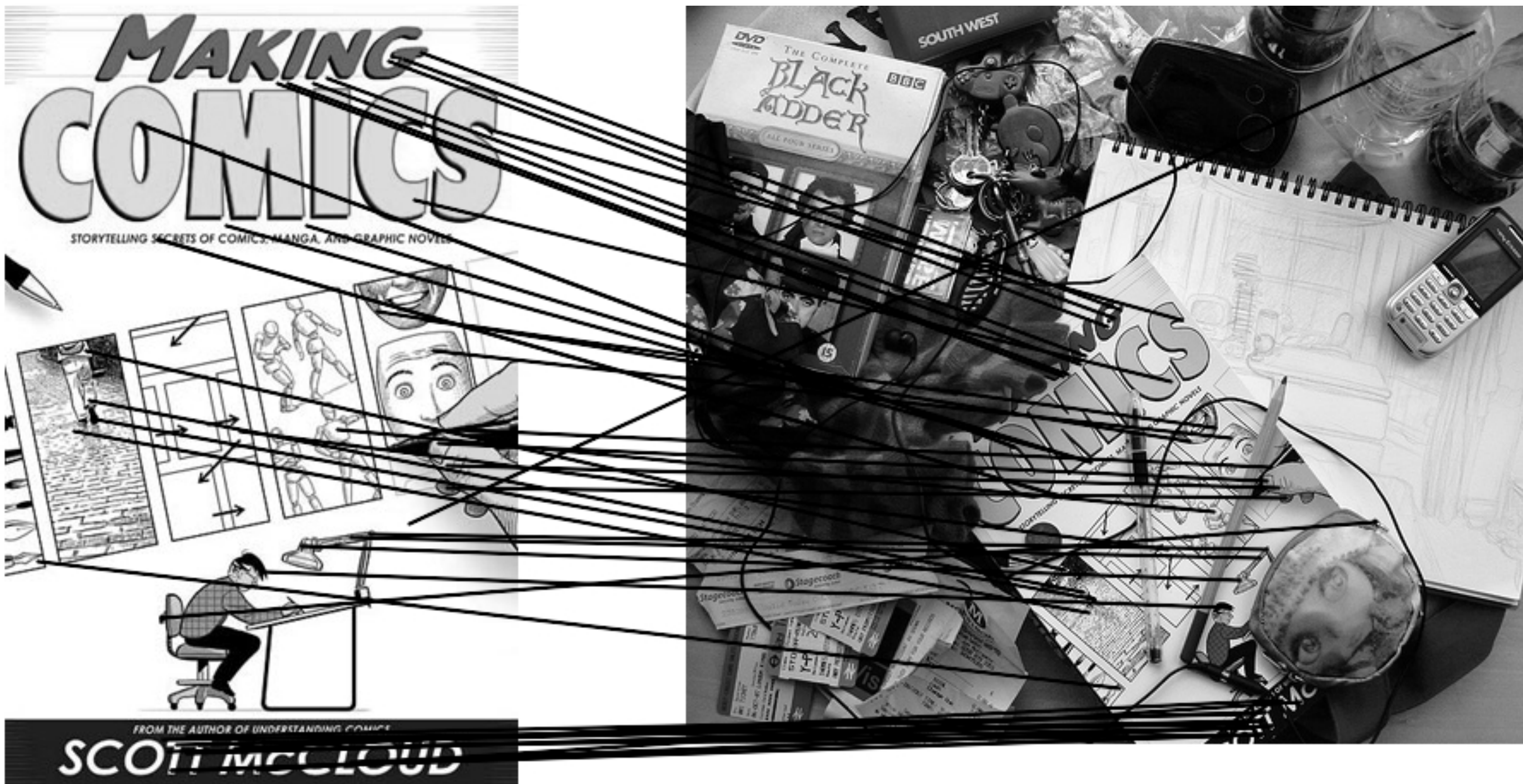



$I_2$

# Feature distance

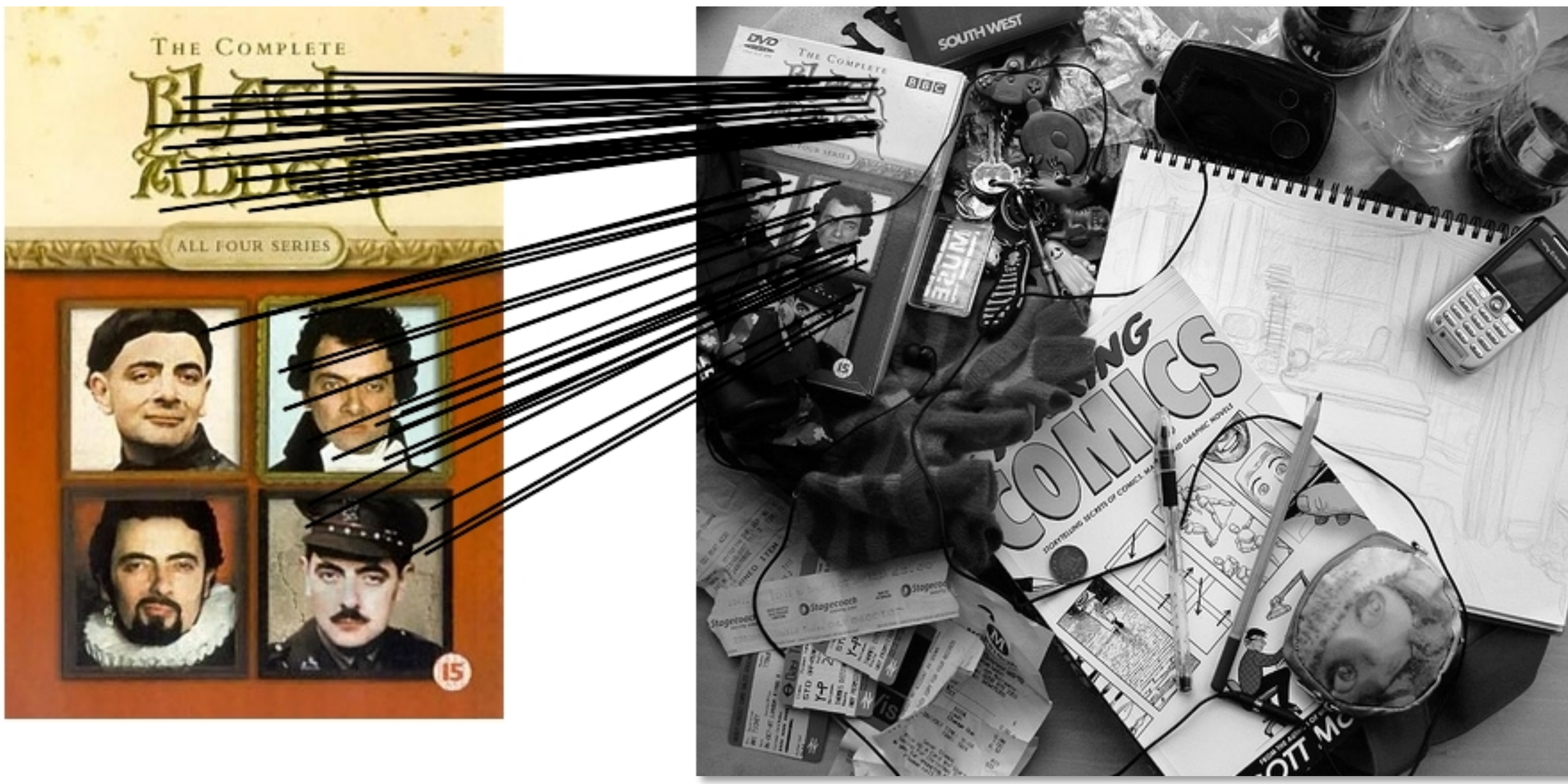
- Does the SSD vs “ratio distance” change the best match to a given feature in image 1?

# Feature matching example




**51 matches (thresholded by ratio score)**

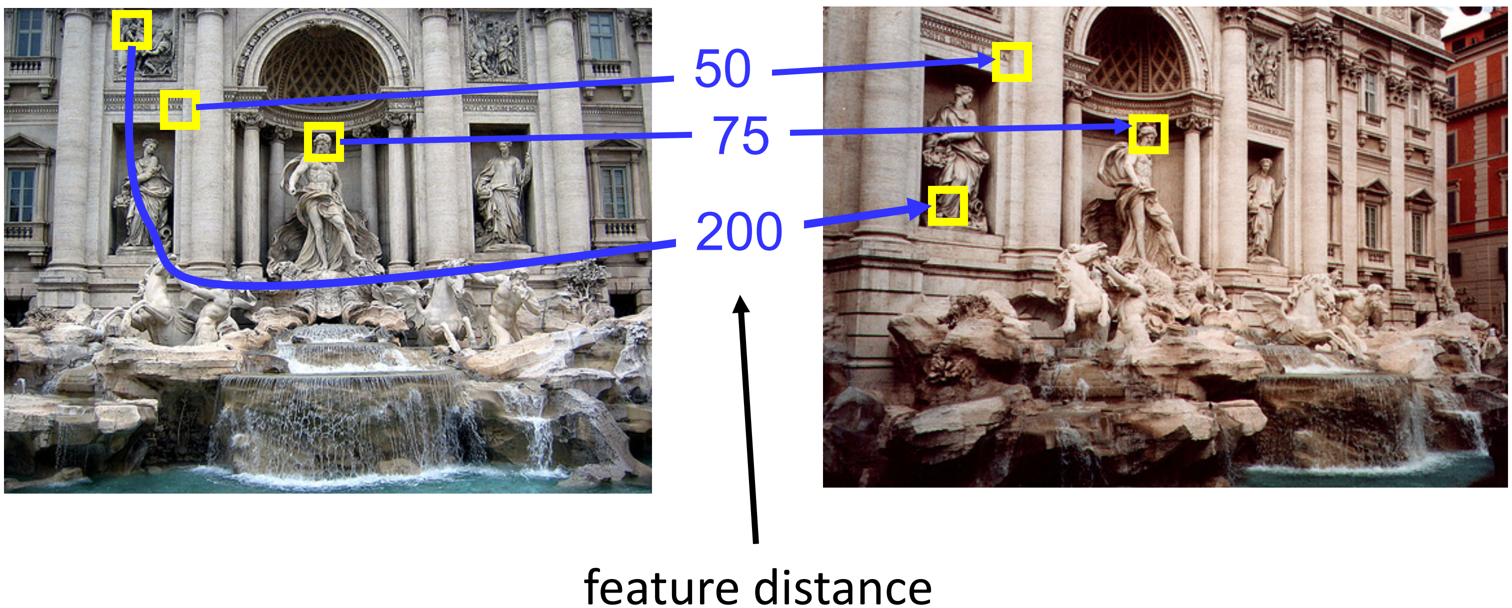
# Feature matching example




**58 matches (thresholded by ratio score)**

# Evaluating the results

How can we measure the performance of a feature matcher?



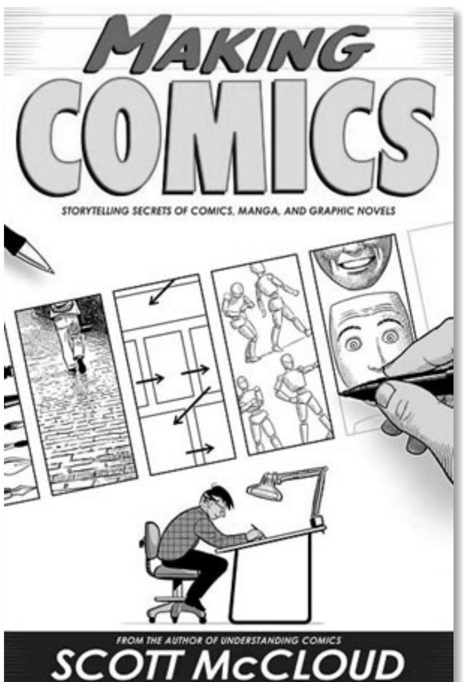
# Available at a web site near you…

- For most local feature detectors, executables are available online:
  - http://www.robots.ox.ac.uk/~vgg/research/affine
  - http://www.cs.ubc.ca/~lowe/keypoints/
  - http://www.vision.ee.ethz.ch/~surf

# Image alignment

Why don’t these image line up exactly?

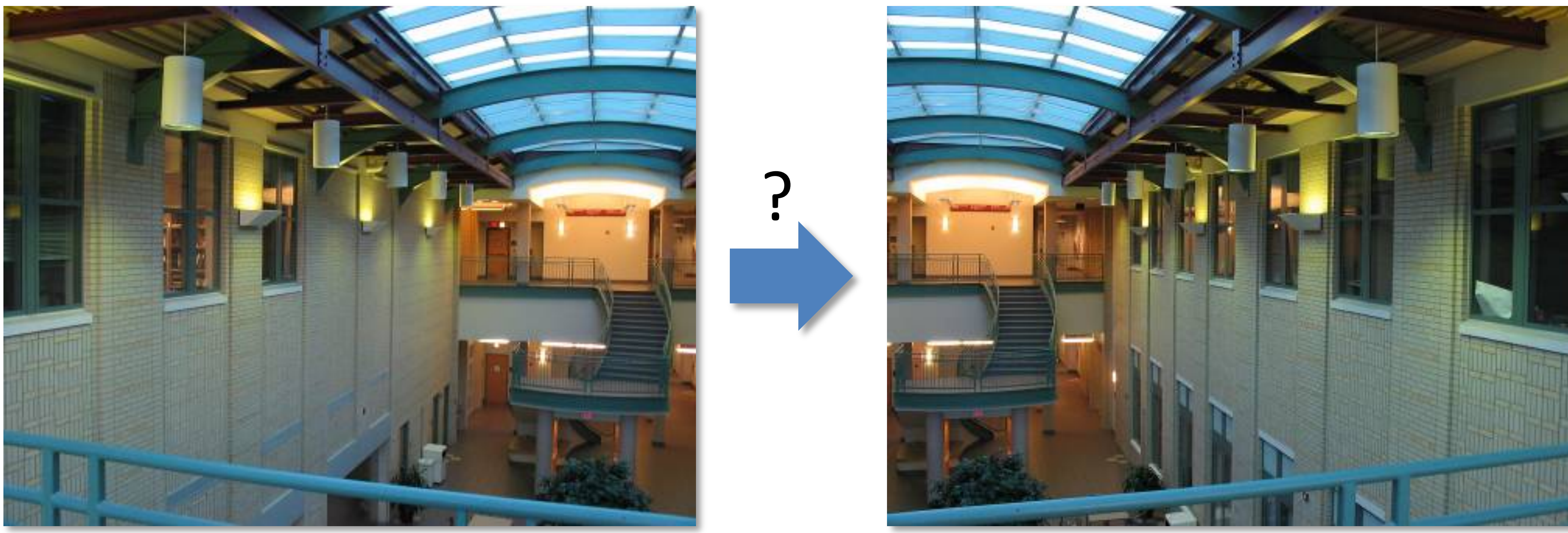
# What is the geometric relationship between these two images?




?


**Answer: Similarity transformation** (translation, rotation, uniform scale)

# What is the geometric relationship between these two images?



# What is the geometric relationship between these two images?

**Very important for creating mosaics!**

# Parametric (global) warping

- Examples of parametric warps:

translation

rotation

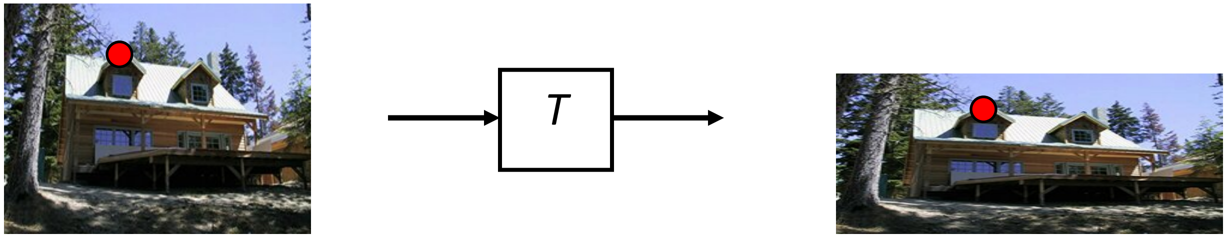
aspect

affine

perspective

cylindrical

# Parametric (global) warping




**p** = (x,y) **p’** = (x’,y’)

- Transformation T is a coordinate-changing machine:

  p’ = *T*(p)

- What does it mean that *T* is global?
  - Is the same for any point p
  - can be described by just a few numbers (parameters)
- Let’s consider *linear* forms (can be represented by a 2D matrix):

$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

# Common linear transformations

- Uniform scaling by *s*:

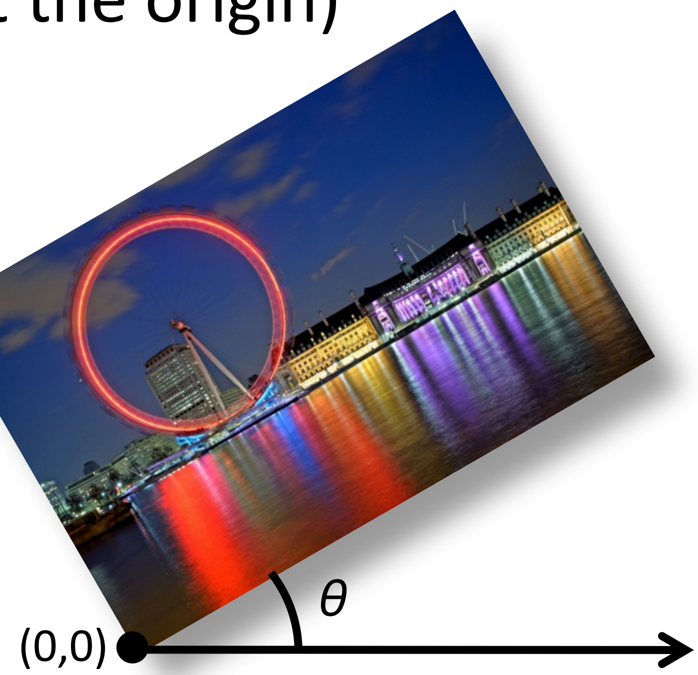


$$\mathbf{S} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

What is the inverse?

# Common linear transformations

- Rotation by angle $\theta$ (about the origin)


$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

What is the inverse?

For rotations:

$$\mathbf{R}^{-1} = \mathbf{R}^T$$

# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D mirror about Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

2D mirror across line y = x?

$$\begin{aligned} x' &= y \\ y' &= x \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$
\begin{aligned}
x' &= x + t_x \\
y' &= y + t_y
\end{aligned}
$$

NO!

Translation is not a linear operation on 2D coordinates

# All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

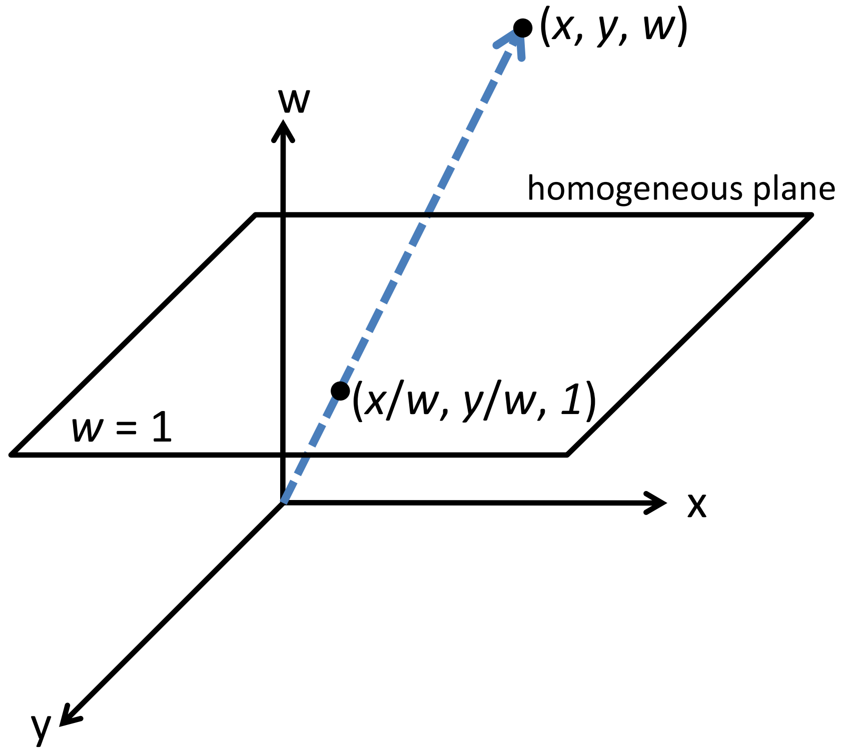
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

# Homogeneous coordinates

Trick: add one more coordinate:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates




Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

# Translation

- Solution: homogeneous coordinates to the rescue

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

# Affine transformations

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

any transformation with last row [ 0 0 1 ] we call an *affine* transformation

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

# Basic affine transformations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s_x} & 0 & 0 \\ 0 & \boldsymbol{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D *in-plane* rotation

$$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh_x} & 0 \\ \boldsymbol{sh_y} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

# Affine Transformations

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

# Is this an affine transformation?

# Where do we go from here?

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

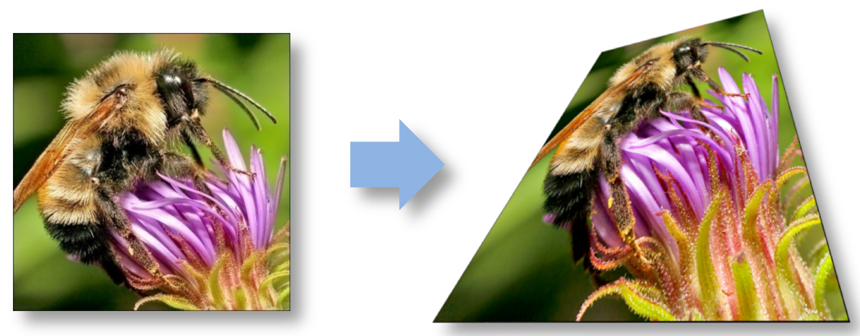
what happens when we mess with this row?

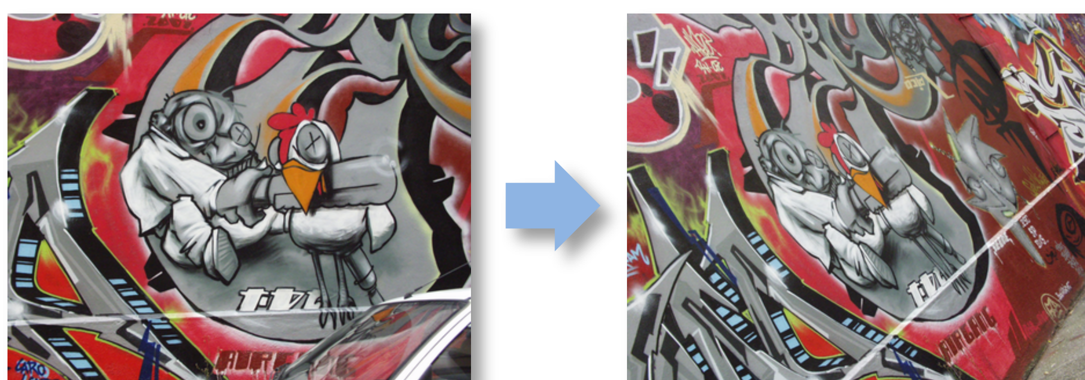
affine transformation

# Projective Transformations aka Homographies aka Planar Perspective Maps

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

Called a *homography*
(or *planar perspective map*)





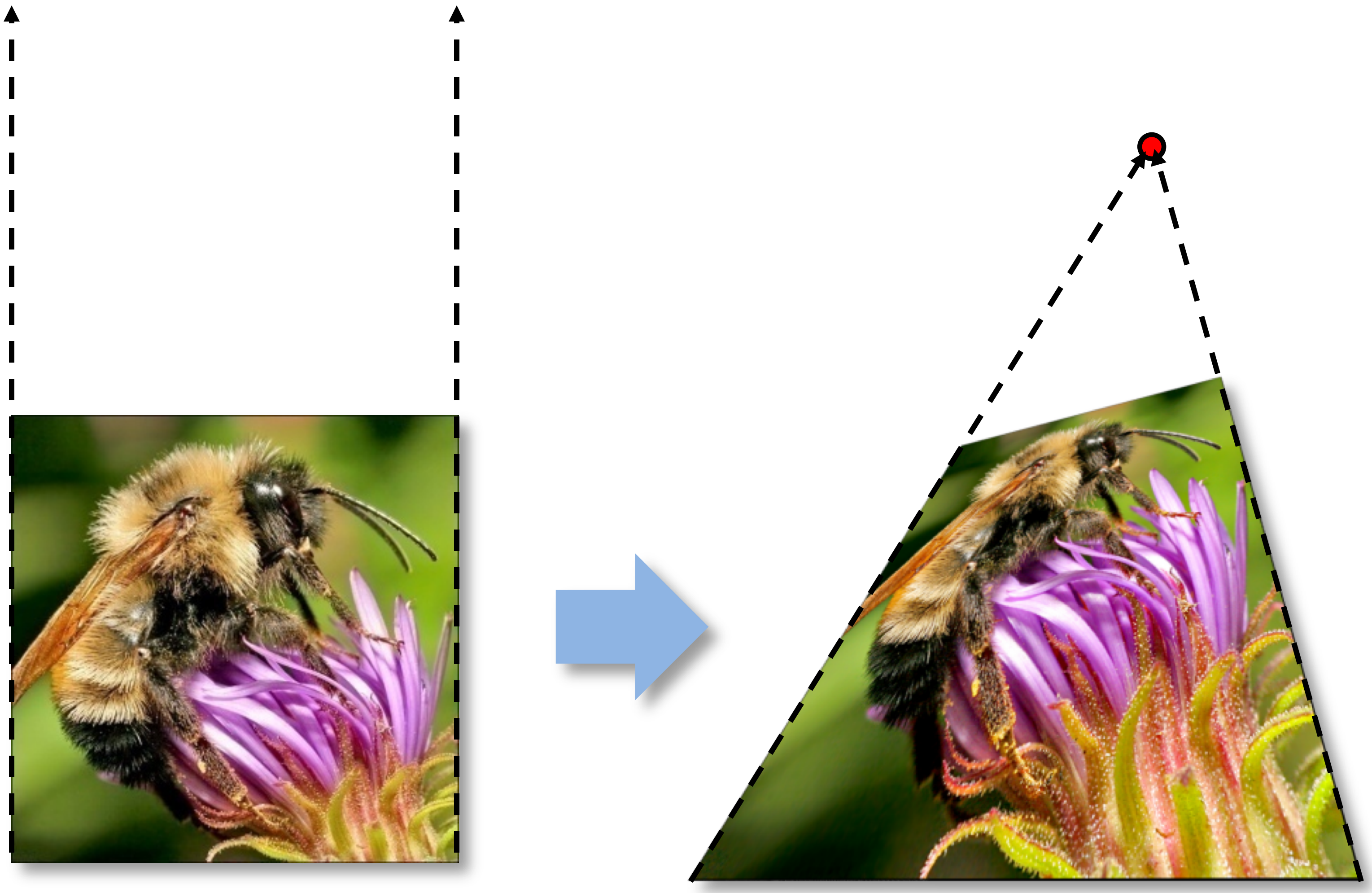
# Homographies

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

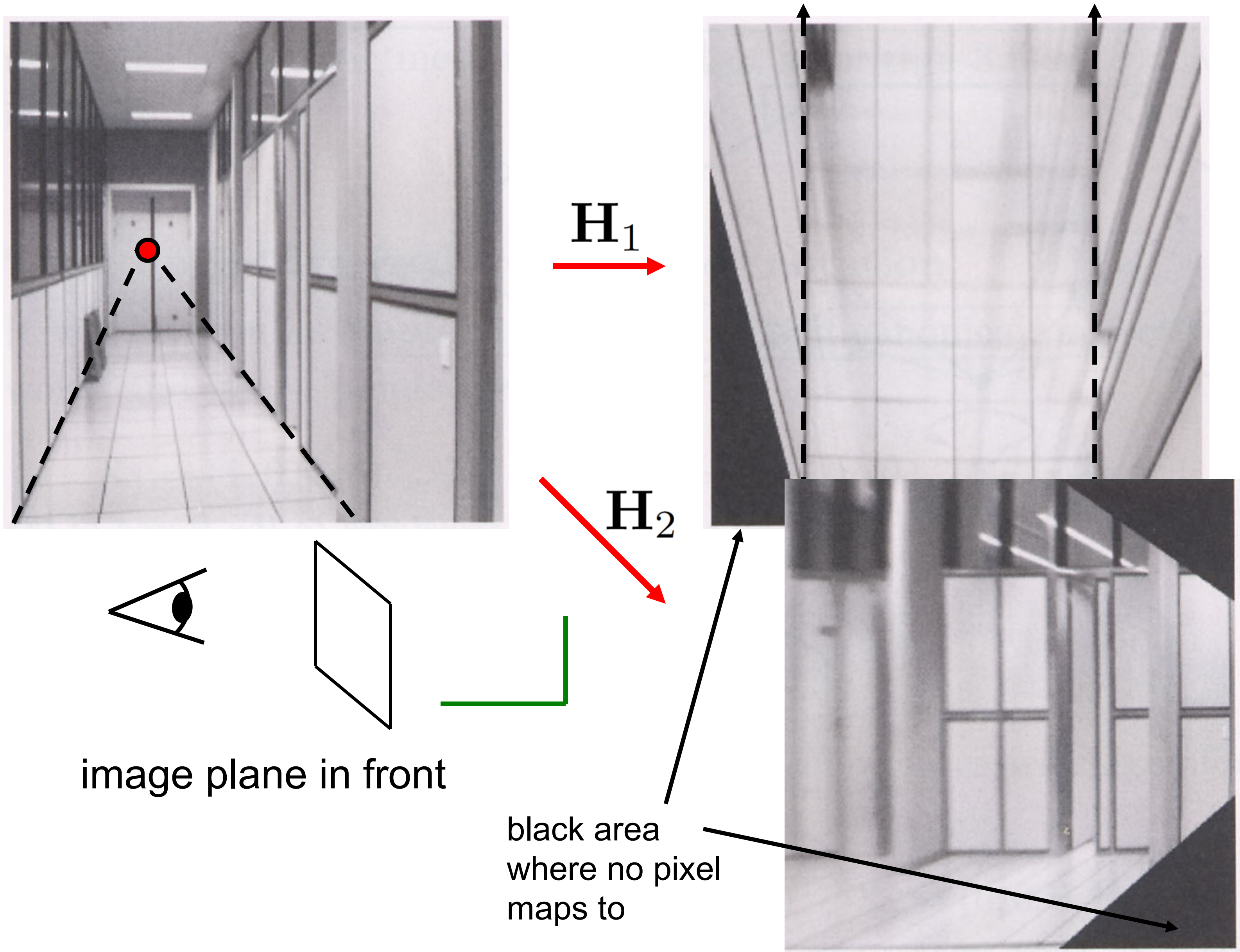
**What happens when the denominator is 0?**

$$\sim \begin{bmatrix} \frac{ax+by+c}{gx+hy+1} \\ \frac{dx+ey+f}{gx+hy+1} \\ 1 \end{bmatrix}$$

# Points at infinity



# Image warping with homographies



# Homographies

# Homographies

- Homographies …
  - Affine transformations, and
  - Projective warps

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

- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
  - Closed under composition

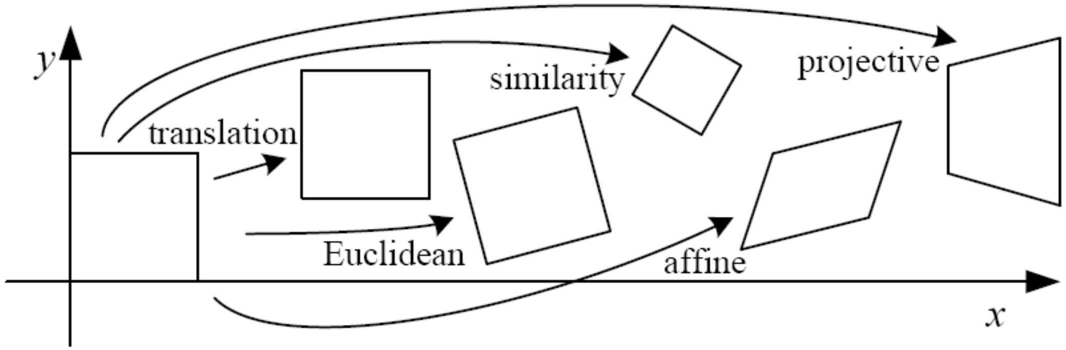
# Affine Transformations

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

# 2D image transformations

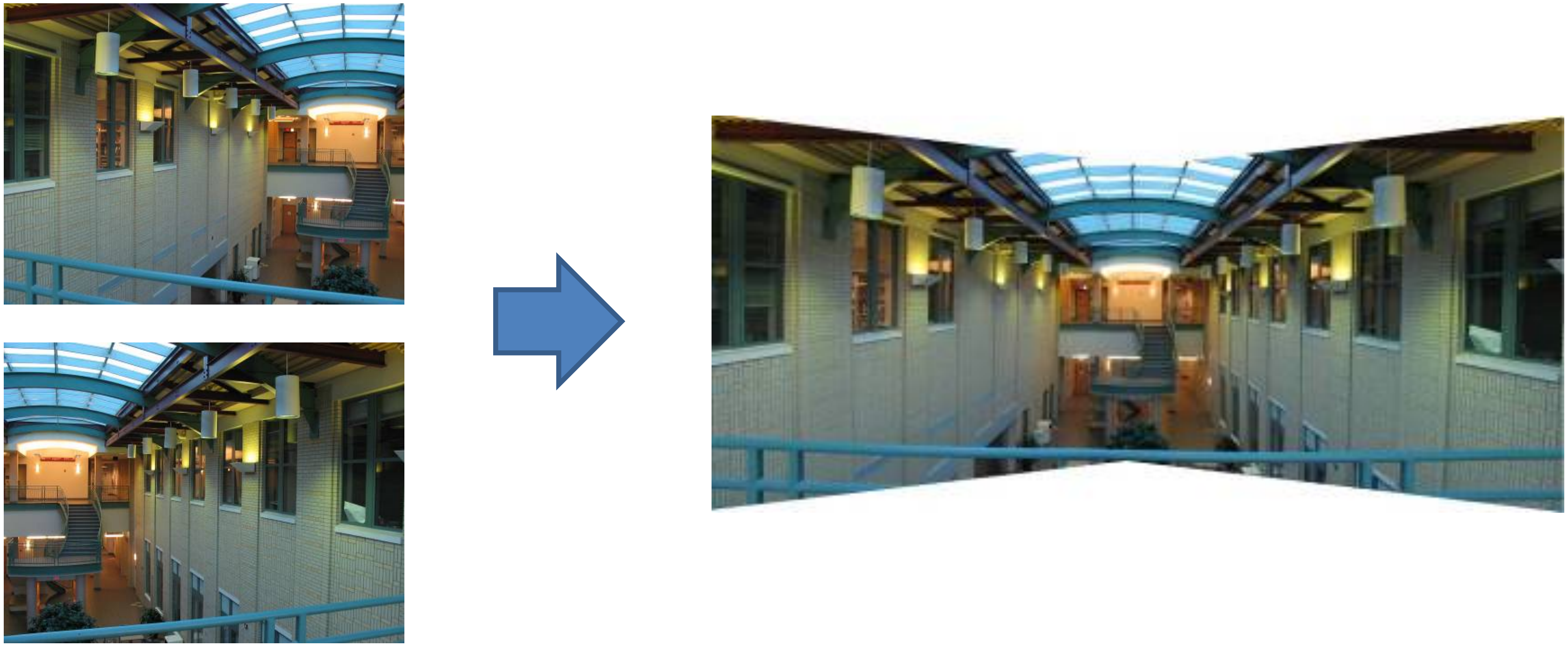



| Name | Matrix | # D.O.F. | Preserves: | Icon |
|---|---|---|---|---|
| translation | $\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$ | 2 | orientation $+ \cdots$ | |
| rigid (Euclidean) | $\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$ |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

These transformations are a nested set of groups

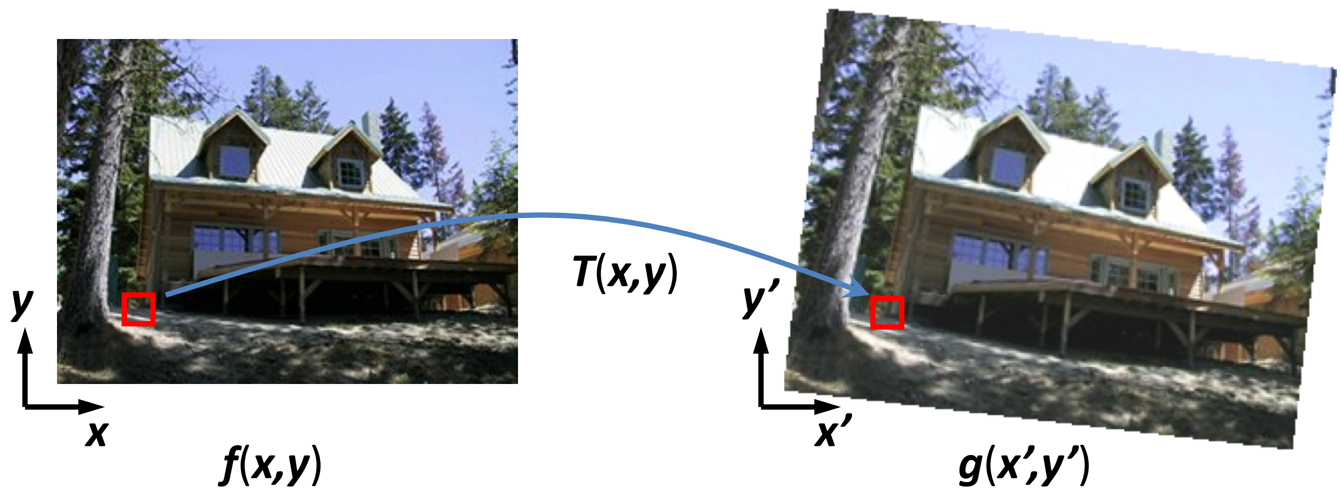
- Closed under composition and inverse is a member

# Homographies



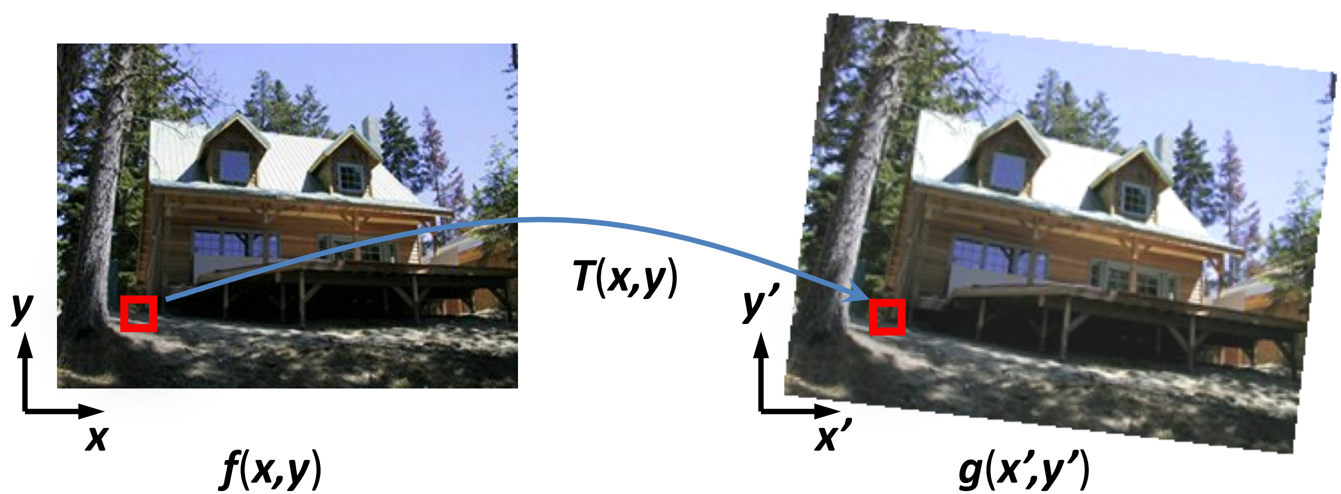
# Image Warping

- Given a coordinate xform $(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})$ and a source image $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$, how do we compute an xformed image $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{f}(\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}))$?



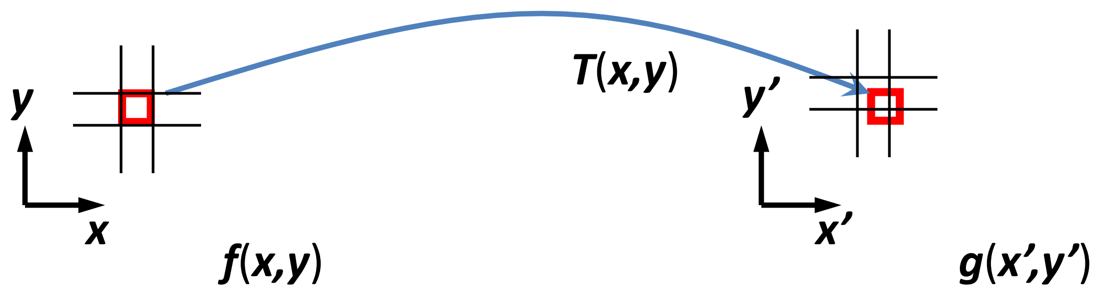
# Forward Warping

- Send each pixel ***f***(***x***) to its corresponding location (***x'***,***y'***) = ***T***(***x***,***y***) in ***g***(***x'***,***y'***)
  - What if pixel lands "between" two pixels?



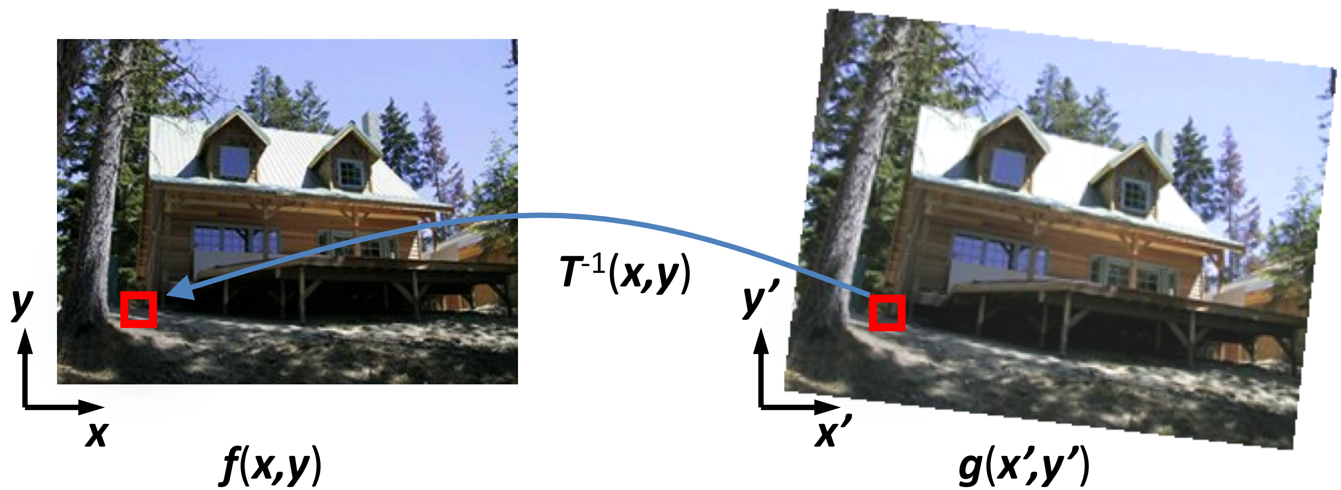
# Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$ to its corresponding location $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x},\boldsymbol{y})$ in $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'})$
  - What if pixel lands "between" two pixels?
  - Answer: add "contribution" to several pixels, normalize later (*splatting*)
  - Can still result in holes



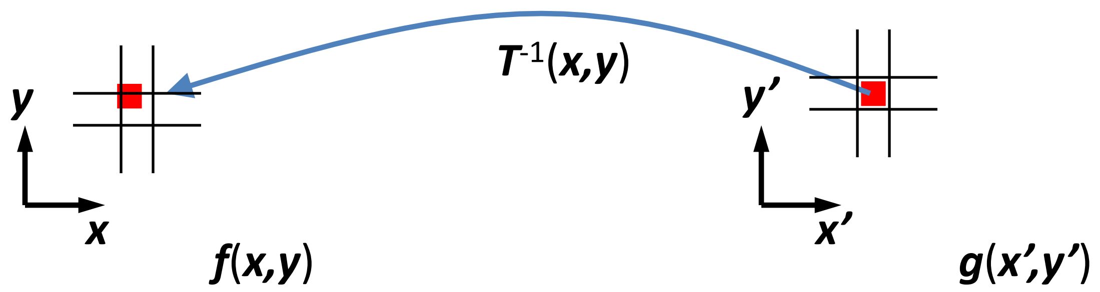
# Inverse Warping

- Get each pixel $\boldsymbol{g(x',y')}$ from its corresponding location $(\boldsymbol{x,y}) = \boldsymbol{T}^{-1}(\boldsymbol{x,y})$ in $\boldsymbol{f(x,y)}$
  - Requires taking the inverse of the transform
  - What if pixel comes from “between” two pixels?



# Inverse Warping

- Get each pixel ***g***(***x'***) from its corresponding location ***x'*** = ***h***(***x***) in ***f***(***x***)
  - What if pixel comes from "between" two pixels?
  - Answer: *resample* color value from *interpolated* (*prefiltered*) source image



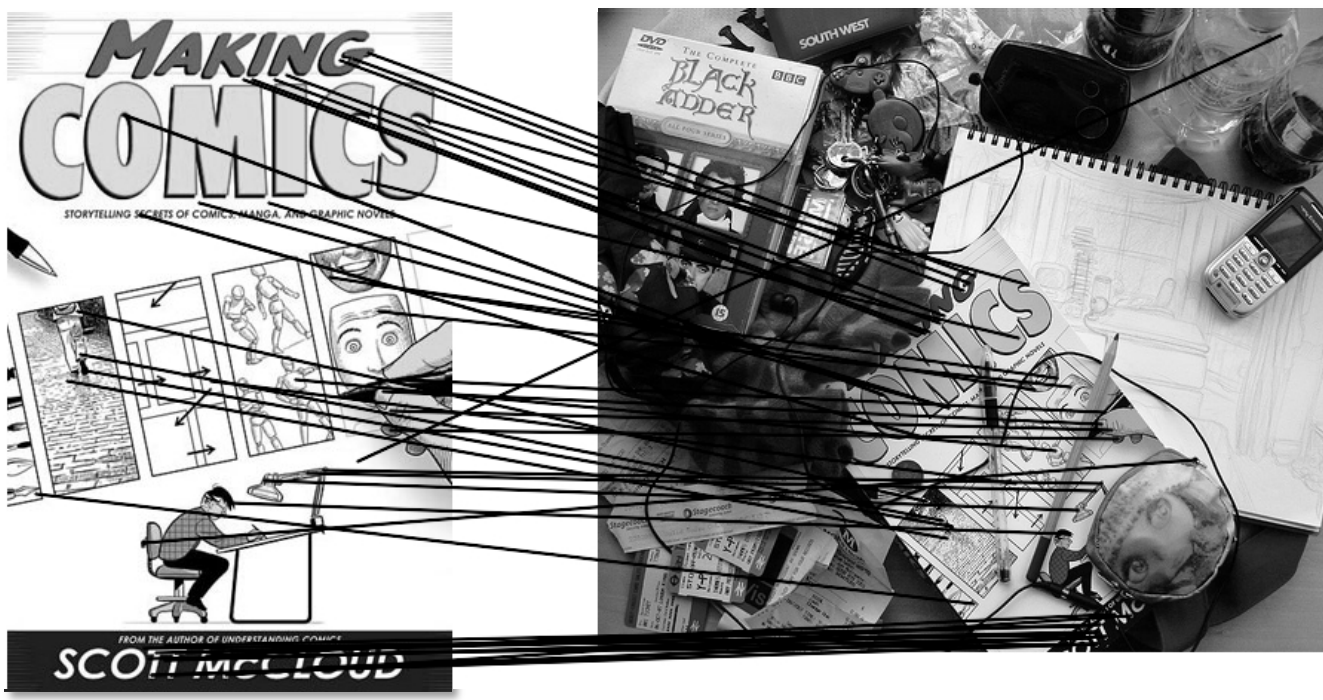
# Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)
  - sinc
- Needed to prevent "jaggies" and "texture crawl"

  (with prefiltering)

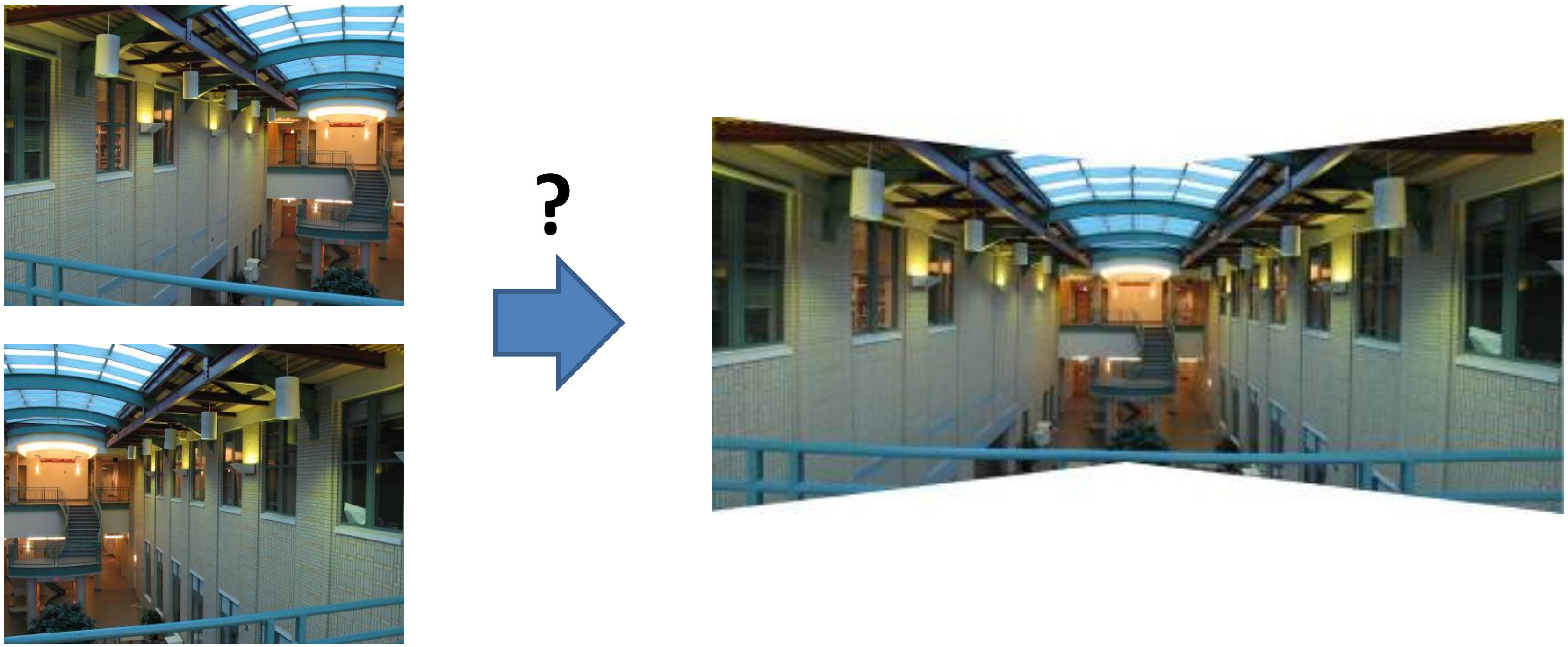
# Computing transformations

- Given a set of matches between images A and B
  - How can we compute the transform T from A to B?

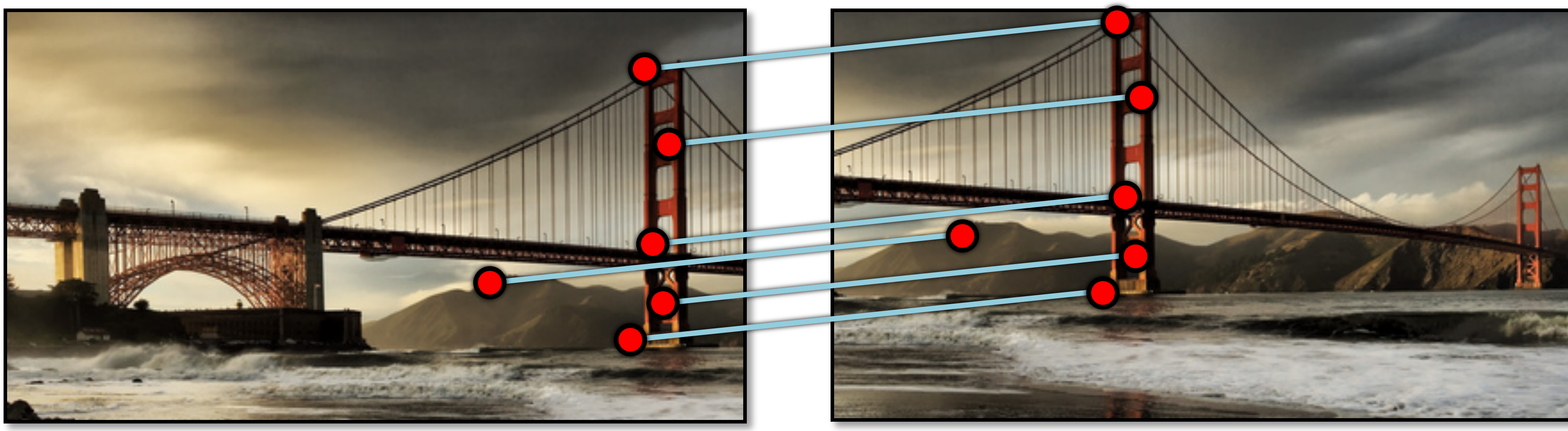



  - Find transform T that best “agrees” with the matches

# Computing transformations



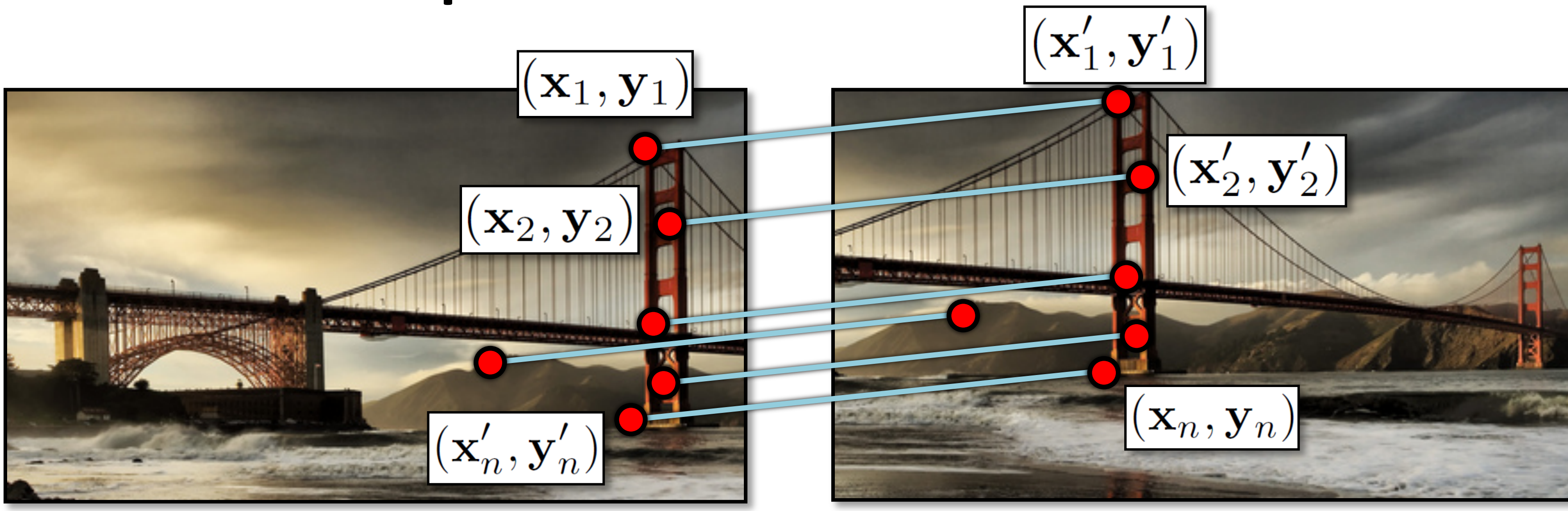
# Simple case: translations



$(\mathbf{x}_t, \mathbf{y}_t)$

**How do we solve for** $(\mathbf{x}_t, \mathbf{y}_t)$**?**

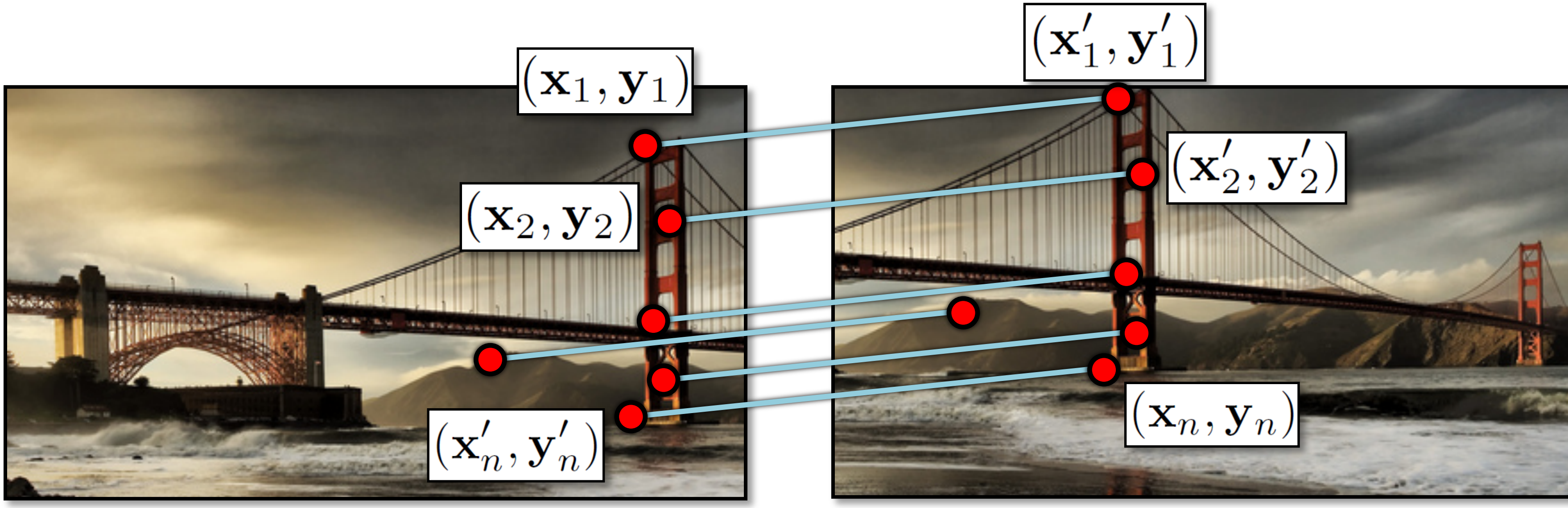
# Simple case: translations




Displacement of match *i* = $(\mathbf{x}'_i - \mathbf{x}_i, \mathbf{y}'_i - \mathbf{y}_i)$

$$(\mathbf{x}_t, \mathbf{y}_t) = \left( \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}'_i - \mathbf{x}_i, \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}'_i - \mathbf{y}_i \right)$$

# Another view

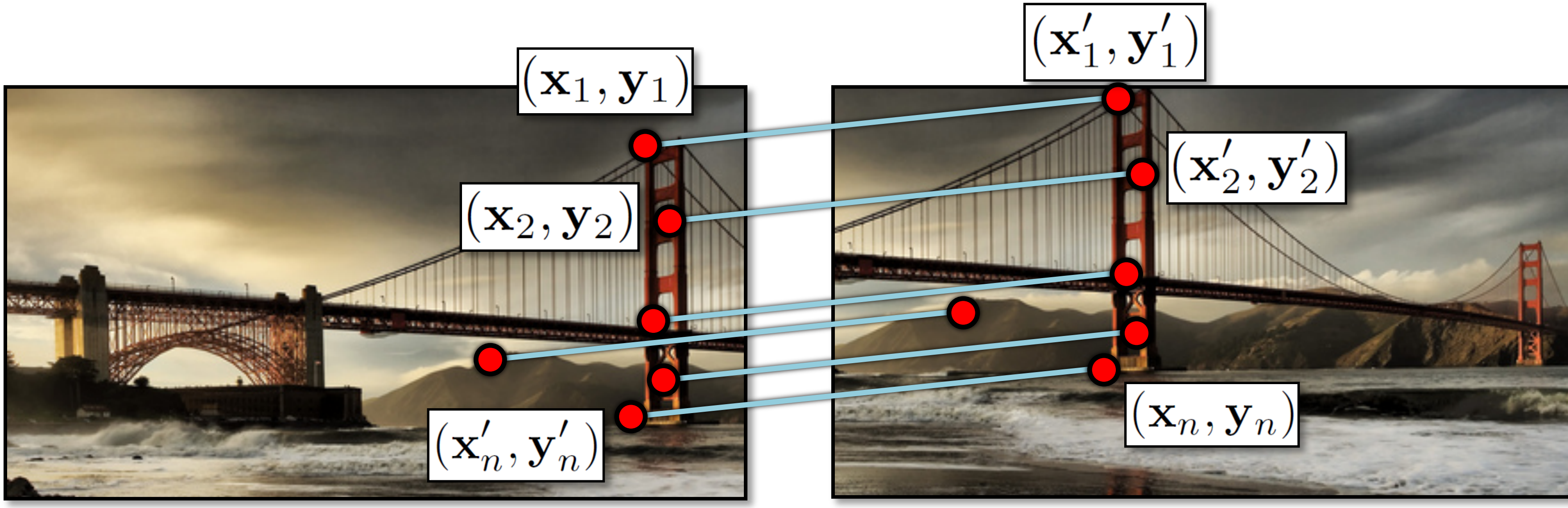



$$\mathbf{x}_i + \mathbf{x_t} \quad = \quad \mathbf{x}'_i$$

$$\mathbf{y}_i + \mathbf{y_t} \quad = \quad \mathbf{y}'_i$$

- System of linear equations
  - What are the knowns? Unknowns?
  - How many unknowns? How many equations (per match)?

# Another view




$$
\begin{aligned}
\mathbf{x}_i + \mathbf{x_t} &= \mathbf{x}'_i \\
\mathbf{y}_i + \mathbf{y_t} &= \mathbf{y}'_i
\end{aligned}
$$

- Problem: more equations than unknowns
  - “Overdetermined” system of equations
  - We will find the *least squares* solution

# Least squares formulation

- For each point $(\mathbf{x}_i, \mathbf{y}_i)$

$$
\begin{aligned}
\mathbf{x}_i + \mathbf{x_t} &= \mathbf{x}_i' \\
\mathbf{y}_i + \mathbf{y_t} &= \mathbf{y}_i'
\end{aligned}
$$

- we define the *residuals* as

$$
\begin{aligned}
r_{\mathbf{x}_i}(\mathbf{x}_t) &= (\mathbf{x}_i + \mathbf{x}_t) - \mathbf{x}_i' \\
r_{\mathbf{y}_i}(\mathbf{y}_t) &= (\mathbf{y}_i + \mathbf{y}_t) - \mathbf{y}_i'
\end{aligned}
$$

# Least squares formulation

- Goal: minimize sum of squared residuals

$$C(\mathbf{x}_t, \mathbf{y}_t) = \sum_{i=1}^{n} \left( r_{\mathbf{x}_i}(\mathbf{x}_t)^2 + r_{\mathbf{y}_i}(\mathbf{y}_t)^2 \right)$$

- "Least squares" solution
- For translations, is equal to mean (average) displacement

# Least squares formulation

- Can also write as a matrix equation

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x'_1 - x_1 \\ y'_1 - y_1 \\ x'_2 - x_2 \\ y'_2 - y_2 \\ \vdots \\ x'_n - x_n \\ y'_n - y_n \end{bmatrix}$$

$$\underset{2n \times 2}{\mathbf{A}} \quad \underset{2 \times 1}{\mathbf{t}} = \underset{2n \times 1}{\mathbf{b}}$$

# Least squares

$$\mathbf{A}\mathbf{t} = \mathbf{b}$$

- Find **t** that minimizes

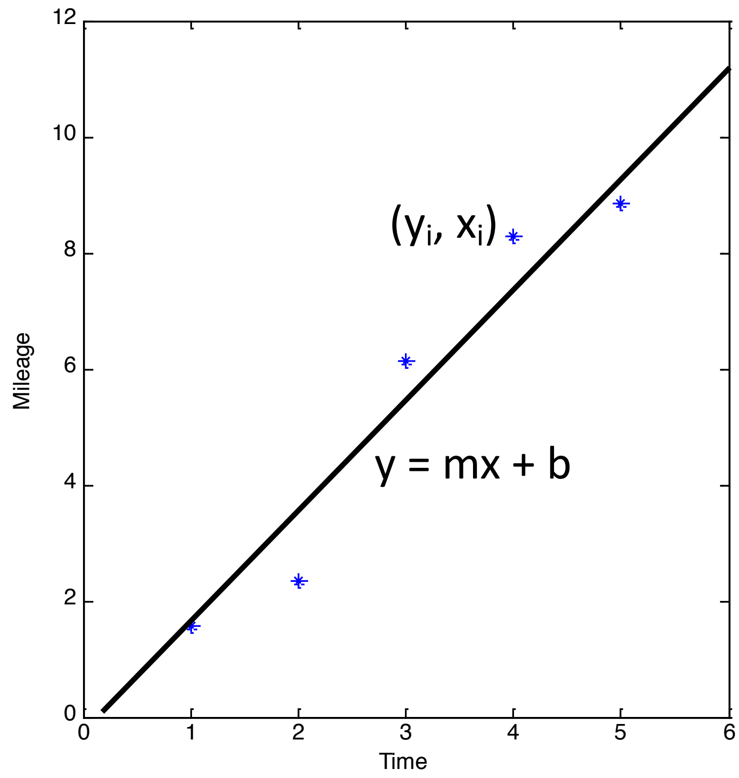
$$||\mathbf{A}\mathbf{t} - \mathbf{b}||^2$$

- To solve, form the *normal equations*

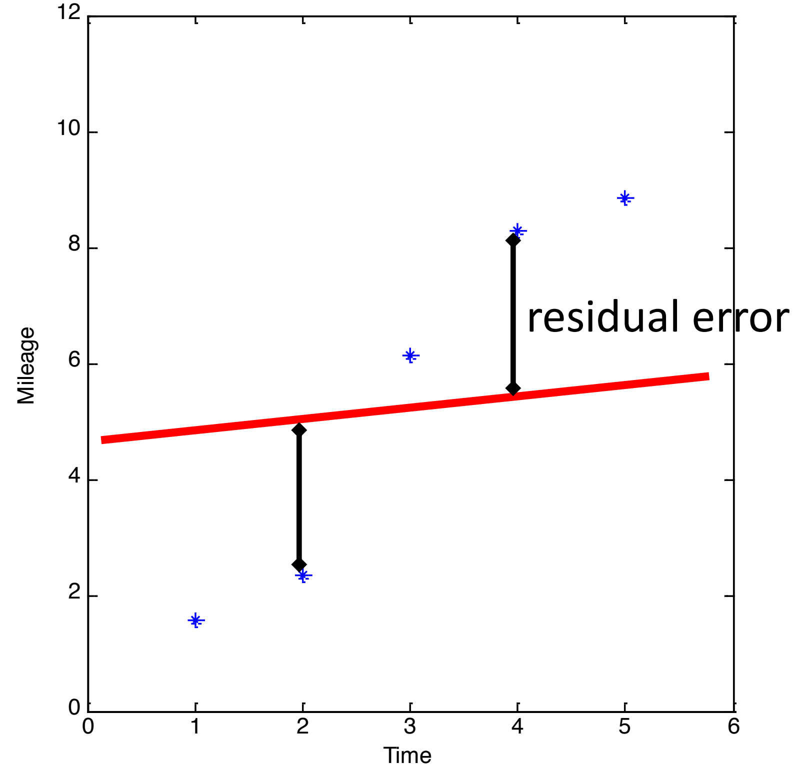
$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{t} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$$

$$\mathbf{t} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

# Least squares: linear regression



# Linear regression



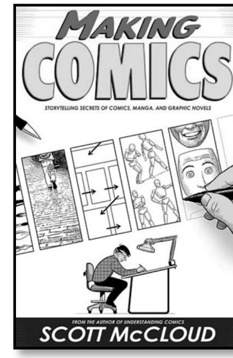

$$\mathrm{Cost}(m, b) = \sum_{i=1}^{n} |y_i - (mx_i + b)|^2$$

# Linear regression

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

# Affine transformations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$




- How many unknowns?
- How many equations per match?
- How many matches do we need?

# Affine transformations

- Residuals:

$$
\begin{aligned}
r_{x_i}(a,b,c,d,e,f) &= (ax_i + by_i + c) - x'_i \\
r_{y_i}(a,b,c,d,e,f) &= (dx_i + ey_i + f) - y'_i
\end{aligned}
$$

- Cost function:

$$
C(a,b,c,d,e,f) = \sum_{i=1}^{n} \left( r_{x_i}(a,b,c,d,e,f)^2 + r_{y_i}(a,b,c,d,e,f)^2 \right)
$$

# Affine transformations

- Matrix form

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ & & & \vdots & & \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix}$$

$$\underset{2n \times 6}{\mathbf{A}} \qquad \underset{6 \times 1}{\mathbf{t}} = \underset{2n \times 1}{\mathbf{b}}$$

# Optimization Problem to Find Transformation

**Problem statement**

$$\text{minimize } \left\| \mathbf{Ax} - \mathbf{b} \right\|^2$$

least squares solution to $\mathbf{Ax} = \mathbf{b}$

**Solution**

$$\mathbf{x} = \left(\mathbf{A}^T \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{b}$$

$\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$ (matlab)

# Image Alignment Algorithm

Given images A and B

1. Compute image features for A and B
2. Match features between A and B
3. Compute homography (or affine transformation) between A and B using least squares on set of matches

What could go wrong?

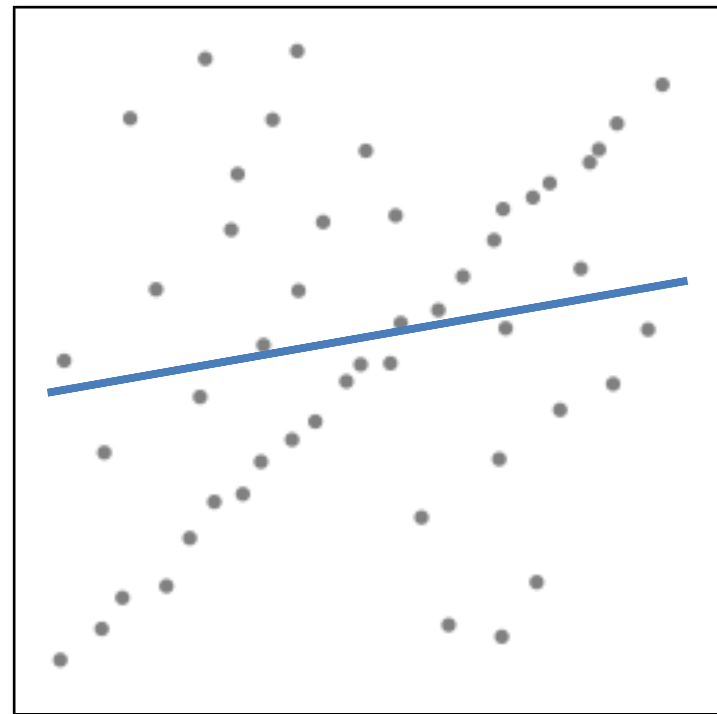
# Outliers

outliers
MAKING COMICS
STORYTELLING SECRETS OF COMICS, MANGA, AND GRAPHIC NOVELS
FROM THE AUTHOR OF UNDERSTANDING COMICS
SCOTT McCLOUD
DVD
THE COMPLETE BLACK ADDER
BBC
SOUTH WEST
inliers

# Robustness

- Let’s consider a simpler example… linear regression

Problem: Fit a line to these datapoints



Least squares fit

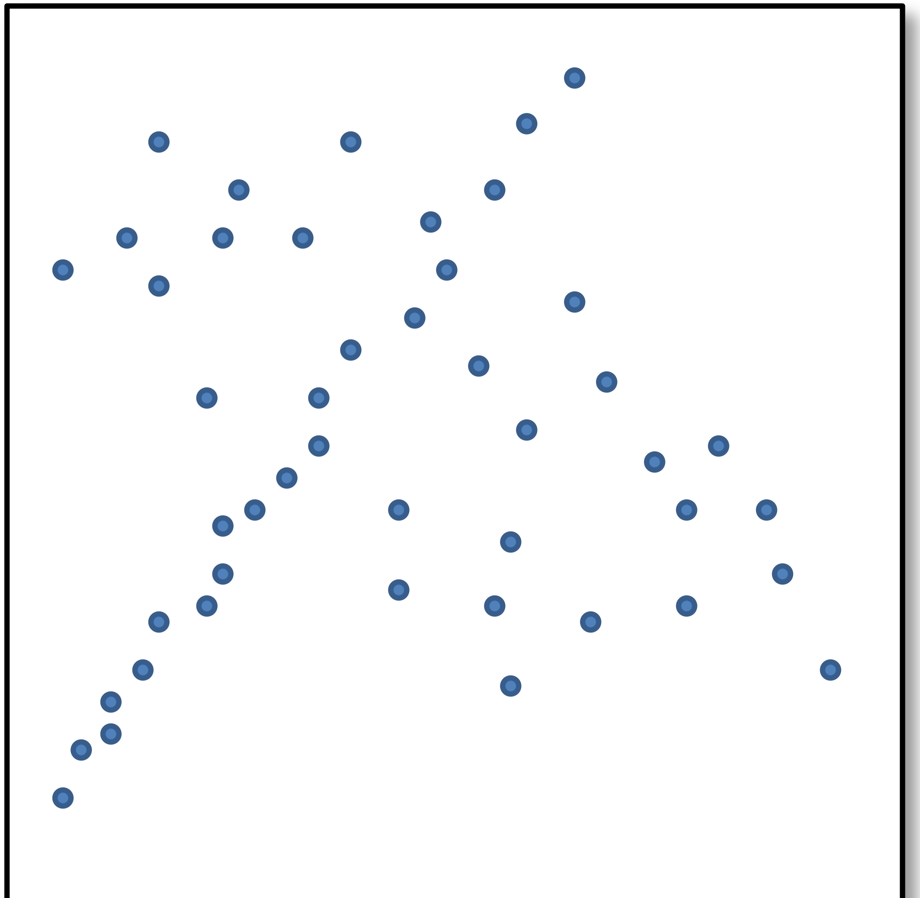
- How can we fix this?

# Idea

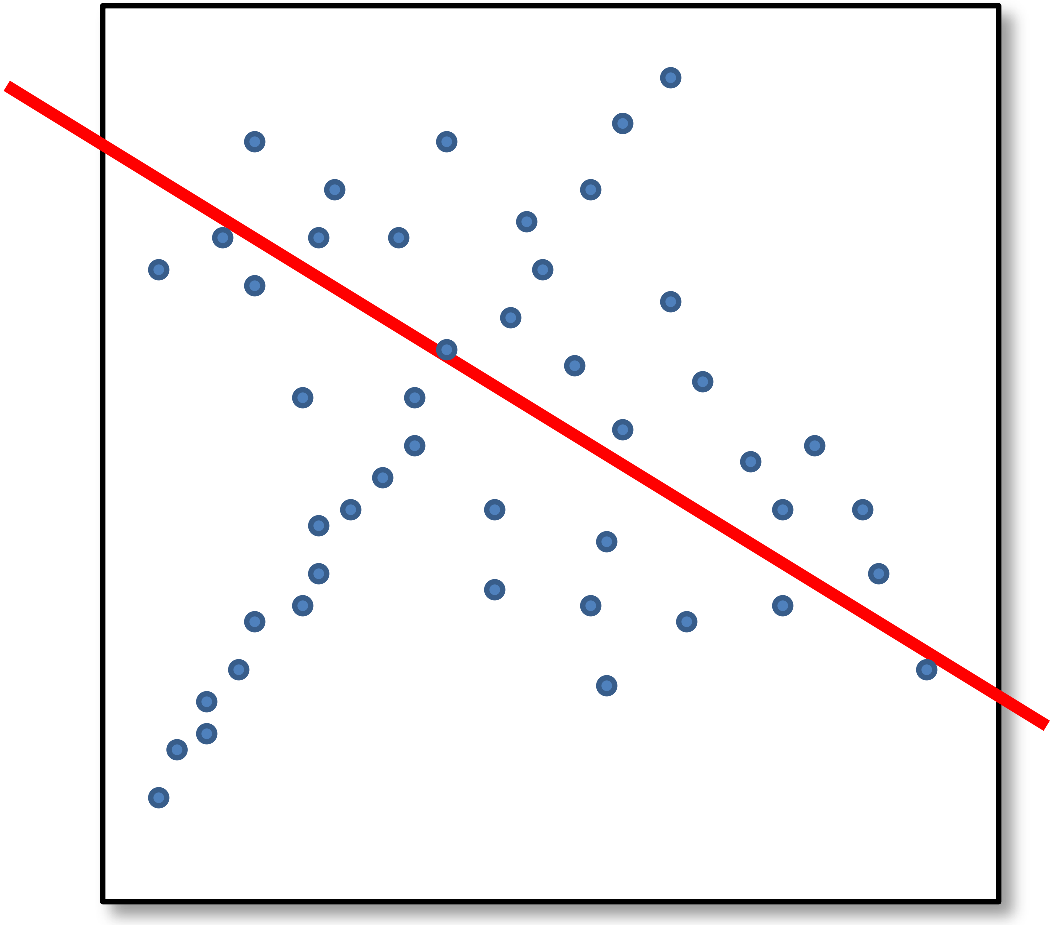
- Given a hypothesized line
- Count the number of points that “agree” with the line
  - “Agree” = within a small distance of the line
  - I.e., the **inliers** to that line

- For all possible lines, select the one with the largest number of inliers

# Counting inliers

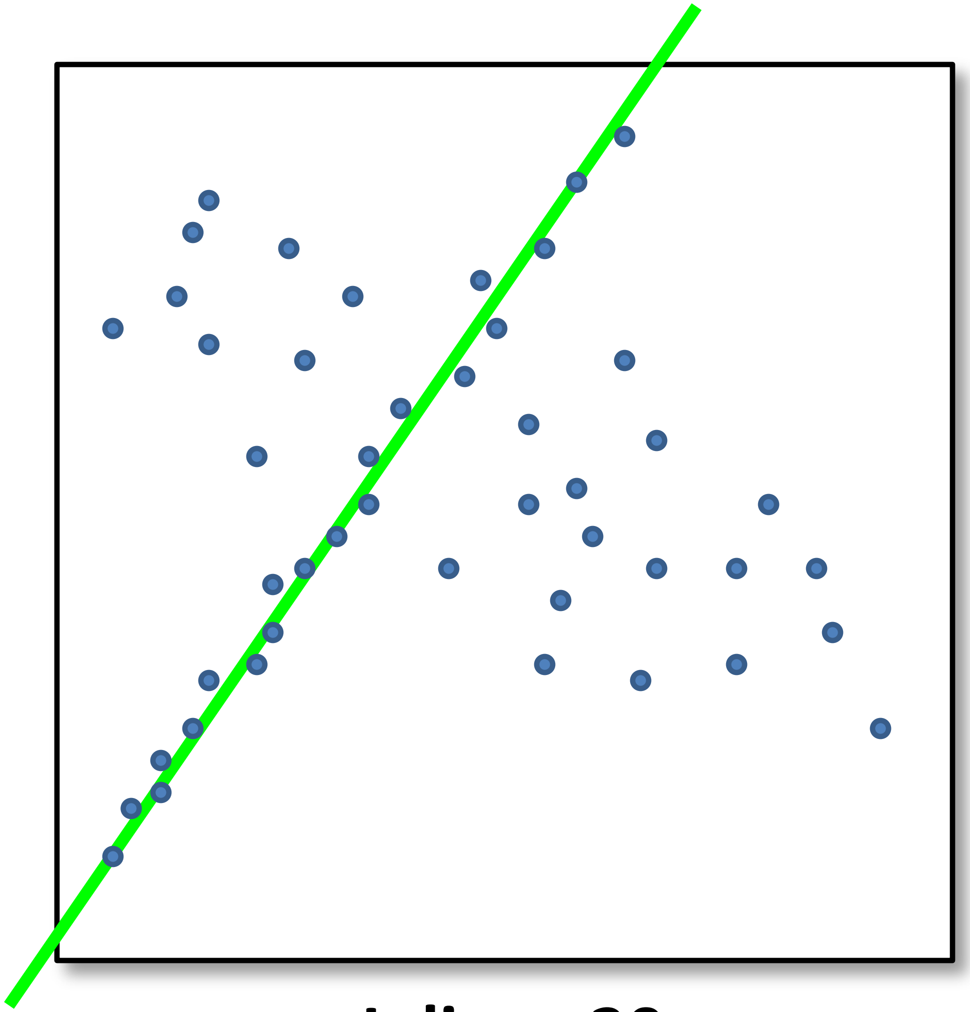


# Counting inliers



**Inliers: 3**

# Counting inliers

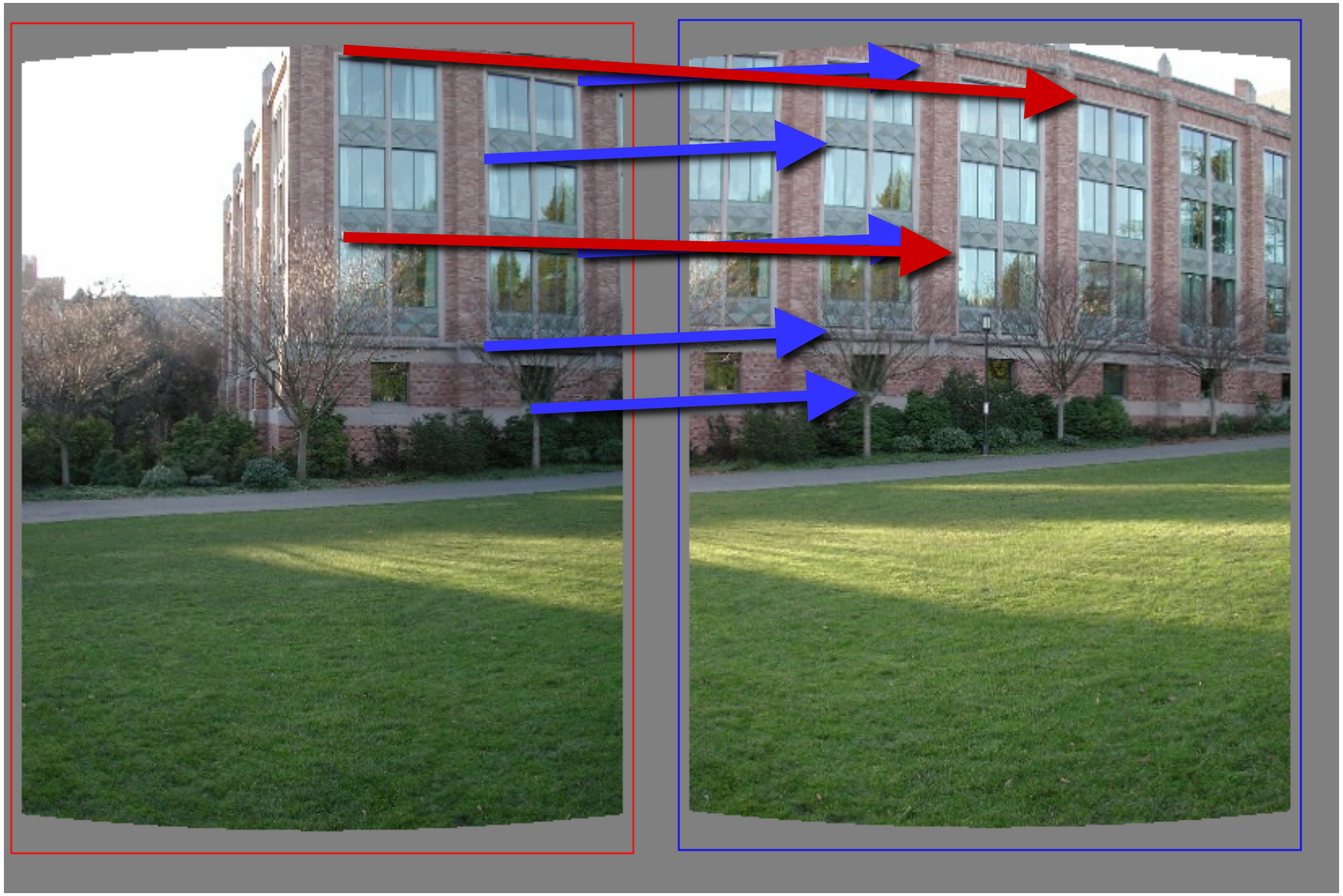


**Inliers: 20**

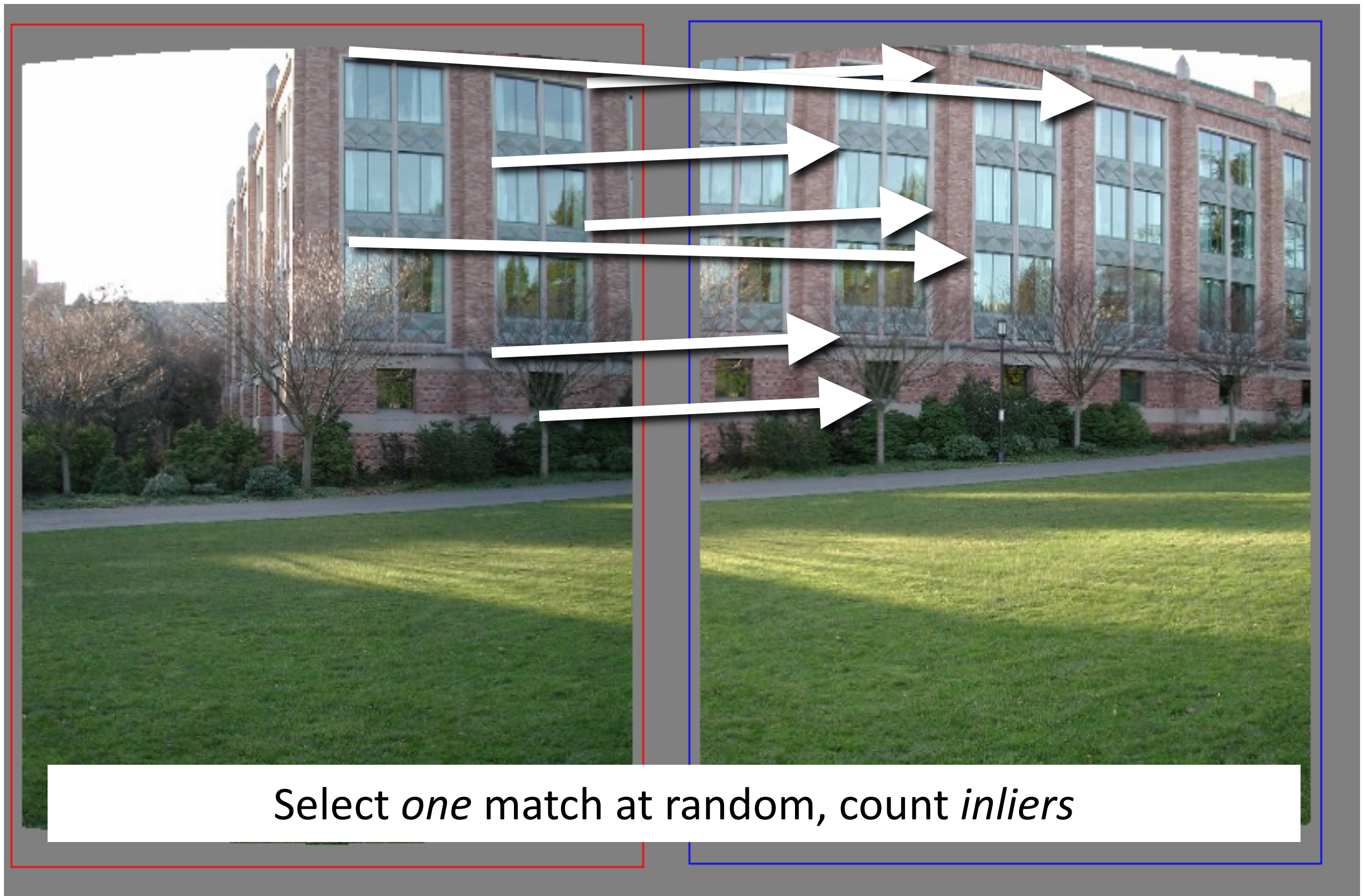
# How do we find the best line?

- Unlike least-squares, no simple closed-form solution
- Hypothesize-and-test
  - Try out many lines, keep the best one
  - Which lines?

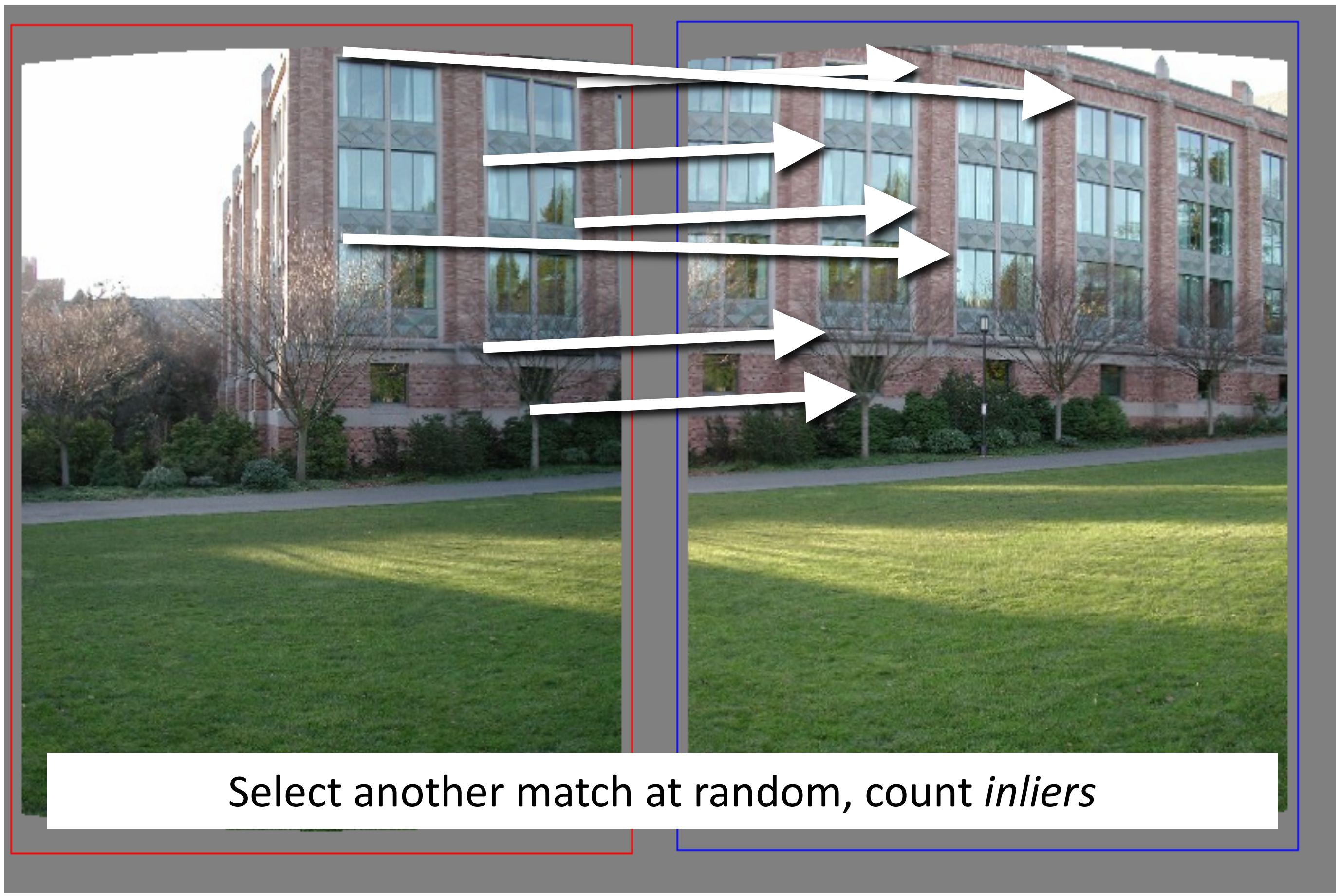
# Translations



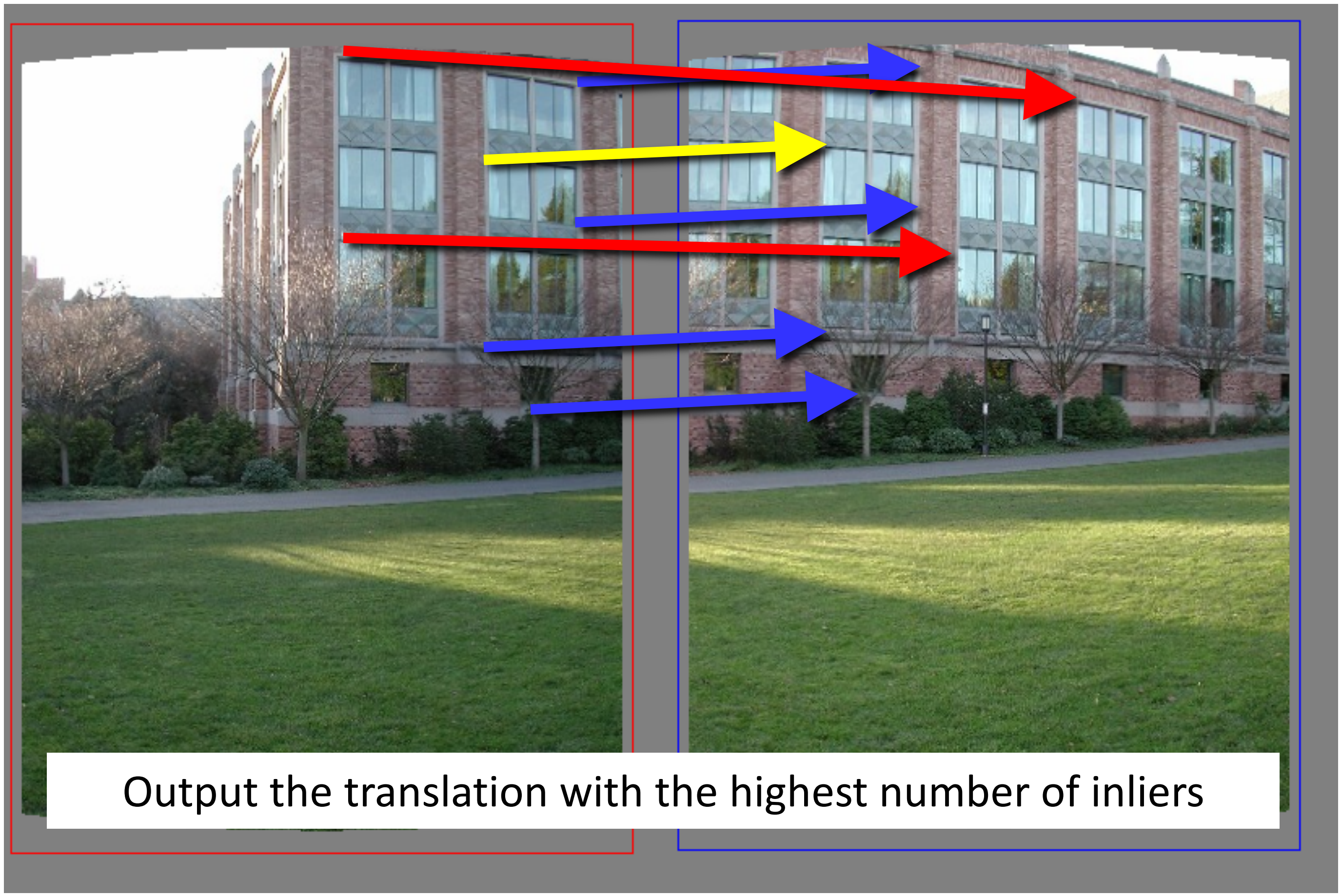
# <u>RA</u>ndom <u>SA</u>mple <u>C</u>onsensus



# <u>RA</u>ndom <u>SA</u>mple <u>C</u>onsensus



# <u>RA</u>ndom <u>SA</u>mple <u>C</u>onsensus



# RANSAC

- Idea:
  - All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
    - RANSAC only has guarantees if there are < 50% outliers

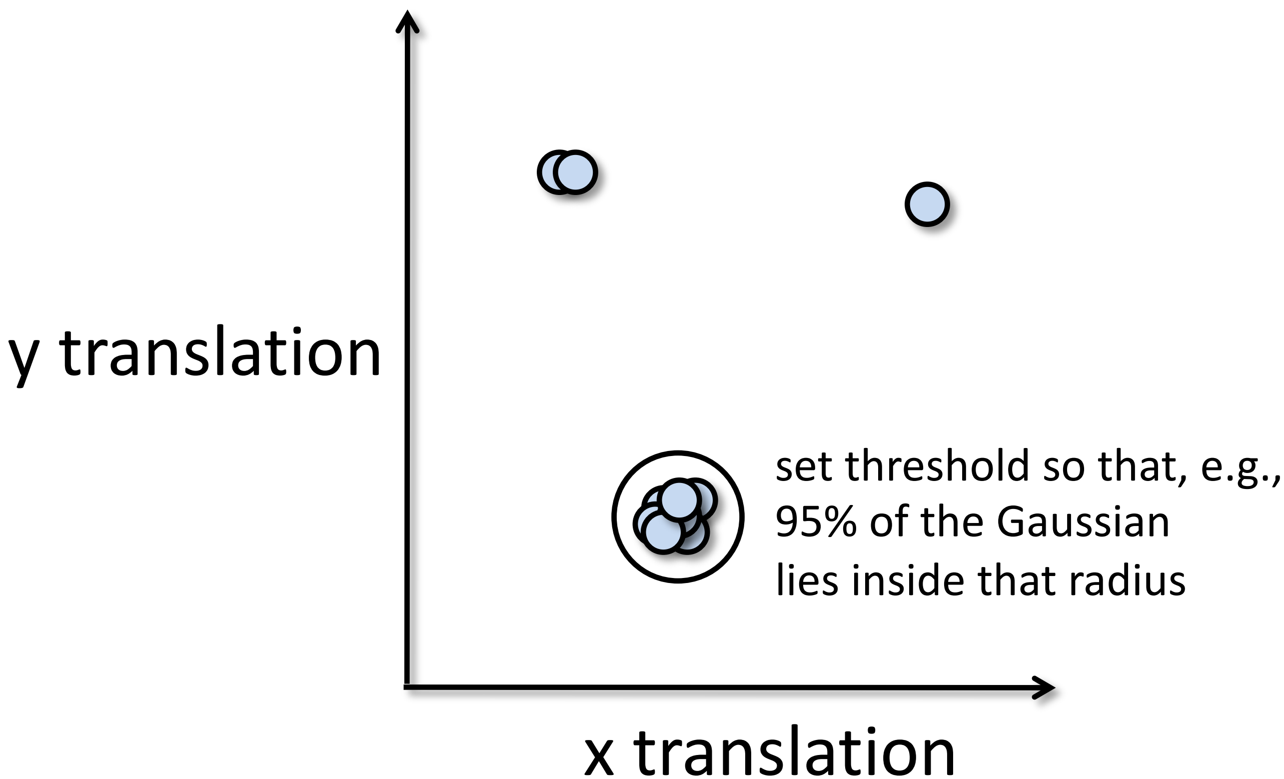
  - "All good matches are alike; every bad match is bad in its own way."

– Tolstoy via Alyosha Efros

# RANSAC

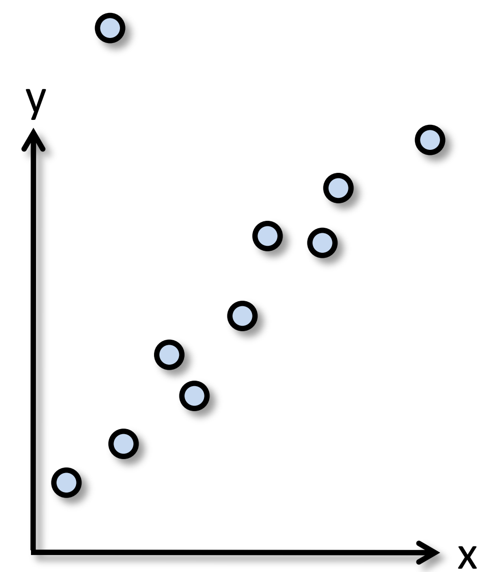
- **Inlier threshold** related to the amount of noise we expect in inliers
  - Often model noise as Gaussian with some standard deviation (e.g., 3 pixels)
- **Number of rounds** related to the percentage of outliers we expect, and the probability of success we'd like to guarantee
  - Suppose there are 20% outliers, and we want to find the correct answer with 99% probability
  - How many rounds do we need?

# RANSAC


y translation
set threshold so that, e.g.,
95% of the Gaussian
lies inside that radius
x translation

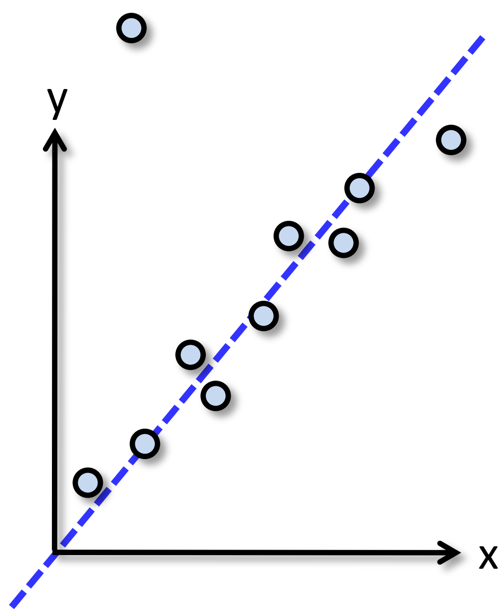
# RANSAC

- Back to linear regression
- How do we generate a hypothesis?



# RANSAC

- Back to linear regression
- How do we generate a hypothesis?



# RANSAC

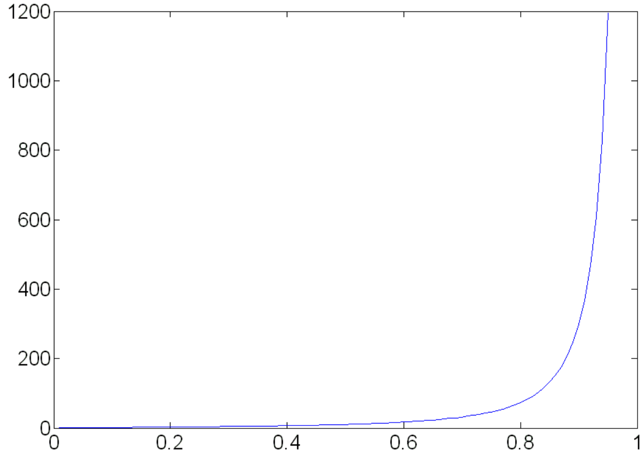
- General version:
  1. Randomly choose *s* samples
     - Typically *s* = minimum sample size that lets you fit a model
  2. Fit a model (e.g., line) to those samples
  3. Count the number of inliers that approximately fit the model
  4. Repeat *N* times
  5. Choose the model that has the largest set of inliers

# How many rounds?

- If we have to choose *s* samples each time
  - with an outlier ratio *e*
  - and we want the right answer with probability *p*

| | proportion of outliers *e* | | | | | | |
|---|---|---|---|---|---|---|---|
| s | 5% | 10% | 20% | 25% | 30% | 40% | 50% |
| 2 | 2 | 3 | 5 | 6 | 7 | 11 | 17 |
| 3 | 3 | 4 | 7 | 9 | 11 | 19 | 35 |
| 4 | 3 | 5 | 9 | 13 | 17 | 34 | 72 |
| 5 | 4 | 6 | 12 | 17 | 26 | 57 | 146 |
| 6 | 4 | 7 | 16 | 24 | 37 | 97 | 293 |
| 7 | 4 | 8 | 20 | 33 | 54 | 163 | 588 |
| 8 | 5 | 9 | 26 | 44 | 78 | 272 | 1177 |

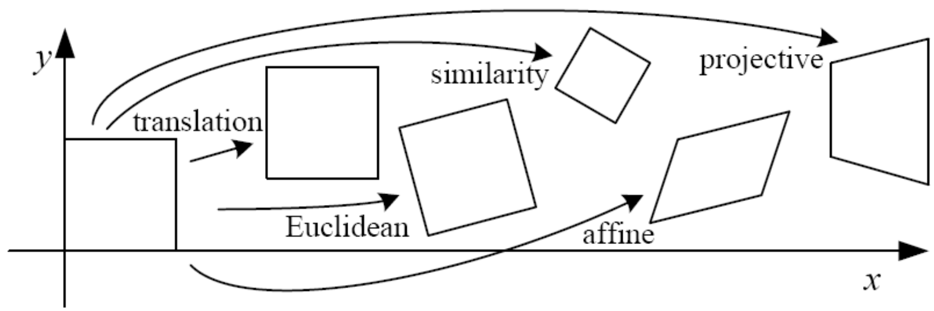
$p = 0.99$




Source: M. Pollefeys

# How big is *s*?

- For alignment, depends on the motion model
  - Here, each sample is a correspondence (pair of matching points)

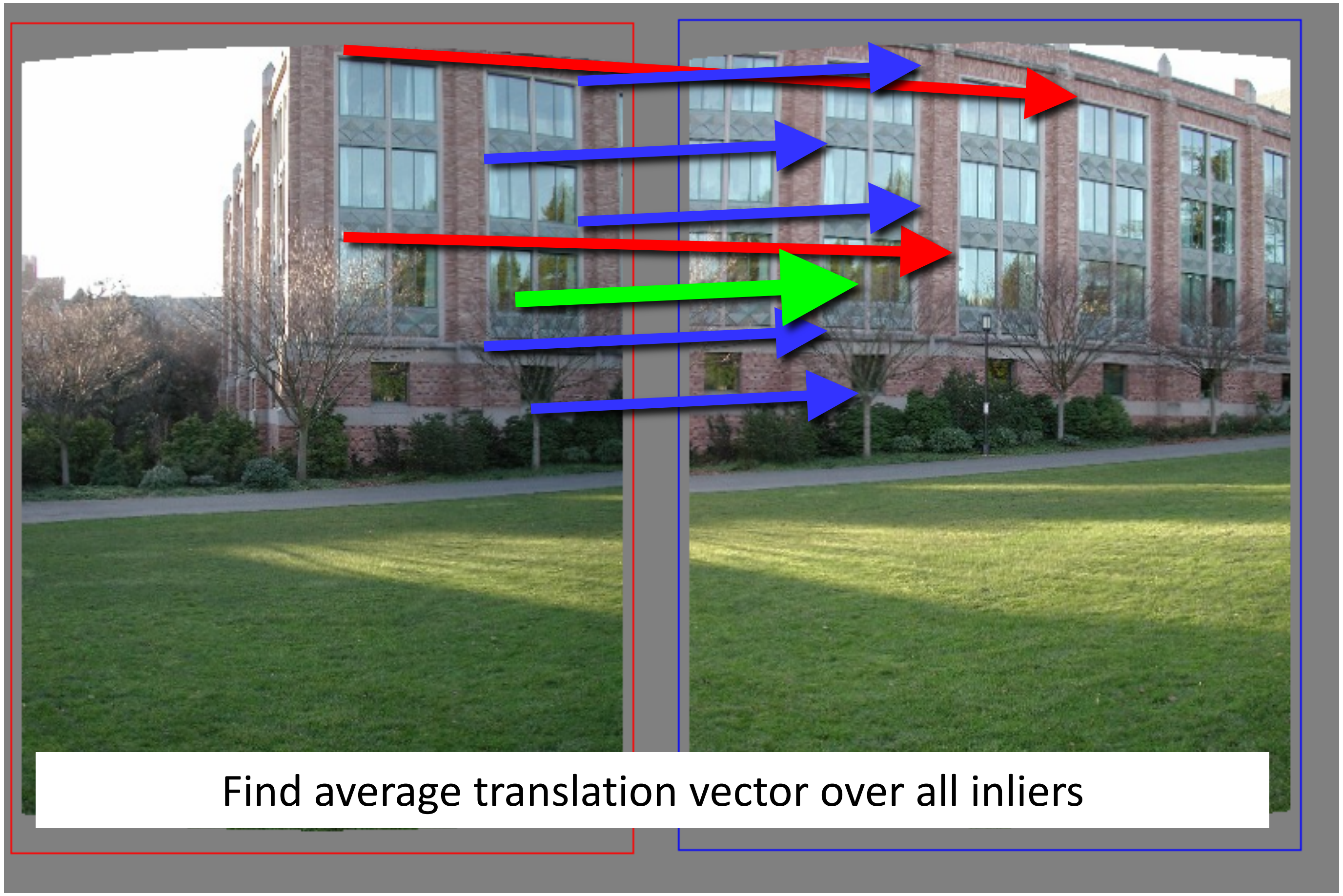



| Name | Matrix | # D.O.F. | Preserves: | Icon |
|---|---|---|---|---|
| translation | $\left[\, \boldsymbol{I} \mid \boldsymbol{t} \,\right]_{2\times 3}$ | 2 | orientation $+\cdots$ | |
| rigid (Euclidean) | $\left[\, \boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$ | 3 | lengths $+\cdots$ | |
| similarity | $\left[\, s\boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$ | 4 | angles $+\cdots$ | |
| affine | $\left[\, \boldsymbol{A} \,\right]_{2\times 3}$ | 6 | parallelism $+\cdots$ | |
| projective | $\left[\, \tilde{\boldsymbol{H}} \,\right]_{3\times 3}$ | 8 | straight lines | |

# RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Parameters to tune
  - Sometimes too many iterations are required
  - Can fail for extremely low inlier ratios
  - We can often do better than brute-force sampling

# Final step: least squares fit



# RANSAC

- An example of a “voting”-based fitting scheme
- Each hypothesis gets voted on by each data point, best hypothesis wins

- There are many other types of voting schemes
  - E.g., Hough transforms…

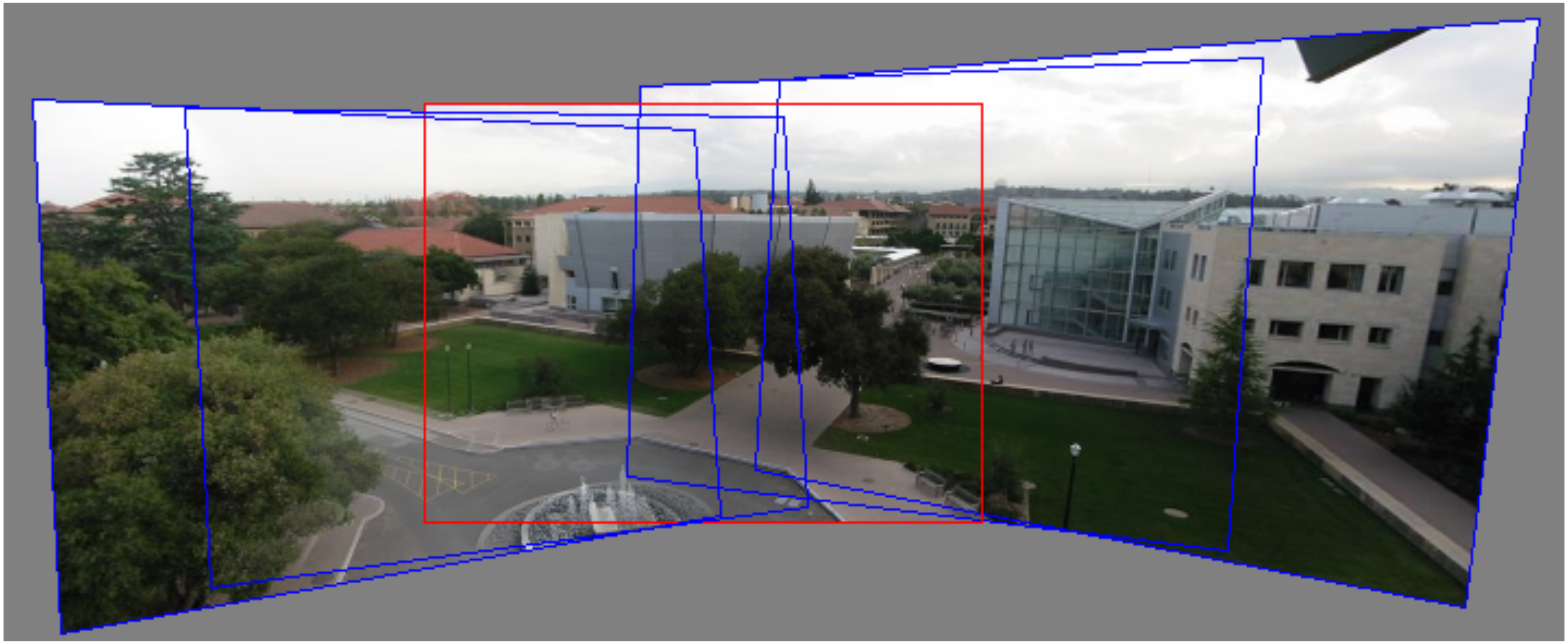
# Panoramas

- Now we know how to create panoramas!

- Given two images:
  - Step 1: Detect features
  - Step 2: Match features
  - Step 3: Compute a homography using RANSAC
  - Step 4: Combine the images together (somehow)

- What if we have more than two images?

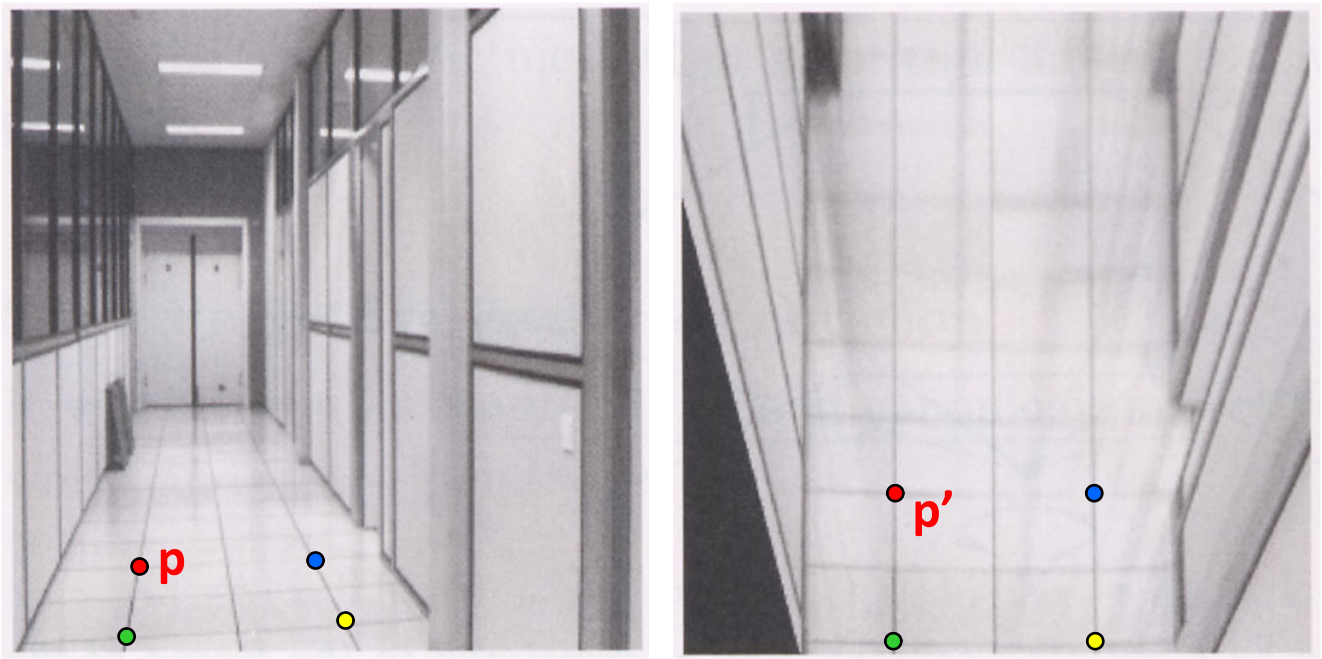
# Can we use homographies to create a 360 panorama?



- In order to figure this out, we need to learn what a **camera** is

# 360 panorama

# Homographies




To unwarp (rectify) an image

- solve for homography **H** given **p** and **p’**
- solve equations of the form: w**p’** = **Hp**
  - linear in unknowns: w and coefficients of **H**
  - H is defined up to an arbitrary scale factor
  - how many points are necessary to solve for **H**?

# Solving for homographies

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x'_i = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$y'_i = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

**Not linear!**

$$\begin{aligned} x'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

# Solving for homographies

$$
\begin{aligned}
x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\
y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12}
\end{aligned}
$$

$$
\begin{bmatrix}
x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\
0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i'
\end{bmatrix}
\begin{bmatrix}
h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22}
\end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \end{bmatrix}
$$

# Solving for homographies

$$
\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\
0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\
 & & & & & \vdots & & & \\
x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\
0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n
\end{bmatrix}
\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}
$$

$$
\underset{2n \times 9}{\mathbf{A}} \qquad \underset{9}{\mathbf{h}} \qquad \underset{2n}{\mathbf{0}}
$$

Defines a least squares problem: minimize $\|\mathbf{Ah} - \mathbf{0}\|^2$

- Since $\mathbf{h}$ is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more points

# Recap: Two Common Optimization Problems

| Problem statement | Solution |
| --- | --- |
| minimize $\left\Vert \mathbf{Ax} - \mathbf{b} \right\Vert^2$ | $\mathbf{x} = \left(\mathbf{A}^T \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{b}$ |
| least squares solution to $\mathbf{Ax} = \mathbf{b}$ | $\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$ (matlab) |

| Problem statement | Solution |
| --- | --- |
| minimize $\mathbf{x}^T \mathbf{A}^T \mathbf{Ax}$ s.t. $\mathbf{x}^T \mathbf{x} = 1$ | $[\mathbf{v}, \lambda] = \mathrm{eig}(\mathbf{A}^T \mathbf{A})$ |
| | $\lambda_1 < \lambda_{2..n} : \mathbf{x} = \mathbf{v}_1$ |
| non - trivial lsq solution to $\mathbf{Ax} = 0$ | |