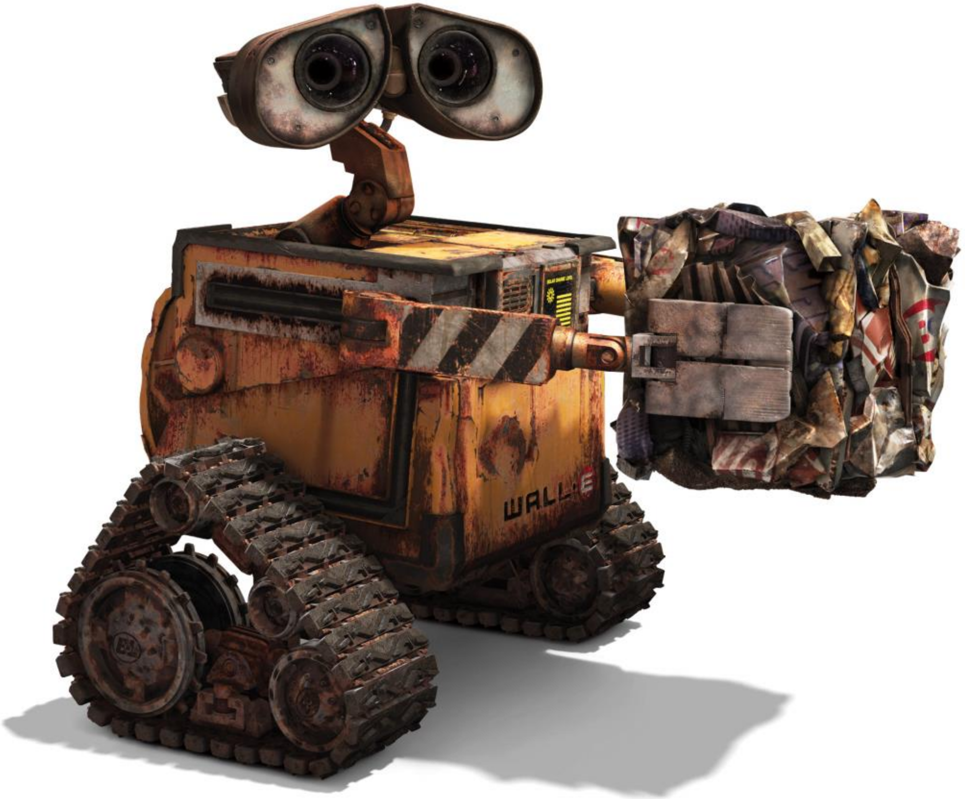
INF 5300 Advanced Topic: Video Content Analysis

Observing from a moving platform

Asbjørn Berge



Demo: Realtime 3D mapping

- Track features in 3D data from a Kinect to simultaneously map the surroundings and locate the camera.
- Fundamentally these ideas behind autonomous robot navigation.

Reading materials and tools

R. Szeliski: **Computer Vision: Algorithms and Applications**
Chapters 4.1, 6.1 and 7.1+7.2, http://szeliski.org/Book/

David G. Lowe, **Distinctive image features from scale-invariant keypoints,** *International Journal of Computer Vision,* 60, 2 (2004), pp. 91-110. [PDF]

M. Zuliani: **Ransac for dummies**
http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf

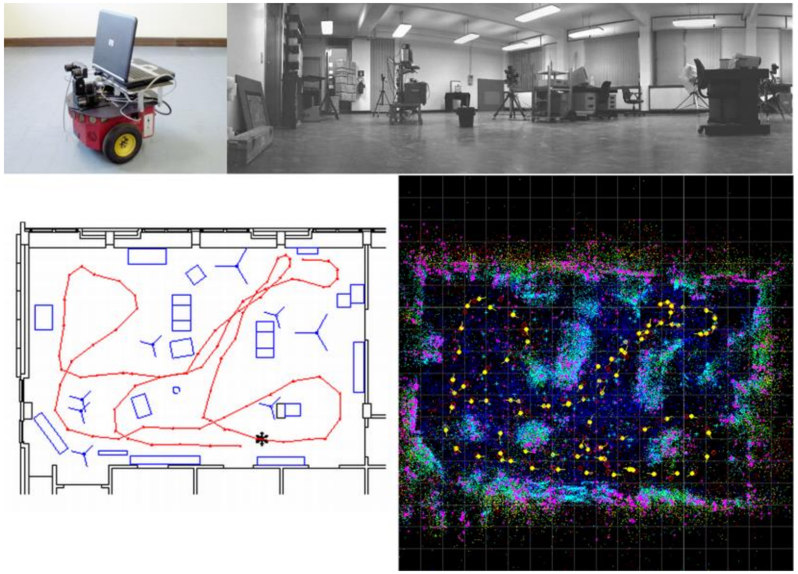
Tools

Ransac toolbox : https://github.com/RANSAC/RANSAC-Toolbox
VlFeat toolbox : http://www.vlfeat.org
OpenCV 3D reconstruction:
http://opencv.itseez.com/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Inferring 3D

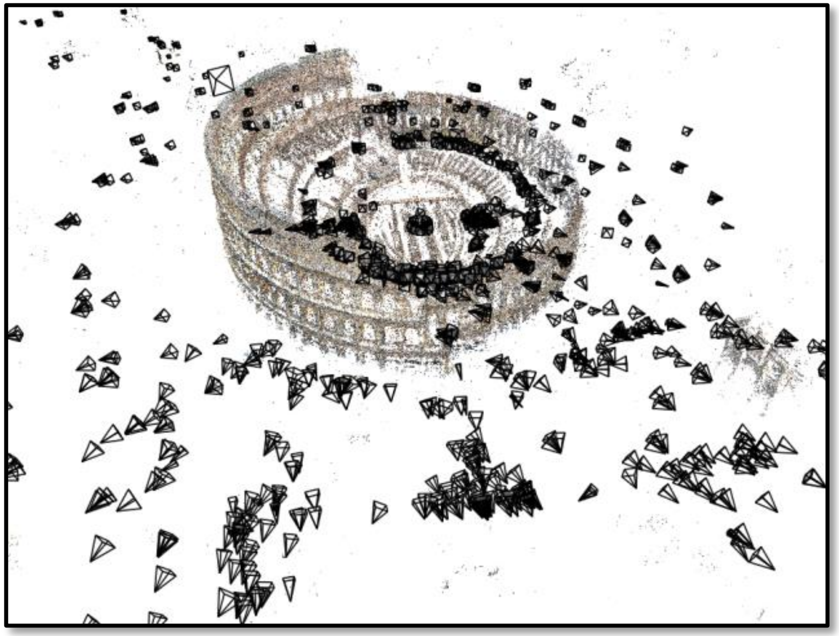
- Structure from motion
  - Obtain 3D scene structure from multiple images from the same camera in different locations, poses
  - Typically, camera location & pose treated as unknowns
  - Track points across frames, infer camera pose & scene structure from correspondences

- Simultaneous Location And Mapping (SLAM)
  - Localize a robot and map its surroundings with a single camera



3D Reconstruction

Internet Photos ("Colosseum")



Reconstructed 3D cameras and points

http://photosynth.net/default.aspx

http://phototour.cs.washington.edu/applet/index.html

Some panorama examples

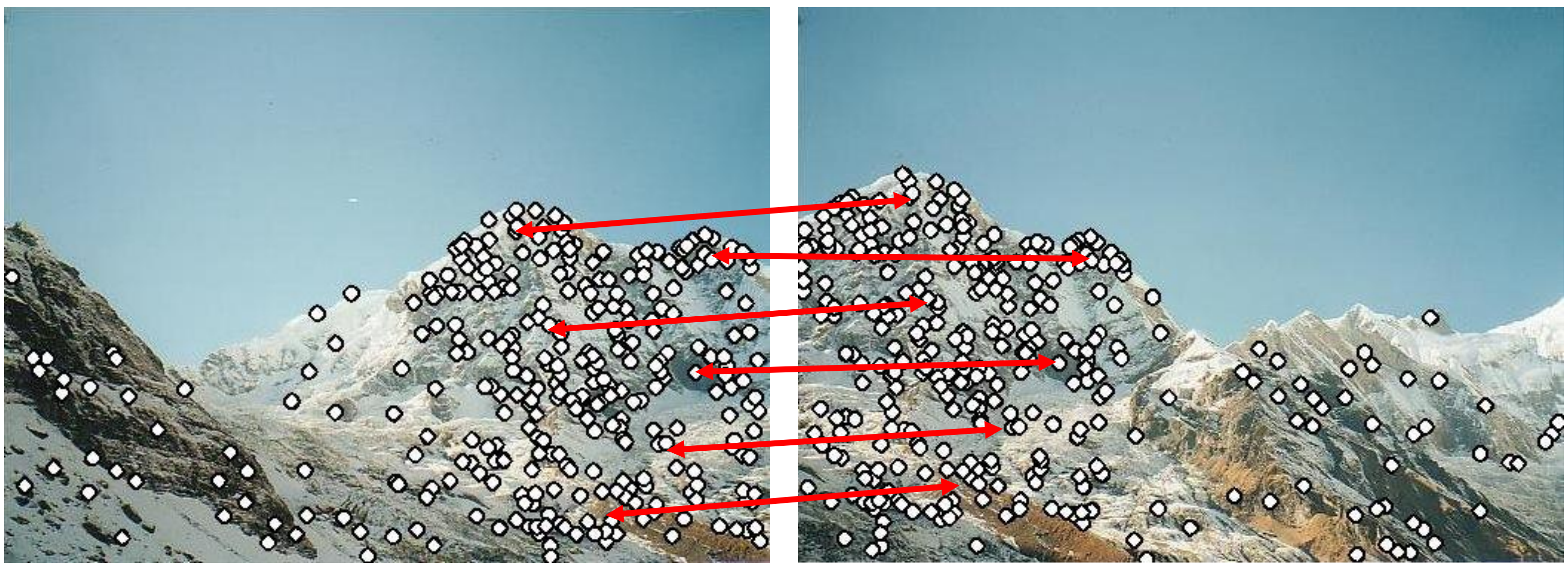
- Every image on Google Streetview

Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?

Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?



Step 1: extract features
Step 2: match features

Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?

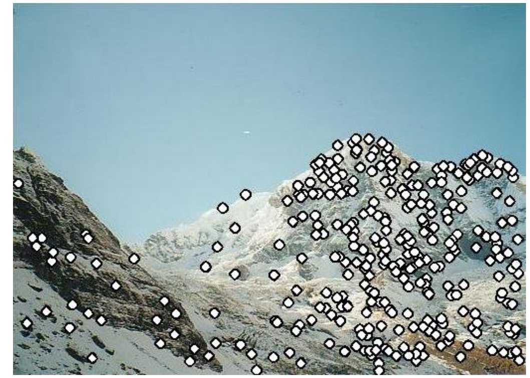
Step 1: extract features
Step 2: match features
Step 3: align images

Local invariant features: outline

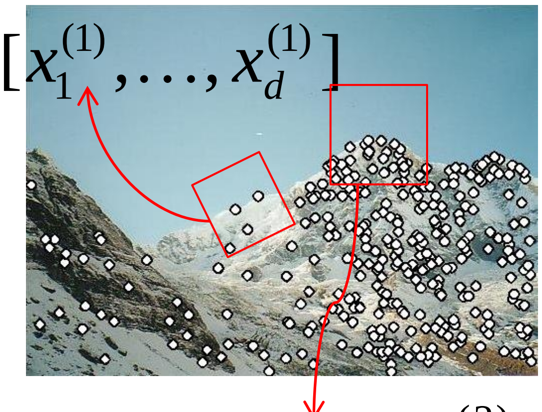
1) **Detection:** Identify the interest points

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

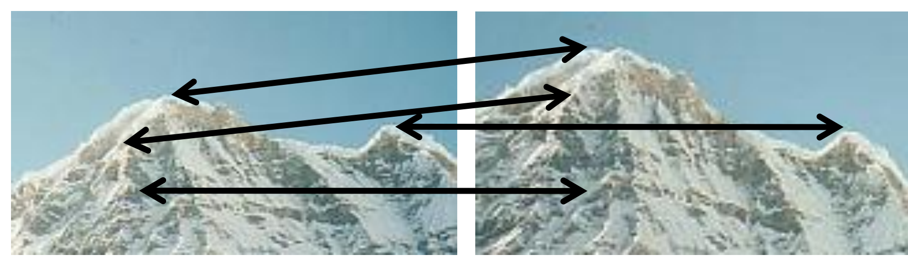
3) **Matching:** Determine correspondence between descriptors in two views



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



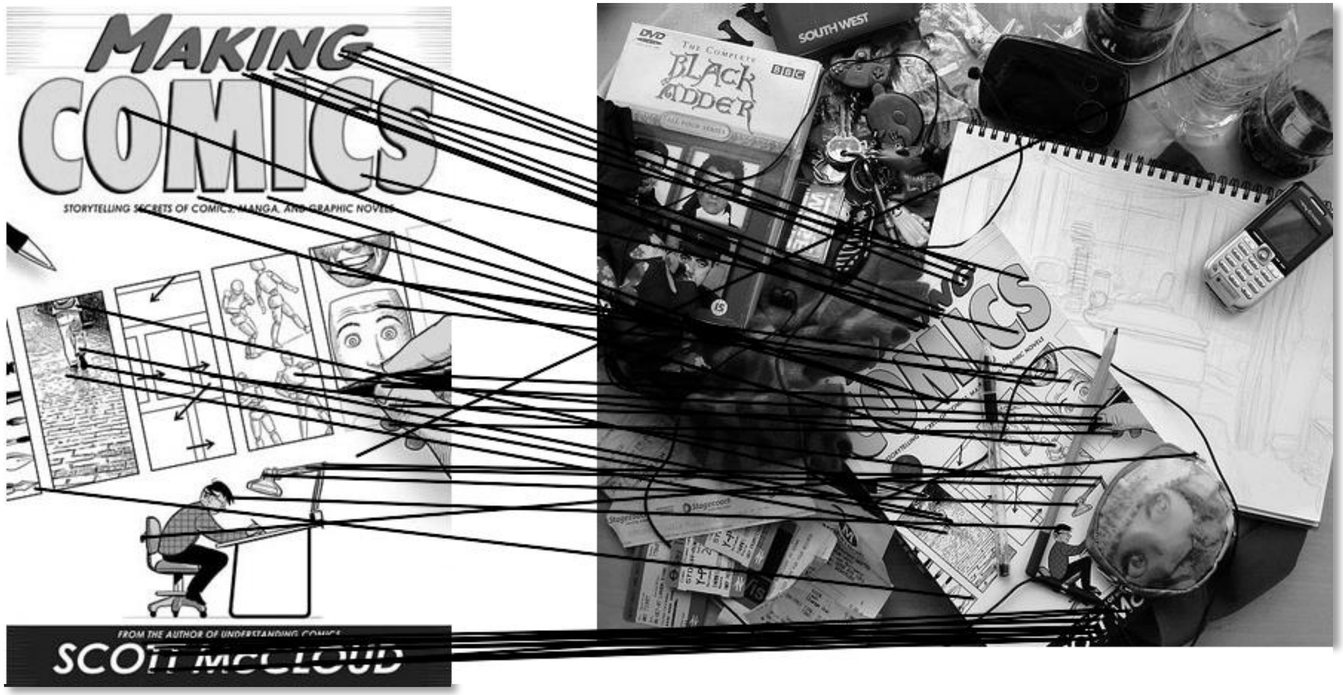
$$\mathbf{x}_2 = [x_1^{(2)}, \dots, x_d^{(2)}]$$



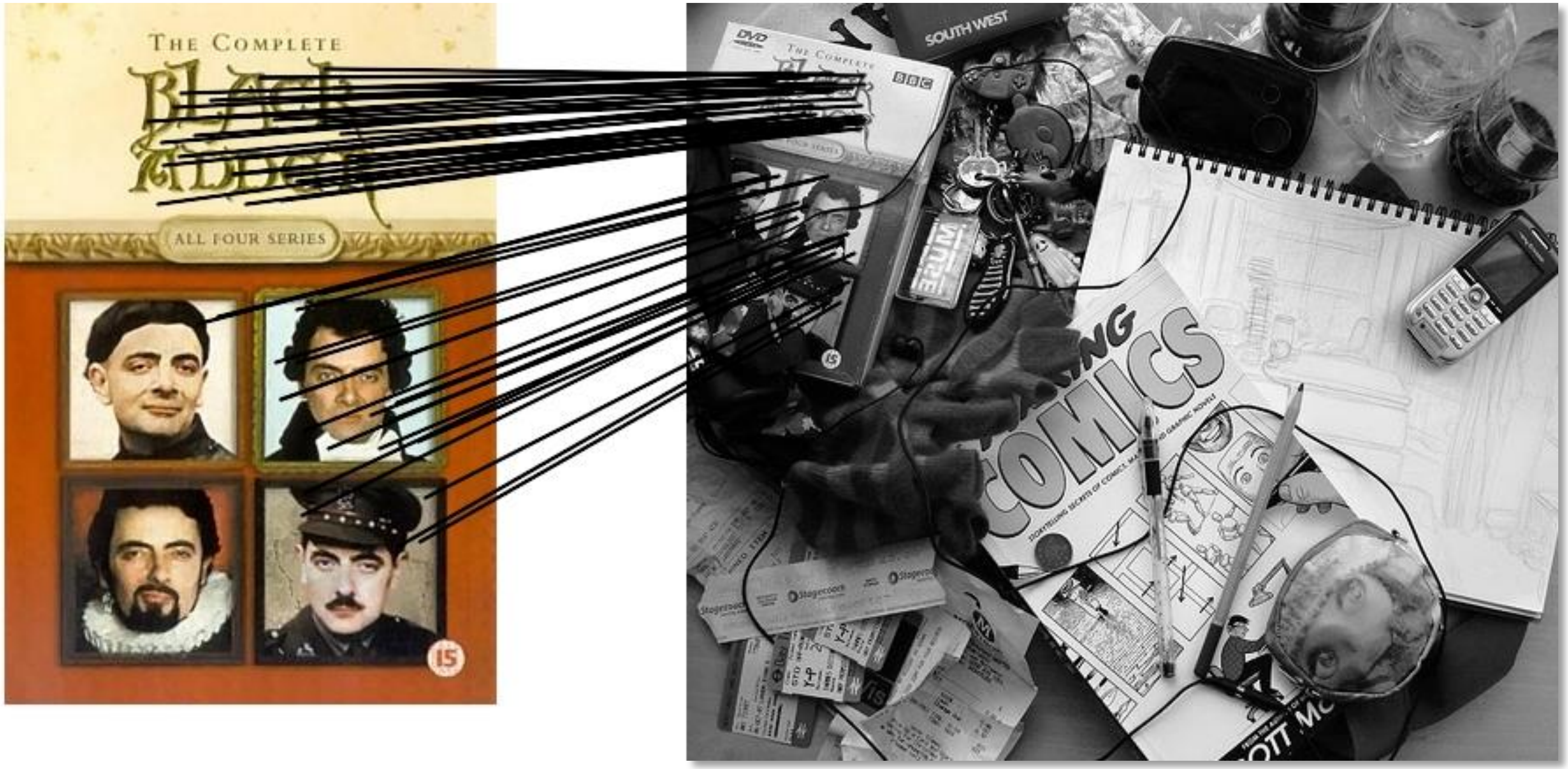
Computing transformations

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

  –



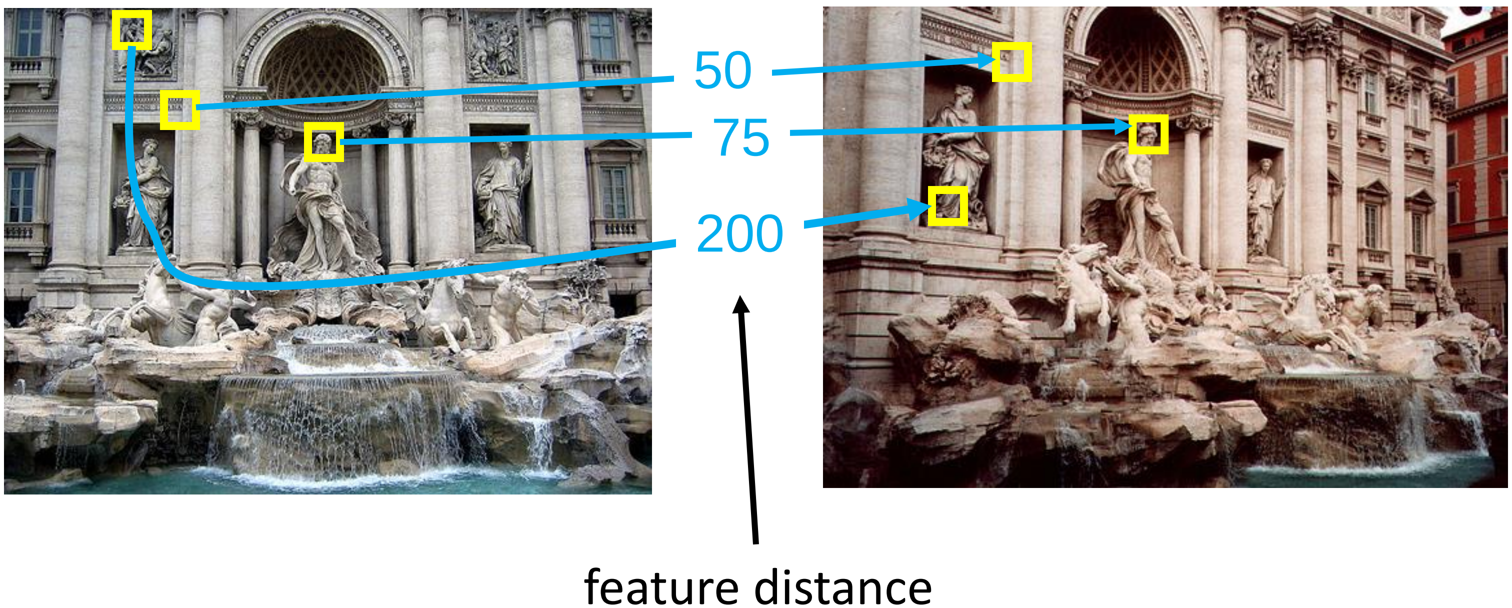
Feature matching example



58 matches

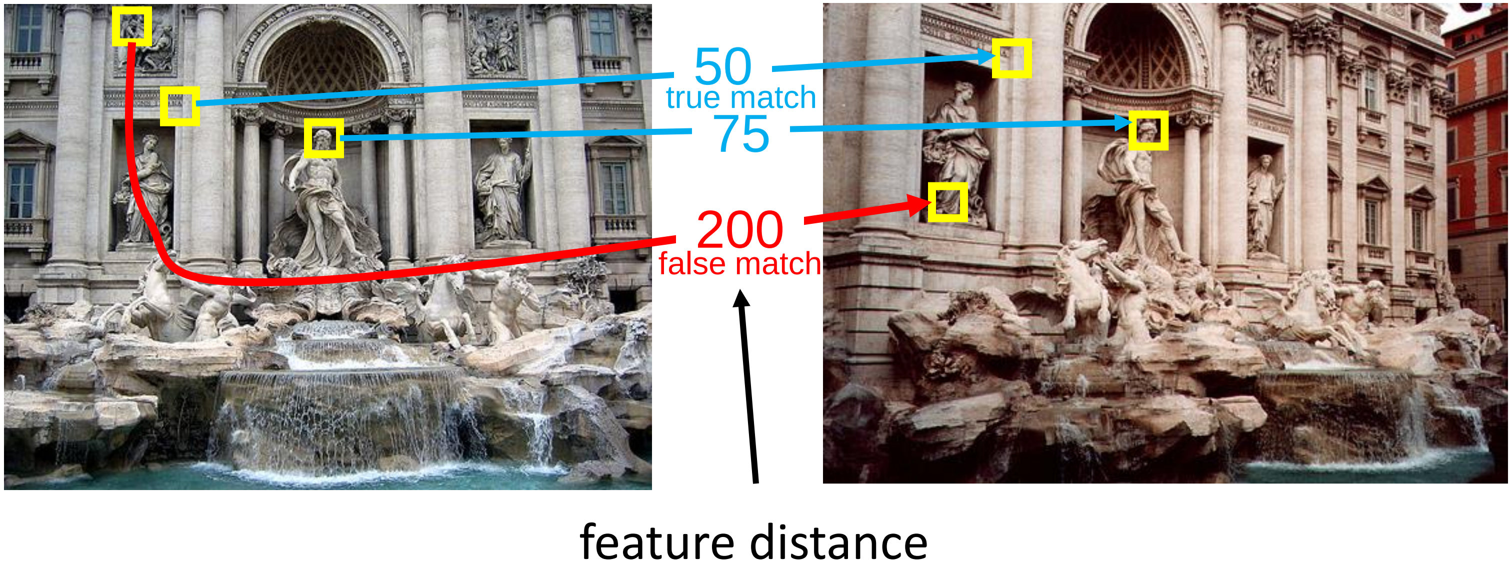
Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives

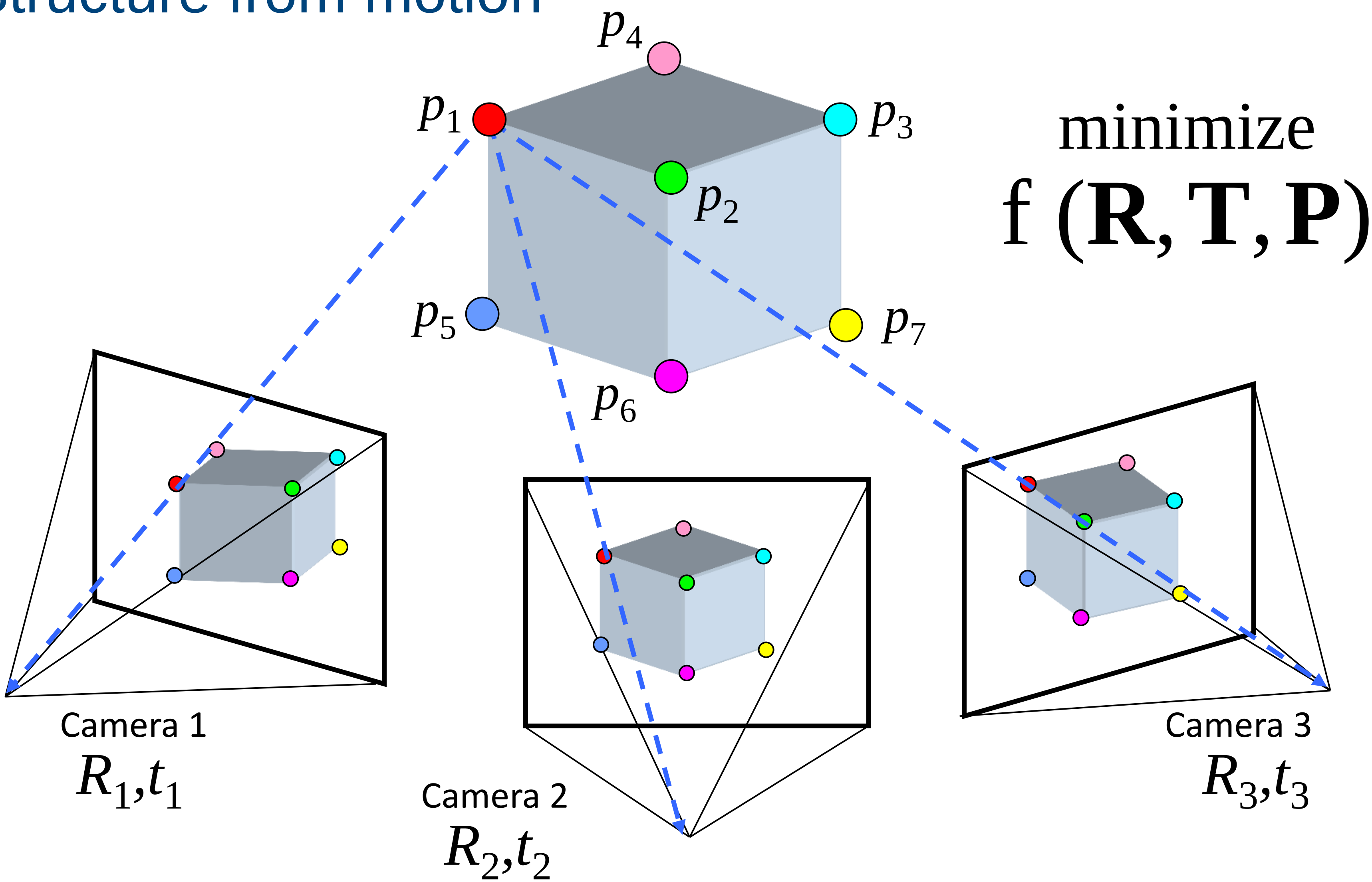
How can we measure the performance of a feature matcher?



The distance threshold affects performance

- True positives = # of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?
- False positives = # of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?

Structure from motion



SfM objective function

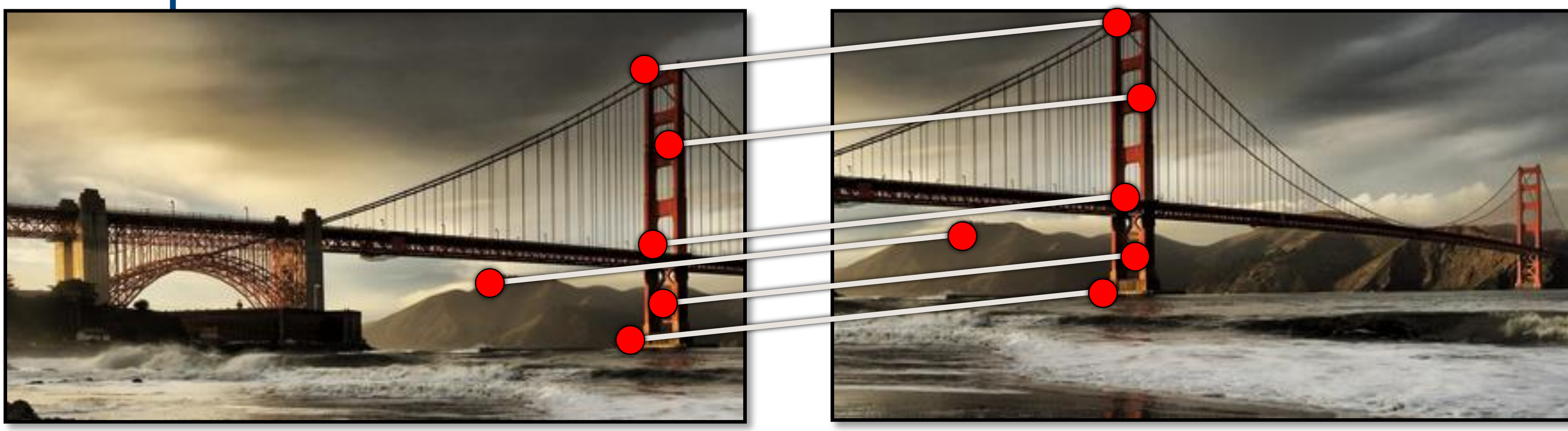
- Given point **x** and rotation and translation **R**, **t**

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \mathbf{R}\mathbf{x} + \mathbf{t} \qquad u' = \frac{f x'}{z'} \quad v' = \frac{f y'}{z'} \qquad \begin{bmatrix} u' \\ v' \end{bmatrix} = \mathbf{P}(\mathbf{x}, \mathbf{R}, \mathbf{t})$$

- Minimize sum of squared reprojection errors:

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} \cdot \left\| \underbrace{\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)}_{\substack{\textit{predicted} \\ \text{image location}}} - \underbrace{\begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix}}_{\substack{\textit{observed} \\ \text{image location}}} \right\|^2$$

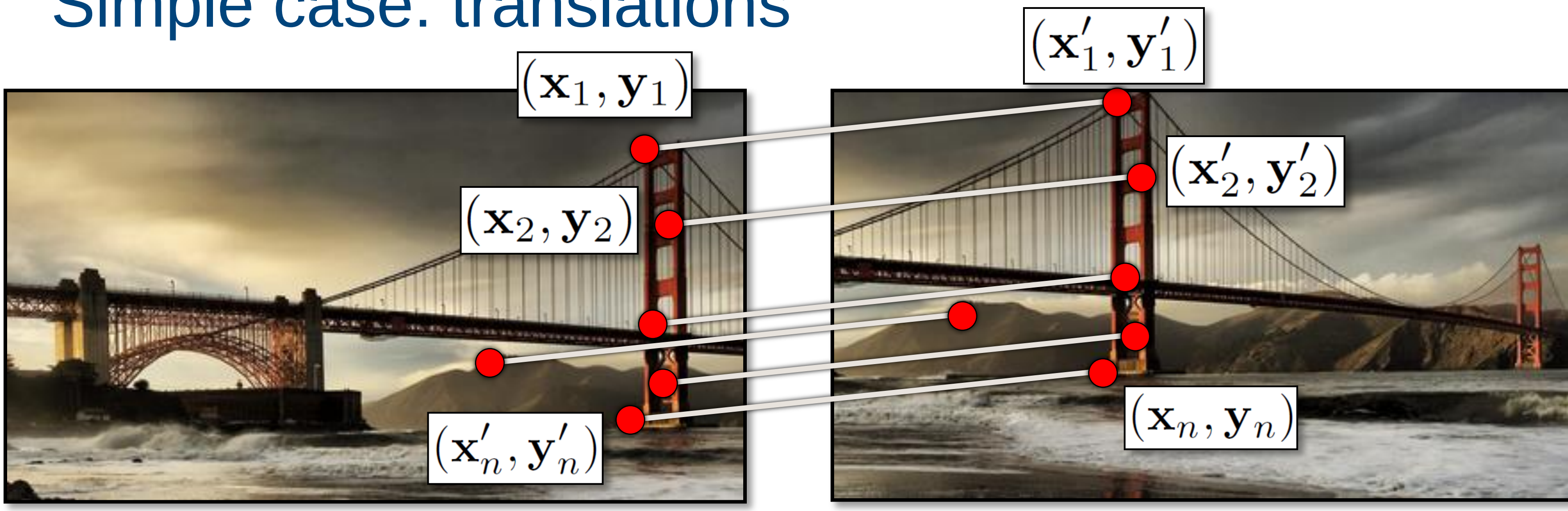
Simple case: translations



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

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

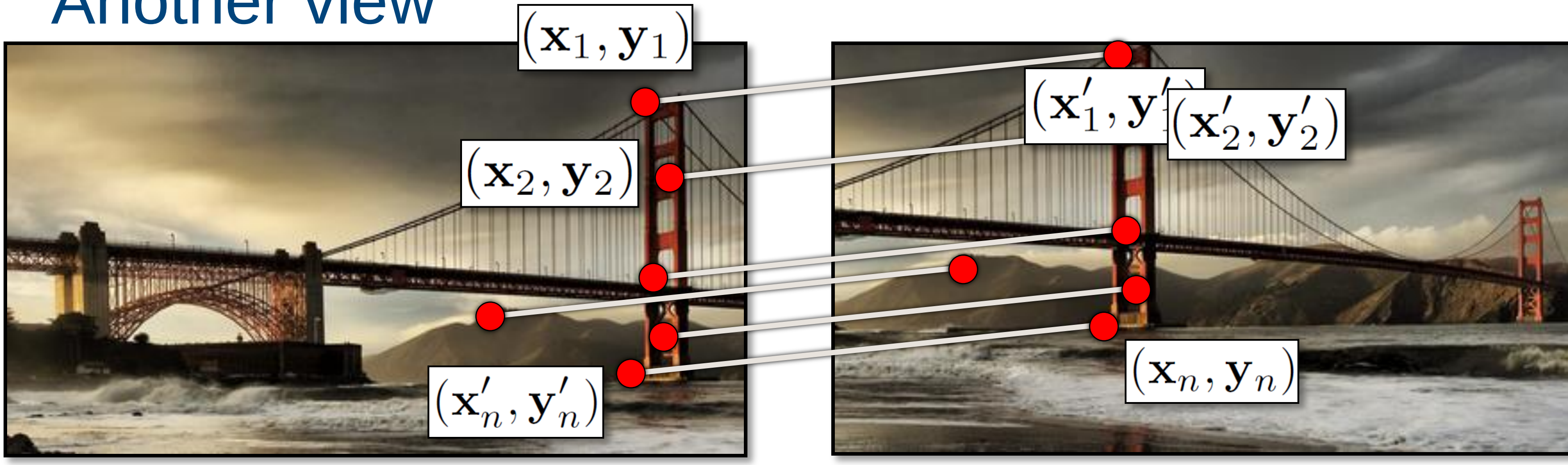
Simple case: translations



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

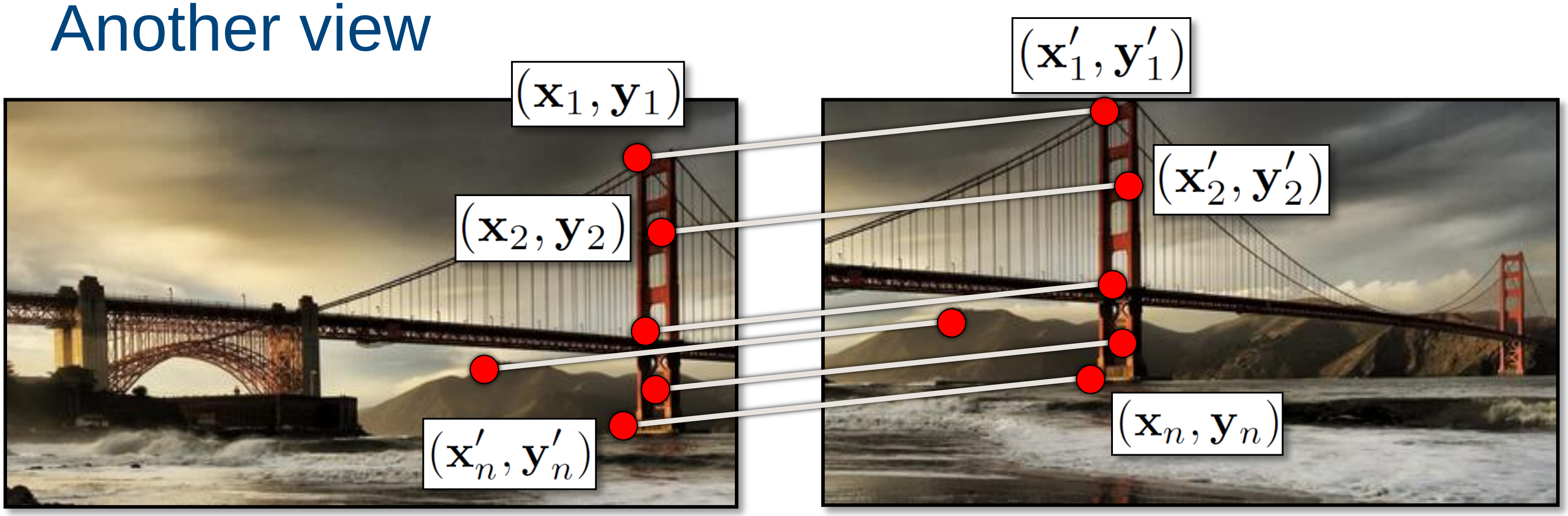


$$\mathbf{x}_i + \mathbf{x_t} = \mathbf{x}'_i$$

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

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

Another view



$$\mathbf{x}_i + \mathbf{x_t} = \mathbf{x}'_i$$
$$\mathbf{y}_i + \mathbf{y_t} = \mathbf{y}'_i$$

- 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 displacement

Least squares formulation

- Can also write as a matrix equation

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

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

$2n$ x 2   2 x 1 $2n$ x 1

Least squares

- Find **t** that minimizes

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

- To solve, form the *normal equations*

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

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

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

Projection matrix

$$\mathbf{\Pi} = \mathbf{K} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0\;0\;0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3\times 3} & -\mathbf{c} \\ 0\;0\;0 & 1 \end{bmatrix}$$

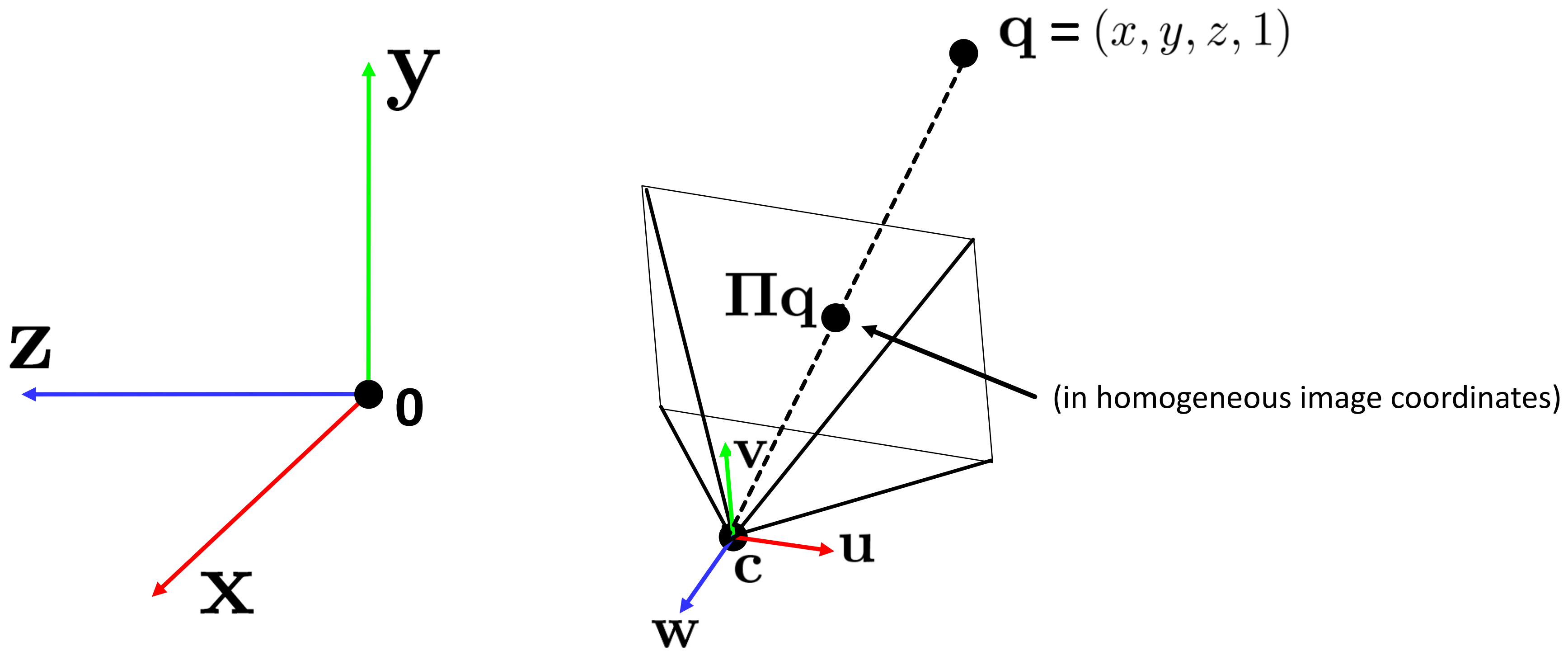
intrinsics, projection, rotation, translation

$$\begin{bmatrix} \mathbf{R} & | & -\mathbf{Rc} \end{bmatrix}$$

($\mathbf{t}$ in book's notation)

$$\mathbf{\Pi} = \mathbf{K} \begin{bmatrix} \mathbf{R} & | & -\mathbf{Rc} \end{bmatrix}$$

Projection matrix



Why Mosaic?

- Are you getting the whole picture?
  - Compact Camera FOV = 50 x 35°
  - Human FOV = 200 x 135°
  - Panoramic Mosaic = 360 x 180°

Projective Transformations aka Homographies aka Planar Perspective Maps

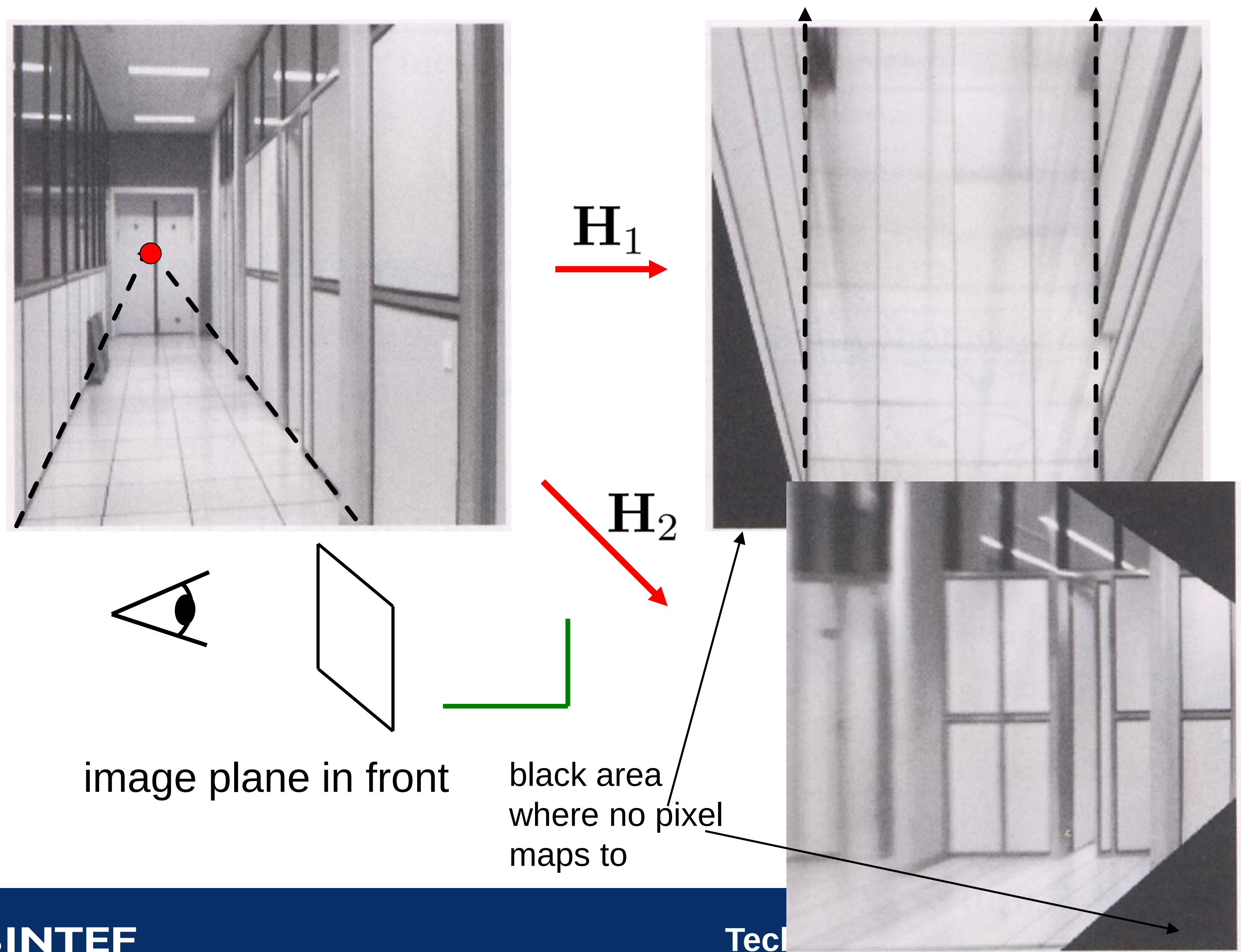
$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

Called a *homography*

(or *planar perspective map*)

projection of 3D plane can be explained by a (homogeneous) 2D transform

Image warping with homographies



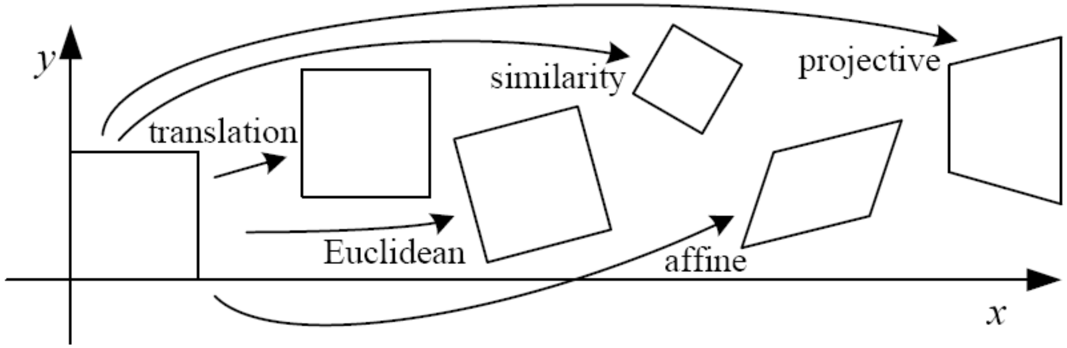
Homographies

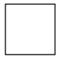
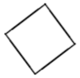
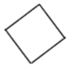
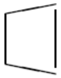
- Homographies …
  - Affine transformations, and
  - Projective warps

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

- 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

2D image transformations

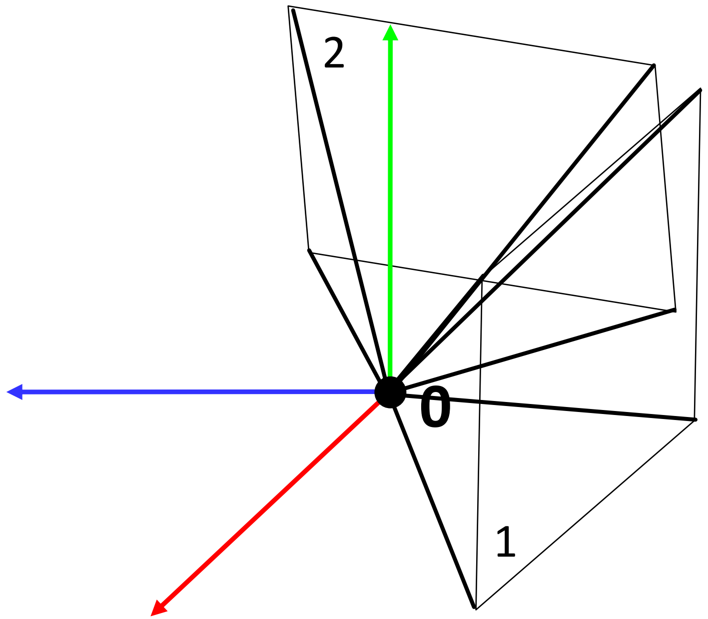


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation $+ \cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths $+ \cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	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	

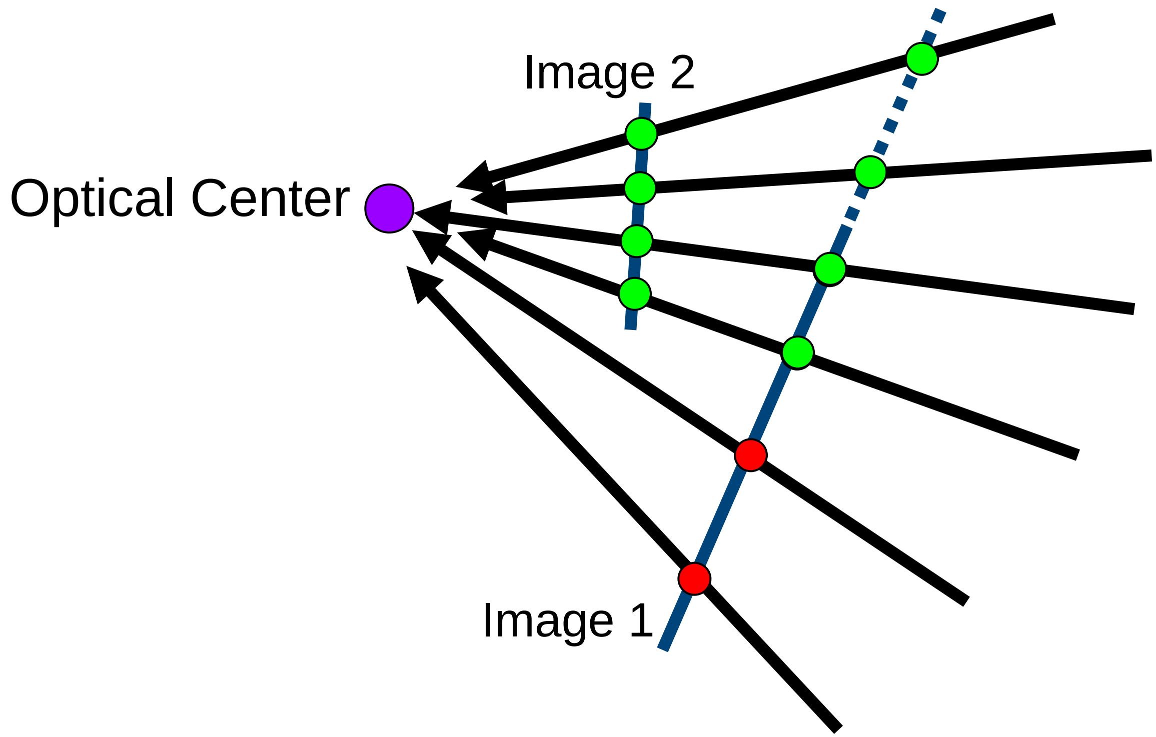
These transformations are a nested set of groups

- Closed under composition and inverse is a member

Geometric interpretation of mosaics

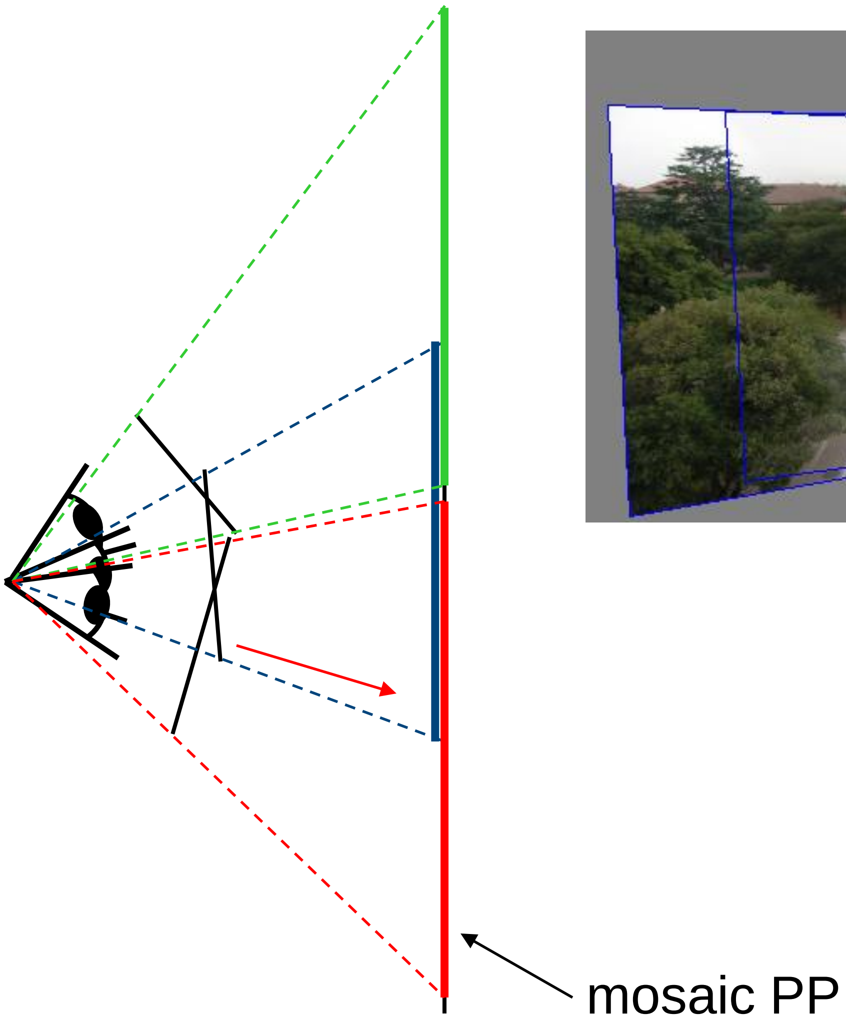


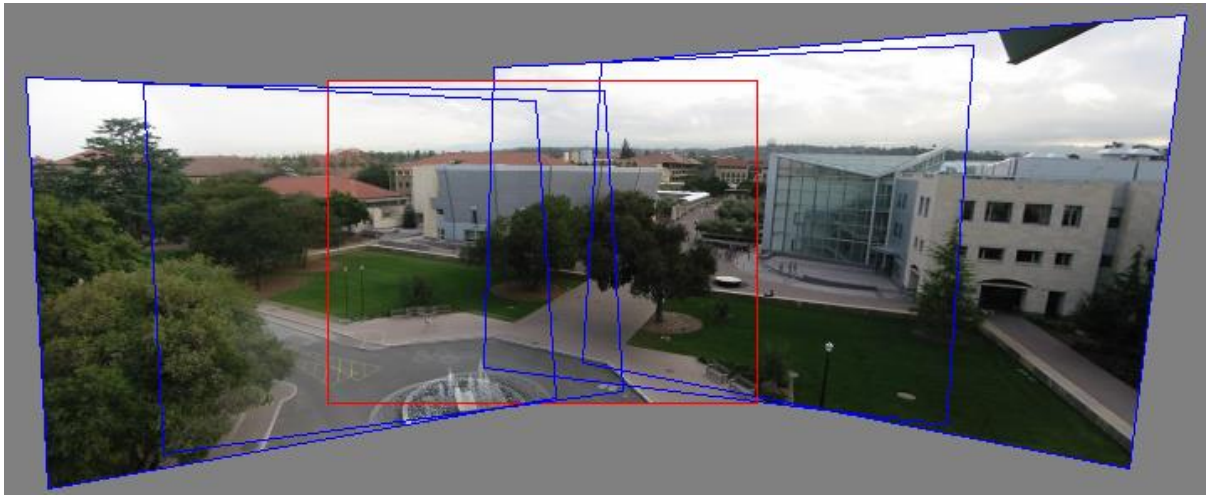
Geometric Interpretation of Mosaics



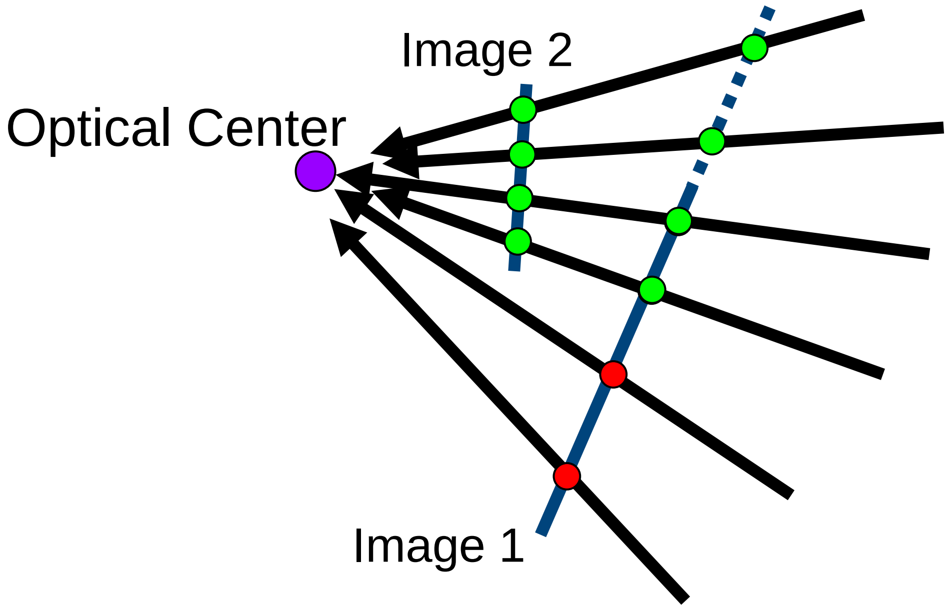
- If we capture all 360º of rays, we can create a 360º panorama
- The basic operation is projecting an image from one plane to another
- The projective transformation is scene-INDEPENDENT
  - This depends on all the images having the same optical center
    - http://archive.bigben.id.au/tutorials/360/photo/nodal.html

Projecting images onto a common plane





What is the transformation?



$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = \mathbf{K}_2^{-1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

3D ray coords (in camera 2) — image coords (in image 2)

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} = \mathbf{R}_1 \mathbf{R}_2^{\mathrm{T}} \mathbf{K}_2^{-1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

3D ray coords (in camera 1) — image coords (in image 2)

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \sim \mathbf{K}_1 \mathbf{R}_1 \mathbf{R}_2^{\mathrm{T}} \mathbf{K}_2^{-1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

image coords (in image 1) — **3x3 homography** — image coords (in image 2)

How do we transform image 2 onto image 1's projection plane?

image 1	image 2
$\mathbf{K}_1$	$\mathbf{K}_2$
$\mathbf{R}_1 = \mathbf{I}_{3\times 3}$	$\mathbf{R}_2$

Image alignment

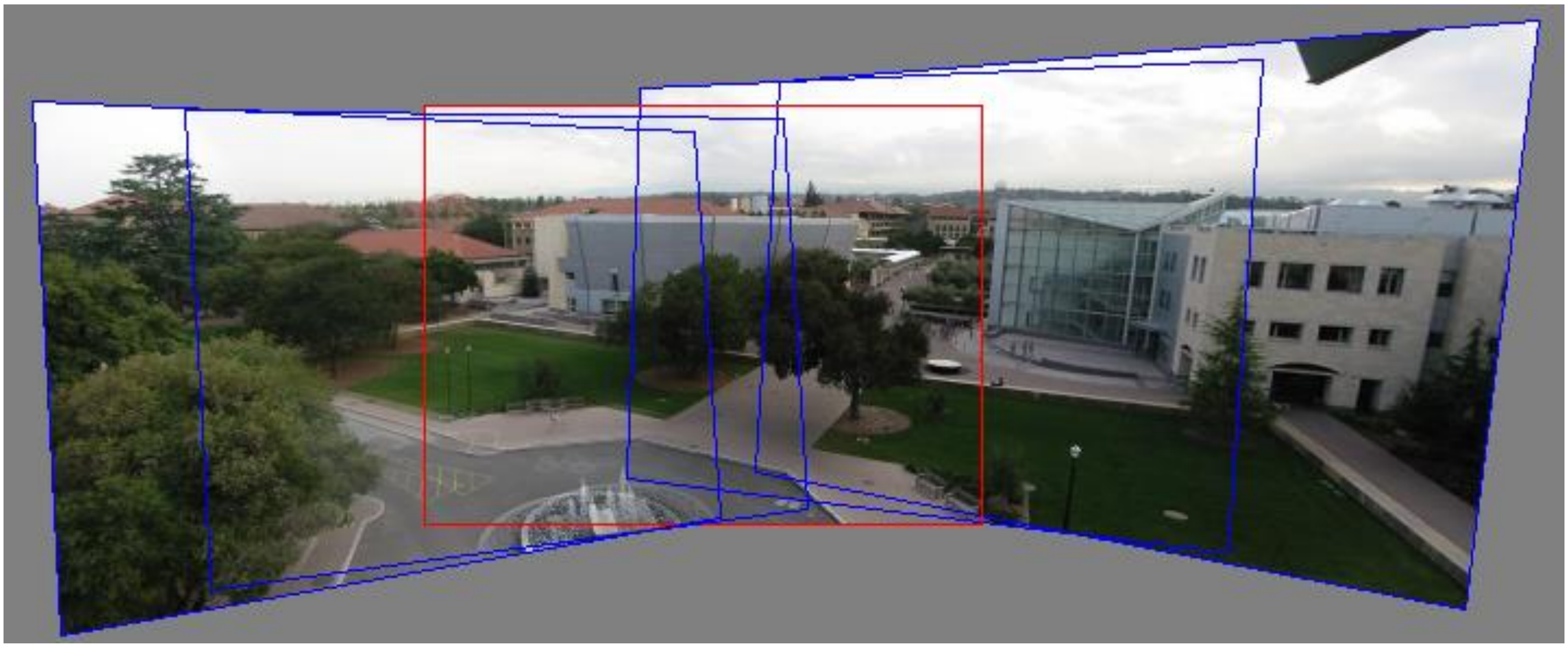
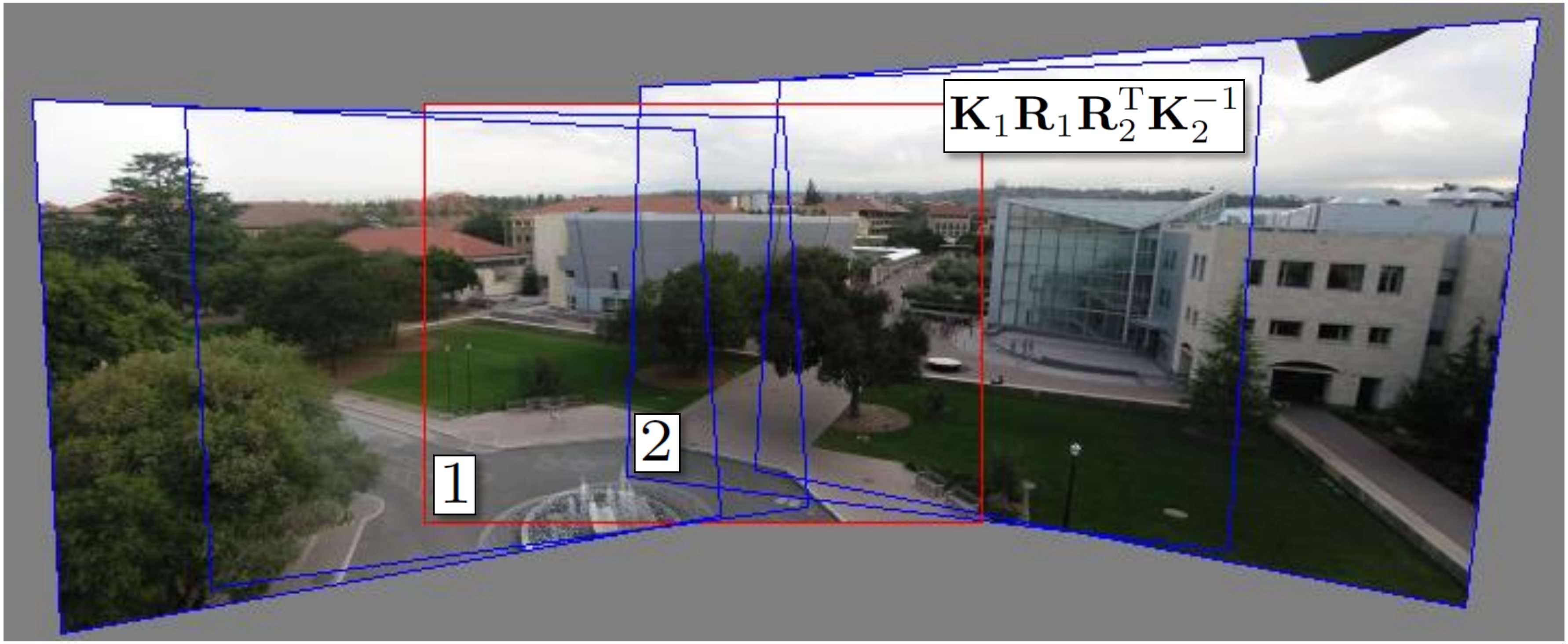
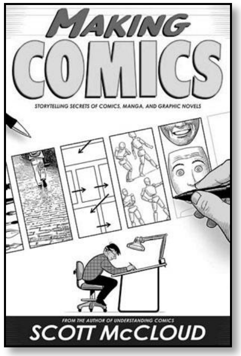


Image alignment



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:

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

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

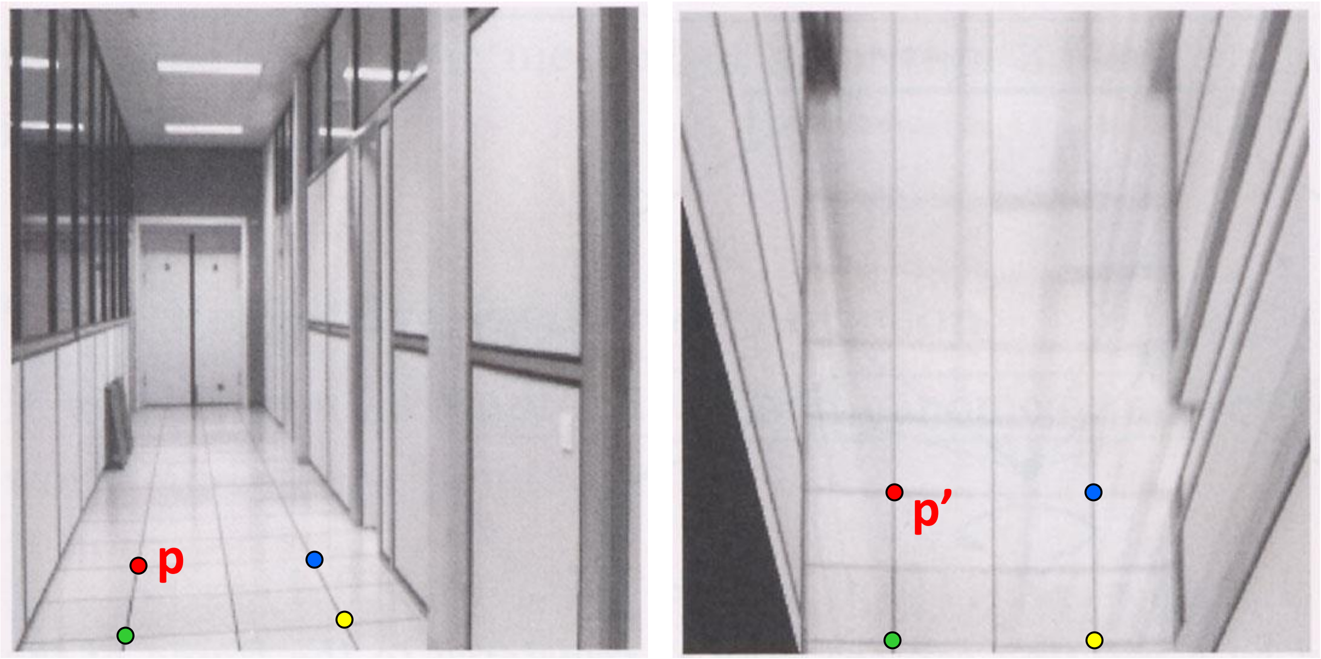
Affine transformations

- Matrix form

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

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

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**?

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 between A and B using least squares on set of matches

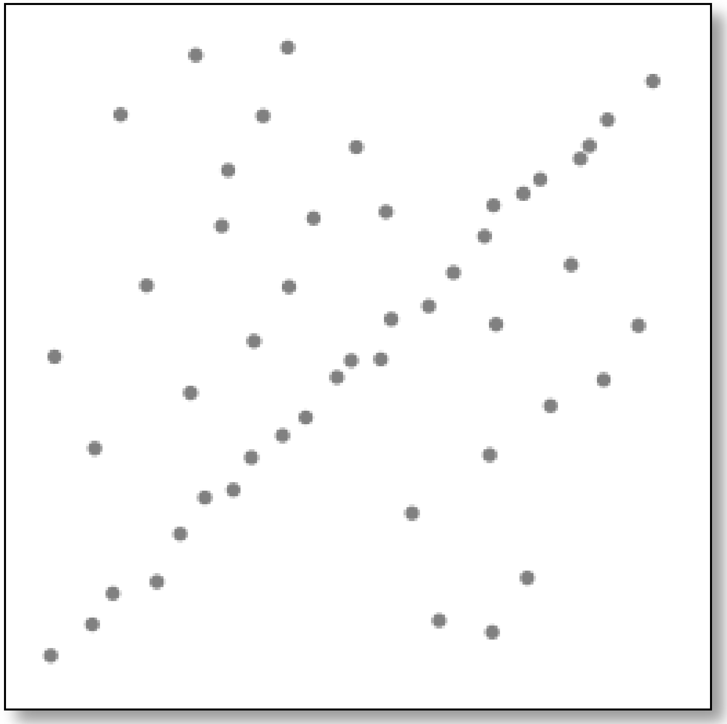
What could go wrong?

Robustness

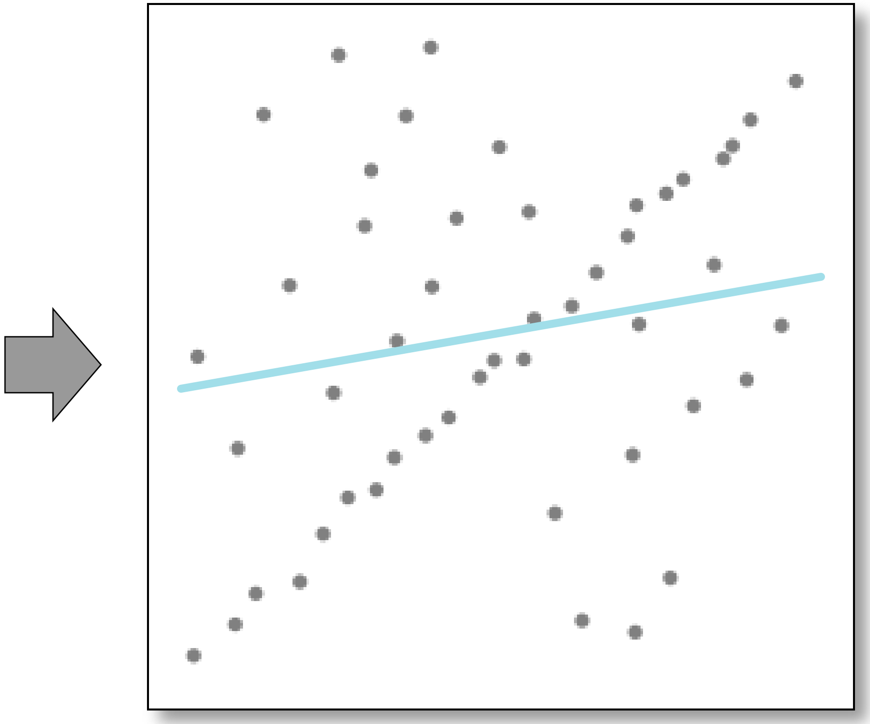

outliers
MAKING COMICS
STORYTELLING SECRETS OF COMICS, MANGA, AND GRAPHIC NOVELS
SCOTT McCLOUD
THE COMPLETE BLACK ADDER
SOUTH WEST

Robustness

- Let's consider a simpler example…



Problem: Fit a line to these datapoints



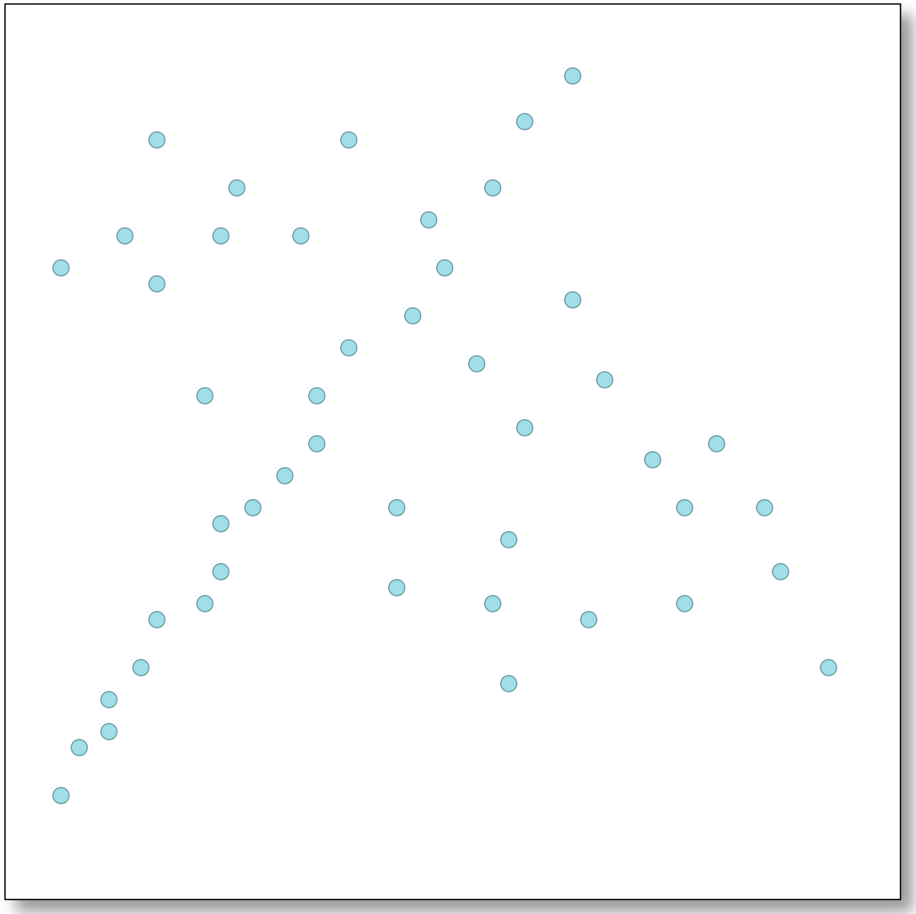
Least squares fit

Idea

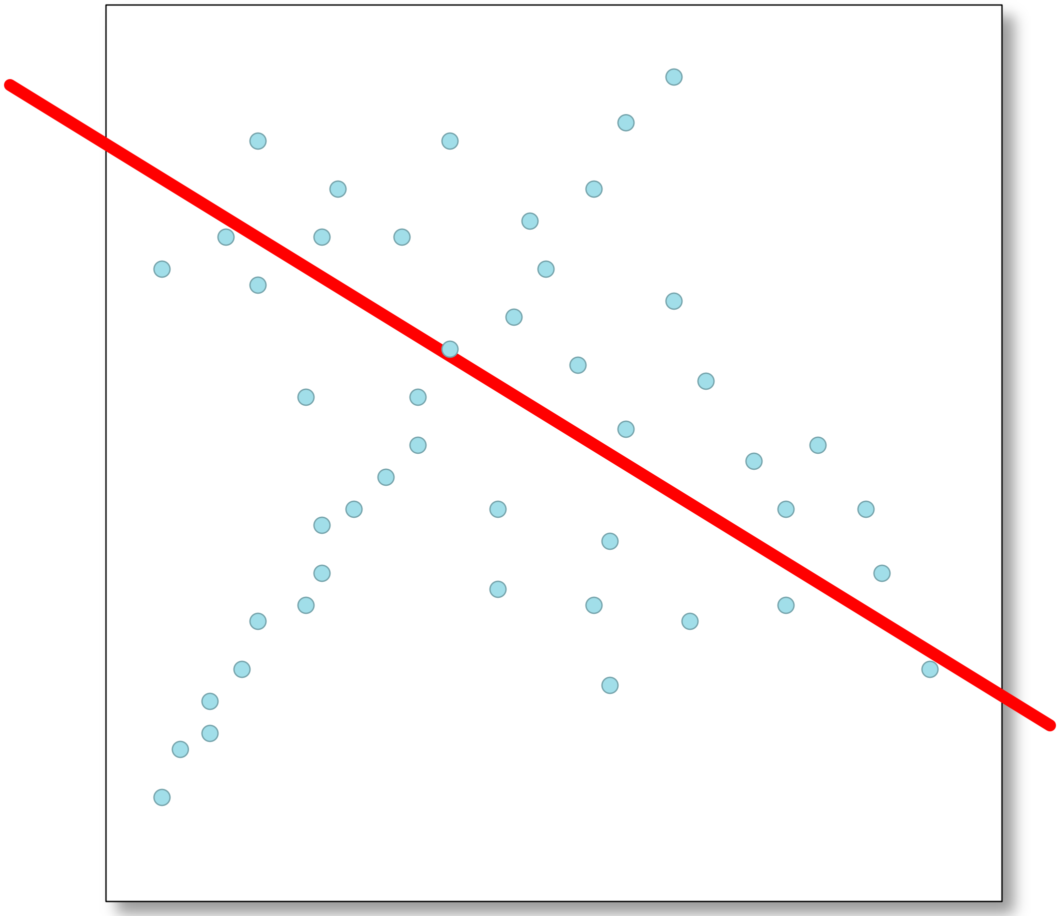
- Given a hypothesized line
- Count the number of points that “agree” with the line
  - “Agree” = within a small distance of the line
  - I.e., the **inliers** to that line

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

Counting inliers

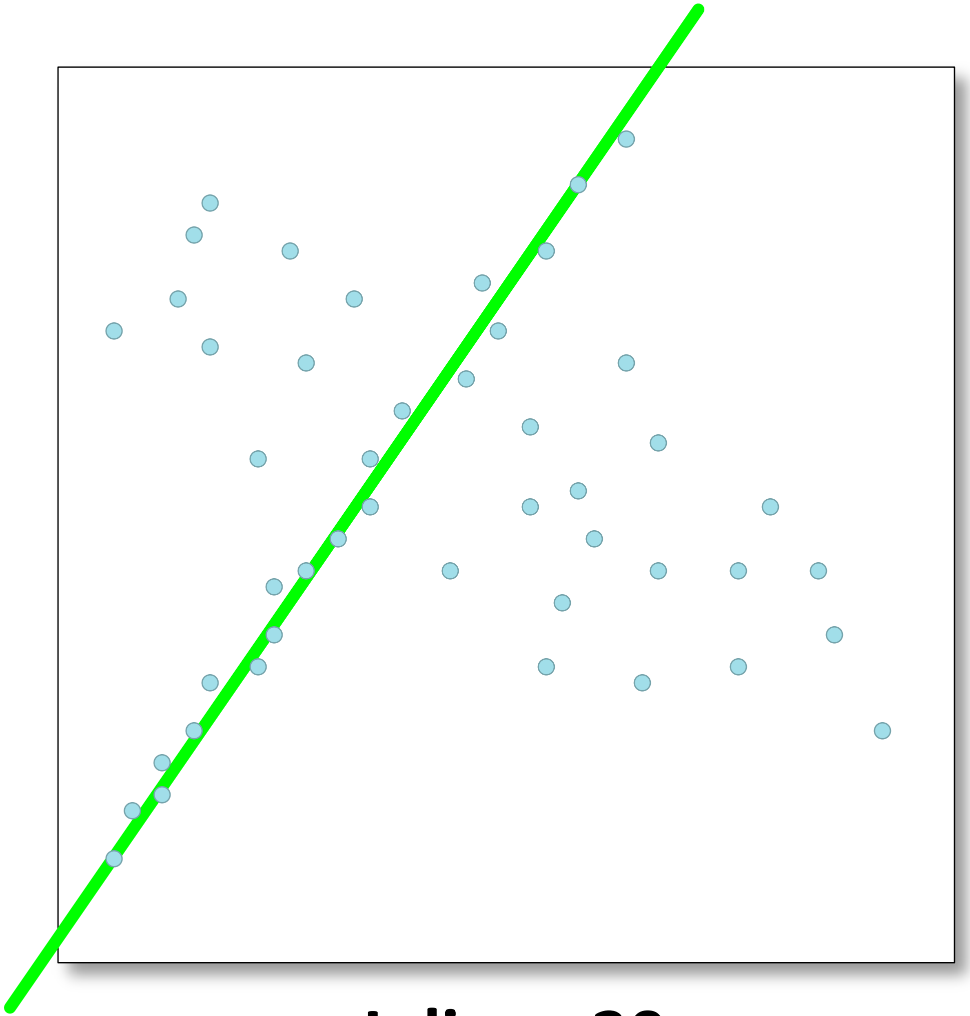


Counting inliers



Inliers: 3

Counting inliers

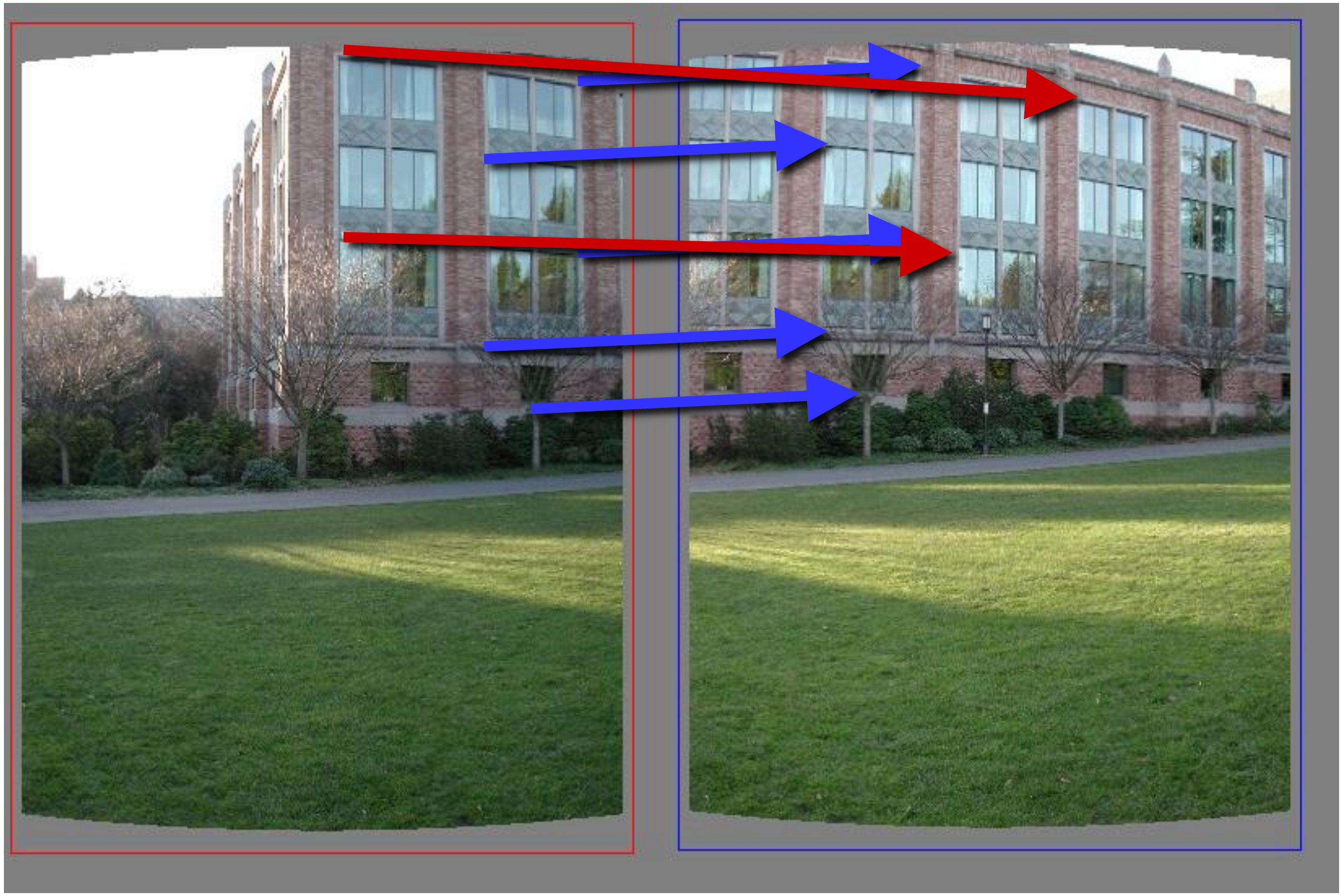


Inliers: 20

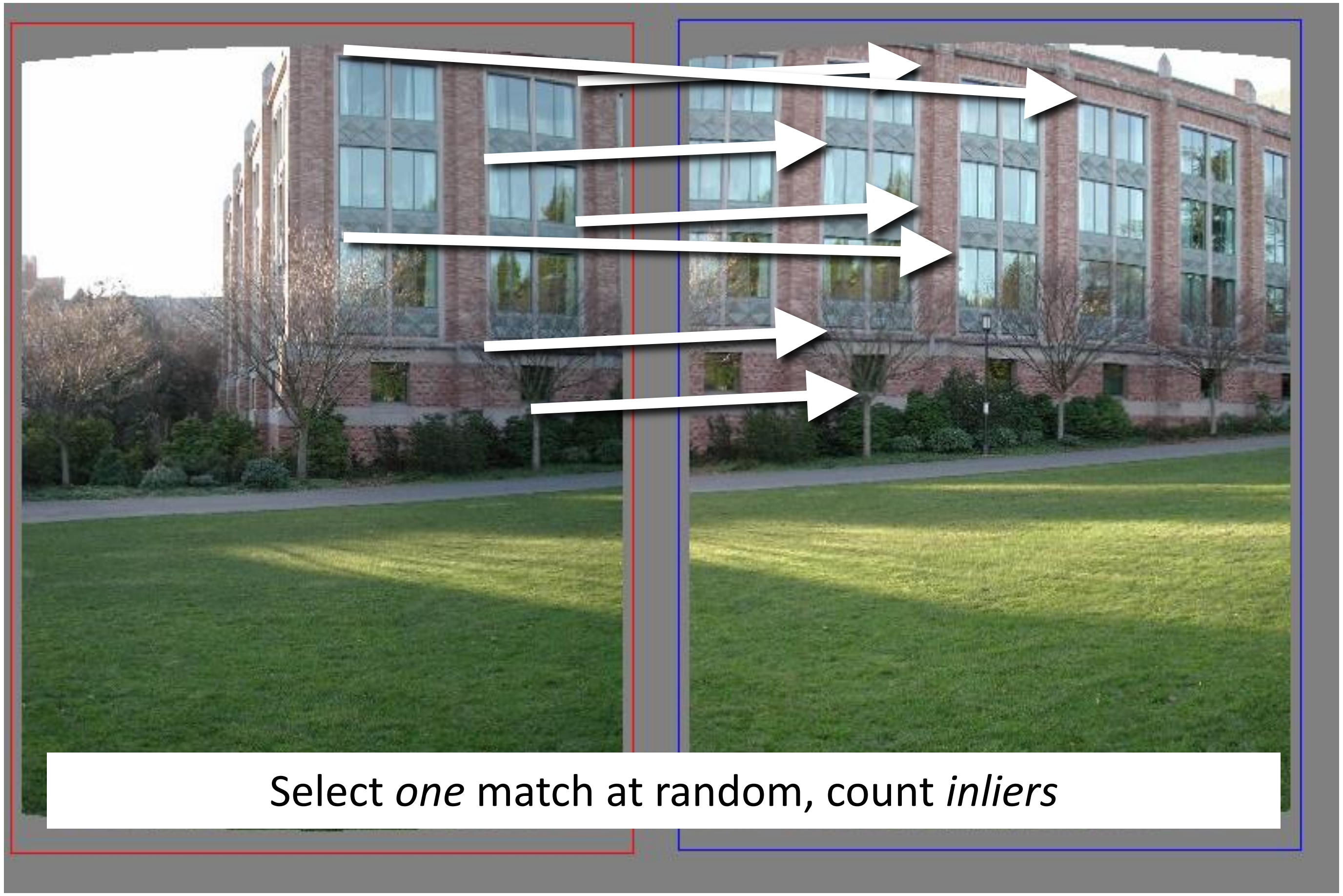
How do we find the best line?

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

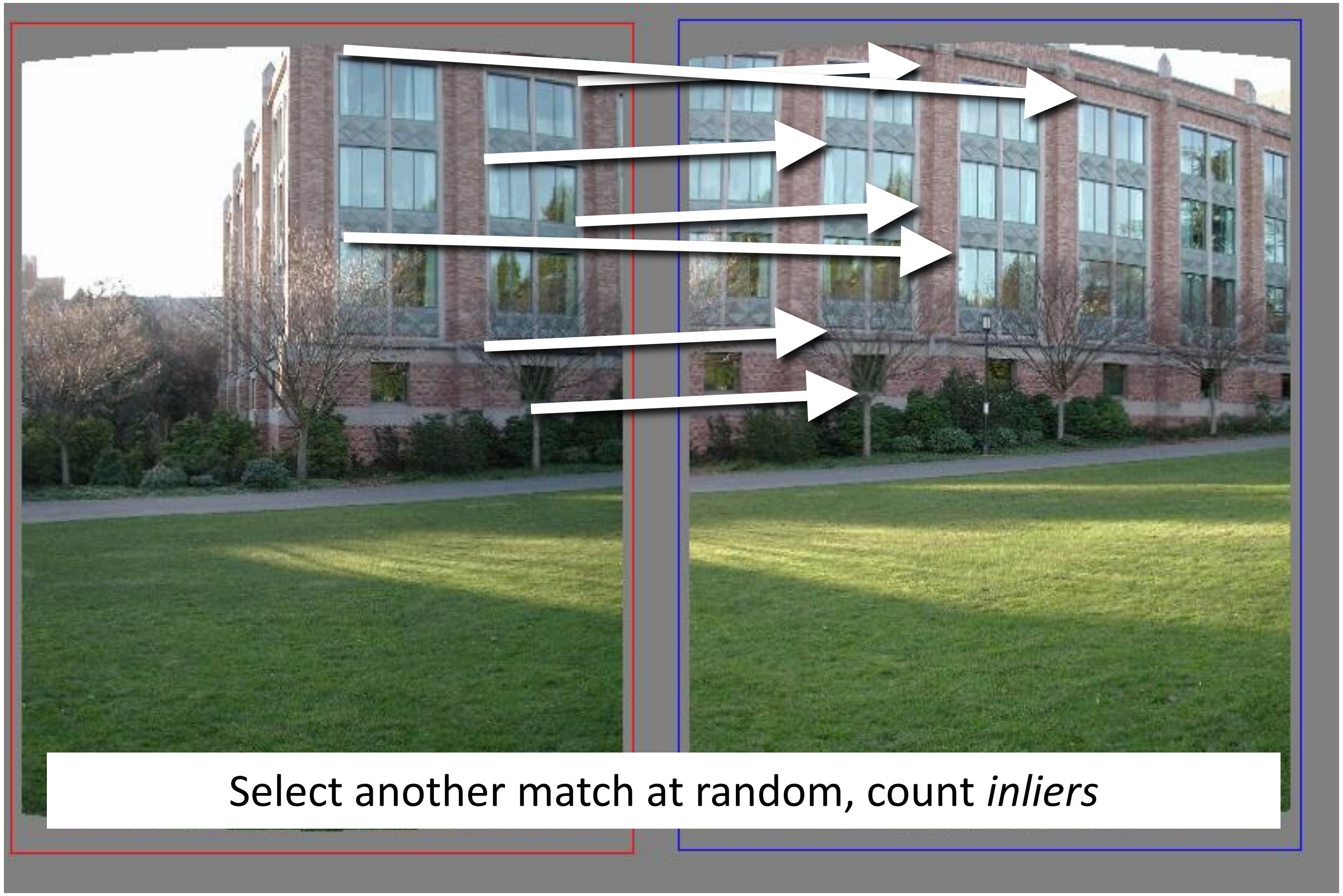
Translations



RAndom SAmple Consensus

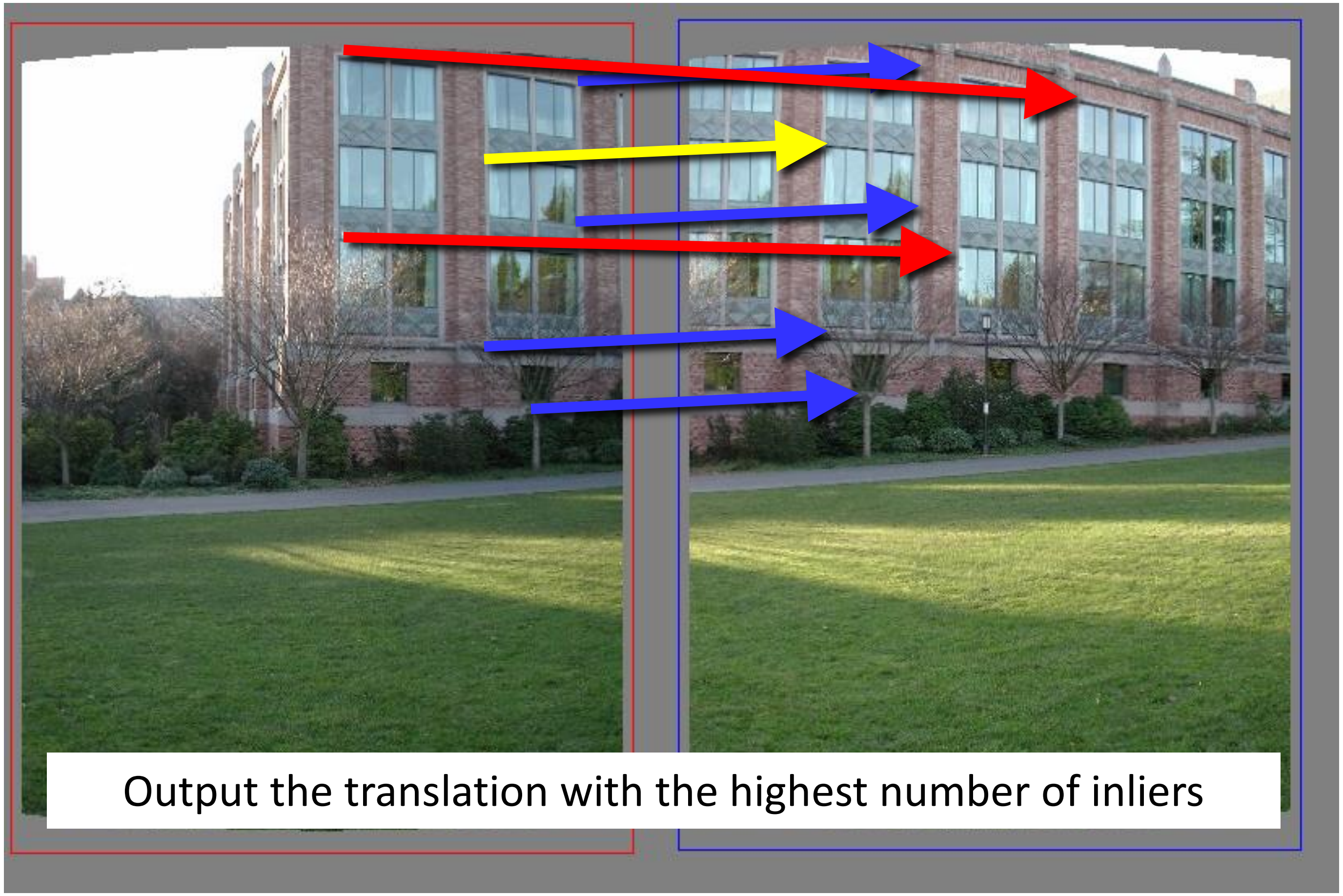


RAndom SAmple Consensus



Select another match at random, count *inliers*

RAndom SAmple Consensus



Output the translation with the highest number of inliers

RANSAC

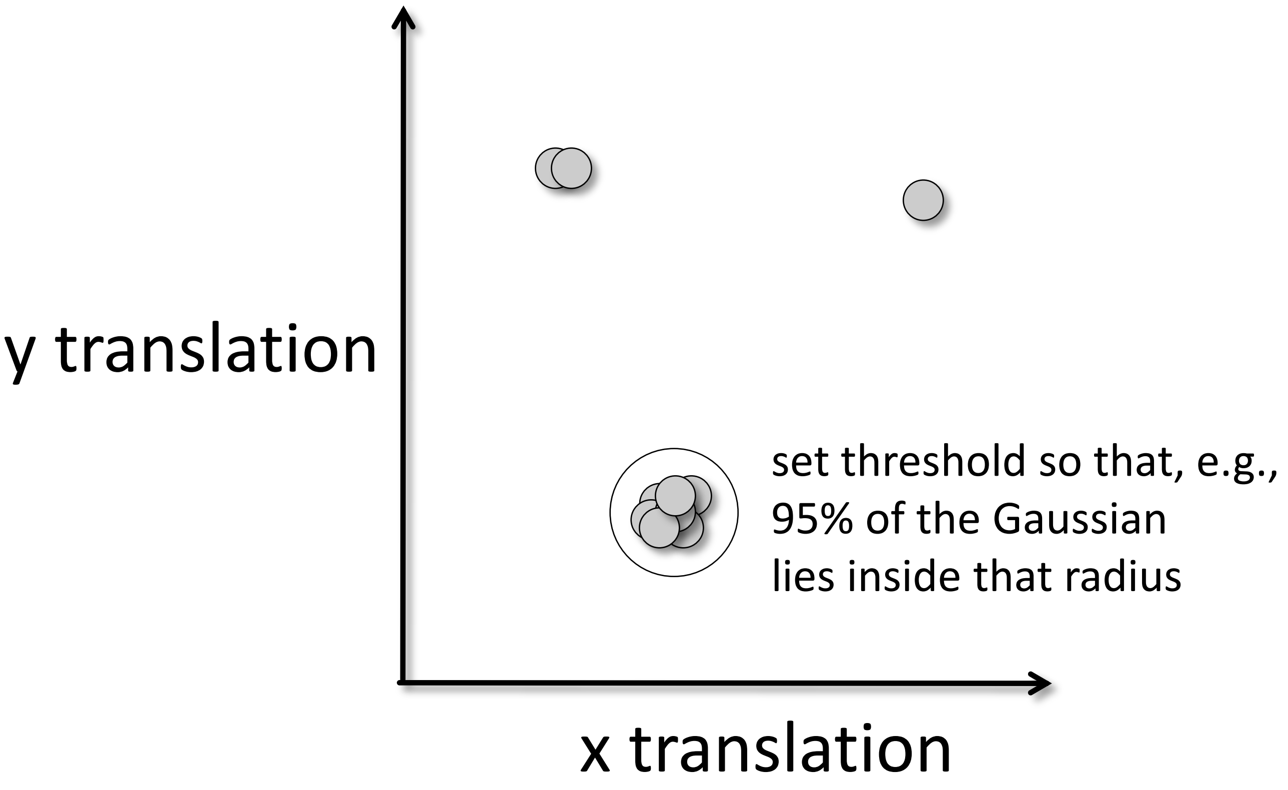
- Idea:
  - All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
    - RANSAC only has guarantees if there are < 50% outliers
  - “All good matches are alike; every bad match is bad in its own way.”

– 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?

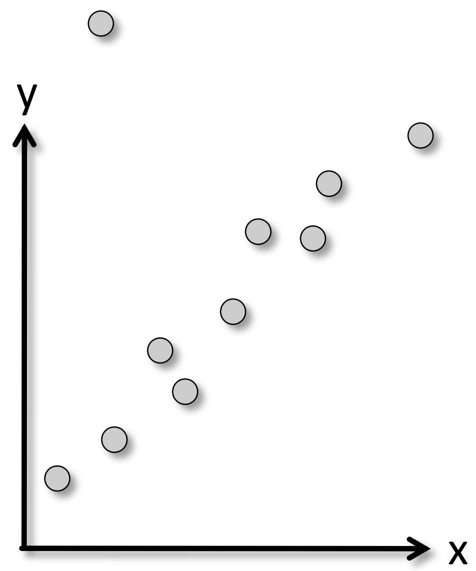
RANSAC



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

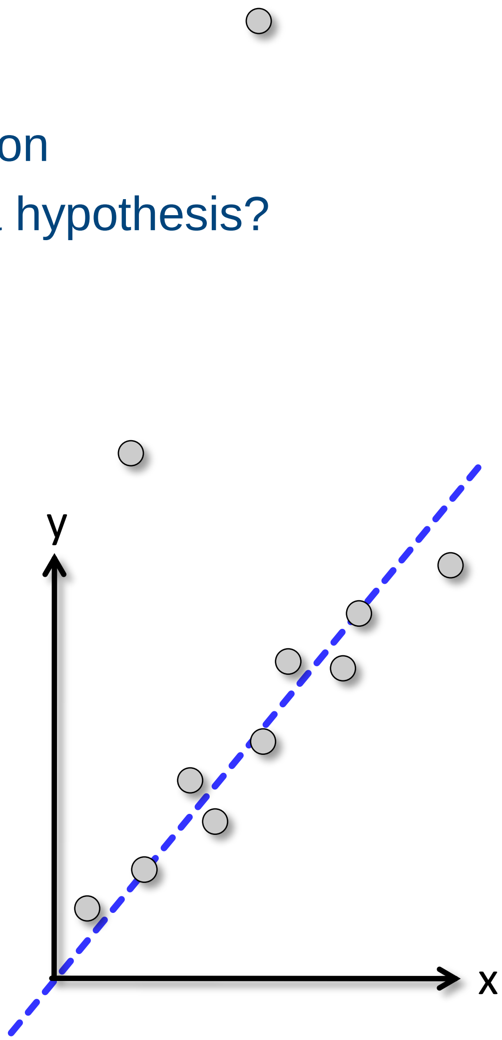
RANSAC

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



RANSAC

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



RANSAC

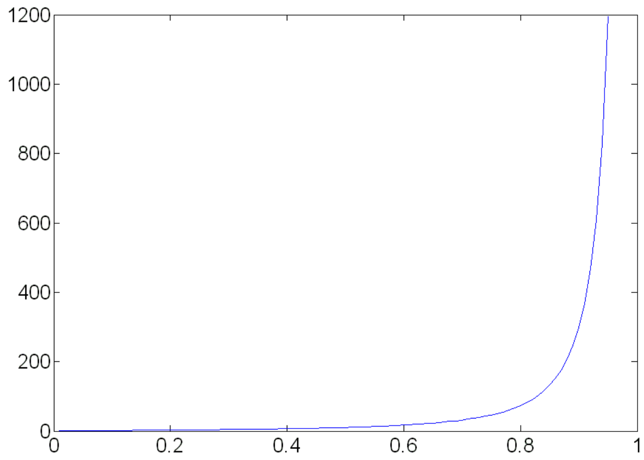
- General version:
  1. Randomly choose *s* samples
     - Typically *s* = minimum sample size that lets you fit a model
  2. Fit a model (e.g., line) to those samples
  3. Count the number of inliers that approximately fit the model
  4. Repeat *N* times
  5. Choose the model that has the largest set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

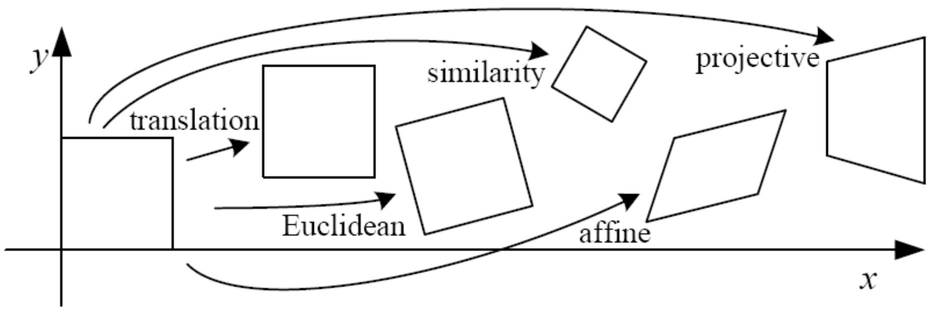
	proportion of outliers *e*						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

$p = 0.99$



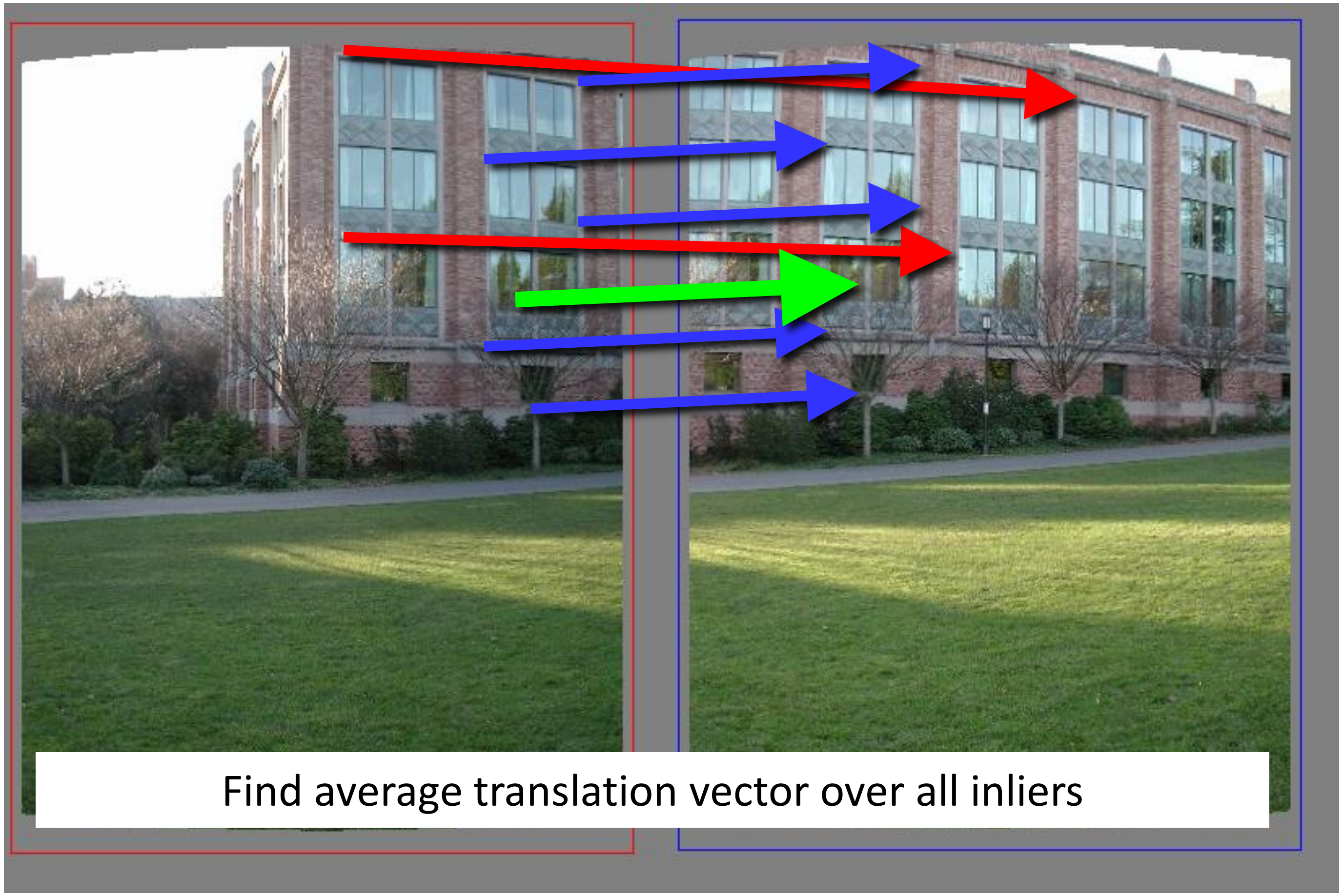
How big is *s*?

- 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	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation $+\cdots$	□
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths $+\cdots$	◇
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	4	angles $+\cdots$	◇
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times 3}$	6	parallelism $+\cdots$	▱
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times 3}$	8	straight lines	⏢

Final step: least squares fit



Choosing the parameters

- Initial number of points *s*
  - Typically minimum number needed to fit the model
- Distance threshold *t*
  - Choose *t* so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples *N*
  - Choose *N* so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)

$$\left(1-\left(1-e\right)^s\right)^N = 1-p$$

$$N = \log\left(1-p\right)/\log\left(1-\left(1-e\right)^s\right)$$

	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

RANSAC conclusions

Good

- Robust to outliers
- Applicable for larger number of parameters than Hough transform
- Parameters are easier to choose than Hough transform

Bad

- Computational time grows quickly with fraction of outliers and number of parameters
- Not good for getting multiple fits

Common applications

- Computing a homography (e.g., image stitching)
- Estimating fundamental matrix (relating two views)

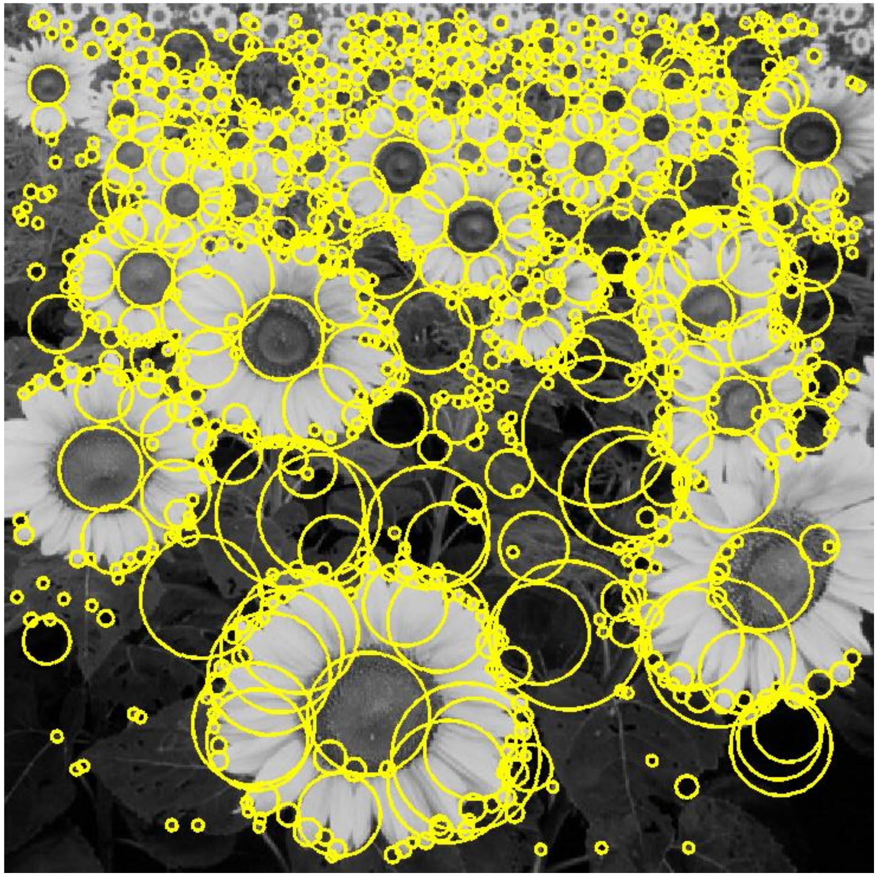
VLFeat demo of Ransac Homography fit

335 tentative matches

209 (62.39%) inliner matches out of 335

Mosaic

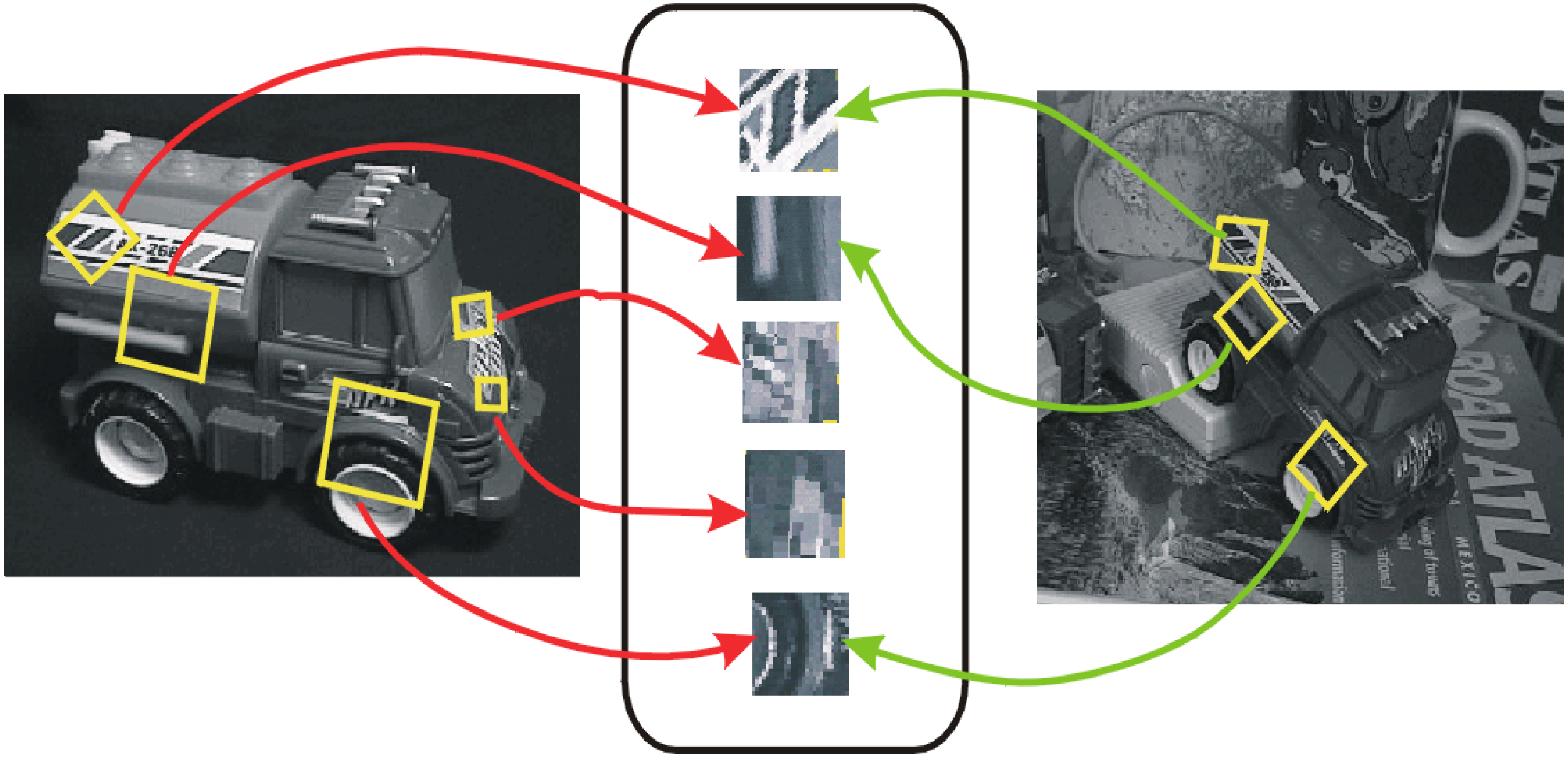
Feature extraction: Corners and blobs



Invariant local features

Find features that are invariant to transformations

- geometric invariance: translation, rotation, scale
- photometric invariance: brightness, exposure, …



Advantages of local features

Locality

– features are local, so robust to occlusion and clutter

Quantity

– hundreds or thousands in a single image

Distinctiveness:

– can differentiate a large database of objects

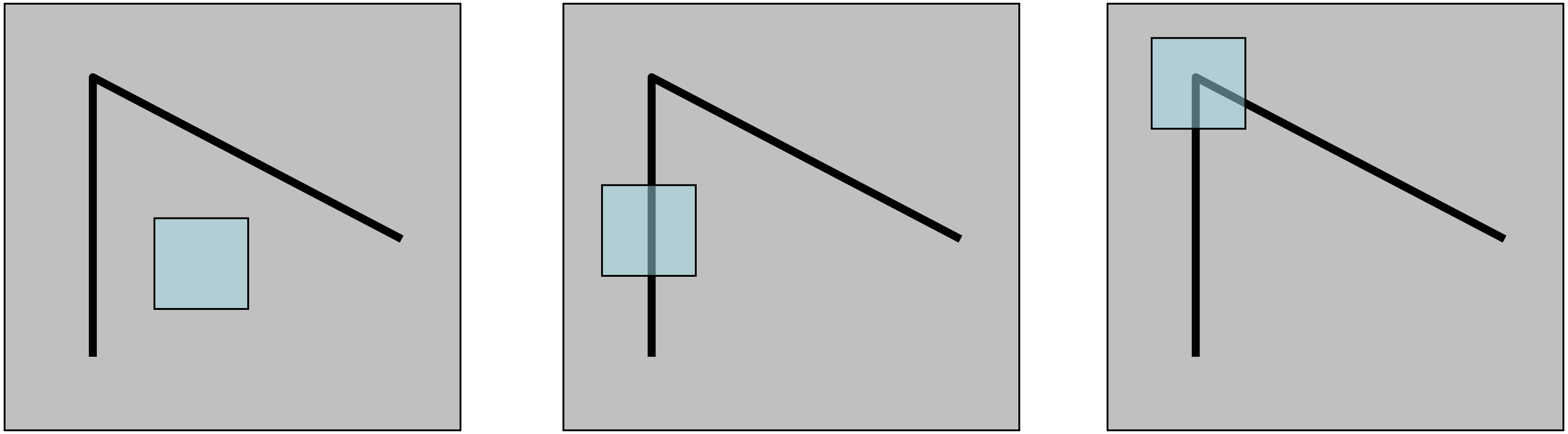
Efficiency

– real-time performance achievable

Local measures of uniqueness

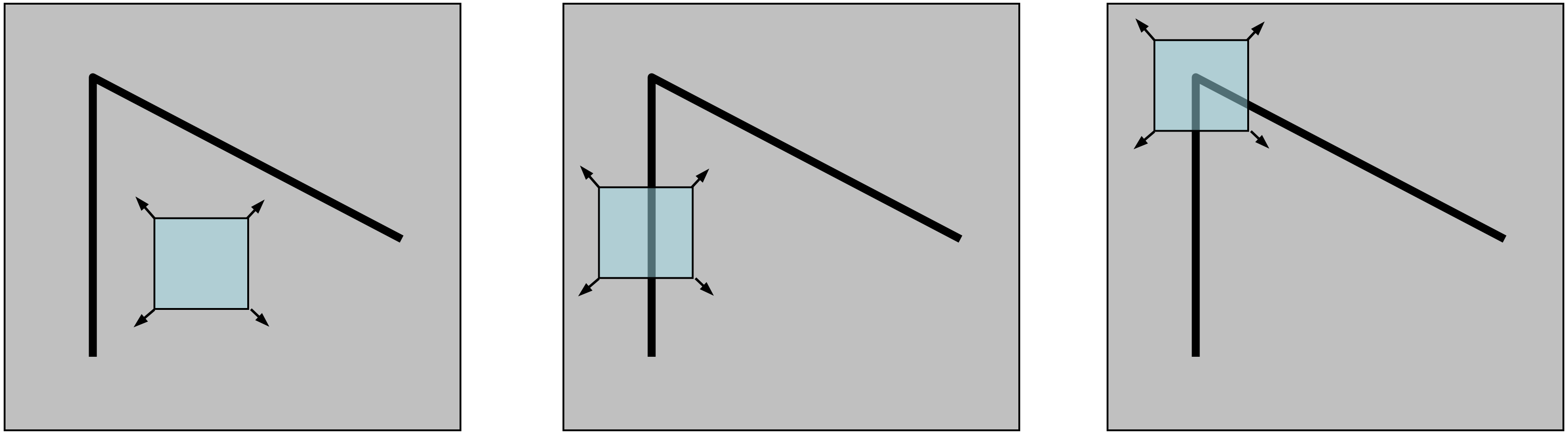
Suppose we only consider a small window of pixels

– What defines whether a feature is a good or bad candidate?



Local measure of feature uniqueness

- How does the window change when you shift it?
- Shifting the window in any direction causes a big change



"flat" region:
no change in all directions

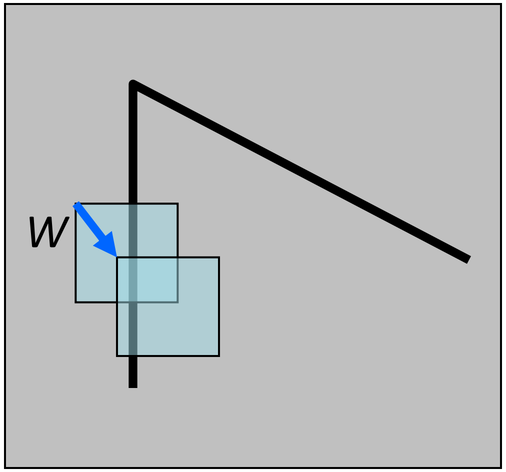
"edge":
no change along the edge direction

"corner":
significant change in all directions

Harris corner detection: the math

Consider shifting the window *W* by (*u,v*)

- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD "error" *E*(*u,v*):



$$E(u,v) = \sum_{(x,y)\in W} \left[I(x+u,y+v) - I(x,y)\right]^2$$

Small motion assumption

Taylor Series expansion of *I*:

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

If the motion (u,v) is small, then first order approximation is good

$$\begin{aligned} I(x+u,y+v) &\approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x,y) + [I_x \;\; I_y]\begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

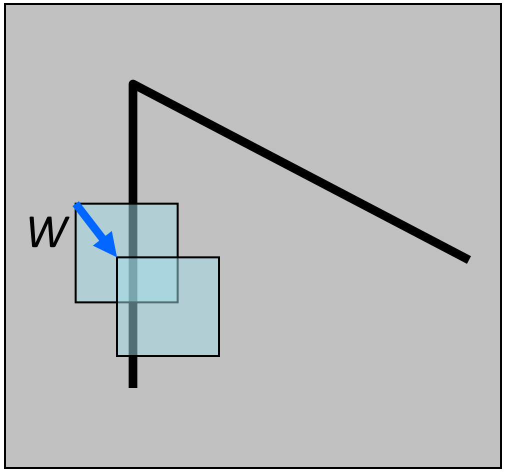
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Corner detection: the math

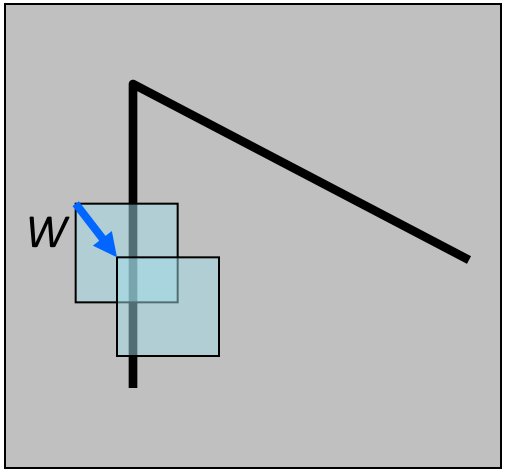
Consider shifting the window *W* by (*u,v*)

- define an SSD “error” *E(u,v)*:



$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} \left[I(x+u,y+v) - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I(x,y) + I_x u + I_y v - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I_x u + I_y v\right]^2
\end{aligned}
$$

Corner detection: the math



Consider shifting the window *W* by (*u*,*v*)

- define an SSD "error" *E(u,v)*:

$$E(u,v) \approx \sum_{(x,y)\in W} [I_x u + I_y v]^2$$

$$\approx Au^2 + 2Buv + Cv^2$$

$$A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_x I_y \qquad C = \sum_{(x,y)\in W} I_y^2$$

- Thus, *E*(*u*,*v*) is locally approximated as a quadratic error function

The second moment matrix

The surface *E*(*u*,*v*) is locally approximated by a quadratic form.

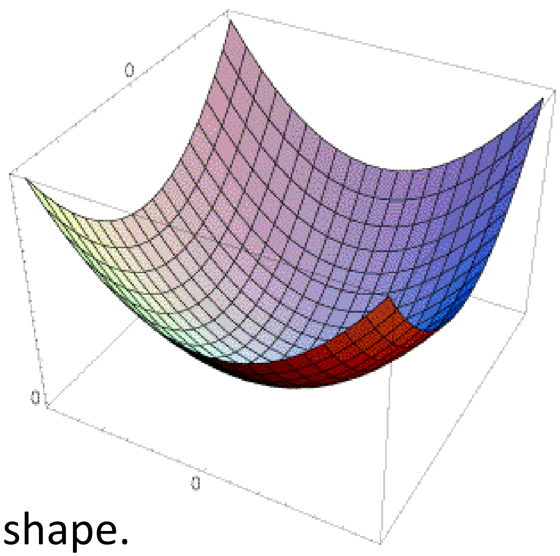
$$E(u,v) \approx Au^2 + 2Buv + Cv^2$$

$$\approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



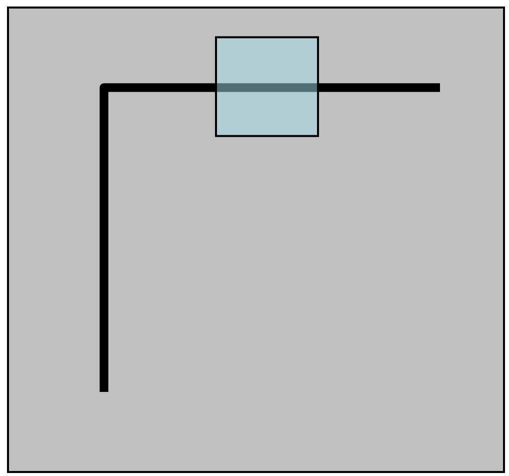
Let’s try to understand its shape.

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

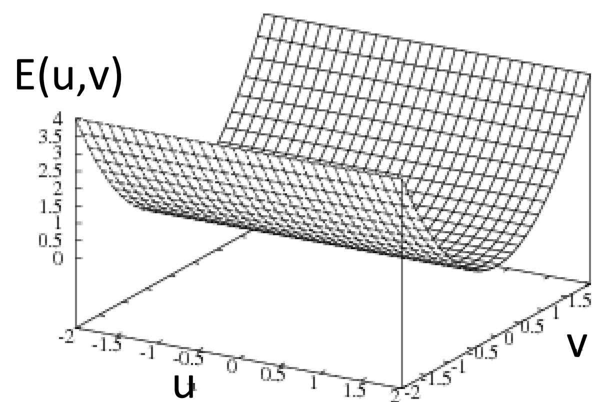
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Horizontal edge: $I_x = 0$

$$H = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix}$$

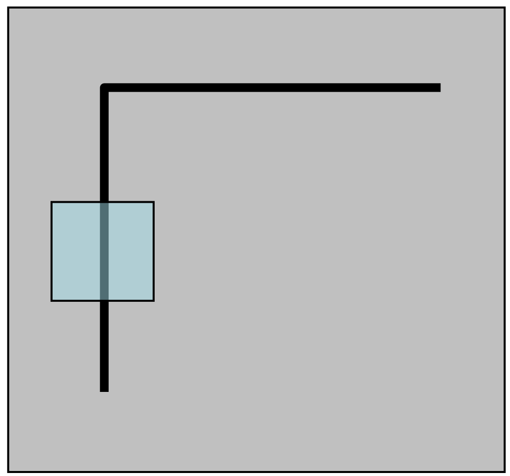


$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

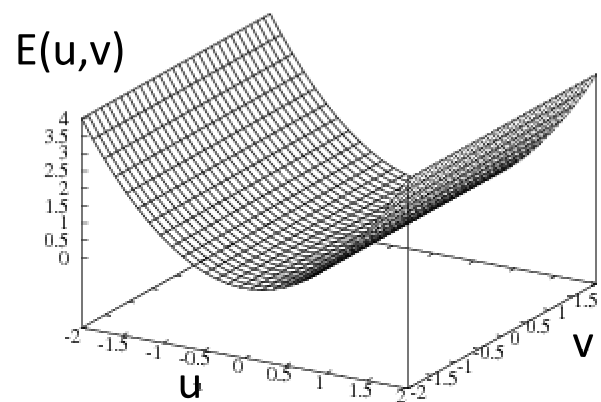
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Vertical edge: $I_y = 0$

$$H = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$$

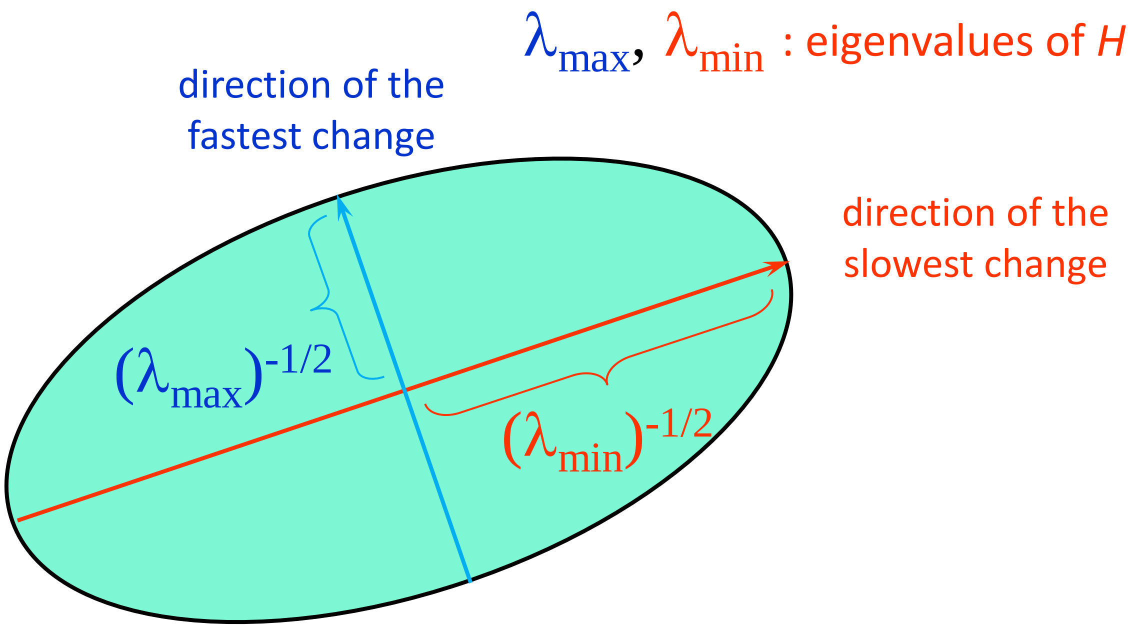


General case

We can visualize *H* as an ellipse with axis lengths determined by the *eigenvalues* of *H* and orientation determined by the *eigenvectors* of *H*

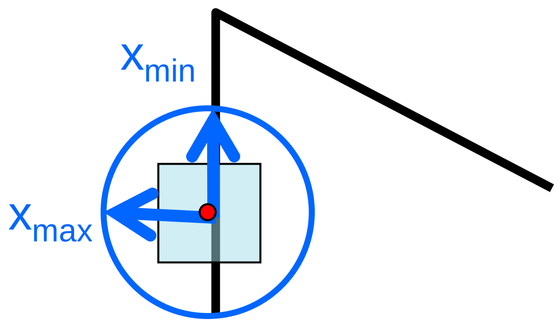
Ellipse equation:

$$[u \;\; v] \; H \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



Corner detection: the math

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$



$$Hx_{\max} = \lambda_{\max} x_{\max}$$

$$Hx_{\min} = \lambda_{\min} x_{\min}$$

Eigenvalues and eigenvectors of H

- Define shift directions with the smallest and largest change in error
- x_{max} = direction of largest increase in *E*
- λ_{max} = amount of increase in direction x_{max}
- x_{min} = direction of smallest increase in *E*
- λ_{min} = amount of increase in direction x_{min}

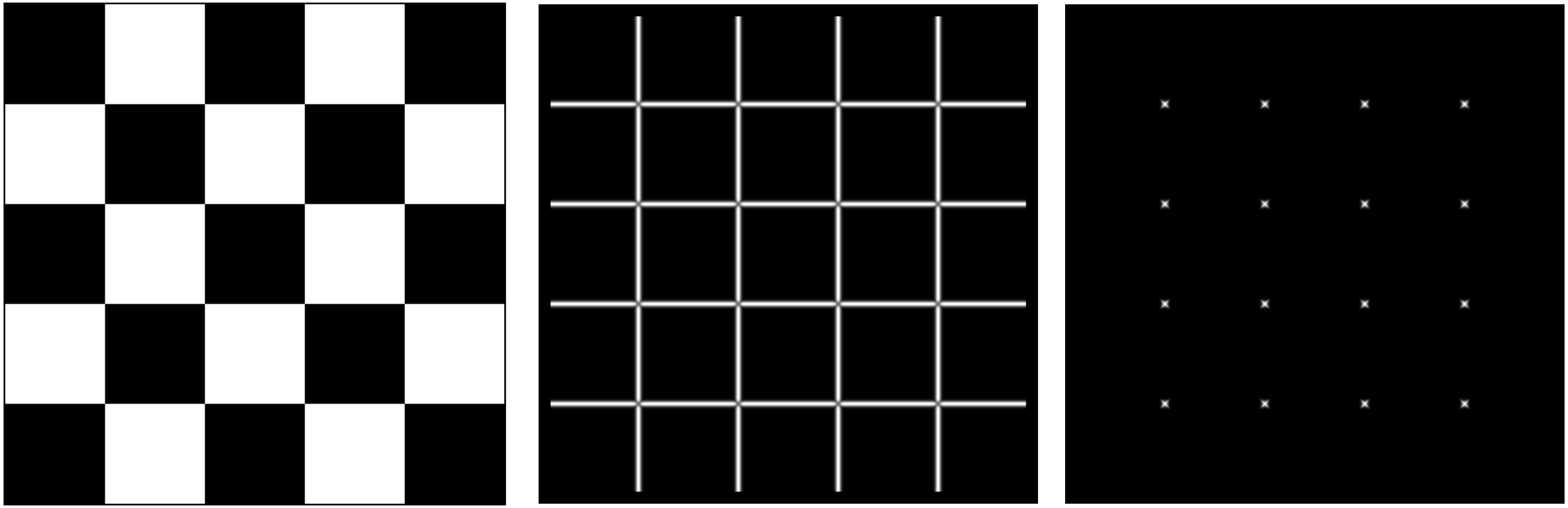
Corner detection: the math

How are λ_{max}, x_{max}, λ_{min}, and x_{min} relevant for feature detection?

- What's our feature scoring function?

Want $E(u,v)$ to be large for small shifts in all directions

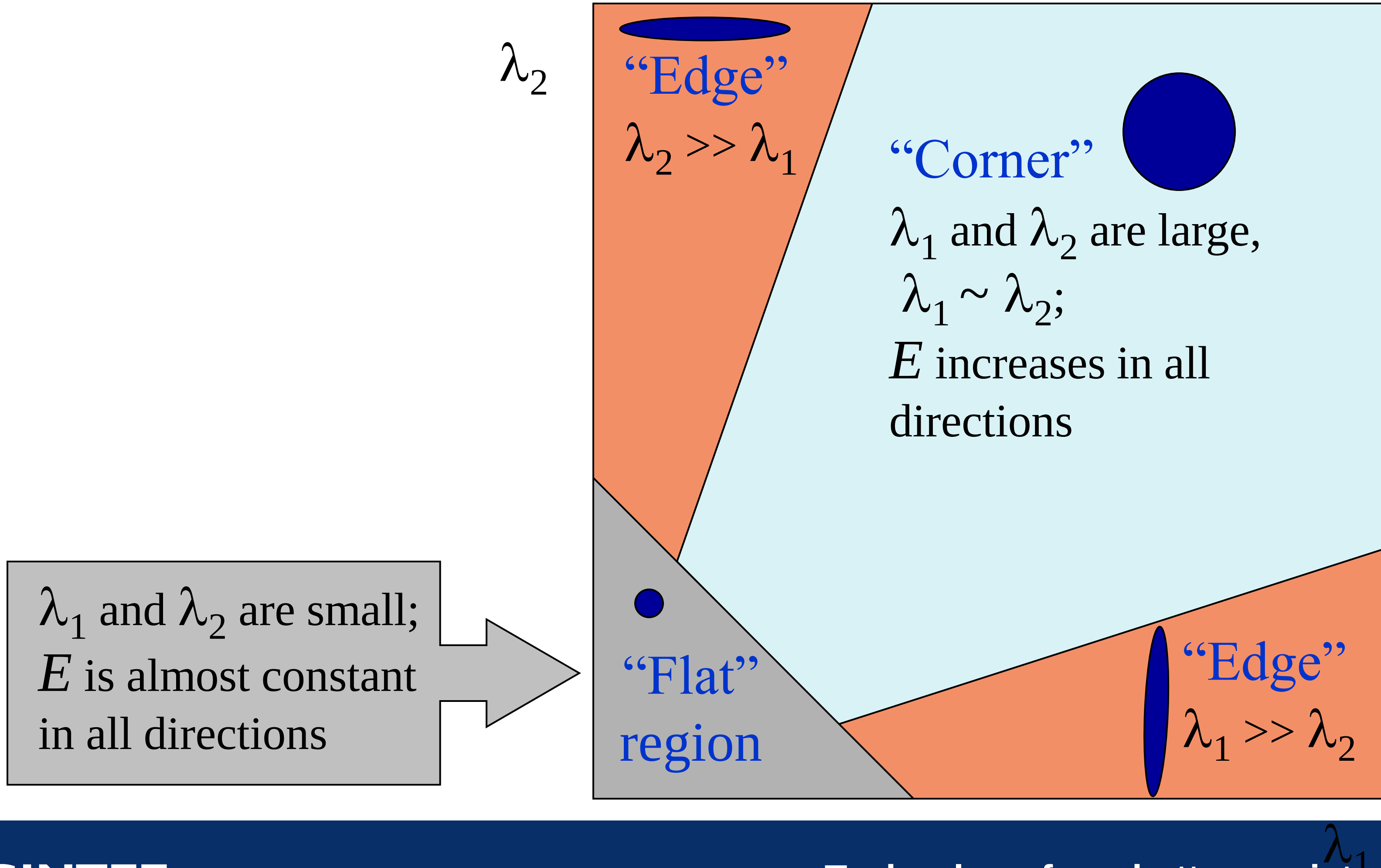
- the minimum of $E(u,v)$ should be large, over all unit vectors $[u\ v]$
- this minimum is given by the smaller eigenvalue (λ_{min}) of H



I λ_{max} λ_{min}

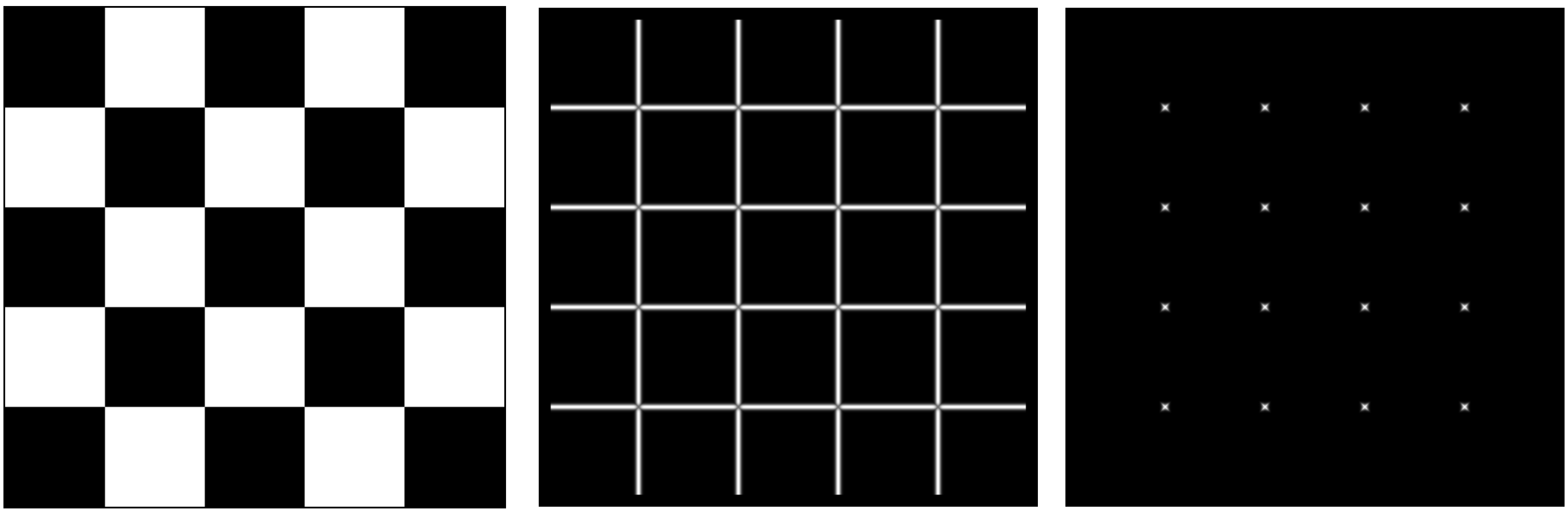
Interpreting the eigenvalues

Classification of image points using eigenvalues of *M*:



Corner detection summary

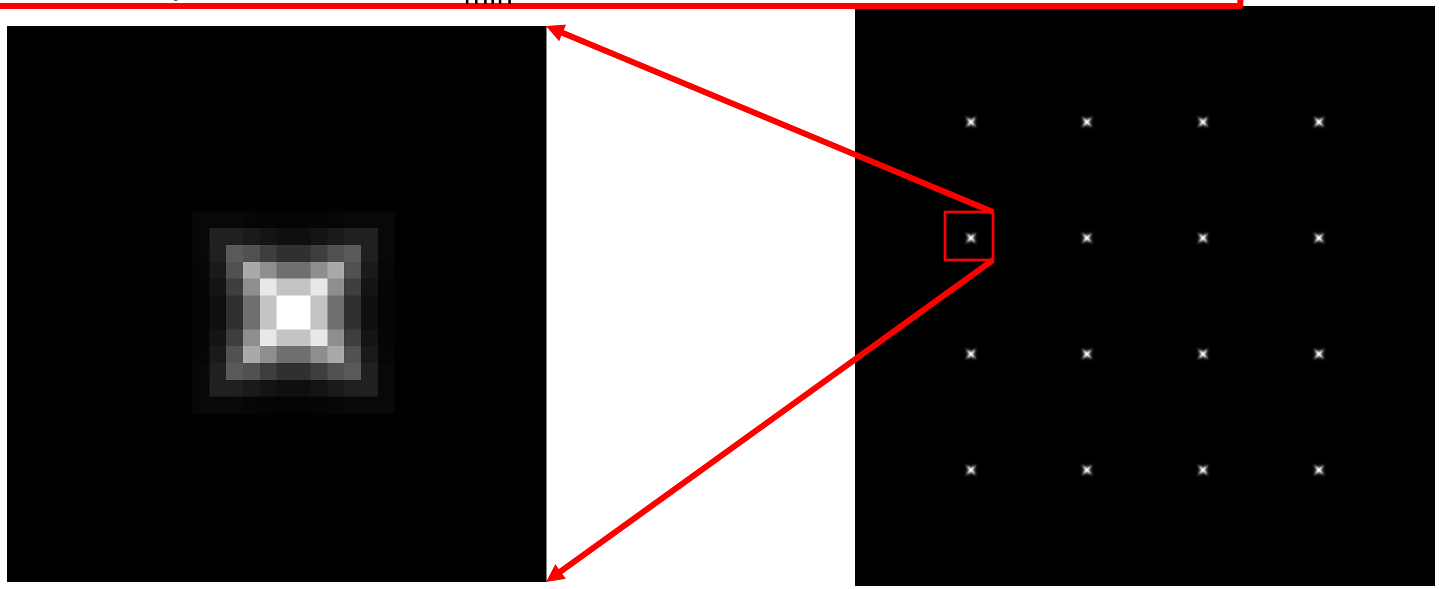
- Compute the gradient at each point in the image
- Create the *H* matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_{min} > threshold)
- Choose those points where λ_{min} is a local maximum as features



I λ_{max} λ_{min}

Corner detection summary

- Compute the gradient at each point in the image
- Create the *H* matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_{min} > threshold)
- Choose those points where λ_{min} is a local maximum as features



λ_{min}

The Harris operator

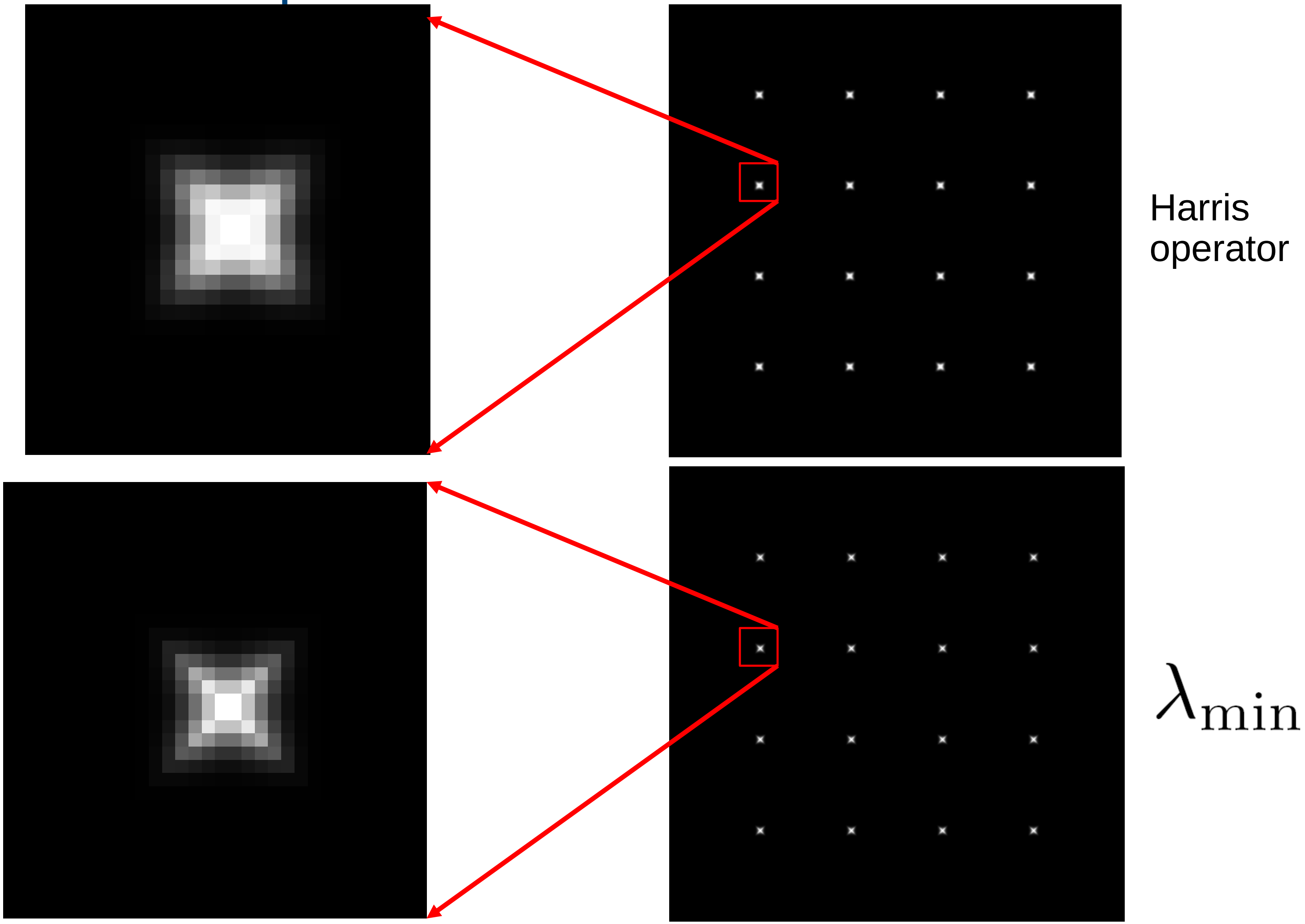
λ_{min} is a variant of the "Harris operator" for feature detection

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

$$= \frac{determinant(H)}{trace(H)}$$

- The *trace* is the sum of the diagonals, i.e., $trace(H) = h_{11} + h_{22}$
- Very similar to λ_{min} but less expensive (no square root)
- Called the "Harris Corner Detector" or "Harris Operator"
- Lots of other detectors, this is one of the most popular

The Harris operator



Harris Detector – Responses [Harris88]

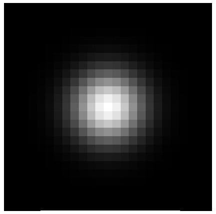
Weighting the derivatives

- In practice, using a simple window *W* doesn't work too well

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

- Instead, we'll *weight* each derivative value based on its distance from the center pixel

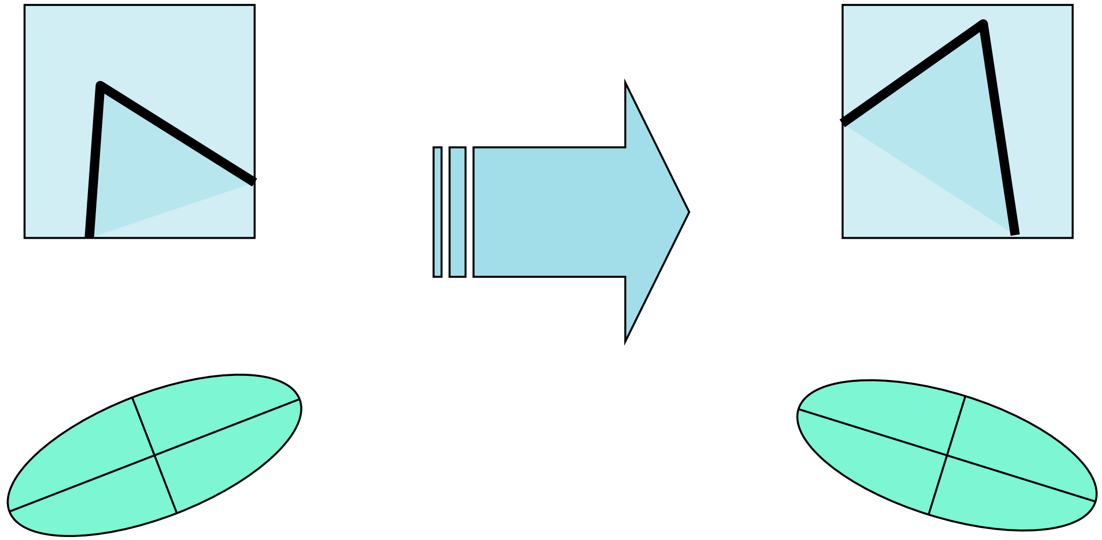
$$H = \sum_{(x,y)\in W} w_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



$w_{x,y}$

Harris Detector: Invariance Properties

- Rotation



Ellipse rotates but its shape (i.e. eigenvalues) remains the same

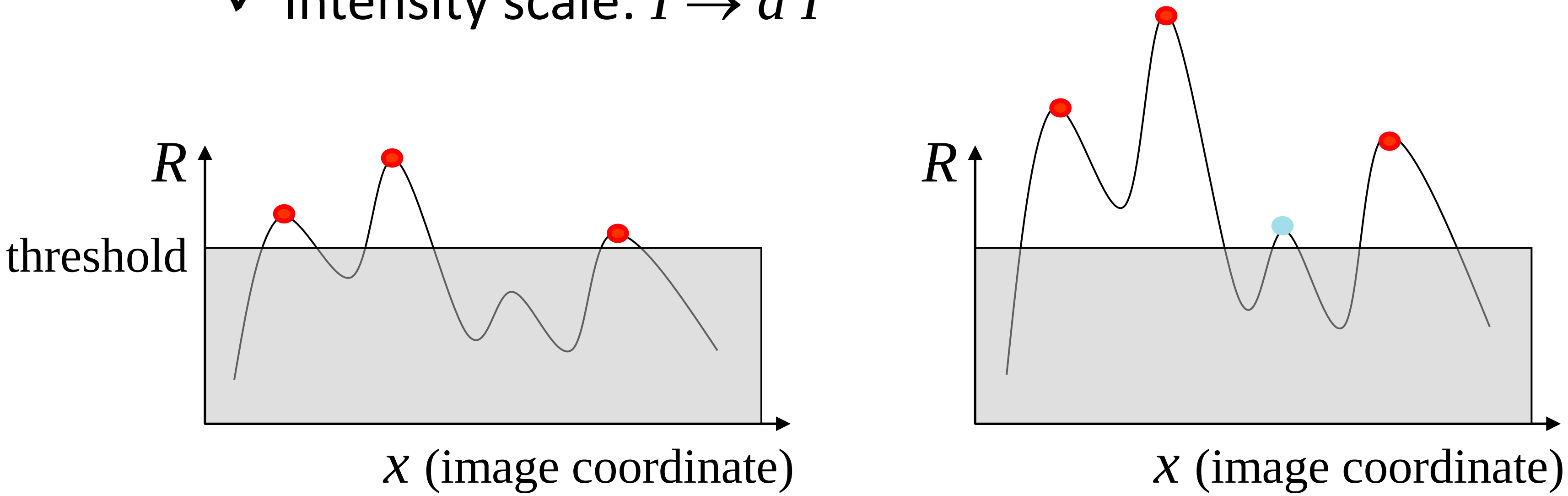
Corner response is invariant to image rotation

Harris Detector: Invariance Properties

- Affine intensity change: $I \rightarrow aI + b$

✓ Only derivatives are used =>
invariance to intensity shift $I \rightarrow I + b$

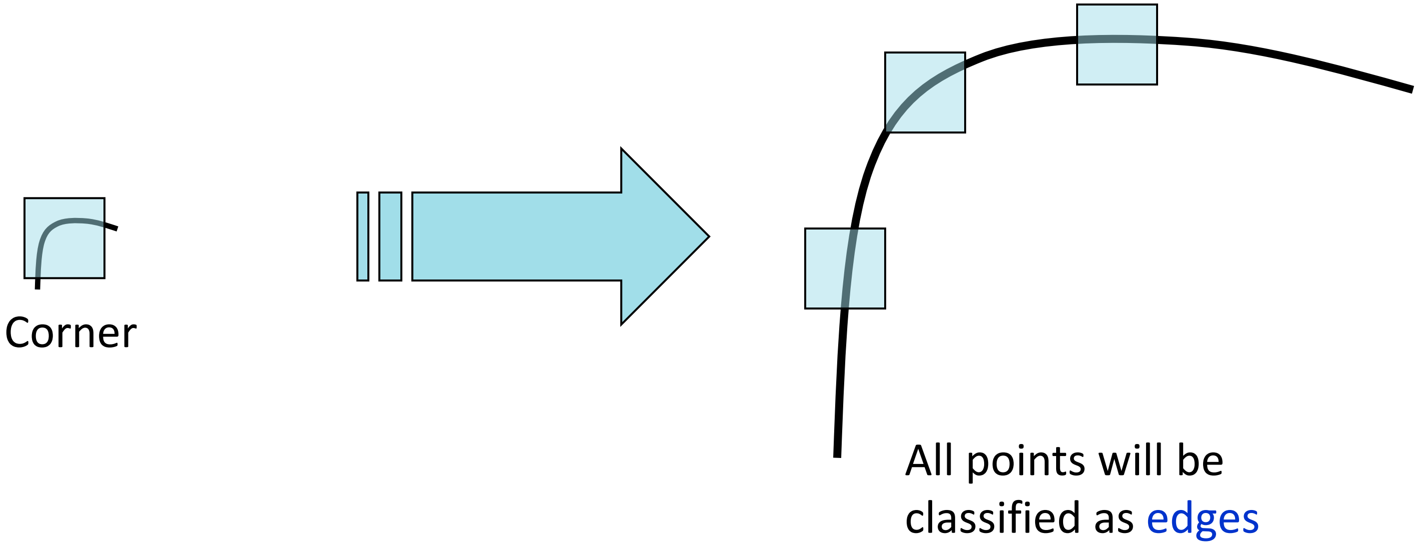
✓ Intensity scale: $I \rightarrow a\,I$



Partially invariant to affine intensity change

Harris Detector: Invariance Properties

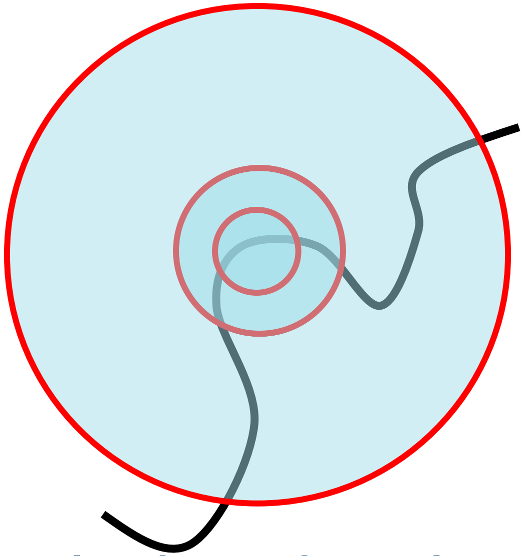
- Scaling



Not invariant to scaling

Scale invariant detection

Suppose you're looking for corners



Key idea: find scale that gives local maximum of *f*

- in both position and scale
- One definition of *f* : the Harris operator

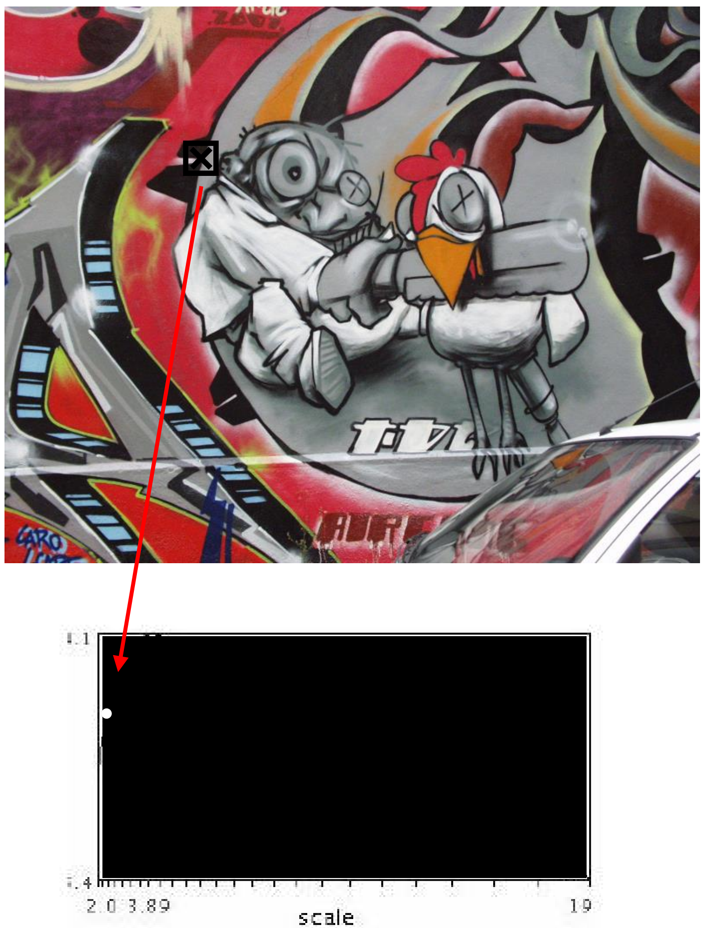
Automatic Scale Selection

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

Same operator responses if the patch contains the same image up to scale factor. How to find corresponding patch sizes?

Automatic Scale Selection

- Function responses for increasing scale (scale signature)

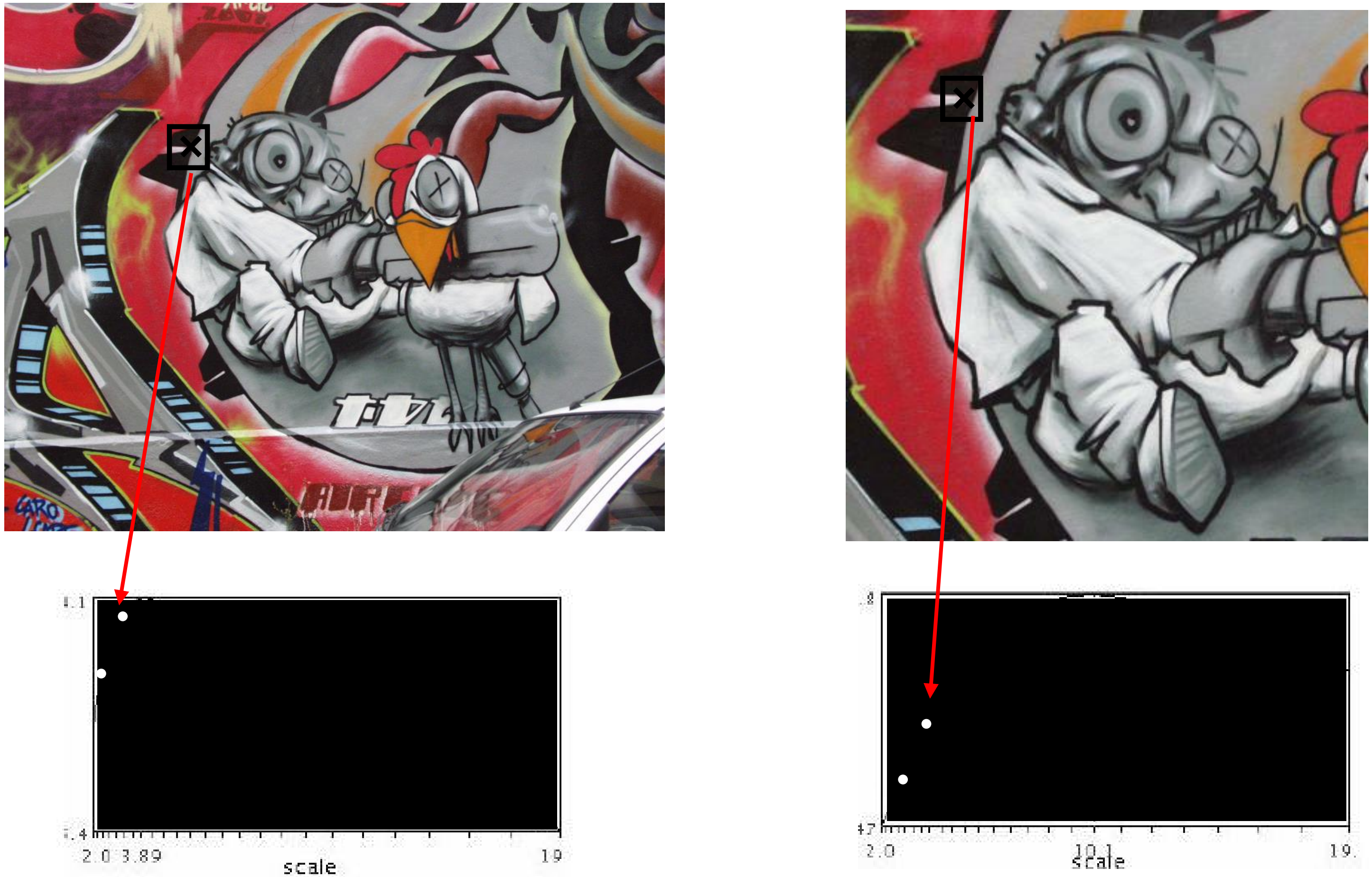


$f(I_{i_1 \ldots i_m}(x,\sigma))$

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

Automatic Scale Selection

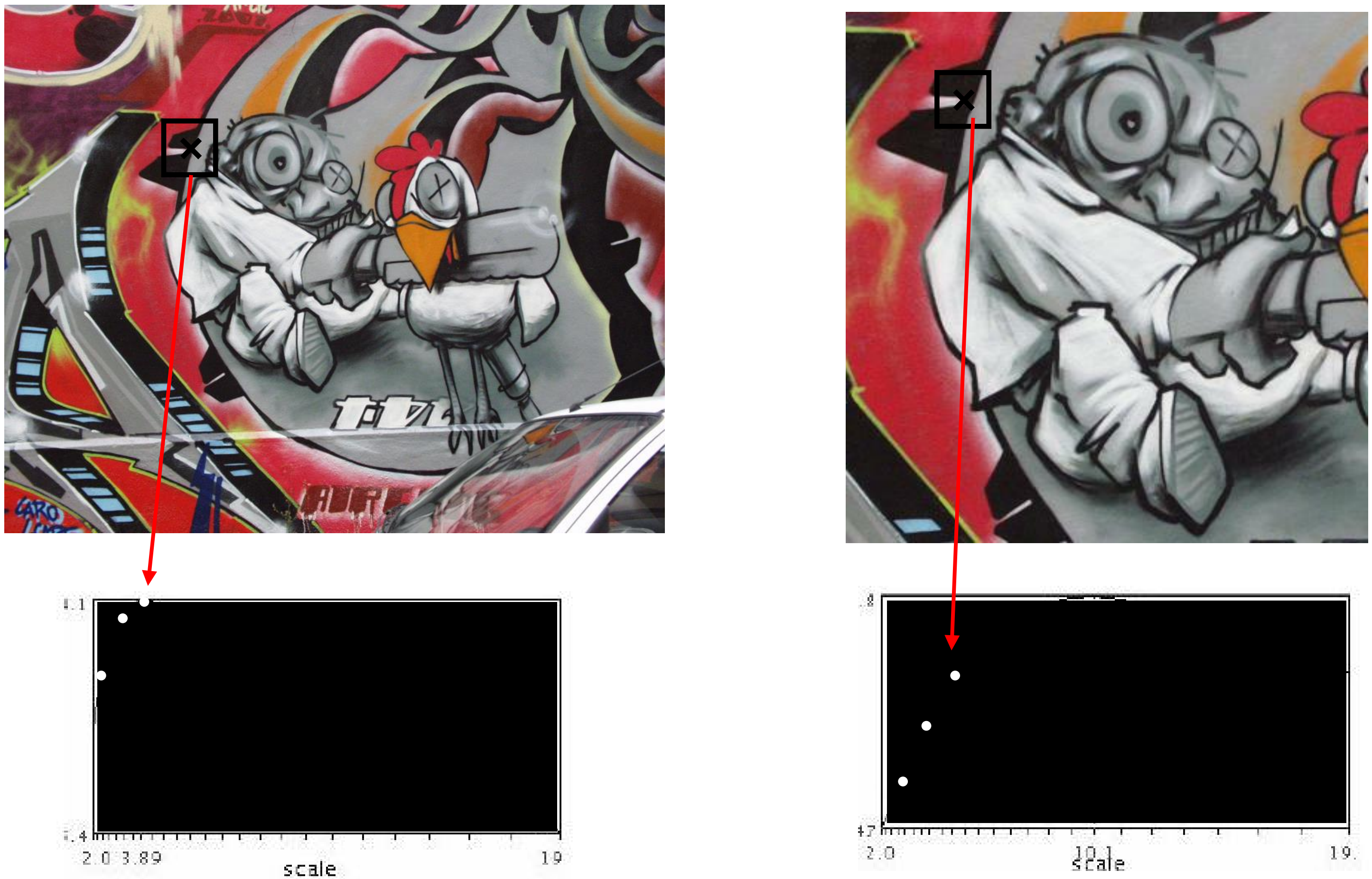
- Function responses for increasing scale (scale signature)



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

Automatic Scale Selection

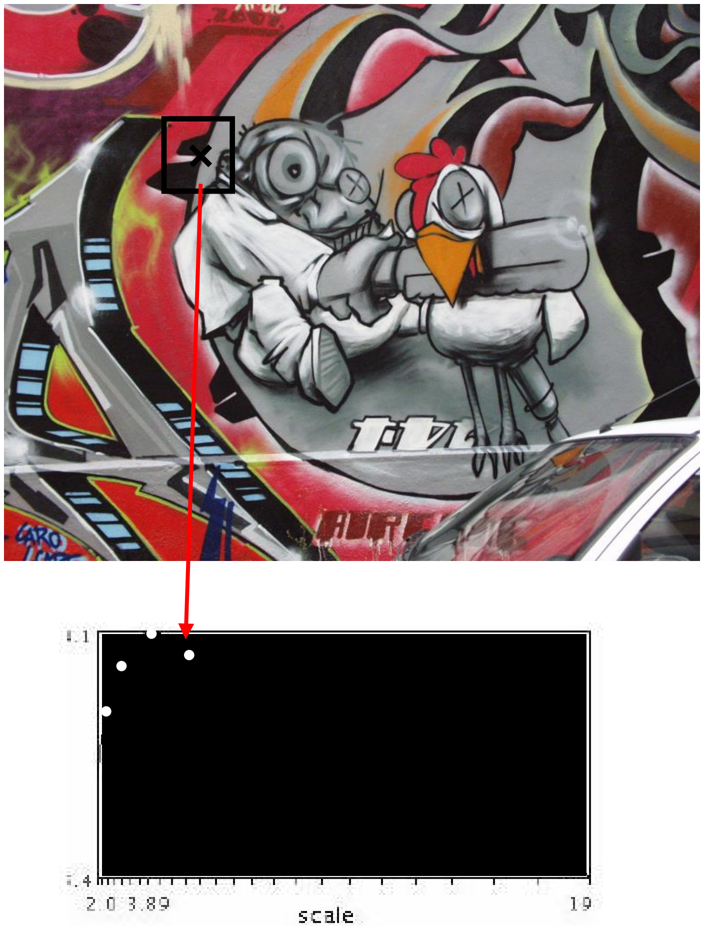
- Function responses for increasing scale (scale signature)



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

Automatic Scale Selection

- Function responses for increasing scale (scale signature)

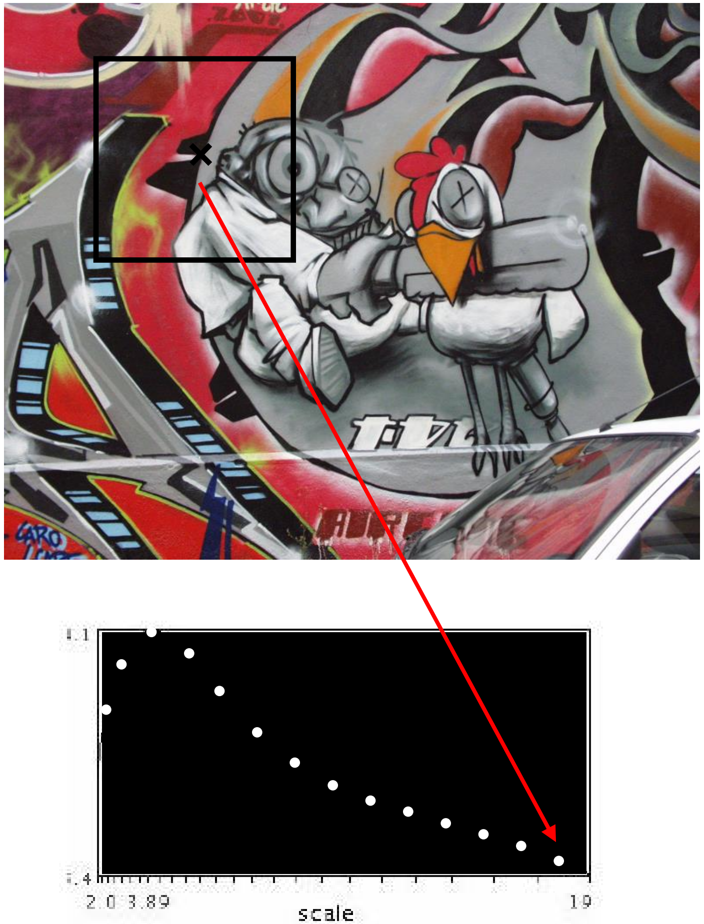


$f(I_{i_1 \ldots i_m}(x, \sigma))$

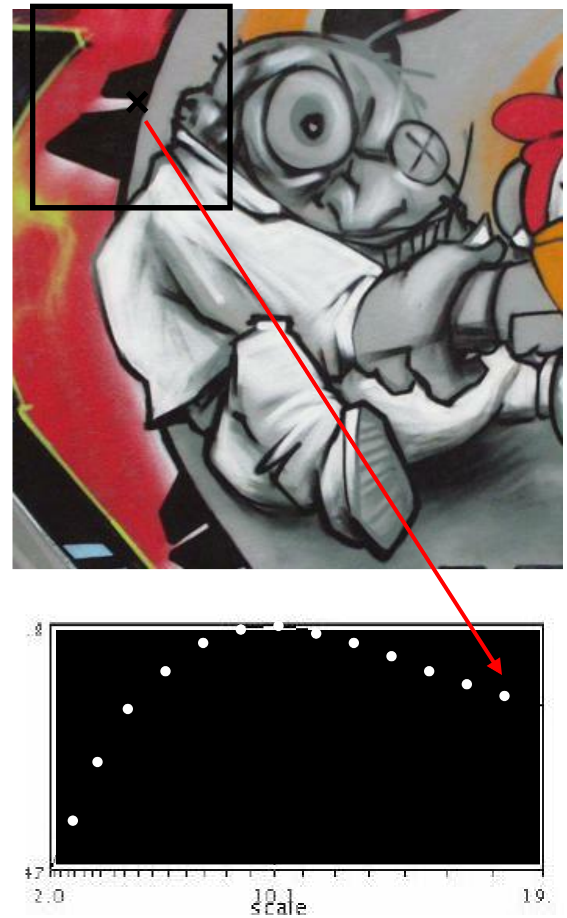
$f(I_{i_1 \ldots i_m}(x', \sigma))$

Automatic Scale Selection

- Function responses for increasing scale (scale signature)



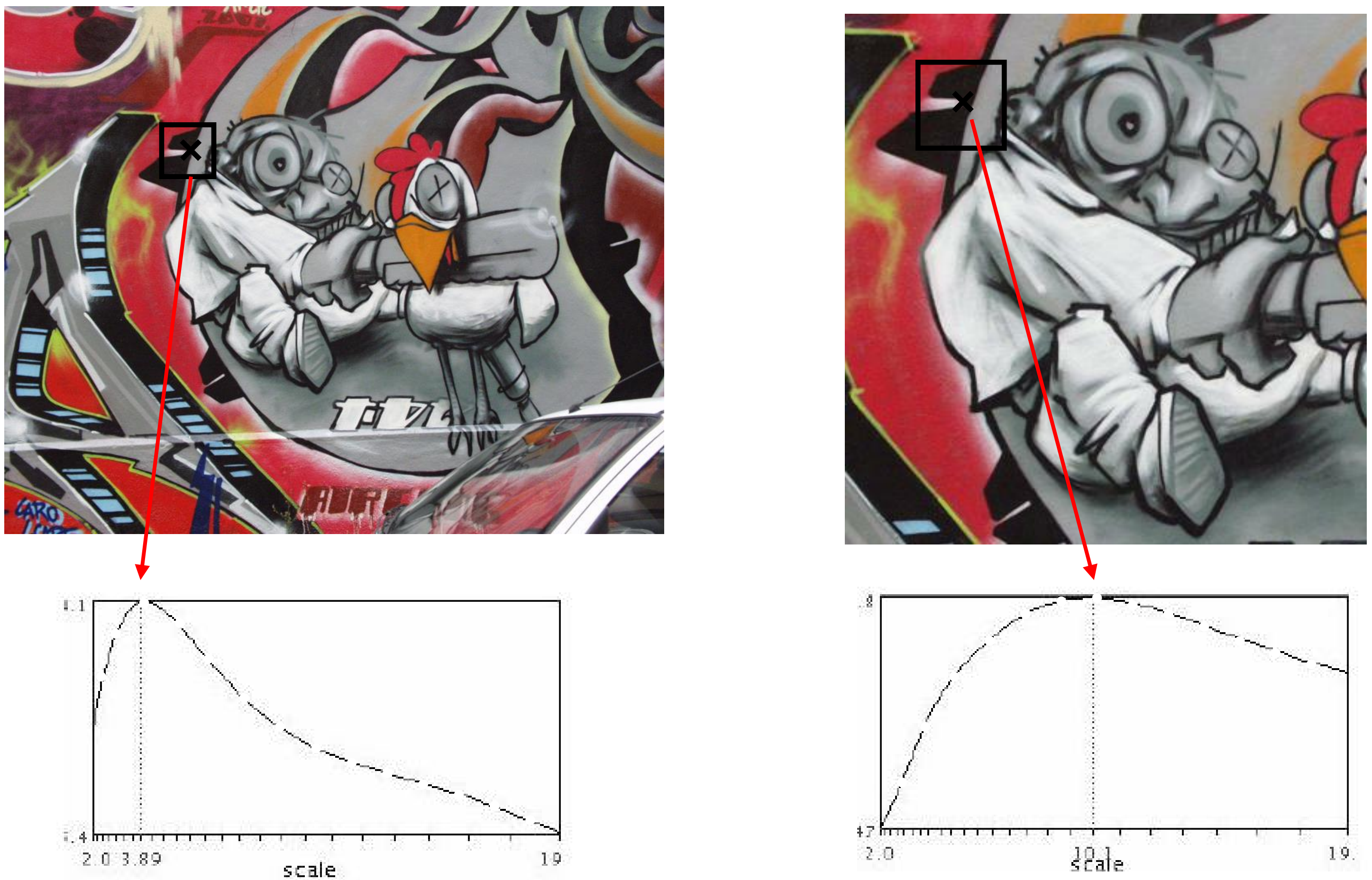
$f(I_{i_1 \ldots i_m}(x, \sigma))$



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

Automatic Scale Selection

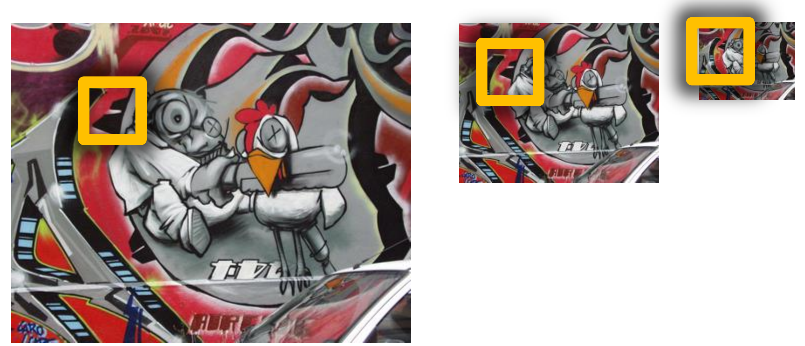
- Function responses for increasing scale (scale signature)



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

Implementation

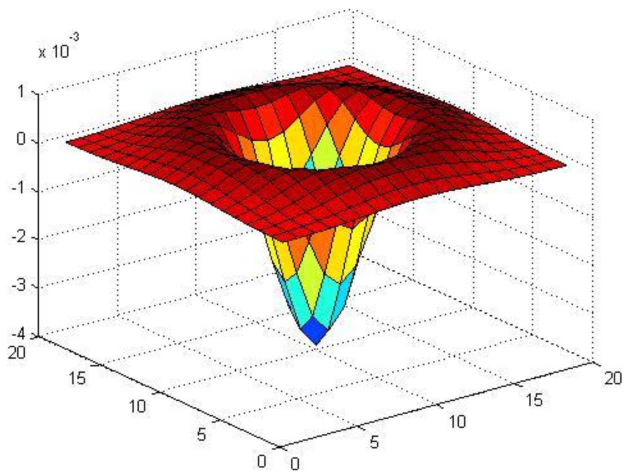
- Instead of computing *f* for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid

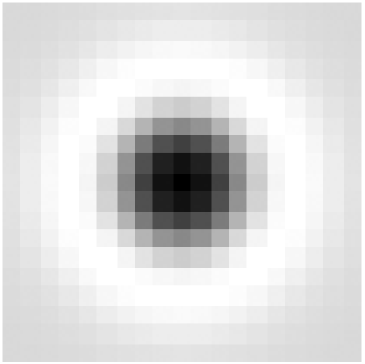


(sometimes need to create in-between levels, e.g. a ¾-size image)

Another common definition of *f*

- The *Laplacian of Gaussian* (*LoG*)



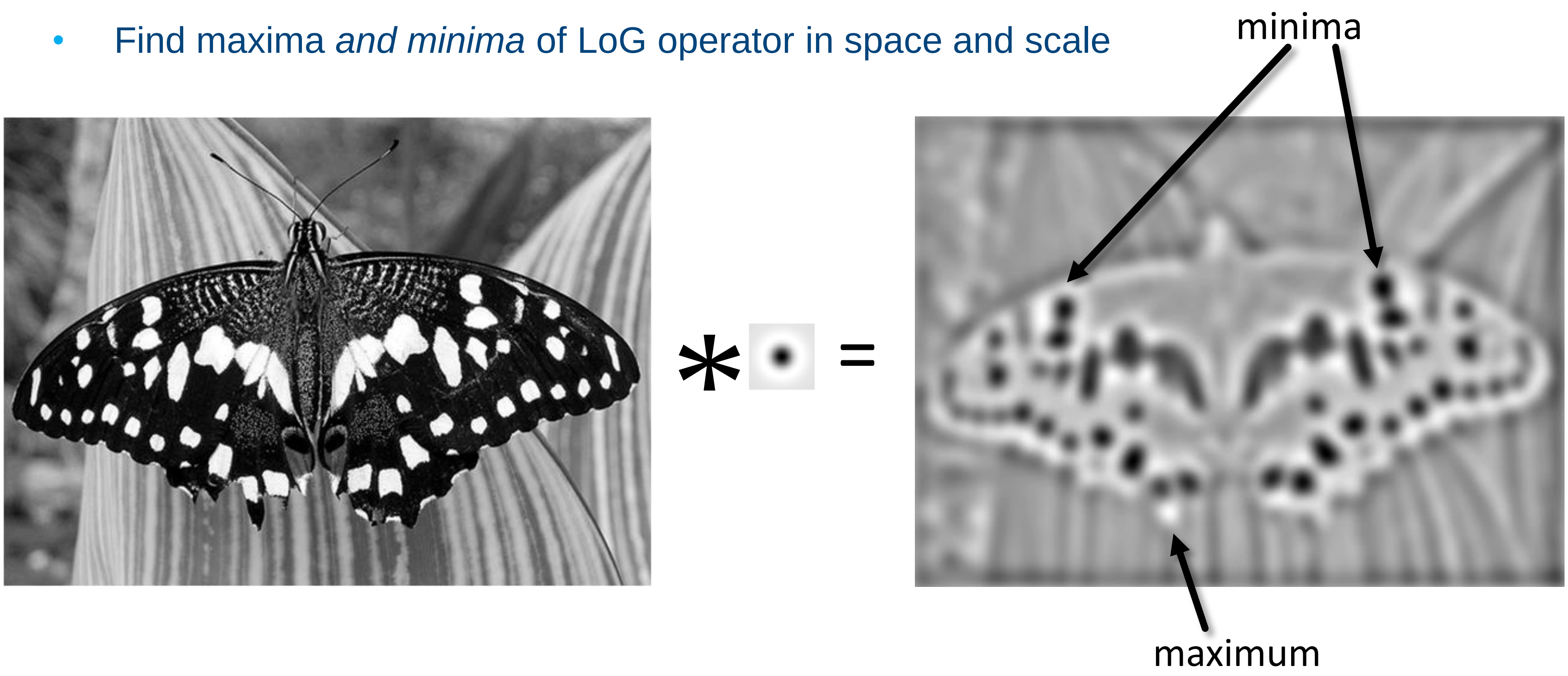


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

(very similar to a Difference of Gaussians (DoG) – i.e. a Gaussian minus a slightly smaller Gaussian)

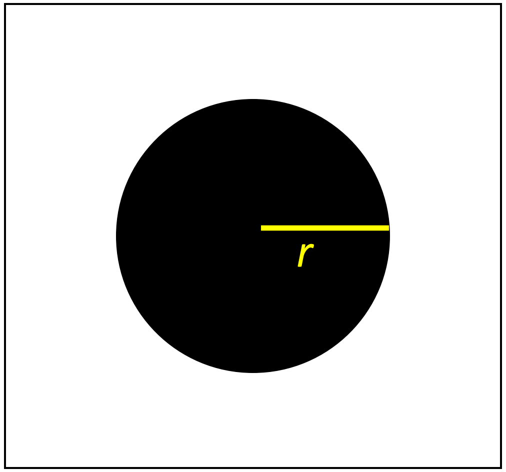
Laplacian of Gaussian

- “Blob” detector
- Find maxima *and minima* of LoG operator in space and scale

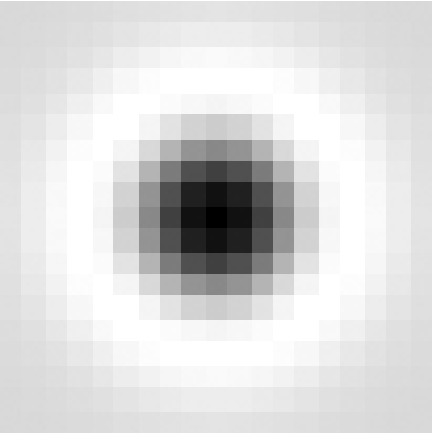


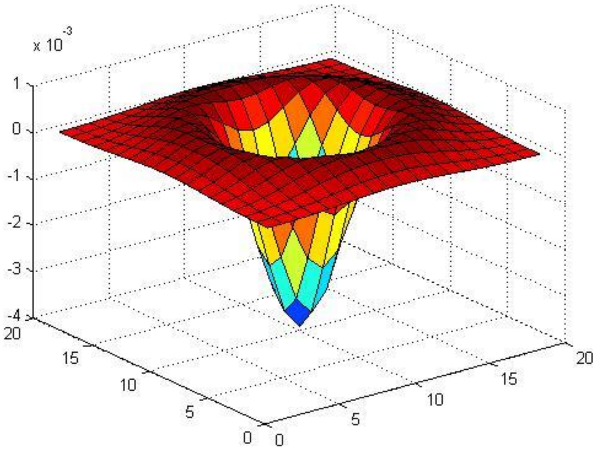
Scale selection

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



image

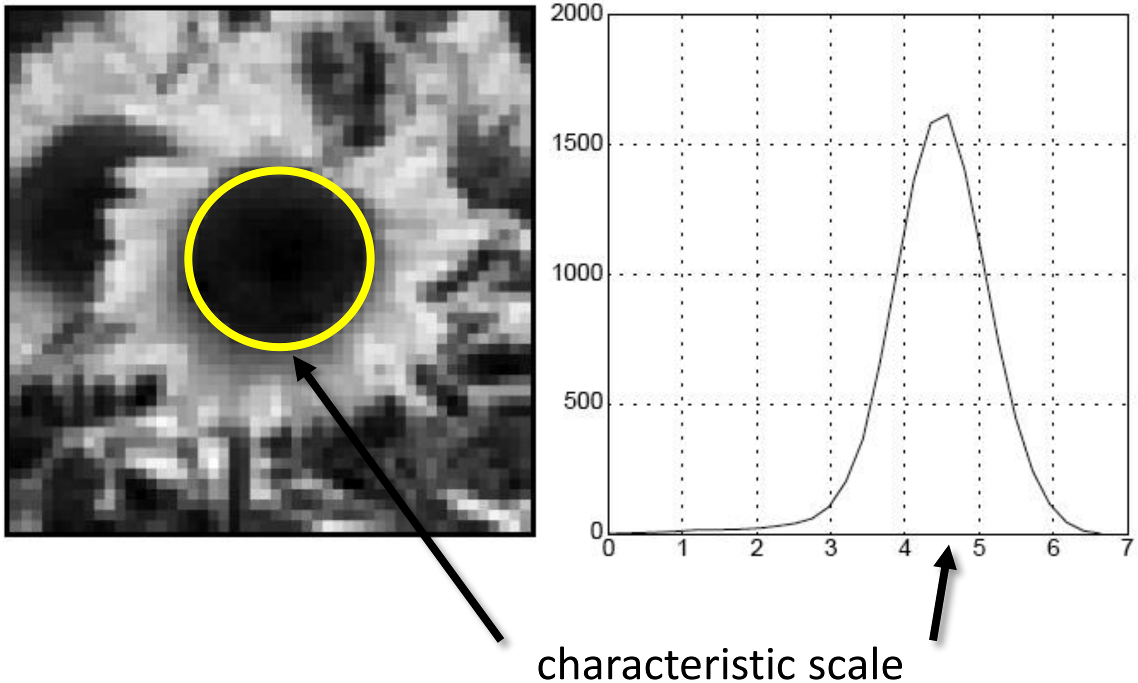




Laplacian

Characteristic scale

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



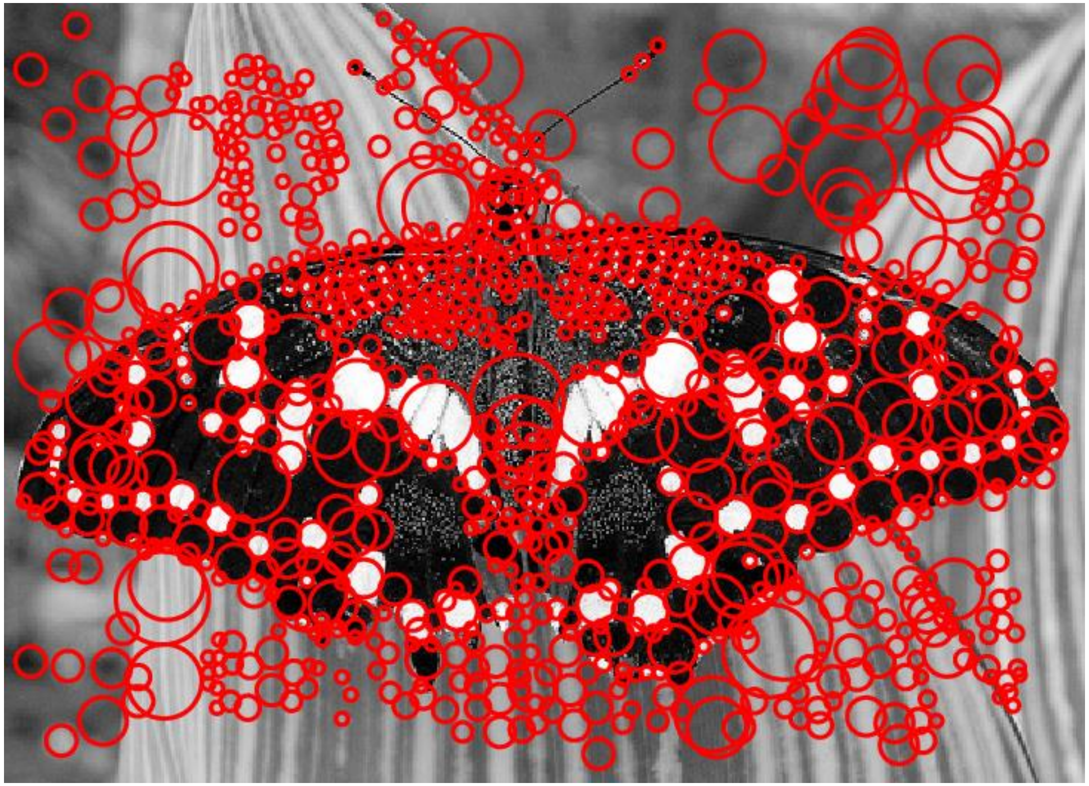
T. Lindeberg (1998). "Feature detection with automatic scale selection." *International Journal of Computer Vision* **30** (2): pp 77--116.

Scale-space blob detector: Example

Scale-space blob detector: Example

sigma = 11.9912

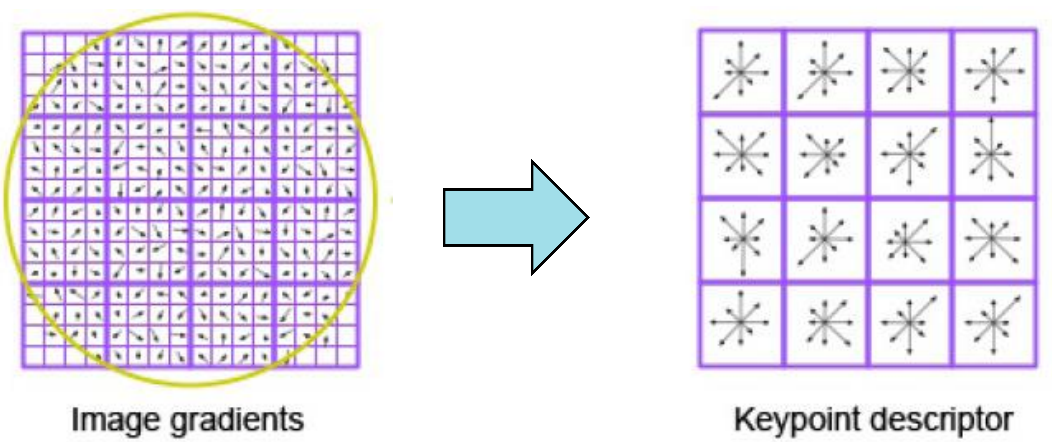
Scale-space blob detector: Example



Scale Invariant Feature Transform (SIFT)

1. Take a 16 x16 window around interest point (i.e., at the scale detected).
2. Divide into a 4x4 grid of cells.
3. Compute histogram of image gradients in each cell (8 bins each).

16 histograms x 8 orientations
= 128 features



SIFT Computation – Steps

(1) Scale-space extrema detection

- Extract scale and rotation invariant interest points (i.e., keypoints).

(2) Keypoint localization

- Determine location and scale for each interest point.
- Eliminate “weak” keypoints

(3) Orientation assignment

- Assign one or more orientations to each keypoint.

(4) Keypoint descriptor

- Use local image gradients at the selected scale.

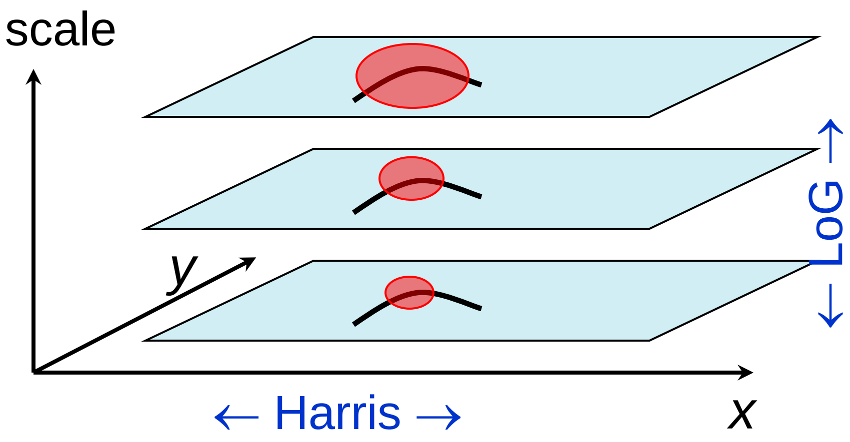
D. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints”, **International Journal of Computer Vision**, 60(2):91-110, 2004.

Cited 13629 times (as of 17/4/2012)

Scale-space Extrema Detection

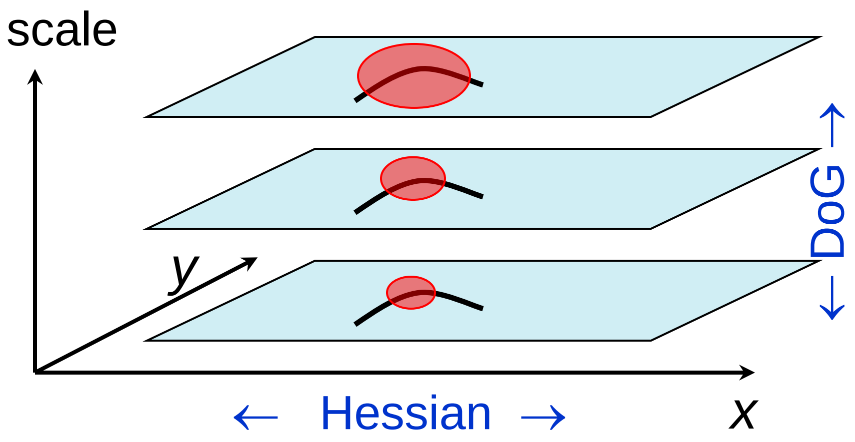
- Harris-Laplace

- *Find local maxima of:*
  - Harris detector in space
  - LoG in scale



- SIFT

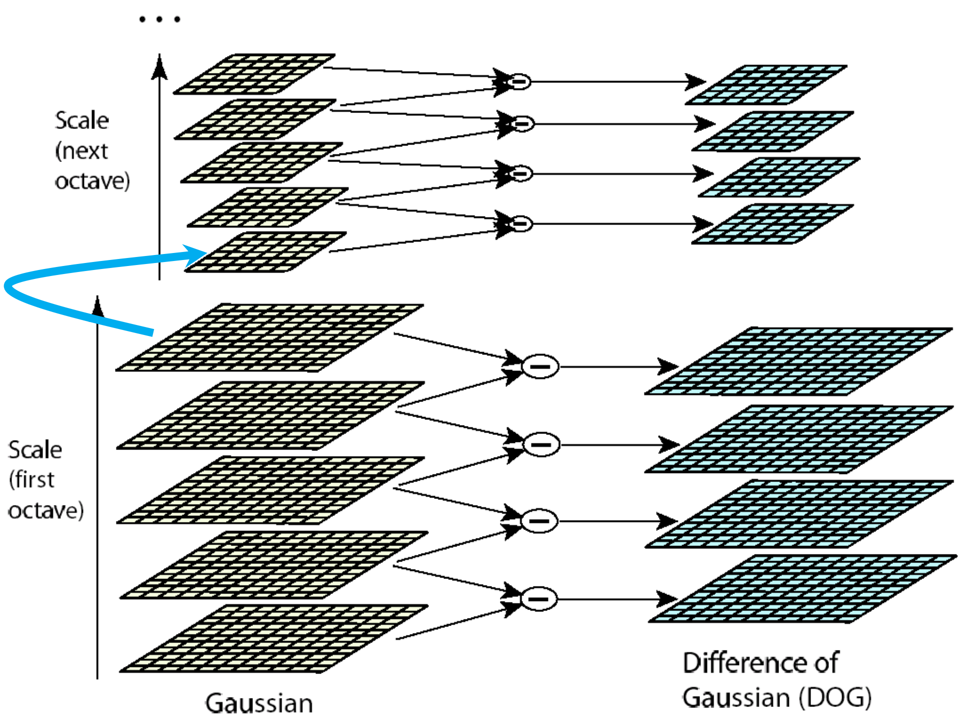
  Find local maxima of:
  - Hessian in space
  - DoG in scale



Scale-space Extrema Detection

- DoG images are grouped by octaves (i.e., <u>doubling</u> of σ_0)
- Fixed number of levels per octave

down-sample



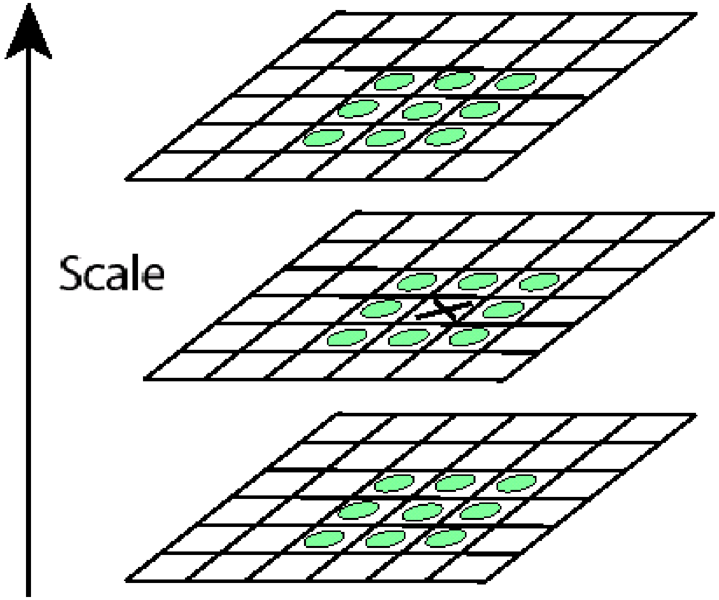
$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

where

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

Scale-space Extrema Detection

- Extract local extrema (i.e., minima or maxima) in DoG pyramid.
  - -Compare each point to its 8 neighbors at the same level, 9 neighbors in the level above, and 9 neighbors in the level below (i.e., 26 total).



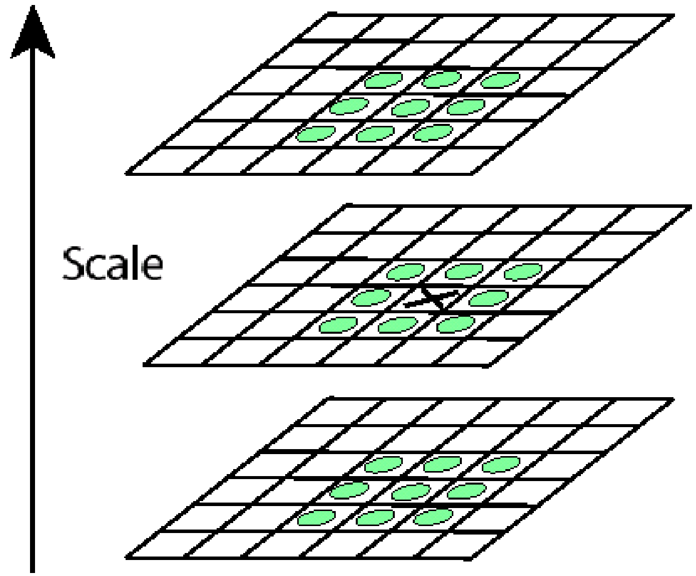
Keypoint Localization

- Determine the location and scale of keypoints to **sub-pixel** and **sub-scale** accuracy by fitting a 3D quadratic polynomial:

$$X_i = (x_i, y_i, \sigma_i)$$ keypoint location

$$\Delta X = (x - x_i, y - y_i, \sigma - \sigma_i)$$ offset

$$X_i \leftarrow X_i + \Delta X$$ sub-pixel, sub-scale Estimated location



Substantial improvement to matching and stability!

Keypoint Localization

- Use Taylor expansion to locally approximate D(x,y,σ) (i.e., DoG function) and estimate Δx:

$$D(\Delta X) = D(X_i) + \frac{\partial D^T(X_i)}{\partial \mathrm{X}} \Delta \mathrm{X} + \frac{1}{2} \Delta \mathrm{X}^T \frac{\partial^2 D(X_i)}{\partial \mathrm{X}^2} \Delta \mathrm{X}$$

- Find the extrema of D(ΔX):

$$\frac{\partial D(X_i)}{\partial X} + \frac{\partial^2 D(X_i)}{\partial X^2} \Delta X = 0$$

Keypoint Localization

$$\frac{\partial^2 D(X_i)}{\partial X^2}\Delta X = -\frac{\partial D(X_i)}{\partial X} \rightarrow \Delta X = -\frac{\partial^2 D^{-1}(X_i)}{\partial X^2}\frac{\partial D(X_i)}{\partial X}$$

- ΔX can be computed by solving a 3x3 linear system:

$$\begin{bmatrix} \frac{\partial^2 D}{\partial \sigma^2} & \frac{\partial^2 D}{\partial \sigma y} & \frac{\partial^2 D}{\partial \sigma x} \\ \frac{\partial^2 D}{\partial \sigma y} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial yx} \\ \frac{\partial^2 D}{\partial \sigma x} & \frac{\partial^2 D}{\partial yx} & \frac{\partial^2 D}{\partial x^2} \end{bmatrix} \begin{bmatrix} \Delta\sigma \\ \Delta y \\ \Delta x \end{bmatrix} = - \begin{bmatrix} \frac{\partial D}{\partial \sigma} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial x} \end{bmatrix}$$

use finite differences:

$$\frac{\partial D}{\partial \sigma} = \frac{D_{k+1}^{i,j} - D_{k-1}^{i,j}}{2}$$

$$\frac{\partial^2 D}{\partial \sigma^2} = \frac{D_{k-1}^{i,j} - 2D_k^{i,j} + D_{k+1}^{i,j}}{1}$$

$$\frac{\partial^2 D}{\partial \sigma y} = \frac{(D_{k+1}^{i+1,j} - D_{k-1}^{i+1,j}) - (D_{k+1}^{i-1,j} - D_{k-1}^{i-1,j})}{4}$$

If $\Delta X > 0.5$ in any dimension, repeat.

Keypoint Localization

- Reject keypoints having low contrast.
  - i.e., sensitive to noise

If $|D(X_i + \Delta X)| < 0.03$ reject keypoint

– i.e., assumes that image values have been normalized in [0,1]

Keypoint Localization

- Reject points lying on edges (or being close to edges)

- Harris uses the auto-correlation matrix:

$$A_W(x,y) = \sum_{x \in W, y \in W} \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}$$

$$R(A_W) = \det(A_W) - \alpha \, \mathrm{trace}^2(A_W)$$

or $$R(A_W) = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2$$

Keypoint Localization

- SIFT uses the Hessian matrix (for efficiency).
  - i.e., Hessian encodes principal curvatures

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

α: largest eigenvalue (λ_{max})
β: smallest eigenvalue (λ_{min})
(proportional to principal curvatures)

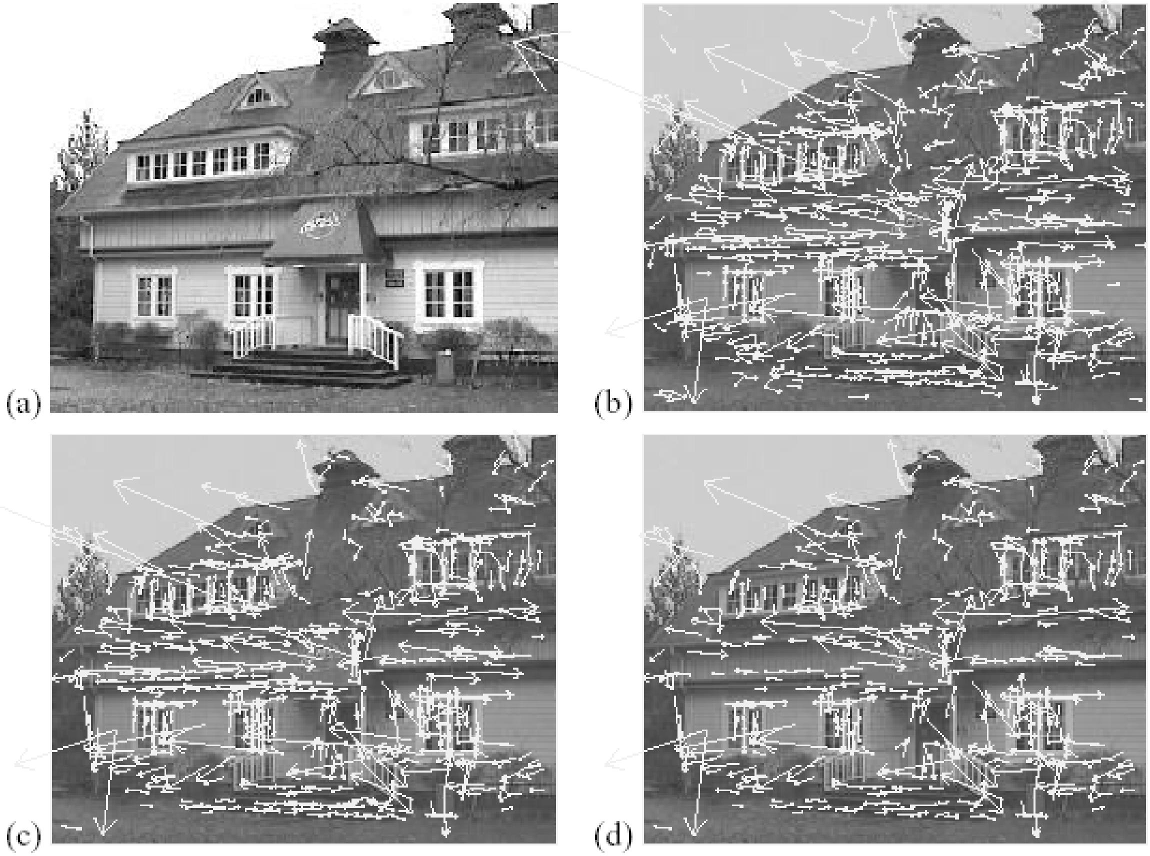
$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$
$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

$$\rightarrow \frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

($r = \alpha/\beta$)

Reject keypoint if: $\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$ (SIFT uses r = 10)

Keypoint Localization



(a) 233x189 image

(b) 832 DoG extrema

(c) 729 left after low contrast threshold

(d) 536 left after testing ratio based on Hessian

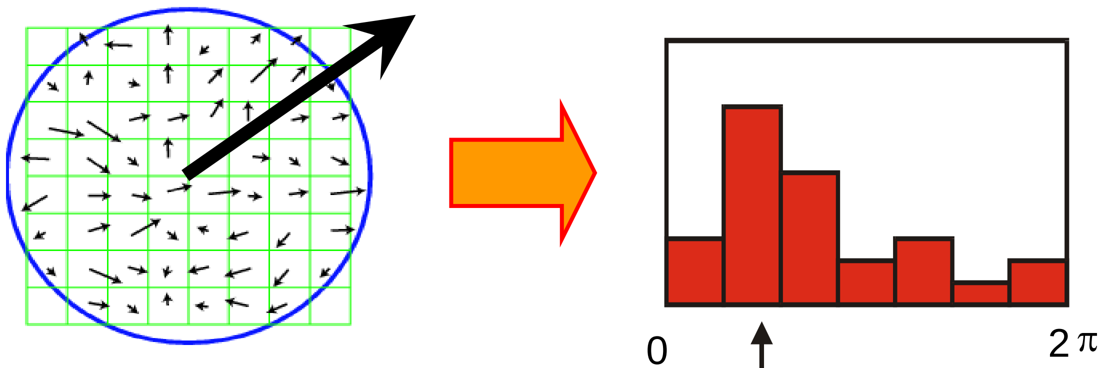
Orientation Assignment

- Create histogram of gradient directions, within a region around the keypoint, at selected scale:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = a\tan 2((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)))$$

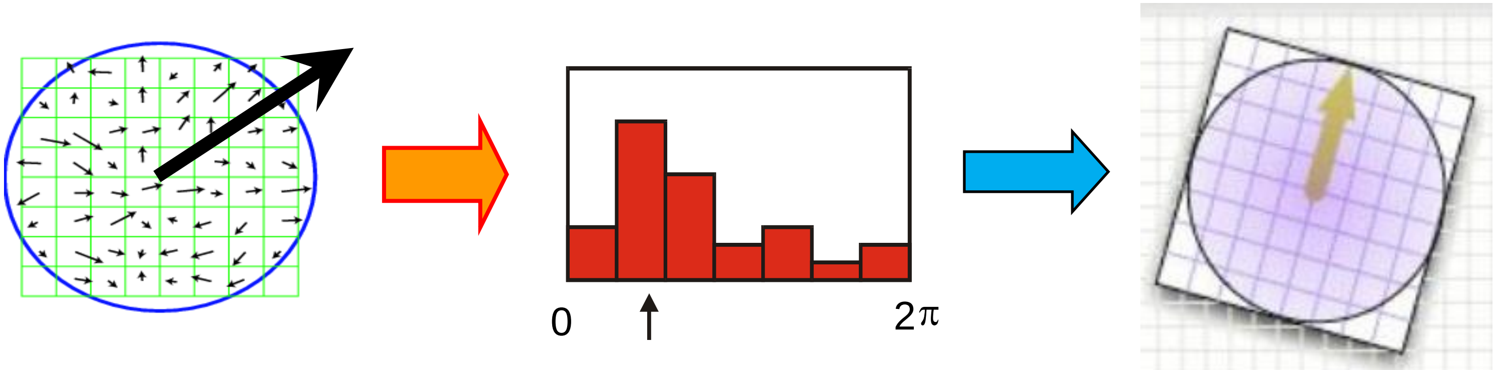


36 bins (i.e., 10º per bin)

- Histogram entries are **weighted** by (i) gradient magnitude and (ii) a Gaussian function with σ equal to 1.5 times the scale of the keypoint.

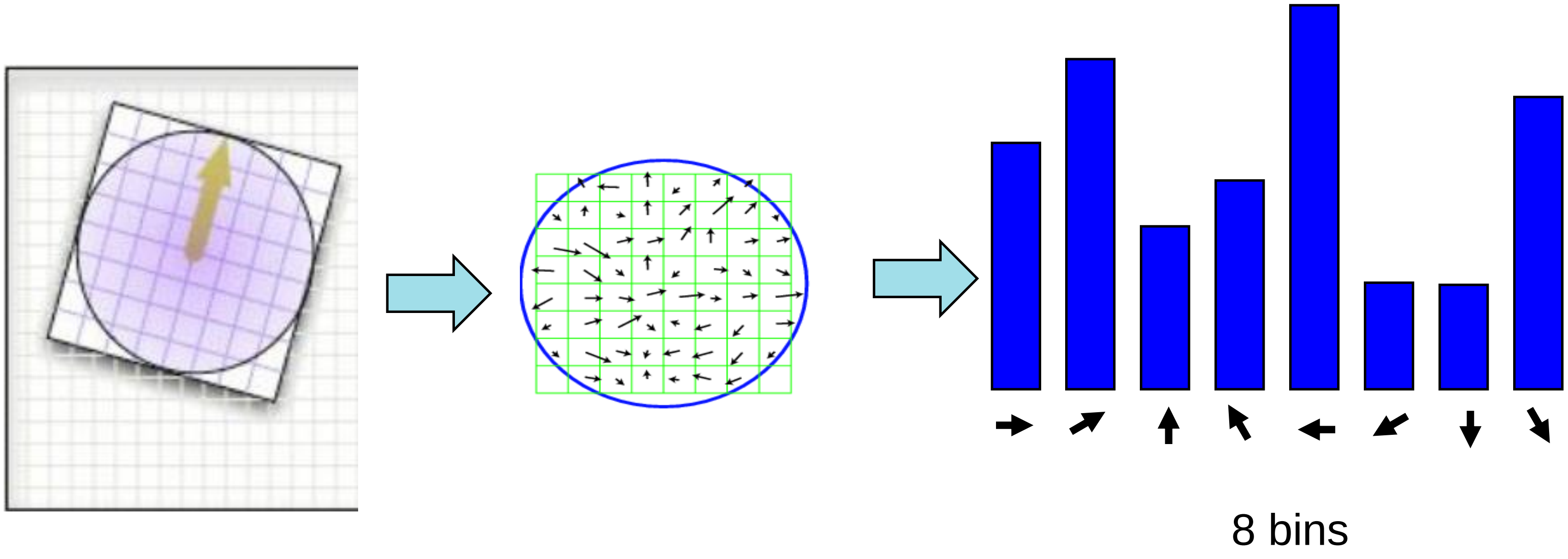
Orientation Assignment

- Assign canonical orientation at peak of smoothed histogram (fit parabola to better localize peak).



- In case of peaks within 80% of highest peak, multiple orientations assigned to keypoints.
  - About 15% of keypoints has multiple orientations assigned.
  - Significantly improves stability of matching.

Keypoint Descriptor

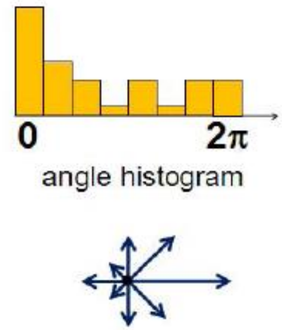


Keypoint Descriptor

1. Take a 16 x16 window around detected interest point.

2. Divide into a 4x4 grid of cells.

3. Compute histogram in each cell.



(8 bins)

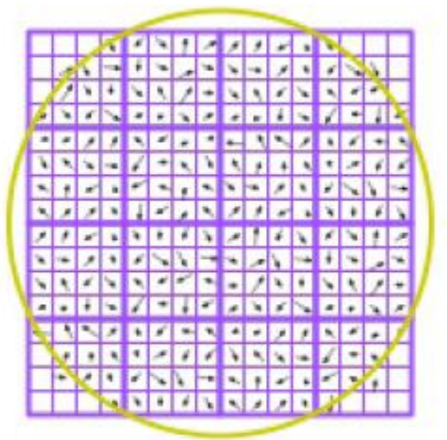
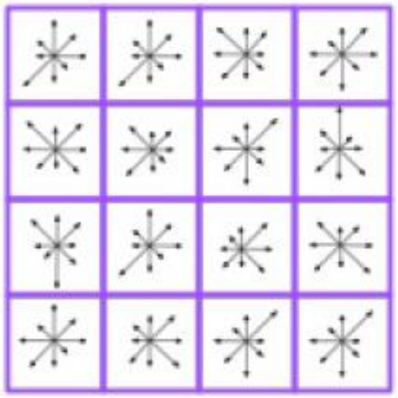


Image gradients

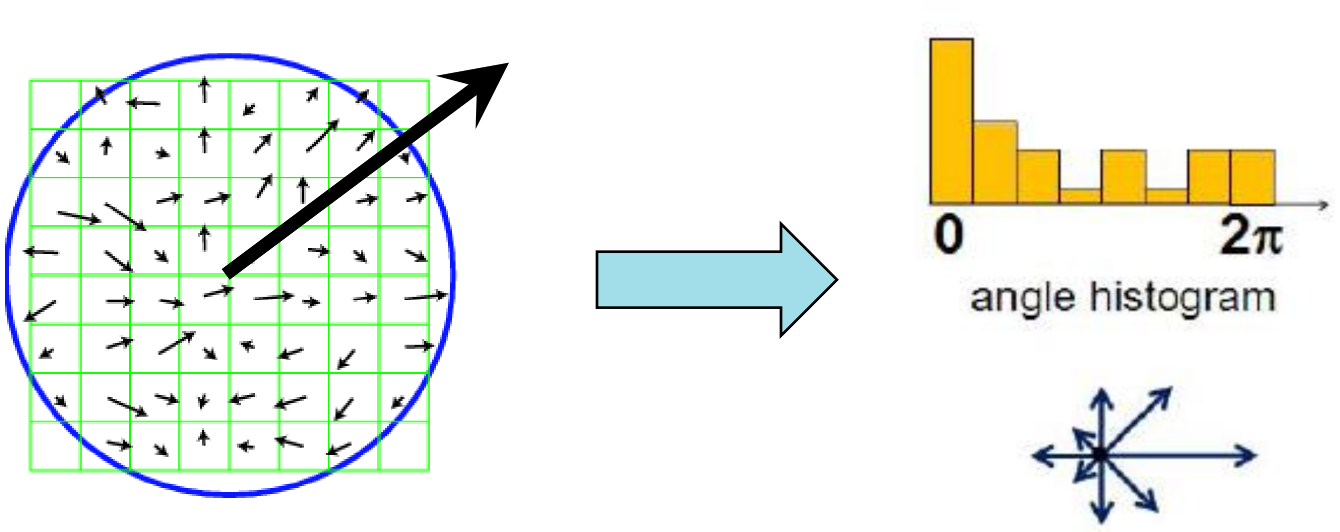


Keypoint descriptor

16 histograms x 8 orientations
= 128 features

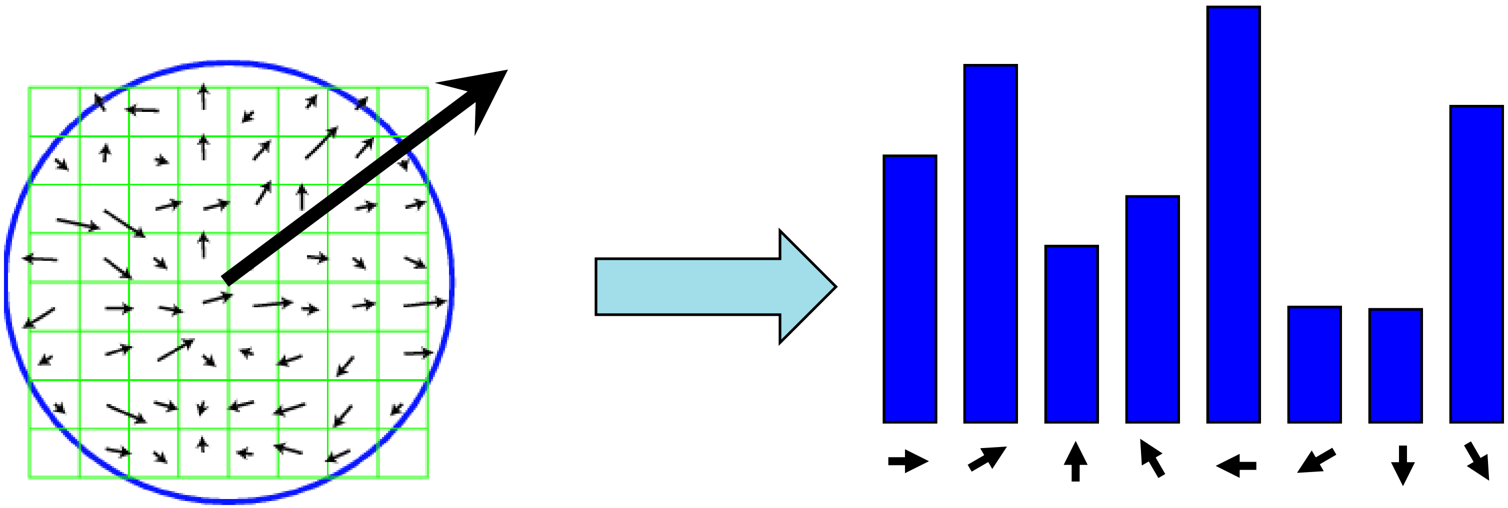
Keypoint Descriptor

- Each histogram entry **is** **weighted** by (i) gradient magnitude and (ii) a Gaussian function with σ equal to 0.5 times the width of the descriptor window.



Keypoint Descriptor

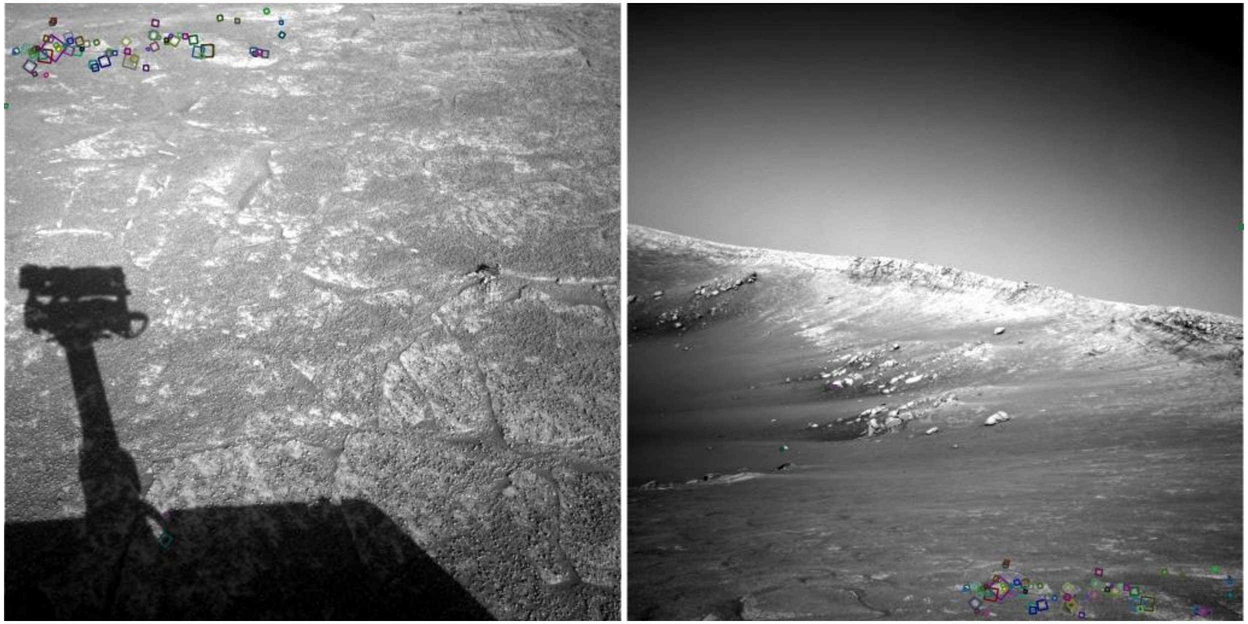
- Partial Voting: distribute histogram entries into adjacent bins (i.e., additional robustness to shifts)
  - Each entry is added to all bins, multiplied by a weight of 1-d, where d is the distance from the bin it belongs.



Properties of SIFT

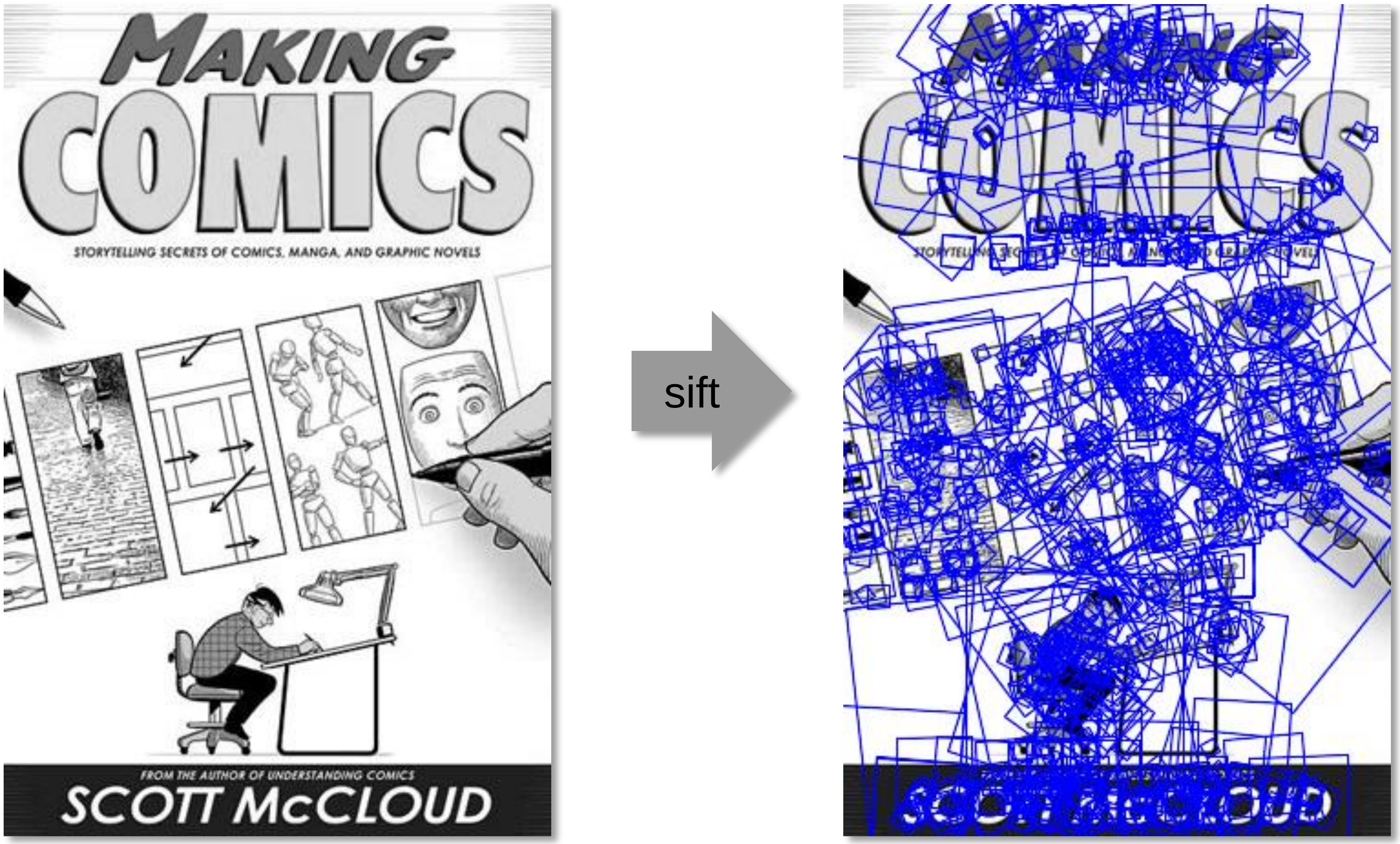
Extraordinarily robust matching technique

- 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



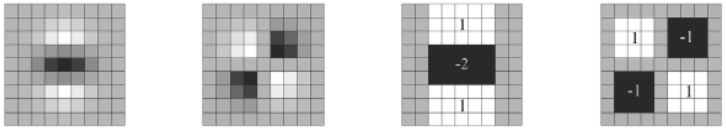
NASA Mars Rover images with SIFT feature matches

SIFT Example



868 SIFT features

Local Descriptors: SURF



Fast approximation of SIFT idea

- Efficient computation by 2D box filters & integral images
  ⇒ 6 times faster than SIFT
- Equivalent quality for object identification

GPU implementation available

- Feature extraction @ 100Hz (detector + descriptor, 640 × 480 img)
- http://www.vision.ee.ethz.ch/~surf

[Bay, ECCV'06], [Cornelis, CVGPU'08]

Main points of this lecture

- Moving the same camera restricts the geometry allowing inference about 3D
  Potential uses range from mosaicing to egomotion estimation
  - In principle the same mechanism that human depth perception is based on
- Learn the RANSAC algorithm and understand why it works
  - Simple, fast algorithm applicable in very many tasks
  - Important part of your toolbox
- Grasp the concept of scale-invariant features
  - Example: SIFT algorithm (location and description)
- Geometry and image transforms is out of scope for this course
  - But part of INF 2310 – so you know all this!

Properties of SIFT

- Highly distinctive
  - A single feature can be correctly matched with high probability against a large database of features from many images.
- Scale and rotation invariant.
- Partially invariant to 3D camera viewpoint
  - Can tolerate up to about 60 degree out of plane rotation
- Partially invariant to changes in illumination
- Can be computed fast and efficiently.