Dense Correspondence

Vinay P. Namboodiri

- Slide credit to James Hays, Derek Hoiem, Kirsten Grauman and Bastian Leibe, Robert T. Collins

Outline

- Review over Hough, Ransac and ICP
- Finding the object
- Dense Correspondence using correlation

Review: Hough Transform

1. Create a grid of parameter values

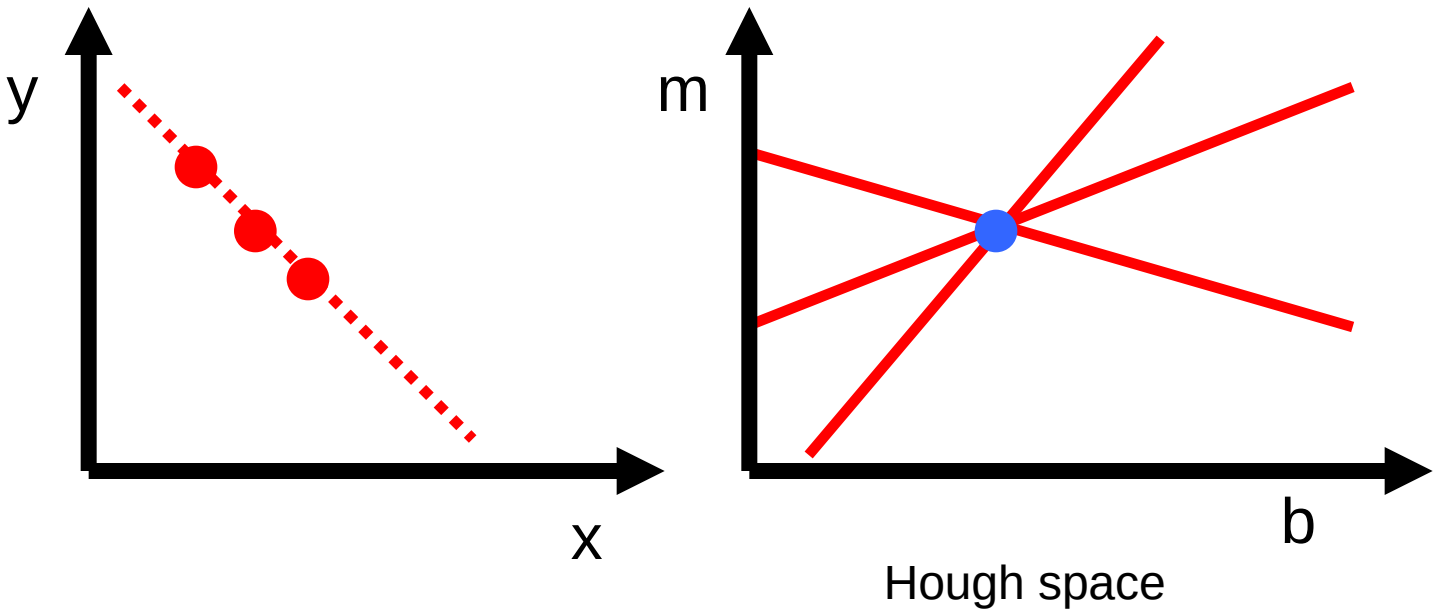
2. Each point votes for a set of parameters, incrementing those values in grid

3. Find maximum or local maxima in grid

Review: Hough transform

P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

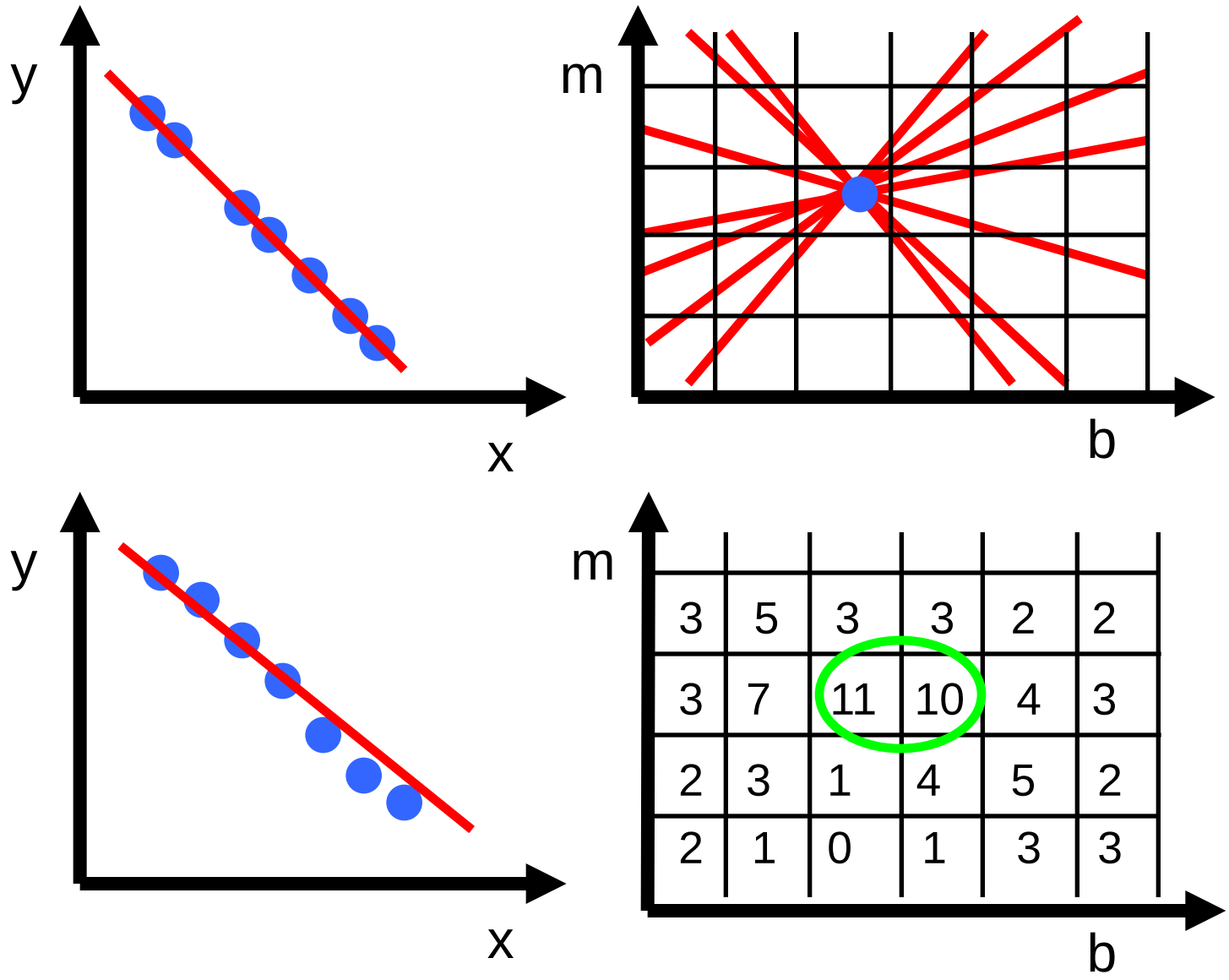
Given a set of points, find the curve or line that explains the data points best



$$y = m\,x + b$$

Slide from S. Savarese

Review: Hough transform



Slide from S. Savarese

Hough transform conclusions

Good

- Robust to outliers: each point votes separately
- Fairly efficient (much faster than trying all sets of parameters)
- Provides multiple good fits

Bad

- Some sensitivity to noise
- Bin size trades off between noise tolerance, precision, and speed/memory
  - Can be hard to find sweet spot
- Not suitable for more than a few parameters
  - grid size grows exponentially

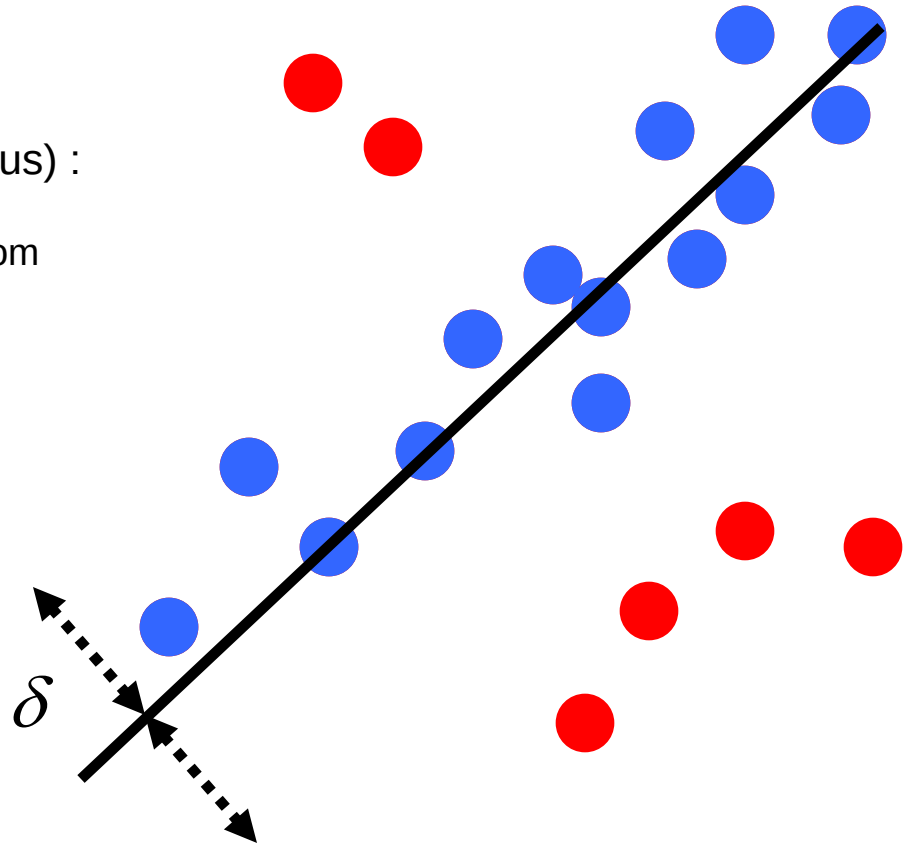
Common applications

- Line fitting (also circles, ellipses, etc.)
- Object instance recognition (parameters are affine transform)
- Object category recognition (parameters are position/scale)

RANSAC

(RANdom SAmple Consensus) :
Learning technique to estimate parameters of a model by random sampling of observed data

Fischler & Bolles in ‘81.



$$\pi : \boldsymbol{I} \rightarrow \{\boldsymbol{P}, \boldsymbol{O}\} \qquad \min_{\pi} |\boldsymbol{O}|$$

such that:

$$\boldsymbol{f}(\boldsymbol{P}, \beta) < \delta \qquad \boldsymbol{f}(\boldsymbol{P}, \beta) = \left\| \beta - \left(\boldsymbol{P}^{\boldsymbol{T}} \boldsymbol{P}\right)^{-1} \boldsymbol{P}^{\boldsymbol{T}} \right\|$$

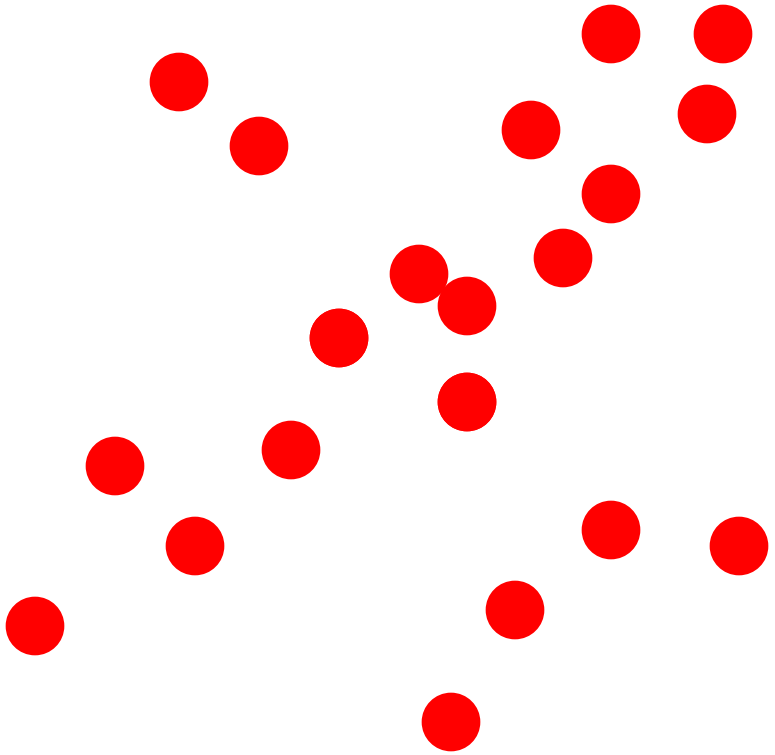
Model parameters

Source: Savarese

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in ‘81.



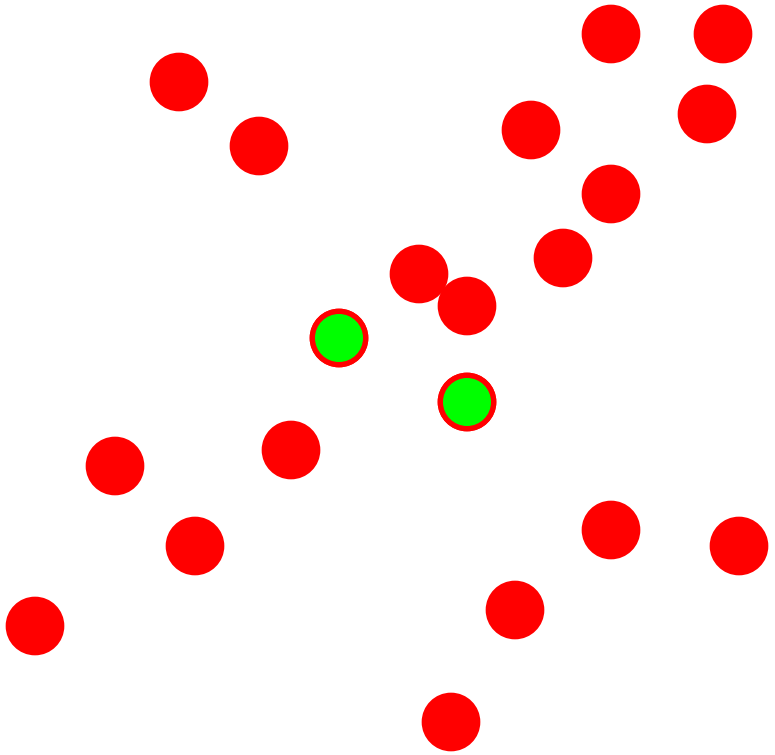
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



Algorithm:

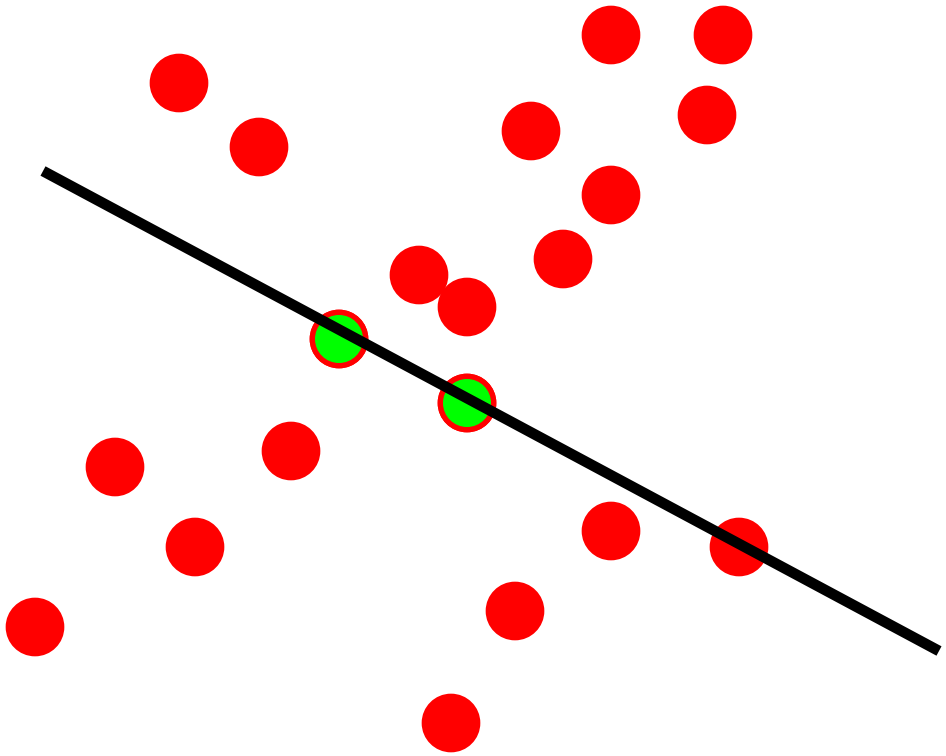
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Illustration by Savarese

RANSAC

Line fitting example



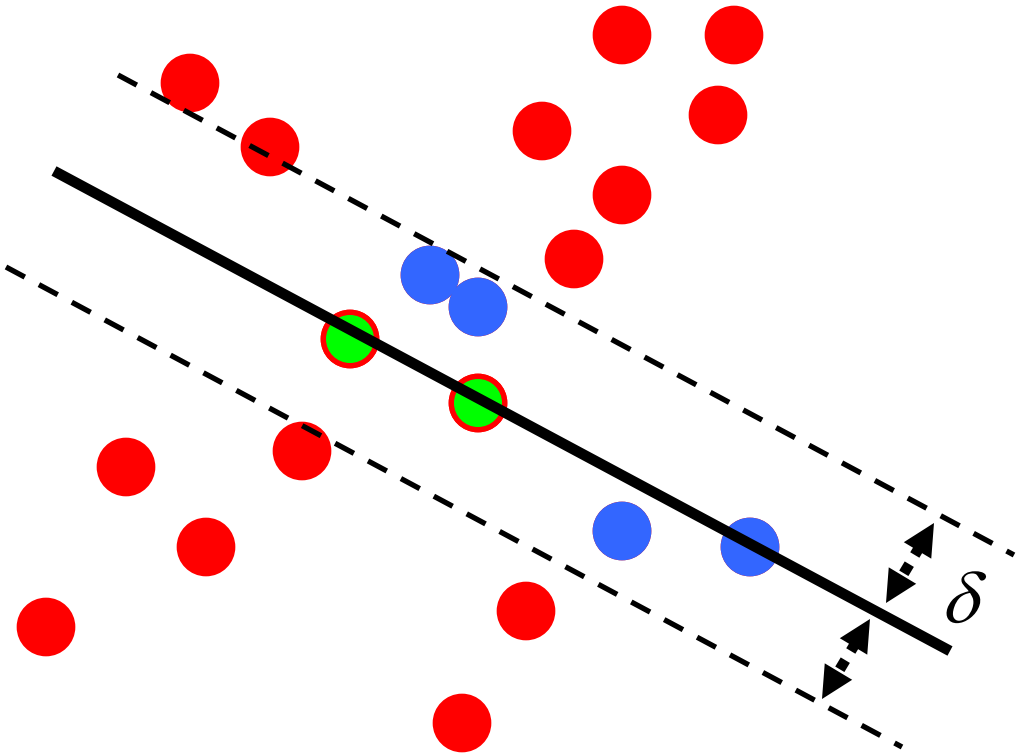
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



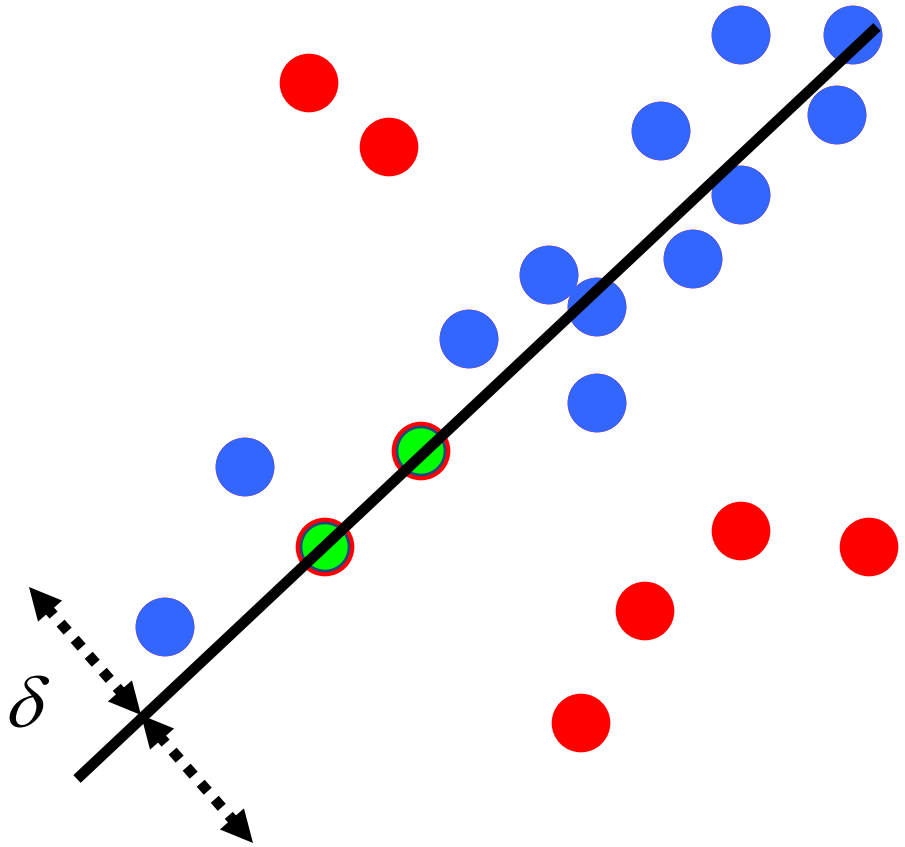
$$N_I = 6$$

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC



$N_I = 14$

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

How to choose parameters?

- 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*)
- Number of sampled points *s*
  - Minimum number needed to fit the model
- Distance threshold δ
  - Choose δ so that a good point with noise is likely (e.g., prob=0.95) within threshold
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$

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

	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 objective function parameters than Hough transform
- Optimization parameters are easier to choose than Hough transform

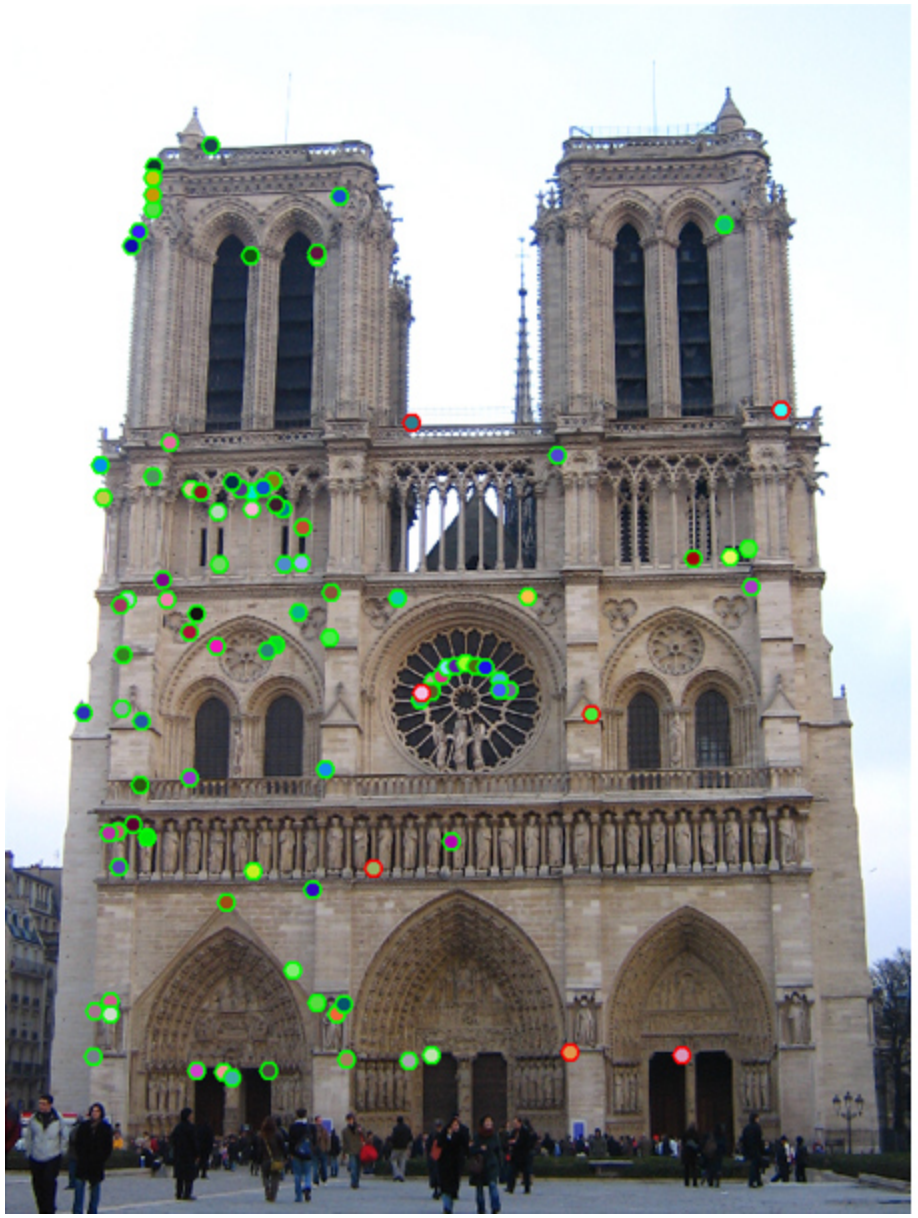
Bad

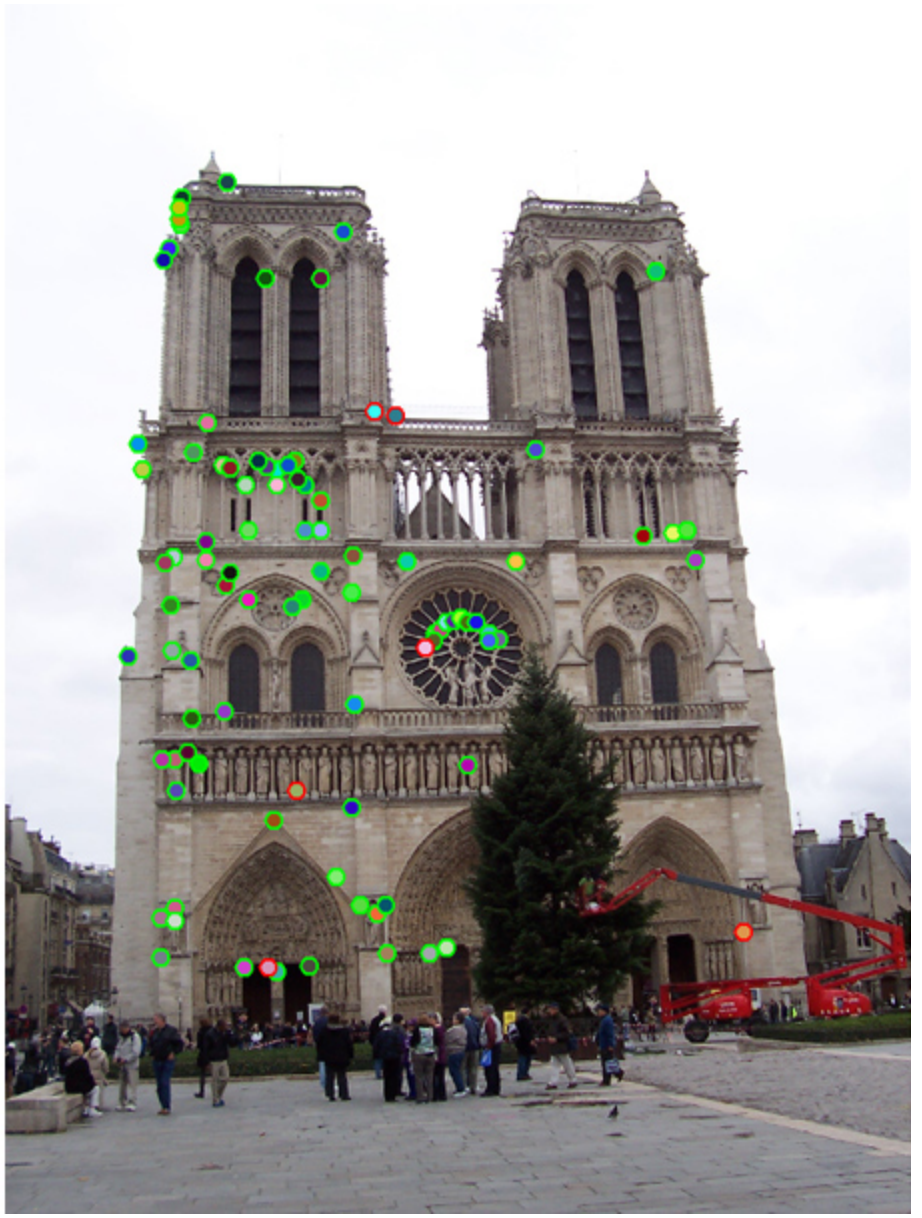
- Computational time grows quickly with fraction of outliers and number of parameters
- Not good for getting multiple fits

Common applications

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

How do we fit the best alignment?

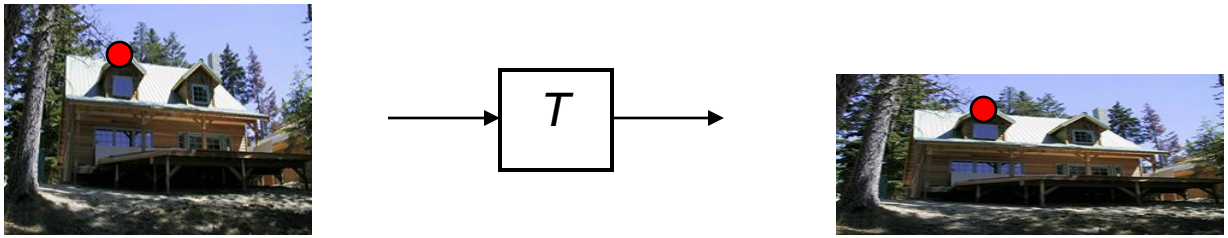




Alignment

- Alignment: find parameters of model that maps one set of points to another
- Typically want to solve for a global transformation that accounts for *most* true correspondences
- Difficulties
  - Noise (typically 1-3 pixels)
  - Outliers (often 50%)
  - Many-to-one matches or multiple objects

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)

For linear transformations, we can represent T as a matrix

$$p' = \mathbf{T}p$$

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

Common transformations

original

Transformed

translation

rotation

aspect

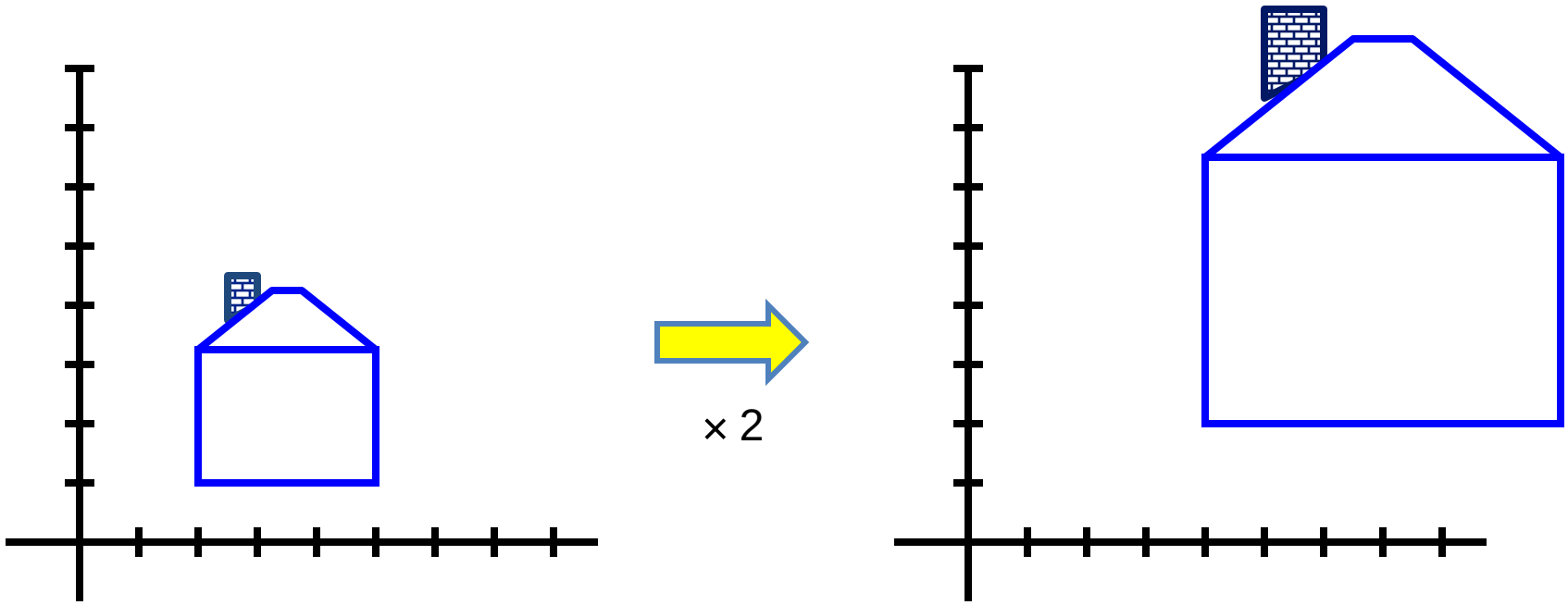
affine

perspective

Slide credit (next few slides):
A. Efros and/or S. Seitz

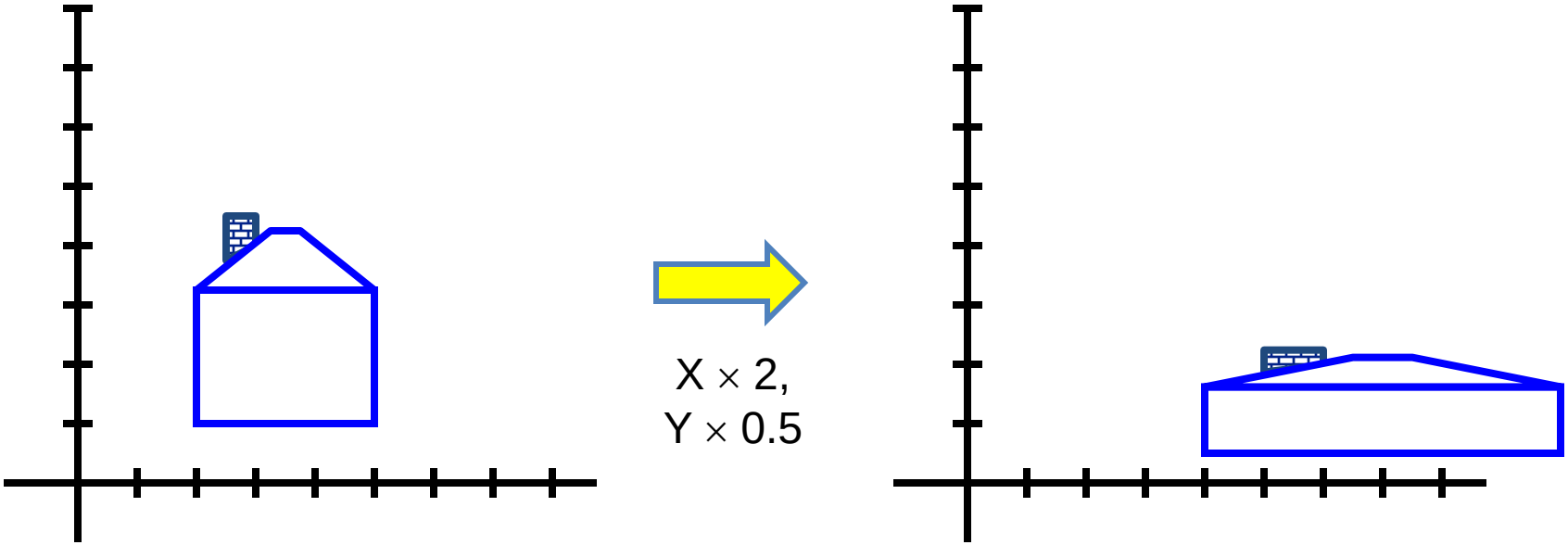
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component:



Scaling

- Scaling operation:

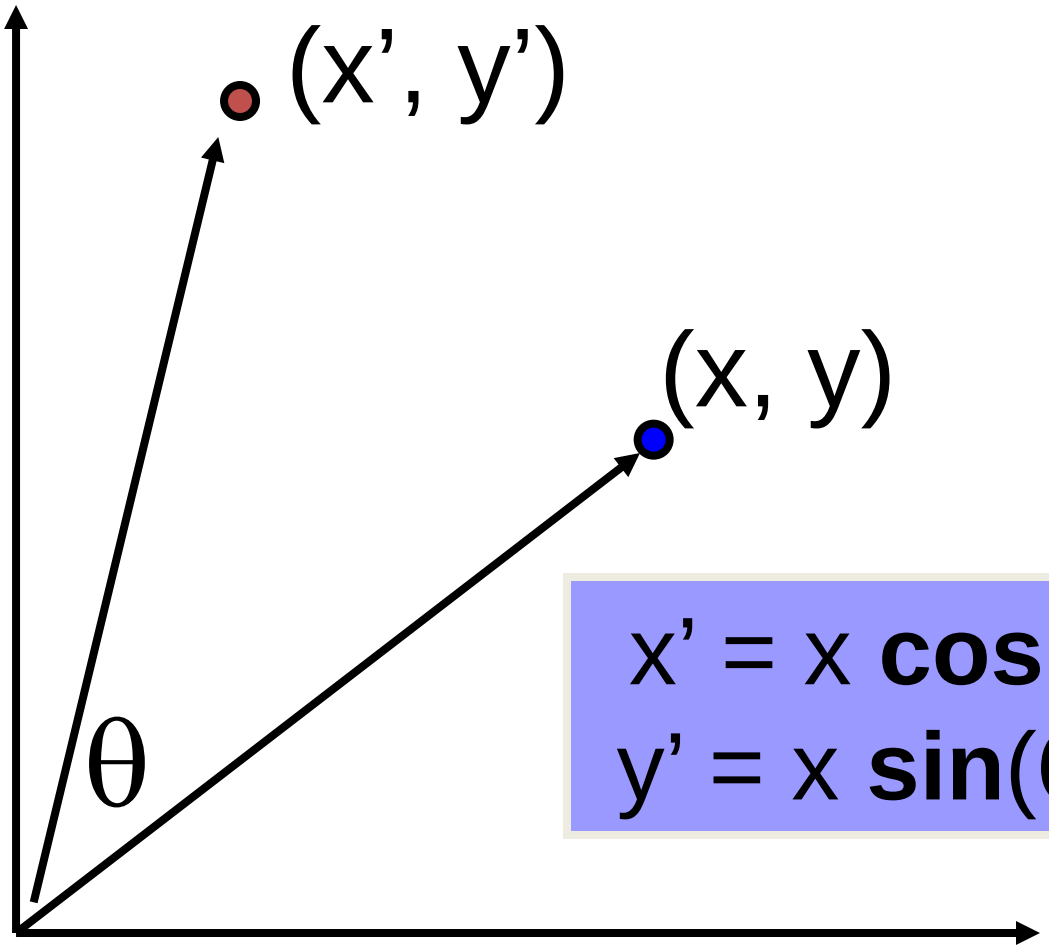
$$x' = ax$$

$$y' = by$$

- Or, in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

2-D Rotation



$$x' = x\ \mathbf{cos}(\theta) - y\ \mathbf{sin}(\theta)$$
$$y' = x\ \mathbf{sin}(\theta) + y\ \mathbf{cos}(\theta)$$

2-D Rotation

This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

Even though sin(θ) and cos(θ) are nonlinear functions of θ,

- ***x' is a linear combination of x and y***
- ***y' is a linear combination of x and y***

What is the inverse transformation?

- Rotation by –θ
- For rotation matrices $\mathbf{R}^{-1} = \mathbf{R}^{T}$

Basic 2D transformations

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

Scale

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & \alpha_x \\ \alpha_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Shear

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

Rotate

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

Translate

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

Affine

Affine is any combination of translation, scale, rotation, shear

Affine Transformations

Affine transformations are combinations of

- Linear transformations, and
- Translations

Properties of affine transformations:

- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

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

or

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

Projective Transformations

Projective transformations are combos of

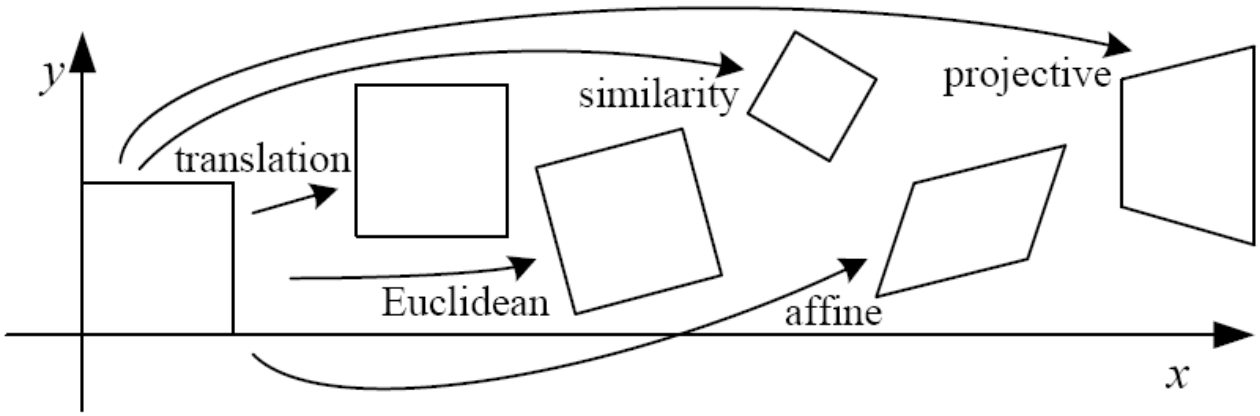
- Affine transformations, and
- Projective warps

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

Properties of projective transformations:

- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis
- Projective matrix is defined up to a scale (8 DOF)

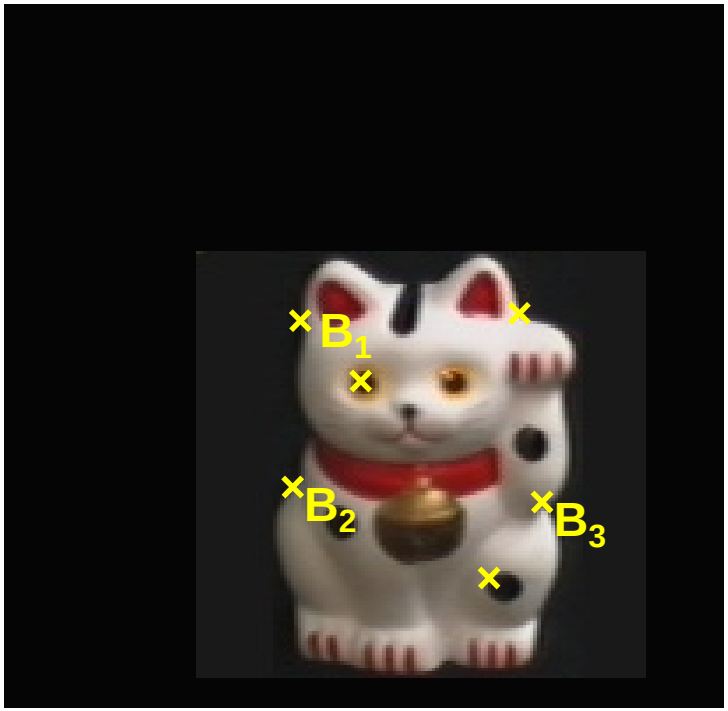
2D image transformations (reference table)



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	3	lengths $+\cdots$	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	4	angles $+\cdots$	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times3}$	6	parallelism $+\cdots$	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times3}$	8	straight lines	

Szeliski 2.1

Example: solving for translation

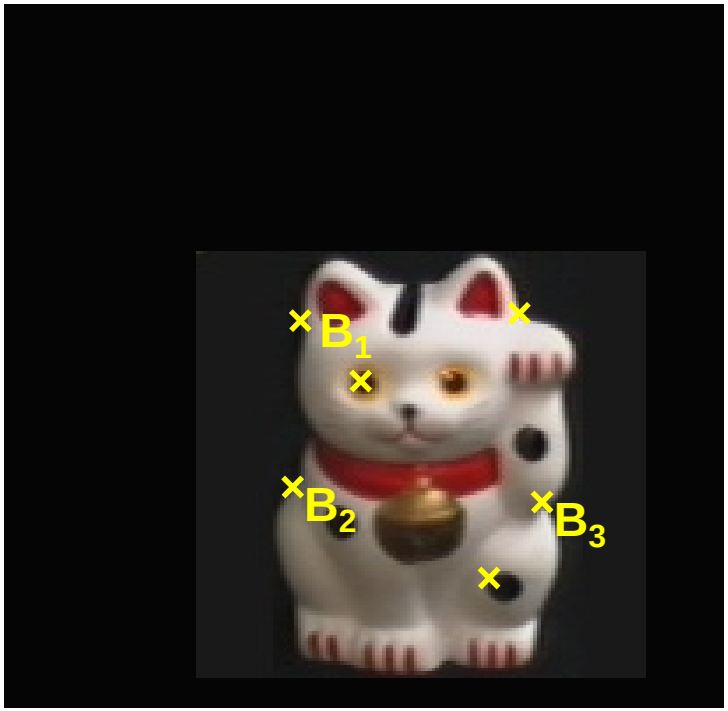


Given matched points in {A} and {B}, estimate the translation of the object

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Example: solving for translation

(t_x, t_y)



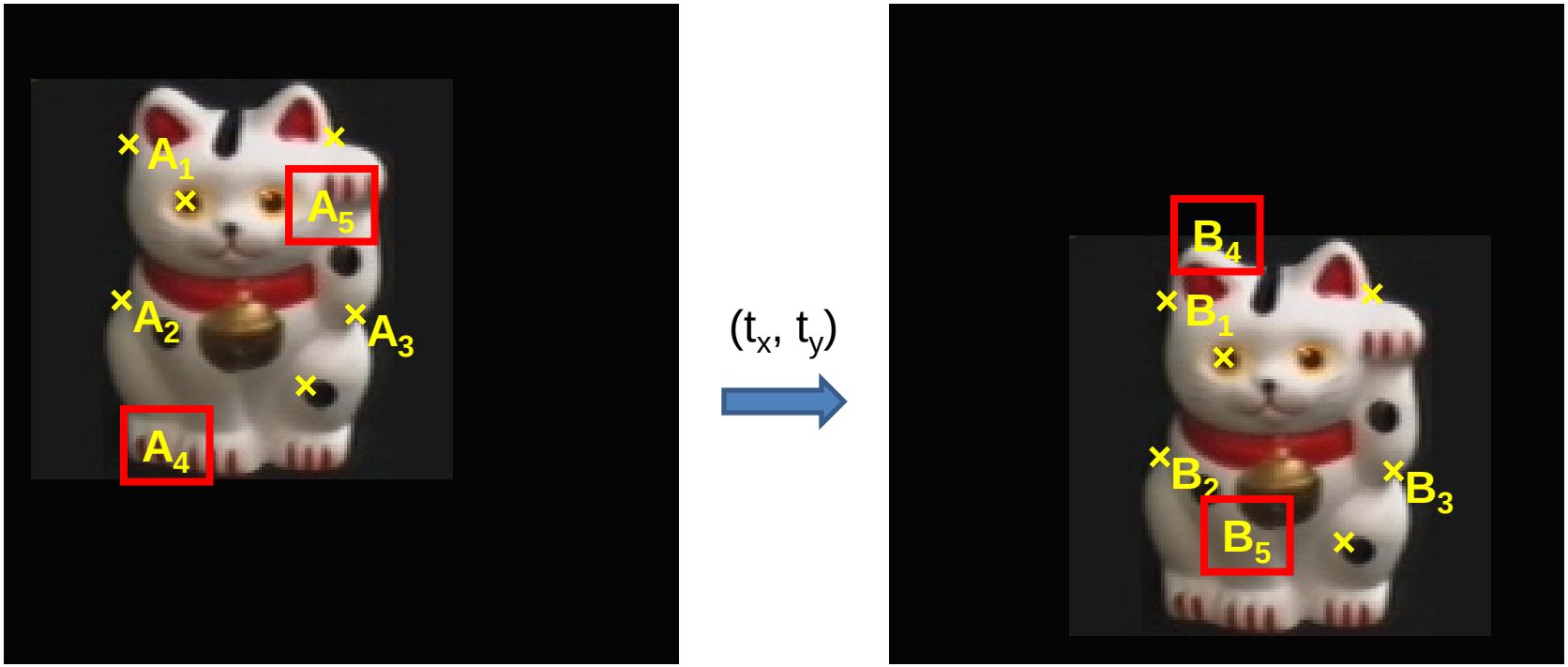
Least squares solution

1. Write down objective function
2. Derived solution
   a) Compute derivative
   b) Compute solution
3. Computational solution
   a) Write in form Ax=b
   b) Solve using pseudo-inverse or eigenvalue decomposition

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x_1^B - x_1^A \\ y_1^B - y_1^A \\ \vdots \\ x_n^B - x_n^A \\ y_n^B - y_n^A \end{bmatrix}$$

Example: solving for translation



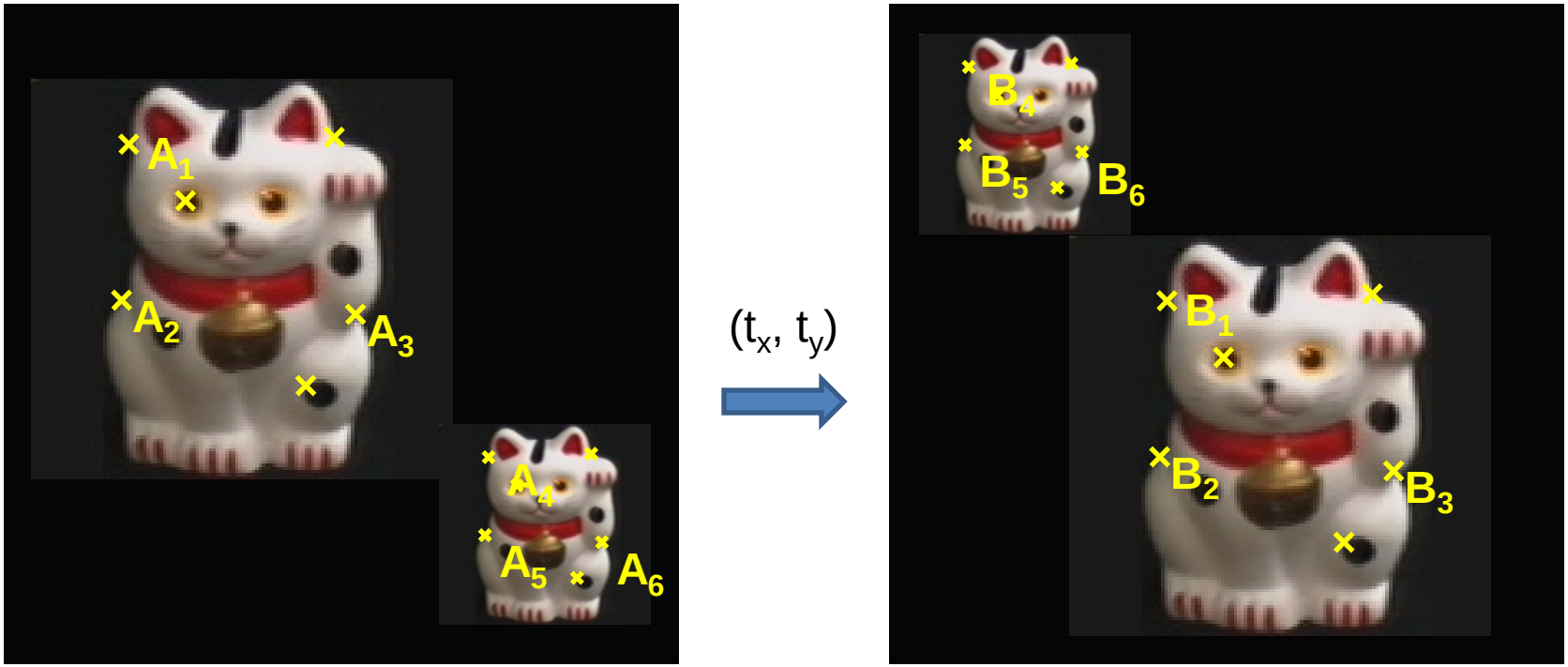
Problem: outliers

RANSAC solution

1. Sample a set of matching points (1 pair)
2. Solve for transformation parameters
3. Score parameters with number of inliers
4. Repeat steps 1-3 N times

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Example: solving for translation



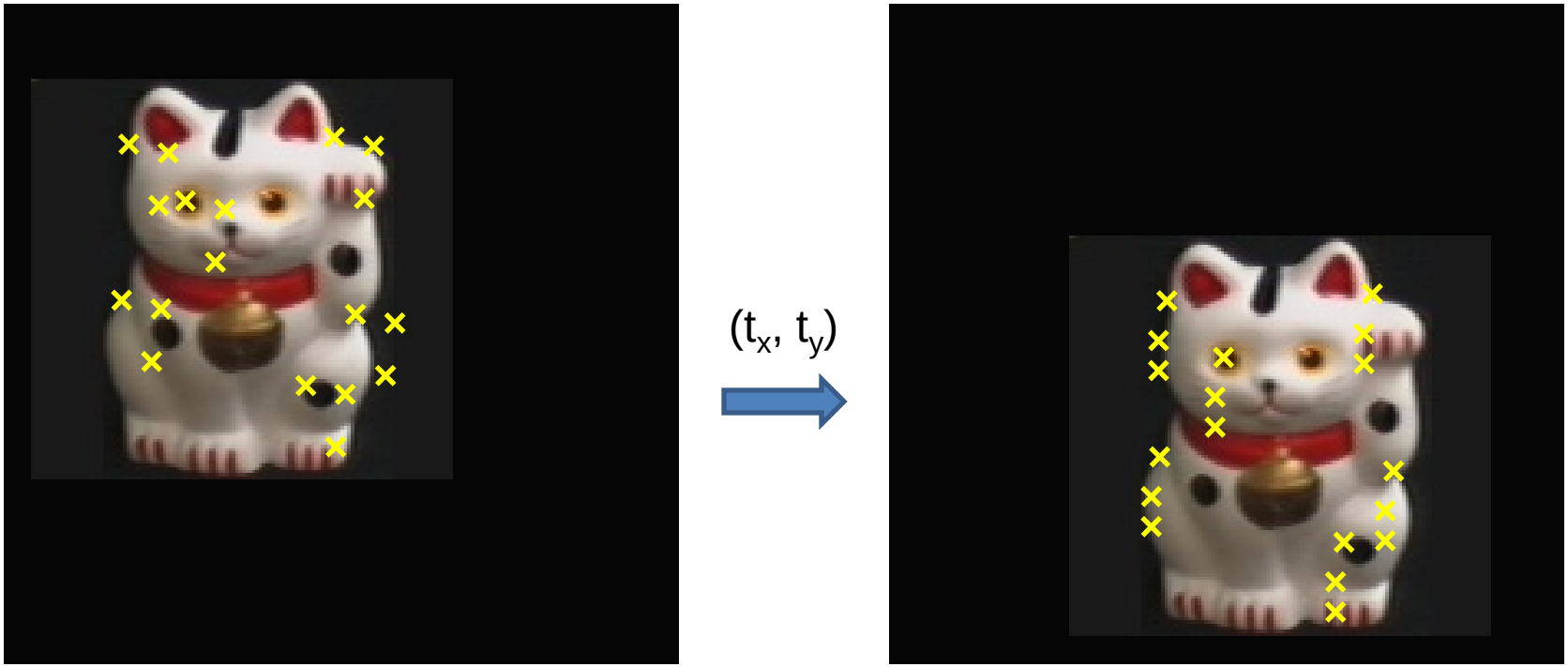
Problem: outliers, multiple objects, and/or many-to-one matches

Hough transform solution

1. Initialize a grid of parameter values
2. Each matched pair casts a vote for consistent values
3. Find the parameters with the most votes
4. Solve using least squares with inliers

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Example: solving for translation

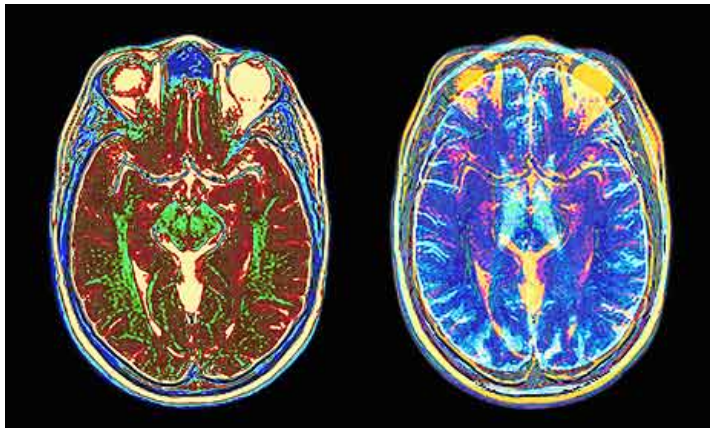


Problem: no initial guesses for correspondence

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

What if you want to align but have no prior matched pairs?

- Hough transform and RANSAC not applicable
- Important applications



Medical imaging: match brain scans or contours

Robotics: match point clouds

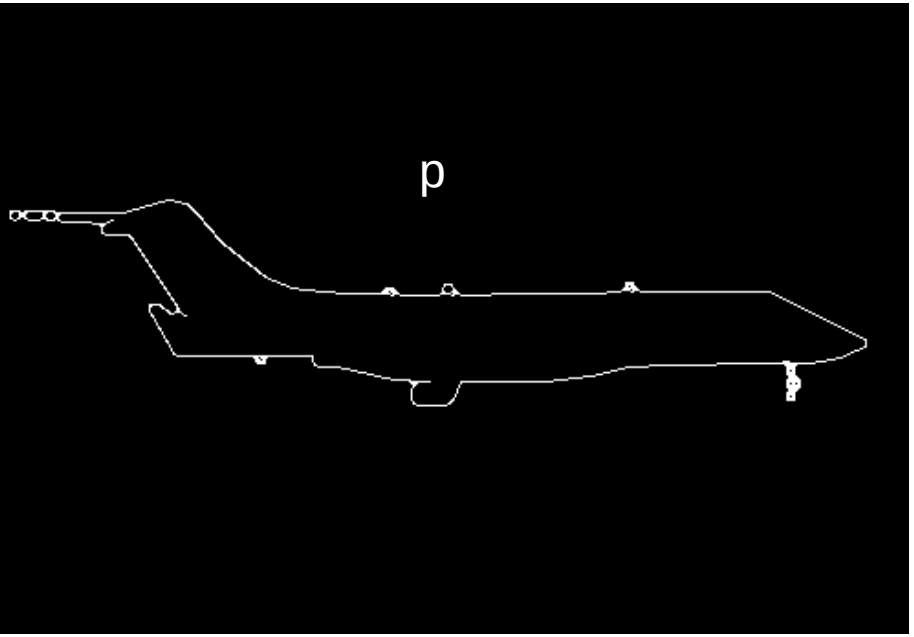
Iterative Closest Points (ICP) Algorithm

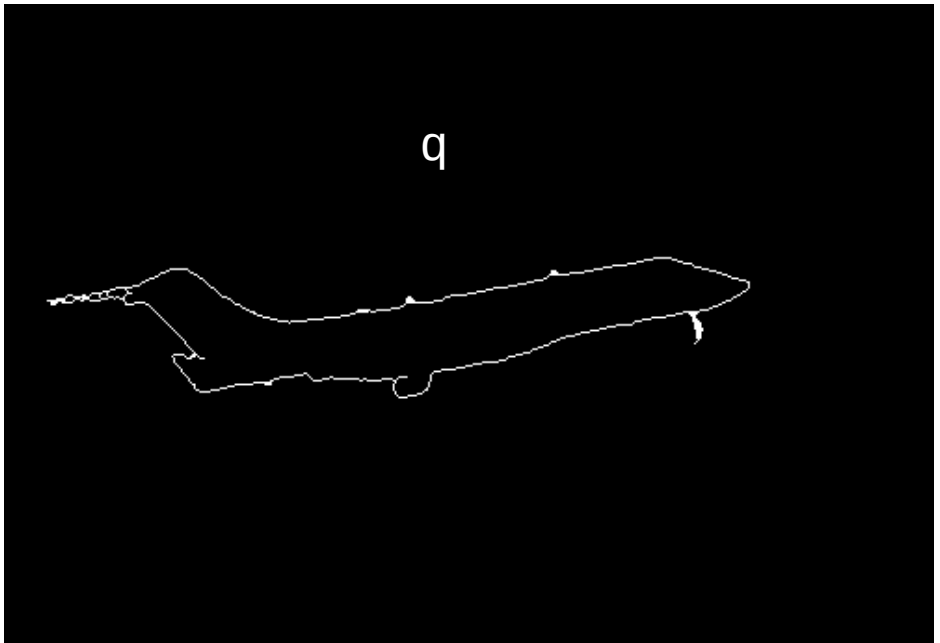
Goal: estimate transform between two dense sets of points

1. **Initialize** transformation (e.g., compute difference in means and scale)
2. **Assign** each point in {Set 1} to its nearest neighbor in {Set 2}
3. **Estimate** transformation parameters
   - e.g., least squares or robust least squares
4. **Transform** the points in {Set 1} using estimated parameters
5. **Repeat** steps 2-4 until change is very small

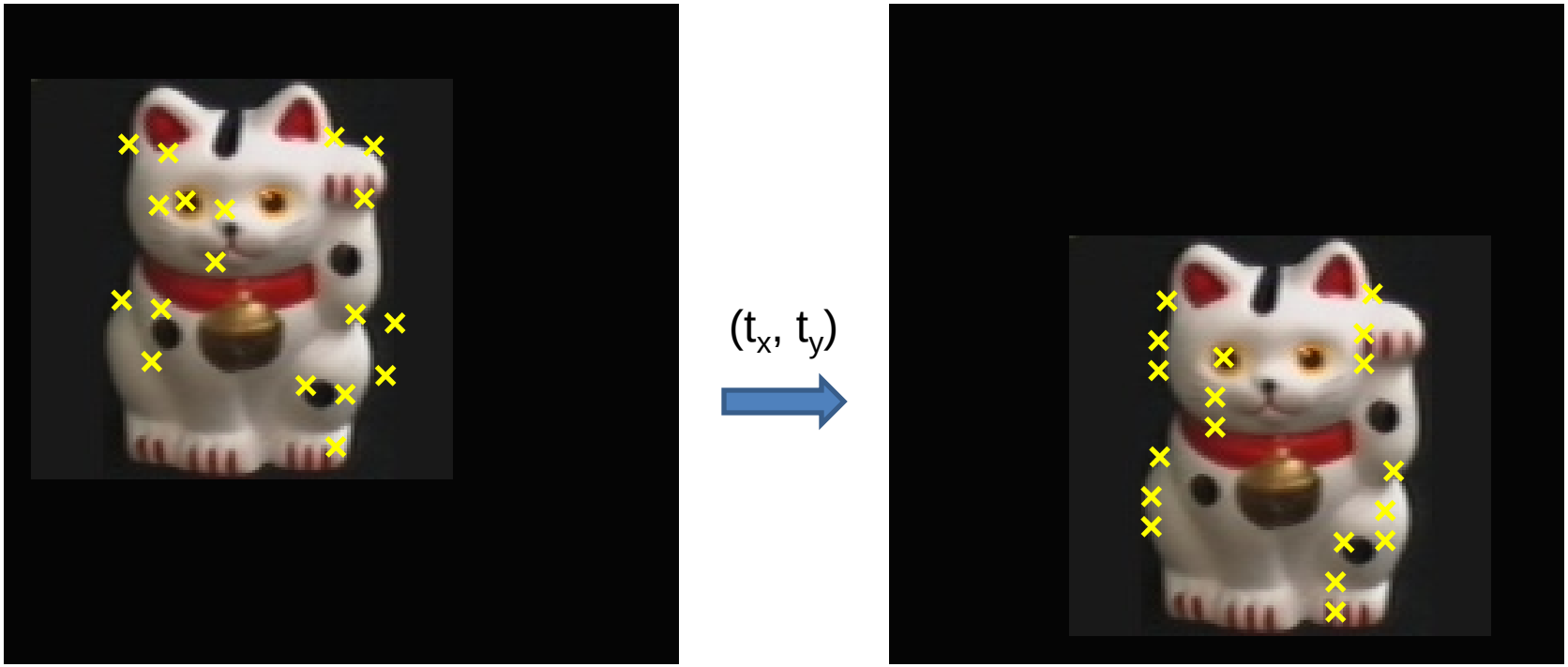
Example: aligning boundaries

1. Extract edge pixels $p_1 .. p_n$ and $q_1 .. q_m$
2. Compute initial transformation (e.g., compute translation and scaling by center of mass, variance within each image)
3. Get nearest neighbors: for each point p_i find corresponding $\text{match(i)} = \text{argmin}_j \, dist(p_i, q_j)$
4. Compute transformation ***T*** based on matches
5. Warp points ***p*** according to ***T***
6. Repeat 3-5 until convergence





Example: solving for translation



Problem: no initial guesses for correspondence

ICP solution

1. Find nearest neighbors for each point
2. Compute transform using matches
3. Move points using transform
4. Repeat steps 1-3 until convergence

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Algorithm Summary

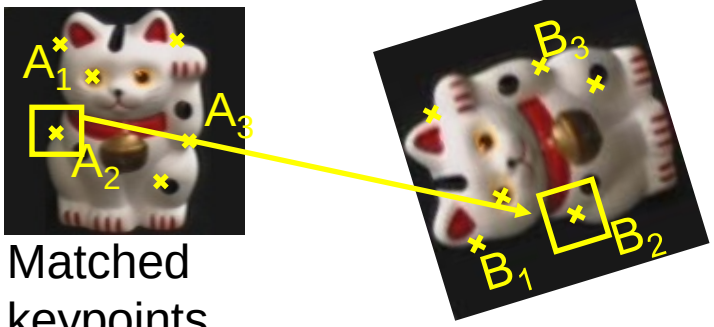
- Least Squares Fit
  - closed form solution
  - robust to noise
  - not robust to outliers
- Robust Least Squares
  - improves robustness to noise
  - requires iterative optimization
- Hough transform
  - robust to noise and outliers
  - can fit multiple models
  - only works for a few parameters (1-4 typically)
- RANSAC
  - robust to noise and outliers
  - works with a moderate number of parameters (e.g, 1-8)
- Iterative Closest Point (ICP)
  - For local alignment only: does not require initial correspondences

Object Instance Recognition

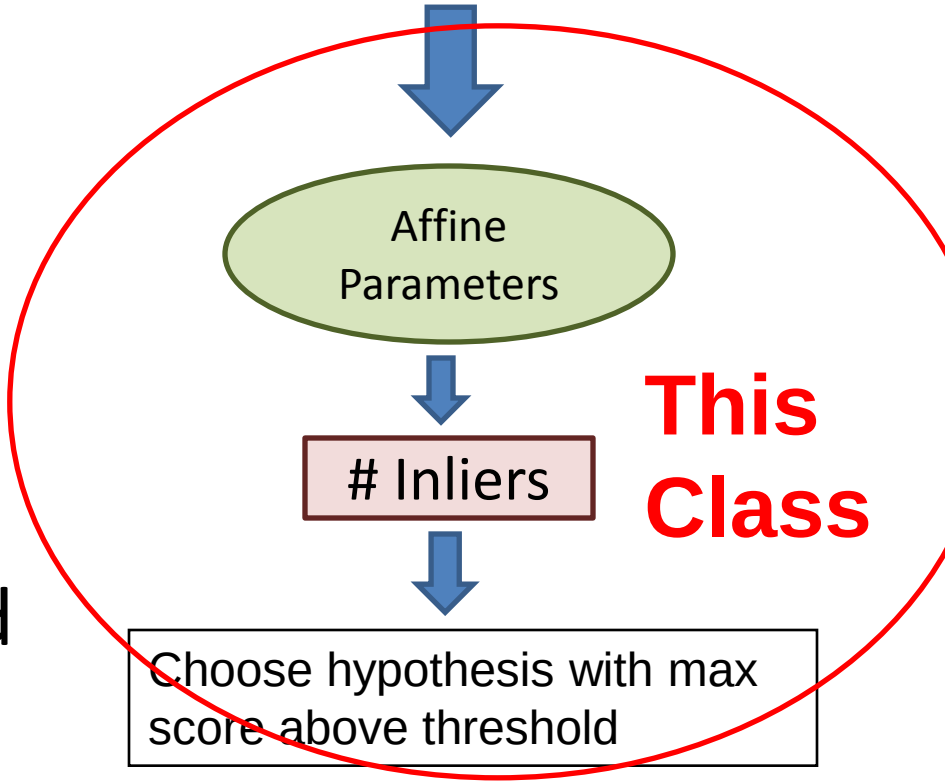
1. Match keypoints to object model

2. Solve for affine transformation parameters

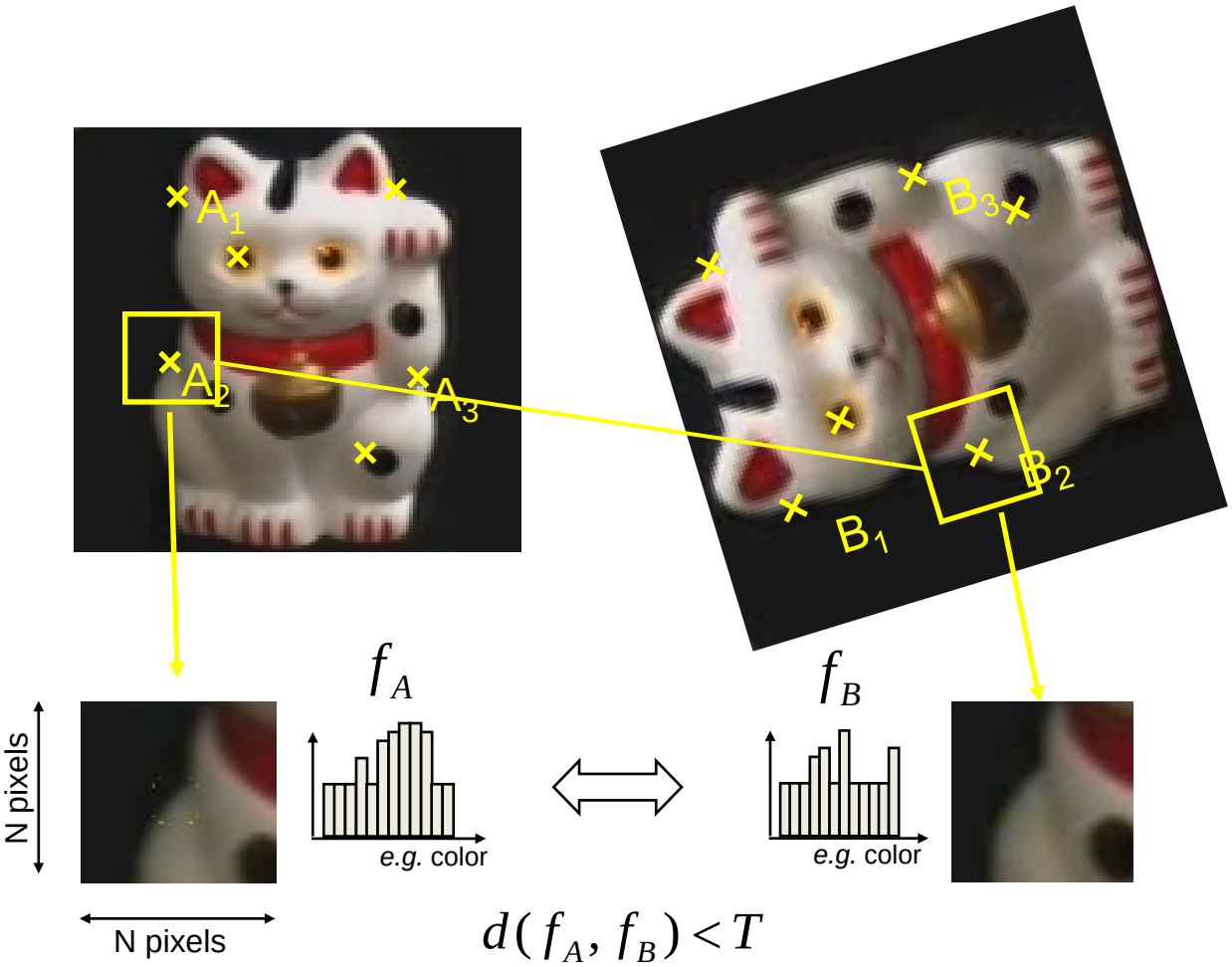
3. Score by inliers and choose solutions with score above threshold



Matched keypoints

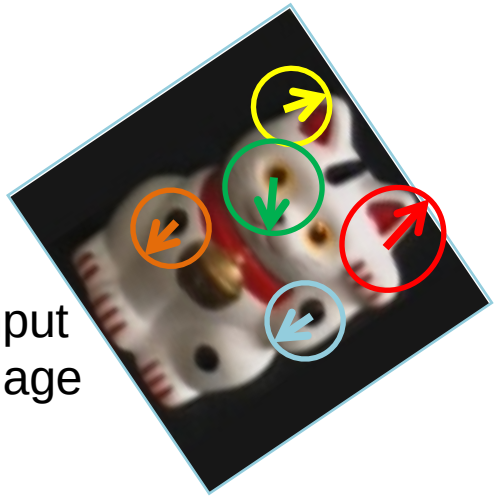


Overview of Keypoint Matching



1. **Find a set of distinctive key-points**
2. **Define a region around each keypoint**
3. **Extract and normalize the region content**
4. **Compute a local descriptor from the normalized region**
5. **Match local descriptors**

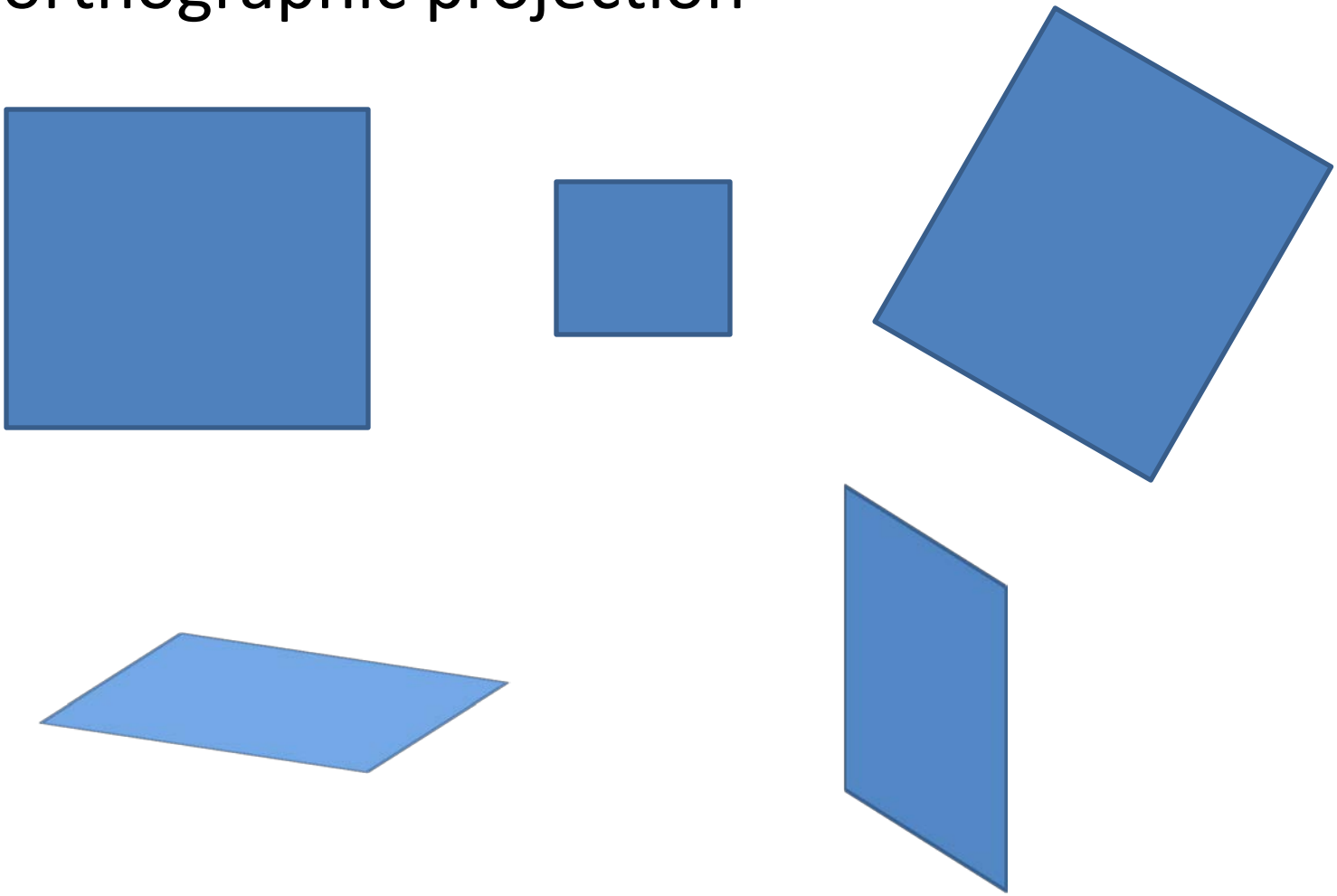
Finding the objects (overview)



1. Match interest points from input image to database image
2. Matched points vote for rough position/orientation/scale of object
3. Find position/orientation/scales that have at least three votes
4. Compute affine registration and matches using iterative least squares with outlier check
5. Report object if there are at least T matched points

Affine Object Model

- Accounts for 3D rotation of a surface under orthographic projection



Affine Object Model

- Accounts for 3D rotation of a surface under orthographic projection

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

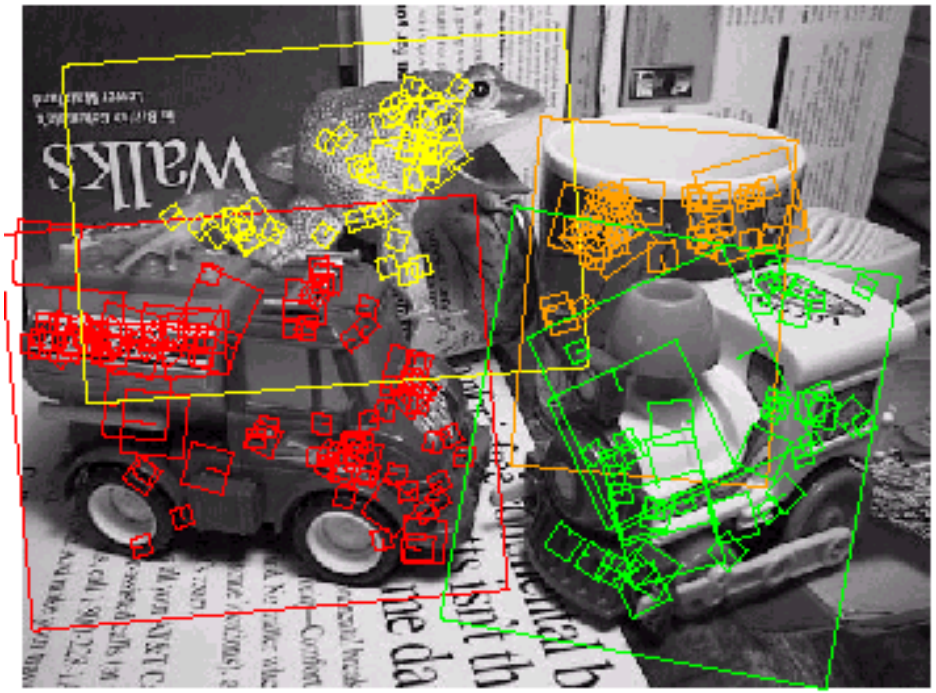
$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ & & \vdots & & & \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ \vdots \end{bmatrix}$$

$$\mathbf{x} = [\mathbf{A}^{\mathrm{T}}\mathbf{A}]^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

Finding the objects (SIFT, Lowe 2004)

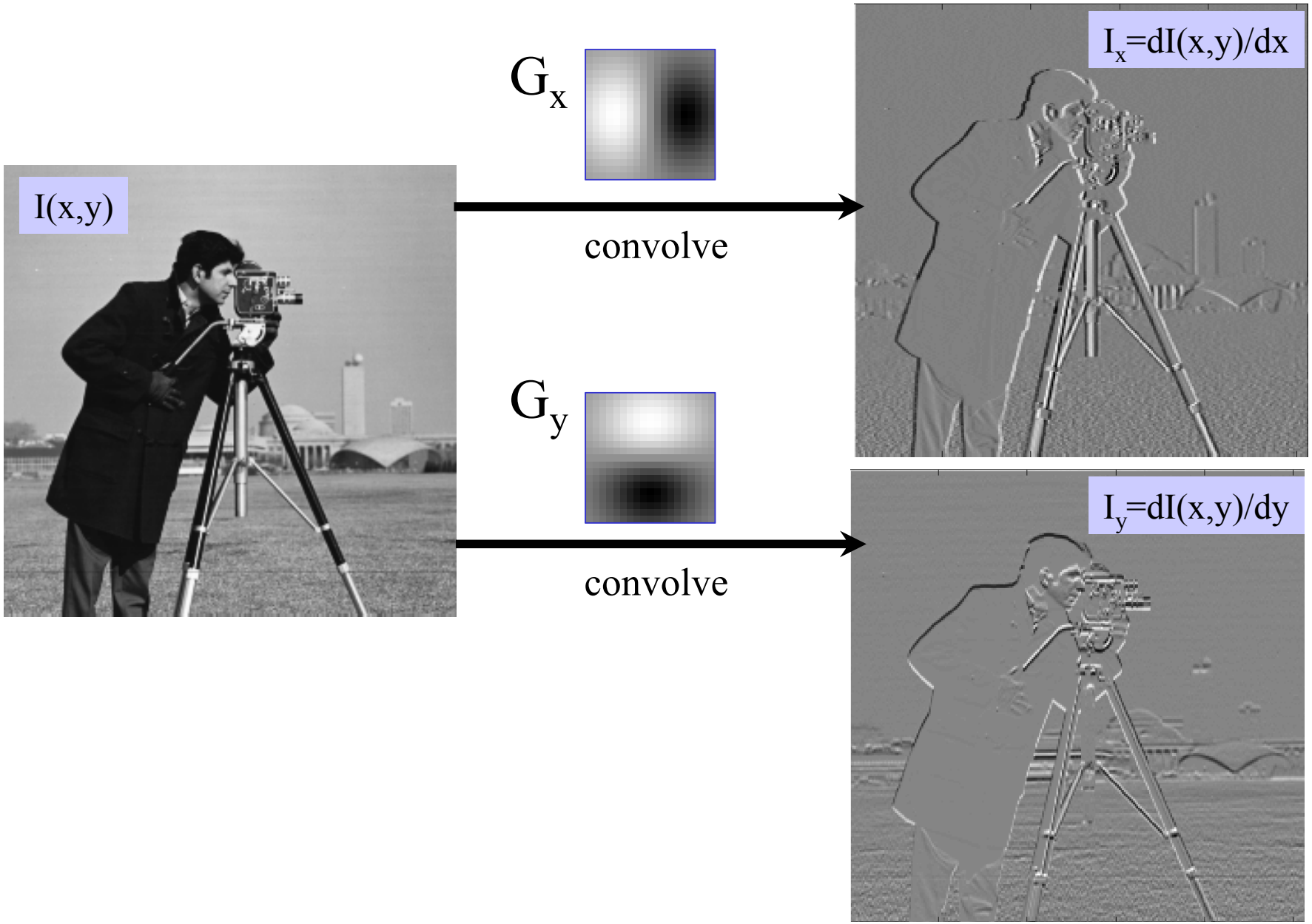
1. Match interest points from input image to database image
2. Get location/scale/orientation using Hough voting
   - In training, each point has known position/scale/orientation wrt whole object
   - Matched points vote for the position, scale, and orientation of the entire object
   - Bins for x, y, scale, orientation
     - Wide bins (0.25 object length in position, 2x scale, 30 degrees orientation)
     - Vote for two closest bin centers in each direction (16 votes total)
3. Geometric verification
   - For each bin with at least 3 keypoints
   - Iterate between least squares fit and checking for inliers and outliers
4. Report object if > T inliers (T is typically 3, can be computed to match some probabilistic threshold)

Examples of recognized objects



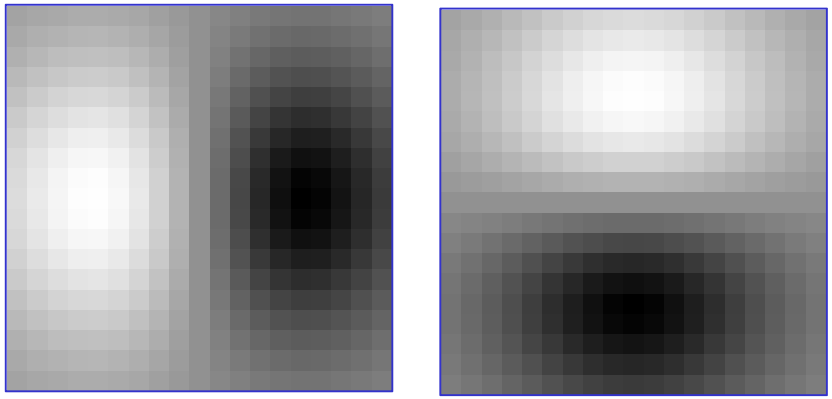
Dense Correspondence

Recall: Derivative of Gaussian Filter



Observe and Generalize

Derivative of Gaussian



Looks like vertical and horizontal step edges

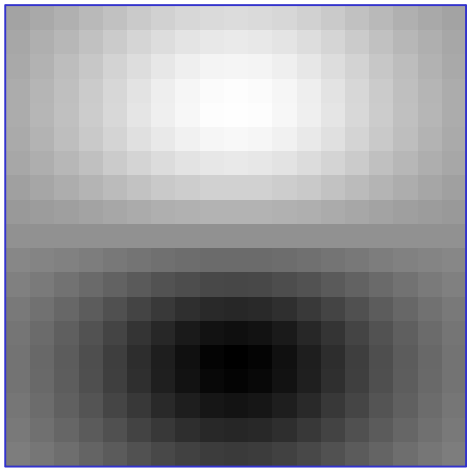
Key idea: Convolution (and cross correlation) with a filter can be viewed as comparing a little “picture” of what you want to find against all local regions in the image.

Robert Collins
CSE486, Penn State

Observe and Generalize

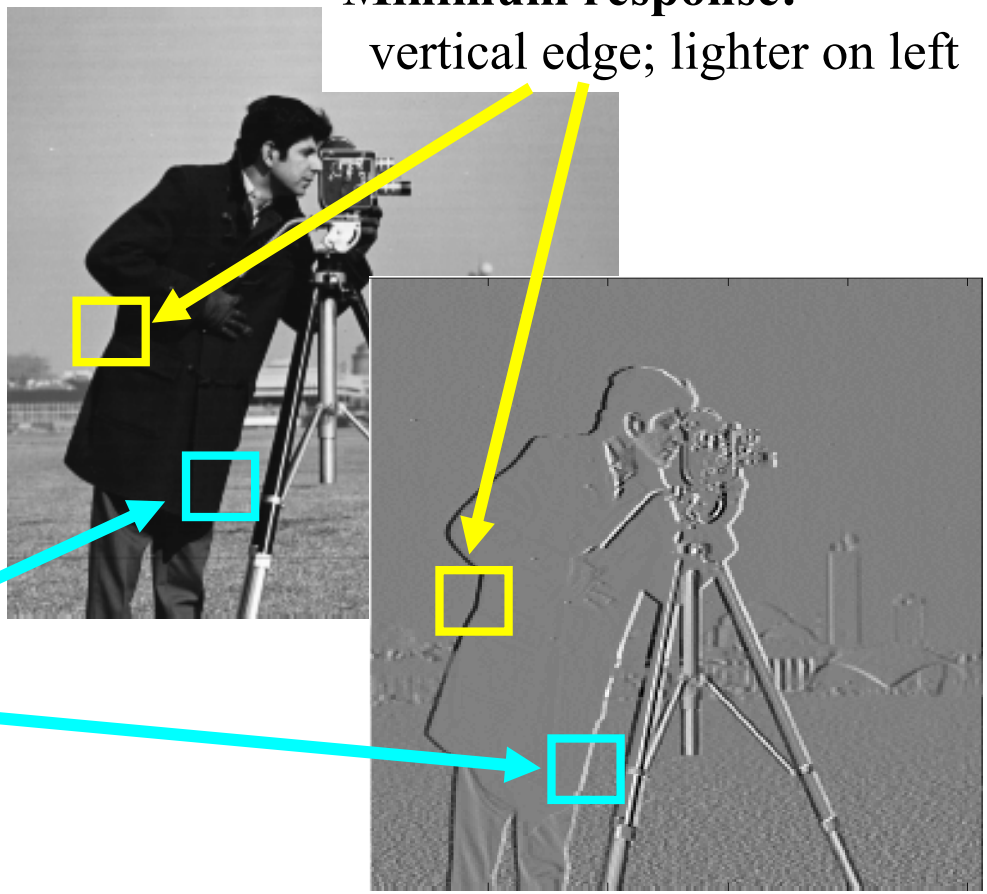
Key idea: Cross correlation with a filter can be viewed as comparing a little "picture" of what you want to find against all local regions in the image.

looks like vertical edge; lighter on right



Minimum response:
vertical edge; lighter on left

Maximum response:
vertical edge; lighter on right



Observe and Generalize

Key idea: Cross correlation with a filter can be viewed as comparing a little "picture" of what you want to find against all local regions in the image.

For this reason, it is sometimes called "matched filtering"

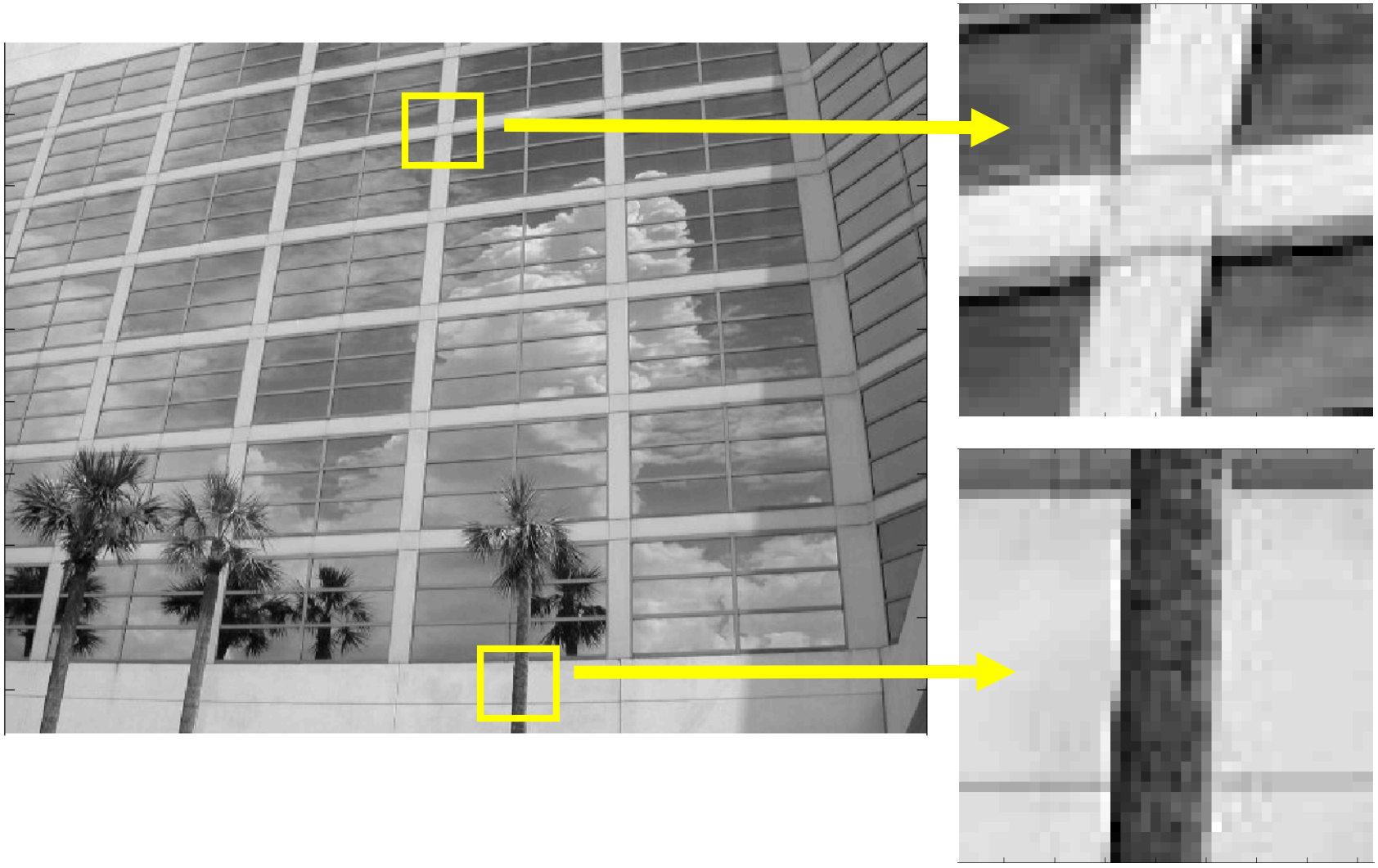
In fact, you can prove that the best linear operator for finding an image patch is essentially the patch itself

(using variational calculus, outside scope of our course).

Robert Collins
CSE486, Penn State

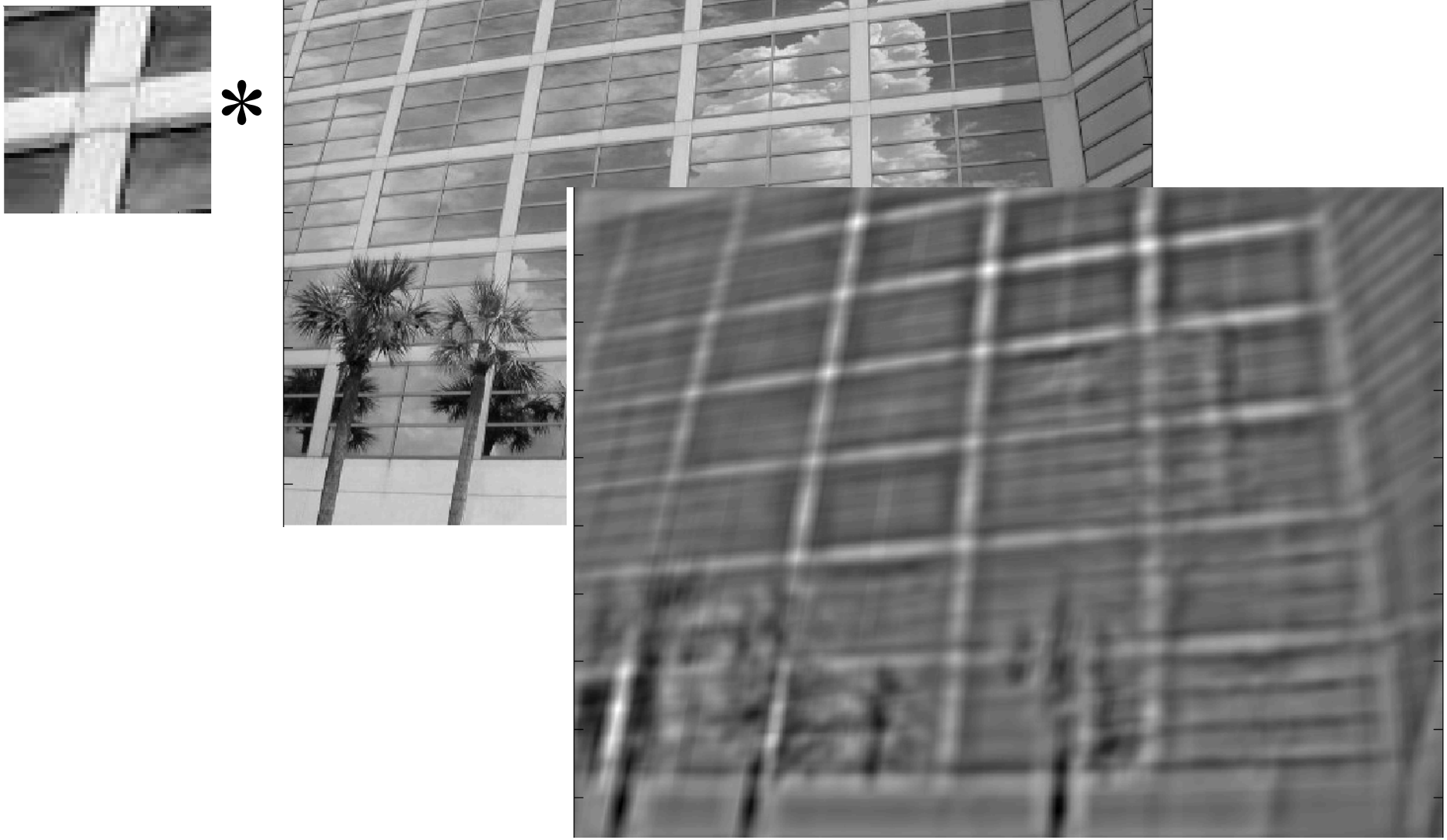
Template Matching

What if we cut little pictures out from an image, then tried convolve them with the same or other images?



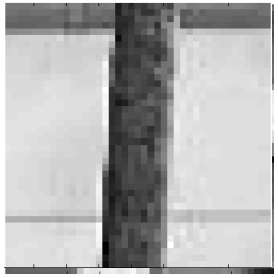
Robert Collins
CSE486, Penn State

Template Matching

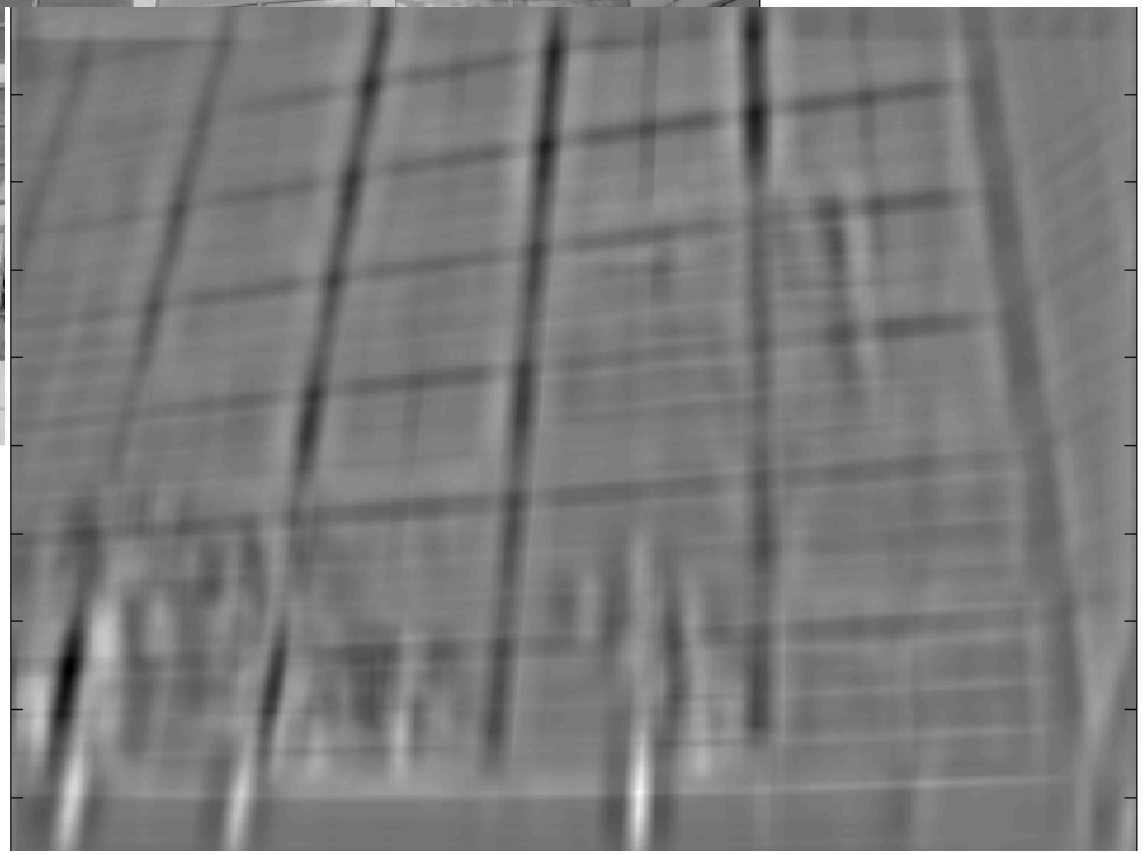


Robert Collins
CSE486, Penn State

Template Matching



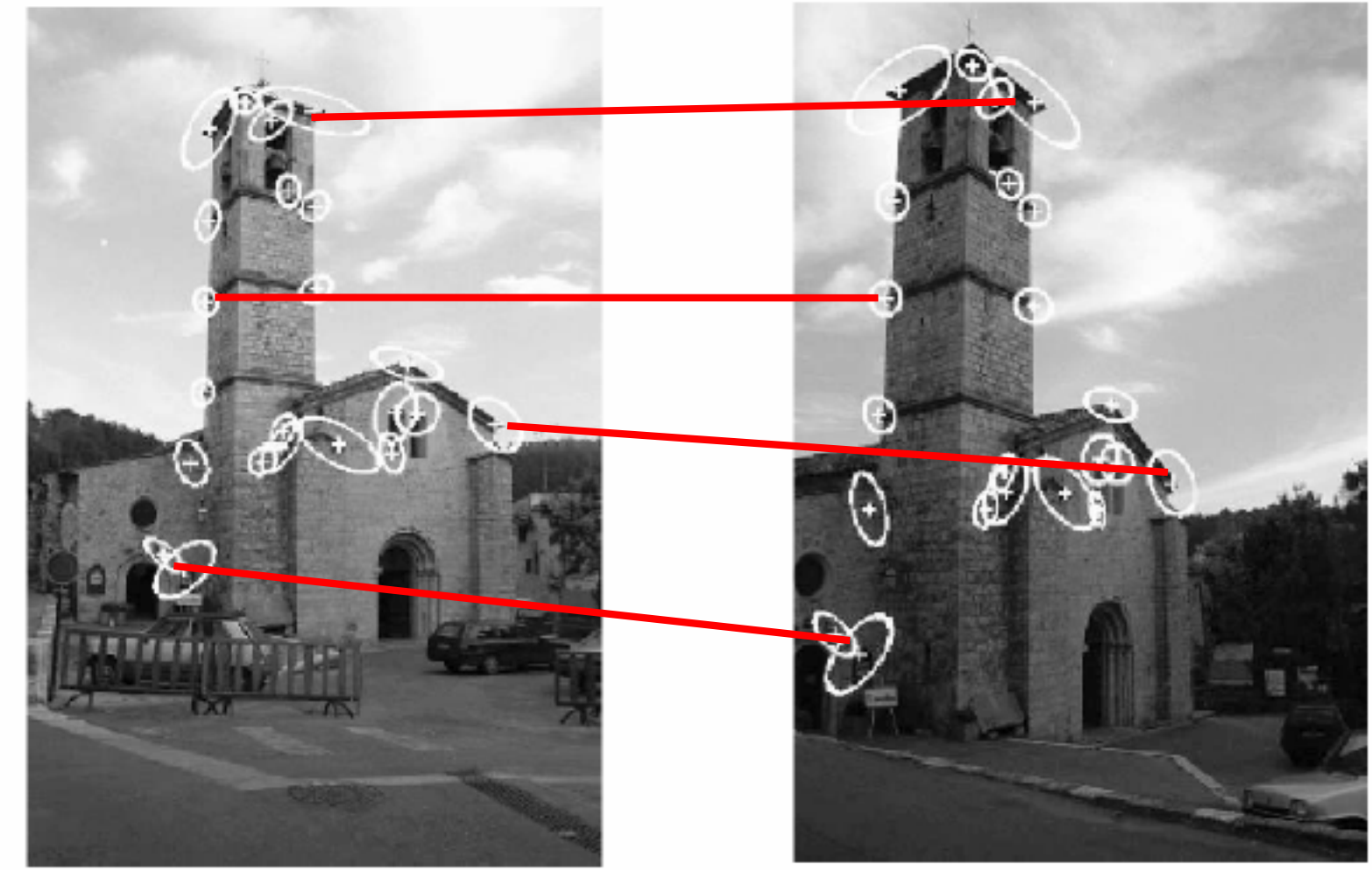
*



Robert Collins
CSE486, Penn State

Correspondence Problem

Vision tasks such as stereo and motion estimation require finding corresponding features across two or more views.



The Correspondence Problem

- Basic assumptions:
  - Most scene points are visible in both images
  - Corresponding image regions are similar
- These assumptions hold if:
  - The distance of points from the cameras is much larger than the distance between cameras

The Correspondence Problem

- Is a "search" problem:
  - Given an element in the left image, search for the corresponding element in the right image.
  - We will typically need geometric constraints to reduce the size of the search space
- We must choose:
  - Elements to match
  - A similarity measure to compare elements

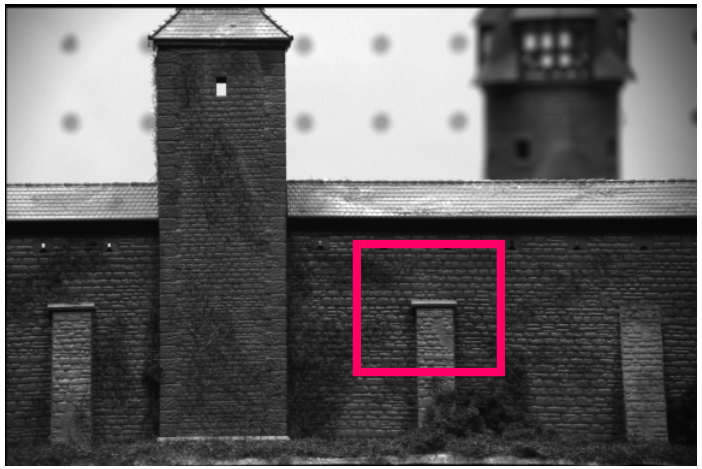
Correspondence Problem

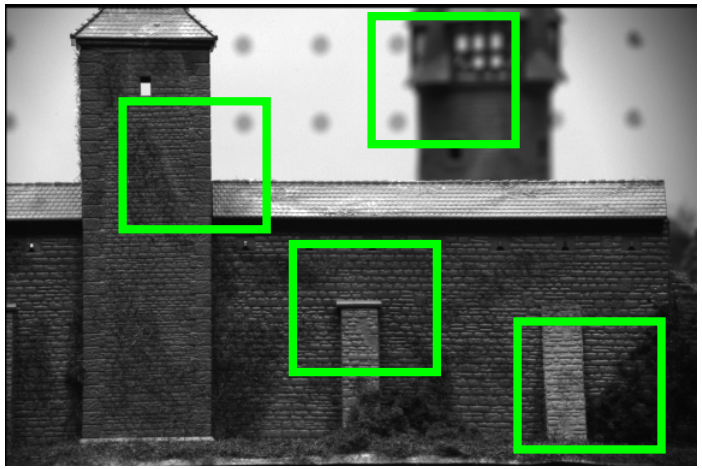
- Two classes of algorithms:
  - Correlation-based algorithms
    - Produce a DENSE set of correspondences
  - Feature-based algorithms
    - Produce a SPARSE set of correspondences

Robert Collins
CSE486, Penn State

Correlation-based Algorithms

Elements to be matched are image patches of fixed size





Task: what is the corresponding patch in a second image?

Robert Collins
CSE486, Penn State

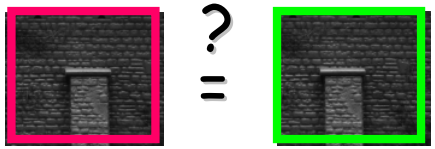
Correlation-based Algorithms

Task: what is the corresponding patch in a second image?

1) Need an appearance similarity function.

2) Need a search strategy to find location with highest similarity. Simplest (but least efficient) approach is exhaustive search.

Comparing Windows:



f g

Some possible measures:

$$\max_{[i,j]\in R} |f(i,j) - g(i,j)|$$

$$\sum_{[i,j]\in R} |f(i,j) - g(i,j)|$$

$$SSD = \sum_{[i,j]\in R} (f(i,j) - g(i,j))^2$$

$$C_{fg} = \sum_{[i,j]\in R} f(i,j)g(i,j)$$

Most popular

Correlation C_{fg}

$$C_{fg} = \sum_{[i,j] \in R} f(i,j) g(i,j)$$

If we are doing exhaustive search over all image patches in the second image, this becomes cross-correlation of a template with an image. We have seen this before – imfilter(im,template,'corr').

Example

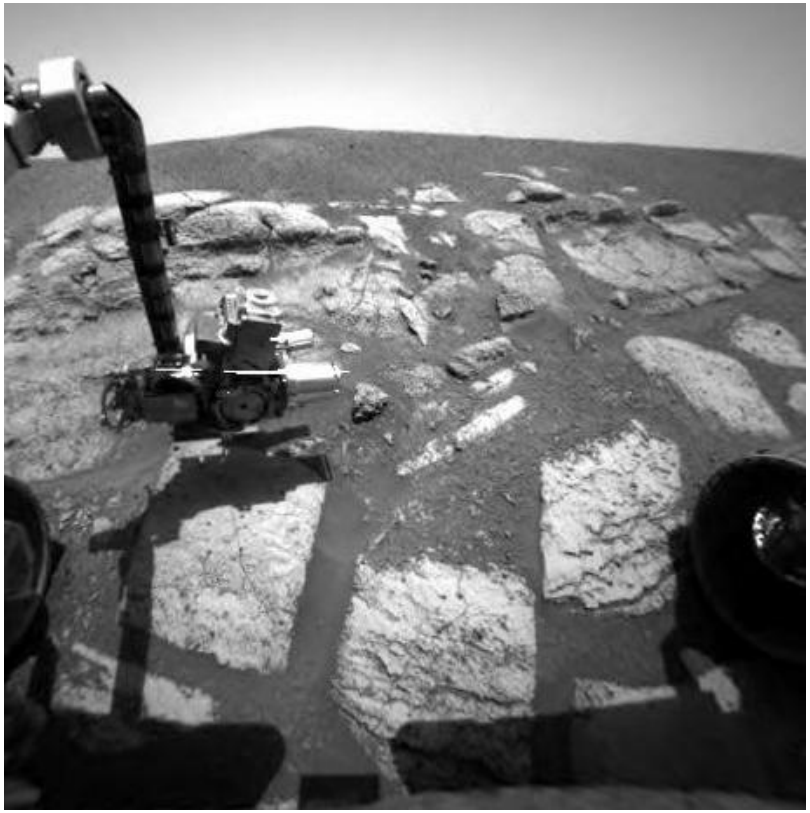


Image 1

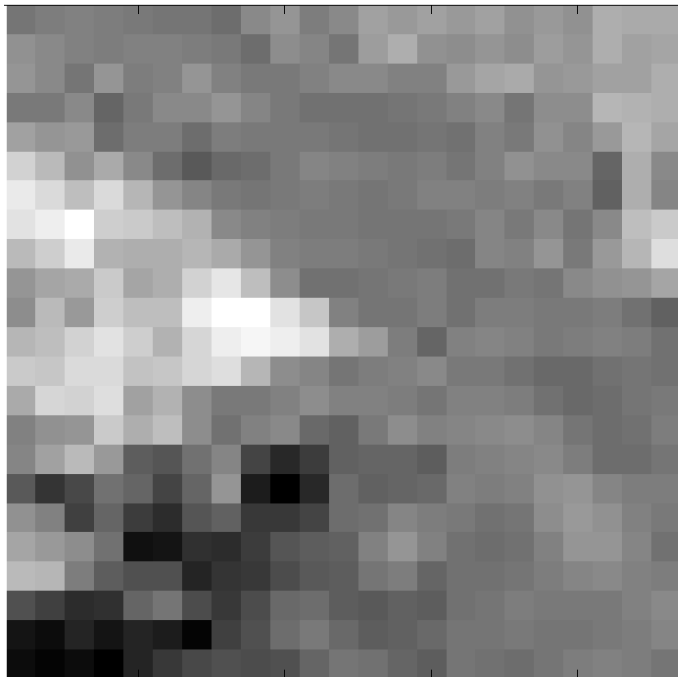
Image 2

Note: this is a stereo pair from the NASA mars rover. The rover is exploring the “El Capitan” formation.

Robert Collins
CSE486, Penn State

Example

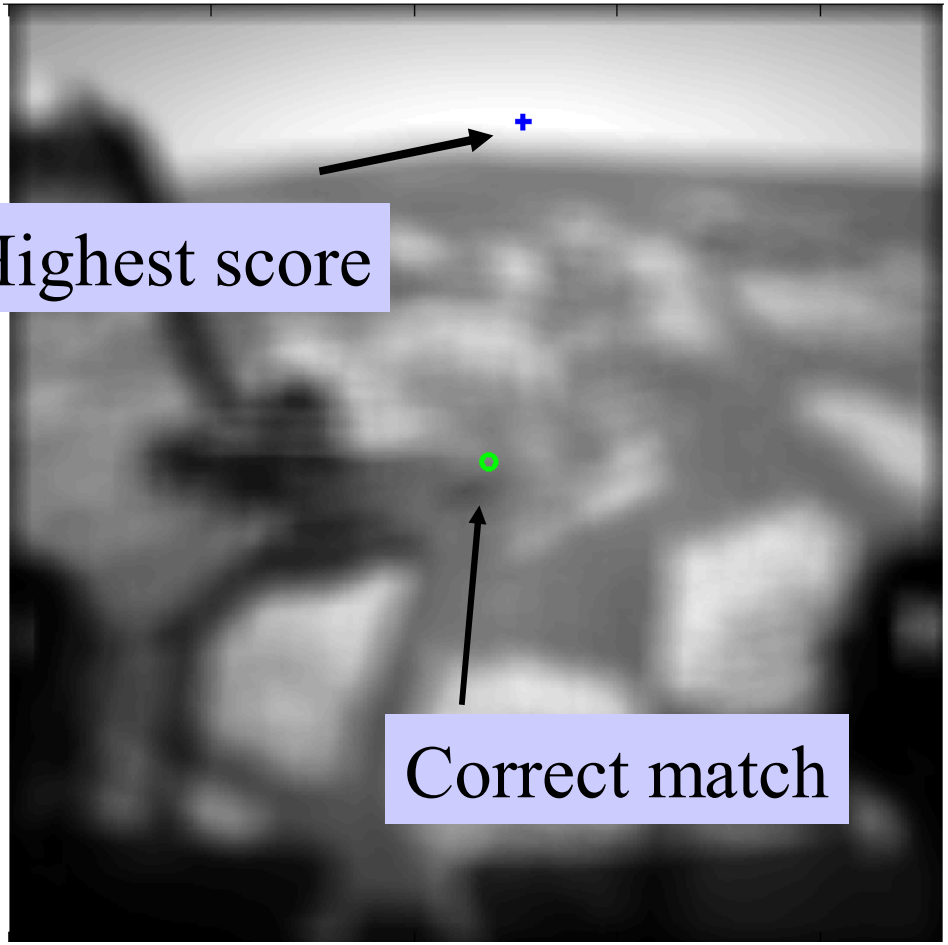
Image 1



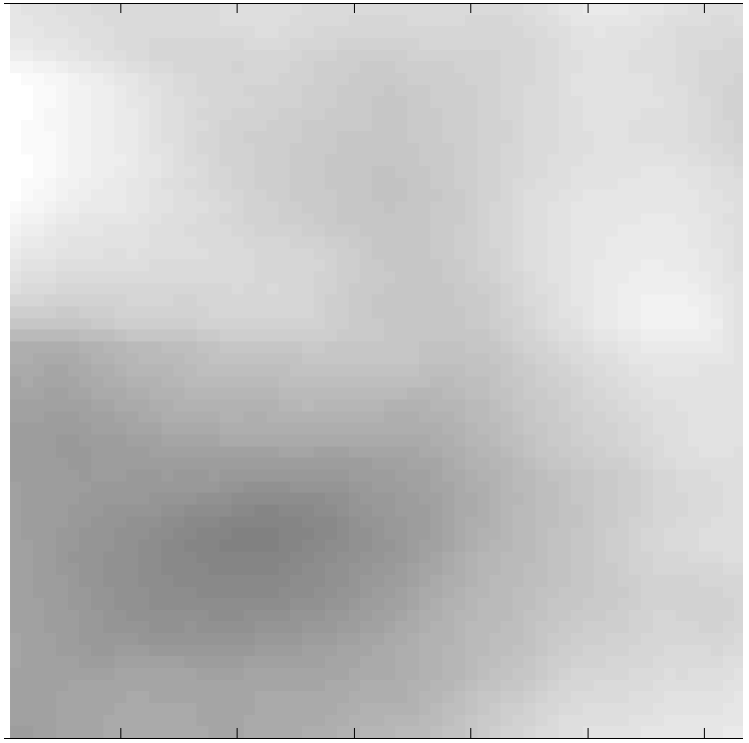
Template
(image patch)

Example: Raw Cross-correlation

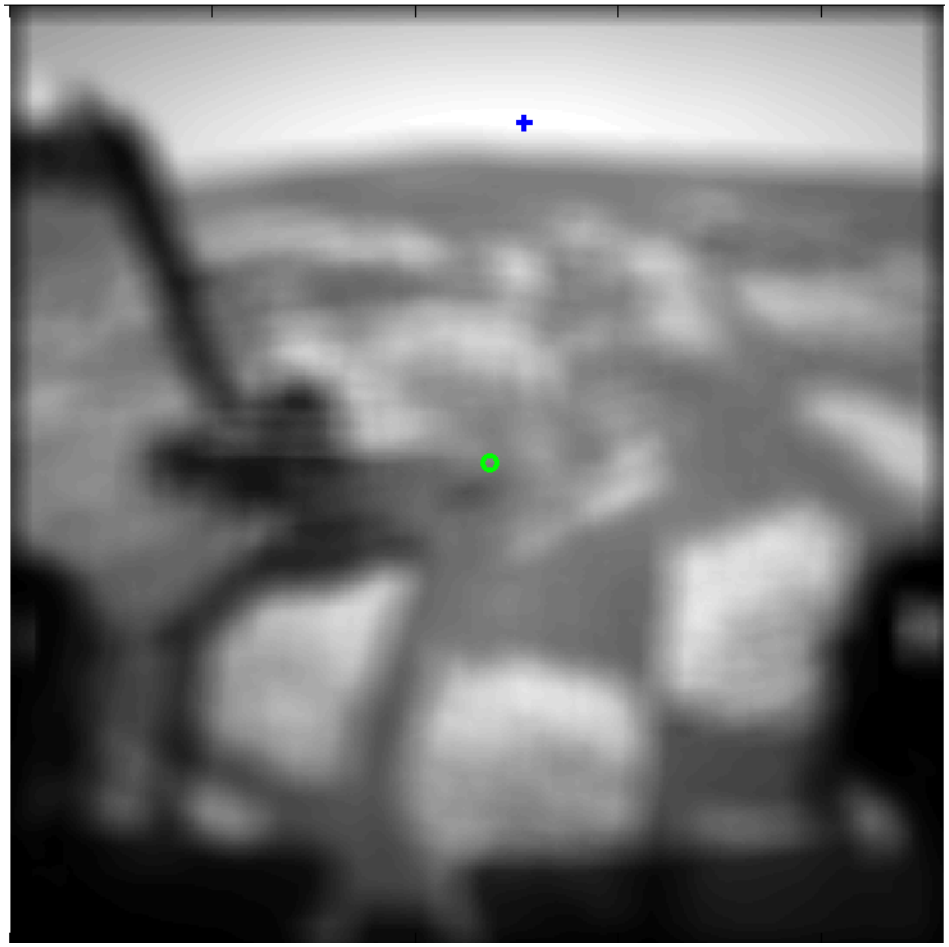
score = imfilter(image2,tmpl)



Score around correct match



Example: Cross-correlation



Note that score image looks a lot like a blurry version of image 2.

This clues us in to the problem with straight correlation with an image template.

Robert Collins
CSE486, Penn State

Problem with Correlation of Raw Image Templates

Consider correlation of template with an image of constant grey value:

a	b	c
d	e	f
g	h	i

⊗

v	v	v
v	v	v
v	v	v

Result: v*(a+b+c+d+e+f+g+h+i)

Robert Collins
CSE486, Penn State

Problem with Correlation of Raw Image Templates

Now consider correlation with a constant image that is twice as bright.

a	b	c
d	e	f
g	h	i

ⓧ

2v	2v	2v
2v	2v	2v
2v	2v	2v

Result: 2*v*(a+b+c+d+e+f+g+h+i)
> v*(a+b+c+d+e+f+g+h+i)

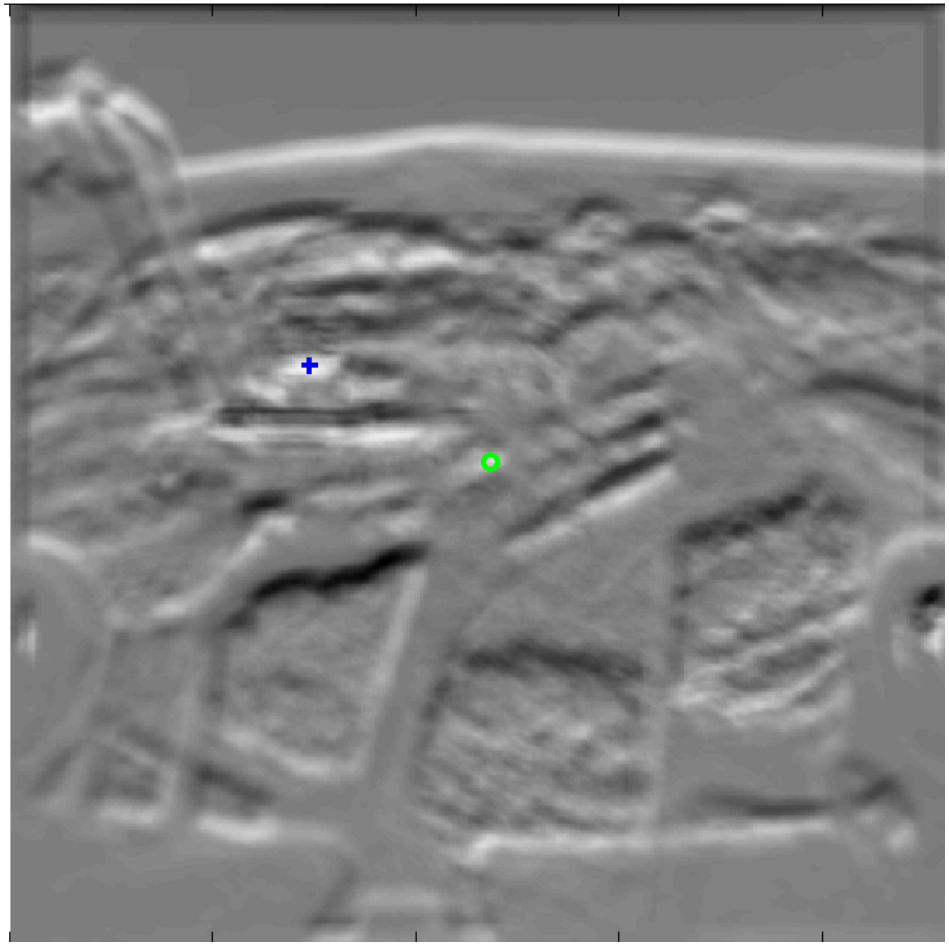
Larger score, regardless of what the template is!

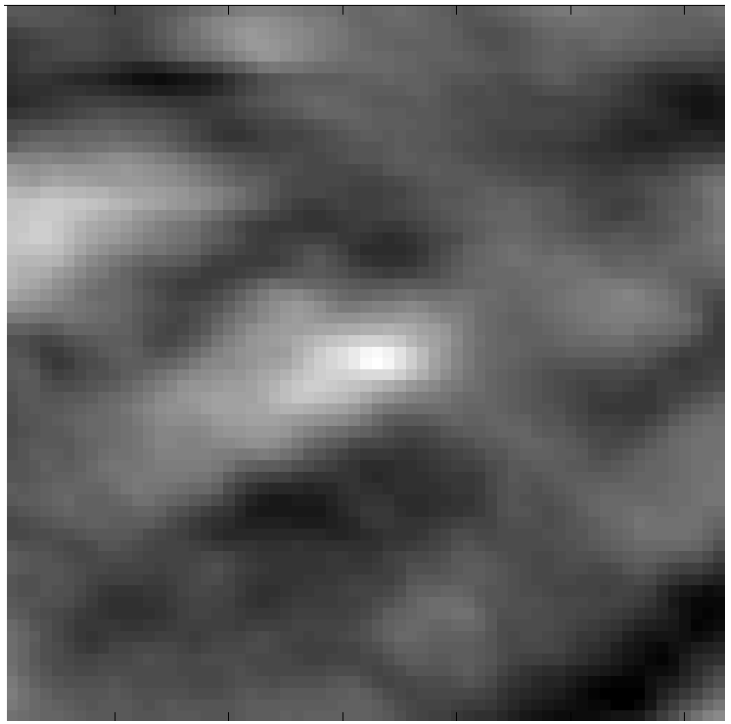
Solution

Subtract off the mean value of the template.

In this way, the correlation score is higher only when darker parts of the template overlap darker parts of the image, and brighter parts of the template overlap brighter parts of the image.

Correlation, zero-mean template





Better! But highest score is still not the correct match.

Note: highest score IS best within local neighborhood of correct match.

Robert Collins
CSE486, Penn State

"SSD" or "block matching" (Sum of Squared Differences)

$$\sum_{[i,j] \in R} (f(i,j) - g(i,j))^2$$

1) The most popular matching score.
2) We used it when deriving Harris corners
3) T&V claim it works better than cross-correlation

Relation between SSD and Correlation

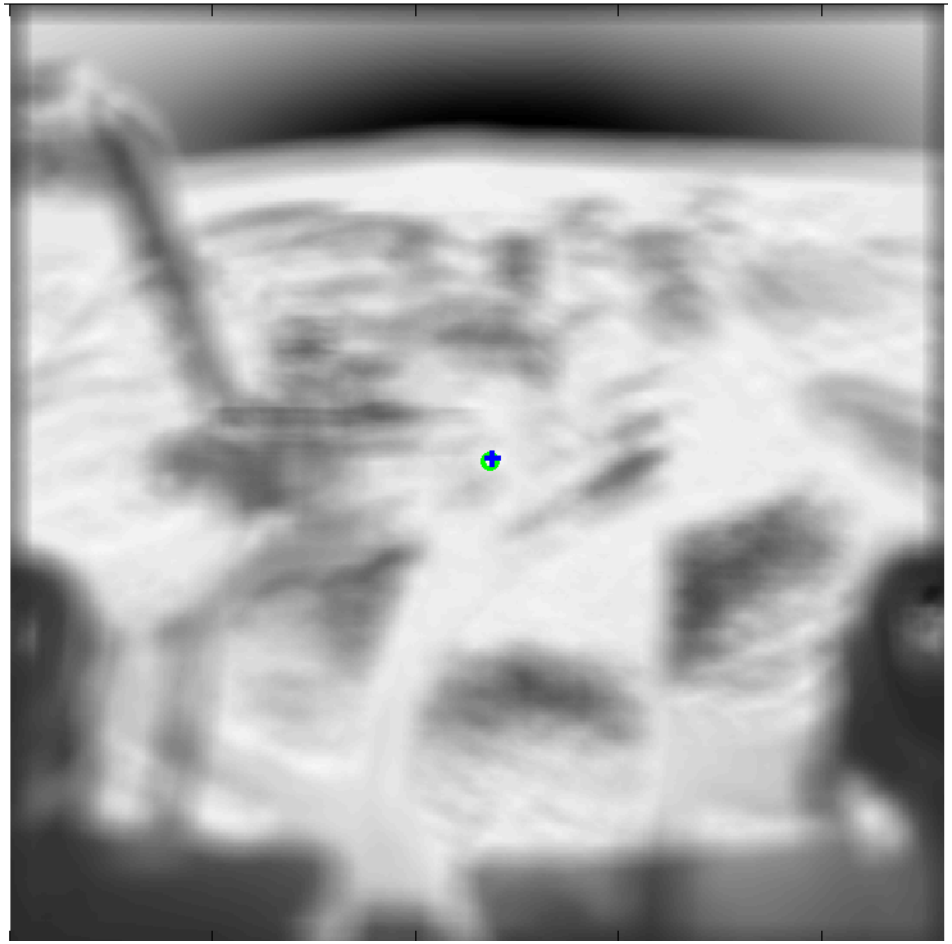
$$SSD = \sum_{[i,j] \in R} (f - g)^2$$

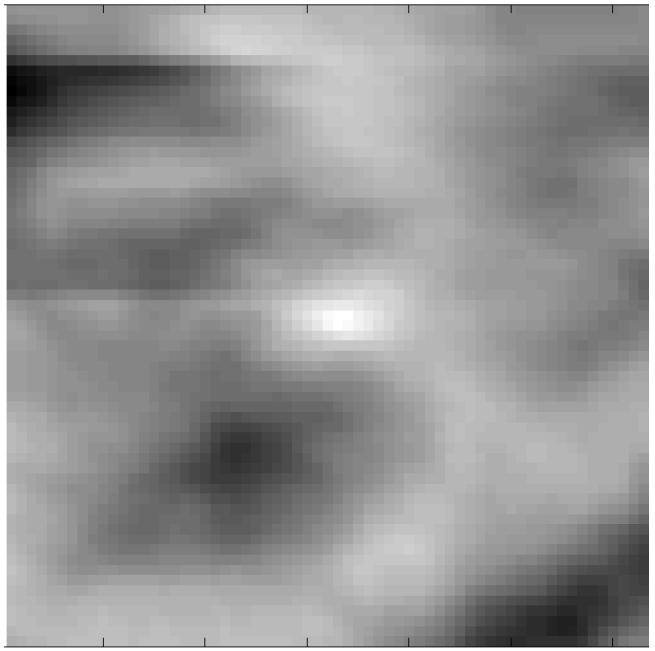
$$= \sum_{[i,j] \in R} f^2 + \sum_{[i,j] \in R} g^2 - 2 \sum_{[i,j] \in R} fg$$

$$C_{fg} = \sum_{[i,j] \in R} f(i,j) g(i,j)$$

Correlation!

SSD





Best match (highest score) in image coincides with correct match in this case!

Robert Collins
CSE486, Penn State

Handling Intensity Changes

Intensity Changes:

•the camera taking the second image might have different intensity response characteristics than the camera taking the first image

•Illumination in the scene could change

•The camera might have auto-gain control set, so that it's response changes as it moves through the scene.

Robert Collins
CSE486, Penn State

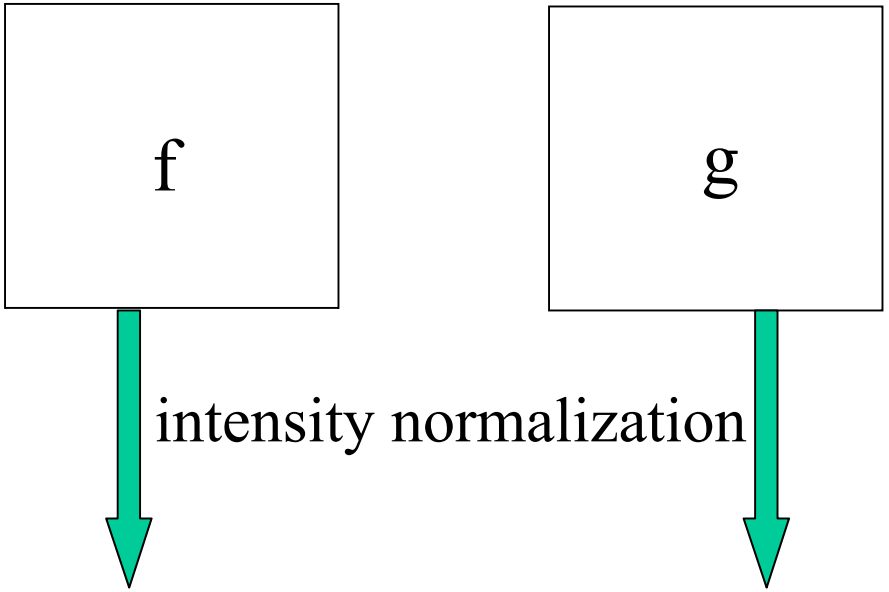
Intensity Normalization

- When a scene is imaged by different sensors, or under different illumination intensities, both the SSD and the C_{fg} can be large for windows representing the same area in the scene!
- A solution is to NORMALIZE the pixels in the windows before comparing them by subtracting the mean of the patch intensities and dividing by the std.dev.

$$\hat{f} = \frac{f - \bar{f}}{\sqrt{\sum (f - \bar{f})^2}} \qquad \hat{g} = \frac{g - \bar{g}}{\sqrt{\sum (g - \bar{g})^2}}$$

Robert Collins
CSE486, Penn State

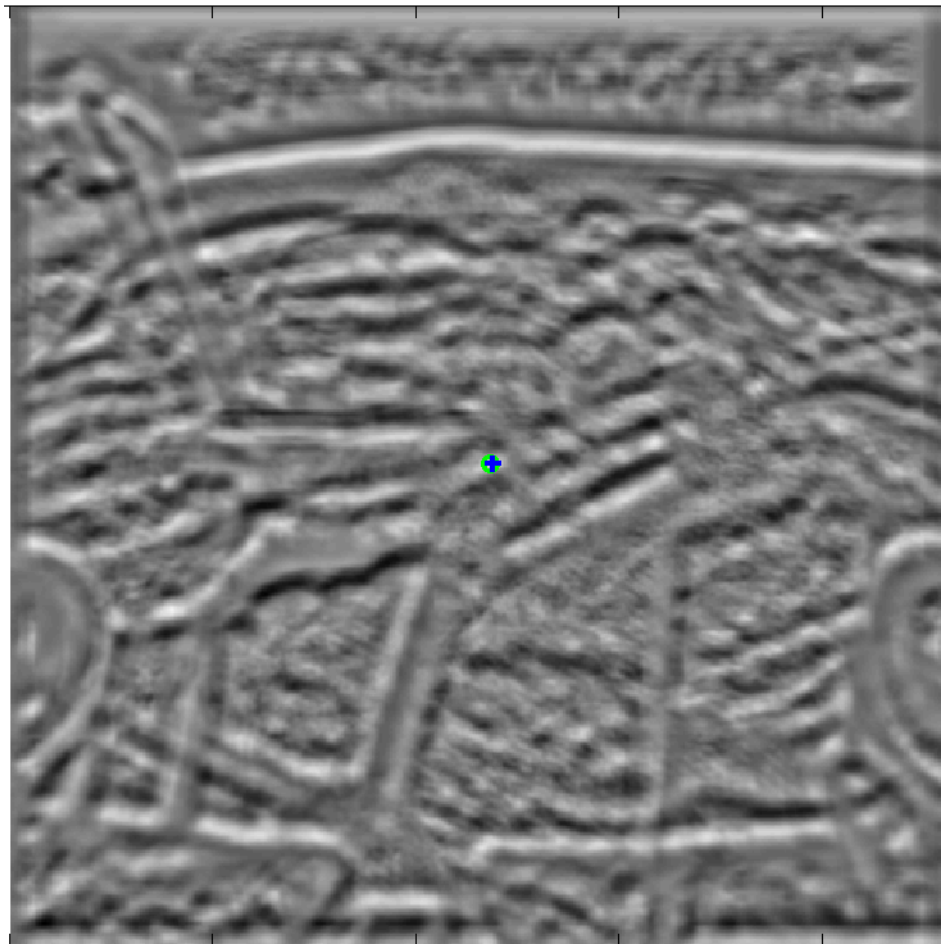
Normalized Cross Correlation

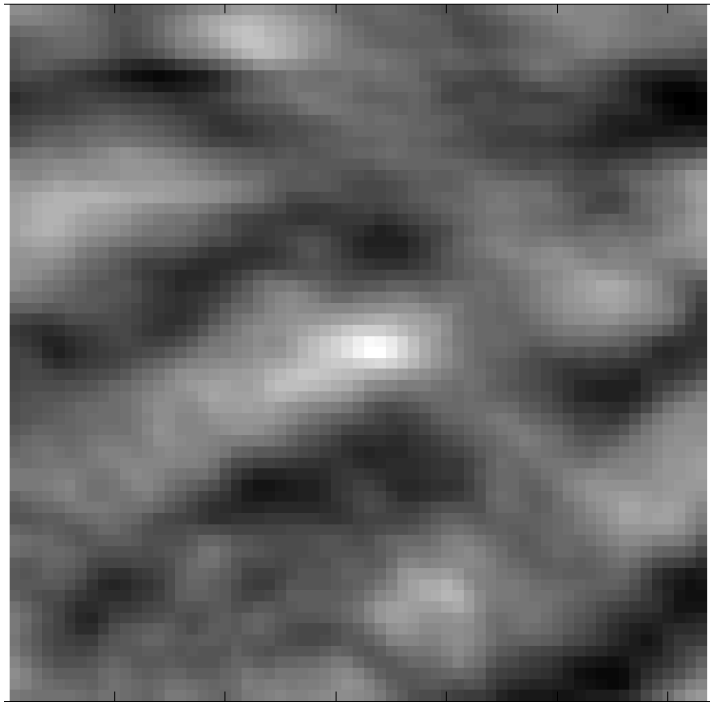


$$\hat{f} = \frac{f - \bar{f}}{\sqrt{\sum (f - \bar{f})^2}} \qquad \hat{g} = \frac{g - \bar{g}}{\sqrt{\sum (g - \bar{g})^2}}$$

$$\text{NCC(f,g)} = C_{fg}(\hat{f}, \hat{g}) = \sum_{[i,j] \in R} \hat{f}(i,j)\hat{g}(i,j)$$

Normalized Cross Correlation





Highest score also coincides with correct match.
Also, looks like less chances of getting a wrong match.

Normalized Cross Correlation

Important point about NCC:

Score values range from 1 (perfect match)
to -1 (completely anti-correlated)

Intuition: treating the normalized patches as vectors, we see they are unit vectors. Therefore, correlation becomes dot product of unit vectors, and thus must range between -1 and 1.