Alignment and Object Instance Recognition

Computer Vision

CS 543 / ECE 549

University of Illinois

Derek Hoiem

Today's class

- Fitting/Alignment (continued)
- Object instance recognition
- Example of alignment-based category recognition

Methods discussed last class

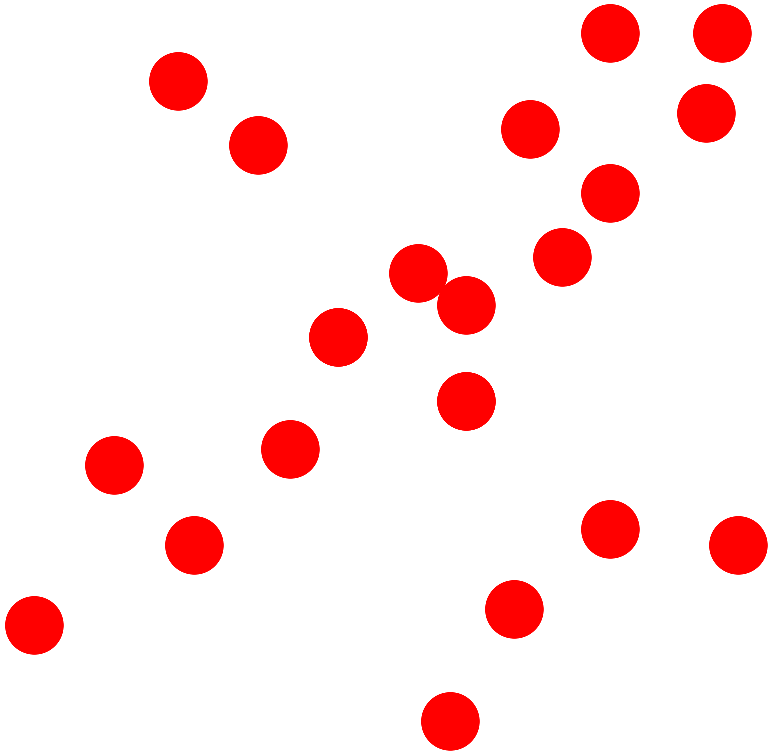
- Global optimization / Search for parameters
  - **Least squares fit**
  - **Robust least squares**
  - Iterative closest point (ICP)

- Hypothesize and test
  - **Generalized Hough transform**
  - RANSAC

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in ‘81.



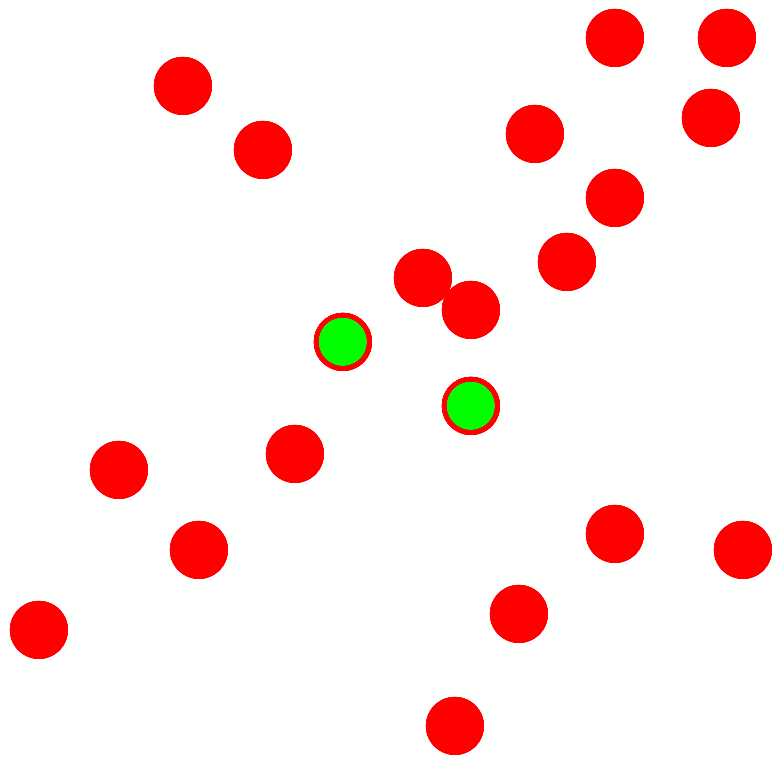
Algorithm:

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:

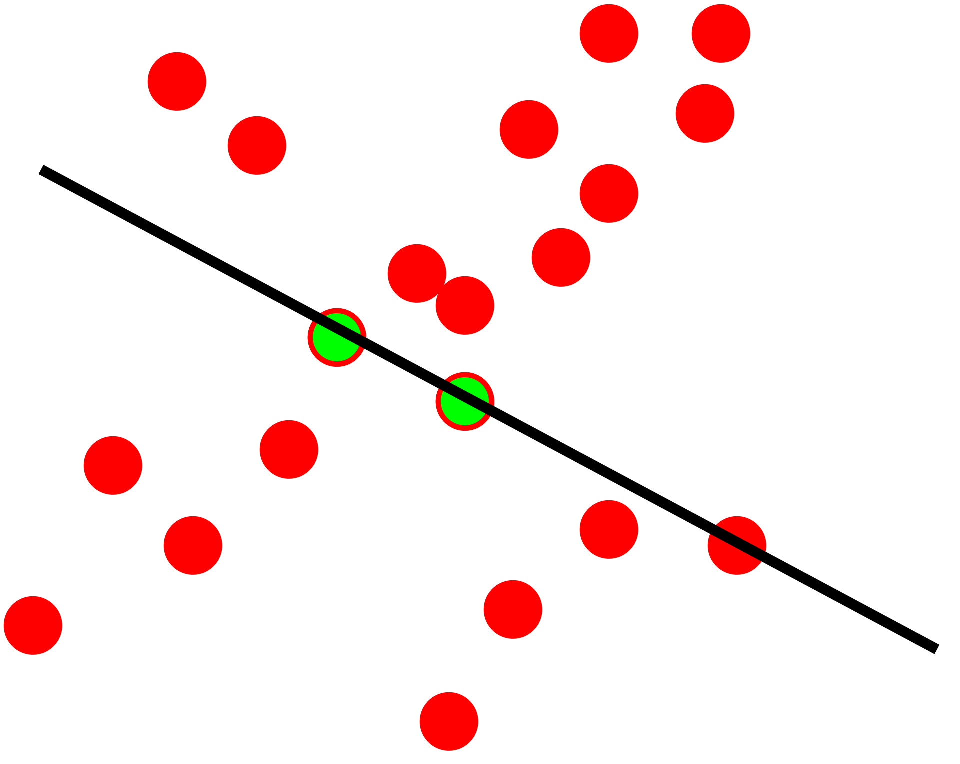
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

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

Illustration by Savarese

RANSAC

Line fitting example



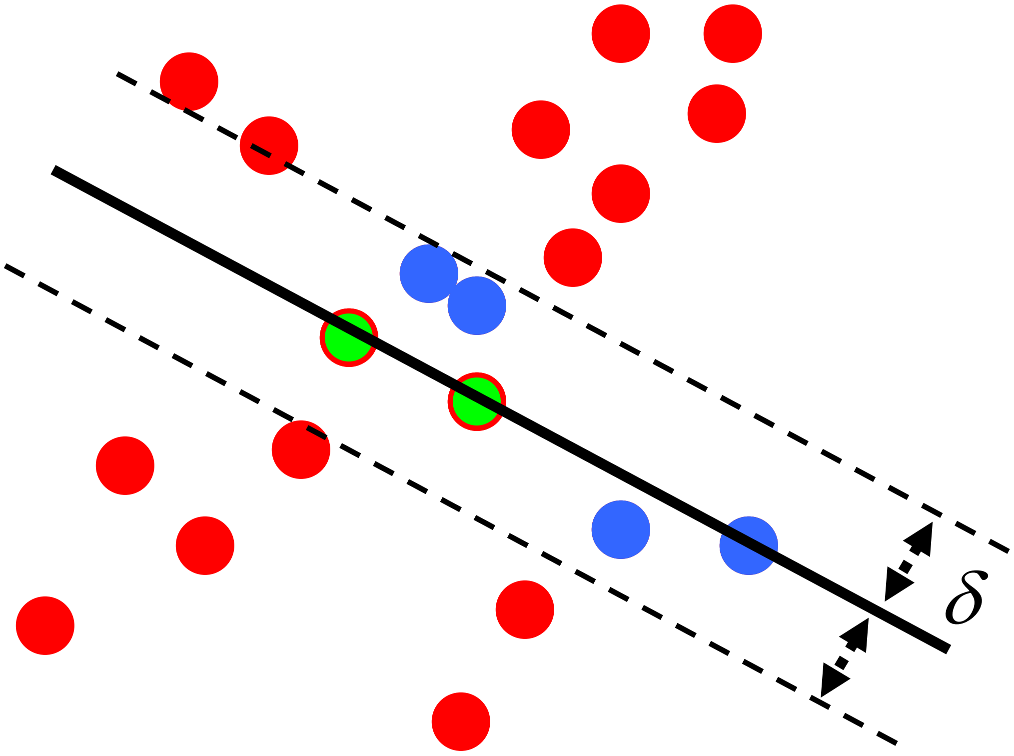
Algorithm:

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



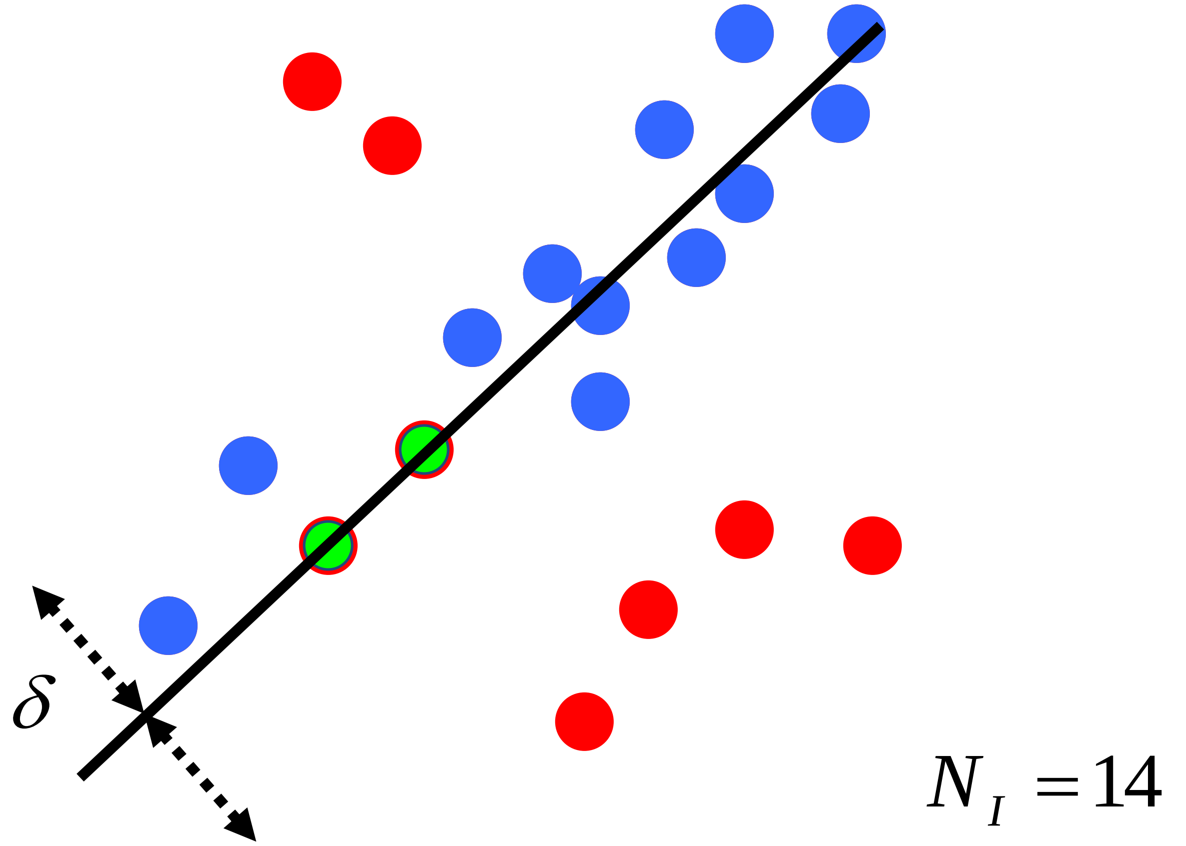
$$N_I = 6$$

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



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 sampled points *s*
  - Minimum number needed to fit the model
- 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*)
- 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: $t^2=3.84\sigma^2$

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

	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

modified from M. Pollefeys

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
- Sensitive to noise (with high noise might not be able to estimate parameters from any sample)
- Not as good for getting multiple fits (though one solution is to remove inliers after each fit and repeat)

Common applications

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

Line Fitting Demo (Part 2)

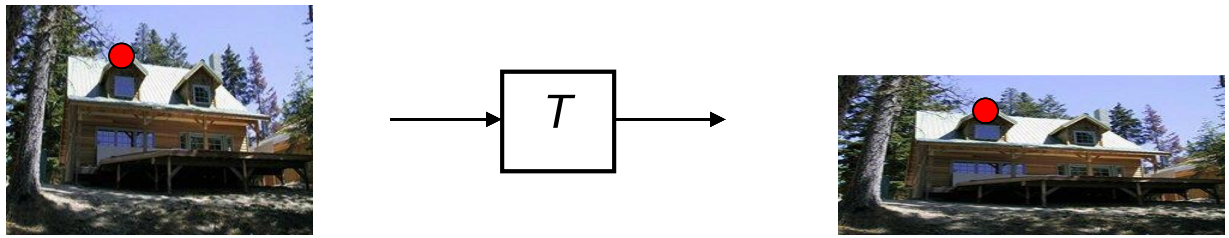
Alignment

- 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 30-50%)
  - 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’ = **T**p

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

Common transformations

original

Transformed

translation

rotation

aspect

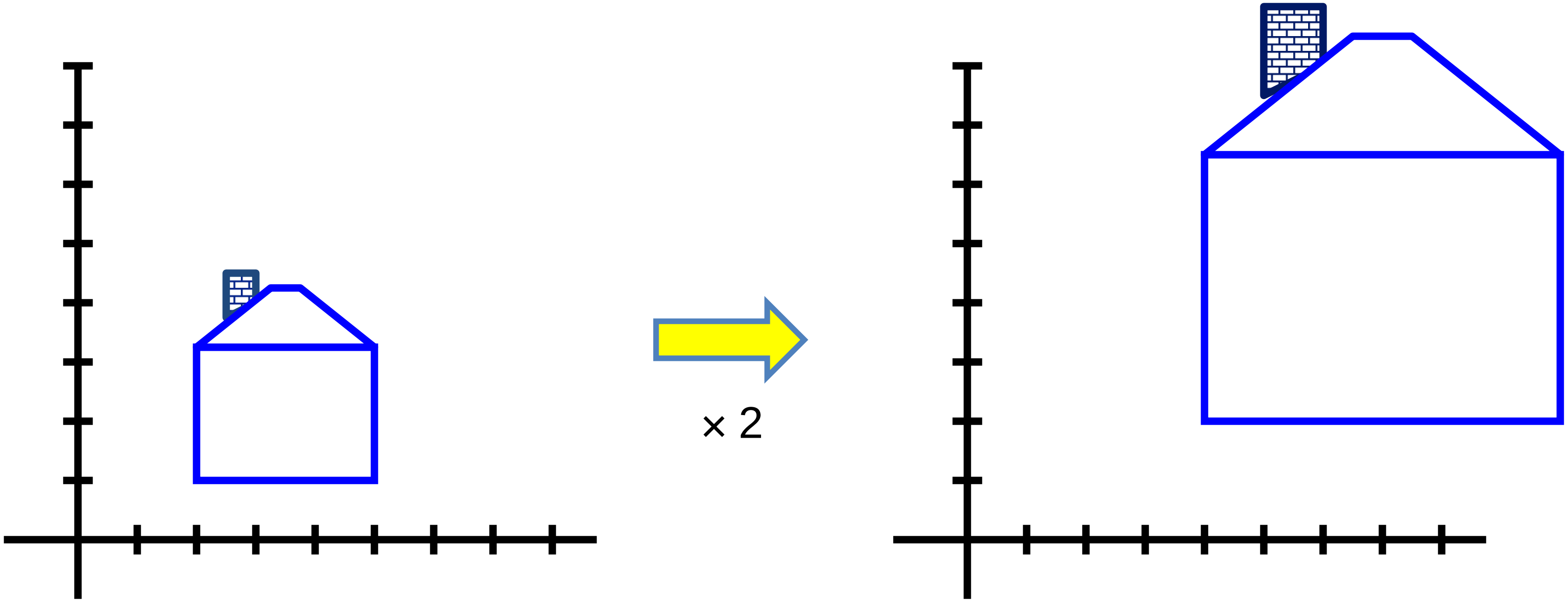
affine

perspective

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

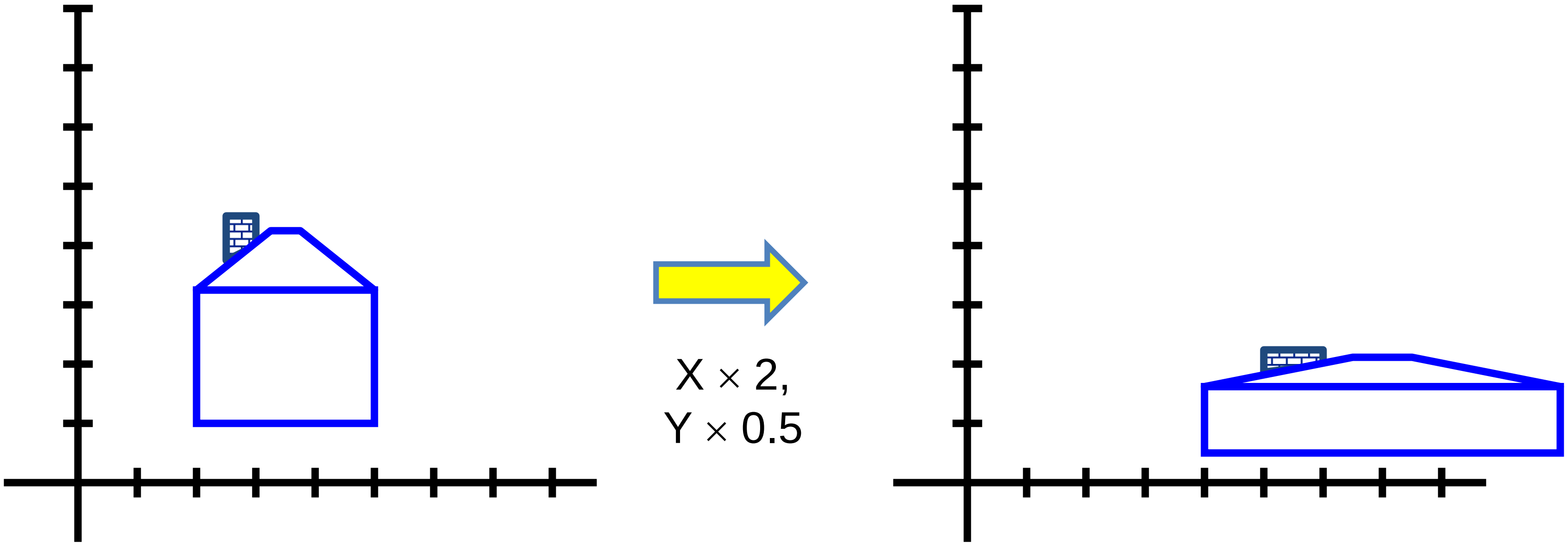
Scaling

- *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:

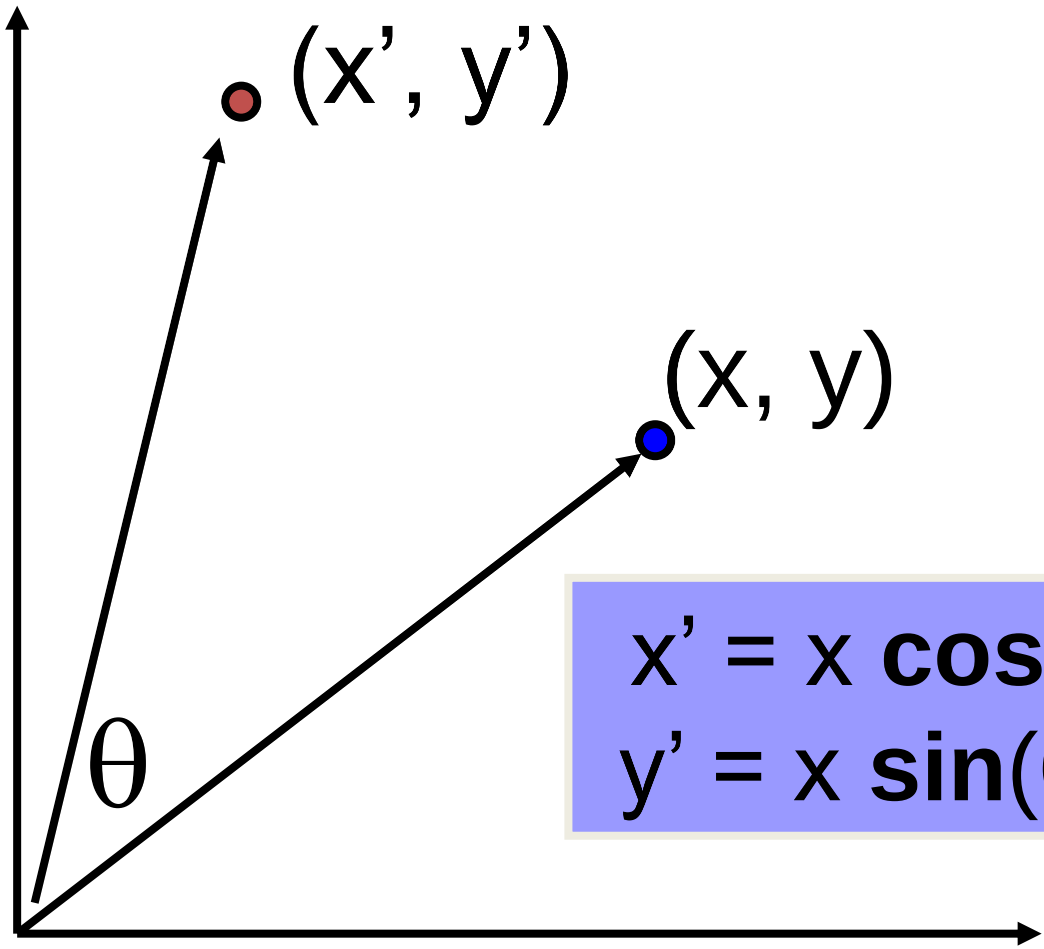
$$x' = ax$$

$$y' = by$$

- Or, in matrix form:

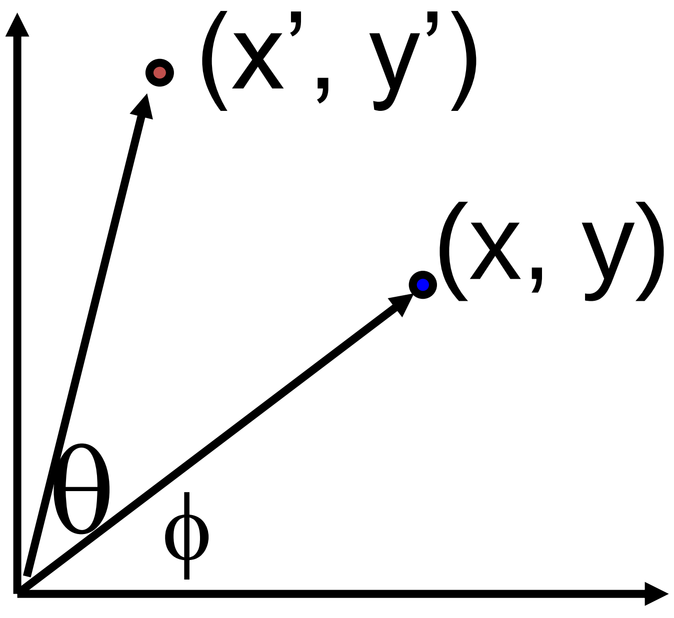
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\textit{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

2-D Rotation



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

2-D Rotation



Polar coordinates…

$x = r\cos(\phi)$

$y = r\sin(\phi)$

$x' = r\cos(\phi + \theta)$

$y' = r\sin(\phi + \theta)$

Trig Identity…

$x' = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$

$y' = r\sin(\phi)\cos(\theta) + r\cos(\phi)\sin(\theta)$

Substitute…

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

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

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

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

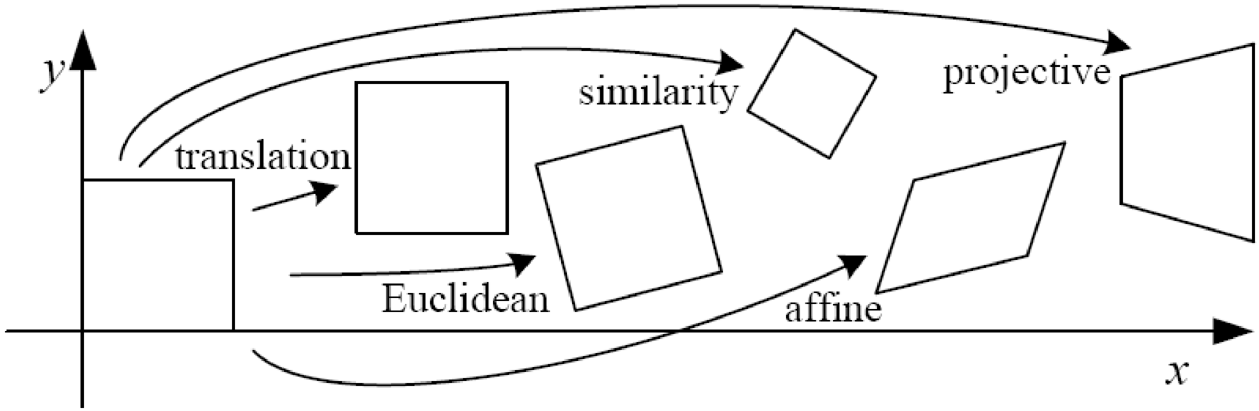
Projective transformations are combos of

- Affine transformations, and
- Projective warps

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)



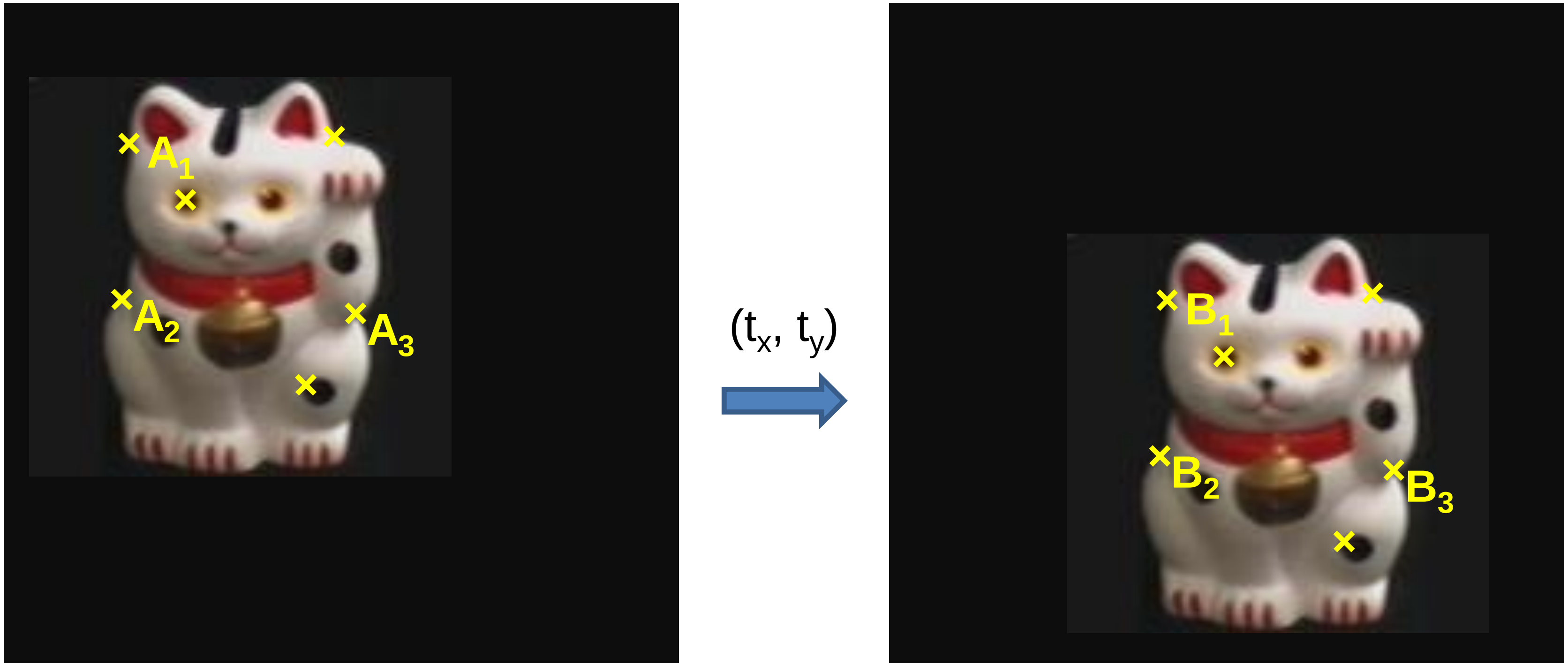
Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R}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

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



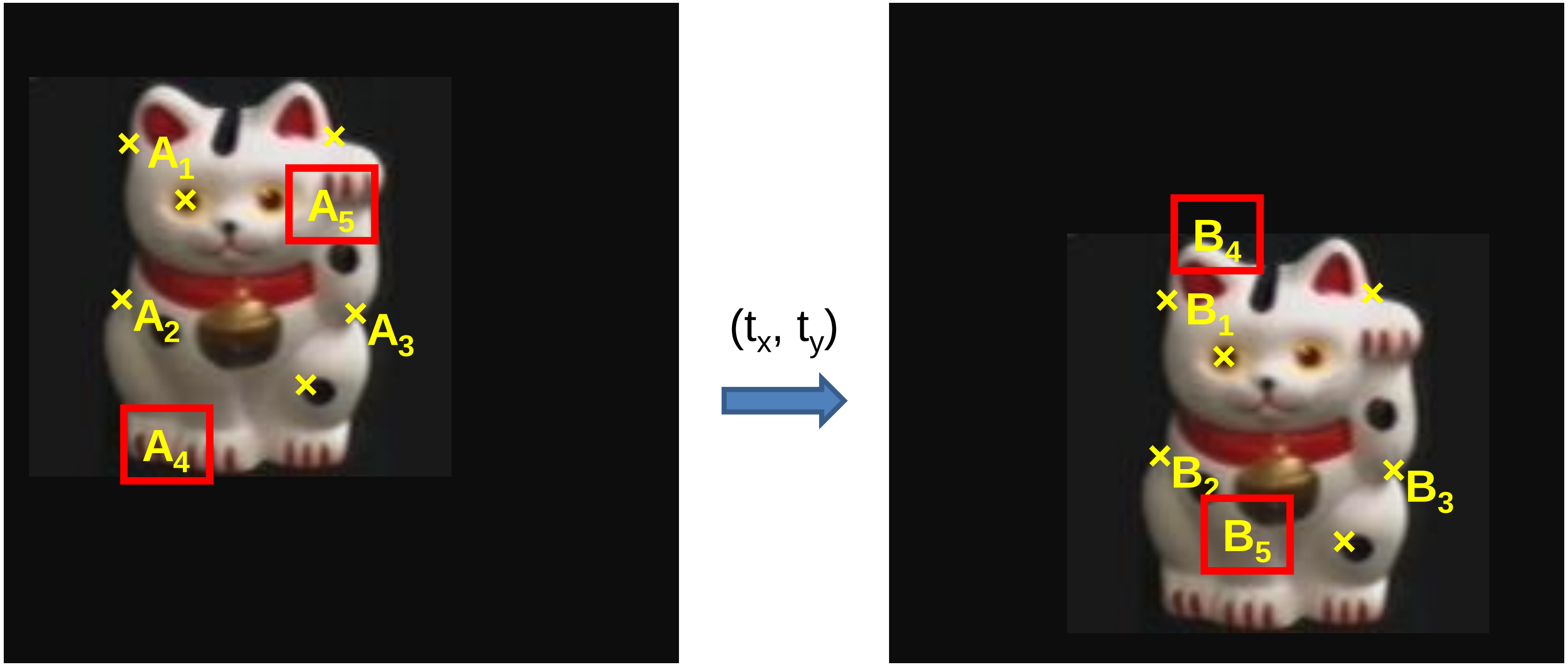
Least squares solution

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



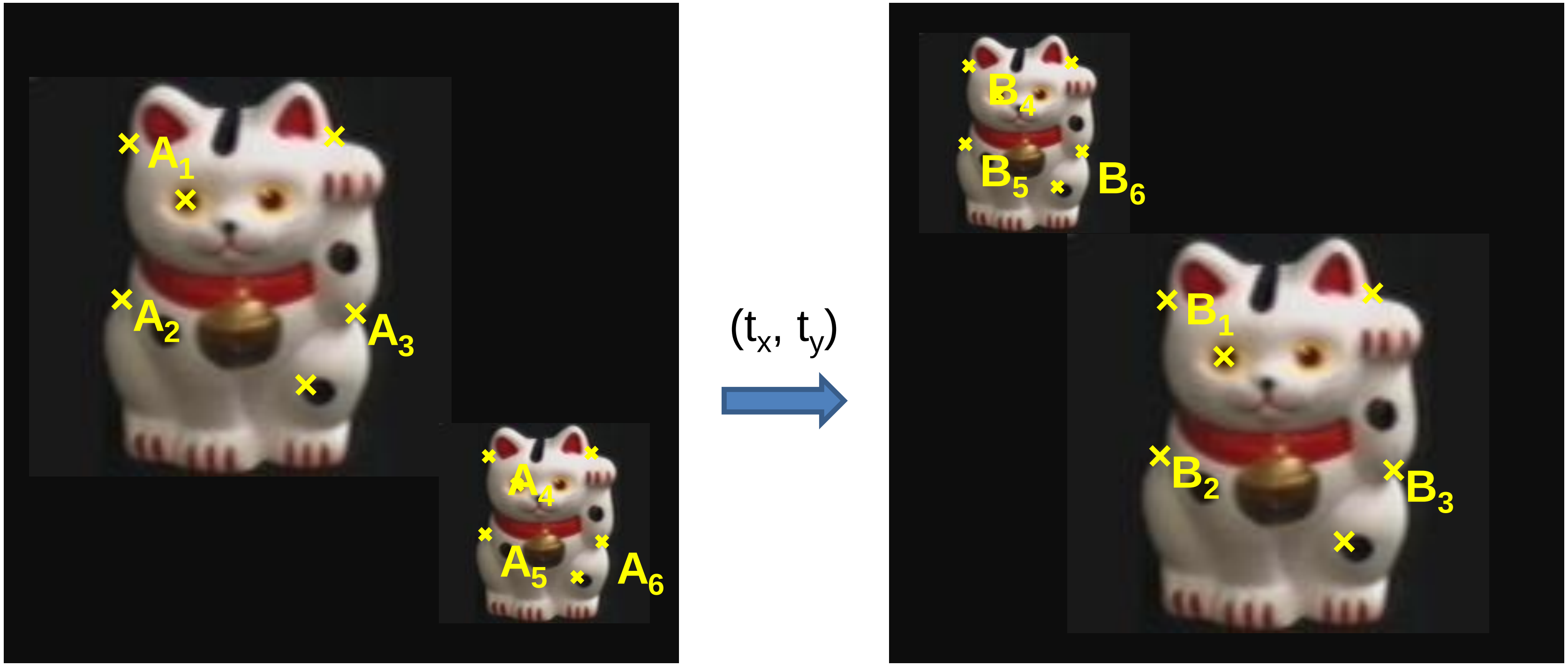
Problem: outliers

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



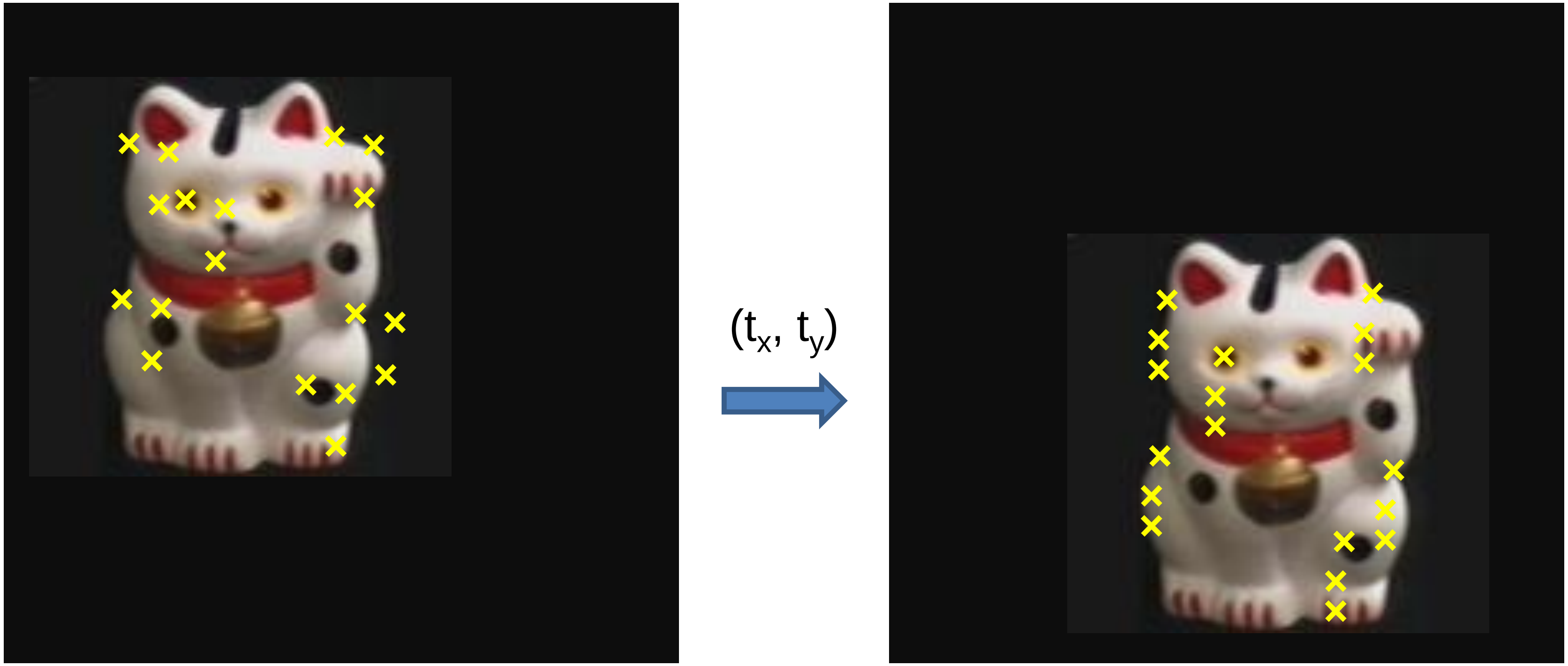
Problem: outliers, multiple objects, and/or many-to-one matches

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



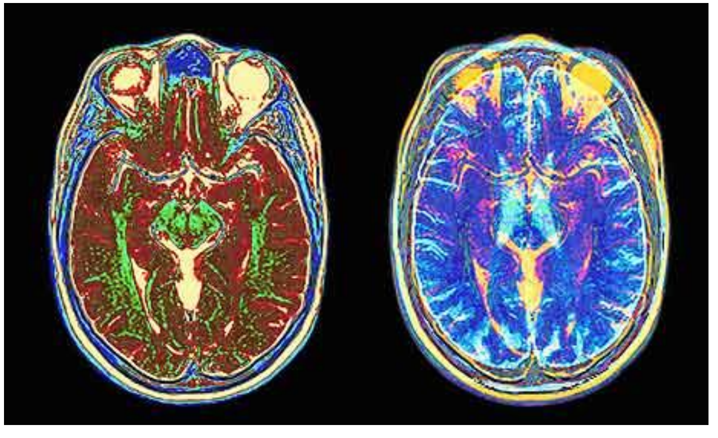
Problem: no initial guesses for correspondence

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

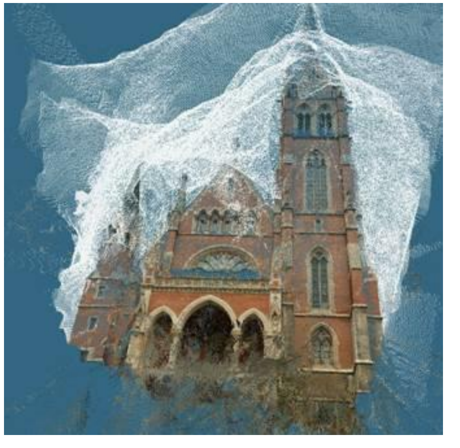
What if you want to align but have no prior matched pairs?

- Hough transform and RANSAC not applicable

- Important applications



Medical imaging: match brain scans or contours



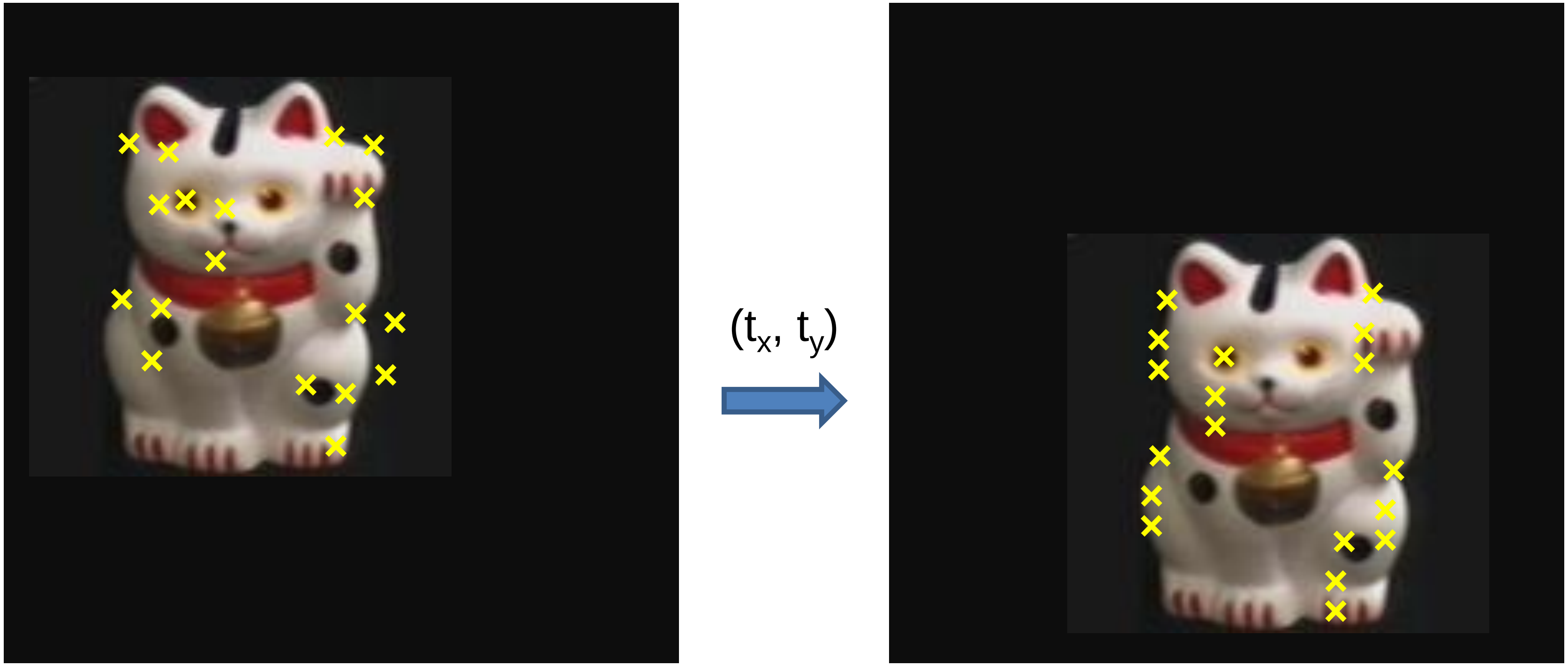
Robotics: match point clouds

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: solving for translation



Problem: no initial guesses for correspondence

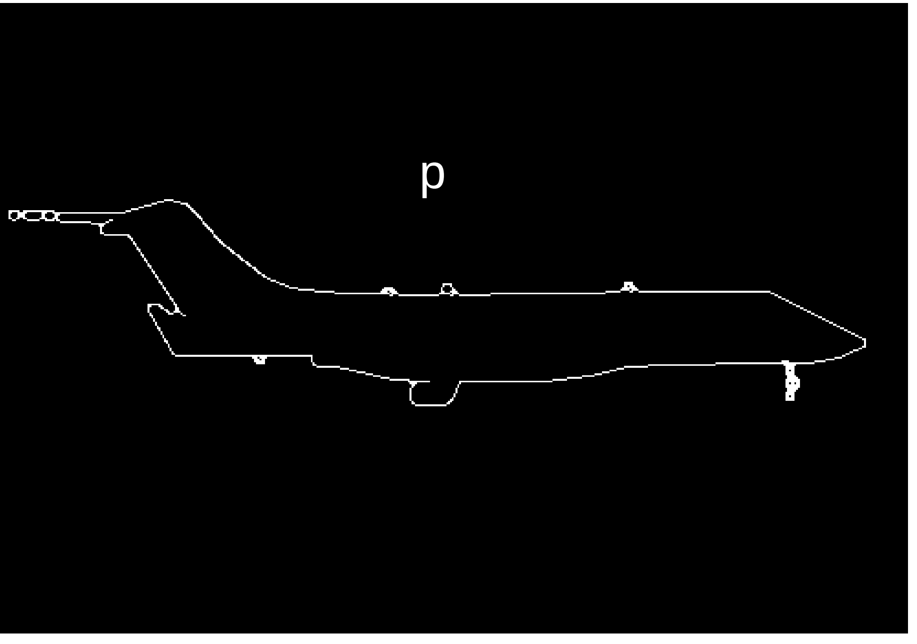
ICP solution

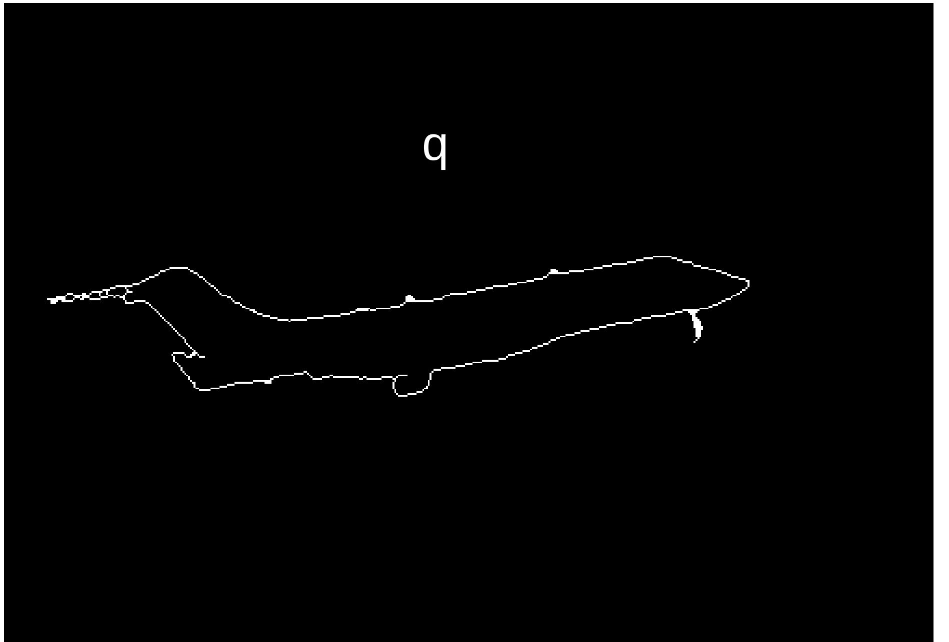
1. Find nearest neighbors for each point
2. Compute transform using matches
3. Move points using transform
4. Repeat steps 1-3 until convergence

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

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 $\text{match(i)} = \underset{j}{\text{argmin}}\, dist(pi, qj)$
4. Compute transformation ***T*** based on matches
5. Warp points ***p*** according to ***T***
6. Repeat 3-5 until convergence





Algorithm Summary

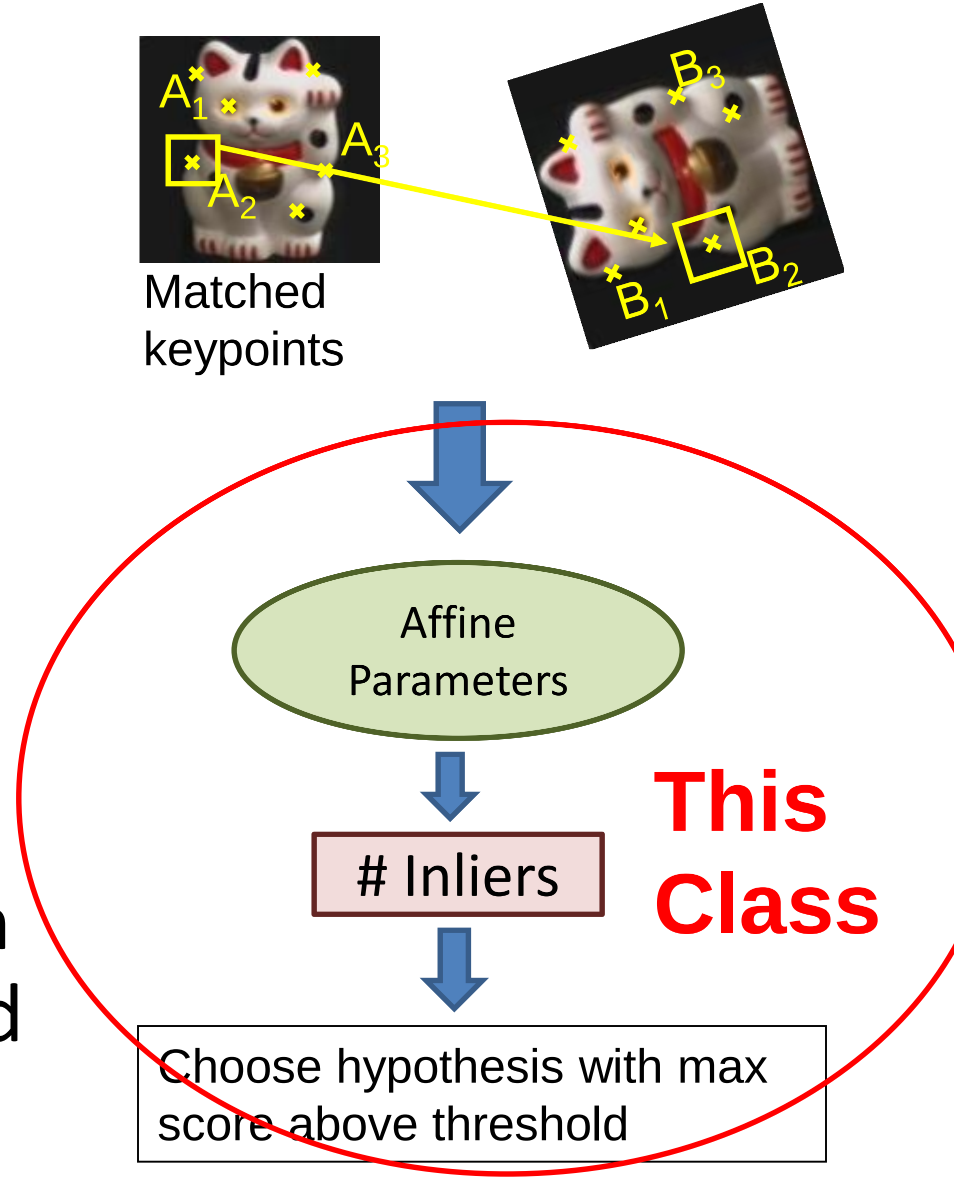
- Least Squares Fit
  - closed form solution
  - robust to noise
  - not robust to outliers
- Robust Least Squares
  - improves robustness to noise
  - requires iterative optimization
- Hough transform
  - robust to noise and outliers
  - can fit multiple models
  - only works for a few parameters (1-4 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

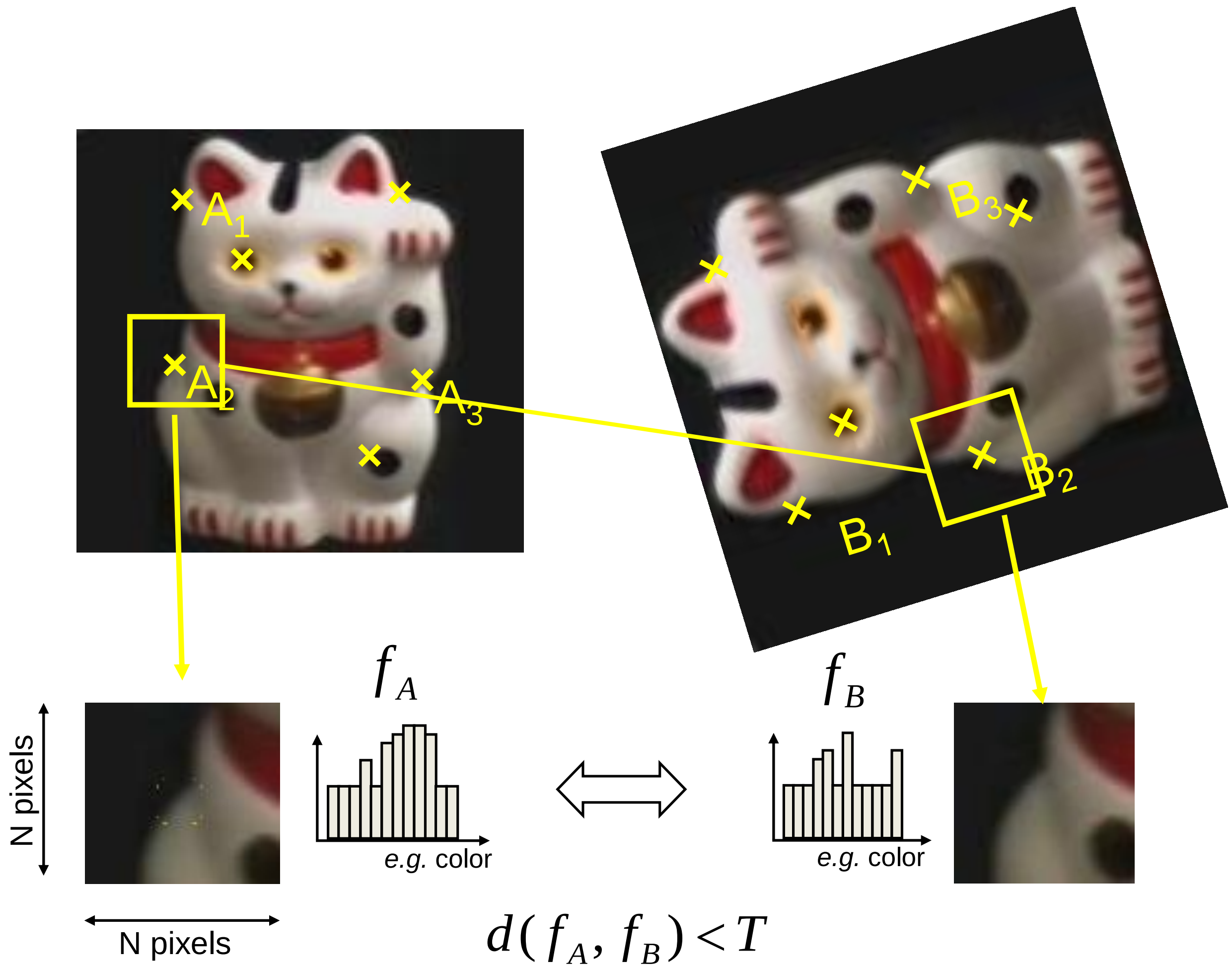
1. Match keypoints to object model

2. Solve for affine transformation parameters

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

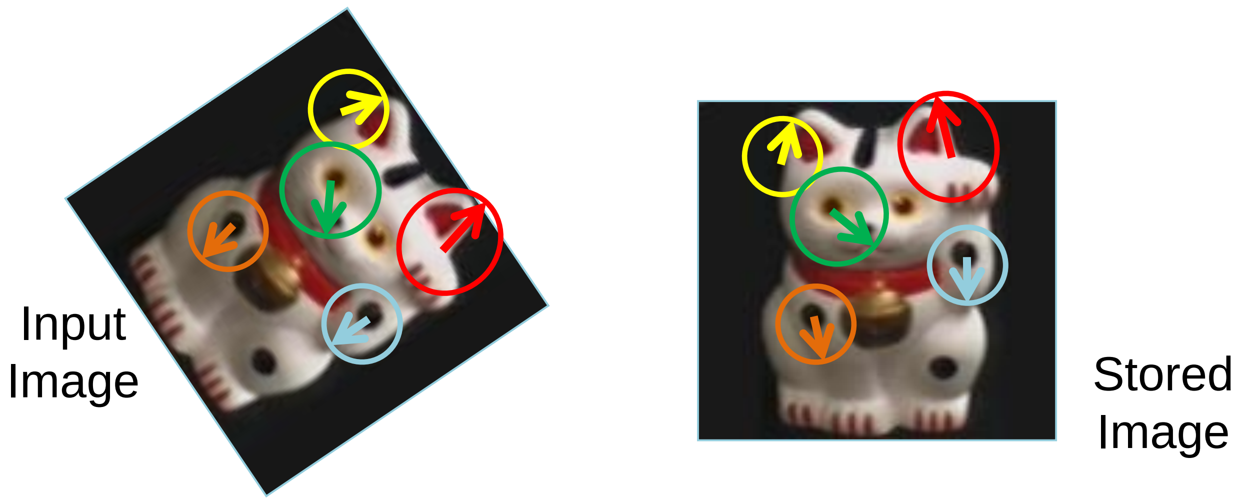


Overview of Keypoint Matching



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

Matching Keypoints

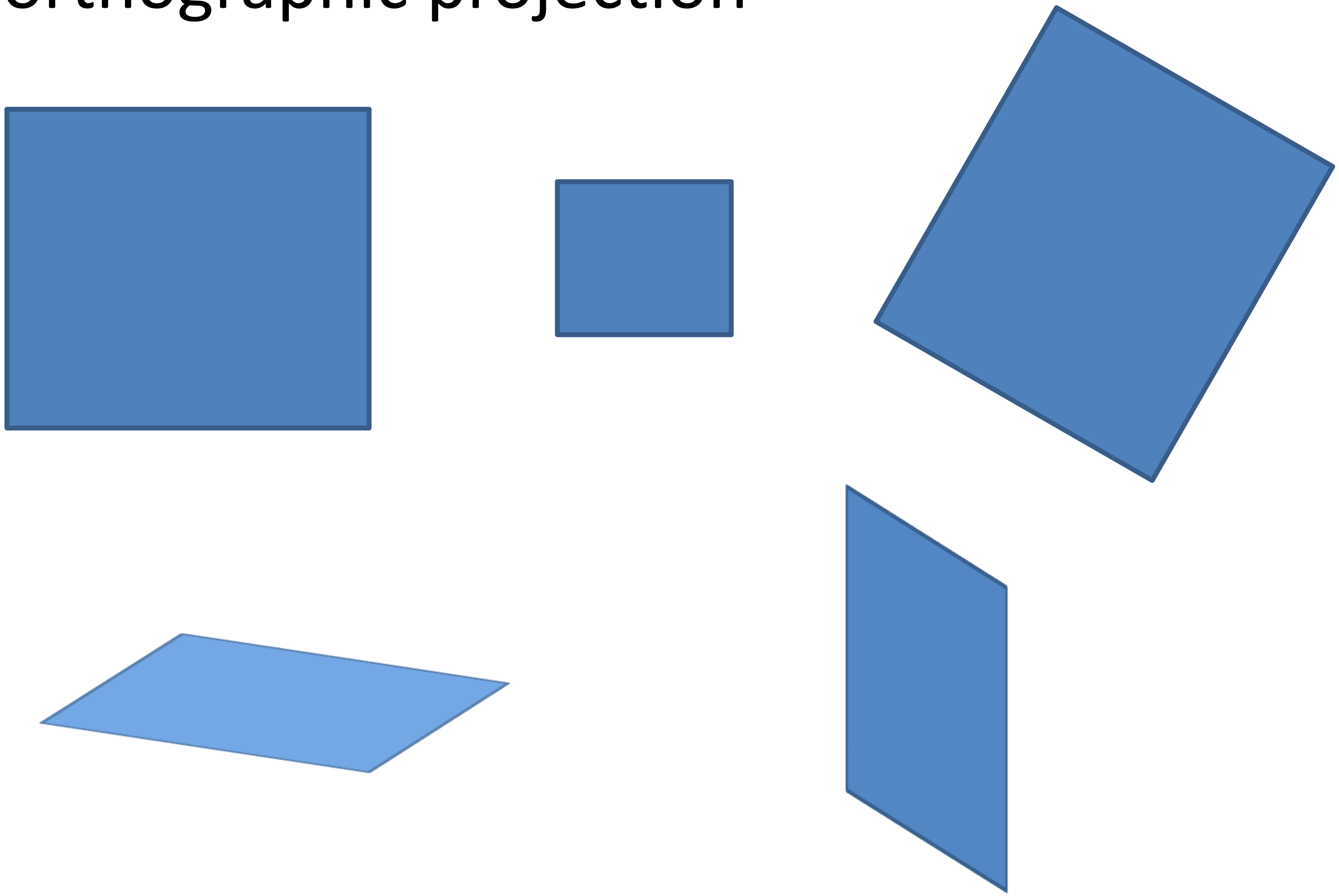
- Want to match keypoints between:
  1. Query image
  2. Stored image containing the object

- Given descriptor x_0, find two nearest neighbors x_1, x_2 with distances d_1, d_2

- x_1 matches x_0 if $d_1/d_2 < 0.8$
  – This gets rid of 90% false matches, 5% of true matches in Lowe's study

Affine Object Model

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



Affine Object Model

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

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

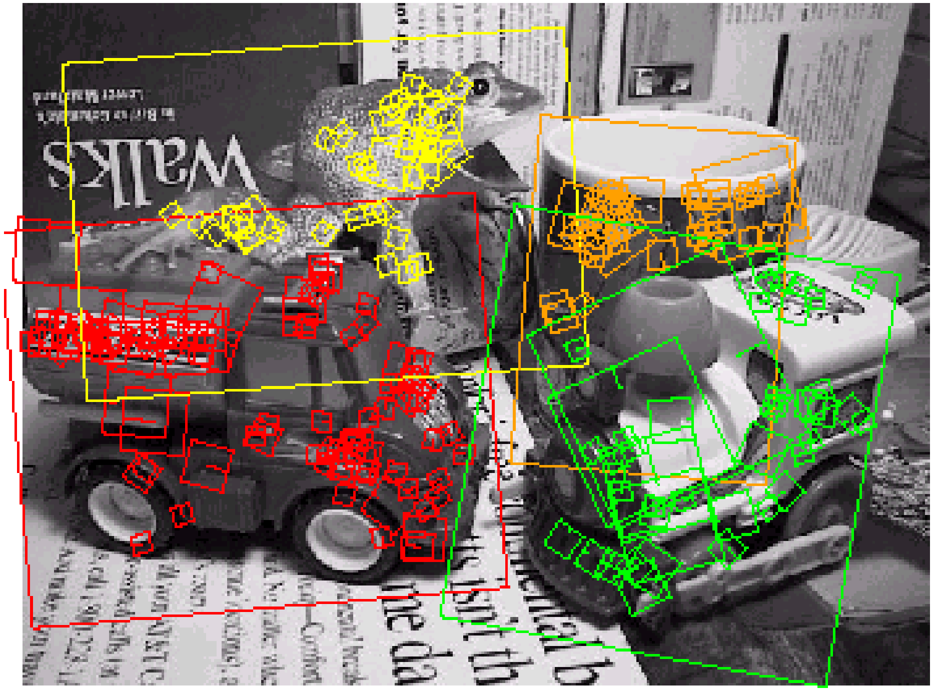
$$\mathbf{x} = [\mathbf{A}^{\mathbf{T}}\mathbf{A}]^{-1}\mathbf{A}^{\mathbf{T}}\mathbf{b}$$

What is the minimum number of matched points that we need?

Finding the objects (in detail)

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)

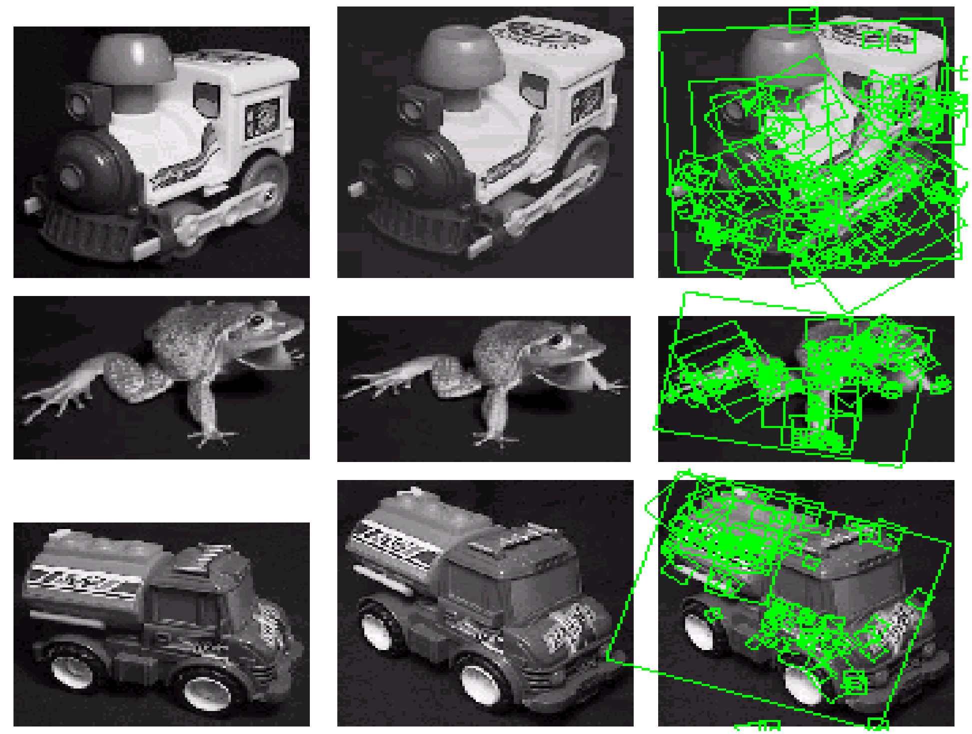
Examples of recognized objects



View interpolation

- Training
  - Given images of different viewpoints
  - Cluster similar viewpoints using feature matches
  - Link features in adjacent views
- Recognition
  - Feature matches may be spread over several training viewpoints

  ⇒ Use the known links to "transfer votes" to other viewpoints



[Lowe01]

Slide credit: David Lowe

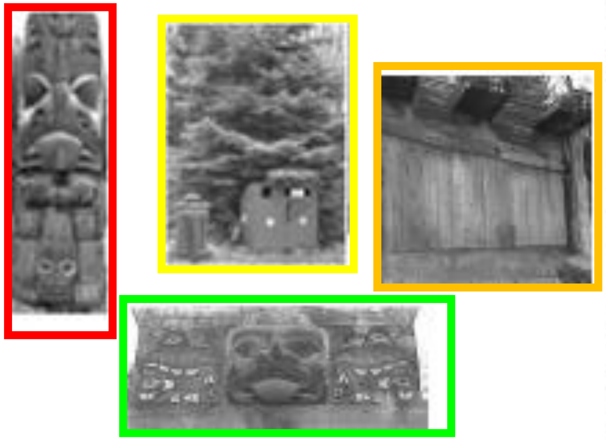
Applications

- Sony Aibo
  (Evolution Robotics)

- SIFT usage
  – Recognize docking station
  – Communicate with visual cards

- Other uses
  – Place recognition
  – Loop closure in SLAM

Location Recognition



Training

[Lowe04]

Slide credit: David Lowe

Another application: category recognition

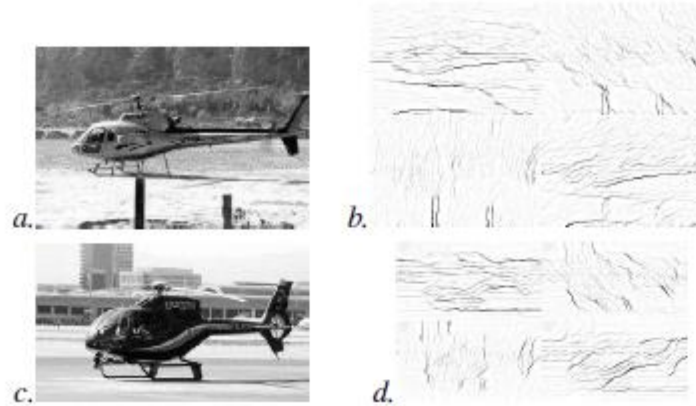
- Goal: identify what type of object is in the image
- Approach: align to known objects and choose category with best match

?

"Shape matching and object recognition using low distortion correspondence", Berg et al., CVPR 2005: http://www.cnbc.cmu.edu/cns/papers/berg-cvpr05.pdf

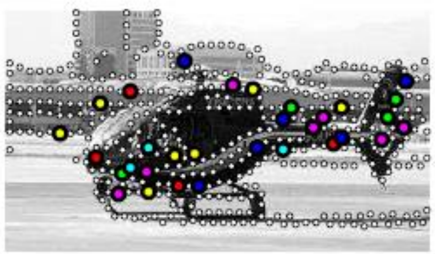
Summary of algorithm

- Input: query *q* and exemplar *e*
- For each: sample edge points and create "geometric blur" descriptor
- Compute match cost **c** to match points in *q* to each point in *e*
- Compute deformation cost **H** that penalizes change in orientation and scale for pairs of matched points
- Solve a binary quadratic program to get correspondence that minimizes **c** and **H**, using thin-plate spline deformation
- Record total cost for *e*, repeat for all exemplars, choose exemplar with minimum cost

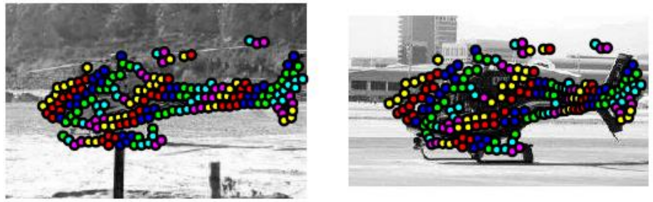


Input, Edge Maps

Geometric Blur

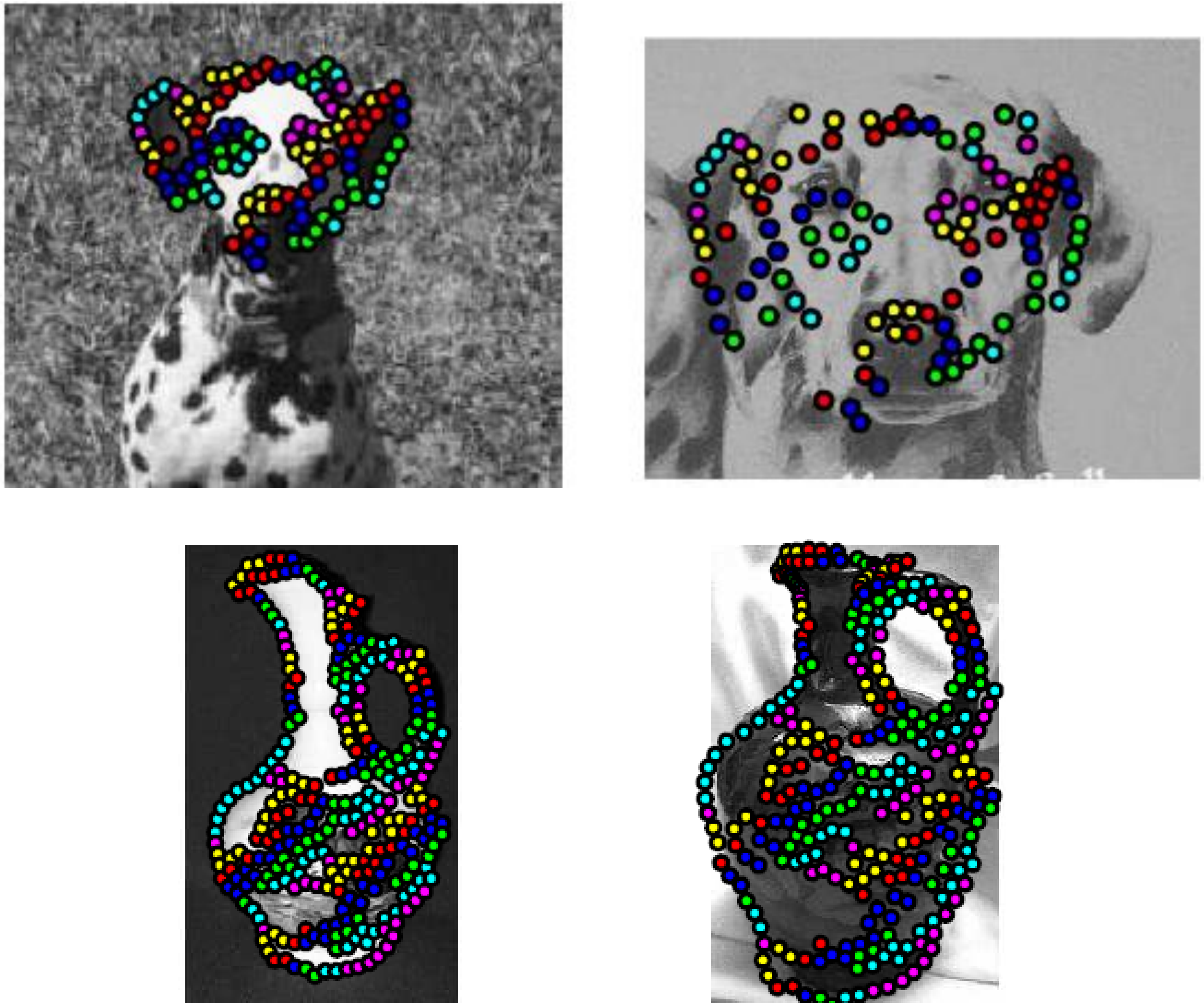


Feature Points

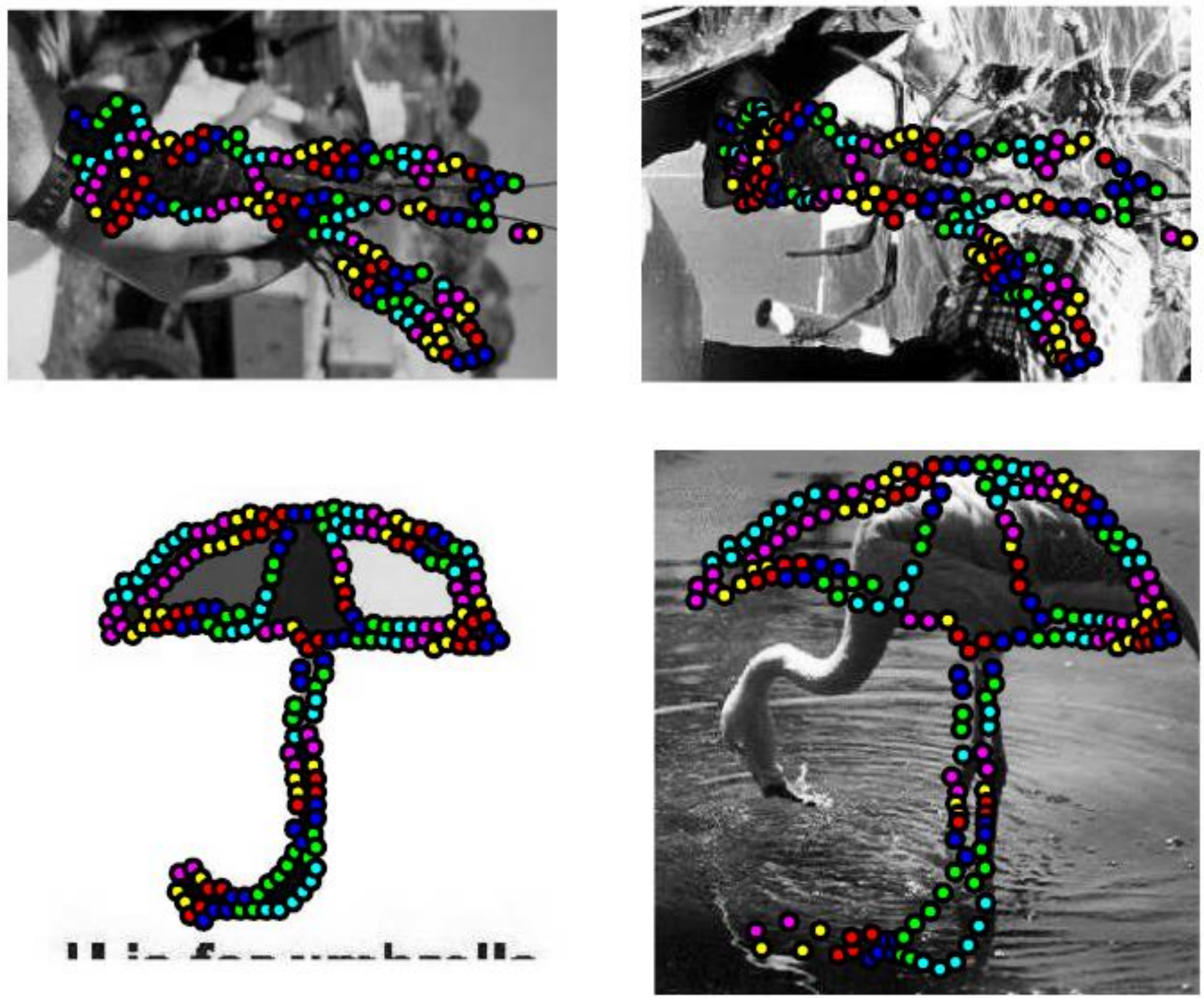


Correspondences

Examples of Matches



Examples of Matches



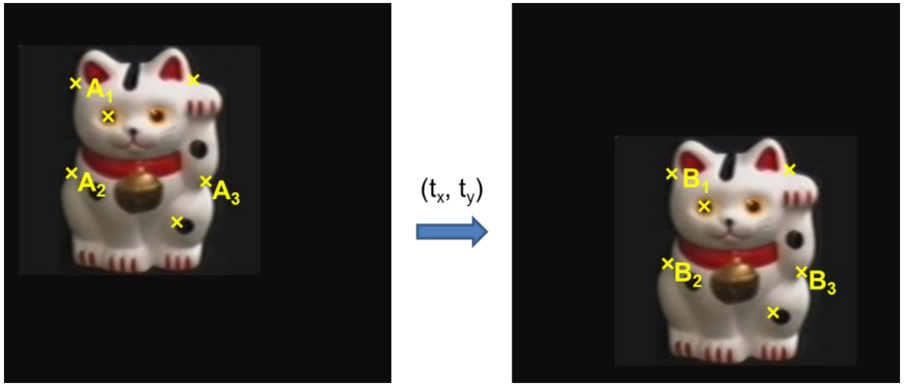
Other ideas worth being aware of

- Thin-plate splines: combines global affine warp with smooth local deformation

- Robust non-rigid point matching: http://noodle.med.yale.edu/~chui/tps-rpm.html (includes code, demo, paper)

Key concepts

- Alignment
  - Hough transform
  - RANSAC
  - ICP



- Object instance recognition
  - Find keypoints, compute descriptors
  - Match descriptors
  - Vote for / fit affine parameters
  - Return object if # inliers > T