

Fitting and Alignment

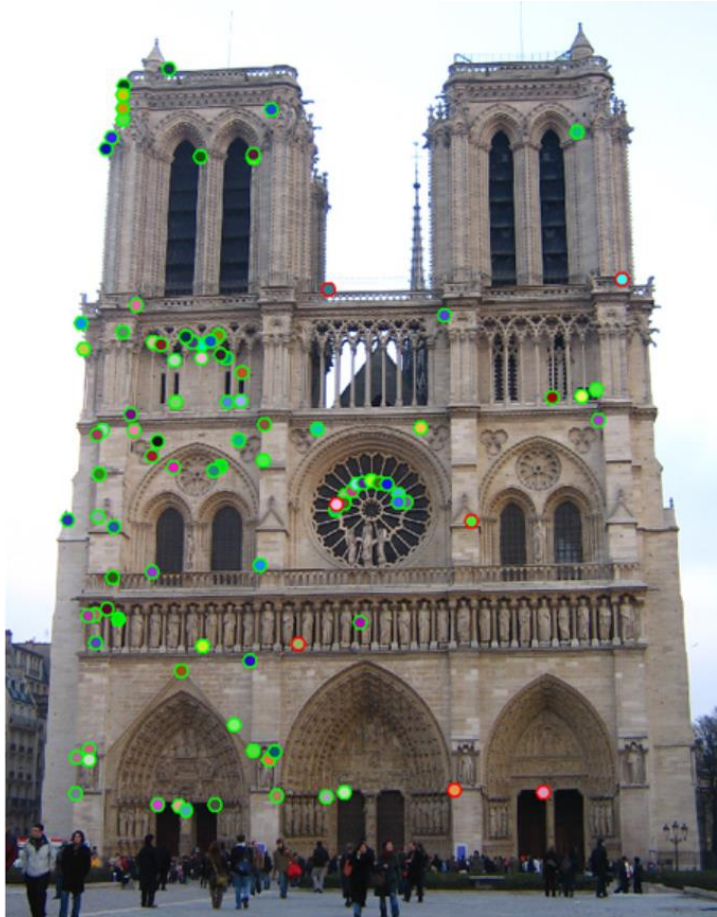
Szeliski 6.1

Computer Vision

James Hays

Acknowledgment: Many slides from Derek Hoiem, Lana Lazebnik, and Grauman&Leibe 2008 AAAI Tutorial

Project 2 – due Monday



The top 100 most confident local feature matches from a baseline implementation of project 2. In this case, 93 were correct (highlighted in green) and 7 were incorrect (highlighted in red).

Project 2: Local Feature Matching

Review

Fitting: find the parameters of a model that best fit the data

Alignment: find the parameters of the transformation that best align matched points

Review: Fitting and Alignment

- Design challenges
 - Design a suitable **goodness of fit** measure
 - Similarity should reflect application goals
 - Encode robustness to outliers and noise
 - Design an **optimization** method
 - Avoid local optima
 - Find best parameters quickly

Fitting and Alignment: Methods

- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC

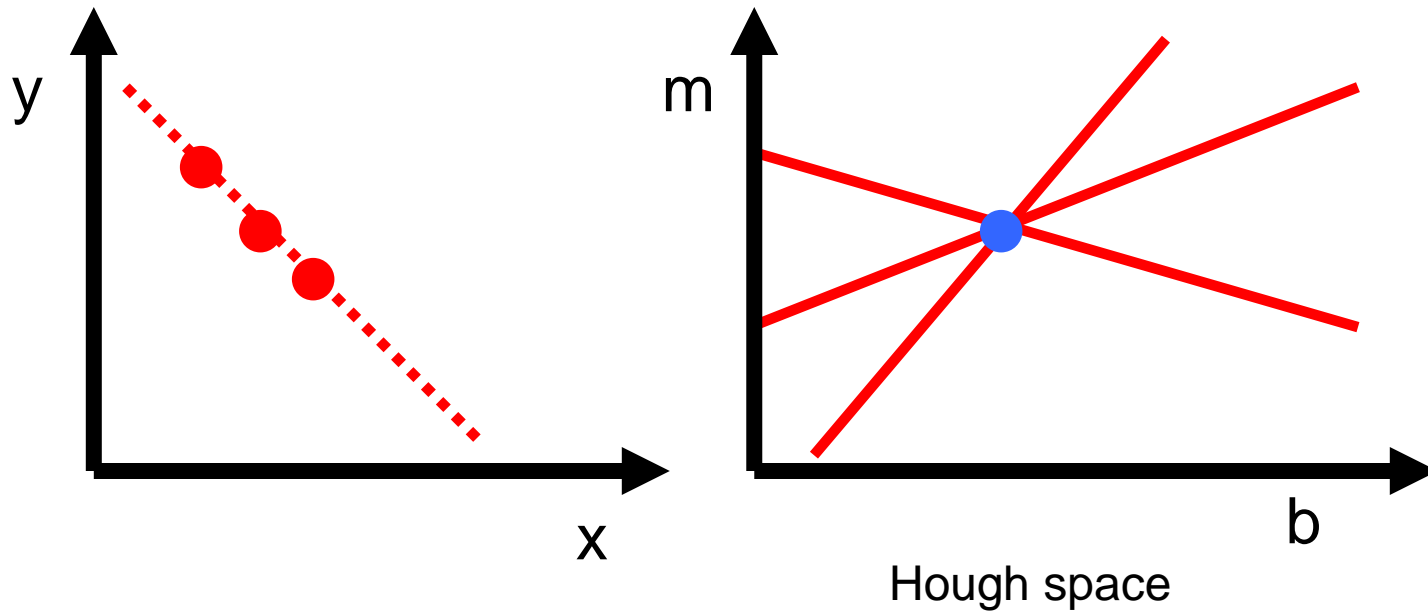
Review: Hough Transform

1. Create a grid of parameter values
2. Each point (or correspondence) votes for a set of parameters, incrementing those values in grid
3. Find maximum or local maxima in grid

Review: Hough transform

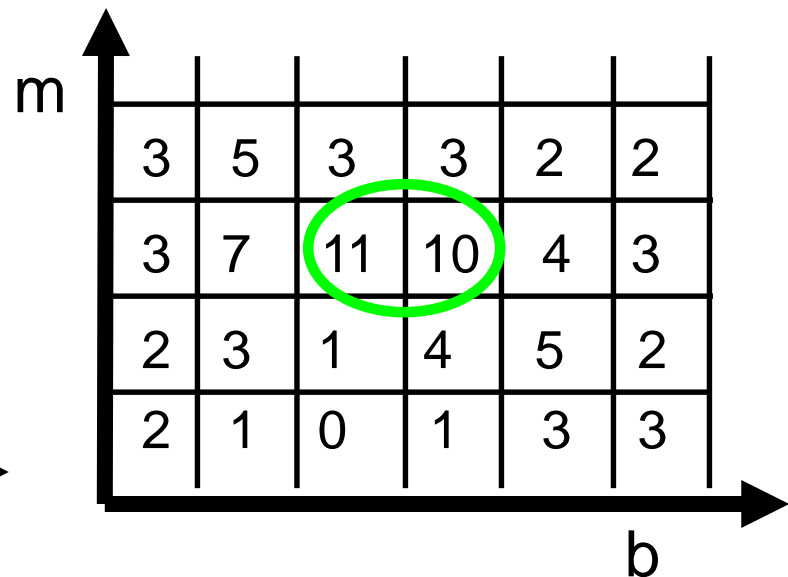
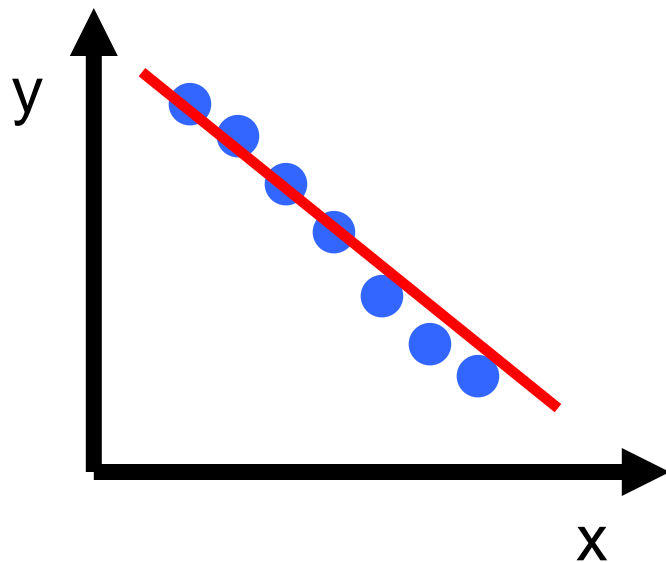
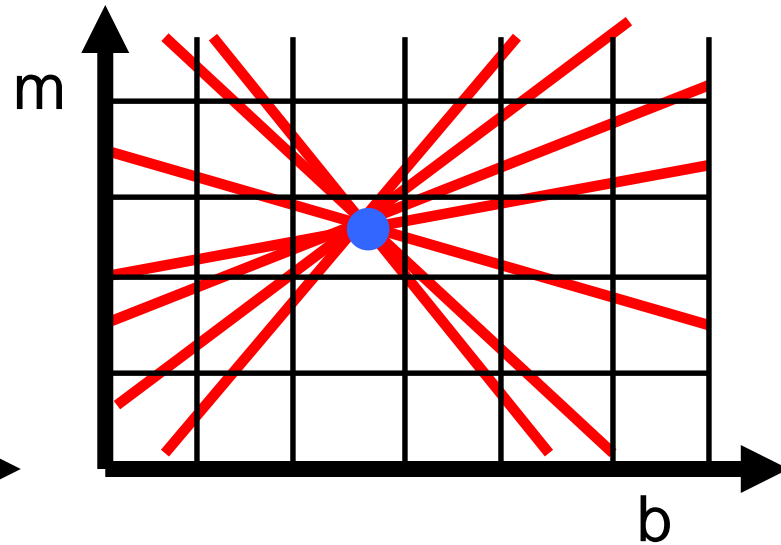
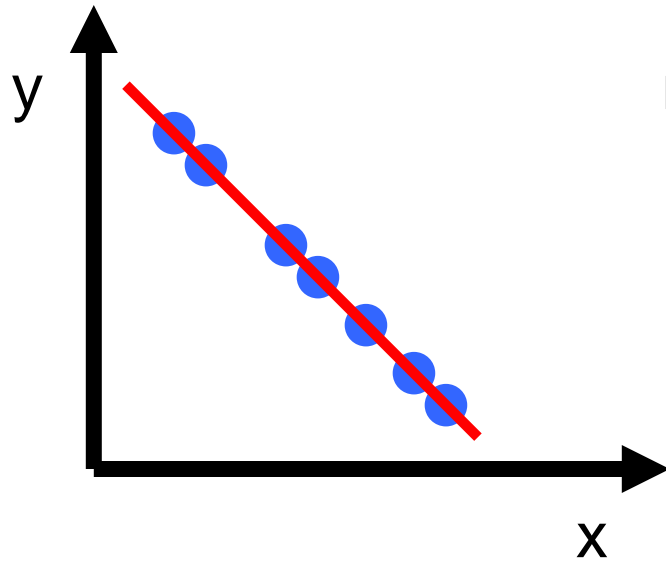
P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures*, Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

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



$$y = m x + b$$

Review: Hough transform

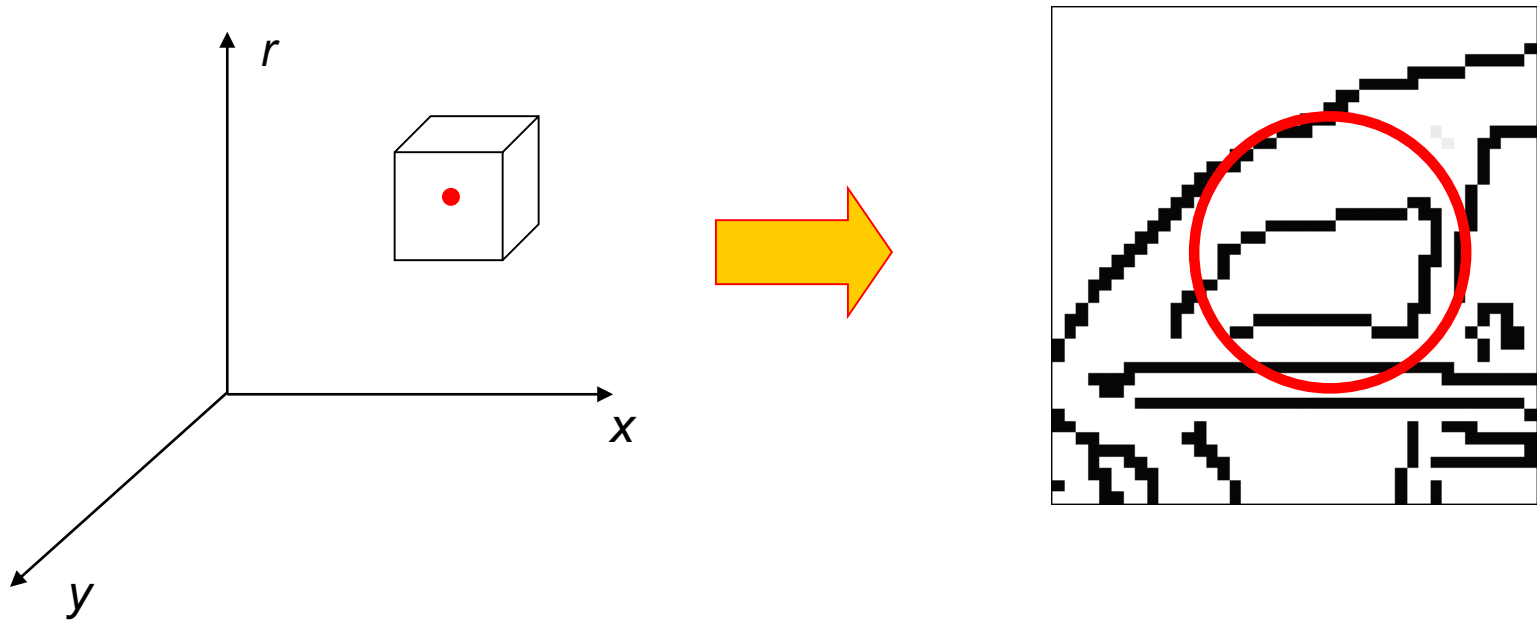


Hough Transform

- How would we find circles?
 - Of fixed radius
 - Of unknown radius
 - Of unknown radius but with known edge orientation

Hough transform for circles

- Conceptually equivalent procedure: for each (x,y,r) , draw the corresponding circle in the image and compute its “support”

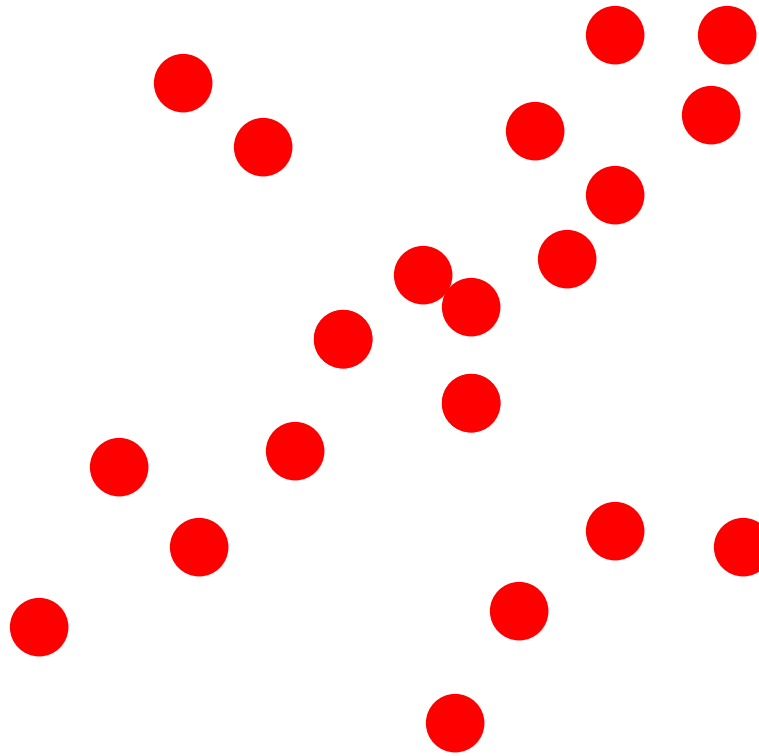


Is this more or less efficient than voting with features?

RANSAC

(**RAN**dom **SA**mples **C**onsensus) :

Fischler & Bolles in '81.



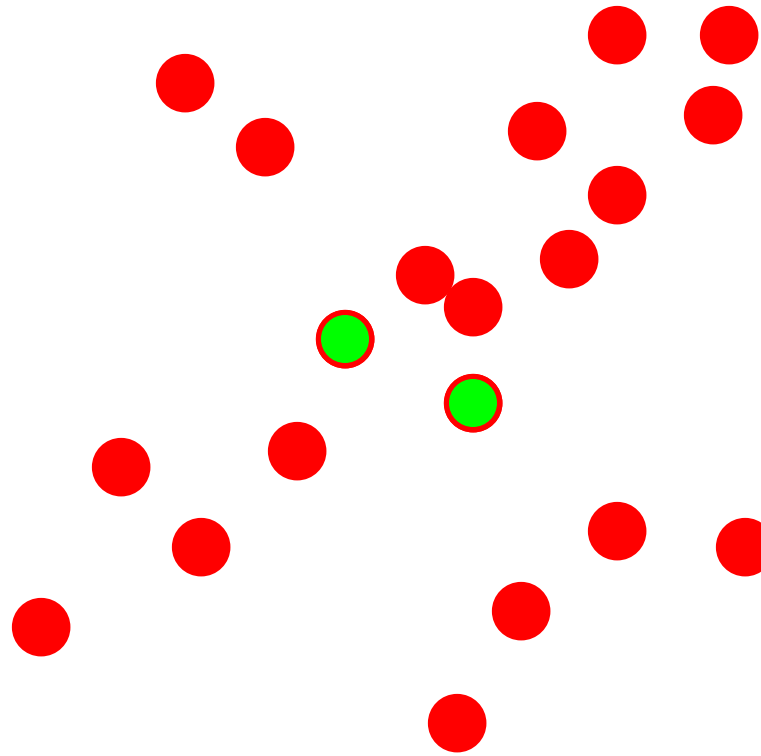
Algorithm:

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



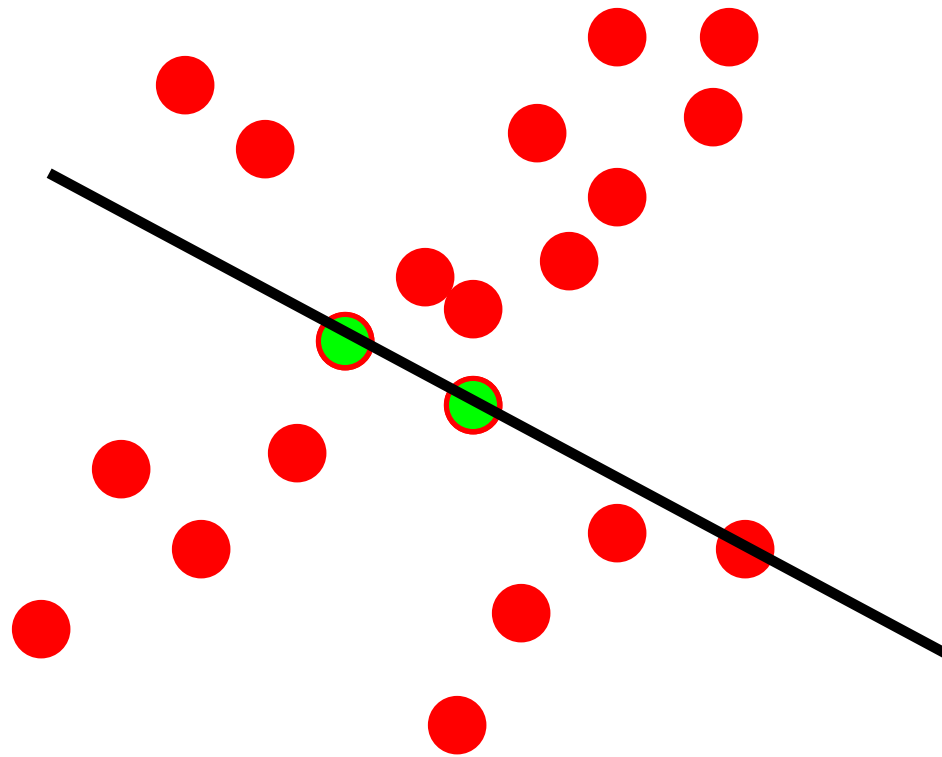
Algorithm:

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

RANSAC

Line fitting example



Algorithm:

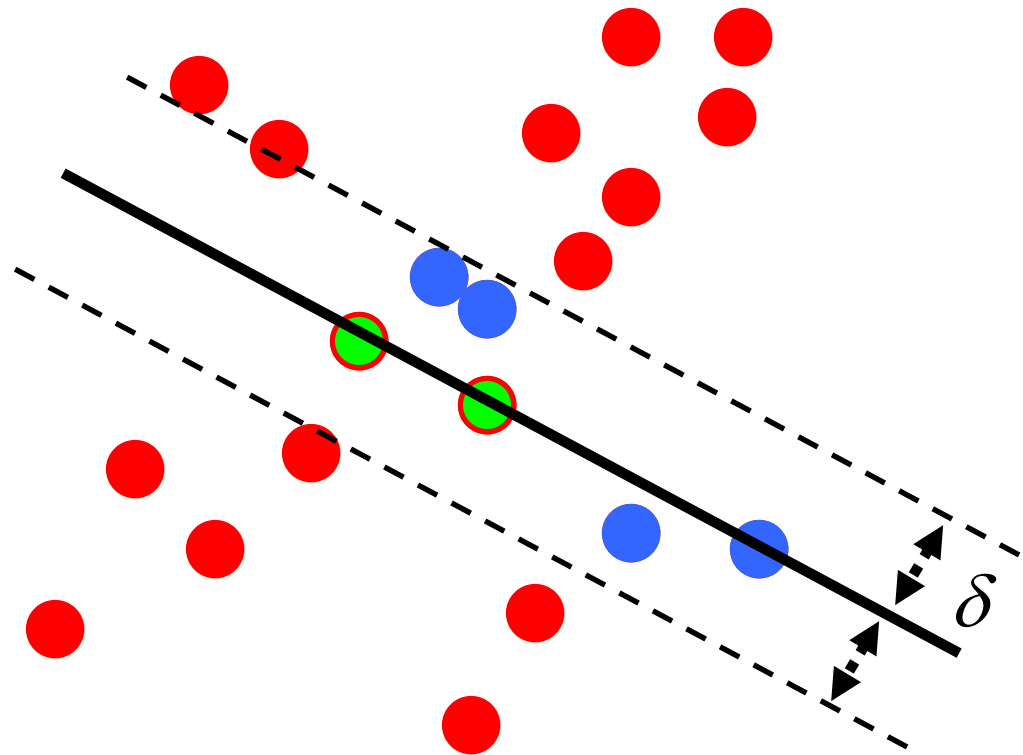
1. **Sample** (randomly) the number of points required to fit the model ($\# = 2$)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

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

RANSAC

Line fitting example

$$N_I = 6$$

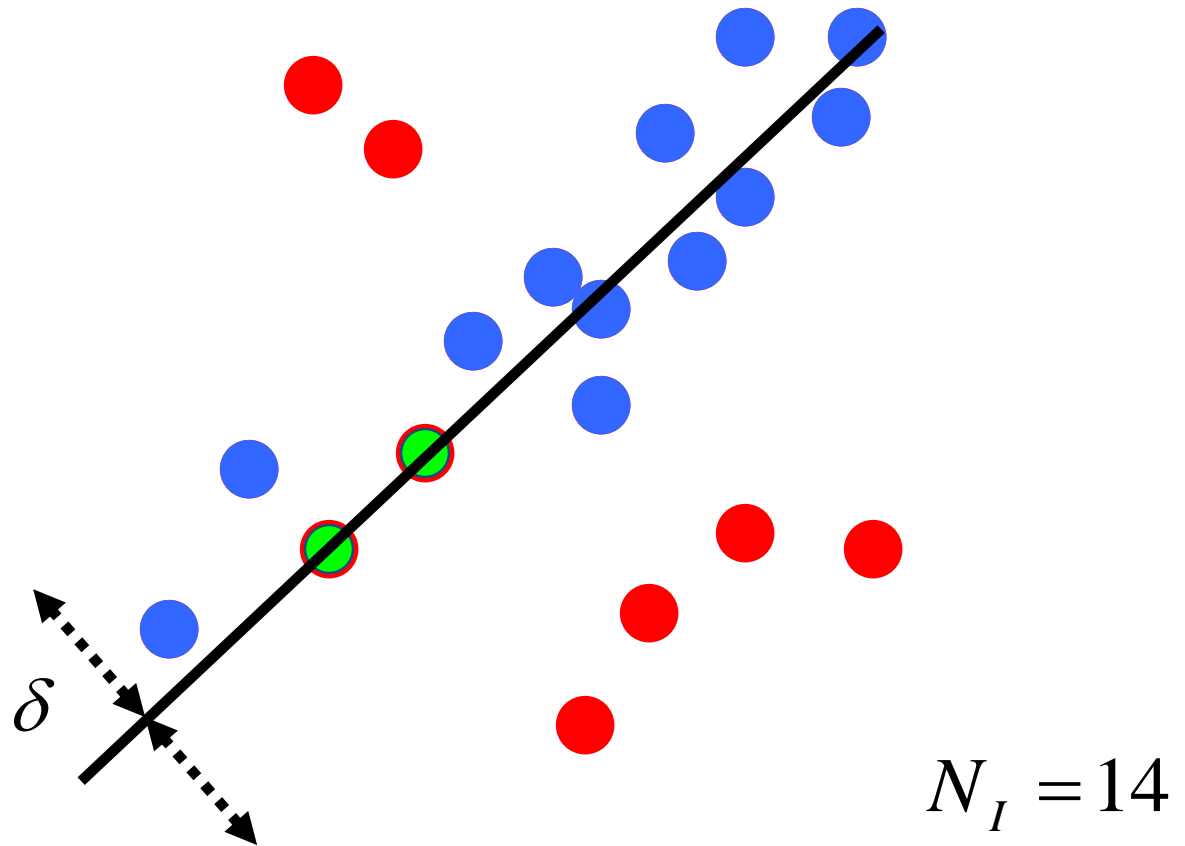


Algorithm:

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 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 : $t^2=3.84\sigma^2$

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

s	proportion of outliers e						
	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

For $p = 0.99$

RANSAC conclusions

Good

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

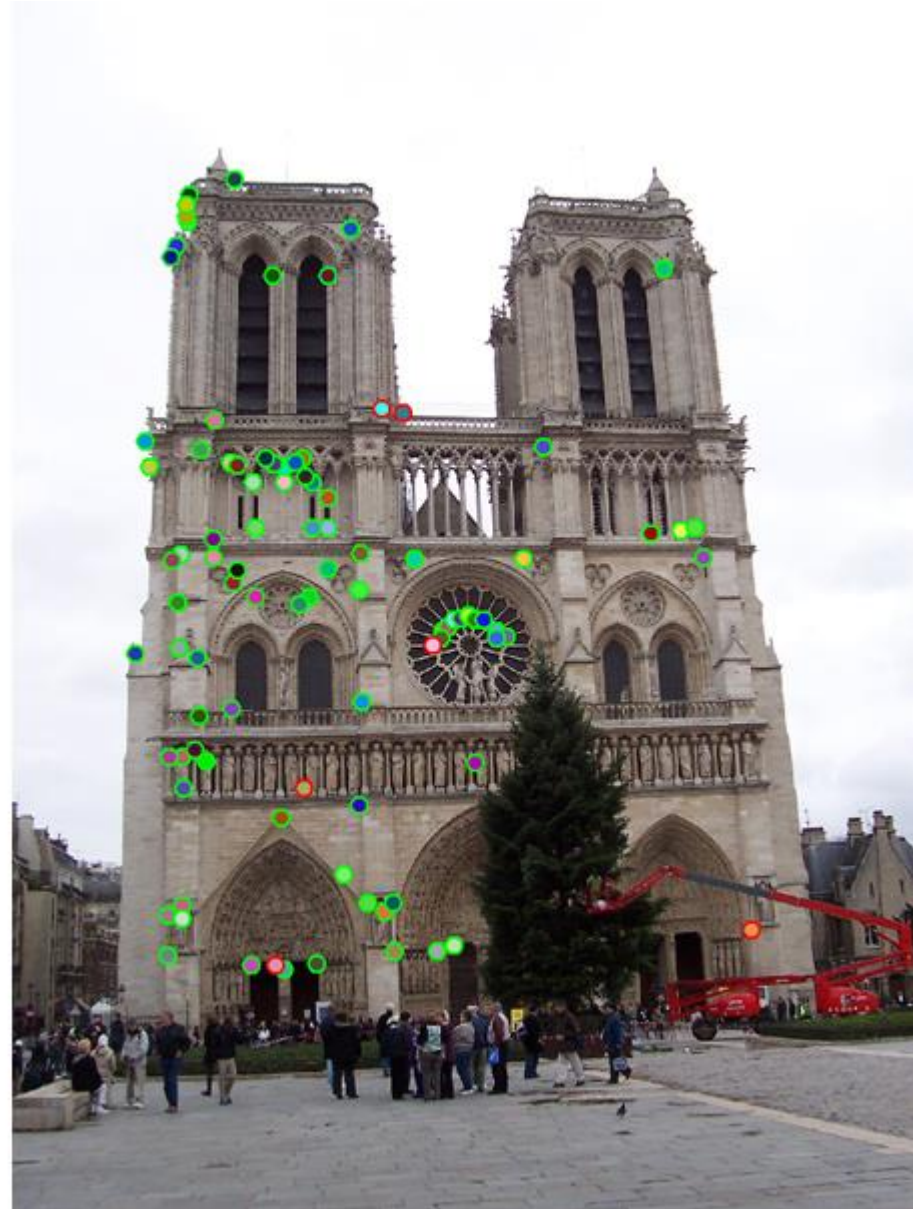
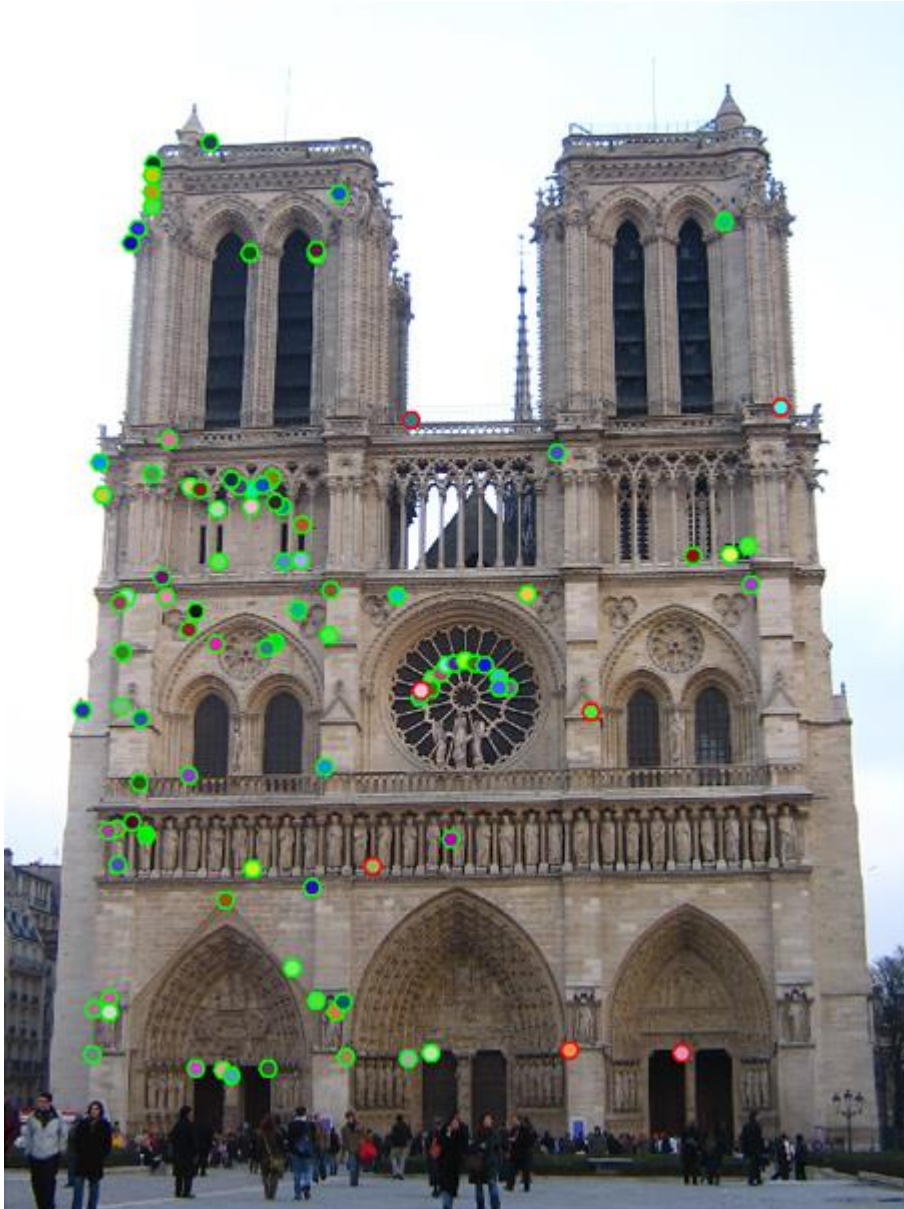
Bad

- 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?



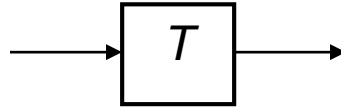
Alignment

- 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



$$\mathbf{p} = (x, y)$$



$$\mathbf{p}' = (x', y')$$

Transformation T is a coordinate-changing machine:

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

What does it mean that T is global?

- Is the same for any point \mathbf{p}
- can be described by just a few numbers (parameters)

For linear transformations, we can represent T as a matrix

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

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

Common transformations



original

Transformed



translation



rotation



aspect



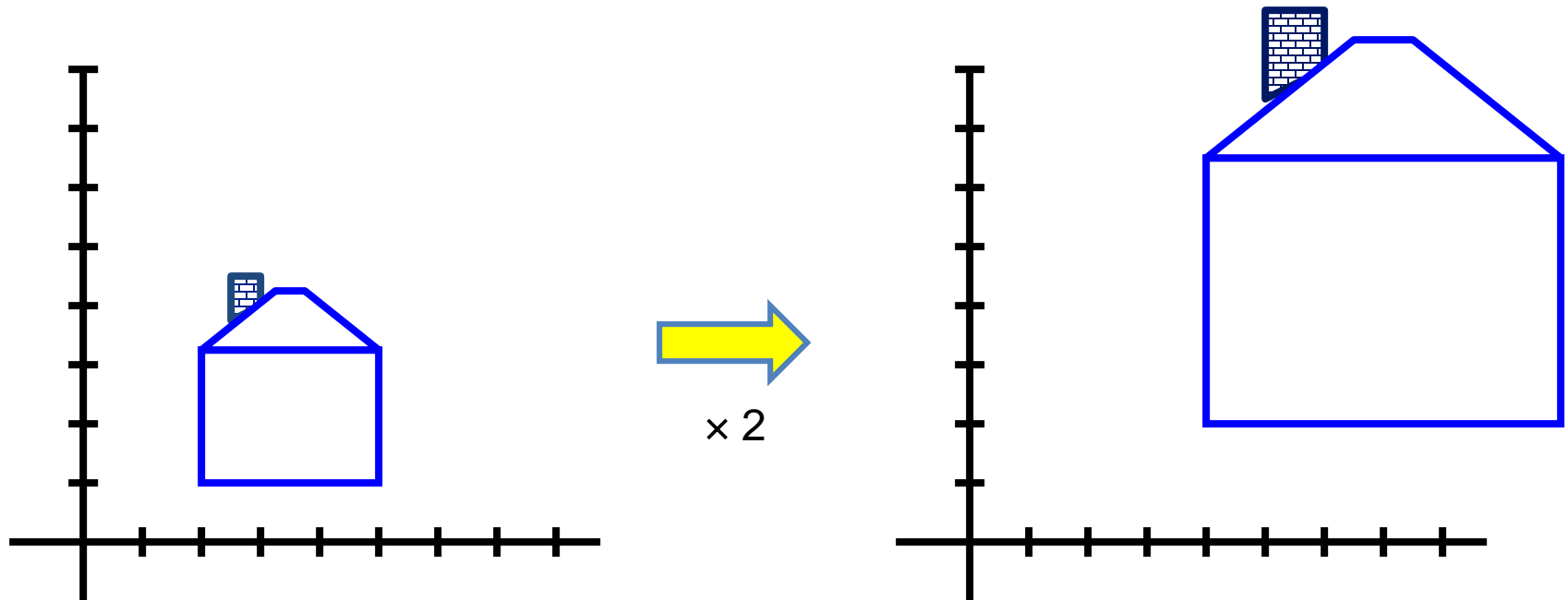
affine



perspective

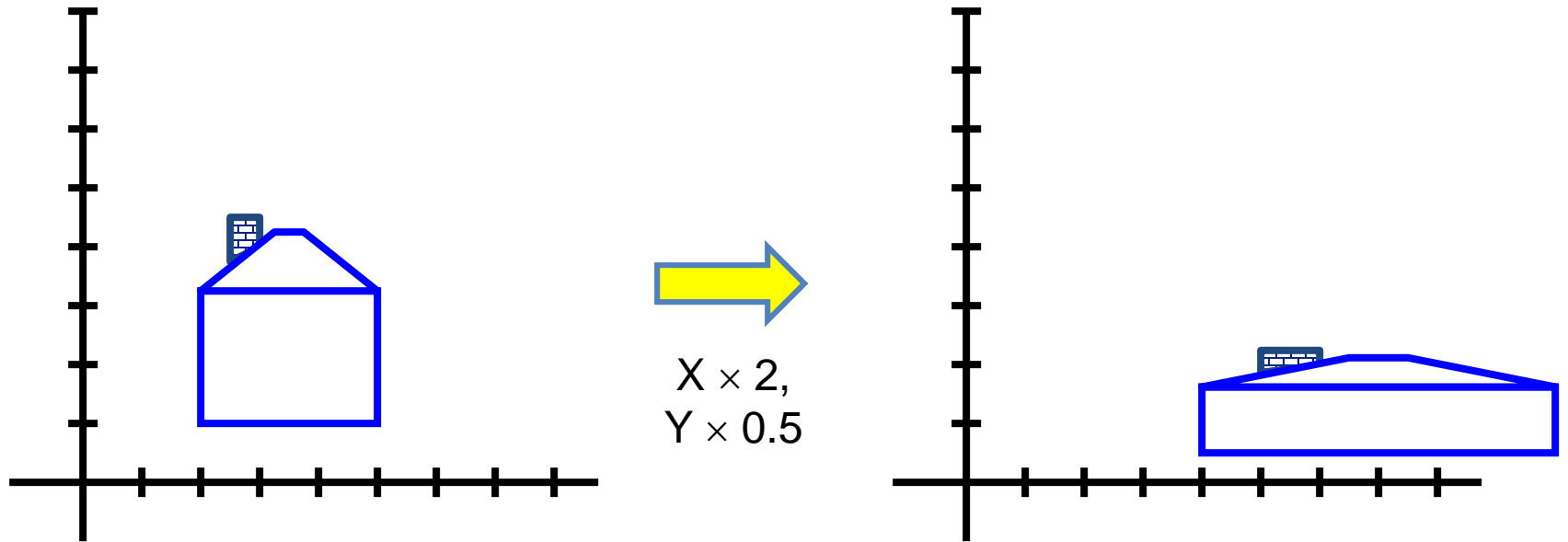
Scaling

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



Scaling

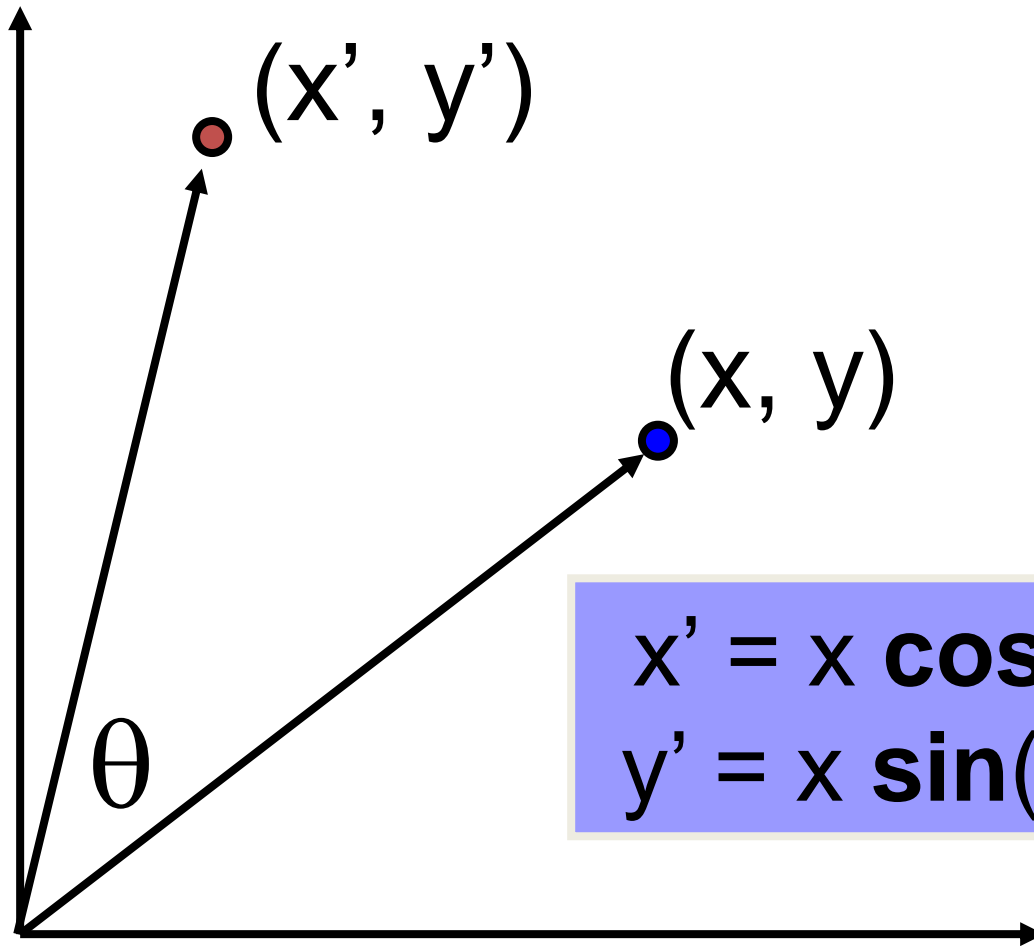
- *Non-uniform scaling*: different scalars per component:



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



$$\begin{aligned}x' &= x \cos(\theta) - y \sin(\theta) \\y' &= x \sin(\theta) + y \cos(\theta)\end{aligned}$$

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 $\sin(\theta)$ and $\cos(\theta)$ 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 $-\theta$
- 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

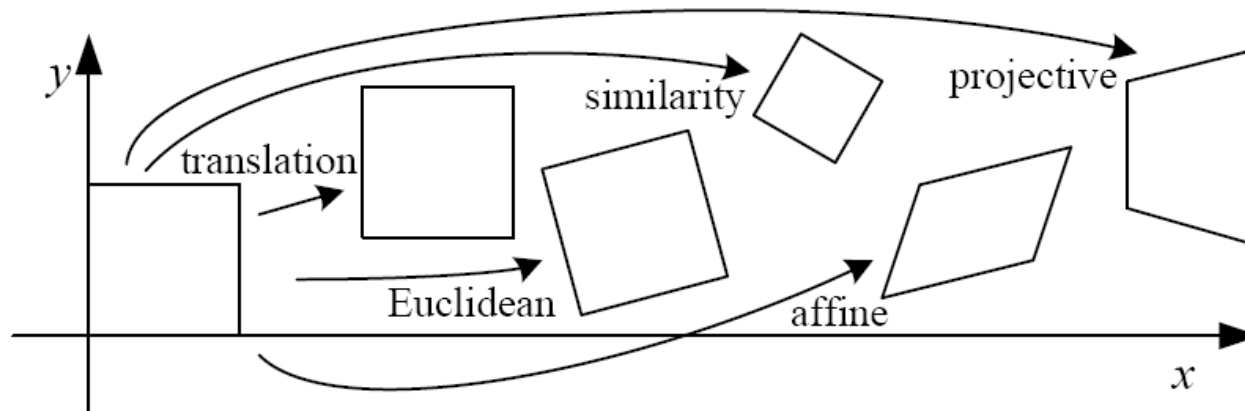
- Affine transformations, and
- Projective warps

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

Properties of projective transformations:

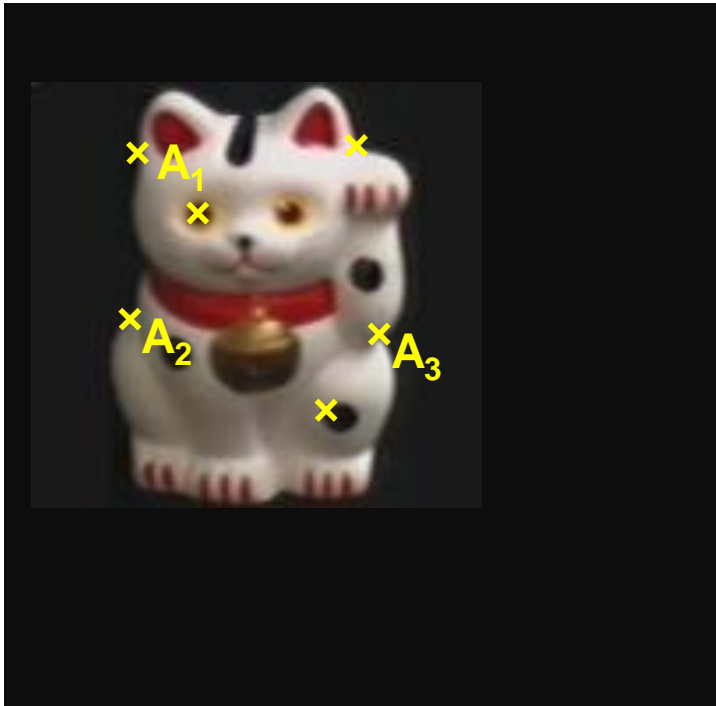
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis
- Projective matrix is defined up to a scale (8 DOF)

2D image transformations (reference table)



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

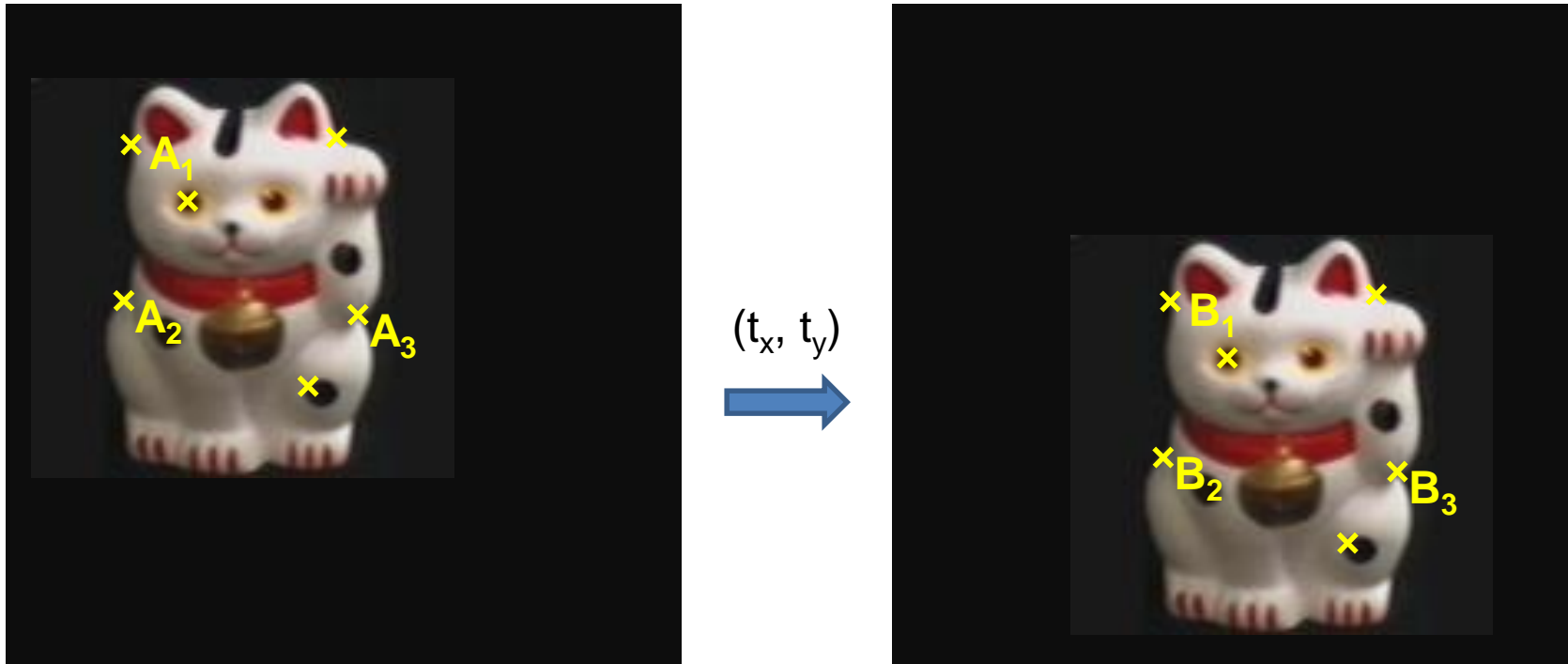
Example: solving for translation



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



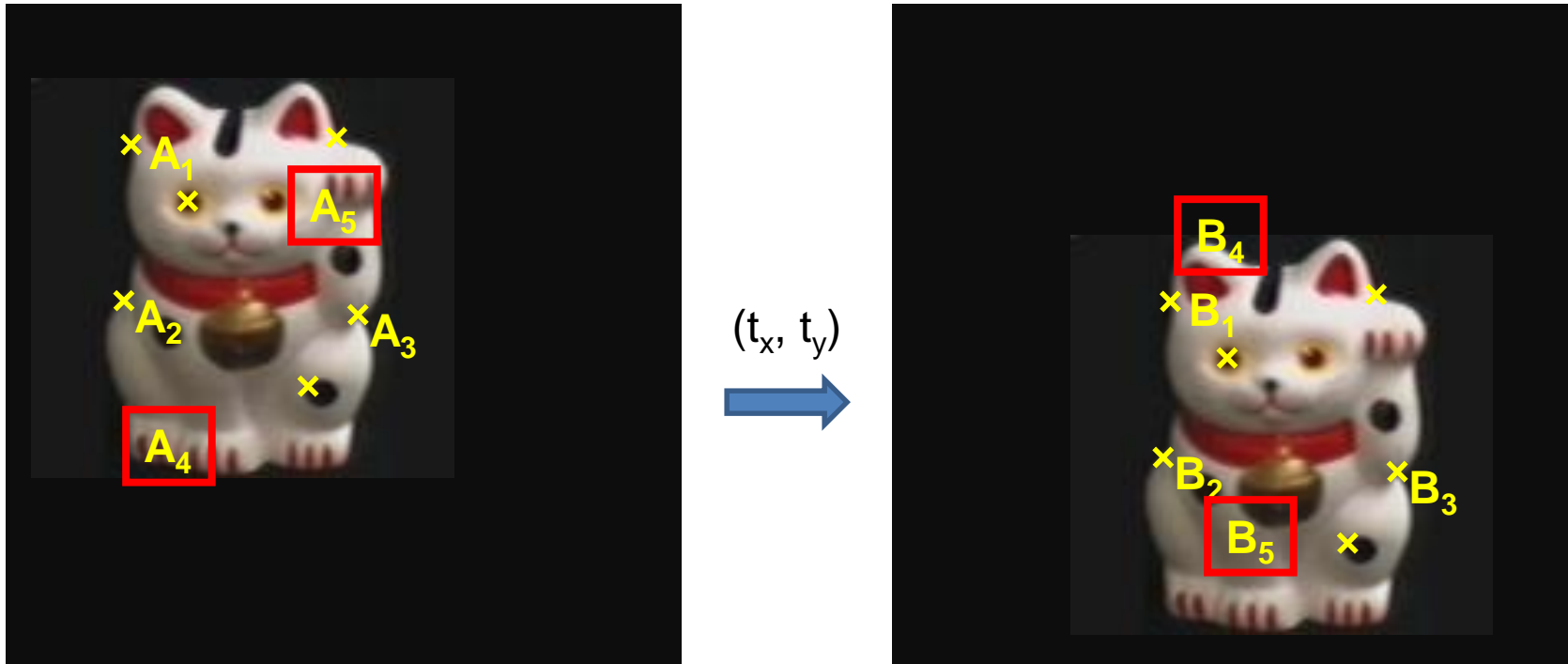
Least squares solution

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



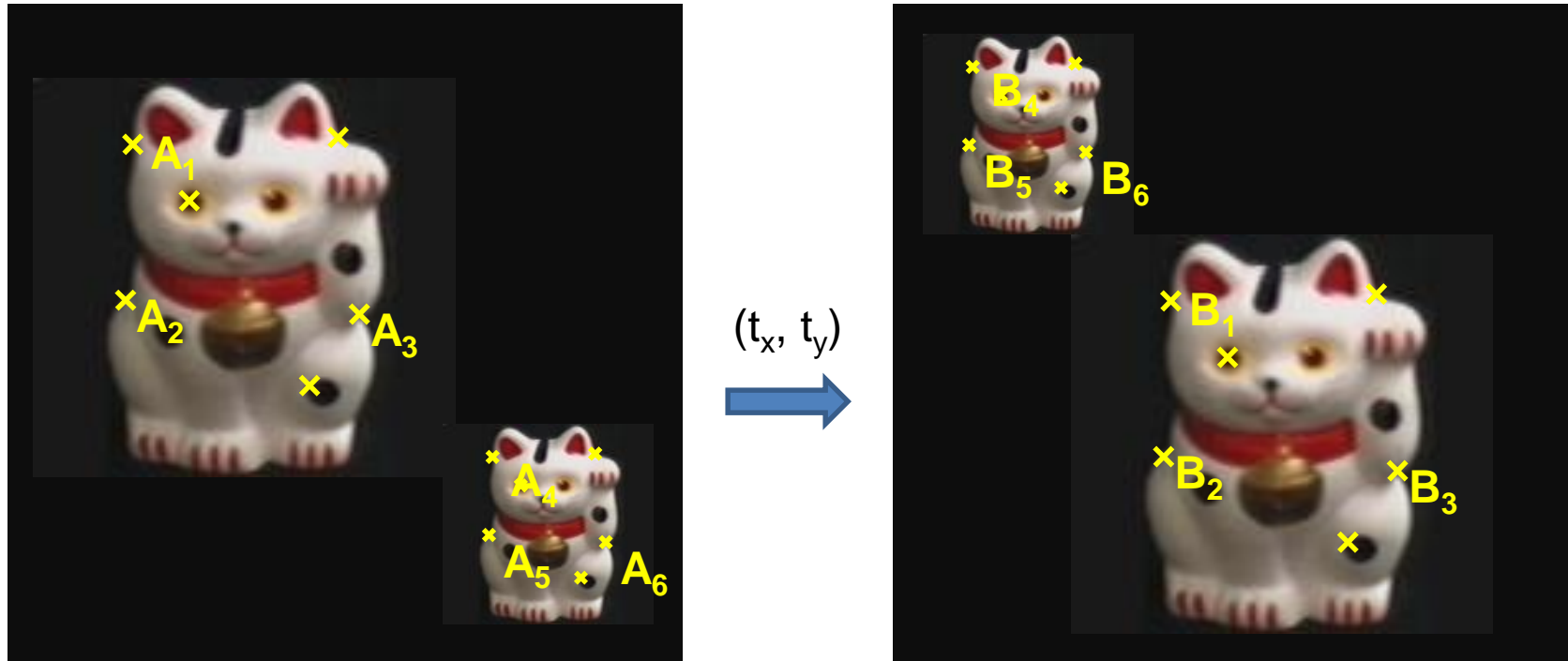
Problem: outliers

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



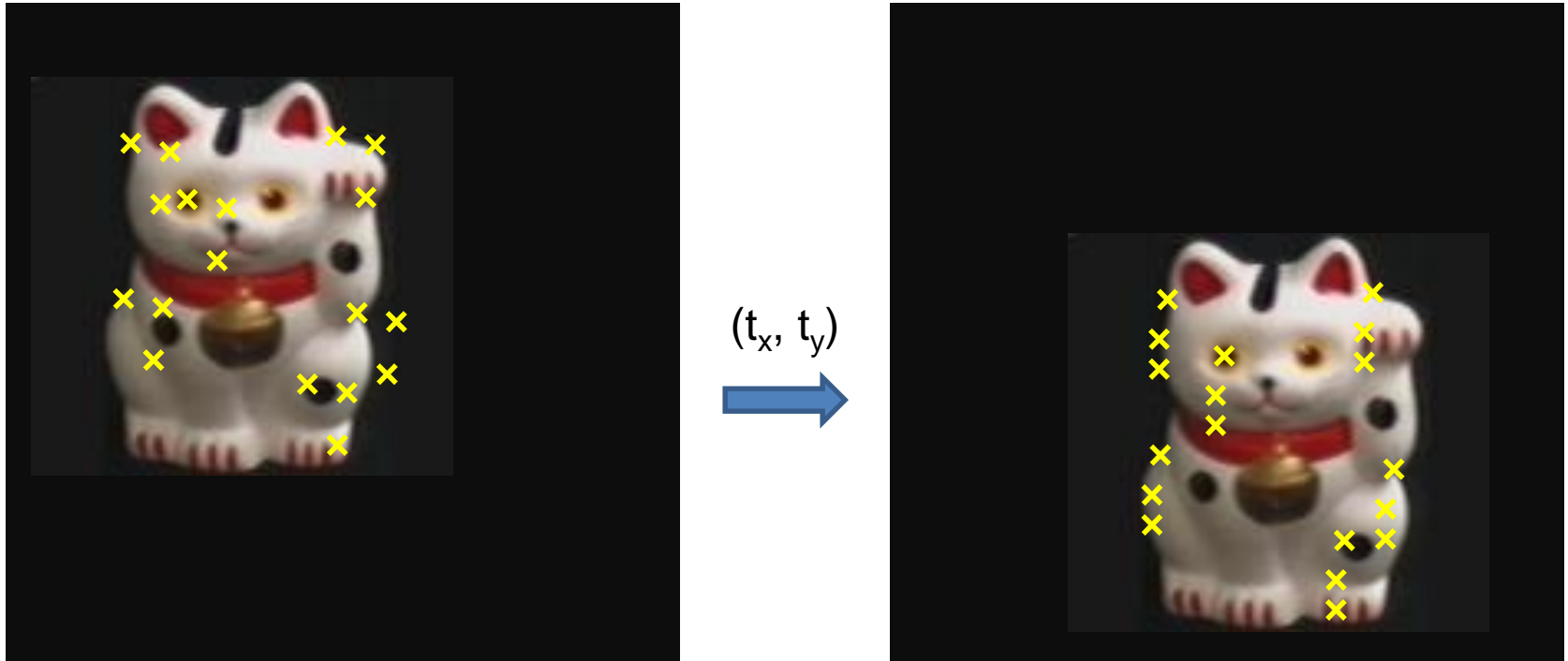
Problem: outliers, multiple objects, and/or many-to-one matches

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

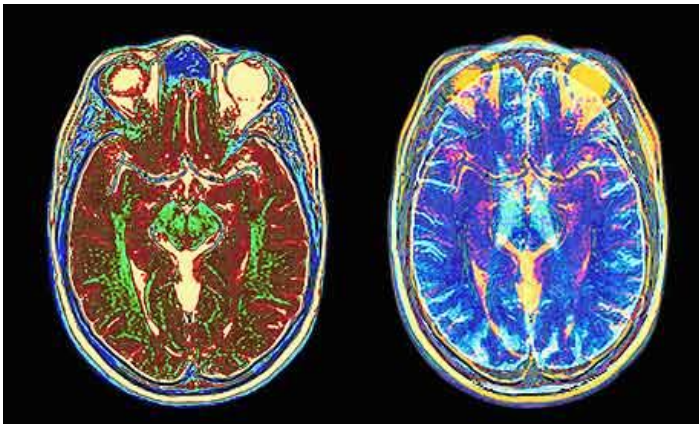
$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Fitting and Alignment: Methods

- Global optimization / Search for parameters
 - Least squares fit
 - Robust least squares
 - Other parameter search methods
- Hypothesize and test
 - Generalized Hough transform
 - RANSAC
- Iterative Closest Points (ICP)

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

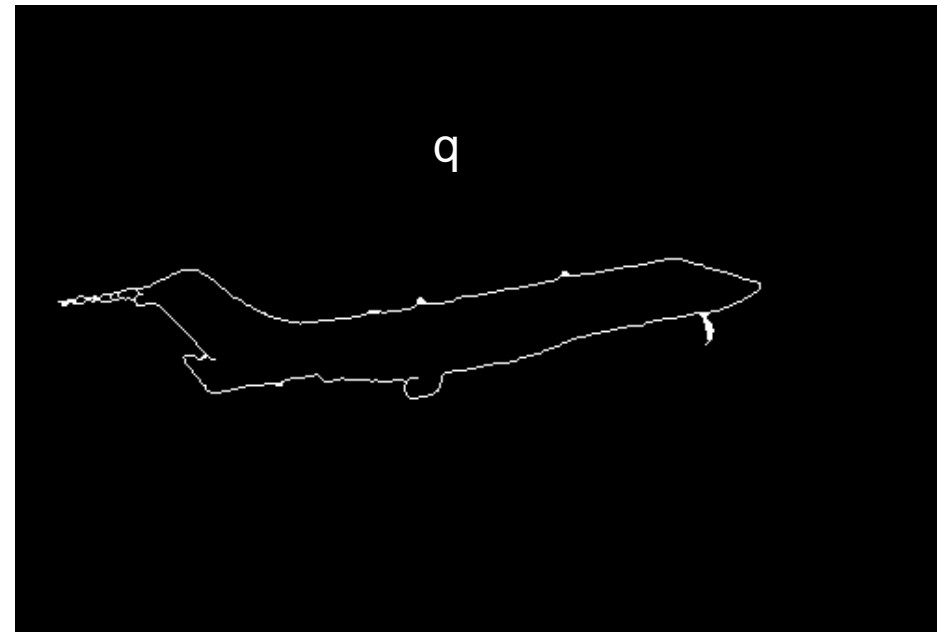
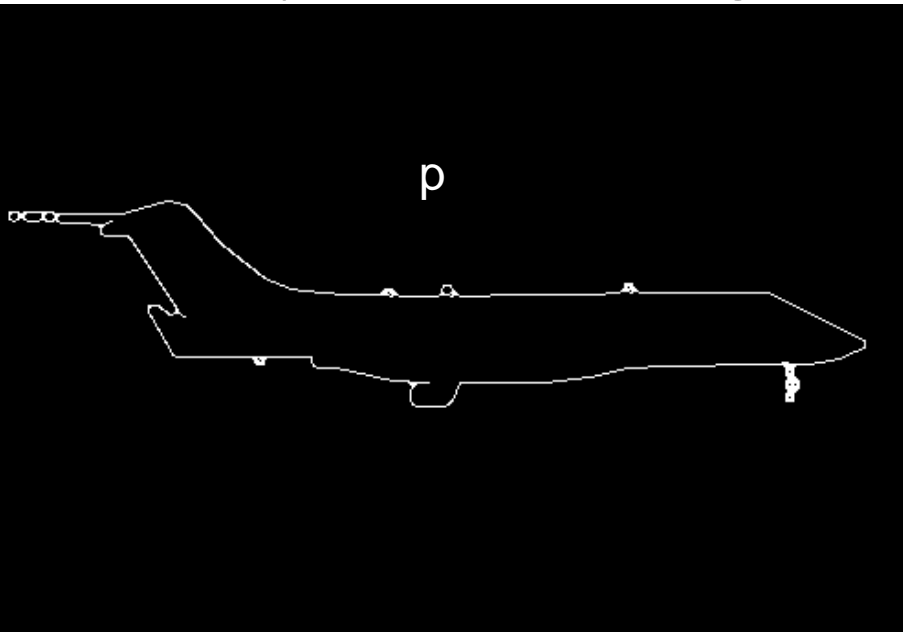
Iterative Closest Points (ICP) Algorithm

Goal: estimate transform between two dense sets of points

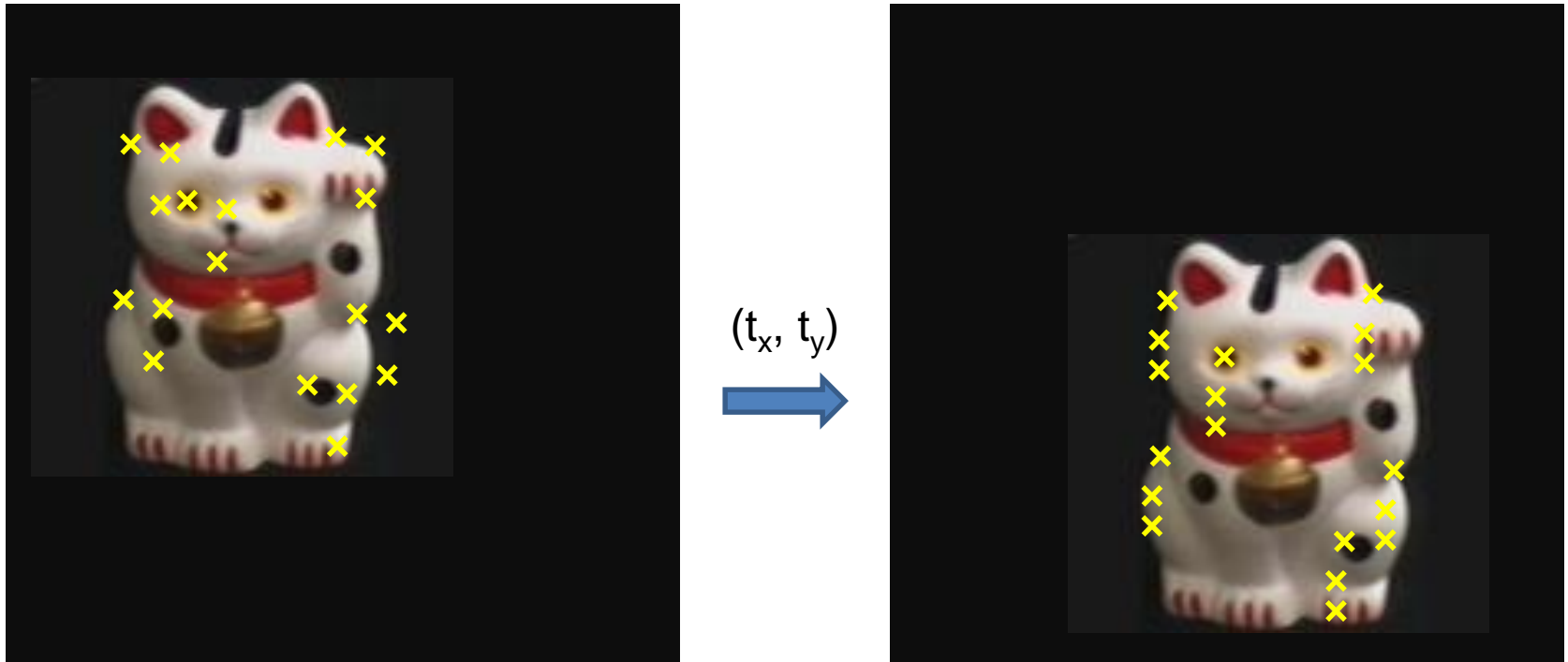
1. **Initialize** transformation (e.g., compute difference in means and scale)
2. **Assign** each point in {Set 1} to its nearest neighbor in {Set 2}
3. **Estimate** transformation parameters
 - e.g., least squares or robust least squares
4. **Transform** the points in {Set 1} using estimated parameters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

Example: aligning boundaries

1. Extract edge pixels $p_1 \dots p_n$ and $q_1 \dots q_m$
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 $\text{match}(i) = \underset{j}{\operatorname{argmin}} \operatorname{dist}(p_i, q_j)$
4. Compute transformation \mathbf{T} based on matches
5. Warp points \mathbf{p} according to \mathbf{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

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

Sparse ICP

Sofien Bouaziz Andrea Tagliasacchi Mark Pauly



Algorithm Summaries

- Least Squares Fit
 - closed form solution
 - robust to noise
 - not robust to outliers
- Robust Least Squares
 - improves robustness to outliers
 - 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