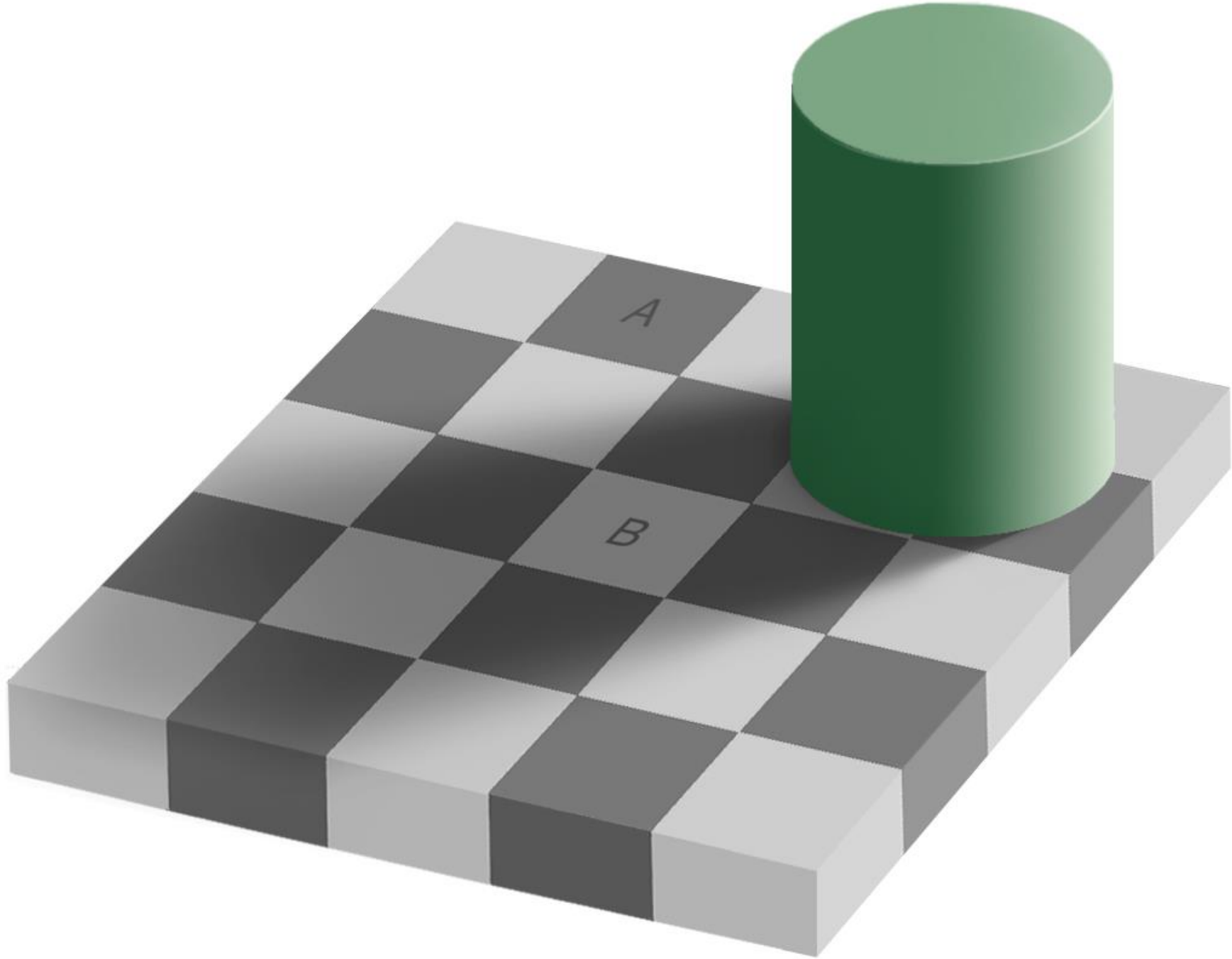
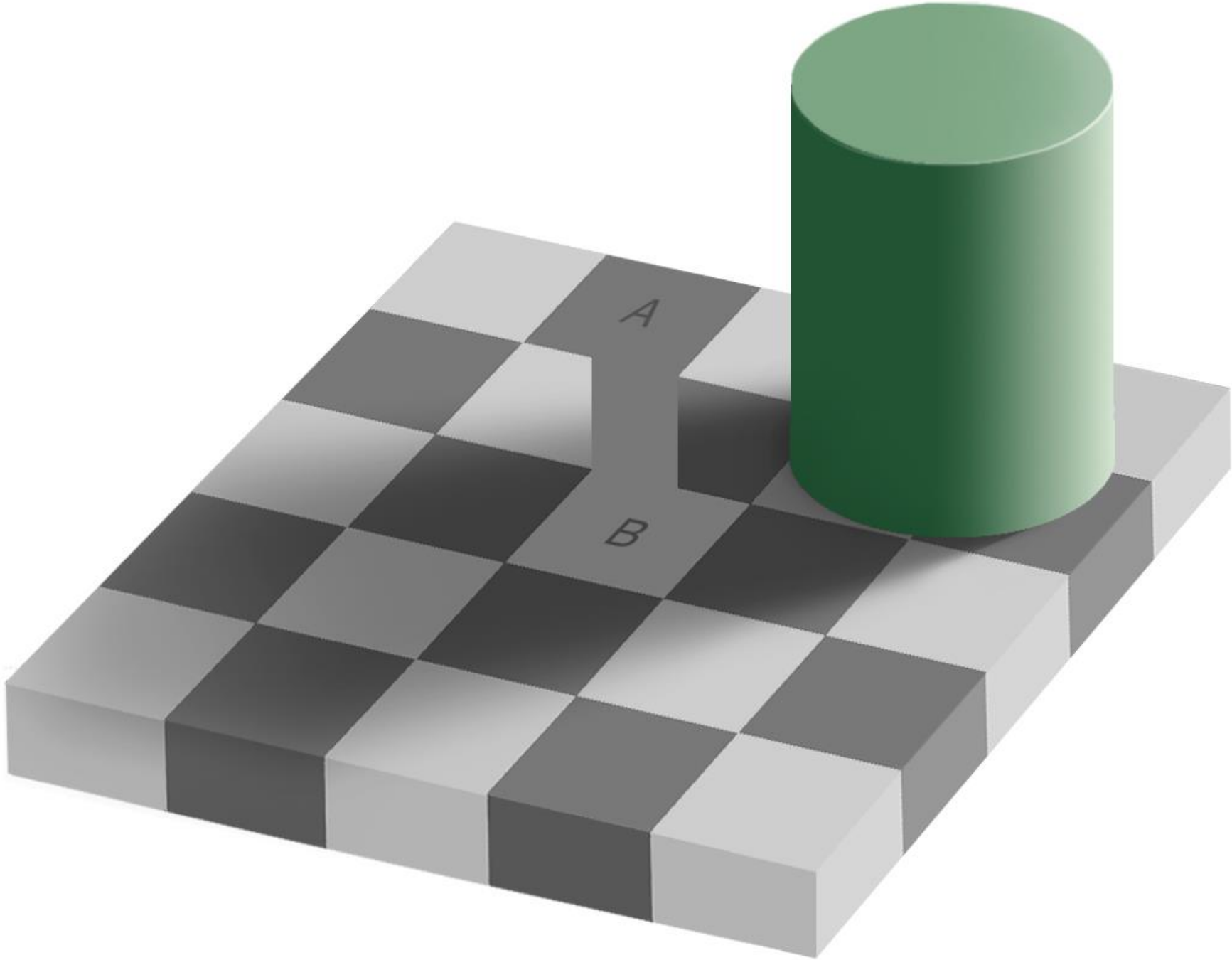
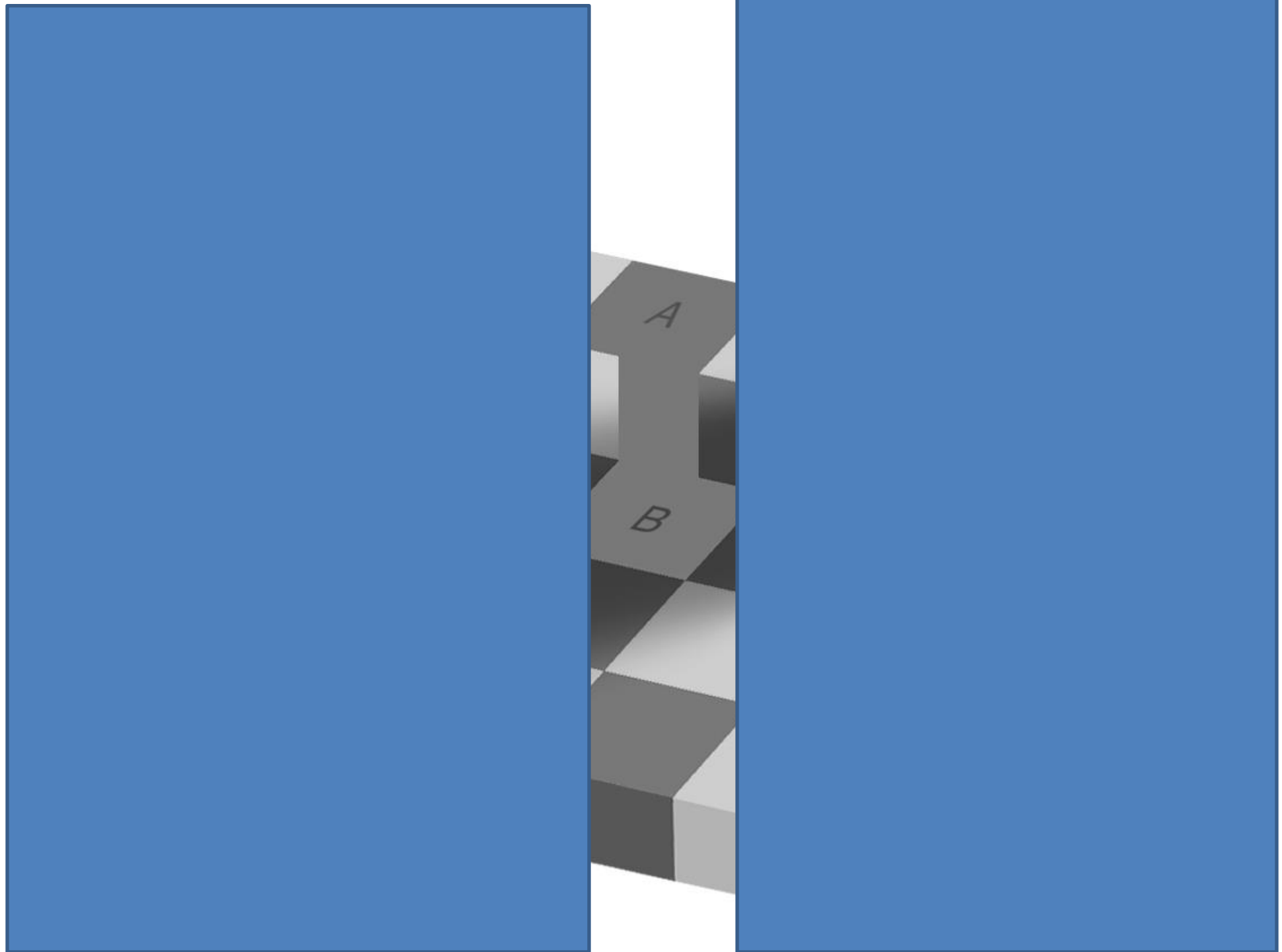
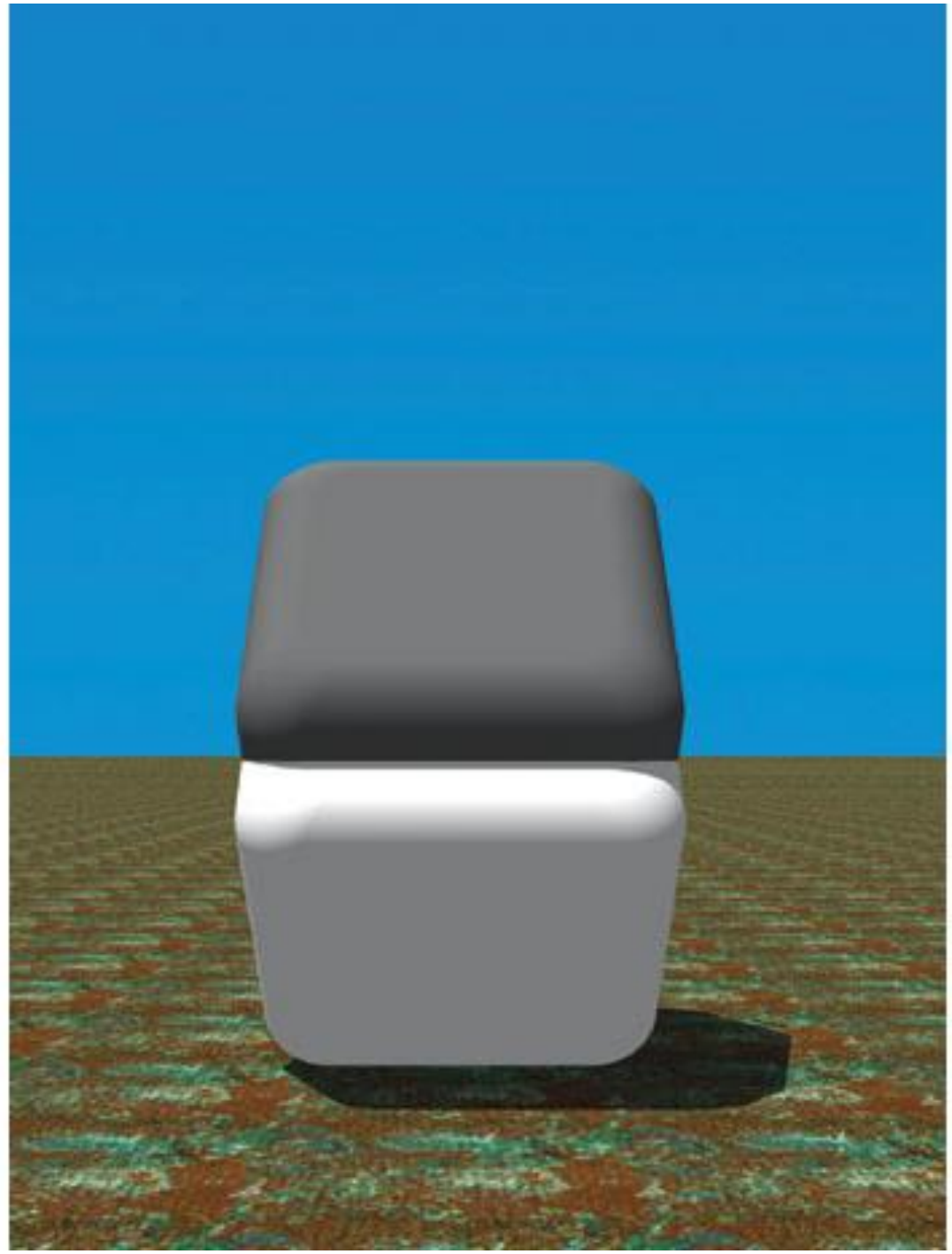

A
B


A
B


A
B



RANSAC, ICP, Fitting and Alignment

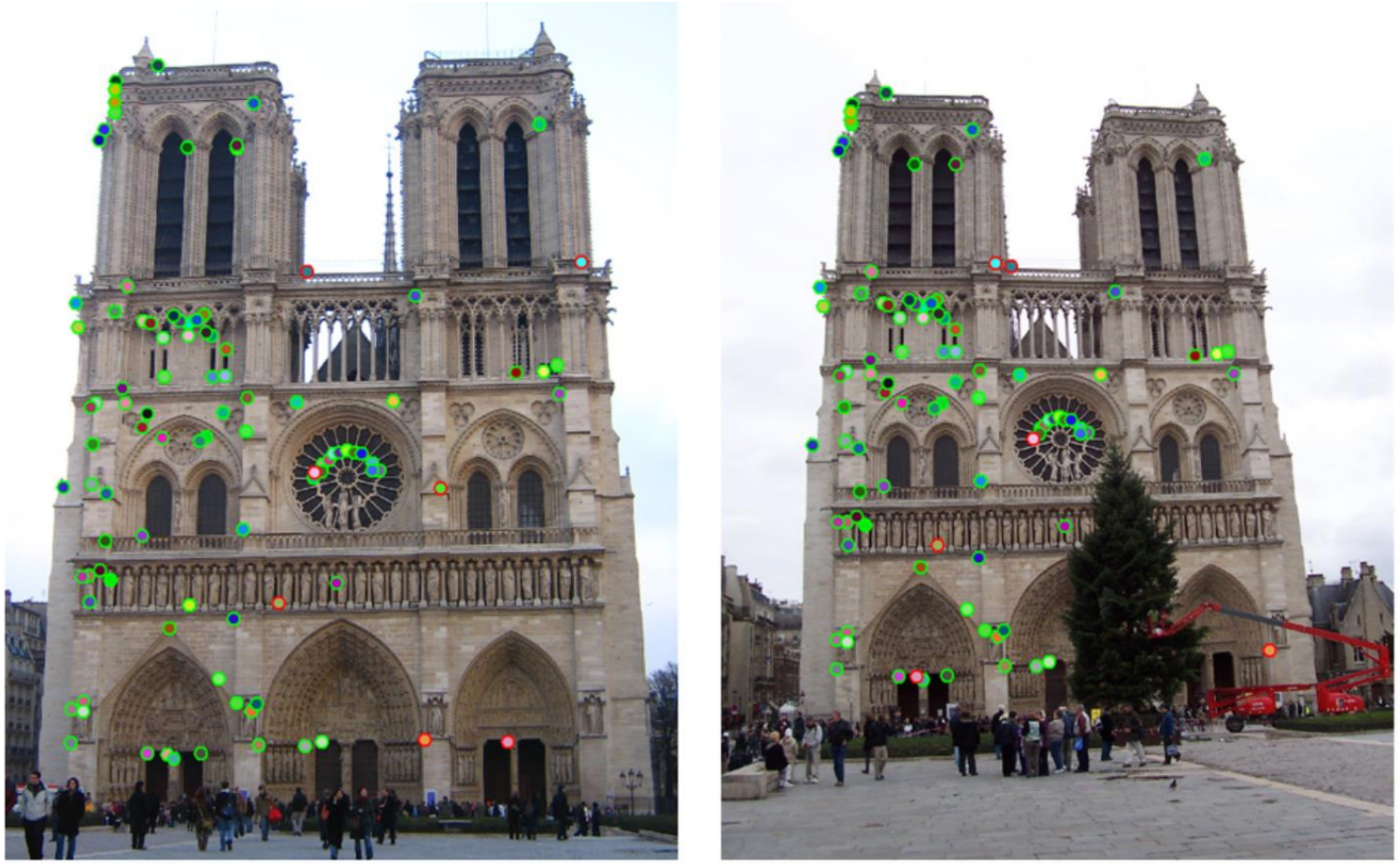
Computer Vision

Szeliski 2.1
and 8.1

James Hays

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

Project 2



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

Fitting and Alignment: Methods

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

Review: Hough Transform

1. Create a grid of parameter values

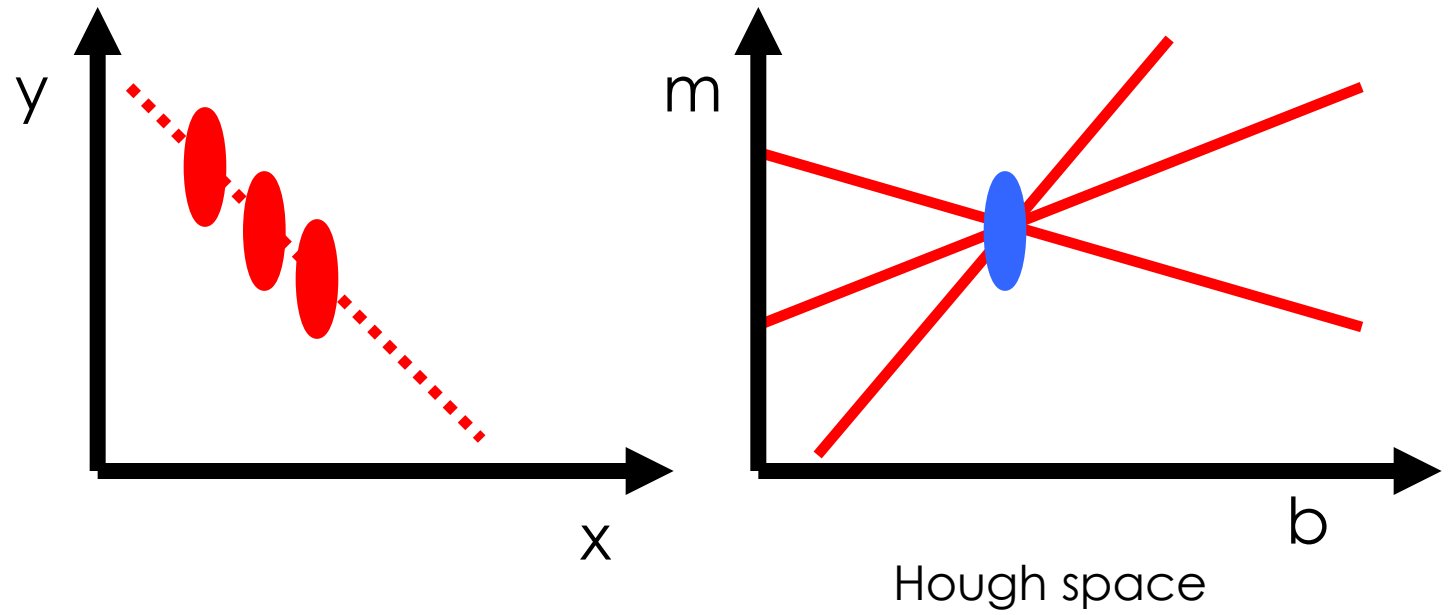
2. Each point (or correspondence) votes for a set of parameters, incrementing those values in grid

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

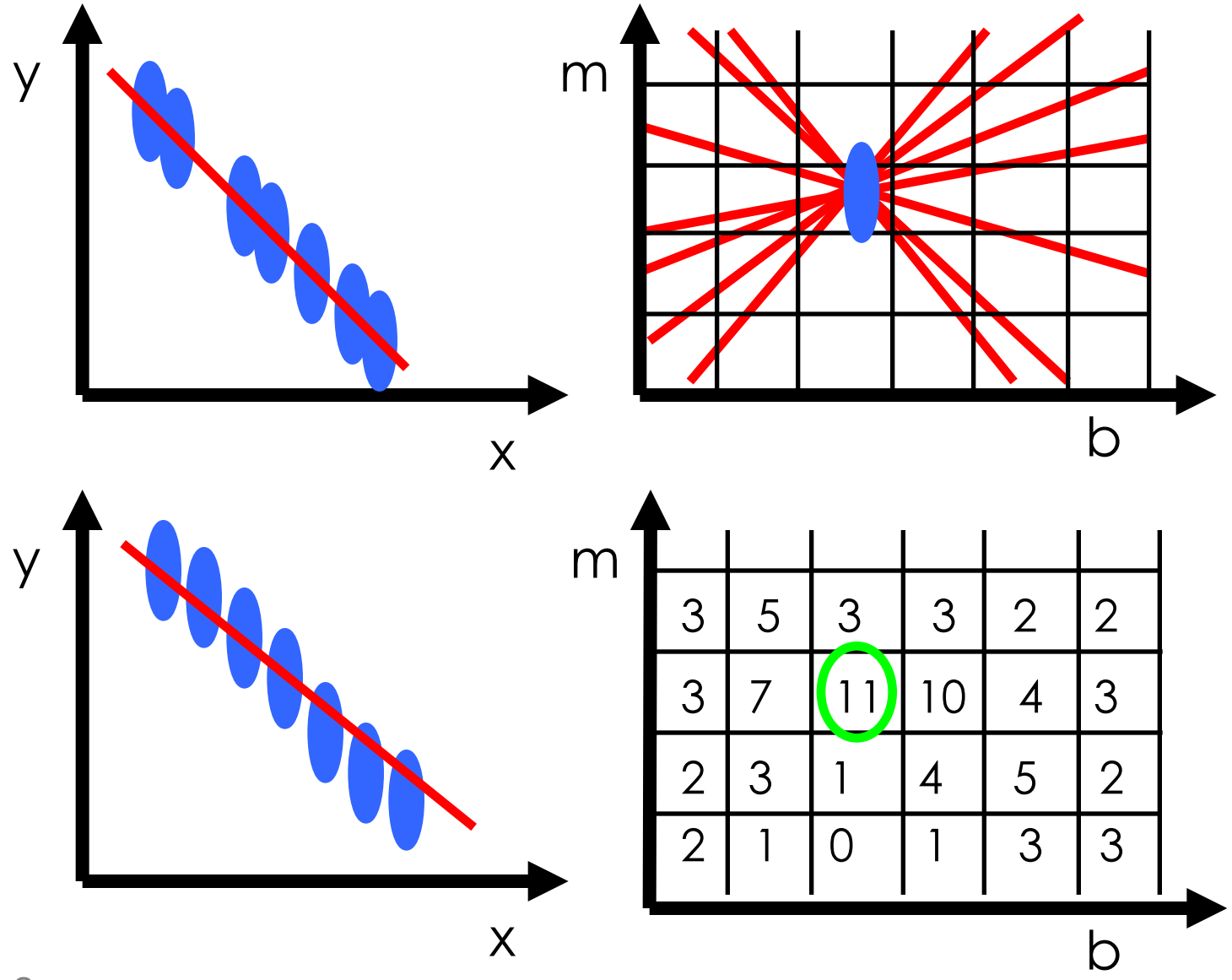
Given a set of points, find the curve or line that explains the data points best



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

Slide from S. Savarese

Review: Hough transform



Slide from S. Savarese

Hough transform for circles

For every edge pixel (x,y) :

  For each possible radius value r:

    For each possible direction θ:

      // or use estimated gradient at (x,y)

        $a = x - r\cos(\theta)$ // column

        $b = y + r\sin(\theta)$ // row

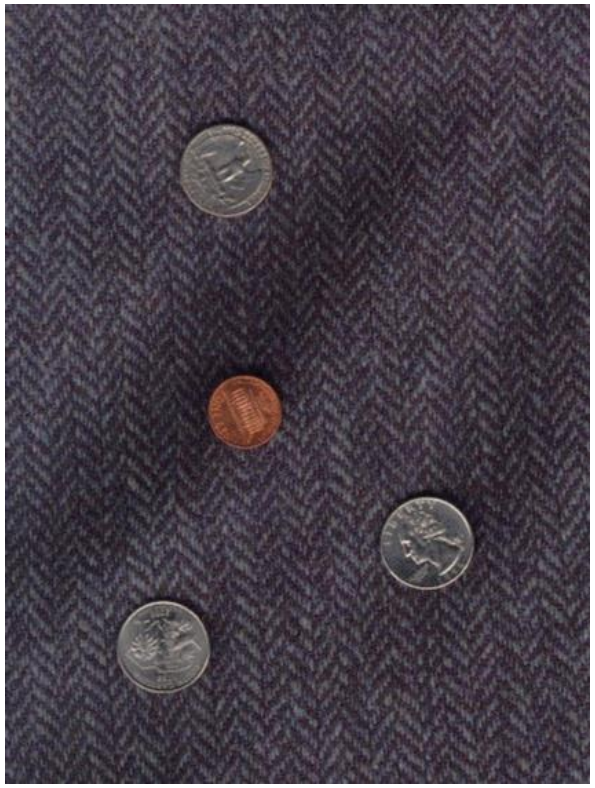
        H[a,b,r] += 1

  end

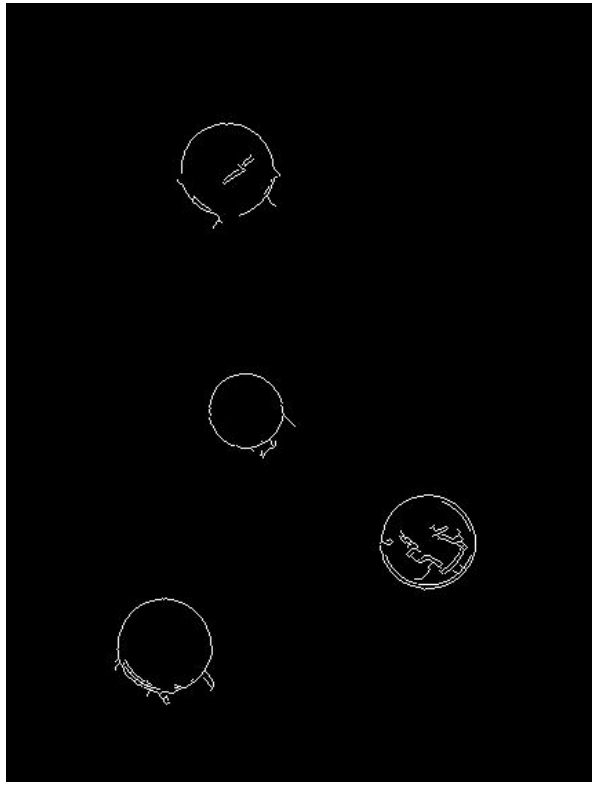
end

Example: detecting circles with Hough

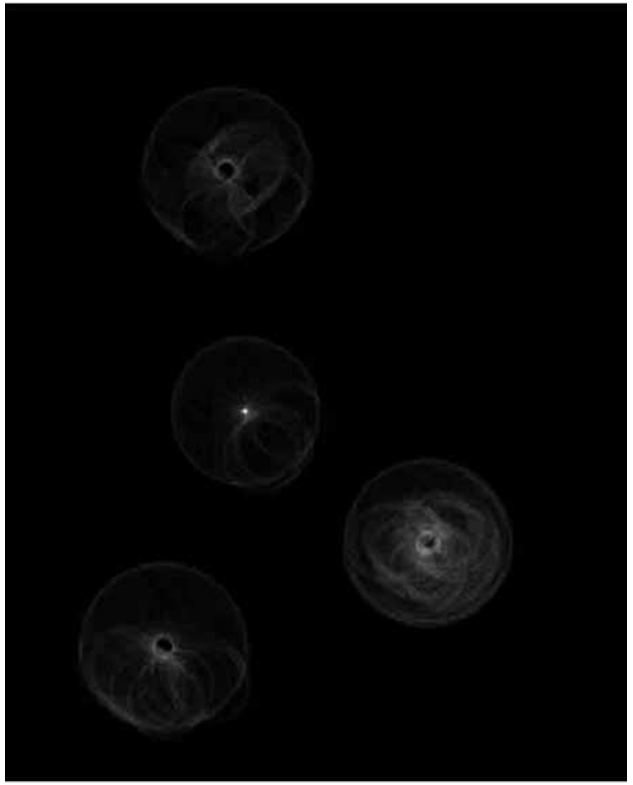
Original



Edges



Votes: Penny

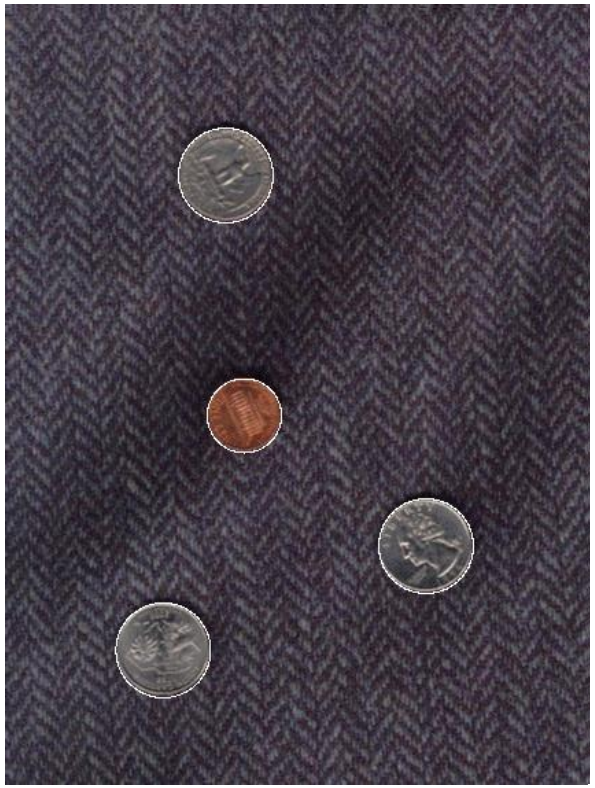


Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

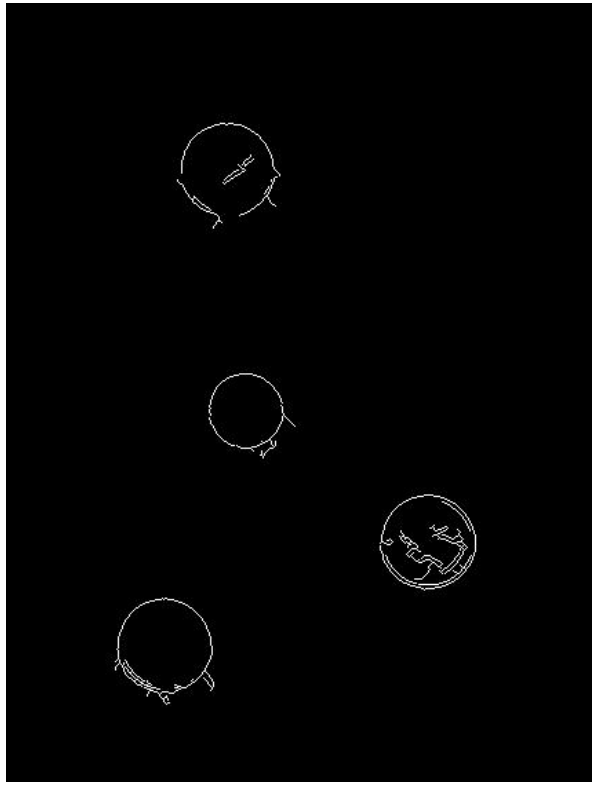
Slide credit: Kristen Grauman

Example: detecting circles with Hough

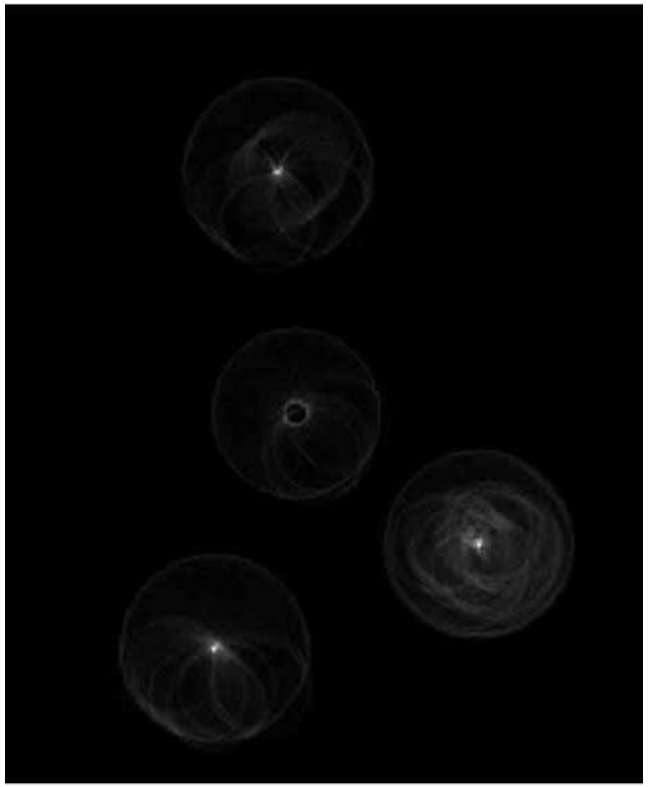
Original



Edges



Votes: Quarter



Combined detections

Slide credit: Kristen Grauman

Coin finding sample images from: Vivek Kwatra

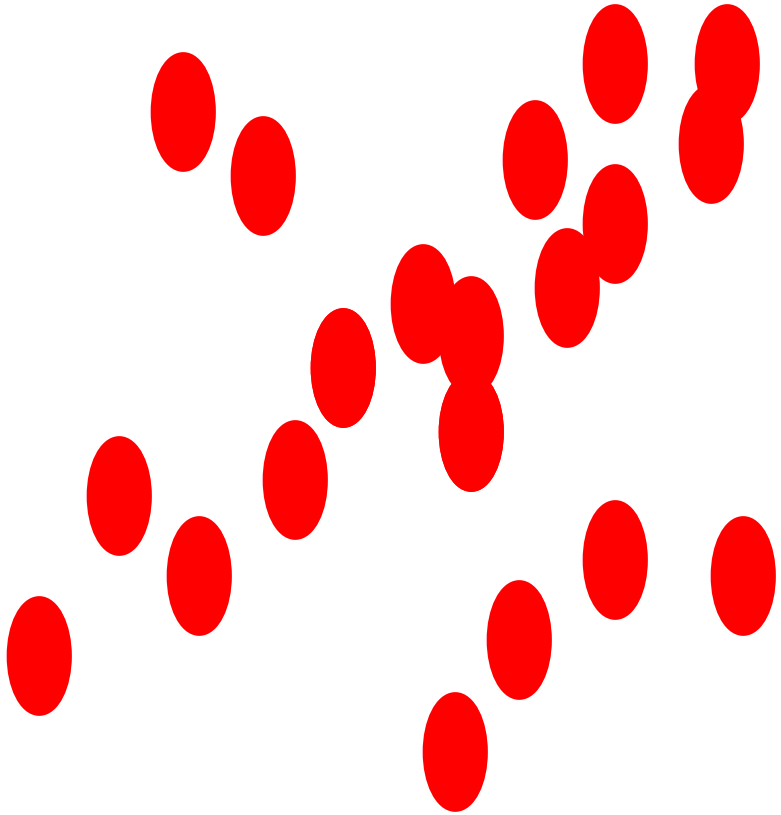
Fitting and Alignment: Methods

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

RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in ‘81.



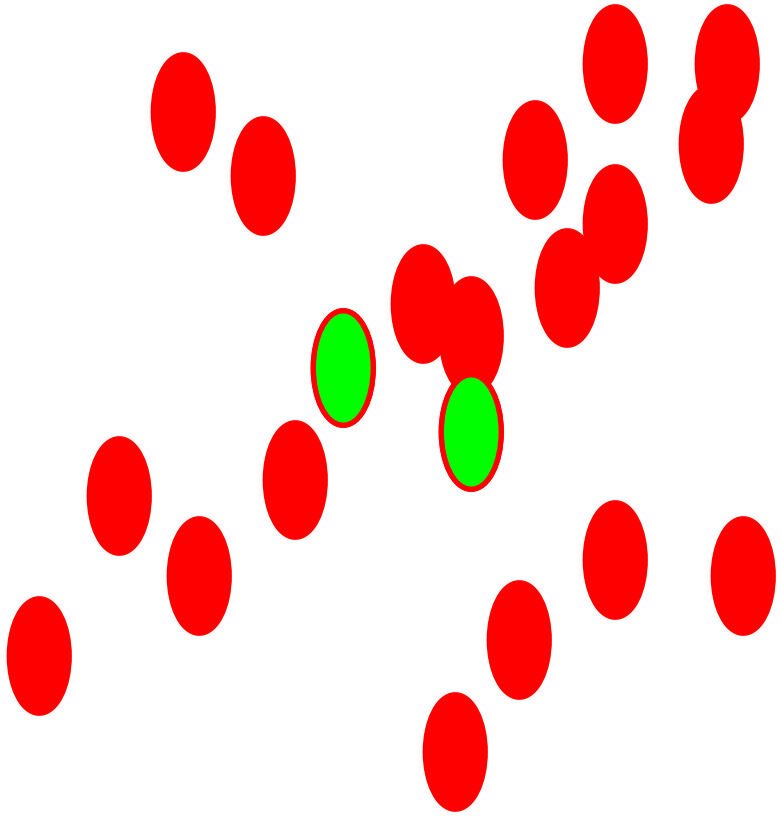
Algorithm:

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:

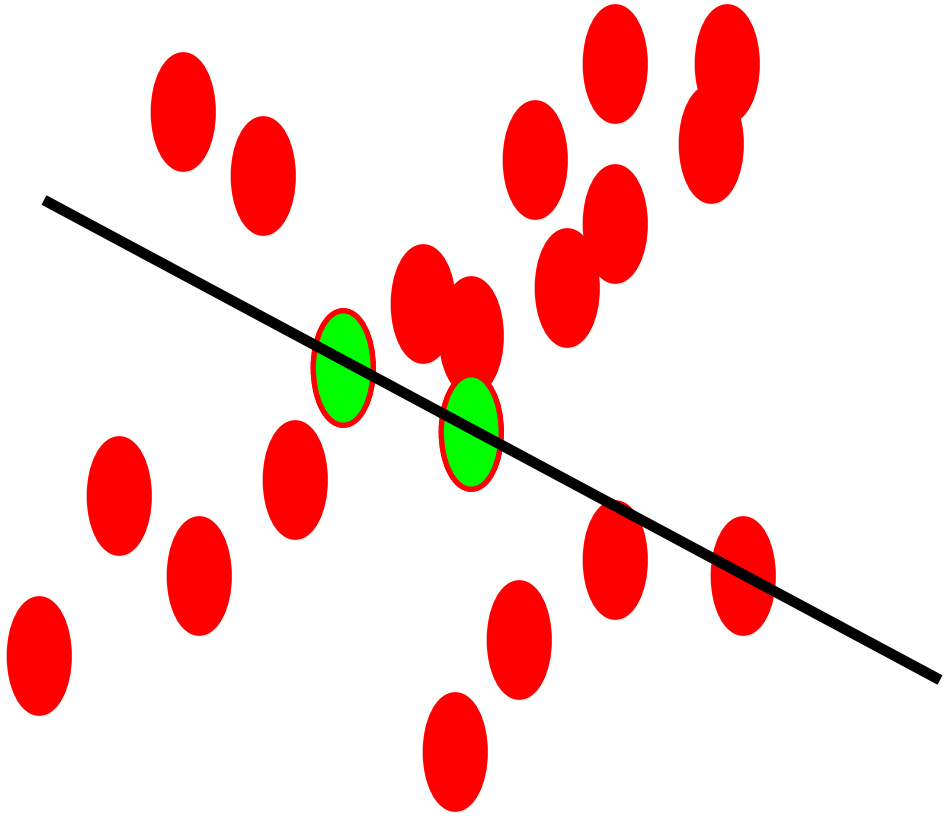
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

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

Illustration by Savarese

RANSAC

Line fitting example



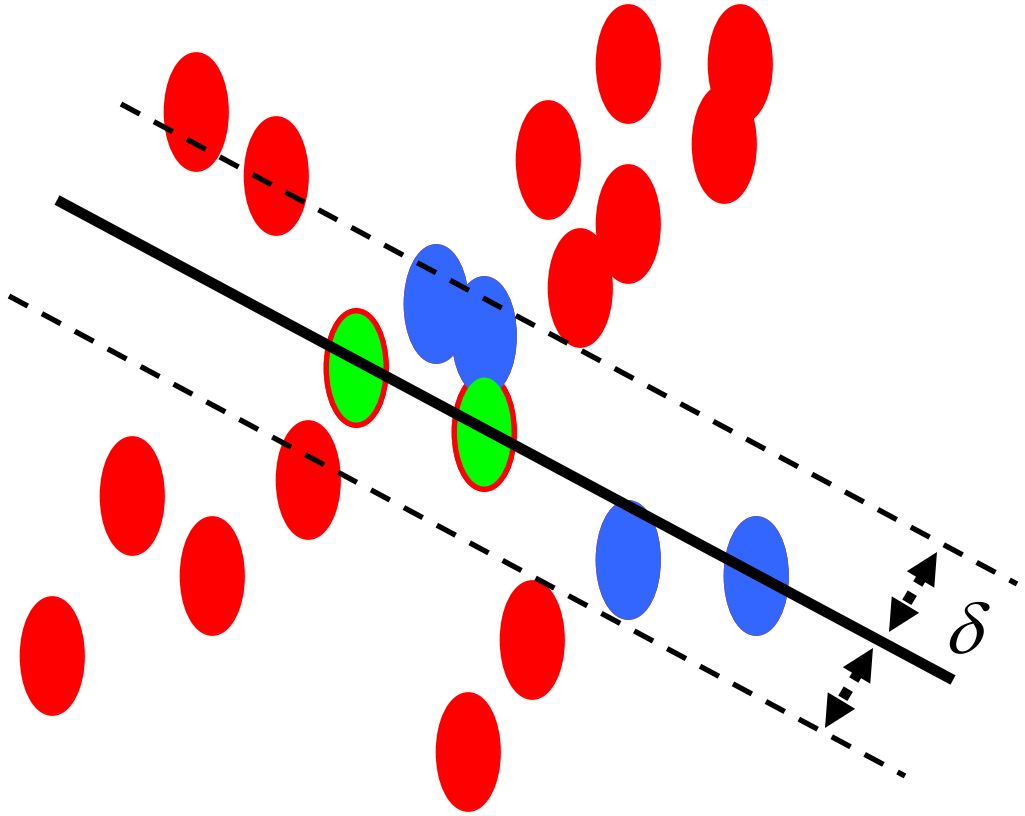
Algorithm:

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



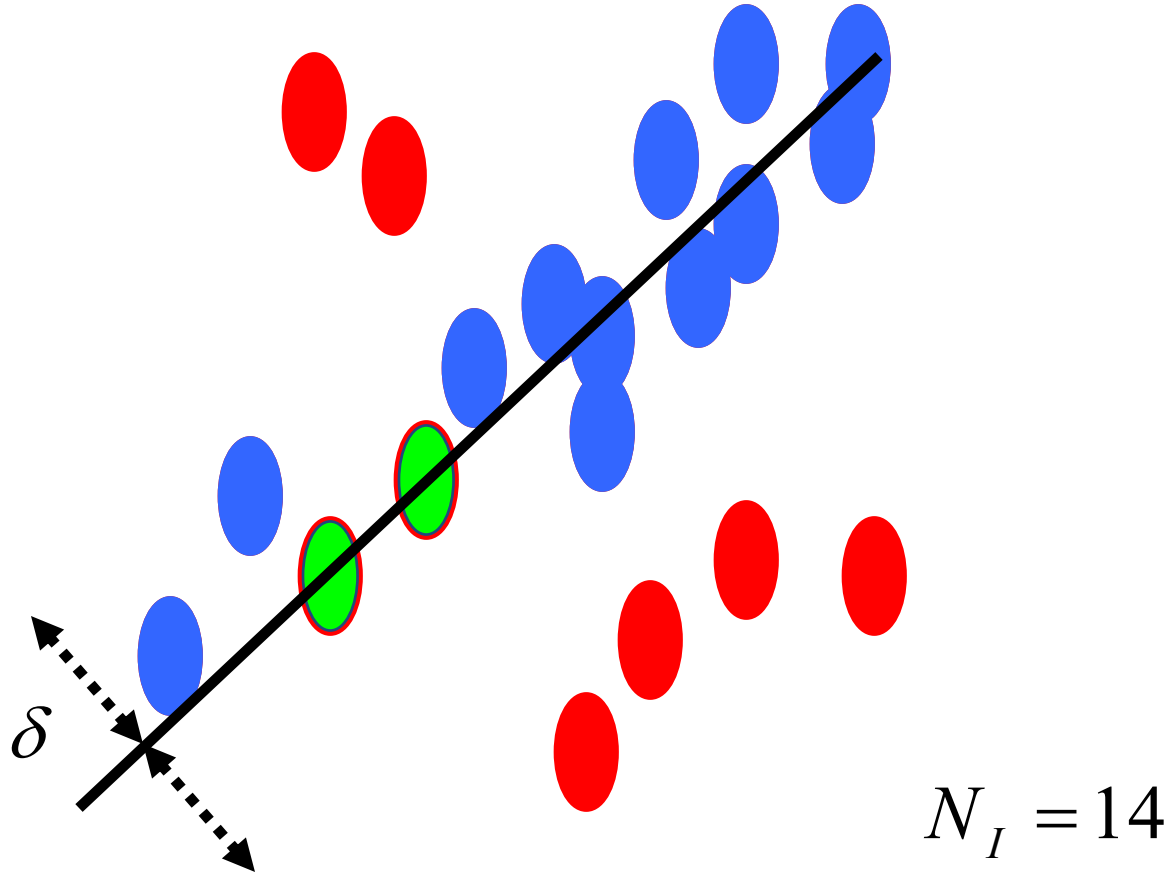
$$N_I = 6$$

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 samples (iterations) *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

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

	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

For p = 0.99

modified from M. Pollefeys

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

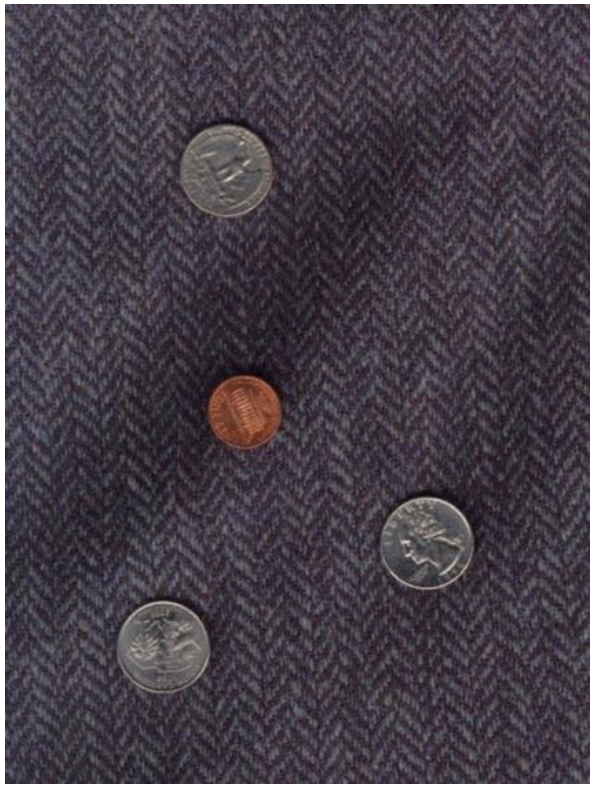
- Computational time grows quickly with fraction of outliers and number of parameters
- Not good for getting multiple fits

Common applications

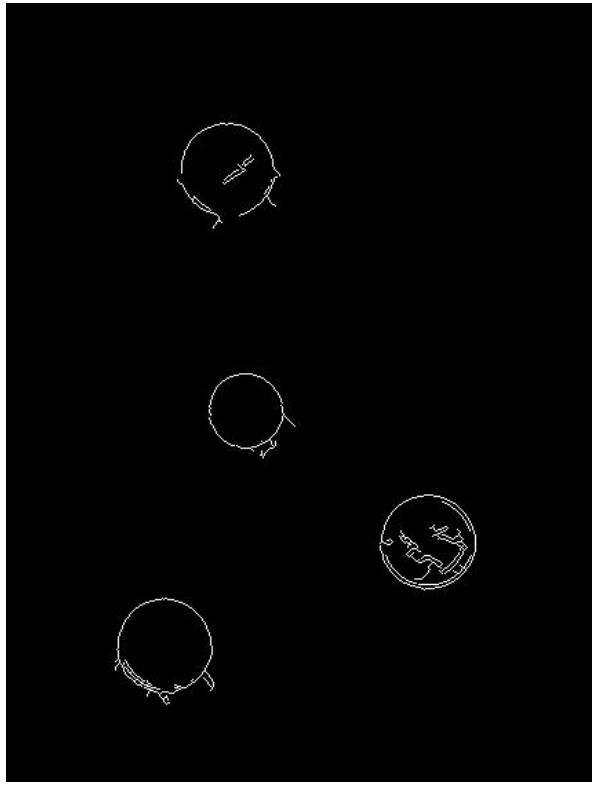
- Computing a homography (e.g., image stitching)
- Estimating fundamental matrix (relating two views)

Can we use RANSAC instead of Hough transform?

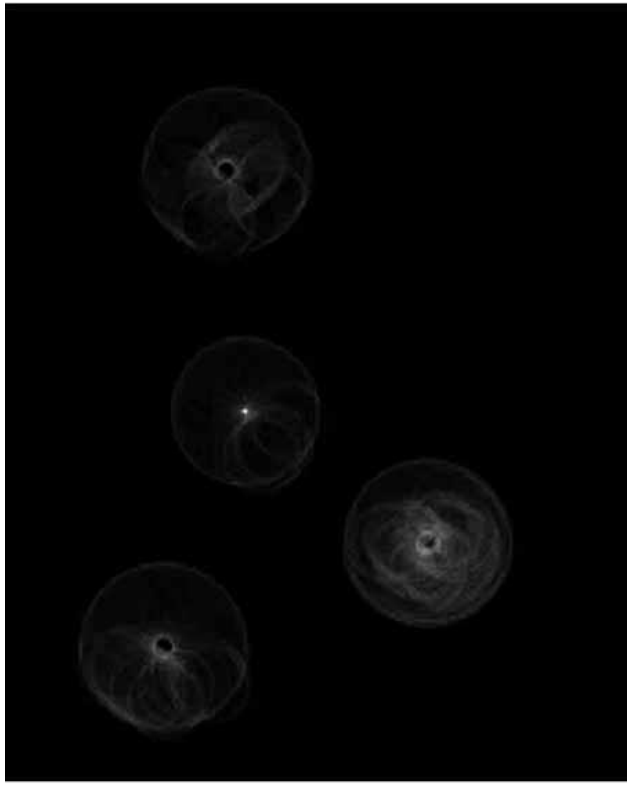
Original



Edges



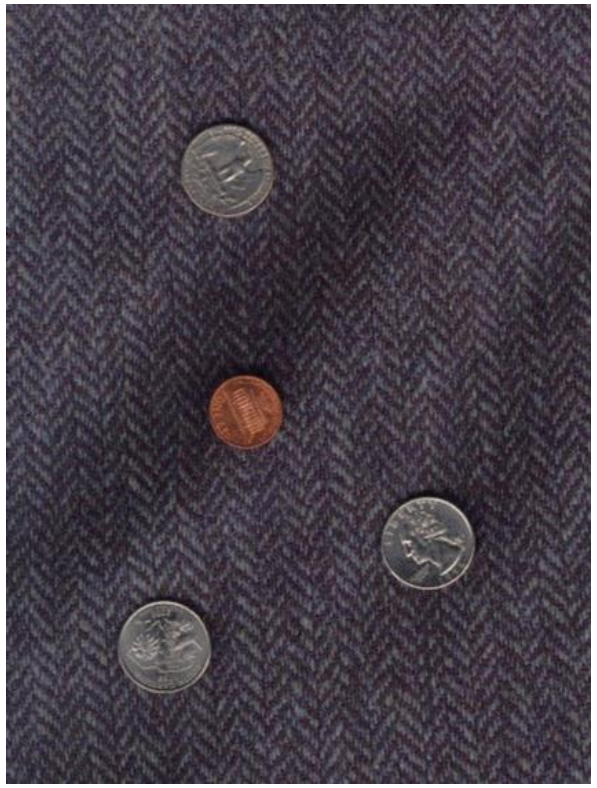
Votes: Penny



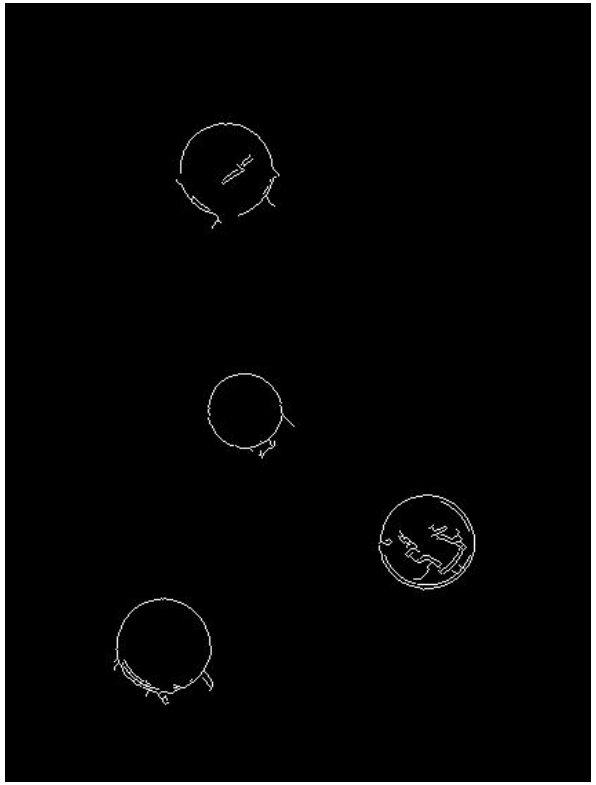
Slide credit: Kristen Grauman

Can we use RANSAC instead of Hough transform?

Original



Edges



Let's find circles of any radius from 6 to 55 pixels

Let's assume that for a particular coin, 10% of the overall edge pixels are "inliers" (on the perimeter of that coin)

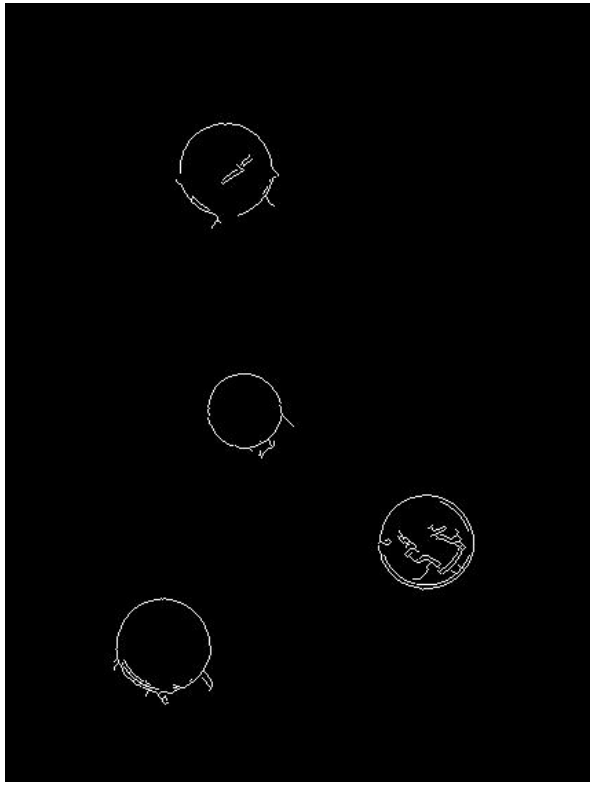
Recall this equation to estimate the number of RANSAC iterations needed, N

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

s = number of samples needed to fit a model
p = desired probability of finding an outlier free solution
e = proportion of outliers

Can we use RANSAC instead of Hough transform?

Edges



Let's find circles of any radius from 6 to 55 pixels

Let's assume that for a particular coin, 10% of the overall edge pixels are "inliers" (on the perimeter of that coin)

Recall this equation to estimate the number of RANSAC iterations needed, N

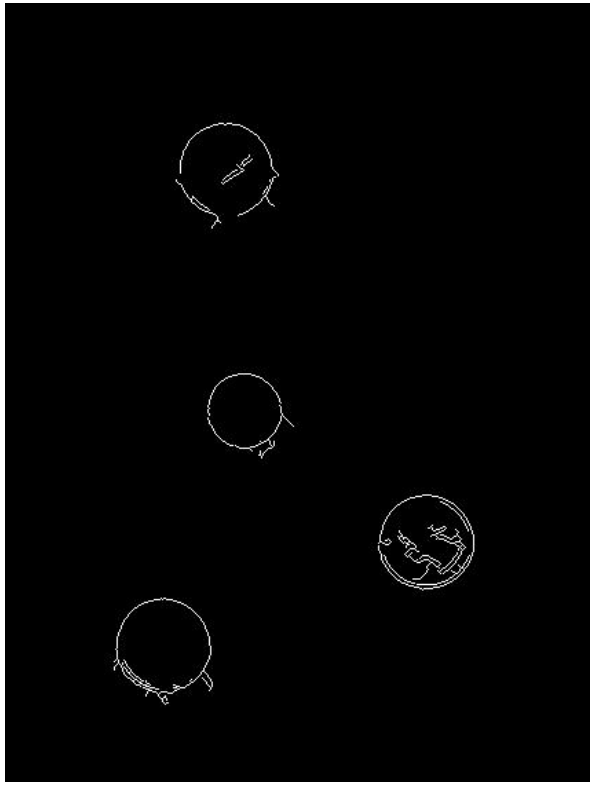
$$N = \log(1-p) / \log\left(1-(1-e)^s\right)$$

s = number of samples needed to fit a model
p = desired probability of finding an outlier free solution
e = proportion of outliers

s = 3
p = .99
e = .9

Can we use RANSAC instead of Hough transform?

Edges



Let's find circles of any radius from 6 to 55 pixels

Let's assume that for a particular coin, 10% of the overall edge pixels are "inliers" (on the perimeter of that coin)

Recall this equation to estimate the number of RANSAC iterations needed, N

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

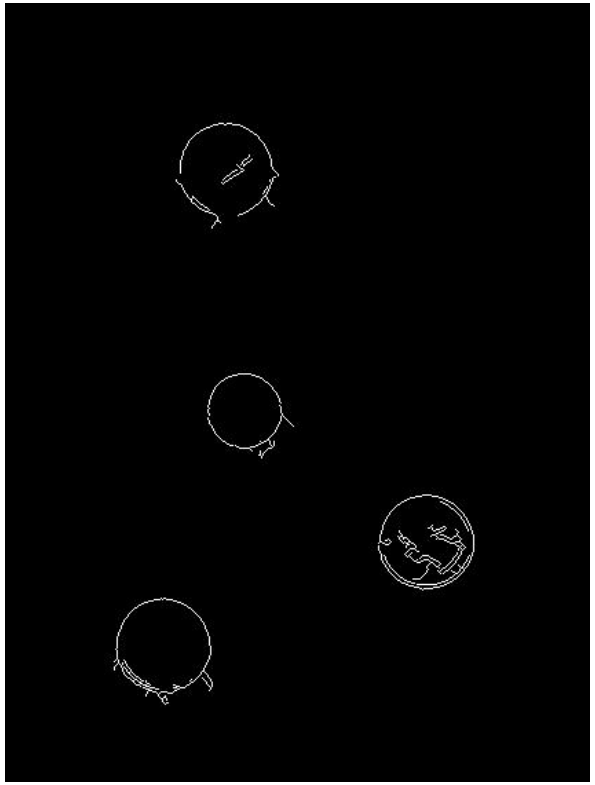
s = number of samples needed to fit a model
p = desired probability of finding an outlier free solution
e = proportion of outliers

s = 3
p = .99
e = .9

N = log(1 - .99) /
log(1 - .001)

Can we use RANSAC instead of Hough transform?

Edges



Let's find circles of any radius from 6 to 55 pixels

Let's assume that for a particular coin, 10% of the overall edge pixels are "inliers" (on the perimeter of that coin)

Recall this equation to estimate the number of RANSAC iterations needed, N

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

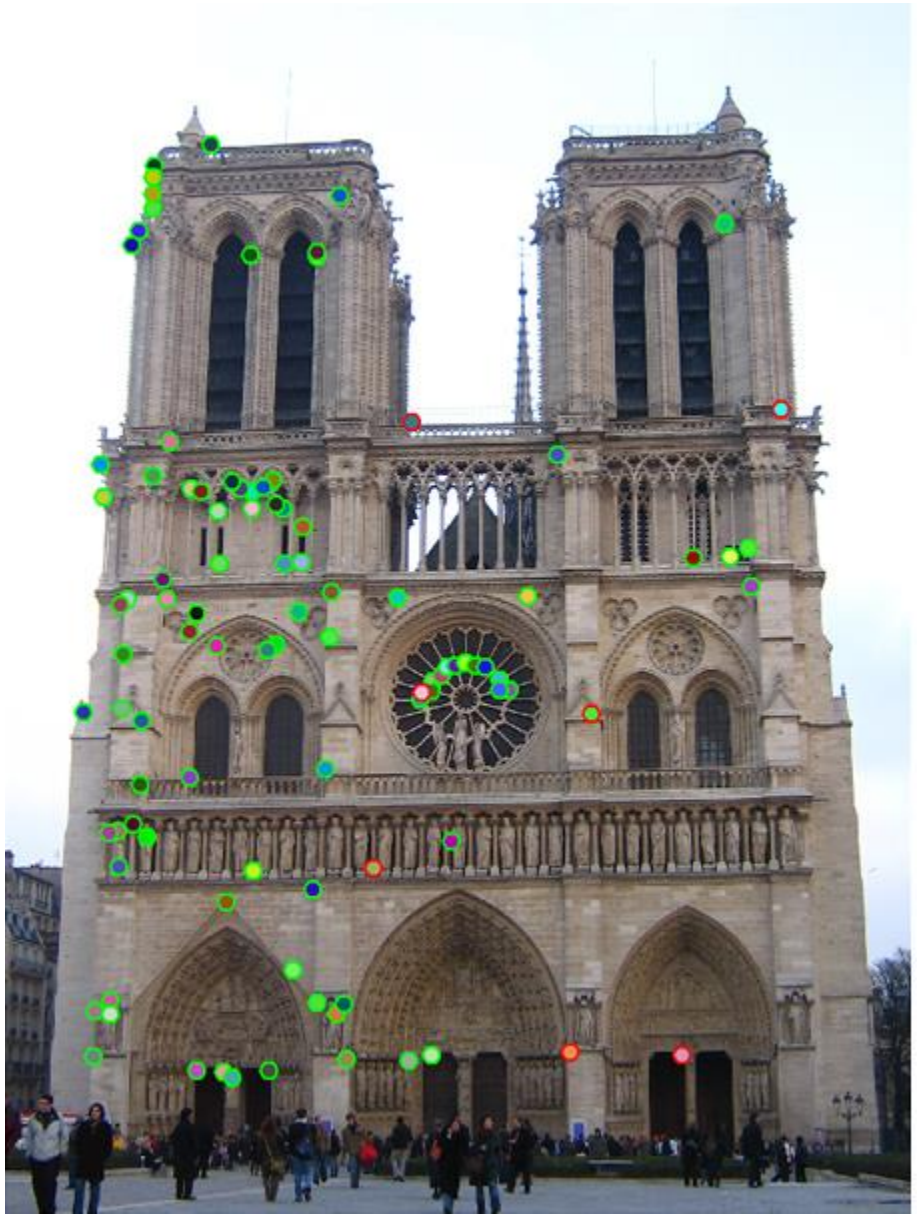
s = number of samples needed to fit a model
p = desired probability of finding an outlier free solution
e = proportion of outliers

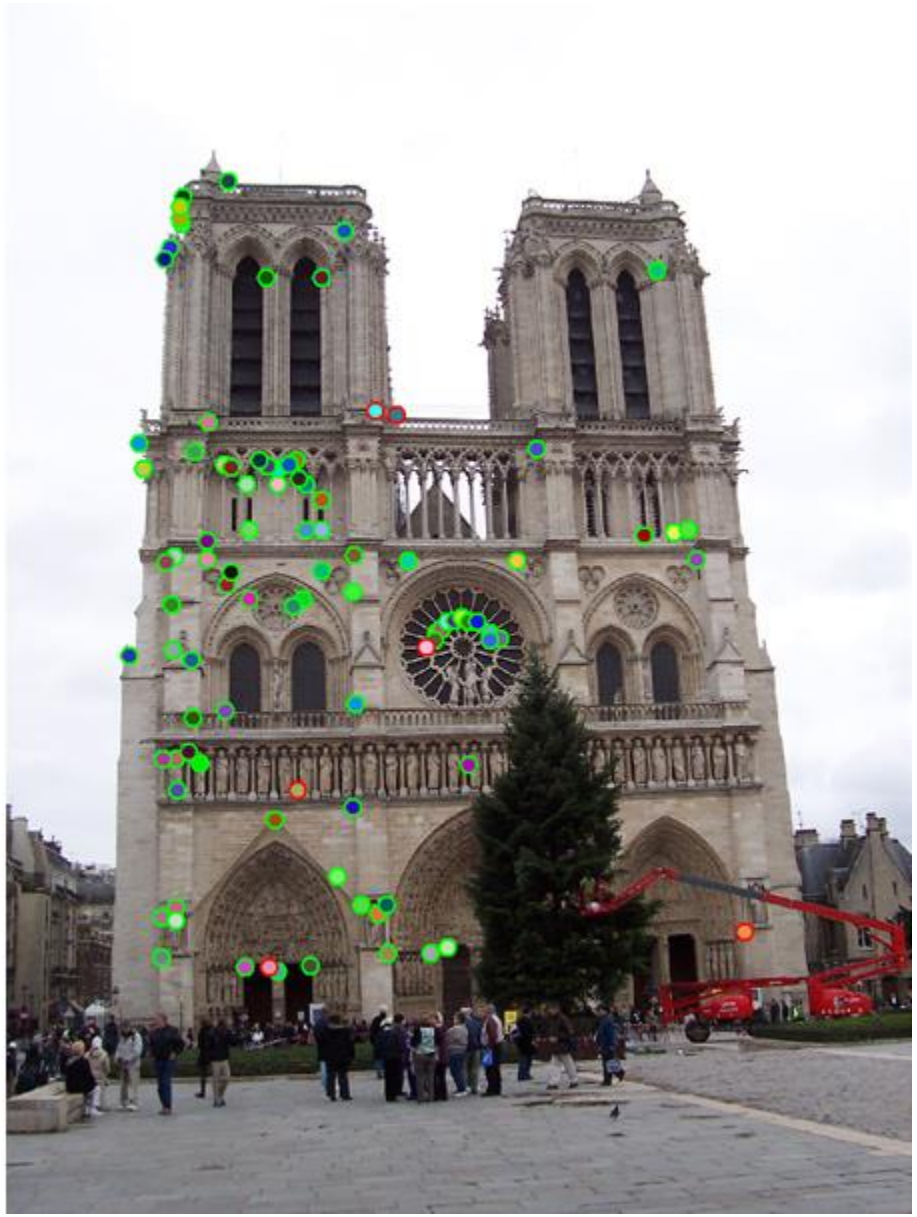
s = 3
p = .99
e = .9

N = log(1 - .99) / log(1 - .001)

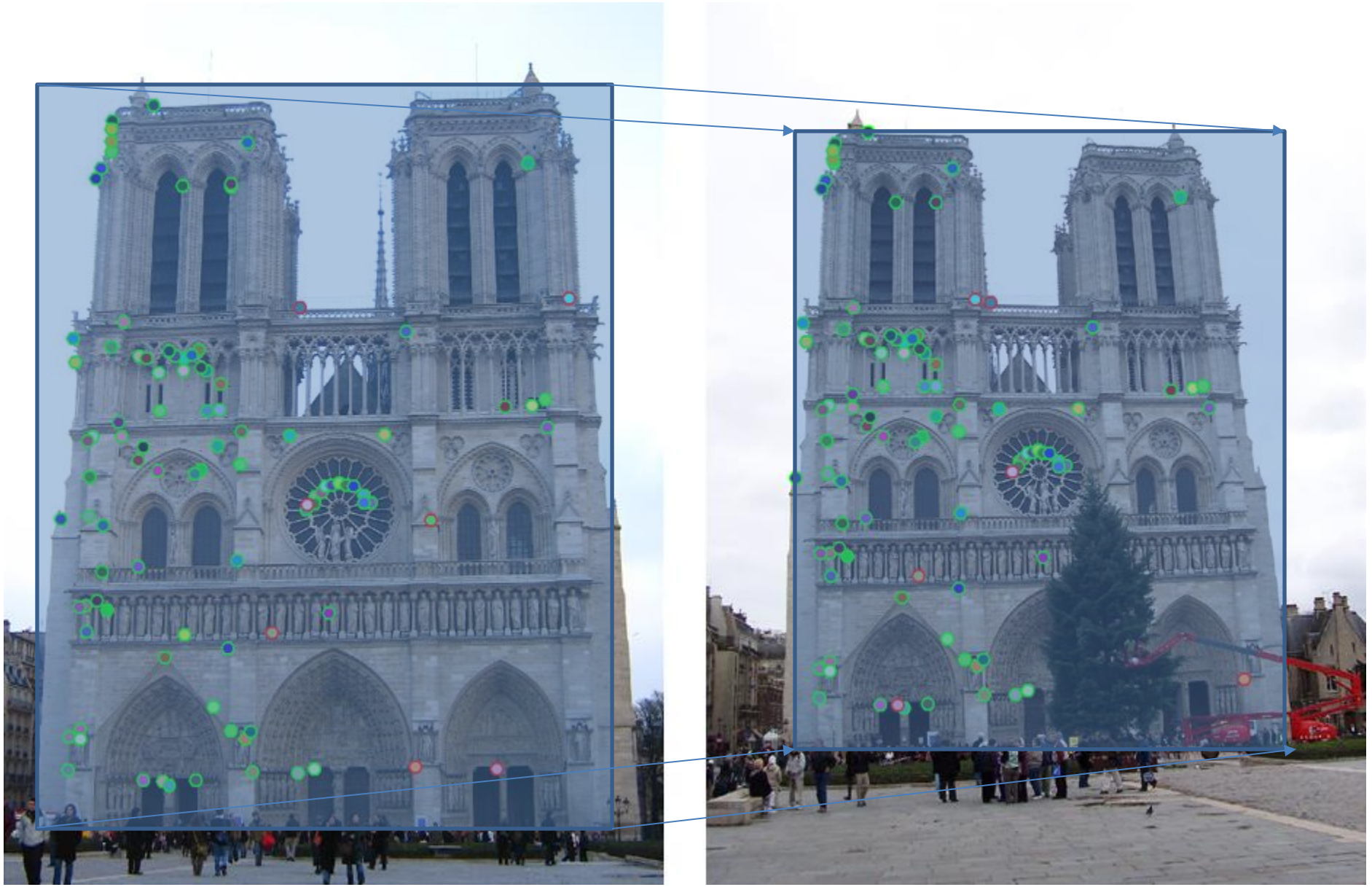
N = 4,602

How do we fit the best alignment?

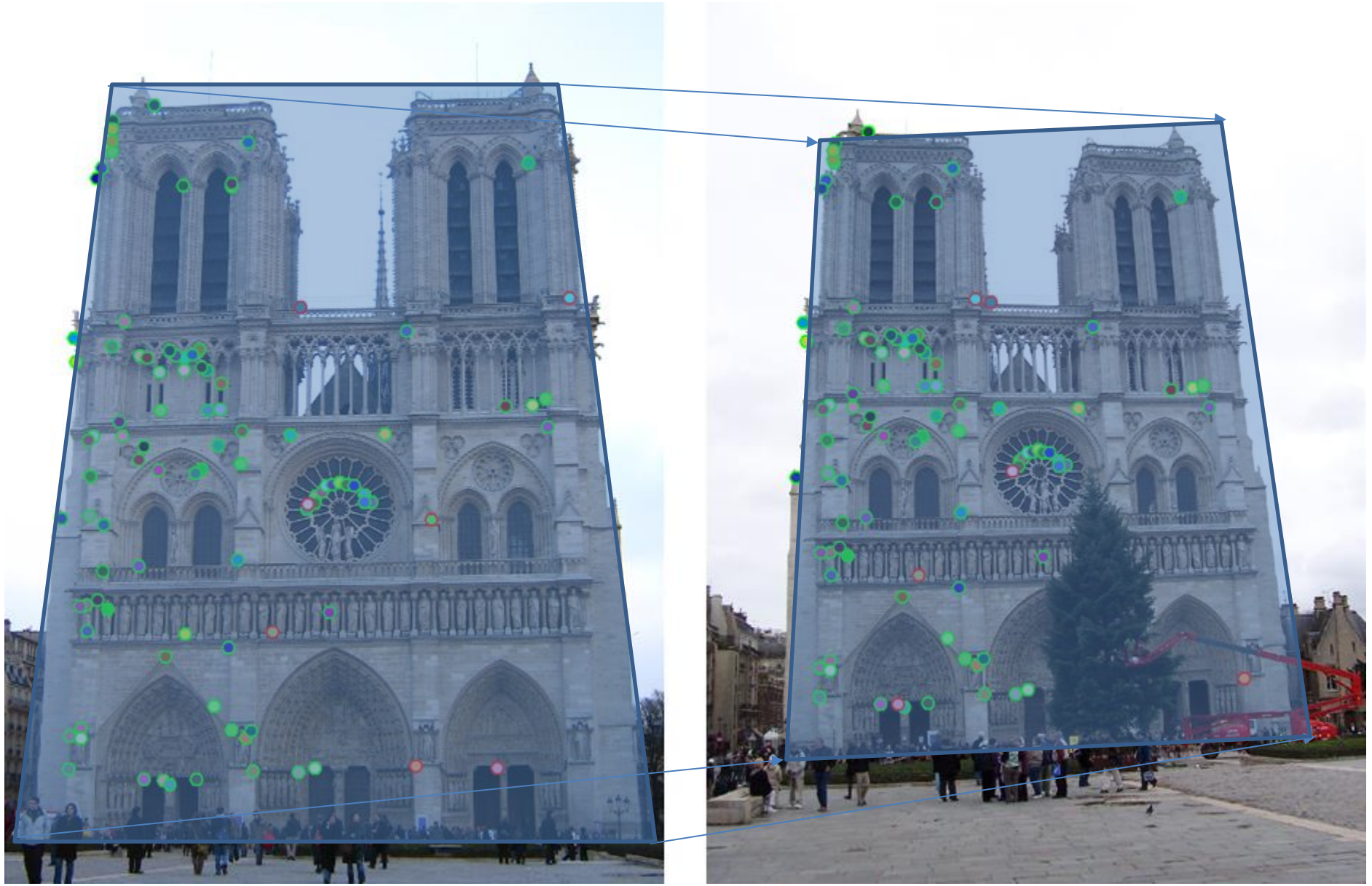




How do we fit the best alignment?



How do we fit the best alignment?



Part 6: Homography with RANSAC

For two photographs of a scene, it's unlikely that you'd have perfect point correspondence with which to solve for the homography matrix. So, next you are going to compute the homography with point correspondences computed using SIFT. As discussed in class, a direct least-squares solution alone is not appropriate in this scenario due to the presence of multiple outliers. In order to estimate the homography from this noisy data, you'll need to use RANSAC. This technique is especially powerful for aligning images of planar surfaces, like building facades or the ground plane.

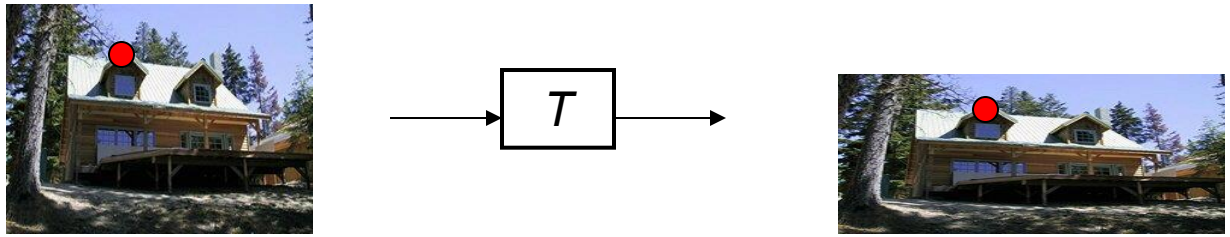
You'll use these putative point correspondences and RANSAC to find the "best" homography matrix. You will iteratively choose a minimal set of point correspondences (4 for a homography), solve for the homography matrix, and then count the number of inliers. Inliers in this context will be point correspondences that "agree" with the estimated homography. In order to count how many inliers a homography has, you'll need a distance metric. For a homography, this is typically the reprojection error: the geometric distance between a point in the second image (x') and the location predicted by mapping its corresponding point from the first image (Hx). You'll need to pick a threshold between inliers and outliers; your results are very sensitive to this threshold, so explore a range of values. You don't want to be too permissive about what you consider an inlier, nor do you want to be too conservative. Return the homography with the most inliers.

Recall from lecture the expected number of iterations of RANSAC to find the "right" solution in the presence of outliers. For example, if half of your input correspondences are wrong, then you have a $(0.5)^4 = 6.25\%$

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 and parametric?

- Global: Is the same for any point p
- Parametric: can be described by just a few numbers

We're going to focus on *linear* transformations. We can represent T as a matrix multiplication

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

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

Common transformations

original

Transformed

translation

rotation

aspect

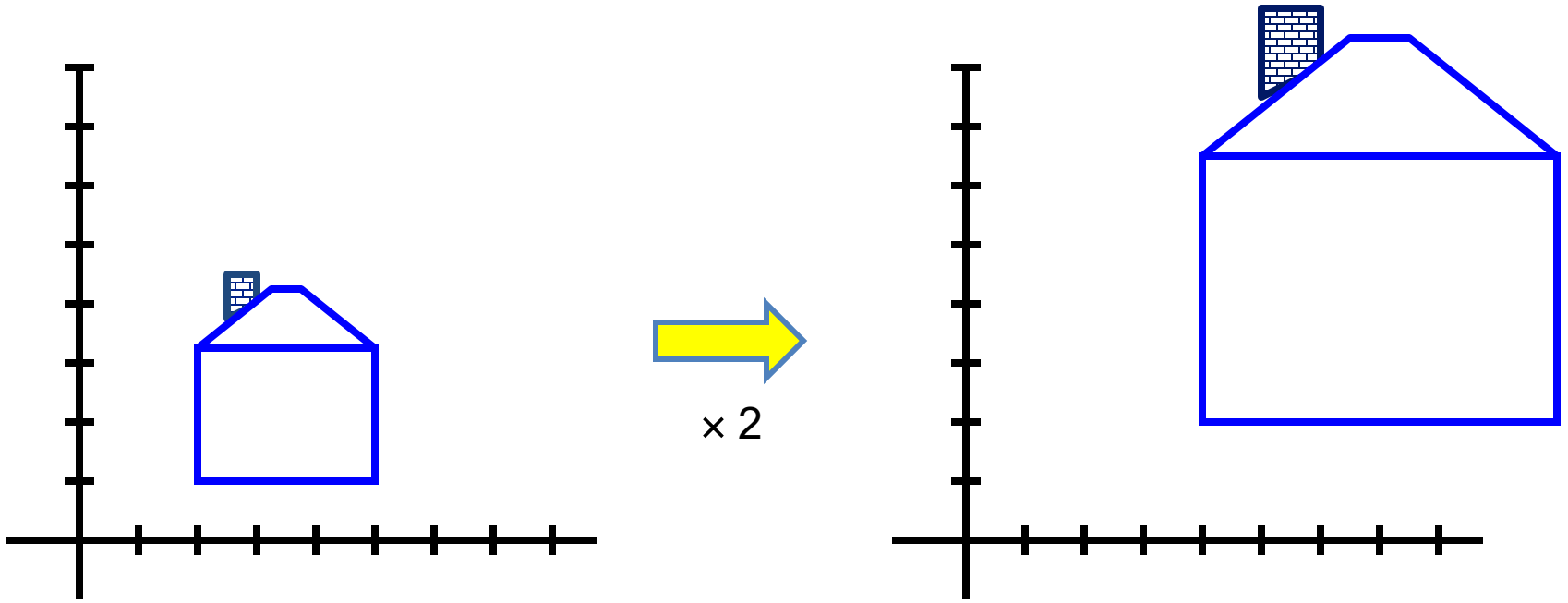
affine

perspective

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

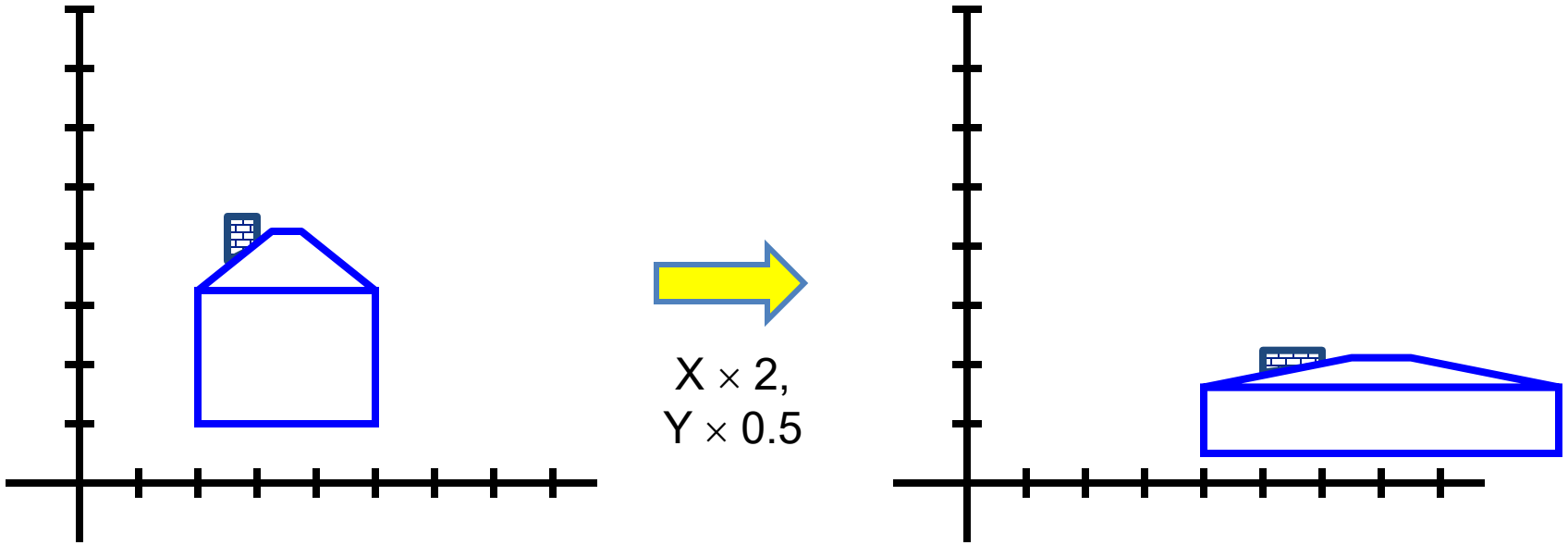
Scaling

- *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:

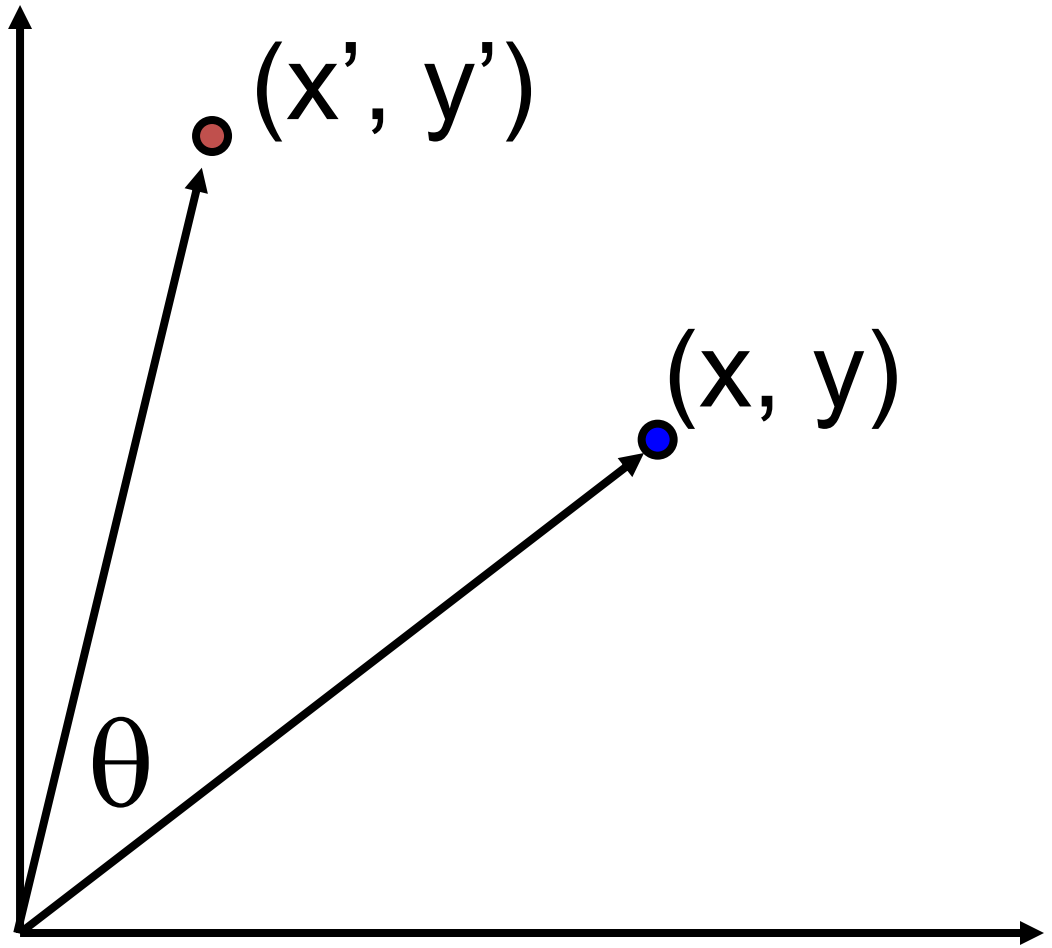
$$x' = ax$$

$$y' = by$$

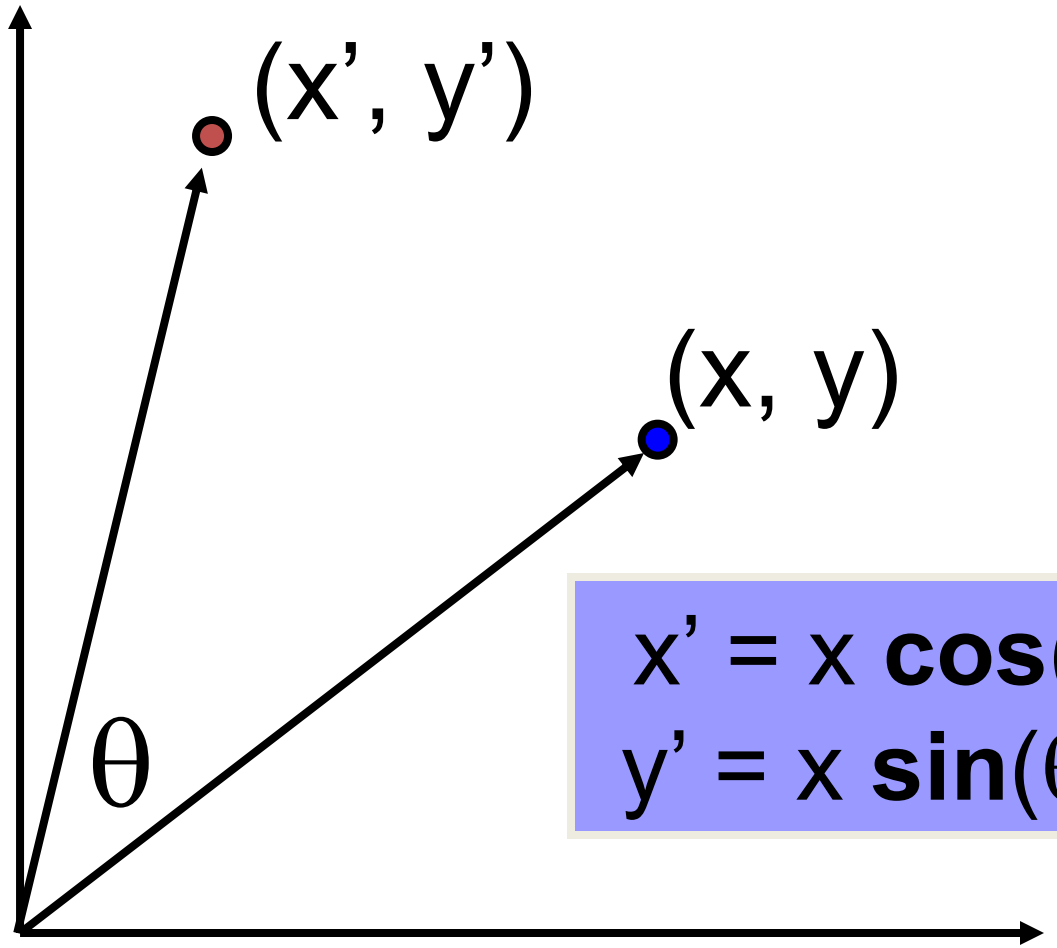
- Or, in matrix form:

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

2-D Rotation (around the origin)



2-D Rotation (around the origin)



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

2-D Rotation

This is easy to capture in matrix form:

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

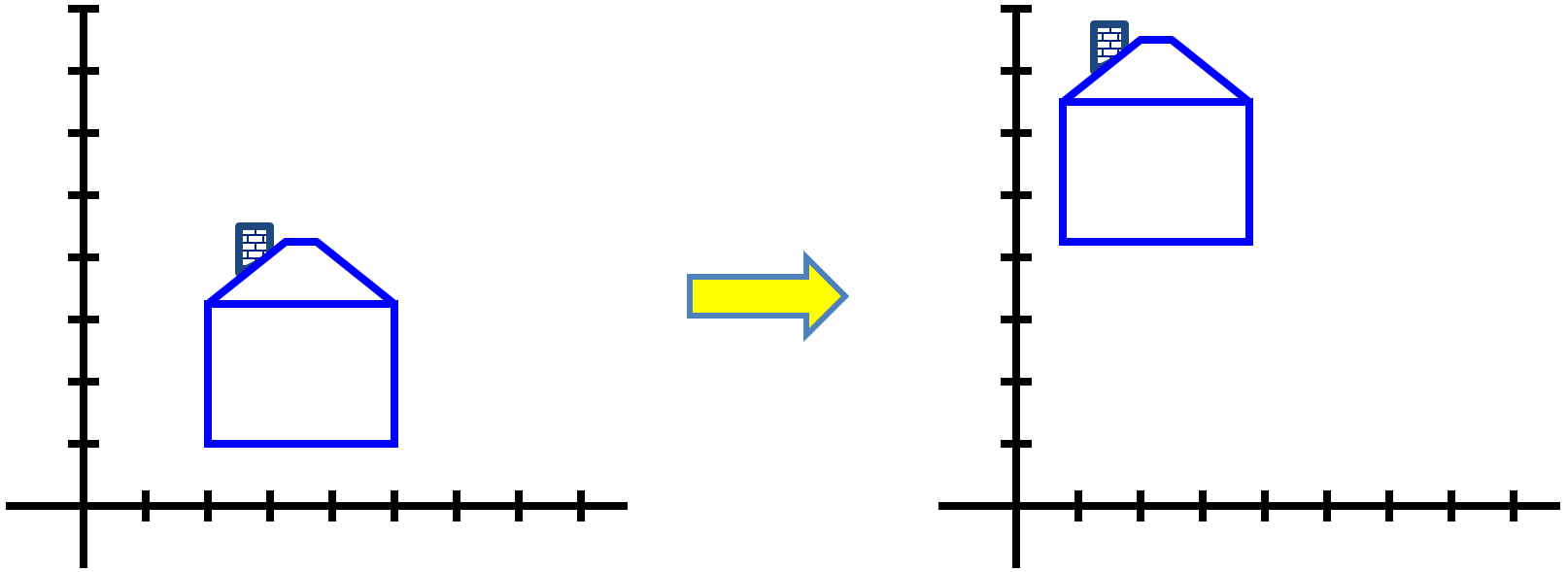
Even though sin(θ) and cos(θ) are nonlinear functions of θ,

- ***For a particular*** θ, ***x' is a linear combination of x and y***
- ***For a particular*** θ, ***y' is a linear combination of x and y***

What is the inverse transformation?

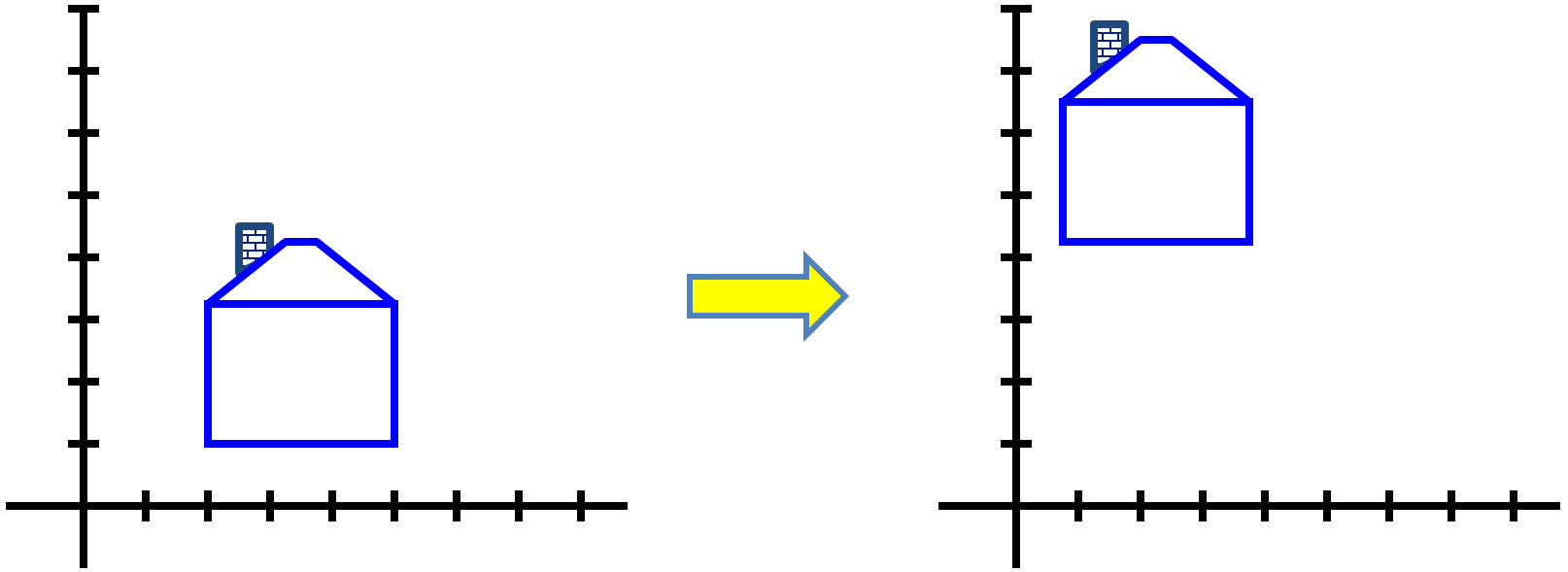
- Rotation by –θ
- For rotation matrices $\mathbf{R}^{-1} = \mathbf{R}^{T}$

How about translation?



How about translation?

$$x' = x + t_x$$
$$y' = y + t_y$$



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

From the original SIFT paper

Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

7.4 Solution for affine parameters

The Hough transform is used to identify all clusters with at least 3 entries in a bin. Each such cluster is then subject to a geometric verification procedure in which a least-squares solution is performed for the best affine projection parameters relating the training image to the new image.

An affine transformation correctly accounts for 3D rotation of a planar surface under orthographic projection, but the approximation can be poor for 3D rotation of non-planar objects. A more general solution would be to solve for the fundamental matrix (Luong and Faugeras, 1996; Hartley and Zisserman, 2000). However, a fundamental matrix solution requires at least 7 point matches as compared to only 3 for the affine solution and in practice requires even more matches for good stability. We would like to perform recognition with as few as 3 feature matches, so the affine solution provides a better starting point and we can account for errors in the affine approximation by allowing for large residual errors. If we imagine placing a sphere around an object, then rotation of the sphere by 30 degrees will move no point within the sphere by more than 0.25 times the projected diameter of the sphere. For the examples of typical 3D objects used in this paper, an affine solution works well given that we allow residual errors up to 0.25 times the maximum projected dimension of the object. A more general approach is given in (Brown and Lowe, 2002), in which the initial solution is based on a similarity transform, which then progresses to solution for the fundamental matrix in those cases in which a sufficient number of matches are found.

The affine transformation of a model point $[x\ y]^T$ to an image point $[u\ v]^T$ can be written as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

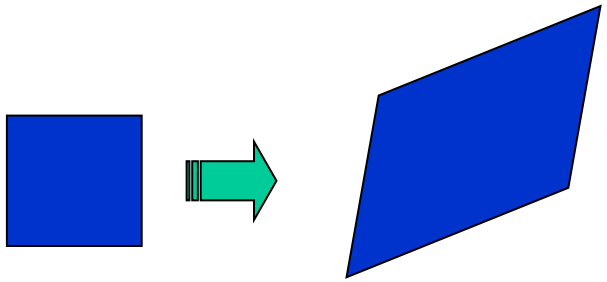
2D Affine Transformations

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

Affine transformations are combinations of …

- Linear transformations, and
- Translations

Parallel lines remain parallel



Slide credit: Kristen Grauman

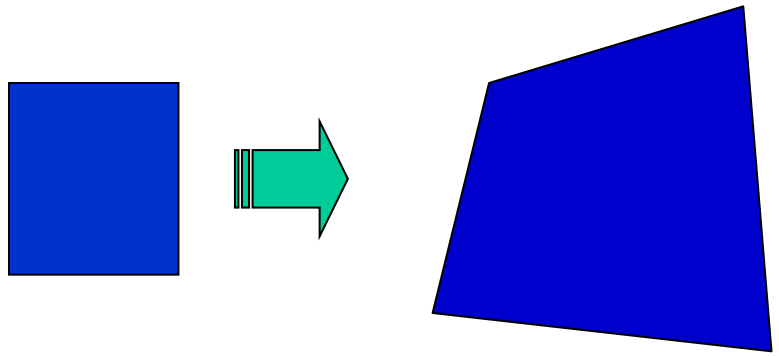
Projective Transformations (or Homographies)

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

Projective transformations:

- Affine transformations, and
- Projective warps

Parallel lines do not necessarily remain parallel



Slide credit: Kristen Grauman

Textbook 2.1.1

Projective. This transformation, also known as a *perspective transform* or *homography*, operates on homogeneous coordinates,

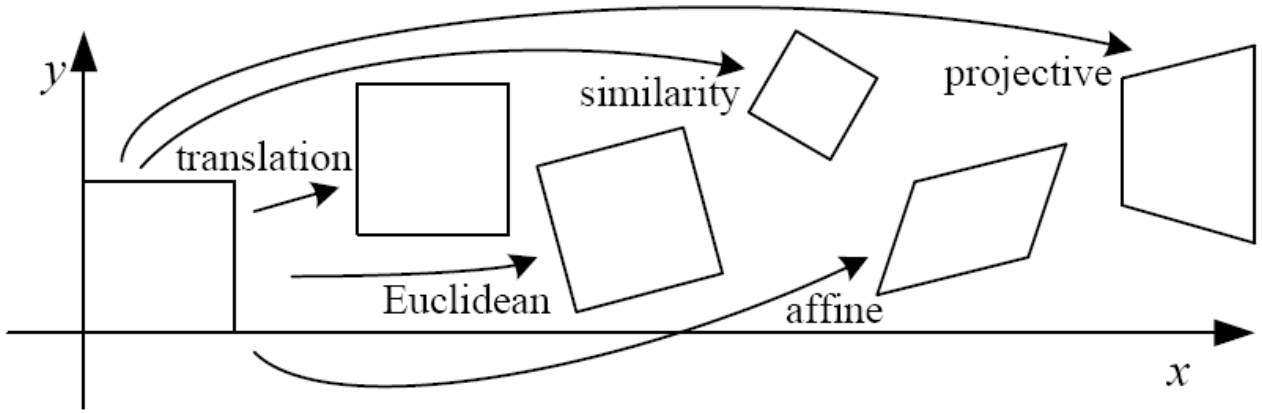
$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}, \tag{2.20}$$

where $\tilde{\mathbf{H}}$ is an arbitrary 3×3 matrix. Note that $\tilde{\mathbf{H}}$ is homogeneous, i.e., it is only defined up to a scale, and that two $\tilde{\mathbf{H}}$ matrices that differ only by scale are equivalent. The resulting homogeneous coordinate $\tilde{\mathbf{x}}'$ must be normalized in order to obtain an inhomogeneous result $\mathbf{x}$, i.e.,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}. \tag{2.21}$$

Perspective transformations preserve straight lines (i.e., they remain straight after the transformation).

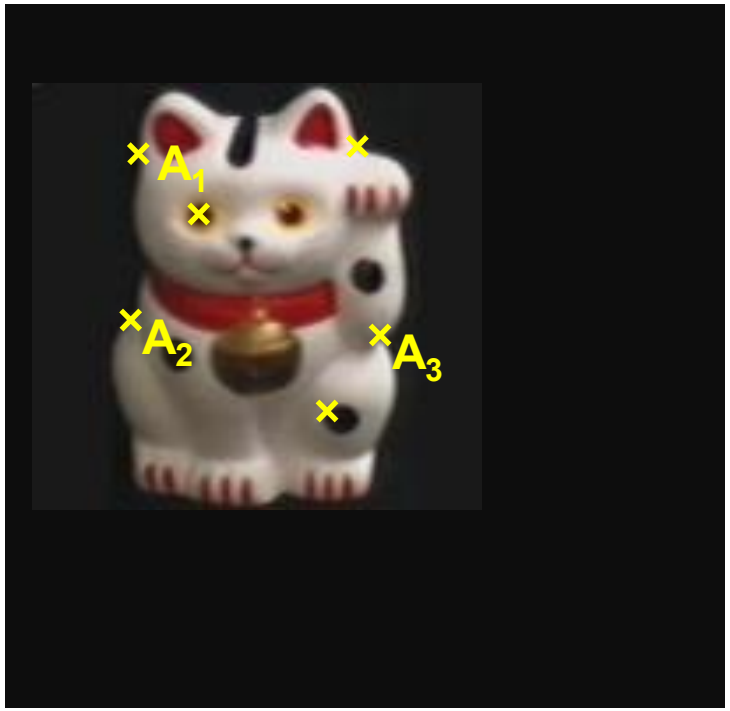
2D image transformations (reference table)

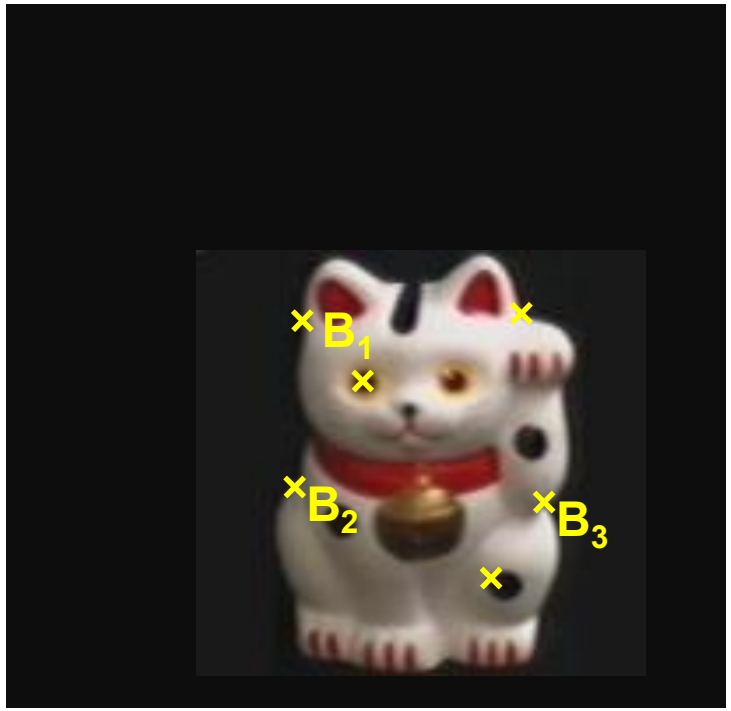


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R}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

Szeliski 2.1

Example: solving for translation

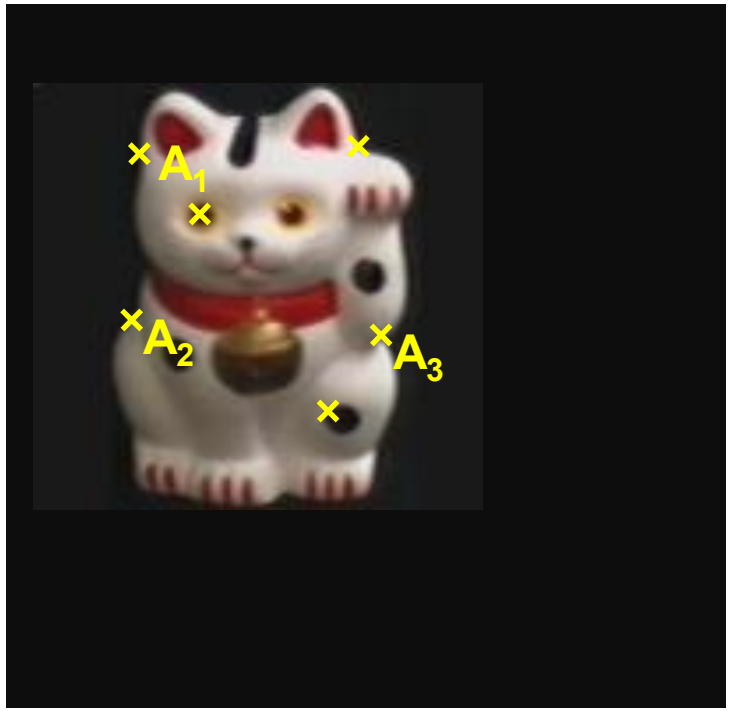




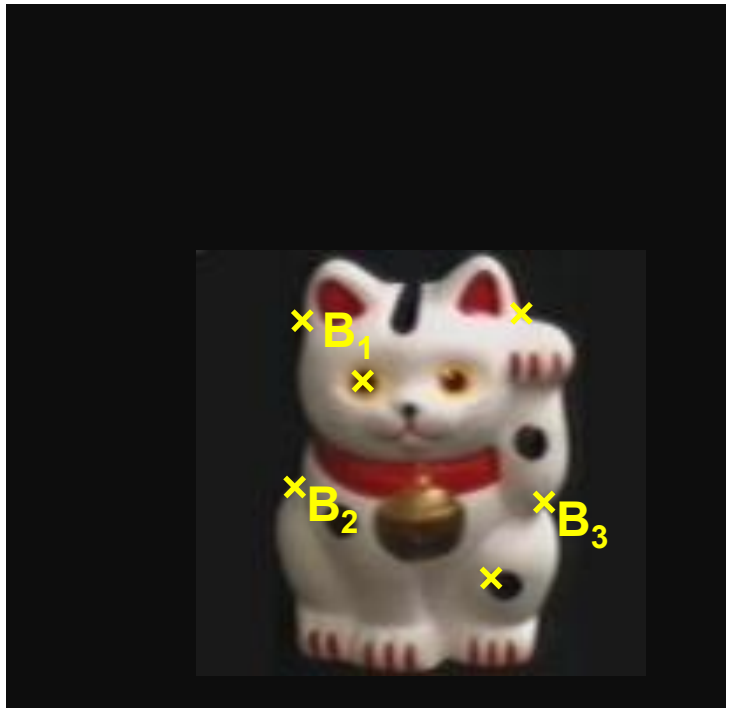
Given matched points in {A} and {B}, estimate the translation of the object

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

Example: solving for translation



(t_x, t_y)



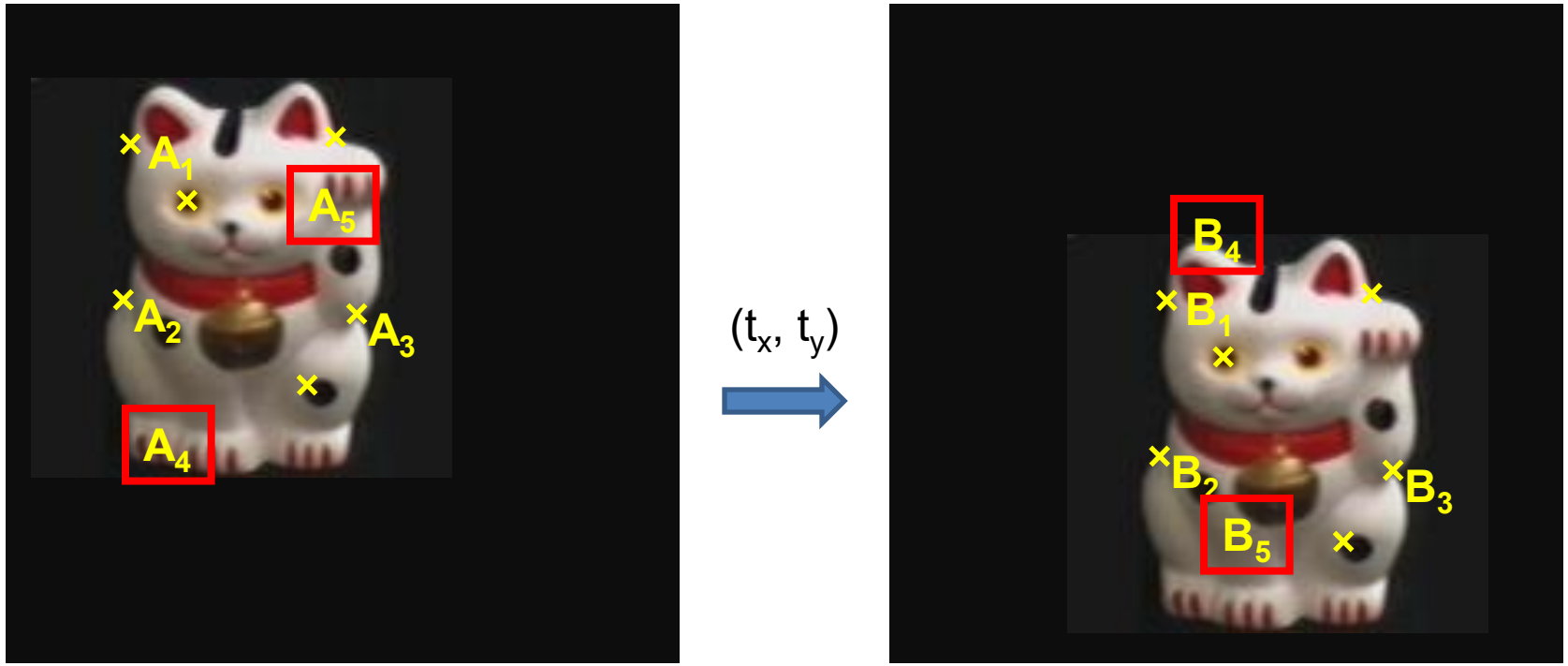
Least squares solution

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



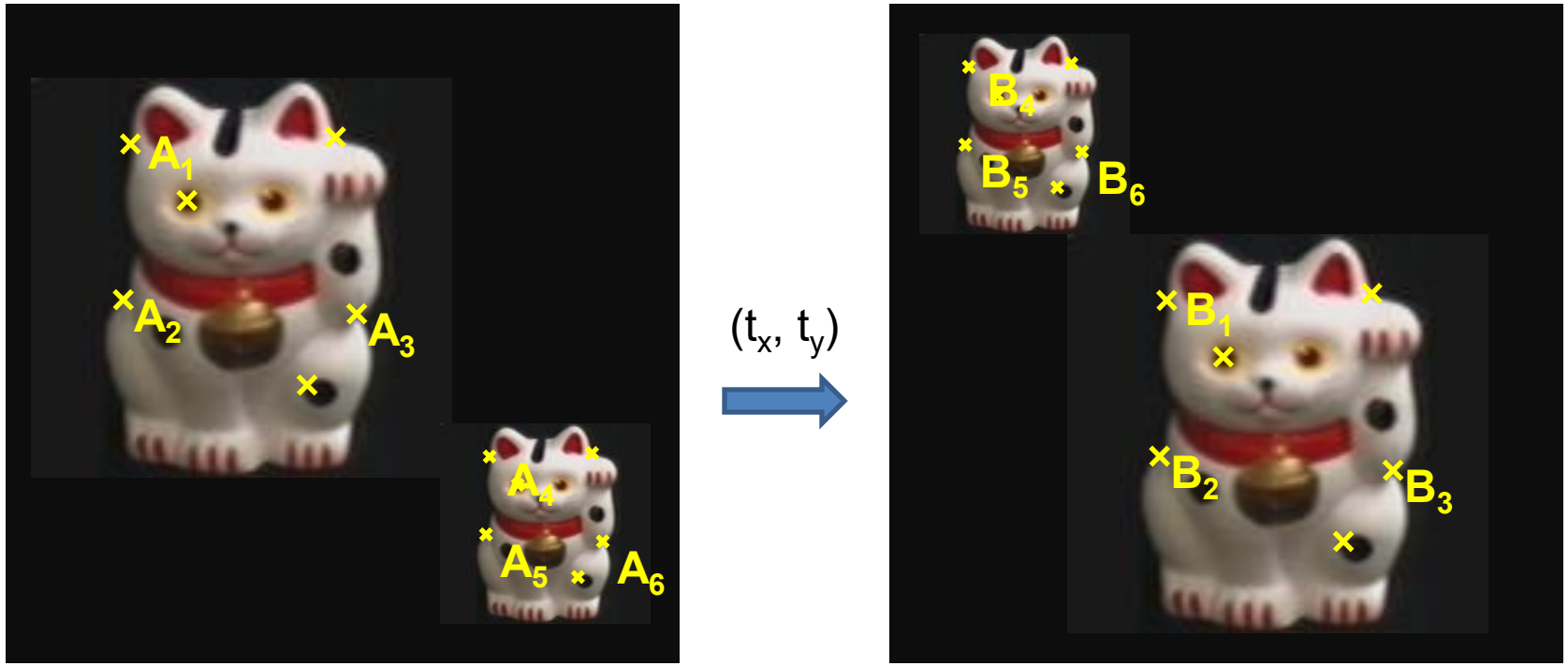
Problem: outliers

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



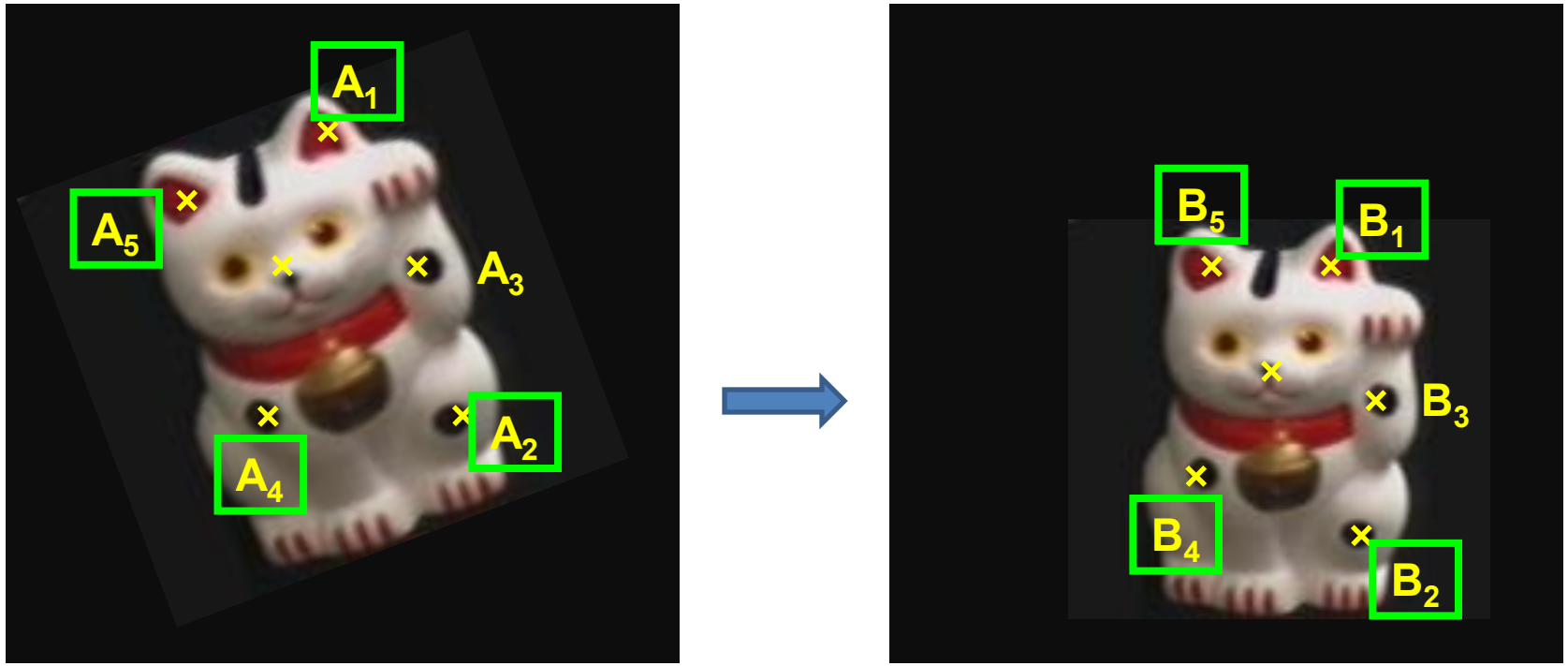
Problem: outliers, multiple objects, and/or many-to-one matches

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

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

But what about solving for a homography?



RANSAC solution

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

Setup. Given putative correspondences $A_i \leftrightarrow B_i$ with $A_i = (x_i, y_i)$ in image A and $B_i = (u_i, v_i)$ in image B, let

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{x}' = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

The projective mapping is $\mathbf{x}' \sim H\mathbf{x}$, i.e.

$$u = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \qquad v = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}.$$

Per-correspondence linear equations. Cross-multiplying and rearranging yields, for each match $(x_i, y_i) \mapsto (u_i, v_i)$,

$$x_i h_{11} + y_i h_{12} + h_{13} - u_i x_i h_{31} - u_i y_i h_{32} - u_i h_{33} = 0, \tag{1}$$
$$x_i h_{21} + y_i h_{22} + h_{23} - v_i x_i h_{31} - v_i y_i h_{32} - v_i h_{33} = 0. \tag{2}$$

Minimal 4-point linear solve (fixing scale). With exactly four non-degenerate correspondences ($i = 1, \ldots, 4$), fix the global scale by setting $h_{33} = 1$ and solve the 8×8 system for $\mathbf{h} = [h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}]^\top$:

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -v_1x_1 & -v_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -v_2x_2 & -v_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -u_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -v_3x_3 & -v_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -u_4x_4 & -u_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -v_4x_4 & -v_4y_4 \end{bmatrix}}_{A_{8\times 8}} \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}}_{\mathbf{h}} = \underbrace{\begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix}}_{\mathbf{b}}.$$

Recover H by appending $h_{33} = 1$.

Part 6: Homography with RANSAC

Your task is to implement the RANSAC algorithm to robustly align the Notre Dame image pair, which is challenging due to many incorrect initial feature matches. Since the cathedral facade is a planar surface, the correct geometric model is a Homography (H), which accurately accounts for perspective distortion.

```python
from vision.part6_ransac import (
    calculate_num_ransac_iterations,
    ransac_homography,
)
from vision.utils import single2im
```

[ ] Python

```python
# Load the data

# Notre Dame
img_a_path = "data/Notre_Dame/921919841_a30df938f2_o.jpg"
img_b_path = "data/Notre_Dame/4191453057_c86028ce1f_o.jpg"

try:
    img_a_bgr = cv2.imread(img_a_path)
    img_b_bgr = cv2.imread(img_b_path)
    if img_a_bgr is None or img_b_bgr is None: raise FileNotFoundError
    img_a = cv2.cvtColor(img_a_bgr, cv2.COLOR_BGR2RGB)
    img_b = cv2.cvtColor(img_b_bgr, cv2.COLOR_BGR2RGB)
except FileNotFoundError:
    print("Error: One or both images could not be loaded. Check paths.")
```

[ ] Python

```python
from tests.test_part6_ransac import (
    test_calculate_num_ransac_iterations,
    test_ransac_homography,
)

print(
    "test_calculate_num_ransac_iterations():",
    verify(test_calculate_num_ransac_iterations),
)
print("test_ransac_homography():", verify(test_ransac_homography))
```

part6_ransac.py 2

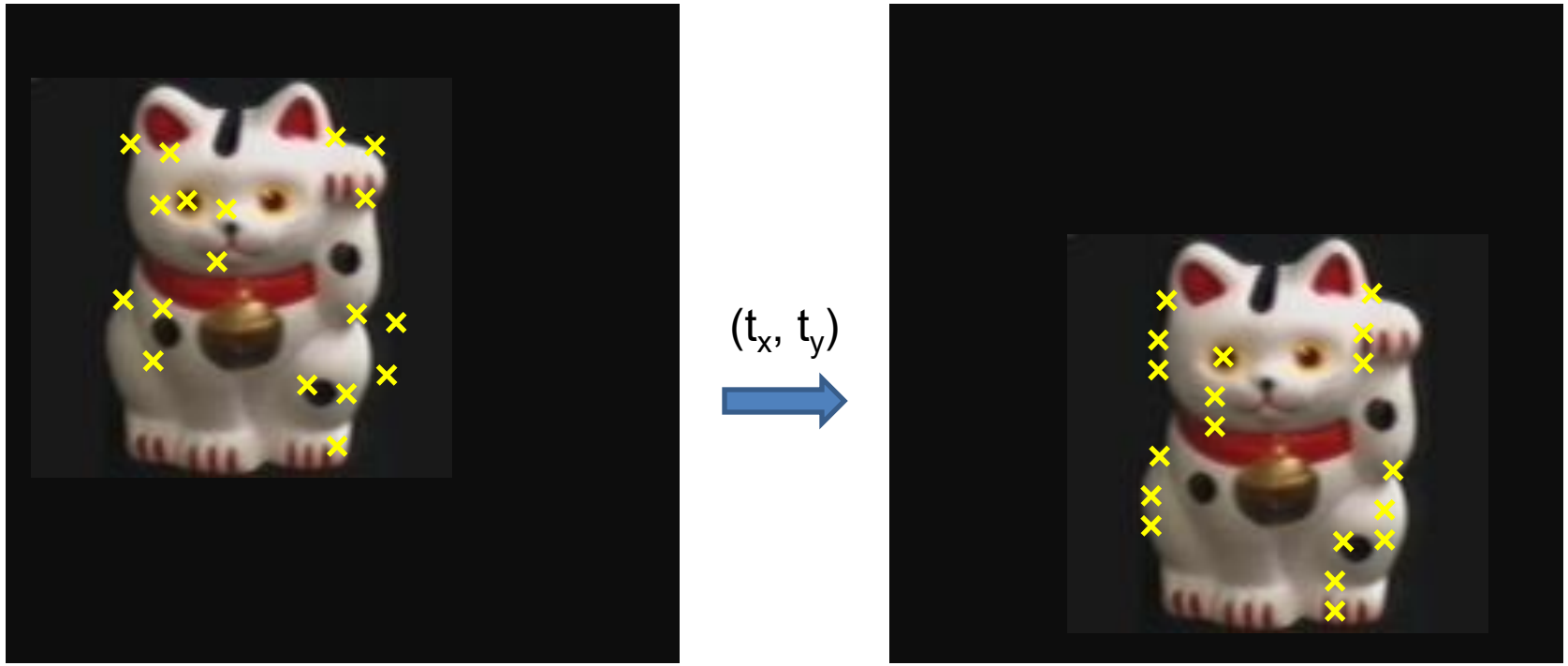
C: > Users > james > Desktop > compvision > project-2-student > src > vision > part6_ransac.py > ...

```python
def ransac_homography(
    points_a: np.ndarray, points_b: np.ndarray
) -> (np.ndarray, np.ndarray, np.ndarray):
    """
    Uses the RANSAC algorithm to robustly estimate a homography matrix.

    Args:
    -   points_a: A numpy array of shape (N, 2) of points from image A.
    -   points_b: A numpy array of shape (N, 2) of corresponding points from image B.

    Returns:
    -   best_H: The best homography matrix of shape (3, 3).
    -   inliers_a: The subset of points_a that are inliers (M, 2).
    -   inliers_b: The subset of points_b that are inliers (M, 2).
    """
    ############################################################################
    # TODO: YOUR CODE HERE                                                     #
    #                                                                          #
    # HINT: You are allowed to use the `cv2.findHomography` function to        #
    # compute the homography from a sample of points. To compute a direct      #
    # solution without OpenCV's built-in RANSAC, use it like this:             #
    #   H, _ = cv2.findHomography(sample_a, sample_b, 0)                       #
    # The `0` flag ensures it computes a direct least-squares solution.        #
    ############################################################################

    raise NotImplementedError(
        "`ransac_homography` function in "
        + "`part6_ransac.py` needs to be implemented"
    )

    ############################################################################
    #                             END OF YOUR CODE                             #
    ############################################################################

    return best_H, inliers_a, inliers_b

```

Fitting and Alignment: Methods

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

Example: solving for translation

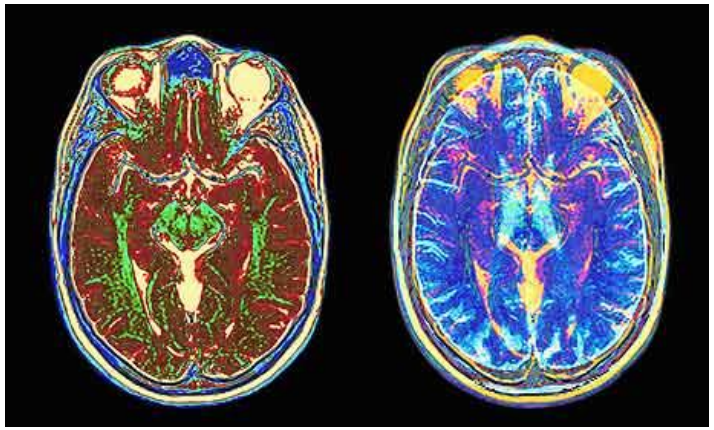


Problem: no initial guesses for correspondence

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

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

- Hough transform and RANSAC not applicable
- Important applications



Medical imaging: align brain scans or contours

Robotics: align point clouds

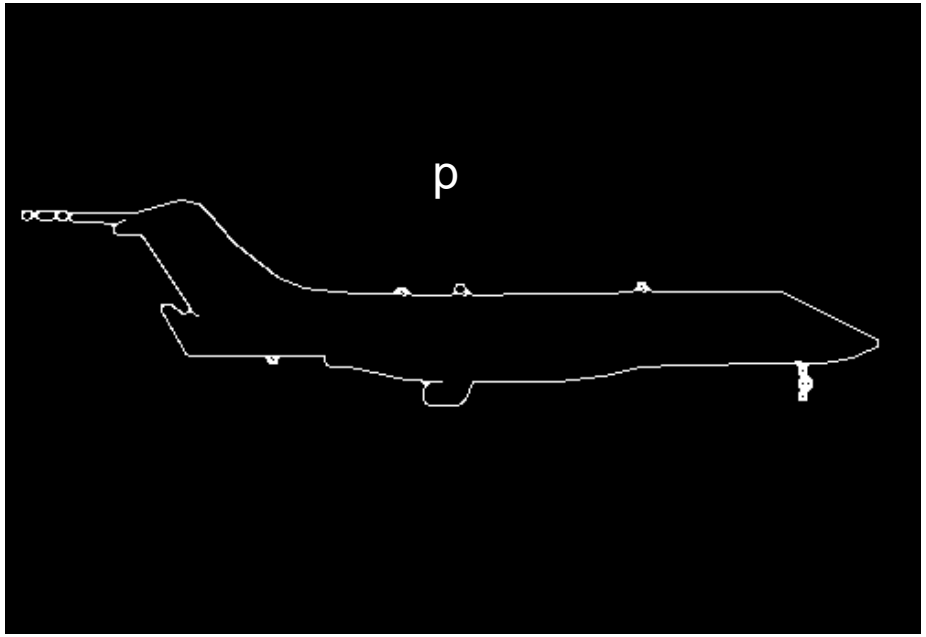
Iterative Closest Points (ICP) Algorithm

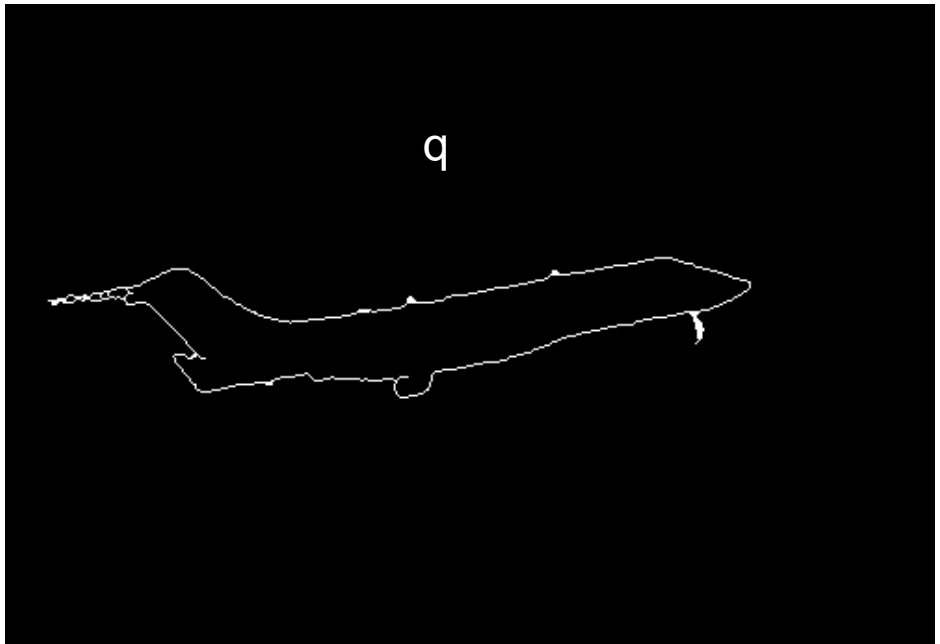
Goal: estimate transform between two dense sets of points

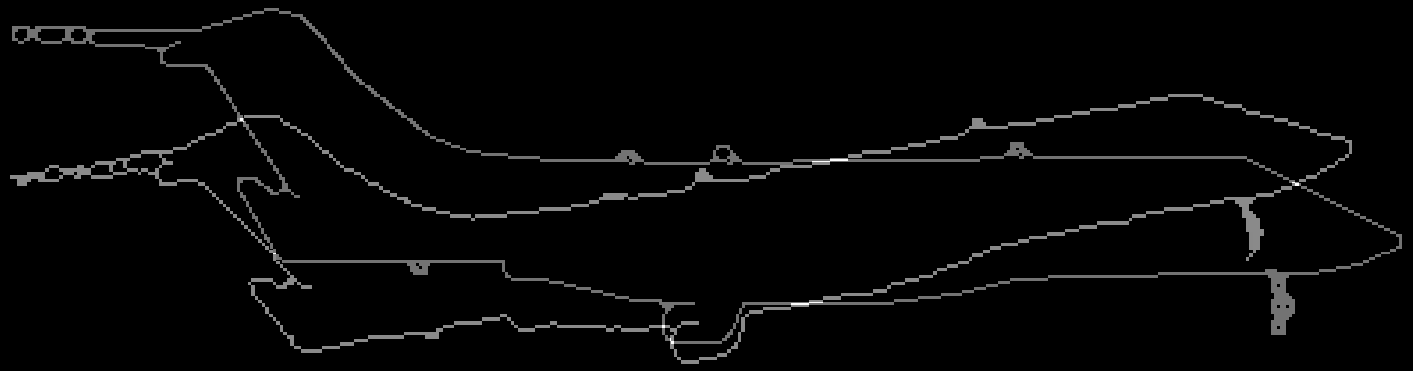
1. **Initialize** transformation (e.g., compute difference in means and scale)
2. **Assign** each point in {Set 1} to its nearest spatial neighbor in {Set 2}
3. **Estimate** transformation parameters
   - e.g., least squares or robust least squares
4. **Transform** the points in {Set 1} using estimated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

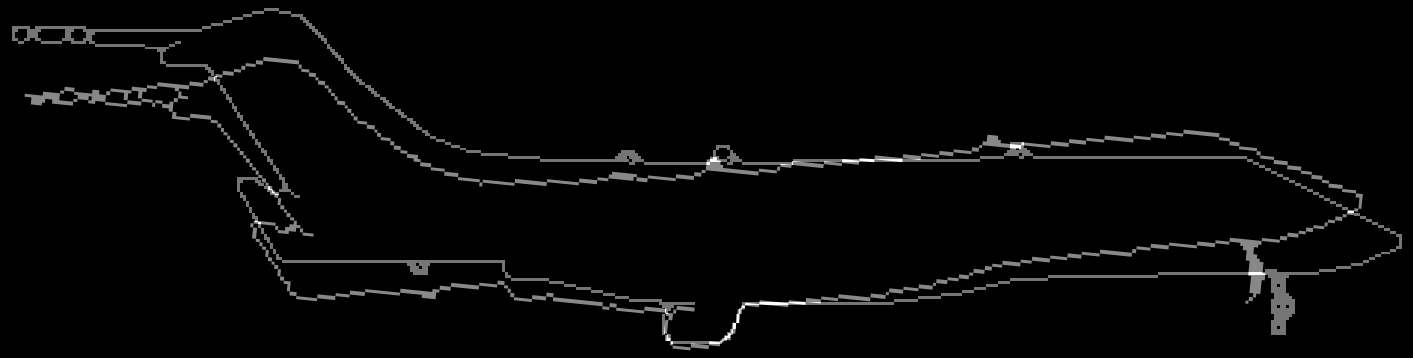
Example: aligning boundaries

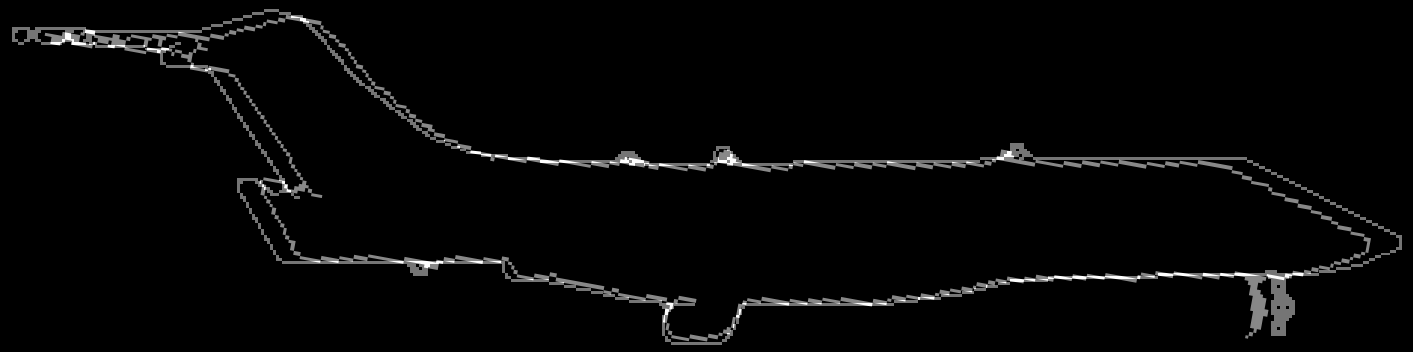
1. Extract edge pixels $p_1..pn$ and $q_1..qm$
2. Compute initial transformation (e.g., compute translation and scaling by center of mass, variance within each image)
3. Get nearest neighbors: for each point p_i find corresponding $\text{match(i)} = \underset{j}{\text{argmin}}\, dist(pi, qj)$
4. Compute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



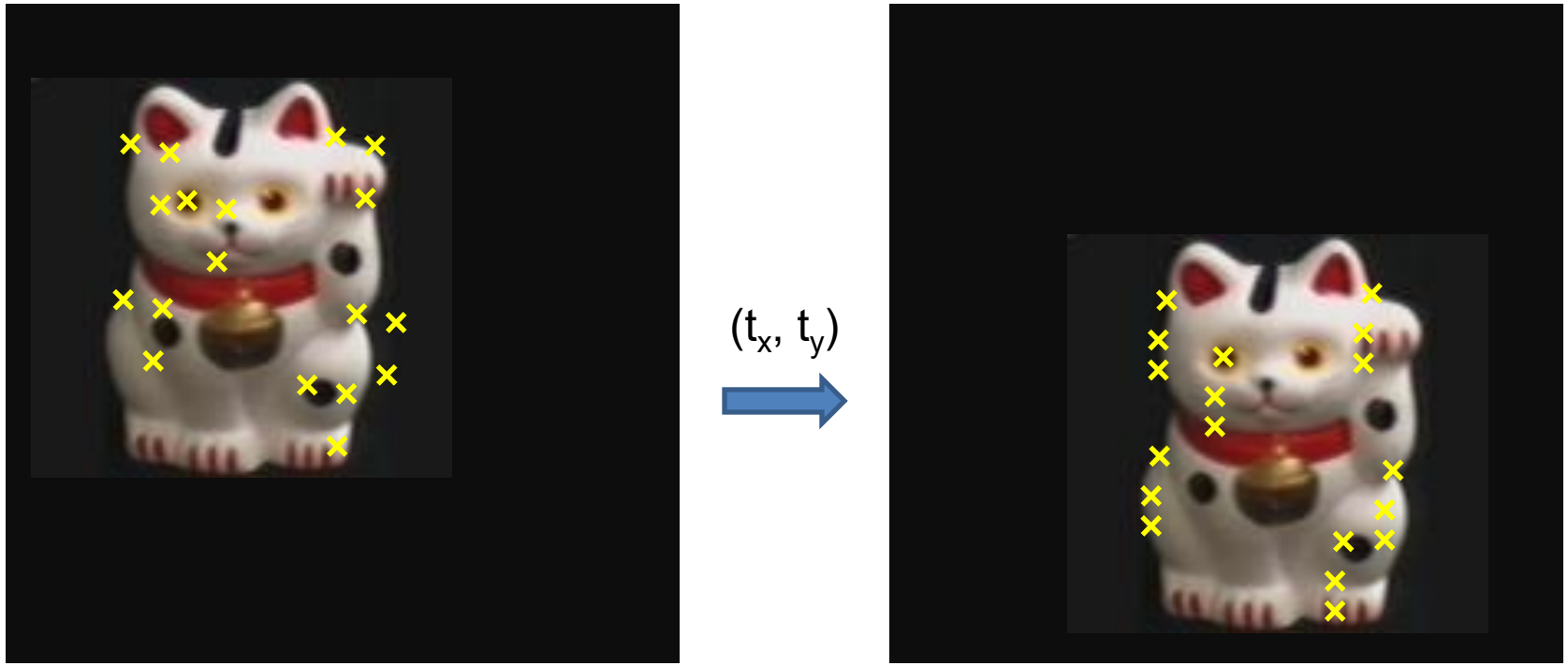








Example: solving for translation



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

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

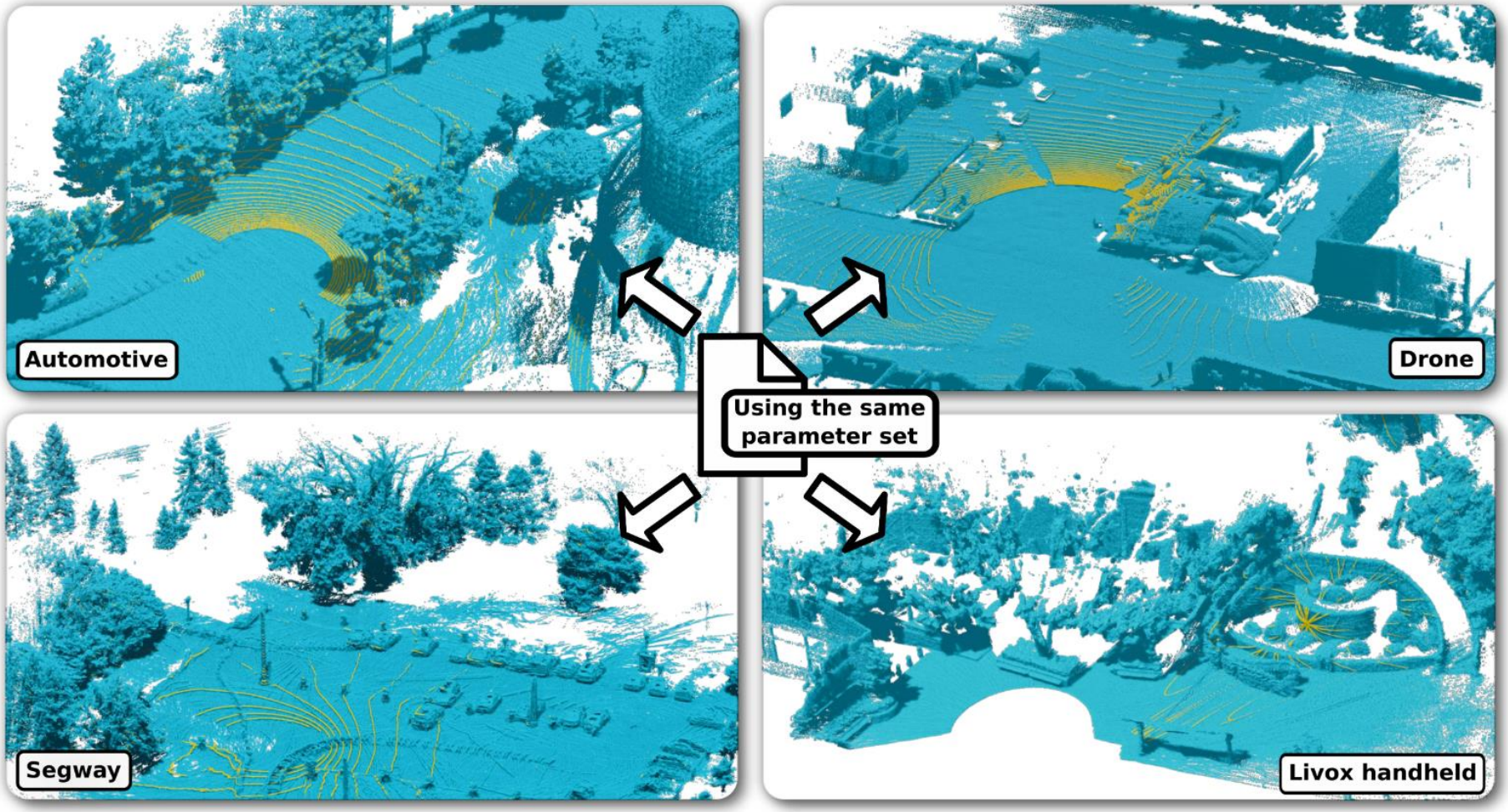
Sparse ICP

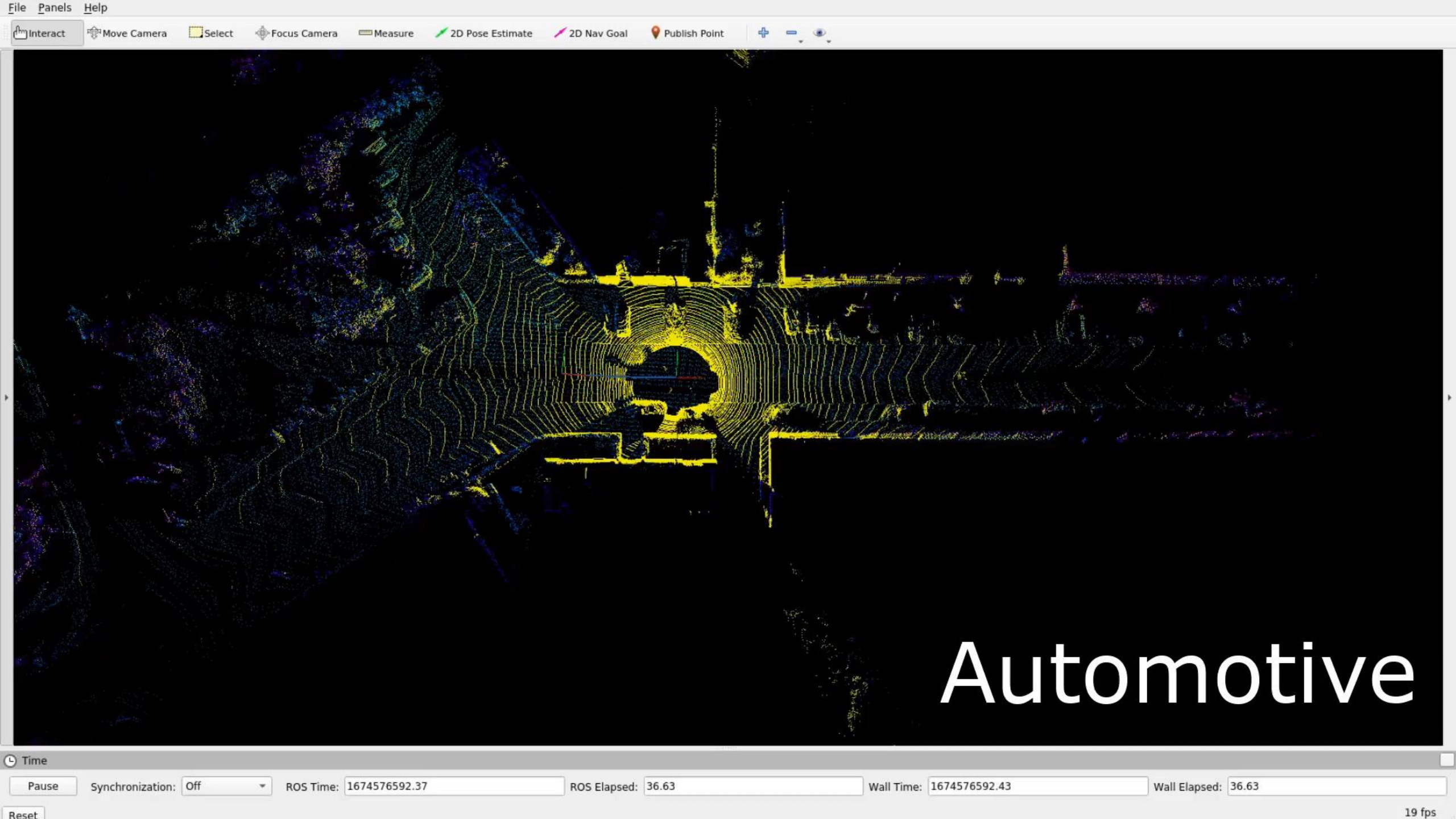
Sofien Bouaziz *Andrea Tagliasacchi* *Mark Pauly*

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way

Ignacio Vizzo Tiziano Guadagnino Benedikt Mersch Louis Wiesmann Jens Behley Cyrill Stachniss




File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Automotive
Time
Pause
Synchronization: Off
ROS Time: 1674576592.37
ROS Elapsed: 36.63
Wall Time: 1674576592.43
Wall Elapsed: 36.63
Reset
19 fps

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
  - Sensitive to initialization

Rough count of mentions in recent literature

- Keypoint 2,180 mentions
- SIFT 3,530 mentions
- “Least Squares” 2,290 mentions
- “Robust Least Squares” 4 mentions
- Hough: 901 mentions
- RANSAC: 1,690 mentions
- ICP: 895 mentions
- Affine 2,970
- ResNet: 8,510 mentions

Google search for site:https://openaccess.thecvf.com [term]
Seems to find results since 2013.

Rough count of mentions in recent literature

- Keypoint 7,600 mentions
- SIFT 3,400 mentions
- “Least Squares” 3,300 mentions
- Hough: 1,200 mentions
- RANSAC: 2,500 mentions
- ICP: 1,300 mentions
- Affine 4,600
- ResNet: 16,400 mentions

Google search for site:https://openaccess.thecvf.com [term]
Seems to find results since 2013.