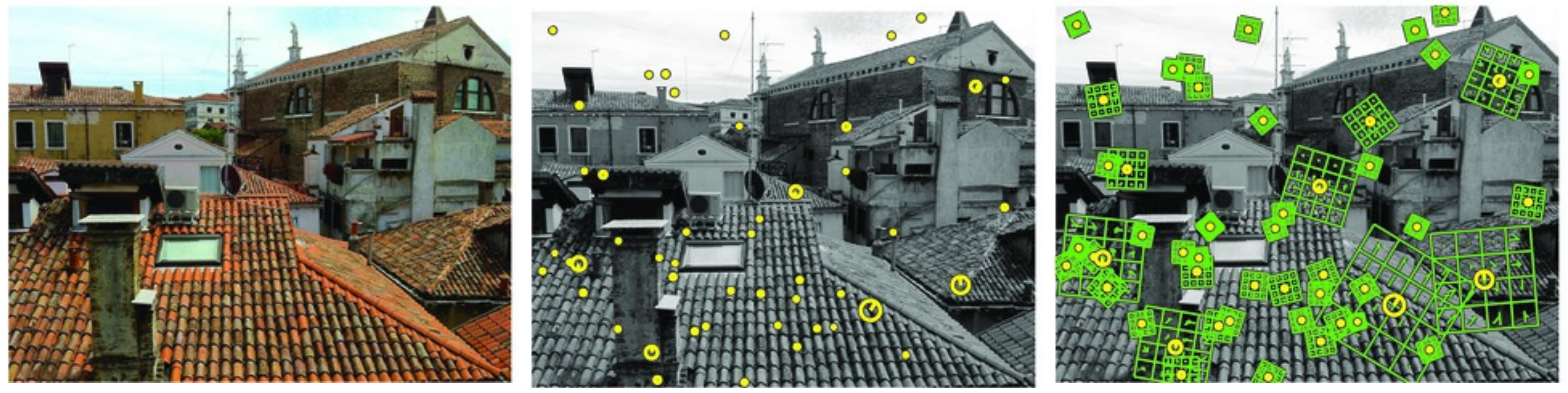
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 14: Planar Geometry and RANSAC

Menu for Today (**October 28, 2024**)

Topics:

- **Planar** Geometry
- **Image Alignment**, Object Recognition
- **RANSAC**

Readings:

- **Today's** Lecture: Szeliski 2.1, 8.1, Forsyth & Ponce 10.4.2

Reminders:

- **Assignment 4**: RANSAC and Panorama Stitching

Today's "**fun**" Example: COTR

Image 1

Image 2

With COTR, we find dense correspondences, which we can reconstruct a dense 3D model from just two calibrated views.

Today's "**fun**" Example: COTR

Image 1

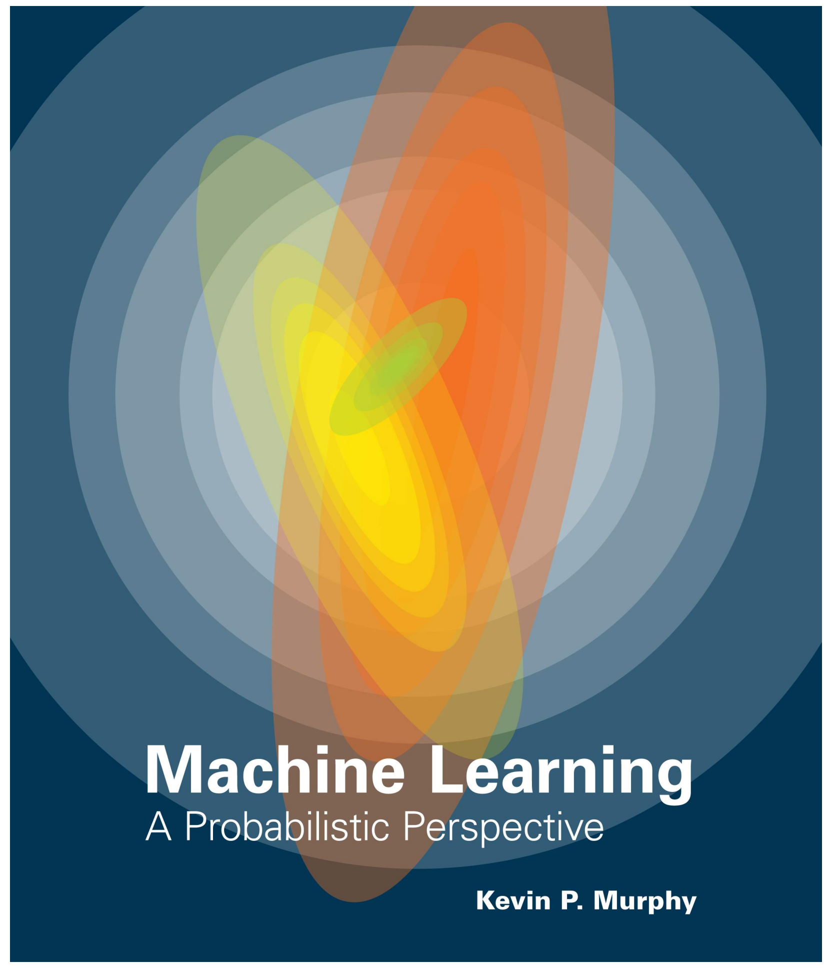
Image 2

With COTR, we find dense correspondences, which we can reconstruct a dense 3D model from just two calibrated views.

Today's "**fun**" Example: Im2Calories

ICCV 2015 paper by **Kevin Murphy**

(UBC's former faculty)



Coincidently Kevin is also author of one of the most prominent ML books

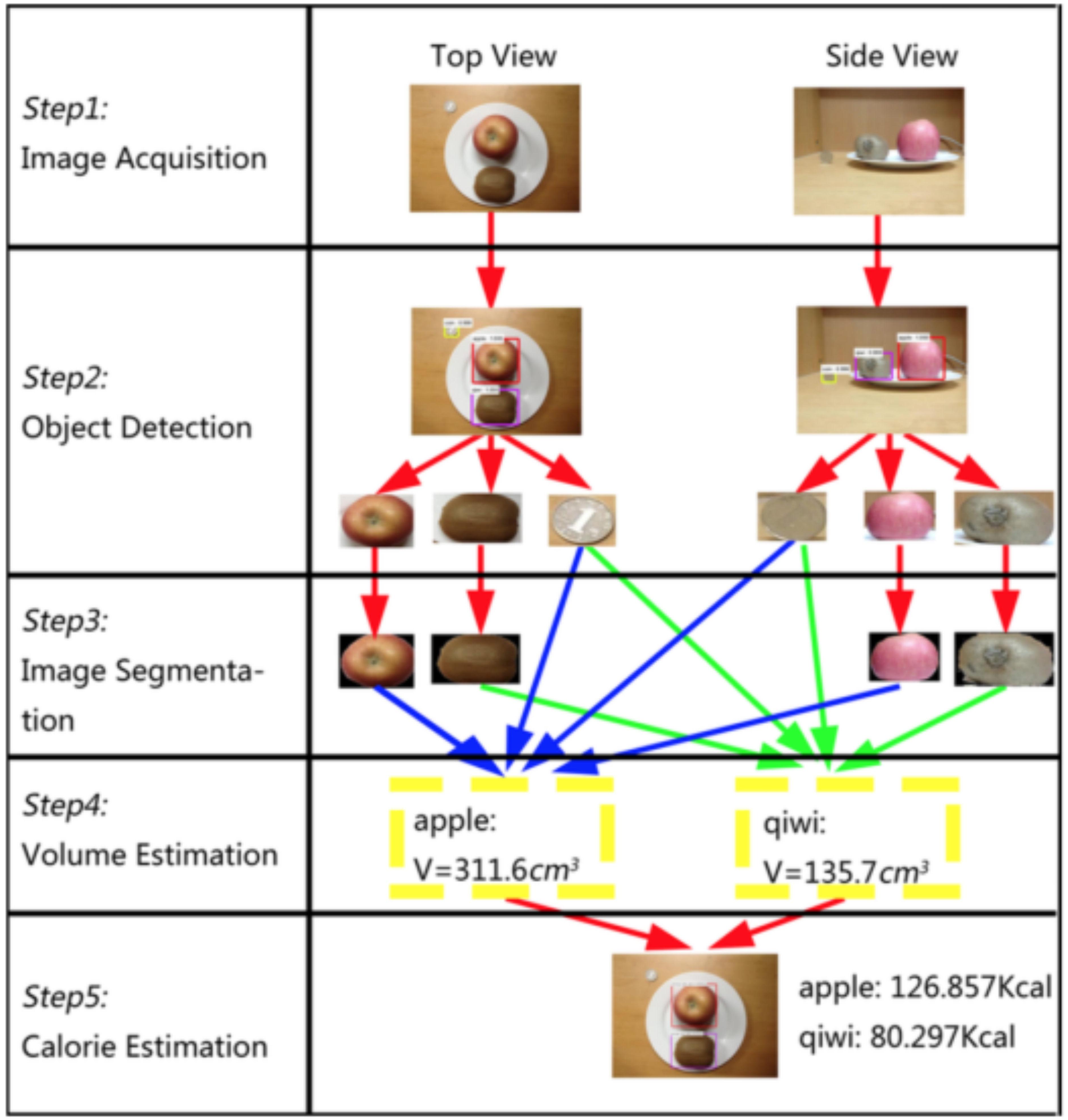


Figure 1: Calorie Estimation Flowchart

Today's "**fun**" Example: Im2Calories

Im2Calories: towards an automated mobile vision food diary

Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, Kevin Murphy amyers@umd.edu, (nickj, rathodv, kbanoop, gorban)@google.com (nsilberman, sguada, gpapan, jonathanhuang, kpmurphy)@google.com

Today's "**fun**" Example: Im2Calories

Im2Calories: towards an automated mobile vision food diary

Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, Kevin Murphy amyers@umd.edu, (nickj, rathodv, kbanoop, gorban)@google.com (nsilberman, sguada, gpapan, jonathanhuang, kpmurphy)@google.com

Today's "**fun**" Example: Im2Calories

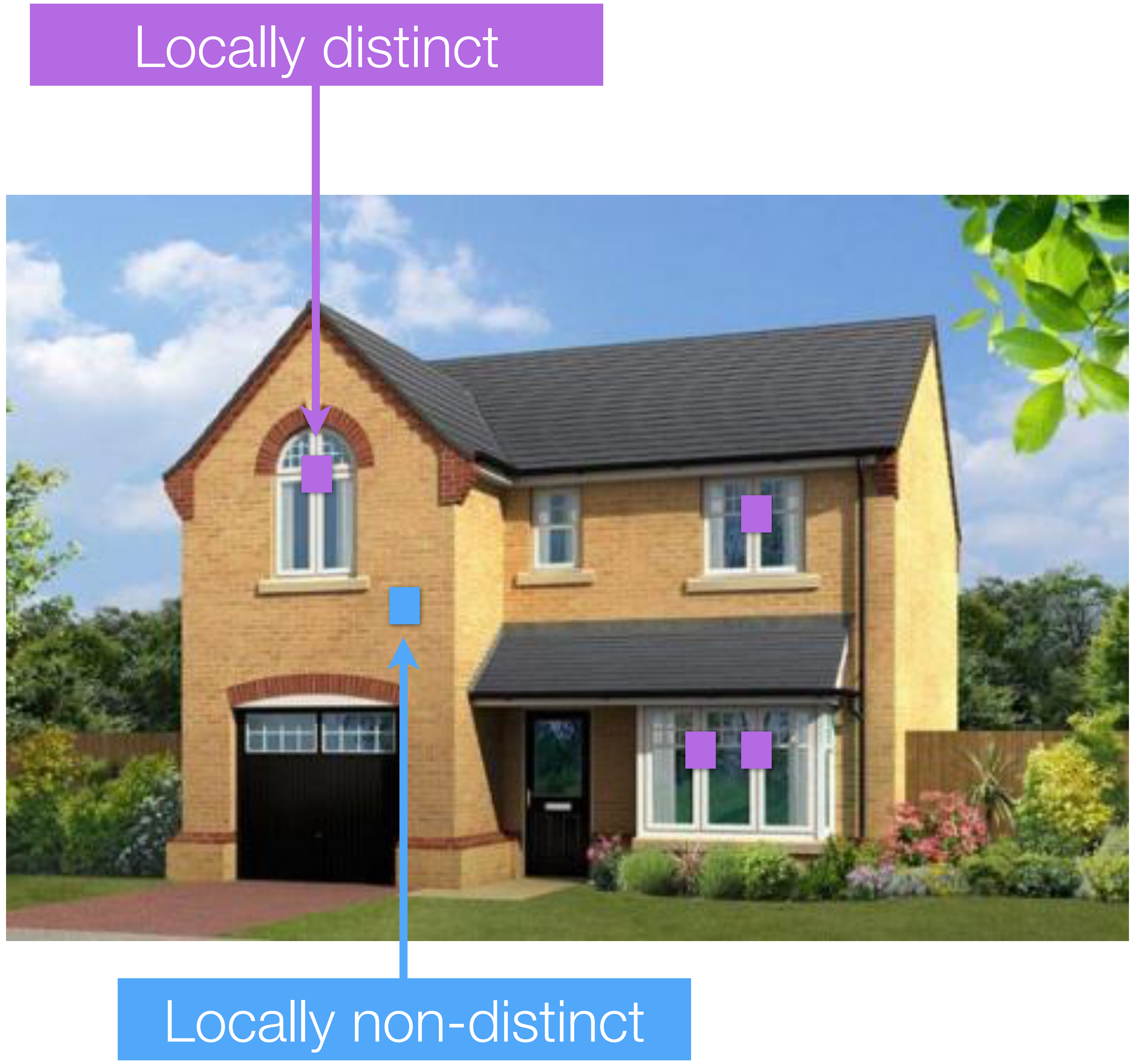
Fun **on-line demo**: http://www.caloriemama.ai/api

Lecture 13: Re-Cap

Keypoint is an image location at which a descriptor is computed

— Locally distinct points

— Easily localizable and identifiable



Lecture 13: Re-Cap

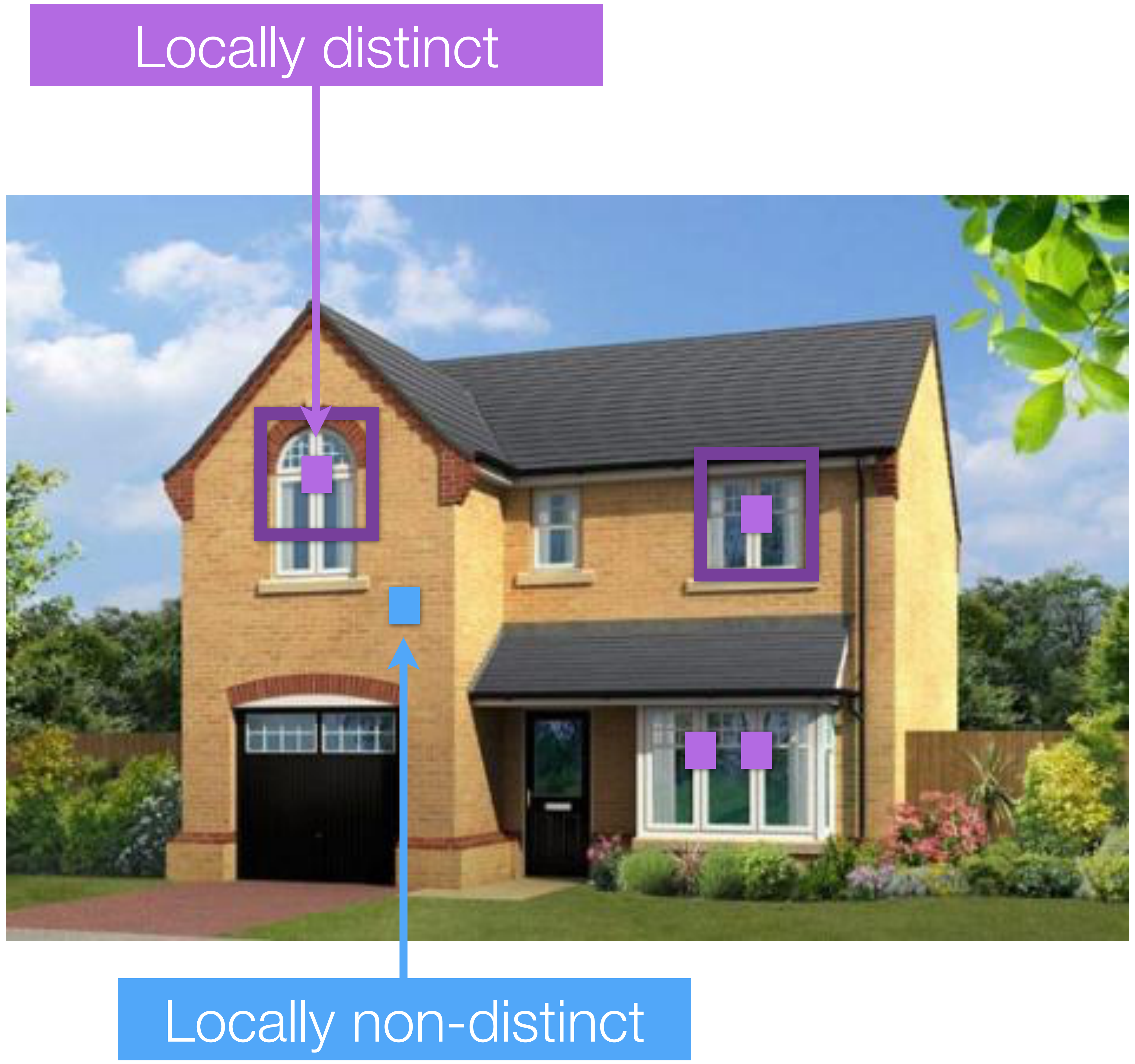
Keypoint is an image location at which a descriptor is computed

— Locally distinct points

— Easily localizable and identifiable

The feature **descriptor** summarizes the local structure around the key point

— Allows us to (hopefully) unique matching of keypoints in presence of object pose variations, image and photometric deformations



Lecture 13: Re-Cap

Keypoint is an image location at which a descriptor is computed

— Locally distinct points

— Easily localizable and identifiable

The feature **descriptor** summarizes the local structure around the key point

— Allows us to (hopefully) unique matching of keypoints in presence of object pose variations, image and photometric deformations

Note, for repetitive structure this would still not give us unique matches.

Lecture 13: Re-Cap

— We motivated SIFT for identifying locally distinct keypoints in an image (**detection**)

— SIFT features (**description**) are invariant to translation, rotation, and scale; robust to 3D pose and illumination

1. Multi-scale extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

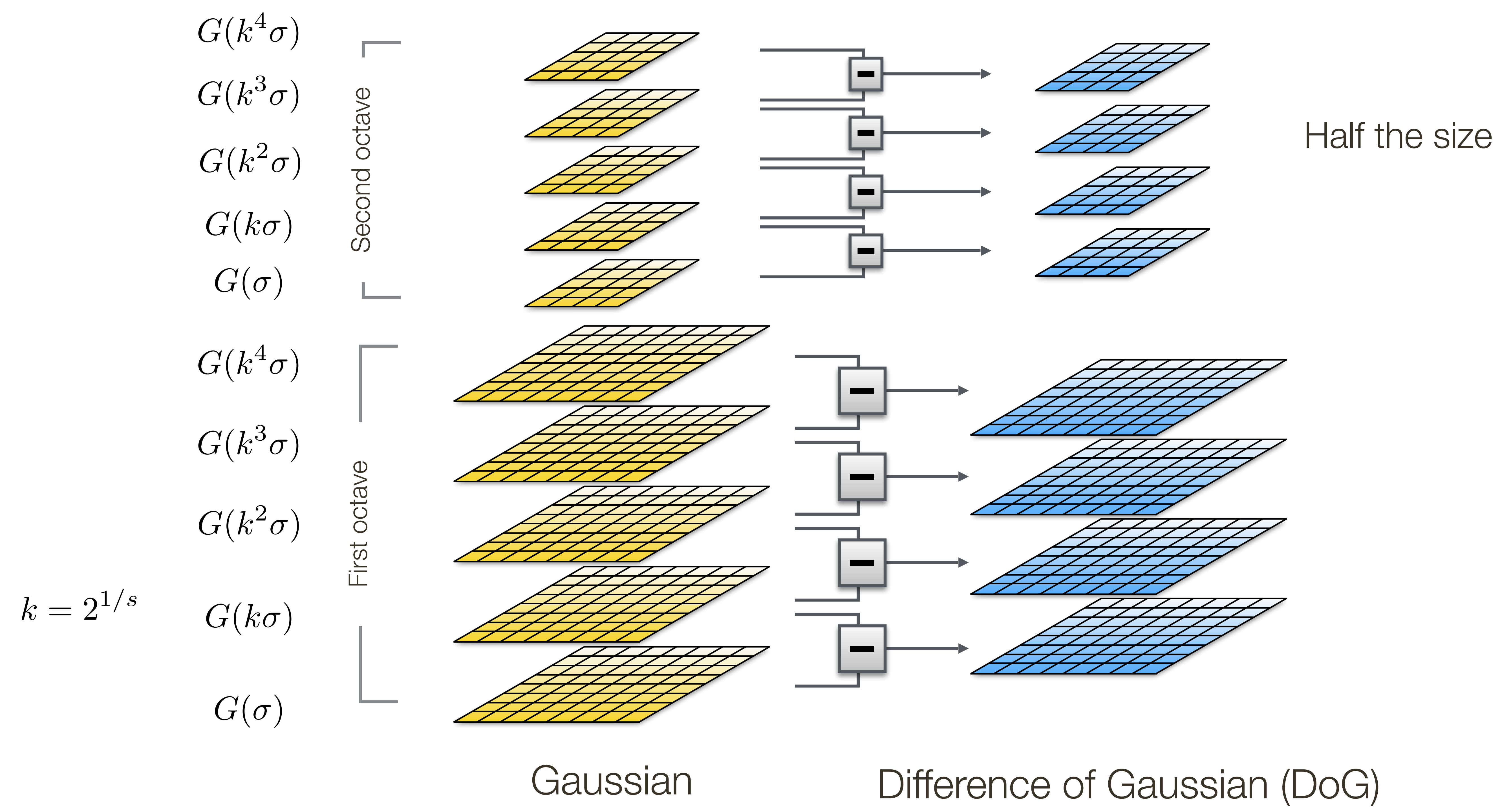
Lecture 13: Re-Cap

Four steps to SIFT feature generation:

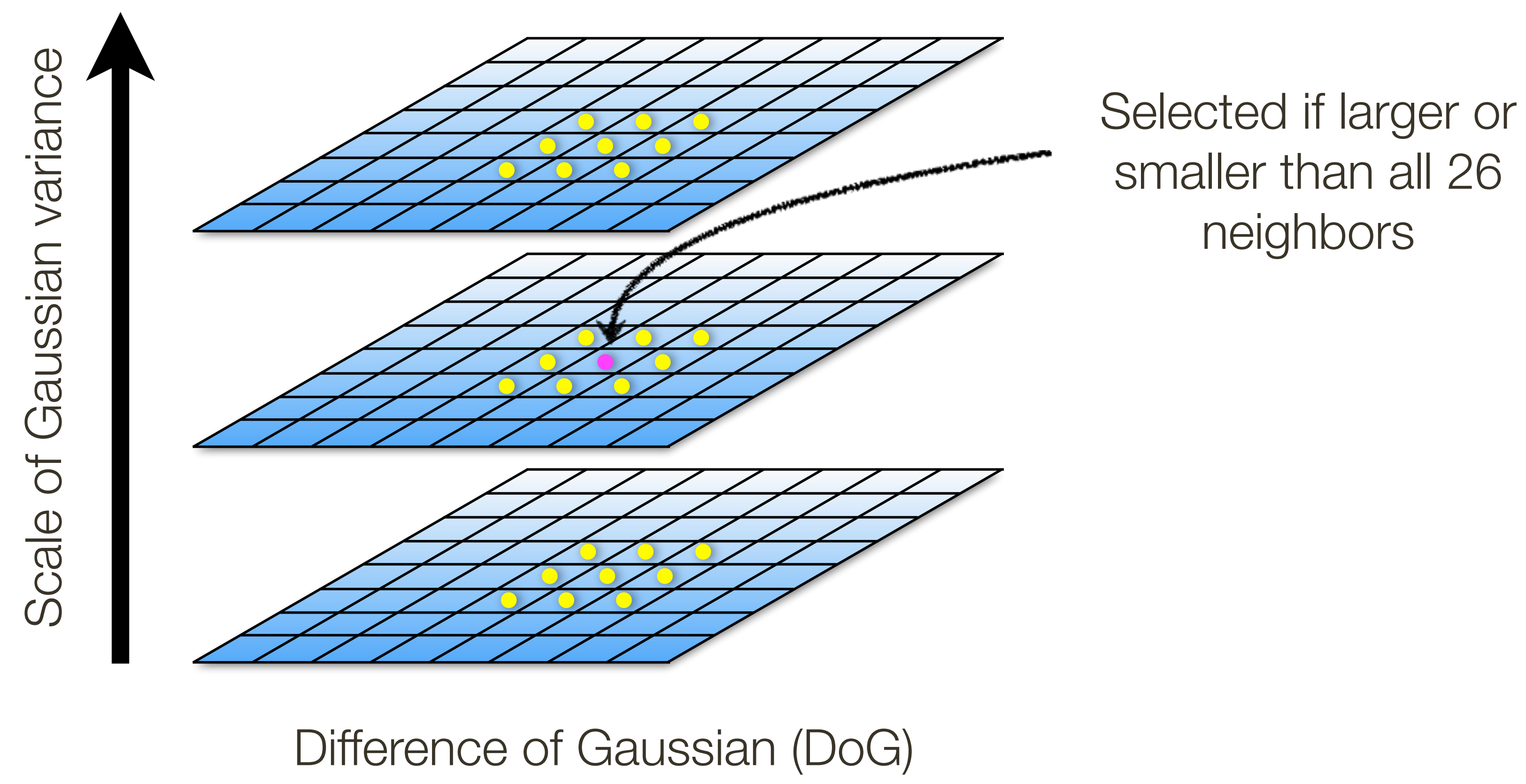
1. **Scale-space representation and local extrema detection**
   — use DoG pyramid
   — 3 scales/octave, down-sample by factor of 2 each octave

Output: (x, y, s) for each keypoint

Lecture 13: Re-Cap — Multi-scale Extrema Detection



Detect maxima and minima of Difference of Gaussian in scale space



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Lecture 13: Re-Cap

Four steps to SIFT feature generation:

1. **Scale-space representation and local extrema detection**
   — use DoG pyramid **Output**: (x, y, s) for each keypoint
   — 3 scales/octave, down-sample by factor of 2 each octave
2. **Keypoint localization**
   — select stable keypoints (threshold on magnitude of extremum, ratio of principal curvatures) **Output**: Remove some (weak) keypoints

Lecture 13: Re-Cap — Keypoint Localization

— After keypoints are detected, we remove those that have **low contrast** or are **poorly localized** along an edge

How do we decide whether a keypoint is poorly localized, say along an edge, vs. well-localized?

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

Lecture 13: Re-Cap

Four steps to SIFT feature generation:

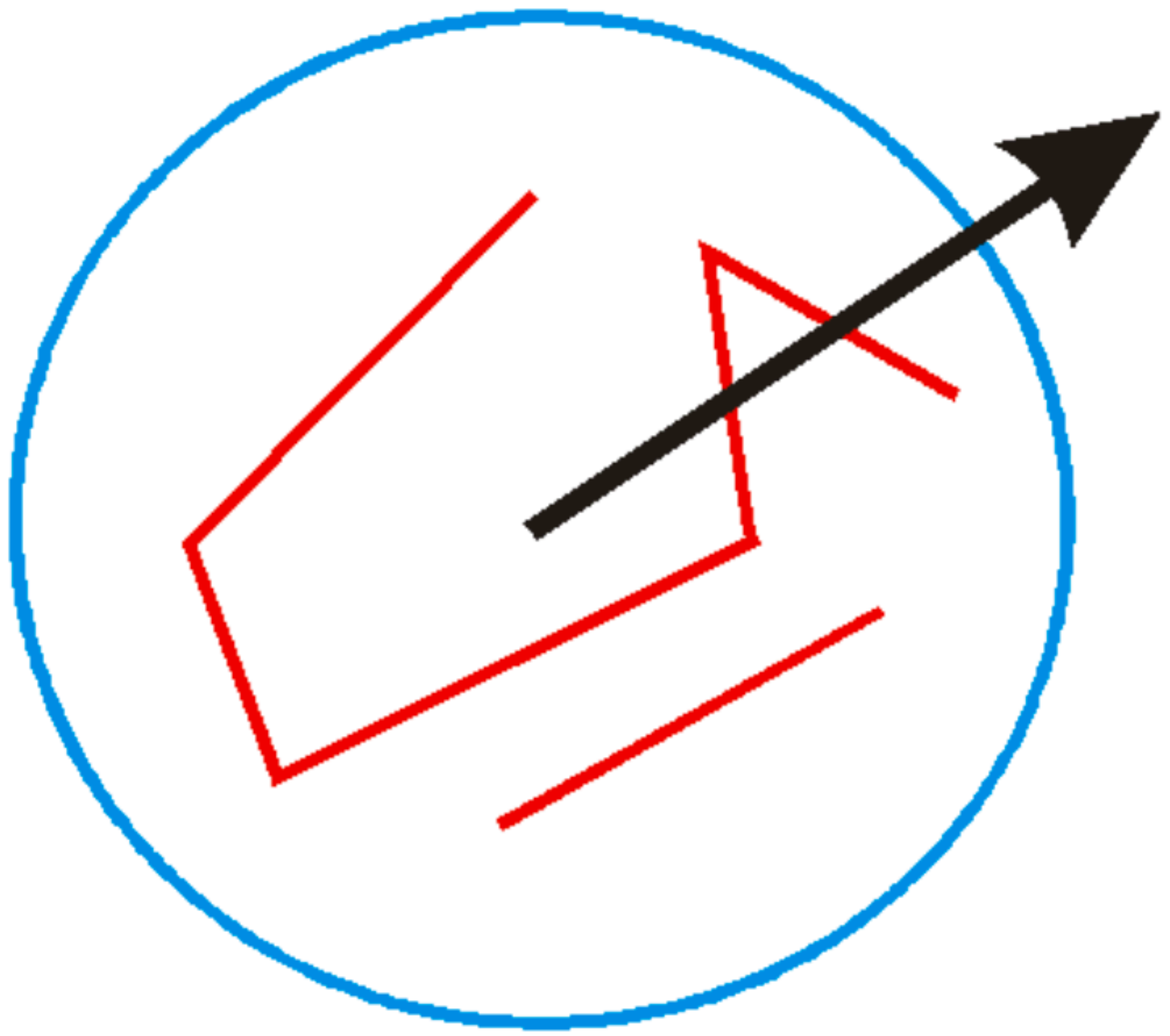
1. **Scale-space representation and local extrema detection**
   — use DoG pyramid **Output**: (x, y, s) for each keypoint
   — 3 scales/octave, down-sample by factor of 2 each octave
2. **Keypoint localization**
   — select stable keypoints (threshold on magnitude of extremum, ratio of principal curvatures) **Output**: Remove some (weak) keypoints
3. **Keypoint orientation assignment** **Output**: Orientation for each keypoint
   — based on histogram of local image gradient directions

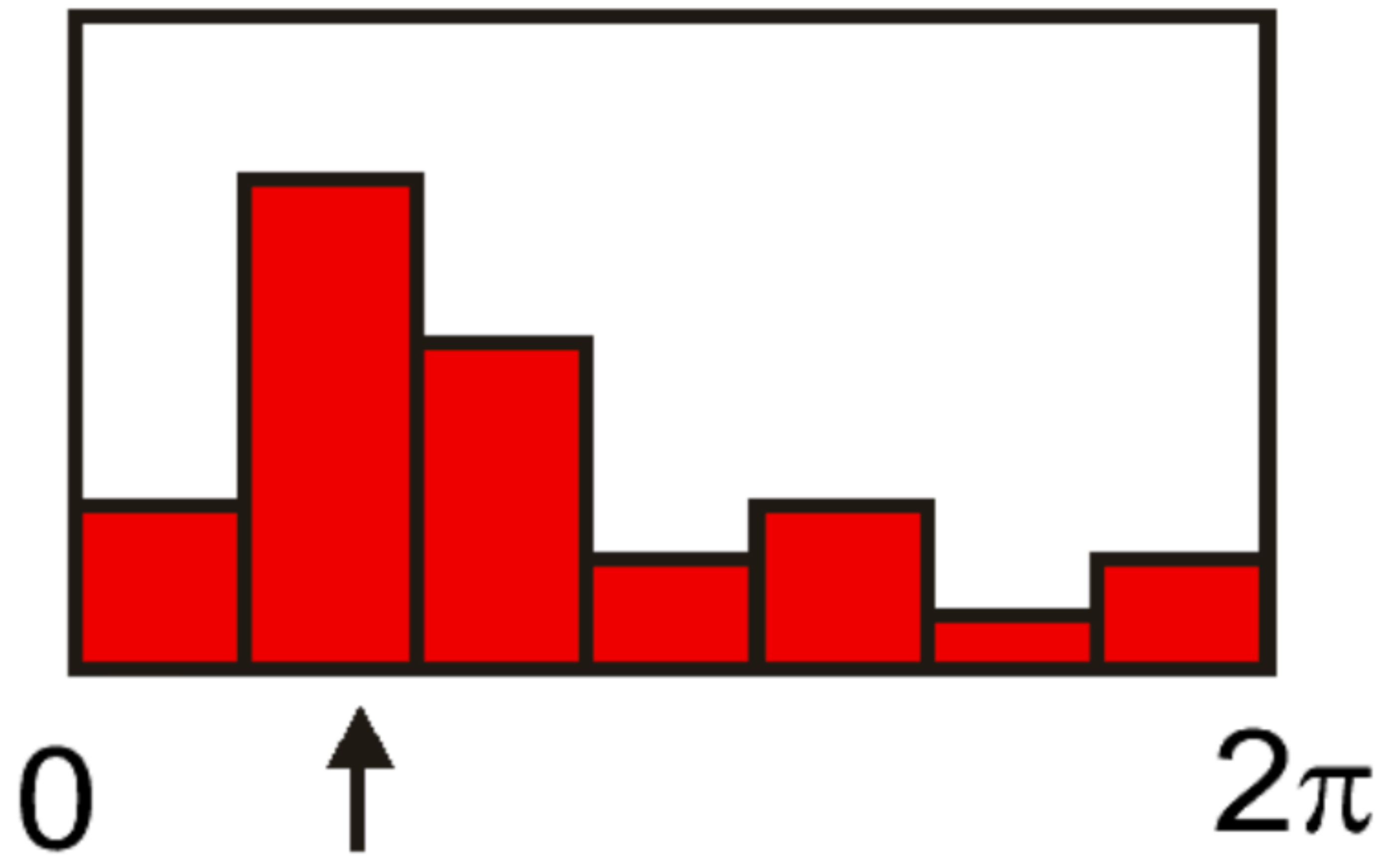
Lecture 13: Re-Cap — Orientation Assignment

— Create **histogram** of local gradient directions computed at selected scale

— Assign **canonical orientation** at peak of smoothed histogram

— Each key specifies stable 2D coordinates (x , y , scale, orientation)





Lecture 13: Re-Cap

Four steps to SIFT feature generation:

1. **Scale-space representation and local extrema detection**
   — use DoG pyramid
   Output: (x, y, s) for each keypoint
   — 3 scales/octave, down-sample by factor of 2 each octave

2. **Keypoint localization**
   — select stable keypoints (threshold on magnitude of extremum, ratio of principal curvatures)
   Output: Remove some (weak) keypoints

3. **Keypoint orientation assignment**
   Output: Orientation for each keypoint
   — based on histogram of local image gradient directions

4. **Keypoint descriptor**
   — histogram of local gradient directions — vector with $8 \times (4 \times 4) = 128$ dim
   — vector normalized (to unit length)
   Output: 128D normalized vector characterizing the keypoint region

Lecture 13: Histogram of Oriented Gradients (**HOG**)

1 cell step size

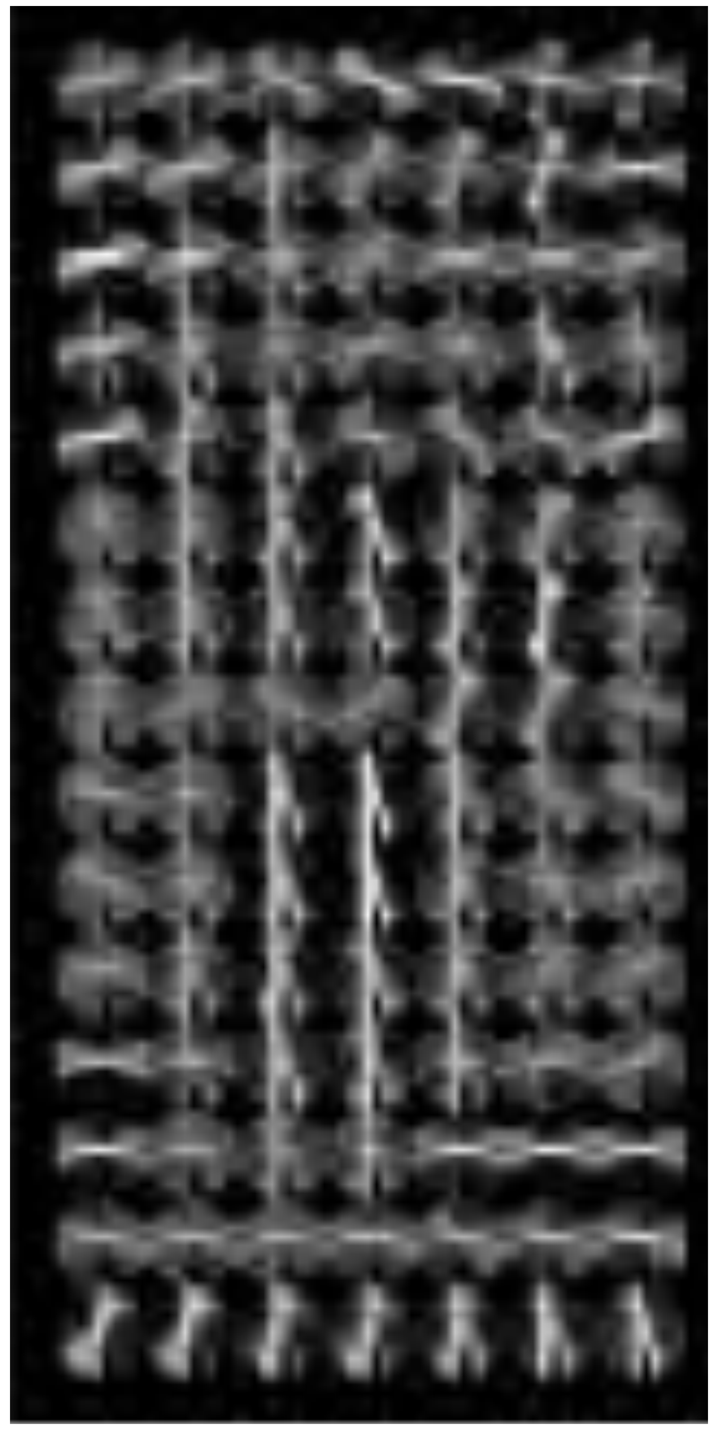
visualization

Pedestrian detection

128 pixels
16 cells
15 blocks

64 pixels
8 cells
7 blocks

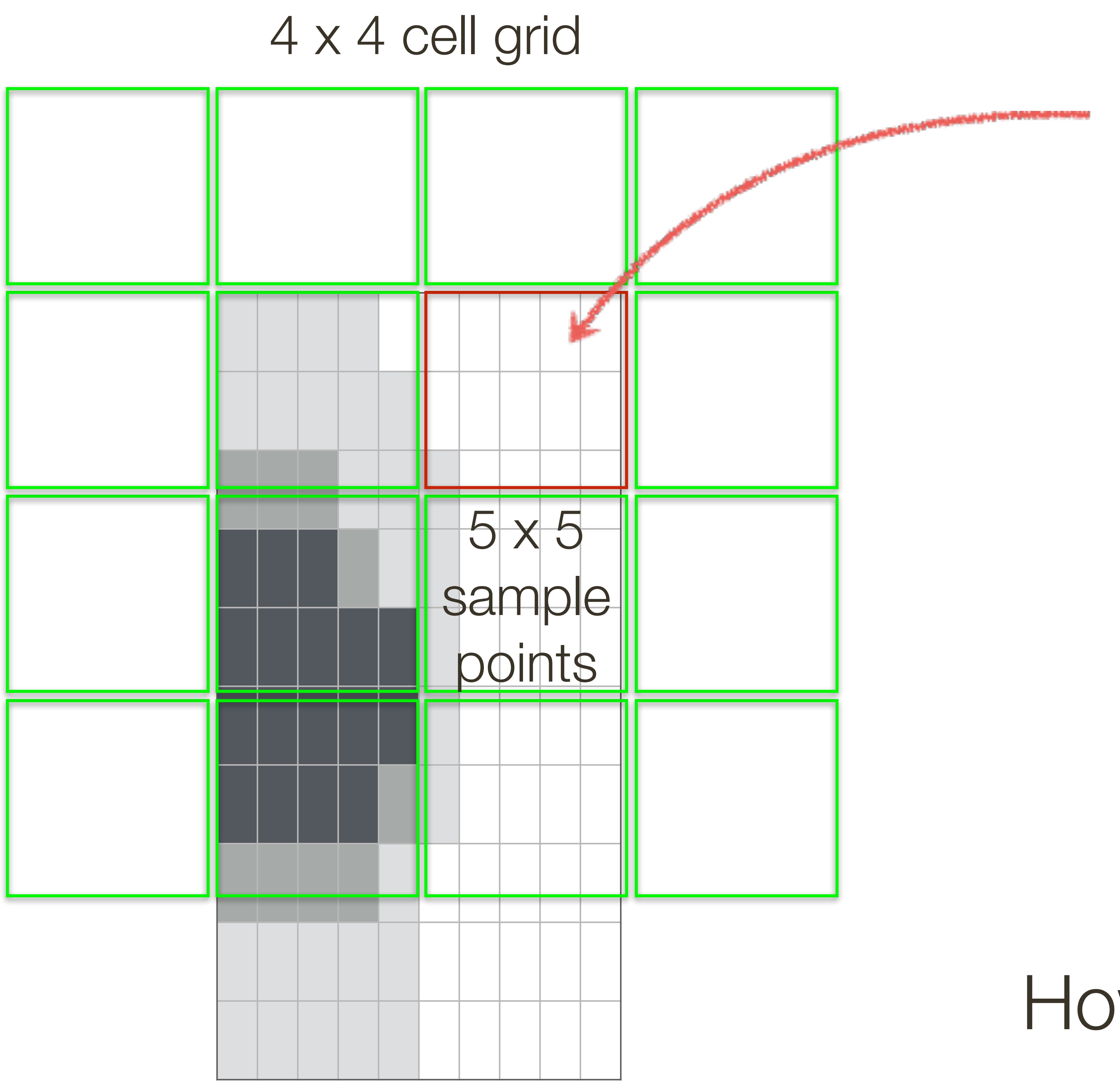
15 x 7 x 4 x 9 =
3780



Redundant representation due to overlapping blocks

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

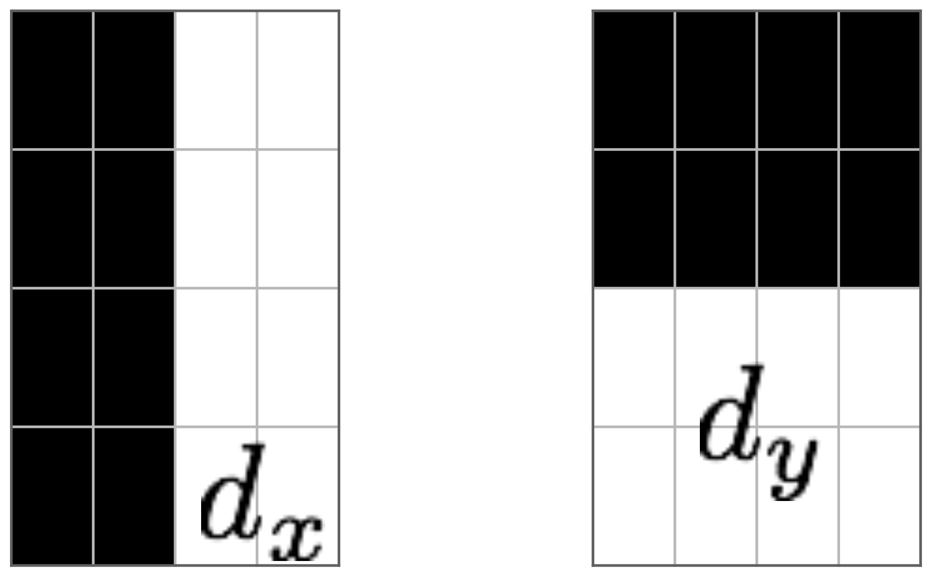
Lecture 13: 'Speeded' Up Robust Features



Each cell is represented by 4 values:

$$\left[\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right]$$

Haar wavelets filters
(Gaussian weighted from center)

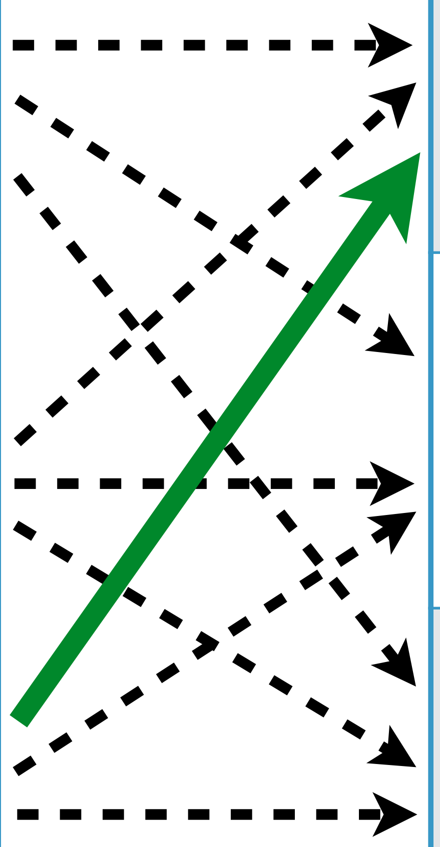


How big is the SURF descriptor?

64 dimensions

Lecture 13: Summary

Keypoint Detection Algorithms	**Representation**
Harris Corners	(x,y,s)
LoG / Blobs	(x,y,s)
SIFT	(x,y,s,theta)

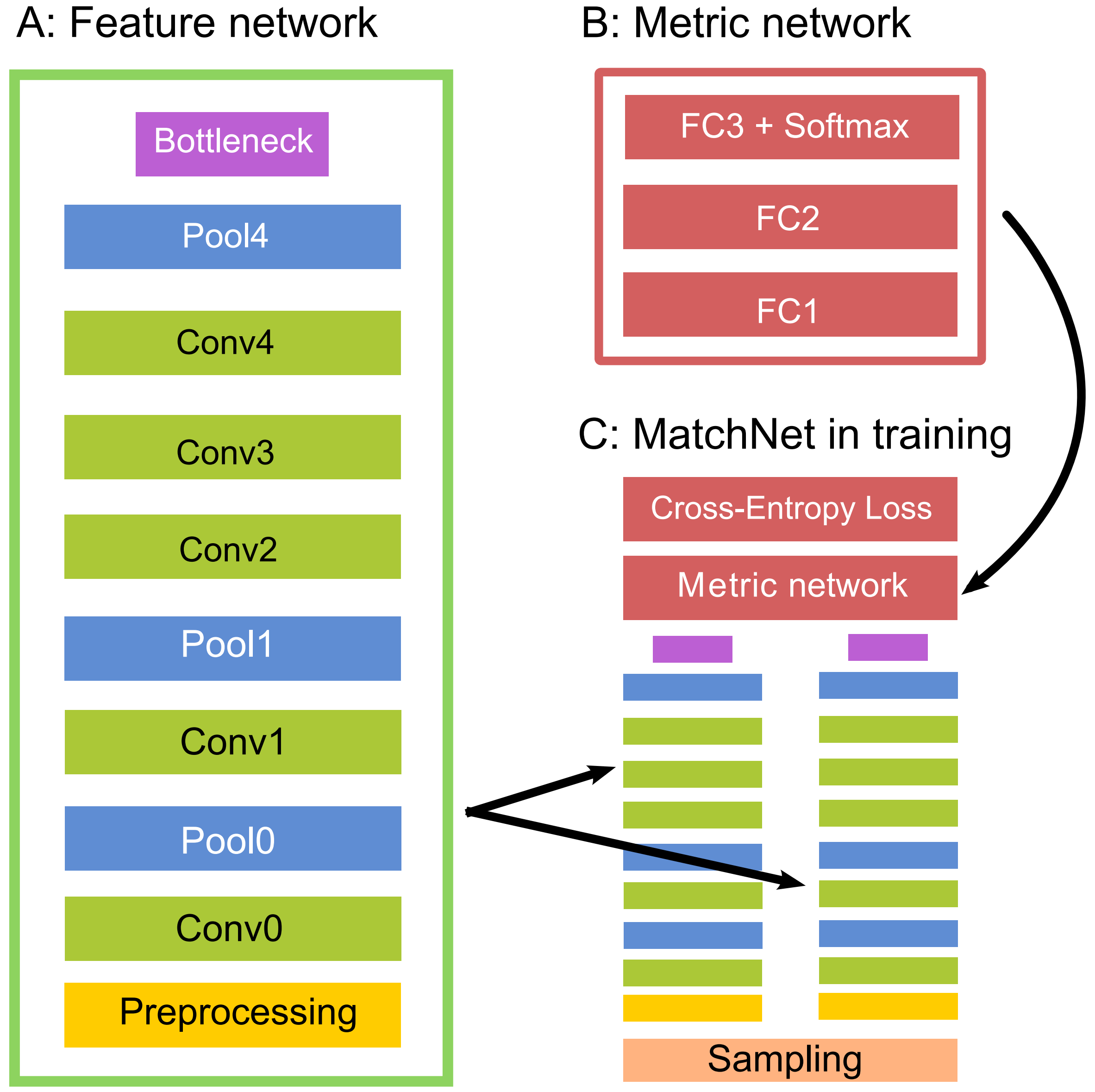


Keypoint Description Algorithms	**Representation**
SIFT	128D
Histogram of Oriented Gradients	3780D
SURF	64D

Learning Descriptors

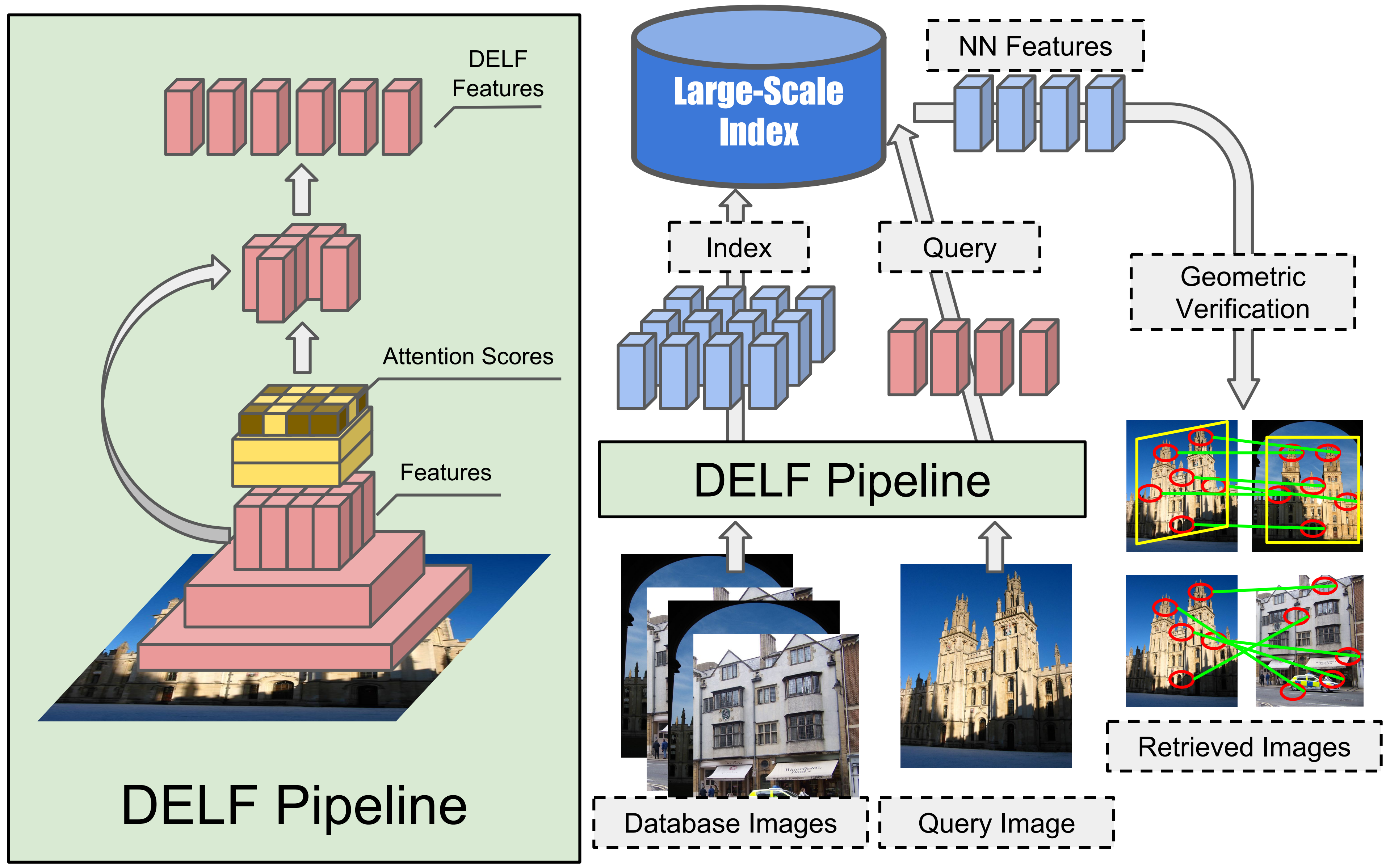
- Deep networks for descriptor learning

Patch labels



[MatchNet
Han et al 2015]

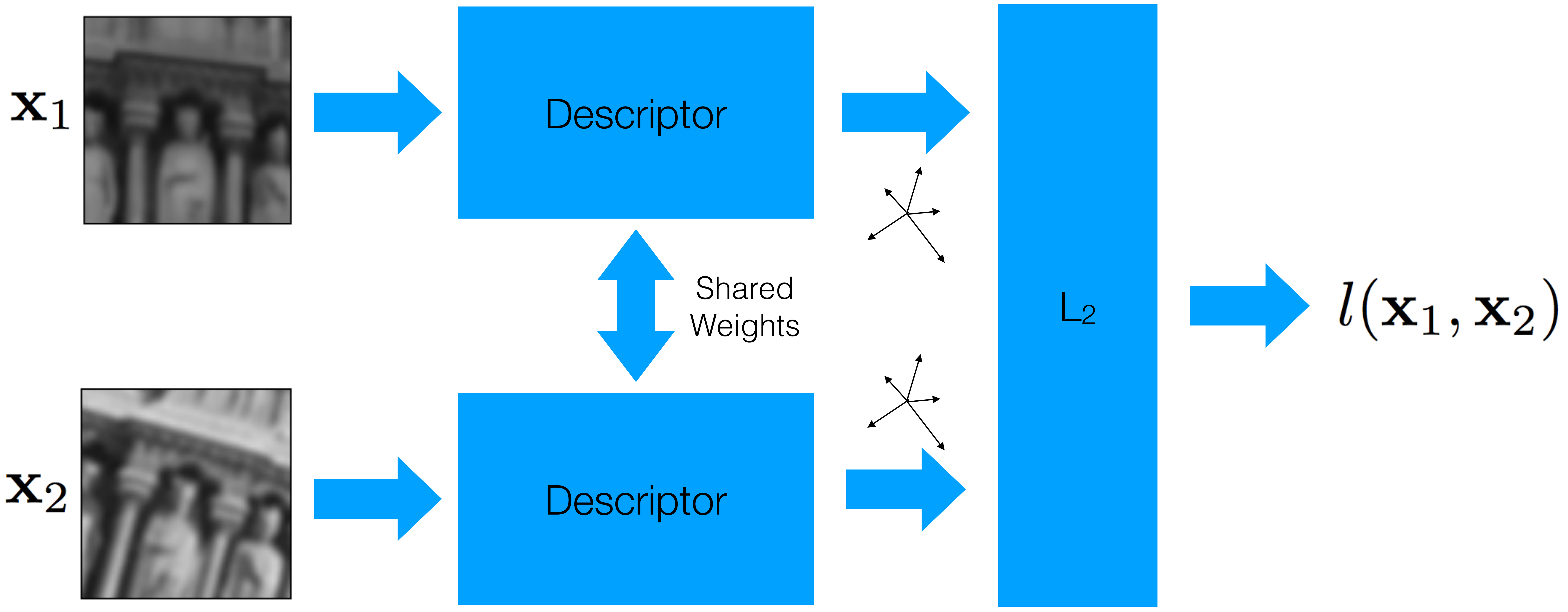
Image labels, also learns interest function



[DELF
Noh et al 2017]

DeepDesc [ICCV 2015]

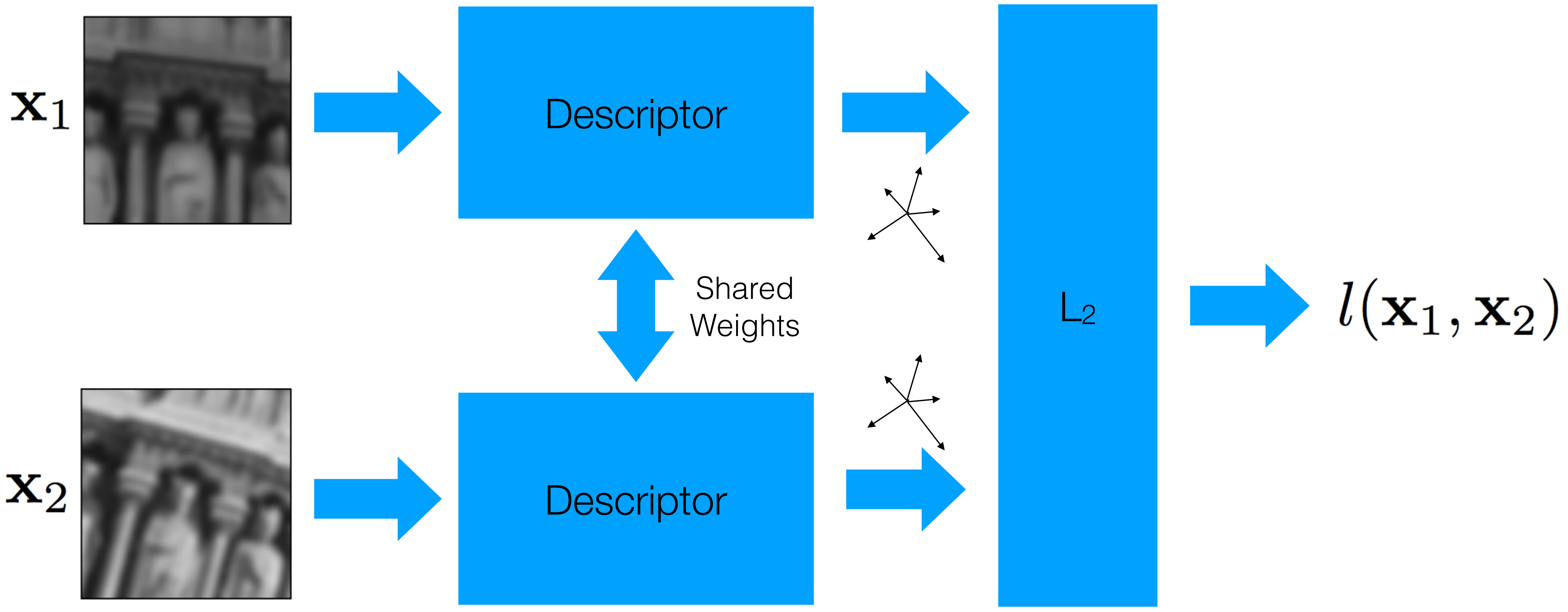
Learning an "embedding"



Slide credits, Eduard Trulls

DeepDesc [ICCV 2015]

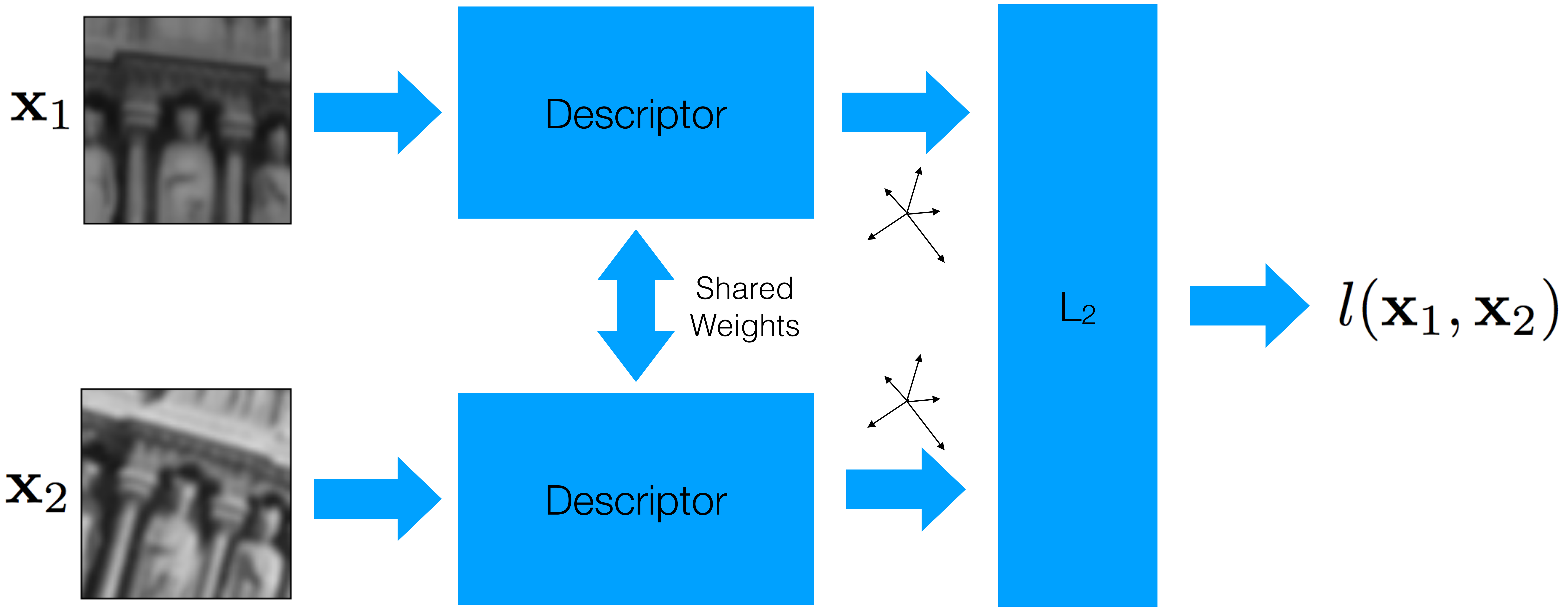
Learning an "embedding"



Minimize the distance for corresponding matches.

DeepDesc [ICCV 2015]

Learning an “embedding”

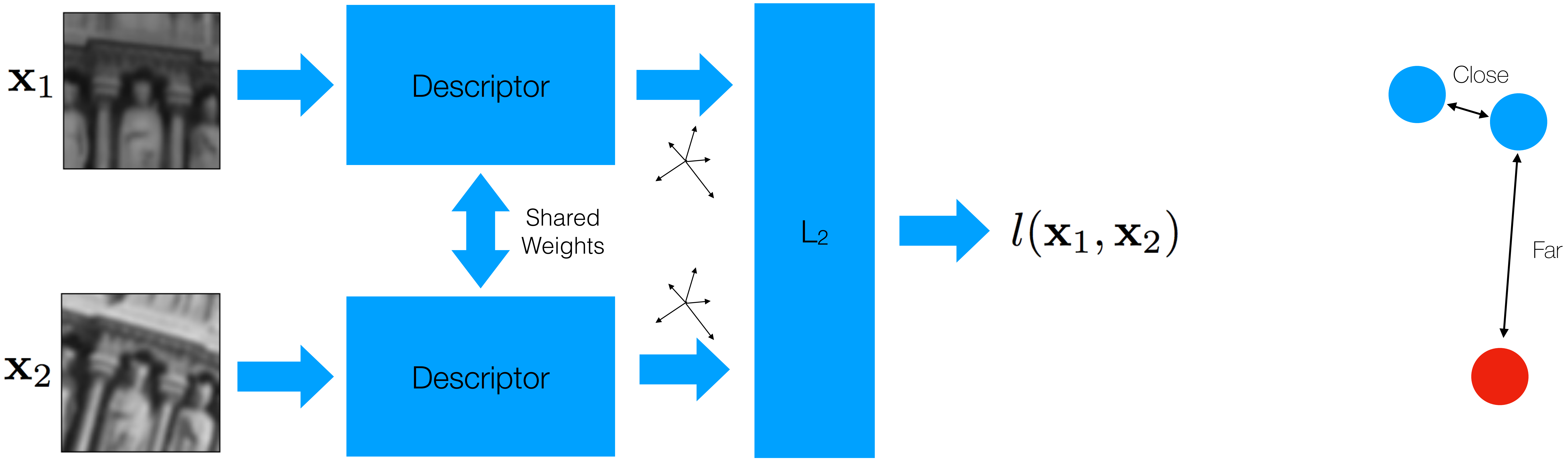


Minimize the distance for corresponding matches.

Maximize it for non-corresponding patches.

DeepDesc [ICCV 2015]

Learning an "embedding"



Minimize the distance for corresponding matches.

Maximize it for non-corresponding patches.

Slide credits, Eduard Trulls

Image **Panoramas**

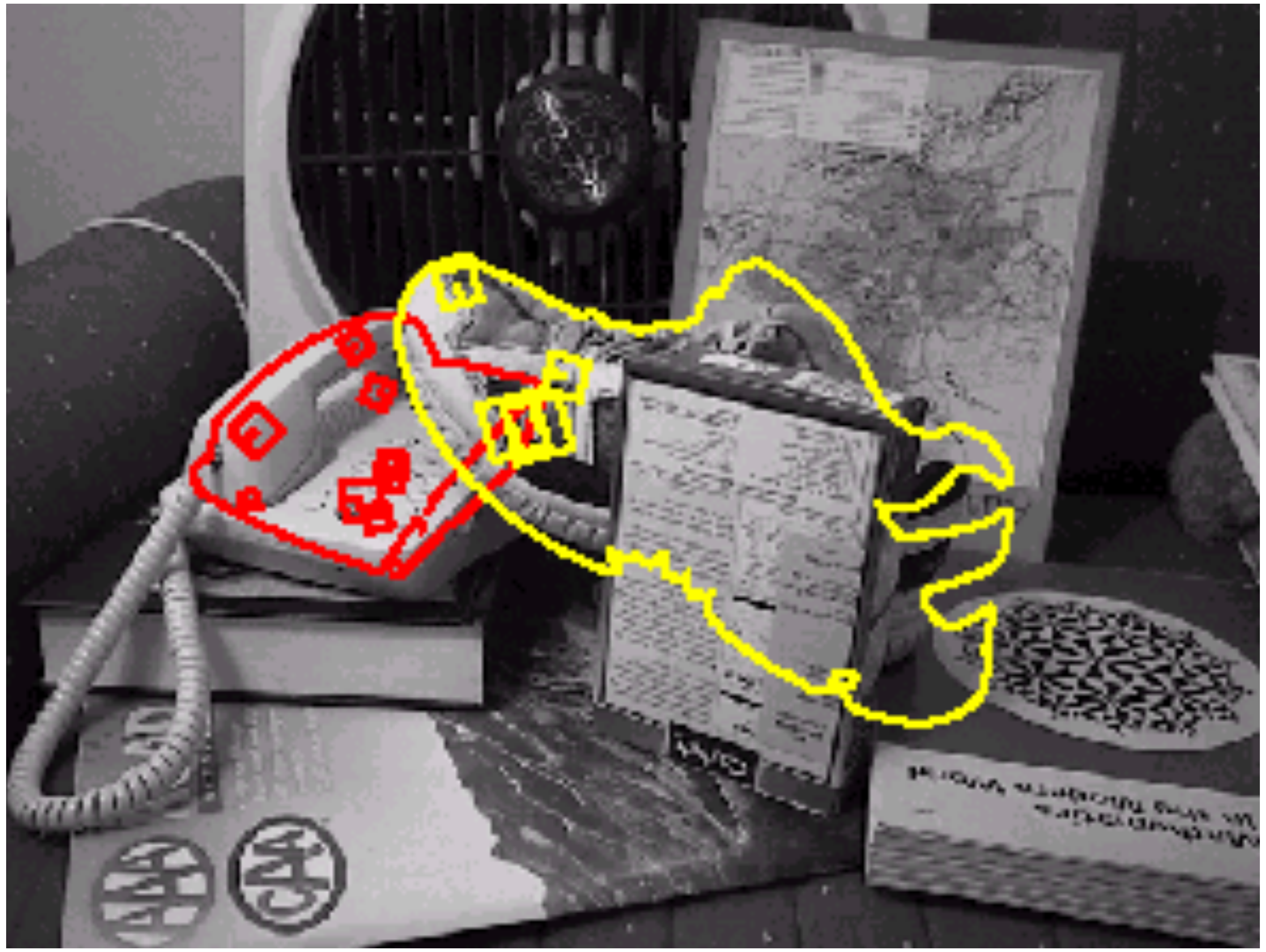
Planar Object **Instance Recognition**

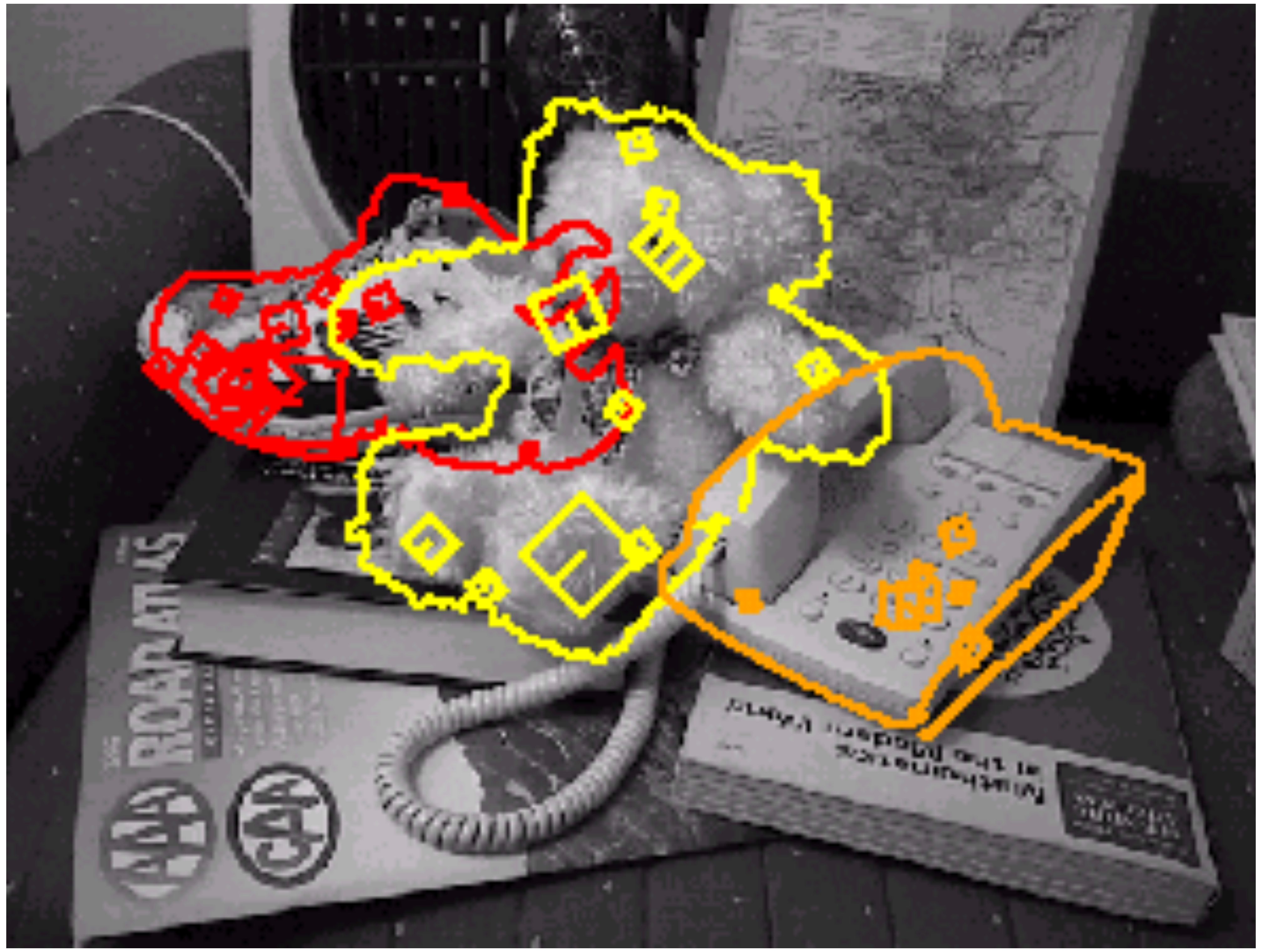
Database of planar objects

Instance recognition

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Recognition under **Occlusion**





Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Learning **Goals**

1. Linear (Projective) Transformations
2. Good results don't happen by chance (or do they?)
3. Good == more support

Image **Alignment**

Aim: Warp one image to align with another

Image **Alignment**

Aim: Warp one image to align with another using a 2D transformation

Image **Alignment**

Step 1: Find correspondences (matching points) across two images

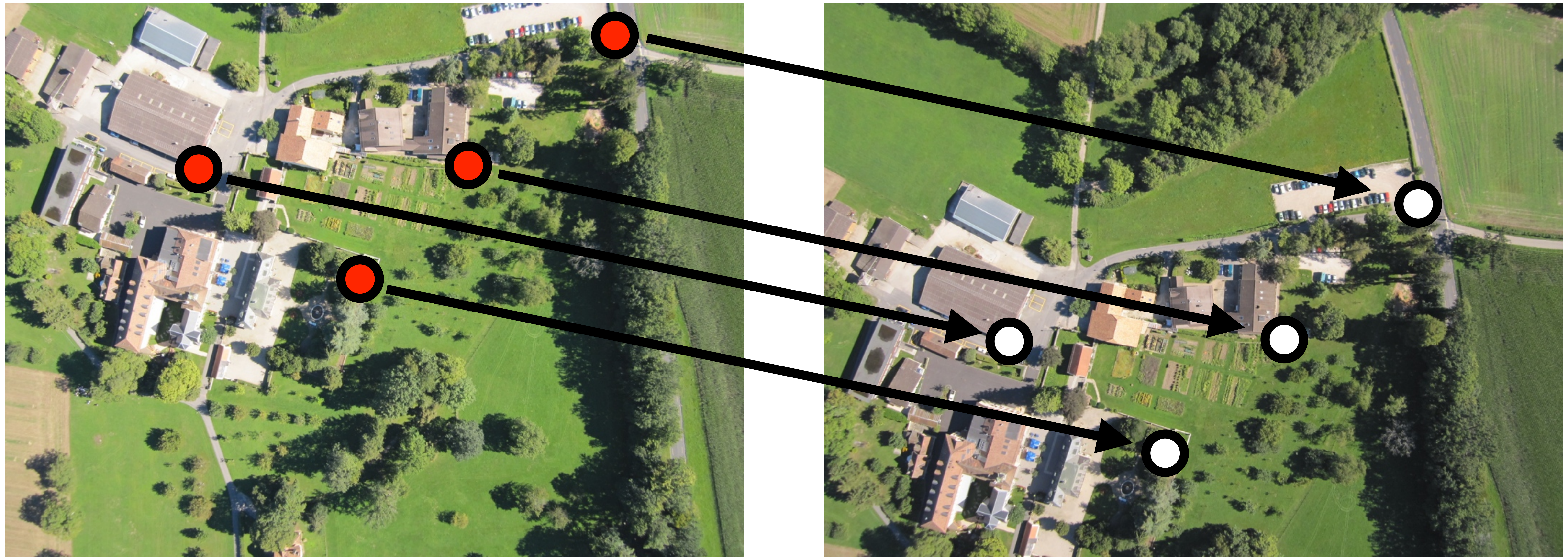


Image **Alignment**

Step 2: Compute the transformation to align the two images

Image **Alignment**

Not all points will match across two images, we can also reject outliers

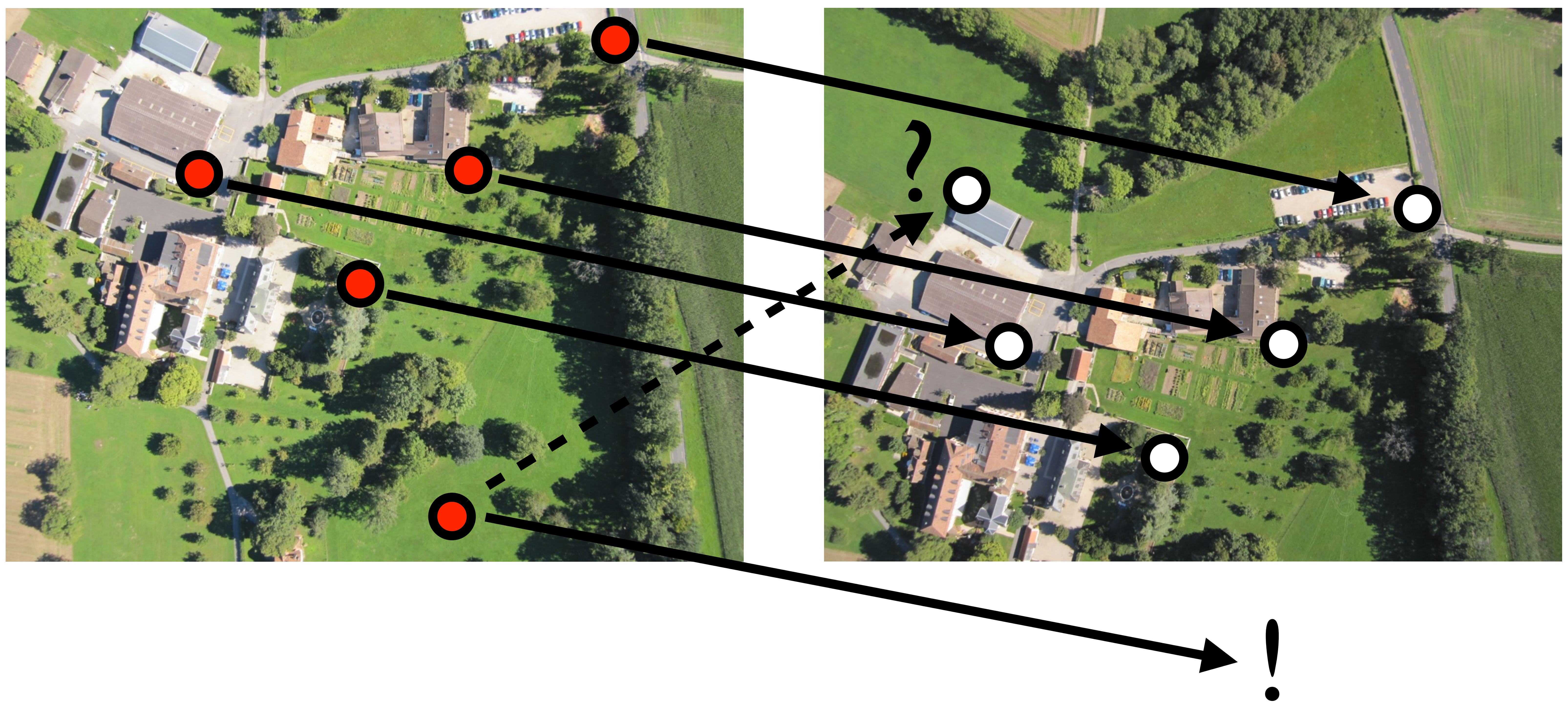


Image **Alignment**

Not all points will match across two images, we can also reject outliers

Planar Geometry

— 2D Linear + **Projective** transformations

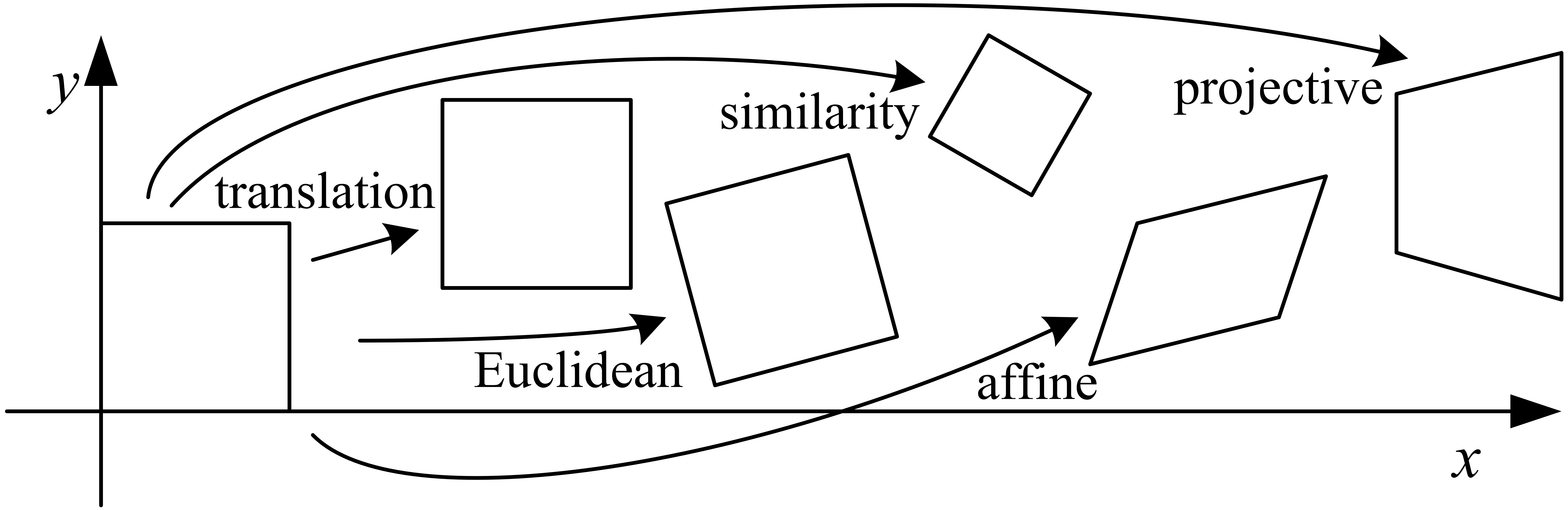
Euclidean, Similarity, Affine, Homography

— Robust Estimation and **RANSAC**

Estimating 2D transforms with noisy correspondences

2D **Transformations**

— We will look at a family that can be represented by 3x3 matrices



— This group represents perspective projections of **planar surfaces**

Affine Transformation

- Transformed points are a **linear function** of the input points

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix}$$

Affine Transformation

— Transformed points are a **linear function** of the input points

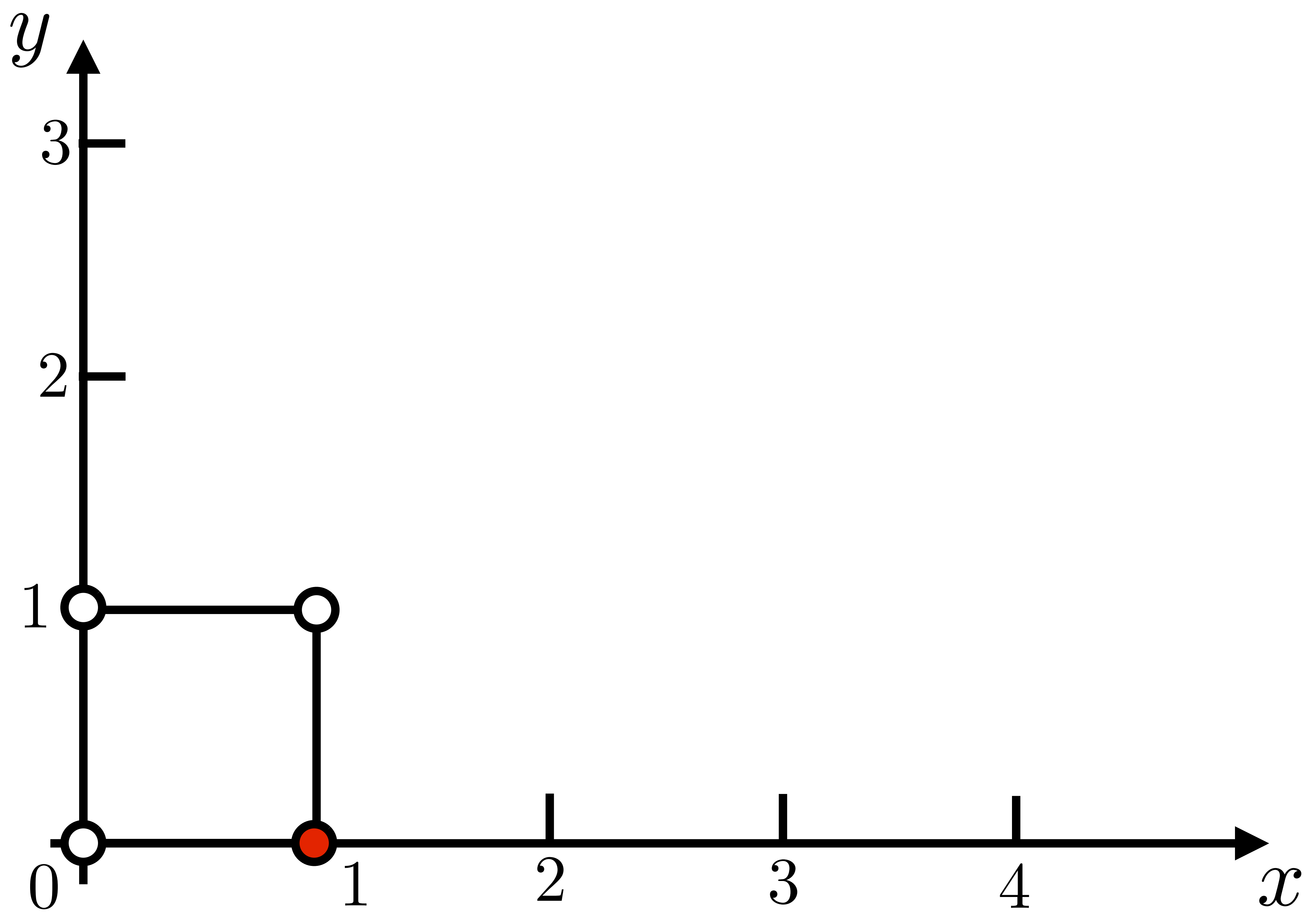
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix}$$

— This can be written as a **single matrix multiplication** using **homogeneous** coordinates

$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

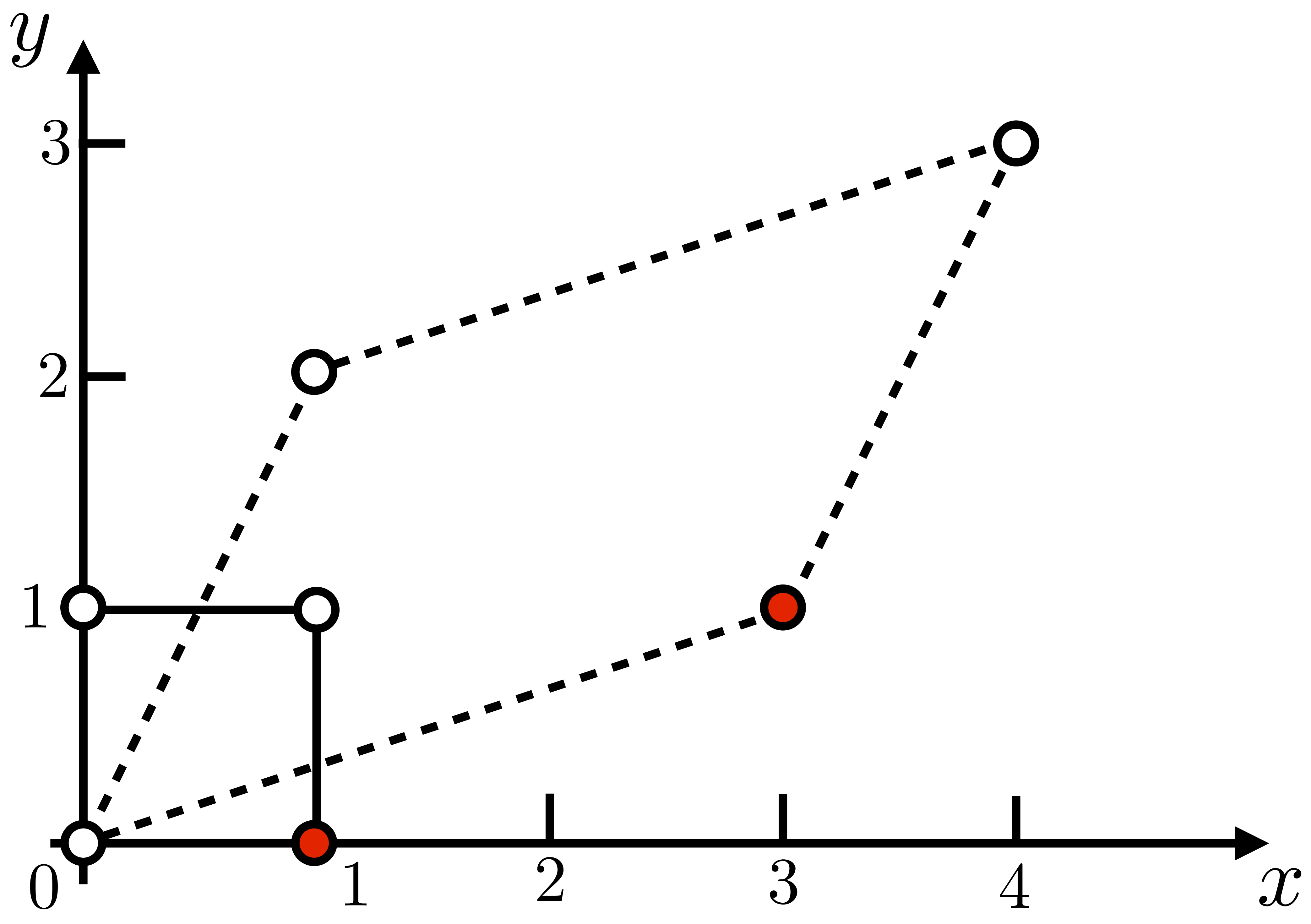
Linear Transformation

— Consider the action of the unit square under, sample transform $\begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



Linear Transformation

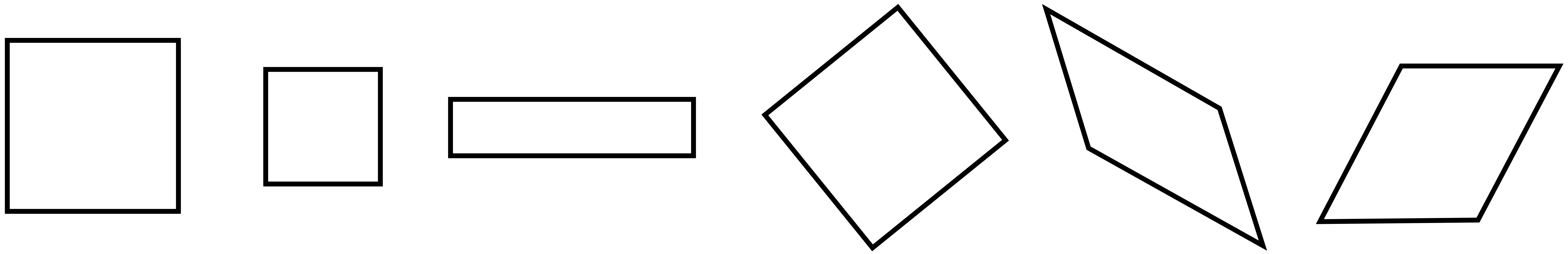
— Consider the action of the unit square under, sample transform $\begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



$$\begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3 & 4 \\ 0 & 2 & 1 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

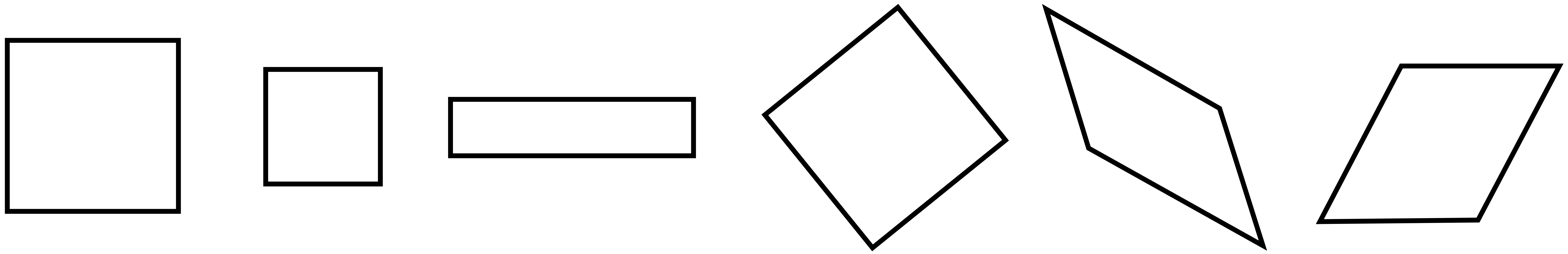
Transformation — Points — Transformed Points

Linear (or Affine) Transformations

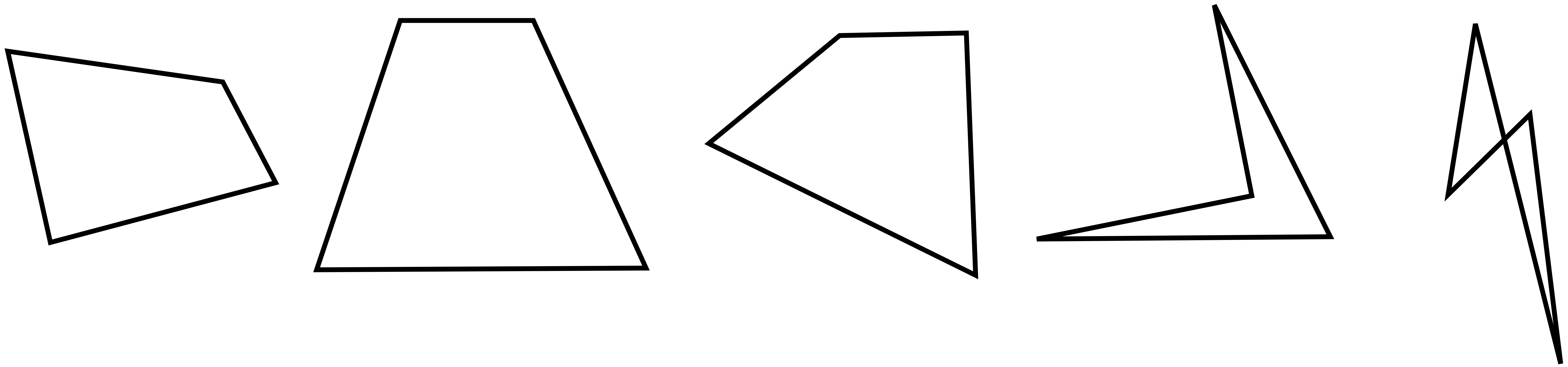


Translation, rotation, scale, shear (parallel lines preserved)

Linear (or Affine) Transformations



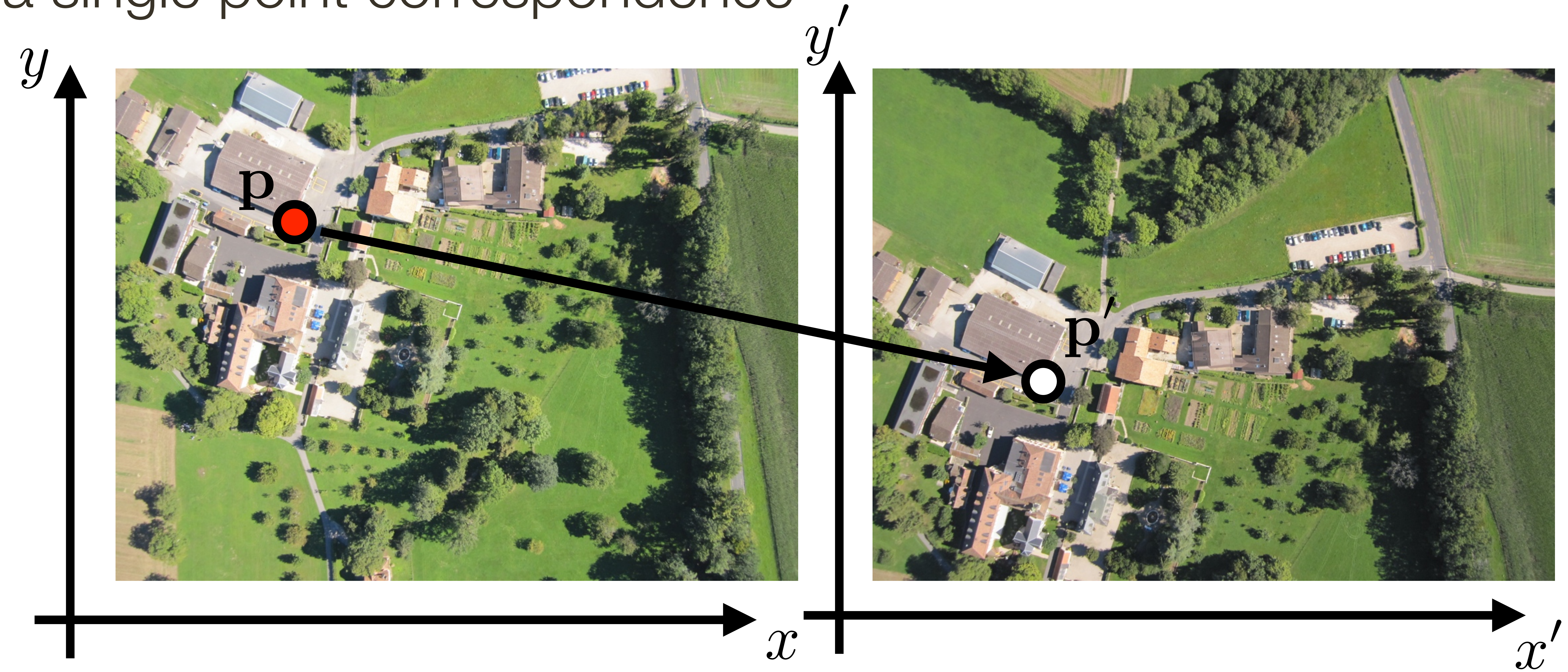
Translation, rotation, scale, shear (parallel lines preserved)



These transforms are not affine (parallel lines not preserved)

Linear (or Affine) Transformations

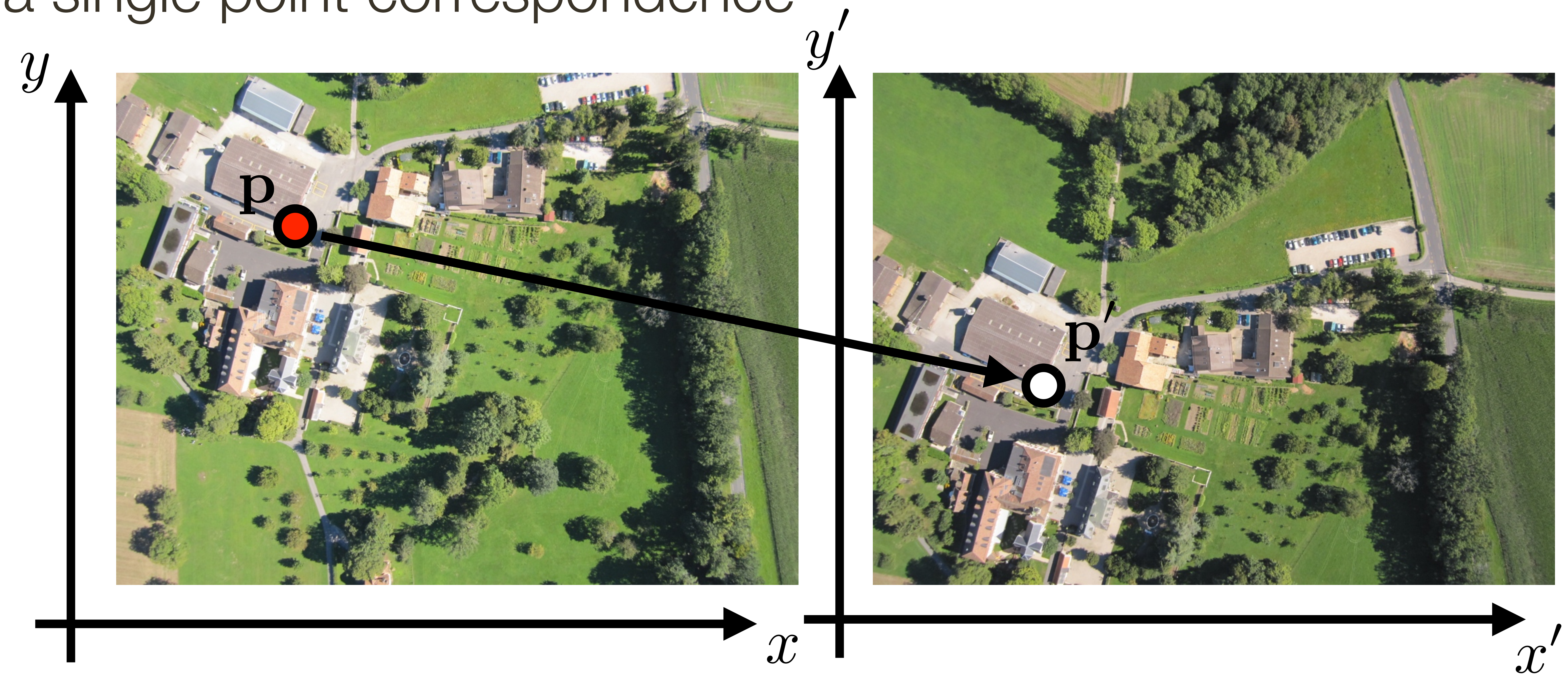
Consider a single point correspondence



$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Linear (or Affine) Transformations

Consider a single point correspondence



$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

How many points are needed to solve for **a**?

Computing Affine Transform

Lets compute an affine transform from correspondences:

$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Computing Affine Transform

Lets compute an affine transform from correspondences:

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Re-arrange unknowns into a vector

$$\begin{bmatrix} x_1' \\ y_1' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} 0 & x_1 \\ 0 & y_1 \\ 0 & 1 \\ x_2 & 0 \\ y_2 & 0 \\ 1 & 0 \end{bmatrix}$$

Computing Affine Transform

Linear system in the unknown parameters **a**

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix}$$

Of the form

$$\mathbf{Ma} = \mathbf{y}$$

Computing Affine Transform

Linear system in the unknown parameters **a**

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix}$$

Of the form

$$\mathbf{Ma} = \mathbf{y}$$

Solve for **a** using Gaussian Elimination

Computing Affine Transform

Once we solve for a transform, we can now map any other points between the two images … or resample one image in the coordinate system of the other

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Computing Affine Transform

Once we solve for a transform, we can now map any <u>other points</u> between the two images … or resample one image in the coordinate system of the other

This allows us to "stitch" the two images

Linear Transformations

Other linear transforms are special cases of **affine** transform:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Linear Transformations

Other linear transforms are special cases of **affine** transform:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

e.g.,

$$\begin{bmatrix} 1 & 0 & a_{13} \\ 0 & 1 & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

translation transform

Linear Transformations

Other linear transforms are special cases of **affine** transform:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

e.g.,

$$\begin{bmatrix} \cos\theta & \sin\theta & a_{13} \\ -\sin\theta & \cos\theta & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

euclidian transform

Linear Transformations

Other linear transforms are special cases of **affine** transform:

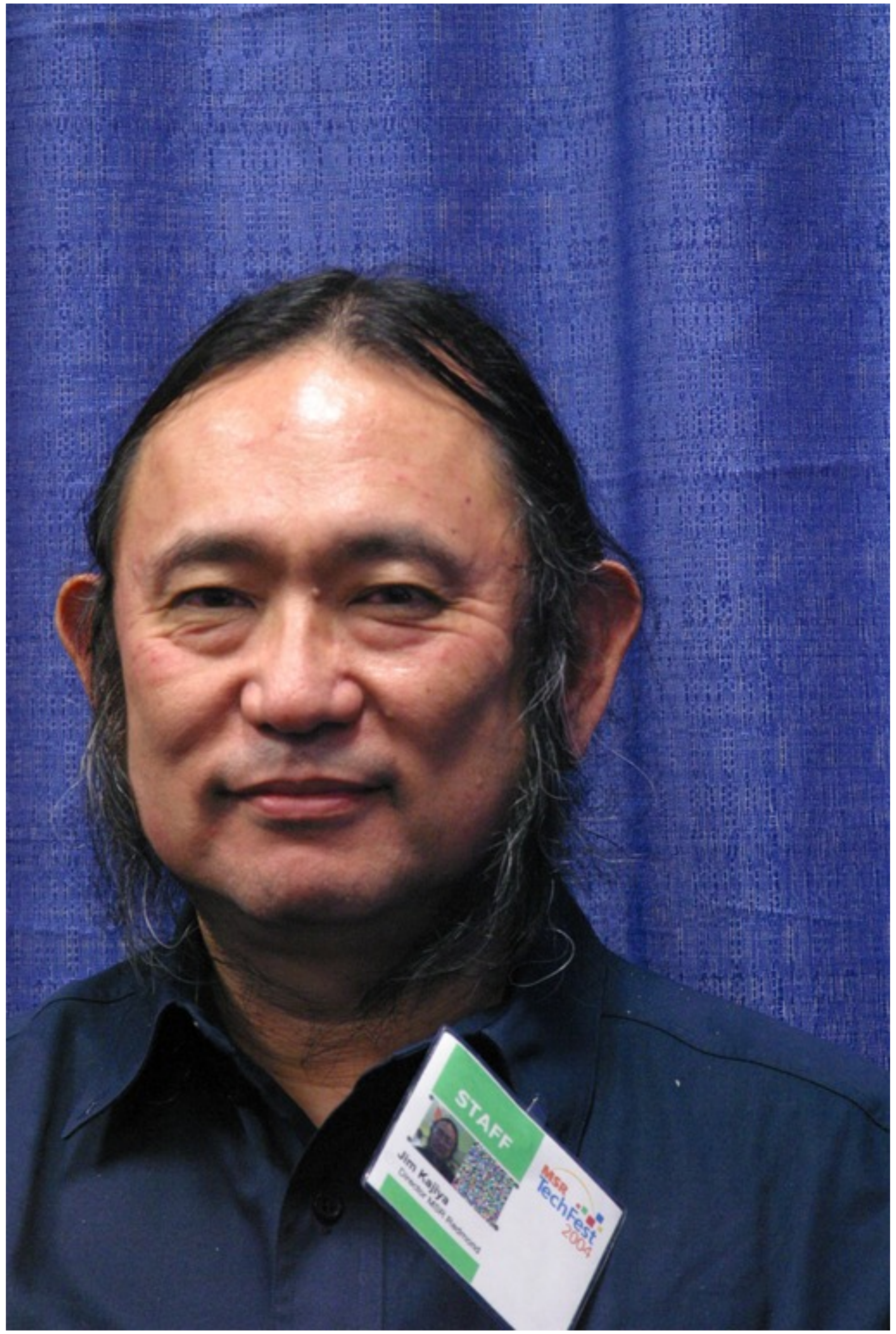
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

e.g.,

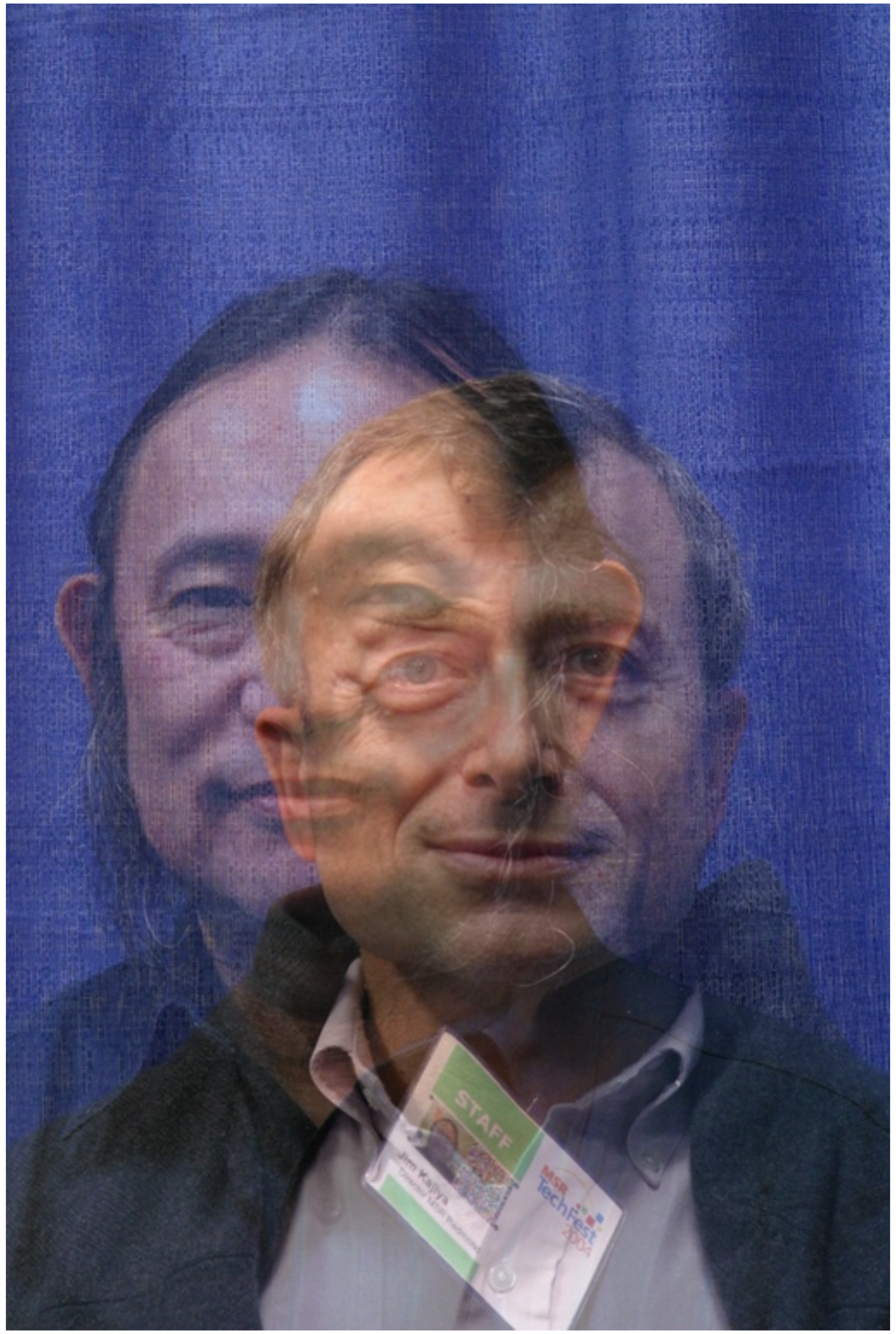
$$\begin{bmatrix} s\cos\theta & s\sin\theta & a_{13} \\ -s\sin\theta & s\cos\theta & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

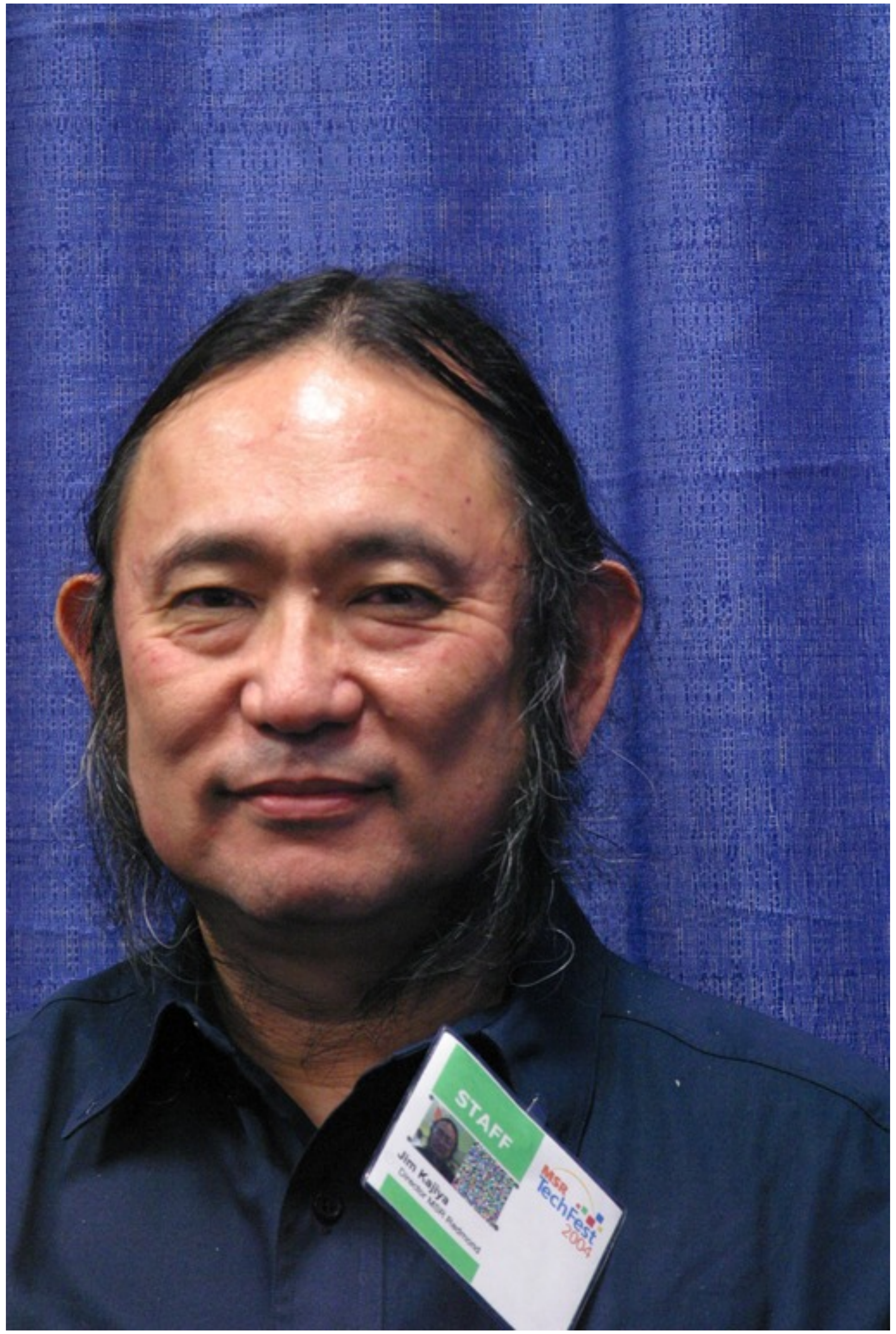
similarity transform

Face Alignment

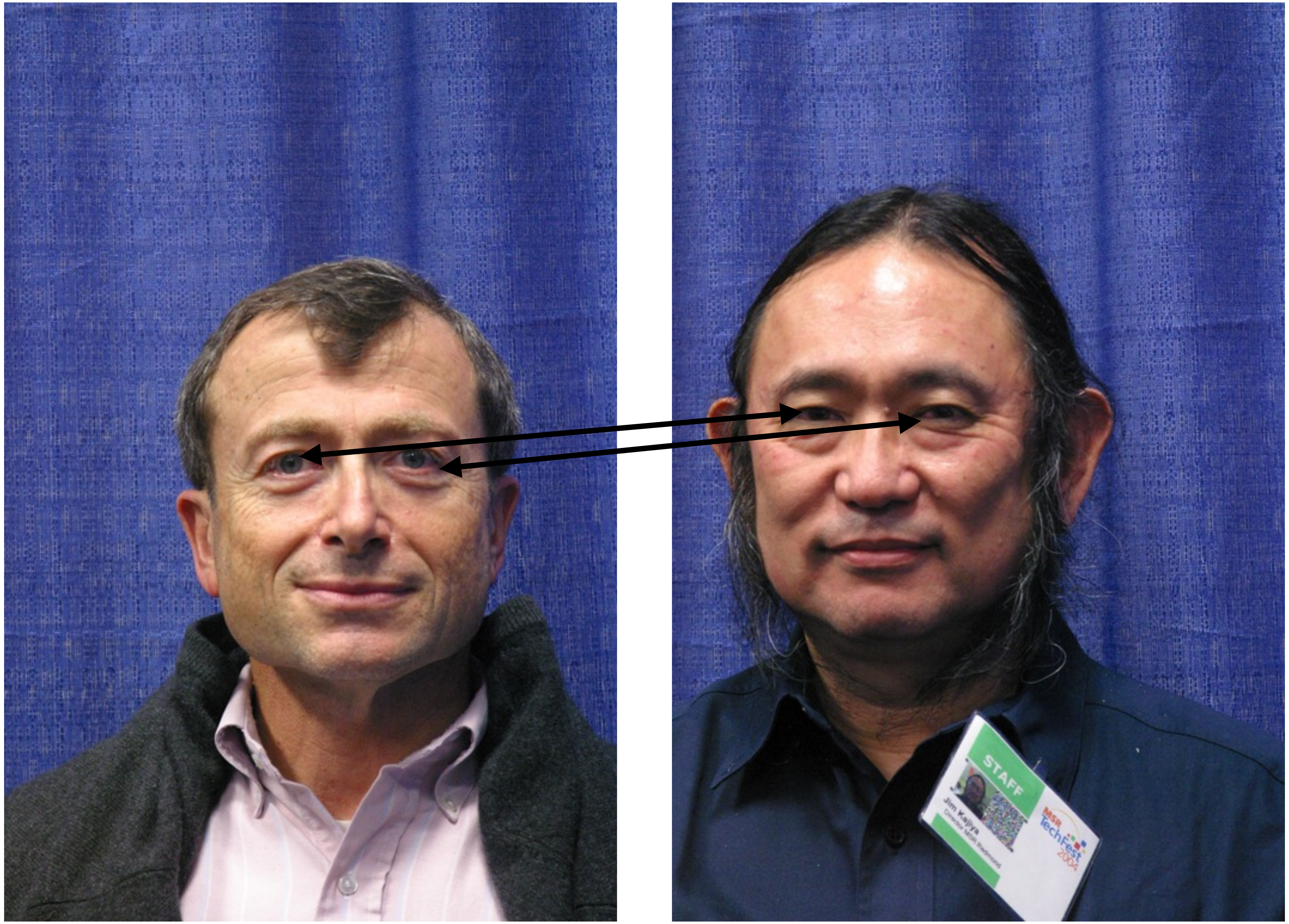


Face Alignment

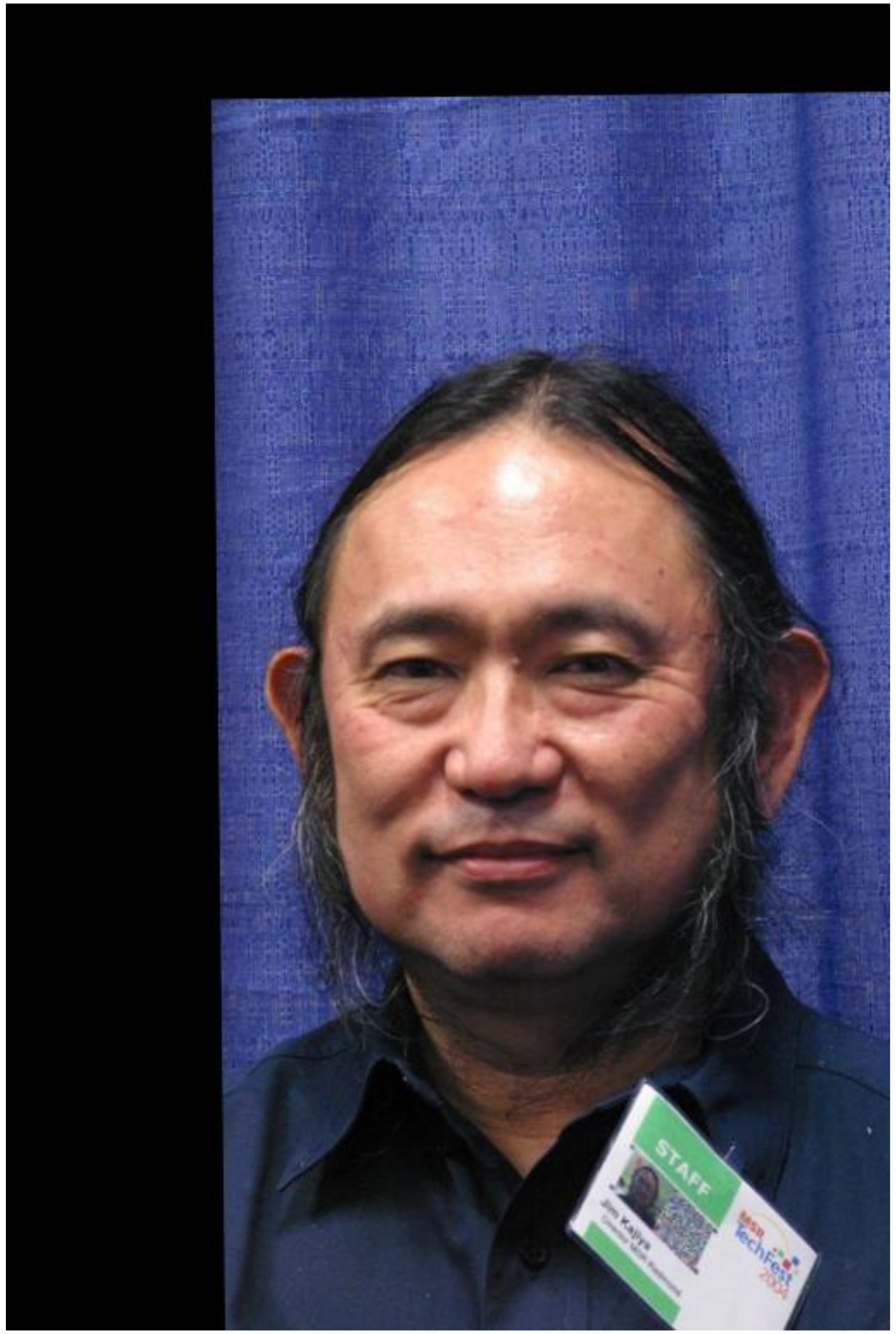


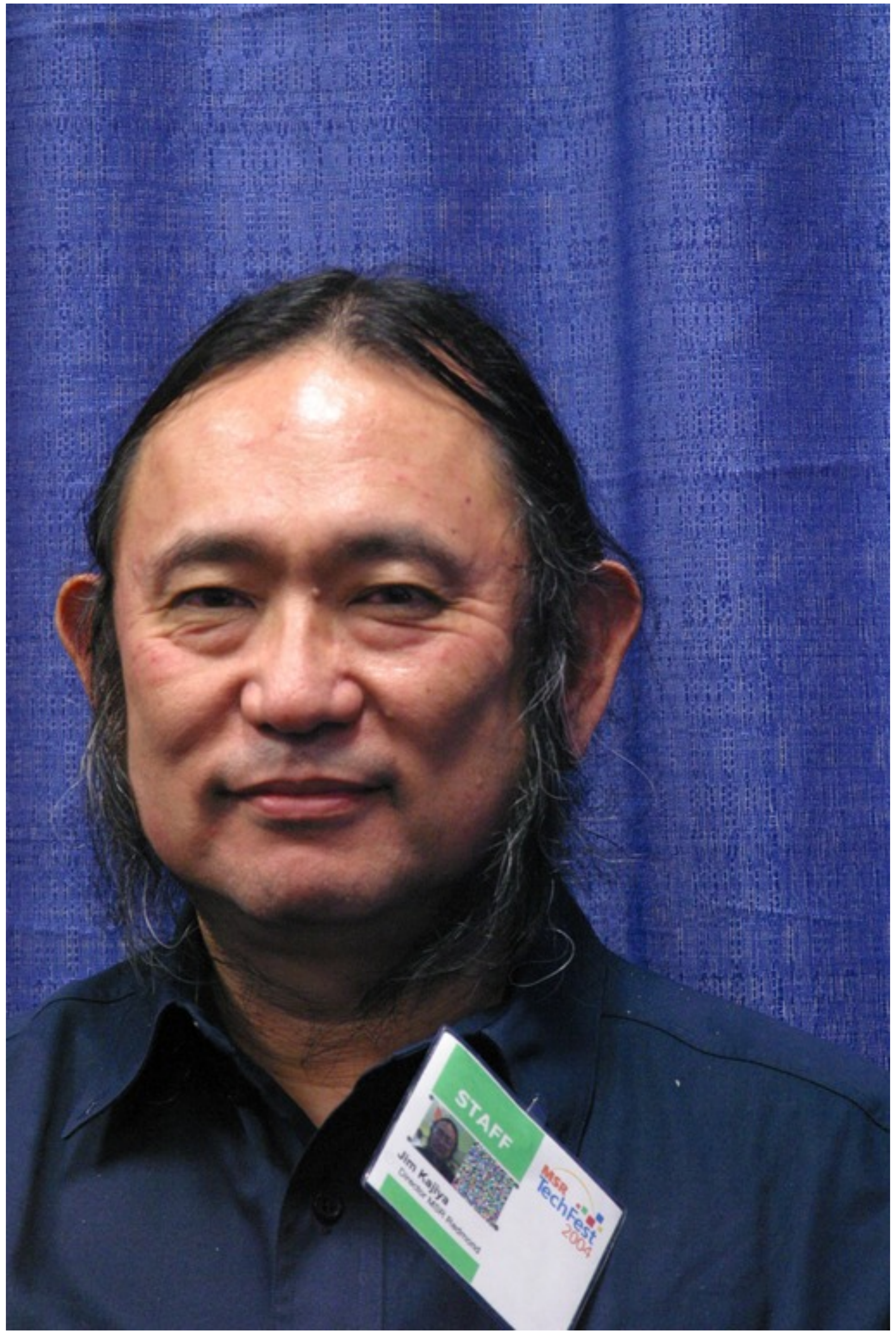


Face Alignment

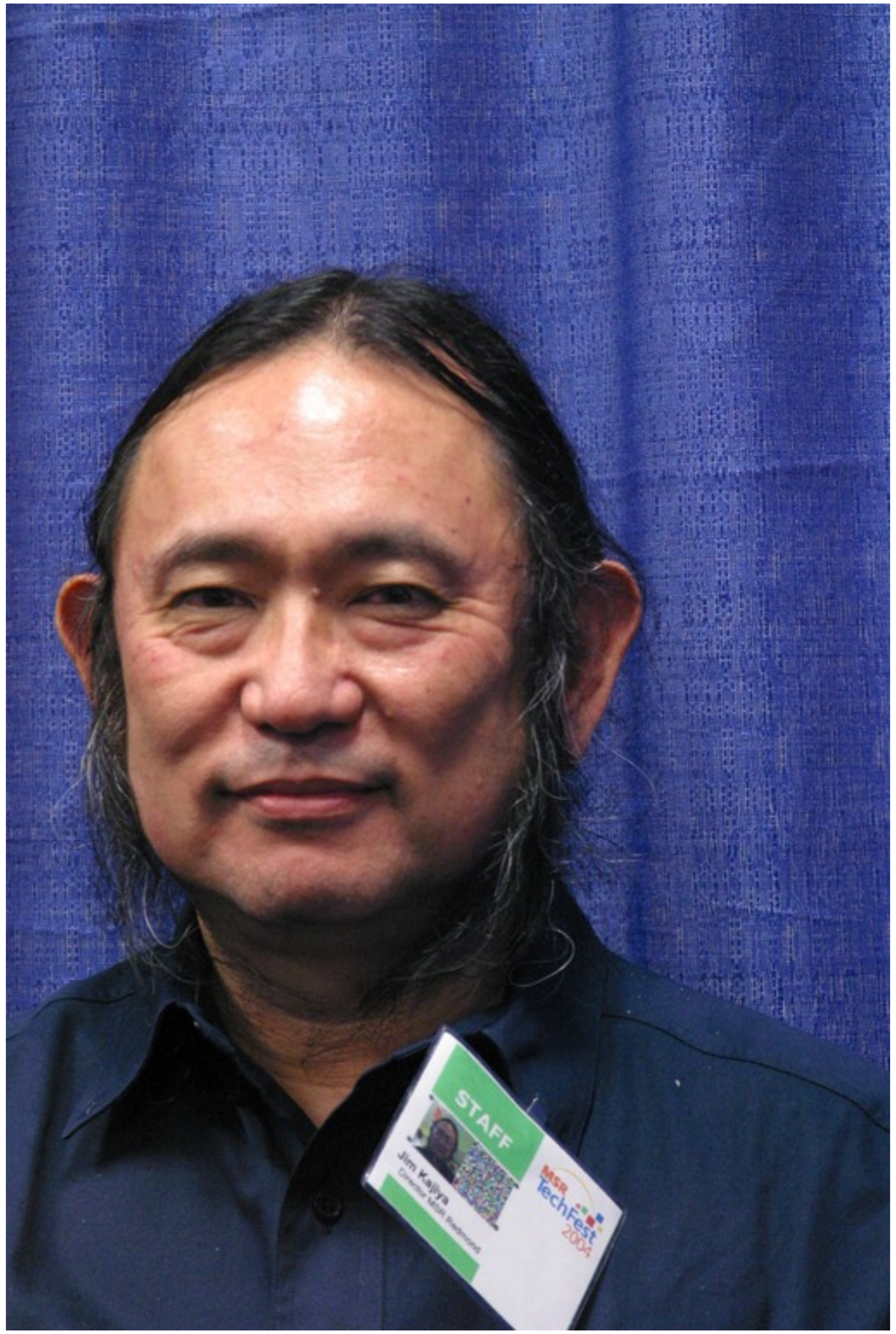


Face Alignment

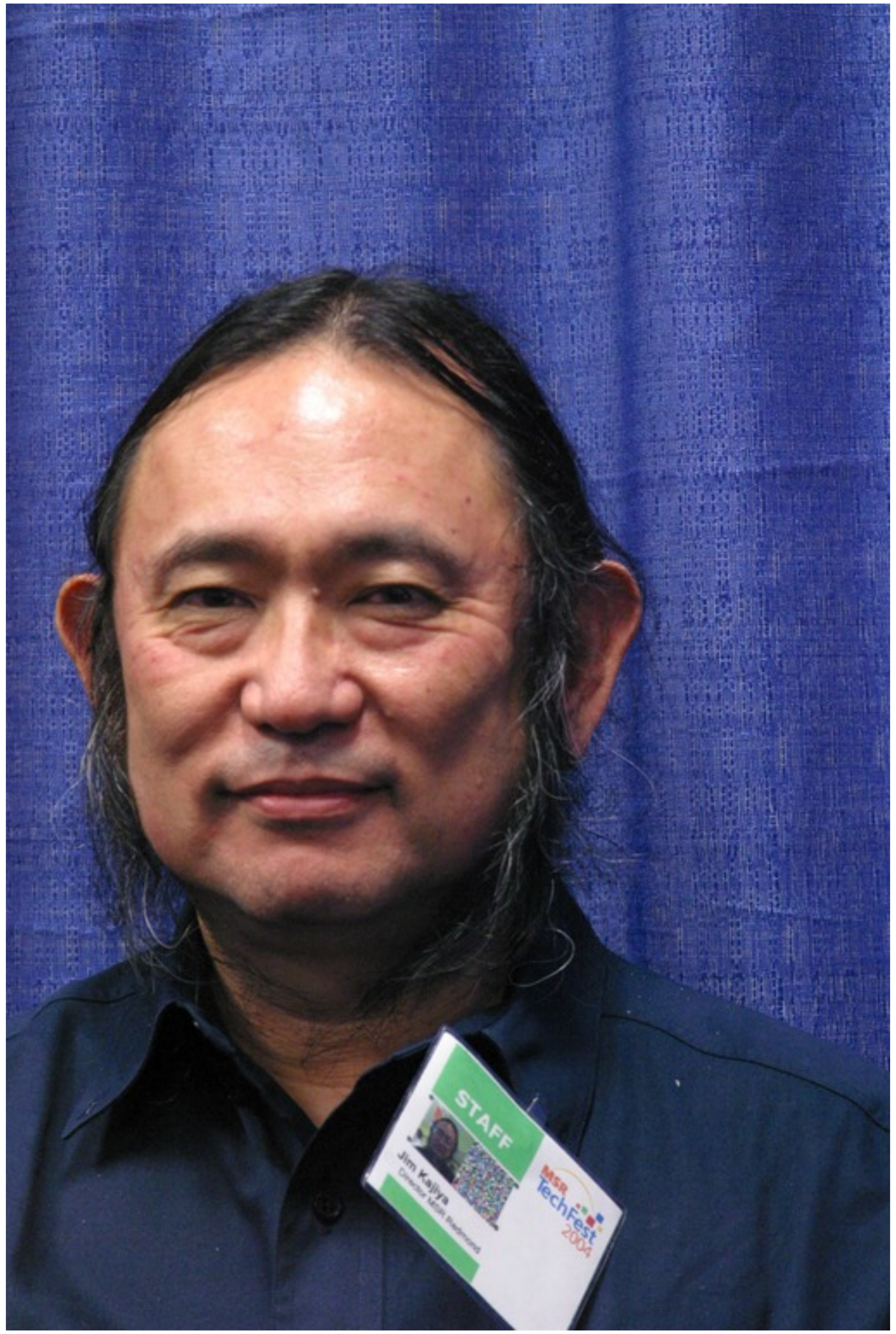




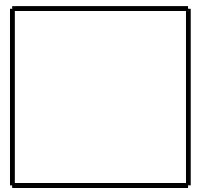
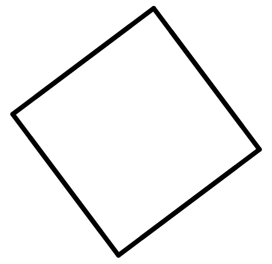
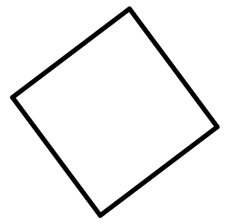
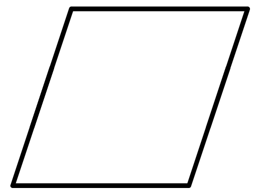
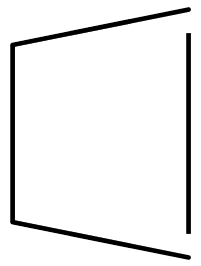
Face Alignment



Face Alignment

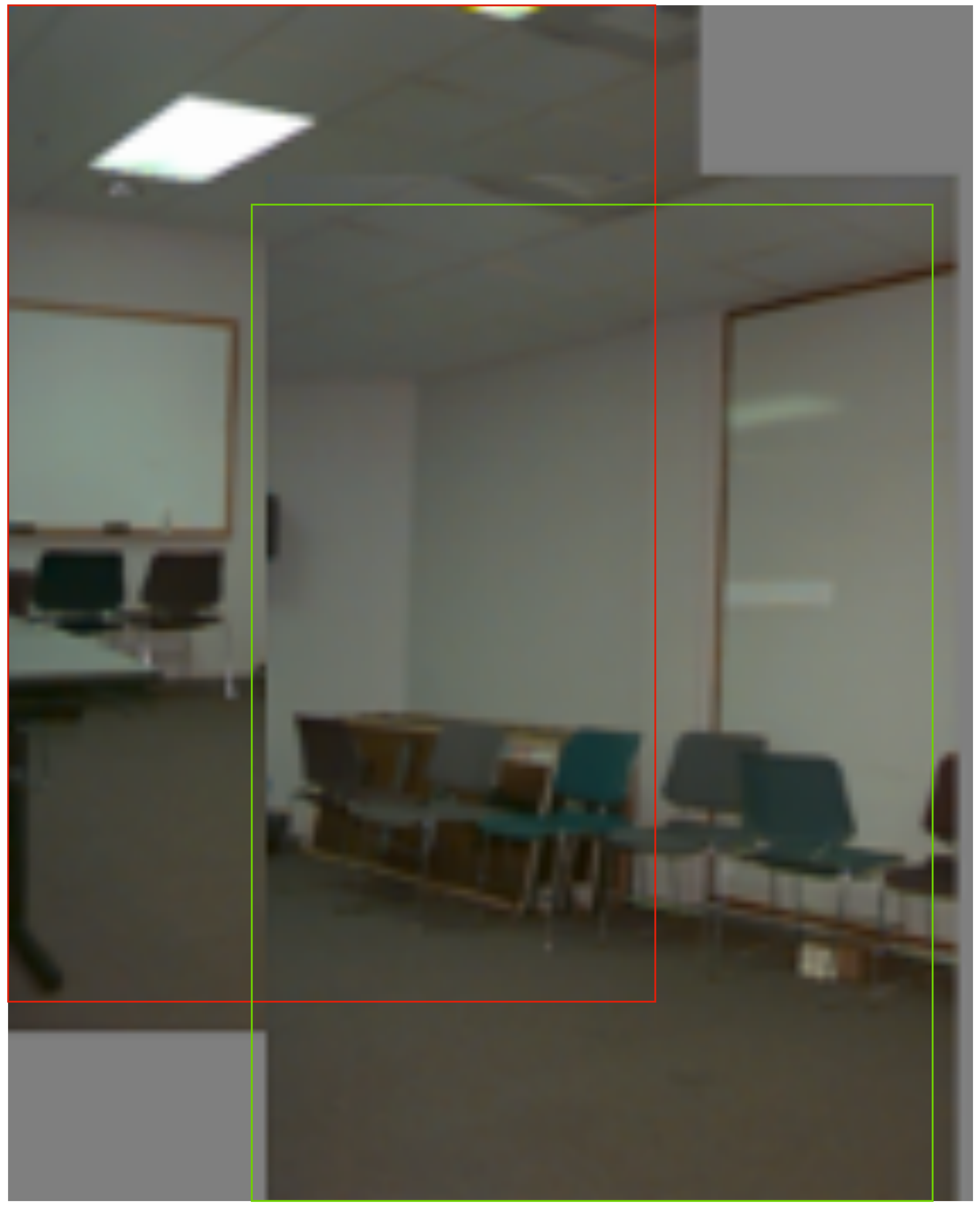


2D **Transformations**

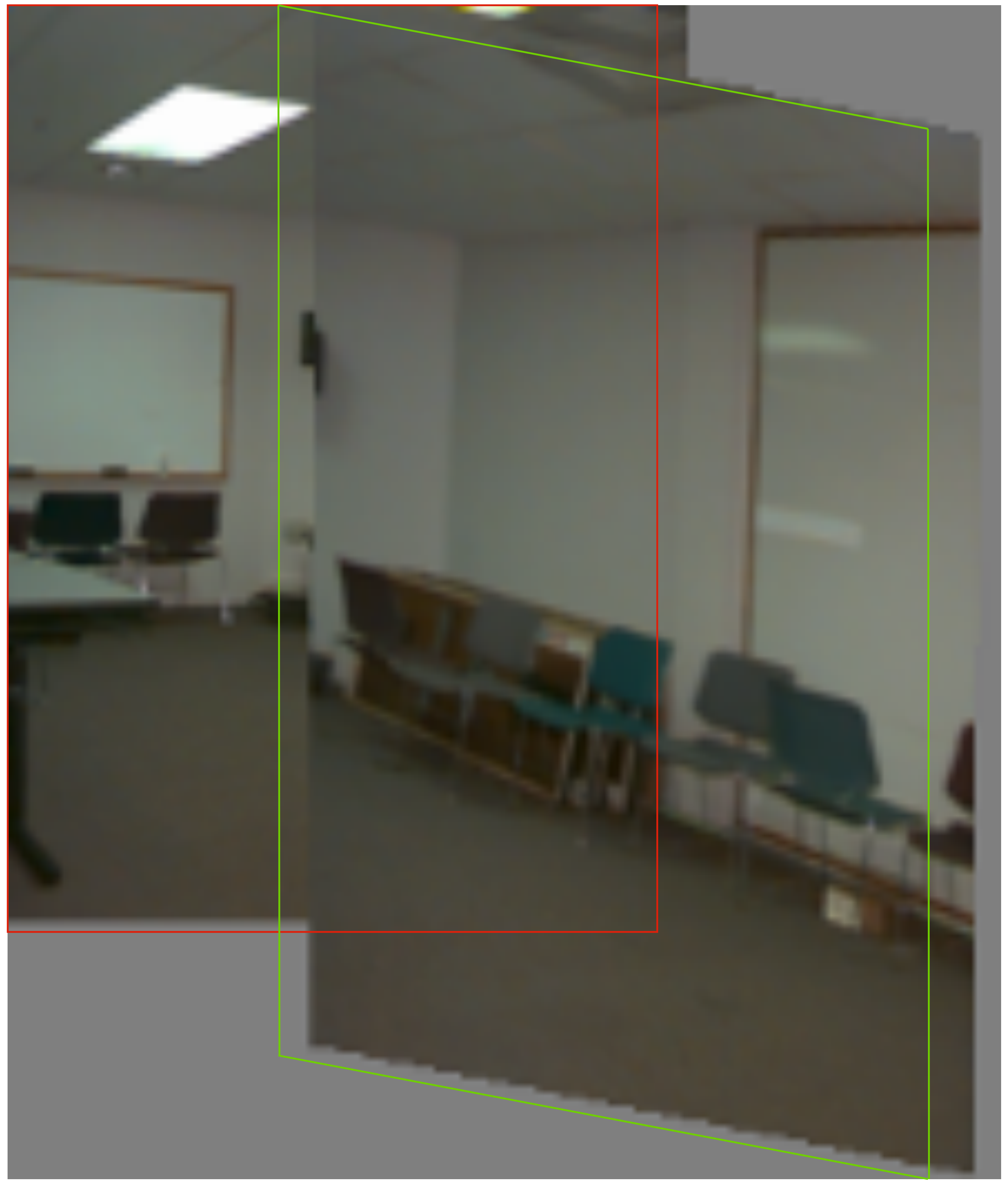
Transformation	**Matrix**	**# DoF**	**Preserves**	**Icon**
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

Example: Warping with Different Transformations

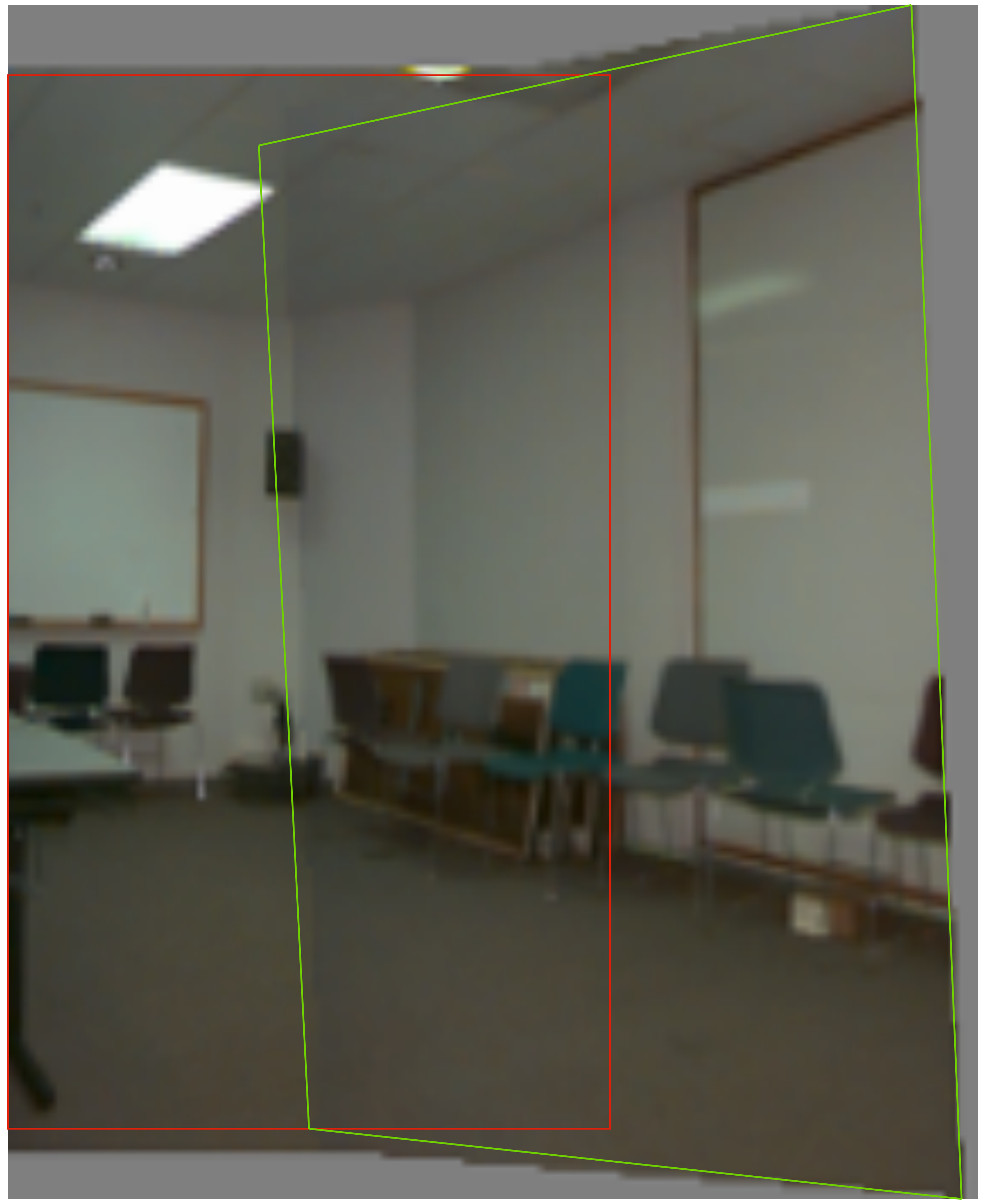
Translation



Affine



Projective
(homography)



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Aside: We can use homographies when …

1.… the scene is planar; or

2.… the scene is very far or has small (relative) depth variation → scene is approximately planar

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Aside: We can use homographies when …

3.… the scene is captured under camera rotation only (no translation or pose change)

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Projective Transformation

General 3x3 matrix transformation

$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Projective Transformation

General 3x3 matrix transformation

$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Lets try an example:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix}$$

Transformation Points Transformed Points

Projective Transformation

General 3x3 matrix transformation

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Lets try an example:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix}$$

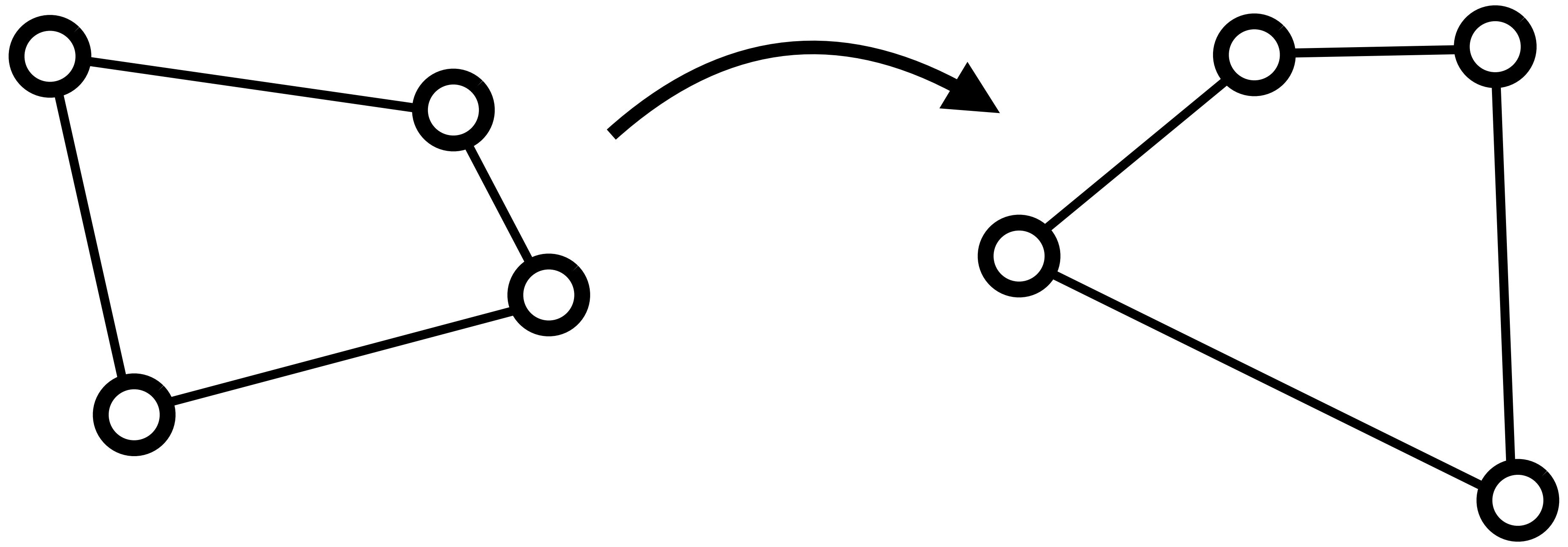
Transformation   Points   Transformed Points

Divide by the last row: $\begin{bmatrix} 0 & 0 & 1 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Compute **H** from Correspondences

Each match gives 2 equations to solve for **8** parameters

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$



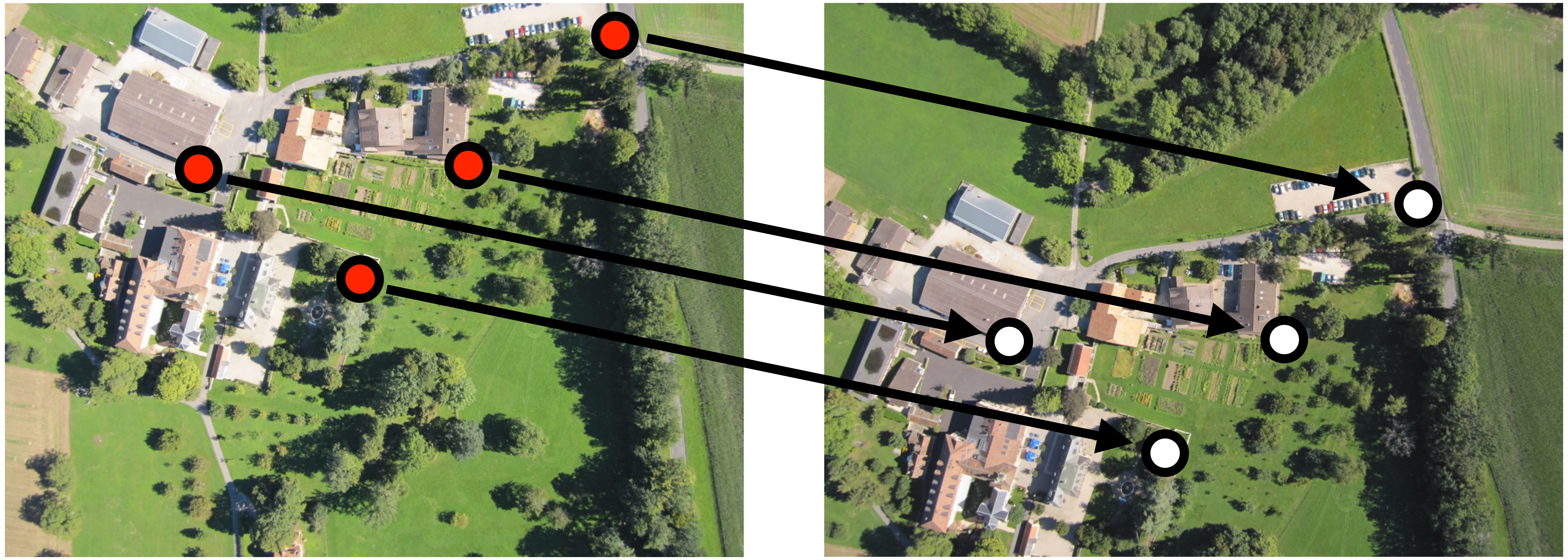
→ 4 correspondences to solve for **H** matrix

Solution uses **Singular Value Decomposition** (SVD)

In **Assignment 4** you can compute this using `cv2.findHomography`

Image **Alignment**

Find **corresponding** (matching) points between the image



$$\mathbf{u} = \mathbf{H}\mathbf{x}$$

2 points for Similarity

3 for Affine

4 for Homography

Image **Alignment**

In practice we have many noisy correspondences + **outliers**

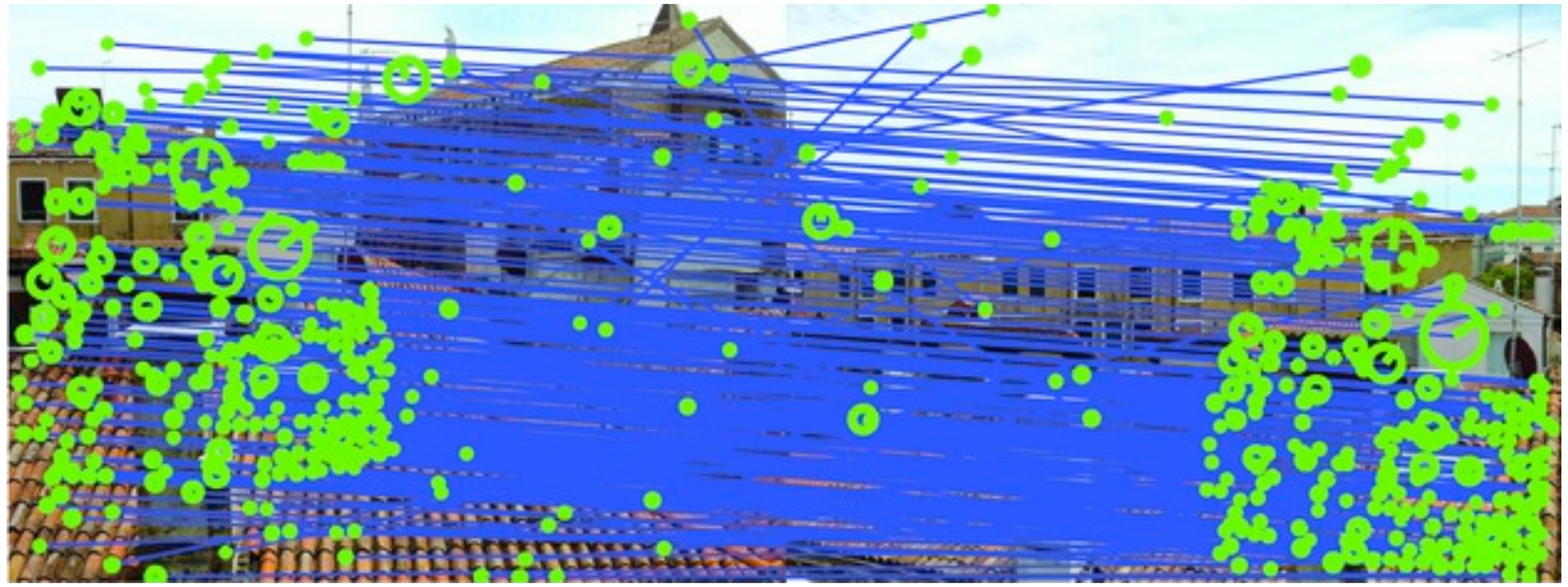


Image **Alignment**

In practice we have many noisy correspondences + **outliers**

e.g., for an affine transform we have a linear system in the parameters **a**:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \\ & & & \vdots & & \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ \vdots \end{bmatrix}$$

It is **overconstrained** (more equations than unknowns) and subject to **outliers** (some rows are completely wrong)

Image **Alignment**

In practice we have many noisy correspondences + **outliers**

e.g., for an affine transform we have a linear system in the parameters **a**:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \\ & & & \vdots & & \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ \vdots \end{bmatrix}$$

It is **overconstrained** (more equations than unknowns) and subject to **outliers** (some rows are completely wrong)

Let's deal with these problems in a simpler context …

Fitting a Model to Noisy Data

Suppose we are **fitting a line** to a dataset that consists of 50% outliers

We can fit a line using two points

If we draw pairs of points uniformly at random, what fraction of pairs will consist entirely of ‘good’ data points (inliers)?

Fitting a Model to Noisy Data

Suppose we are **fitting a line** to a dataset that consists of 50% outliers

We can fit a line using two points

— If we draw pairs of points uniformly at random, then about 1/4 of these pairs will consist entirely of 'good' data points (inliers)

— We can identify these good pairs by noticing that a large collection of other points lie close to the line fitted to the pair

— A better estimate of the line can be obtained by refitting the line to the points that lie close to the line

RANSAC (**RAN**dom **SA**mple **C**onsensus)

1. Randomly choose minimal subset of data points necessary to fit model (a **sample**)

2. Points within some distance threshold, t, of model are a **consensus set**. Size of consensus set is model's **support**

3. Repeat for N samples; model with biggest support is most robust fit
   - Points within distance t of best model are inliers
   - Fit final model to all inliers

Slide Credit: Christopher Rasmussen

RANSAC (**RAN**dom **SA**mple **C**onsensus)

1. Randomly choose minimal subset of data points necessary to fit model (a **sample**)

2. Points within some distance threshold, t, of model are a **consensus set**. Size of consensus set is model's **support**

3. Repeat for N samples; model with biggest support is most robust fit
   - Points within distance t of best model are inliers
   - Fit final model to all inliers

RANSAC is very useful for variety of applications

Slide Credit: Christopher Rasmussen

RANSAC (**RAN**dom **SA**mple **C**onsensus)

1. Randomly choose minimal subset of data points necessary to fit model (a **sample**)

   Fitting a Line: 2 points

2. Points within some distance threshold, t, of model are a **consensus set**. Size of consensus set is model's **support**

3. Repeat for N samples; model with biggest support is most robust fit
   — Points within distance t of best model are inliers
   — Fit final model to all inliers

Slide Credit: Christopher Rasmussen

Example 1: Fitting a Line

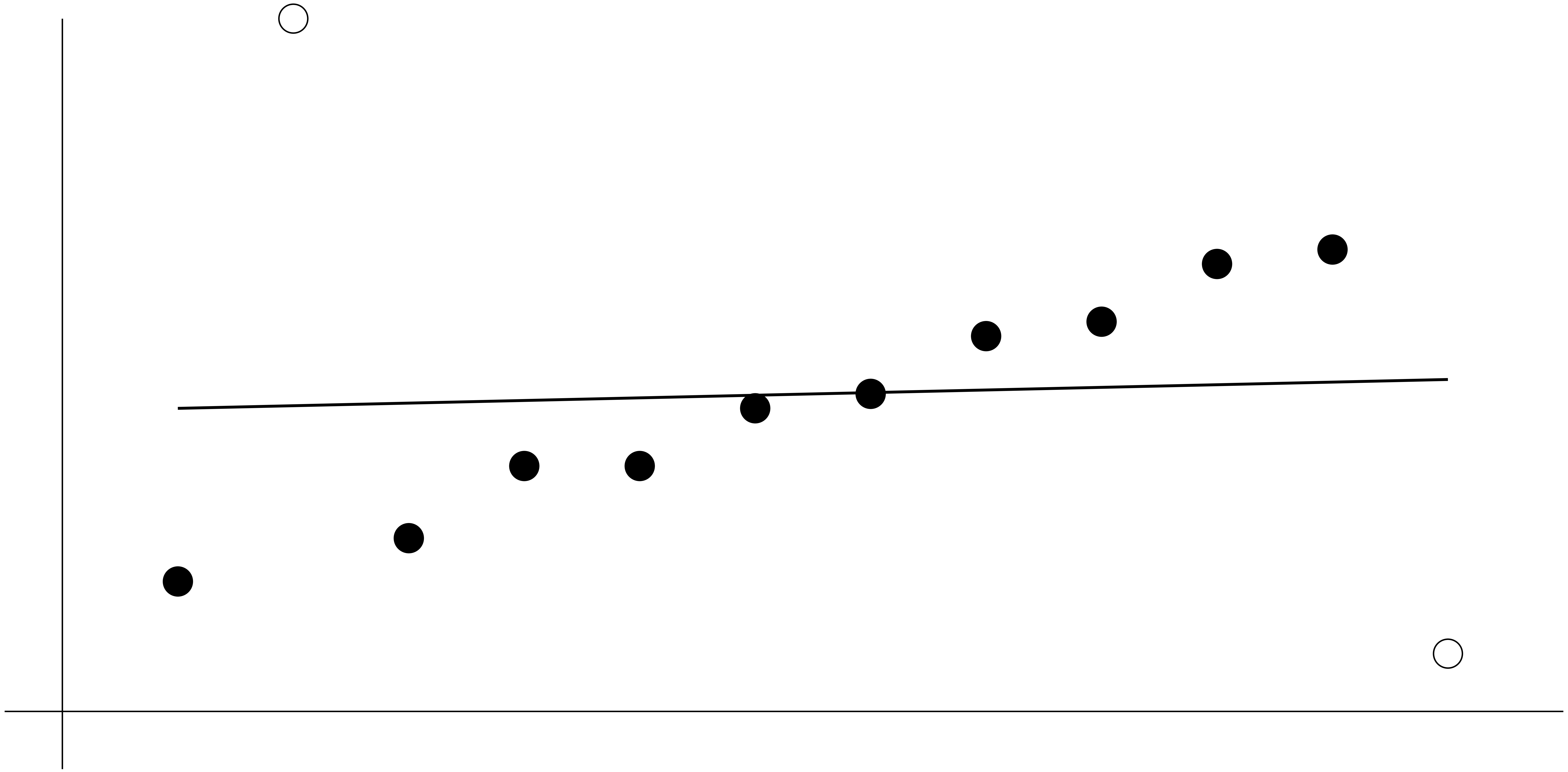


Figure Credit: Hartley & Zisserman

Example 1: Fitting a Line

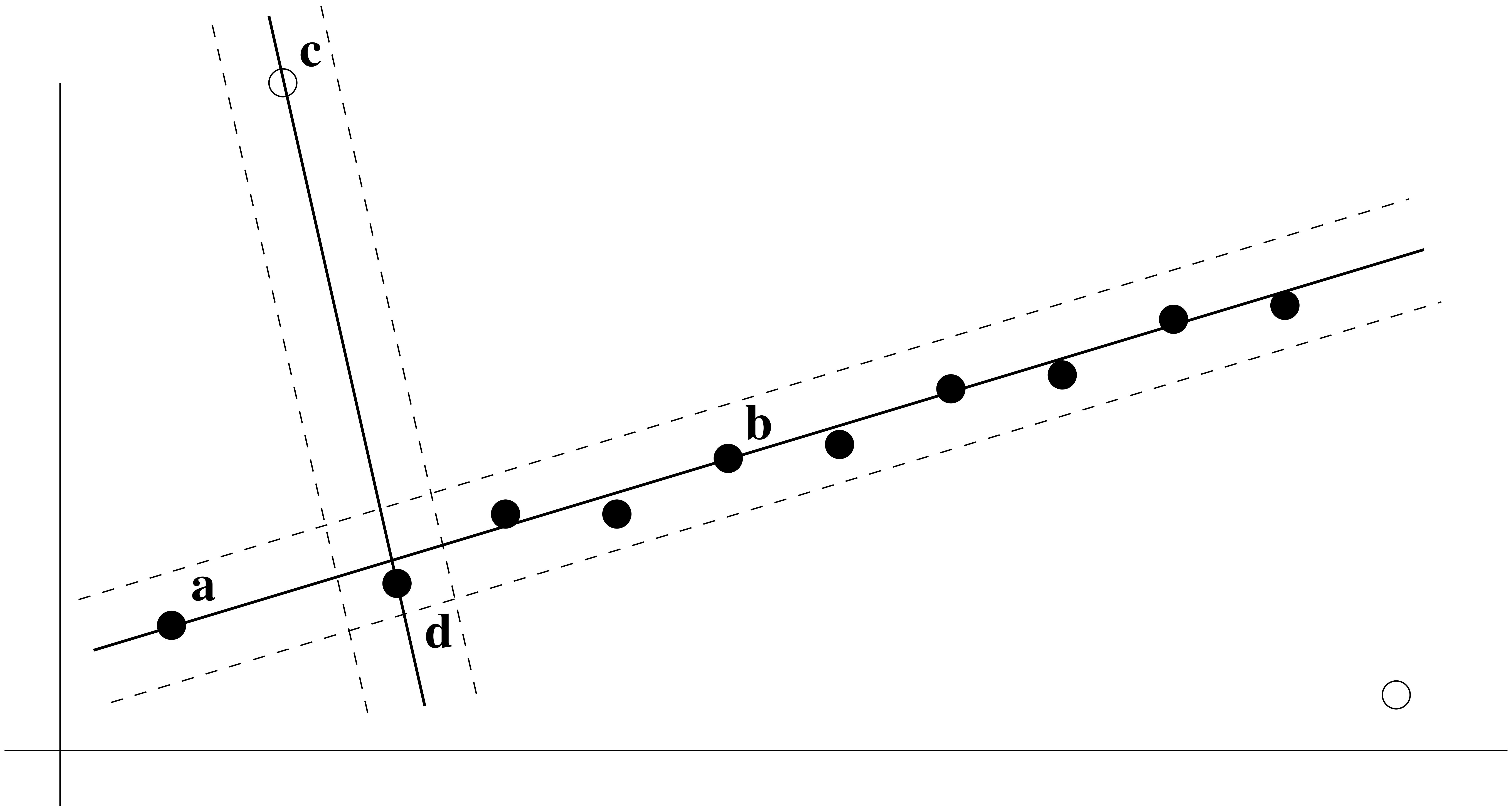


Figure Credit: Hartley & Zisserman

Example 1: Fitting a Line

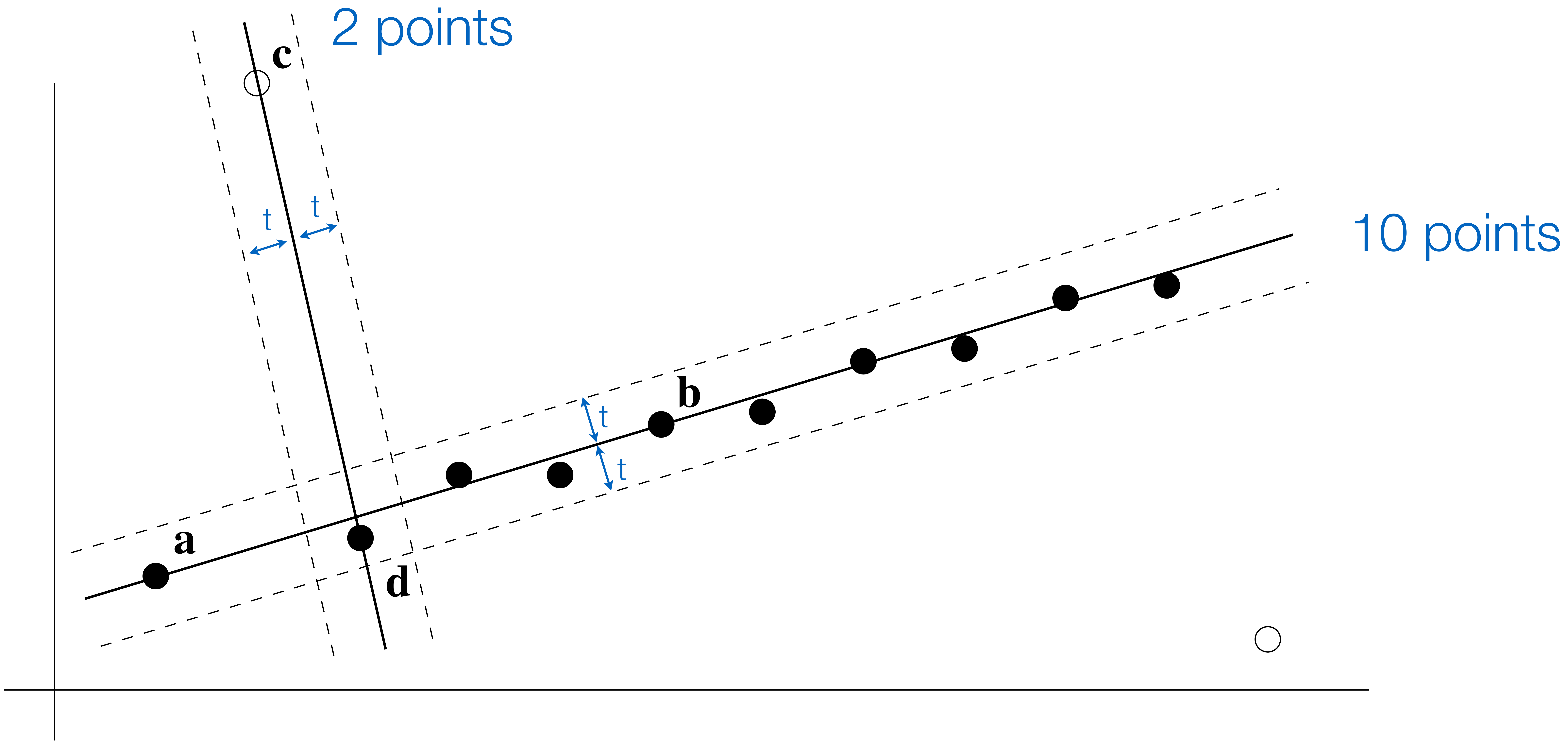


Figure Credit: Hartley & Zisserman

RANSAC: How many samples?

Let ω be the fraction of inliers (i.e., points on line)

Let n be the number of points needed to define hypothesis

$(n = 2$ for a line in the plane)

Suppose k samples are chosen

The probability that a single sample of n points is correct (all inliers) is

RANSAC: How many samples?

Let ω be the fraction of inliers (i.e., points on line)

Let n be the number of points needed to define hypothesis

($n = 2$ for a line in the plane)

Suppose k samples are chosen

The probability that a single sample of n points is correct (all inliers) is

$$\omega^n$$

The probability that all k samples fail is

RANSAC: How many samples?

Let ω be the fraction of inliers (i.e., points on line)

Let n be the number of points needed to define hypothesis
$(n = 2$ for a line in the plane)

Suppose k samples are chosen

The probability that a single sample of n points is correct (all inliers) is

$$\omega^n$$

The probability that all k samples fail is

$$(1 - \omega^n)^k$$

Choose k large enough (to keep this below a target failure rate)

RANSAC: *k* Samples Chosen (p = 0.99)

Sample size	Proportion of outliers						
n	**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

Figure Credit: Hartley & Zisserman

After RANSAC

RANSAC divides data into inliers and outliers and yields estimate computed from minimal set of inliers

Improve this initial estimate with estimation over all inliers (e.g., with standard least-squares minimization)

But this may change inliers, so alternate fitting with re-classification as inlier/outlier

Example 2: Fitting a Line

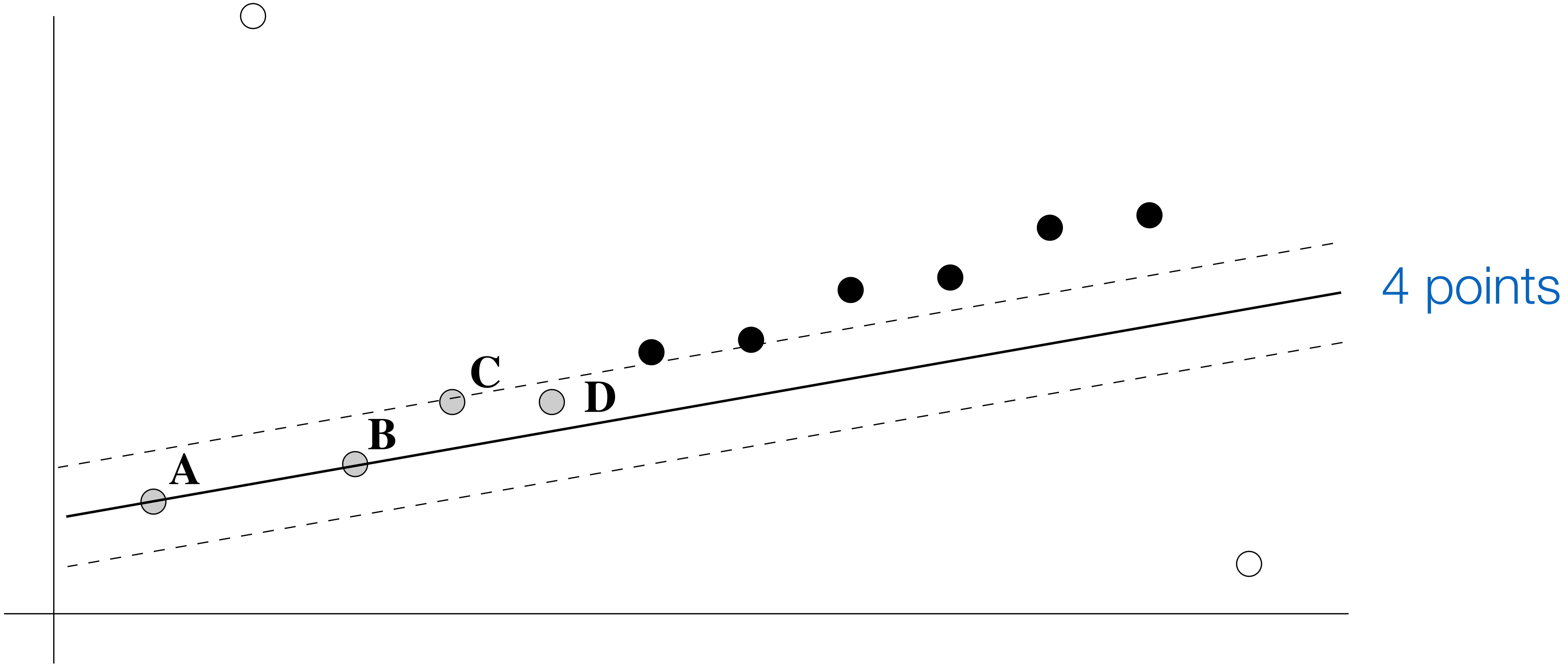


Figure Credit: Hartley & Zisserman

Example 2: Fitting a Line

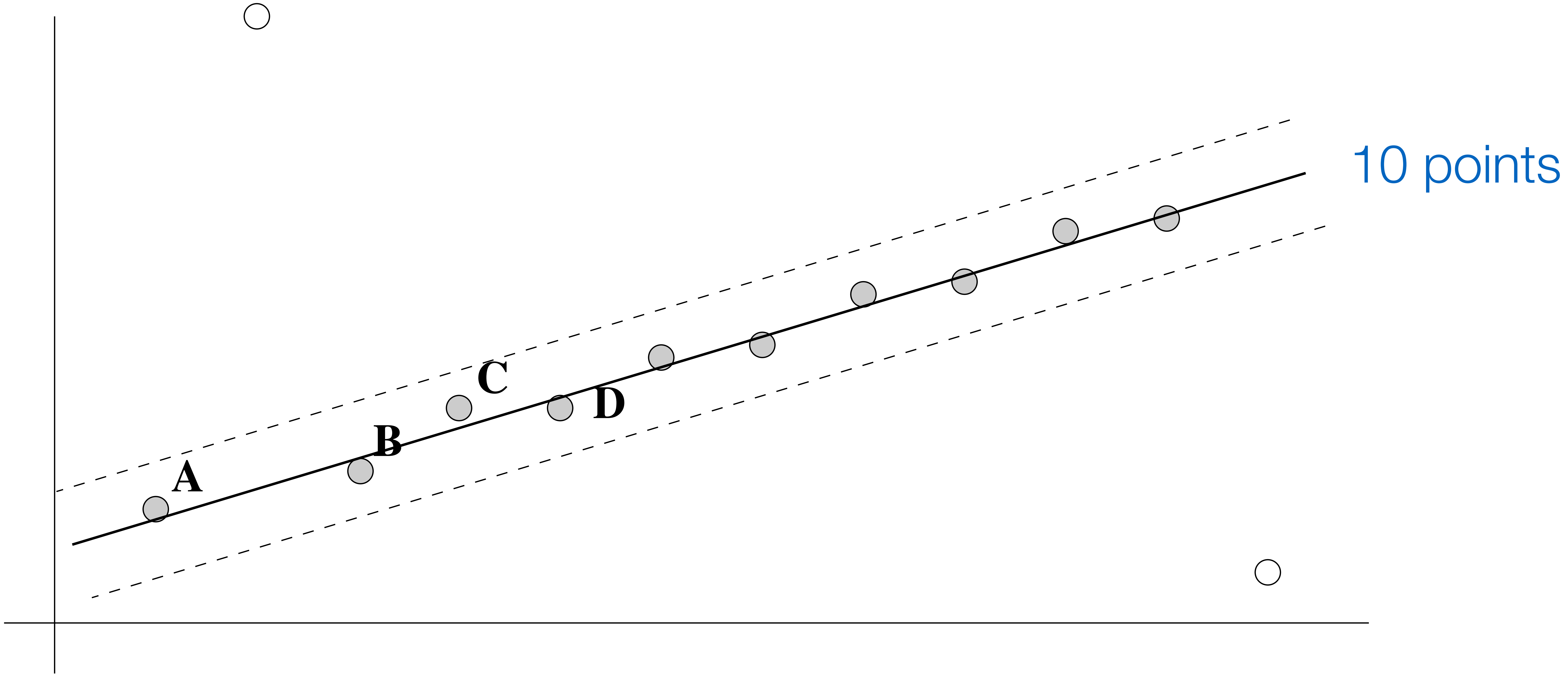


Figure Credit: Hartley & Zisserman

Image **Alignment + RANSAC**

In practice we have many noisy correspondences + **outliers**

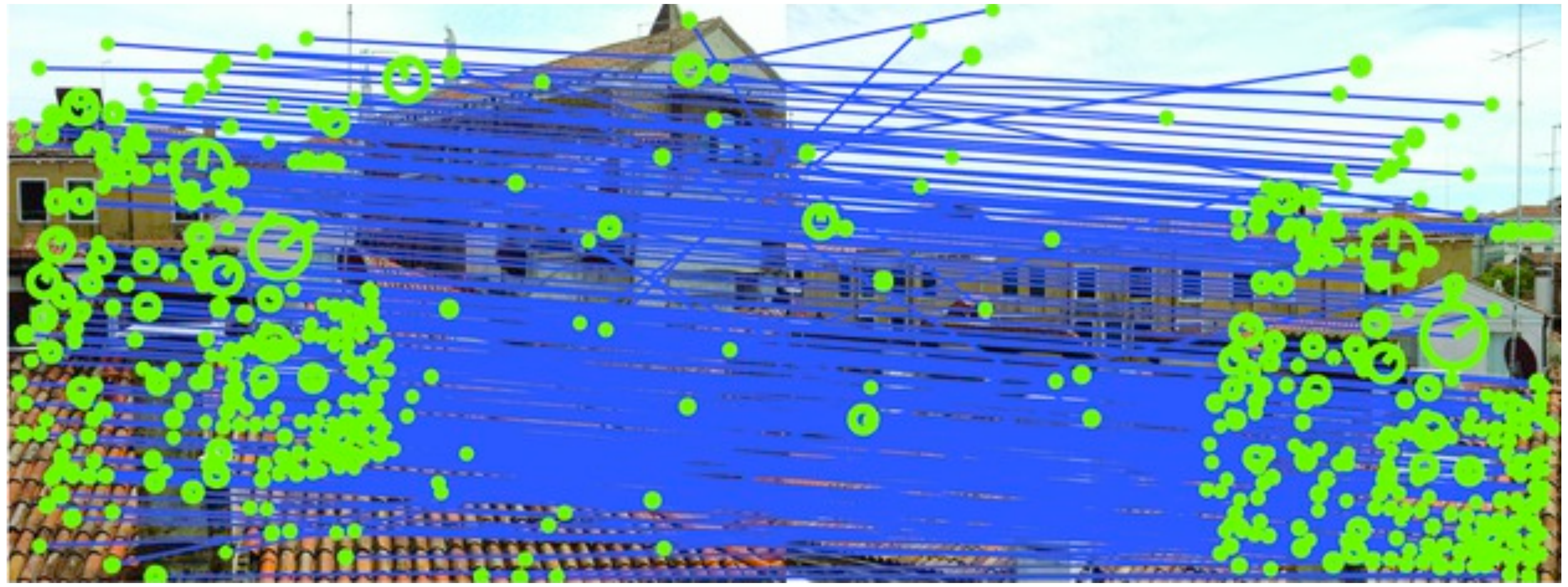


Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

4 inliers (red, yellow, orange, brown),

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

4 outliers (blue, light blue, purple, pink)

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

4 inliers (red, yellow, orange, brown),
4 outliers (blue, light blue, purple, pink)

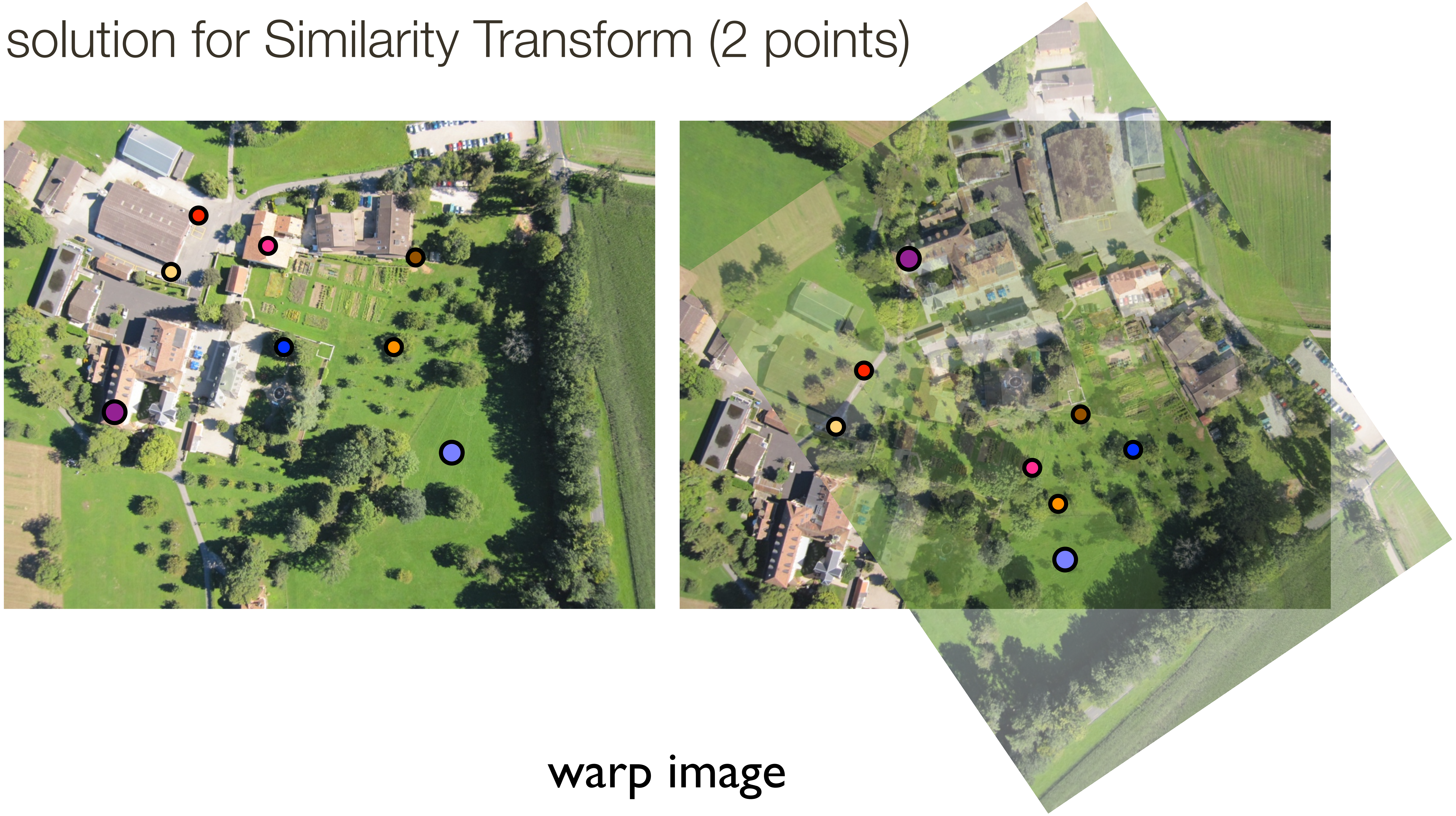
Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

choose light blue, purple

Image **Alignment + RANSAC**

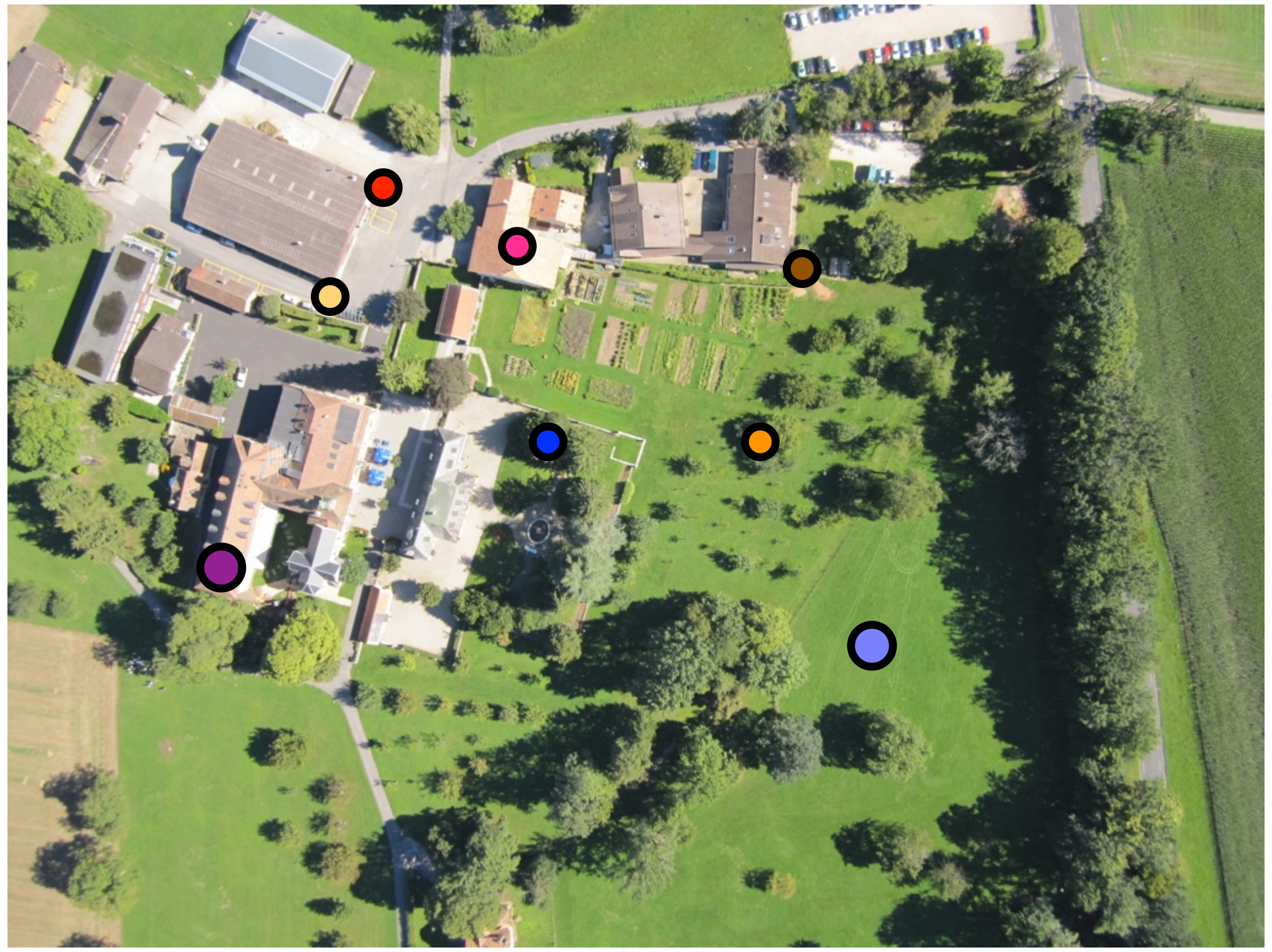
RANSAC solution for Similarity Transform (2 points)



warp image

Image **Alignment + RANSAC**

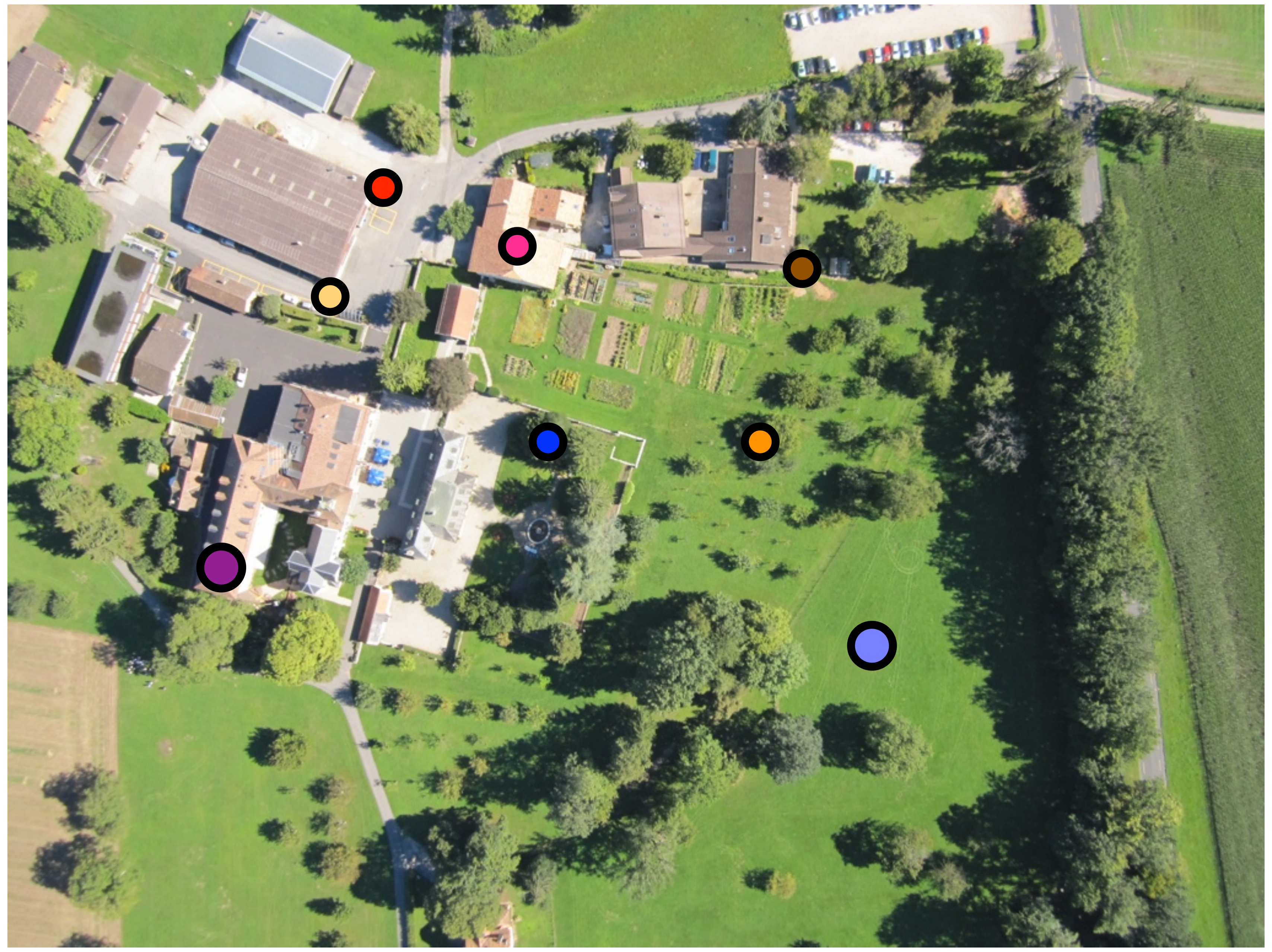
RANSAC solution for Similarity Transform (2 points)



check match distances

Image **Alignment + RANSAC**

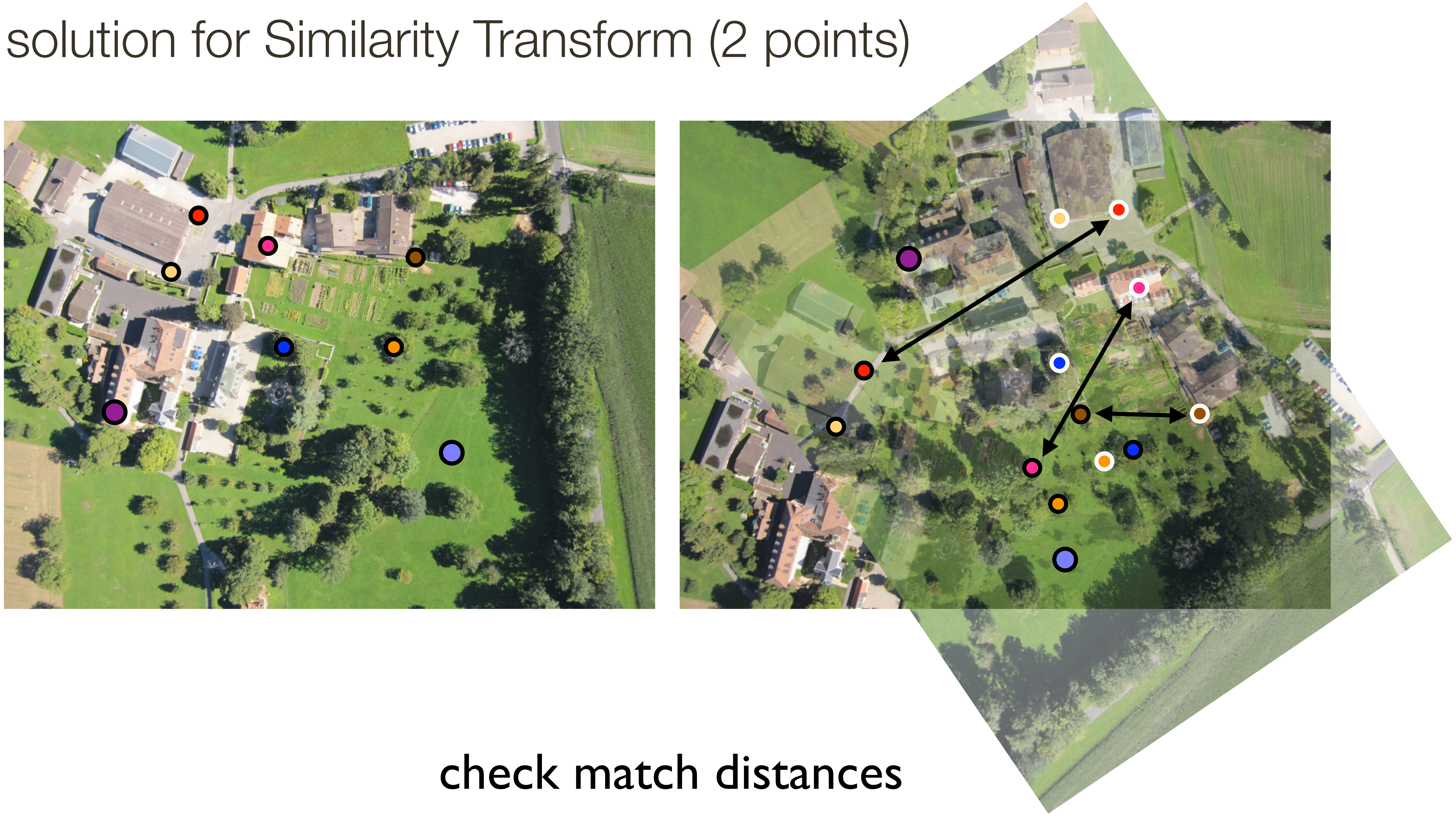
RANSAC solution for Similarity Transform (2 points)



check match distances

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)



check match distances

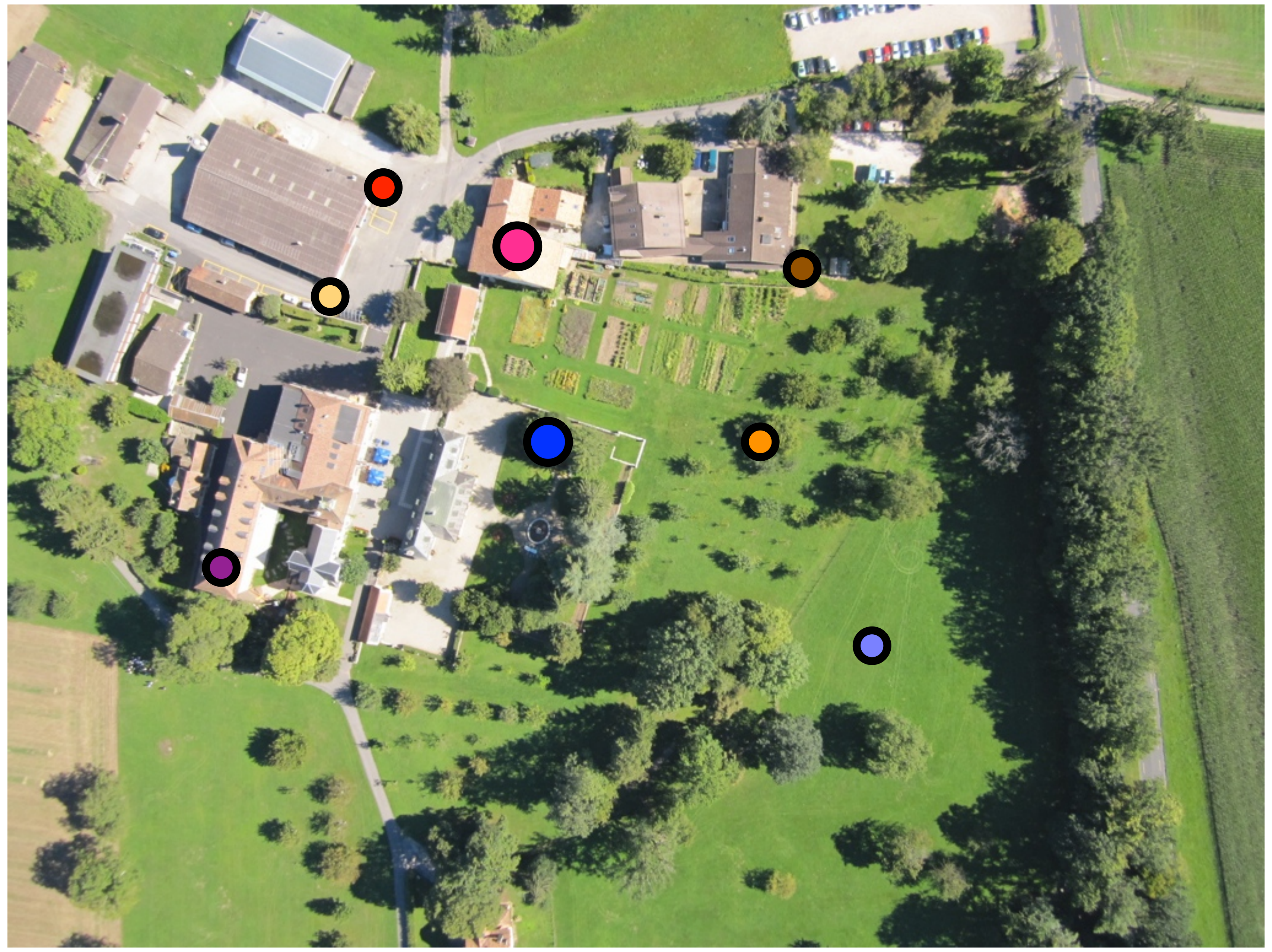
#inliers = 2

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)



choose pink, blue

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

warp image

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

check match distances

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

check match distances

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

check match distances

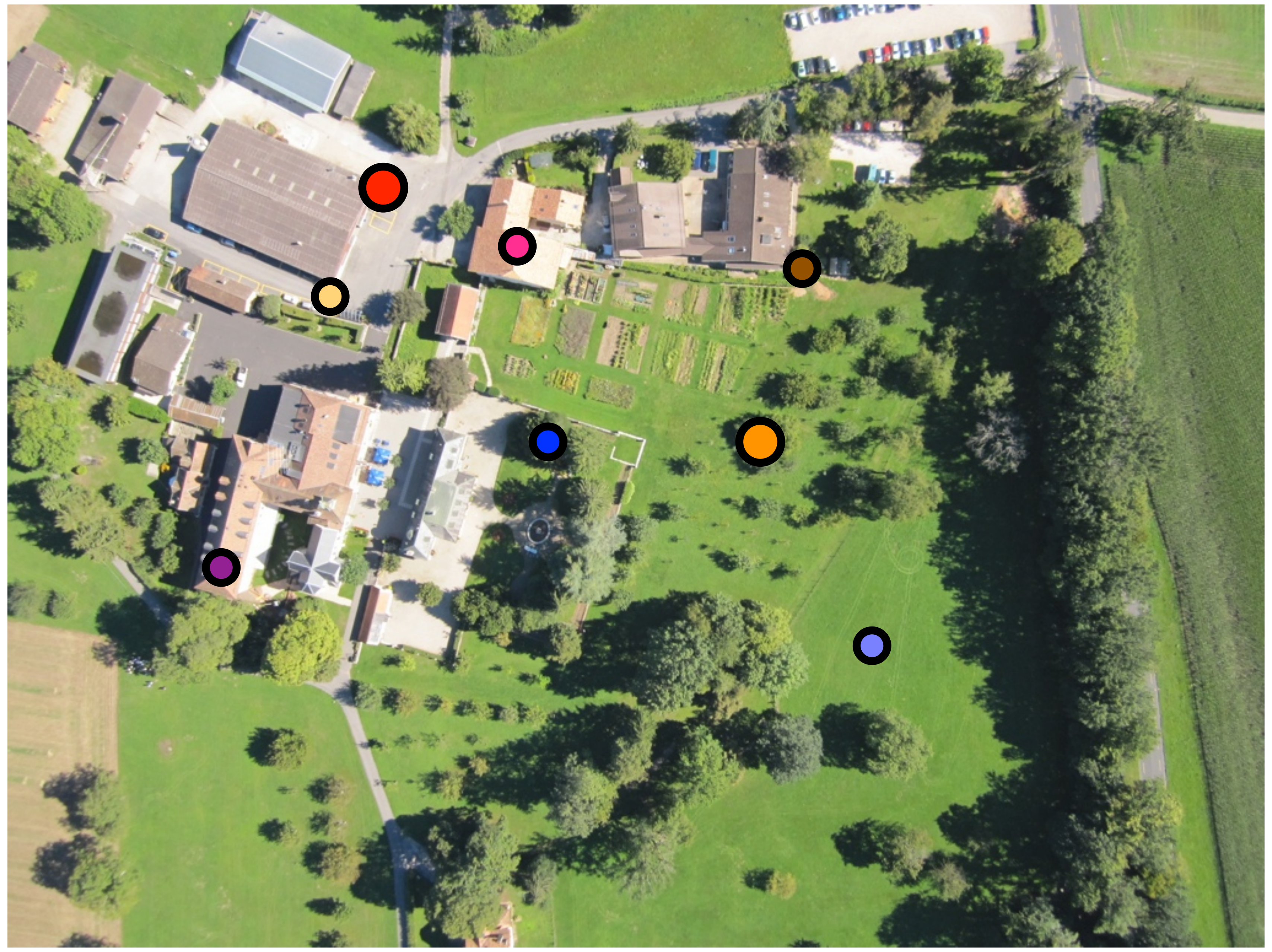
#inliers = 2

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

Image **Alignment + RANSAC**

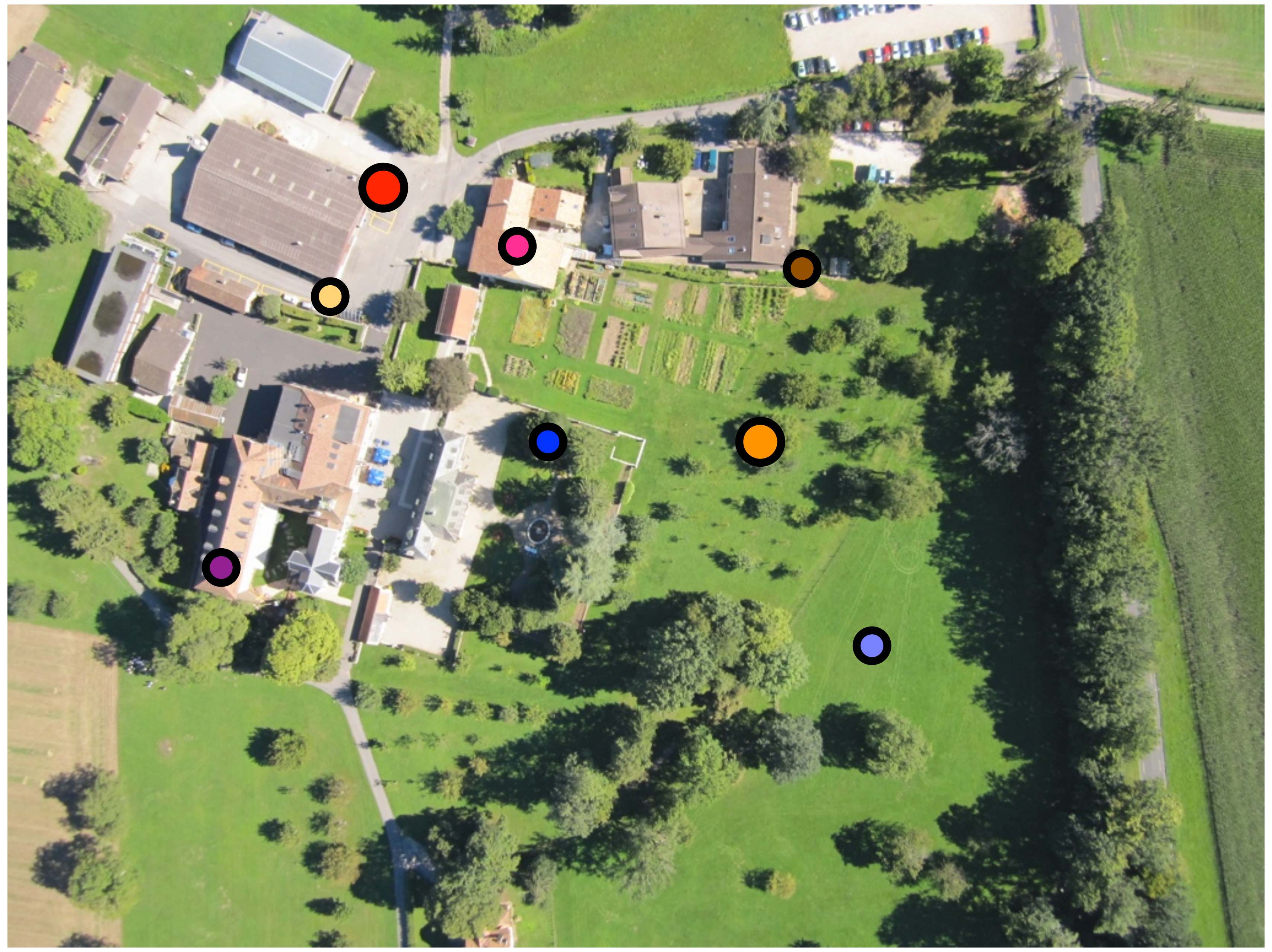
RANSAC solution for Similarity Transform (2 points)

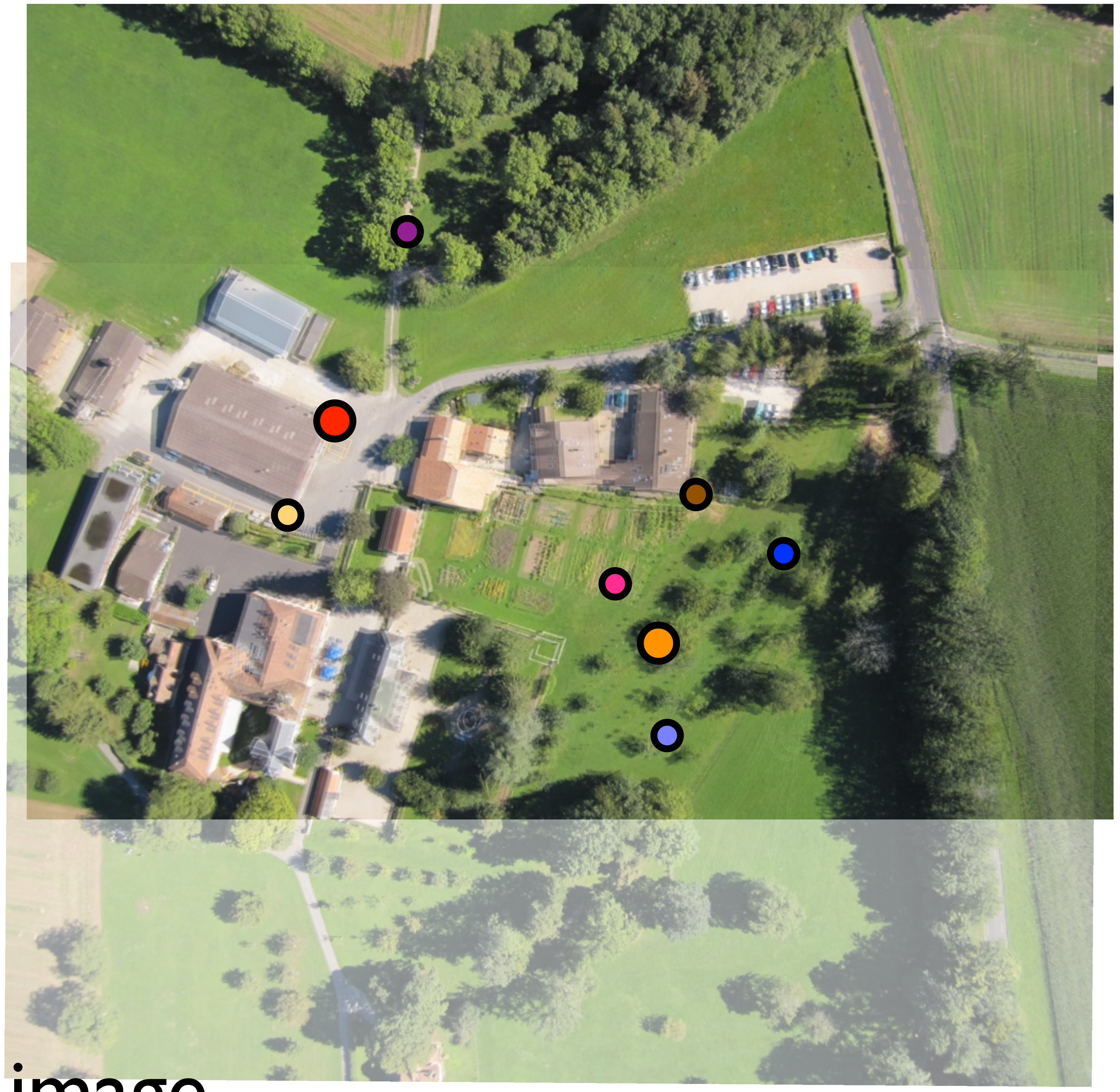


choose red, orange

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)





warp image

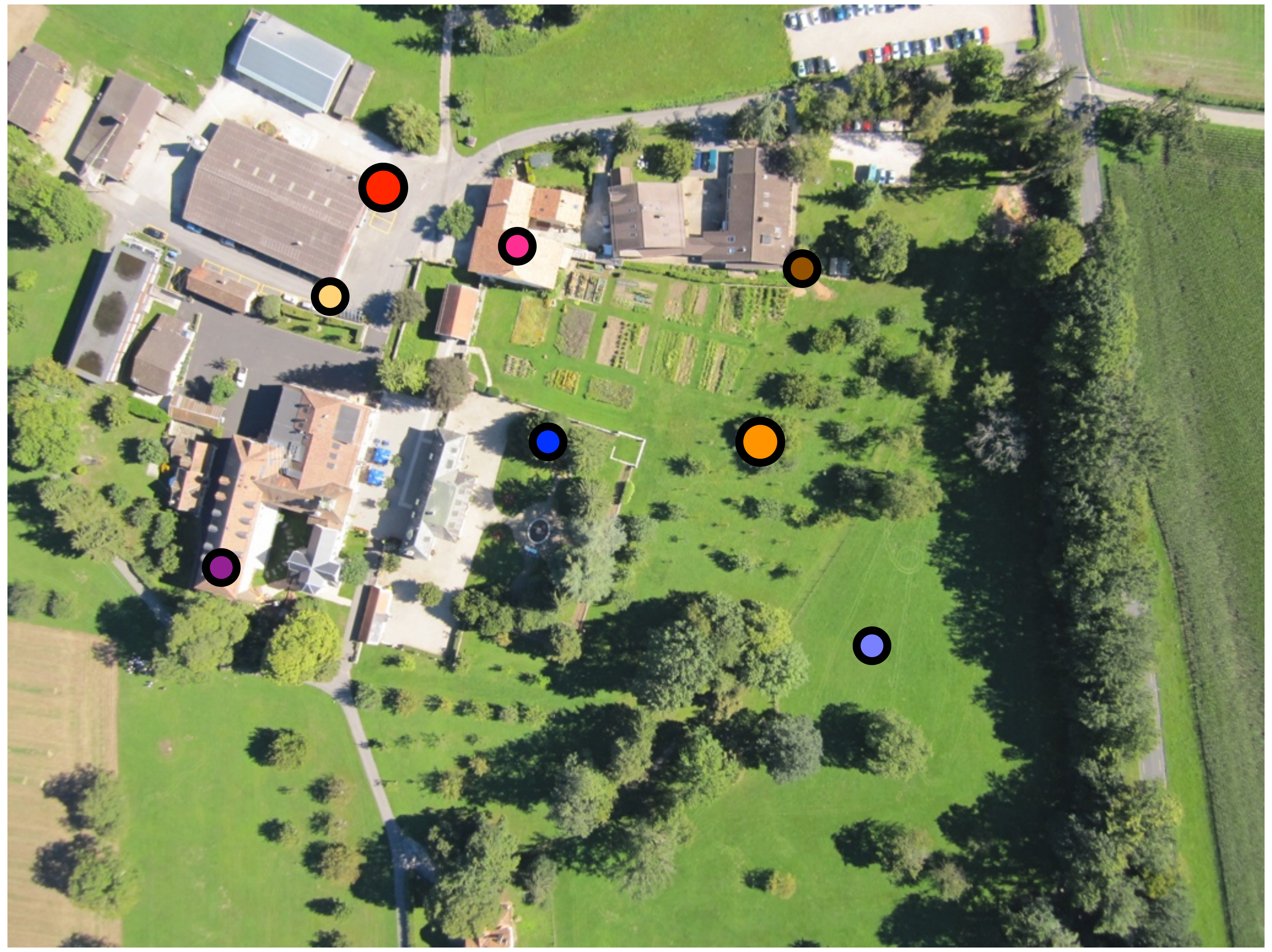
Image **Alignment + RANSAC**

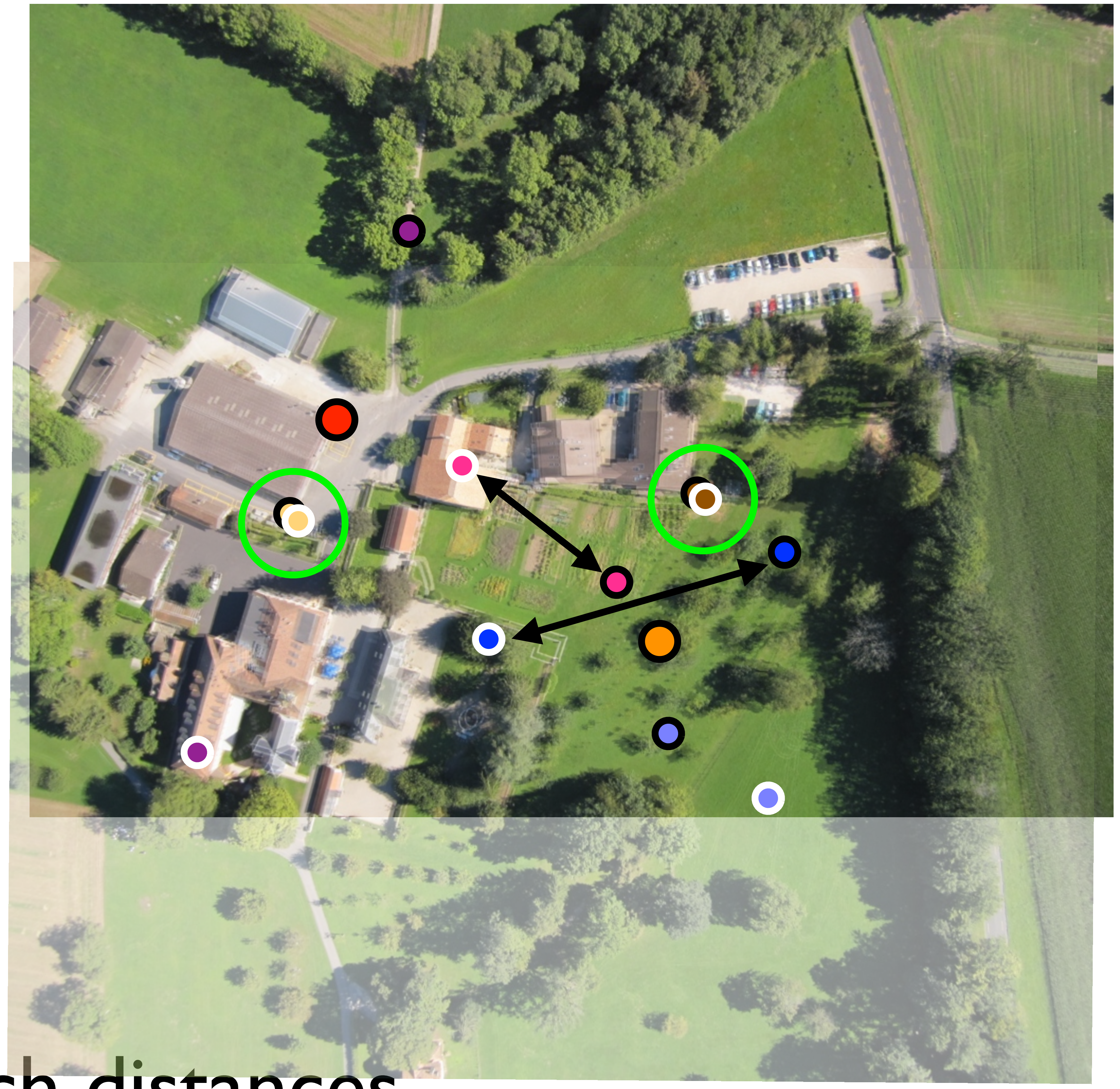
RANSAC solution for Similarity Transform (2 points)

check match distances

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)





check match distances

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

check match distances

#inliers = 4

Image **Alignment + RANSAC**

RANSAC solution for Similarity Transform (2 points)

Image **Alignment + RANSAC**

Assignment 4

1. Match feature points between 2 views

2. Select minimal subset of matches*

3. Compute transformation T using minimal subset

4. Check consistency of all points with T — compute projected position and count #inliers with distance < threshold

5. Repeat steps 2-4 to maximize #inliers

* Similarity transform = 2 points, Affine = 3, Homography = 4

RANSAC: *k* Samples Chosen (p = 0.99)

Sample size	Proportion of outliers						
n	**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

Figure Credit: Hartley & Zisserman

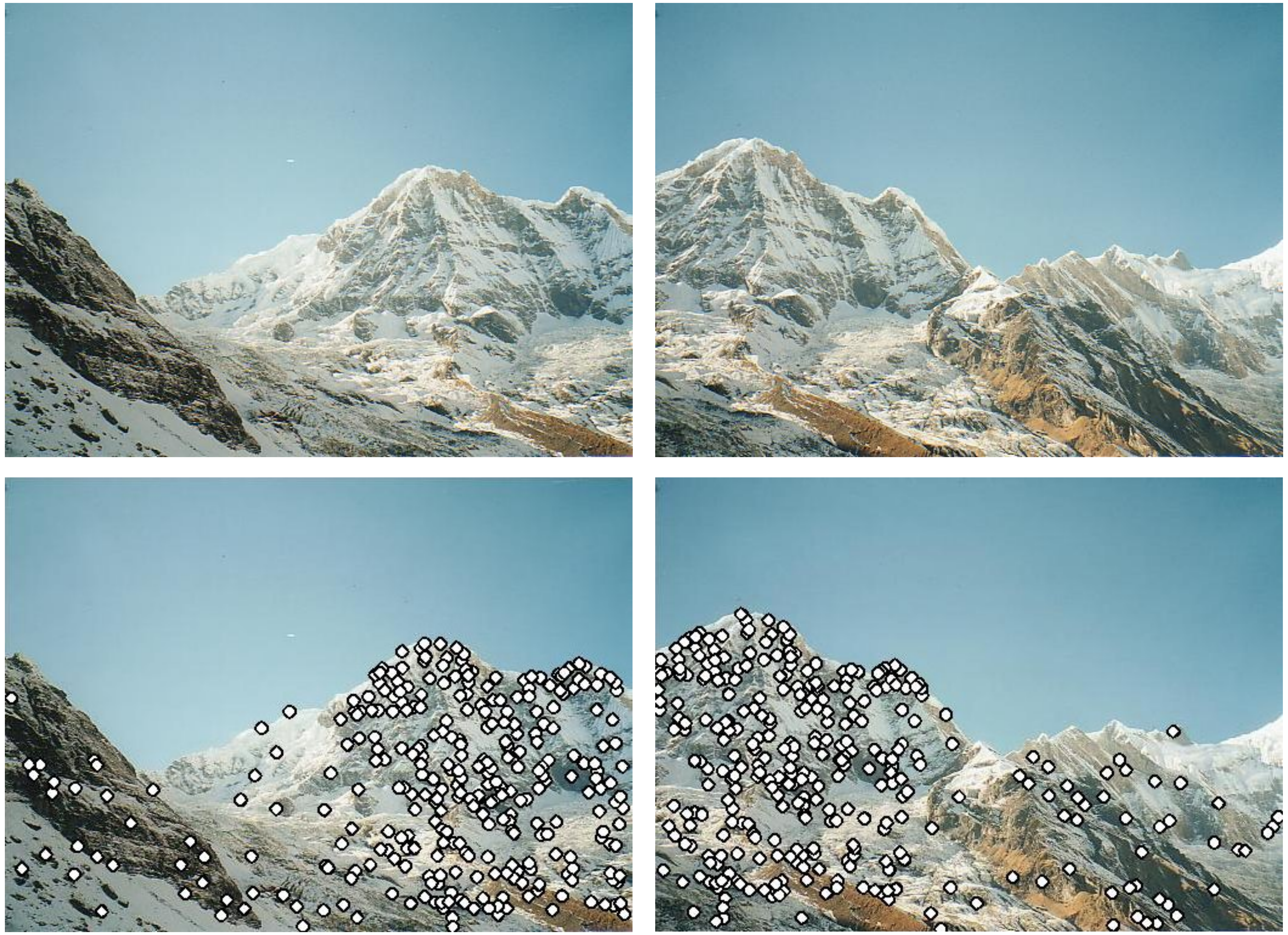
RANSAC: *k* Samples Chosen (p = 0.99)

Sample size	Proportion of outliers						
n	**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

Figure Credit: Hartley & Zisserman

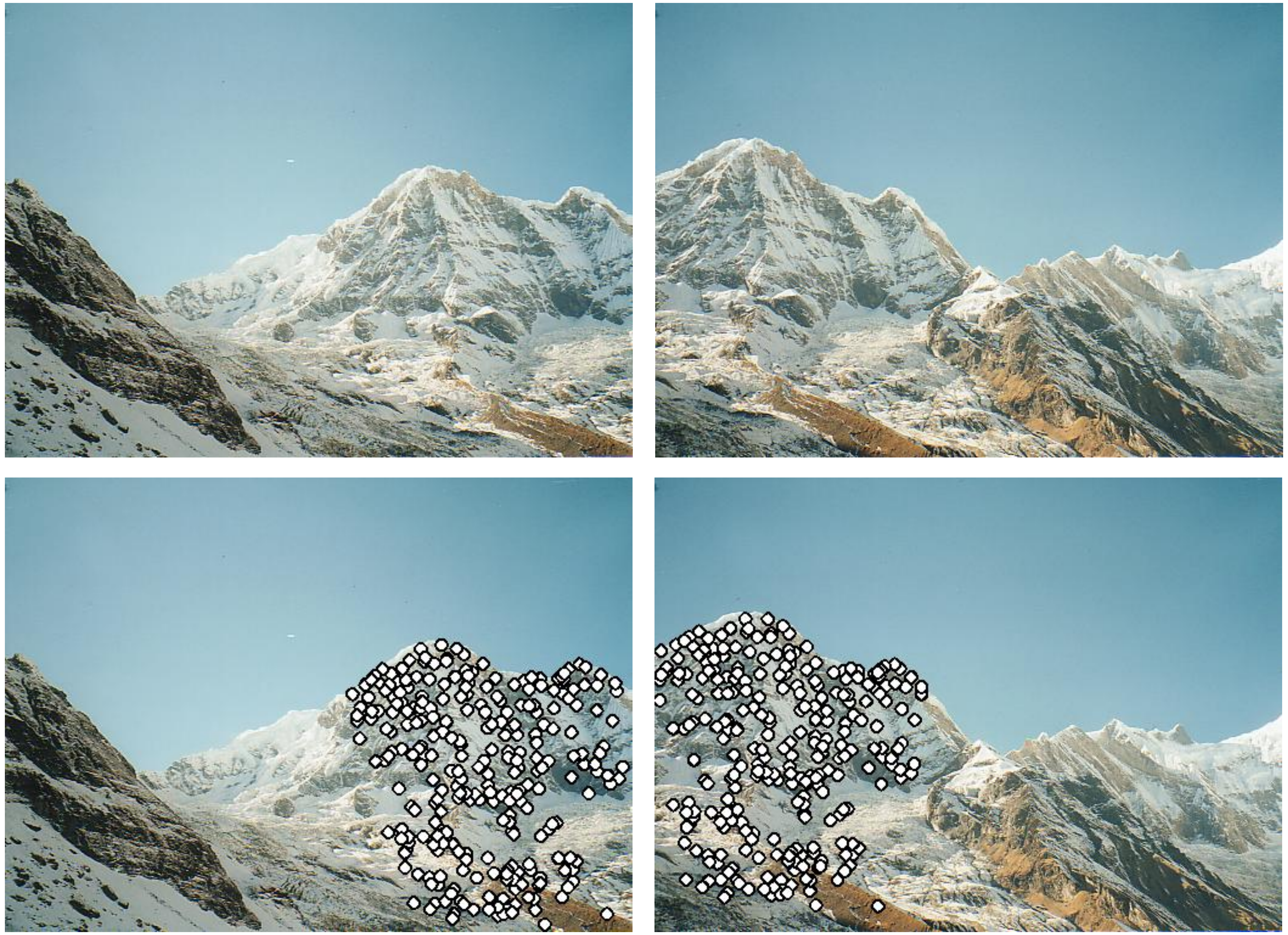
2-view **Rotation** Estimation

Find features + raw matches, use RANSAC to find Similarity



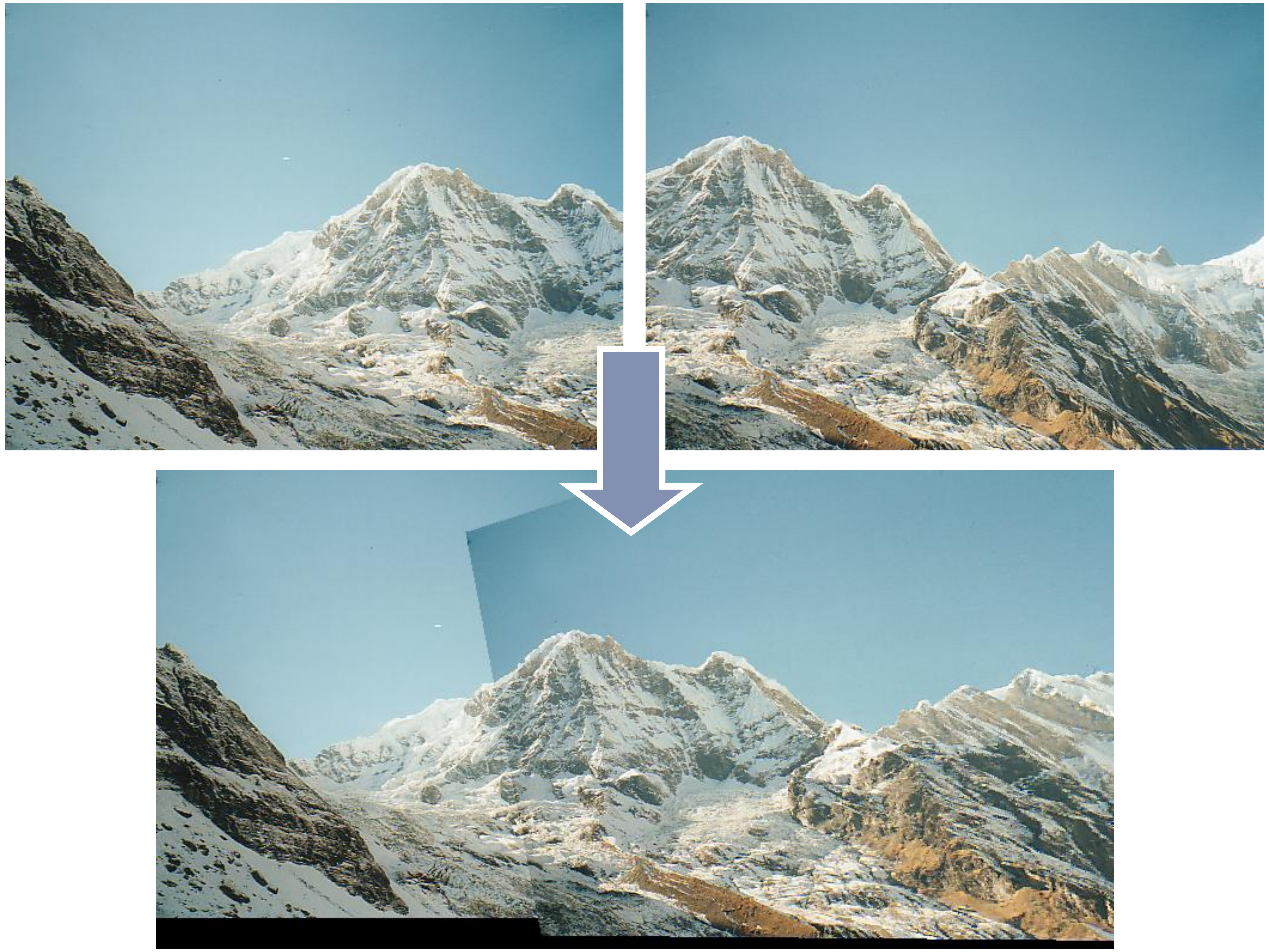
2-view **Rotation** Estimation

Remove outliers, can now solve for R using least squares



2-view **Rotation** Estimation

Final rotation estimation



Object **Instance Recognition**

Database of planar objects

Instance recognition

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Object **Instance Recognition** with SIFT

Match SIFT descriptors between **query image** and a database of known keypoints extracted from **training examples**

— use fast (approximate) nearest neighbour matching

— threshold based on ratio of distances between 1NN and 2NN

Use **RANSAC** to find a **subset of matches** that all agree on an object and geometric transform (e.g., **affine transform**)

Optionally **refine pose estimate** by recomputing the transformation using all the RANSAC inliers

Re-cap RANSAC

RANSAC is a technique to fit data to a model

— divide data into inliers and outliers

— estimate model from minimal set of inliers

— improve model estimate using all inliers

— alternate fitting with re-classification as inlier/outlier

RANSAC is a general method suited for a wide range of model fitting problems

— easy to implement

— easy to estimate/control failure rate

RANSAC only handles a moderate percentage of outliers without cost blowing up