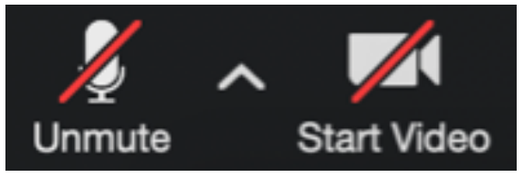
AEROSP 584 - Navigation and Guidance: From Perception to Control



Lectures start at 10:30am EST

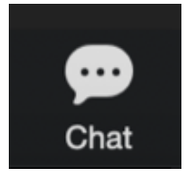
Vasileios Tzoumas

Lecture 19

COLLEGE OF ENGINEERING
AEROSPACE ENGINEERING
UNIVERSITY OF MICHIGAN

To ask questions:

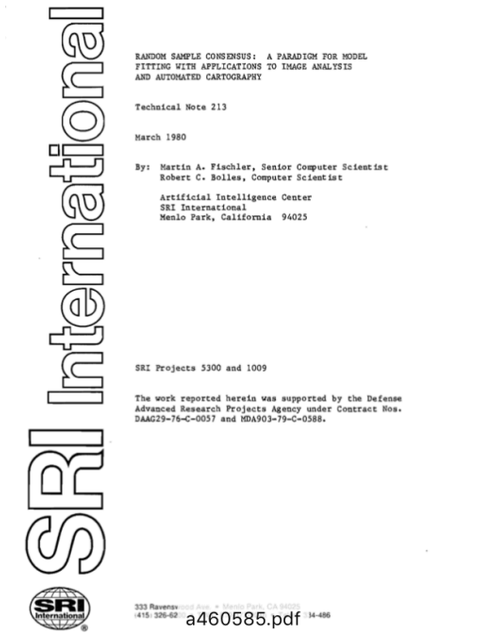
or



Based on slides made by Luca Carlone @ AEROASTRO MIT

Today

- Recap on 2-view
- RANSAC
- 3D-3D correspondences



SRI International

RANDOM SAMPLE CONSENSUS: A PARADIGM FOR MODEL FITTING WITH APPLICATIONS TO IMAGE ANALYSIS AND AUTOMATED CARTOGRAPHY

Technical Note 213

March 1980

By: Martin A. Fischler, Senior Computer Scientist
Robert C. Bolles, Computer Scientist

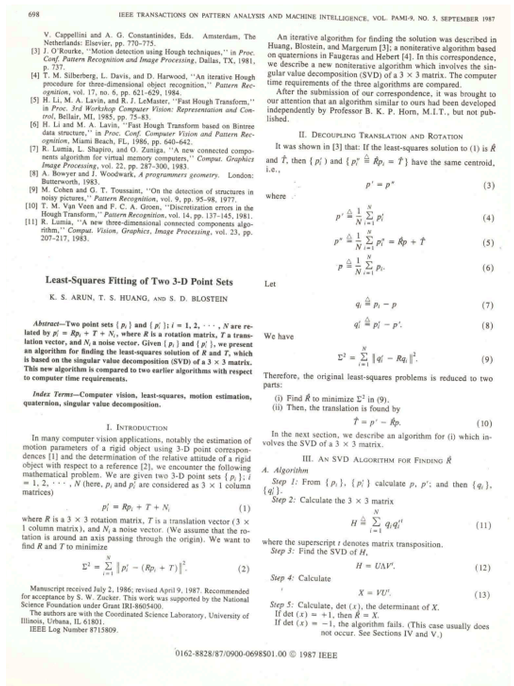
Artificial Intelligence Center
SRI International
Menlo Park, California 94025

SRI Projects 5300 and 1009

The work reported herein was supported by the Defense Advanced Research Projects Agency under Contract Nos. DAAG29-76-C-0037 and MDA903-79-C-0588.

333 Ravenswood Avenue · Menlo Park, CA 94025
(415) 326-6200 a460585.pdf 34-486

[1]



698 IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-9, NO. 5, SEPTEMBER 1987

V. Cappellini and A. G. Constantinides, Eds. Amsterdam, The Netherlands: Elsevier, pp. 770–775.
[3] J. O'Rourke, "Motion detection using Hough techniques," in *Proc. Conf. Pattern Recognition and Image Processing*, Dallas, TX, 1981, p. 737.
[4] T. M. Silberberg, L. Davis, and D. Harwood, "An iterative Hough procedure for three-dimensional object recognition," *Pattern Recognition*, vol. 17, no. 6, pp. 621–629, 1984.
[5] H. Li, M. A. Lavin, and R. J. LeMaster, "Fast Hough Transform," in *Proc. 3rd Workshop Computer Vision: Representation and Control*, Bellair, MI, 1985, pp. 75–83.
[6] H. Li and M. A. Lavin, "Fast Hough Transform based on Bintree data structure," in *Proc. Conf. Computer Vision and Pattern Recognition*, Miami Beach, FL, 1986, pp. 640–642.
[7] R. Lumia, L. Shapiro, and O. Zuniga, "A new connected components algorithm for virtual memory computers," *Comput. Graphics Image Processing*, vol. 22, pp. 287–300, 1983.
[8] A. Bowyer and J. Woodwark, *A programmers geometry*. London: Butterworth, 1983.
[9] M. Cohen and G. T. Toussaint, "On the detection of structures in noisy pictures," *Pattern Recognition*, vol. 9, pp. 95–98, 1977.
[10] T. M. Van Veen and F. C. A. Groen, "Discretization errors in the Hough Transform," *Pattern Recognition*, vol. 14, pp. 137–145, 1981.
[11] R. Lumia, "A new three-dimensional connected components algorithm," *Comput. Vision, Graphics, Image Processing*, vol. 23, pp. 207–217, 1983.

Least-Squares Fitting of Two 3-D Point Sets

K. S. ARUN, T. S. HUANG, AND S. D. BLOSTEIN

Abstract—**Two point sets $\{p_i\}$ and $\{p'_i\}; i = 1, 2, \cdots, N$ are related by $p'_i = Rp_i + T + N_i$, where R is a rotation matrix, T a translation vector, and N_i a noise vector. Given $\{p_i\}$ and $\{p'_i\}$, we present an algorithm for finding the least-squares solution of R and T, which is based on the singular value decomposition (SVD) of a 3×3 matrix. This new algorithm is compared to two earlier algorithms with respect to computer time requirements.**

Index Terms—**Computer vision, least-squares, motion estimation, quaternion, singular value decomposition.**

I. Introduction

In many computer vision applications, notably the estimation of motion parameters of a rigid object using 3-D point correspondences [1] and the determination of the relative attitude of a rigid object with respect to a reference [2], we encounter the following mathematical problem. We are given two 3-D point sets $\{p_i\}; i = 1, 2, \cdots, N$ (here, p_i and p'_i are considered as 3×1 column matrices)

$$p'_i = Rp_i + T + N_i \tag{1}$$

where R is a 3×3 rotation matrix, T is a translation vector (3×1 column matrix), and N_i a noise vector. (We assume that the rotation is around an axis passing through the origin). We want to find R and T to minimize

$$\Sigma^2 = \sum_{i=1}^{N} \|p'_i - (Rp_i + T)\|^2. \tag{2}$$

Manuscript received July 2, 1986; revised April 9, 1987. Recommended for acceptance by S. W. Zucker. This work was supported by the National Science Foundation under Grant IRI-8605400.

The authors are with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

IEEE Log Number 8715809.

An iterative algorithm for finding the solution was described in Huang, Blostein, and Margerum [3]; a noniterative algorithm based on quaternions in Faugeras and Hebert [4]. In this correspondence, we describe a new noniterative algorithm which involves the singular value decomposition (SVD) of a 3×3 matrix. The computer time requirements of the three algorithms are compared.

After the submission of our correspondence, it was brought to our attention that an algorithm similar to ours had been developed independently by Professor B. K. P. Horn, M.I.T., but not published.

II. Decoupling Translation and Rotation

It was shown in [3] that: If the least-squares solution to (1) is $\hat{R}$ and $\hat{T}$, then $\{p'_i\}$ and $\{p''_i \triangleq \hat{R}p_i + \hat{T}\}$ have the same centroid, i.e.,

$$p' = p'' \tag{3}$$

where

$$p' \triangleq \frac{1}{N}\sum_{i=1}^{N} p'_i \tag{4}$$

$$p'' \triangleq \frac{1}{N}\sum_{i=1}^{N} p''_i = \hat{R}p + \hat{T} \tag{5}$$

$$p \triangleq \frac{1}{N}\sum_{i=1}^{N} p_i. \tag{6}$$

Let

$$q_i \triangleq p_i - p \tag{7}$$

$$q'_i \triangleq p'_i - p'. \tag{8}$$

We have

$$\Sigma^2 = \sum_{i=1}^{N} \|q'_i - Rq_i\|^2. \tag{9}$$

Therefore, the original least-squares problems is reduced to two parts:

(i) Find $\hat{R}$ to minimize Σ^2 in (9).
(ii) Then, the translation is found by

$$\hat{T} = p' - \hat{R}p. \tag{10}$$

In the next section, we describe an algorithm for (i) which involves the SVD of a 3×3 matrix.

III. An SVD Algorithm for Finding $\hat{R}$

A. Algorithm

Step 1: From $\{p_i\}$, $\{p'_i\}$ calculate p, p'; and then $\{q_i\}$, $\{q'_i\}$.

Step 2: Calculate the 3×3 matrix

$$H \triangleq \sum_{i=1}^{N} q_i q'^t_i \tag{11}$$

where the superscript t denotes matrix transposition.

Step 3: Find the SVD of H,

$$H = UAV^t. \tag{12}$$

Step 4: Calculate

$$X = VU^t. \tag{13}$$

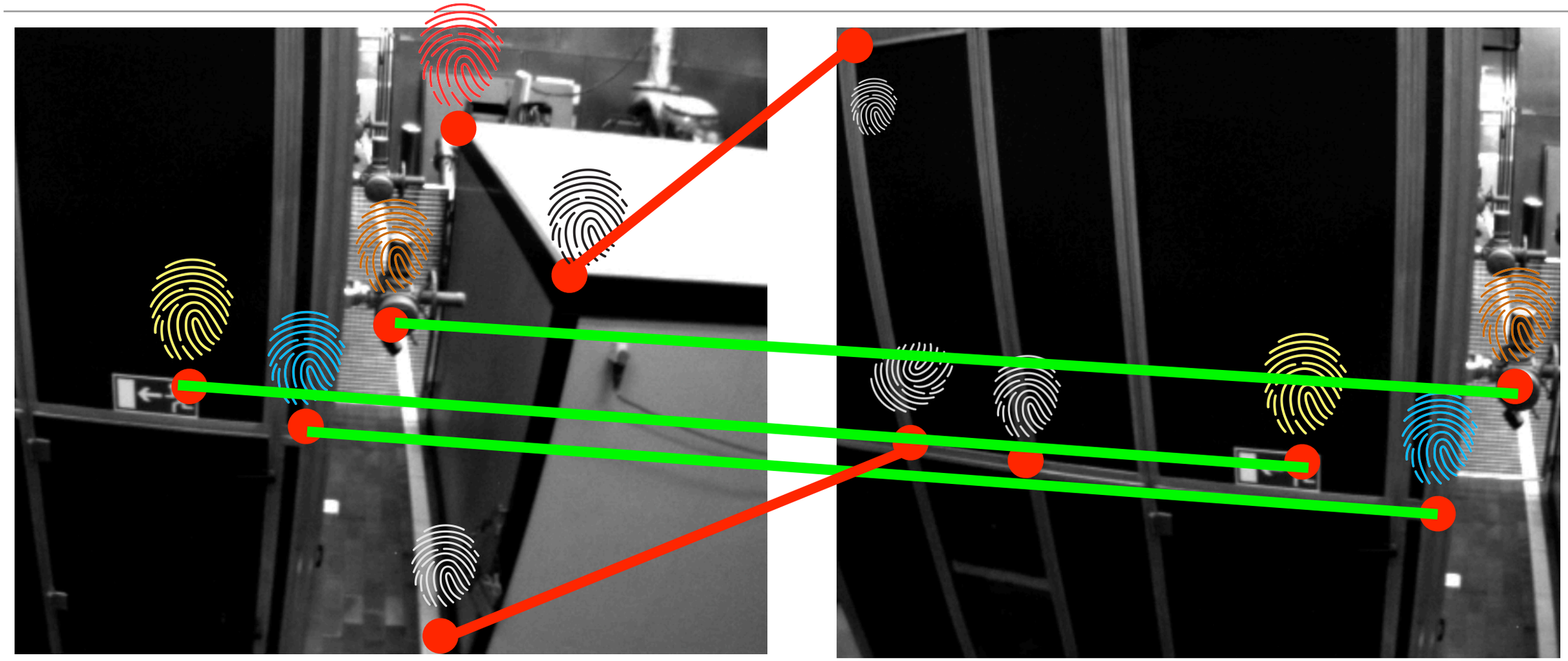
Step 5: Calculate, $\det(x)$, the determinant of X.
If $\det(x) = +1$, then $\hat{R} = X$.
If $\det(x) = -1$, the algorithm fails. (This case usually does not occur. See Sections IV and V.)

[2]

[1] M .Fisher, R. Bollets, “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”, SRI Technical Note, 1980.

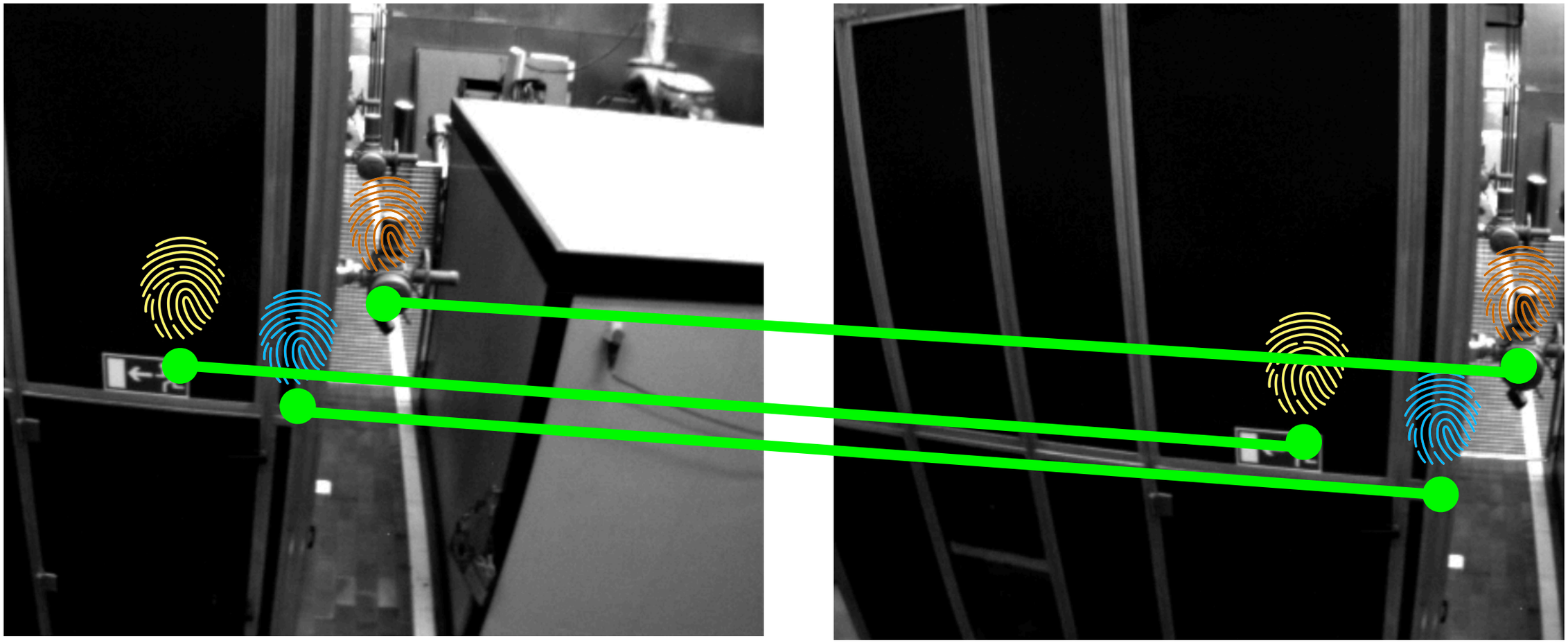
[2] K.S. Arun, T.S. Huang, S.D. Blostein, "Least-Squares Fitting of Two 3-D Point Sets", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 9(5), 698-700, 1987.

2-view Geometry



Question: can we estimate the motion of the camera between I_1 and I_2 using pixel correspondences?

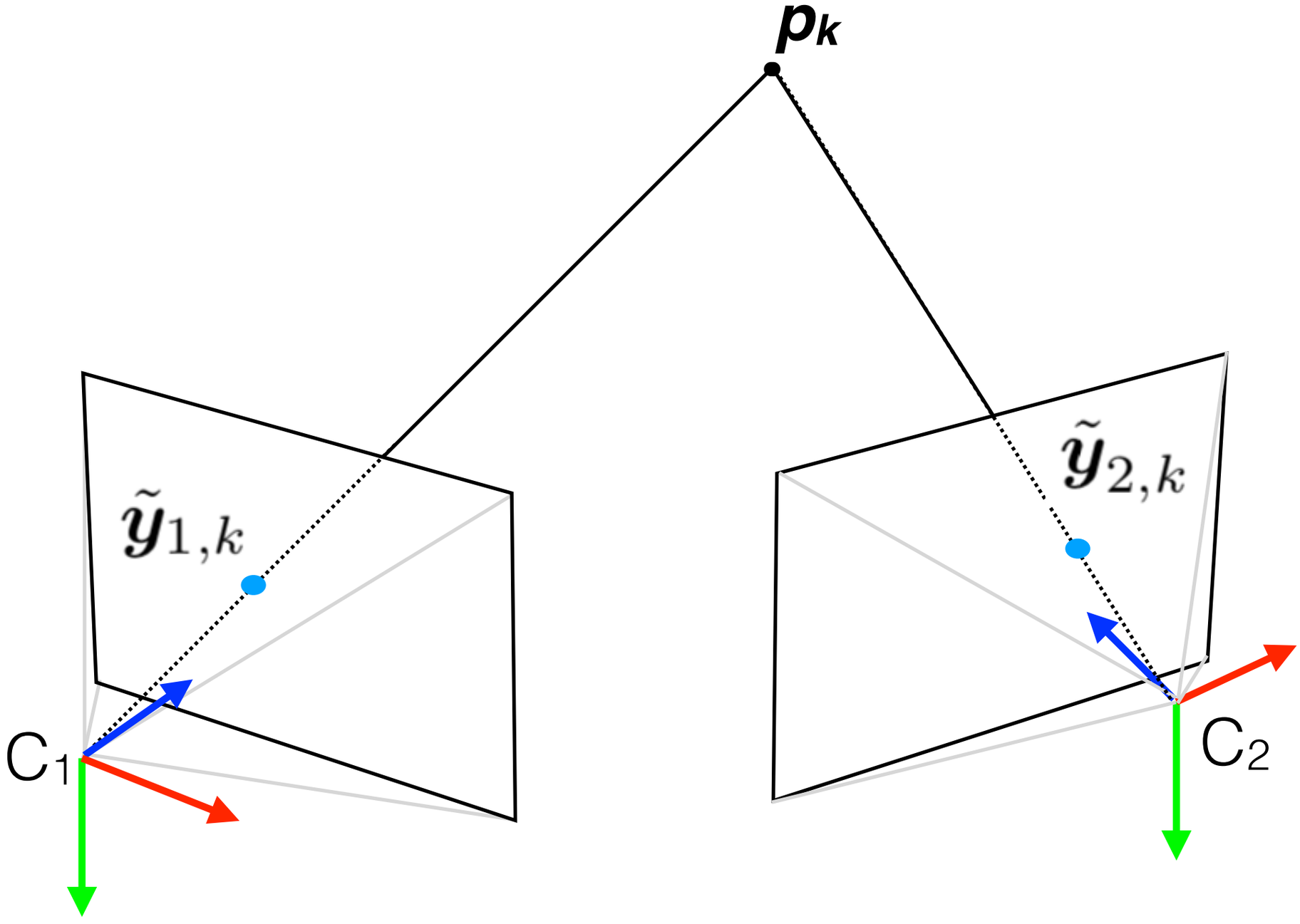
2-view Geometry



Monday's assumptions:

- no wrong correspondences (outliers)
- 3D point is not moving
- camera calibration is known

2-view Geometry



$$\tilde{\boldsymbol{y}}_{2,k}^{\mathsf{T}}\, \boldsymbol{E}\, \tilde{\boldsymbol{y}}_{1,k} = 0$$

Essential matrix encodes relative pose (up to scale) between C_1 and C_2

Estimating Poses from Correspondences

Given *N* calibrated pixel correspondences:

$$(\tilde{\boldsymbol{y}}_{1,k}, \tilde{\boldsymbol{y}}_{2,k}) \text{ for } k = 1, \ldots, N$$

1. leverage the epipolar constraints to estimate the essential matrix ***E***

$$\tilde{\boldsymbol{y}}_{2,k}^{\mathsf{T}} \boldsymbol{E} \tilde{\boldsymbol{y}}_{1,k} = 0$$

For 8 points: $\boldsymbol{A}\boldsymbol{e} = 0$ N>8 points: $\arg\min_{\|\boldsymbol{e}\|=1} \|\boldsymbol{A}\boldsymbol{e}\|^2$

2. Retrieve the rotation and translation (up to scale) from the ***E***

$$\boldsymbol{E} = [\boldsymbol{t}]_{\times} \boldsymbol{R}$$

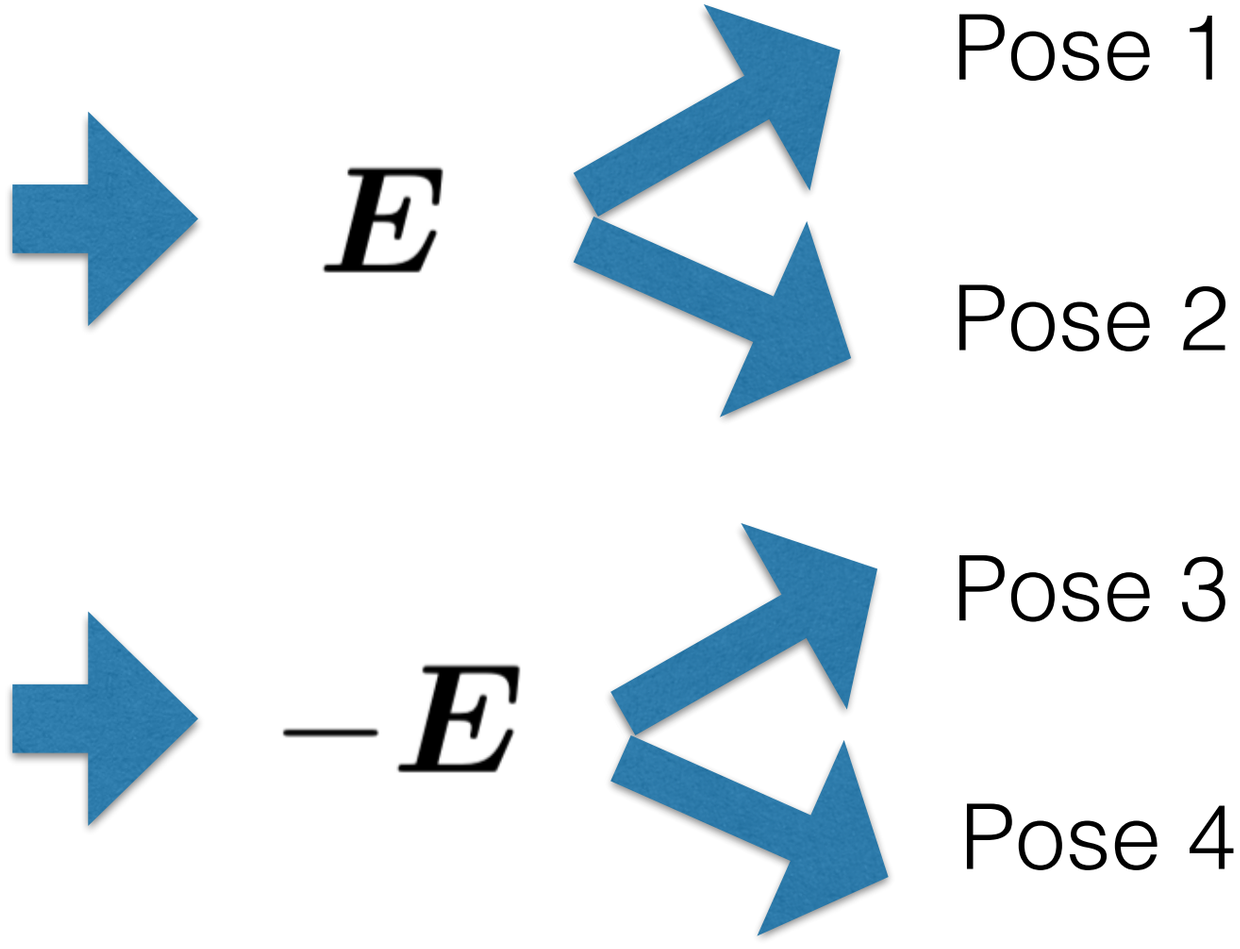
Retrieving Pose from Essential Matrix

Theorem 1 (Pose recovery from essential matrix, Thm 5.7 in [1]). *There exist exactly two relative poses* $(\boldsymbol{R}, \boldsymbol{t})$ *with* $\boldsymbol{R} \in \mathrm{SO}(3)$ *and* $\boldsymbol{t} \in \mathbb{R}^3$ *corresponding to a nonzero essential matrix* $\boldsymbol{E}$ *(i.e., such that* $\boldsymbol{E} = [\boldsymbol{t}]_\times \boldsymbol{R}$*):*

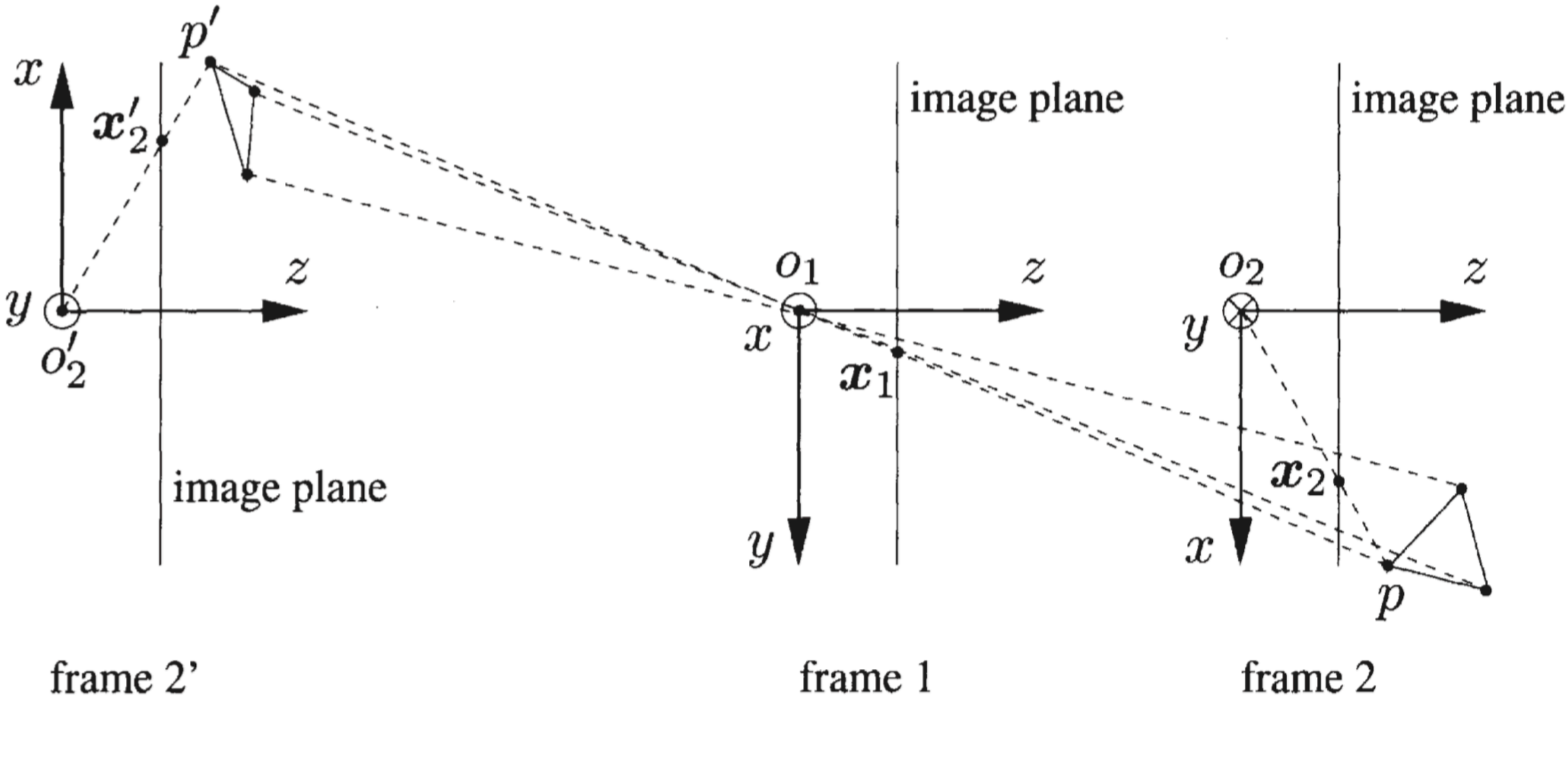
$$\boldsymbol{t}_1 = \boldsymbol{U}\boldsymbol{R}_z(+\pi/2)\boldsymbol{\Sigma}\boldsymbol{U}^\mathsf{T} \qquad \boldsymbol{R}_1 = \boldsymbol{U}\boldsymbol{R}_z(+\pi/2)\boldsymbol{V}^\mathsf{T} \tag{13.19}$$

$$\boldsymbol{t}_2 = \boldsymbol{U}\boldsymbol{R}_z(-\pi/2)\boldsymbol{\Sigma}\boldsymbol{U}^\mathsf{T} \qquad \boldsymbol{R}_2 = \boldsymbol{U}\boldsymbol{R}_z(-\pi/2)\boldsymbol{V}^\mathsf{T} \tag{13.20}$$

where $\boldsymbol{E} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\mathsf{T}$ *is the singular value decomposition of the matrix* $\boldsymbol{E}$*, and* $\boldsymbol{R}_z(+\pi/2)$ *is an elementary rotation around the z-axis of an angle* $\pi/2$*.*

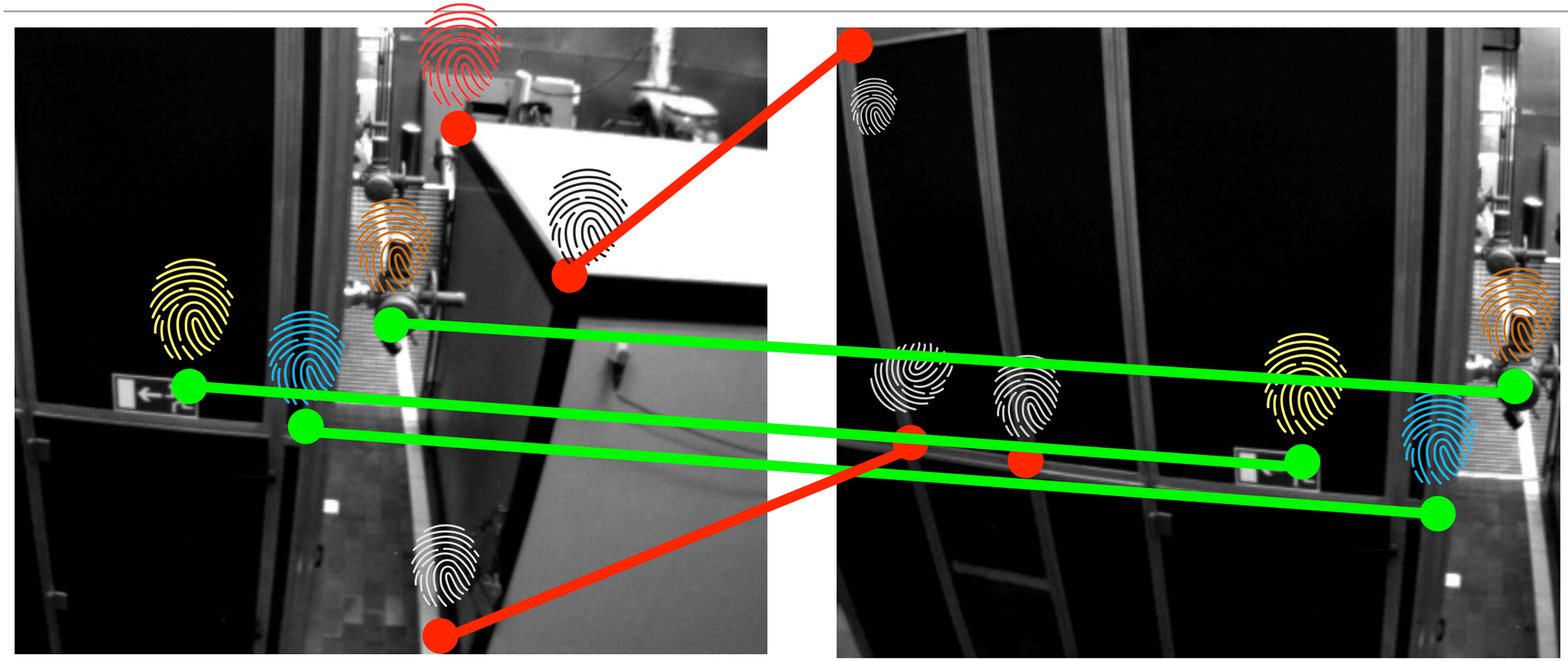


Cheirality constraints



Points must be in front of the cameras!

2-view Geometry



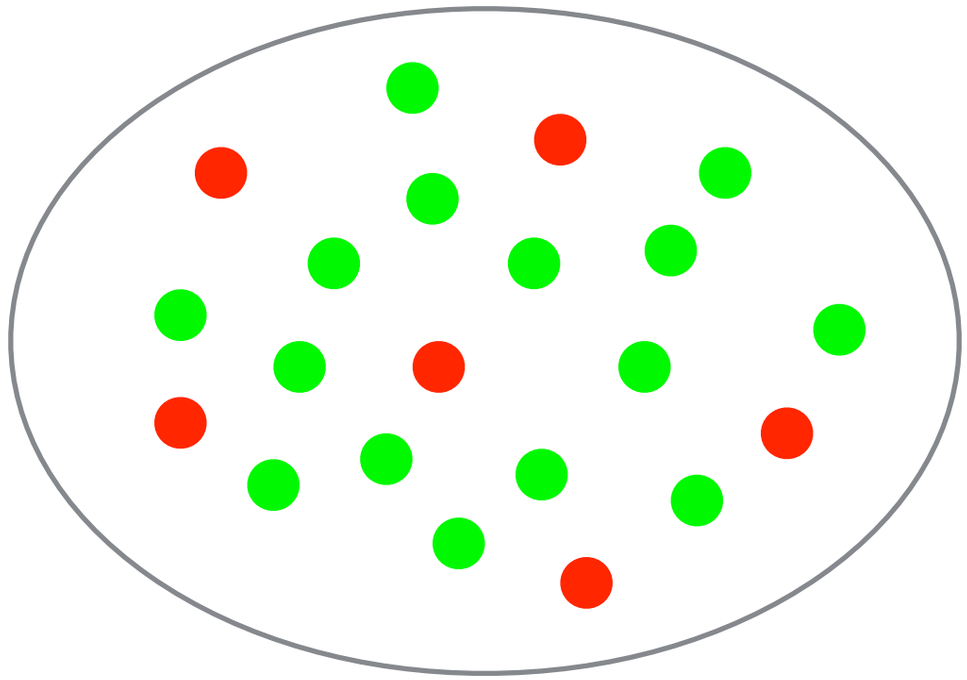
In practice:

- Many wrong correspondences (outliers)
- Some 3D points might be moving

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.



Assume: we have an algorithm to estimate *P* from *n* data points (n << N)

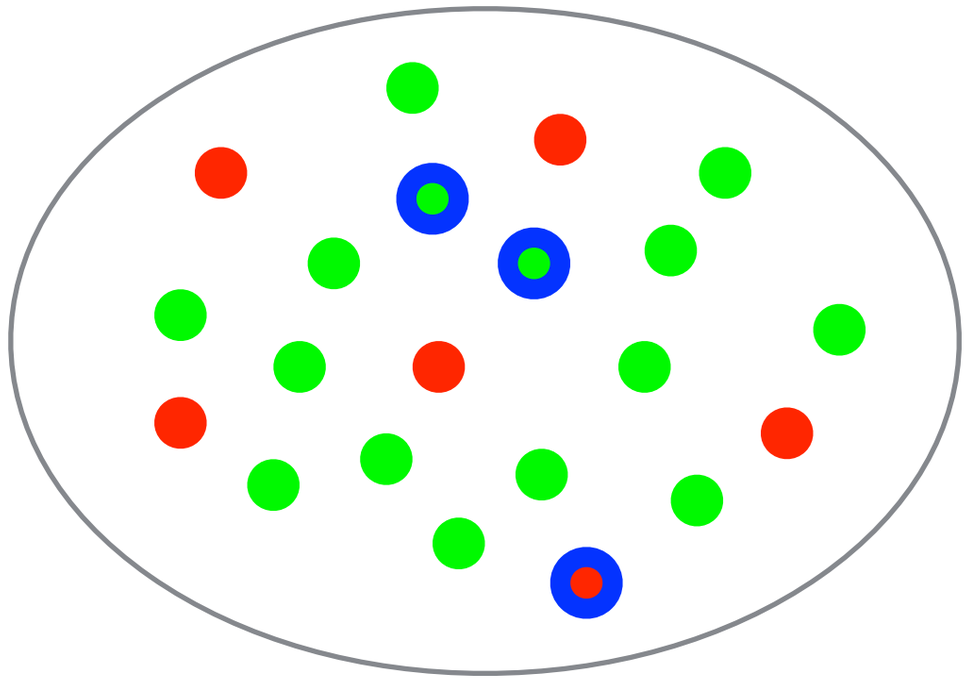
Basic idea:

1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.



Assume: we have an algorithm to estimate *P* from *n* data points (n << N)

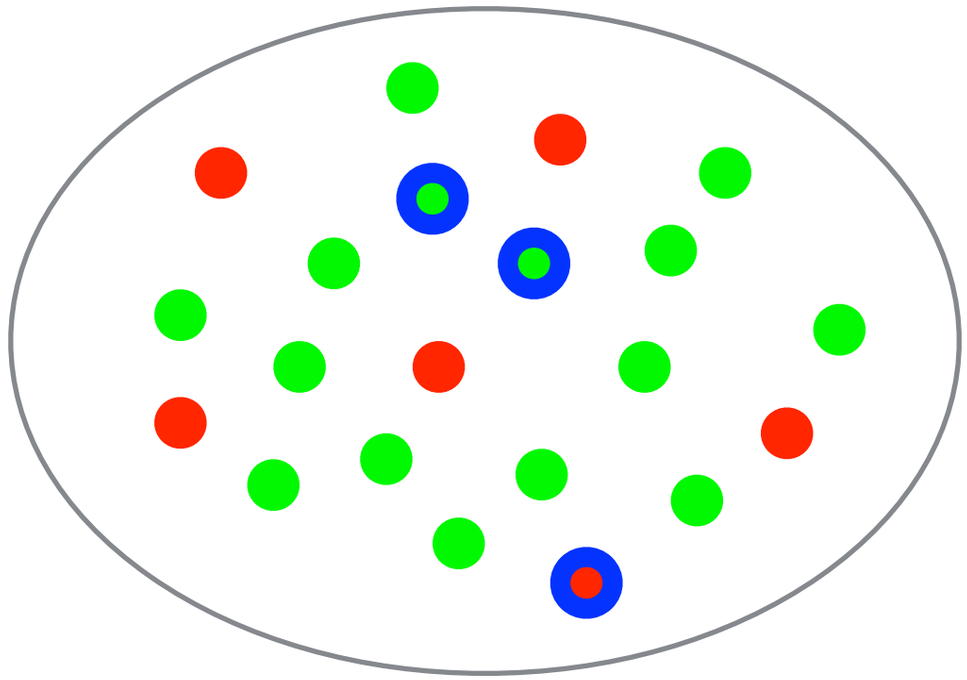
Basic idea:

1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.



P'

Assume: we have an algorithm to estimate *P* from *n* data points (n << N)

Basic idea:

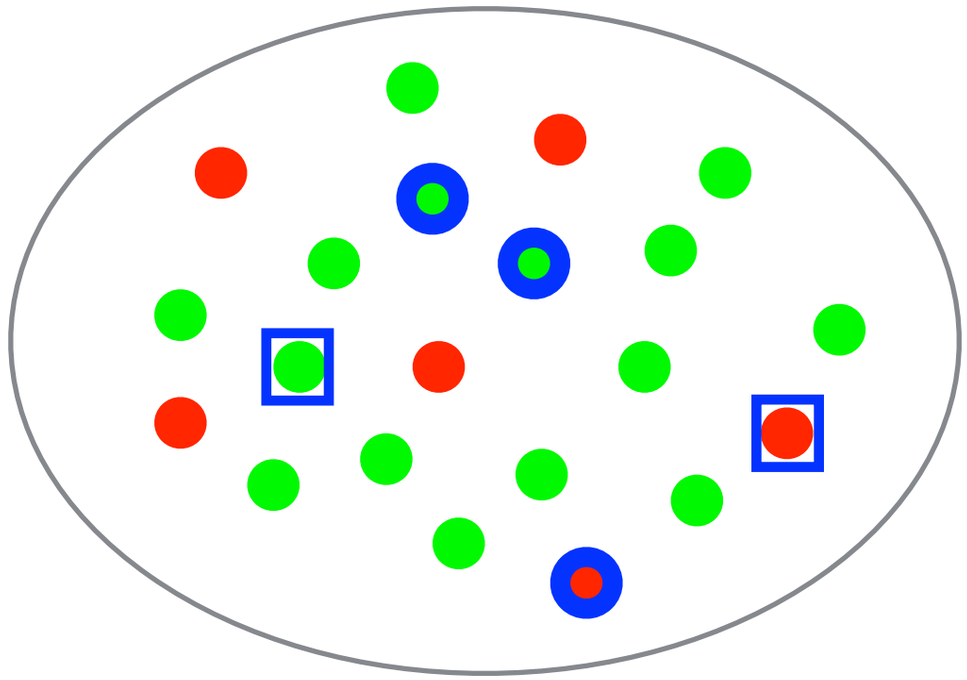
1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.

Assume: we have an algorithm to estimate *P* from *n* data points (n << N)



P’

Basic idea:

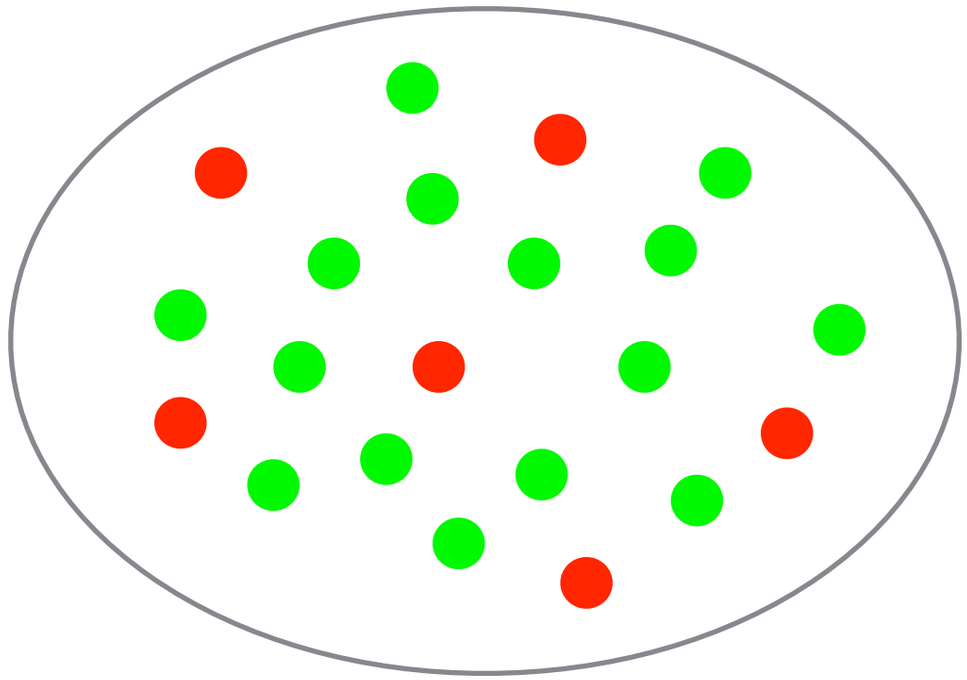
1. sample *n* points
2. compute an estimate *P’* of *P*
3. count how many other points agree with *P’*
4. repeat until you get a *P’* that agrees with many points

Consensus Set

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.



Assume: we have an algorithm to estimate *P* from *n* data points (n << N)

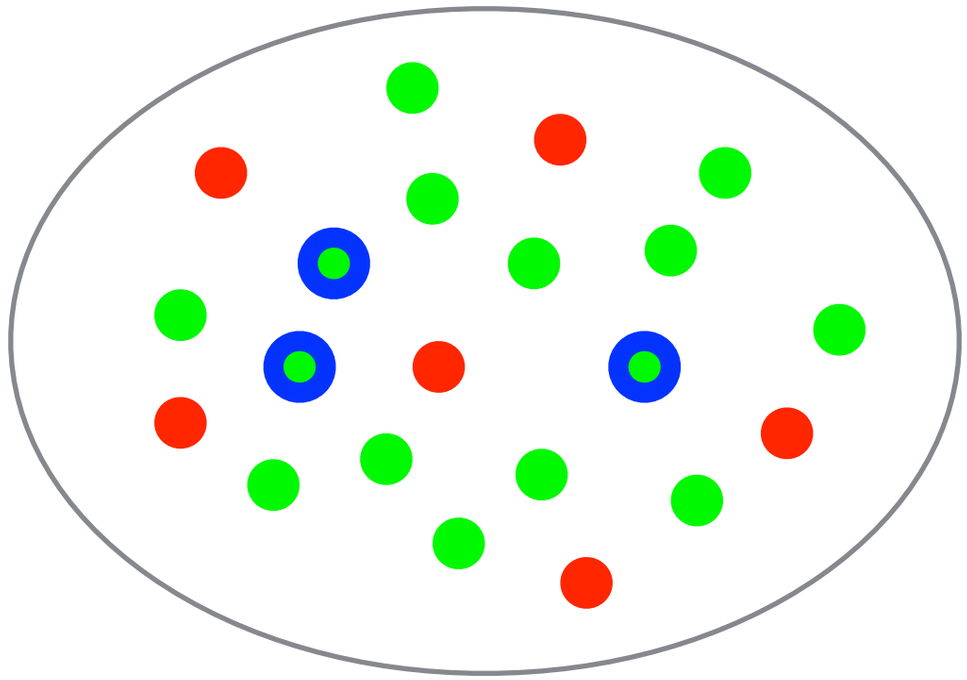
Basic idea:

1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.



Assume: we have an algorithm to estimate *P* from *n* data points (n << N)

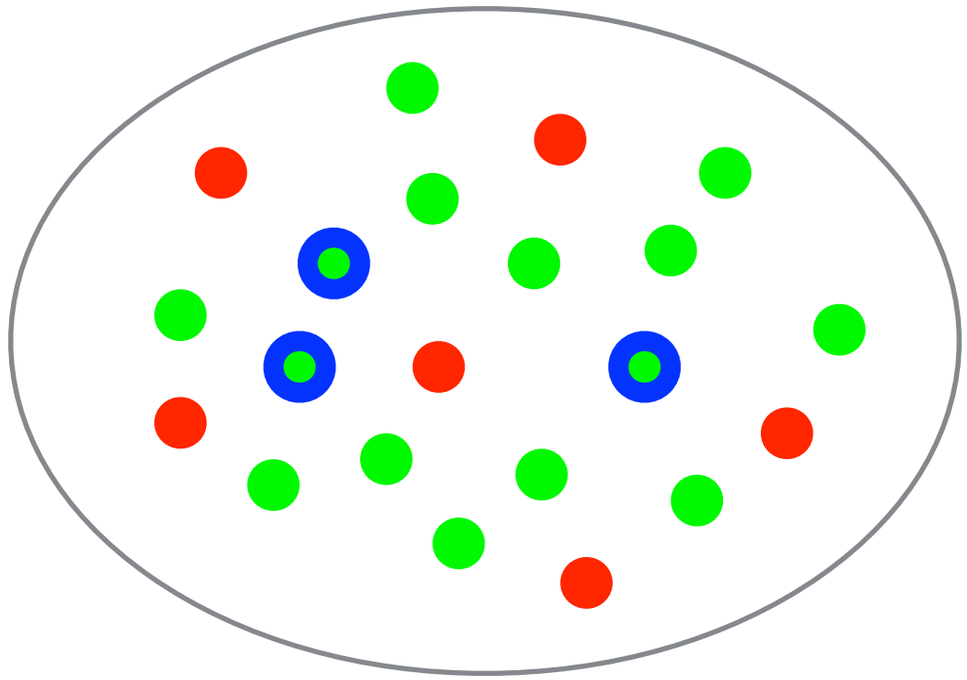
Basic idea:

1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.



P’

Assume: we have an algorithm to estimate *P* from *n* data points (n << N)

Basic idea:

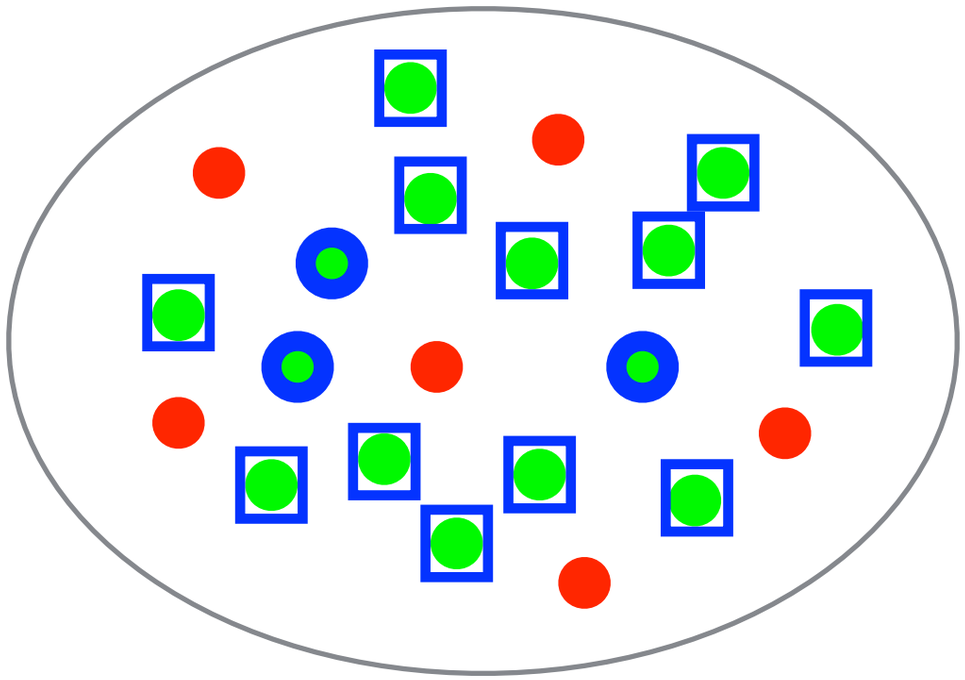
1. sample *n* points
2. compute an estimate *P’* of *P*
3. count how many other points agree with *P’*
4. repeat until you get a *P’* that agrees with many points

RANSAC

RANdom **SA**mple **C**onsensus

Problem: estimate model *P* from N data points, possibly corrupted with outliers.

Assume: we have an algorithm to estimate *P* from *n* data points (n << N)



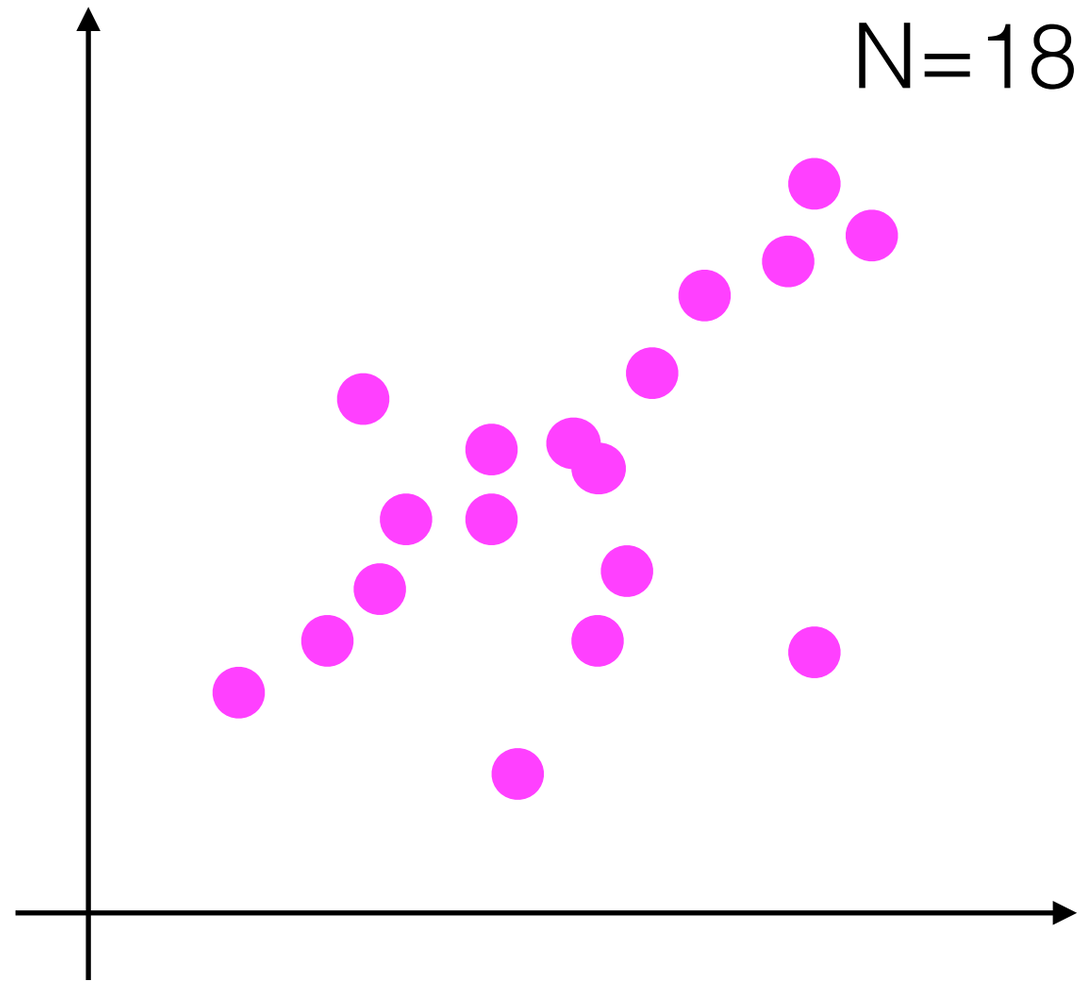
Basic idea:

1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm to estimate a line from n=2 points



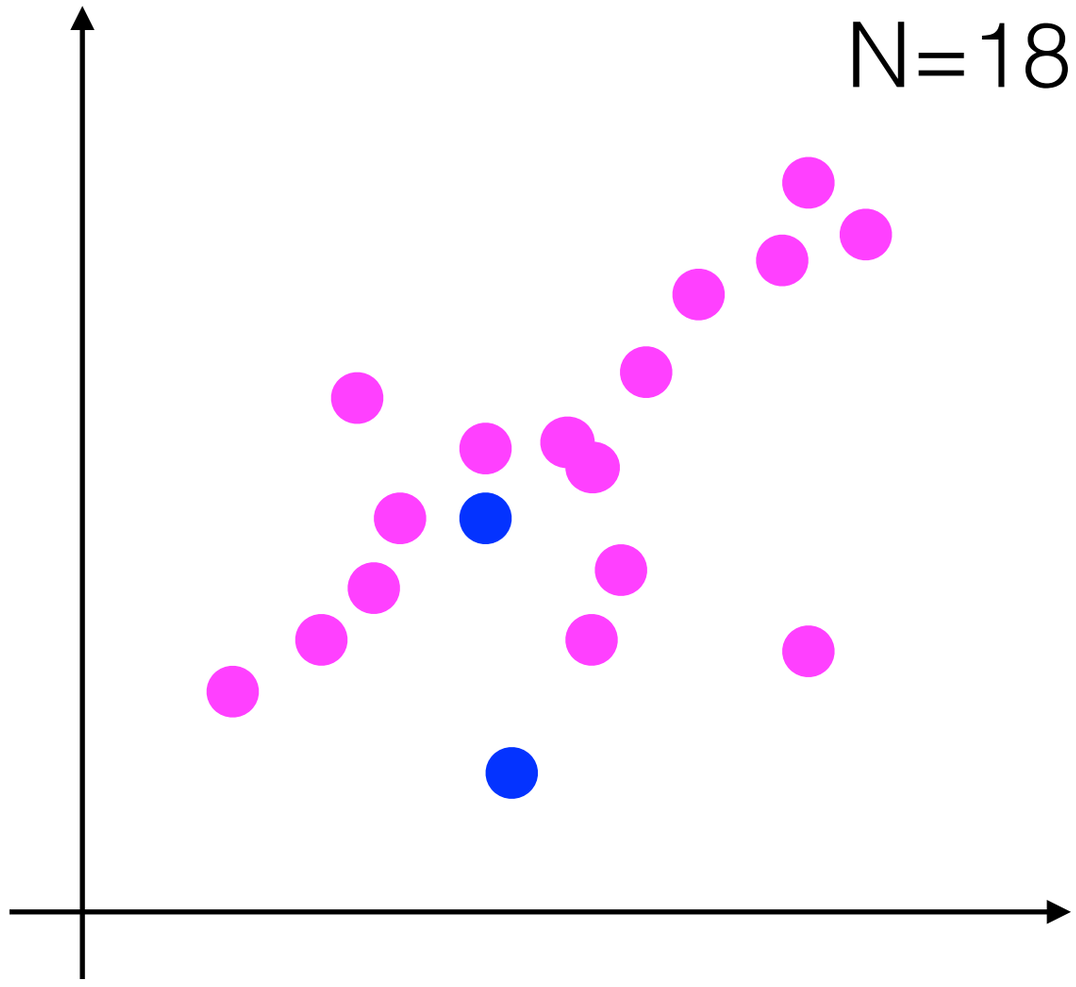
RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm to estimate a line from n=2 points



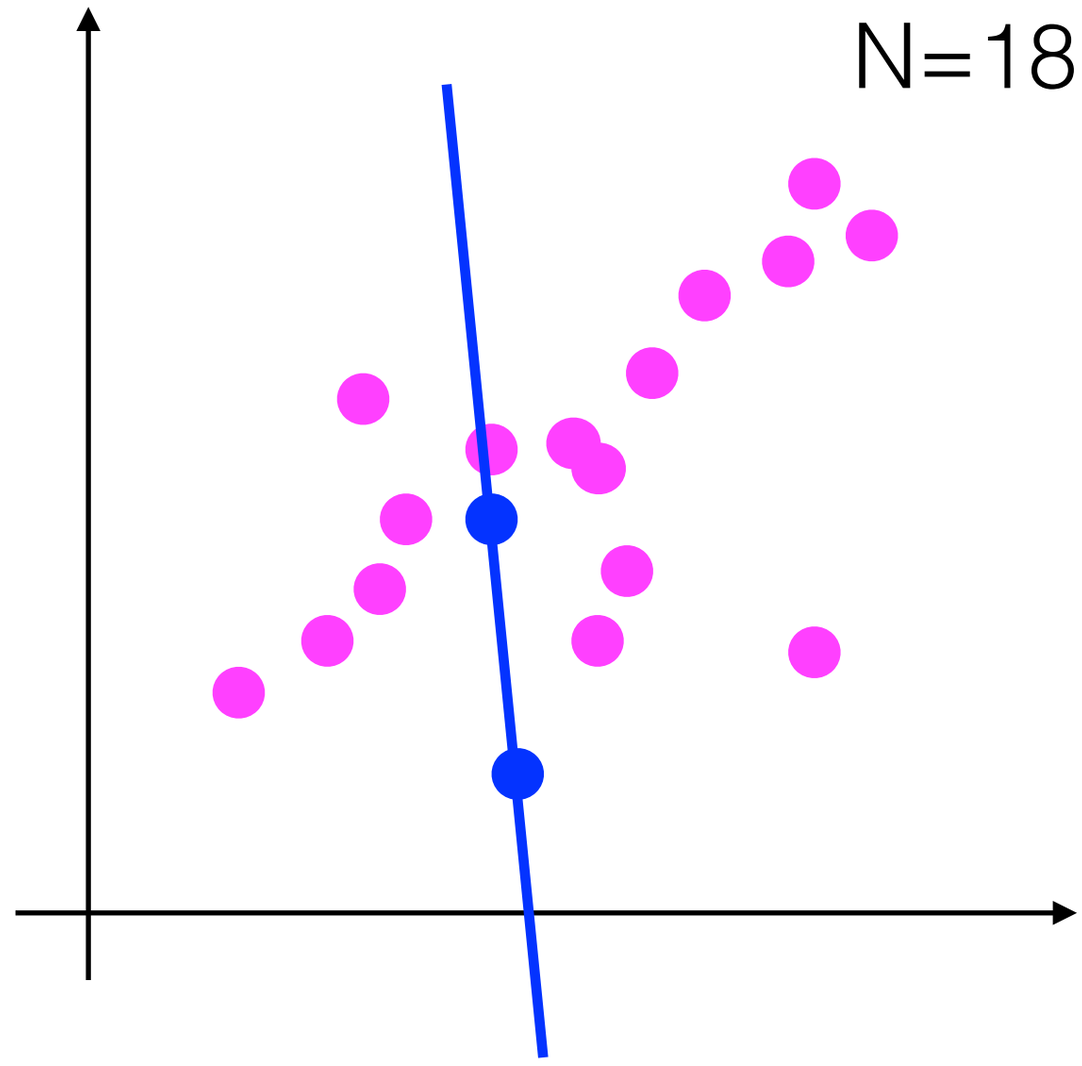
RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm to estimate a line from n=2 points



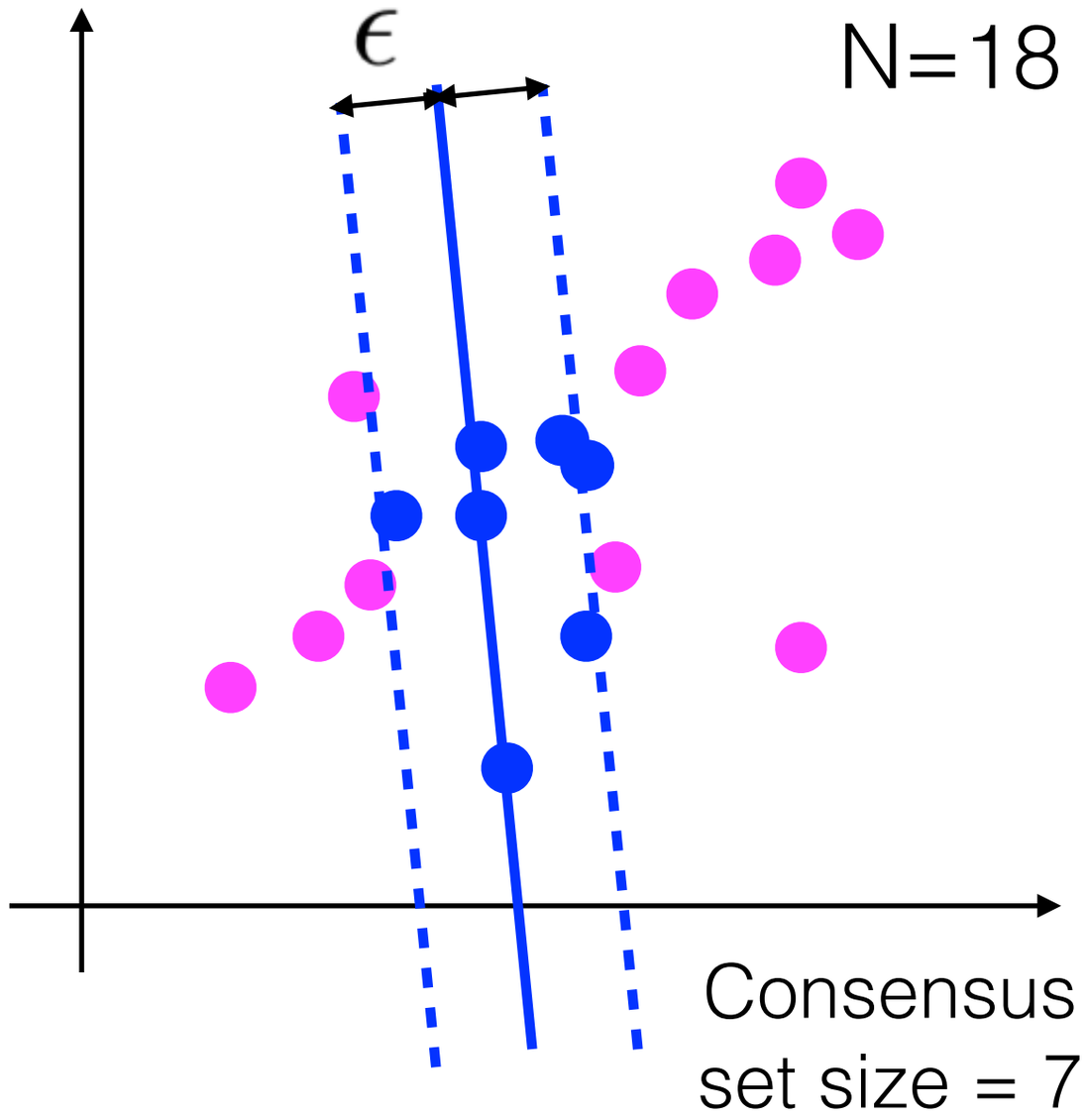
RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm to estimate a line from n=2 points



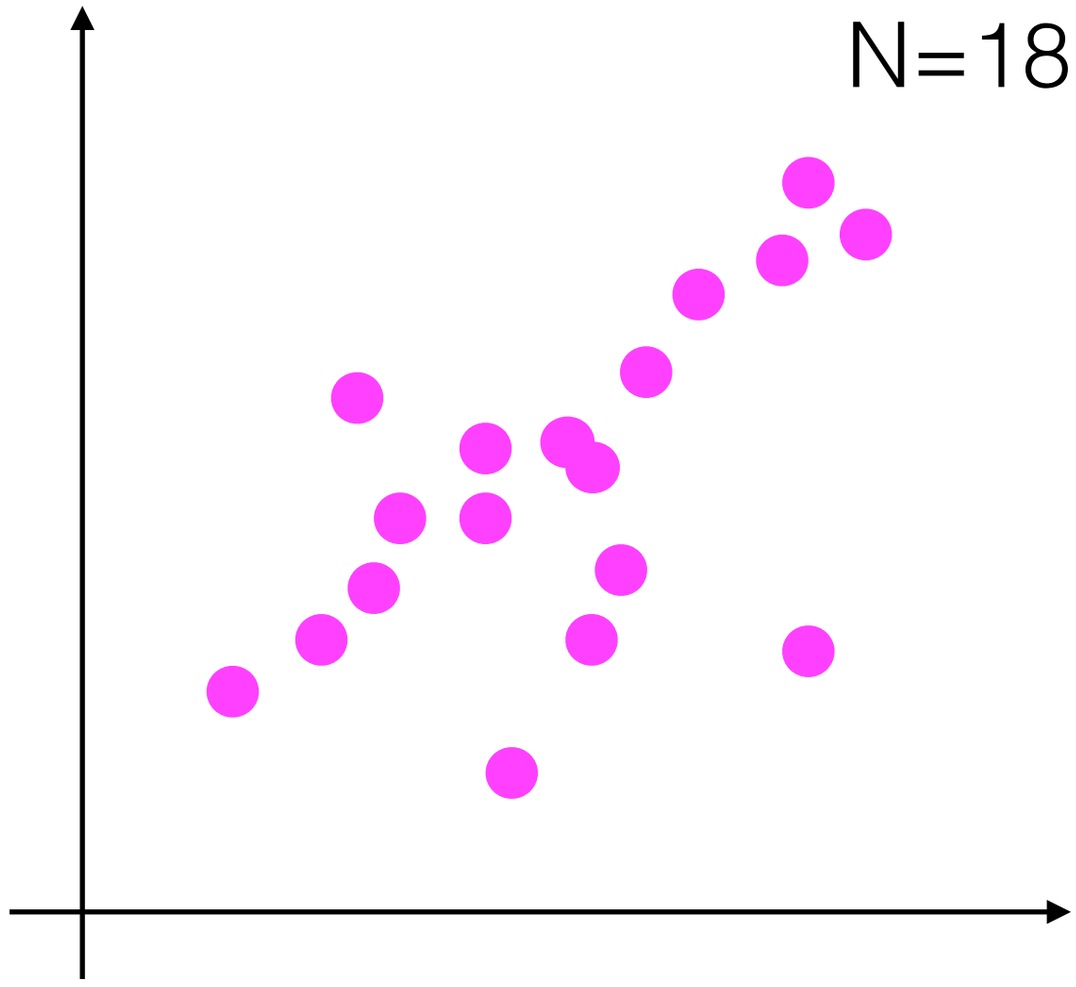
RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm
to estimate a line from n=2 points



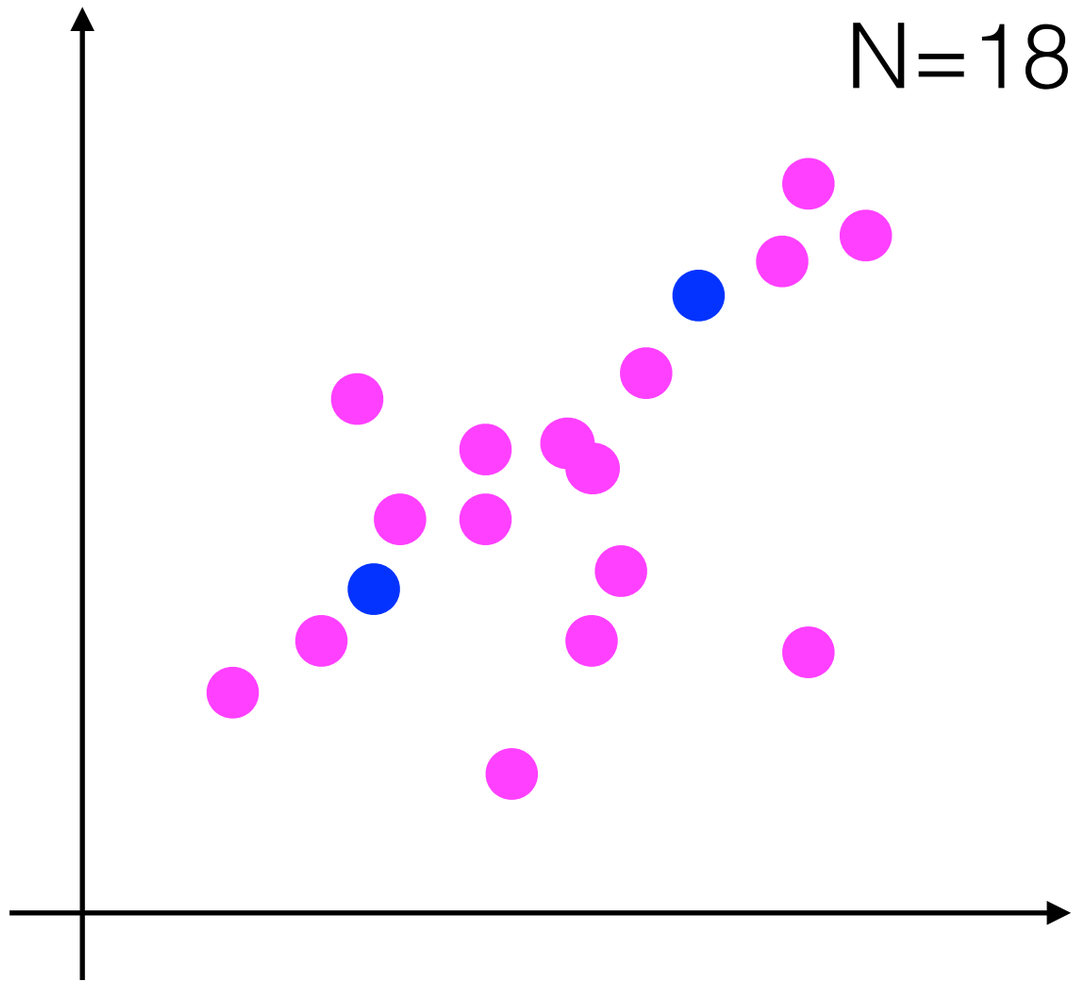
RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm to estimate a line from n=2 points



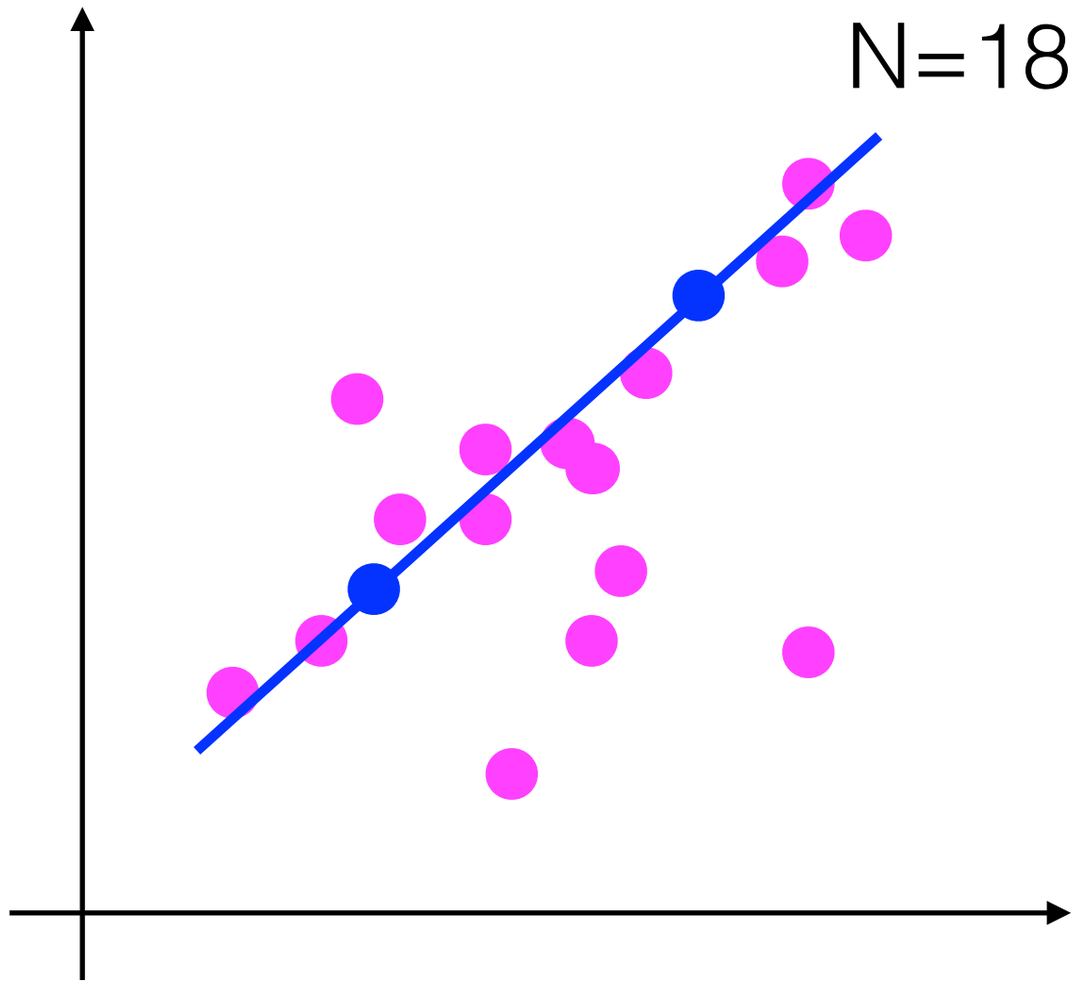
RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm
to estimate a line from n=2 points



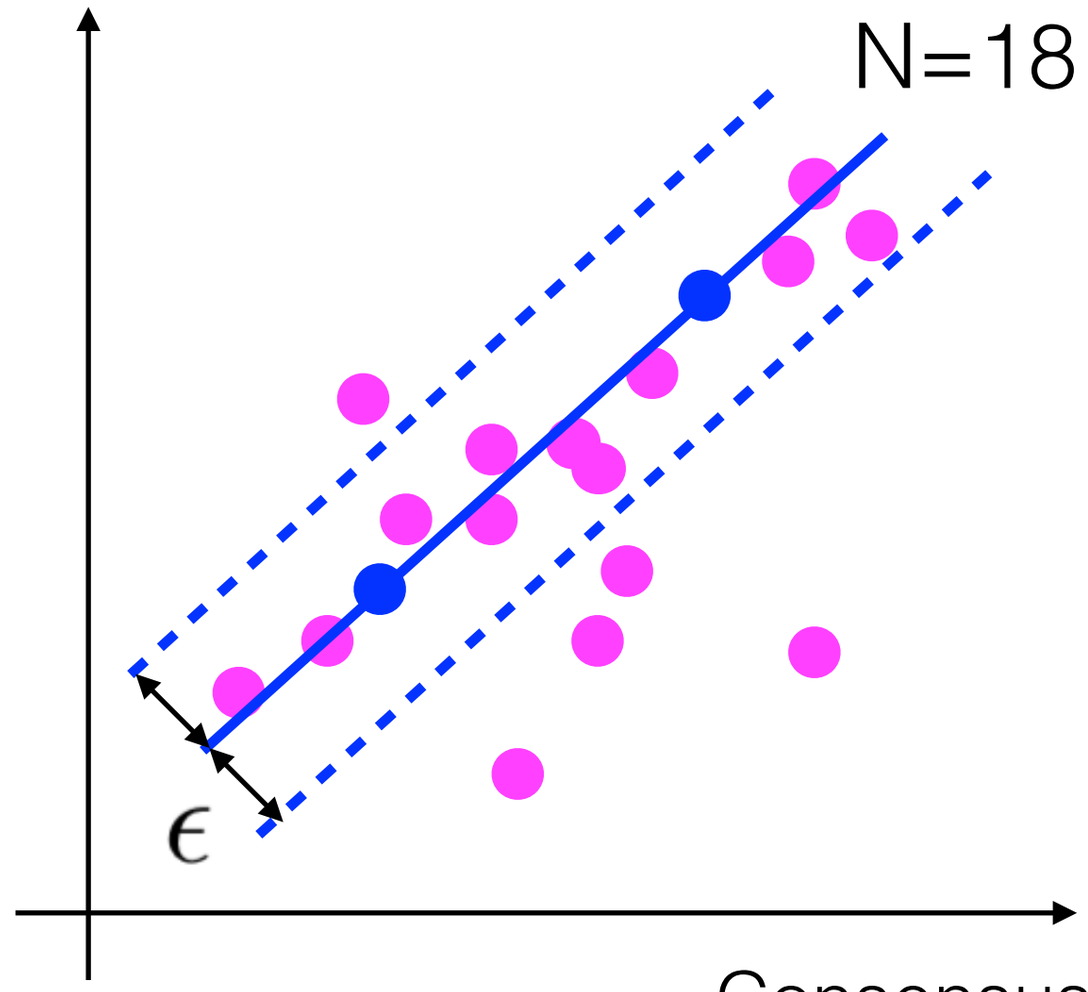
RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

Example: Linear Regression

Fit a line through
N 2D points, possibly
corrupted with outliers.

Note: we have an algorithm
to estimate a line from n=2 points



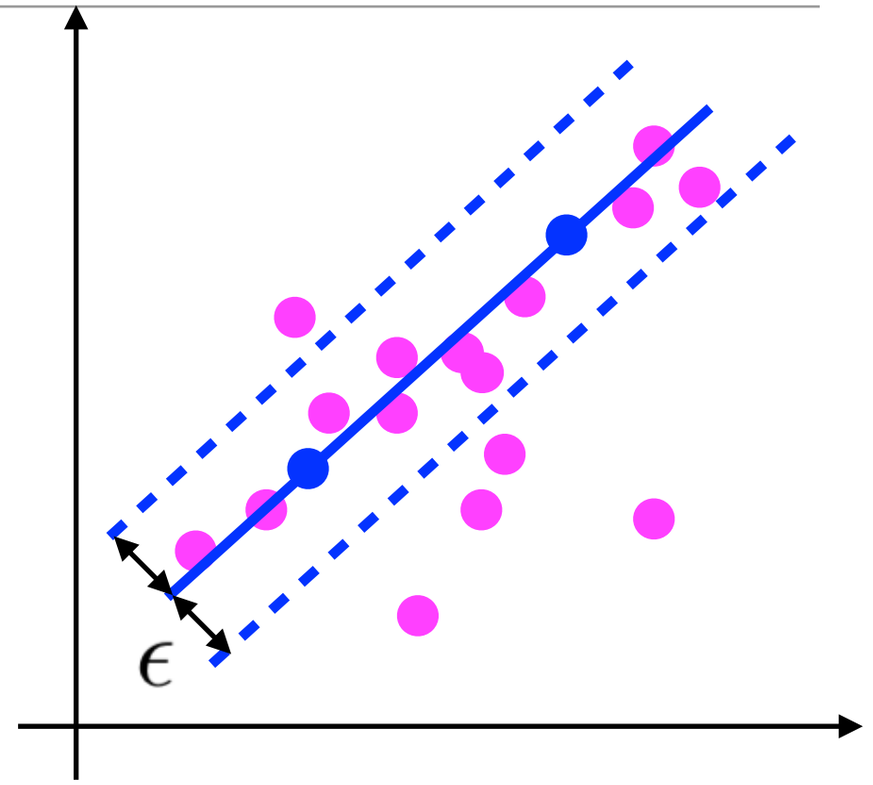
Consensus
set size = 13

RANSAC:

1. sample *2* points
2. compute a line estimate *P'* of *P*
3. count how many points are within a **tolerance** from *P'*
4. repeat until you get a *P'* that agrees with many points

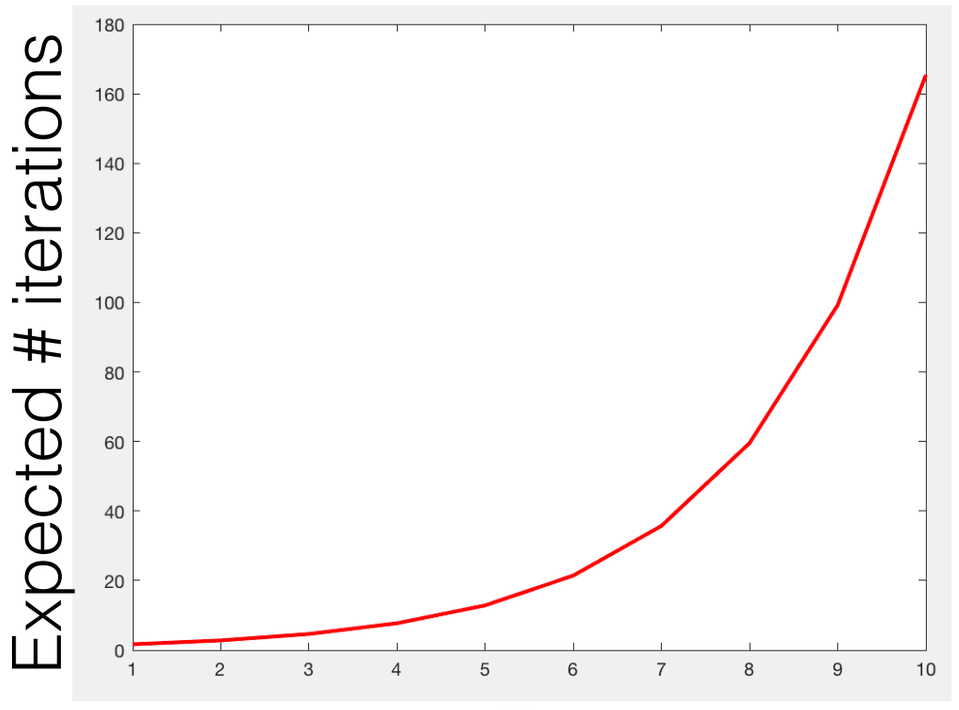
RANSAC: Parameter Tuning

1. **Error Tolerance** ϵ :
   depends on the noise

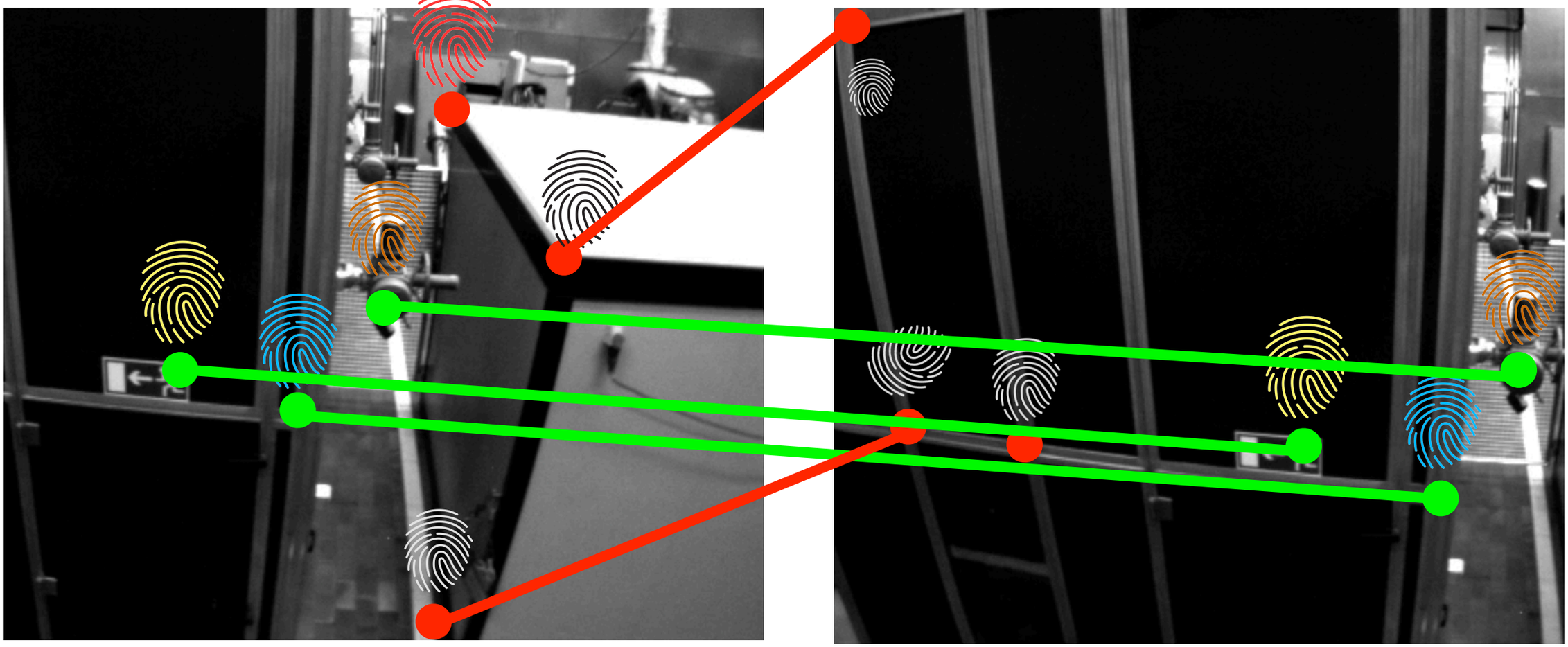


2. **Acceptable consensus set**:
   - from the paper: n+5
   - rule of thumb: >50% of points

3. **Maximum number of iterations**



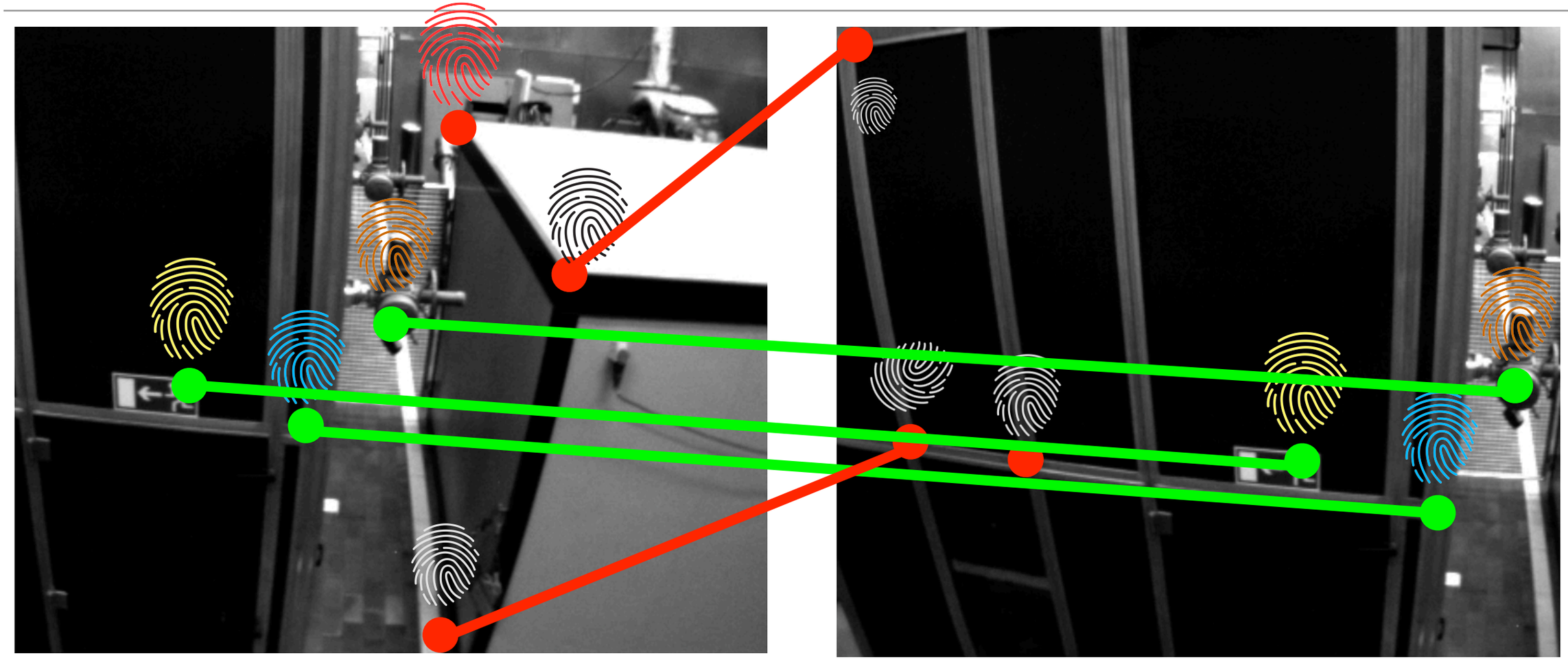
Example: RANSAC for Essential Matrix estimation



RANSAC:

1. sample *n* point correspondences
2. compute an estimate E’ of the essential matrix E
3. count how many points are within a **tolerance** from *E’*
4. repeat until you get a *E’* that agrees with many points

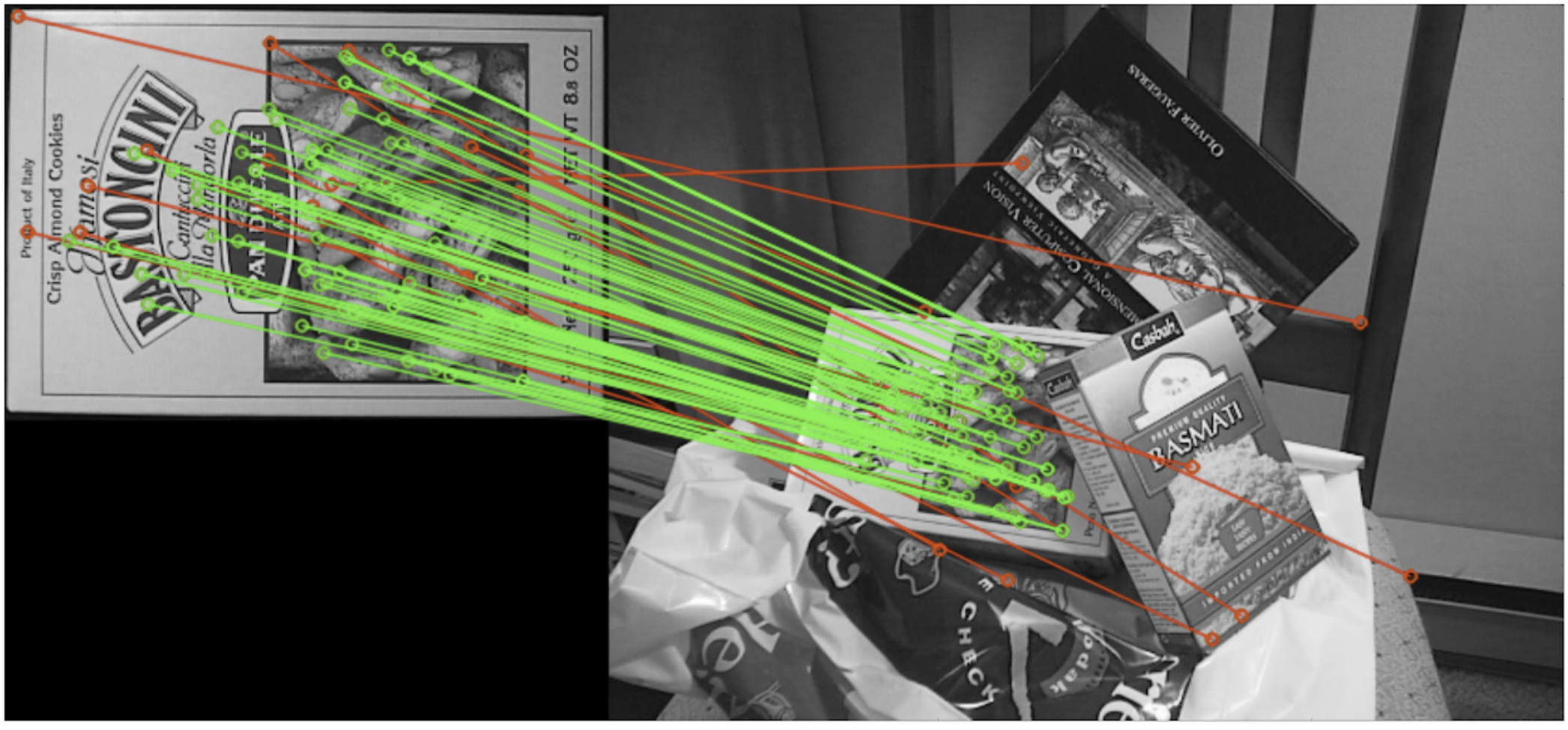
Example: RANSAC for Essential Matrix estimation



RANSAC
- essentially selects the set of inliers
- provides **geometric verification** for the correspondences

Beyond Motion Estimation

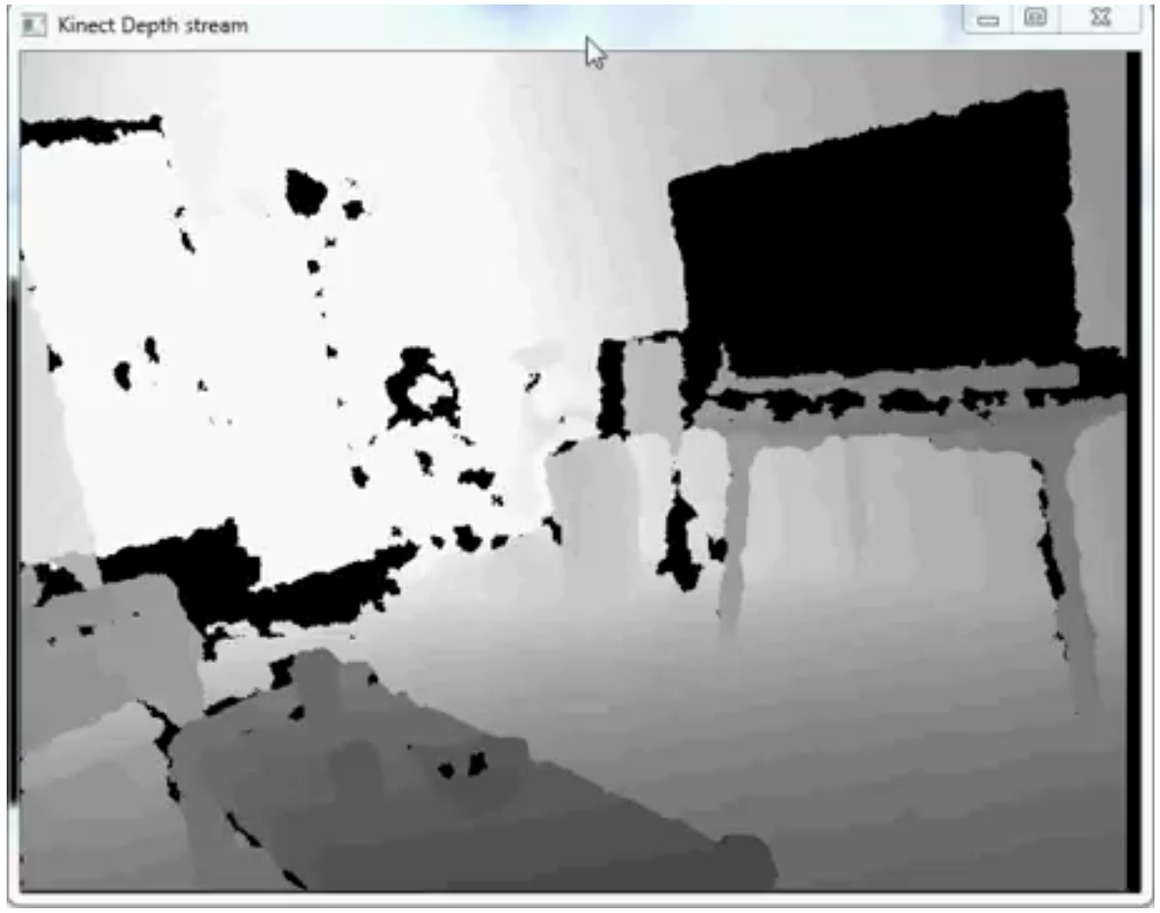
The tools we discussed (feature matching, essential matrix estimation, RANSAC) can be used also for **object detection and localization**



So far: pixel correspondences,
a.k.a., **2D-2D correspondences**

3D-3D Point Correspondences

Structured Light Cameras



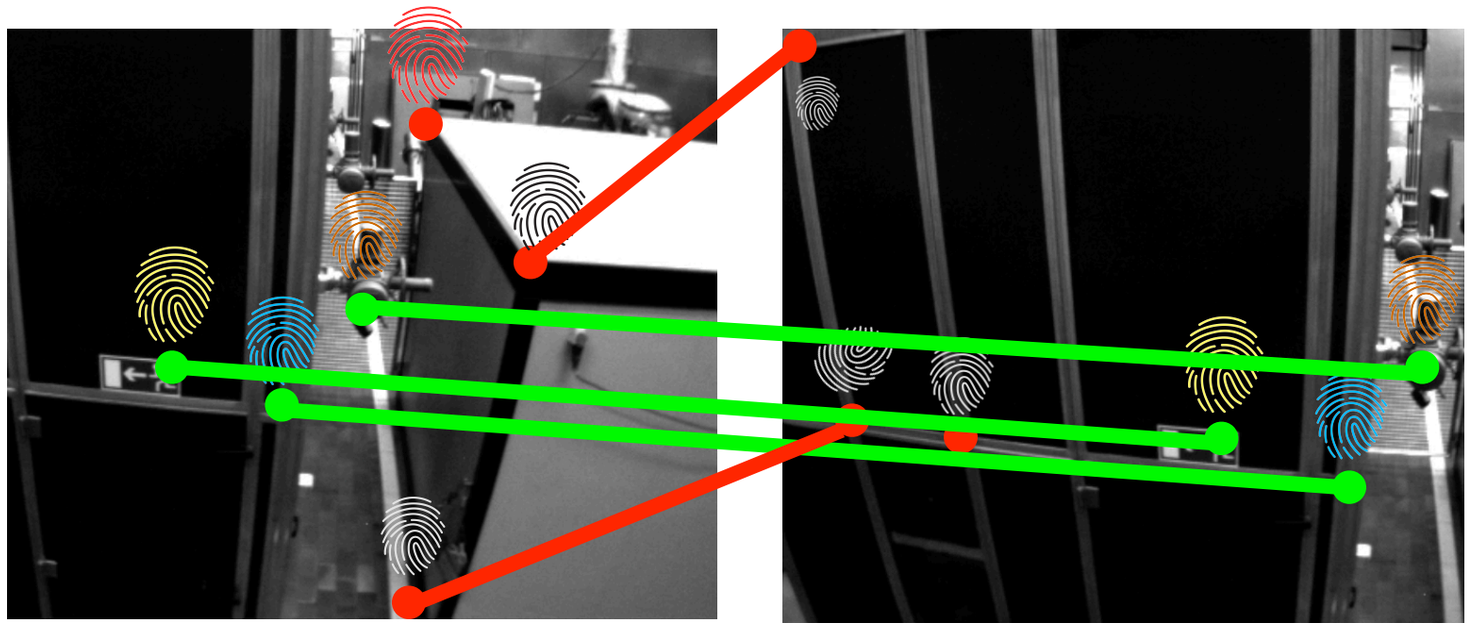
RGB-D cameras can measure depth (D) and image (RBG)

How can we use the depth information to estimate the relative pose between two RGB-D cameras observing the same scene?

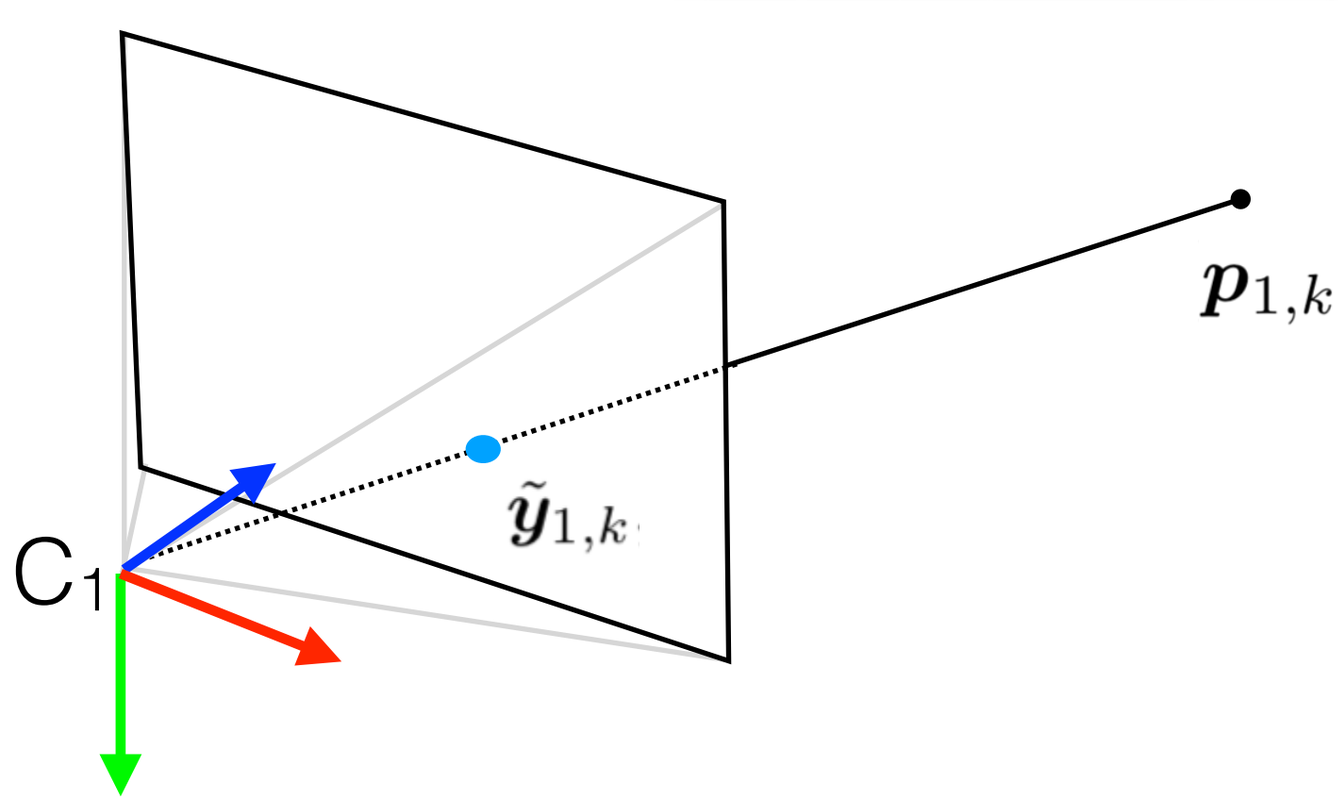
3D-3D Point Correspondences

1. We can use camera images to establish 2D-2D correspondences:

$$(\tilde{\boldsymbol{y}}_{1,k}, \tilde{\boldsymbol{y}}_{2,k}) \text{ for } k = 1, \ldots, N$$

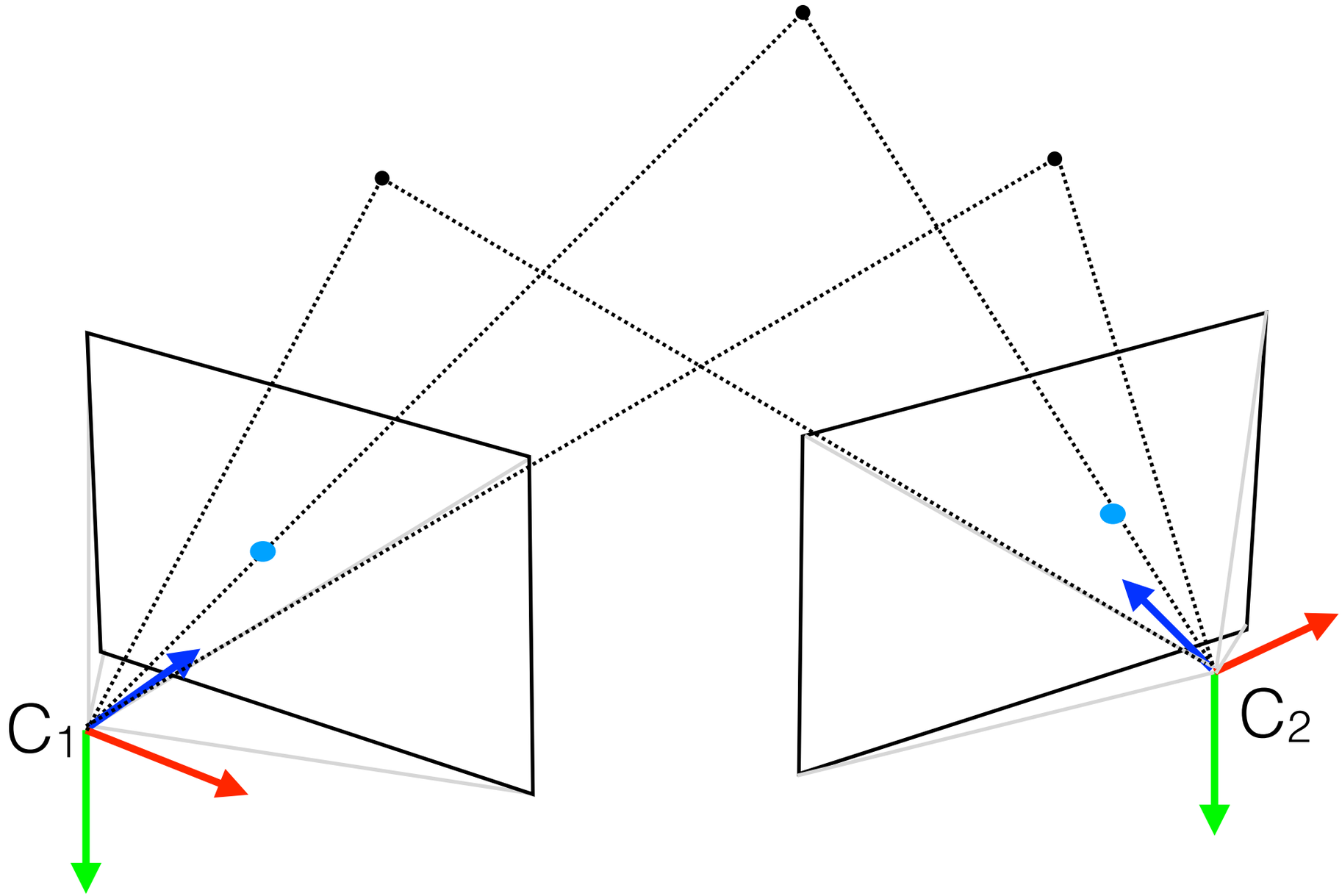


2. For each camera we can compute the set of 3D points corresponding to pixels



We obtain 3D-3D correspondences: $(\boldsymbol{p}_{1,k}, \boldsymbol{p}_{2,k})$ $k = 1, \ldots, N$

2-view Geometry from 3D-3D Correspondences



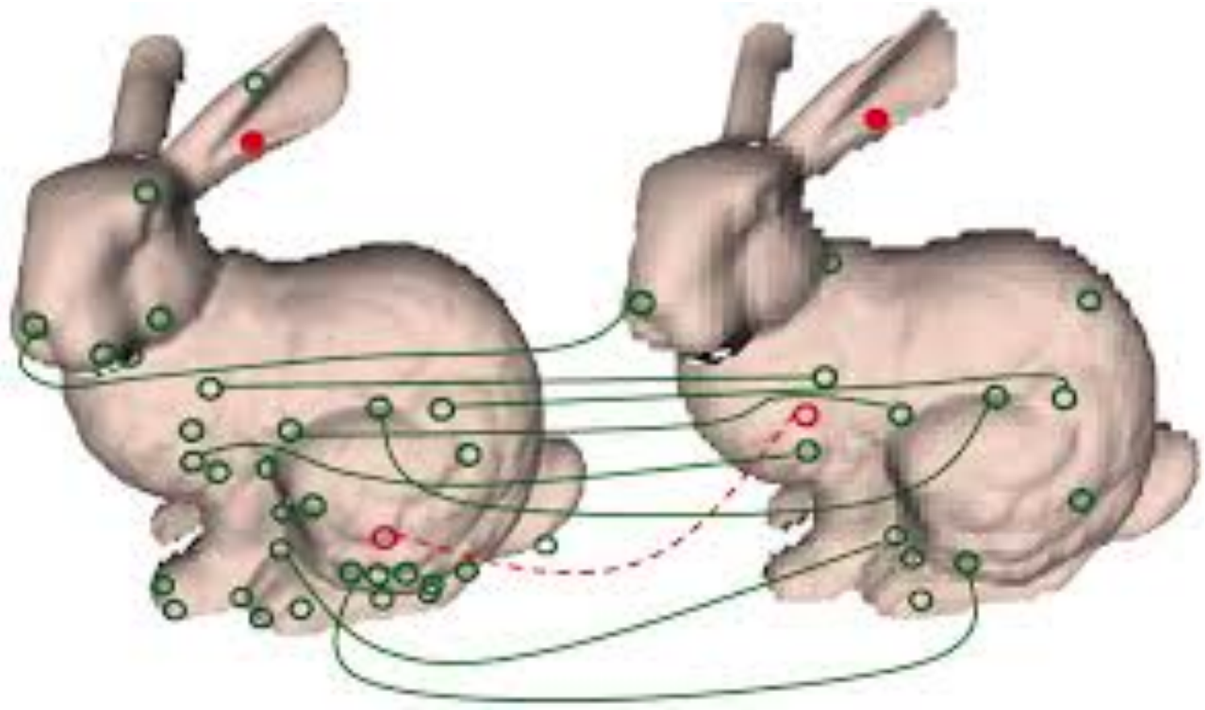
How to estimate the relative pose between the cameras from 3D-3D correspondences $(\boldsymbol{p}_{1,k}, \boldsymbol{p}_{2,k})$ with $k = 1, \ldots, N$?

Few More Comments:

3 points are sufficient to compute the relative pose from 3D-3D correspondences

We can use the solver seen today as a 3-point minimal solver within a **RANSAC** method

Also useful for 3D objects localization:



Other names: vector registration, point cloud alignment, ..