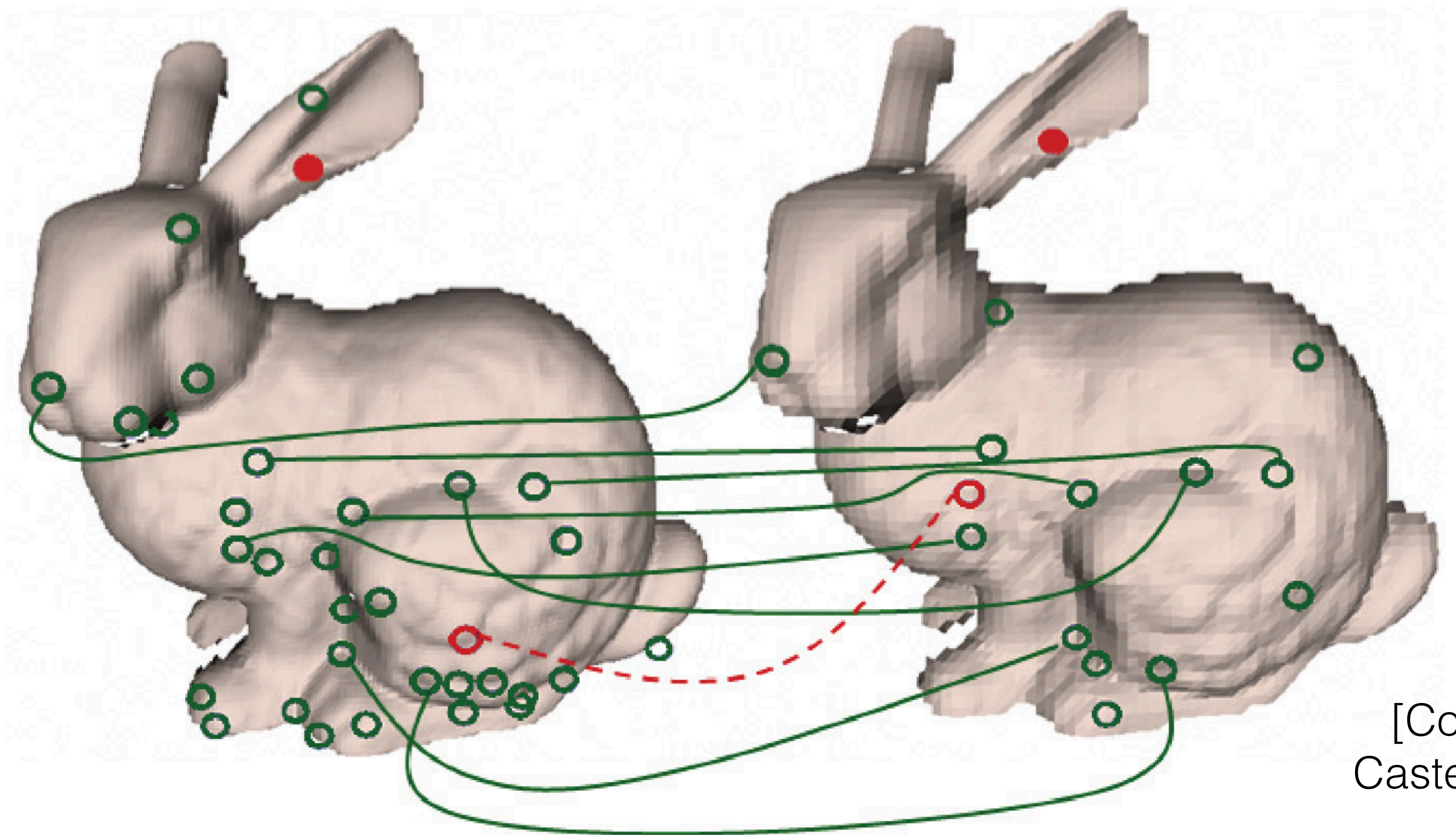


[Courtesy of Castellani et al.]

© Umberto Castellani et al. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/

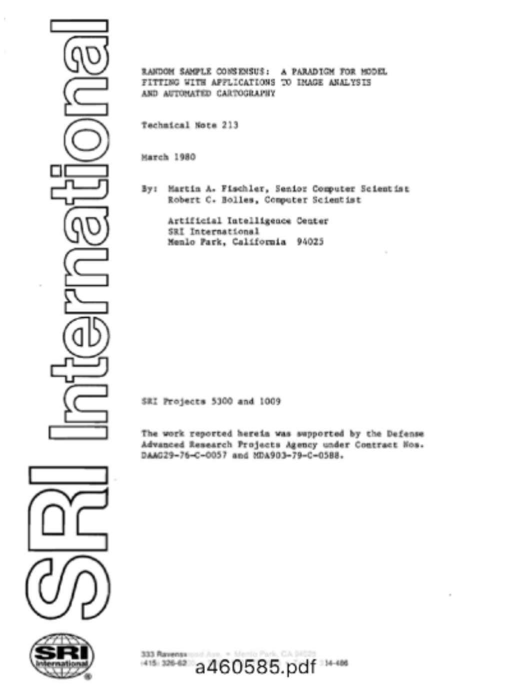
16.485: VNAV - Visual Navigation for Autonomous Vehicles

Luca Carlone

Lecture 15: RANSAC and 3D-3D correspondences

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

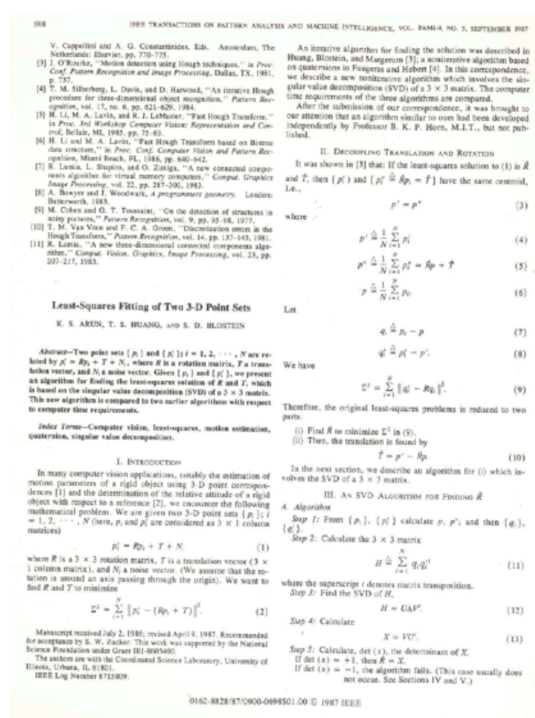
Artificial Intelligence Center
SRI International
Menlo Park, California 94025

SRI Projects 5300 and 1009

The work reported herein was supported by the Defense Advanced Research Projects Agency under Contract Nos. DAAG29-76-C-0057 and MDA903-79-C-0588.

a460585.pdf

[1]



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

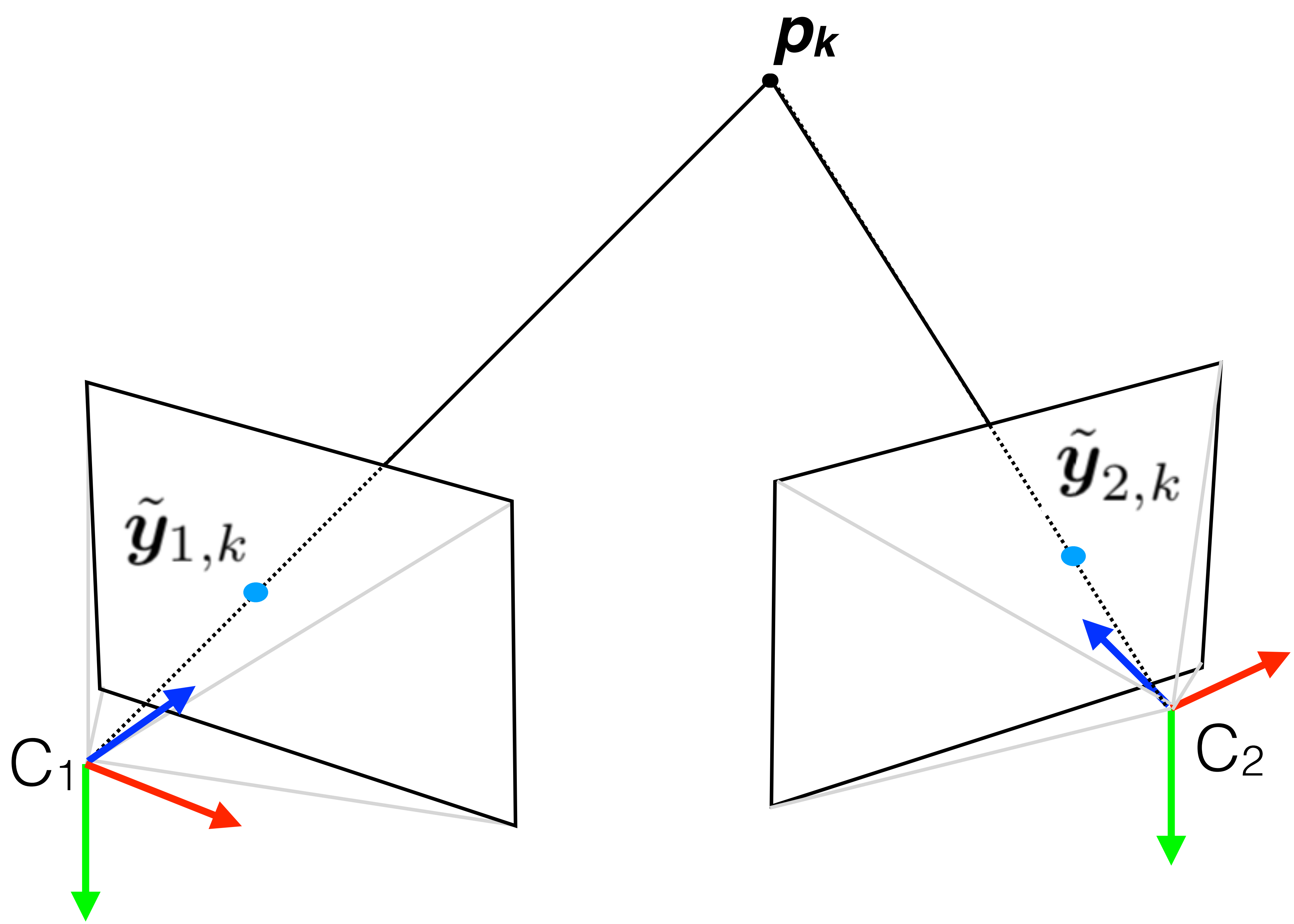
In many computer vision applications, notably the estimation of motion parameters of a rigid object using 3-D point correspondences [1] and the determination of the relative attitude of a rigid object with respect to a reference [2], we encounter the following mathematical problem. We are given two 3-D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

[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.

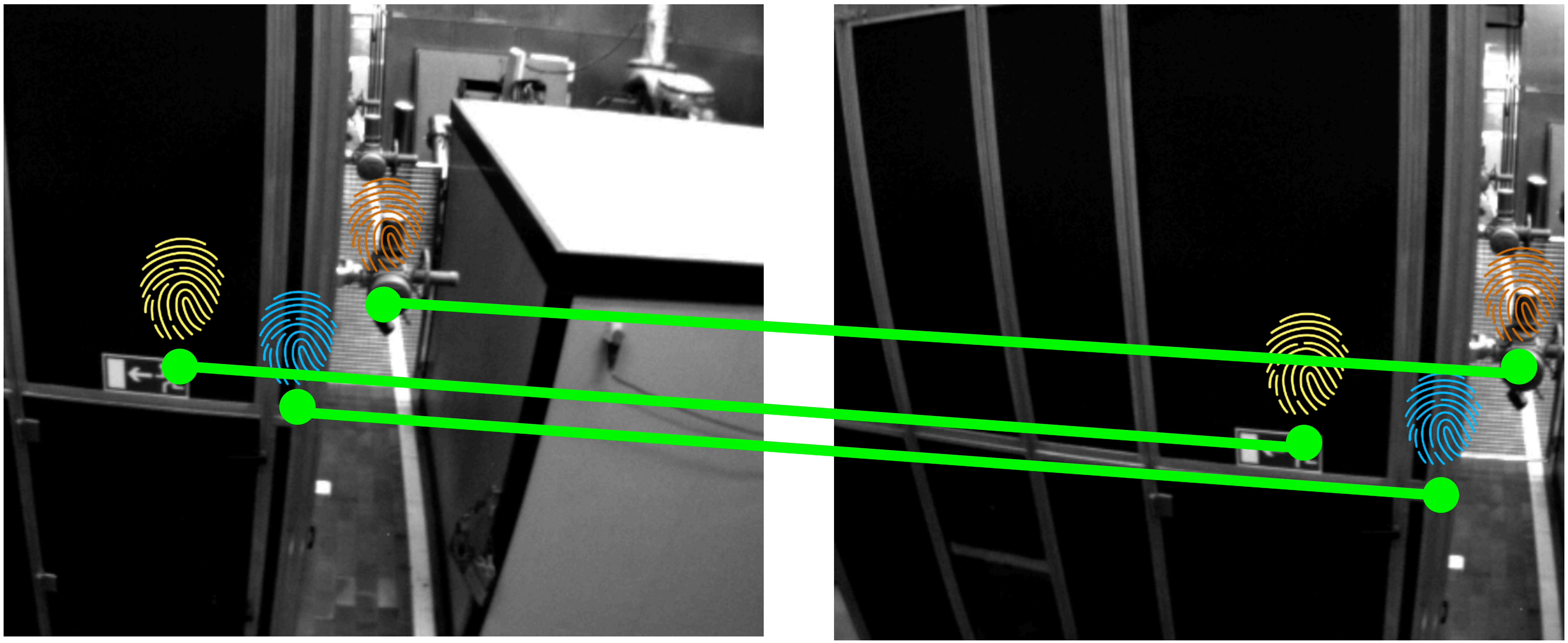
2-view Geometry



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

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

2-view Geometry



Last week's assumptions:

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

Estimating Poses from Correspondences

Given N calibrated pixel correspondences:

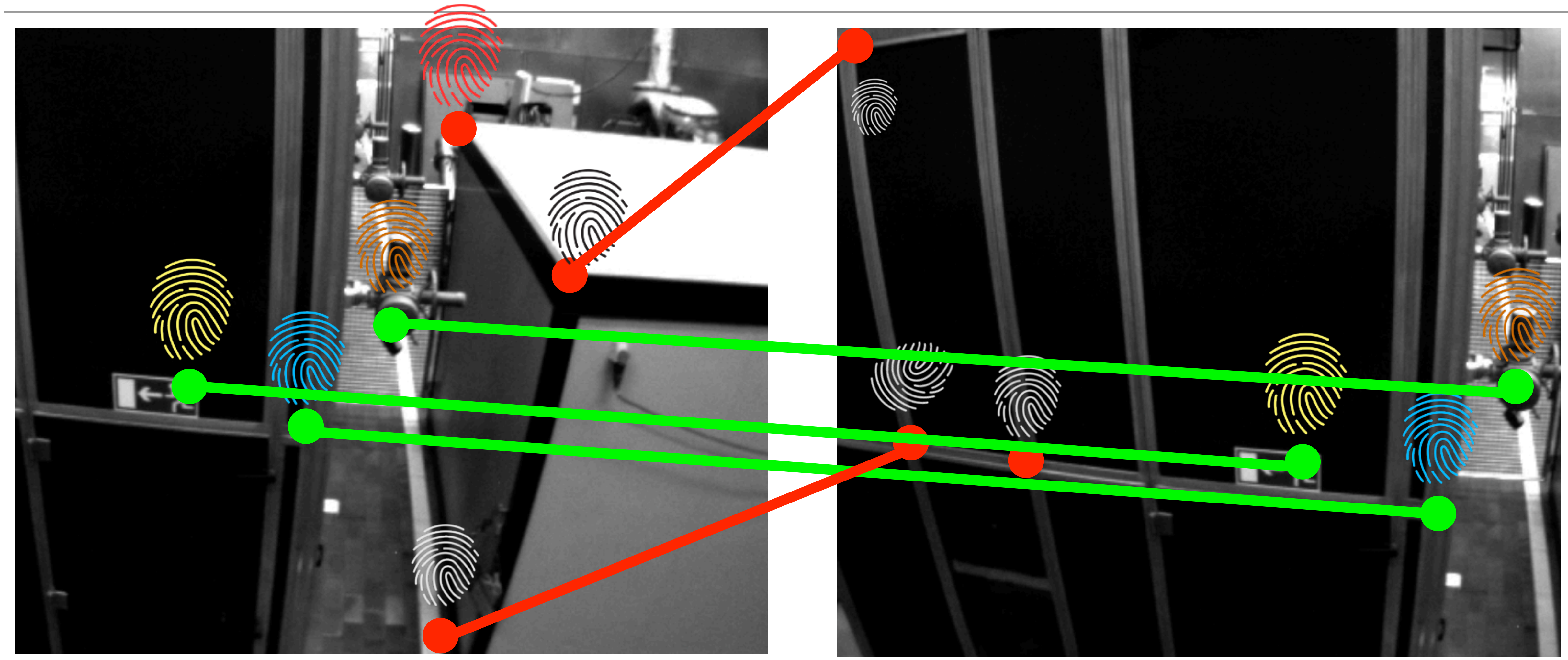
$$(\tilde{\boldsymbol{y}}_{1,k}, \tilde{\boldsymbol{y}}_{2,k}) \text{ for } k = 1, \ldots, N$$

1. leverage the epipolar constraints to estimate the essential matrix ***E*** $\quad \tilde{\boldsymbol{y}}_{2,k}^{\mathsf{T}} \boldsymbol{E} \, \tilde{\boldsymbol{y}}_{1,k} = 0$

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

2-view Geometry



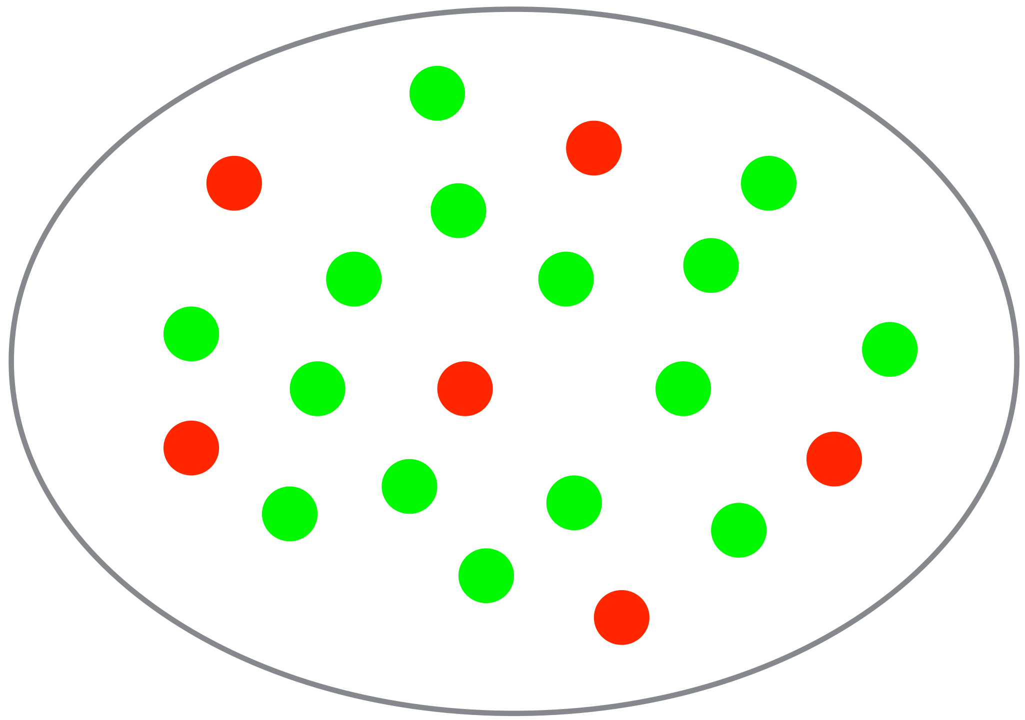
In practice:

- 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.



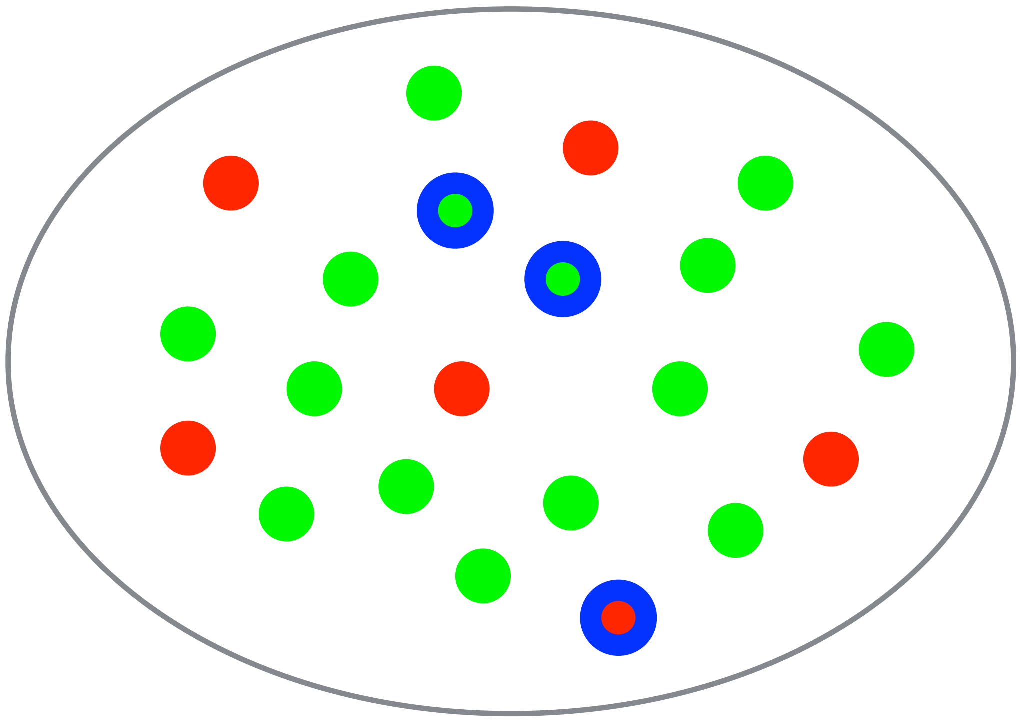
Assume: we have an algorithm to estimate *P* from *n* data points ($n << N$)

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:

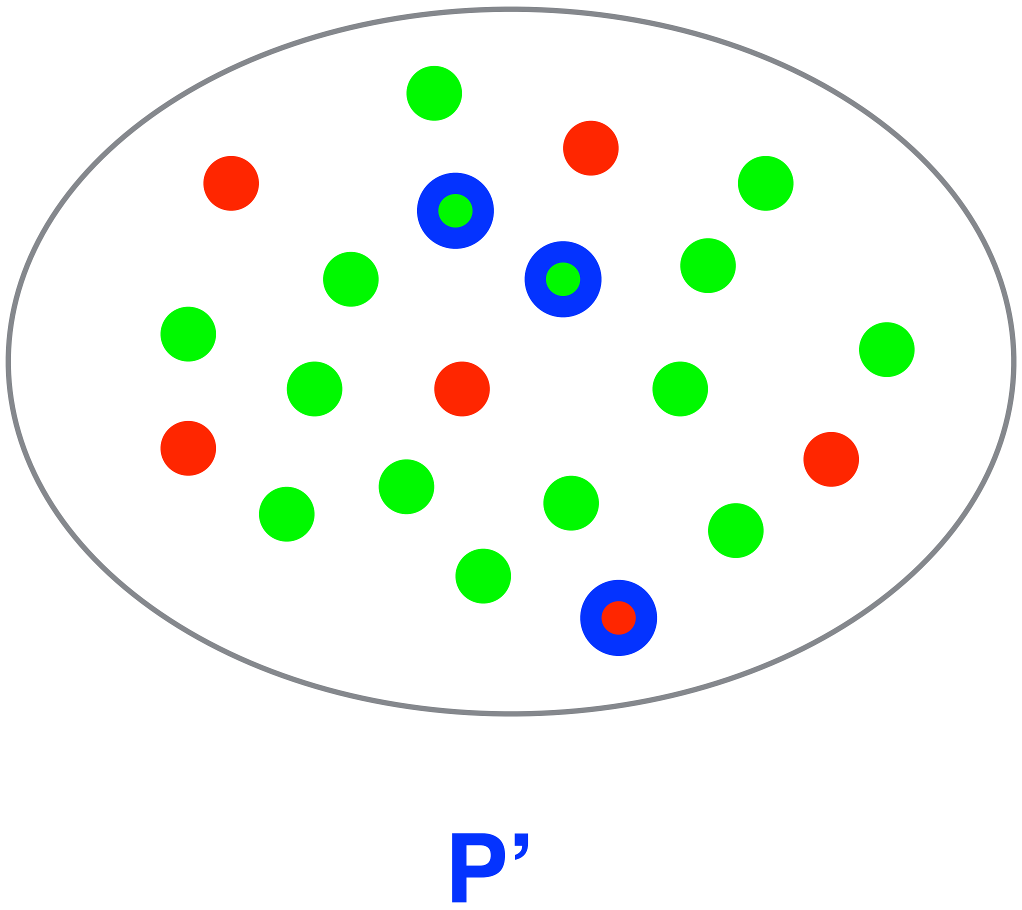
1. sample *n* points
2. compute an estimate *P’* of *P*
3. count how many other points agree with *P’*
4. repeat until you get a *P’* that agrees with many points

RANSAC

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

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

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



Basic idea:

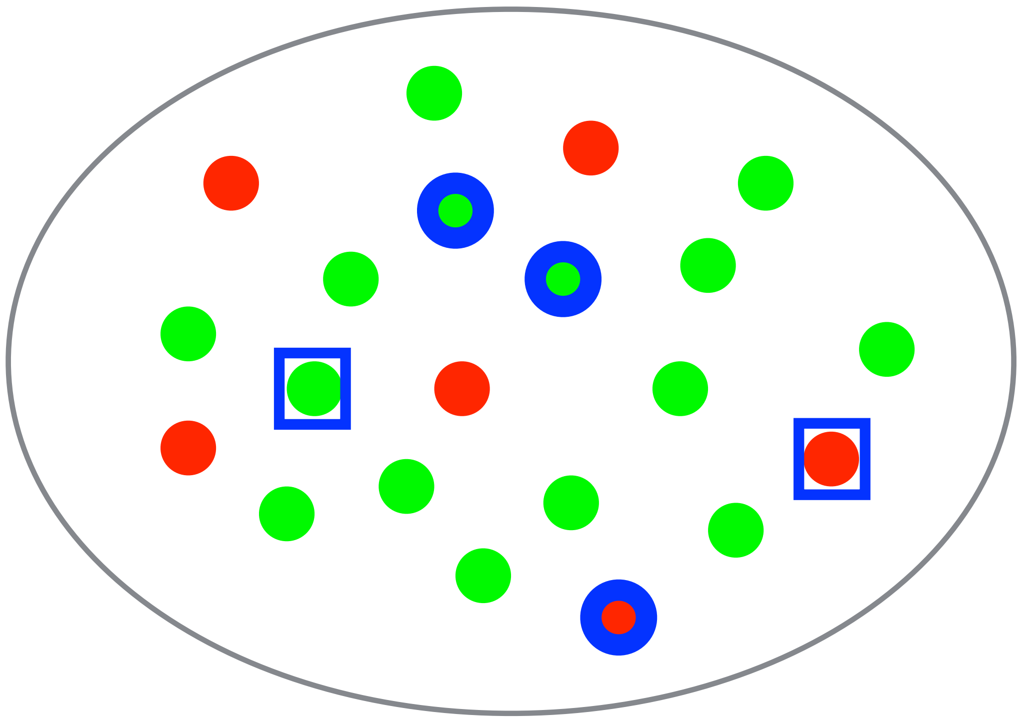
1. sample *n* points
2. compute an estimate *P’* of *P*
3. count how many other points agree with *P’*
4. repeat until you get a *P’* that agrees with many points

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'

Consensus Set

Basic idea:

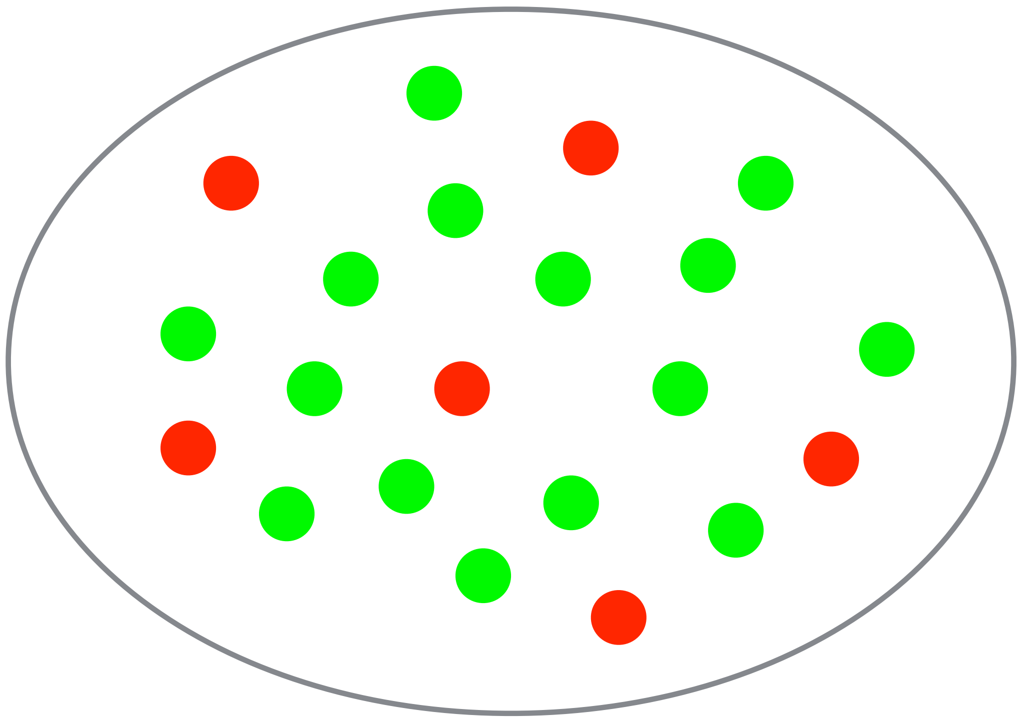
1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

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)

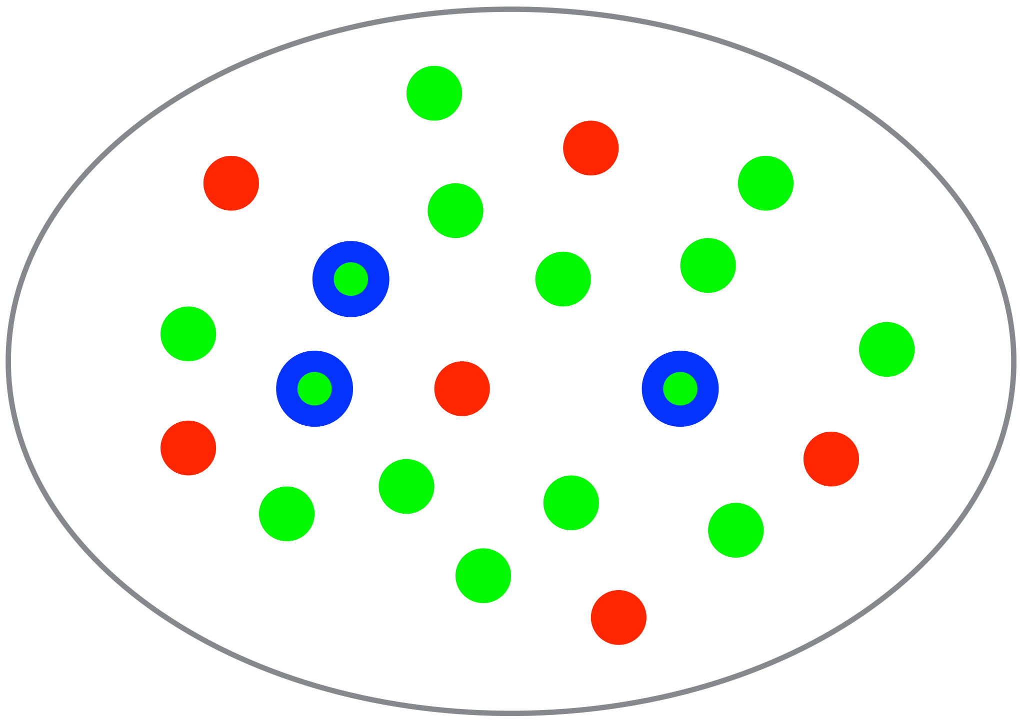


Basic idea:

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.

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

Basic idea:

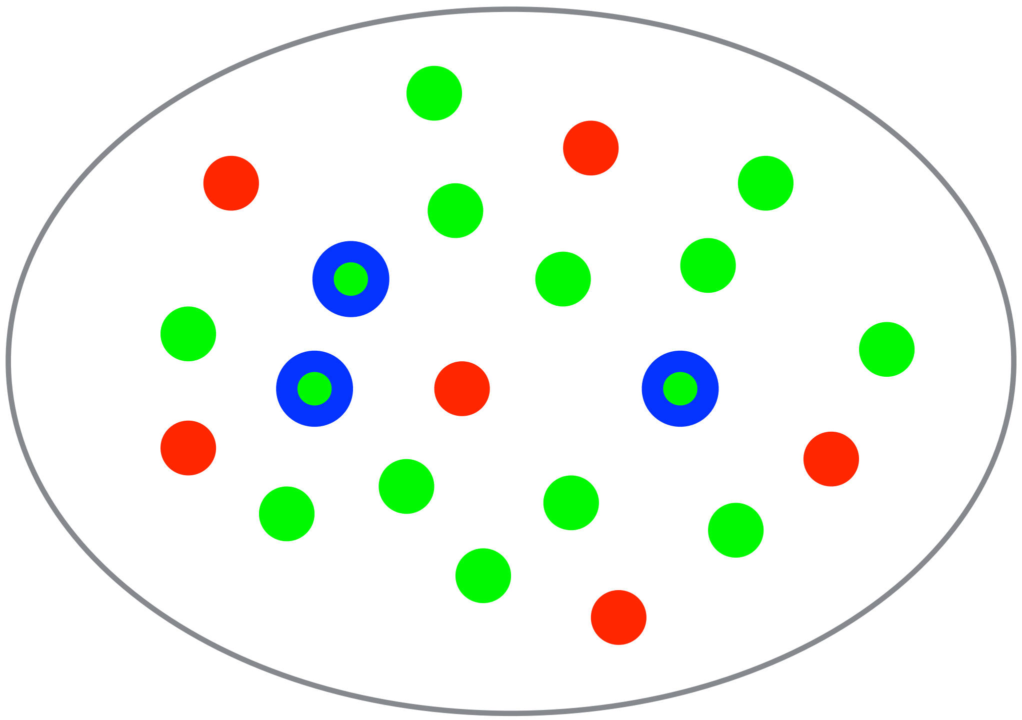
1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

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:

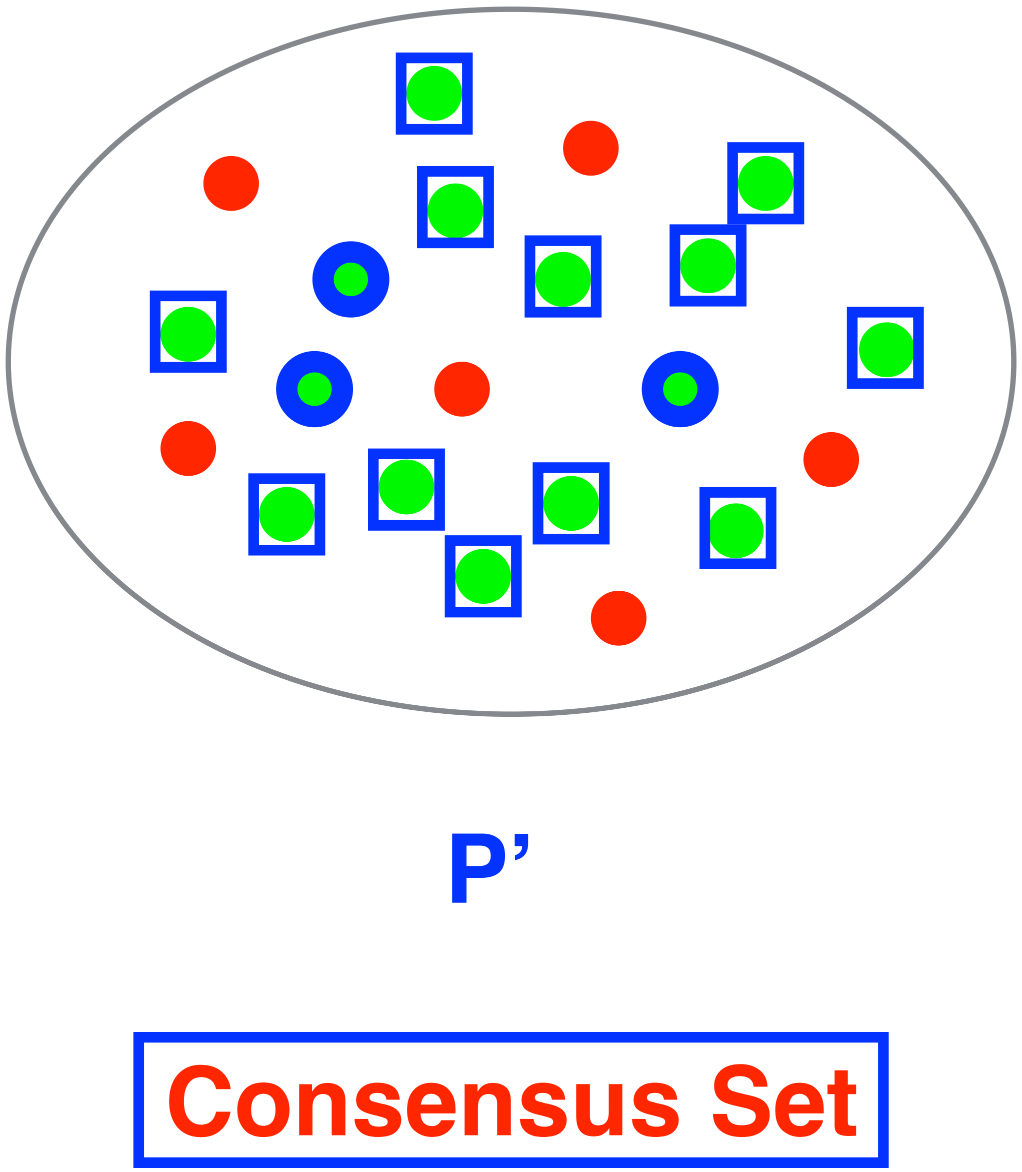
1. sample *n* points
2. compute an estimate *P'* of *P*
3. count how many other points agree with *P'*
4. repeat until you get a *P'* that agrees with many points

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)



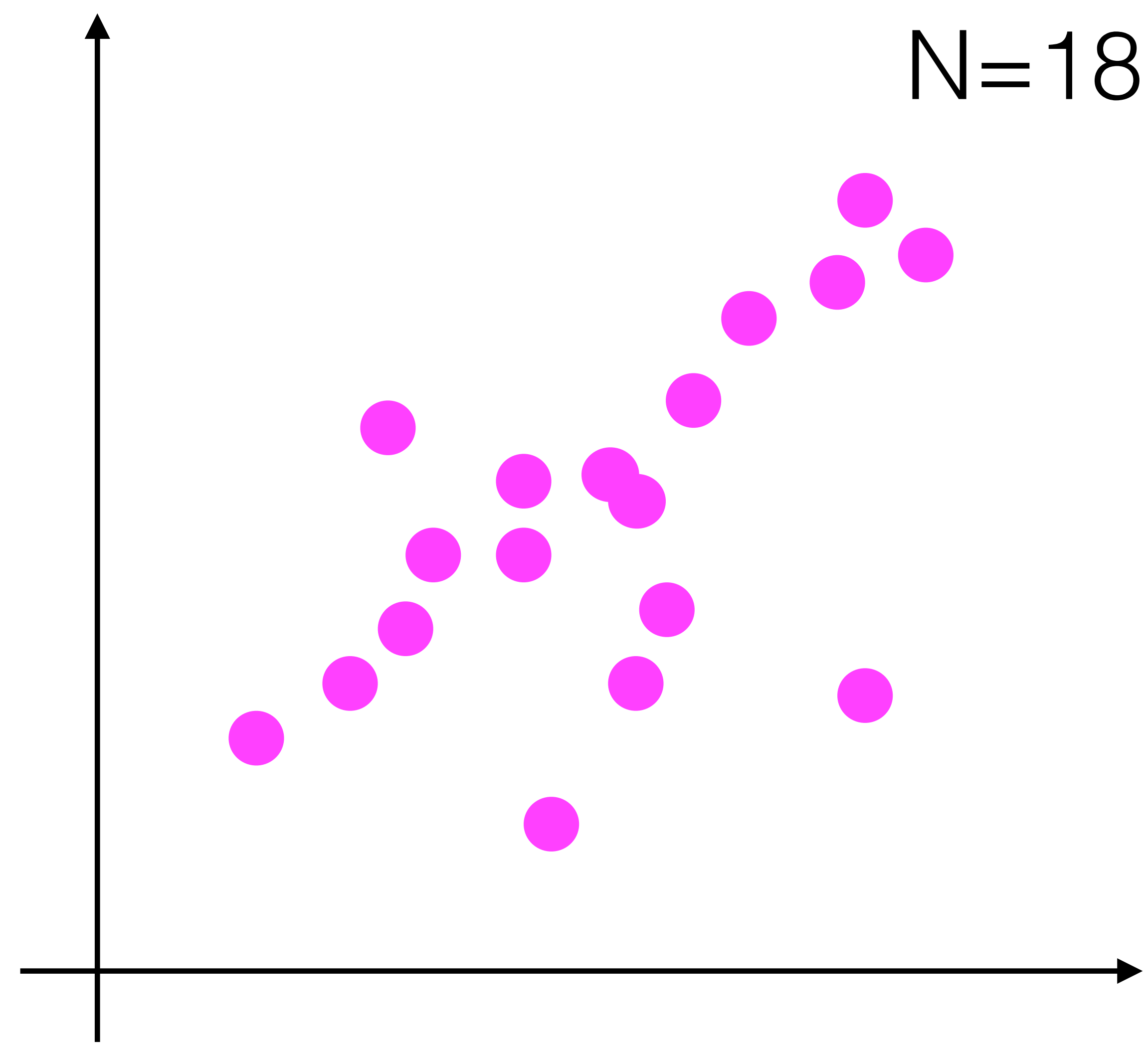
Basic idea:

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



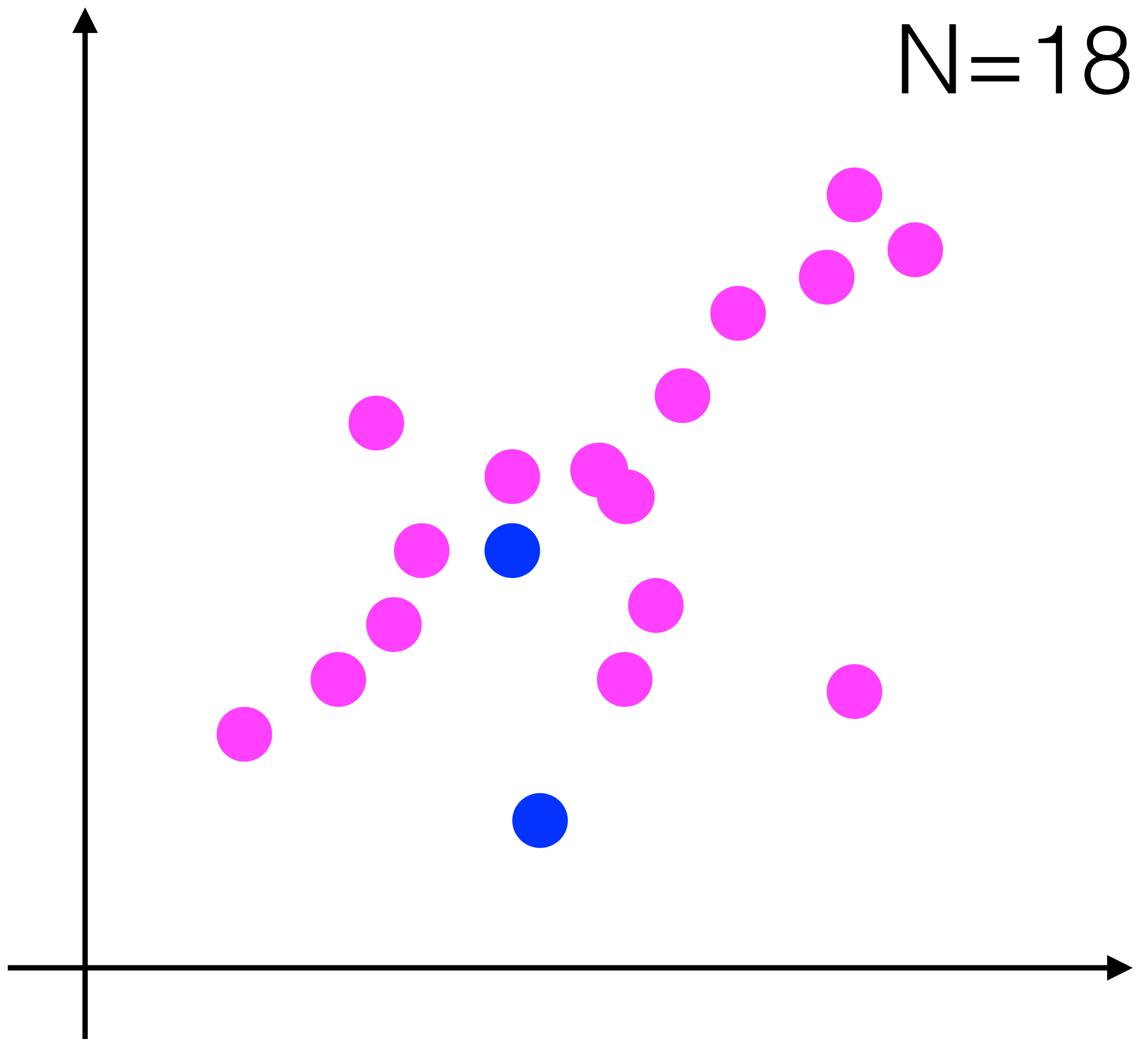
RANSAC:

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



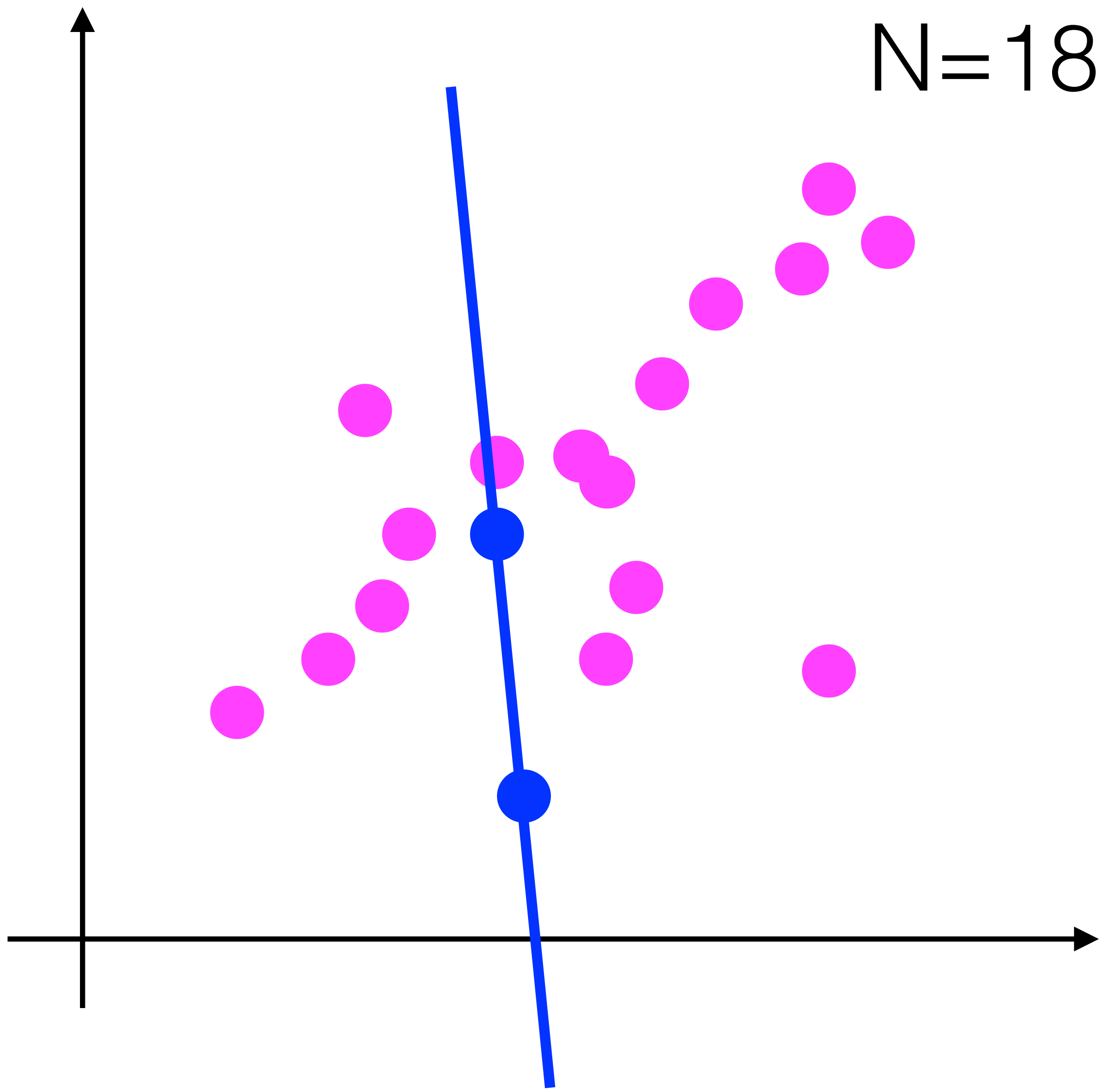
RANSAC:

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



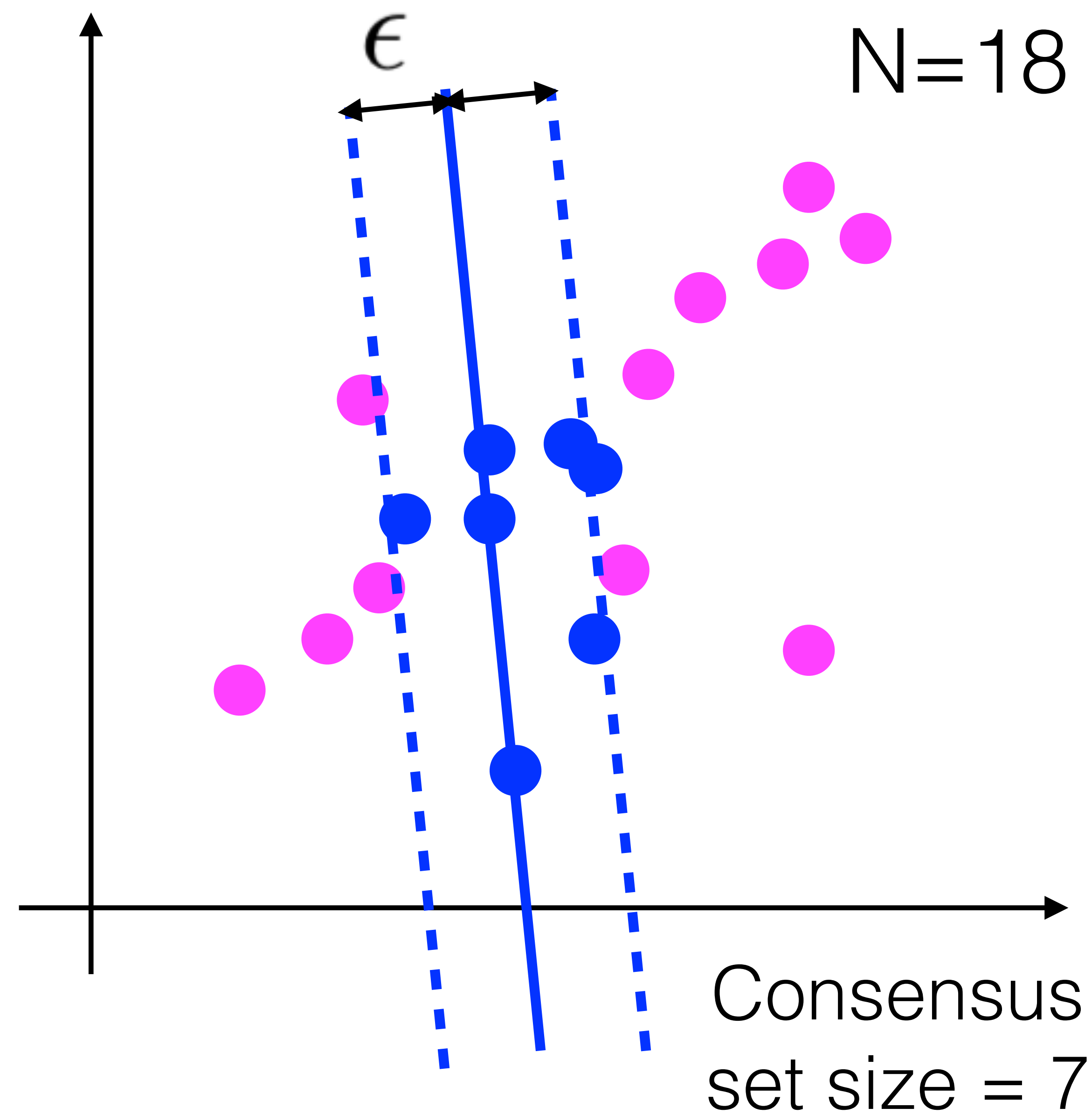
RANSAC:

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



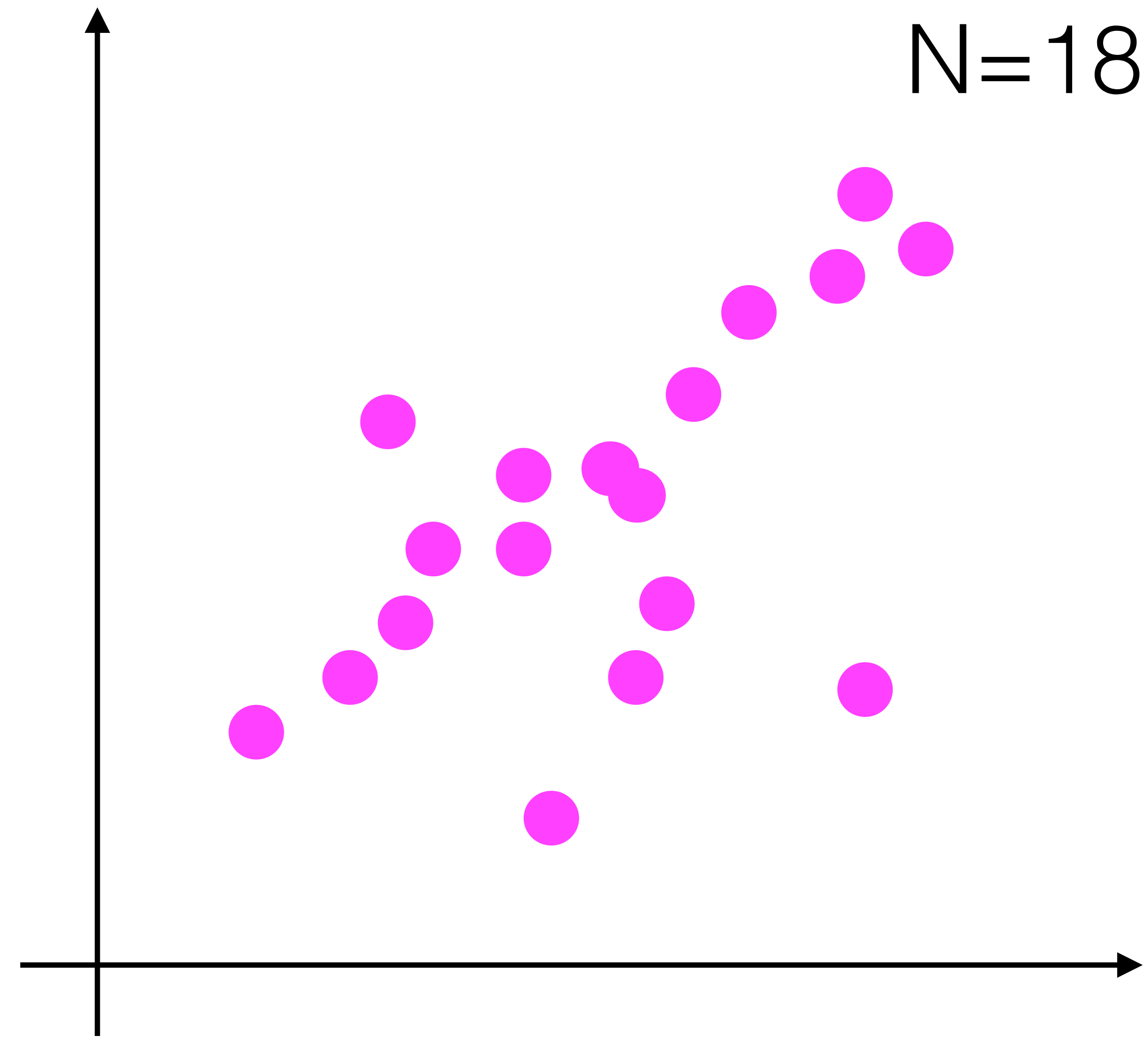
RANSAC:

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



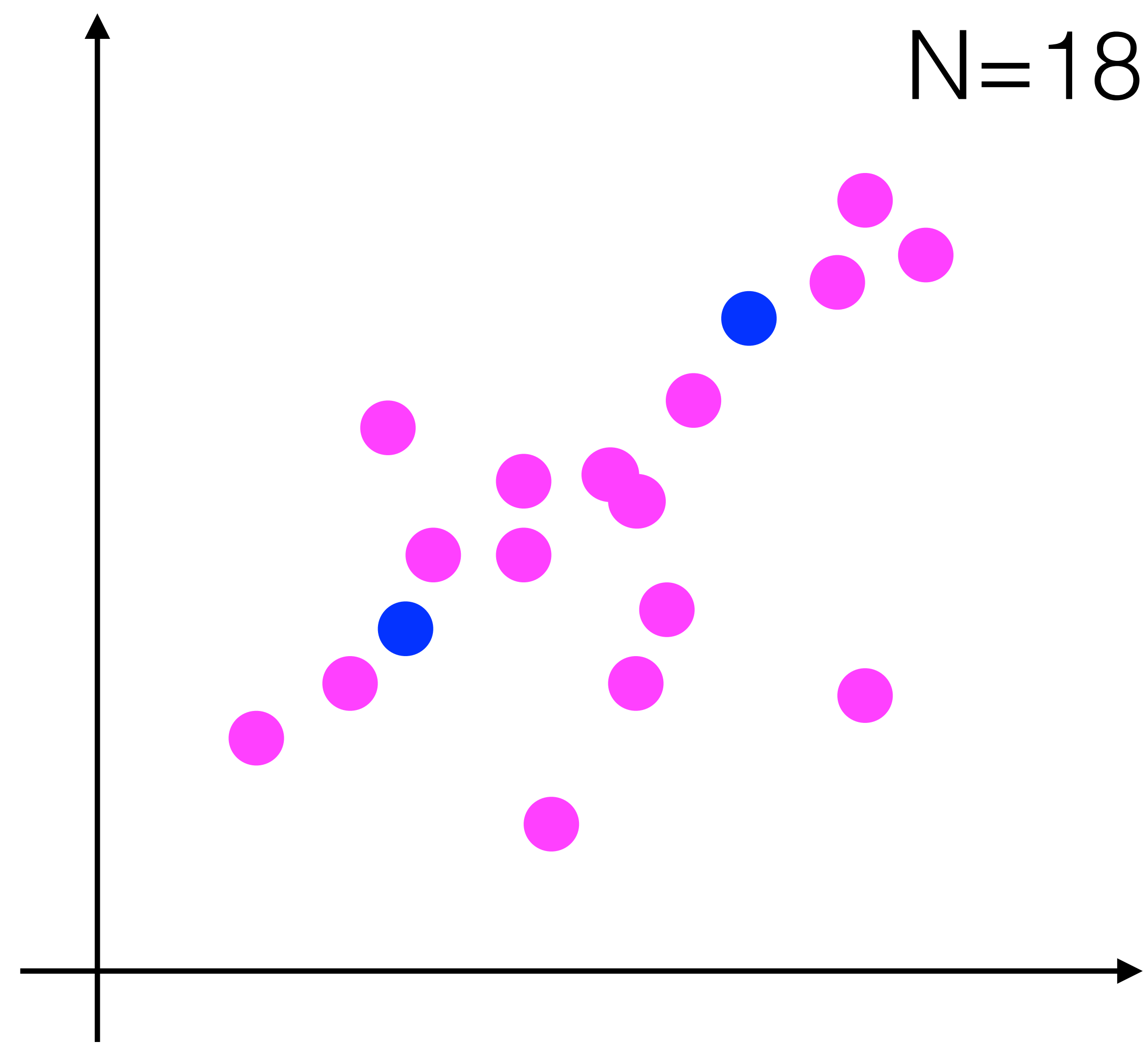
RANSAC:

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



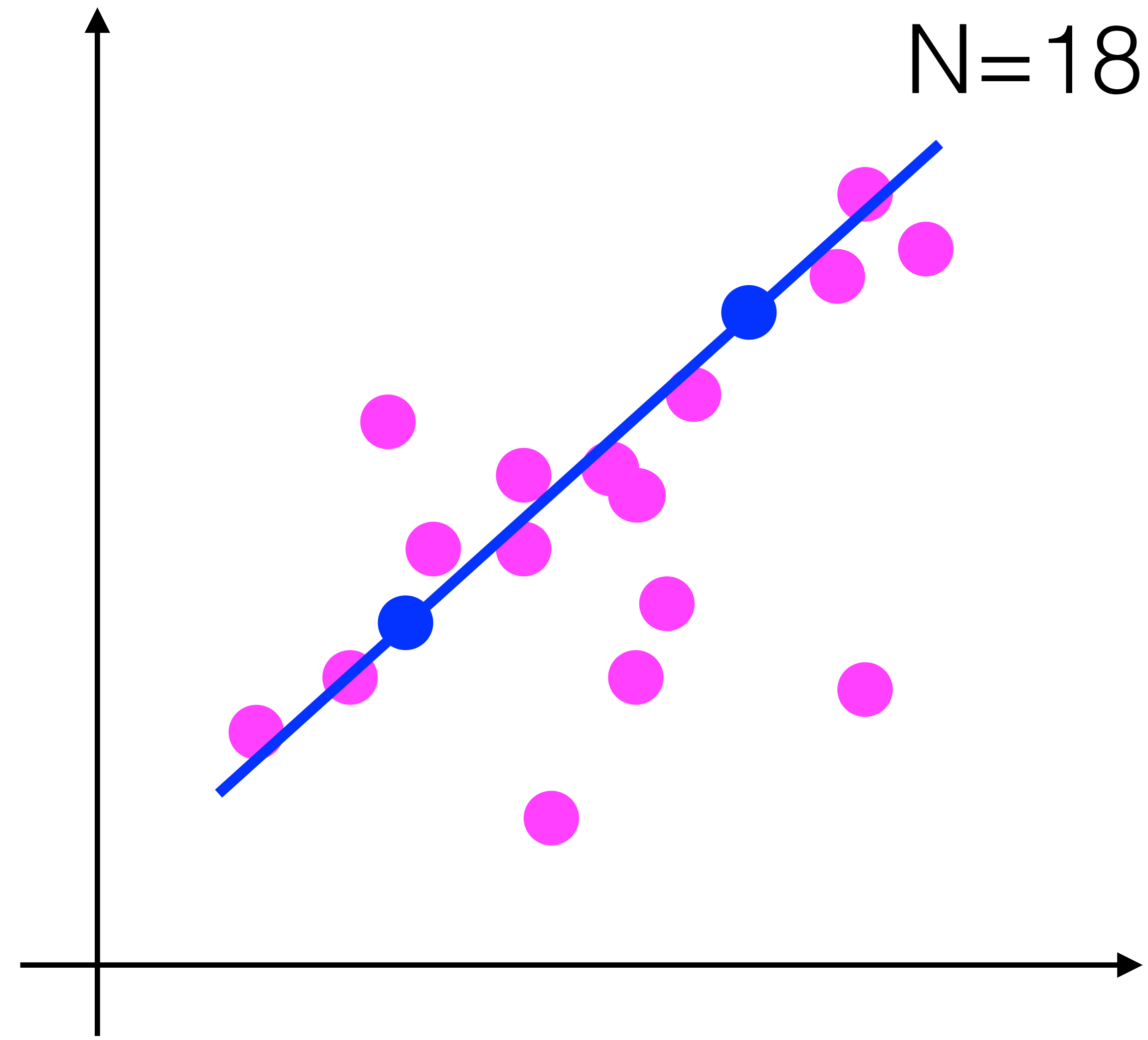
RANSAC:

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



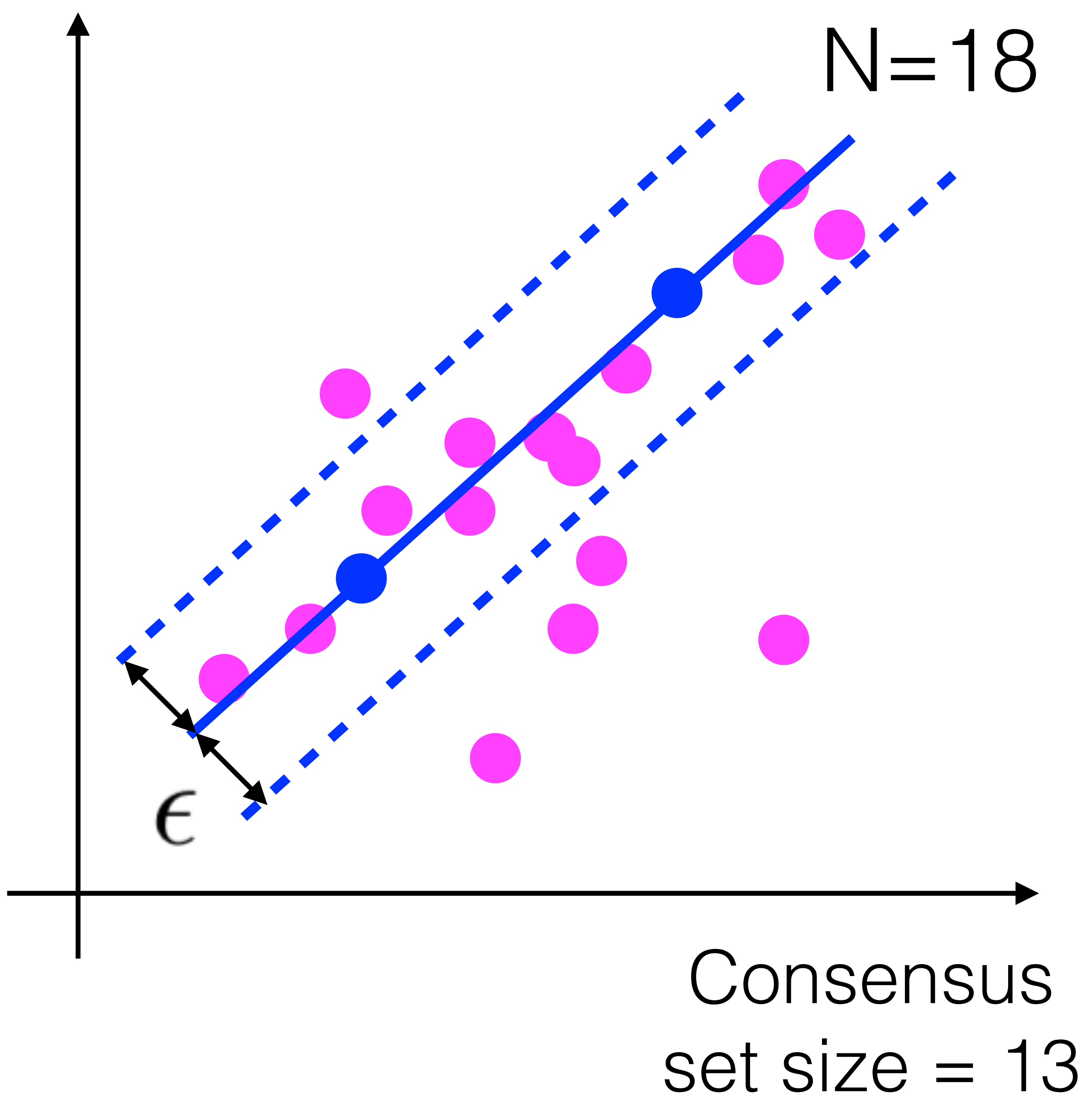
RANSAC:

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

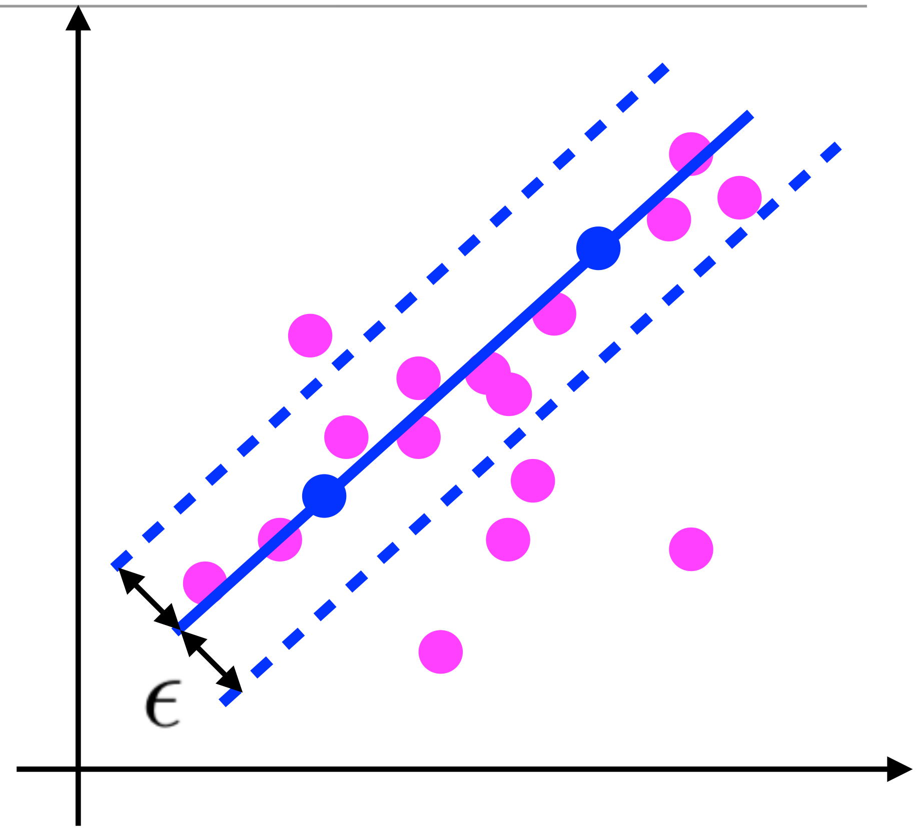


RANSAC:

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

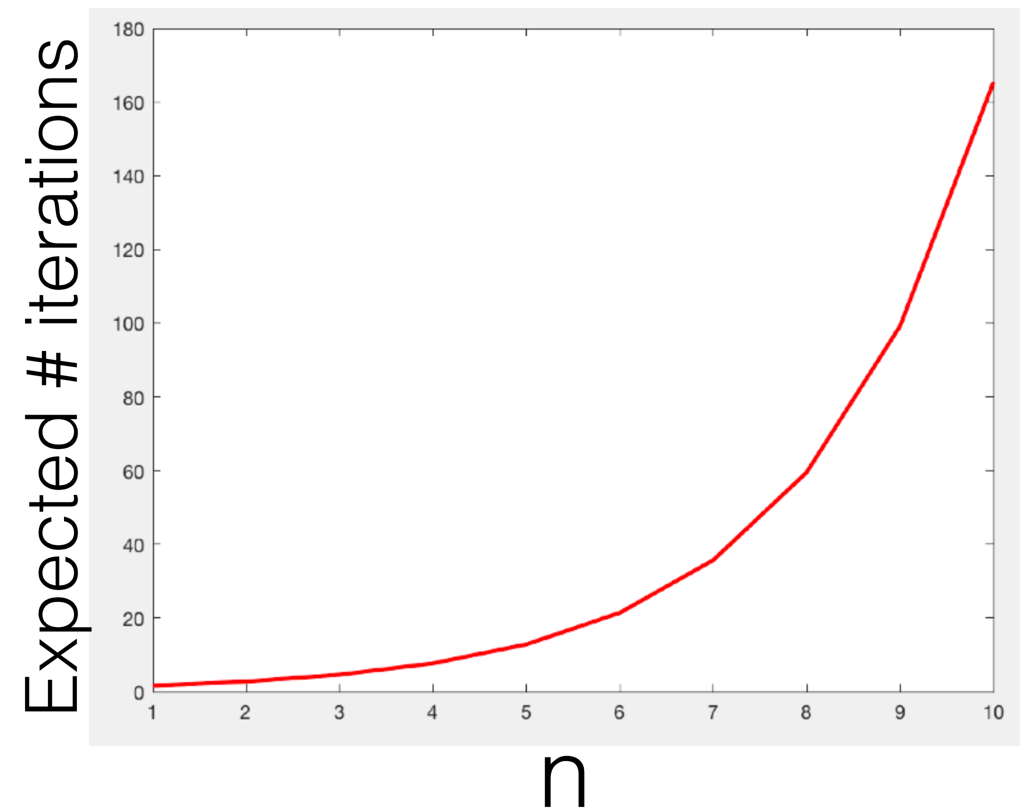
RANSAC: Parameter Tuning

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

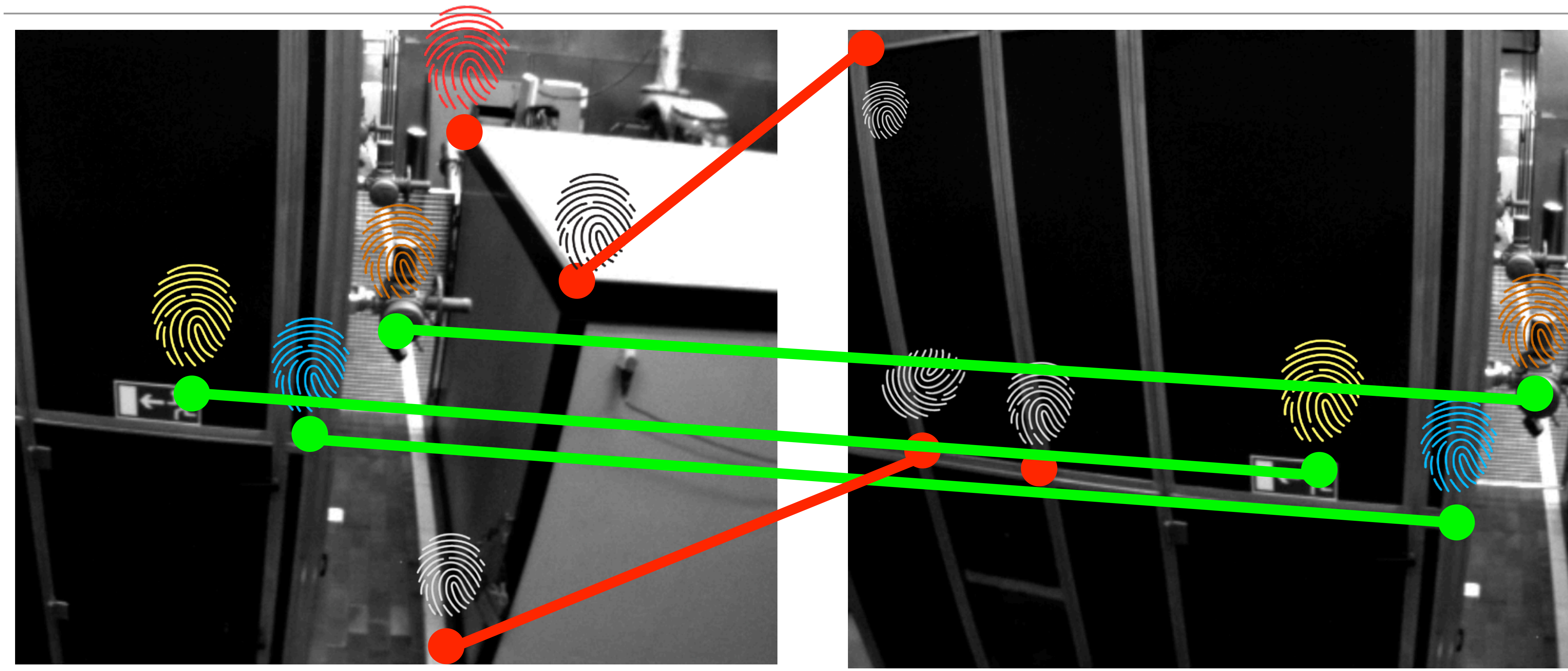


2. **Acceptable consensus set**:
   - from the paper: n+5
   - rule of thumb: >50% of points

3. **Maximum number of iterations**



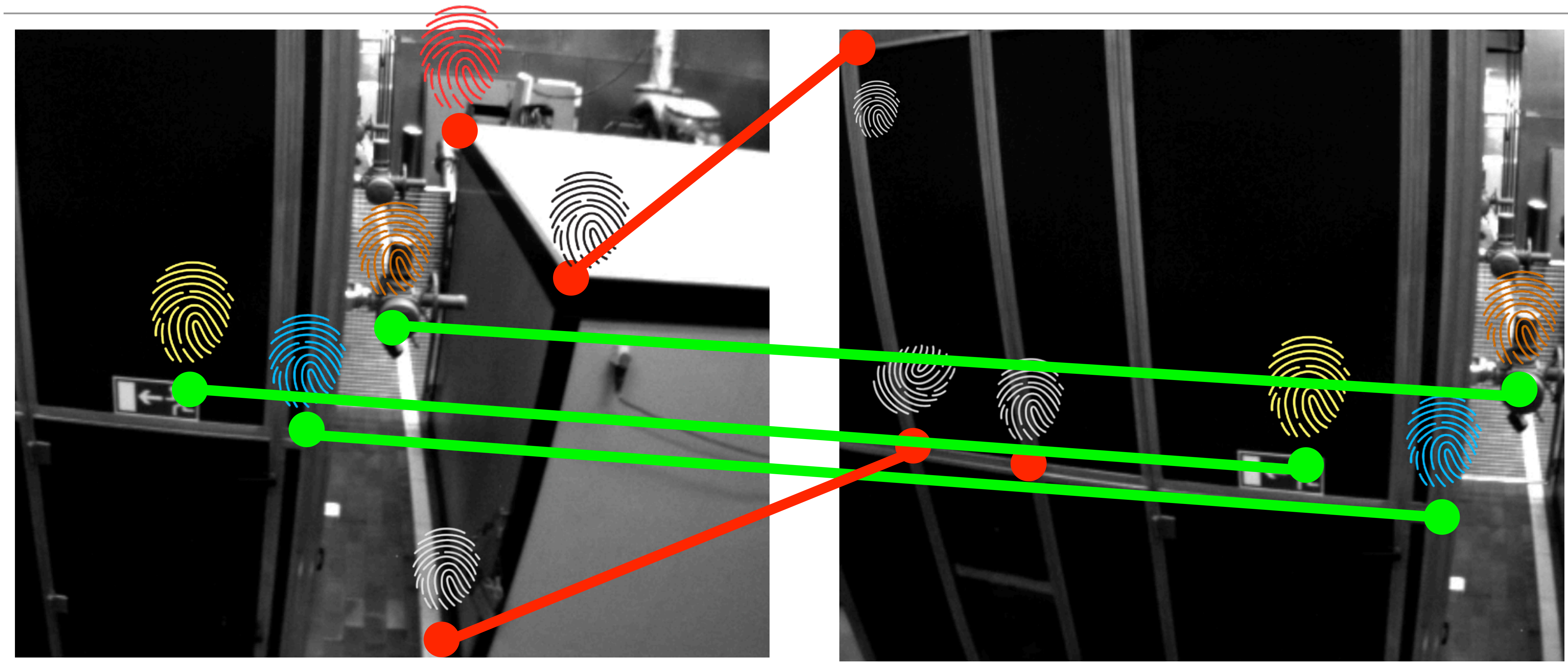
Example: RANSAC for Essential Matrix estimation



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

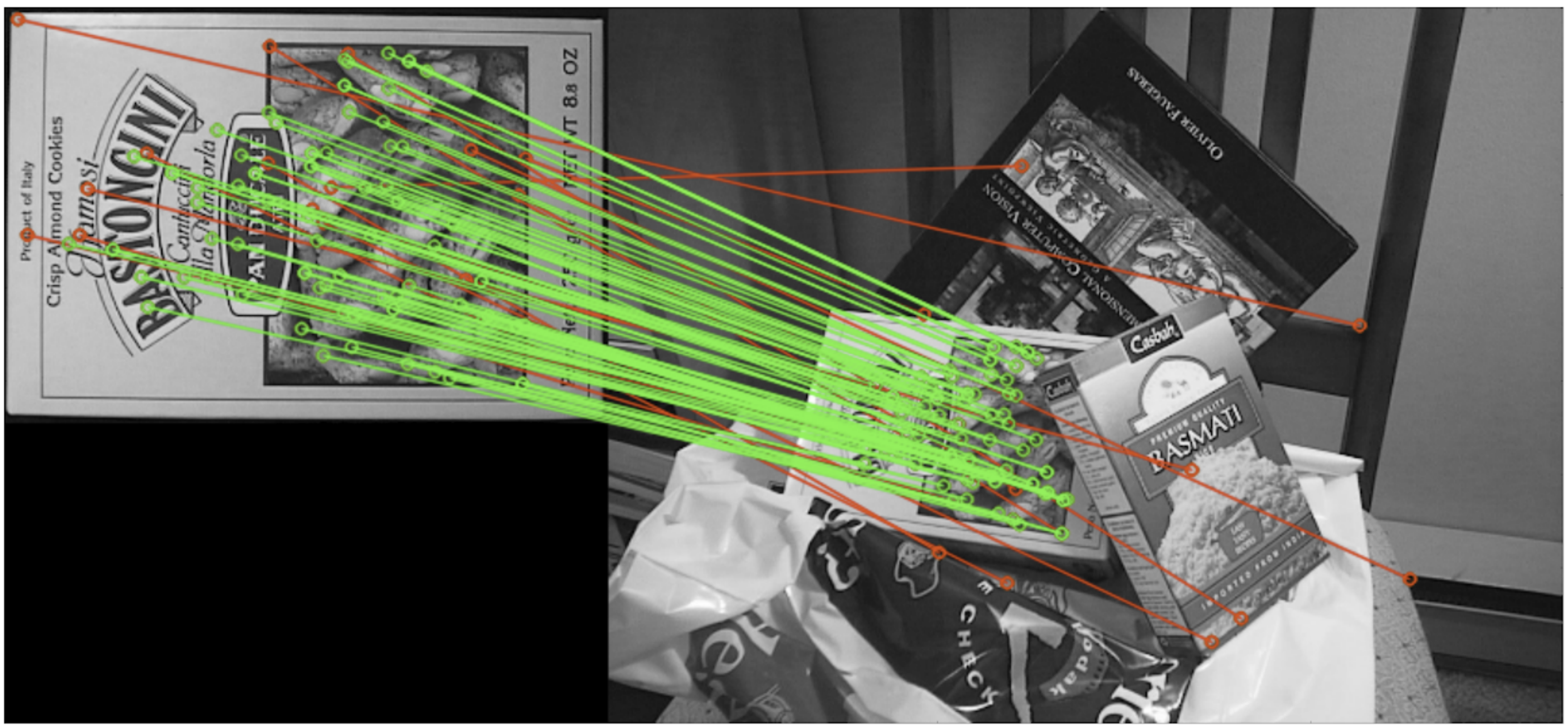
Example: RANSAC for Essential Matrix estimation



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**

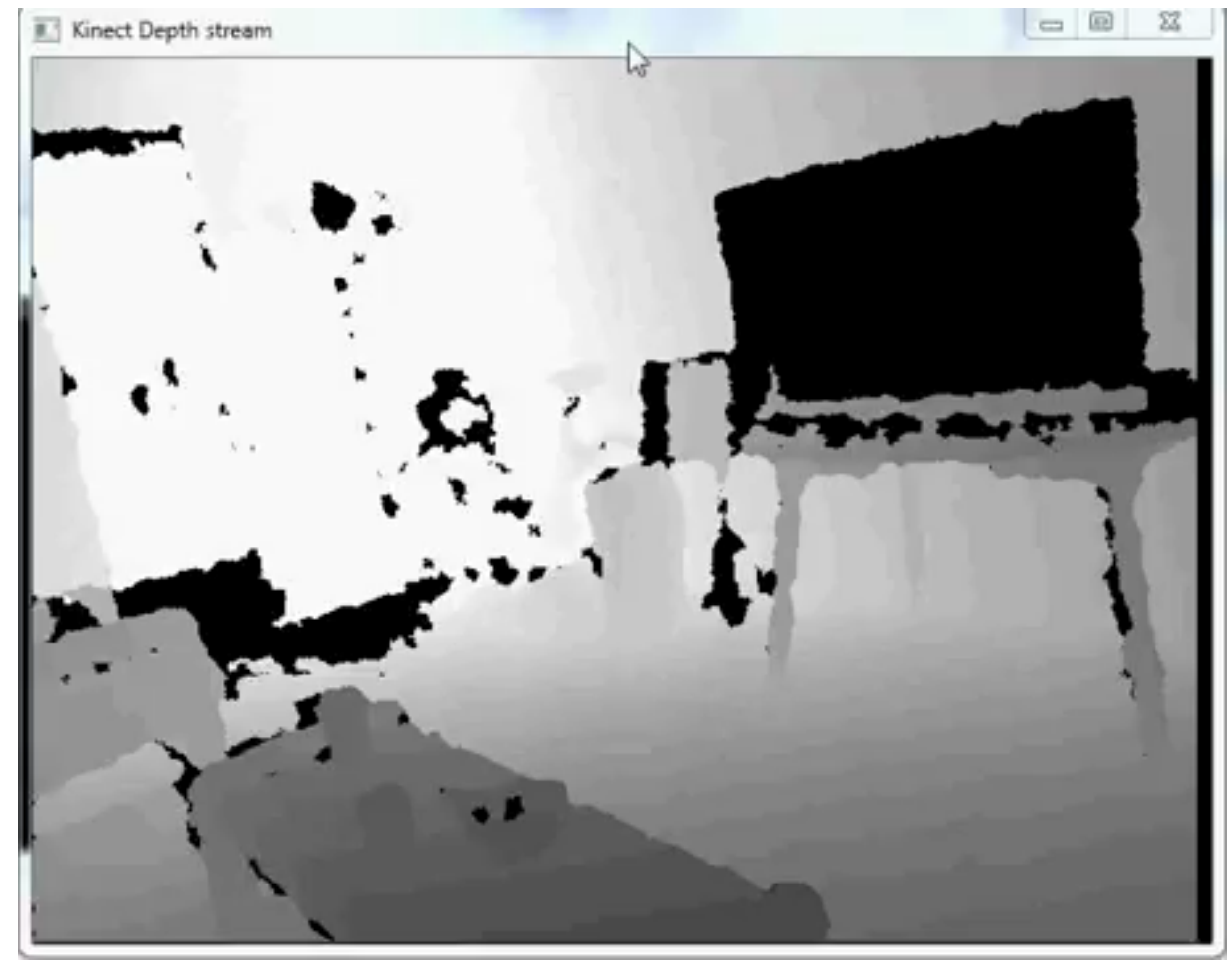


So far: pixel correspondences,
a.k.a., **2D-2D correspondences**

3D-3D Point Correspondences

Structured Light Cameras

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/



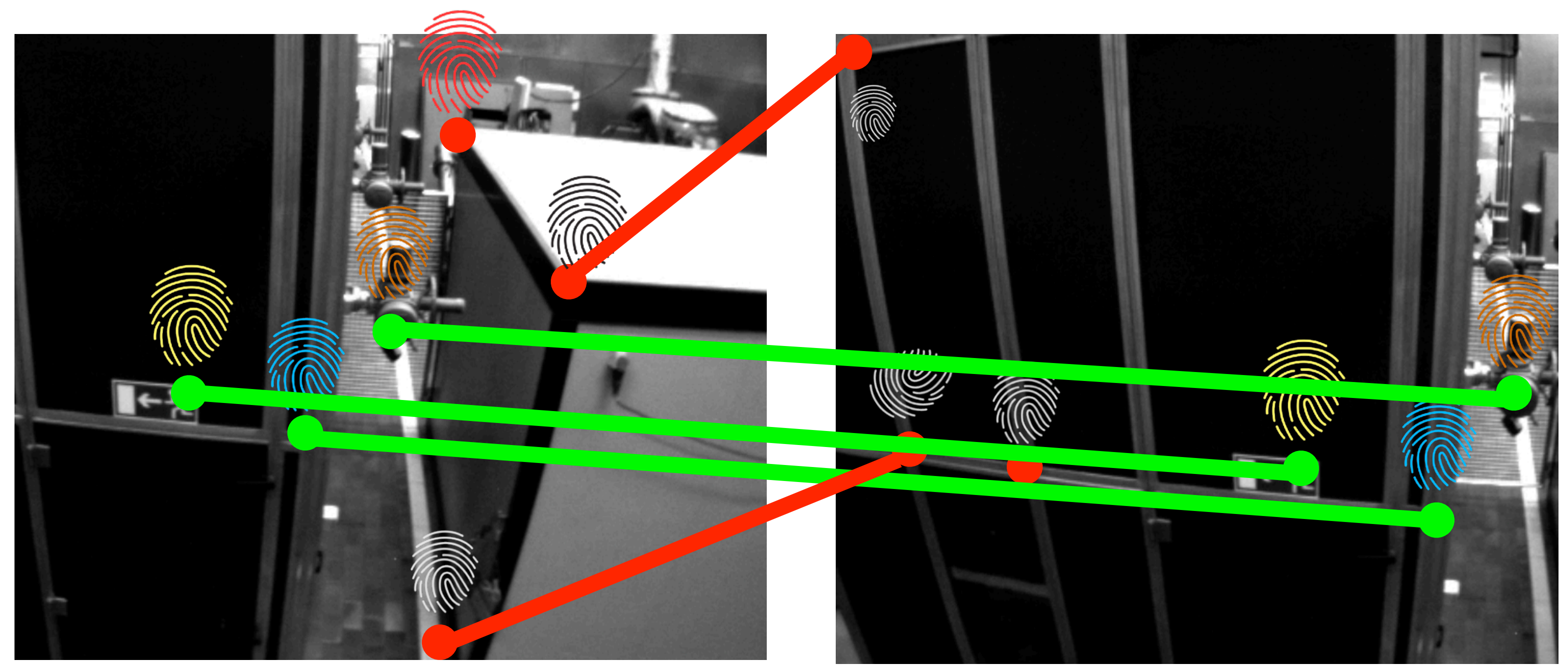
RGB-D cameras can measure depth (D) and image (RBG)

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

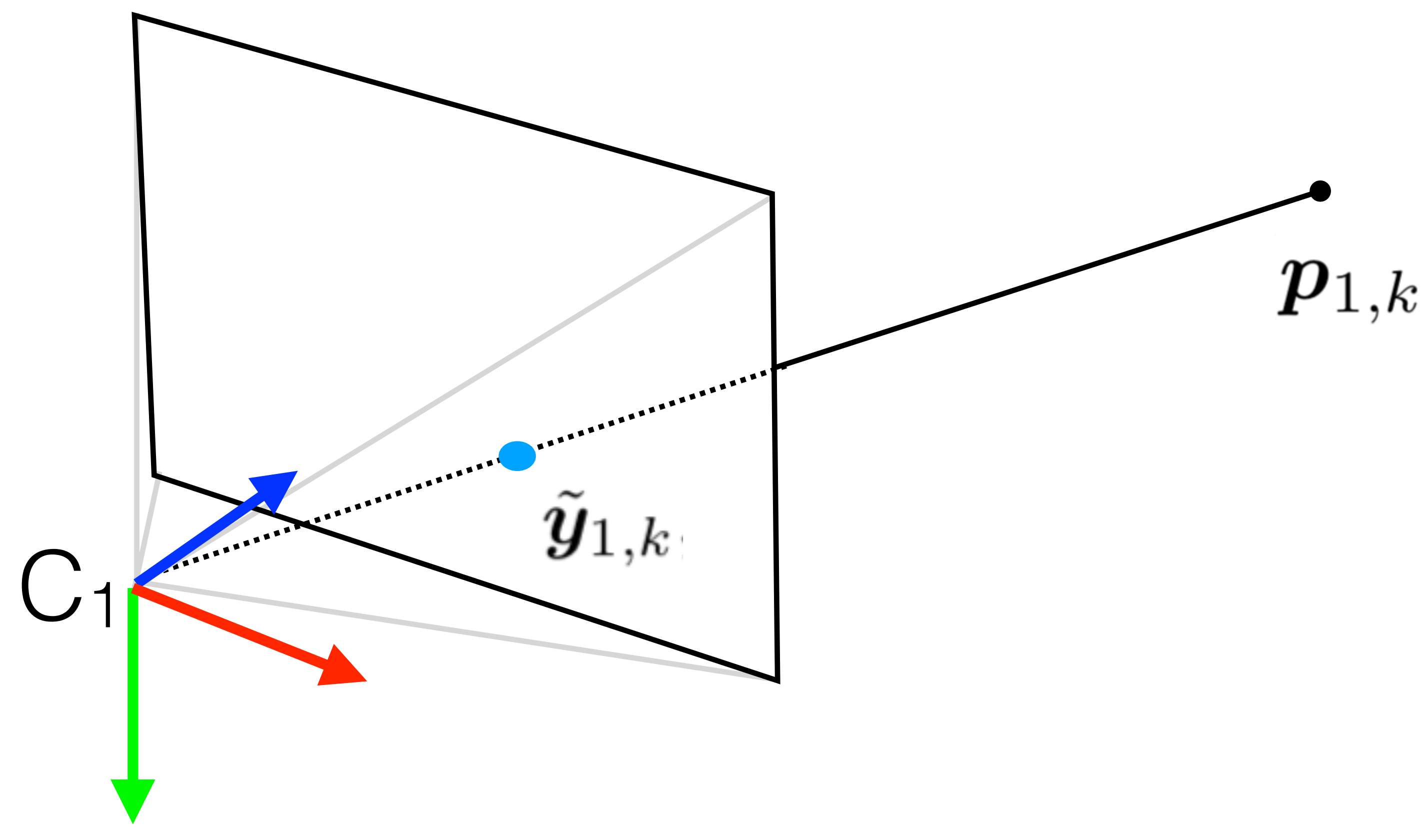
3D-3D Point Correspondences

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

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

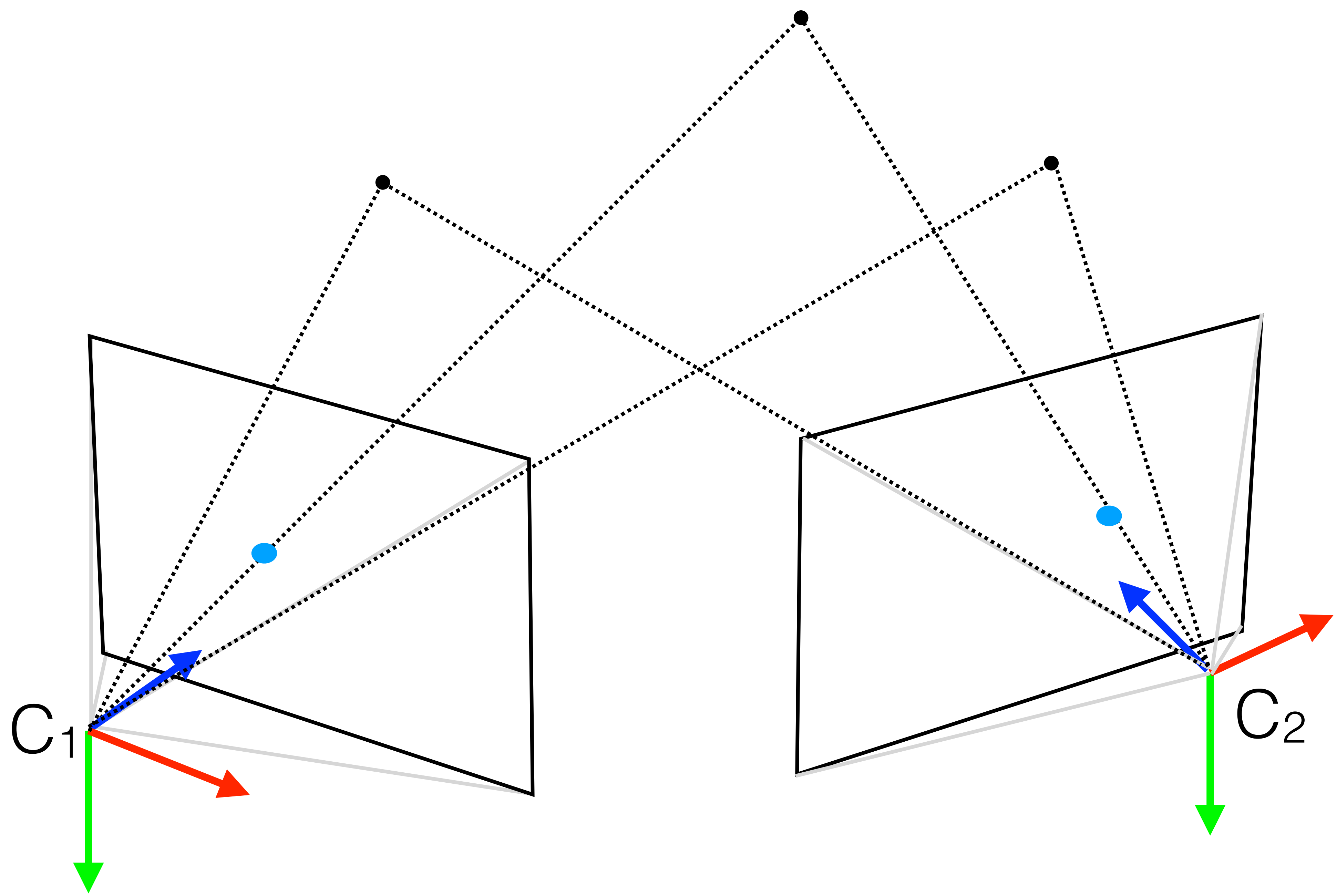


2. For each camera we can compute the set of 3D points corresponding to pixels



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

2-view Geometry from 3D-3D Correspondences



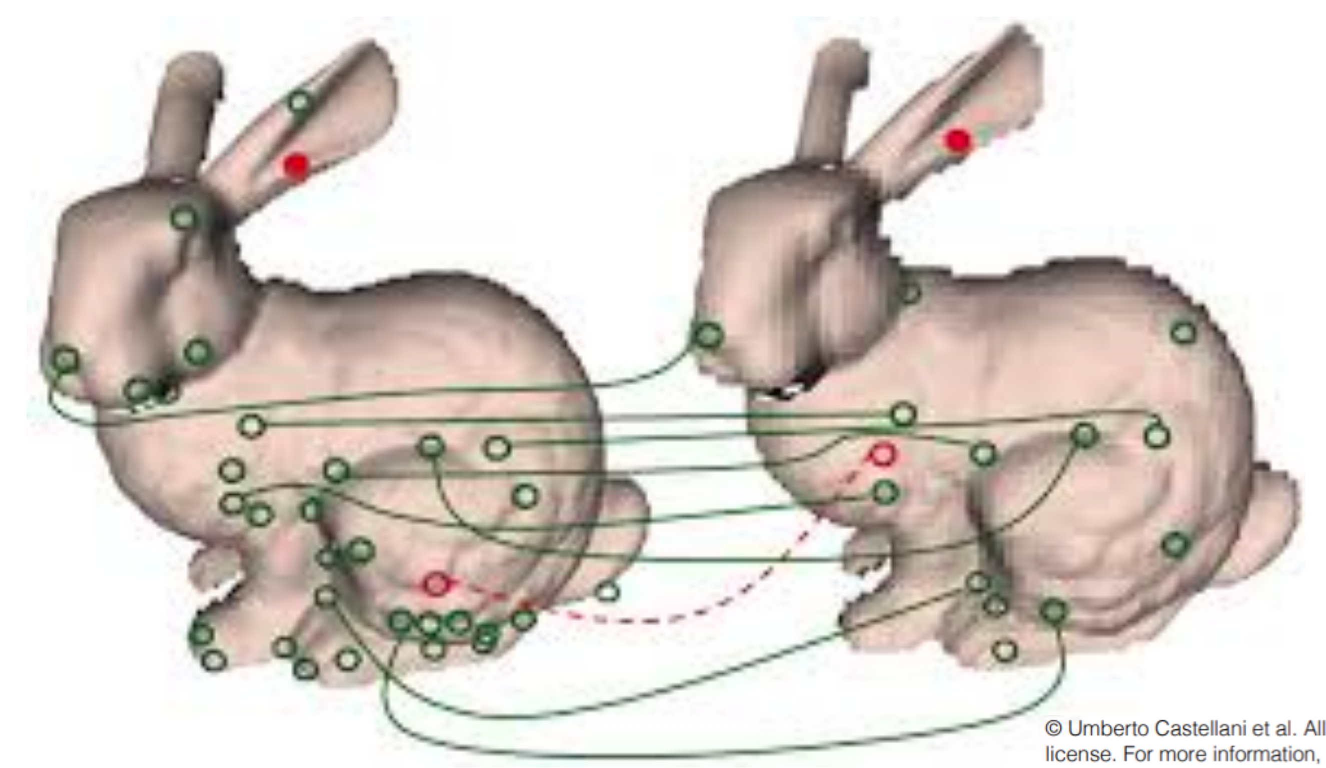
How to estimate the relative pose between the cameras from 3D-3D correspondences $(\boldsymbol{p}_{1,k}, \boldsymbol{p}_{2,k})$ **with** $k = 1, \ldots, N$ **?**

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:



© Umberto Castellani et al. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/

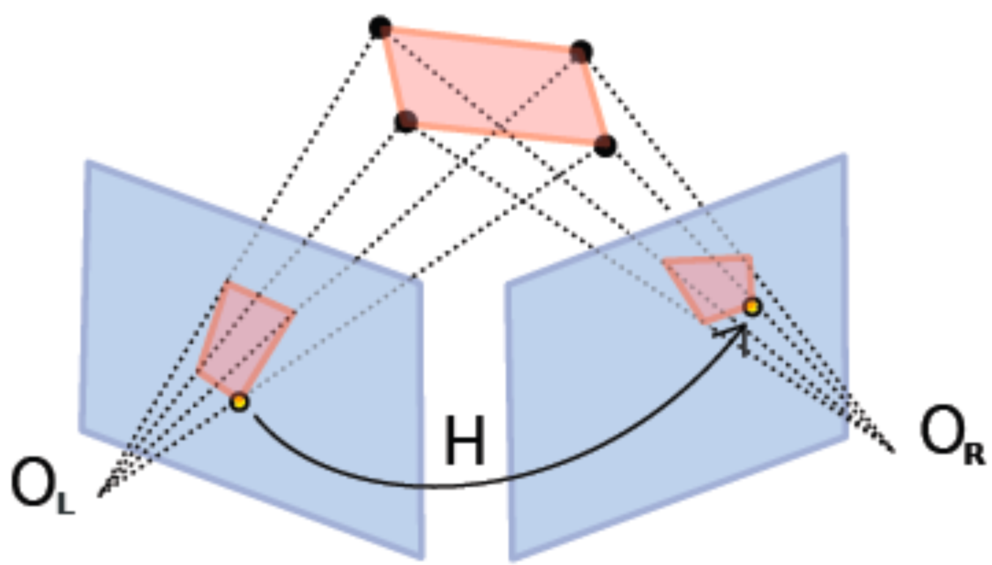
Other names: vector registration, point cloud alignment, ..

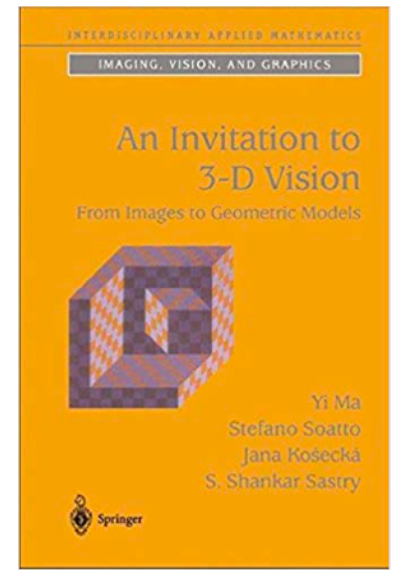
Backup

Other Matrices in 2-view Geometry

Homography matrix **H**

$$\lambda_2 \boldsymbol{x}_2 = H \lambda_1 \boldsymbol{x}_1$$

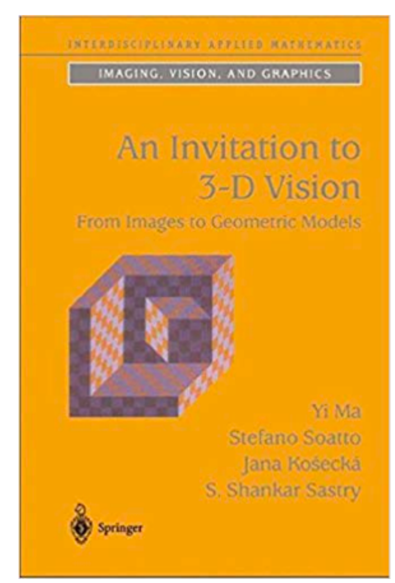




Section 5.3

Fundamental matrix **F**

$$\boldsymbol{F} = \boldsymbol{K}_2^{-\top} \; [\boldsymbol{t}]_\times \boldsymbol{R} \; \boldsymbol{K}_1^{-1}$$



Chapter 6

Essential Matrix Properties

- A matrix is an essential matrix *if and only* if it has singular values $\{\sigma, \sigma, 0\}$

- The space of the essential matrices is called the *Essential space* $\mathcal{S}_E$ (i.e., the space of 3×3 matrices that can be written as $[\boldsymbol{t}]_\times \boldsymbol{R}$ for some $\boldsymbol{R} \in \mathrm{SO}(3)$ and $\boldsymbol{t} \in \mathbb{R}^3$). The projection of a matrix $\boldsymbol{M}$ onto the Essential space can be computed as prescribed in [1, Thm 5.9]:

$$\underset{\boldsymbol{E} \in \mathcal{S}_E}{\arg\min} \|\boldsymbol{E} - \boldsymbol{M}\|_F^2 = \boldsymbol{U} \begin{bmatrix} \frac{\lambda_1+\lambda_2}{2} & 0 & 0 \\ 0 & \frac{\lambda_1+\lambda_2}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{V}^\top$$

where $\boldsymbol{M} = \boldsymbol{U} \operatorname{diag}(\lambda_1, \lambda_2, \lambda_3) \boldsymbol{V}^\top$ is a singular value decomposition of $\boldsymbol{M}$.

[1]