University of Zurich[UZH]

ROBOTICS & PERCEPTION GROUP

Vision Algorithms for Mobile Robotics

Lecture 09
Multiple View Geometry 3

Davide Scaramuzza

http://rpg.ifi.uzh.ch

Lab Exercise 7 – Today

Implement the P3P algorithm and RANSAC.

Additionally, we will outline the mini projects

Outline

- Robust Structure from Motion
- Bundle Adjustment

Robust Estimation

- Matched points are usually contaminated by **outliers** (i.e., wrong image matches).

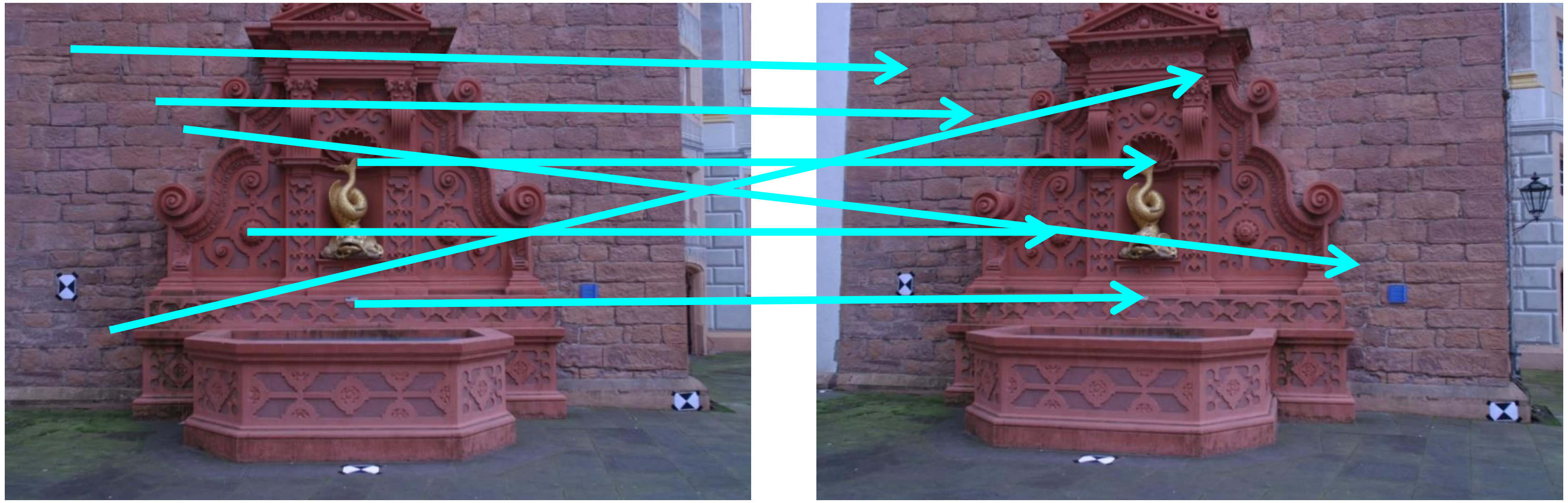


Image 1 Image 2

Robust Estimation

- Matched points are usually contaminated by **outliers** (i.e., wrong image matches).
- Causes of outliers are:
  - Repetitive features (i.e., features with the same appearance)
  - Geometric and photometric changes to which the descriptor is not invariant
  - Large image noise
  - Occlusions
  - Moving objects
  - Image or motion blur
- For reliable and accurate visual odometry, outliers must be removed
- This is the task of **Robust Estimation**

Image 1

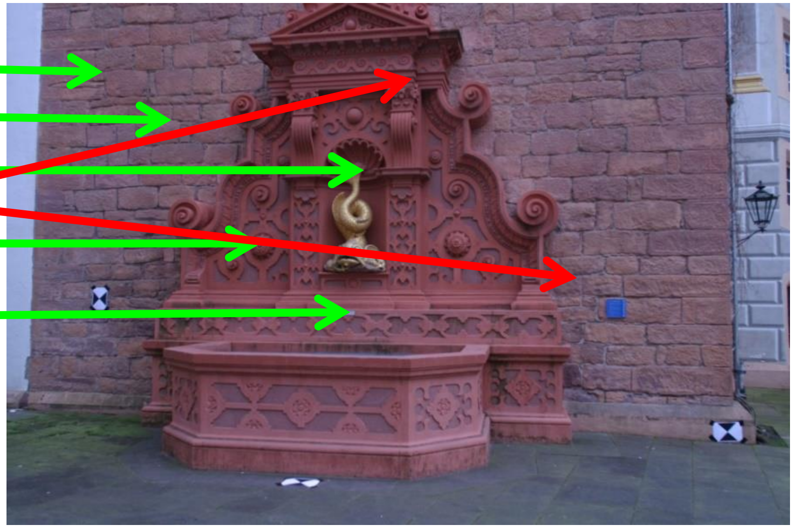
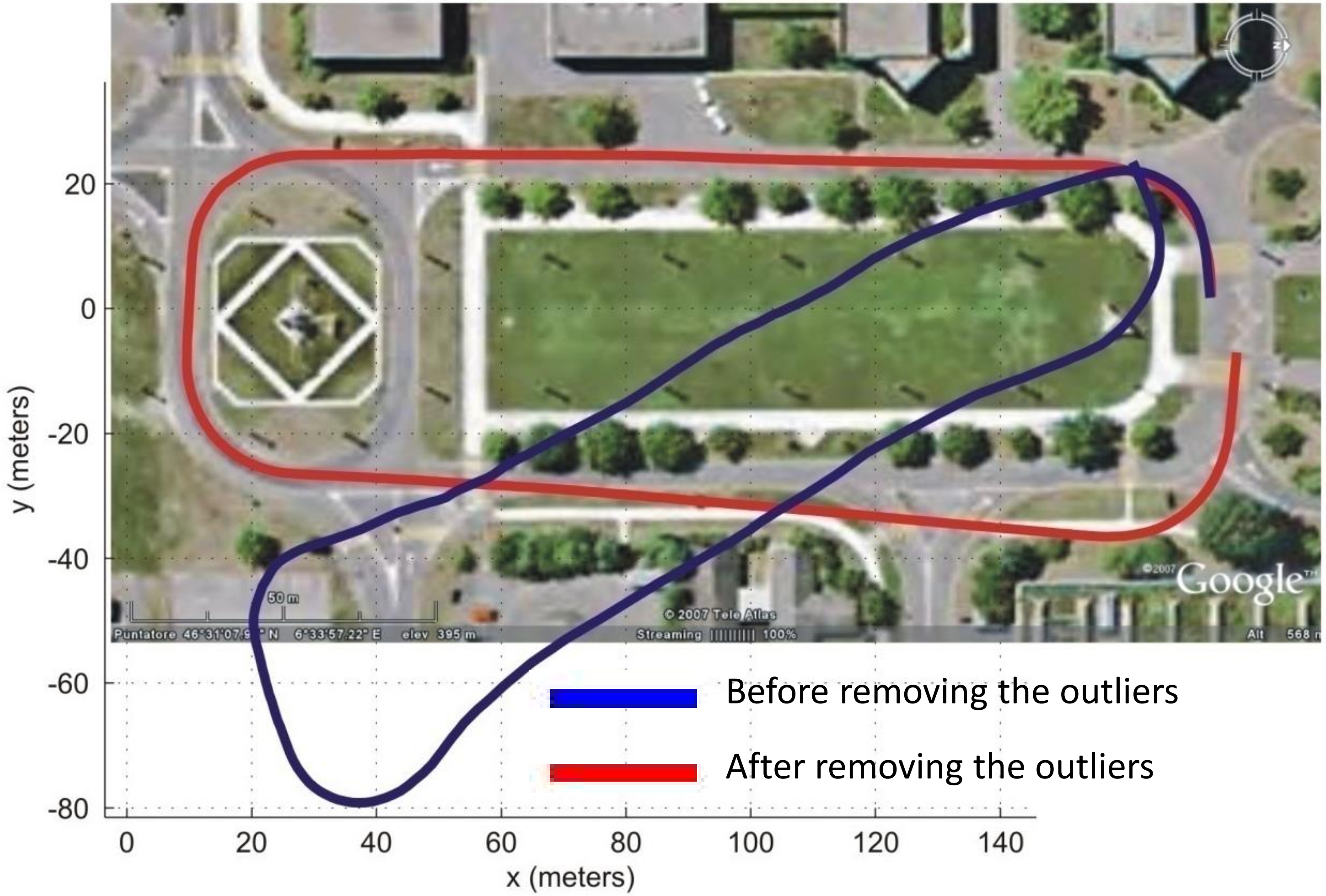


Image 2

Effect of Outliers on Visual Odometry



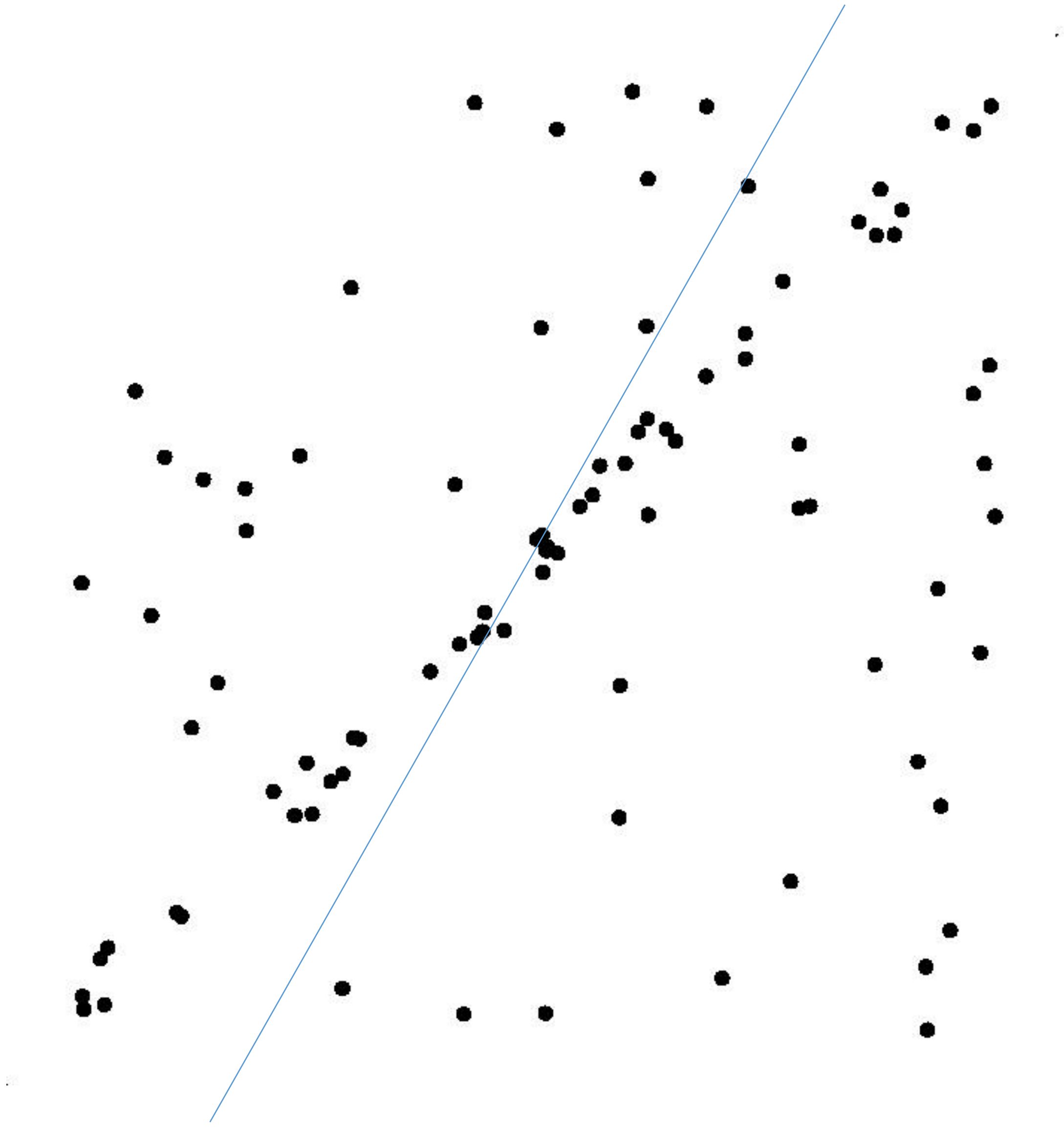
Expectation Maximization (EM) algorithm

- EM is a simple **method for model fitting in the presence of outliers** (very noisy points or wrong data)
- It can be applied to all sorts of problems where the goal is to **estimate the parameters of a model from the data** (e.g., camera calibration, Structure from Motion, DLT, PnP, P3P, Homography, etc.)
- Let's review EM applied to the line fitting problem

[1] Dellaert, The *expectation maximization algorithm,* Georgia Institute of Technology, 2002. PDF (explains the original papers below)
[2] Hartley, *Maximum likelihood estimation from incomplete data*, Biometrics, 1958.
[3] Dempster, Laird, Rubin, *Maximum likelihood from incomplete data via the EM algorithm*, Journal of the Royal Statistical Society, 1977.

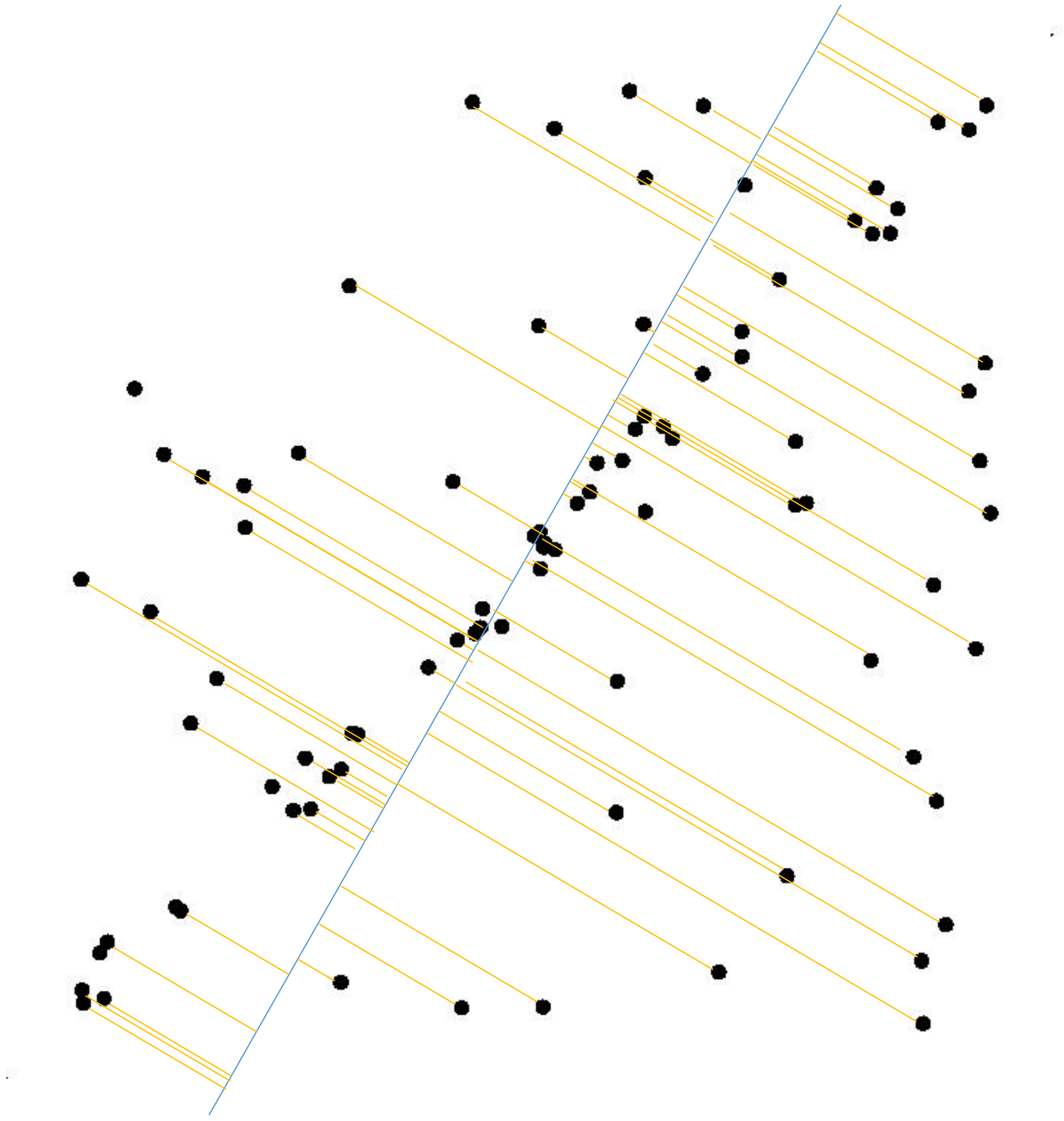
EM applied to line fitting

EM applied to line fitting



1. Estimate line parameters that fit all data points (e.g., using least-square: $min \sum r_i^2$, where r_i is the point-to-line distance)

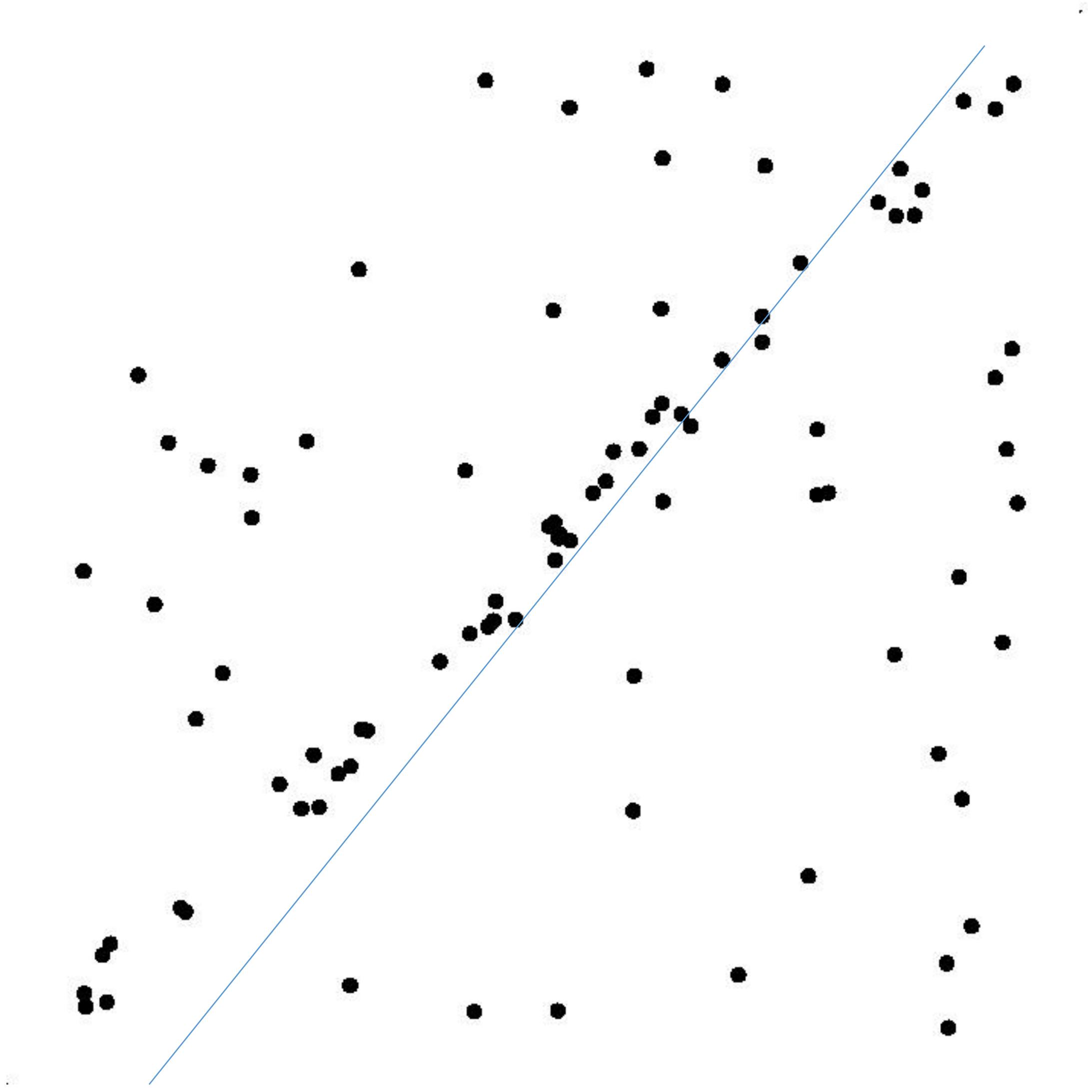
EM applied to line fitting



1. Estimate line parameters that fit all data points (e.g., using least-square: $min \sum r_i^2$, where r_i is the point-to-line distance)

2. Calculate residual error r_i for each data point and assign it a weight (e.g., $w_i = e^{-r_i^2}$ representing the likelihood that such assignment is correct (estimates the **Expectation**)

EM applied to line fitting

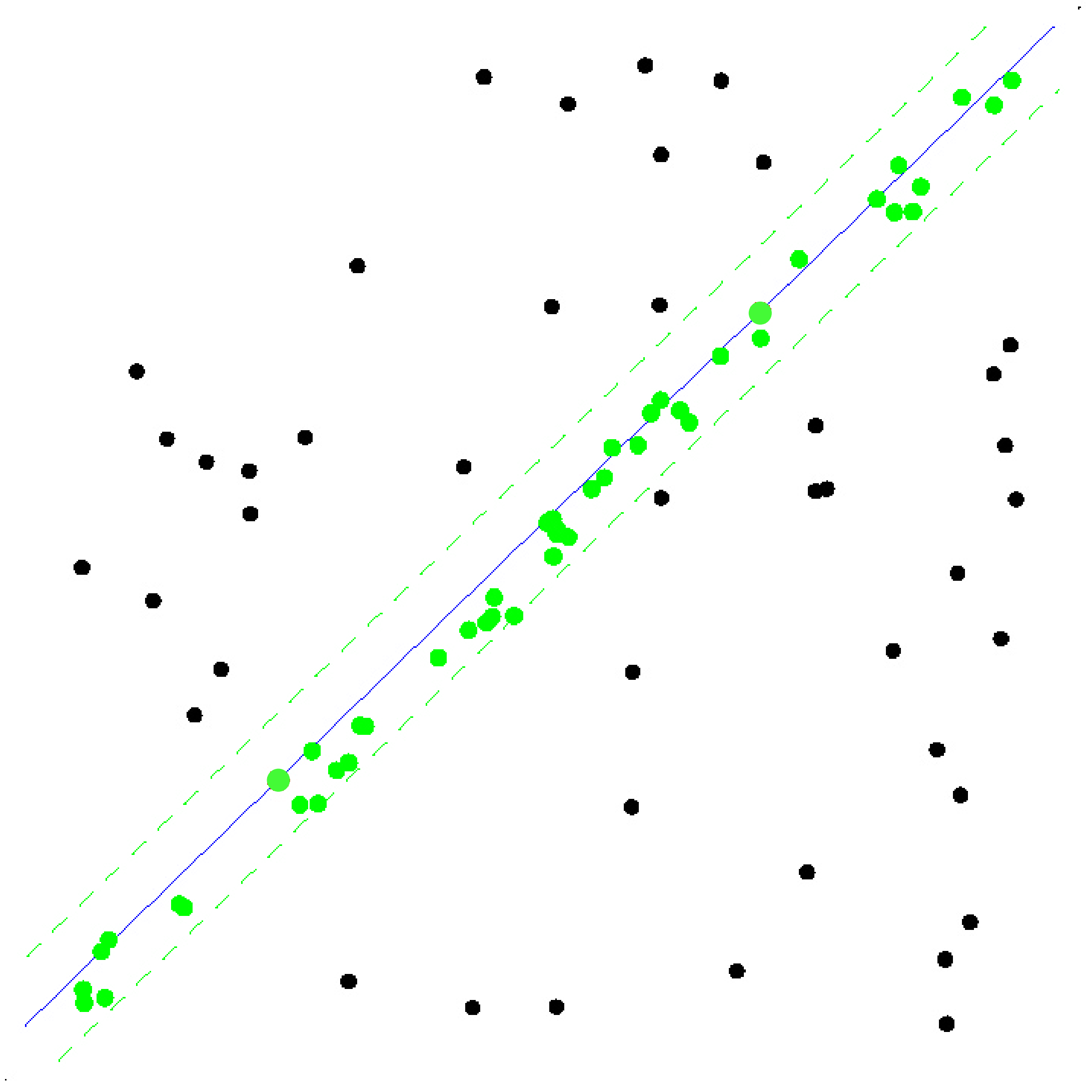


1. Estimate line parameters that fit all data points (e.g., using least-square: $\min \sum r_i^2$, where r_i is the point-to-line distance)

2. Calculate residual error r_i for each data point and assign it a weight (e.g., $w_i = e^{-r_i^2}$ representing the likelihood that such assignment is correct (estimates the **Expectation**)

3. Re-estimate line parameters (e.g., using weighted least-squares: $\min \sum w_i r_i^2$) **(Maximization Step)**

EM applied to line fitting



1. Estimate line parameters that fit all data points (e.g., using least-square: $\min \sum r_i^2$, where r_i is the point-to-line distance)

2. Calculate residual error r_i for each data point and assign it a weight (e.g., $w_i = e^{-r_i^2}$ representing the likelihood that such assignment is correct (estimates the **Expectation**)

3. Re-estimate line parameters (e.g., using weighted least-squares: $\min \sum w_i r_i^2$) **(Maximization Step)**

4. Iterate 2 and 3 till convergence

5. Select as **inliers the** data points with weight higher than a threshold

Problem of EM algorithm

Very sensitive to initial condition:

- This is because EM selects the initial condition by minimizing the sum of squared residuals $\sum r_i^2$.
- While this is a convex function, the result is strongly influenced by a few large error values (e.g., outliers).
- Thus, EM converges to the wrong solution if initial condition is far from the true one

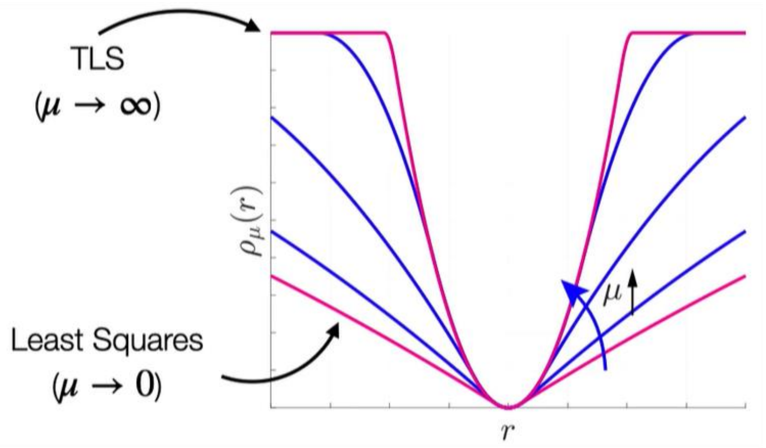
- Alternative options:
  - GNC algorithm
  - RANSAC algorithm

Graduated Non-Convexity algorithm (GNC)

Won't be asked at the exam ☺

Idea: optimize a surrogate function $\sum \rho_\mu(r_i)$, where μ controls the amount of non-convexity.

- Start by solving the non-robust convex optimization function (μ → 0, i.e., least squares)
- At each iteration, gradually increase non-convexity (μ → ∞) and recompute weights w_i till we achieve the desired level of robustness.
- It is shown in [1] to be robust up to 90% of outliers with five times fewer iterations than RANSAC.
- However, RANSAC can cope with even more than 90% outliers.



[1] Yang, Antonante, Tzoumas, Carlone, *Graduated Non-Convexity for Robust Spatial Perception: From Non-Minimal Solvers to Global Outlier Rejection*, International Conference on Robotics and Automation **(ICRA), 2020. Best paper award in Robot Vision**. PDF. Code.
[2] Blake, Zisserman, *Visual Reconstruction*. MIT Press, Cambridge, Massachusetts, 1987.

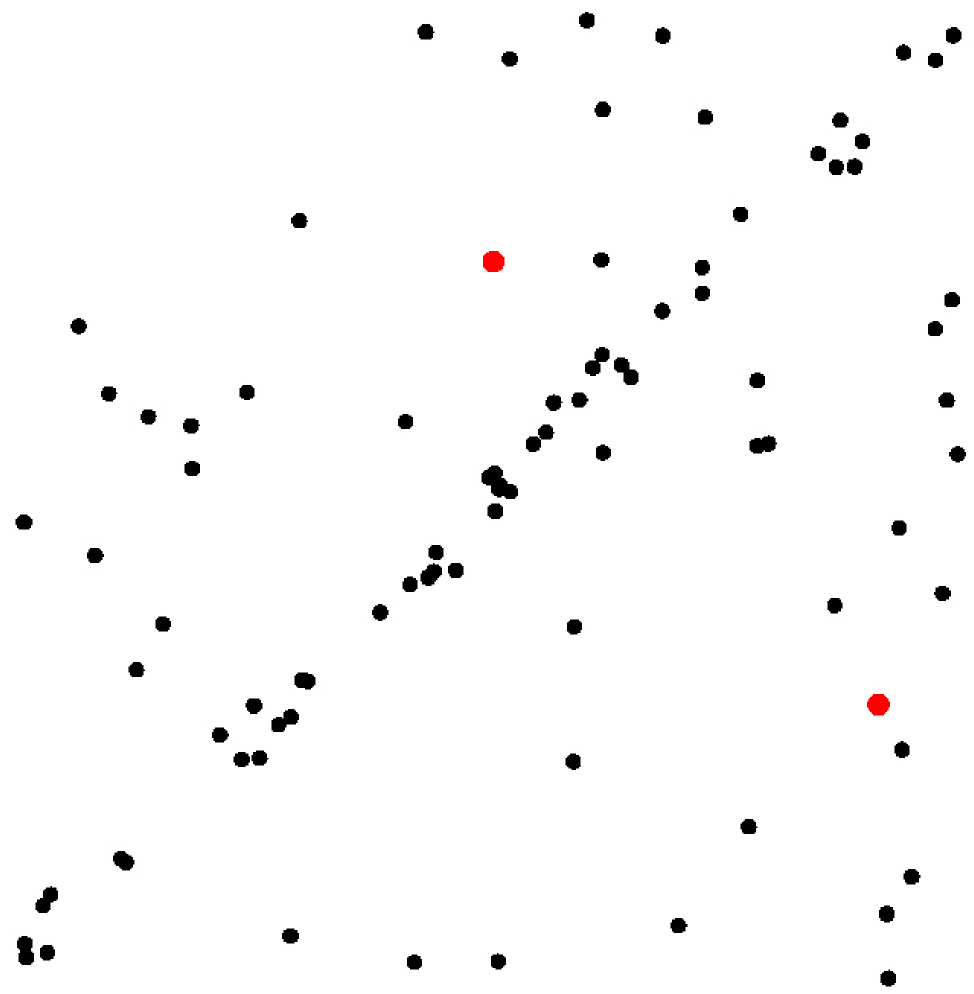
RANSAC (RAndom SAmple Consensus)

- RANSAC is the **standard method for model fitting in the presence of outliers** (very noisy points or wrong data)
- It is **non-deterministic**: you get a different result everytime you run it
- It is **not sensitive to the initial condition**, and **does not get stuck in local maxima**
- It can be applied to all sorts of problems where the goal is to **estimate the parameters of a model from the data** (e.g., camera calibration, Structure from Motion, DLT, PnP, P3P, Homography, etc.)
- Let's review RANSAC for line fitting and see how we can use it to do Structure from Motion

M. A.Fischler and R. C.Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Graphics and Image Processing, 1981. PDF.

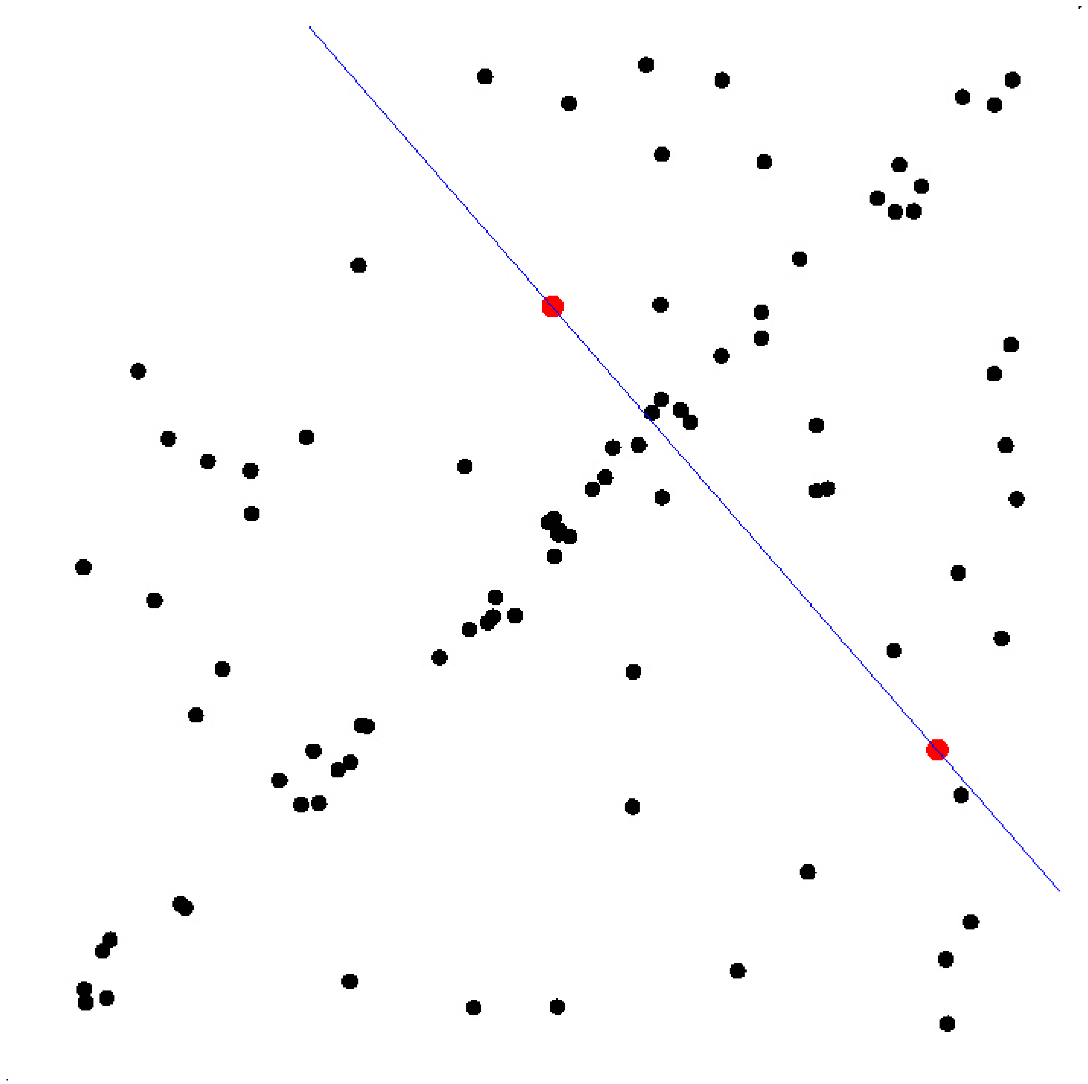
RANSAC

RANSAC



1. **Select a sample of 2 points at *random***

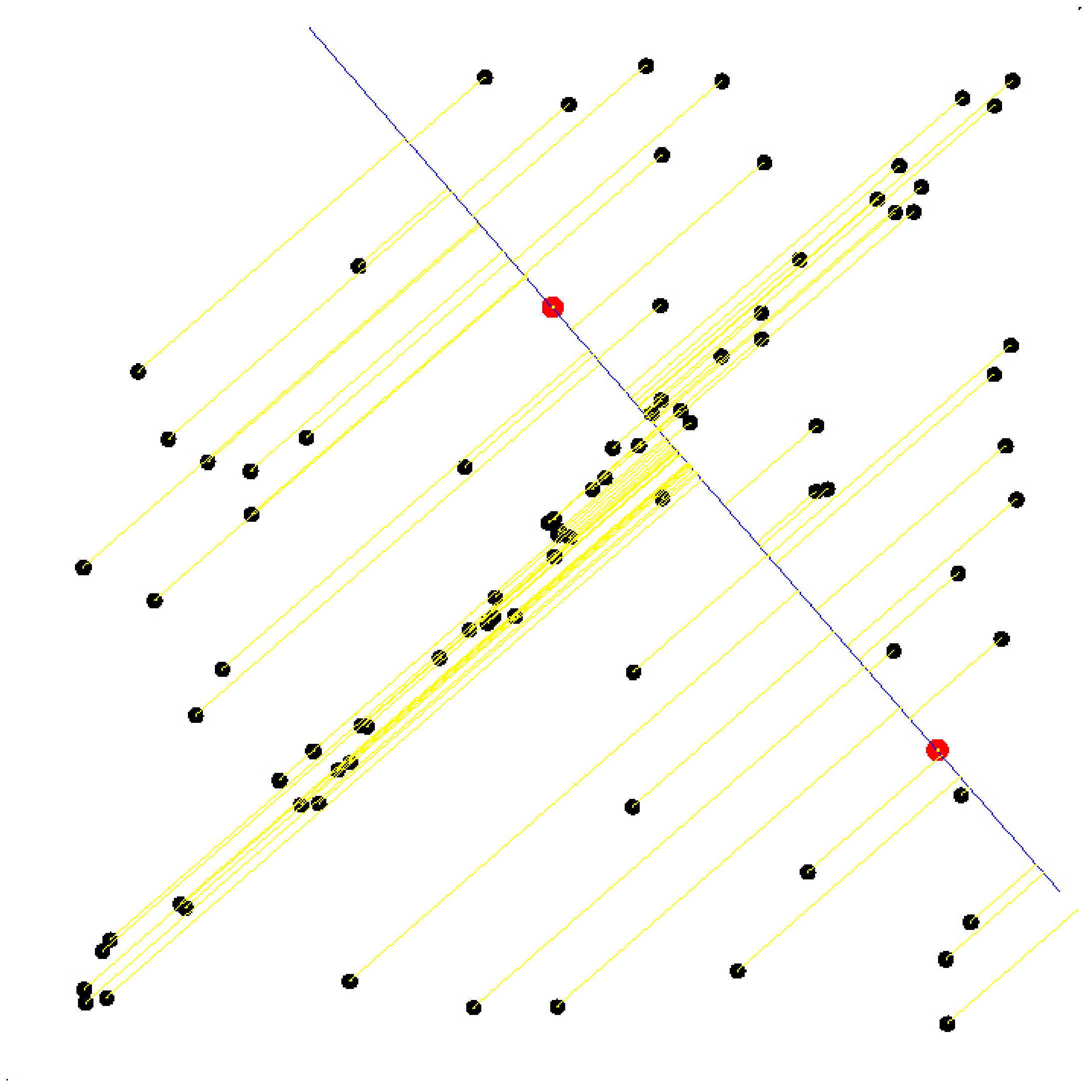
RANSAC



1. Select a sample of 2 points at *random*

2. **Calculate model parameters that fit the data in the sample**

RANSAC

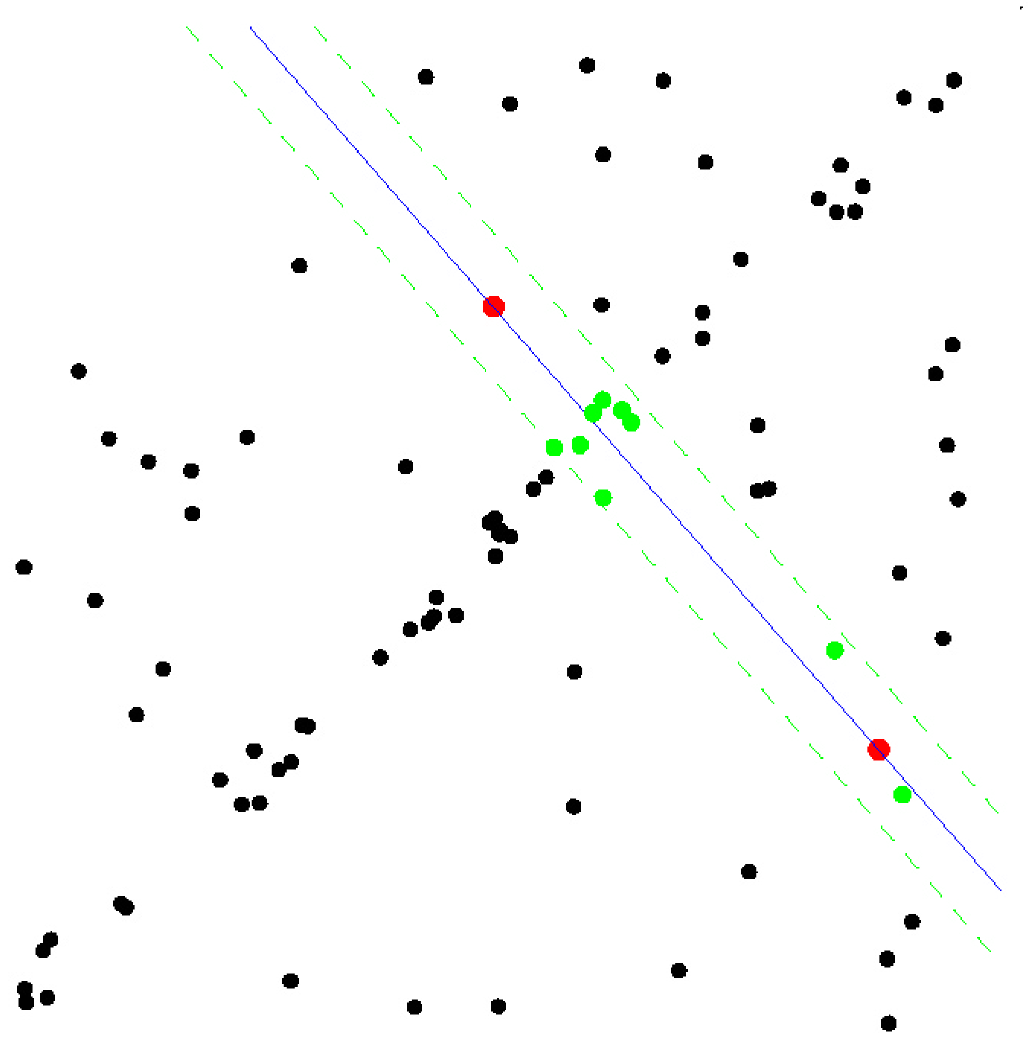


1. Select a sample of 2 points at *random*

2. Calculate model parameters that fit the data in the sample

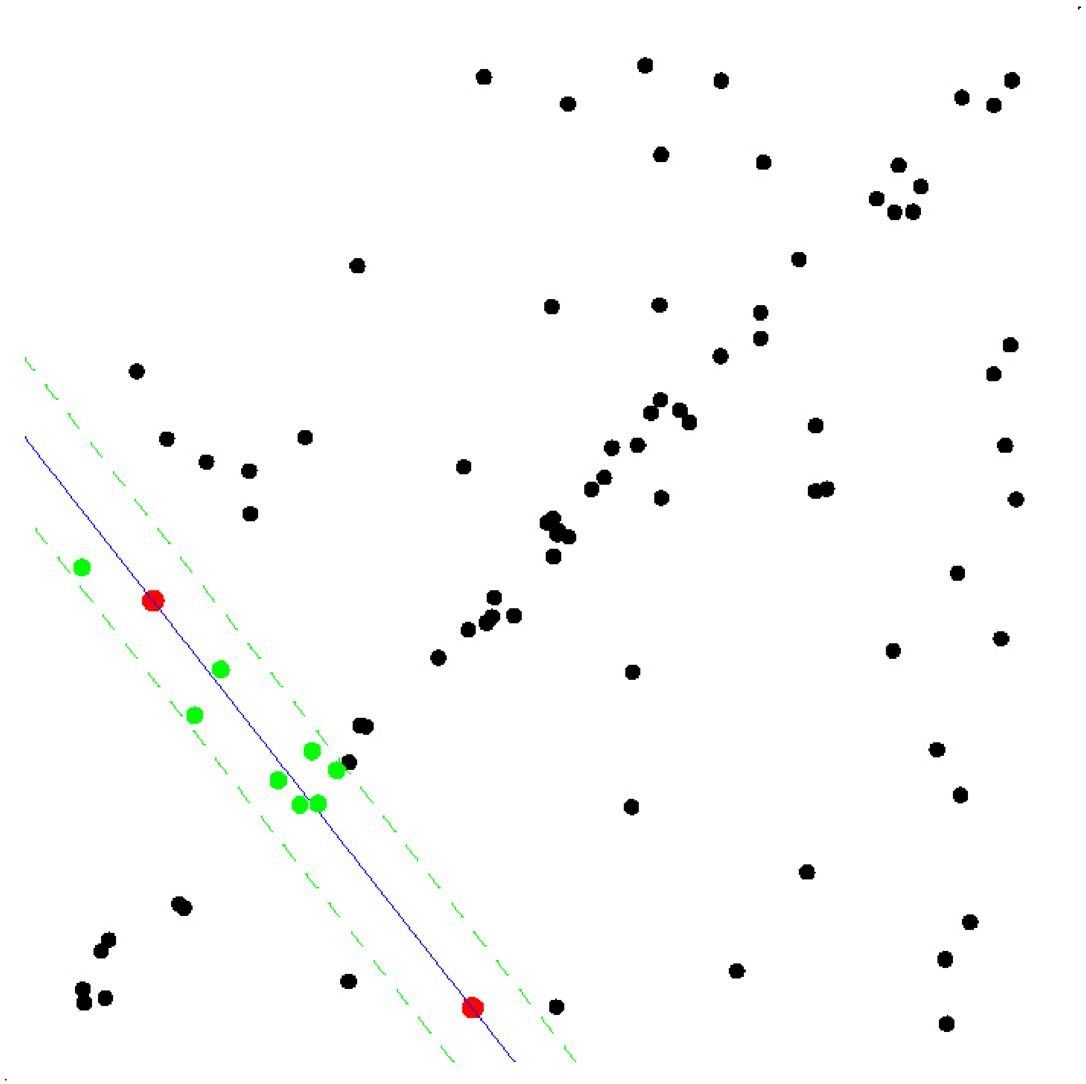
3. **Calculate the residual error for each data point**

RANSAC



1. Select a sample of 2 points at *random*
2. Calculate model parameters that fit the data in the sample
3. Calculate the residual error for each data point
4. **Select data that support current hypothesis**

RANSAC



1. Select a sample of 2 points at *random*

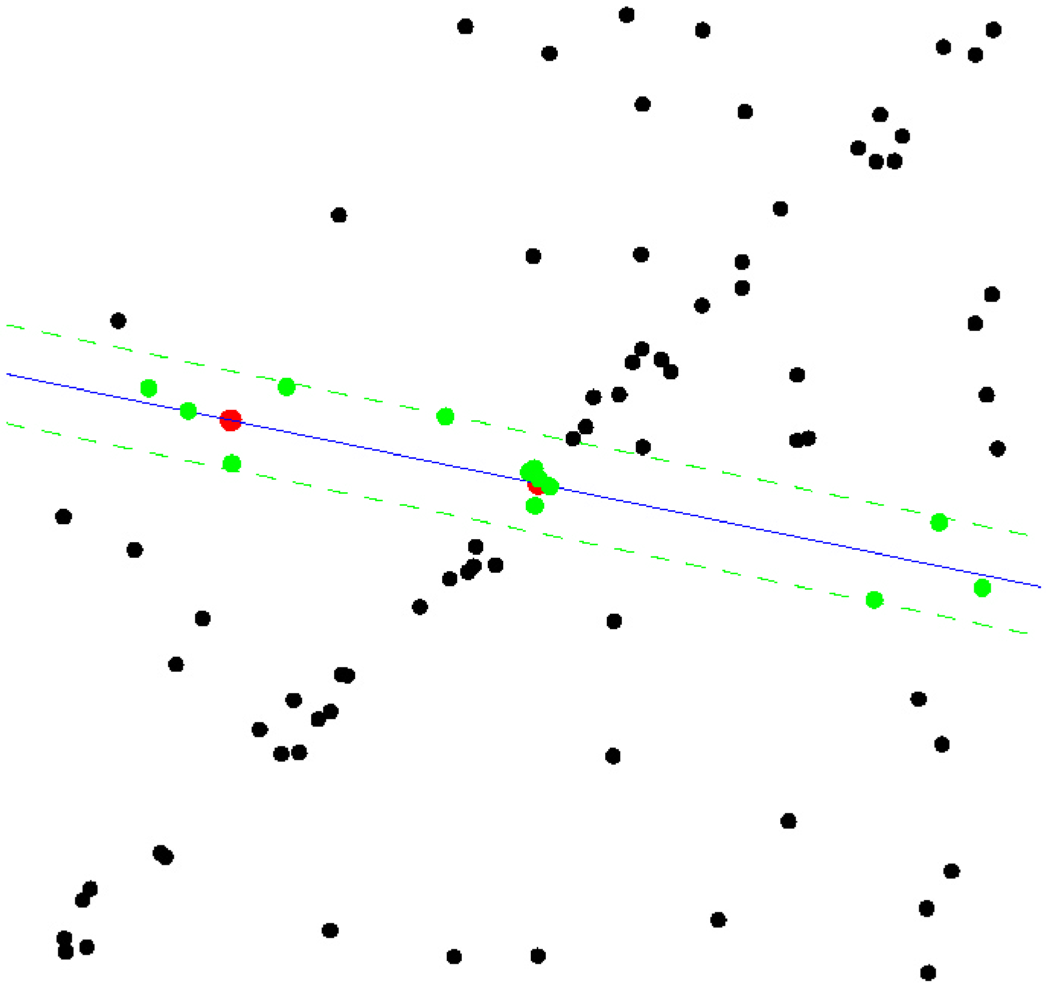
2. Calculate model parameters that fit the data in the sample

3. Calculate the residual error for each data point

4. Select data that support current hypothesis

5. **Repeat from step 1 for k times**

RANSAC



1. Select a sample of 2 points at *random*

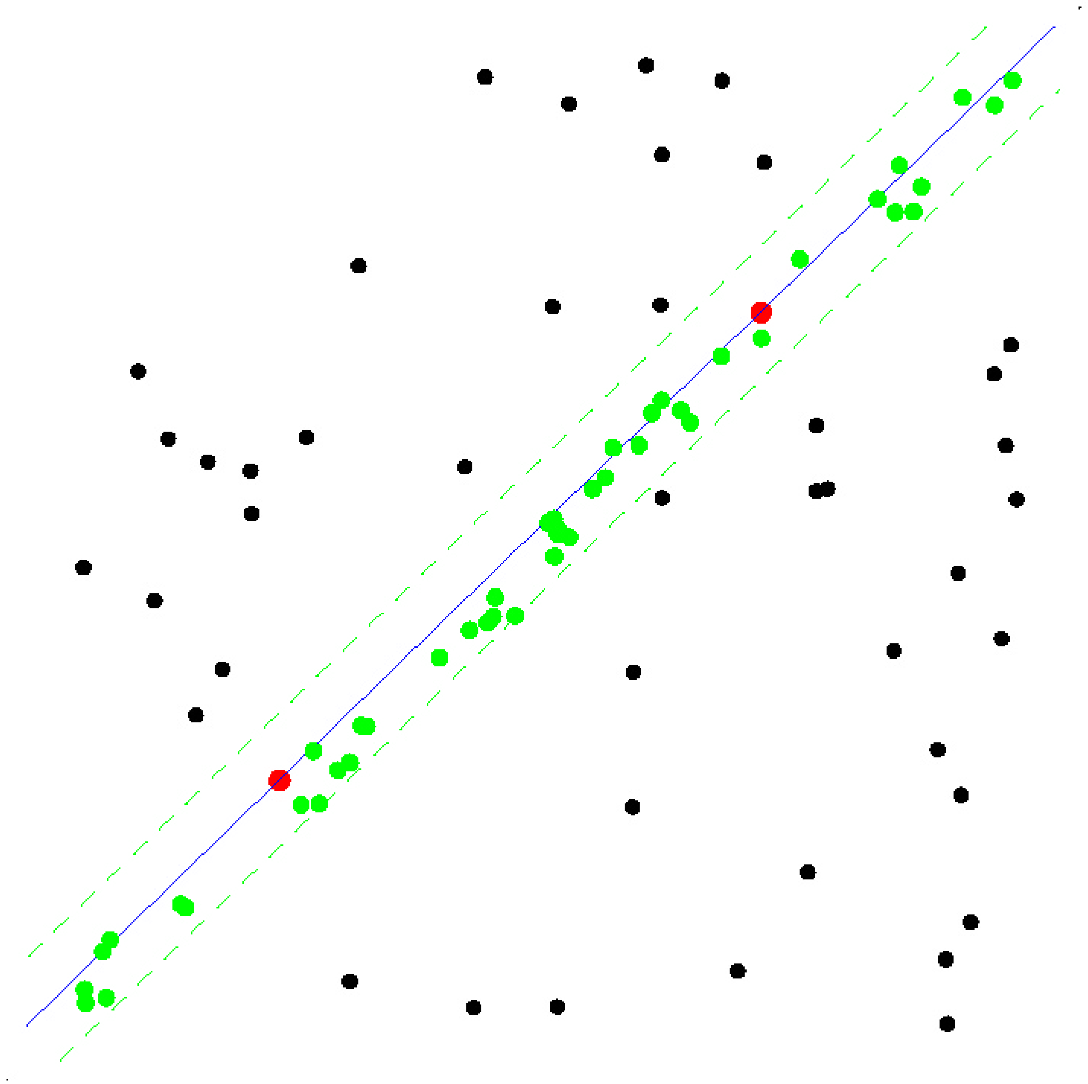
2. Calculate model parameters that fit the data in the sample

3. Calculate the residual error for each data point

4. Select data that support current hypothesis

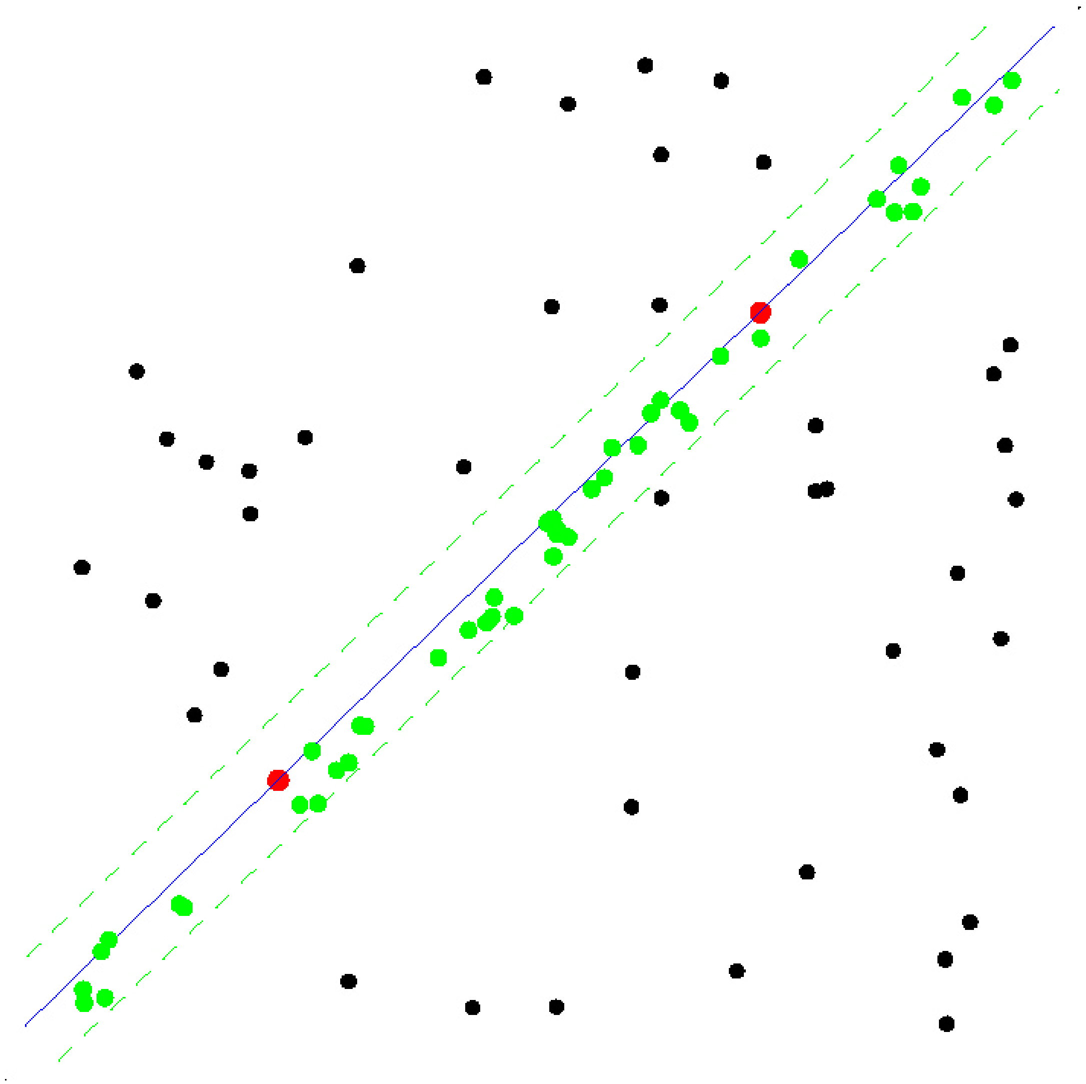
5. **Repeat from step 1 for k times**

RANSAC



1. Select a sample of 2 points at *random*
2. Calculate model parameters that fit the data in the sample
3. Calculate the residual error for each data point
4. Select data that support current hypothesis
5. Repeat from step 1 for k times
6. **Select the set with the maximum number of inliers obtained within k iterations**

RANSAC



1. Select a sample of 2 points at *random*

2. Calculate model parameters that fit the data in the sample

3. Calculate the residual error for each data point

4. Select data that support current hypothesis

5. Repeat from step 1 for k times

6. **Select the set with the maximum number of inliers obtained within k iterations**

7. Finally, calculate the model parameters using **all the inliers**

NB: RANSAC is non deterministic: every time you run it you may get a different result (due to the random hypotheses' generation process). Conversely, **EM and GNC** are **deterministic**

RANSAC

- How many iterations does RANSAC need?
- Ideally: check all possible combinations of 2 points in a dataset of N points.
- Number of all pairwise combinations: $\frac{N(N-1)}{2}$
  - computationally unfeasible if N is too large.
    Example, for 1000 points you need to check all 1000×999/2 ≅ 500'000 possibilities!

- Do we really need to check all possibilities or can we stop RANSAC after some iterations?
  - We will see that it is **enough to check a subset of all combinations if we have** a rough **estimate of the percentage of inliers** in our dataset
  - This can be done in a **probabilistic way**

RANSAC

- How many iterations does RANSAC need?
- $\boldsymbol{N}$:= total number of data points
- $\boldsymbol{w}$:= number of inliers $/N \rightarrow \boldsymbol{w}$: fraction of inliers in the dataset $\rightarrow$ $\boldsymbol{w}$ **=** P(selecting an inlier-point out of the dataset)
- Assumption: the 2 points necessary to estimate a line are selected independently
  - $\rightarrow \boldsymbol{w}^{\mathbf{2}} =$ P(both selected points are inliers)
  - $\rightarrow \mathbf{1} - \boldsymbol{w}^{\mathbf{2}} =$ P(at least one of these two points is an outlier)
- Let $\boldsymbol{k}$ be the number of RANSAC iterations executed so far
- $\rightarrow$ $(\mathbf{1} - \boldsymbol{w}^{\mathbf{2}})^{\boldsymbol{k}} =$ P(RANSAC never selected two points that are both inliers after $\boldsymbol{k}$ iterations)
- Let $\boldsymbol{p}$:= Probability to have selected at least two points that are both inliers after $\boldsymbol{k}$ iterations. We call $\boldsymbol{p}$ *Probability of Success*
- $\rightarrow \mathbf{1} - \boldsymbol{p} = (\mathbf{1} - \boldsymbol{w}^{\mathbf{2}})^{\boldsymbol{k}}$ and therefore:

$$k = \frac{\log(1 - p)}{\log(1 - w^2)}$$

RANSAC

- How many iterations does RANSAC need?

$$k = \frac{\log(1-p)}{\log(1-w^2)}$$

$\rightarrow$ knowing the fraction of inliers $\boldsymbol{w}$, after $\boldsymbol{k}$ iterations we will have a probability $\boldsymbol{p}$ of finding a set of points free of outliers

- Example: if we want a probability of success $\boldsymbol{p} = \mathbf{99}\%$ and we know that $\boldsymbol{w} = \mathbf{50}\% \rightarrow \boldsymbol{k} = 16$ iterations
    - these are **significantly fewer** than the number of **all possible combinations (500,000)**!
    - **Notice: the number of data points does not influence the minimum number of iterations $\boldsymbol{k}$, only $\boldsymbol{w}$ does!**

- In practice we only need a rough estimate of $\boldsymbol{w}$. More advanced variants of RANSAC estimate the fraction of inliers and adaptively update it at every iteration (how?)

RANSAC applied to Line Fitting

1. Initial: let *A* be a set of *N* points
2. **repeat**
3. Randomly select a sample of **2** points from *A*
4. **Fit a line** through the **2** points
5. Compute the **distances** of all other points **from this line**
6. Construct the inlier set (i.e. count the number of points whose distance $< d$)
7. Store these inliers
8. **until** maximum number of iterations $\boldsymbol{k}$ reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

$$k = \frac{\log(1-p)}{\log(1-w^2)}$$

RANSAC applied to General Model Fitting

1. Initial: let *A* be a set of *N* points
2. **repeat**
3. Randomly select a sample of $\boldsymbol{s}$ points from *A*
4. **Fit a model** from the $\boldsymbol{s}$ points
5. Compute the **distances** of all other points **from this model**
6. Construct the inlier set (i.e. count the number of points whose distance $< \ d$)
7. Store these inliers
8. **until** maximum number of iterations $\boldsymbol{k}$ reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

$$k = \frac{\log(1-p)}{\log(1-w^s)}$$

RANSAC applied to General Model Fitting

1. Initial: let *A* be a set of *N* points
2. **repeat**
3. Randomly select a sample of $\boldsymbol{s}$ points from *A*
4. **Fit a model** from the $\boldsymbol{s}$ points
5. Compute the **distances** of all other points **from this model**
6. Construct the inlier set (i.e. count the number of points whose distance $< d$)
7. Store these inliers
8. **until** maximum number of iterations $\boldsymbol{k}$ reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)}$$

NB: The formula is more commonly written as a function of the **fraction of outliers** $\boldsymbol{\varepsilon}$

The Three Key Ingredients of RANSAC

In order to implement RANSAC for Structure From Motion (SFM), we need three key ingredients:

1. What's the **model** in SFM?
2. What's the **minimum number of points** to estimate the model?
3. How do we compute the distance of a point from the model? In other words, can we define a **distance metric** that measures how well a point fits the model?

Answers

1. **What's the model** in SFM?
    - The **Essential Matrix** (for calibrated cameras) or the **Fundamental Matrix** (for uncalibrated cameras)
    - Alternatively, **R** and **T**

2. What's the **minimum number of points** to estimate the model?
    1. We know that 5 points is the theoretical minimum number of points for calibrated cameras
    2. However, if we use the *8-point algorithm*, then **8** is the minimum (for both calibrated or uncalibrated cameras)

3. How do we compute the **distance** of a point from the model?
    1. Algebraic error
    2. Directional error
    3. Epipolar line distance
    4. Reprojection error

Example: 8-point RANSAC applied to SFM

- Let's consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows

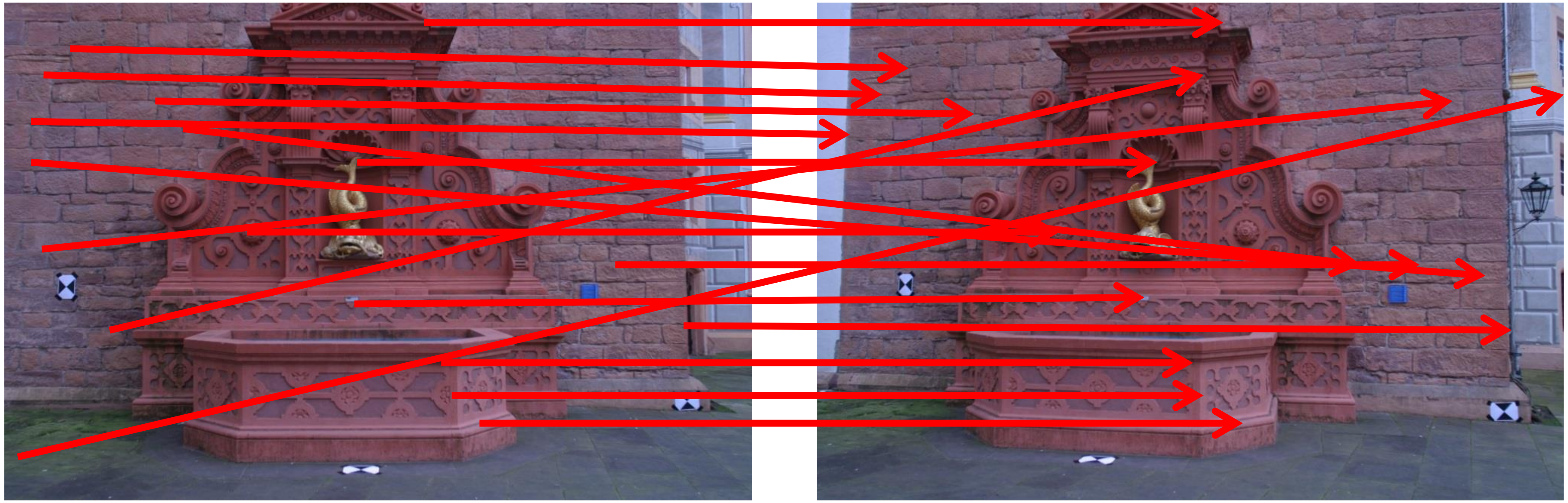


Image 1 Image 2

Example: 8-point RANSAC applied to SFM

- Let's consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

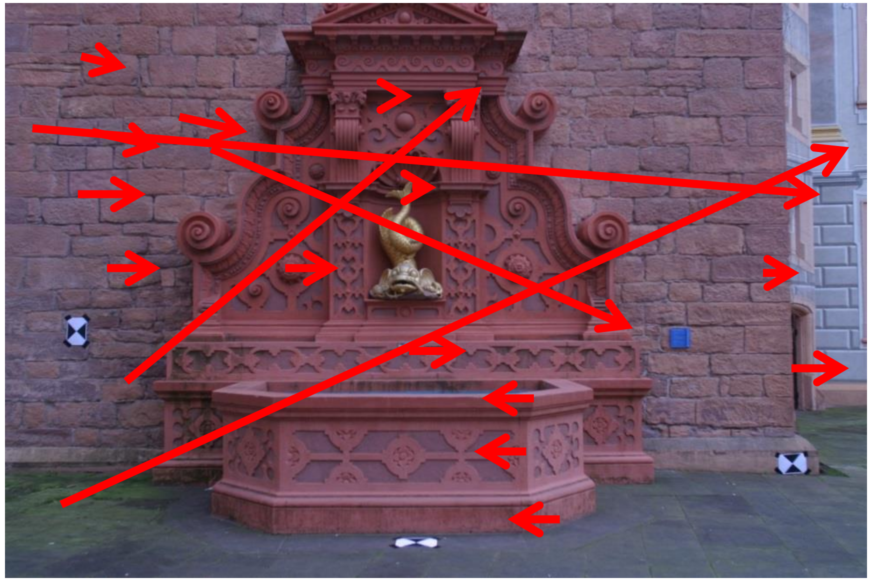


Image 1

Example: 8-point RANSAC applied to SFM

- Let’s consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

1. Randomly select 8 point correspondences and compute the model

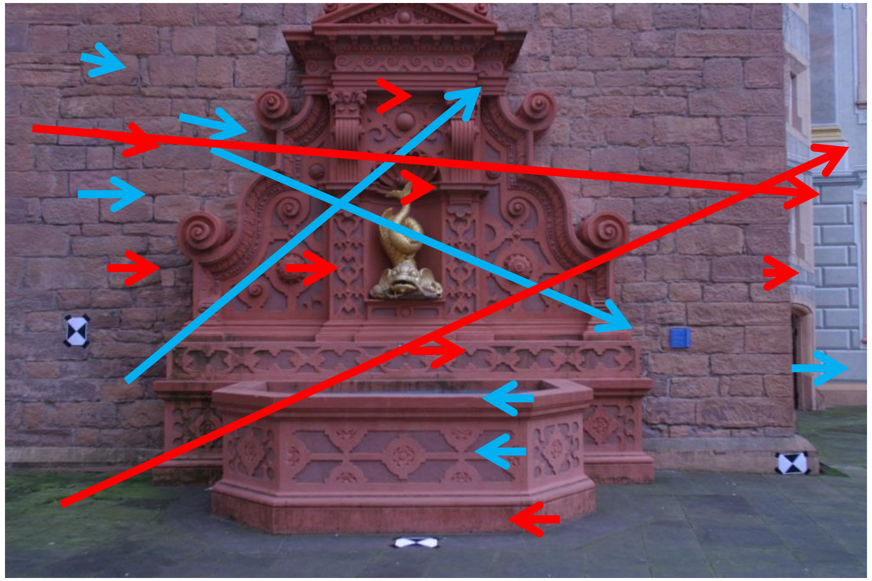


Image 1

Example: 8-point RANSAC applied to SFM

- Let’s consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

1. Randomly select 8 point correspondences and compute the model
2. Compute distance of all other points from this model and count the inliers

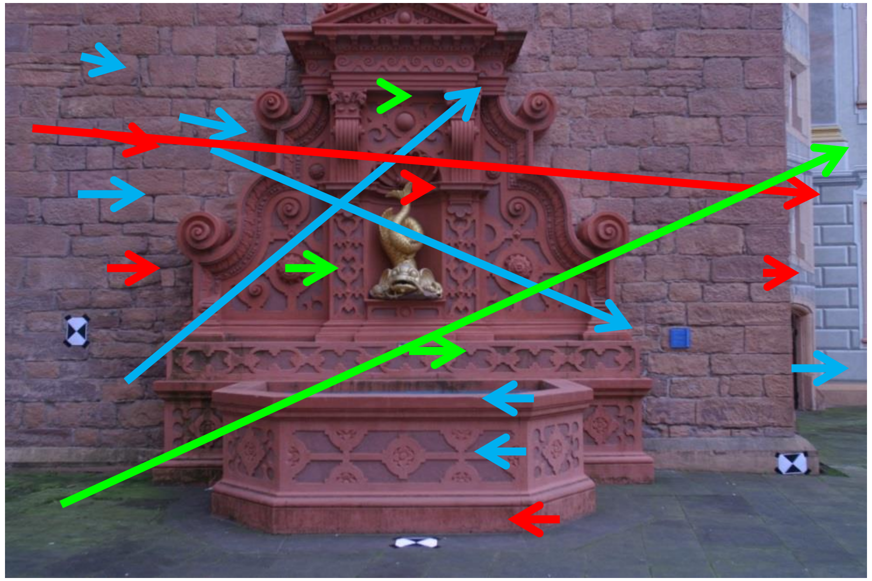


Image 1

Example: 8-point RANSAC applied to SFM

- Let's consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

1. Randomly select 8 point correspondences and compute the model
2. Compute distance of all other points from this model and count the inliers
3. Repeat from 1

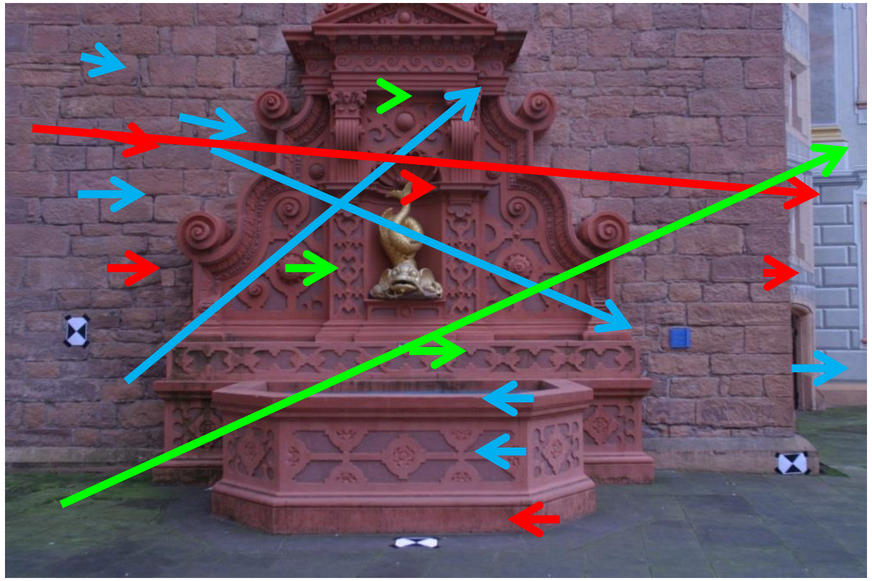


Image 1

Example: 8-point RANSAC applied to SFM

- Let's consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

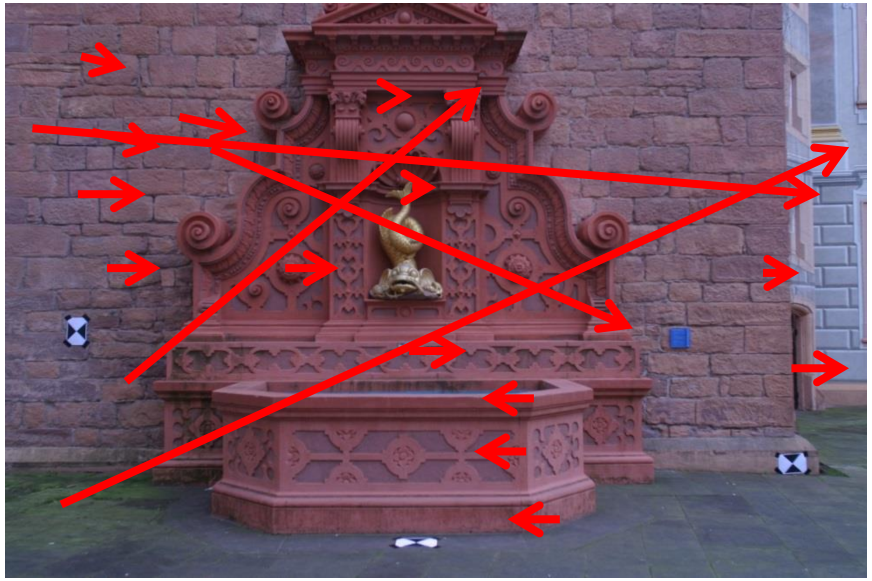


Image 1

Example: 8-point RANSAC applied to SFM

- Let’s consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

1. Randomly select 8 point correspondences and compute the model

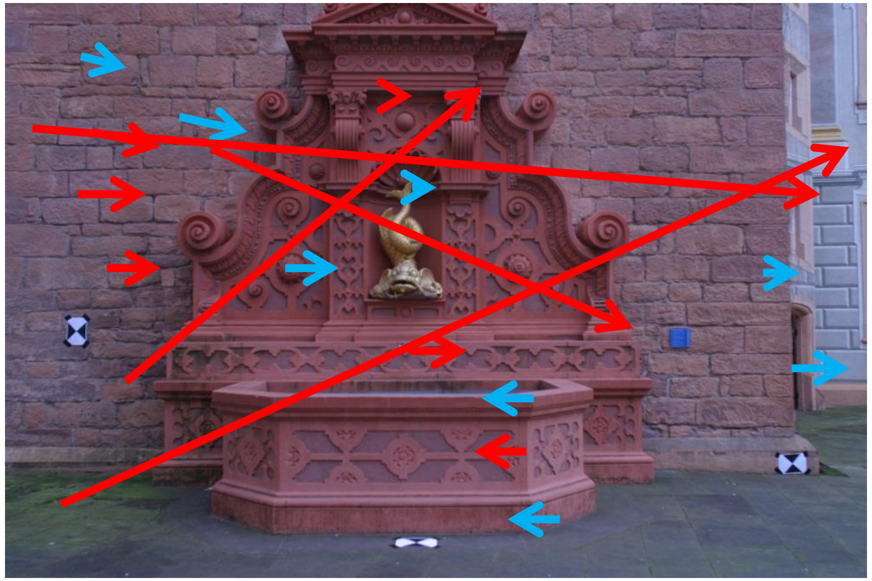


Image 1

Example: 8-point RANSAC applied to SFM

- Let's consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

1. Randomly select 8 point correspondences and compute the model
2. Compute distance of all other points from this model and count the inliers

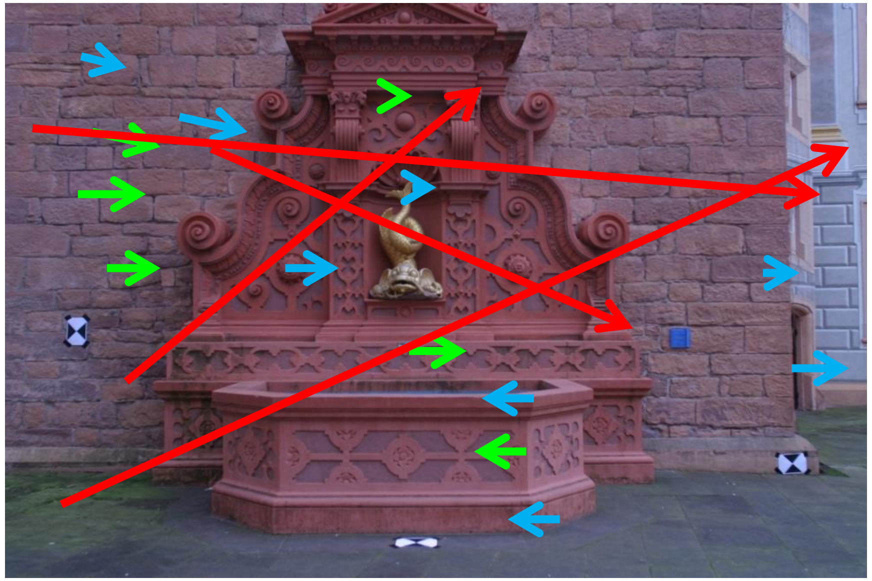


Image 1

Example: 8-point RANSAC applied to SFM

- Let's consider the following image pair and its image correspondences (e.g., Harris, SIFT, etc.), denoted by arrows
- For convenience, we overlay the features of the second image on the first image and use arrows to denote the *motion vectors* of the features

1. Randomly select 8 point correspondences and compute the model
2. Compute distance of all other points from this model and count the inliers
3. Repeat from 1 for $\boldsymbol{k}$ times

$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^8)}$$

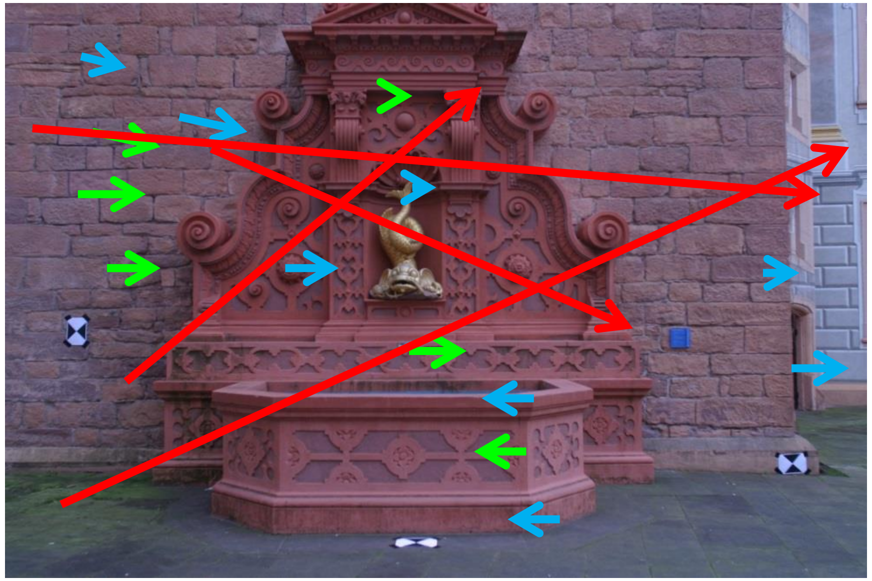


Image 1

RANSAC iterations $\boldsymbol{k}$ vs. $\boldsymbol{s}$

$\boldsymbol{k}$ increases exponentially with the number of points $\boldsymbol{s}$ estimate the model

Let's assume $\boldsymbol{p}$ = 99% and $\boldsymbol{\varepsilon}$ = 50% (fraction of outliers):

- **8-point RANSAC**
  - $\boldsymbol{s}$ = 8 points (8-point algorithm)

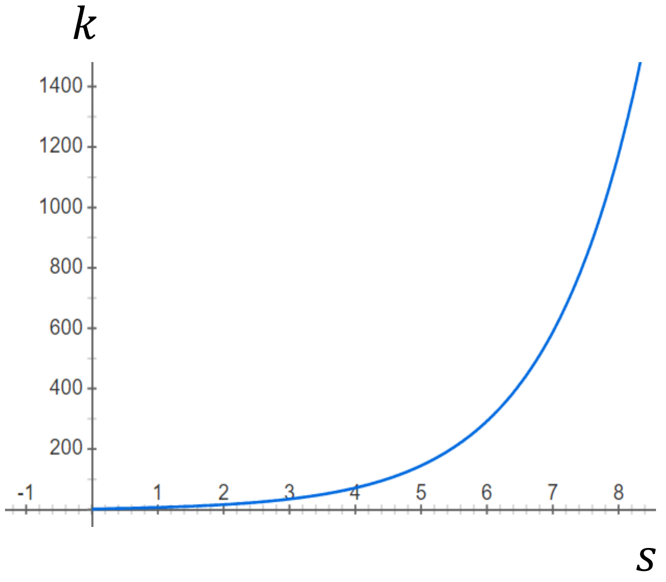
$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^8)} = 1177 \quad \textit{iterations}$$

- **5-point RANSAC**
  - $\boldsymbol{s}$ = 5 points (5-point algorithm)

$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^5)} = 145 \quad \textit{iterations}$$

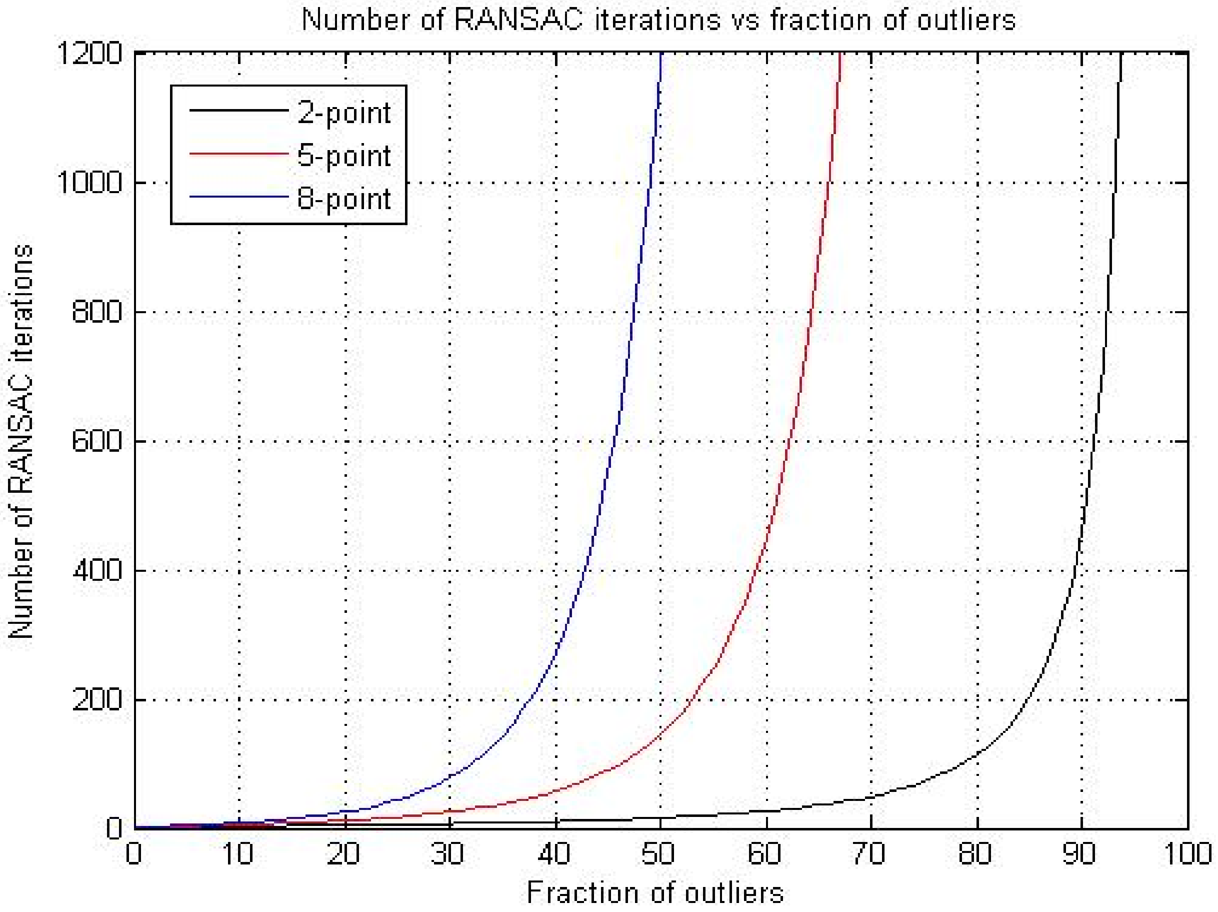
- **2-point RANSAC (e.g., line fitting)**
  - $\boldsymbol{s}$ = 2 points

$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^2)} = 16 \quad \textit{iterations}$$



RANSAC iterations $\boldsymbol{k}$ vs. $\boldsymbol{\varepsilon}$

$\boldsymbol{k}$ is increases exponentially with the fraction of outliers $\boldsymbol{\varepsilon}$:



These plots were computed assuming p = 99%

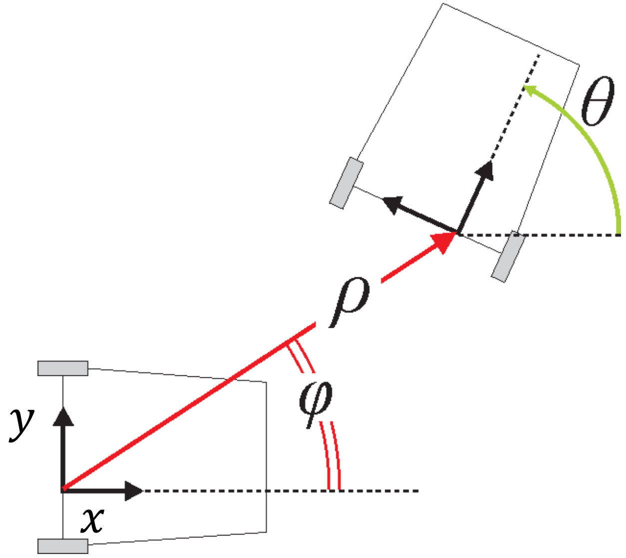
RANSAC iterations

- As observed, $\boldsymbol{k}$ is exponential with the number of points $\boldsymbol{s}$ necessary to estimate the model
- The **8-point algorithm** is extremely simple and was very successful; however, it requires more than **1177 iterations**
- Because of this, there has been a large interest by the research community in **using smaller motion parameterizations** (i.e., smaller $\boldsymbol{s}$)
- The first efficient solution to the minimal-case solution (5-point algorithm) took almost a century (Kruppa 1913 → Nister 2004)
- The **5-point RANSAC** (Nister 2004) only requires **145 iterations**; however:
  - The **5-point algorithm** can return **up to 10 solutions of E (worst case scenario)**
  - The **8-point algorithm** only returns a **unique solution of E**

Can we use less than 5 points?

Yes, if you use motion constraints!

Planar Motion



Planar motion is described by three parameters: *ϑ, φ, ρ*

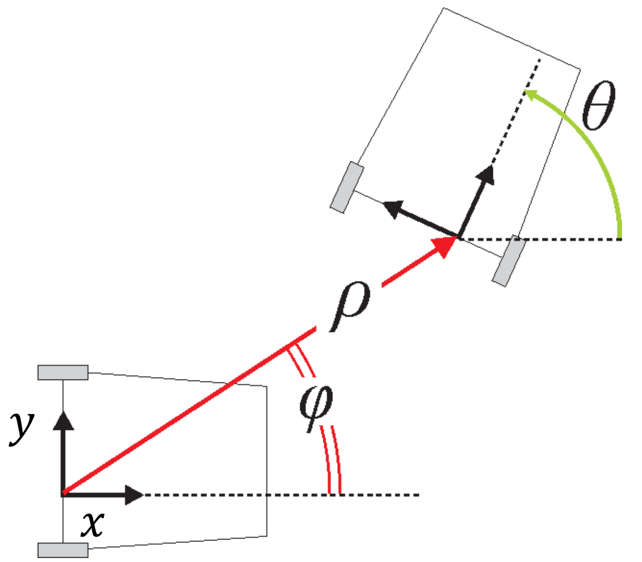
$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\varphi \\ \rho\sin\varphi \\ 0 \end{bmatrix}$$

Let’s compute the Epipolar Geometry

$$\mathrm{E} = [T_\times]R$$ Essential matrix

$$\overline{p}_2^T \; E \; \overline{p}_1 = 0$$ Epipolar constraint

Planar Motion



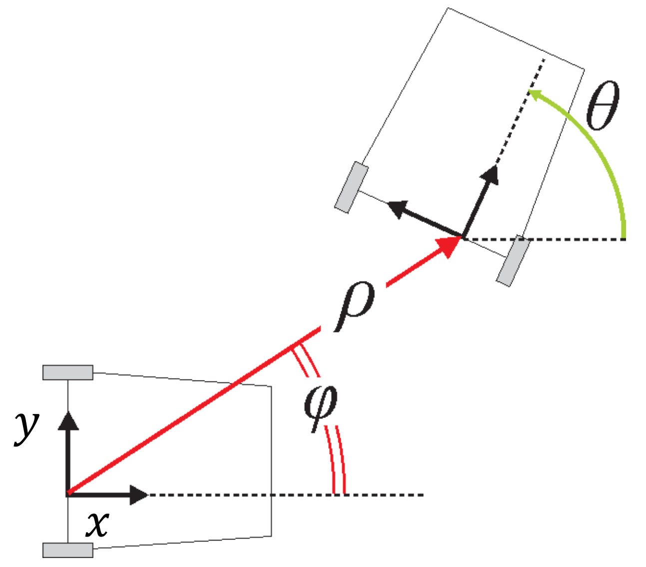
Planar motion is described by three parameters: *ϑ, φ, ρ*

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\varphi \\ \rho\sin\varphi \\ 0 \end{bmatrix}$$

Let’s compute the Epipolar Geometry

$$[T_\times] = \begin{bmatrix} 0 & 0 & \rho\sin\varphi \\ 0 & 0 & -\rho\cos\varphi \\ -\rho\sin\varphi & \rho\cos\varphi & 0 \end{bmatrix}$$

Planar Motion



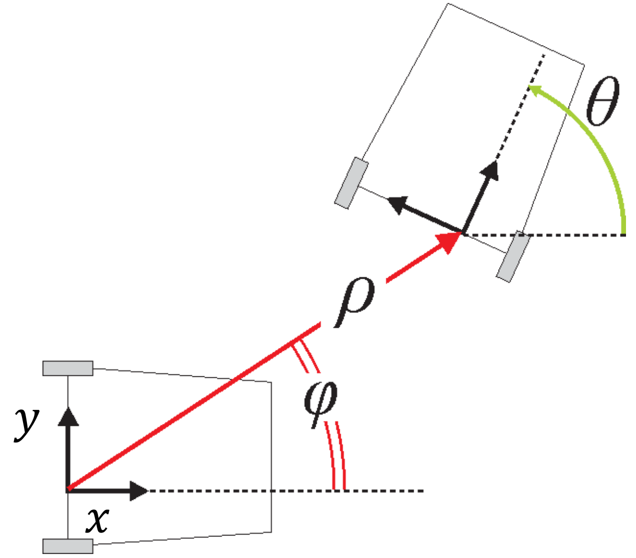
Planar motion is described by three parameters: ϑ, φ, ρ

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\varphi \\ \rho\sin\varphi \\ 0 \end{bmatrix}$$

Let's compute the Epipolar Geometry

$$E = [T_\times]R = \begin{bmatrix} 0 & 0 & \rho\sin\varphi \\ 0 & 0 & -\rho\cos\varphi \\ -\rho\sin\varphi & \rho\cos\varphi & 0 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Planar Motion



Planar motion is described by three parameters: ϑ, φ, ρ

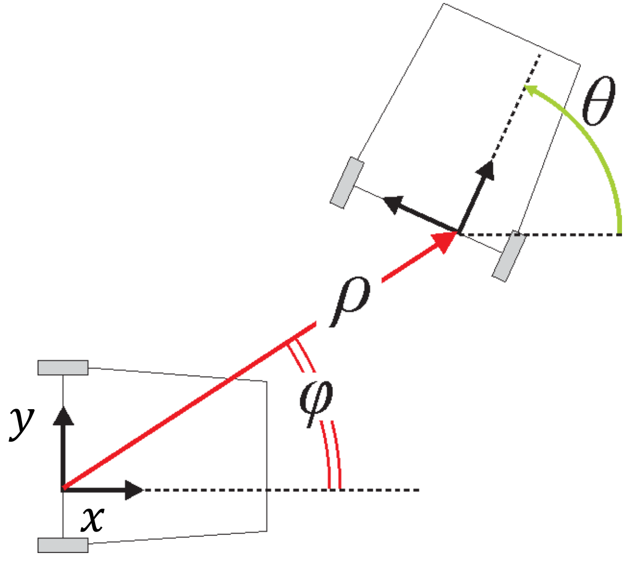
$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\varphi \\ \rho\sin\varphi \\ 0 \end{bmatrix}$$

Let's compute the Epipolar Geometry

$$E = [T_\times]R = \begin{bmatrix} 0 & 0 & \rho\sin(\varphi) \\ 0 & 0 & -\rho\cos(\varphi) \\ -\rho\sin(\varphi-\theta) & \rho\cos(\varphi-\theta) & 0 \end{bmatrix}$$

"2-Point RANSAC", Ortin & Montiel, Indoor robot motion based on monocular images, Robotica, 2001. PDF.

Planar Motion



Planar motion is described by three parameters: *ϑ, φ, ρ*

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\varphi \\ \rho\sin\varphi \\ 0 \end{bmatrix}$$

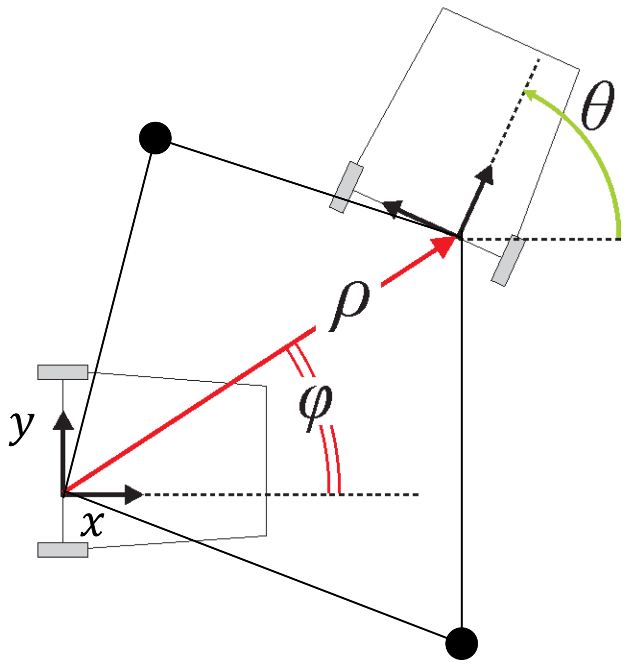
Let's compute the Epipolar Constraint: $\overline{p}_2^T \; E \; \overline{p}_1 = 0$

$$-u_1 sin(\phi-\theta) + v_1\cos(\phi-\theta) + u_2\sin(\phi) - v_2\cos(\phi) = 0$$

"2-Point RANSAC", Ortin & Montiel, Indoor robot motion based on monocular images, Robotica, 2001. PDF.

Planar Motion

Planar motion is described by three parameters: *ϑ, φ, ρ*



$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\varphi \\ \rho\sin\varphi \\ 0 \end{bmatrix}$$

Observe that ρ was cancelled out. Since only θ, φ can be determined and every point correspondence provides one scalar equation, then **2 point correspondences are sufficient** to estimate θ and φ

$$-u_1 sin(\phi - \theta) + v_1 \cos(\phi - \theta) + u_2 \sin(\phi) - v_2 \cos(\phi) = 0$$

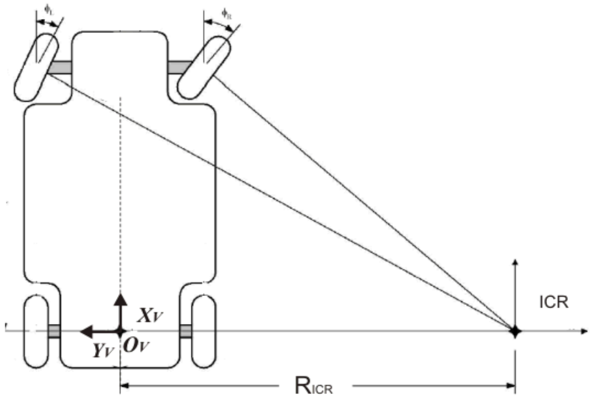
"2-Point RANSAC", Ortin & Montiel, Indoor robot motion based on monocular images, Robotica, 2001. PDF.

Less than 2 points?

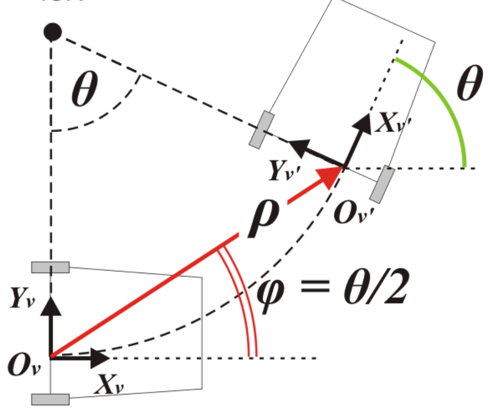
- Can we use less than 2 point correspondences?
    - Yes, if we exploit wheeled vehicles with **non-holonomic** constraints

Planar & Circular Motion (e.g., cars)

Wheeled vehicles, like cars, follow locally-planar circular motion about the Instantaneous Center of Rotation (ICR)



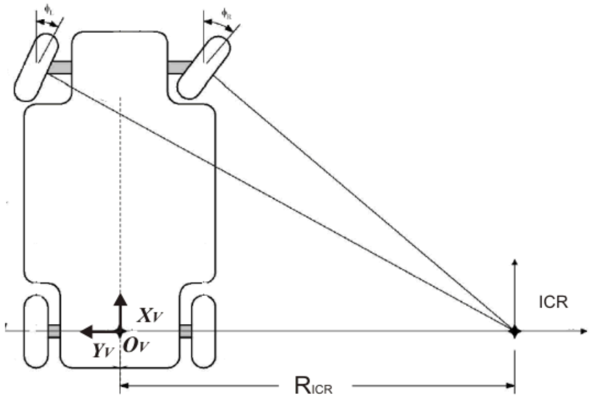
Example of Ackerman steering principle



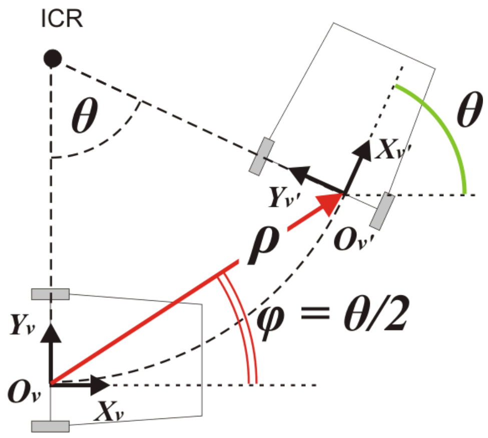
Locally-planar circular motion

Planar & Circular Motion (e.g., cars)

Wheeled vehicles, like cars, follow locally-planar circular motion about the Instantaneous Center of Rotation (ICR)



Example of Ackerman steering principle



Locally-planar circular motion

$\varphi = \theta/2$ => only 1 DoF (θ); thus, **only 1 point correspondence** is sufficient [Scaramuzza, 2011]

This is the smallest parameterization possible and results in the most efficient algorithm for removing outliers

Scaramuzza, 1-Point-RANSAC Structure from Motion for Vehicle-Mounted Cameras by Exploiting Non-Holonomic Constraints, International Journal of Computer Vision, 2011. PDF.

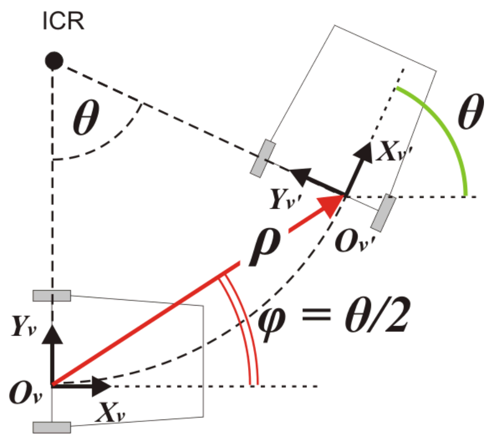
Planar & Circular Motion (e.g., cars)

Let's compute the Epipolar Geometry

$\mathrm{E} = [T_\times]R$ *Essential matrix*

$\bar{p}_2^T \; E \; \bar{p}_1 = 0$ *Epipolar constraint*

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\frac{\theta}{2} \\ \rho\sin\frac{\theta}{2} \\ 0 \end{bmatrix}$$



Locally-planar circular motion

Scaramuzza, 1-Point-RANSAC Structure from Motion for Vehicle-Mounted Cameras by Exploiting Non-Holonomic Constraints, International Journal of Computer Vision, 2011. PDF.

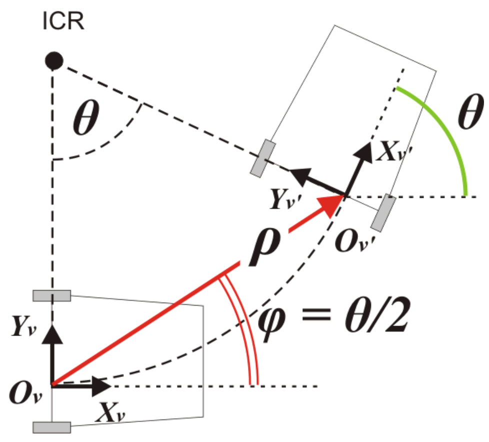
Planar & Circular Motion (e.g., cars)

Let's compute the Epipolar Geometry

$\mathrm{E} = [T_\times]R$ *Essential matrix*

$\bar{p}_2^T \ \mathrm{E} \ \bar{p}_1 = 0$ *Epipolar constraint*

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\frac{\theta}{2} \\ \rho\sin\frac{\theta}{2} \\ 0 \end{bmatrix}$$



Locally-planar circular motion

$$\mathrm{E} = [T_\times]R = \begin{bmatrix} 0 & 0 & \rho\sin\frac{\theta}{2} \\ 0 & 0 & -\rho\cos\frac{\theta}{2} \\ -\rho\sin\frac{\theta}{2} & \rho\cos\frac{\theta}{2} & 0 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \rho\sin\frac{\theta}{2} \\ 0 & 0 & \rho\cos\frac{\theta}{2} \\ \rho\sin\frac{\theta}{2} & -\rho\cos\frac{\theta}{2} & 0 \end{bmatrix}$$

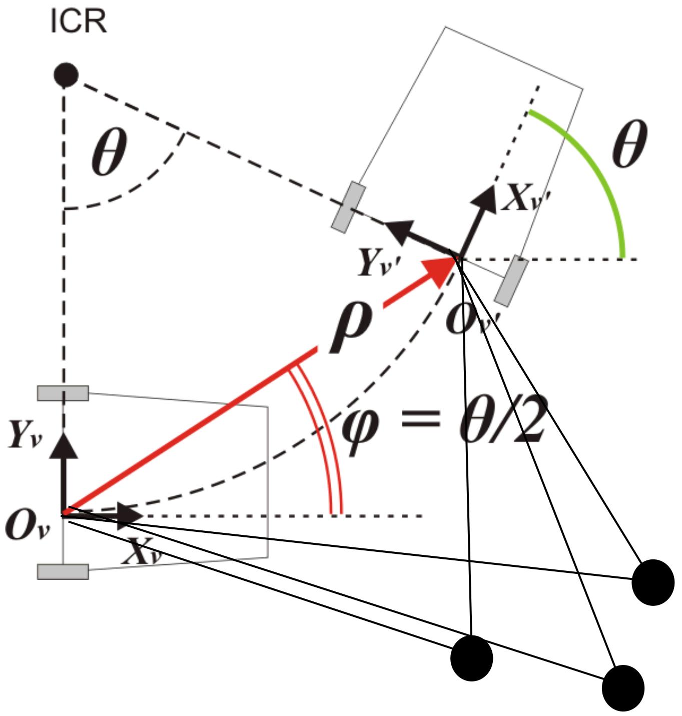
Notice that ρ can be cancelled out

$$\bar{p}_2^T \ \mathrm{E} \ \bar{p}_1 = 0 \ \Rightarrow \ \sin\left(\frac{\theta}{2}\right)\cdot(u_2 + u_1) + \cos\left(\frac{\theta}{2}\right)\cdot(v_2 - v_1) = 0$$

$$\theta = -2\tan^{-1}\left(\frac{v_2 - v_1}{u_2 + u_1}\right)$$

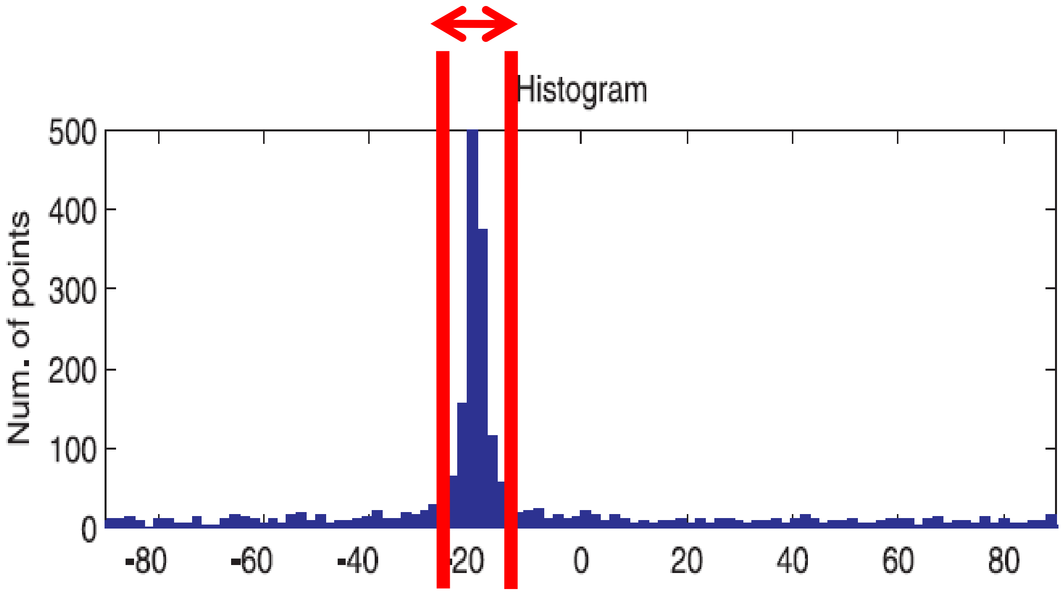
Scaramuzza, 1-Point-RANSAC Structure from Motion for Vehicle-Mounted Cameras by Exploiting Non-Holonomic Constraints, International Journal of Computer Vision, 2011. PDF.

1-Point RANSAC Algorithm



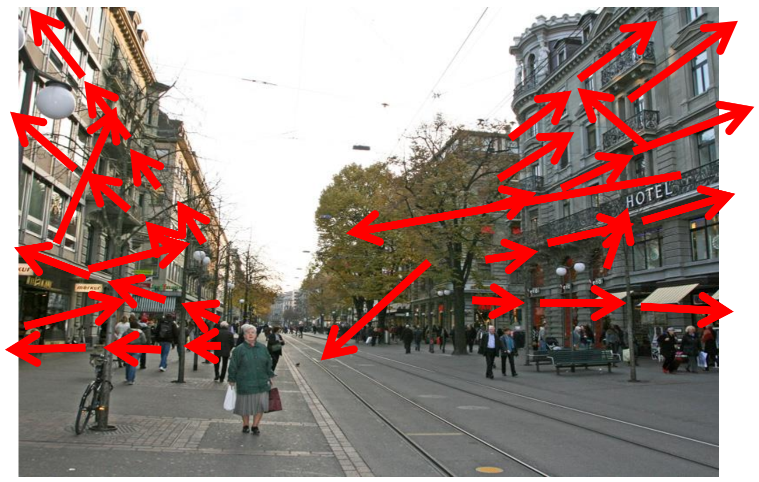
Compute θ for every point correspondence

$$\theta = -2\tan^{-1}\left(\frac{v_2 - v_1}{u_2 + u_1}\right)$$



Only 1 iteration!

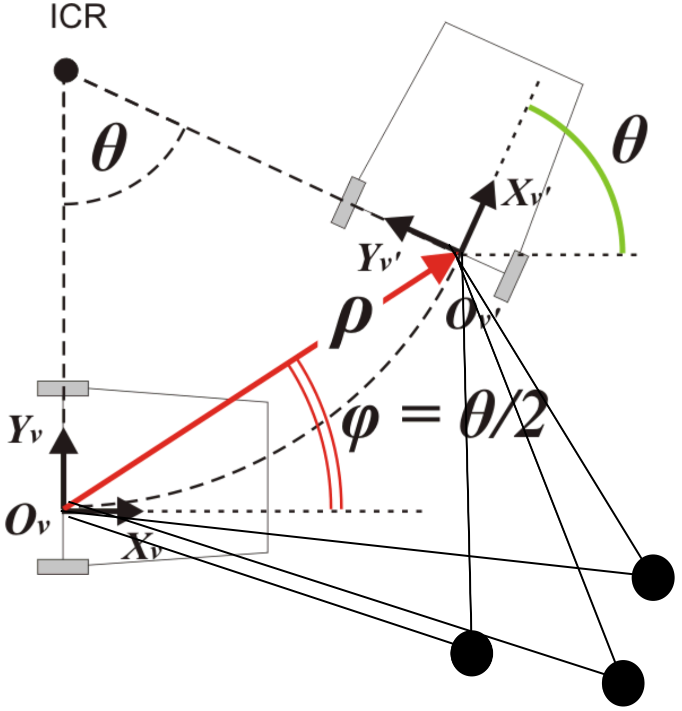
The most efficient algorithm for removing outliers (<1ms)



1-Point RANSAC is ONLY used to find the inliers.

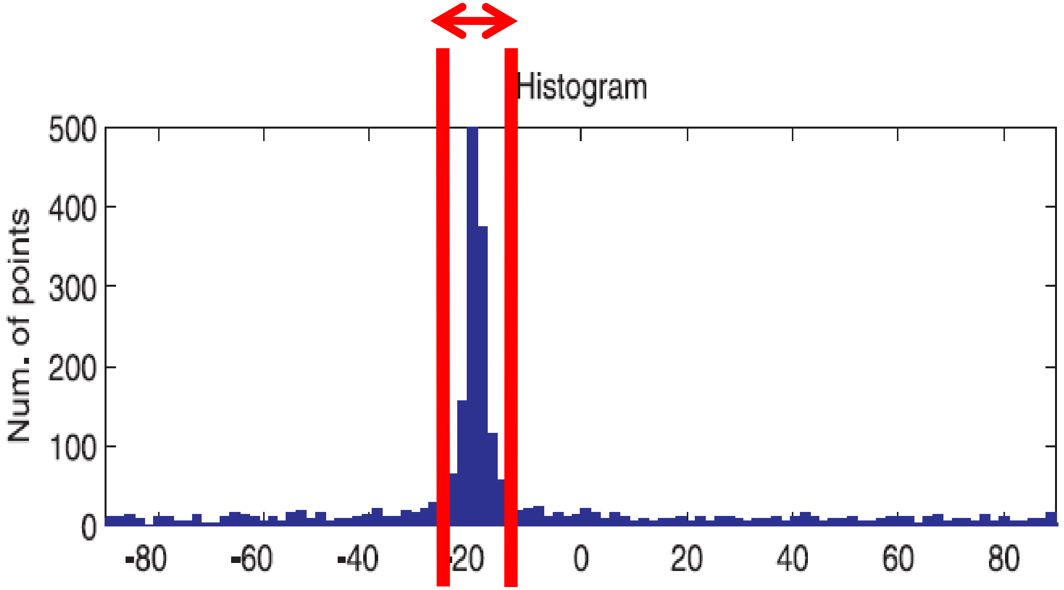
Motion is then estimated from them in 6DOF

1-Point RANSAC Algorithm



Compute ϑ for every point correspondence

$$\theta = -2\tan^{-1}\left(\frac{v_2 - v_1}{u_2 + u_1}\right)$$



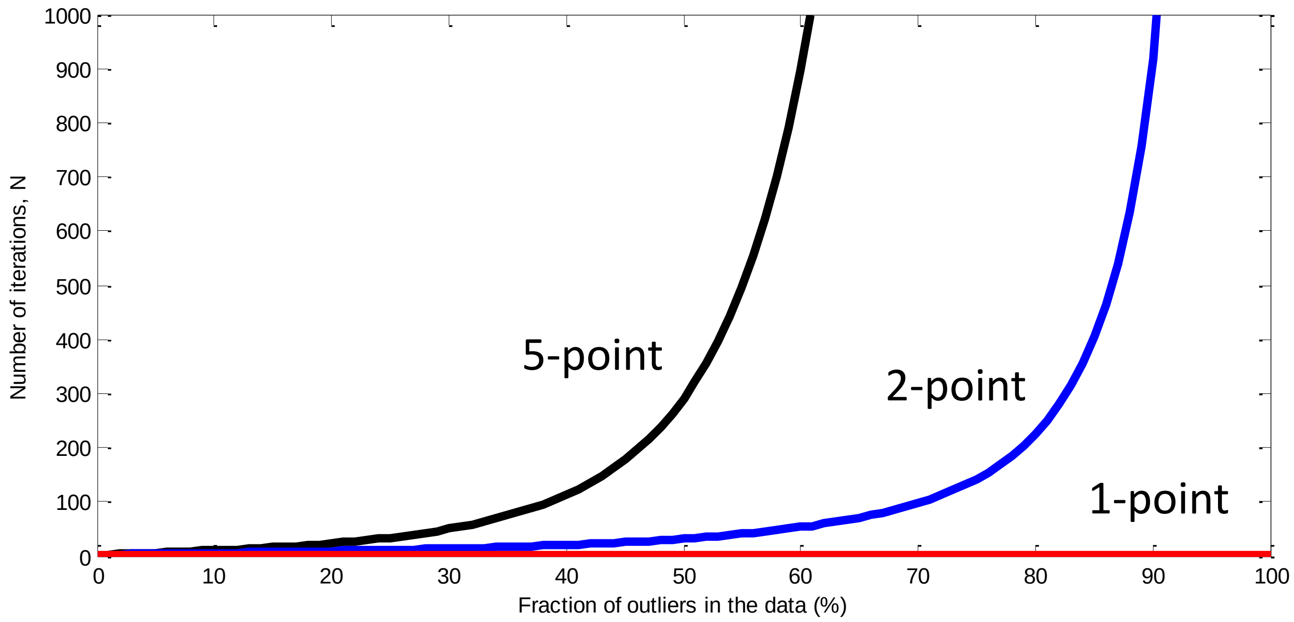
Only 1 iteration!

The most efficient algorithm for removing outliers (<1ms)

1-Point RANSAC is ONLY used to find the inliers.

Motion is then estimated from them in 6DOF

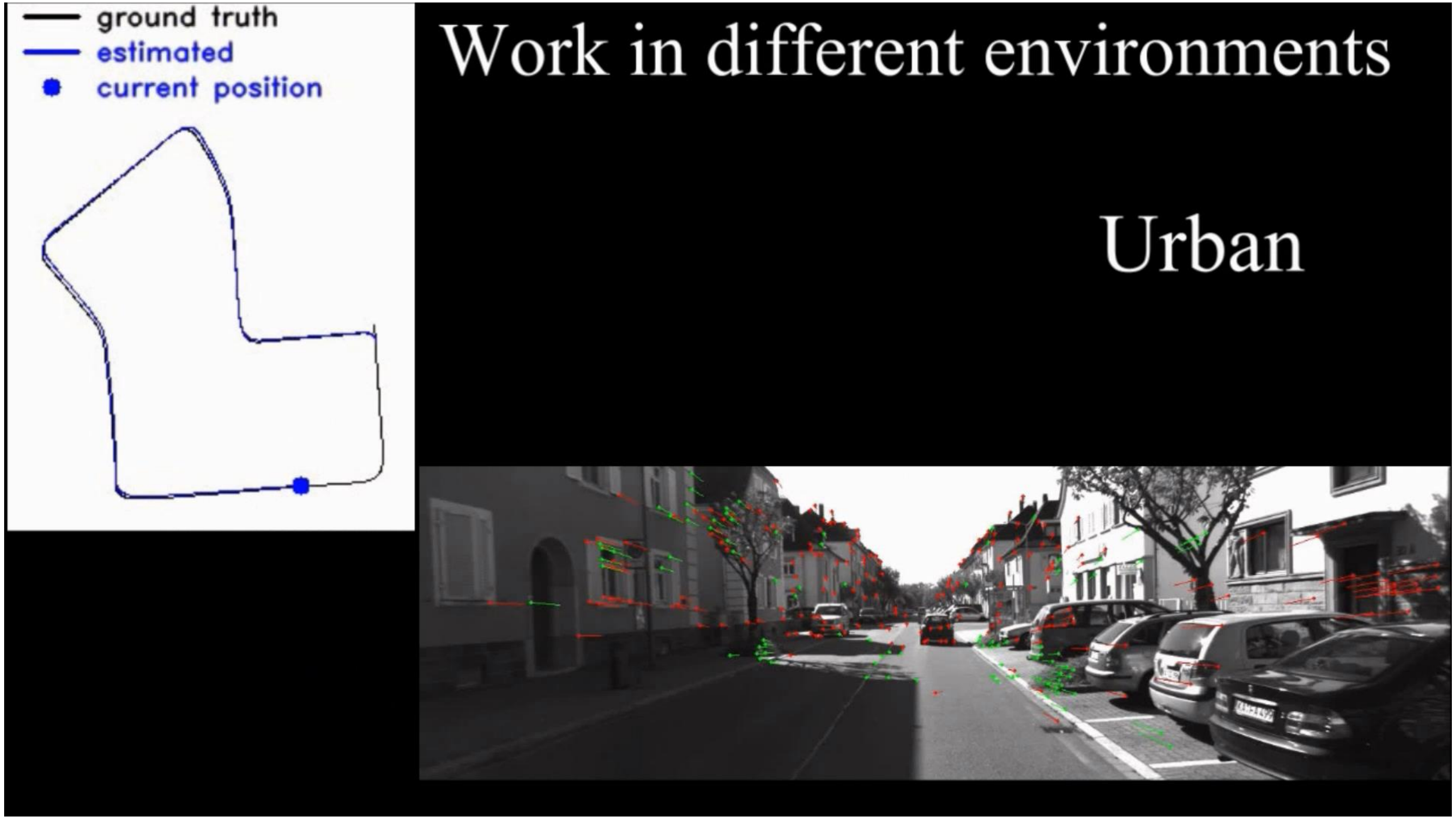
Comparison of RANSAC algorithms



$$N = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)} \quad \text{where we typically use } p = 99\%$$

	8-Point RANSAC [Longuet-Higgins'81]	5-Point RANSAC [Nister'04]	2-Point RANSAC [Ortin'01]	1-Point RANSAC [Scaramuzza'11]
Numb. of iterations	**> 1177**	**>145**	**>16**	**=1**

Visual Odometry with 1-Point RANSAC



Scaramuzza, 1-Point-RANSAC Structure from Motion for Vehicle-Mounted Cameras by Exploiting Non-Holonomic Constraints, International Journal of Computer Vision, 2011. PDF.

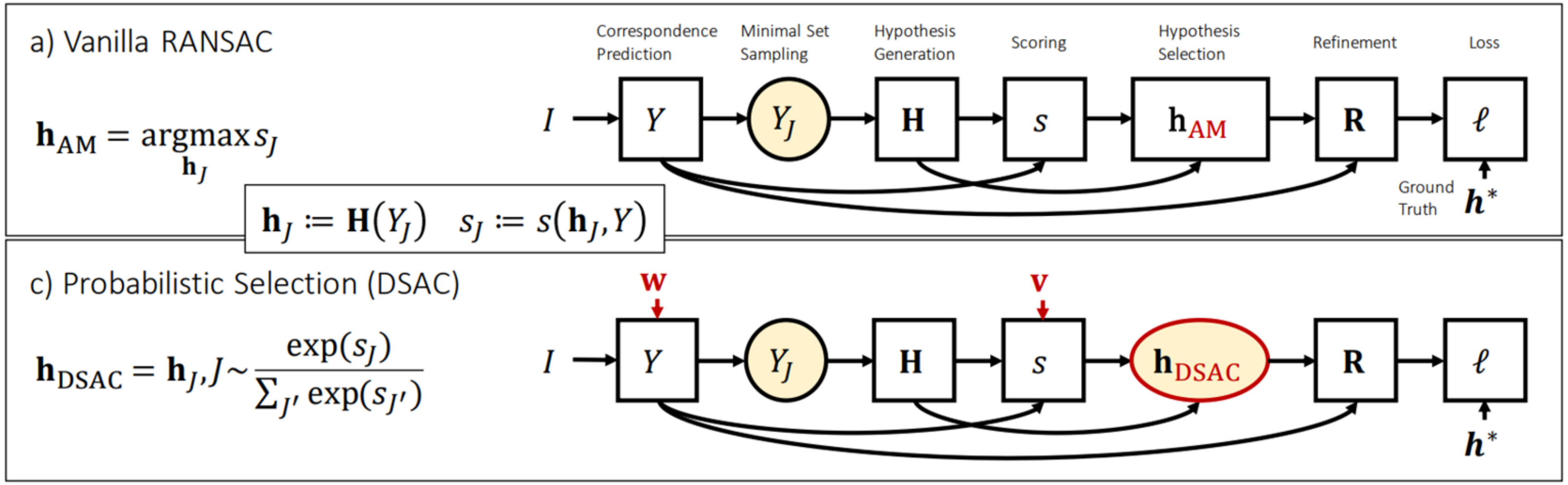
Latest and Greatest ☺

Differentiable RANSAC

- RANSAC is not differentiable since it relies on selecting a hypothesis based on maximizing the number of inliers (i.e., argmax).
- DSAC shows how sample consensus can be used in a differentiable way
- This enables the use of sample consensus in a variety of learning tasks.

Choose the best based on score

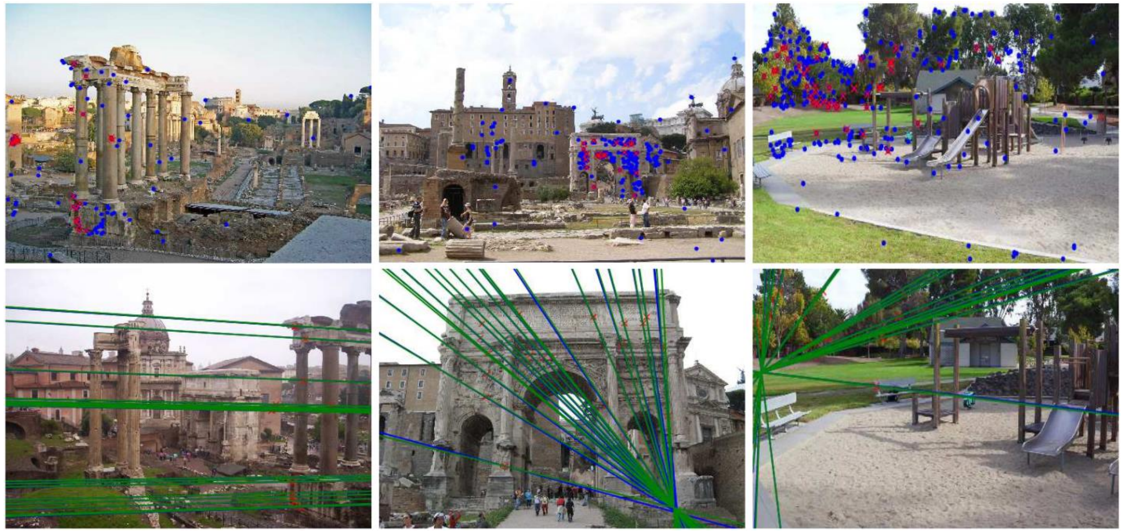
Randomly sample based on score



E. Brachmann et al., DSAC - Differentiable RANSAC for Camera Localization, International Conference on Computer Vision and Pattern Recognition (CVPR), 2017. PDF. Video.

Deep Fundamental Matrix Estimation

- **Input**: two sets of noisy local features (coordinates + descriptors) contaminated by outliers
- **Output**: fundamental matrix
- **Idea**: solve a weighted homogeneous least-squares problem, where robust weights are estimated using deep networks
- **Robust**: handles extreme wide-baseline image pairs



Top-bottom as image-pair

Red: inlier correspondences
Blue: outlier correspondences

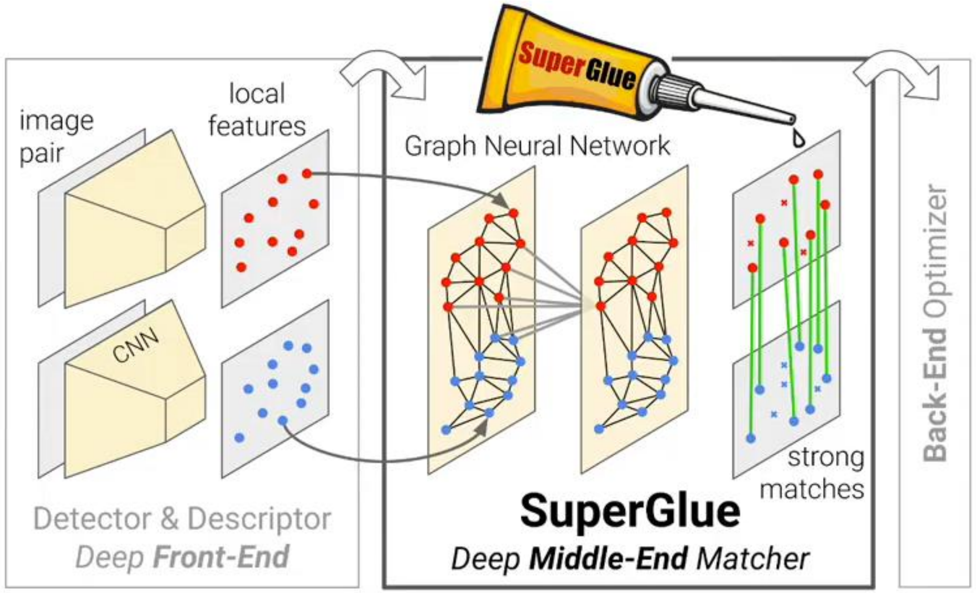
Epipolar lines

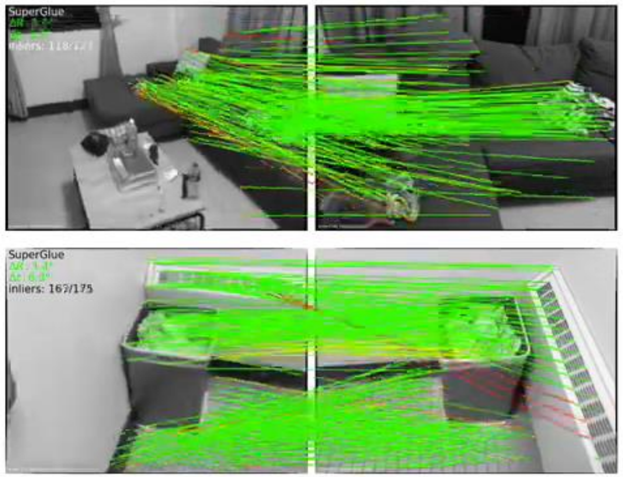
Green: estimated
Blue: ground-truth

Ranftl, Koltun, *Deep Fundamental Matrix Estimation*, European Conference on Computer Vision (ECCV), 2018. PDF.

SuperGlue: Learning Feature Matching with Graph Neural Networks

- **Input**: two sets of noisy local features (coordinates + descriptors) contaminated by outliers
- **Output**: strong & outlier-free matches
- **Combines deep learning with classical optimization** (Graph Neural Networks, Attention, Optimal Transport
- **Robust**: handles extreme wide-baseline image pairs





Sarlin, DeTone, Malisiewicz, Rabinovich, *SuperGlue: Learning Feature Matching with Graph Neural Networks*, International Conference on Computer Vision and Pattern Recognition (CVPR), 2020. PDF. Code.

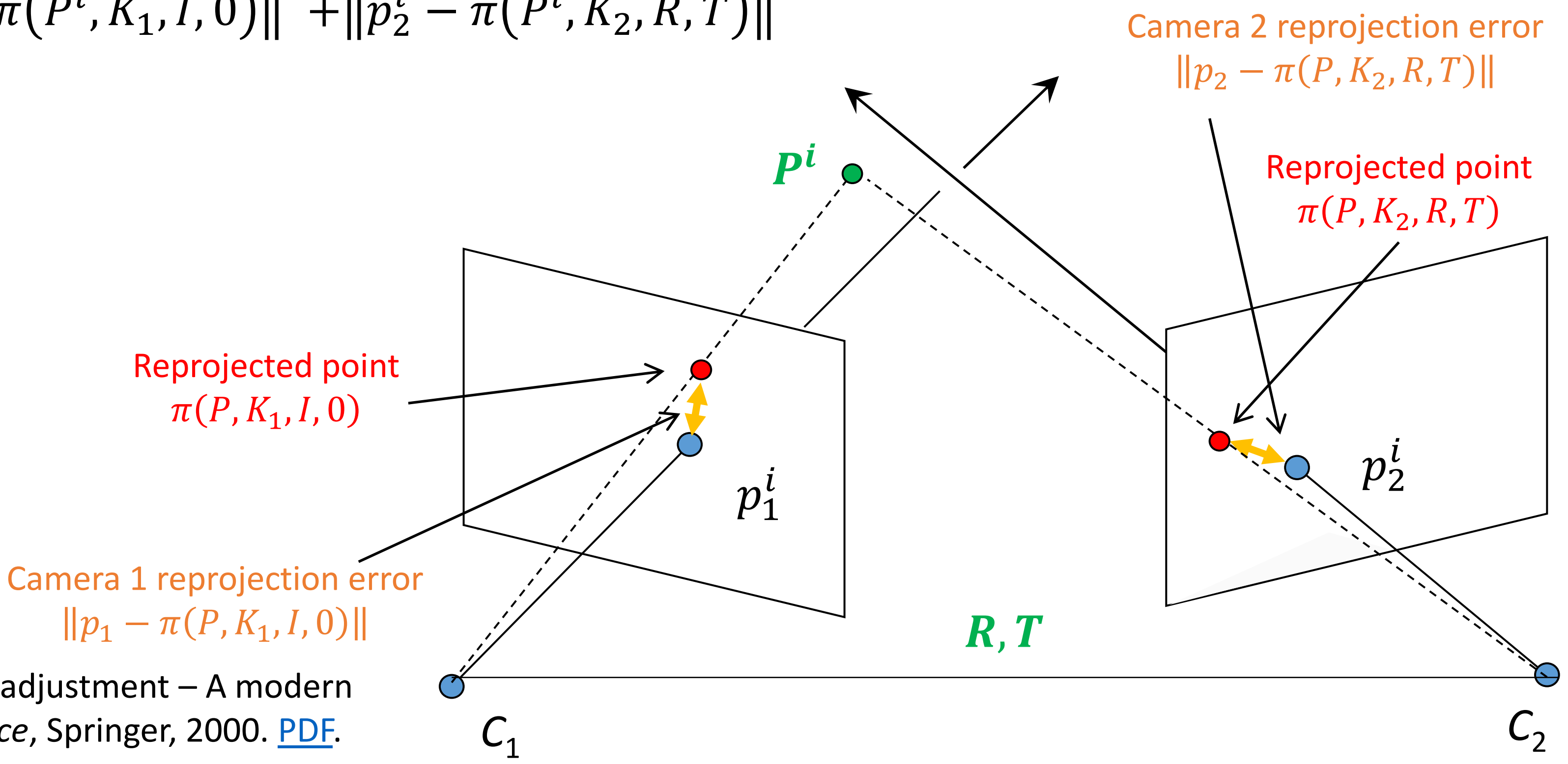
Outline

- Robust Structure from Motion
- Bundle Adjustment

2-View Bundle Adjustment (BA)

- **Non-linear, joint optimization of structure, $\boldsymbol{P^i}$, and motion $\boldsymbol{R, T}$**
- Commonly used after least square estimation of R and T (e.g., after 8- or 5-point algorithm)
- Optimizes P^i, R, T by minimizing the **Sum of Squared Reprojection Errors**:

$$P^i, R, T = argmin_{P^i,R,T} \sum_{i=1}^{N} \left\| p_1^i - \pi\left(P^i, K_1, I, 0\right) \right\|^2 + \left\| p_2^i - \pi\left(P^i, K_2, R, T\right) \right\|^2$$



Triggs, McLauchlan, Hartley, Fitzgibbon, Bundle adjustment – A modern synthesis, *Vision Algorithms: Theory and Practice*, Springer, 2000. PDF.

2-View Bundle Adjustment (BA)

- **Non-linear, joint optimization of structure, $\boldsymbol{P^i}$, and motion $\boldsymbol{R, T}$**
- Commonly used after least square estimation of R and T (e.g., after 8- or 5-point algorithm)
- Optimizes P^i, R, T by minimizing the **Sum of Squared Reprojection Errors**:

$$P^i, R, T = argmin_{P^i,R,T} \sum_{i=1}^{N} \left\| p_1^i - \pi\left(P^i, K_1, I, 0\right) \right\|^2 + \left\| p_2^i - \pi\left(P^i, K_2, R, T\right) \right\|^2$$

Good to know:

- Like in the formula, we typically assume the first camera as the world frame, but it's arbitrary
- Occasionally, the residual terms are weighted
- In order to not get stuck in local minima, the **initial values of** P^i, R, T **should be close to the optimum**
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)
- **Can be modified to also optimize the intrinsic parameters**
- Implementation details in **Exercise 9**

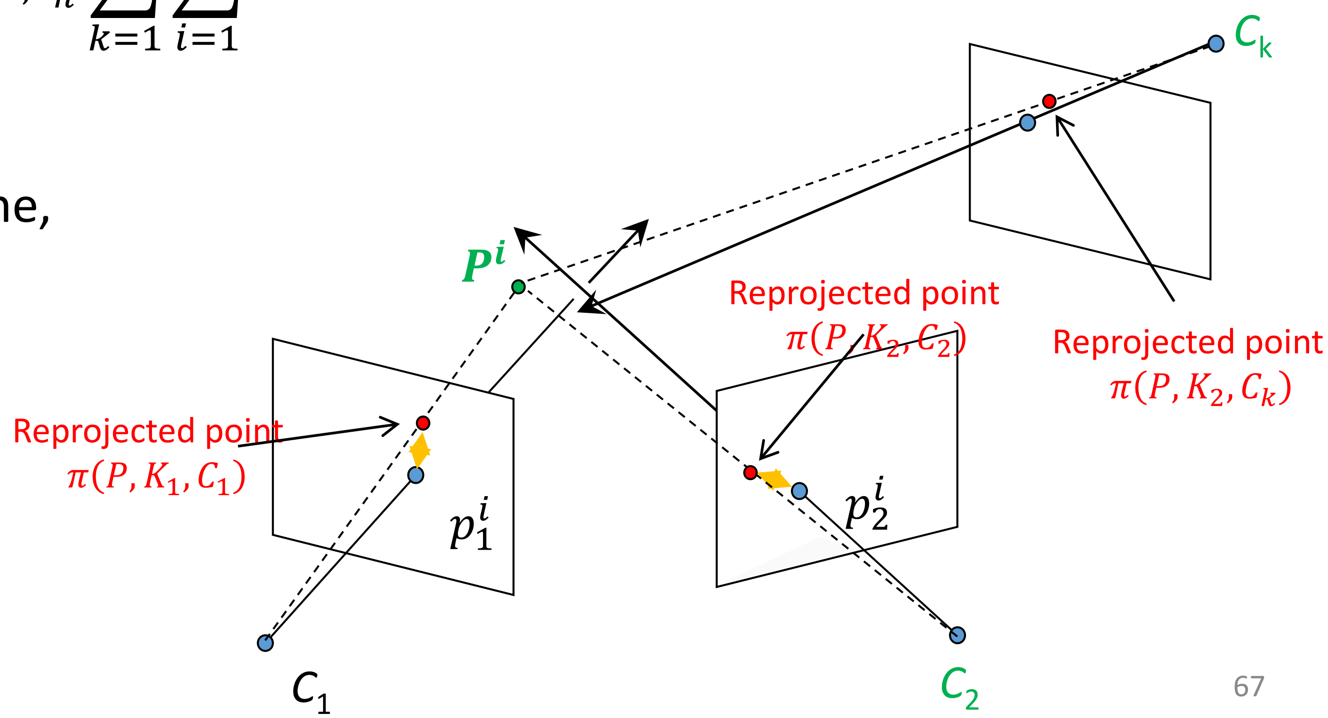
What is the key difference with the reprojection error minimization seen in previous lectures (Lecture 3, slide 21, and Lecture 7, slide 26)?

n-View Bundle Adjustment (BA)

- **Non-linear, joint optimization of structure, $\boldsymbol{P^i}$, and camera poses** $C_1 = [I, 0], \dots, C_k = [R_k, T_k]$
- Minimizes the Sum of Squared Reprojection Errors **across all views**

$$P^i, C_2, \dots, C_n = argmin_{P^i, C_2, \dots, C_n} \sum_{k=1}^{n} \sum_{i=1}^{N} \left\| p_k^i - \pi\left(P^i, K_k, C_k\right) \right\|^2$$

- **NB**: we assume the first camera as the world frame, that's why $C_1 = [I, 0]$



Triggs, McLauchlan, Hartley, Fitzgibbon, Bundle adjustment – A modern synthesis, *Vision Algorithms: Theory and Practice*, Springer, 2000. PDF.

Huber and Tukey Norms

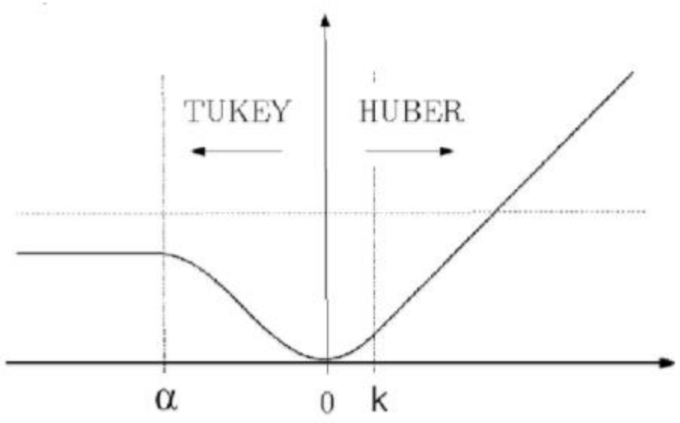
- To prevent that large reprojection errors can negatively impact the optimization, a more robust norm ρ( ) is used instead of the L_2:

$$P^i, C_2, \dots, C_n = argmin_{P^i, C_2, \dots, C_n} \sum_{k=1}^{n} \sum_{i=1}^{N} \rho\left(p_k^i - \pi(P^i, K_k, C_k)\right)$$

- ρ( ) is a robust cost function (**Huber or Tukey**) to alleviate the contribution of wrong matches:

- **Huber norm**: $\rho(x) = \begin{cases} x^2 & \text{if } |x| \leq k \\ k(2|x| - k) & \text{if } |x| \geq k \end{cases}$

- **Tukey norm**: $\rho(x) = \begin{cases} \alpha^2 & \text{if } |x| \geq \alpha \\ \alpha^2\left(1 - \left(1 - \left(\frac{x}{\alpha}\right)^2\right)^3\right) & \text{if } |x| \leq \alpha \end{cases}$



These formulas are not asked at the exam but their plots and meaning is asked ☺

Things to remember

- EM algorithm
- RANSAC algorithm and its application to SFM
- 8 vs 5 vs 1 point RANSAC, pros and cons
- Bundle Adjustment

Reading

- CH. 8.1.4, 8.3.1, 11.3 of Szeliski book, 2nd edition
- Ch. 14.2 of Corke book

Understanding Check

Are you able to answer the following questions?

- What are the causes of outliers?
- What effects may outliers have on VO?
- How does EM work? What are the issues?
- Why do we need RANSAC?
- What is the theoretical maximum number of combinations to explore?
- After how many iterations can RANSAC be stopped to guarantee a given success probability?
- What is the trend of RANSAC vs. iterations, vs. the fraction of outliers, vs. the number of points to estimate the model?
- How do we apply RANSAC to the 8-point algorithm, DLT, P3P?
- How can we reduce the number of RANSAC iterations for the SFM problem? (1- and 2-point RANSAC)
- Bundle Adjustment. Mathematical expression and illustration. Tukey and Huber norms.