ROBOTICS & PERCEPTION GROUP

Universität Zürich[UZH]

Lecture 09
Multiple View Geometry 3

Prof. Dr. Davide Scaramuzza

sdavide@ifi.uzh.ch

Today’s outline

- RANSAC for robust Structure from Motion
- Visual Odometry

“Robust” Structure from Motion

- All Structure-from-Motion algorithms (including the 8-point algorithms) assume that **image correspondences** are **correct**
- However, finding the correct correspondences is not always successful
  - We call false image correspondences **outliers**
  - We call correct image correspondences **inliers**

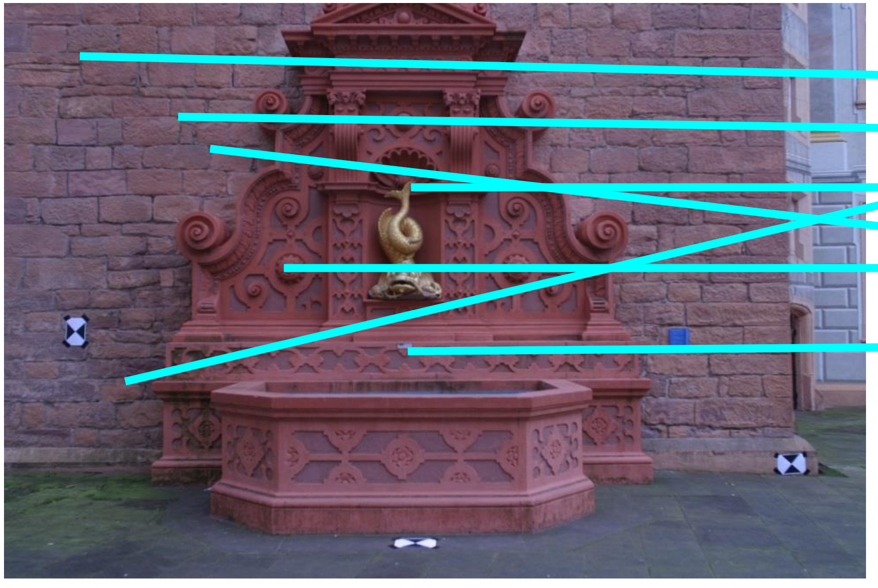


Image 1

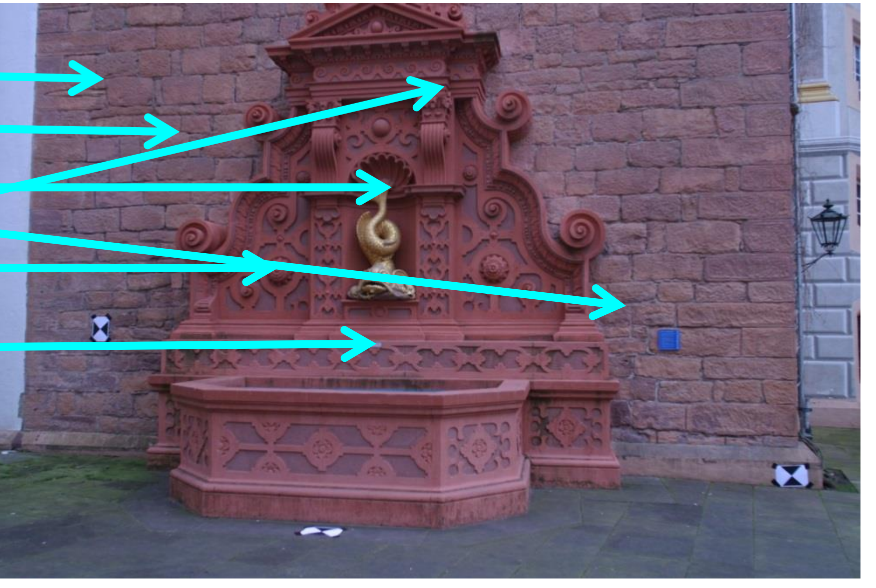


Image 2

“Robust” Structure from Motion

- All Structure-from-Motion algorithms (including the 8-point algorithms) assume that **image correspondences** are **correct**
- However, finding the correct correspondences is not always successful
  - We call false image correspondences **outliers** **(assuming that the scene is static)**
  - We call correct image correspondences **inliers**

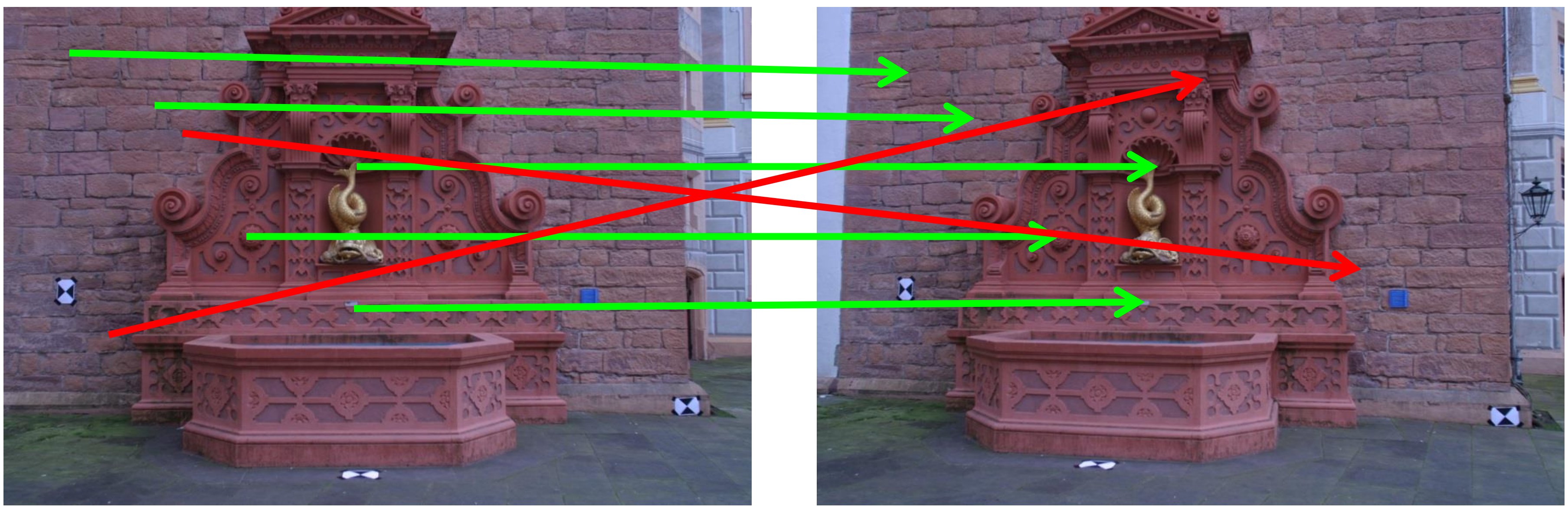


Image 1 Image 2

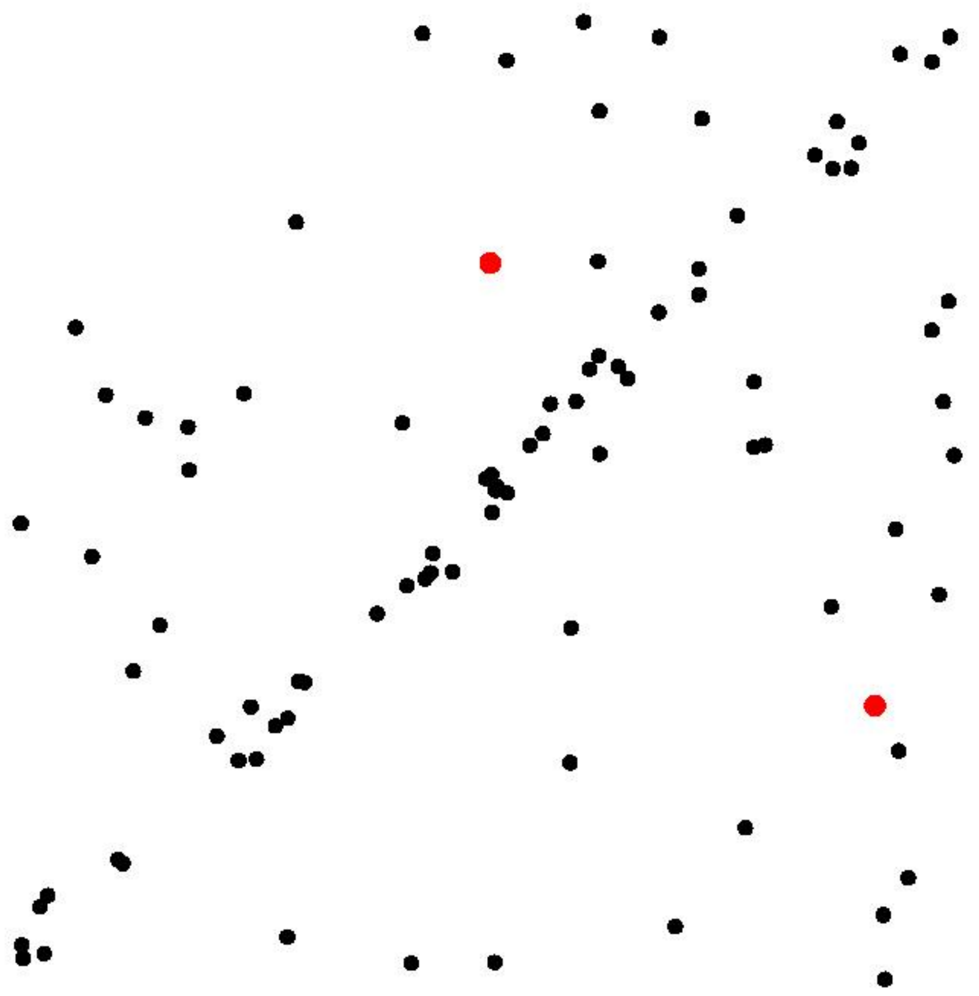
RANSAC (RAndom SAmple Consensus)

- RANSAC is the **standard 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, homography, etc.)
- Let's review RANSAC for line fitting and see how we can adapt it to SfM

M. A.Fischler and R. C.Bolles. Random sample consensus: A paradigm for model fitting with apphcatlons to image analysis and automated cartography. Graphics and Image Processing, 24(6):381–395, 1981.

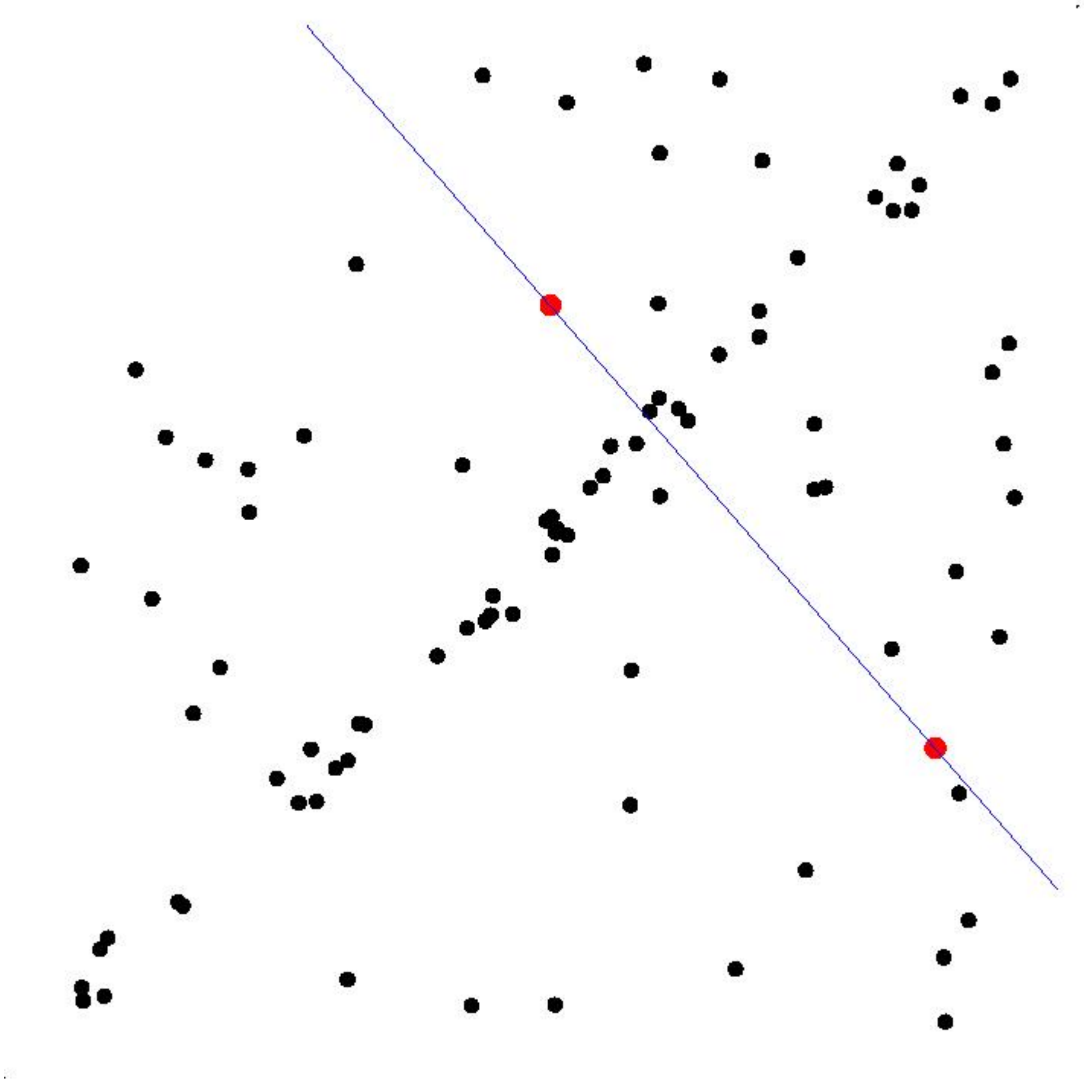
RANSAC

RANSAC



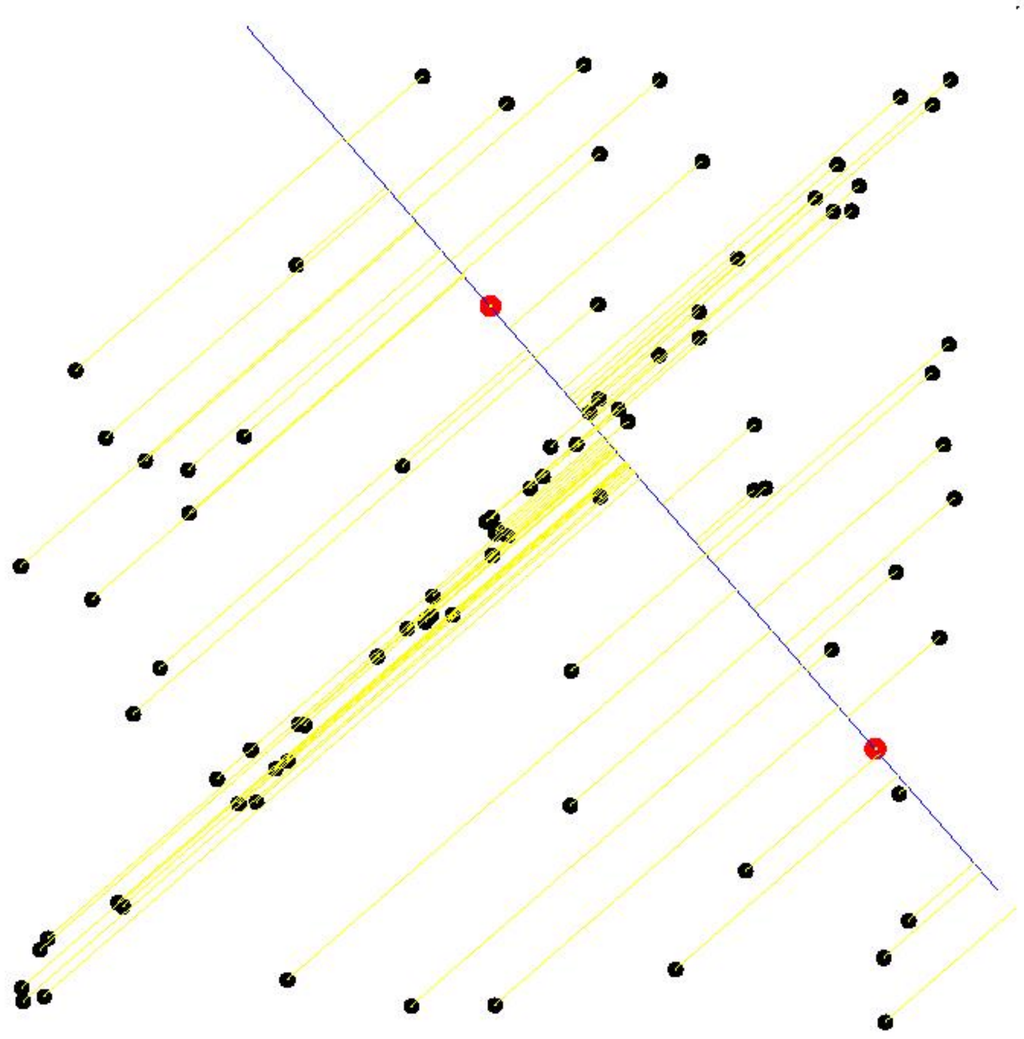
- **Select sample of 2 points at random**

RANSAC



- Select sample of 2 points at random
- **Calculate model parameters that fit the data in the sample**

RANSAC

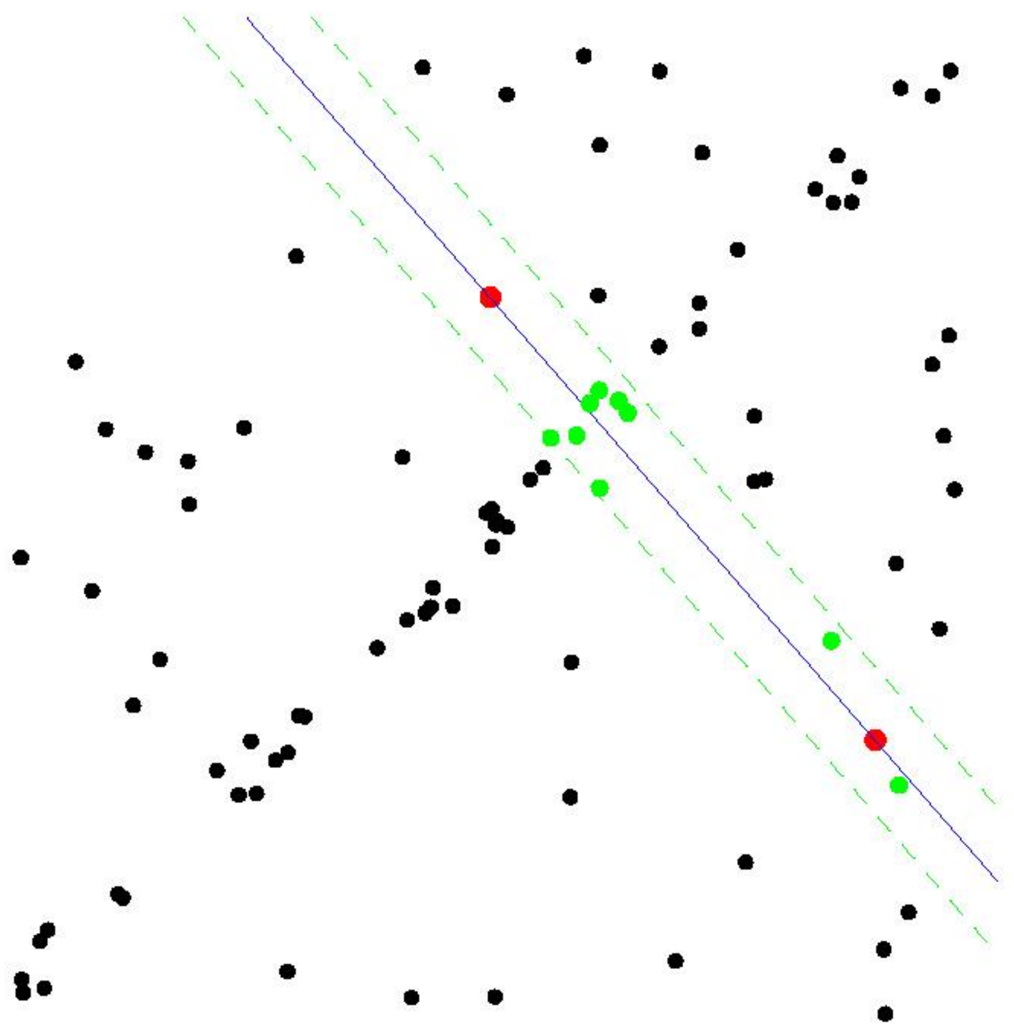


• Select sample of 2 points at random

• Calculate model parameters that fit the data in the sample

• **Calculate error function for each data point**

RANSAC



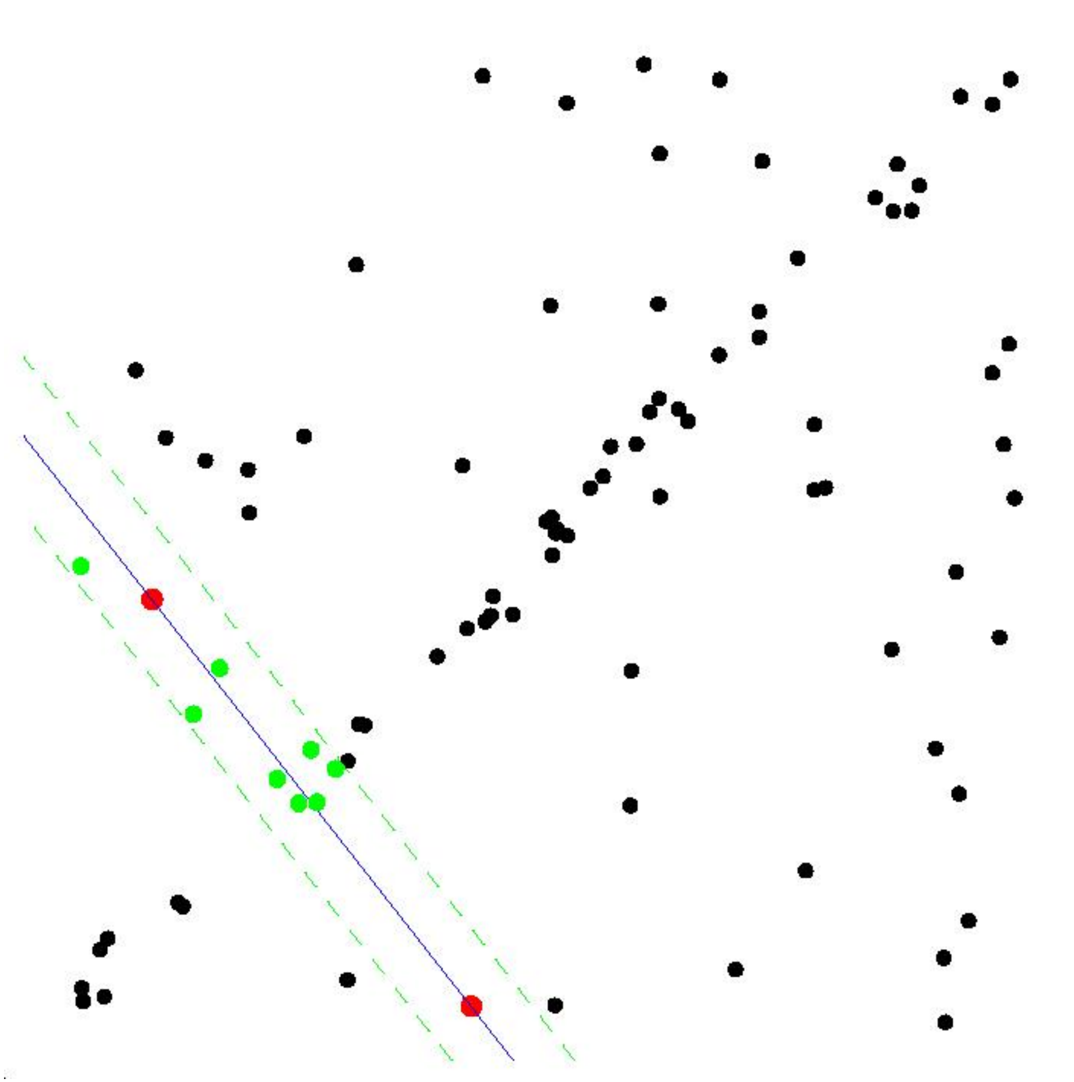
• Select sample of 2 points at random

• Calculate model parameters that fit the data in the sample

• Calculate error function for each data point

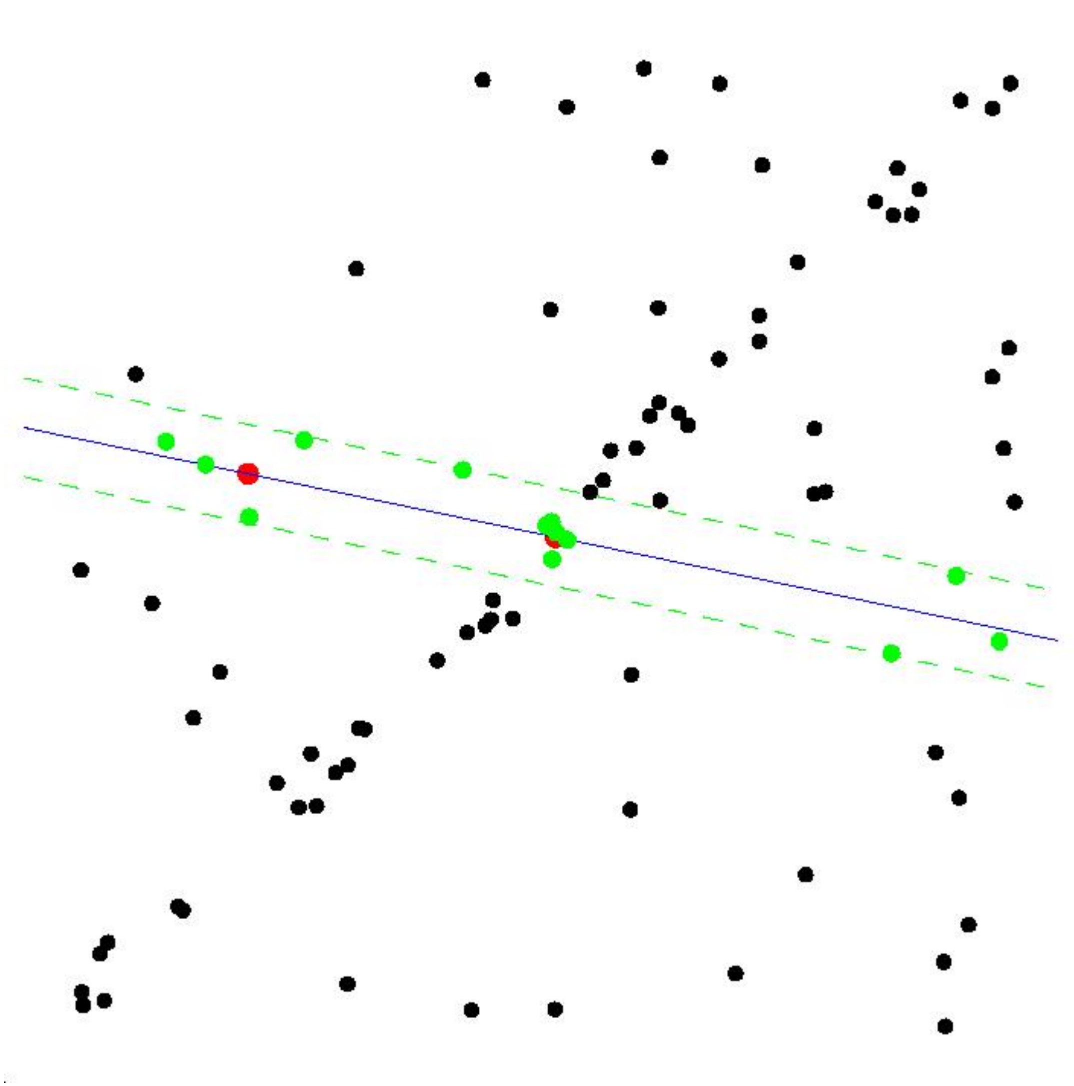
• Select data that supports current hypothesis

RANSAC



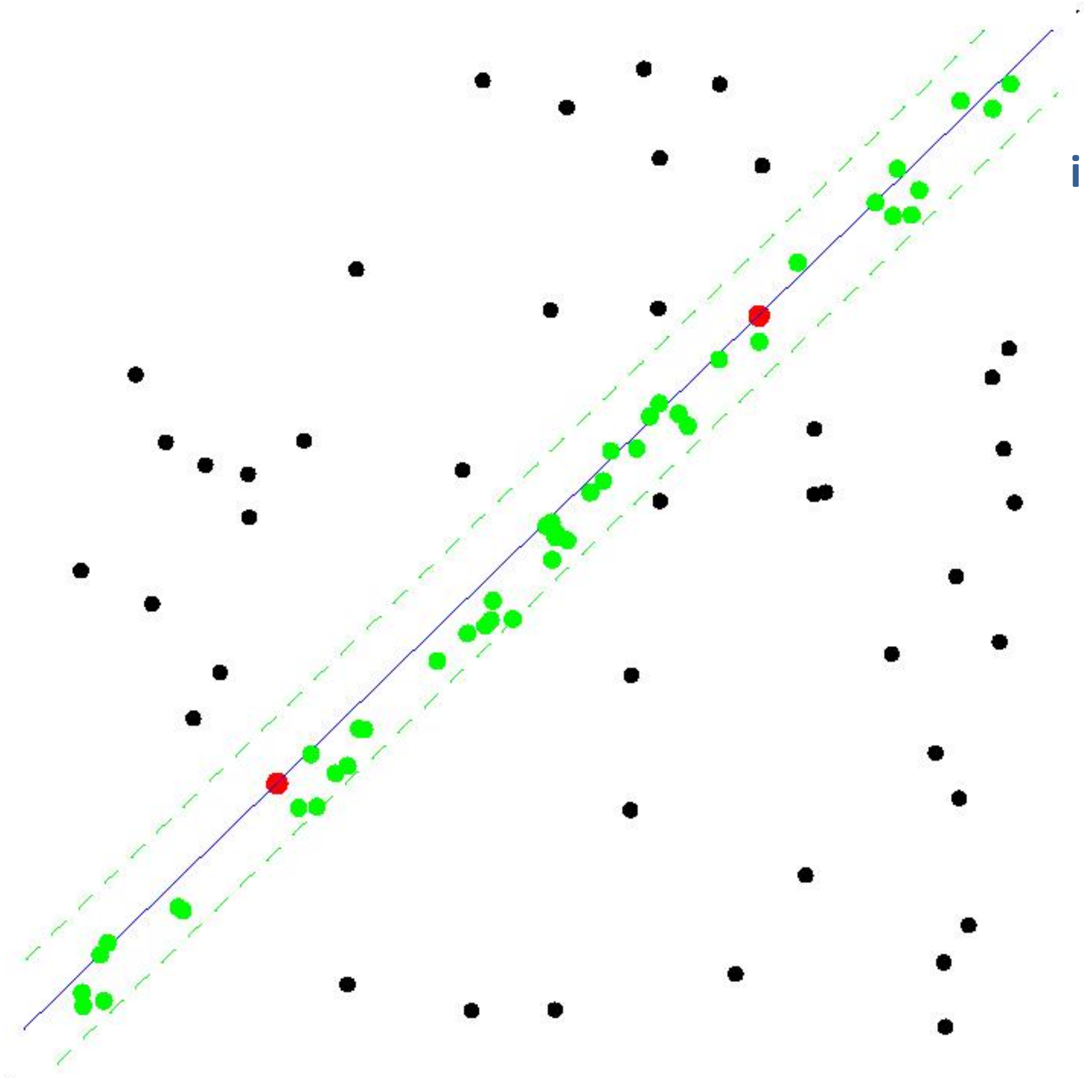
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that supports current hypothesis
- **Repeat sampling**

RANSAC



- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that supports current hypothesis
- **Repeat sampling**

RANSAC



Set with the maximum number of inliers obtained within k iterations

RANSAC

How many iterations does RANSAC need?

- Ideally: check all possible combinations of **2** points in a dataset of **N** points.
- Number all pairwise combinations: **N(N-1)/2**

  ⇨ computationally unfeasible if **N** is too large.

  example: 1000 edge points ⇨ 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?
  Checking a **subset** of combinations is enough 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?

- $\boldsymbol{w}$:= number of inliers/N
  $\boldsymbol{N}$:= total number of data points
  ⇨ $\boldsymbol{w}$: fraction of inliers in the dataset ⇨ $\boldsymbol{w}$ **=** P(selecting an inlier-point out of the dataset)
- Assumption: the 2 points necessary to estimate a line are selected independently
  ⇨ $\boldsymbol{w^2}$ = P(both selected points are inliers)
  ⇨ $\boldsymbol{1-w^2}$ = P(at least one of these two points is an outlier)
- Let $\boldsymbol{k}$:= no. RANSAC iterations executed so far
- ⇨ $(\boldsymbol{1-w^2})^{\boldsymbol{k}}$ = P(RANSAC never selected two points that are both inliers)
- Let $\boldsymbol{p}$:= P(probability of success)
- ⇨ $\boldsymbol{1-p} = (\boldsymbol{1-w^2})^{\boldsymbol{k}}$ and therefore :

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

RANSAC

How many iterations does RANSAC need?

- The number of iterations $\boldsymbol{k}$ is

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

- ⇨ knowing the fraction of inliers $\boldsymbol{w}$, after $\boldsymbol{k}$ RANSAC 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 $\boldsymbol{p}$=99% and we know that $\boldsymbol{w}$=50% ⇨ $\boldsymbol{k}$=16 iterations – these are dramatically fewer than the number of all possible combinations! **As you can see, the number of points does not influence the estimated number of iterations, only 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

RANSAC

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 to 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 ***k*** reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

RANSAC applied to general model fitting

1. Initial: let *A* be a set of *N* points
2. **repeat**
3. Randomly select a sample of ***s*** points from *A*
4. **Fit a model** from the ***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 ***k*** reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

In order to implement RANSAC for Structure from Motion, we need **three 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** that measures how well a point fits the model?

Answers

1. **What's the model** in SfM?
    1. Possible models are:
        1. **R, T**
        2. **E** (i.e., essential Matrix, for calibrated cameras) or **F** (Fundamental matrix, for uncalibrated cameras)

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
    2. However, if we use the *8-point algorithm*, then, **8** is the minimum

3. How do we compute the **error**, i.e., the **distance** of a point from the model?
    1. If we use **E** for the model, then we can use the epipolar constraint $p_2{}^T E p_1 = 0$ to measure how well a correspondence pair (p_1, p_2) verifies the model E. For instance, we can use a threshold $\boldsymbol{th}$ and count as inliers all correspondence pairs that satisfy $|p_2{}^T E p_1| < th$
    2. **In the next three slides, we give an overview of four different popular error measures:**
        1. **Algebraic error**
        2. **Directional error**
        3. **Epipolar-Line distance**
        4. **Reprojection error**

1. Algebraic Error

$$err = ({p^i_2}^T \boldsymbol{E}\, p^i_1)^2$$

Using the definition of dot product, it can be observed that

$$\boldsymbol{p}^T{}_2 \cdot \boldsymbol{E}\boldsymbol{p}_1 = \|\boldsymbol{p}_2\|\|\boldsymbol{E}\boldsymbol{p}_1\|\cos(\theta)$$

which is zero when, $\boldsymbol{p}^T{}_1$, $\boldsymbol{p}_2$, and $\boldsymbol{T}$ are coplanar.

2. Directional Error

Angular distance to the Epipolar plane

$$\mathrm{err} = \cos(\theta)$$

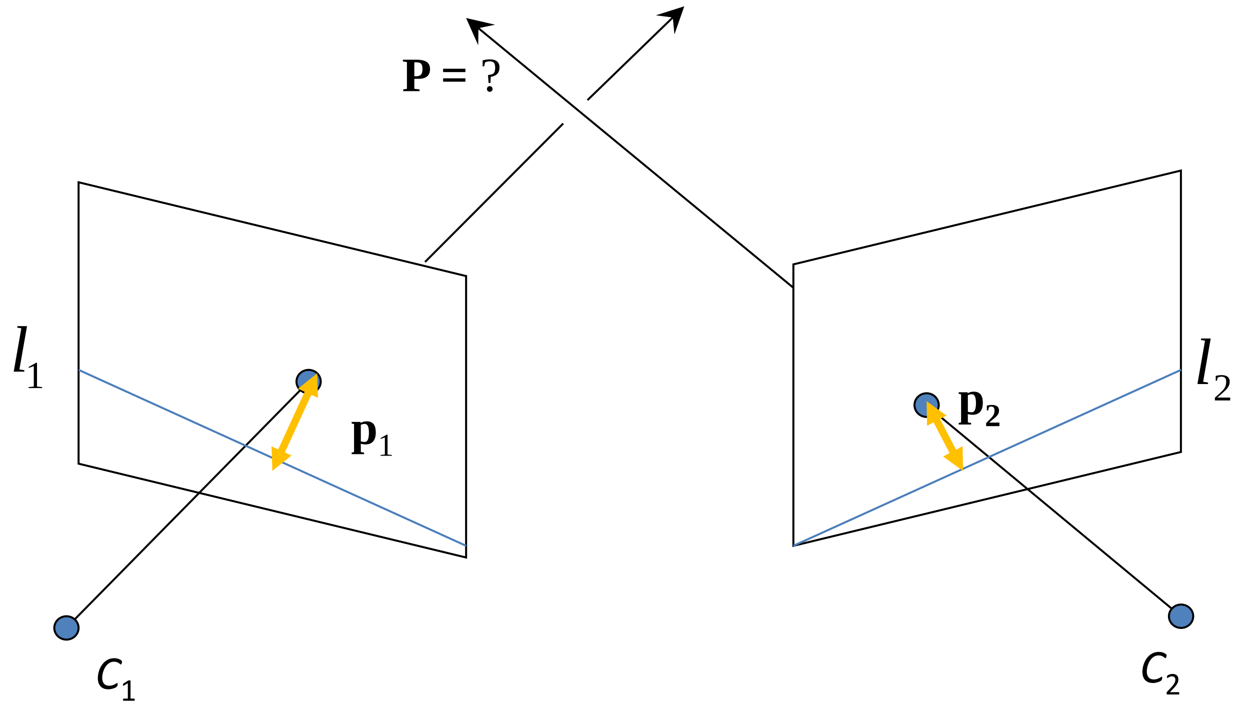
From the previous slide, we obtain:

$$\mathrm{err} = \cos(\theta) = \frac{\boldsymbol{p}^{T}{}_{1} \cdot \boldsymbol{E}\boldsymbol{p}_{2}}{\|\boldsymbol{p}^{T}{}_{1}\|\|\boldsymbol{E}\boldsymbol{p}_{2}\|}$$

3. Epipolar-Line Distance

Minimize sum of squared *epipolar* **distances**

$$err = (\mathrm{d}(p_1, l_1))^2 + (\mathrm{d}(p_2, l_2))^2$$

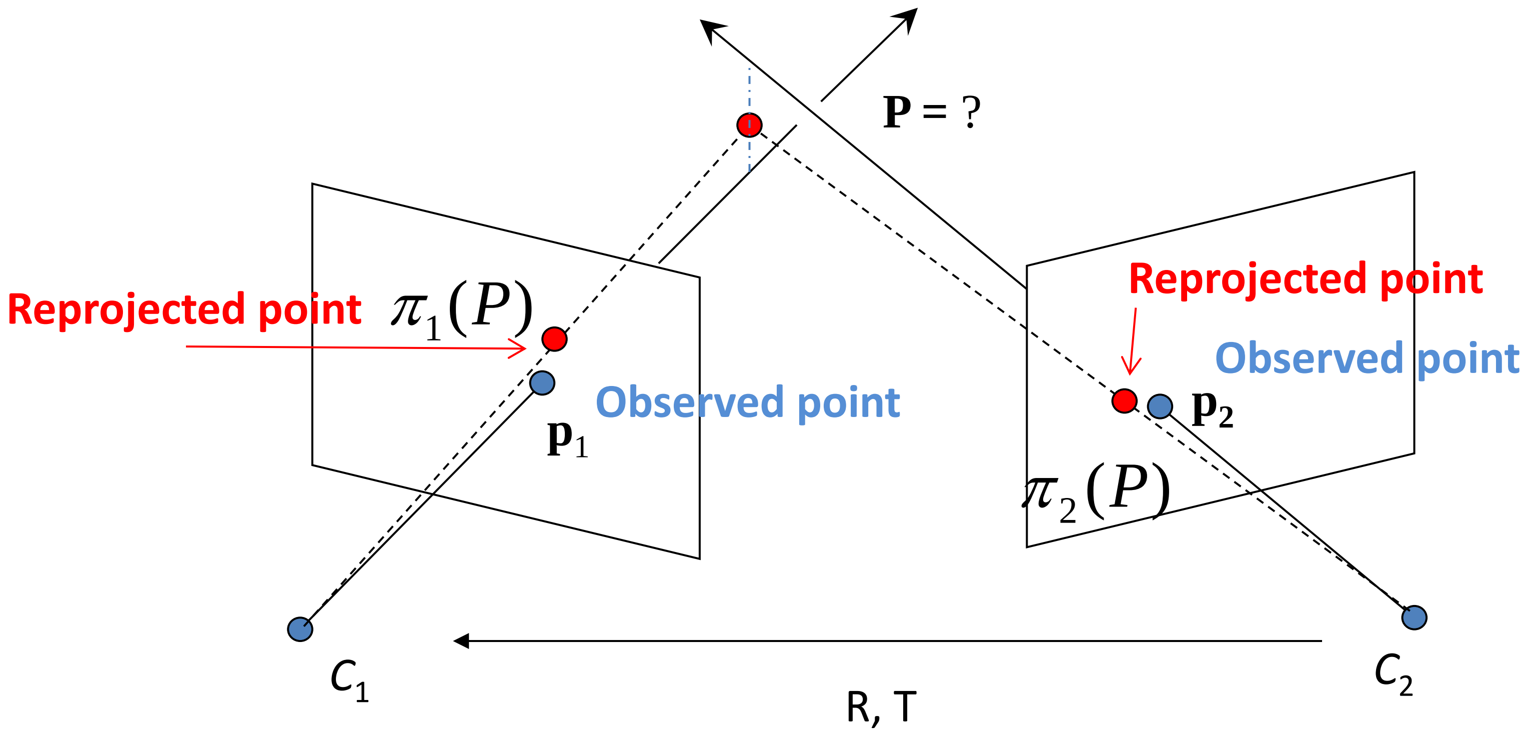


4. Reprojection Error

- Definition: is the sum of the squared distances between the observed image points and the reprojection of the triangulated 3D point

$$err = d^2(p_1, \pi_1(P)) + d^2(p_2, \pi_2(P))$$

$$d(p_1, M_1 P) = \|p_1 - \pi_1(P)\|$$



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

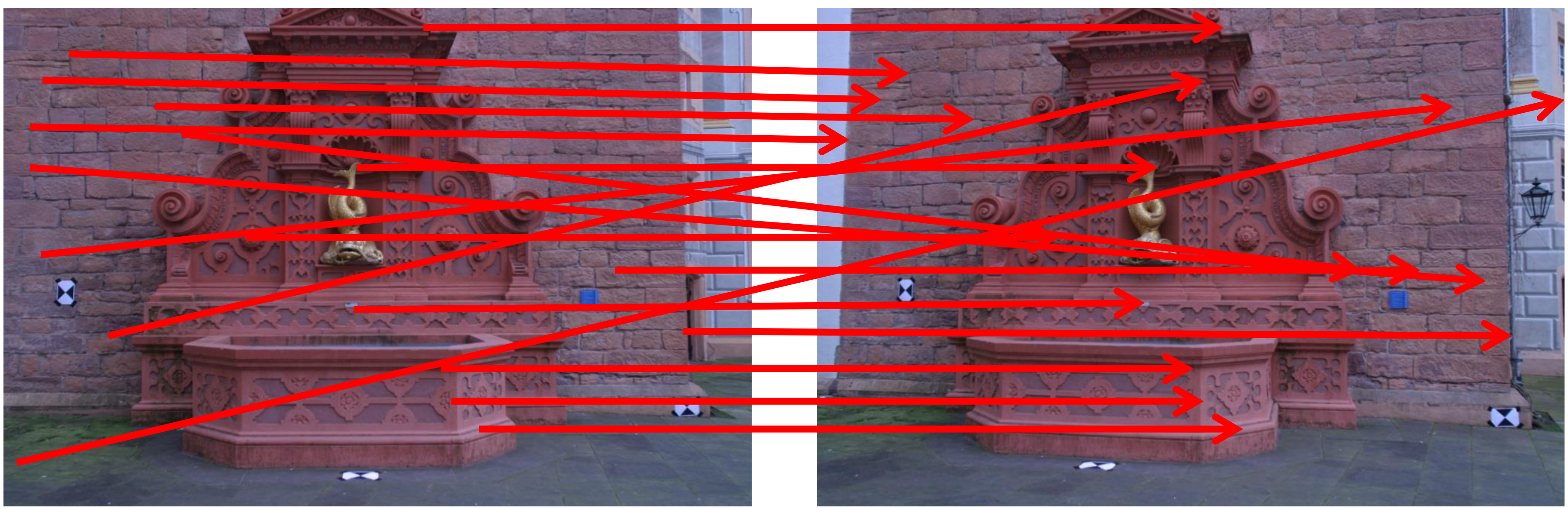


Image 1   Image 2

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 in the first image and use arrows to denote the motion vectors of the features

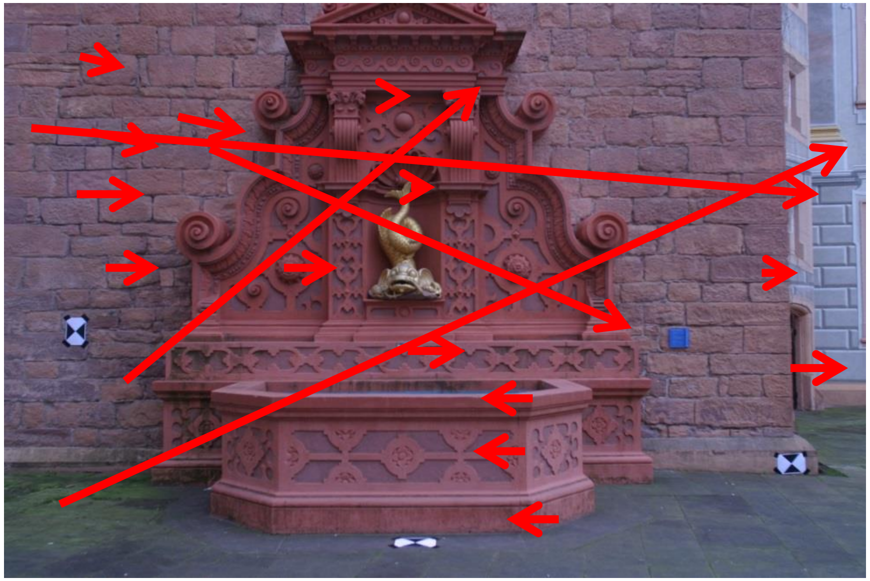


Image 1

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 in the first image and use arrows to denote the motion vectors of the features

1. Randomly select 8 point correspondences

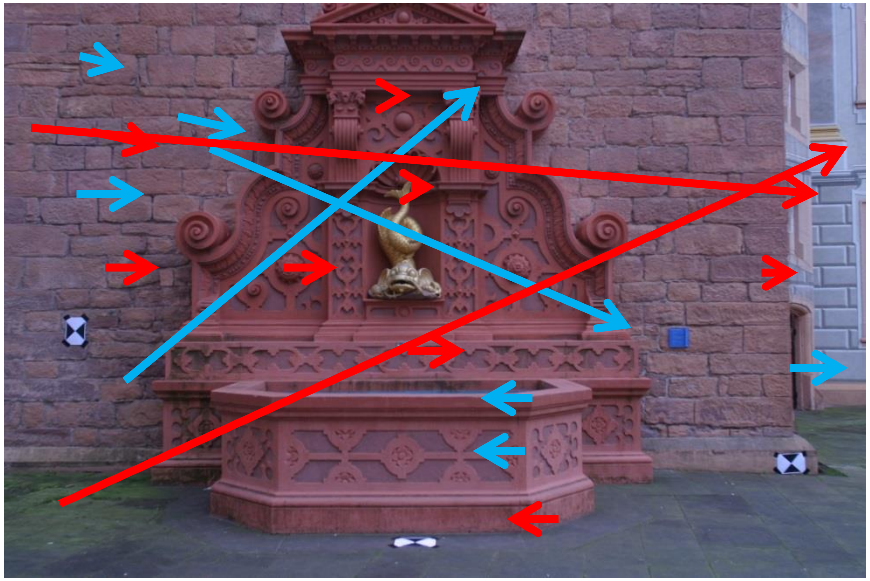


Image 1

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 in the first image and use arrows to denote the motion vectors of the features

1. Randomly select 8 point correspondences
2. Fit the model to all other points and count the inliers

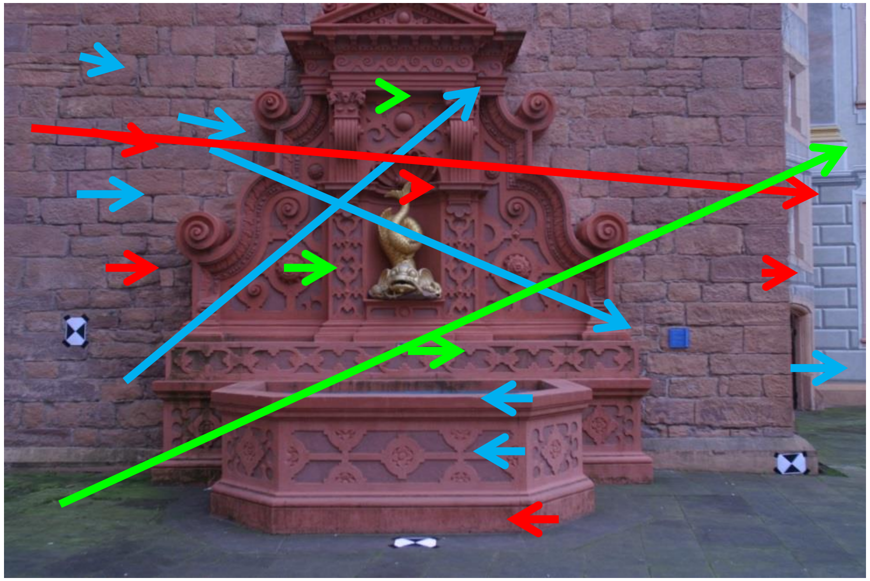


Image 1

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 in the first image and use arrows to denote the motion vectors of the features

1. Randomly select 8 point correspondences
2. Fit the model to all other points and count the inliers
3. Repeat from 1

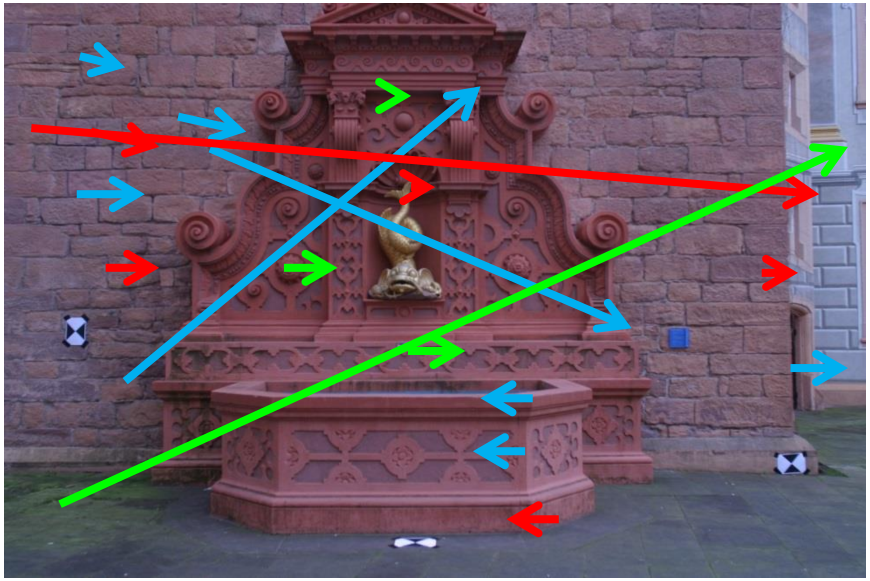


Image 1

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 in the first image and use arrows to denote the motion vectors of the features

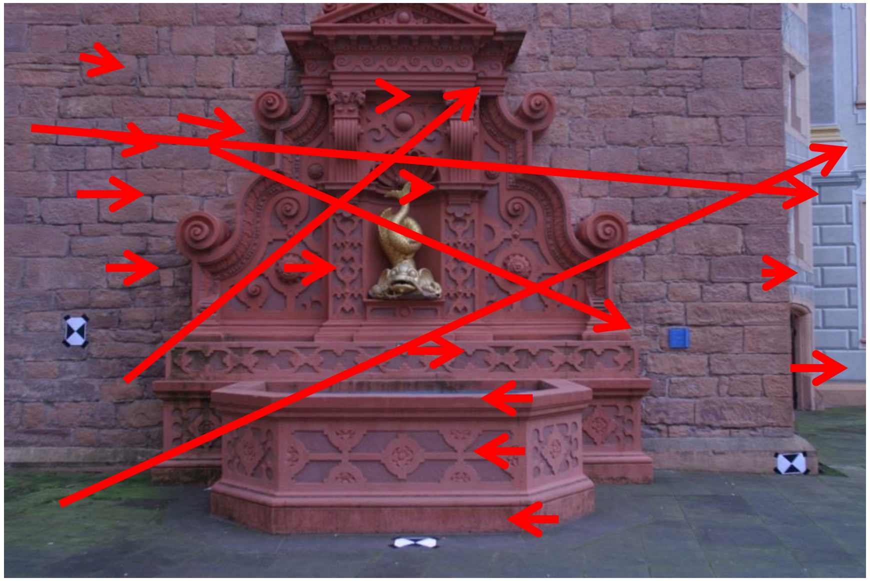


Image 1

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 in the first image and use arrows to denote the motion vectors of the features

1. Randomly select 8 point correspondences

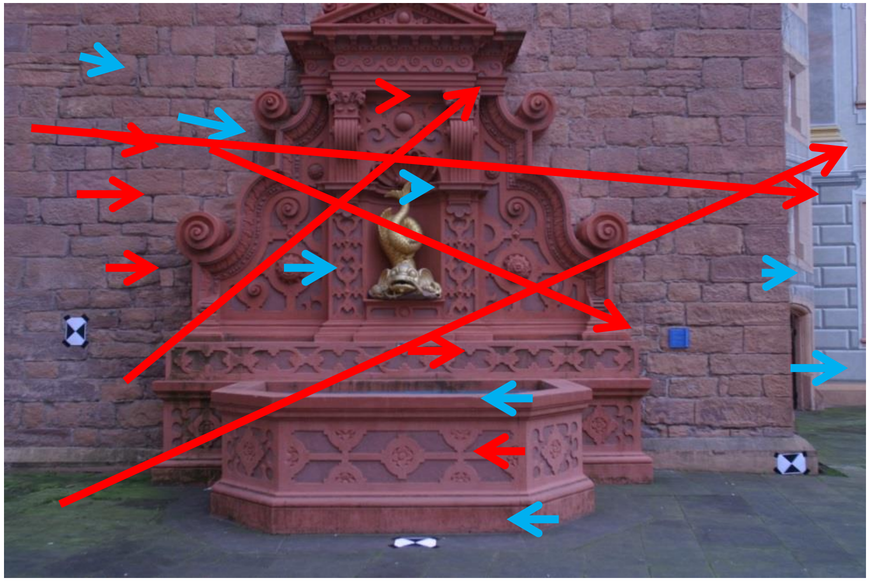


Image 1

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 in the first image and use arrows to denote the motion vectors of the features

1. Randomly select 8 point correspondences
2. Fit the model to all other points and count the inliers

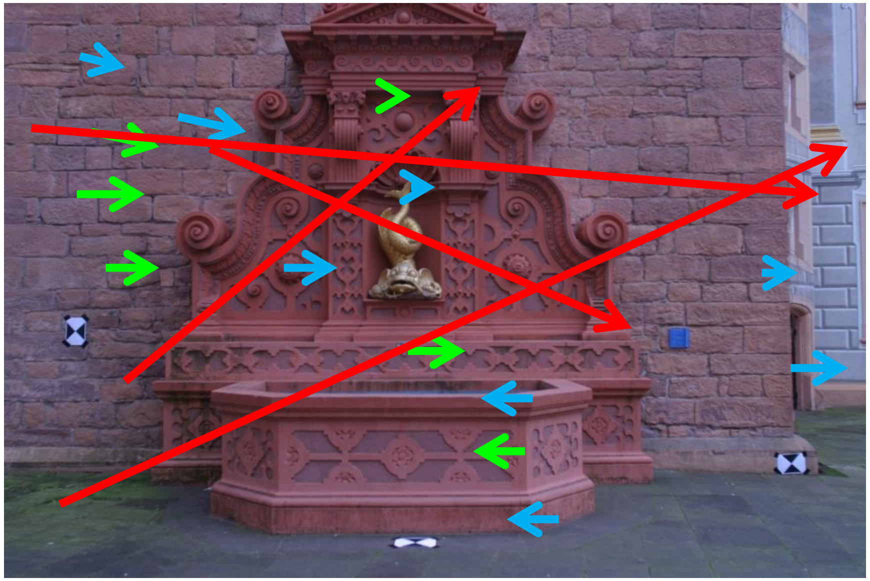


Image 1

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 in the first image and use arrows to denote the motion vectors of the features

1. Randomly select 8 point correspondences
2. Fit the model to all other points and count the inliers
3. Repeat from 1 for $\boldsymbol{k}$ times

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

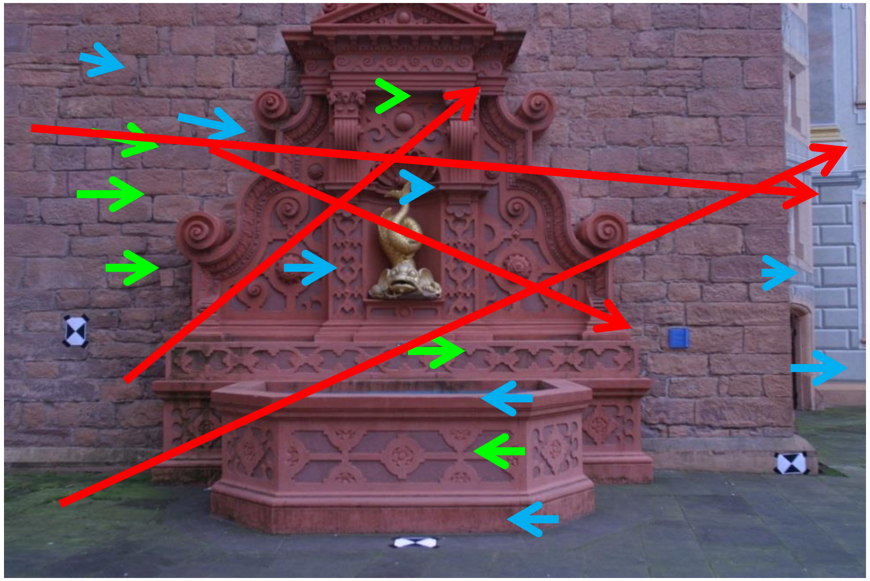


Image 1

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

$\boldsymbol{k}$ is exponential in the number of points $\boldsymbol{s}$ necessary to estimate the model:

- **8-point RANSAC**
  - Assuming
    - $\boldsymbol{p}$ = 99%,
    - $\boldsymbol{\varepsilon}$ = 50% (fraction of outliers)
    - $\boldsymbol{s}$ = 8 points (8-point algorithm)

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

- **5-point RANSAC**
  - Assuming
    - $\boldsymbol{p}$ = 99%,
    - $\boldsymbol{\varepsilon}$ = 50% (fraction of outliers)
    - $\boldsymbol{s}$ = 5 points (5-point algorithm of David Nister (2004))

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

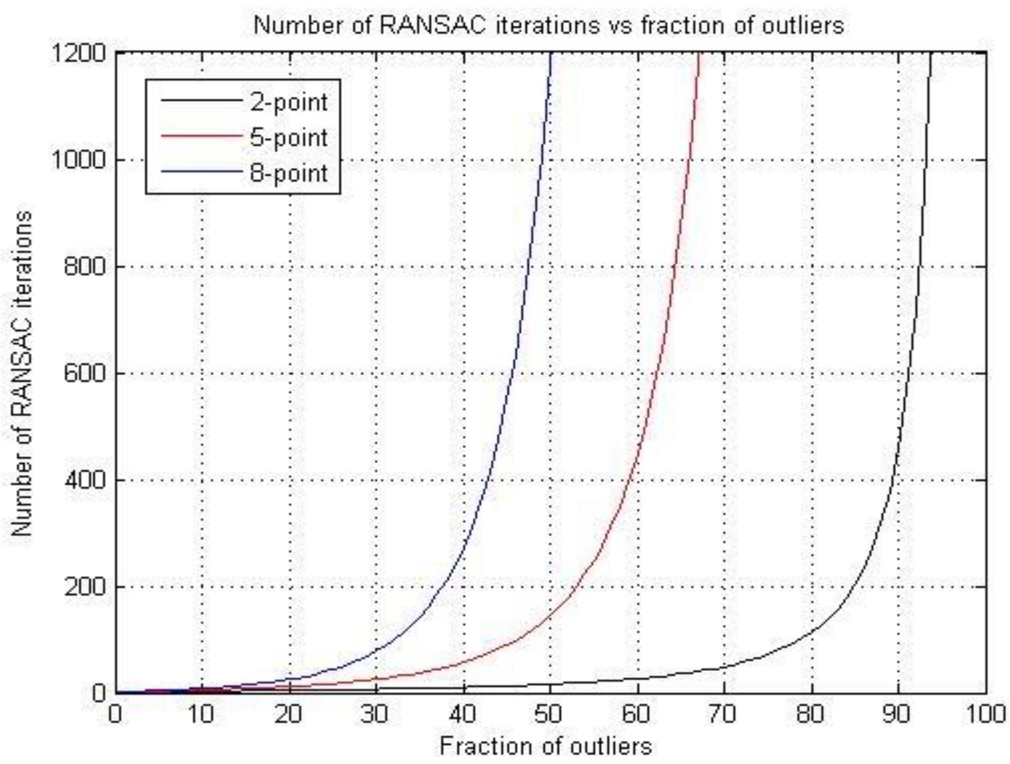
- **2-point RANSAC (e.g., line fitting)**
  - Assuming
    - $\boldsymbol{p}$ = 99%,
    - $\boldsymbol{\varepsilon}$ = 50% (fraction of outliers)
    - $\boldsymbol{s}$ = 2 points

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

Number of points (s):	8	7	6	5	4	2	1
Number of iterations (N):	1177	587	292	145	71	16	7

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

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



RANSAC iterations

- As observed, $\boldsymbol{k}$ is exponential in 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
- The first efficient solution to the minimal-case solution (5-point algorithm) took almost a century (Kruppa 1913 → Nister, 2004)
- The 5-point RANSAC only requires 145 iterations; however:
  - The **5-point algorithm** can return **up to 10 solutions of E**
  - The **8-point algorithm** only returns a **unique solution of E**

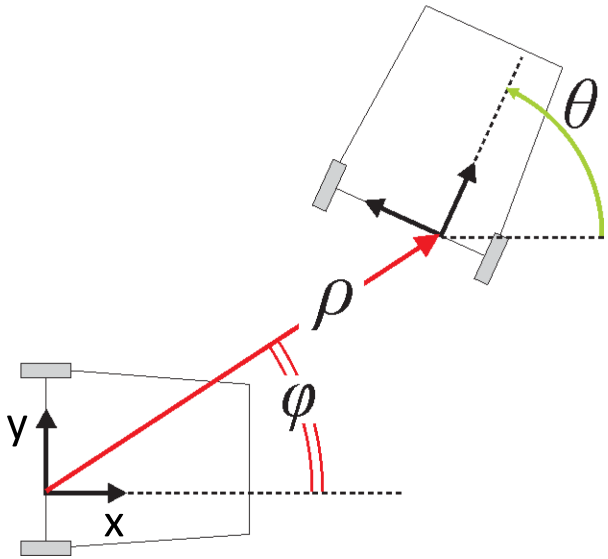
Can we use less than 5 points?

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

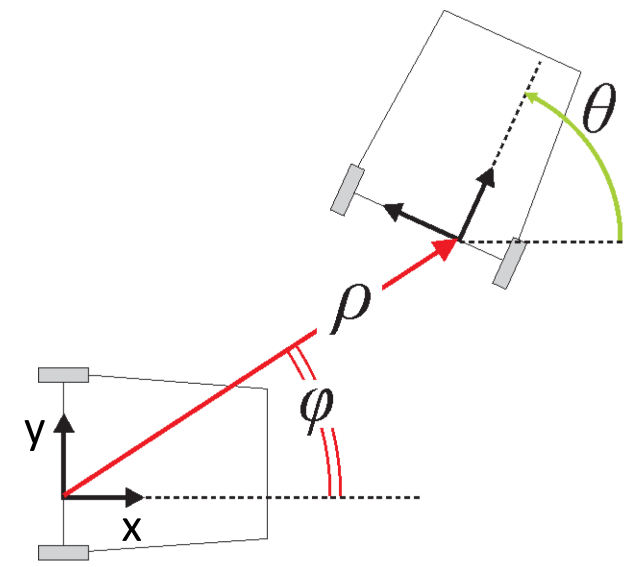
$$\mathrm{E} = [T_\times]R \qquad \textit{Essential matrix}$$

$$p_2^T \; E \; p_1 = 0 \qquad \textit{Epipolar constraint}$$

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

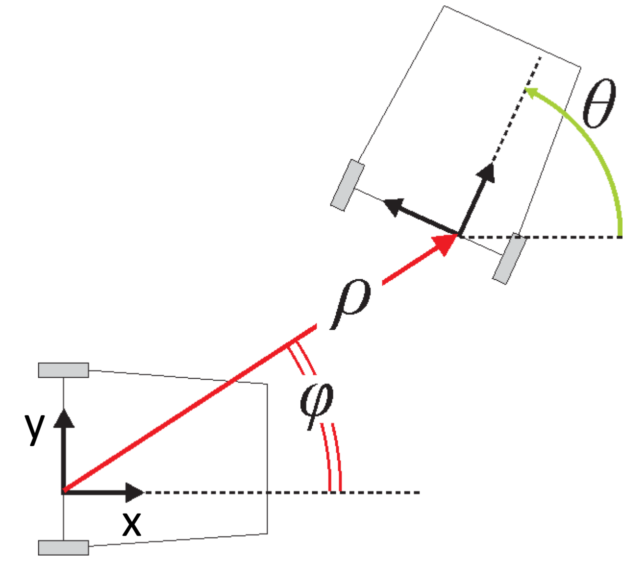
$$[T_\times] = \begin{bmatrix} 0 & 0 & \rho\sin\varphi \\ 0 & 0 & -\rho\cos\varphi \\ -\rho\sin\varphi & \rho\cos\varphi & 0 \end{bmatrix}$$

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

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

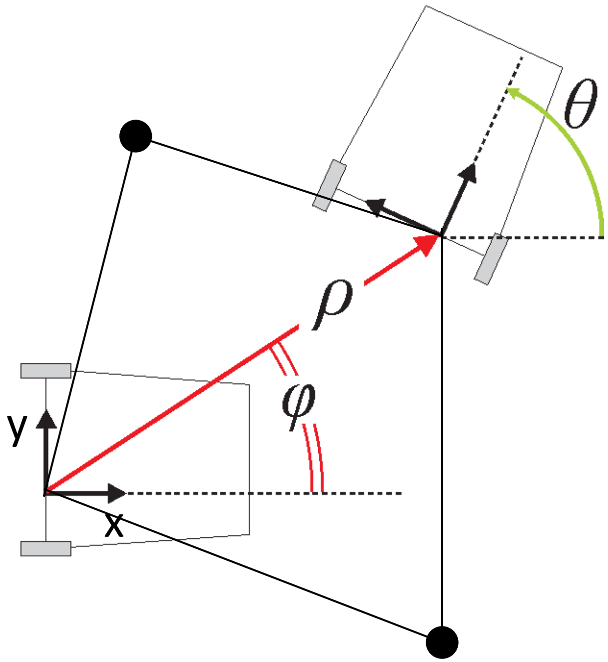
$$[T_\times] = \begin{bmatrix} 0 & 0 & \rho\sin\varphi \\ 0 & 0 & -\rho\cos\varphi \\ -\rho\sin\varphi & \rho\cos\varphi & 0 \end{bmatrix}$$

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

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 E has 2DoF; thus, 2 correspondences are sufficient to estimate θ and ϕ

[“2-Point RANSAC”, Ortin, 2001]

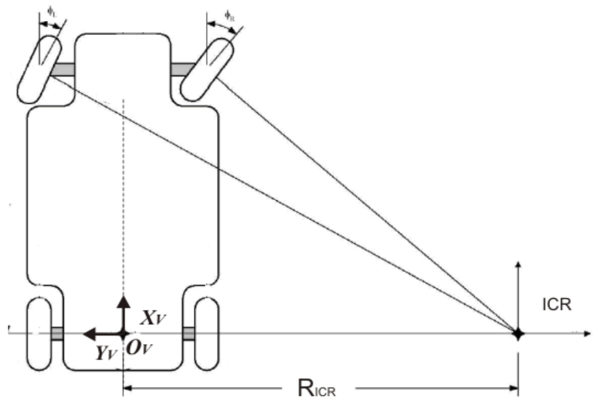
$$E = [T_\times]R = \begin{bmatrix} 0 & 0 & \rho\sin(\varphi) \\ 0 & 0 & -\rho\cos(\varphi) \\ -\rho\sin(\varphi-\theta) & \rho\cos(\varphi-\theta) & 0 \end{bmatrix}$$

Can we use less than 2 point correspondences?

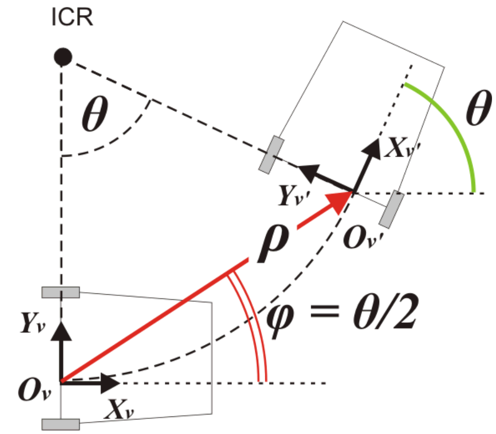
Yes, if we exploit ground, wheeled vehicles with **non-holonomic** *constraints*

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

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



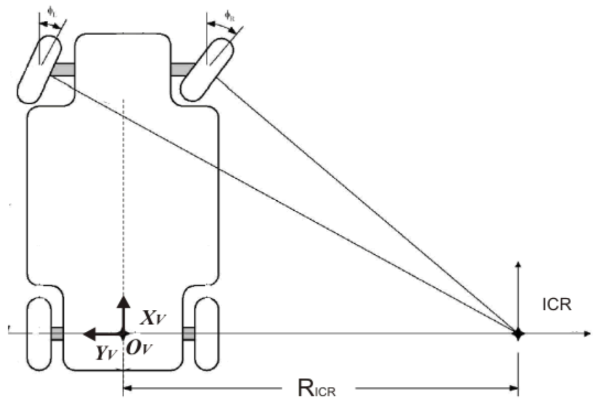
Example of Ackerman steering principle



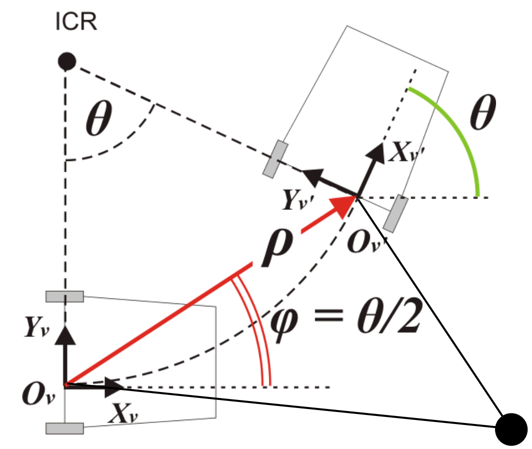
Locally-planar circular motion

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

$\varphi = \theta/2 \Rightarrow$ *only 1 DoF* (θ);

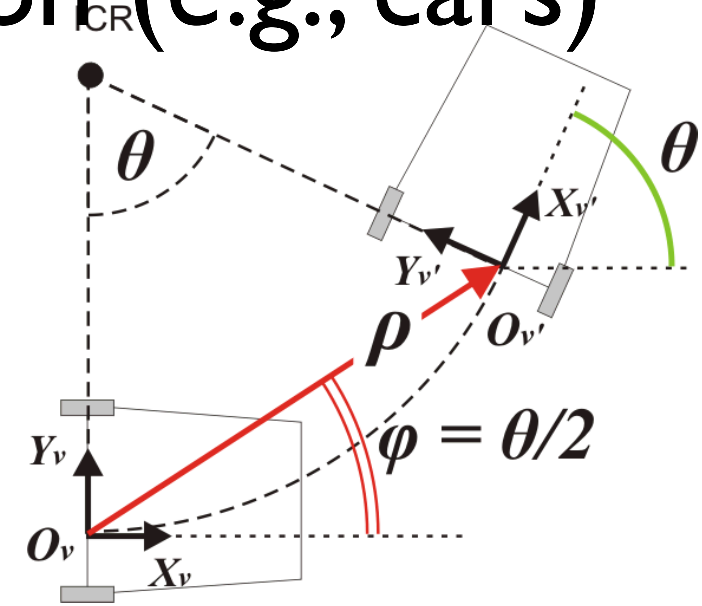
thus, only 1 point correspondence is needed

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

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

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



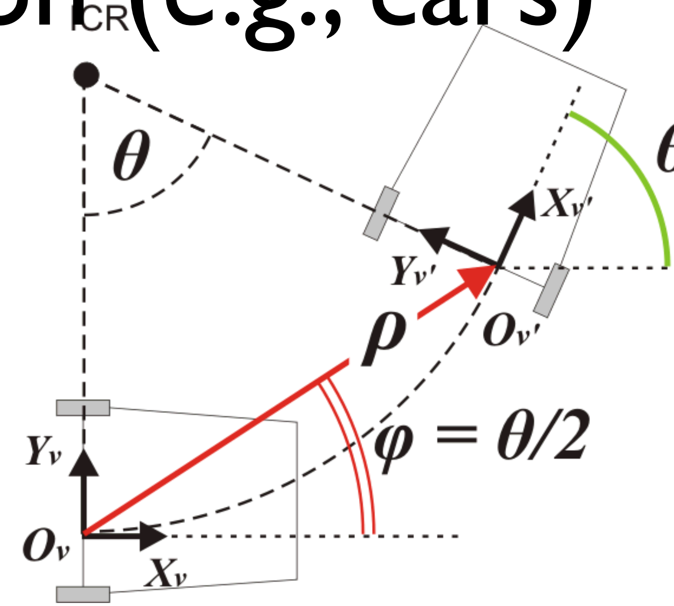
Let's compute the Epipolar Geometry

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

$$p_2^T \; E \; p_1 = 0 \quad \textit{Epipolar constraint}$$

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

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

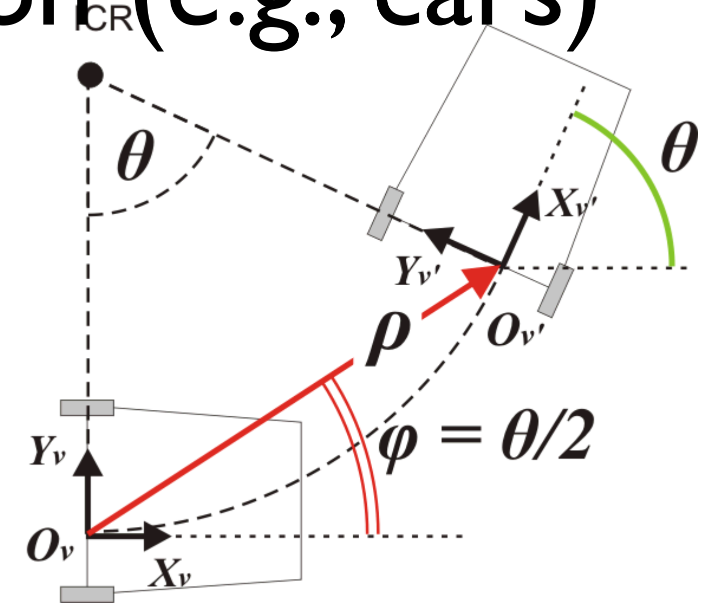


Let's compute the Epipolar Geometry

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

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

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



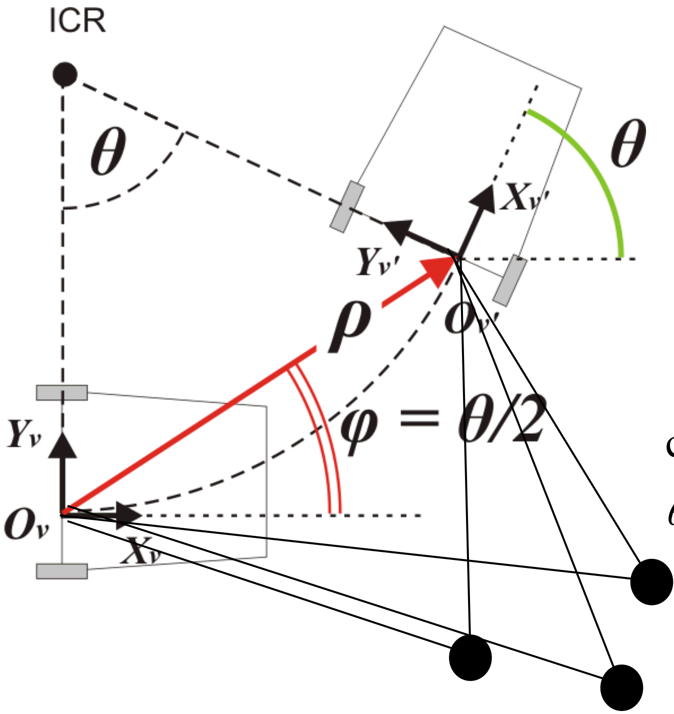
$$E = \rho \begin{bmatrix} 0 & 0 & \sin\frac{\theta}{2} \\ 0 & 0 & \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2} & -\cos\frac{\theta}{2} & 0 \end{bmatrix}$$

Let’s compute the Epipolar Geometry

$$p_2^T \; E \; p_1 = 0 \quad \Rightarrow \quad \sin\left(\frac{\theta}{2}\right)\cdot(u_2 + u_1) + \cos\left(\frac{\theta}{2}\right)\cdot(v_2 - v_1) = 0$$

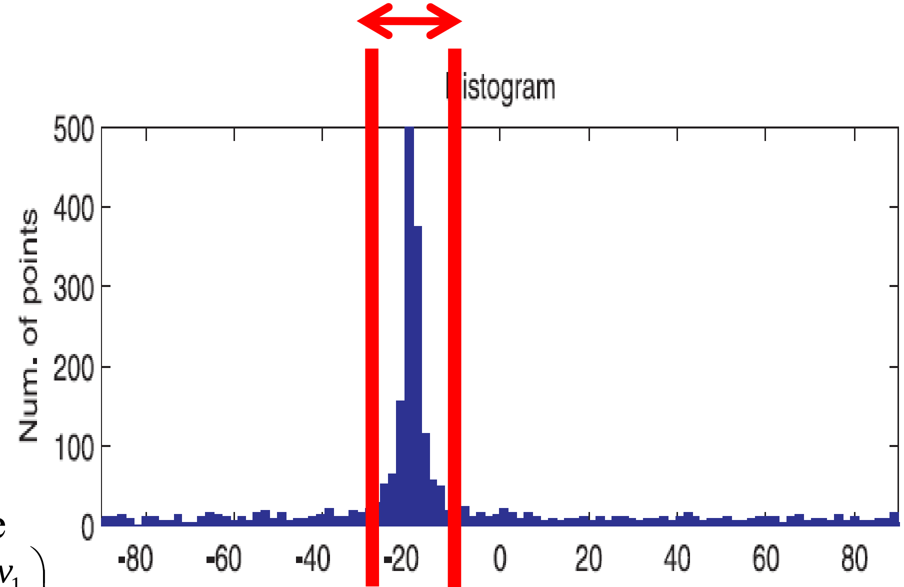
$$\boxed{\theta = -2\tan^{-1}\left(\frac{v_2 - v_1}{u_2 + u_1}\right)}$$

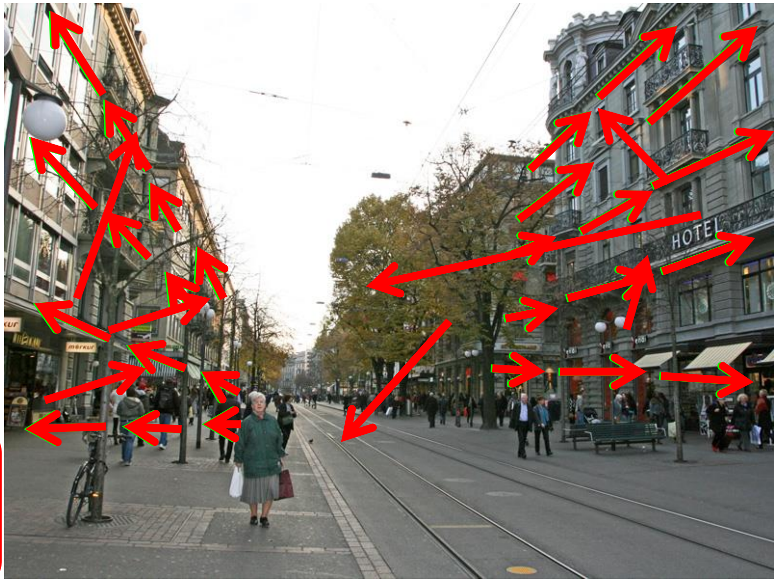
1-Point RANSAC algorithm



Compute θ for every point correspondence

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





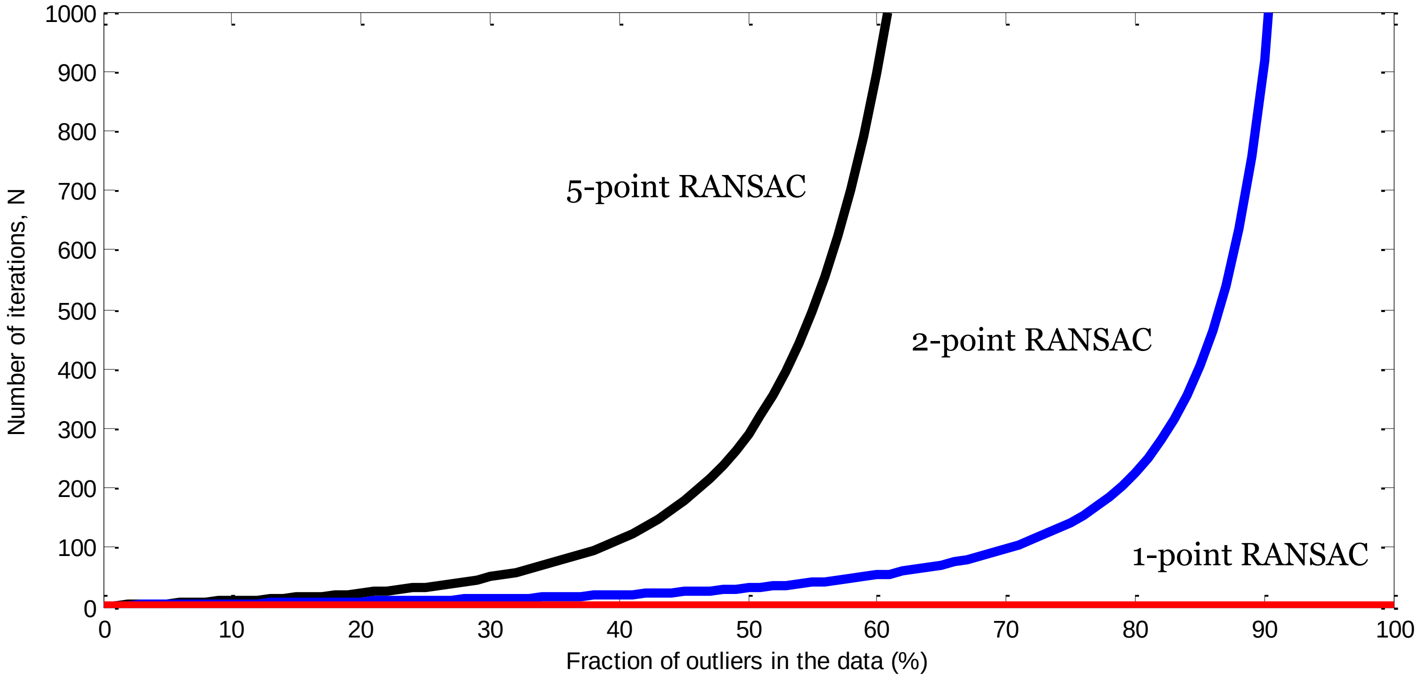
Only 1 iteration!

The most efficient algorithm for removing outliers, up to 1000 Hz!

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

Motion is then estimated from them in 6DOF!

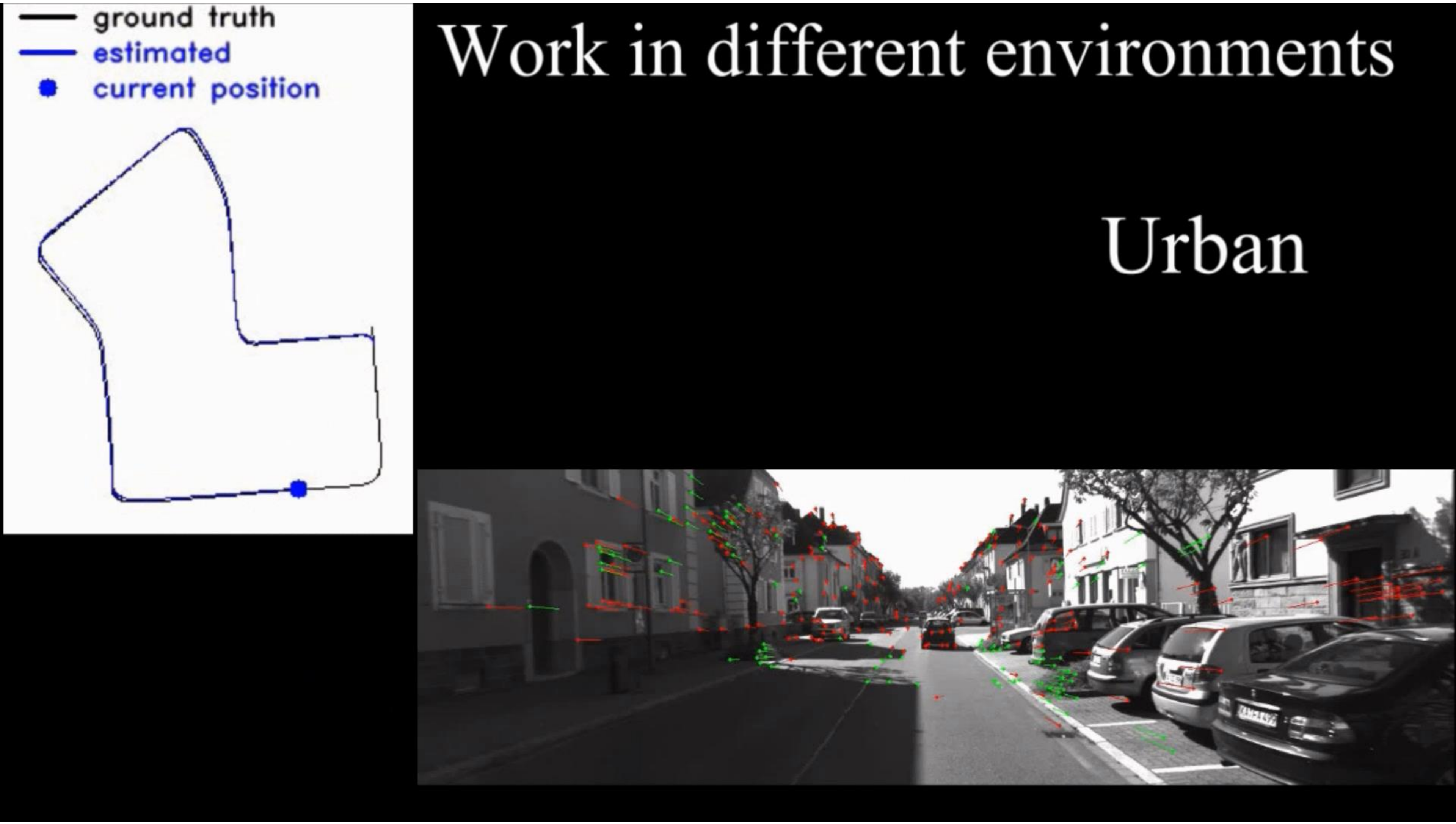
Comparison of RANSAC algorithms



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

	8-Point RANSAC	5-Point RANSAC [Nister'03]	2-Point RANSAC [Ortin'01]	1-Point RANSAC [Scaramuzza, IJCV'10]
Numb. of iterations	**> 1177**	**>145**	**>16**	**=1**

Visual Odometry with 1-Point RANSAC

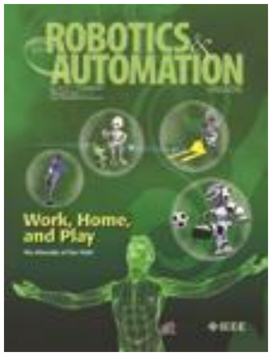


Scaramuzza, **1-Point-RANSAC Structure from Motion for Vehicle-Mounted Cameras by Exploiting Non-holonomic Constraints**, International Journal of Computer Vision, 2011

Today’s outline

- Review of last lecture
- RANSAC for robust Structure from Motion
- Visual Odometry

References: Tutorial on Visual Odometry



- Scaramuzza, D., Fraundorfer, F., Visual Odometry: Part I - The First 30 Years and Fundamentals, IEEE Robotics and Automation Magazine, Volume 18, issue 4, 2011. http://rpg.ifi.uzh.ch/docs/VO_Part_I_Scaramuzza.pdf

- Fraundorfer, F., Scaramuzza, D., Visual Odometry: Part II - Matching, Robustness, and Applications, IEEE Robotics and Automation Magazine, Volume 19, issue 1, 2012. http://rpg.ifi.uzh.ch/docs/VO_Part_II_Scaramuzza.pdf

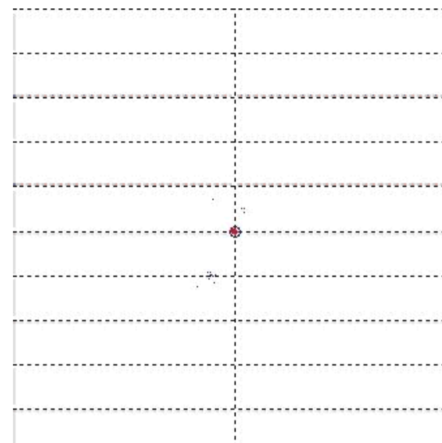
Visual Odometry (VO)

VO is the process of incrementally estimating the pose of the vehicle by examining the changes that motion induces on the images of its onboard cameras

input

Image sequence (or video stream)
from one or more cameras attached to a moving vehicle

output



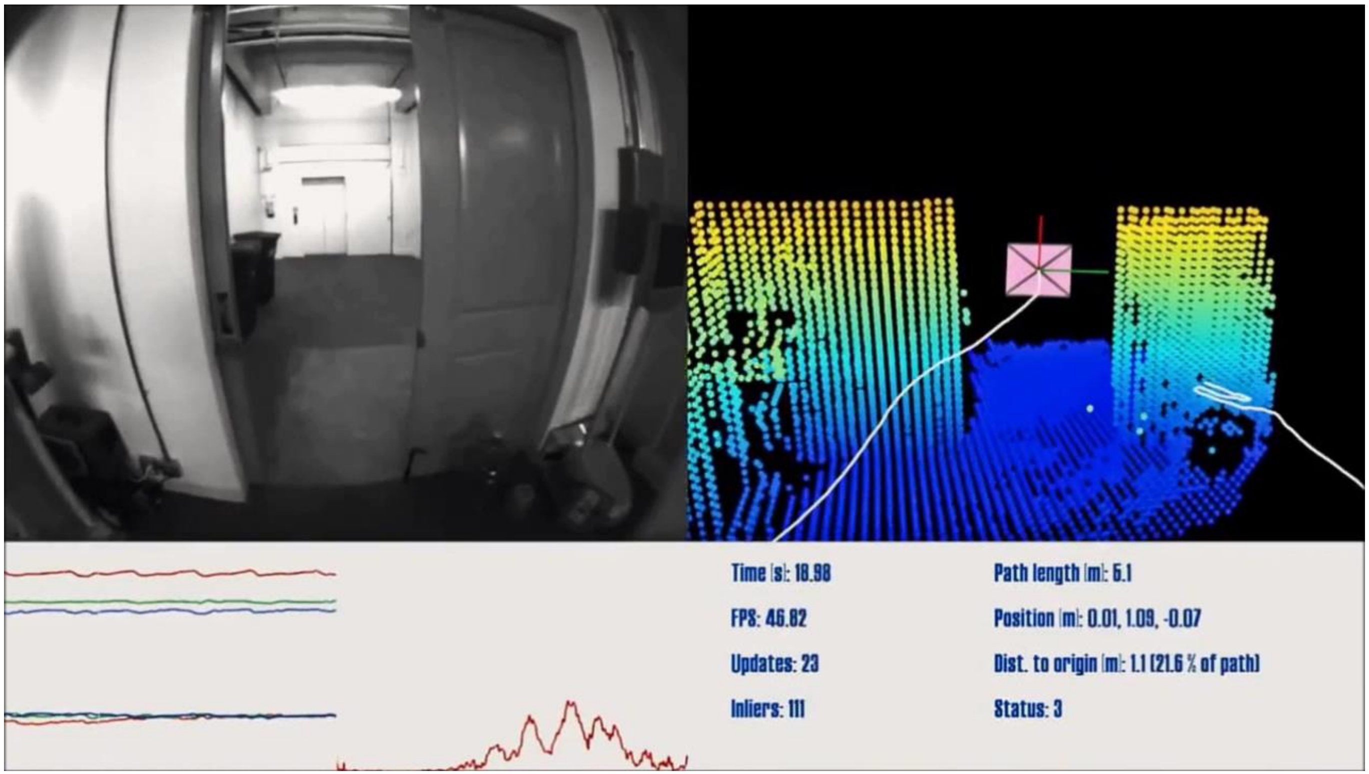
$$R_0, R_1, \ldots, R_i$$

$$t_0, t_1, \ldots, t_i$$

Camera trajectory (3D structure is a plus):

Example 1: VO for Phones

Application to Augmented Reality for smartphones



Project Tango

Example 2: VO for Flying Robots

[Scaramuzza et al., Vision-Controlled Micro Flying Robots: from System Design to Autonomous Navigation and Mapping in GPS-denied Environments, IEEE RAM, September, 2014

Example 3: VO for Mouse Scanners

World-first mouse scanner

Currently distributed by LG: *SmartScan LG LSM100*

A Brief history of VO

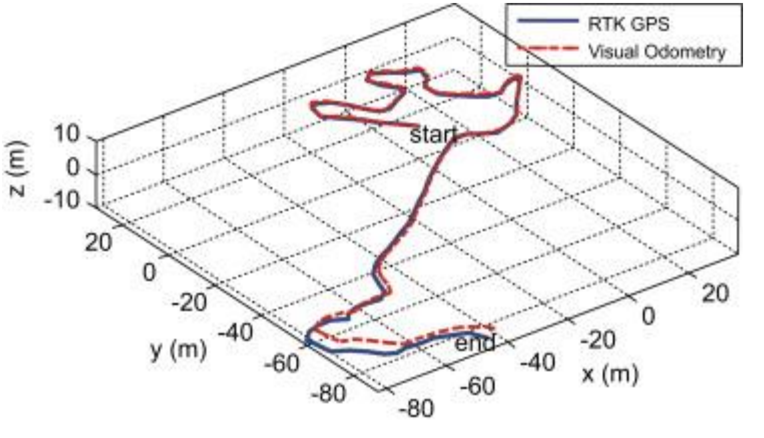
- **1980**: First known stereo VO real-time implementation on a robot by Moraveck PhD thesis (NASA/JPL) for Mars rovers using a sliding camera.
- **1980 to 2000**: The VO research was dominated by NASA/JPL in preparation of 2004 Mars mission (see papers from Matthies, Olson, etc. From JPL)
- **2004**: VO used on a robot on another planet: Mars rovers Spirit and Opportunity
- **2004**: VO was revived in the academic environment by Nister «Visual Odometry» paper.
  The term VO became popular.

Why VO ?

- Contrary to wheel odometry, VO is not affected by wheel slip on uneven terrain or other adverse conditions.

- More accurate trajectory estimates compared to wheel odometry
(relative position error 0.1% – 2%)

- VO can be used as a complement to
  - wheel odometry
  - GPS
  - inertial measurement units (IMUs)
  - laser odometry



- In GPS-denied environments, such as underwater and aerial, VO has utmost importance

VO work flow

- VO computes the camera path incrementally (pose after pose)

Image sequence

↓

Feature detection

↓

Feature matching (tracking)

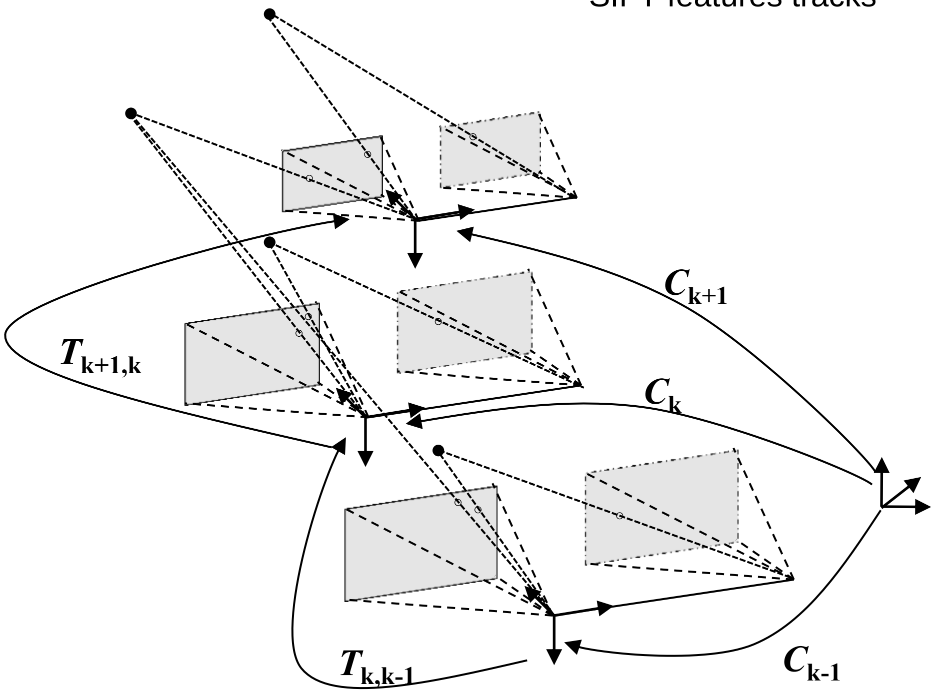
↓

Motion estimation

↓

Local optimization

SIFT features tracks



VO or Structure from Motion (SFM) ?

SFM is more general than VO and tackles the problem of 3D reconstruction of both the structure and camera poses from **unordered image sets**

Reconstruction from 3 million images from Flickr.com
Cluster of 250 computers, 24 hours of computation!
Paper: “Building Rome in a Day”, ICCV’09

VO or Structure from Motion (SFM) ?

- VO is a particular case of SFM

- VO focuses on estimating the 3D motion of the camera sequentially (as a new frame arrives) and in real time.

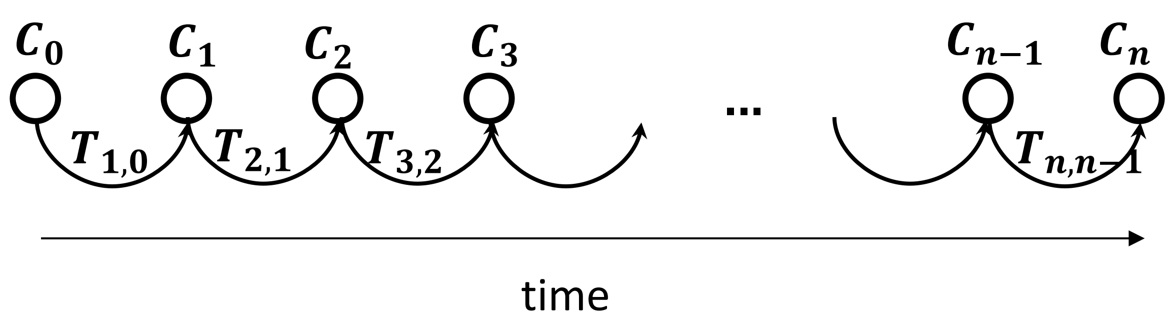
- Terminology: sometimes SFM is used as a synonym of VO

Motion Estimation

- Motion estimation is the core computation step performed for every image in a VO system
- It computes the camera motion T_k between the previous and the current image:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

- By concatenation of all these single movements, the full trajectory of the camera can be recovered , i.e.: $C_k = T_{k,k-1} C_{k-1}$

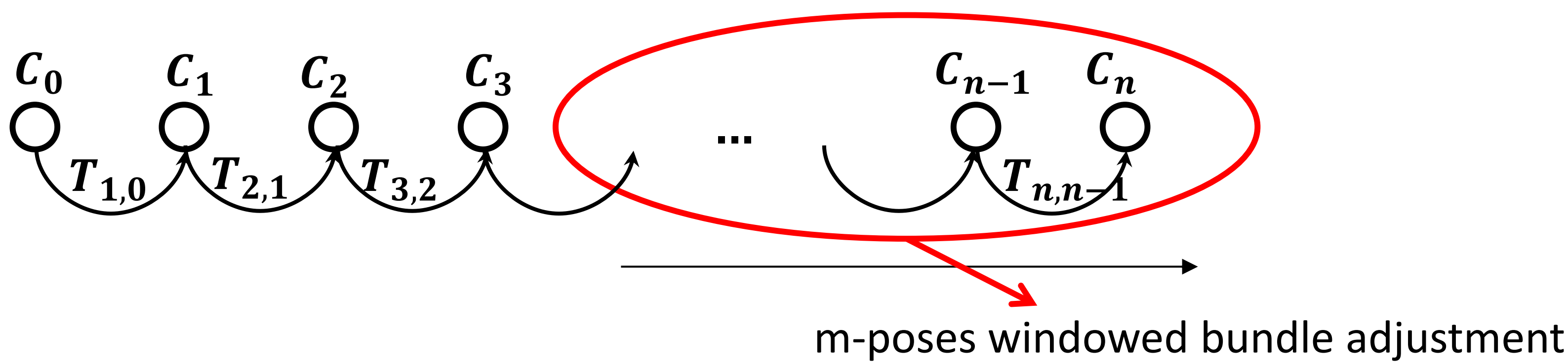


Motion Estimation

- Motion estimation is the core computation step performed for every image in a VO system
- It computes the camera motion T_k between the previous and the current image:

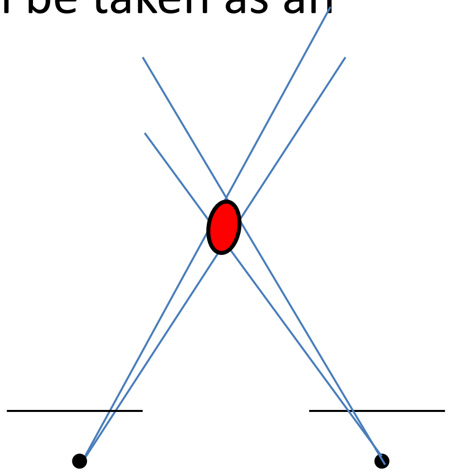
$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

- By concatenation of all these single movements, the full trajectory of the camera can be recovered , i.e.: $C_k = T_{k,k-1} C_{k-1}$
- An iterative refinement over the last m poses can be performed to get a more accurate estimate of the local trajectory



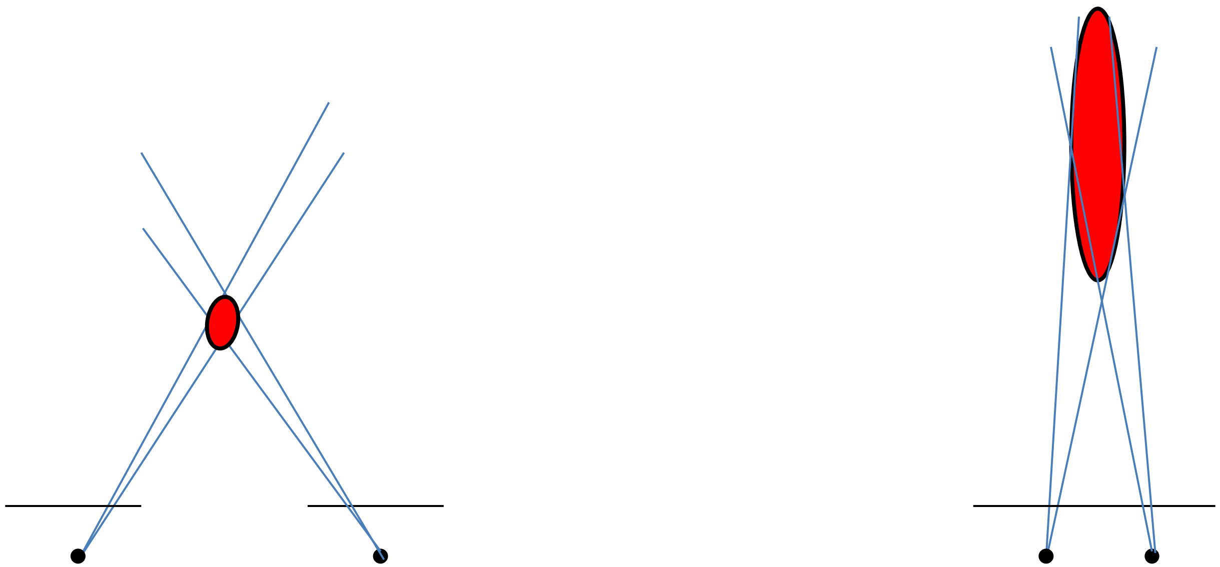
Triangulation and Keyframe Selection

- Triangulated 3D points are determined by intersecting backprojected rays from 2D image correspondences of at least two image frames
- In reality, they never intersect due to
  - image noise,
  - camera model and calibration errors,
  - and feature matching uncertainty
- The point at minimal distance from all intersecting rays can be taken as an estimate of the 3D point position



Triangulation and Keyframe Selection

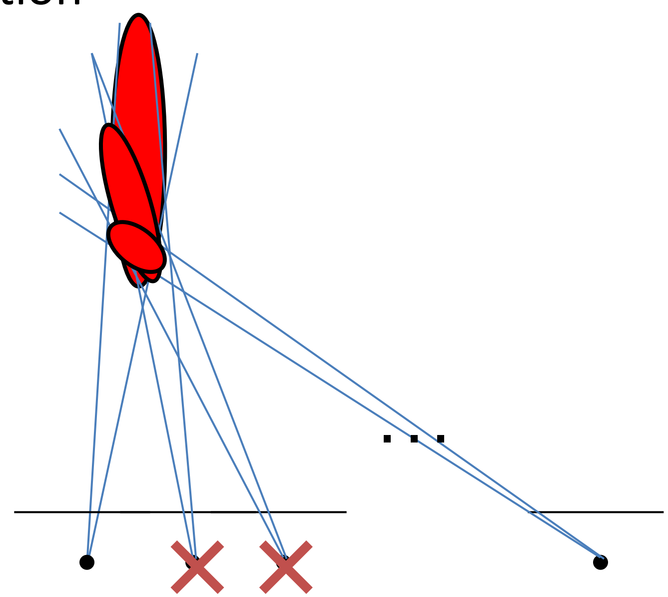
- When frames are taken at nearby positions compared to the scene distance, 3D points will exibit large uncertainty



Triangulation and Keyframe Selection

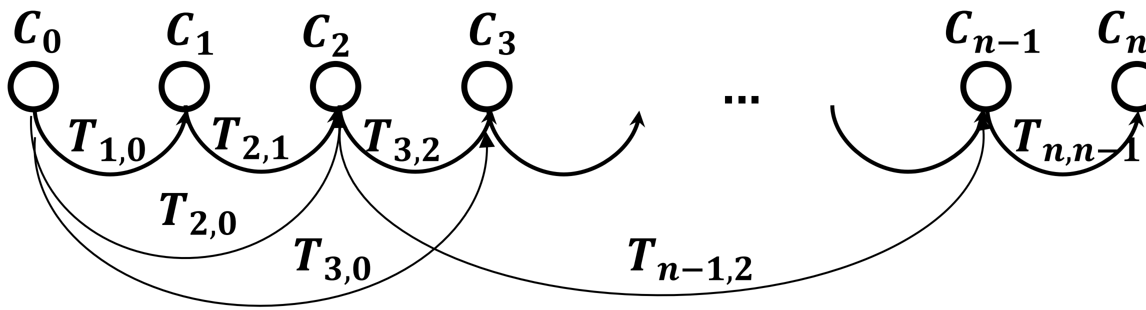
- One way to avoid this consists of skipping frames until the average uncertainty of the 3D points decreases below a certain threshold. The uncertainty can be compute by intersecting back-projected cones. The selected frames are called *keyframes*

- Keyframe selection is a very important step in VO and should always be done before updating the motion



Camera-Pose Optimization

- So far we assumed that the transformations are between consecutive frames

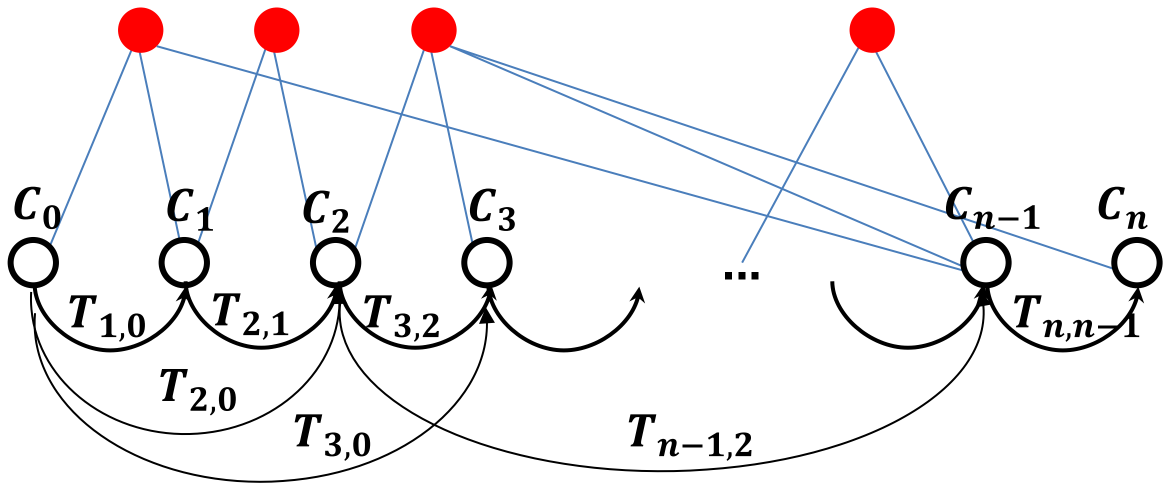


- Transformations can be computed also between non-adjacent frames T_{ij} (e.g., when features from previous keyframes are still observed). They can be used as additional constraints to improve cameras poses by minimizing the following

$$\sum_i \sum_j \left\| C_i - T_{ij} C_j \right\|^2$$

- For efficiency, only the last m keyframes are used
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it

Bundle Adjustment (BA)



- Similar to pose-optimization but it also optimizes 3D points

$$\underset{X^i, C_k}{\arg\min} \sum_{i,k} \| p_k^i - g(X^i, C_k) \|^2$$

- In order to not get stuck in local minima, the initialization should be close the minimum
- Gauss-Newton or Levenberg-Marquadt can be used

How do we estimate the motion?

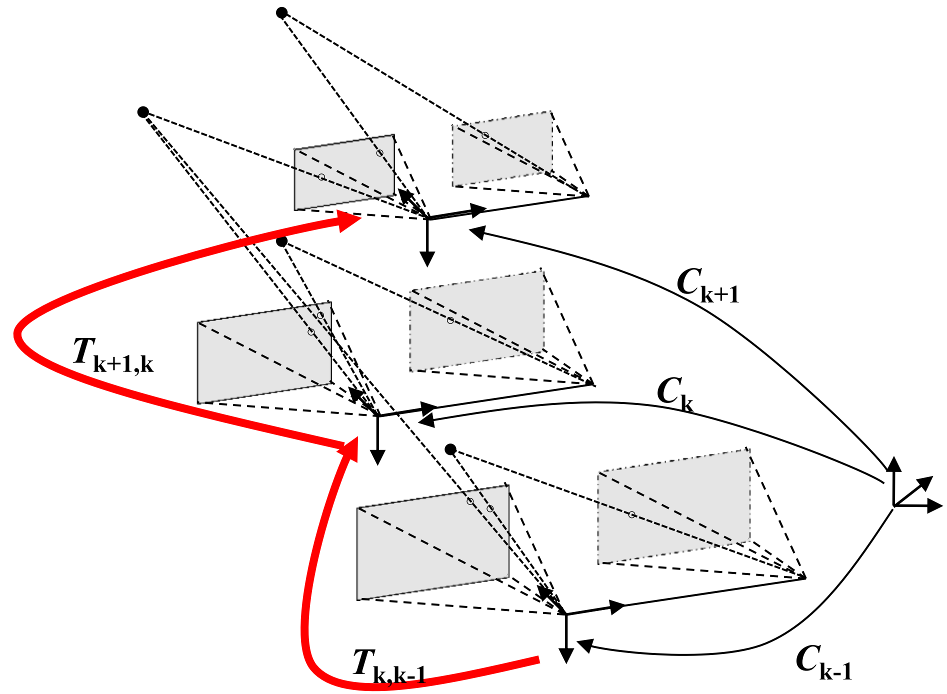
Image sequence

Feature detection

Feature matching (tracking)

Motion estimation

Local optimization



How do we estimate the motion?

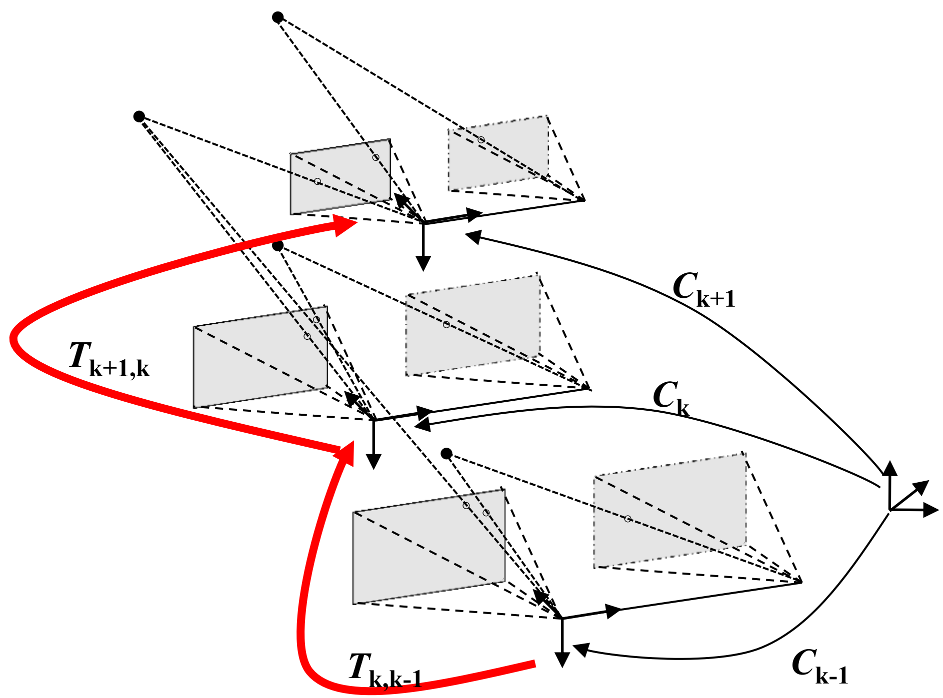
Image sequence

Feature detection

Feature matching (tracking)

Motion estimation		
2D-2D	3D-3D	3D-2D

Local optimization



Motion Estimation

Motion estimation		
2D-2D	3D-3D	3D-2D

Depending on whether the feature correspondences f_{k-1} and f_k are specified in 2D or 3D, there are three different cases:

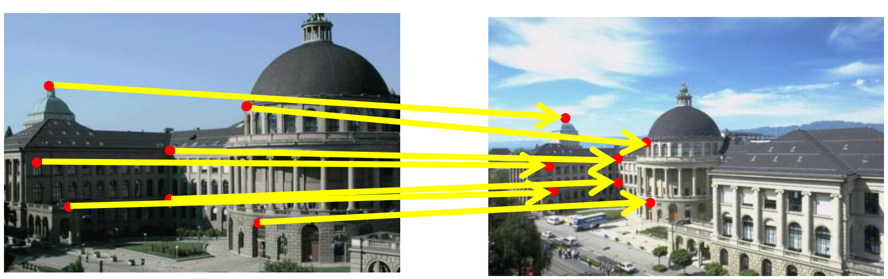
- **2D-to-2D**: both f_{k-1} and f_k are specified in 2D image coordinates
- **3D-to-3D**: both f_{k-1} and f_k are specified in 3D To do this, it is necessary to triangulate 3D points at each time instant, for instance, by using a stereo camera system
- **3D-to-2D**: f_{k-1} are specified in 3D and f_k are their corresponding 2D reprojections on the image I_k

2D-to-2D

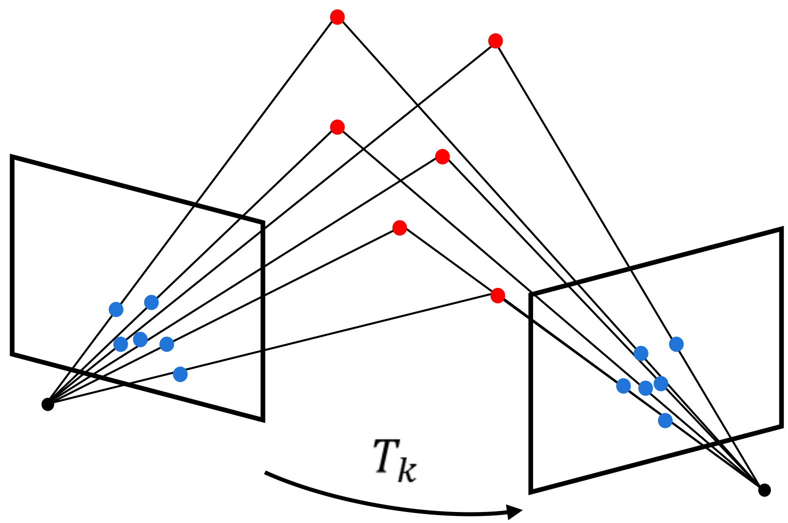
Motion from Image Feature Correspondences

- Both f_{k-1} and f_k are specified in 2D
- The minimal-case solution involves <u>5-point</u> correspondences
- The solution is found by determining the transformation that minimizes the reprojection error of the triangulated points in each image

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg\min_{X^i, C_k} \sum_{i,k} \| p_k^i - g(X^i, C_k) \|^2$$



I_{k-1} I_k



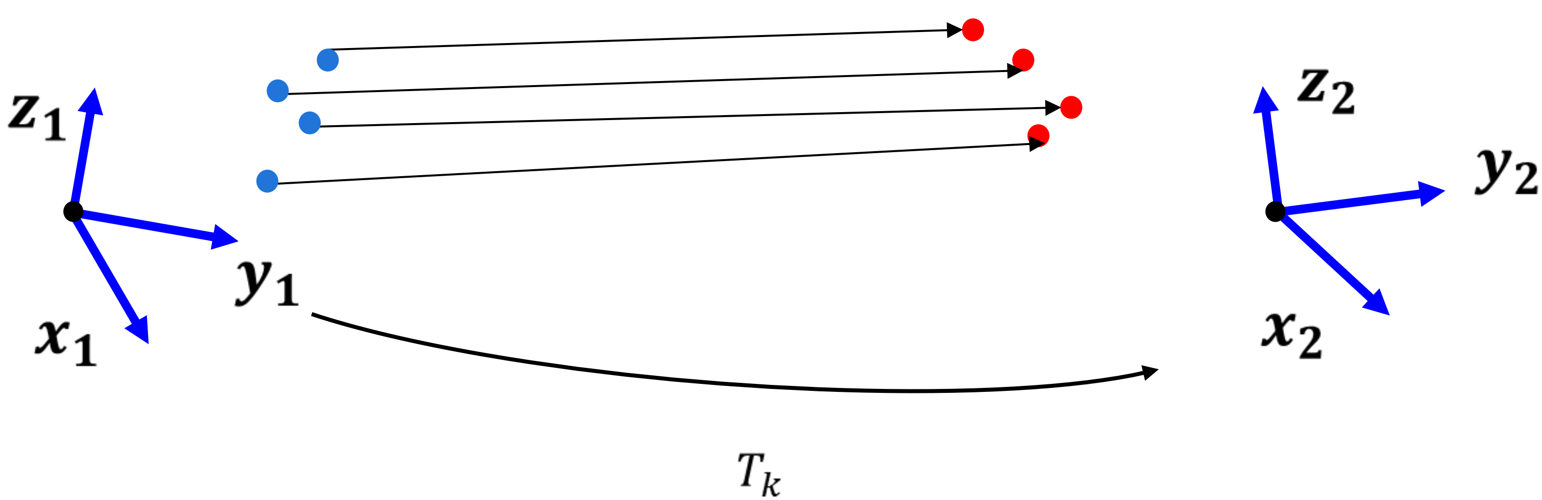
3D-to-3D

Motion estimation		
2D-2D	3D-3D	3D-2D

Motion from 3D-3D Point Correspondences

- Both f_{k-1} and f_k are specified in 3D
- To do this, it is necessary to triangulate 3D points (e.g. use a stereo camera)
- The minimal-case solution involves <u>3 non-collinear correspondences</u>
- The solution is found by determining the aligning transformation that minimizes the 3D-3D distance

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg\min_{T_k} \sum_i ||\tilde{X}_k^i - T_k \tilde{X}_{k-1}^i||$$



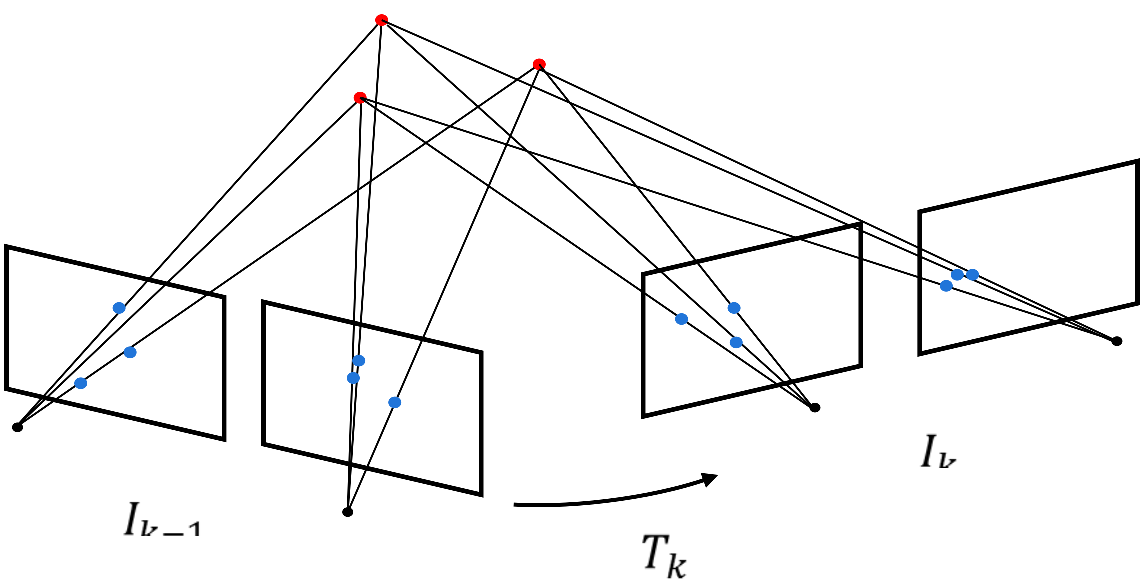
3D-to-3D

Motion estimation		
2D-2D	3D-3D	3D-2D

Motion from 3D-3D Point Correspondences

- Both f_{k-1} and f_k are specified in 3D
- To do this, it is necessary to triangulate 3D points (e.g. use a stereo camera)
- The minimal-case solution involves 3 non-collinear correspondences
- The solution is found by determining the aligning transformation that minimizes the 3D-3D distance

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg\min_{T_k} \sum_i \|\tilde{X}_k^i - T_k \tilde{X}_{k-1}^i\|$$

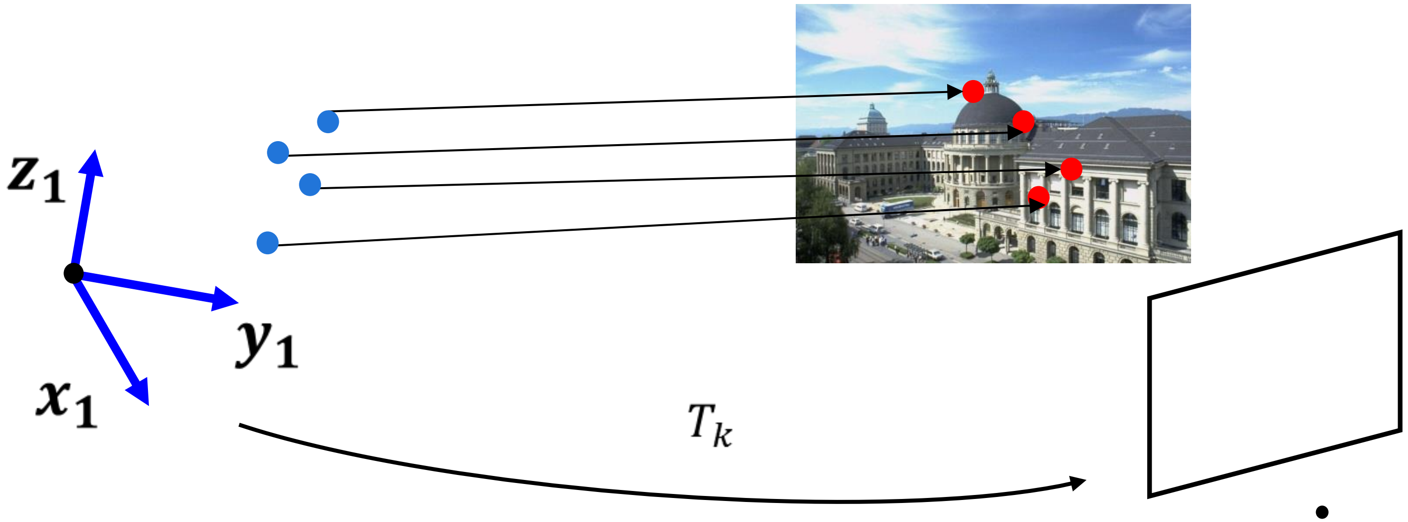


3D-to-2D

Motion from 3D Structure and Image Correspondences

- f_{k-1} is specified in 3D and f_k in 2D
- This problem is known as *camera resection* or PnP (perspective from *n* points)
- The minimal-case solution involves 3 correspondences
- The solution is found by determining the transformation that minimizes the reprojection error

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg\min_{T_k} \sum_i \| p_k^i - \hat{p}_{k-1}^i \|^2$$

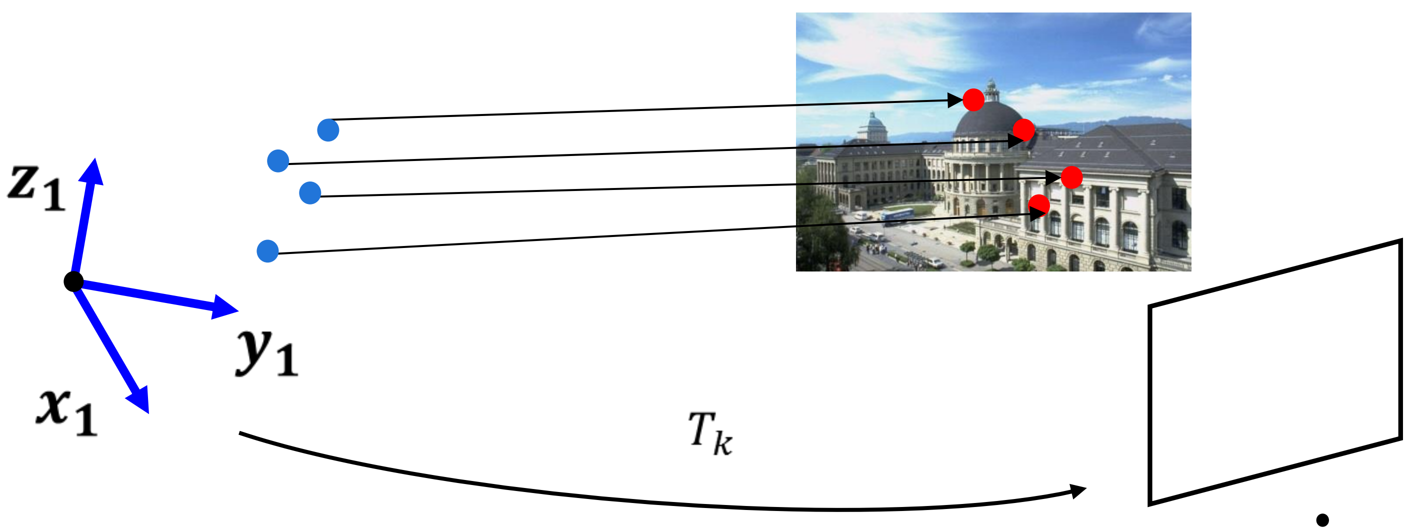


Motion estimation		
2D-2D	3D-3D	3D-2D

3D-to-2D

Motion from 3D Structure and Image Correspondences

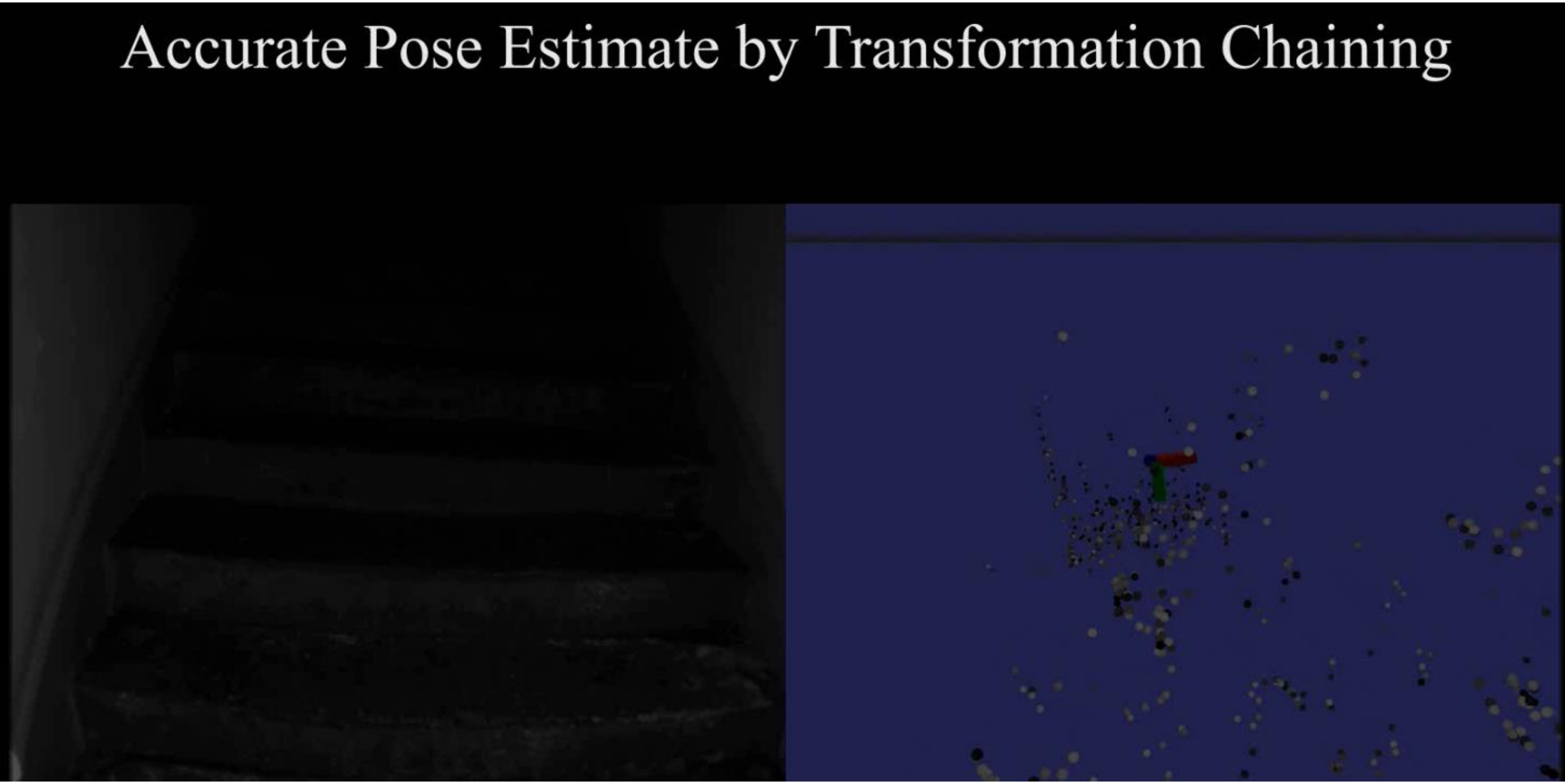
- In the monocular case, the 3D structure needs to be triangulated from two adjacent camera views (e.g., I_{k-2} and I_{k-1}) and then matched to 2D image features in a third view (e.g., I_k).



Motion Estimation: Summary

Type of correspondences	Monocular	Stereo
2D-2D	X	X
3D-3D		X
3D-2D	X	X

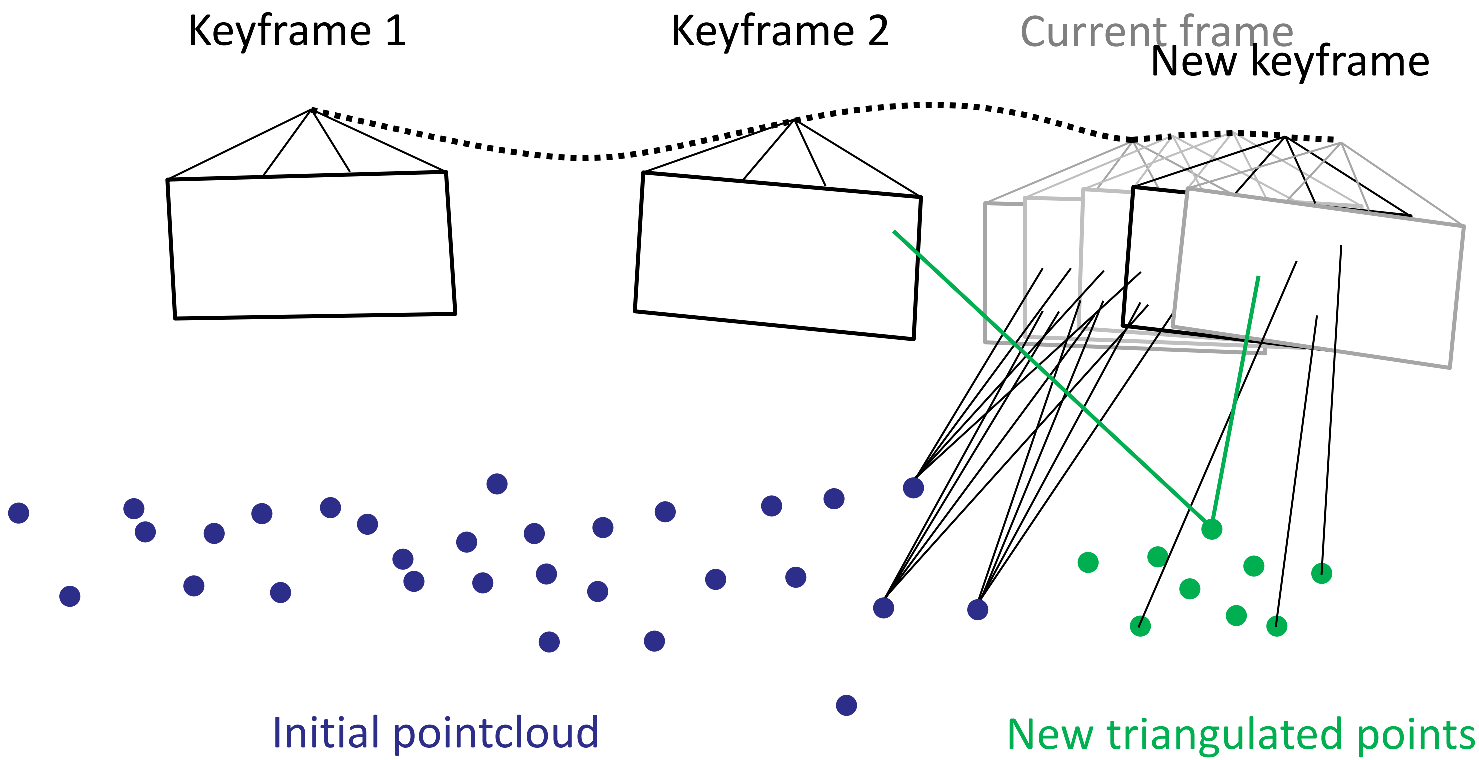
Stereo Visual Odometry (i.e., with 3D-to-3D Motion Estimation)



Courtesy of Stefan Leutenegger

Visual Odometry with 3D-to-2D Motion Estimation

- Compute camera position from known 3D-to-2D feature correspondences



Visual Odometry with 3D-to-2D Motion Estimation

- Compute camera position from known 3D-to-2D feature correspondences
- What's the minimal number of points correspondences
  - 3 for a non linear solution (P3P algorithm)
  - 6 for linear solution (DLT algorithm, see lecture 3, Image Formation 2)

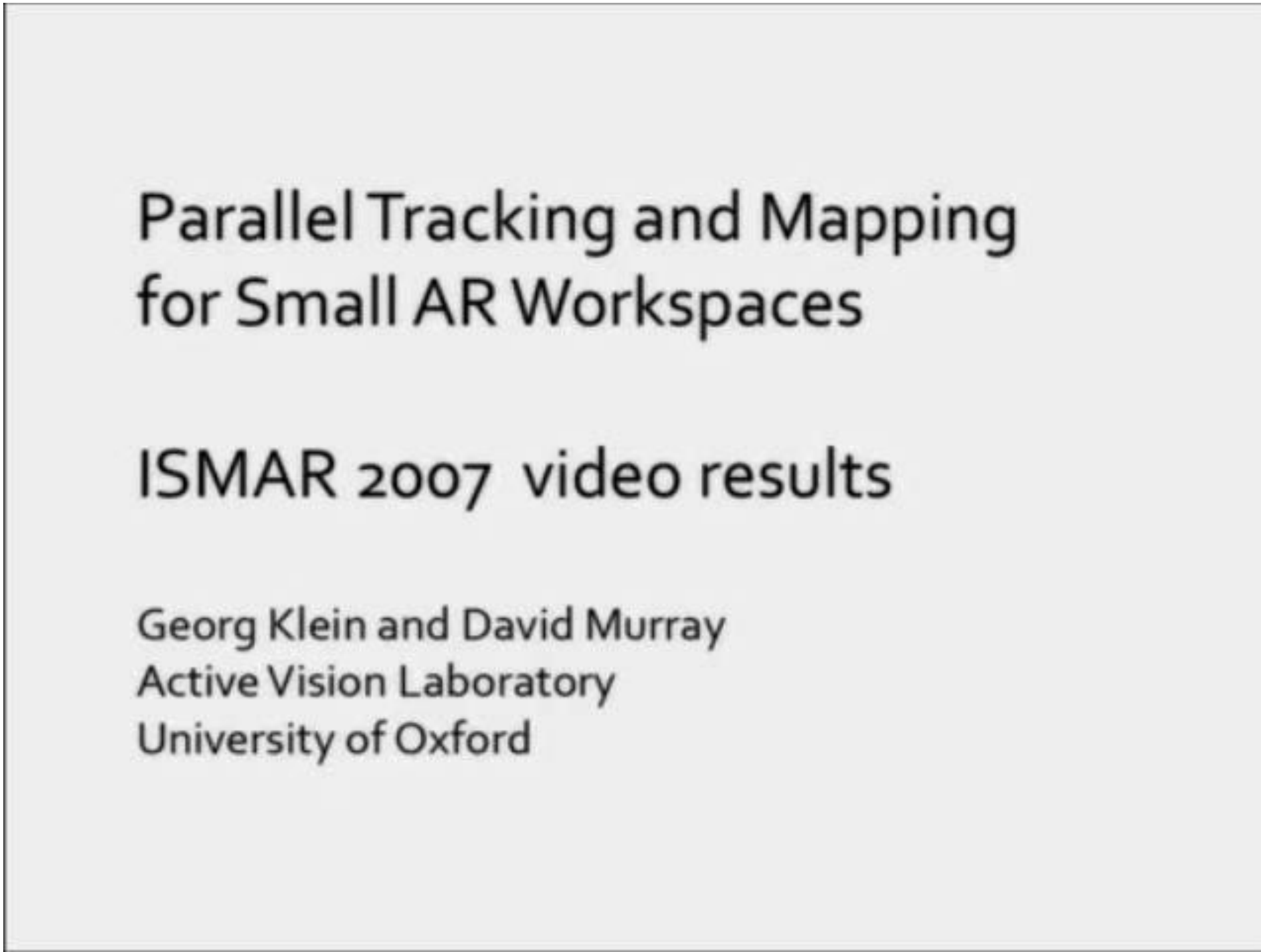
- You want to solve for R, t. K is known

$$\tilde{p} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\left[R \mid T\right] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Visual Odometry with 3D-to-2D Motion Estimation

- Applications
- Visual odometry with 3D-to-2D motion estimation is commonly used in monocular visual odometry
- There are several **open source** packages:
  - **PTAM** [Klein, 2007] -> Oxford, Murray's lab
  - **ORB-SLAM** [Mur-Artal, T-RO, 15] -> Zaragoza, Tardos' lab
  - **LSD-SLAM** [Engel, ECCV'14] -> Munich, Cremers' lab
  - **SVO** [Forster, ICRA'14] -> Zurich, Scaramuzza's lab ;-)

PTAM: Parallel Tracking and Mapping for Smal AR Workspaces, by Klein and Murray, ISMAR’07

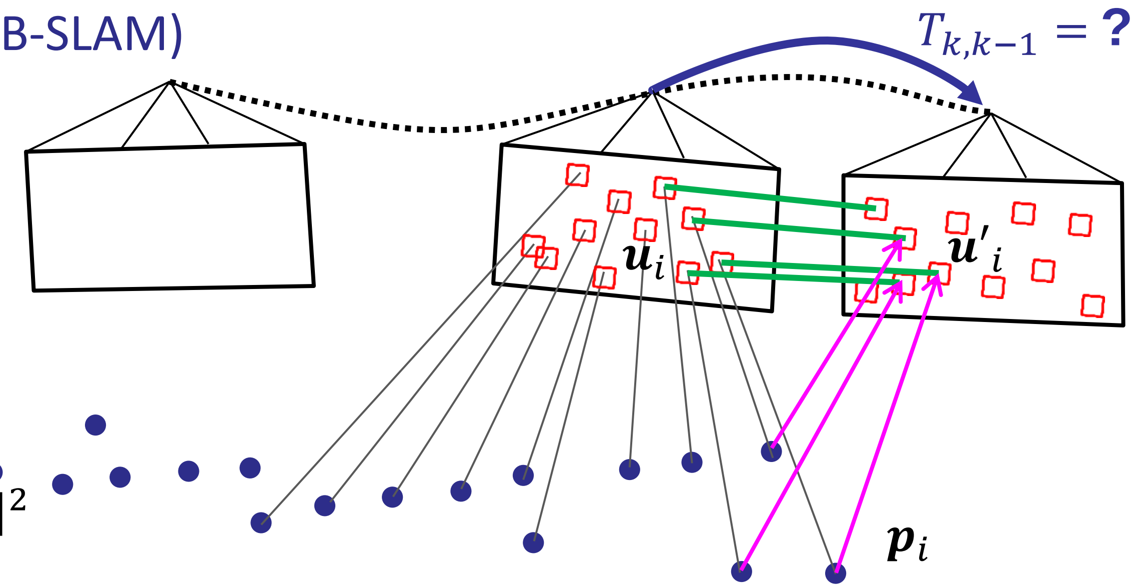


ORB-SLAM, by Mur-Artal, Montiel, Tardos, TRO'15

Feature-based vs. Direct Methods

Feature-based (e.g., PTAM, ORB-SLAM)

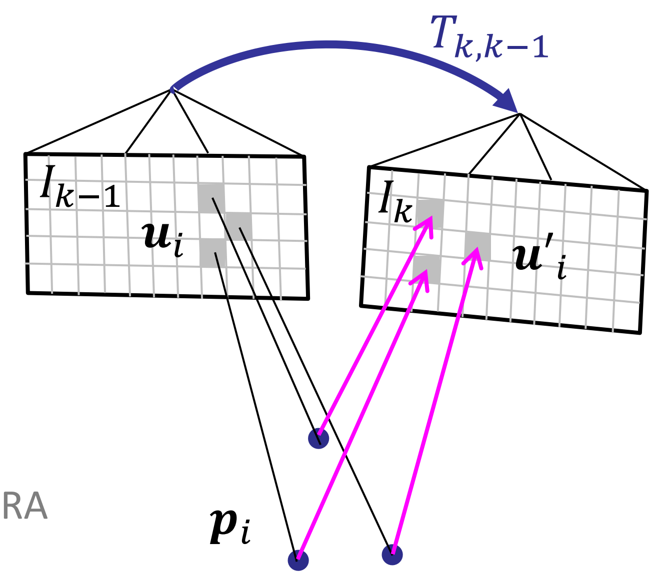
1. Feature extraction
2. Feature matching
3. RANSAC + P3P
4. **Reprojection error** minimization



$$T_{k,k-1} = \arg\min_T \sum_i \|\boldsymbol{u}'_i - \pi(\boldsymbol{p}_i)\|^2$$

Direct approaches (e.g., Meilland'13)

1. **Minimize photometric error**



$$T_{k,k-1} = \arg\min_T \sum_i \|I_k(\boldsymbol{u}'_i) - I_{k-1}(\boldsymbol{u}_i)\|^2$$

[Soatto'95, Meilland and Comport, IROS 2013], DVO [Kerl et al., ICRA 2013], DTAM [Newcombe et al., ICCV '11], ...

Feature-based vs. Direct Methods

Feature-based (e.g., PTAM, ORB-SLAM)

1. Feature extraction
2. Feature matching
3. RANSAC + P3P
4. **Reprojection error** minimization

$$T_{k,k-1} = \arg\min_{T} \sum_{i} \| \boldsymbol{u}'_i - \pi(\boldsymbol{p}_i) \|^2$$

✓ Large frame-to-frame motions

✗ Slow (20-30 Hz) due to costly feature extraction and matching

✗ Not robust to high-frequency and repetive texture

Direct approaches

1. **Minimize photometric error**

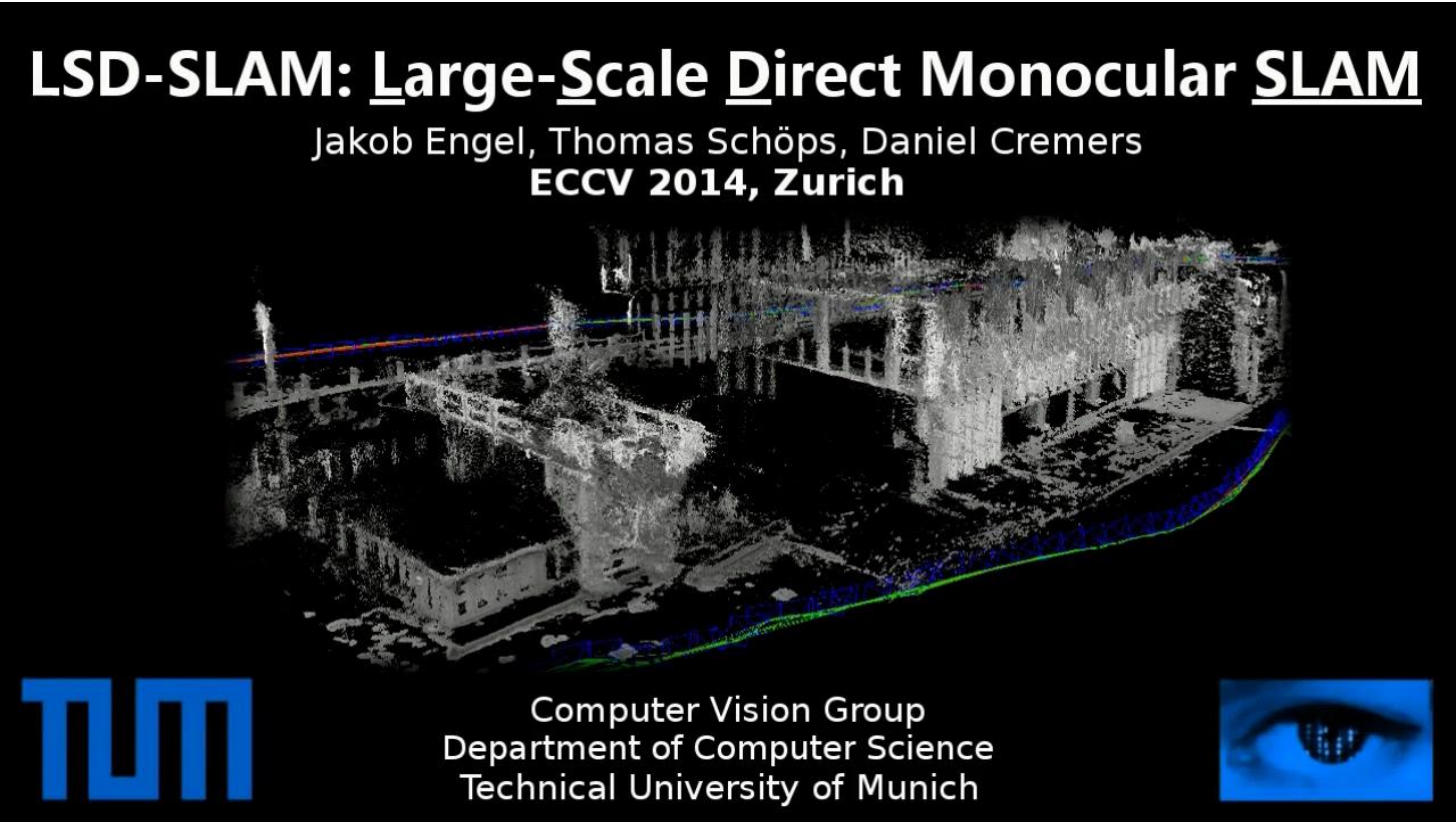
$$T_{k,k-1} = \arg\min_{T} \sum_{i} \| I_k(\boldsymbol{u}'_i) - I_{k-1}(\boldsymbol{u}_i) \|^2$$

✓ Every pixel in the image can be exploited (precision, robustness)

✓ Increasing camera frame-rate reduces computational cost per frame

✗ Limited to small frame-to-frame motion

LSD-SLAM: Large Scale Direct Monocular SLAM, by Engel, Stueckler, Cremers, ECCV'14



Feature-based vs. Direct Methods

Feature-based (e.g., PTAM, ORB-SLAM)

1. Feature extraction
2. Feature matching
3. RANS

✓ Large frame-to-frame motions

✗ Slow (20-30 Hz) due to costly feature extraction and matching

Our solution:

SVO: *Semi-direct* Visual Odometry [ICRA'14]

Combines feature-based and direct methods

1.

$$T_{k,k-1} = \arg\min_{T} \sum_i \| I_k(\boldsymbol{u}'_i) - I_{k-1}(\boldsymbol{u}_i) \|^2$$

✓ Increasing camera frame-rate reduces computational cost per frame

✗ Limited to small frame-to-frame motion

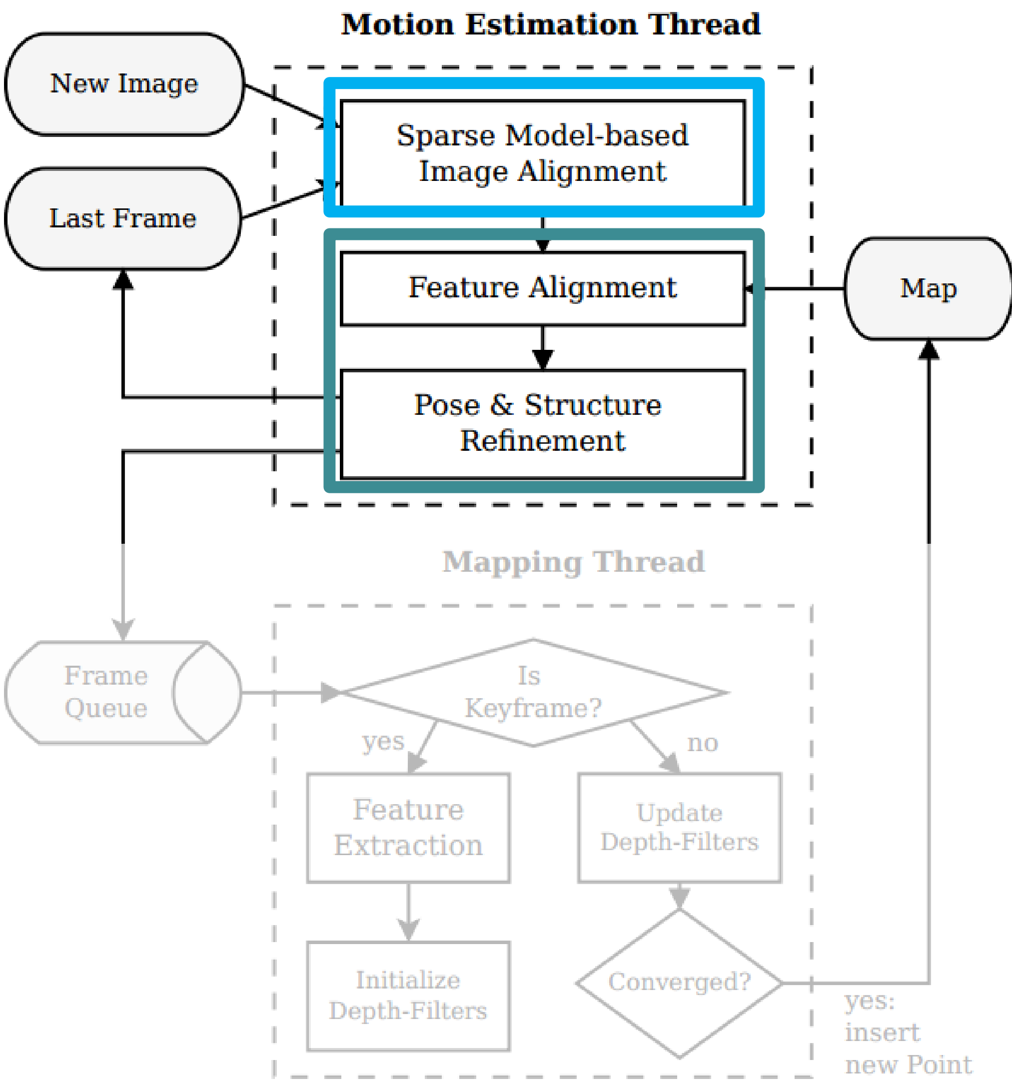
SVO: Semi-Direct Visual Odometry [ICRA'14]

Direct

- Frame-to-frame motion estimation

Feature-based

- Frame-to-Keyframe pose refinement



[Forster, Pizzoli, Scaramuzza, «SVO: Semi Direct Visual Odometry», ICRA'14]

SVO: Semi-Direct Visual Odometry [ICRA'14]

Direct

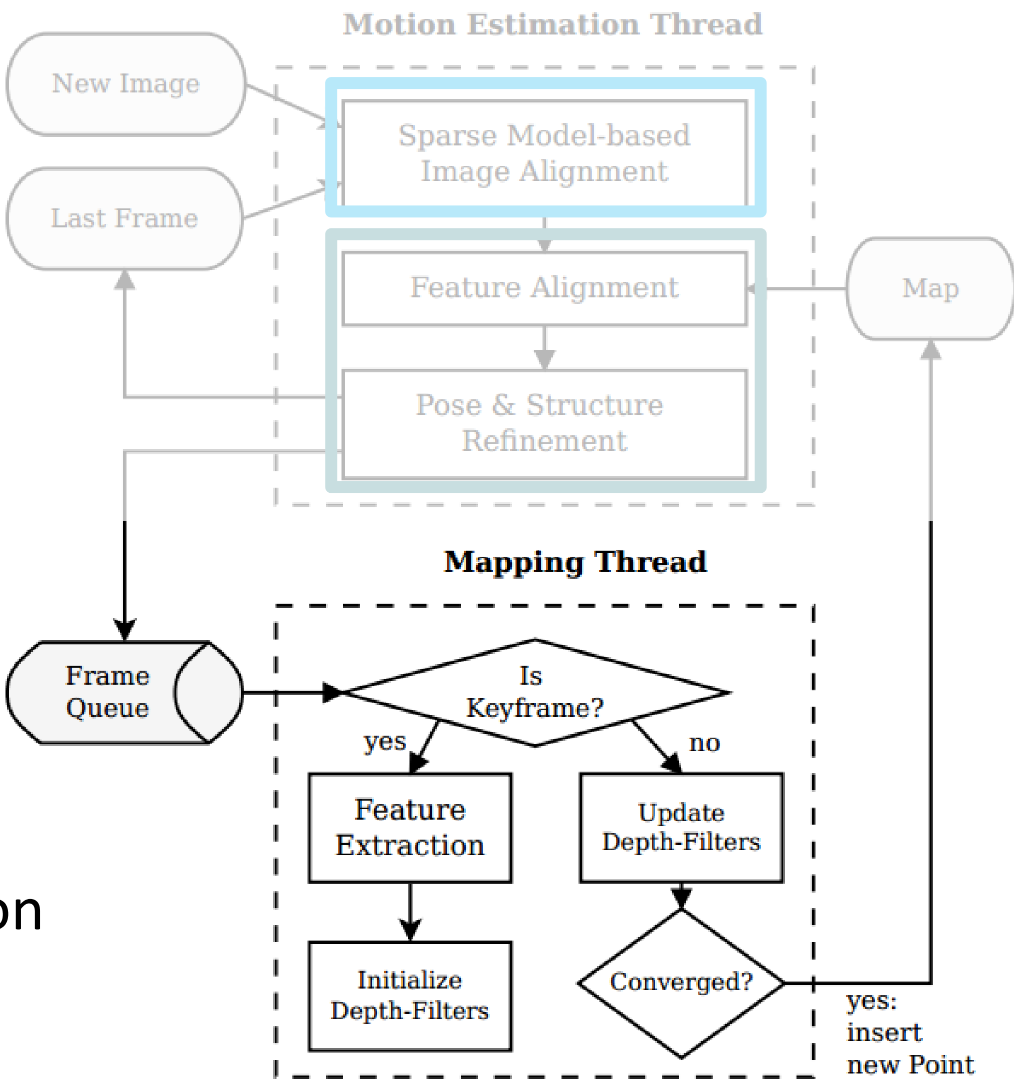
- Frame-to-frame motion estimation

Feature-based

- Frame-to-Kreyframe pose refinement

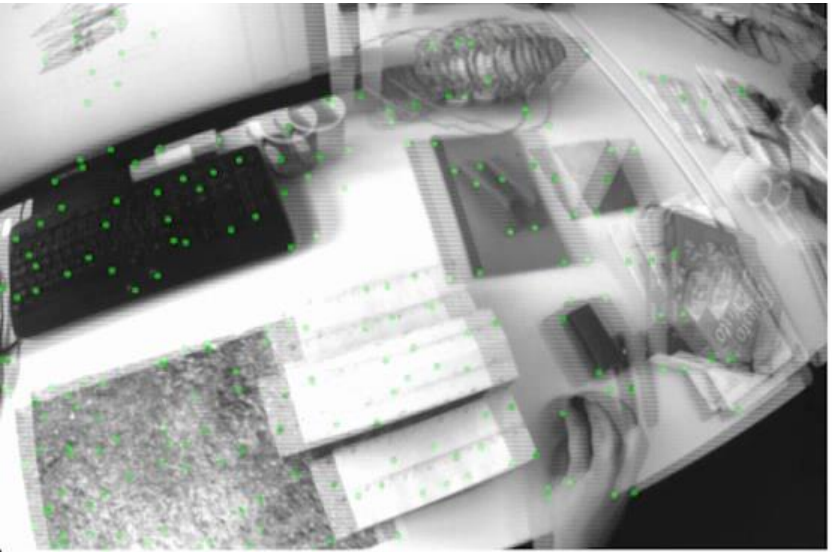
Mapping

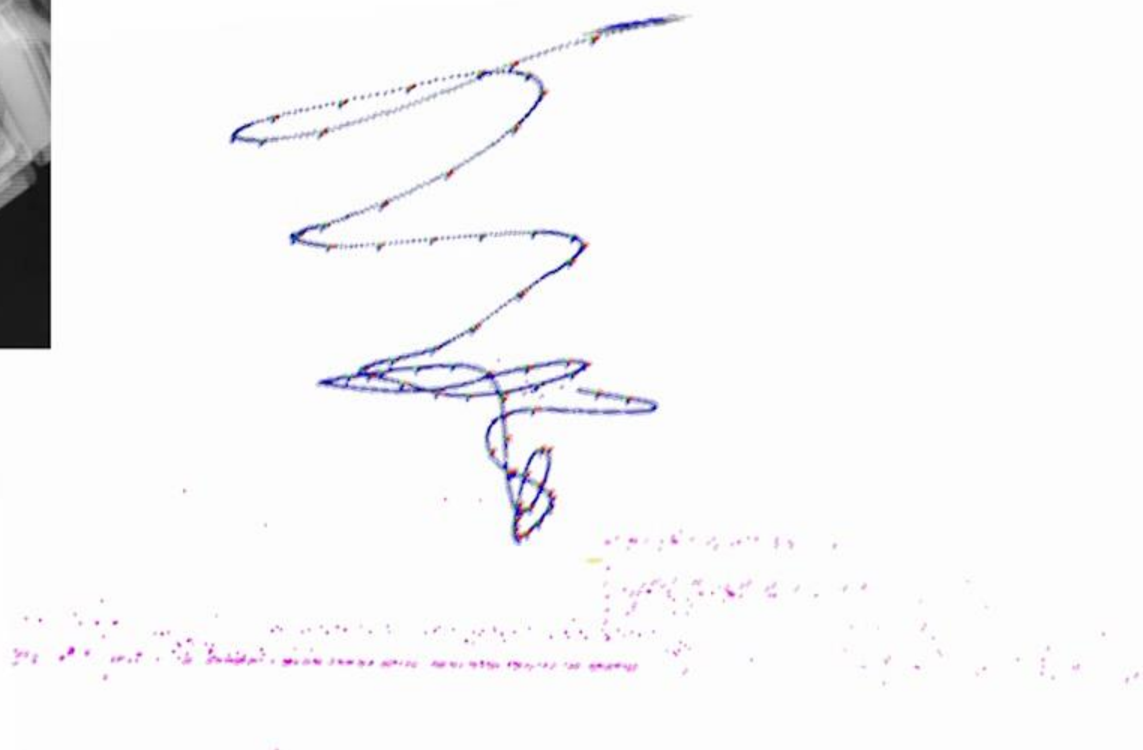
- ➢ Feature extraction only for every keyframe
- ➢ **Probabilistic depth** estimation of 3D points



[Forster, Pizzoli, Scaramuzza, «SVO: Semi Direct Visual Odometry», ICRA'14]

SVO: Semi-direct Visual Odometry, by Forster, Pizzoli, Scaramuzza, ICRA»14





Realtime
Camera at 70fps

Processing Times of SVO

Laptop (Intel i7, 2.8 GHz)

400 frames per second

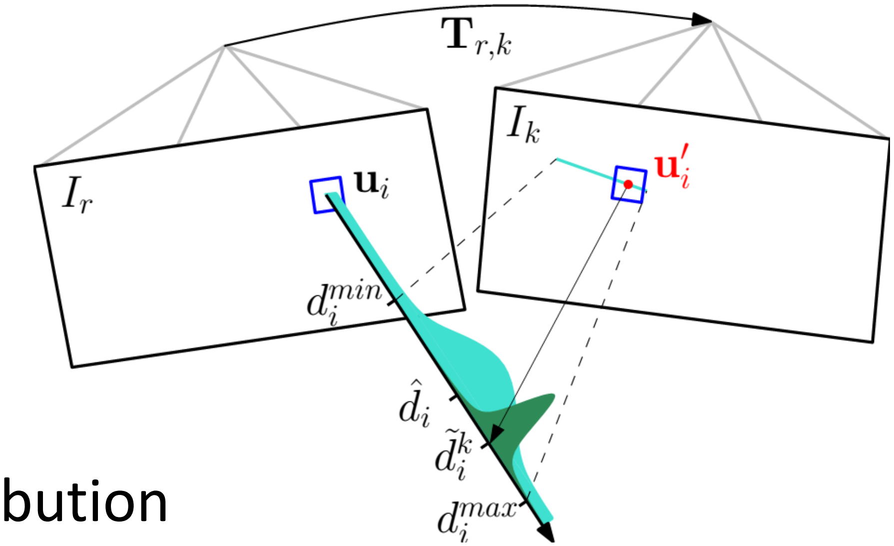
Embedded ARM Cortex-A9, 1.7 GHz

Up to 70 frames per second

Source Code

- Open Source available at: **github.com/uzh-rpg/rpg_svo**
- Works **with and without ROS**
- **Closed-Source professional edition** available for companies

Probabilistic Depth Estimation

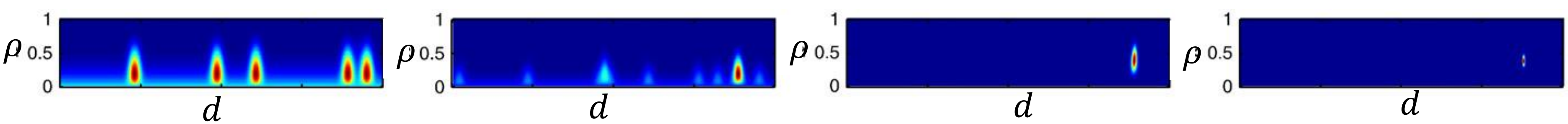


Depth-Filter:

- **Depth Filter** for every feature
- **Recursive Bayesian** depth estimation

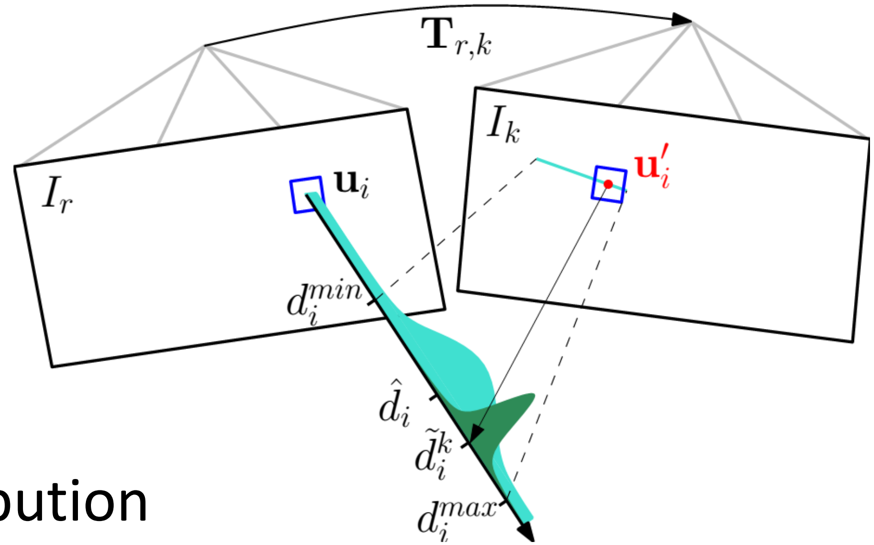
Mixture of Gaussian + Uniform distribution

$$p(\tilde{d}_i^k | d_i, \rho_i) = \rho_i \mathcal{N}\left(\tilde{d}_i^k \middle| d_i, \tau_i^2\right) + (1 - \rho_i)\mathcal{U}\left(\tilde{d}_i^k \middle| d_i^{\min}, d_i^{\max}\right)$$



[Forster, Pizzoli, Scaramuzza, SVO: Semi Direct Visual Odometry, IEEE ICRA'14]

Probabilistic Depth Estimation

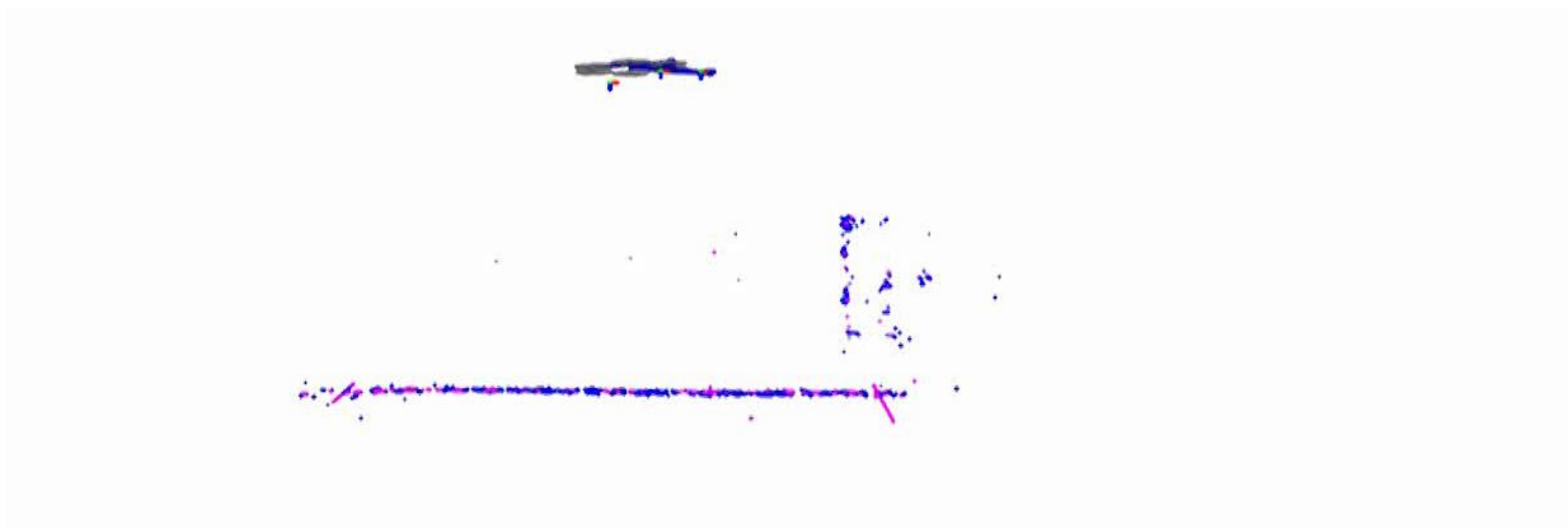


Depth-Filter:

- **Depth Filter** for every feature
- **Recursive Bayesian** depth estimation

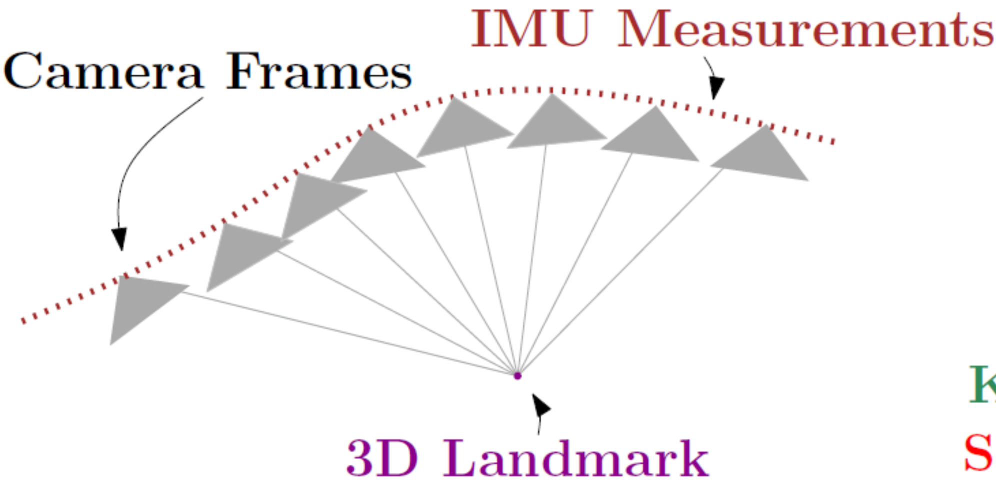
Mixture of Gaussian + Uniform distribution

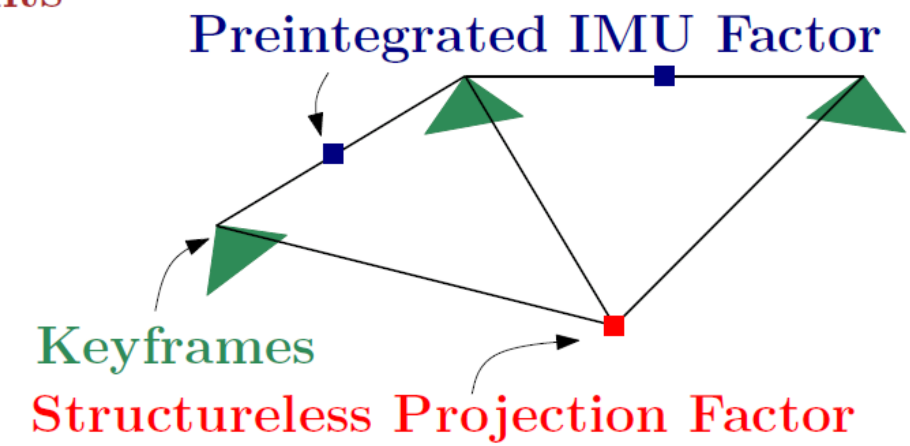
$$p(\tilde{d}_i^k | d_i, \rho_i) = \rho_i \mathcal{N}\left(\tilde{d}_i^k | d_i, \tau_i^2\right) + (1 - \rho_i)\mathcal{U}\left(\tilde{d}_i^k | d_i^{\min}, d_i^{\max}\right)$$



[Forster, Pizzoli, Scaramuzza, SVO: Semi Direct Visual Odometry, IEEE ICRA'14]

Visual-Inertial Fusion [RSS'15]





- Fusion is solved as a non-linear optimization problem (no Kalman filter):
- Increased accuracy over filtering methods

$$\sum_{(i,j)\in\mathcal{K}_k} \|\mathbf{r}_{\mathcal{I}_{ij}}\|^2_{\Sigma_{ij}} + \sum_{i\in\mathcal{K}_k} \sum_{l\in\mathcal{C}_i} \|\mathbf{r}_{\mathcal{C}_{il}}\|^2_{\Sigma_{\mathcal{C}}}$$

IMU residuals *Reprojection residuals*

[Forster, Carlone, Dellaert, Scaramuzza, IMU Preintegration on Manifold for efficient Visual-Inertial Maximum-a-Posteriori Estimation, RSS'15, **Best Paper Award Finalist**]

Visual-Inertial Odometry with Pre-integrated IMU Factors [RSS'15]

Test 1: Indoor Trajectory in Vicon Room

500 m trajectory - Accuracy: 0.1% of the travel distance

[Forster, Carlone, Dellaert, Scaramuzza, IMU Preintegration on Manifold for efficient Visual-Inertial Maximum-a-Posteriori Estimation, RSS'15, **Best Paper Award Finalist**]

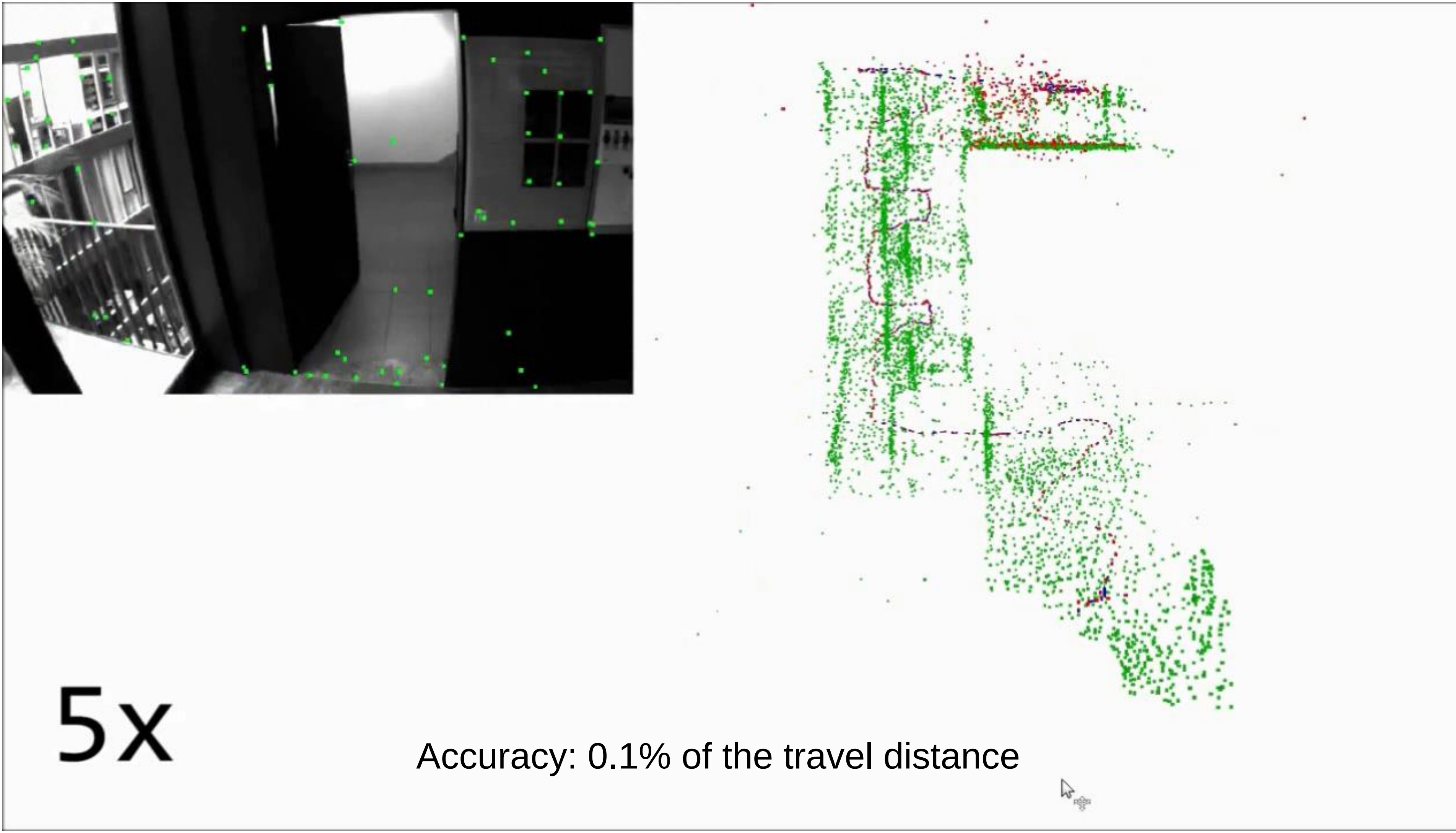
Visual-Inertial Odometry with Pre-integrated IMU Factors [RSS'15]

Test 2: Outdoor Trajectory Around Building

800 m trajectory - Accuracy: 0.1% of the travel distance

[Forster, Carlone, Dellaert, Scaramuzza, IMU Preintegration on Manifold for efficient Visual-Inertial Maximum-a-Posteriori Estimation, RSS'15, **Best Paper Award Finalist]**

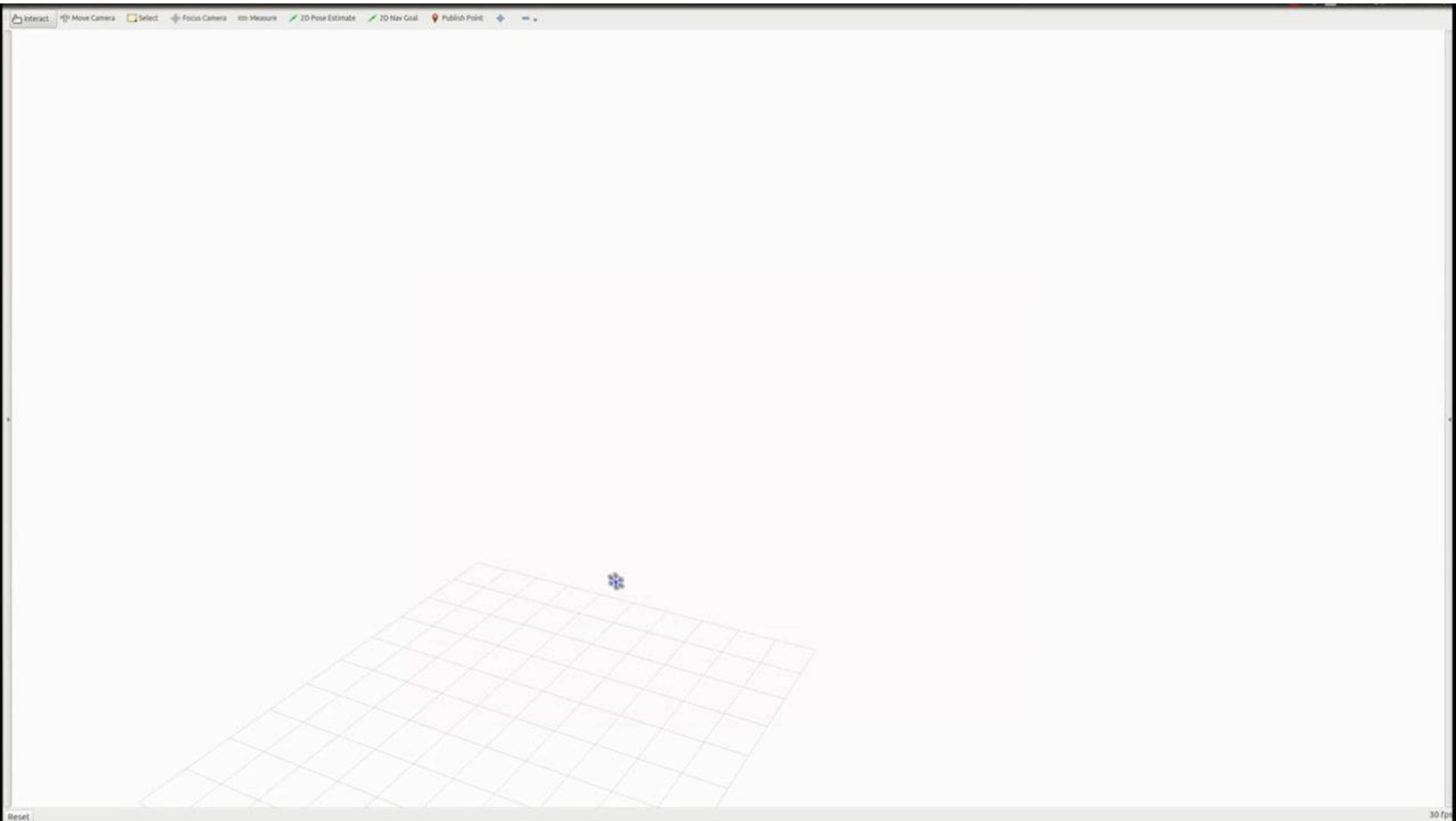
Visual-Inertial Odometry with Pre-integrated IMU Factors [RSS'15]



[Forster, Carlone, Dellaert, Scaramuzza, IMU Preintegration on Manifold for efficient Visual-Inertial Maximum-a-Posteriori Estimation, RSS'15, **Best Paper Award Finalist**]

Generalization to Multiple Cameras

- 4 non overlapping cameras (dataset courtesy of AUDI)
- SVO on 4 camera streams at 50 Hz on CPU



Software and Dataset

SOFTWARE AND DATASETS

Author	Description	Link
Willow Garage	OpenCV: A computer vision library maintained by Willow Garage. The library includes many of the feature detectors mentioned in this tutorial (e.g., Harris, KLT, SIFT, SURF, FAST, BRIEF, ORB). In addition, the library contains the basic motion-estimation algorithms as well as stereo-matching algorithms.	http://opencv.willowgarage.com
Willow Garage	ROS (Robot Operating System): A huge library and middleware maintained by Willow Garage for developing robot applications. Contains a visual-odometry package and many other computer-vision-related packages.	http://www.ros.org
Willow Garage	PCL (Point Cloud Library): A 3D-data-processing library maintained from Willow Garage, which includes useful algorithms to compute transformations between 3D-point clouds.	http://pointclouds.org
Henrik Stewenius et al.	5-point algorithm: An implementation of the 5-point algorithm for computing the essential matrix.	http://www.vis.uky.edu/~stewe/FIVEPOINT/
Changchang Wu et al.	SiftGPU: Real-time implementation of SIFT.	http://cs.unc.edu/~ccwu/siftgpu
Nico Cornelis et al.	GPUSurf: Real-time implementation of SURF.	http://homes.esat.kuleuven.be/~ncorneli/gpusurf
Christopfer Zach	GPU-KLT: Real-time implementation of the KLT tracker.	http://www.inf.ethz.ch/personal/chzach/opensource.html
Edward Rosten	Original implementation of the FAST detector.	http://www.edwardrosten.com/work/fast.html

Software and Dataset

Michael Calonder	Original implementation of the BRIEF descriptor.	http://cvlab.epfl.ch/software/brief/
Leutenegger et al.	BRISK feature detector.	http://www.asl.ethz.ch/people/lestefan/personal/BRISK
Jean-Yves Bouguet	Camera Calibration Toolbox for Matlab.	http://www.vision.caltech.edu/bouguetj/calib_doc
Davide Scaramuzza	OCamCalib: Omnidirectional Camera Calibration Toolbox for MATLAB.	https://sites.google.com/site/scarabotix/ocamcalib-toolbox
Christopher Mei	Omnidirectional Camera Calibration Toolbox for MATLAB	http://homepages.laas.fr/~cmei/index.php/Toolbox
Mark Cummins	FAB-MAP: Visual-word-based loop detection.	http://www.robots.ox.ac.uk/~mjc/Software.htm
Friedrich Fraundorfer	Vocsearch: Visual-word-based place recognition and image search.	http://www.inf.ethz.ch/personal/fraundof/page2.html
Manolis Lourakis	SBA: Sparse Bundle Adjustment	http://www.ics.forth.gr/~lourakis/sba
Christopher Zach	SSBA: Simple Sparse Bundle Adjustment	http://www.inf.ethz.ch/personal/chzach/opensource.html
Rainer Kuemmerle et al.	G2O: Library for graph-based nonlinear function optimization. Contains several variants of SLAM and bundle adjustment.	http://openslam.org/g2o
RAWSEEDS EU Project	RAWSEEDS: Collection of datasets with different sensors (lidars, cameras, IMUs, etc.) with ground truth.	http://www.rawseeds.org
SFLY EU Project	SFLY-MAV dataset: Camera-IMU dataset captured from an aerial vehicle with Vicon data for ground truth.	http://www.sfly.org
Davide Scaramuzza	ETH OMNI-VO: An omnidirectional-image dataset captured from the roof of a car for several kilometers in a urban environment. MATLAB code for visual odometry is provided.	http://sites.google.com/site/scarabotix