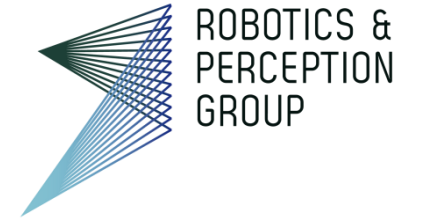




University of
Zurich^{UZH}



Vision Algorithms for Mobile Robotics

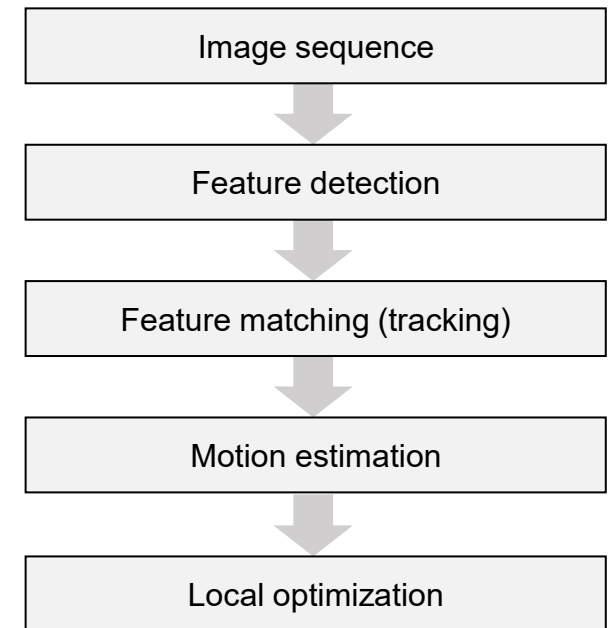
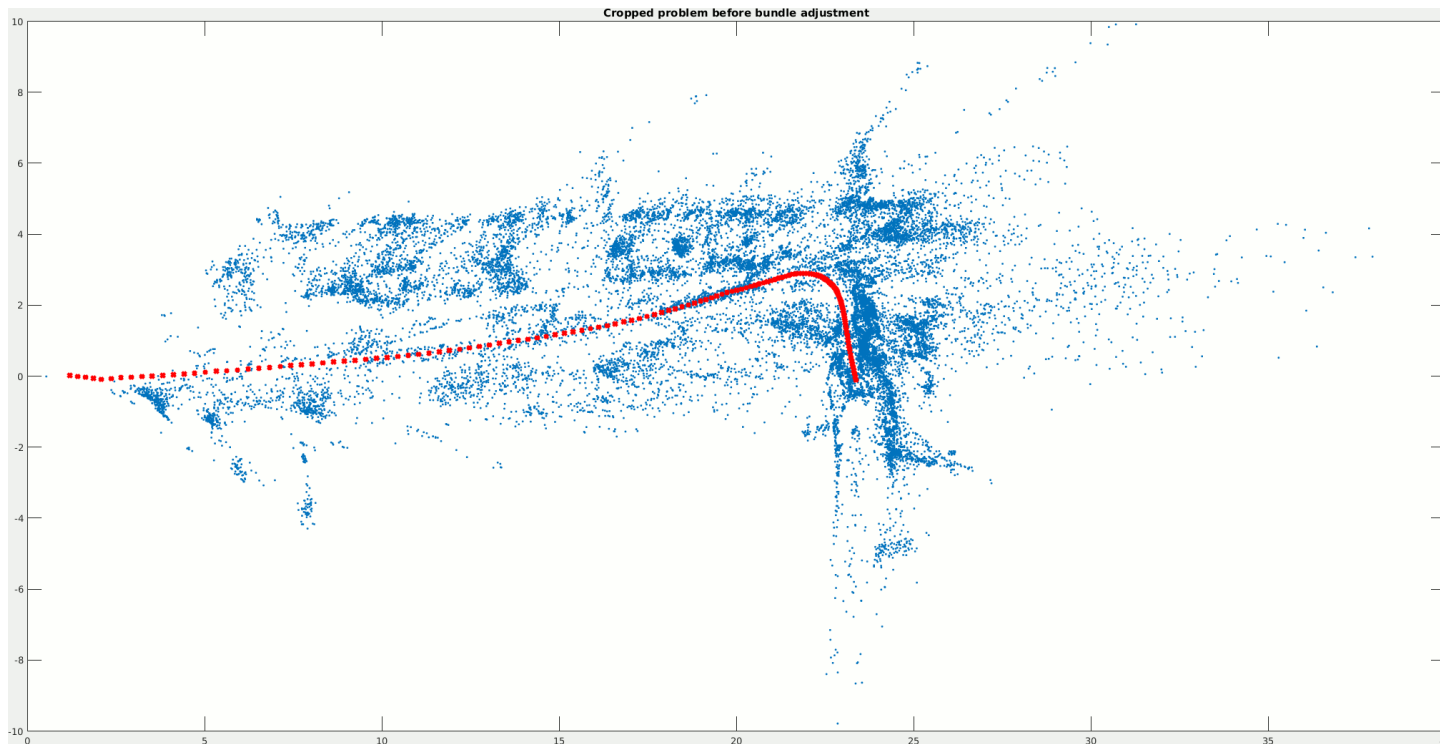
Lecture 10 Multiple View Geometry 4

Davide Scaramuzza

<https://rpg.ifi.uzh.ch>

Today: Lab Exercise

- Bundle Adjustment
- Q&A session on mini-projects and VO integration

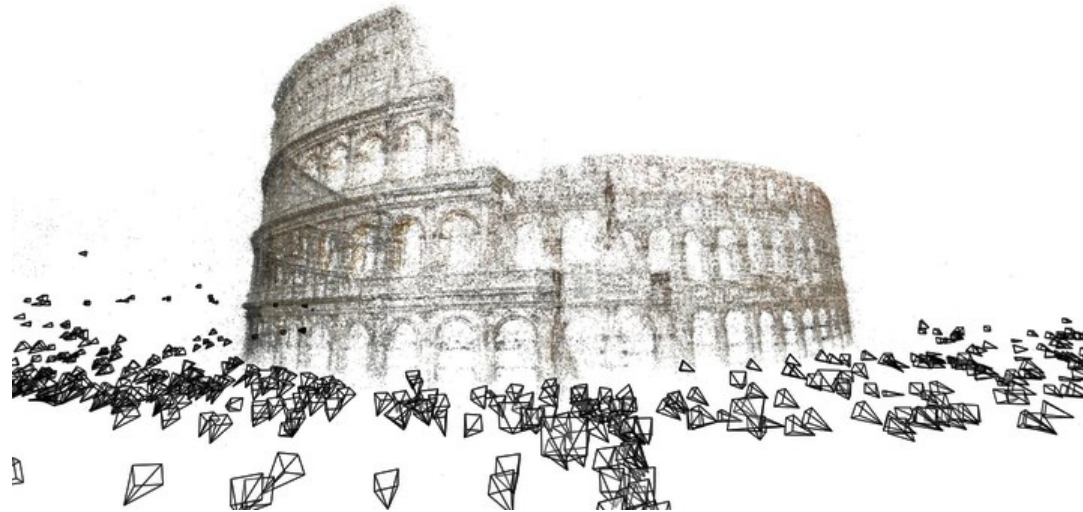


n -View Structure From Motion

- Compute initial structure and motion using either:
 - **Hierarchical SFM**
 - **Sequential SFM** → Visual Odometry (VO)
- Refine simultaneously structure and motion through BA

Hierarchical SFM applied to random internet images

- Reconstruction from 150,000 images from Flickr associated with the tags “Rome”
- 4 million 3D points. Cloud of 496 computers. 21 hours of computation!



Agarwal, Snavely, Simon, Seitz, Szeliski, *Building Rome in a Day*, International Conference on Computer Vision (ICCV), 2009. [PDF, code, datasets](#)

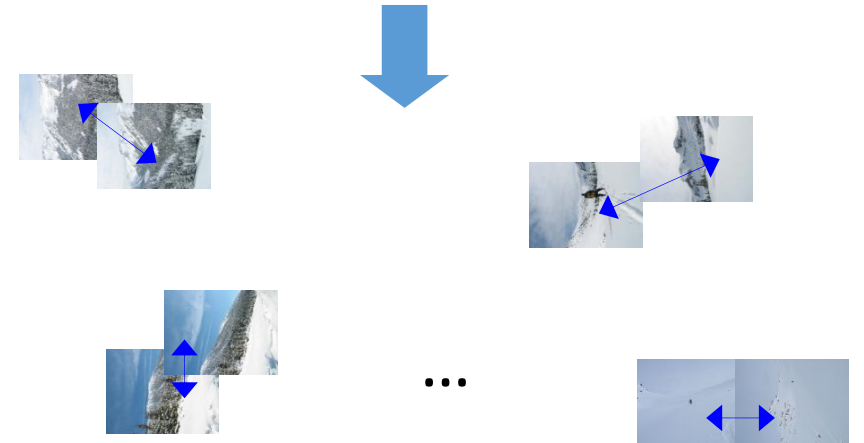
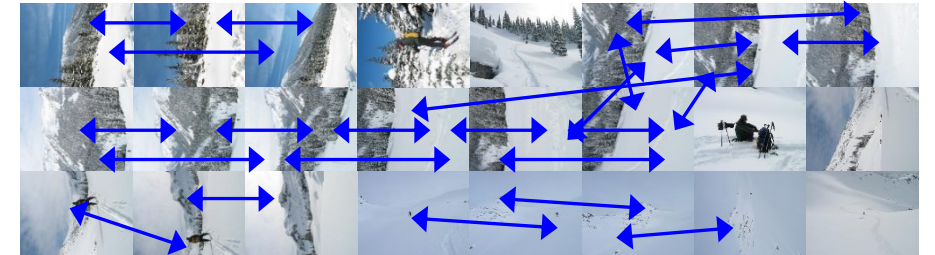
Most influential paper of 2009

State of the art software: [COLMAP](#):

Schoenberger, Frahm, *Structure-from-Motion Revisited*, Conf. on Computer Vision and Pattern Recognition (CVPR), 2016

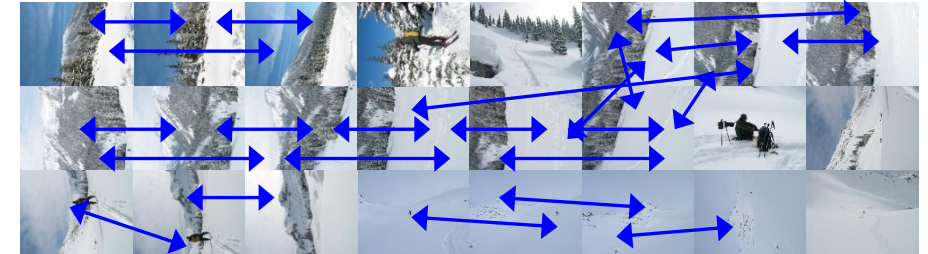
Hierarchical SFM

1. Extract and match features between nearby frames
2. Build clusters consisting of 2 nearby frames

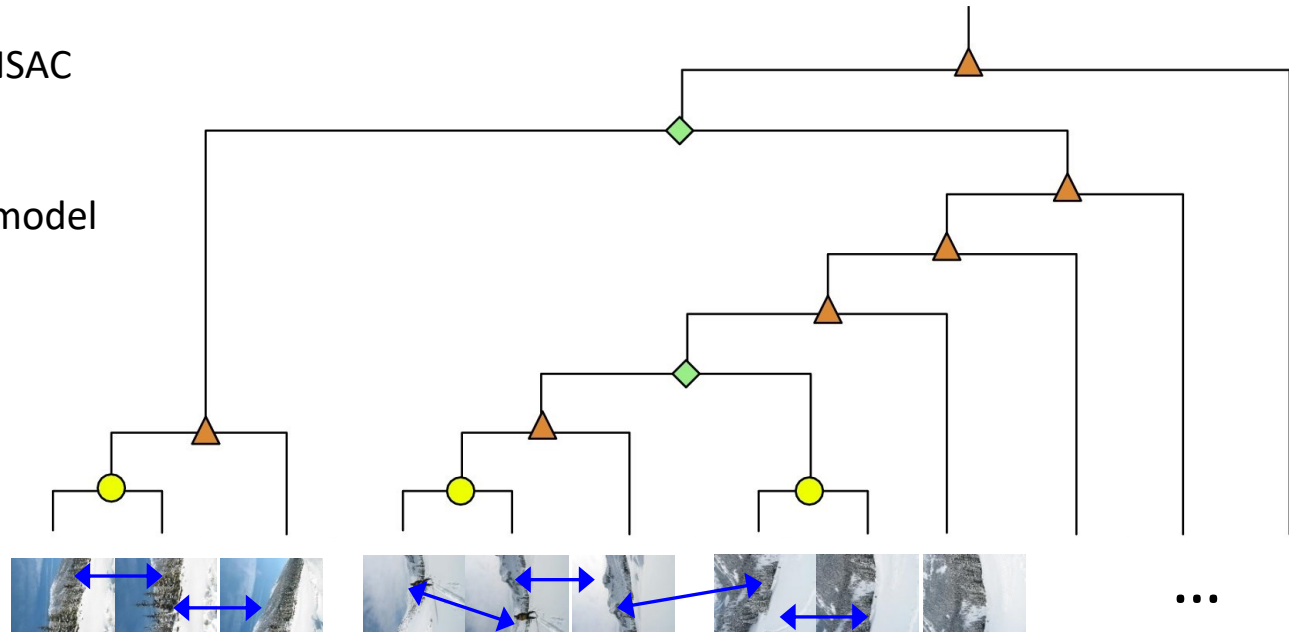


Hierarchical SFM

1. Extract and match features between nearby frames
2. Build clusters consisting of 2 nearby frames
3. Extract topological tree (e.g., count number of SIFT matches)
4. Start from the terminal nodes
 1. Compute 2-view SFM and build 3D model (point cloud)
5. Iterate according to tree structure:
 1. Merge new view by running 3-point RANSAC between 3D model and 3rd view
 2. Merge near-by models by running again 3-point RANSAC between one 3D model and one view of the other 3D model
 3. Bundle adjust



The circle \circ corresponds to the creation of a stereo-model, the triangle \triangle corresponds to applying PNP, the diamond \diamond corresponds to a fusion of two partial independent models.



COLMAP

- COLMAP has become the gold standard SFM unordered and ordered image set
- It can be used for ground truth of VO/VSLAM algorithms
- COLMAP poses and 3D points are even used to initialize dense mapping methods, such as *Nerf* and *Gaussian Splatting*
- Extensions of COLMAP integrating learned-based features or other sensor modalities (IMU, GPS, ...) exist

[1] Mildenhall, Srinivasan, Tancik, Barron, Ramamoorthi, Ng, *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*, ECCV'20

[2] Kerbl, Kopanas, Leimkühler, Drettakis, *3D Gaussian Splatting for Real-Time Radiance Field Rendering*, SIGGRAPH'23

[3] Sarlin, Cadena, Siegwart, Dymczyk, *From Coarse to Fine: Robust Hierarchical Localization at Large Scale*, CVPR'19

[4] Cioffi, Cieslewski, Scaramuzza, *Continuous-time vs. Discrete-time vision-based SLAM: A comparative study*, RA-L'22



Camera poses are depicted in red

n -View Structure From Motion

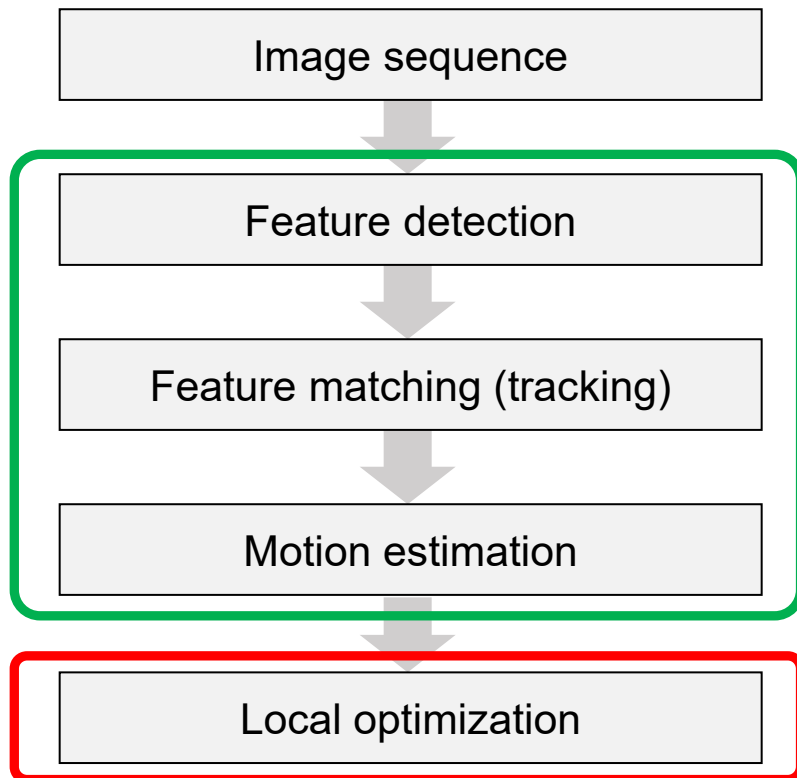
- Compute initial structure and motion using either:
 - **Hierarchical SFM**
 - **Sequential SFM** → Visual Odometry (VO)
- Refine simultaneously structure and motion through BA

Sequential SFM (also called Visual Odometry (VO))

- Initialize structure and motion from 2 views (**bootstrapping**)
- For each additional view
 - Determine pose (**localization**)
 - Extend structure, i.e., extract and triangulate new features (**mapping**)
 - Refine structure and motion through Bundle Adjustment (BA) (**optimization**)

VO Flow Chart: review (Lecture 01)

VO computes the camera path incrementally (pose after pose)

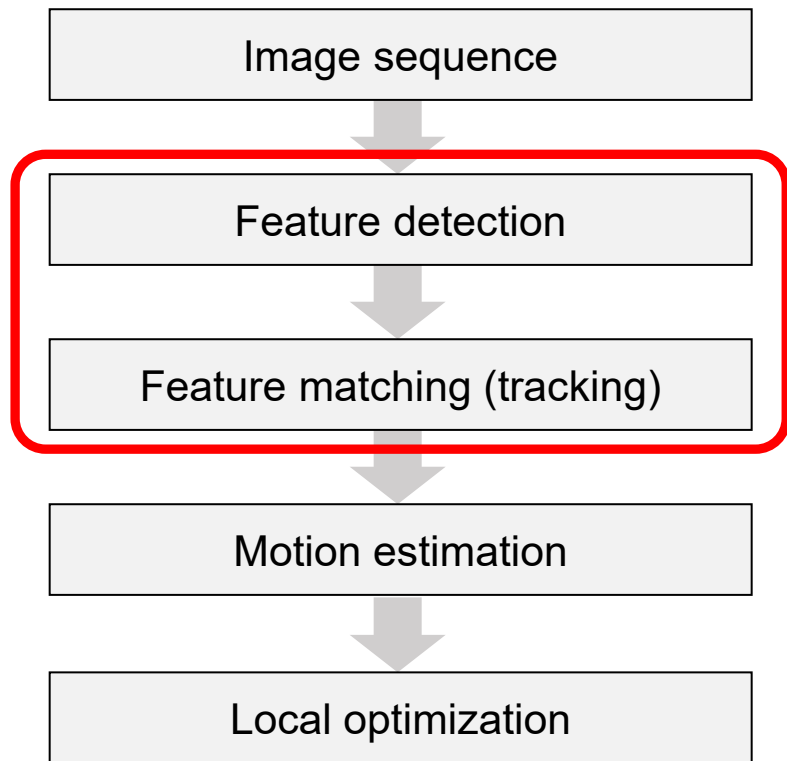


Front-end: outputs the *relative pose* between the *last two frames*

Back-end: “*adjusts*” the relative poses among *multiple recent frames*

VO Flow Chart: review (Lecture 01)

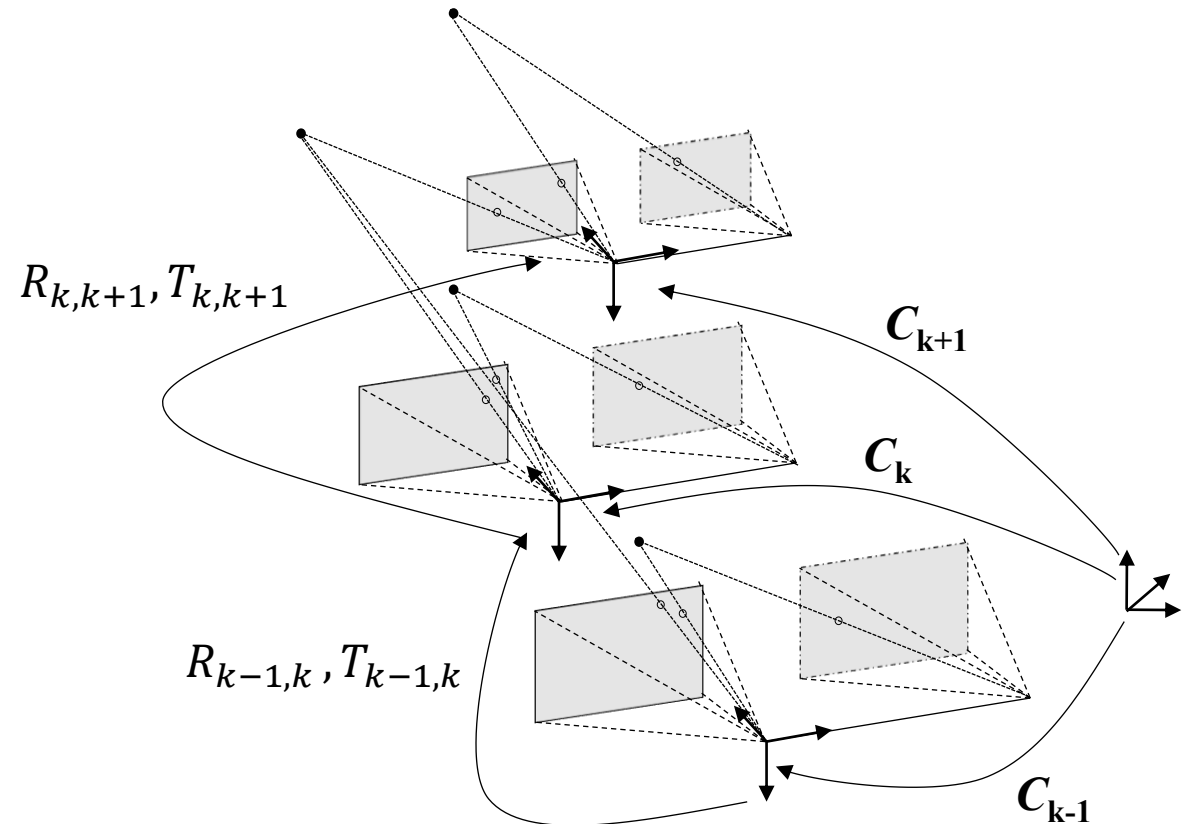
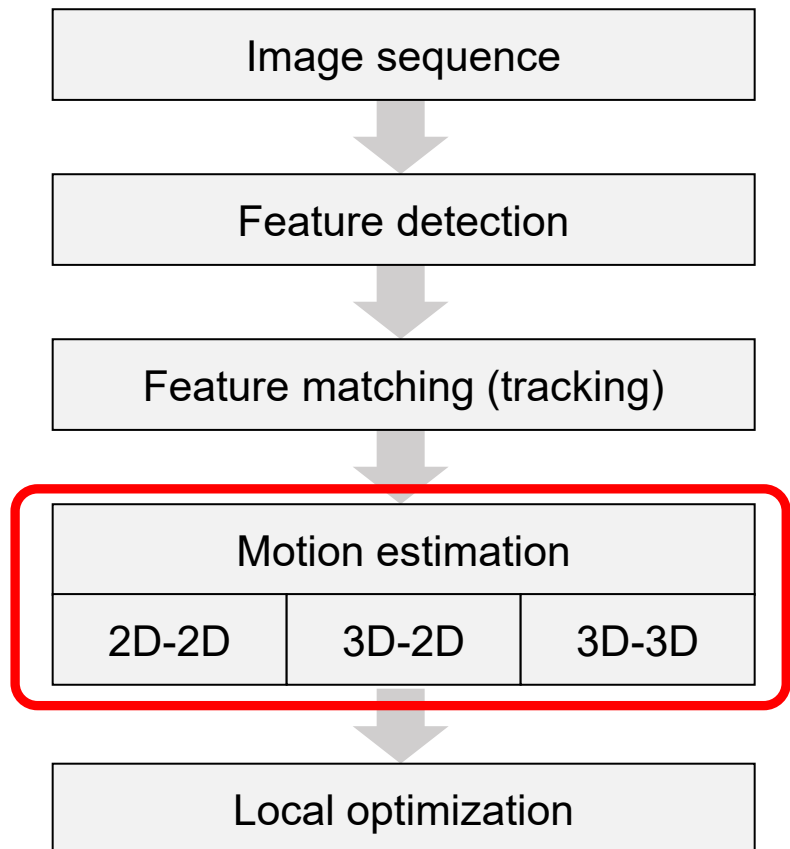
VO computes the camera path incrementally (pose after pose)



Features tracked over multiple recent frames overlaid on the last frame

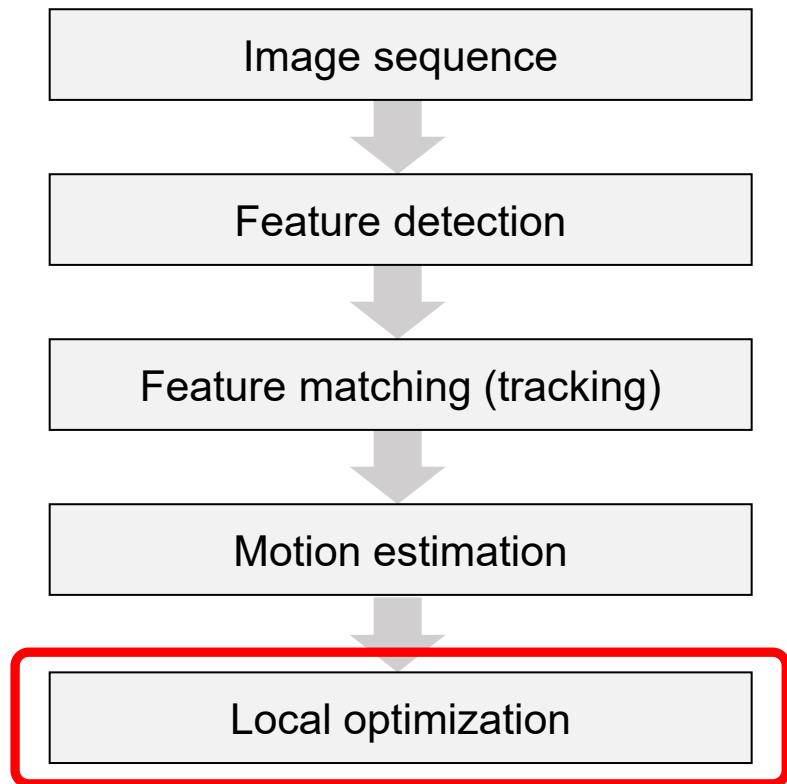
VO Flow Chart: review (Lecture 01)

VO computes the camera path incrementally (pose after pose)



VO Flow Chart: review (Lecture 01)

VO computes the camera path incrementally (pose after pose)



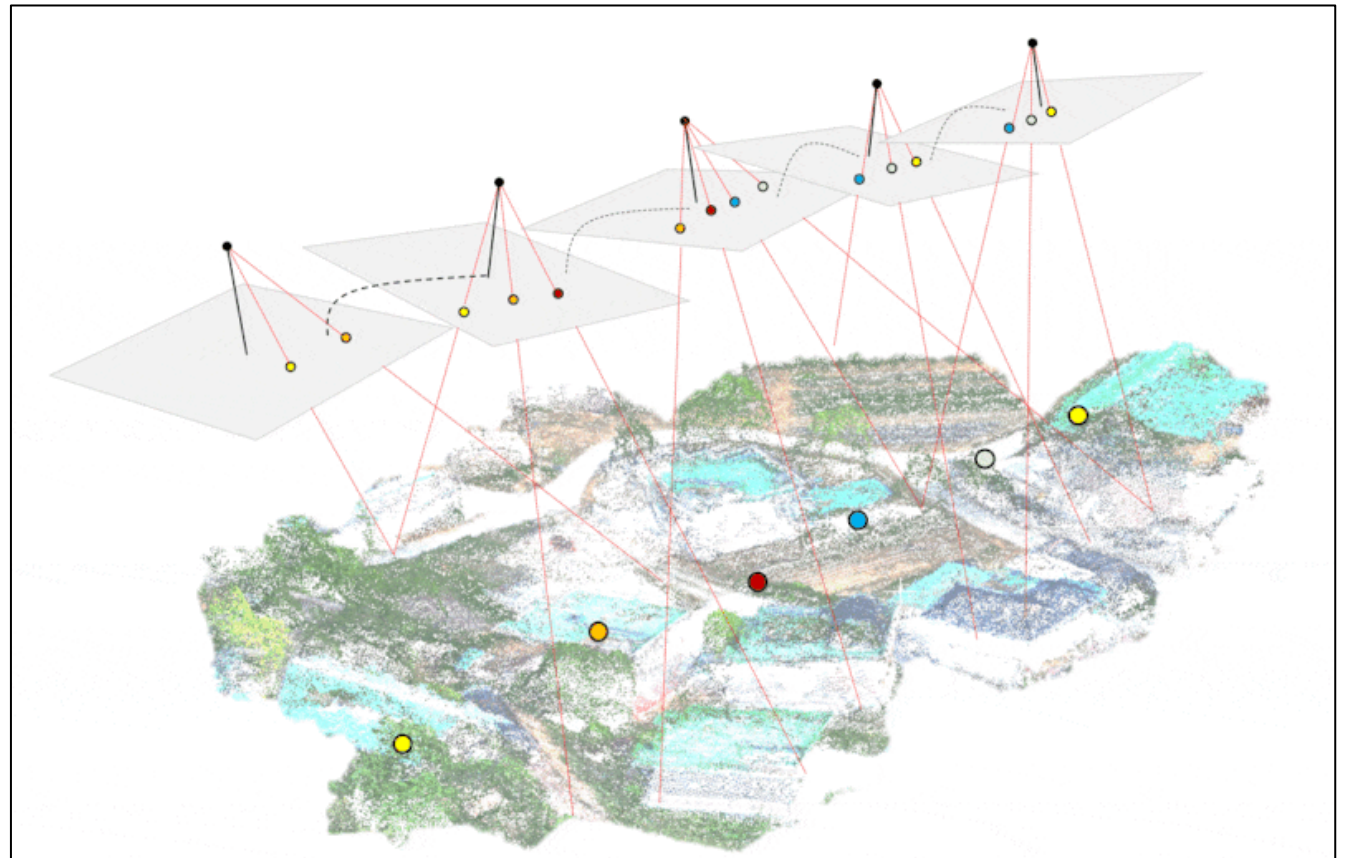
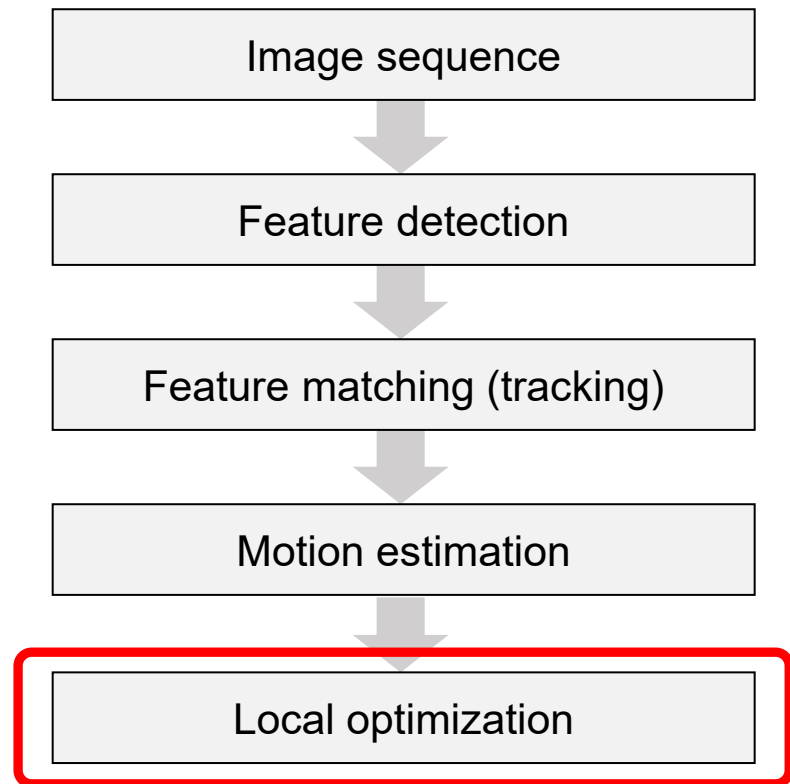
Example: Bundle Adjustment:

$$P^i, C_2, \dots, C_k = \operatorname{argmin}_{P^i, C_2, \dots, C_k} \sum_{k=1}^n \sum_{i=1}^N \|p_k^i - \pi(P^i, K_k, C_k)\|^2$$

Or Pose-Graph Optimization (see later)

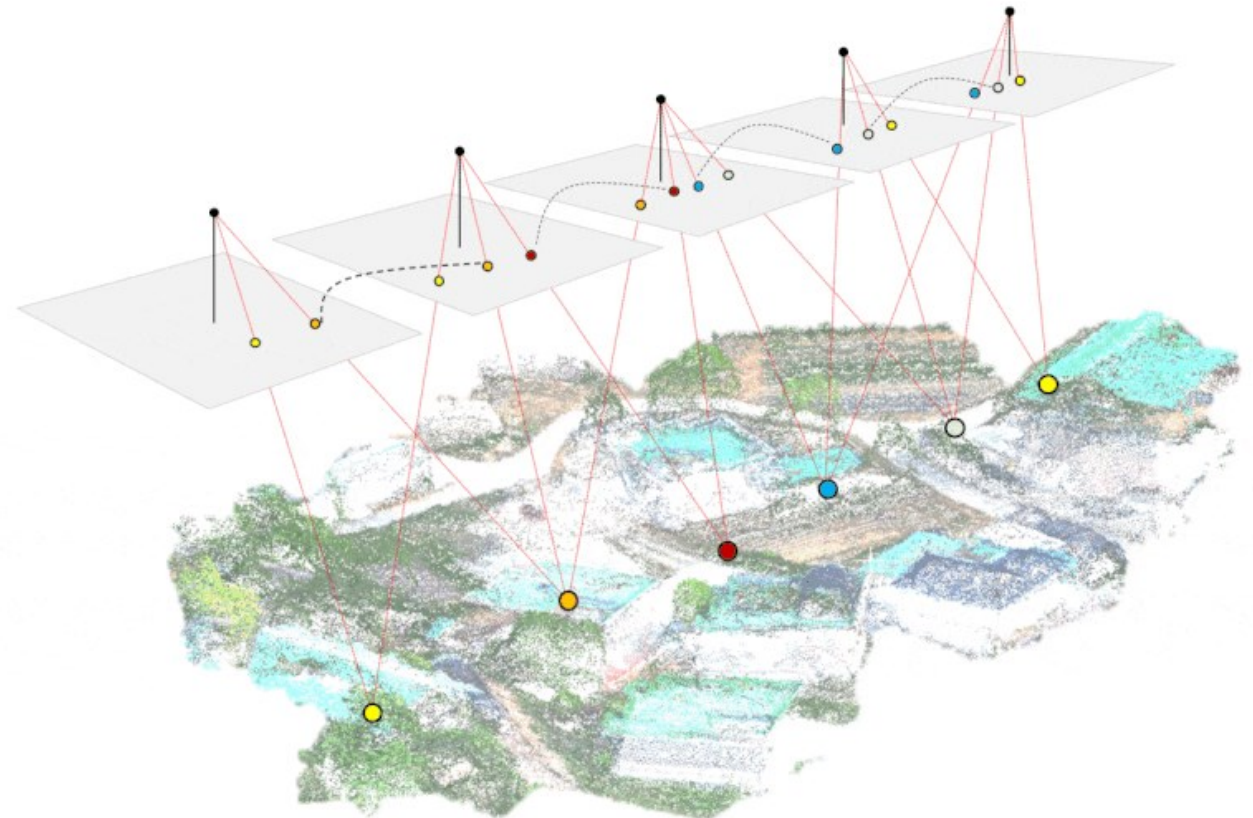
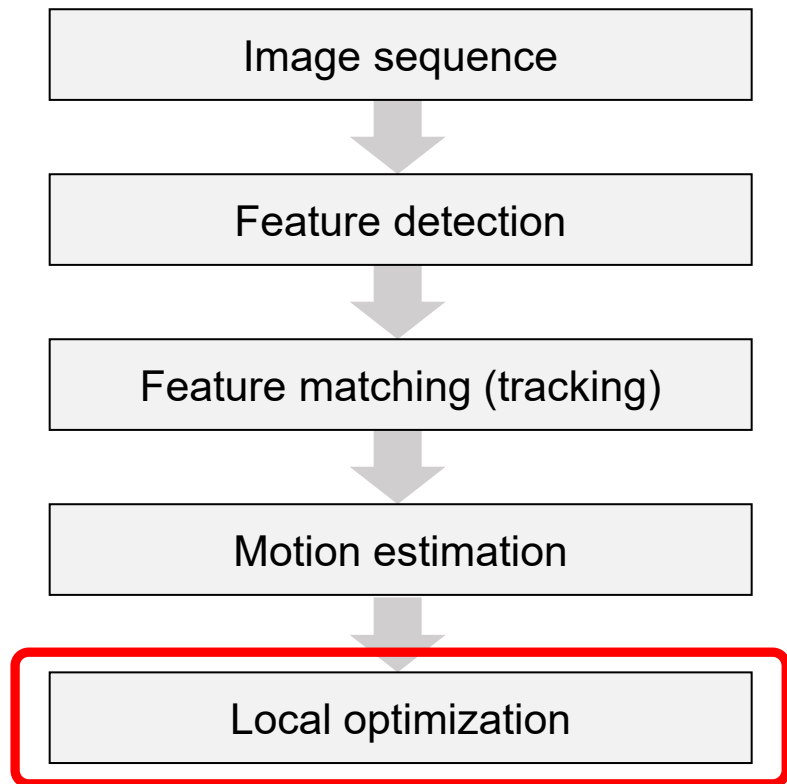
VO Flow Chart: review (Lecture 01)

VO computes the camera path incrementally (pose after pose)



VO Flow Chart: review (Lecture 01)

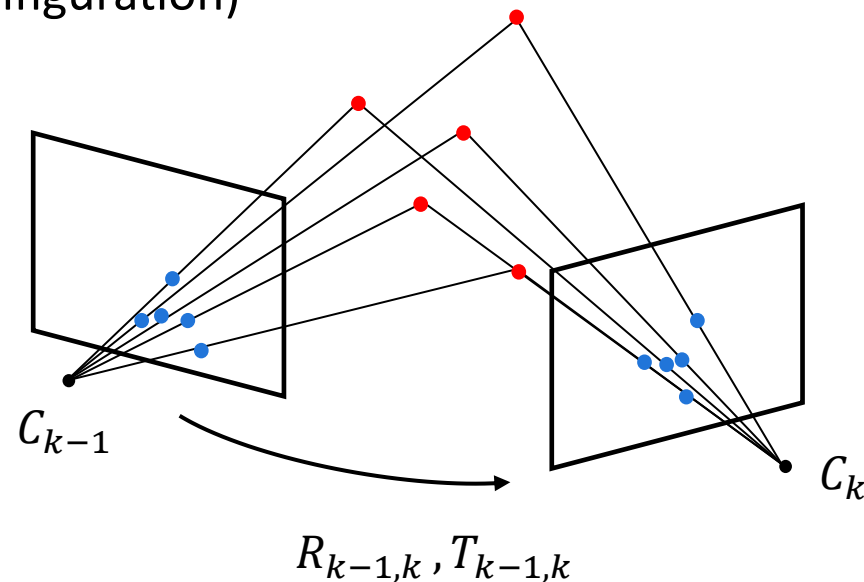
VO computes the camera path incrementally (pose after pose)



2D-to-2D (already seen: Lecture 08)

Motion from 2D-to-2D feature correspondences

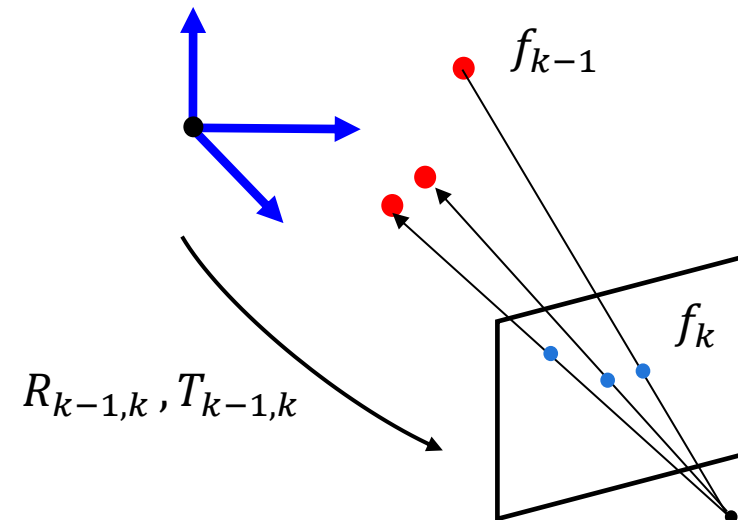
- Both feature correspondences f_{k-1} and f_k are specified in **image coordinates (2D)**
- The **minimal-case** solution involves **5** feature correspondences
- Popular algorithms:
 - **8-point algorithm** (NB: works only for **non-coplanar points** [slide 19 of Lecture 08])
 - **5-point algorithm** (works with any point configuration)



3D-to-2D (already seen: Lecture 03)

Motion from 3D-to-2D feature correspondences (i.e., Perspective from n Points: PnP problem)

- f_{k-1} is specified in 3D and f_k in 2D
- **Minimal case:**
 - DLT algorithm: minimal case: 6 points from 3D objects, or 4 from planar objects
 - P3P algorithm: minimal case: 3 points (+1 for disambiguation)
 - EPNP algorithm: for more than 4 points

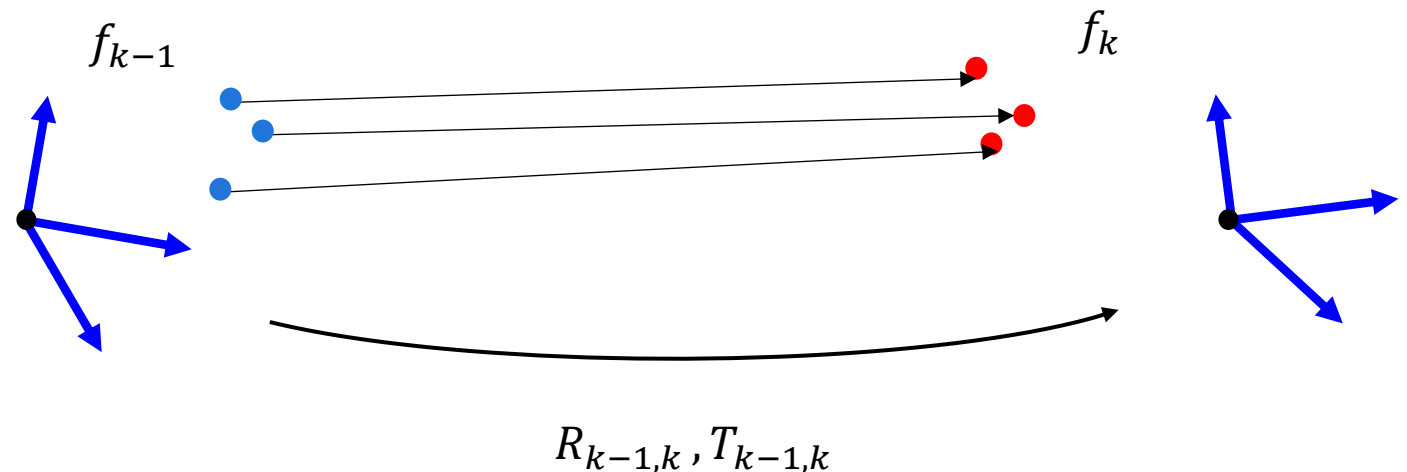


3D-to-3D

- **Motion from 3D-to-3D feature correspondences** (also known as point cloud registration problem)
- Both f_{k-1} and f_k are specified in 3D. To do this, it is necessary to first triangulate 3D points (e.g. use a stereo camera)
- The **minimal-case** solution involves **3 non-collinear** correspondences
- Popular algorithm: [Arun'87]
- Consists of solving the following system of equations with R and T as unknowns:

$$\begin{bmatrix} X^i_{k-1} \\ Y^i_{k-1} \\ Z^i_{k-1} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X^i_k \\ Y^i_k \\ Z^i_k \\ 1 \end{bmatrix}$$

where i is the feature ID.



3D-to-3D

- **Motion from 3D-to-3D feature correspondences** (also known as point cloud registration problem)
- Both f_{k-1} and f_k are specified in 3D. To do this, it is necessary to first triangulate 3D points (e.g. use a stereo camera)
- The **minimal-case** solution involves **3 non-collinear** correspondences
- Popular algorithm: [Arun'87]
- Consists of solving the following system of equations with R and T as unknowns:

$$\begin{bmatrix} X^i_{k-1} \\ Y^i_{k-1} \\ Z^i_{k-1} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X^i_k \\ Y^i_k \\ Z^i_k \\ 1 \end{bmatrix}$$

where i is the feature ID.



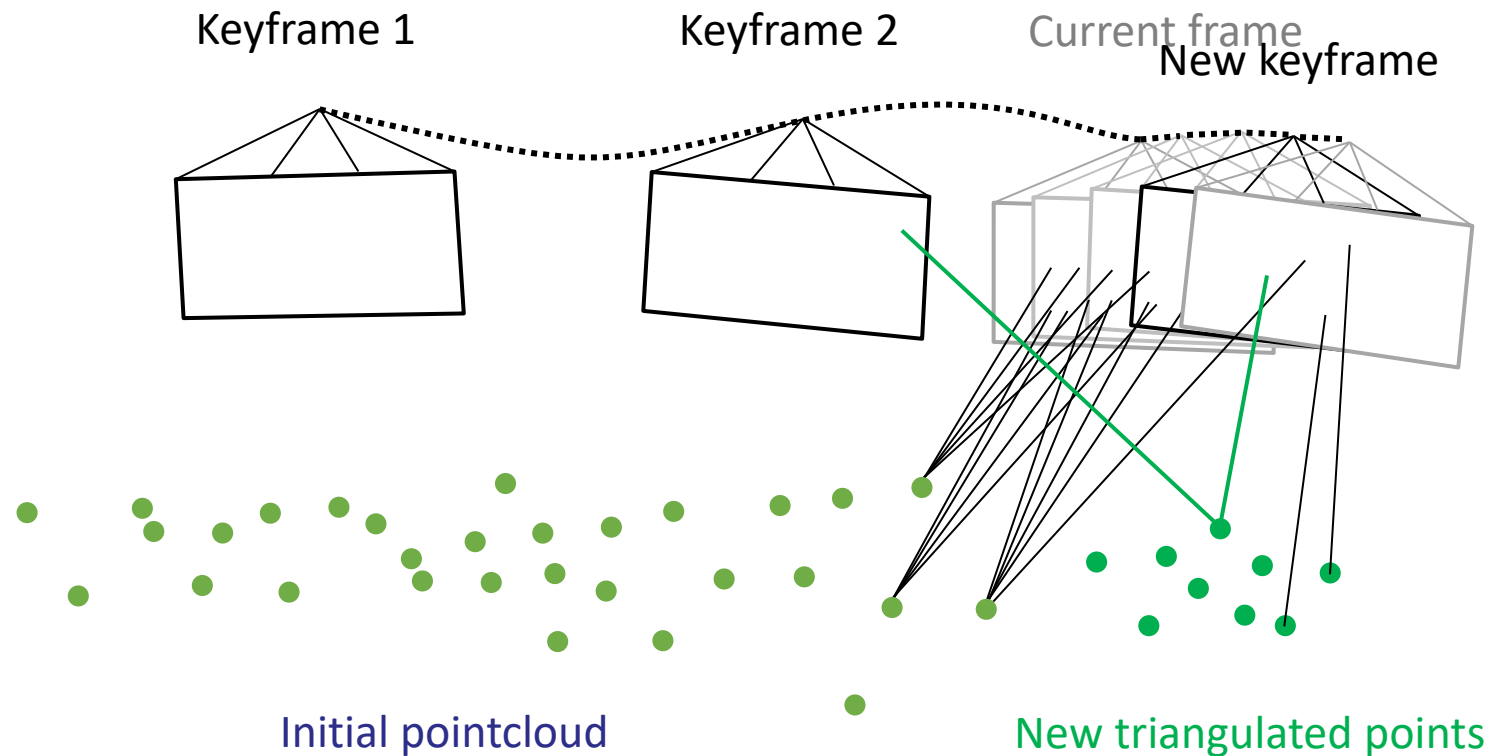
Motion Estimation: Recap

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

Which one provides higher accuracy for Stereo VO, 3D-2D or 3D-3D?

Case Study: Monocular VO (i.e., single camera VO)

This pipeline was initially proposed in PTAM (Parallel Tracking & Mapping) [Klein, ISMAR'07]



Klein, Murray, *Parallel Tracking and Mapping for Small AR Workspaces*, International Symposium on Mixed and Augmented Reality (ISMAR), 2007.

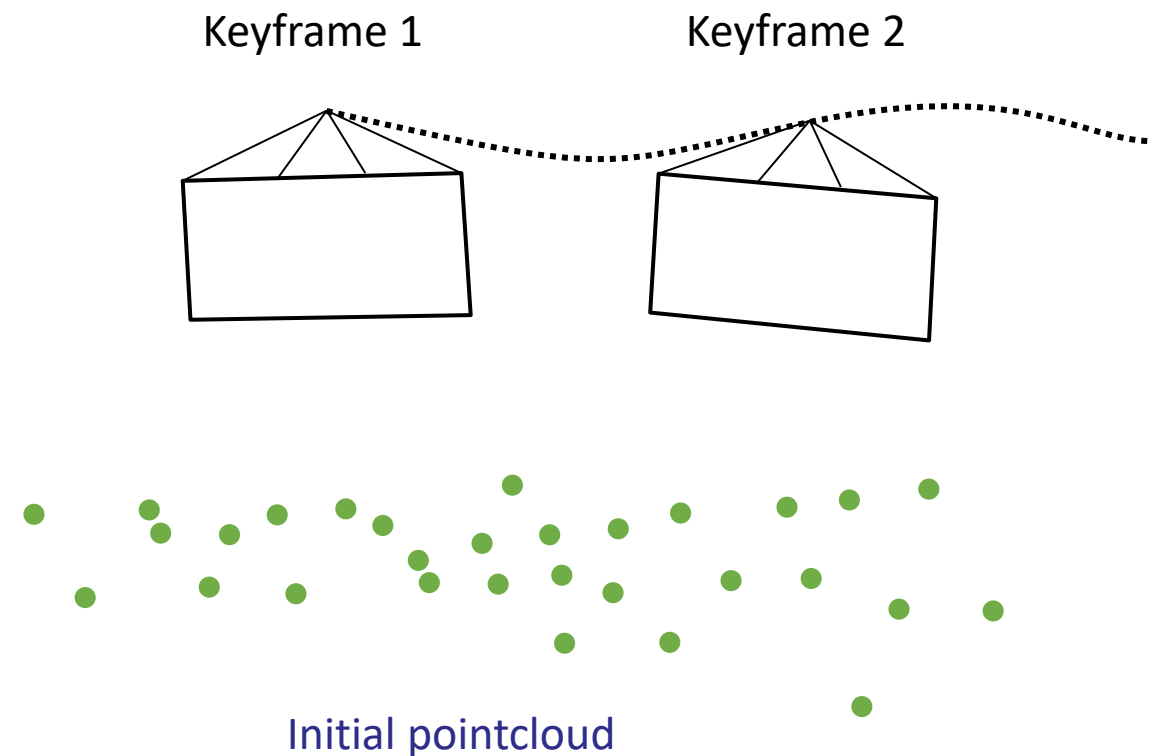
[PDF, code, videos](#). **Best paper award**

Case Study: Monocular VO (i.e., single camera VO)

1. Bootstrapping (i.e., initialization)

- Initialize structure and motion from 2 views: e.g., **5- or 8-point RANSAC**
- Refine structure and motion (**Bundle Adjustment**)
- **How far should the two frames (i.e., keyframes) be?**

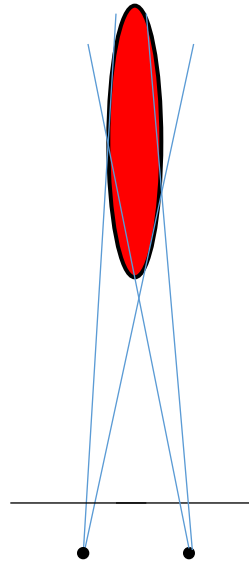
Motion estimation		
2D-2D	3D-2D	3D-3D



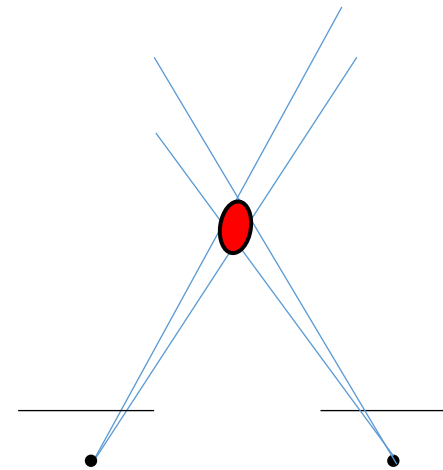
Case Study: Monocular VO (i.e., single camera VO)

2. Keyframe selection (i.e., skipping frames)

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



Small baseline \rightarrow large depth uncertainty

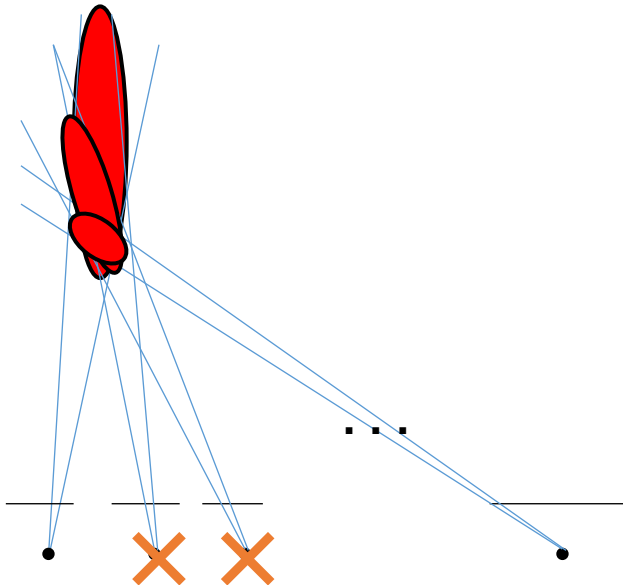


Large baseline \rightarrow small depth uncertainty

Case Study: Monocular VO (i.e., single camera VO)

2. Keyframe selection (i.e., skipping frames)

- When frames are taken at nearby positions compared to the scene distance, 3D points will exhibit large uncertainty
- One way to avoid this consists of **skipping frames** until the average uncertainty of the 3D points, normalized by the average distance from the scene, falls below a certain threshold. The selected frames are called **keyframes**
- **Rule of the thumb:** add a keyframe when $\frac{\text{keyframe distance}}{\text{average-depth}} > \text{threshold}$ (usually 10-20 %)



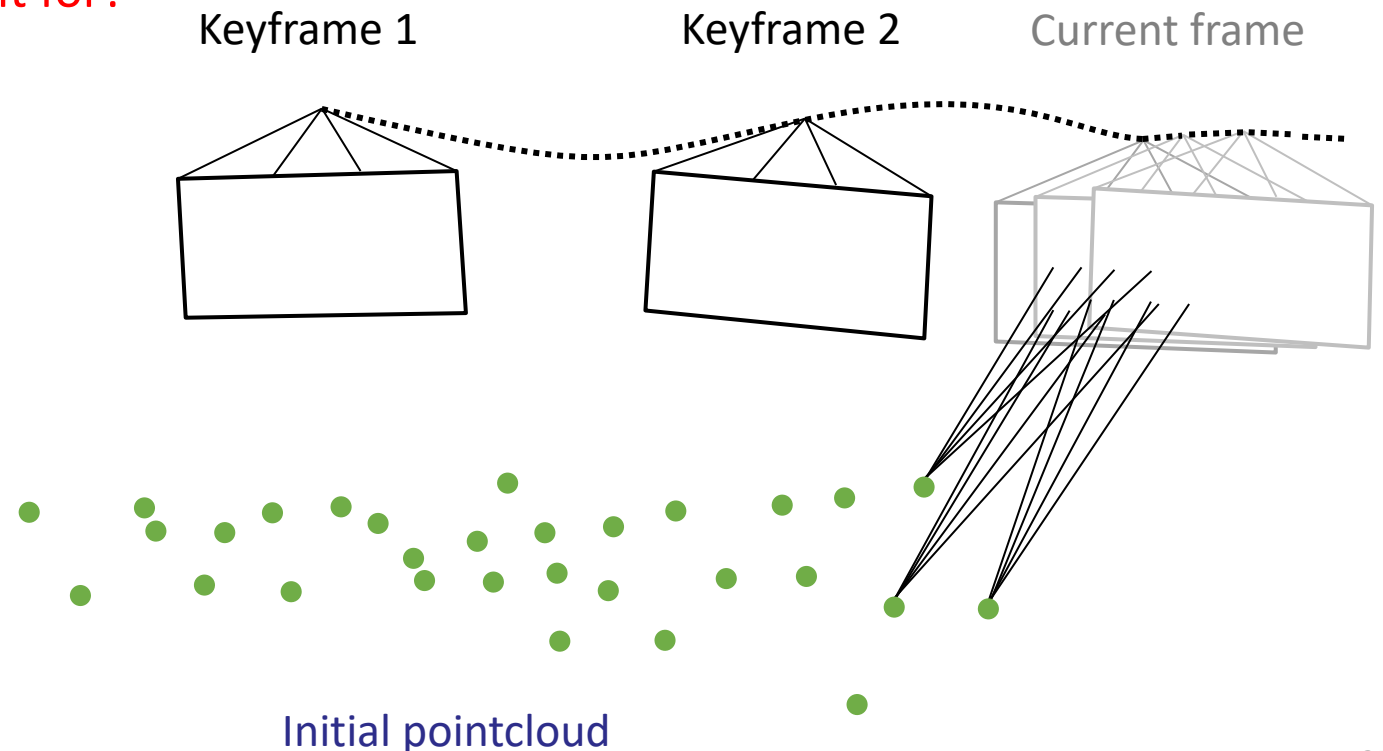
- Where does this come from?
- What about pure rotations?

Case Study: Monocular VO (i.e., single camera VO)

3. Localization (i.e., pose estimation from a given point cloud)

- Given a 3D point cloud (map), determine the pose of each additional view
- What algorithm is used?
- How far from the last keyframe can we use it for?

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

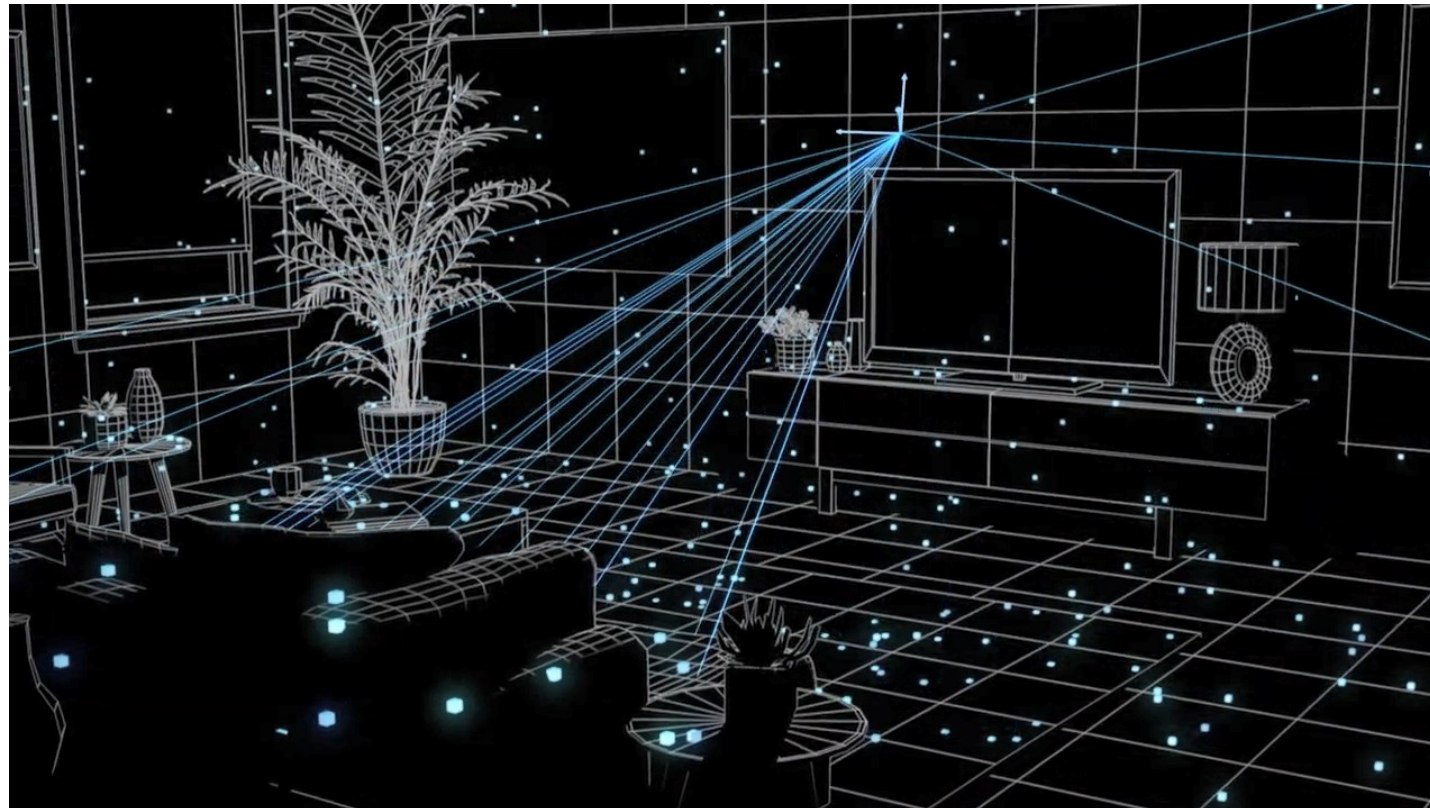


Case Study: Monocular VO (i.e., single camera VO)

3. Localization (i.e., pose estimation from a given point cloud)

- Given a 3D point cloud (map), determine the pose of each additional view

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

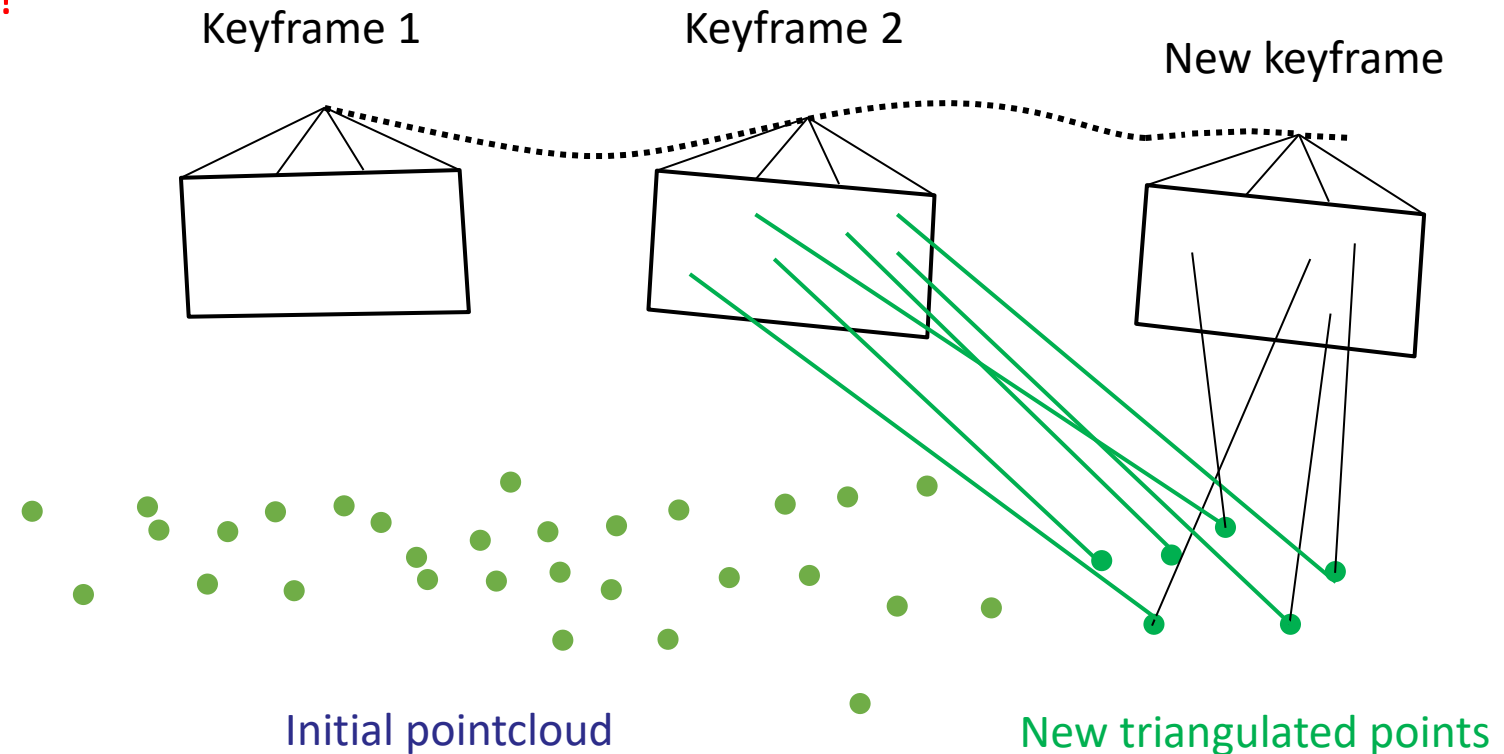


[Video](#) of Oculus Insight (the VIO used in Oculus Quest): built by former [Zurich-Eye team](#), today Meta Zurich.

Case Study: Monocular VO (i.e., single camera VO)

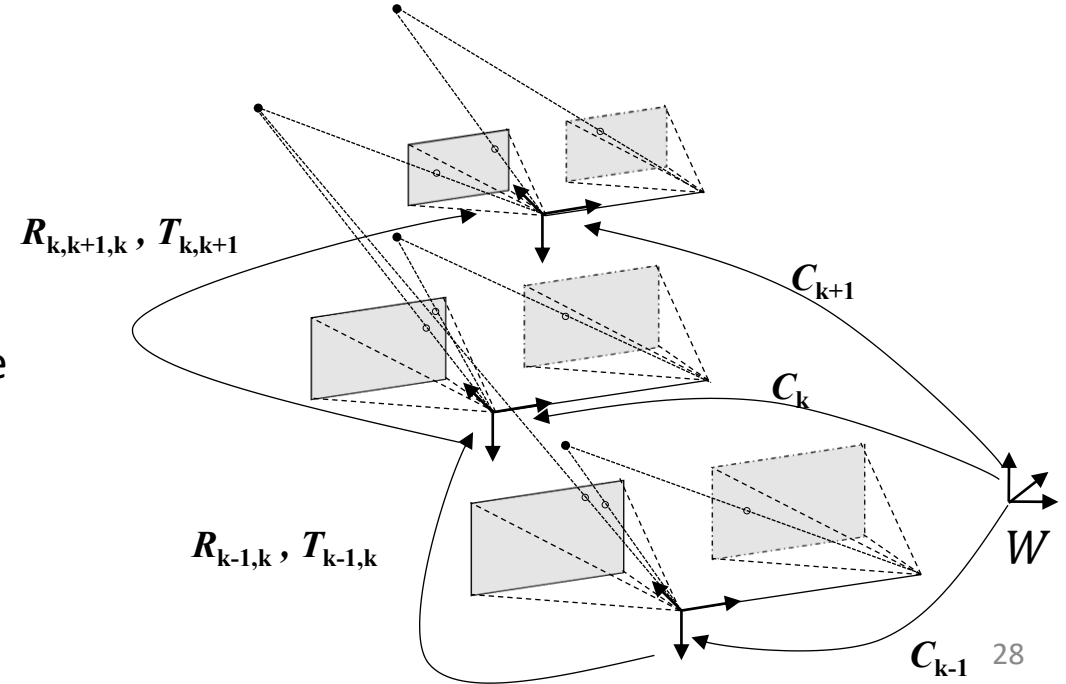
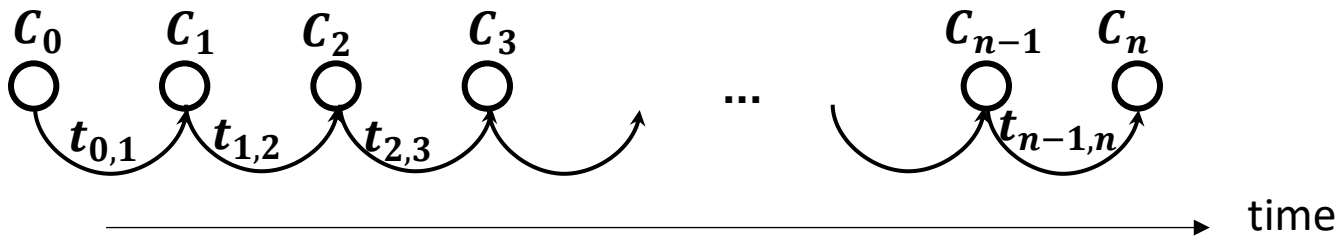
4. Extend Structure (i.e., mapping)

- Extract and triangulate new features
- Is it necessary to do this at every frame or can we just do it at keyframes?
- What are the pros and cons?
- What about pure rotations?



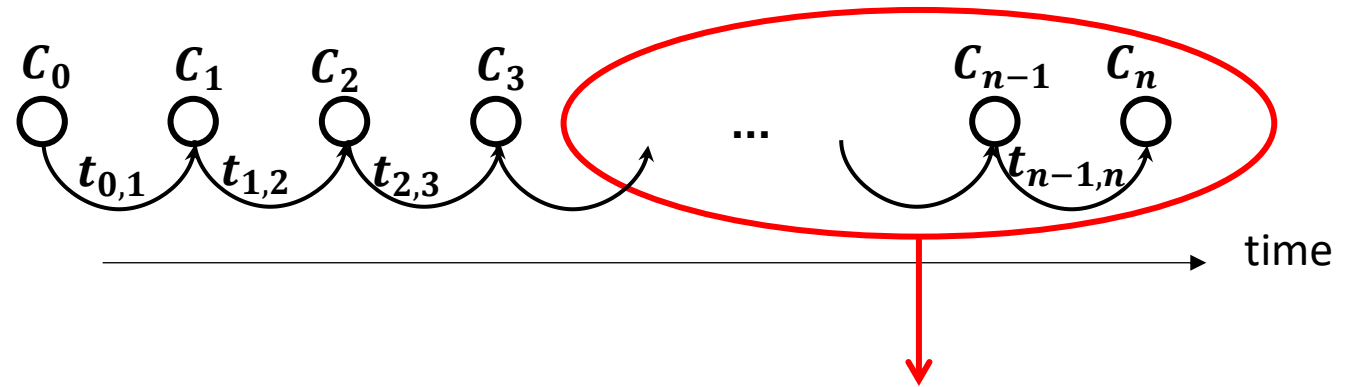
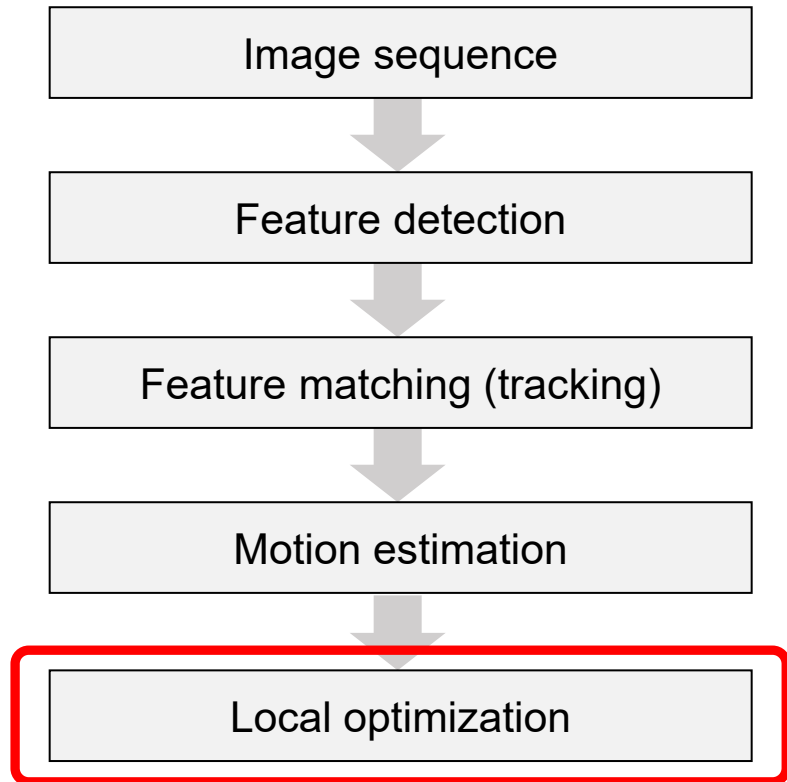
VO: putting all pieces together

- Let the **relative motion** between image I_{k-1} and image I_k be: $t_{k-1,k} = \begin{bmatrix} R_{k-1,k} & T_{k-1,k} \\ 0 & 1 \end{bmatrix}$
- Let C_{k-1} be the **previous camera pose in the world reference frame**
- Then, the **current pose C_k in the world frame** is given by: $C_k = C_{k-1}t_{k-1,k}$



Local Optimization

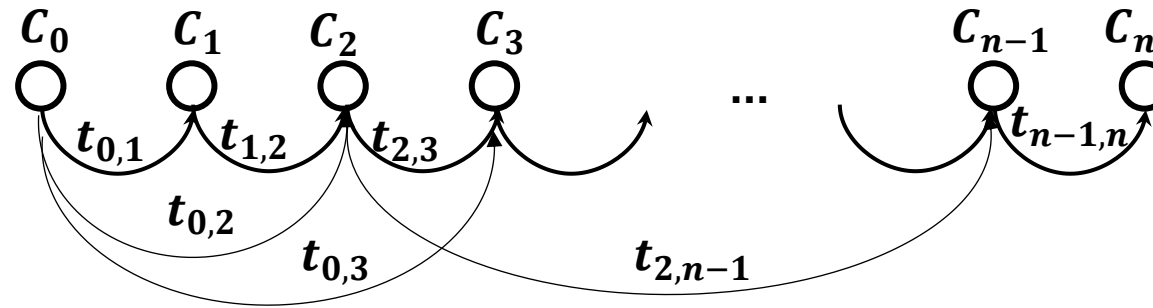
VO flowchart:



Sliding-window bundle adjustment
or Pose-Graph Optimization (see next slide)

Pose-Graph Optimization

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

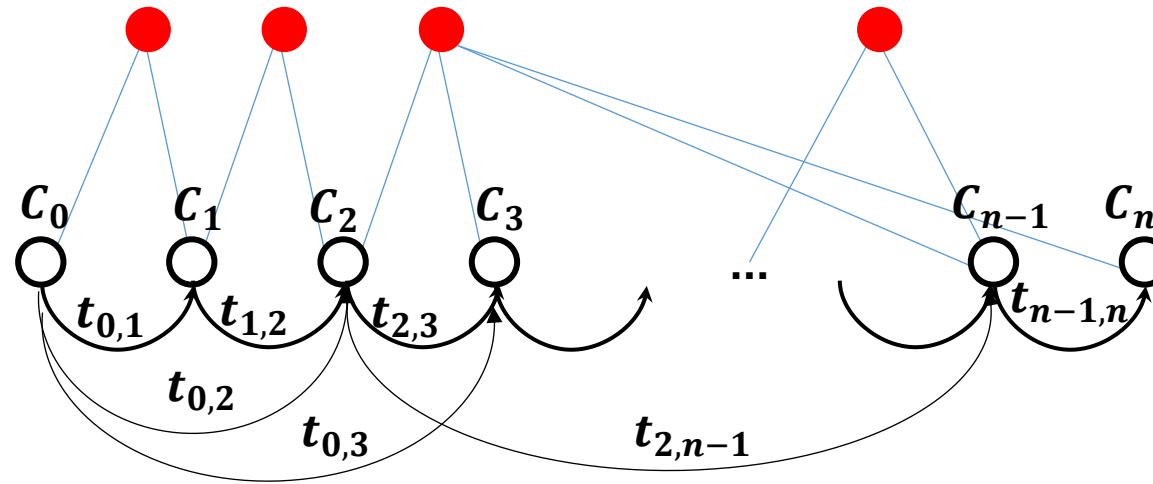


- However, transformations can also be computed between **non-adjacent frames**: $t_{j,i}$ (e.g., when features from previous keyframes are still observed). They can be used as additional constraints to improve camera poses by solving:

$$\{C_1, \dots, C_n\} = \operatorname{argmin}_{\{C_1, \dots, C_n\}} \sum_i \sum_j \|C_i - C_j t_{j,i}\|^2$$

- For efficiency, only the last m keyframes are used (Sliding-Window Pose Graph Optimization)
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools exist: [g2o](#), [GTSAM](#), [SLAM++](#), [Google Ceres](#)

Bundle Adjustment (BA)



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

$$P^i, C_1, \dots, C_n = \underset{P^i, C_1, \dots, C_n}{\operatorname{argmin}} \sum_{k=1}^n \sum_{i=1}^N \rho \left(p_k^i - \pi(P^i, K_k, C_k) \right)$$

- $\rho()$ is the **Huber or Tukey norm**
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools exist: [g2o](#), [GTSAM](#), [SLAM++](#), [Google Ceres](#)

Bundle Adjustment vs Pose-graph Optimization

- BA is **more precise** than pose-graph optimization because it adds additional constraints (*landmark constraints*)
- But **more costly**: $O((qN + lm)^3)$ with N being the number of points, m the number of cameras poses and q and l the number of parameters for points and camera poses. Workarounds:
 - **Sliding-window BA**: A **small window size** limits the number of parameters for the optimization and thus makes real-time bundle adjustment possible.
 - **Motion-only BA**: It is possible to reduce the computational complexity by just optimizing over the camera parameters and keeping the 3-D landmarks fixed

More efficient BA algorithms have recently been developed:

[1] Demmel, Schubert, Sommer, Cremers, Usenko, Square Root Marginalization for Sliding-Window Bundle Adjustment, IEEE International Conference on Computer Vision (ICCV), 2021. [Paper](#), [Video](#), [Code](#).

[2] Demmel, Sommer, Cremers, Usenko, Square Root Bundle Adjustment for Large-Scale Reconstruction, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021. [Paper](#), [Video](#), [Code](#).

Place Recognition

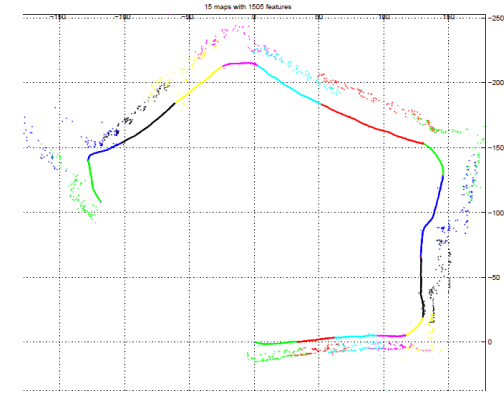
During VO, two problems can occur:

- **Relocalization problem:** camera pose estimation can fail due to:
 1. Feature **tracking can be lost** (due to occlusions, low texture, quick motion, illumination change)
 2. In case of monocular VO: **pure rotation followed by translation** (**why?**)
→ **Solution: Re-localize** camera pose and continue or use other sensors (more cameras or inertial sensors)
- **Loop closing problem**
 - When you go back to a previously mapped area:
 - **Loop closure detection:** to avoid map duplication
 - **Loop correction (or loop closing):** to compensate the accumulated drift
 - In both cases you need a place recognition technique

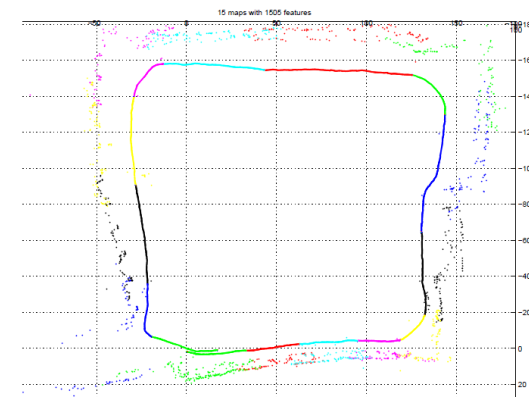
We will address place recognition in Lecture 12

VO vs. Visual SLAM (recap from Lecture 01)

- **Visual Odometry**
 - Focuses on incremental motion estimation
 - **Guarantees local consistency** (i.e., estimated trajectory is locally correct, but not globally, i.e. from the start to the end)
- **Visual SLAM** (Simultaneous Localization And Mapping)
 - **SLAM = visual odometry + loop detection & loop closing**
 - **Guarantees global consistency** (the estimated trajectory is globally correct, i.e. from the start to the end)



Visual odometry



Visual SLAM

Image courtesy of [Clemente et al., RSS'07]

Open Source Monocular VO and SLAM algorithms

- PTAM
- ORB-SLAM
- SVO
- LSD-SLAM
- DSO

Indirect methods: Minimize the feature reprojection error

Direct methods: Minimize the feature photometric error

PTAM: Parallel Tracking and Mapping

- Monocular only
- **Feature based**
 - **FAST corners + patch descriptors**
 - **Minimizes reprojection error**
 - **Jointly optimizes poses & structure** (sliding window BA)
- First to propose **keyframe-based VO**
- First to propose **localization** (i.e., camera tracking) and **mapping** running in **two independent threads**: updated map is used by localization thread asynchronously, as soon as it becomes available
- Includes:
 - Relocalization only in a small neighborhood
 - No global optimization, only local
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

Parallel Tracking and Mapping
for Small AR Workspaces

ISMAR 2007 video results

Georg Klein and David Murray
Active Vision Laboratory
University of Oxford

Klein, Murray, *Parallel Tracking and Mapping for Small AR Workspaces*, International Symposium on Mixed and Augmented Reality (ISMAR), 2007.

[PDF, code, videos](#). **Best paper award**

PTAM: Parallel Tracking and Mapping

- Monocular only
- **Feature based**
 - FAST corners + patch descriptors
 - Minimizes reprojection error
 - Jointly optimizes poses & structure (sliding window BA)
- First to propose **keyframe-based VO**
- First to propose **localization** (i.e., camera tracking) and **mapping** running in **two independent threads**: updated map is used by localization thread asynchronously, as soon as it becomes available
- Includes:
 - Relocalization only in a small neighborhood
 - No global optimization, only local
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

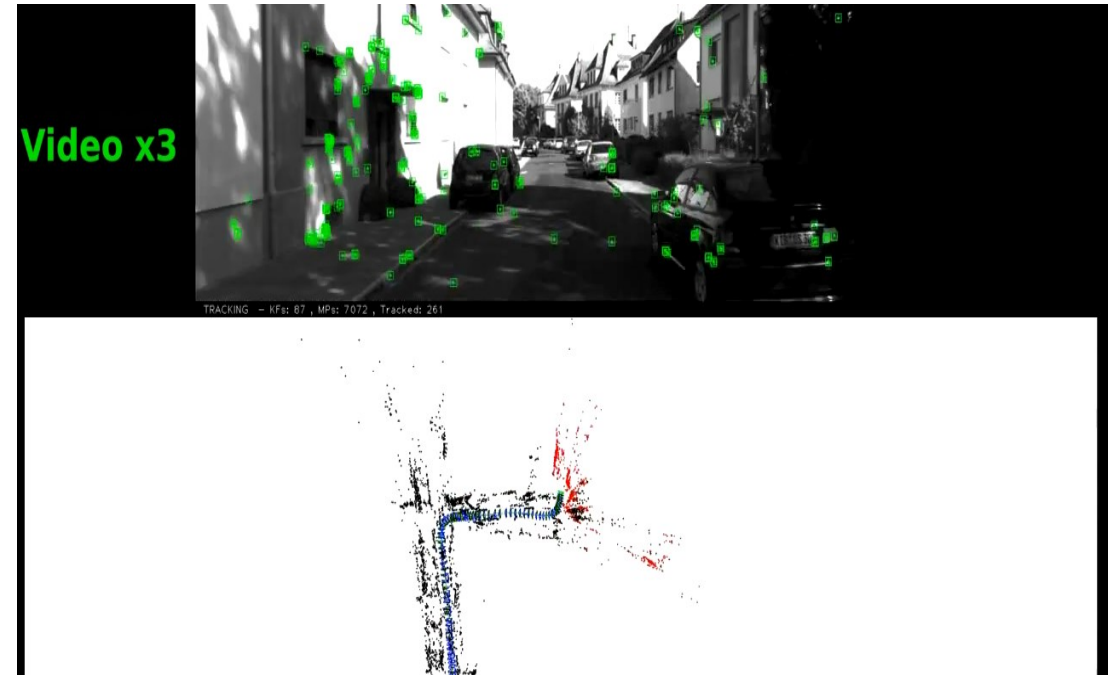


Klein, Murray, *Parallel Tracking and Mapping for Small AR Workspaces*, International Symposium on Mixed and Augmented Reality (ISMAR), 2007.

[PDF, code, videos](#). **Best paper award**

ORB-SLAM

- Supports both **monocular and stereo** cameras
- **Feature based**
 - **FAST corners + ORB descriptors** (recall: ORB is a binary descriptor, thus very fast to compute and match (Hamming distance))
 - **Minimizes reprojection error**
 - **Jointly optimizes poses & structure** (sliding window BA)
- **Same workflow as PTAM** (keyframe based, parallel localization and mapping as independent threads)
- Includes:
 - **Relocalization**
 - **Final optimization**
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

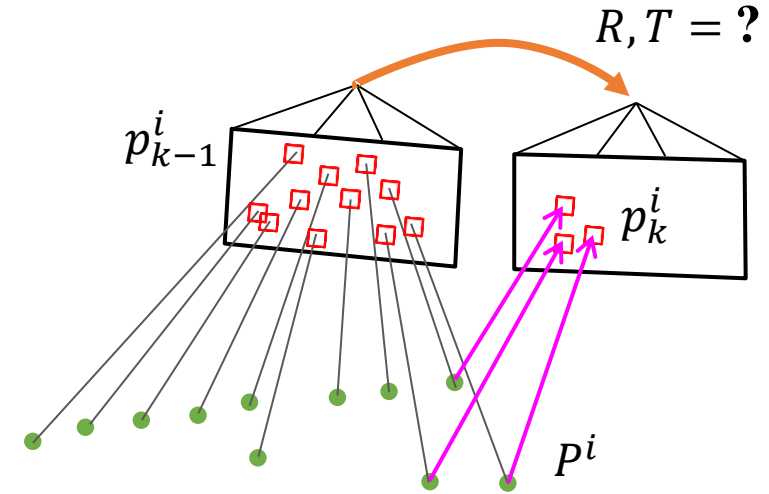


Indirect vs Direct Methods

- **Indirect methods**

1. Extract & match features + 3-point RANSAC
2. Bundle Adjust by minimizing the **Reprojection Error**:

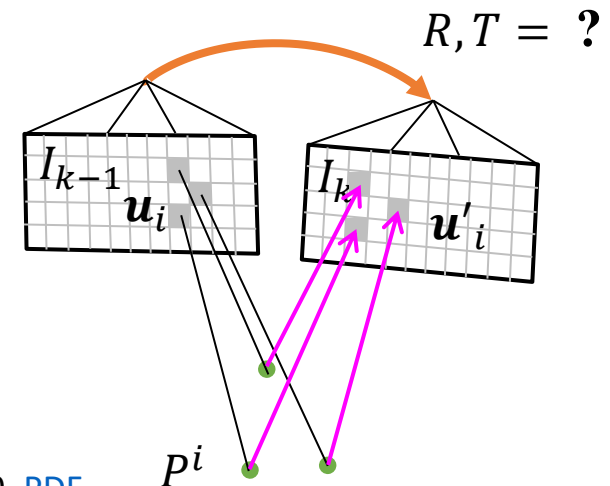
$$P^i, R, T = \arg \min_{P^i, R, T} \sum_{i=1}^N \rho(p_k^i - \pi(P^i, K, R, T))$$



- **Direct methods**

1. No feature extraction, no matching, no RANSAC needed
Instead, directly minimize **Photometric Error**:

$$P^i, R, T = \arg \min_{P^i, R, T} \sum_{i=1}^N \rho(I_{k-1}(p_{k-1}^i) - I_k(\pi(P^i, K, R, T)))$$



What are their pros and cons?

Indirect vs Direct Methods

• Indirect methods

1. Extract & match features + 3-point RANSAC
2. Bundle Adjust by minimizing the **Reprojection Error**:

$$P^i, R, T = \arg \min_{P^i, R, T} \sum_{i=1}^N \rho \left(p_k^i - \pi(P^i, K, R, T) \right)$$

• Direct methods

1. No feature extraction, no matching, no RANSAC needed
Instead, directly minimize **Photometric Error**:

$$P^i, R, T = \arg \min_{P^i, R, T} \sum_{i=1}^N \rho \left(I_{k-1}(p_{k-1}^i) - I_k \left(\pi(P^i, K, R, T) \right) \right)$$

- ✓ Can cope with large frame-to-frame motions (large basin of convergence)
- ✗ Slow due to costly feature extraction, matching, and outlier removal (e.g., RANSAC)

- ✓ All image pixels can in principle be used (higher accuracy, higher robustness to motion blur and weak texture (i.e., weak gradients))
- ✓ Increasing the camera frame-rate reduces computational cost per frame (no RANSAC needed)
- ✗ Very sensitive to initial value → limited frame-to-frame motion (small basin of convergence)

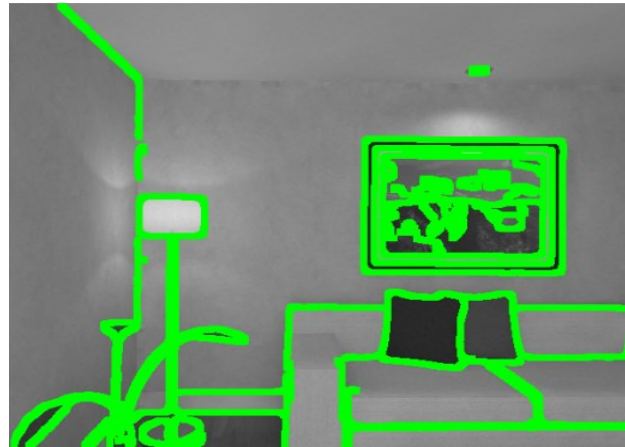
Direct Methods: Dense, Semi-dense, Sparse

Dense methods
track every pixel



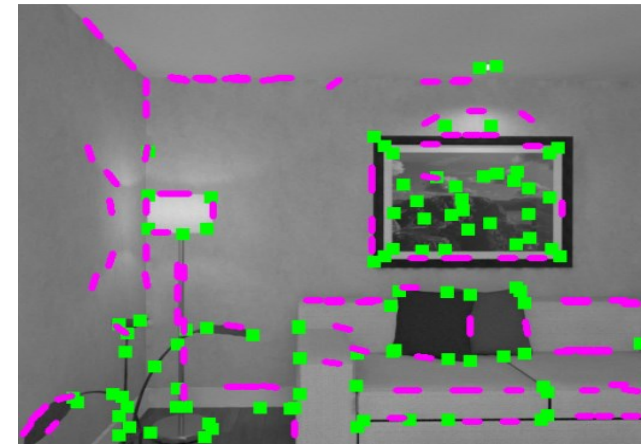
In a VGA image: 300'000+ pixels

Semi-Dense methods
track only edges



In a VGA image: ~10,000 pixels

Sparse methods
track sparse pixels

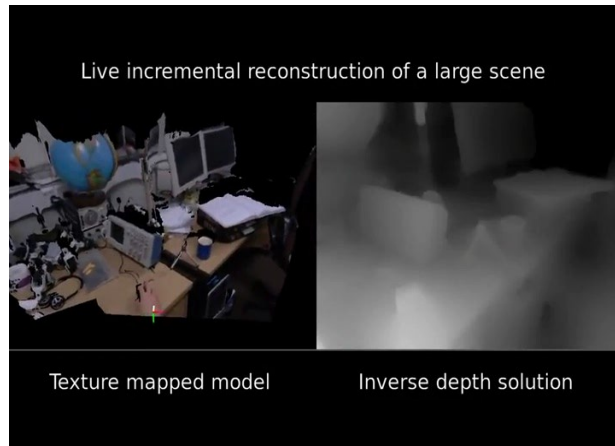


In a VGA image: ~2,000 pixels

Forster, Zhang, Gassner, Werlberger, Scaramuzza, *SVO: Semi Direct Visual Odometry for Monocular and Multi-Camera Systems*, IEEE Transactions on Robotics (T-RO), 2017. [PDF.](#)]

Direct Methods: Dense, Semi-dense, Sparse

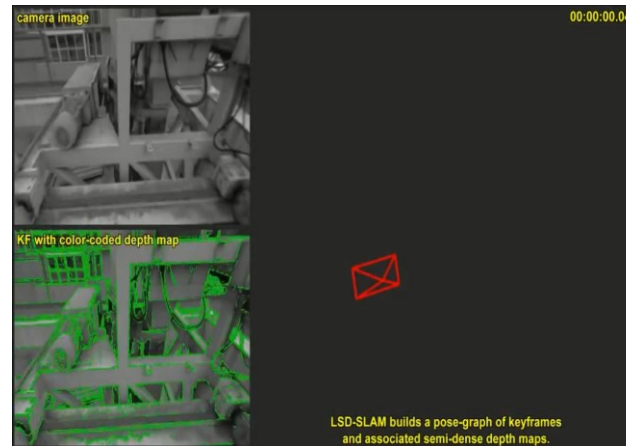
Dense methods
track every pixel



In a VGA image: 300'000+ pixels

DTAM [Newcombe '11], REMODE [Pizzoli'14]

Semi-Dense methods
track only edges



In a VGA image: ~10,000 pixels

LSD-SLAM [Engel'14]

Sparse methods
track sparse pixels

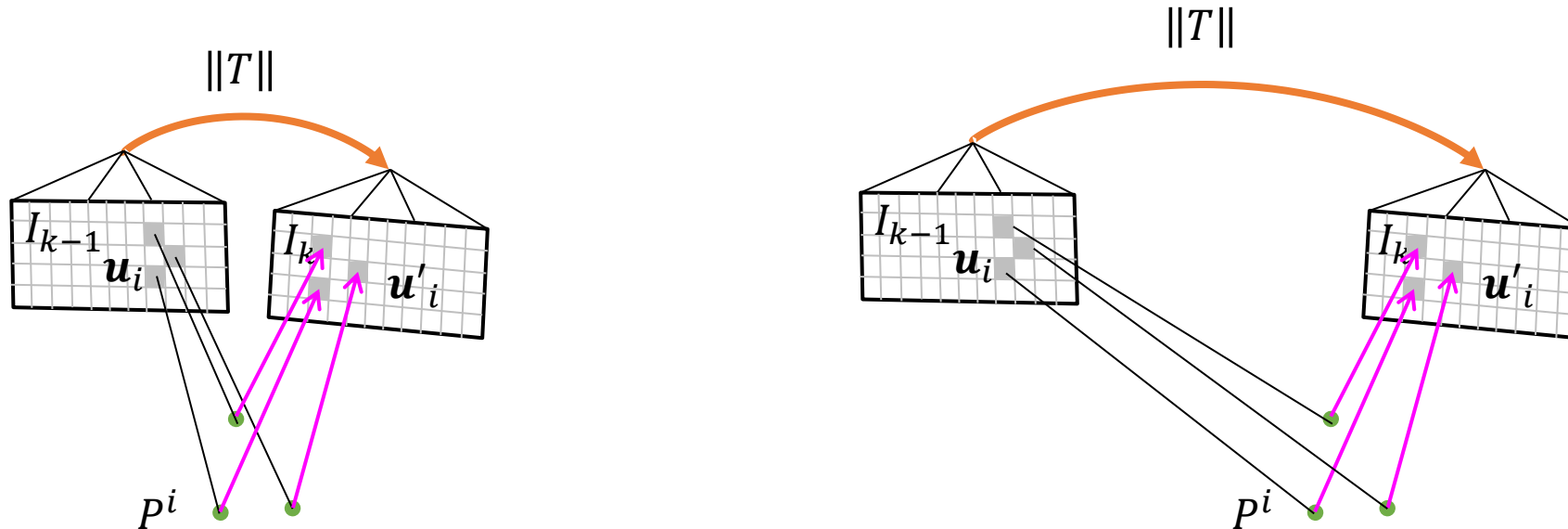


In a VGA image: ~2,000 pixels
e.g., 120 feature patches × (4×4 pixels per patch)

SVO [Forster'14], DSO [Engel'17]

Direct Methods: Dense, Semi-dense, Sparse

- What is the influence of the motion baseline on the convergence rate of direct methods?



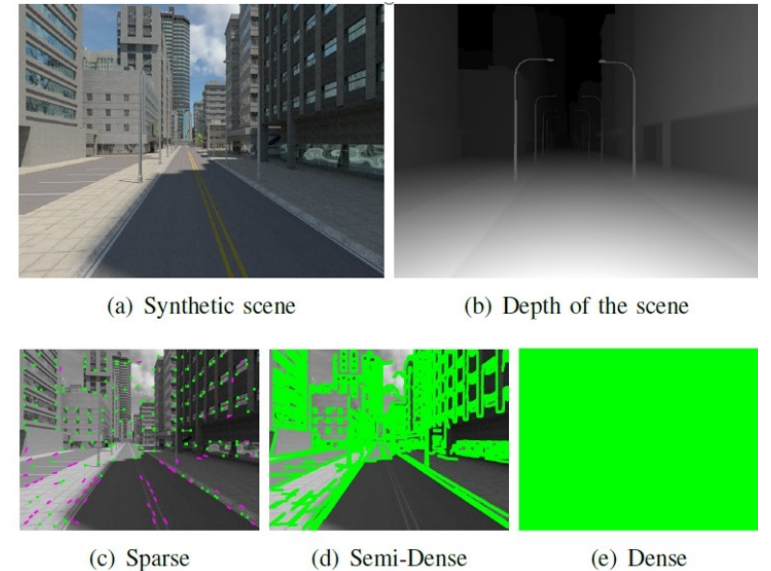
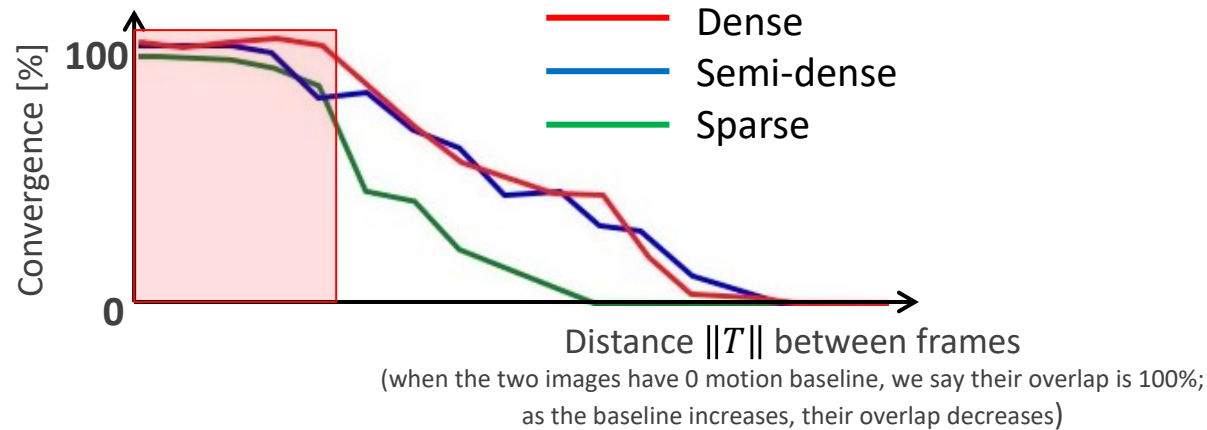
For **small motion** baselines, $\|T\|$,
the **photometric error is usually small**

For **large motion** baselines, $\|T\|$,
the **photometric error is usually large**
(due to large geometric and illumination changes)

Forster, Zhang, Gassner, Werlberger, Scaramuzza, *SVO: Semi Direct Visual Odometry for Monocular and Multi-Camera Systems*,
IEEE Transactions on Robotics (T-RO), 2017. [PDF.](#)]

What is the influence of the motion baseline on the convergence rate of direct methods?

We can use **photorealistic simulation** to answer this question by generating thousands of data



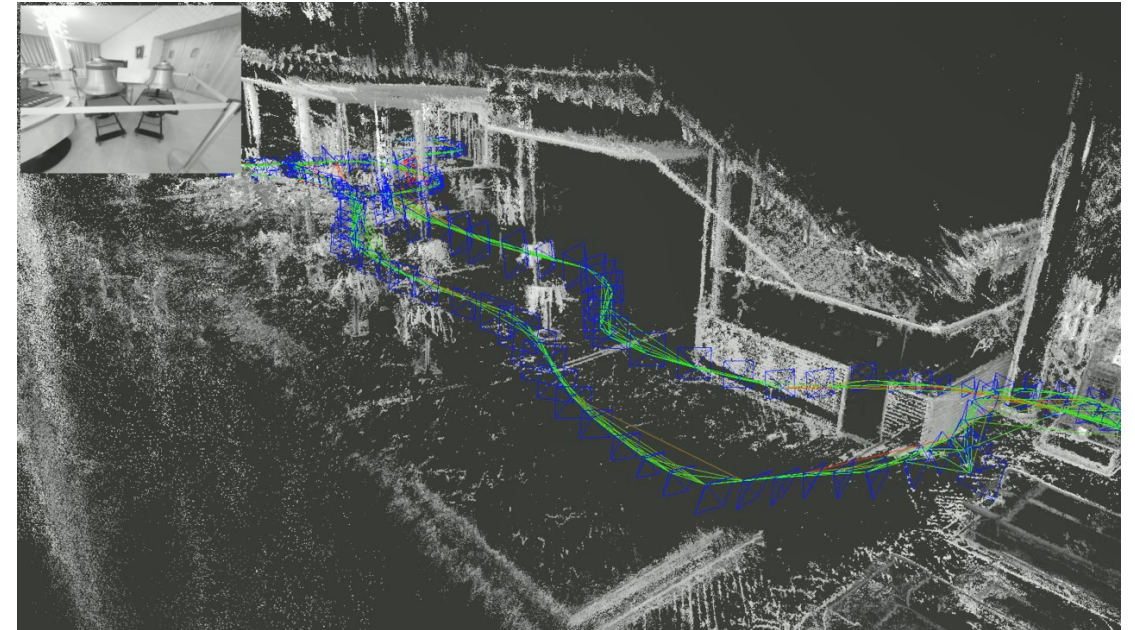
Simulated dataset from [here](#)

- Findings:

- **Dense and Semi-dense** behave similarly, thus **weak gradients are not informative** for the optimization
- **Dense methods** are only useful **with motion blur, defocus, and weak- texture** regions
- **Sparse methods behave equally well as dense or semi-dense methods** for small motion baselines

LSD-SLAM

- Supports both **monocular and stereo** cameras
- **Direct** (photometric error) + **Semi-Dense** formulation
 - **3D structure** represented as **semi-dense** depth map
 - Minimizes **photometric error**
 - **Separately** optimizes poses & structure (sliding window)
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- Includes:
 - **Loop closing**
 - **Relocalization**
 - **Final optimization**
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

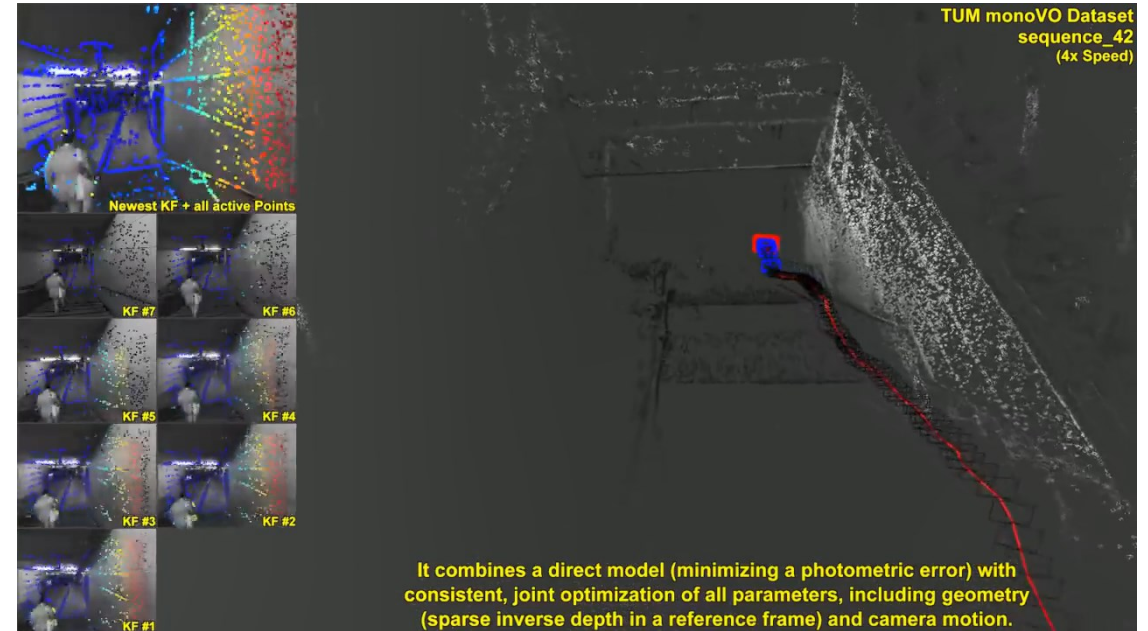


DSO

- Supports both **monocular and stereo** cameras
- **Direct** (photometric error) + **Sparse** formulation
 - **3D structure** represented as **sparse large gradients'** depth map
 - Minimizes **photometric error**
 - **Jointly optimizes poses & structure** (sliding window)
 - Incorporates photometric correction to compensate exposure time change ($\Delta t_{k-1}, \Delta t_k$)

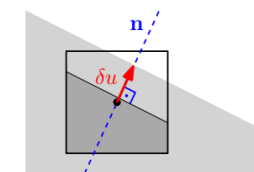
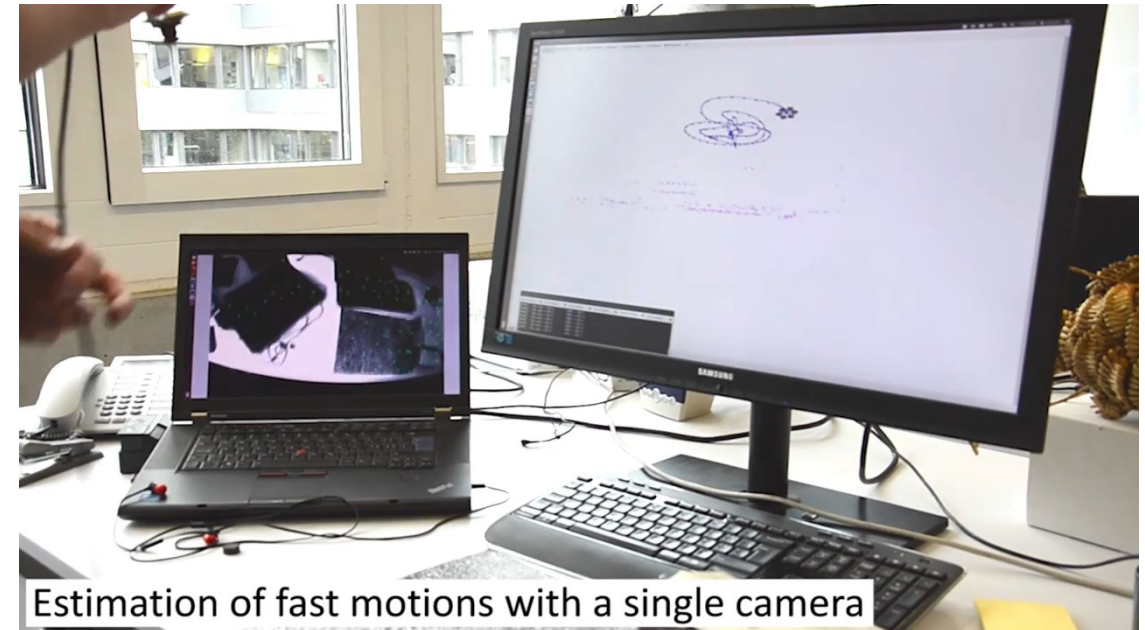
$$P^i, R, K = \arg \min_{P^i, R, K} \sum_{i=1}^N \rho \left(I_{k-1}(p_{k-1}^i) - \frac{\Delta t_{k-1}}{\Delta t_k} I_k \left(\pi(P^i, K, R, T) \right) \right)$$

- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

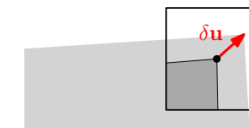


SVO

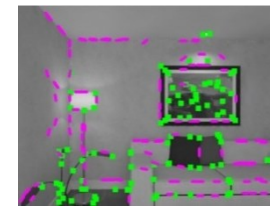
- Supports both **monocular, stereo, multi-camera systems** as well as omnidirectional models (fisheye and catadioptric)
- Combines **indirect + direct methods**
 - **Direct methods** for **frame-to-frame motion estimation**
 - **Indirect methods** for **frame-to-keyframe pose refinement**
- **Mapping**
 - **Probabilistic depth** estimation (heavy-tail Gaussian distribution)
- **Includes:**
 - **Loop closing,**
 - **Relocalization,**
 - **Final optimization**
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- **Faster than real-time: up to 400 fps** on i7 laptops and **100 fps** on smartphone PCs (Odroid (ARM)) or NVIDIA Jetson



Edgelet

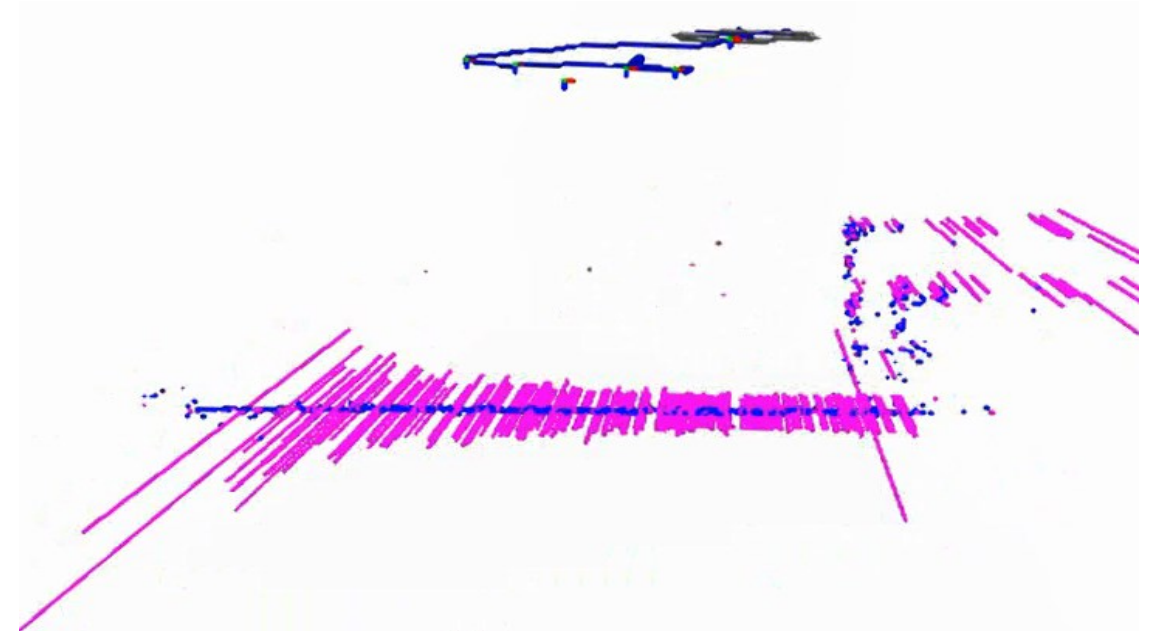


Corner

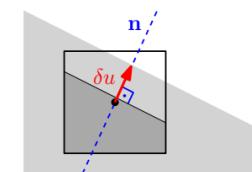


SVO

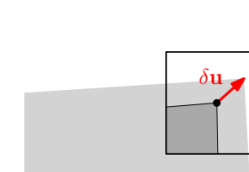
- Supports both **monocular, stereo, multi-camera systems** as well as omnidirectional models (fisheye and catadioptric)
- Combines **indirect + direct methods**
 - **Direct** methods for **frame-to-frame motion estimation**
 - **Indirect** methods for **frame-to-keyframe pose refinement**
- **Mapping**
 - **Probabilistic depth** estimation (heavy-tail Gaussian distribution)
- **Includes:**
 - **Loop closing,**
 - **Relocalization,**
 - **Final optimization**
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- **Faster than real-time: up to 400 fps** on i7 laptops and **100 fps** on smartphone PCs (Odroid (ARM)) or NVIDIA Jetson



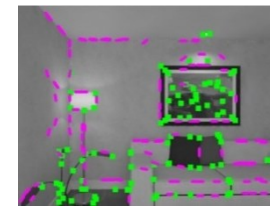
Probabilistic Depth Estimation



Edgelet



Corner



Processing times of ORB-SLAM, LSD-SLAM, DSO, SVO

	Mean	CPU@20 fps
SVO Mono	2.53	55 ±10%
ORB Mono SLAM (No loop closure)	29.81	187 ±32%
LSD Mono SLAM (No loop closure)	23.23	236 ±37%
DSO	20.12	181 ±27%

↑ Processing time in milliseconds ↑ CPU load (100% = 1 core)

SVO and its derivatives are used today in many of products...

- DJI drones
- Magic Leap AR headsets
- Oculus VR headsets
- Huawei phones
- Nikon cameras
- ...



Autonomous quadrotor navigation in dynamic scenes (down-looking camera)
(running on Odroid U3 board (ARM Cortex A9 at 90fps)



Throw-and-go (2015)
(inspired many products, like [DJI Tello drone](#))

20 m/s obstacle free autonomous quadrotor flight at DARPA FLA (2015)



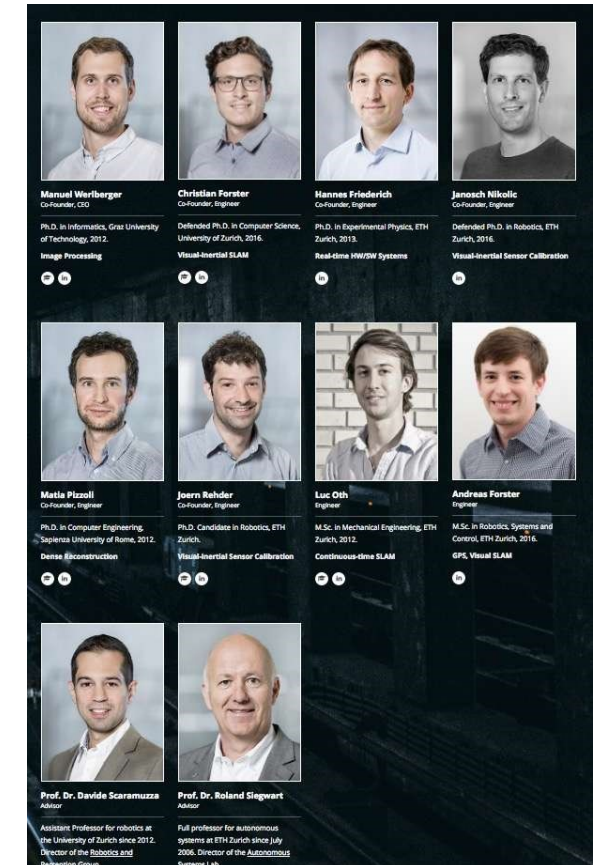
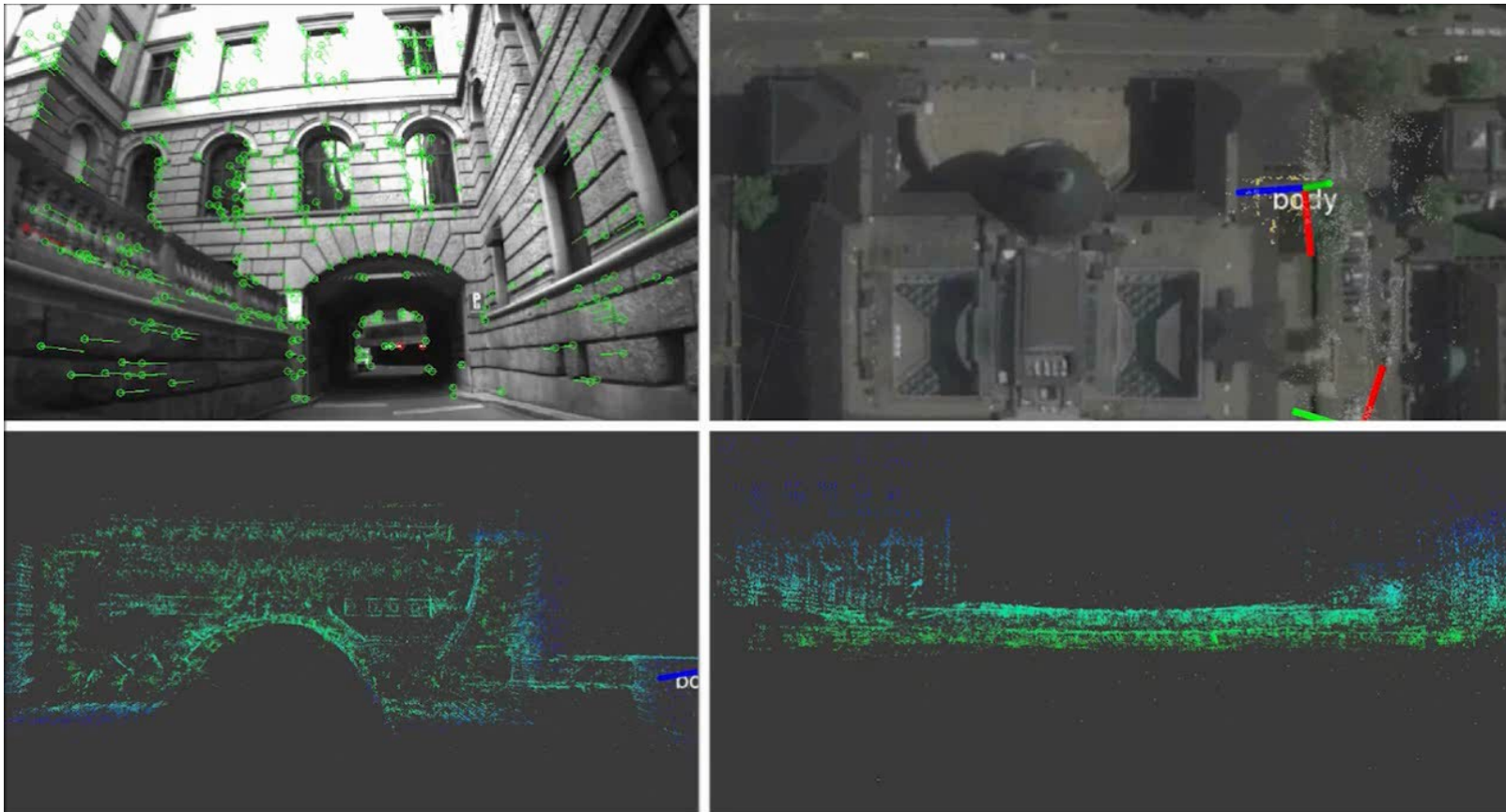
Virtual Reality with SVO running on an iPhone 6 at CES Las Vegas



More videos here: http://rpg.ifi.uzh.ch/svo_pro.html

Startup: “Zurich-Eye” – Today: Facebook-Oculus Zurich

- **Vision-based Localization and Mapping** systems for mobile robots
- Born in Sep. 2015, became **Facebook-Oculus Zurich** in Sep. 2016. Today, **200 employees**.



Startup: “Zurich-Eye” – Today: Facebook-Oculus Zurich

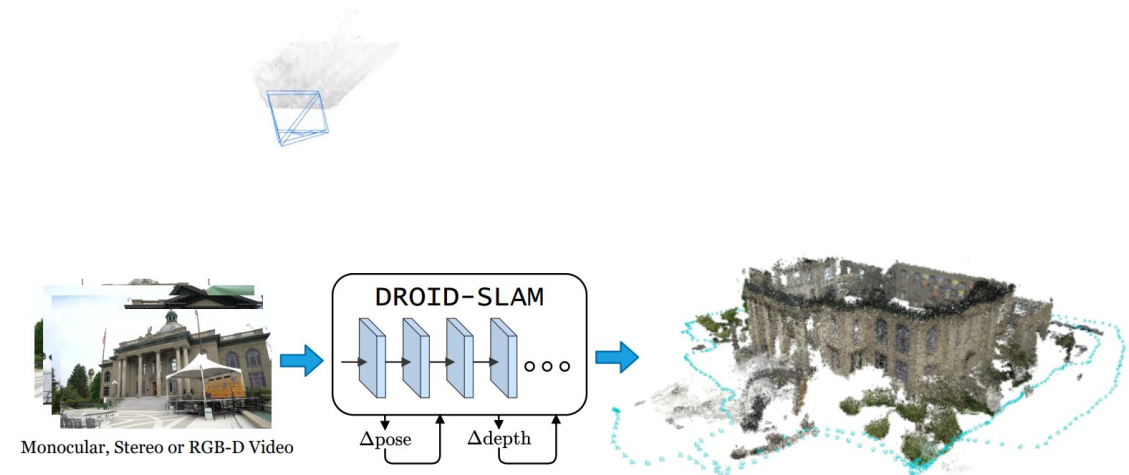
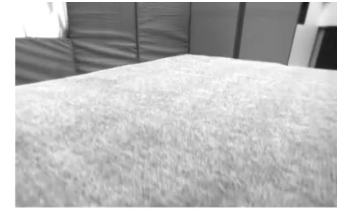
- **Vision-based Localization and Mapping** systems for mobile robots
- Born in Sep. 2015, became **Facebook-Oculus Zurich** in Sep. 2016. Today, **200 employees**.
- In 2018, Zurich-Eye launched **Oculus Quest** (25 million units sold so far)
- **Christian Forster (Meta Zurich & co-founder of Zurich-Eye)’s previous seminar in this course will be shared on OLAT.**



Latest and greatest 😊

DROID-SLAM

- DROID: “Differentiable Recurrent Optimization-Inspired Design” (DROID)
- Leverages a **deep recurrent network** to perform **dense** optical flow (i.e., dense pixel tracking)
- Estimates the pose of the incoming frames in a sliding window in the **local BA**
- Estimates the entire history of keyframes in the **global BA**
 - Selects keyframes based on heuristics
- Trained only on simulated monocular videos and **generalizes** to real scenes
 - Also accepts stereo or RGB-D videos as input
 - Supports loop closing
- Can be considered the **state-of-the-art SLAM** system in terms of accuracy but requires powerful GPU/-s



DPVO: Deep Patch Visual SLAM

- **Sparsified** variant of DROID-SLAM
 - Estimates **optical flow for a sparse set of patches** randomly selected
- Achieves accuracy on-par with DROID-SLAM but it requires **1/3 of the memory** and runs **3x faster**
 - Still requires a powerful GPU
- Trained only on simulated monocular videos (TartanAir Dataset)
 - **Generalizes** to real scenes
- Can be extended to a SLAM system with loop closing and place recognition, *DPV-SLAM*

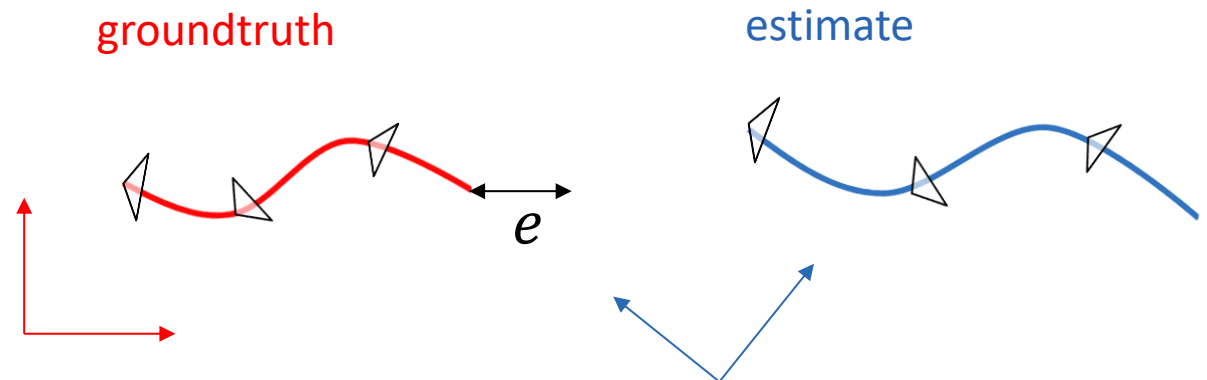


Teed, Lipson, Deng, *Deep Patch Visual Odometry*, Advances in neural information processing systems (NeurIPS), 2023, [PDF](#), [Code](#)

Lipson, Teed, Deng, *Deep Patch Visual SLAM*, European Conference on Computer Vision (ECCV), 2024, [PDF](#), [Code](#)

How can we evaluate the accuracy of VO/SLAM algorithms?

- Idea: **compare the estimated trajectory against ground truth** trajectory (from GPS, motion tracking systems), but the key question is **what error metric should be used?**
- Issues:
 - Different reference frames
 - Different scale
- **Naïve solution (not used anymore):** Maybe align the first poses and **measure the end-pose error?**
- **Not repeatable:**
 - Most VIOs are **non-deterministic** (e.g., **RANSAC, multithreading**) → every time you run your VIO on the same dataset, you get different results
- **Not meaningful:**
 - sensitive to the **trajectory shape** (the number of turns of a trajectory greatly affects the end-pose error)
 - **does not capture the error statistics**



Metric 1: Absolute Trajectory Error (ATE)

- **Step 1:** align the estimated trajectory to the ground truth **from the start to the end** using a similarity transformation (i.e., R, T, s) by minimizing the sum of square position errors

$$R, T, s = \underset{R, T, s}{\operatorname{argmin}} \sum_{k=0}^n \|\hat{C}_k - sRC_k - T\|^2$$

Parameters of the similarity transformation that we want to find

groundtruth positions

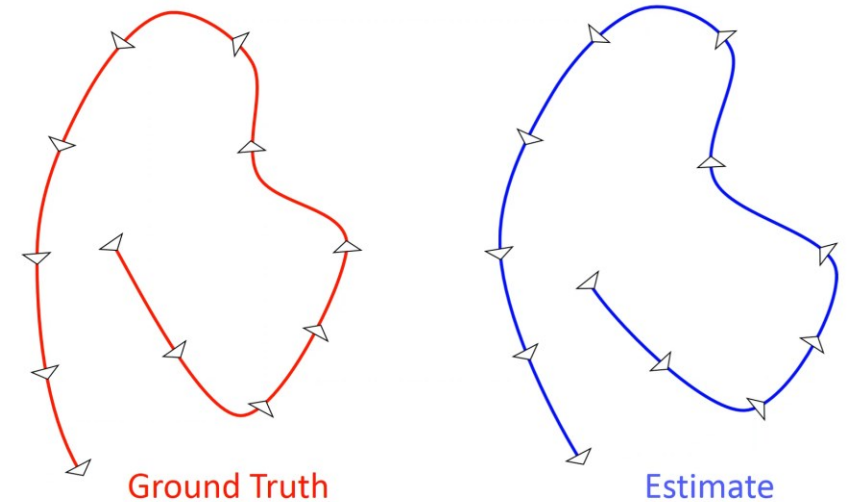
estimated positions

- **Step 2:** compute Root Mean Square Error (RMSE) after alignment:

$$RMSE = \sqrt{\frac{\sum_{k=1}^n \|\hat{C}_k - sRC_k - T\|^2}{n}}$$

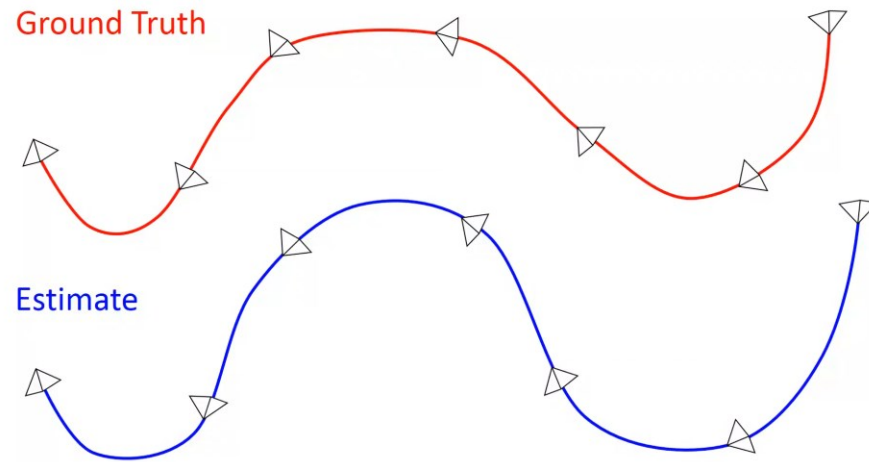
- **Pros and cons:**

- ✓ Single-number metric
- ✓ Captures the global error (accuracy of the global trajectory)
- ✗ Does not capture the relative error (accuracy of the local trajectory estimate)



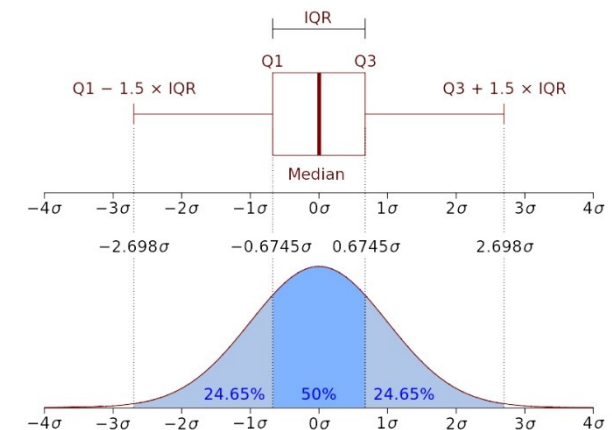
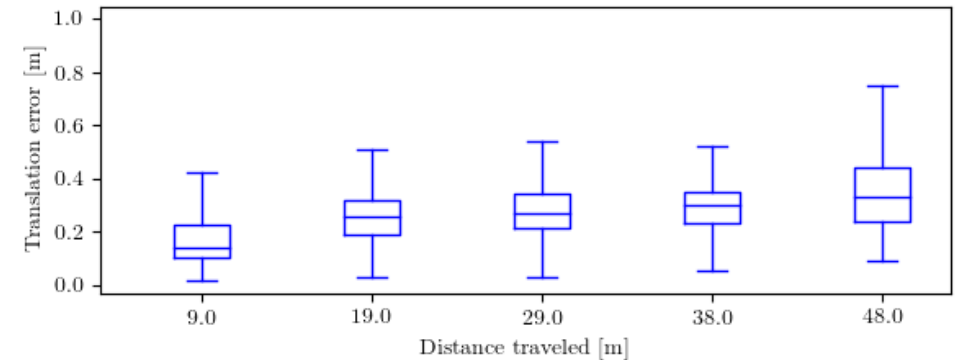
Metric 2: Relative Trajectory Error (RTE)

- Computes **error statistics** of sub-trajectories of specified lengths



- Pros and cons:

- ✓ Informative statistics: captures the relative error (accuracy of the local trajectory estimate)
- ✗ Complicated to compute and rank, but the good news is that there is code for it 😊 (toolbox, link below)



Boxplots are good to visualize error statistics via interquartile ranges ([link](#))



Things to remember

- Hierarchical SFM
- VO flowchart
 - Monocular VO
 - Stereo VO
 - Keyframe selection
- Bundle adjustment vs pose-graph optimization
- Indirect vs direct methods
- Direct methods: Dense, semi-dense, and sparse formulations
- Popular open-source VO algorithms
- ATE and RTE trajectory evaluation metrics

Readings

- Scaramuzza, D., Fraundorfer, F., **Visual Odometry: Part I - The First 30 Years and Fundamentals**, *IEEE Robotics and Automation Magazine*, Volume 18, issue 4, 2011. [PDF](#)
- Fraundorfer, F., Scaramuzza, D., **Visual Odometry: Part II - Matching, Robustness, and Applications**, *IEEE Robotics and Automation Magazine*, Volume 19, issue 1, 2012. [PDF](#)
- C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I.D. Reid, J.J. Leonard, **Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age**, *IEEE Transactions on Robotics*, Vol. 32, Issue 6, 2016. [PDF](#)

Understanding Check

Are you able to answer the following questions:

- Bundle Adjustment and Pose Graph Optimization. Mathematical expressions and illustrations. Pros and cons.
- Are you able to describe hierarchical and sequential SFM for monocular VO?
- What are the building blocks of visual odometry and SLAM?
- What are keyframes? Why do we need them and how can we select them?
- Are you able to define loop closure detection? Why do we need loops? How can we detect loop closures? (make link to other lectures)
- Are you able to describe the differences between feature-based methods and direct methods?
- Sparse vs semi-dense vs dense. What are their pros and cons?
- Are you able to provide a list of the most popular open source VO and VSLAM algorithms?
- Difference between SFM, VO, SLAM (see also lecture 01)
- How do we evaluate the accuracy of visual odometry? What are ATE and RTE, how are they computed and what do they capture?