University of Zurich[UZH] | ETH zürich

Institute of Informatics – Institute of Neuroinformatics

ROBOTICS & PERCEPTION GROUP

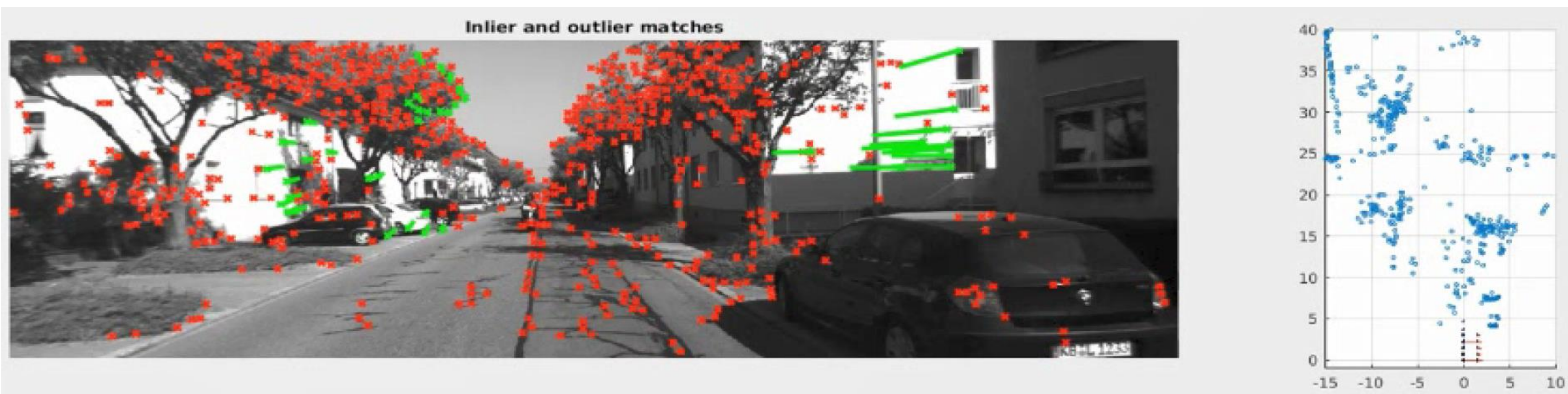
Lecture 09
Multiple View Geometry 3

Davide Scaramuzza

http://rpg.ifi.uzh.ch/

Lab Exercise 7 - Today

- Room ETH HG E 1.1 from 13:15 to 15:00
- Work description: P3P algorithm and RANSAC



Outline

- Bundle Adjustment
- SFM with n views

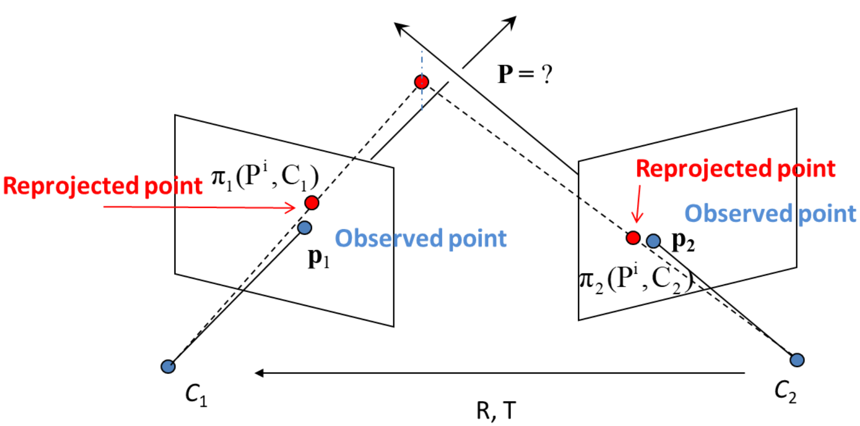
Bundle Adjustment (BA) – More in depth in Exercise 9

- **Non-linear, simultaneous refinement of structure P^i and motion $C = R, T$**
- It is used after linear estimation of R and T (e.g., after 8-point algorithm)
- Computes C, P^i by minimizing the Sum of Squared Reprojection Errors:

$$(P^i, C_2) = \arg\min_{P^i, C_1, C_2} \sum_{i=1}^{N} \left\| p_1^i - \pi_1(P^i, C_1) \right\|^2 + \left\| p_2^i - \pi_2(P^i, C_2) \right\|^2$$

NB: here, by C_1, C_2 we denote the **pose of** each camera in the **world** frame

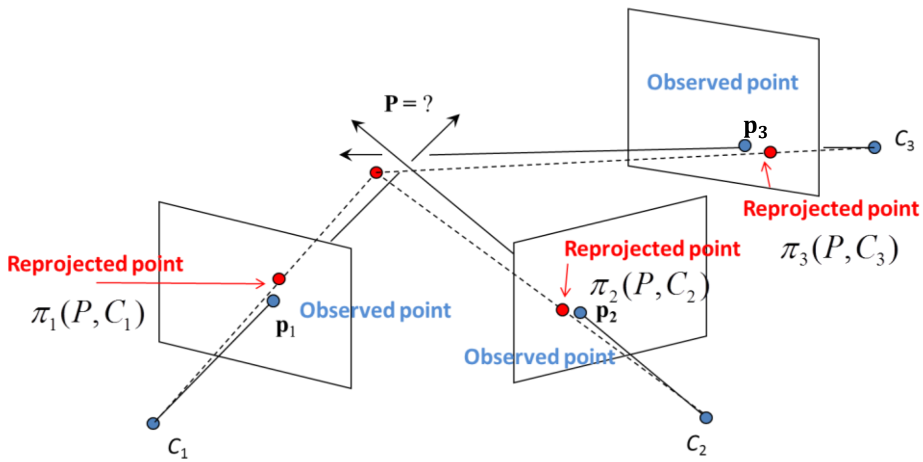
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)
- In order to not get stuck in local minima, the **initialization should be close the minimum**

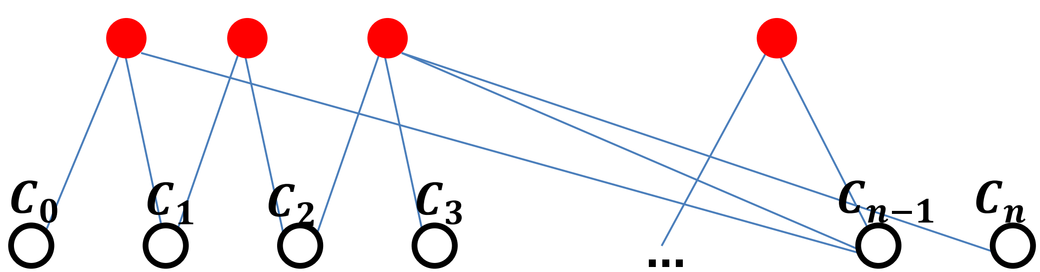


Bundle Adjustment (BA) for n Views

Minimizes the Sum of Squared Reprojection Errors **over each view k**

$$(P^i, C_k) = \arg\min_{P^i, C_k} \sum_{\mathrm{k}} \sum_{\mathrm{i}} \left\| \mathrm{p}_{\mathrm{k}}^{\mathrm{i}} - \pi_{\mathrm{k}}(P^i, C_k) \right\|^2$$





Huber and Tukey Norms

To prevent that large reprojection errors can negatively influence the optimization, a more robust norm $\rho()$ is used instead of the L_2:

$$(P^i, C_k) = \arg\min_{P^i, C_k} \sum_k \sum_i \rho\left(p_k^i - \pi_k(P^i, C_k)\right)$$

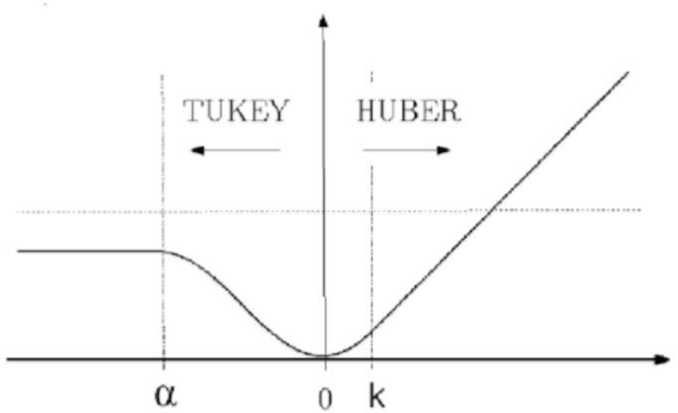
$\rho()$ is a robust cost function (**Huber or Tukey**) to penalize wrong matches:

- **Huber norm**:

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

- **Tukey norm**:

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



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

Outline

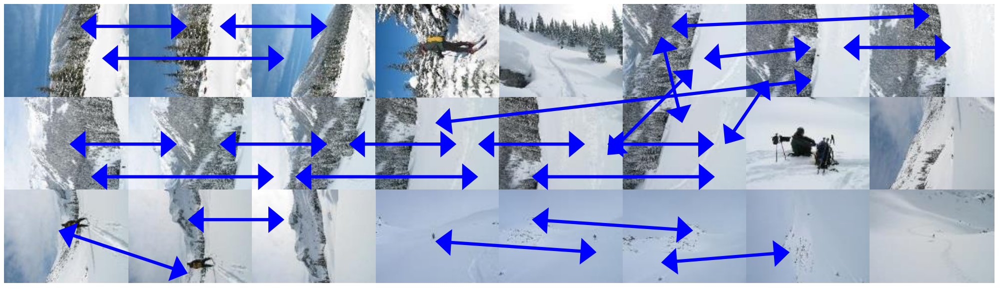
- Bundle Adjustment
- SFM with n views

Structure From Motion with n Views

- Compute initial structure and motion
  - **Hierarchical SFM**
  - **Sequential SFM**
- Refine simultaneously structure and motion through BA

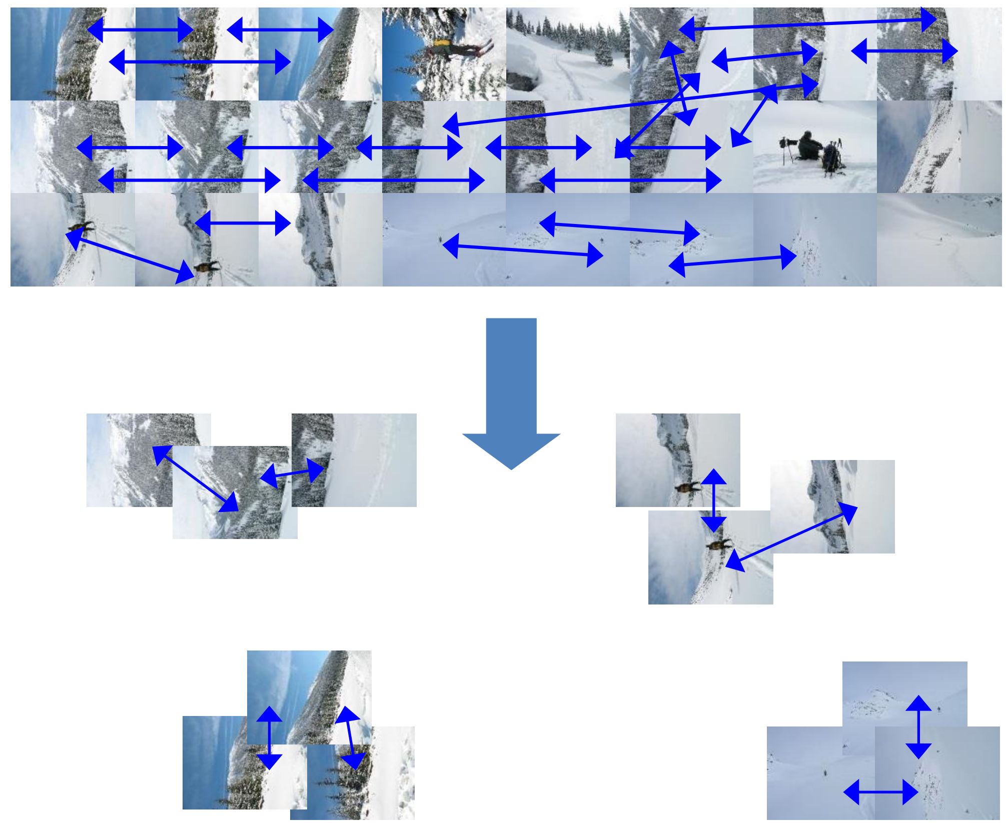
Hierarchical SFM

1. Extract and match features between nearby frames



Hierarchical SFM

1. Extract and match features between nearby frames
2. Identify clusters consisting of 3 nearby frames:
3. Compute SFM for 3 views:
    1. Compute SFM between 1 and 2 and build point cloud
    2. Then merge 3rd view by running 3-point RANSAC between point cloud and 3rd view



Hierarchical SFM

1. Extract and match features between nearby frames
2. Identify clusters consisting of 3 nearby frames:
3. Compute SFM for 3 views:
    1. Compute SFM between 1 and 2 and build point cloud
    2. Then merge 3rd view by running 3-point RANSAC between point cloud and 3rd view
4. Merge clusters pairwise and refine (BA) both structure and motion

How do you merge clusters?

Hierarchical SFM: Example

- Reconstruction from 150,000 images from Flickr associated with the tags "Rome"
- 4m 3D points. Cloud of 496 computers. 21 hours of computation!
- Paper: "*Building Rome in a Day*", ICCV'09: http://grail.cs.washington.edu/rome/ University of Washington, 2009 – Most influential paper of 2009 (link)

Structure From Motion with n Views

- Compute initial structure and motion
  - **Hierarchical SFM**
  - **Sequential SFM**
- 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**)

A Brief history of VO

- **1980**: First known VO real-time implementation on a robot by **Hans Moraveck** PhD thesis **(Stanforfd/NASA/JPL)** for Mars rovers using one sliding camera (*sliding stereo*).

A Brief history of VO

- **1980**: First known VO real-time implementation on a robot by **Hans Moraveck** PhD thesis **(Stanforfd/NASA/JPL)** for Mars rovers using one sliding camera (*sliding stereo*).

- **1980 to 2000**: The VO research was dominated by **NASA/JPL** in preparation of the **2004 mission to Mars**

- **2004**: VO was used on a robot on another planet: Mars rovers Spirit and Opportunity (see seminal paper from NASA/JPL, 2007)

- **2004**. VO was revived in the academic environment by **David Nister**'s «Visual Odometry» paper. The term VO became popular.

More about VO history and tutorials

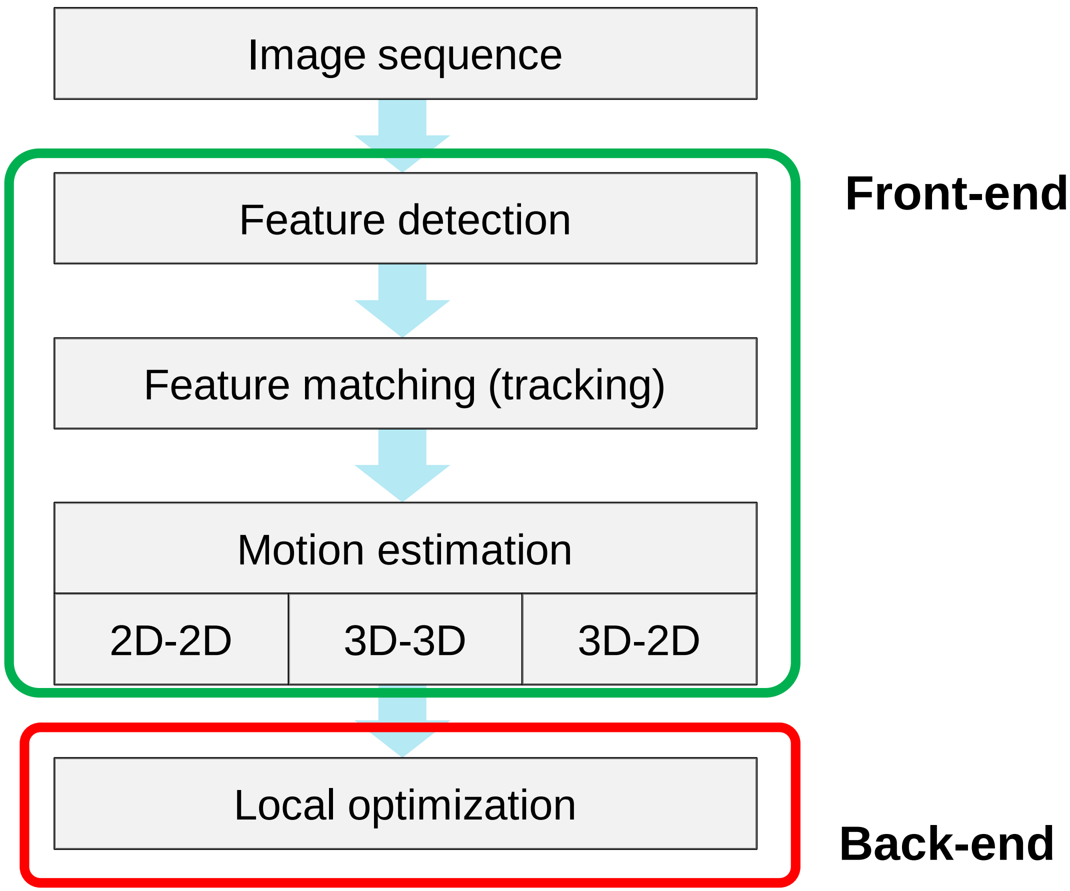
- Scaramuzza, D., Fraundorfer, F., **Visual Odometry: Part I** - The First 30 Years and Fundamentals, *IEEE Robotics and Automation Magazine*, Volume 18, issue 4, 2011. PDF

- 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

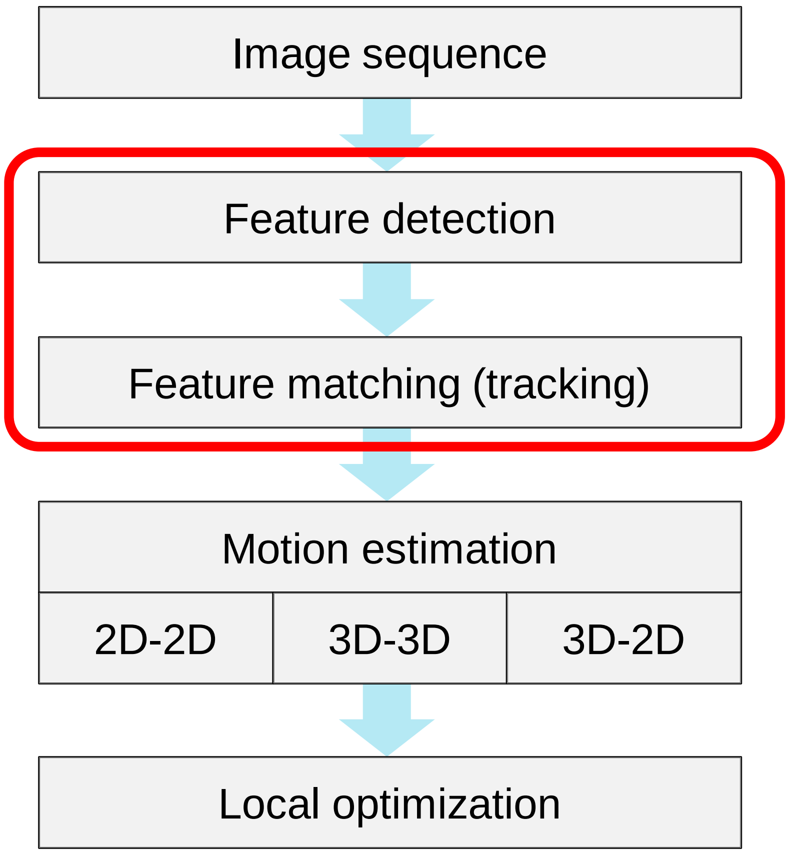
VO Flow Chart

VO computes the camera path incrementally (pose after pose)



VO Flow Chart

VO computes the camera path incrementally (pose after pose)



Example features tracks

VO Flow Chart

VO computes the camera path incrementally (pose after pose)

Image sequence

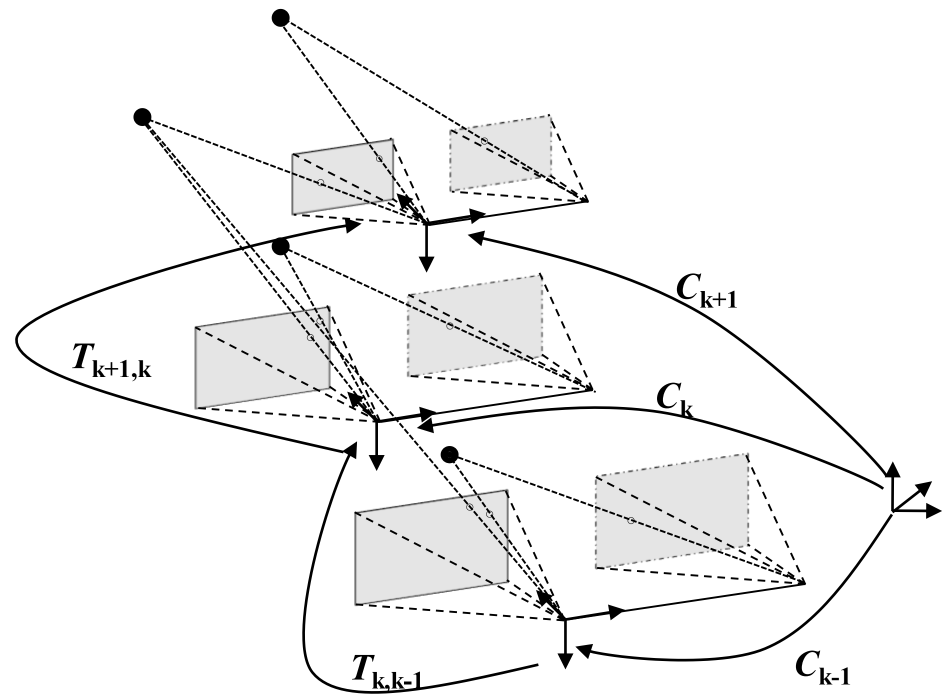
Feature detection

Feature matching (tracking)

Motion estimation

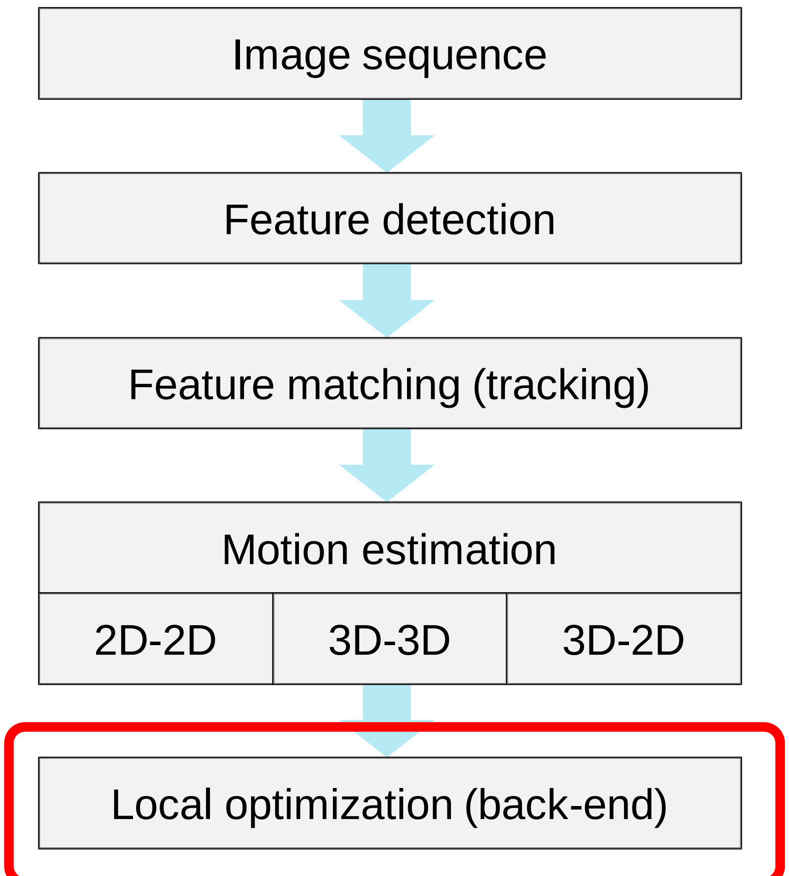
2D-2D	3D-3D	3D-2D

Local optimization

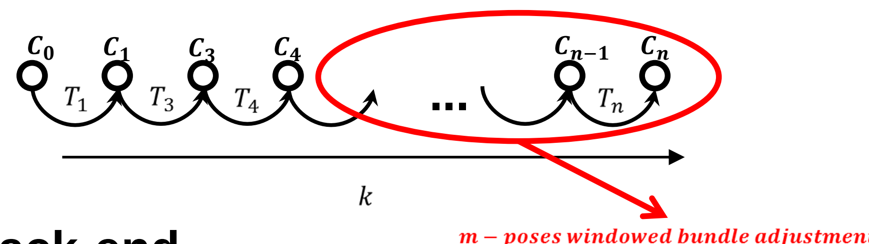


VO Flow Chart

VO computes the camera path incrementally (pose after pose)



Front-end



Back-end

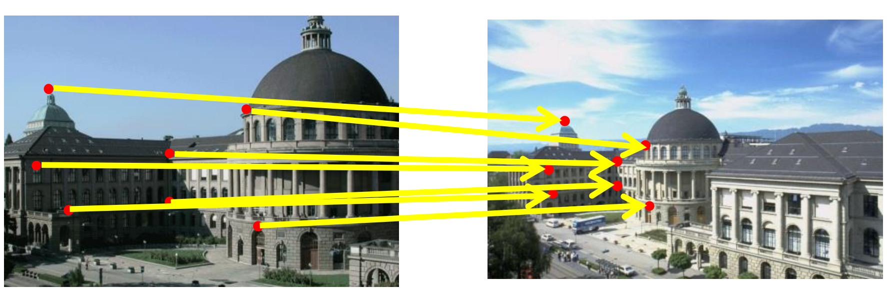
2D-to-2D (already seen: Lecture 08)

Motion from Image Feature Correspondences

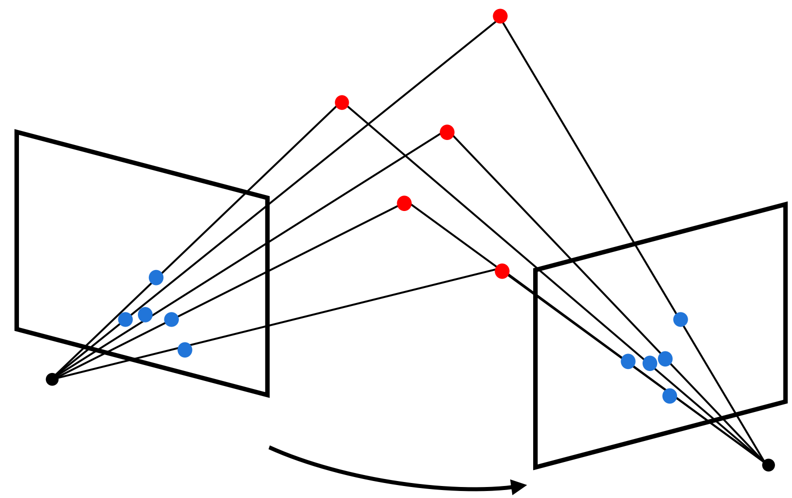
- Both feature points f_{k-1} and f_k are specified **in 2D**
- The minimal-case solution involves **5-point** correspondences
- The solution is found by minimizing the reprojection error:

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

- Popular algorithms: **5- and 8-point algorithms** [Hartley'97, Nister'06]



I_{k-1} I_k



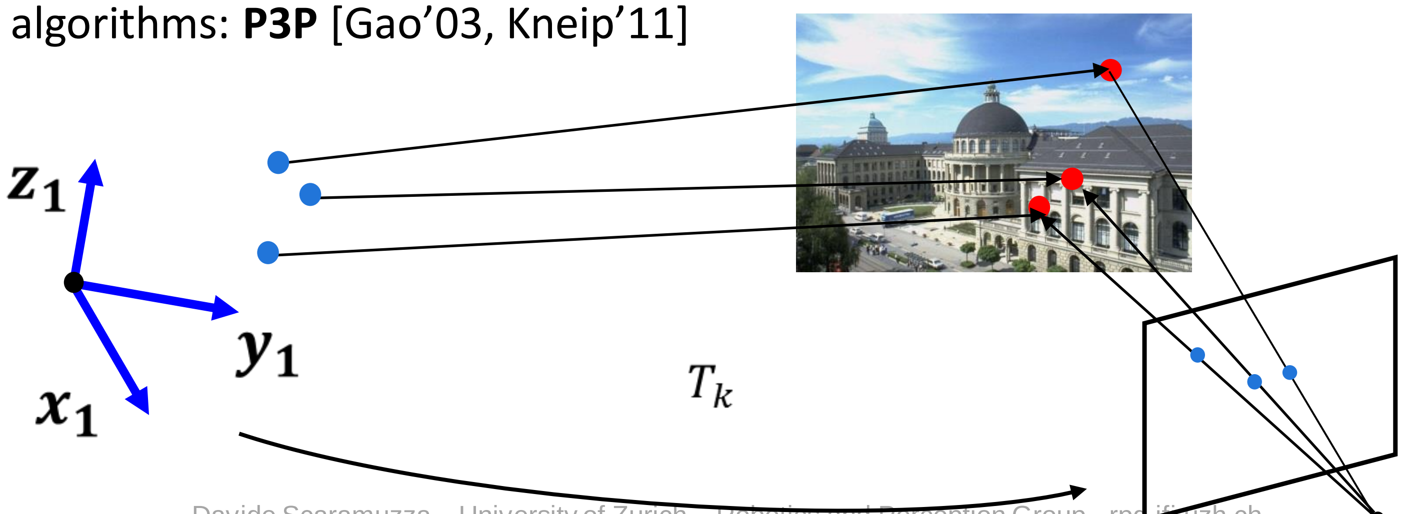
3D-to-2D (already seen: Lecture 03)

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** (**+1** for disambiguating the 4 solutions)
- The solution is found by minimizing the reprojection error:

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

- Popular algorithms: **P3P** [Gao'03, Kneip'11]



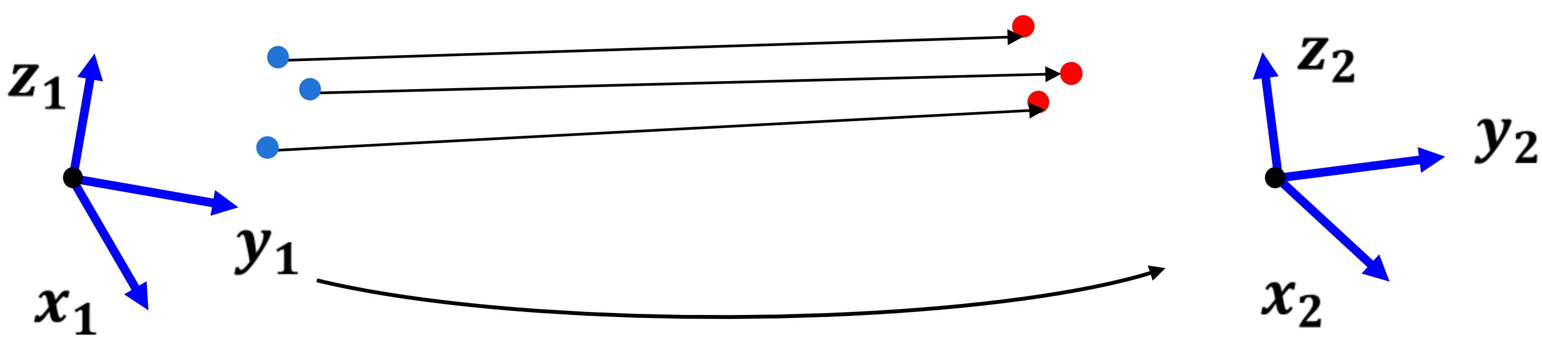
3D-to-3D

Motion from 3D-3D Point Correspondences (point cloud registration)

- 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 minimizing the 3D-3D Euclidean distance:

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

- Popular algorithm: [Arun'87] for global registration plus local refinement with Bundle Adjustment (BA)



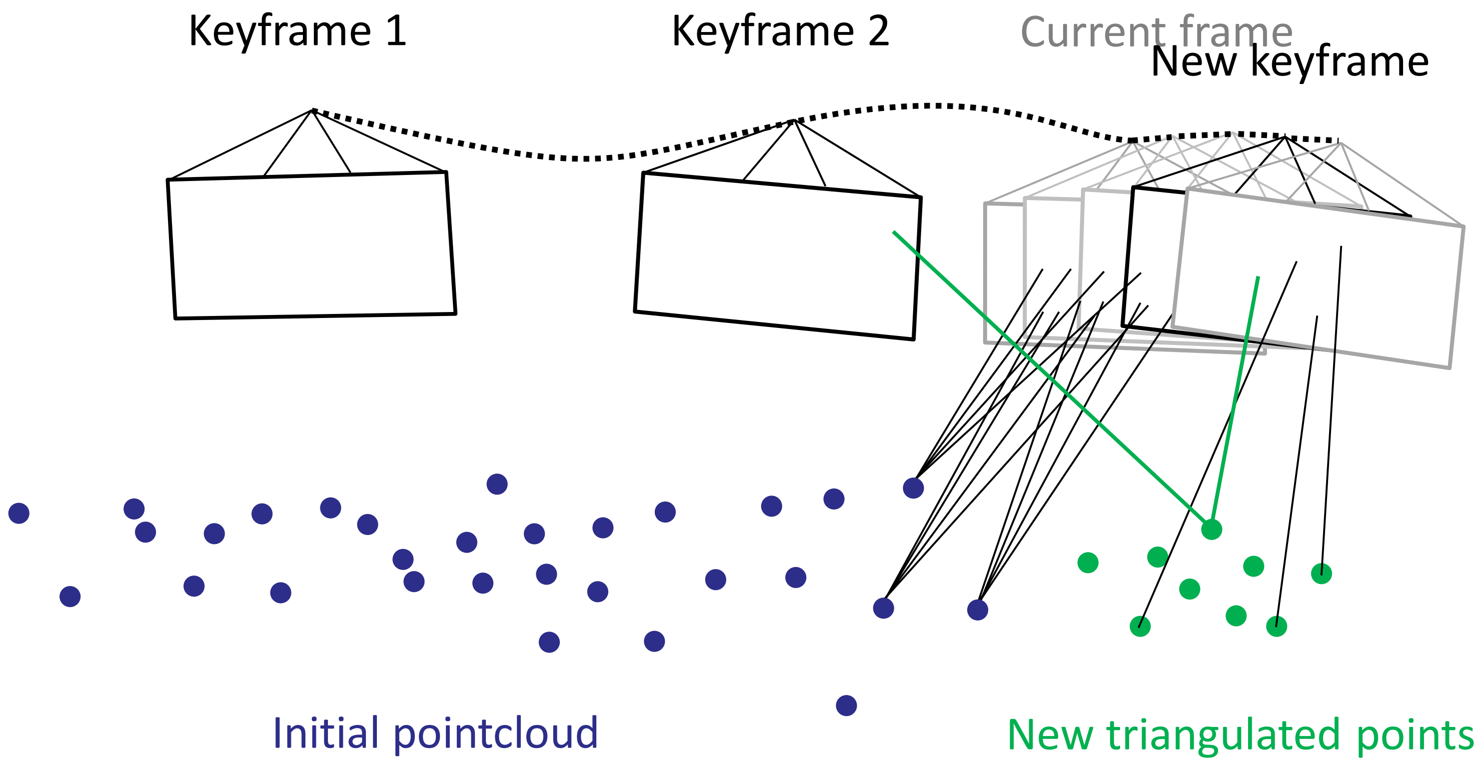
Arun, Huang, Blostein, "Least-squares fitting of two 3-d point sets," PAMI'87. PDF

Motion Estimation: Summary

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

Case Study:
Monocular Visual Odometry

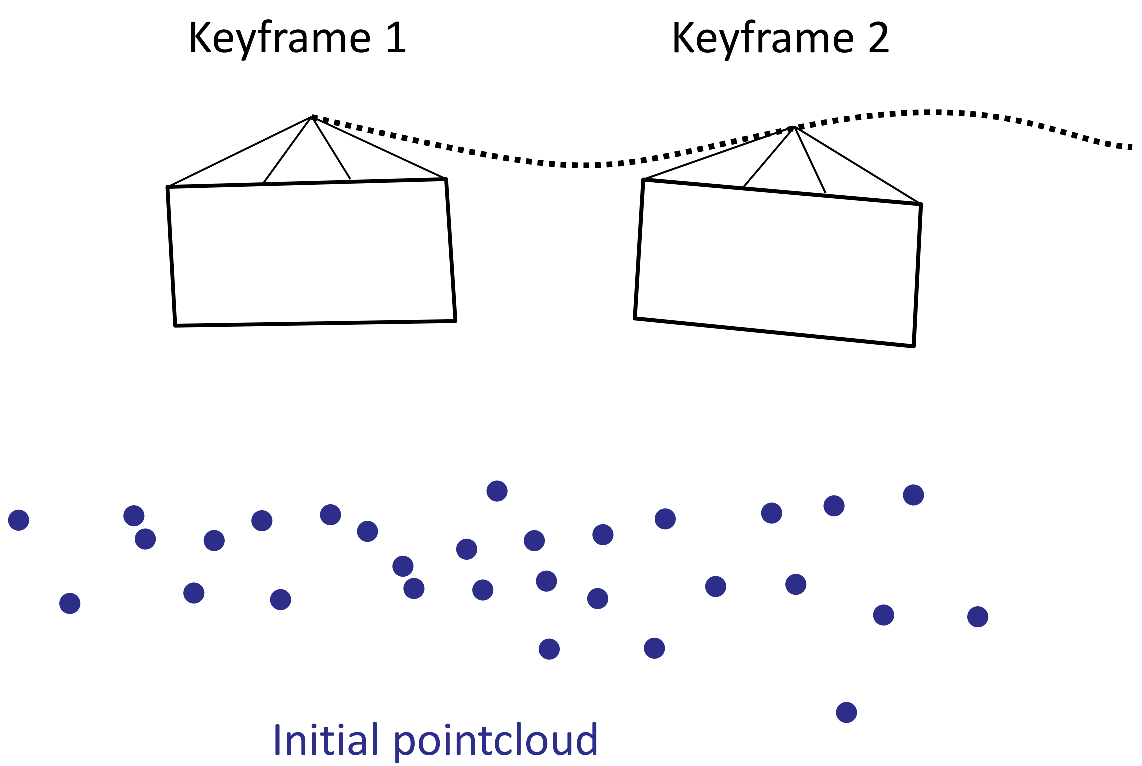
Case study: Monocular VO



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

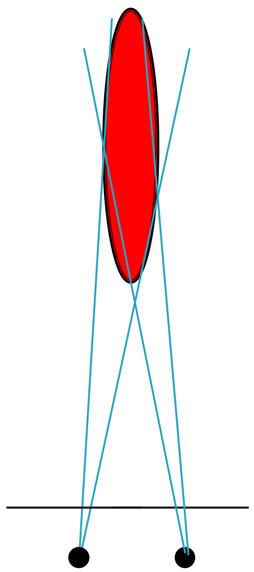
Monocular VO (i.e., with a single camera)

- **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?

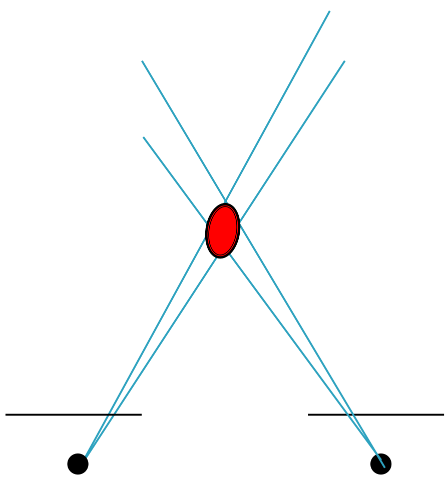


Skipping frames (Keyframe Selection)

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



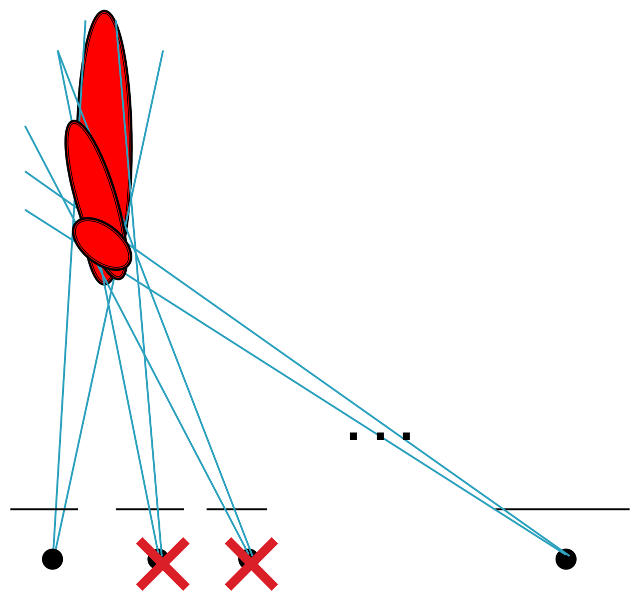
Small baseline → large depth uncertainty



Large baseline → small depth uncertainty

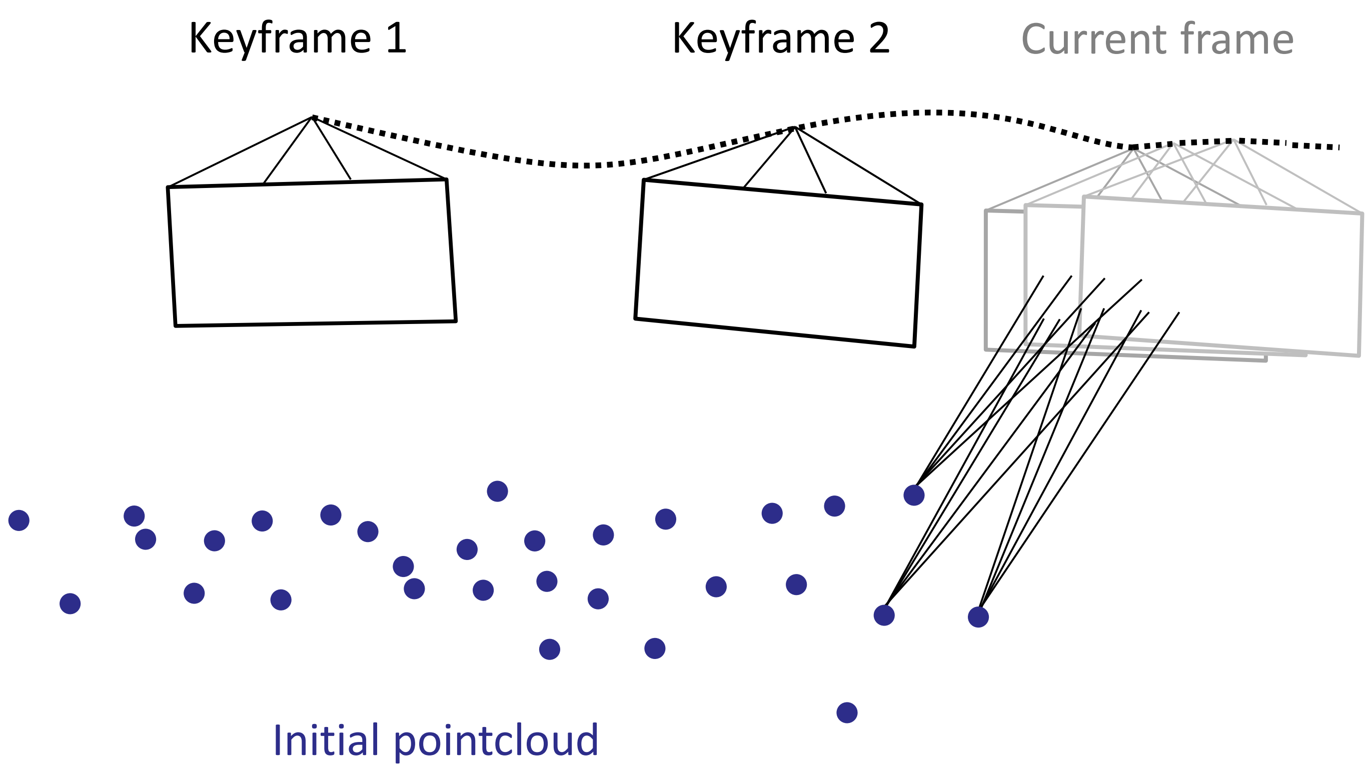
Skipping frames (Keyframe Selection)

- When frames are taken at nearby positions compared to the scene distance, 3D points will exibit large uncertainty
- One way to avoid this consists of **skipping frames** until the average uncertainty of the 3D points decreases below a certain threshold. The selected frames are called ***keyframes***
- **Rule of the thumb:** add a keyframe when $\frac{\textit{keyframe distance}}{\textit{average-depth}} > \textit{threshold (\sim10-20 \%)}$



Monocular VO (i.e., with a single camera)

- **Localization**
  - Given a 3D point cloud (map), determine the pose of each additional view
    - How?
    - How long can I do that?



Monocular VO (i.e., with a single camera)

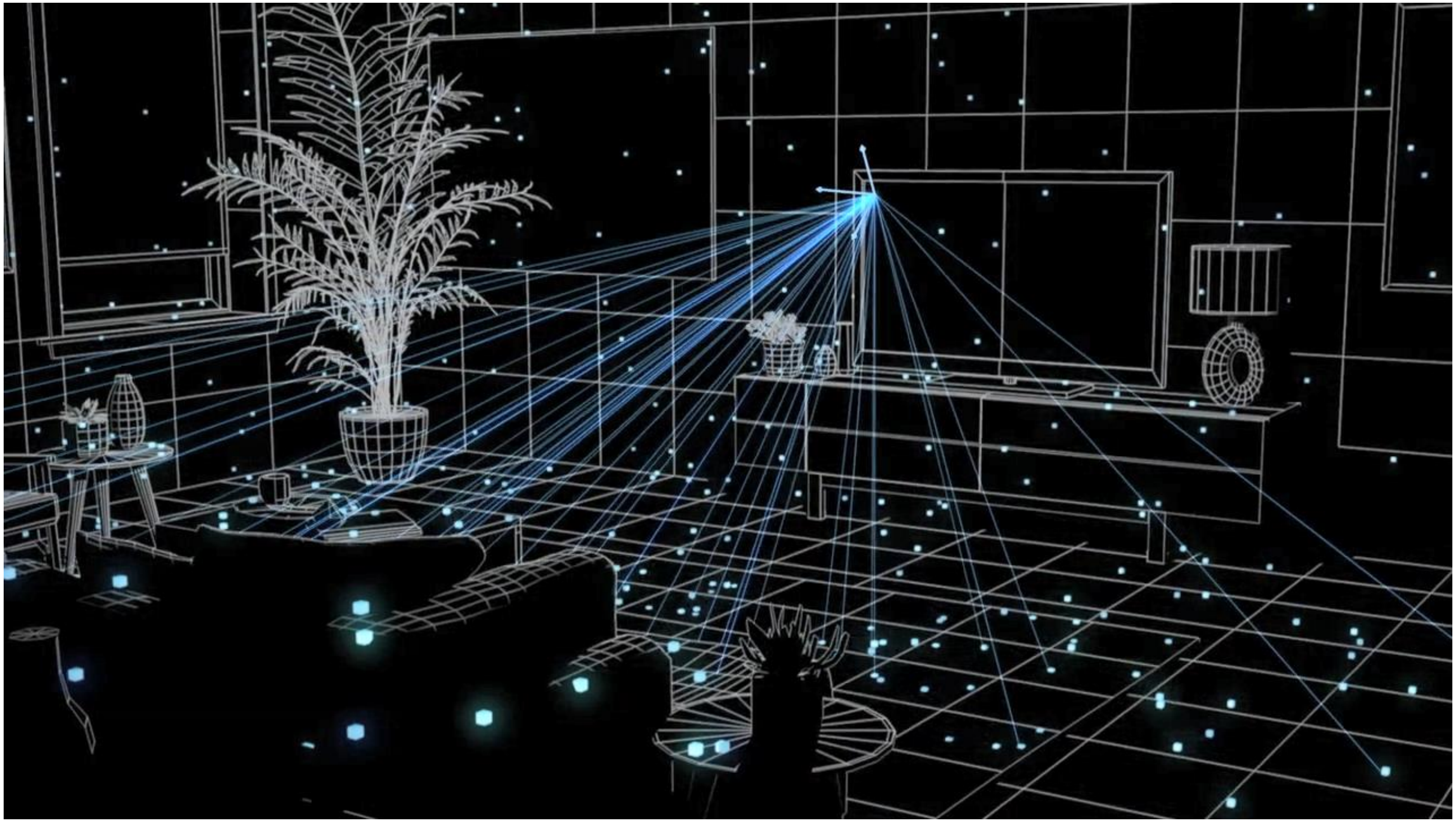
- **Localization**
  - Given a 3D point cloud (map), determine the pose of each additional view
    - How?
    - How long can I do that?

Recall:

- PnP problem (Perspective from n Points)
- What's the minimal number of required point correspondences?
  - Lecture 3:
    - 6 for DLT algorithm (linear solution)
    - 3 (+1) for P3P algorithm (non-linear solution)

Monocular VO (i.e., with a single camera)

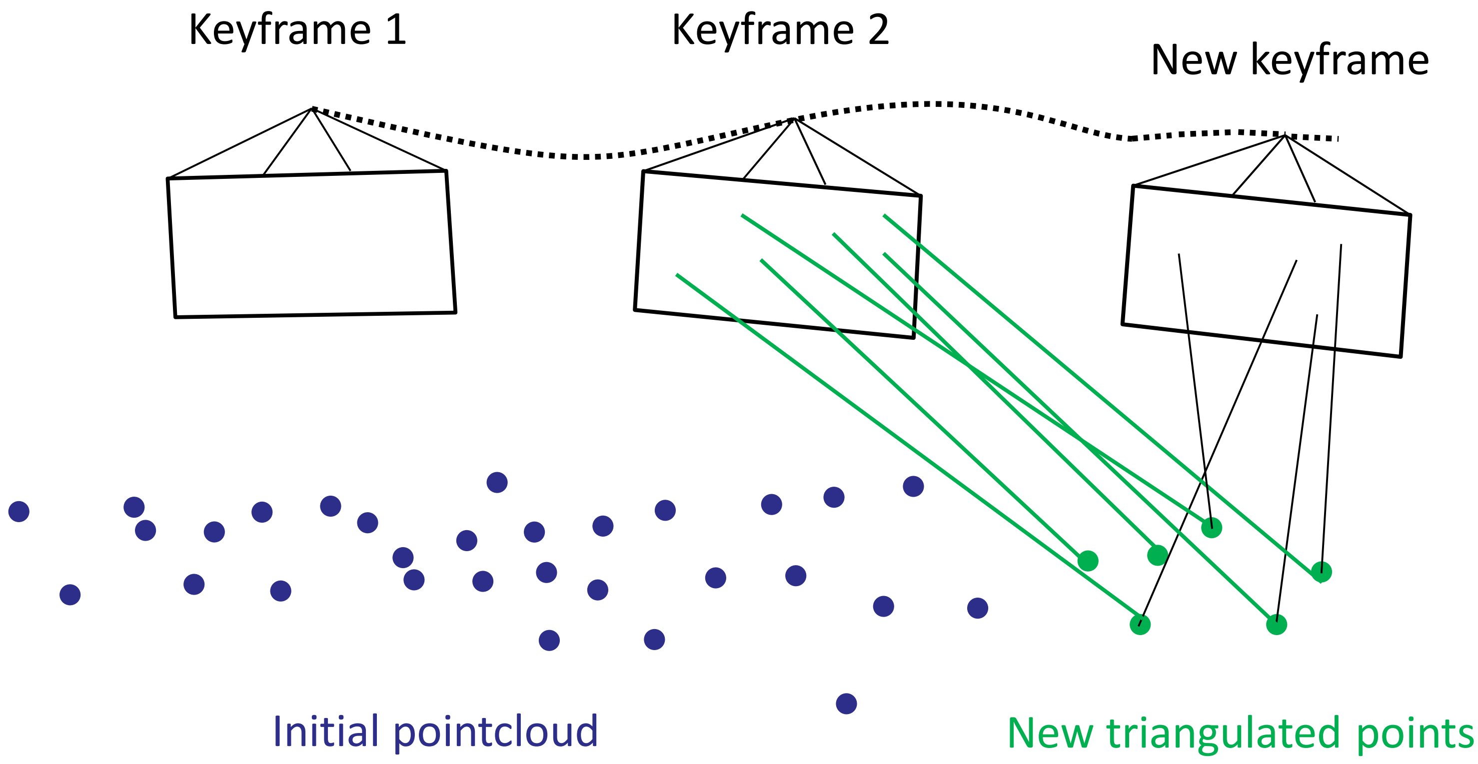
- **Localization**
  - Given a 3D point cloud (map), determine the pose of each additional view



Video of Oculus Insight (the VIO used in Oculus Quest): built by former Zurich-Eye team, today Oculus Zurich. Dr. Christian Forster (Oculus Zurich & co-founder of Zurich-Eye) will give a lecture on Nov. 28

Extend Structure (i.e., mapping)

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



Monocular Visual Odometry: putting all pieces together

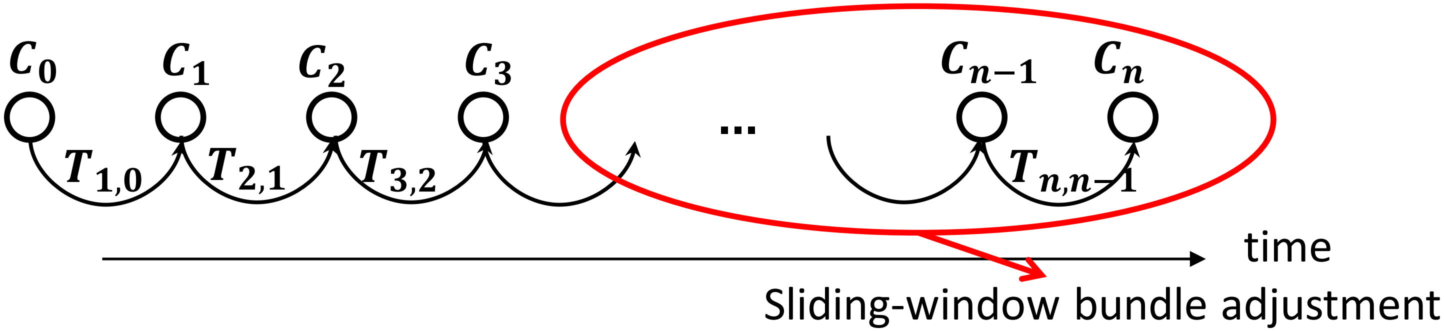
- Let the relative motion T_k from images I_{k-1} to image I_k

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

- Concatenate adjacent transformations to recover the current pose:

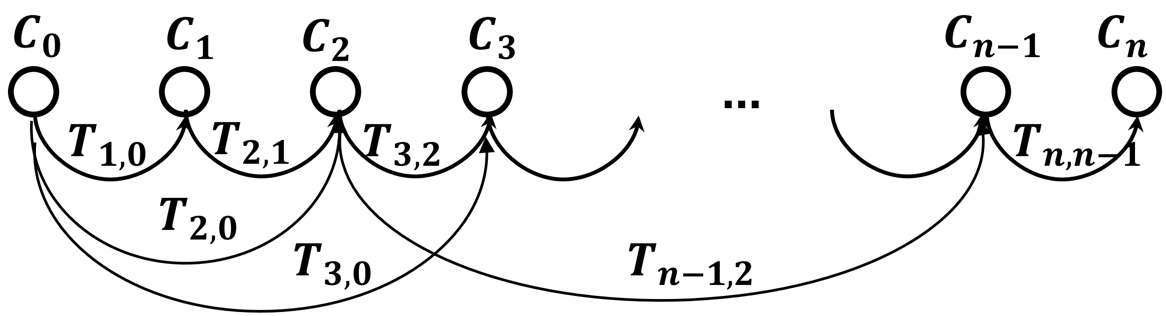
$$C_n = C_{n-1} T_{n,n-1}$$

- Optimize over the last m poses to refine the trajectory (Pose-Graph or Bundle Adjustment)



Pose-Graph Optimization

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

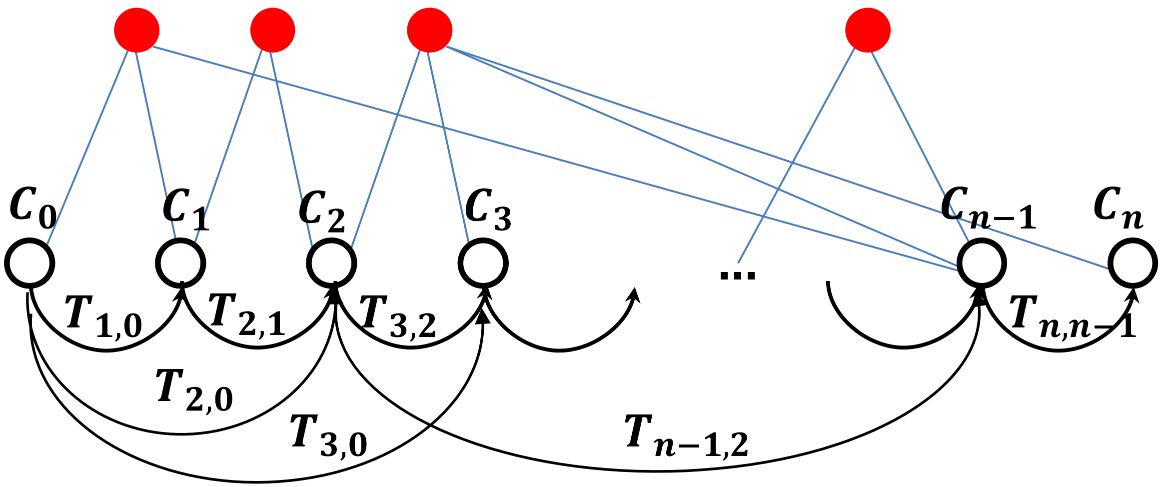


- Transformations can be computed also between **non-adjacent frames $\boldsymbol{T_{ij}}$** (e.g., when features from previous keyframes are still observed). They can be used as additional constraints to improve cameras poses by minimizing the following:

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

- For efficiency, only the last m keyframes are used
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools: **g2o**, **GTSAM**, **SLAM++**, **Google Ceres**

Bundle Adjustment (BA)



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

$$X^i, C_k = argmin_{X^i, C_k,} \sum_i \sum_k \rho\left(p_k^i - \pi(X^i, C_k)\right)$$

- $\rho_H()$ is a robust cost function (e.g., **Huber or Tukey cost**) to penalize wrong matches
- In order to not get stuck in local minima, the initialization should be close to the minimum
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools: [**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**: $O\left((qM + lN)^3\right)$ with M and N being the number of points and cameras poses and q and l the number of parameters for points and camera poses. Workarounds:
  - A **small window size** limits the number of parameters for the optimization and thus makes real-time bundle adjustment possible.
  - It is possible to reduce the computational complexity by just optimizing over the camera parameters and keeping the 3-D landmarks fixed, e.g., (**motion-only BA**)

Loop Closure Detection (i.e., Place Recognition)

- **Relocalization problem:**
  - During VO, tracking can be lost (due to occlusions, low texture, quick motion, illumination change)

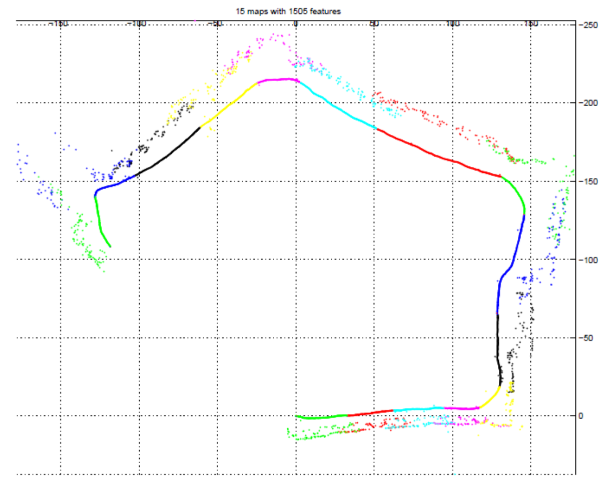
- Solution: **Re-localize** camera pose and continue

- **Loop closing problem**
  - When you go back to a previously mapped area:
    - **Loop detection**: to avoid map duplication
    - **Loop correction**: to compensate the accumulated drift
  - In both cases you need a place recognition technique

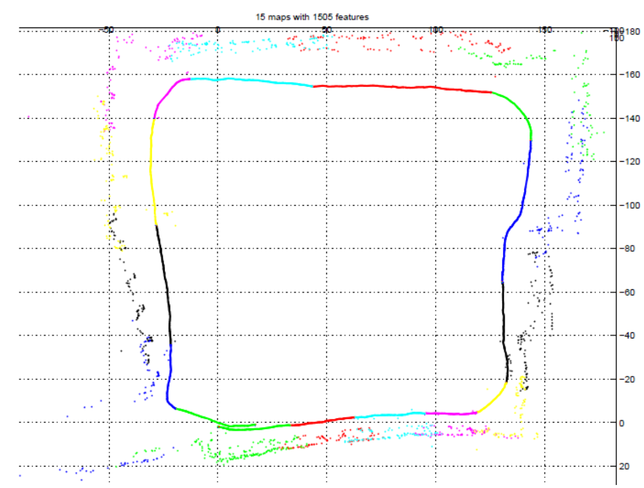
We will address place recognition in Lecture 12

Recall: VO vs. Visual SLAM

- **Visual SLAM = visual odometry + loop detection + graph optimization**



Visual odometry

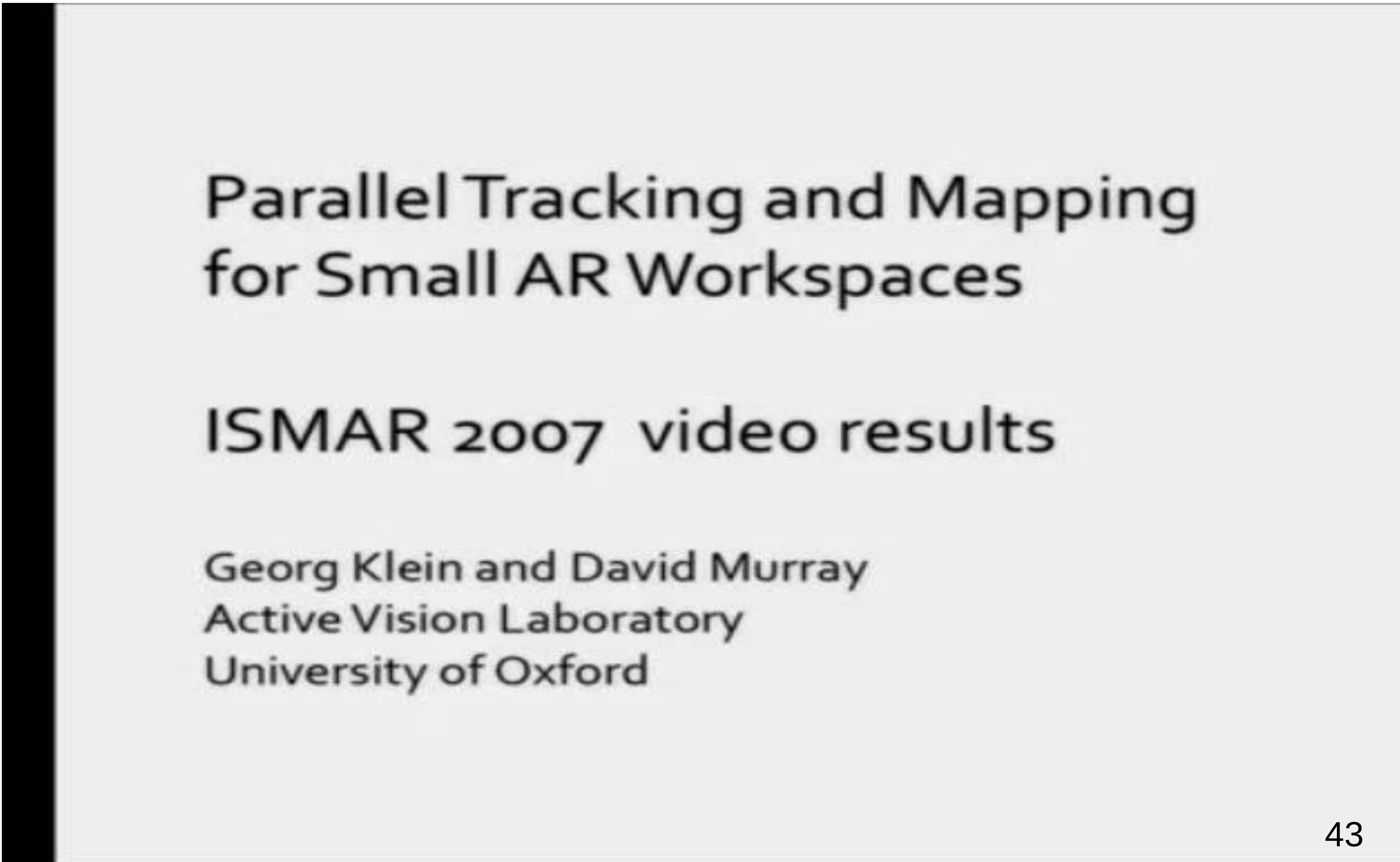


Visual SLAM

Open Source Monocular VO and SLAM algorithms

- **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
- **DSO** [Engel'16] -> Munich, Cremers' lab
- **SVO** [Forster, ICRA'14, TRO'17] -> Zurich, Scaramuzza's lab

PTAM: Parallel Tracking and Mapping for Small AR Workspaces



ORB-SLAM [Mur-Artal, TRO'15]

- **Feature based**
  - FAST corners + ORB descriptors
  - ORB: binary descriptor, very fast to compute and match (Hamming distance)
  - Minimizes reprojection error

- Includes:
  - **Loop closing**
  - **Relocalization**
  - Final optimization

- **Real-time (30Hz)**

Download from
http://webdiis.unizar.es/~raulmur/orbslam/

[Mur-Artal, Montiel, Tardos, ORB-SLAM: Large-scale Feature-based SLAM, TRO'15]

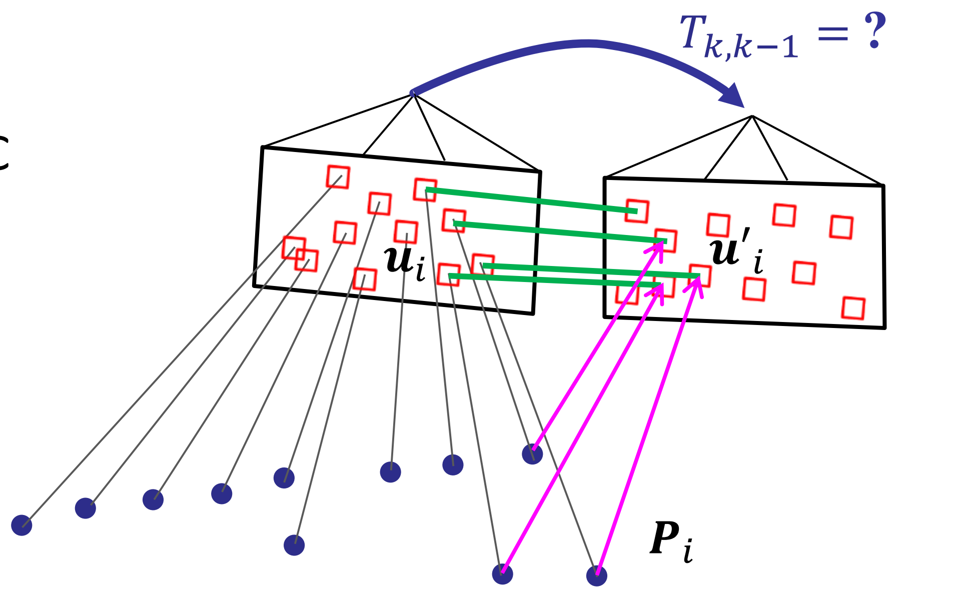
Feature-based methods

1. Extract & match features + RANSAC

2. Bundle Adjust by minimizing the **Reprojection Error**

$$T_{k,k-1} = \arg\min_{T} \sum_{i} \|\boldsymbol{u}'_i - \boldsymbol{u}_i\|_{\Sigma}^2$$

where $\boldsymbol{u}'_i = \pi(\boldsymbol{P}_i, T_{k,k-1})$

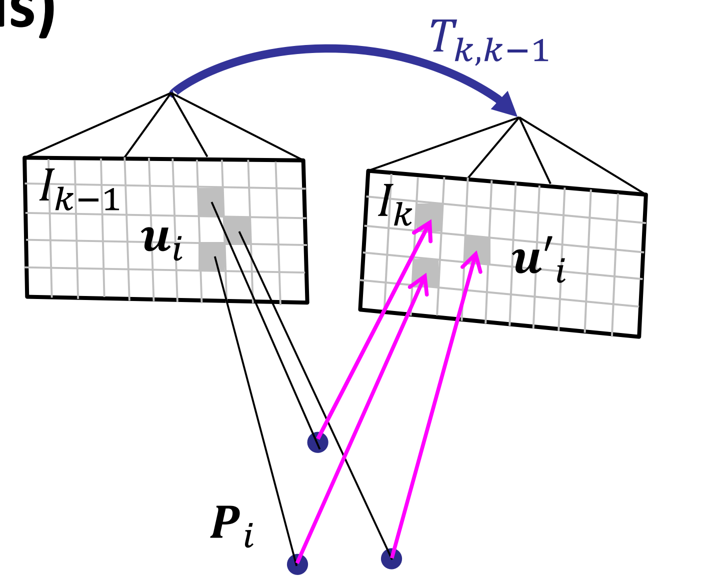


Direct methods (photometric methods)

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

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

where $\boldsymbol{u}'_i = \pi(\boldsymbol{P}_i, T_{k,k-1})$



Irani, Anandau, All about direct methods, Springer'99. PDF

Feature-based methods

1. Extract & match features + RANSAC

2. Bundle Adjust by minimizing the **Reprojection Error**

$$T_{k,k-1} = \arg\min_{T} \sum_{i} \| \boldsymbol{u}'_i - \boldsymbol{u}_i \|_{\Sigma}^2$$

where $\boldsymbol{u}'_i = \pi(\boldsymbol{P}_i, T_{k,k-1})$

✓ Large frame-to-frame motions

✓ Accuracy: Efficient optimization of structure and motion (Bundle Adjustment)

× Slow due to costly feature extraction and matching

× Matching Outliers (RANSAC)

Direct methods (photometric me

1. No feature extraction & no RANSAC. Inst directly minimize **Photometric Error**:

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

where $\boldsymbol{u}'_i = \pi(\boldsymbol{P}_i, T_{k,k-1})$

✓ All information in the image can be exploited (precision, robustness)

✓ Increasing camera frame-rate reduces computational cost per frame

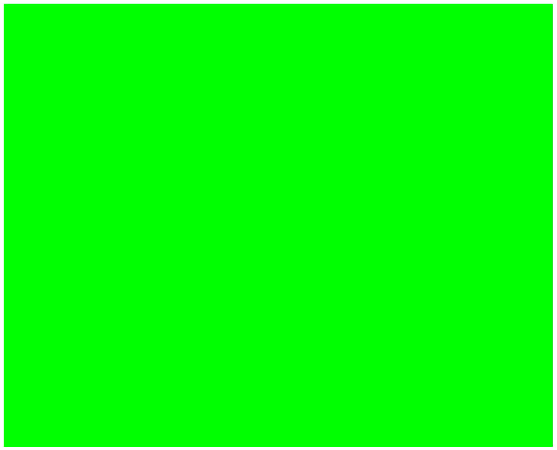
× Limited frame-to-frame motion

× Joint optimization of dense structure and motion too expensive

Irani, Anandau, All about direct methods, Springer'99. PDF

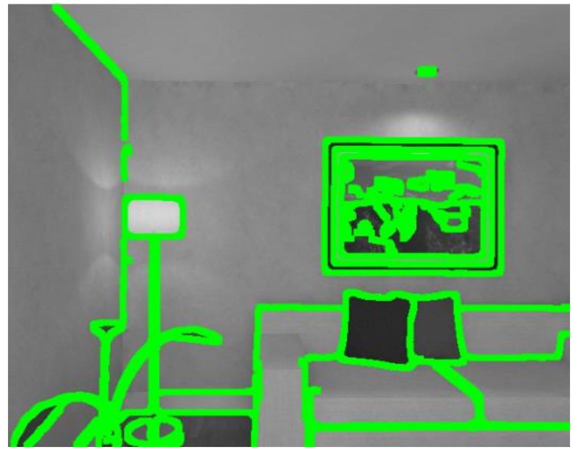
Direct Methods: Dense vs Semi-dense vs Sparse [TRO'16]

Dense



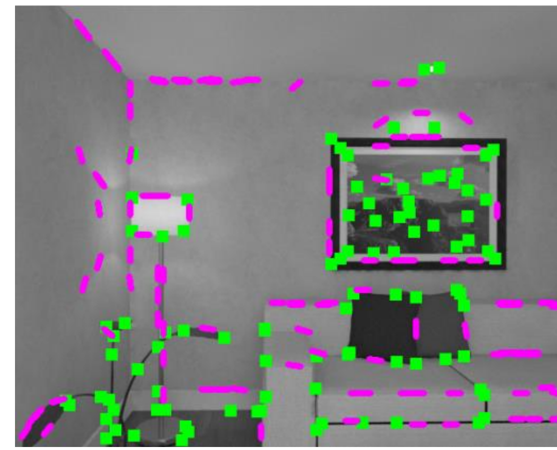
DTAM [Newcombe '11] REMODE [Pizzoli'14]
300'000+ pixels

Semi-Dense



LSD-SLAM [Engel'14]
~10,000 pixels

Sparse

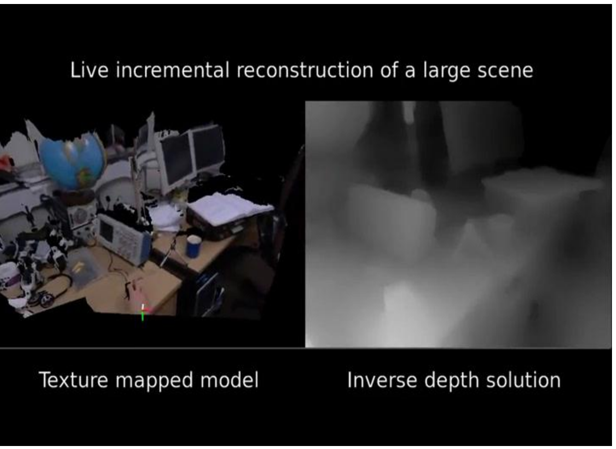


SVO [Forster'14]
100-200 x 4x4 patches ≅ 2,000 pixels

[Forster, et al., SVO: Semi Direct Visual Odometry for Monocular and Multi-Camera Systems, TRO'17]

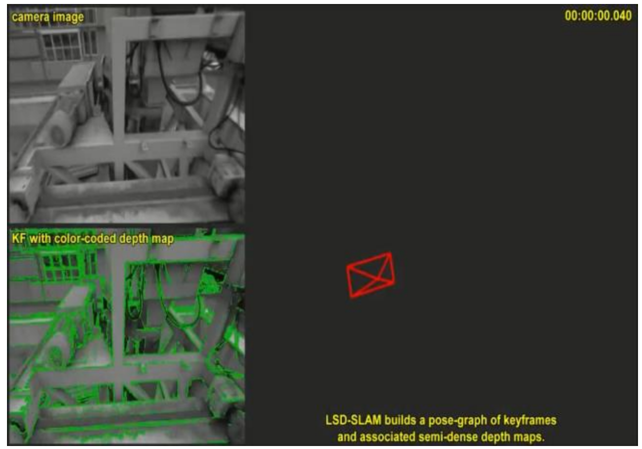
Direct Methods: Dense vs Semi-dense vs Sparse [TRO'16]

Dense



DTAM [Newcombe '11] REMODE [Pizzoli'14]
300'000+ pixels

Semi-Dense



LSD-SLAM [Engel'14]
~10,000 pixels

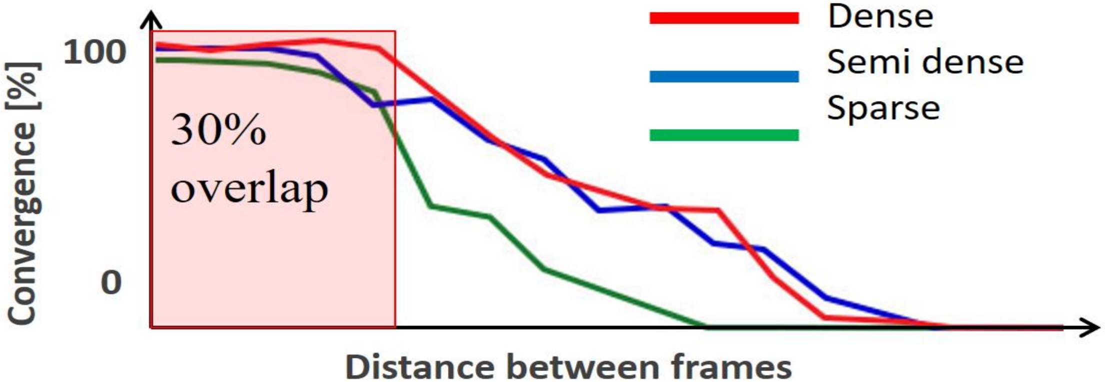
Sparse

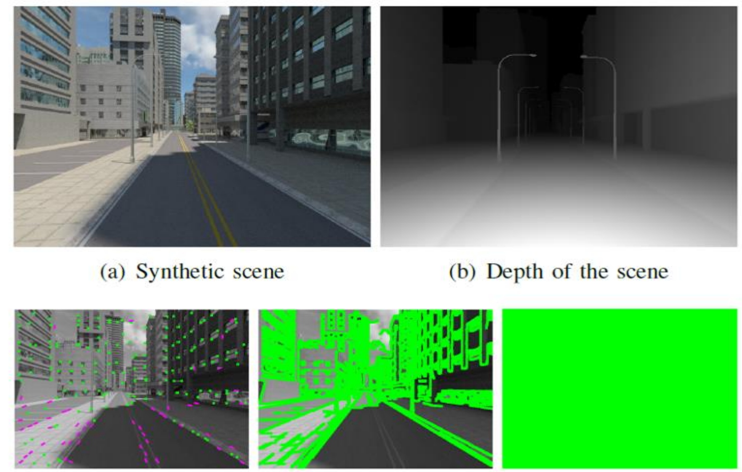
SVO [Forster'14] DSO [Engel'17]
100-200 x 4x4 patches ≅ 2,000 pixels

[Forster, et al., SVO: Semi Direct Visual Odometry for Monocular and Multi-Camera Systems, TRO'17]

Direct Methods: Dense vs Semi-dense vs Sparse [TRO'16]

Robustness to motion baseline (computed from 1,000 Blender simulations)





(a) Synthetic scene (b) Depth of the scene

(c) Sparse (d) Semi-Dense (e) Dense

Images from the synthetic
Multi-FOV Zurich Urban Dataset

- **Dense** and **Semi-dense behave similarly**
  - weak gradients are not informative for the optimization)
- Dense only useful with **motion blur** and **defocus**
- **Sparse** methods behave equally well for image **overlaps up to 30%**

- [Forster, et al., SVO: Semi Direct Visual Odometry for Monocular and Multi-Camera Systems, TRO'17]
- Multi-FOV Zurich Urban Dataset: http://rpg.ifi.uzh.ch/fov.html

LSD-SLAM [Engel, ECCV'14]

- **Direct** (photometric error) **+ Semi-Dense** formulation
  - 3D geometry represented as semi-dense depth maps
  - Minimizes **photometric error**
  - **Separateley** optimizes poses & structure

- Includes:
  - **Loop closing**
  - **Relocalization**
  - Final optimization

- **Real-time (30Hz)**

Download from
https://vision.in.tum.de/research/vslam/lsdslam

[Engel, Schoeps, Cremers, LSD-SLAM: Large-scale Semi-Dense SLAM, ECCV'14]

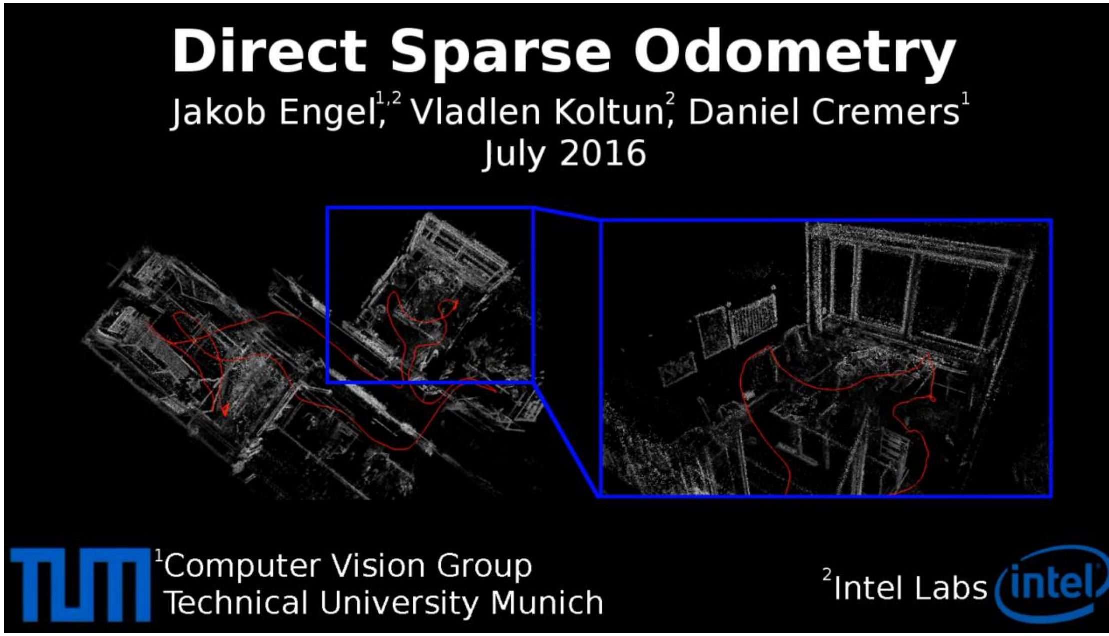
DSO [Engel, PAMI'17]

Download from
https://vision.in.tum.de/research/vslam/dso

- **Direct** (photometric error) **+ Sparse** formulation
  - 3D geometry represented as sparse large gradients
  - Minimizes **photometric error**
  - **Jointly** optimizes poses & structure (sliding window)
  - Incorporate photometric correction to compensate exposure time change

$$E_{\mathbf{p}j} := \sum_{\mathbf{p} \in \mathcal{N}_{\mathbf{p}}} w_{\mathbf{p}} \left\| (I_j[\mathbf{p}'] - b_j) - \frac{t_j e^{a_j}}{t_i e^{a_i}} (I_i[\mathbf{p}] - b_i) \right\|_{\gamma}$$

- **Real-time (30Hz)**



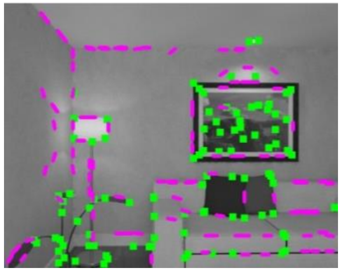
[Engel, Koltun, Cremers, DSO: Direct Sparse Odometry, PAMI'17]

SVO [Forster, ICRA'14, TRO'17]

- **Direct** (minimizes photometric error)
  - **Corners and edgelets**
  - Frame-to-frame motion estimation
- **Feature-based** (minimizes reprojection error)
  - **Frame-to-Keyframe** pose refinement
- **Mapping**
  - **Probabilistic depth** estimation
- **SVO 2.0 includes**
  - Fish-eye & Omni cameras
  - Multi-camera systems

Meant for high speed!
- **400 fps** on i7 laptops
- **100 fps** on smartphone PC

Edgelet **Corner**



Download from http://rpg.ifi.uzh.ch/svo2.html

[Forster, Pizzoli, Scaramuzza, SVO: Fast, Semi-Direct Visual Odometry ICRA'14, TRO'17]

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

	Mean	St.D.	CPU@20 fps
SVO Mono	2.53	0.42	55 $\pm$10%
ORB Mono SLAM (No loop closure)	29.81	5.67	187 $\pm$32%
LSD Mono SLAM (No loop closure)	23.23	5.87	236 $\pm$37%
DSO	20.12	4.03	181 $\pm$27%

TABLE II: The first and second column report mean and standard devitation of the processing time in milliseconds on a laptop with an Intel Core i7 (2.80 GHz) processor. Since all algorithms use multi-threading, the third column reports the average CPU load when providing new images at a constant rate of 20 Hz.

[Forster, et al., SVO: Semi Direct Visual Odometry for Monocular and Multi-Camera Systems, TRO'17]

Processing Times of SVO

- **Laptop** (Intel i7, 2.8 GHz): up to 400 fps
- **Smartphone**, ARM Cortex-A9, 1.7 GHz (Odroid): Up to 100 fps

Timing results on an Intel Core i7 (2.80 GHz) laptop processor:

	Thread	Intel i7 [ms]
Sparse image alignment	1	0.66
Feature alignment	1	1.04
Optimize pose & landmarks	1	0.42
Extract features	2	1.64
Update depth filters	2	1.80

[Forster, et al., SVO: Semi Direct Visual Odometry for Monocular and Multi-Camera Systems, TRO’17]

Applications of SVO

Position error: 5 mm, height: 1.5 m – Down-looking camera

Speed: 4 m/s, height: 3 m – Down-looking camera

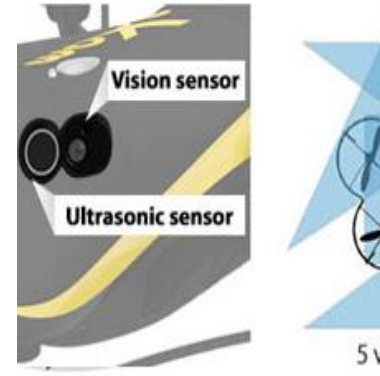
Robustness to dynamic scenes (down-looking camera)

Automatic recovery from aggressive flight [ICRA'15]

[ICRA'10-17, AURO'12, RAM'14, JFR'15, RAL'17]

Parrot: Autonomous Inspection of Bridges and Power Masts

Albris drone



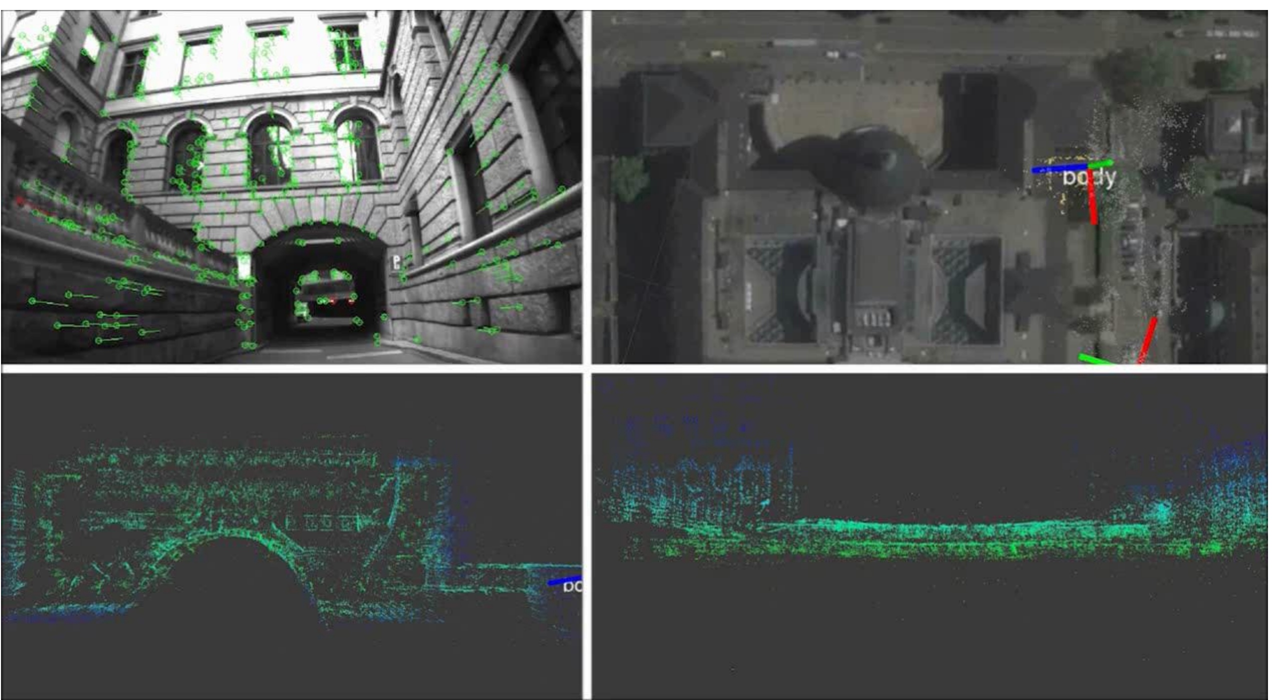
Dacuda 3D (now Magic Leap Zurich)

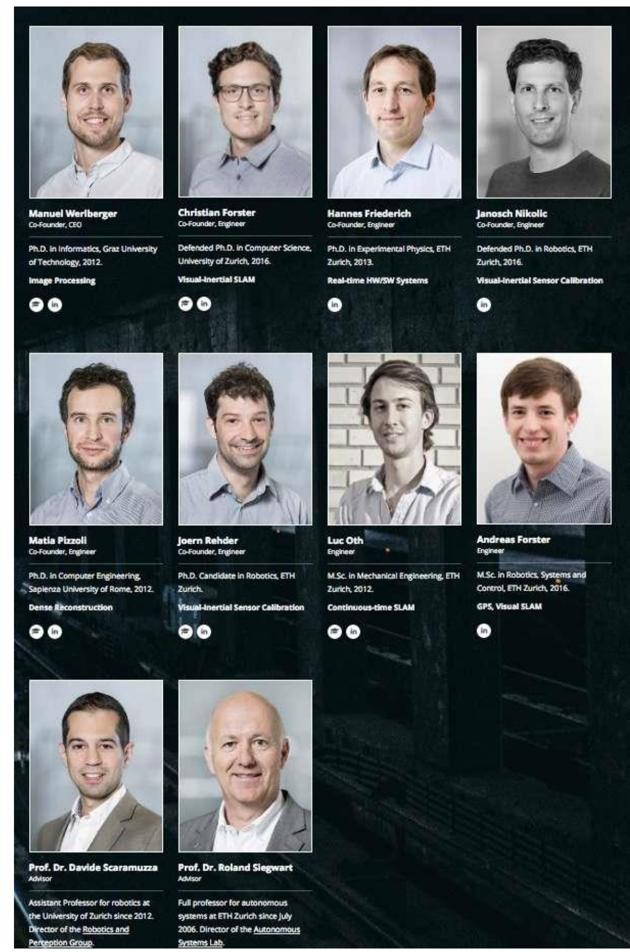
- Fully immersive VR (running on iPhone)
- Powered by SVO

Dacuda's 3D divison ➔ magic leap

Zurich-Eye, first Wyss Zurich project, now Facebook-Oculus Zurich

- Vision-based Localization and Mapping Solutions for Mobile Robots
- Created in Sep. 2015, **became Facebook-Oculus Zurich in Sep. 2016**
- **The Zurich Eye team is behind the new Oculus Quest**





Zurich-Eye, first Wyss Zurich project, now Facebook-Oculus Zurich

- Vision-based Localization and Mapping Solutions for Mobile Robots
- Created in Sep. 2015, **became Facebook-Oculus Zurich in Sep. 2016**
- **The Zurich Eye team is behind the new Oculus Quest**

Understanding Check

Are you able to answer the following questions:

- Are you able to define Bundle Adjustment (via mathematical expression and illustration)?
- Are you able to describe hierarchical and sequential SFM for monocular VO?
- 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?
- Are you able to provide a list of the most popular open source VO and VSLAM algorithms?
- Are you able to describe the differences between feature-based methods and direct methods?