Robotic Mapping & Localization

Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lec15: State-of-the-art SLAM Systems

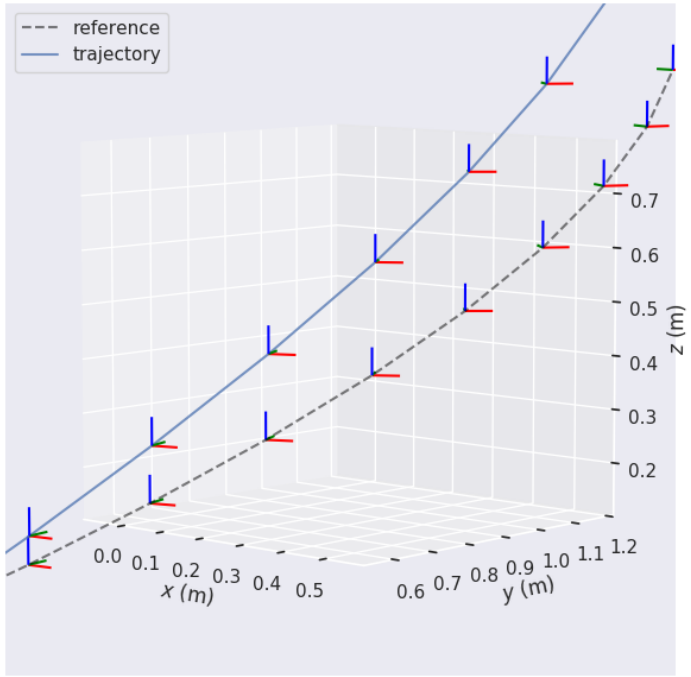
*Courtesy of many people!

SLAM Accuracy Metrics

ATE & RTE

- **Absolute Trajectory Error** (**ATE**) measures the difference between estimated trajectory & ground truth trajectory.

- **ATE** is computed as root mean square error (RMSE) between the ***aligned*** estimated & ground trajectory ***positions*** (not orientations)



- **Relative Trajectory Error** (**RTE**) evaluates error in ***relative*** trajectory, that is, the difference in position change from time t to $t + \Delta t$ in the estimated and ground truth trajectories

- **Absolute** or **Relative Pose Error**s (APE, RPE), are defined similarly, but use pose (position & orientation) to compute the error (instead of only position)

ATE & RTE

2.1 Absolute trajectory error(ATE)

The global consistency can be evaluated by comparing the absolute distances between estimated and ground truth trajectory. As both trajectories can be specified in arbitrary coordinate frames, they first need to be aligned. This can be achieved by using the Horn method [5], which finds the rigid-body transformation S. Let us define absolute trajectory error matrix at time i as:

$$E_i := Q_i^{-1} S P_i$$

The ATE is defined as the root mean square error from error matrices:

$$\mathrm{ATE}_{rmse} = \left(\frac{1}{n} \sum_{i=1}^{n} \| \mathrm{trans}(E_i) \|^2 \right)^{\frac{1}{2}}$$

Translation component

Usually, mean or median values are computed. Actually, absolute trajectory error is the average deviation from ground truth trajectory per frame.

2.2 Relative pose error(RPE)

The relative pose error measures the local accuracy of the trajectory over a fixed time interval Δ. Therefore, the relative pose error corresponds to the drift of the trajectory which is in particular useful for the evaluation of visual odometry systems. Let us define the relative pose error matrix at time step i as:

$$F_i^{\Delta} := (Q_i^{-1} Q_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta})$$

from a sequence of n camera poses we obtain $m = n - \Delta$ individual relative pose error matrices along the sequence. The RPE is usually divided into translation and rotation components. Similar to the absolute trajectory error, we propose to evaluate the root mean squared error over all time indicies for RPE translation error:

$$\mathrm{RPE}_{trans}^{i,\Delta} = \left(\frac{1}{m} \sum_{i=1}^{m} \| \mathrm{trans}(F_i) \|^2 \right)^{\frac{1}{2}}$$

As for rotation component we use mean error approach:

$$\mathrm{RPE}_{rot}^{i,\Delta} = \frac{1}{m} \sum_{i=1}^{m} \angle(\mathrm{rot}(F_i^{\Delta}))$$

For the evaluation of SLAM systems, it makes sense to average over all possible pairs in both translation and rotation component.

Reference: Measuring robustness of Visual SLAM https://arxiv.org/pdf/1910.04755.pdf

ATE & RTE - Example

- **Example**: Kiemra (https://arxiv.org/pdf/1910.02490.pdf)

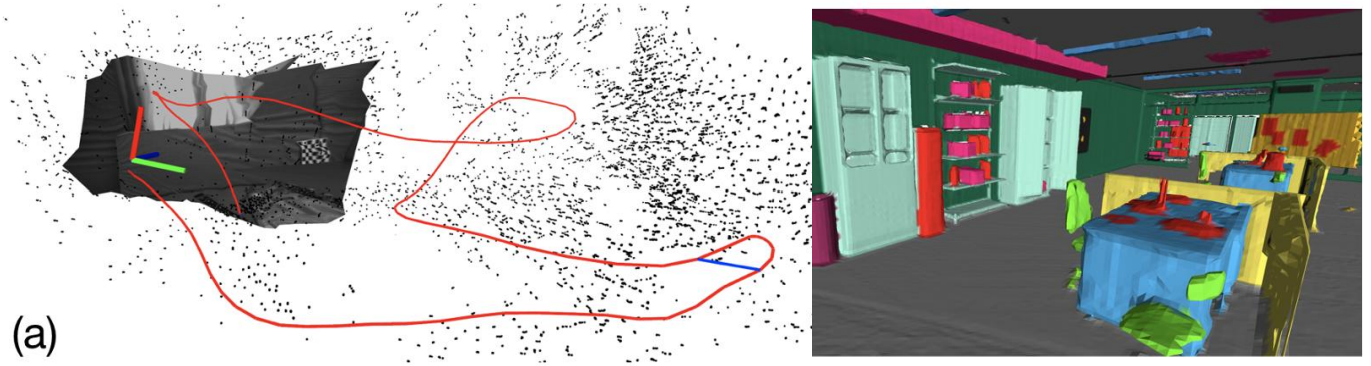


TABLE II: RMSE of state-of-the-art VIO pipelines (reported from [77] and [24]) compared to Kimera, on the EuRoC dataset. In **bold** the best result for each category: fixed-lag smoothing, full smoothing, and PGO with loop closure. × indicates failure.

	RMSE ATE [m]								
	Fixed-lag Smoothing					Full Smoothing		Loop Closure	
Seq.	OKVIS	MSCKF	ROVIO	VINS-Mono	**Kimera-VIO**	SVO-GTSAM	**Kimera-VIO**	VINS-LC	**Kimera-RPGO**
MH_1	0.16	0.42	0.21	0.15	**0.11**	0.05	**0.04**	0.12	**0.08**
MH_2	0.22	0.45	0.25	0.15	**0.10**	**0.03**	0.07	0.12	**0.09**
MH_3	0.24	0.23	0.25	0.22	**0.16**	**0.12**	**0.12**	0.13	**0.11**
MH_4	0.34	0.37	0.49	0.32	**0.24**	**0.13**	0.27	0.18	**0.15**
MH_5	0.47	0.48	0.52	**0.30**	0.35	**0.16**	0.20	**0.21**	0.24
V1_1	0.09	0.34	0.10	0.08	**0.05**	0.07	**0.06**	0.06	**0.05**
V1_2	0.20	0.20	0.10	0.11	**0.08**	0.11	**0.07**	**0.08**	0.11
V1_3	0.24	0.67	0.14	0.18	**0.07**	×	**0.09**	0.19	**0.12**
V2_1	0.13	0.10	0.12	**0.08**	**0.08**	**0.07**	**0.07**	0.08	**0.07**
V2_2	0.16	0.16	0.14	0.16	**0.10**	×	**0.09**	0.16	**0.10**
V2_3	0.29	1.13	**0.14**	0.27	0.21	×	**0.19**	1.39	**0.19**

TABLE III: RMSE ATE [m] vs. loop closure threshold α (V1_01).

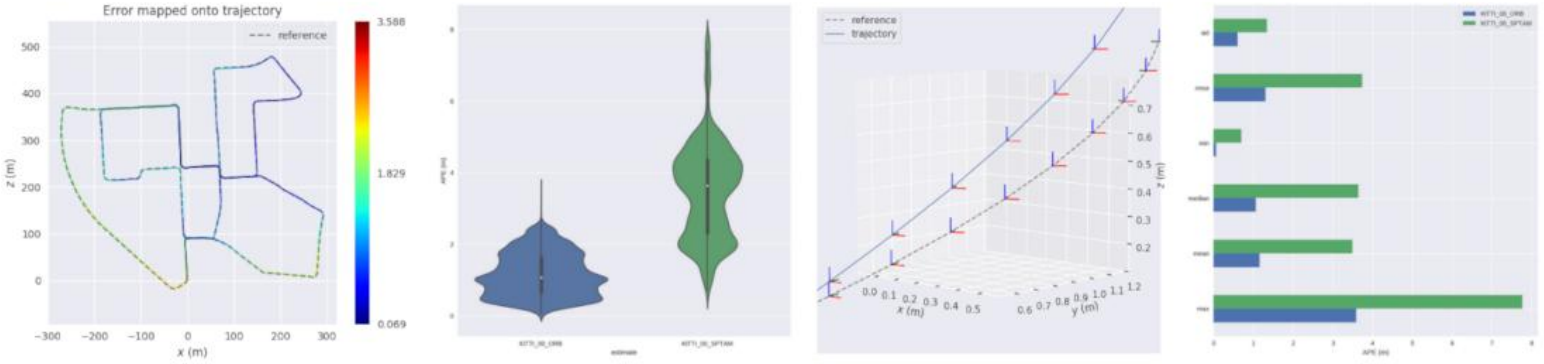
	$\alpha=10$	$\alpha=1$	$\alpha=0.1$	$\alpha=0.01$	$\alpha=0.001$
PGO w/o PCM	0.05	0.45	1.74	1.59	1.59
Kimera-RPGO	0.05	0.05	0.05	0.045	0.049

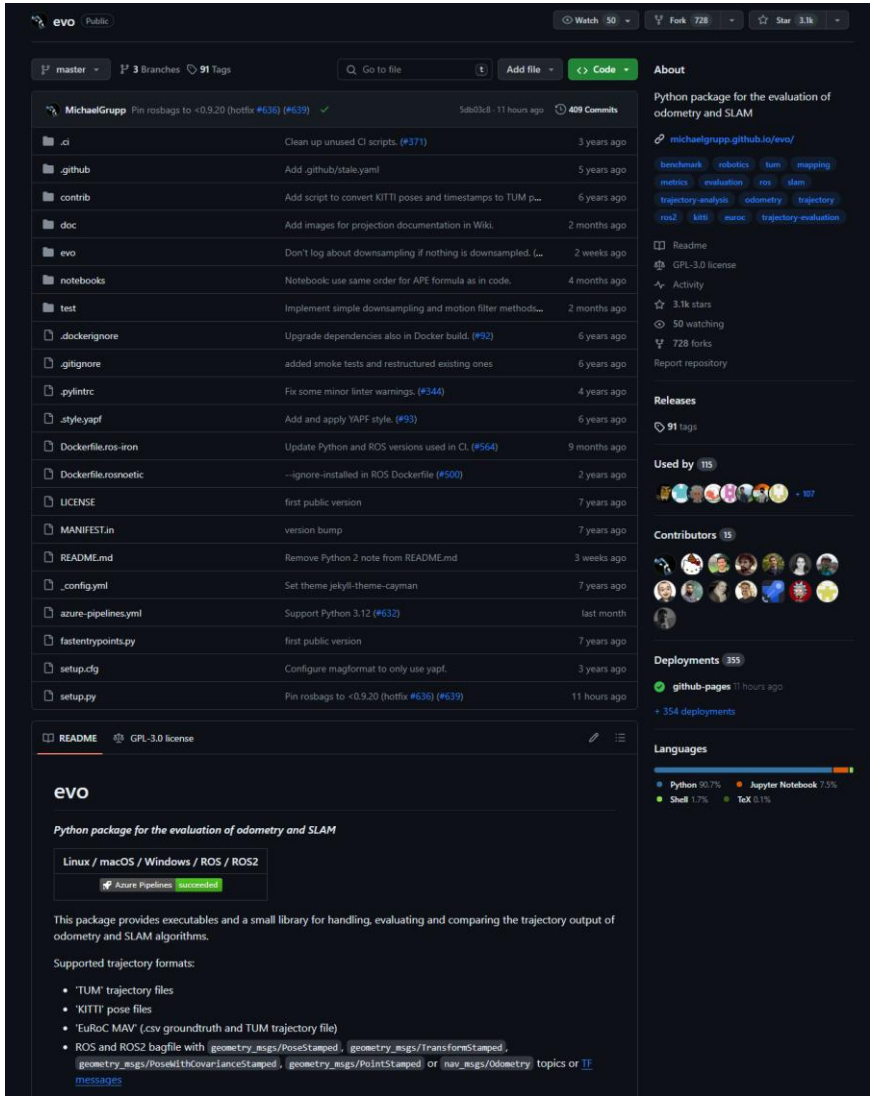
TABLE V: Evaluation of Kimera-Semantics.

Metrics		Kimera-Semantics using: GT Depth GT Poses	GT Depth Kimera-VIO	Dense-Stereo Kimera-VIO
Semantic	mIoU [%]	80.10	80.03	57.23
	Acc [%]	94.68	94.50	80.74
Geometric	ATE [m]	0.0	0.04	0.04
	RMSE [m]	0.079	0.131	0.215

ATE & RTE - Tools

- **EVO**: Python package for the evaluation of odometry and SLAM
- Code: https://github.com/MichaelGrupp/evo

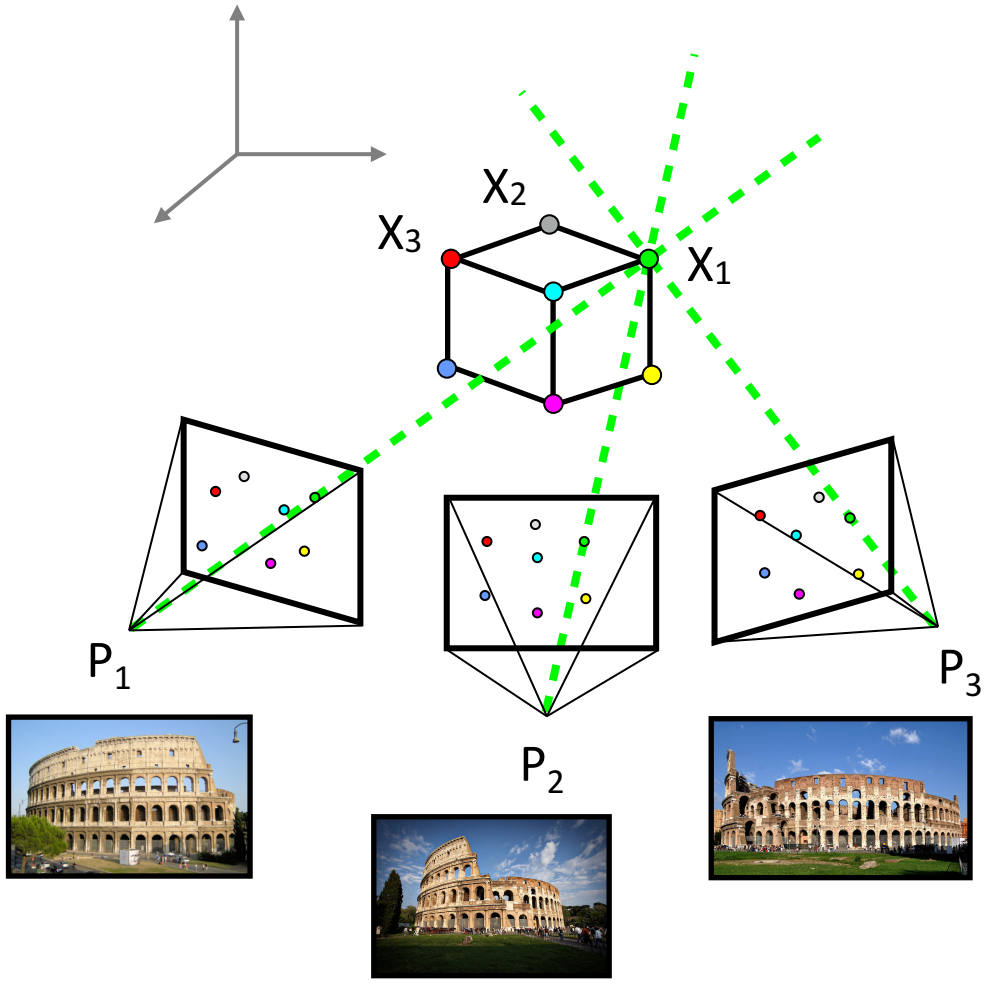




Structure from Motion (SfM)

Structure-from-Motion

- Joint estimation of …
  - Structure $\mathbf{X}_i$
  - Cameras $\mathbf{P}_j$
- … from motion, i.e.
  - images at different viewpoints

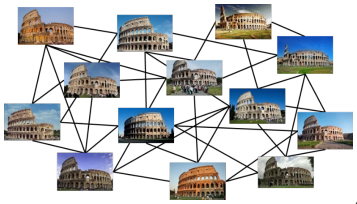


Pipeline

Unstructured Images

Assoc.

Scene Graph



SFM

Sparse Model

MVS

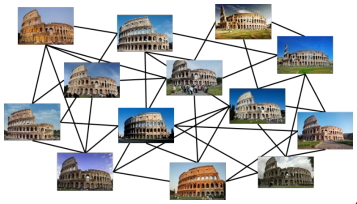
Dense Model

Pipeline

Unstructured Images

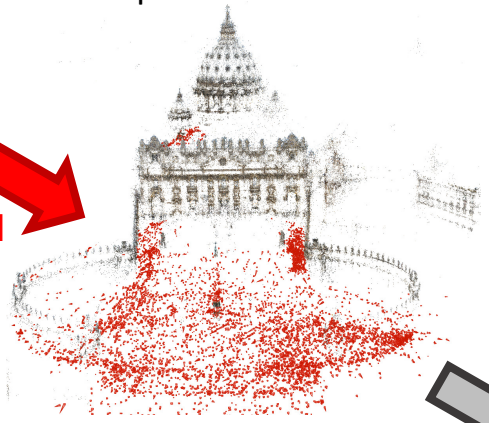
Assoc.

Scene Graph



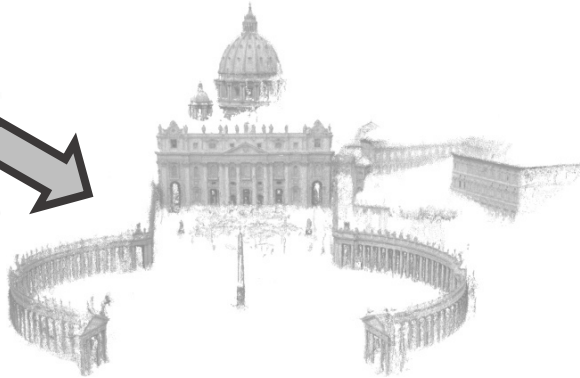
SFM

Sparse Model



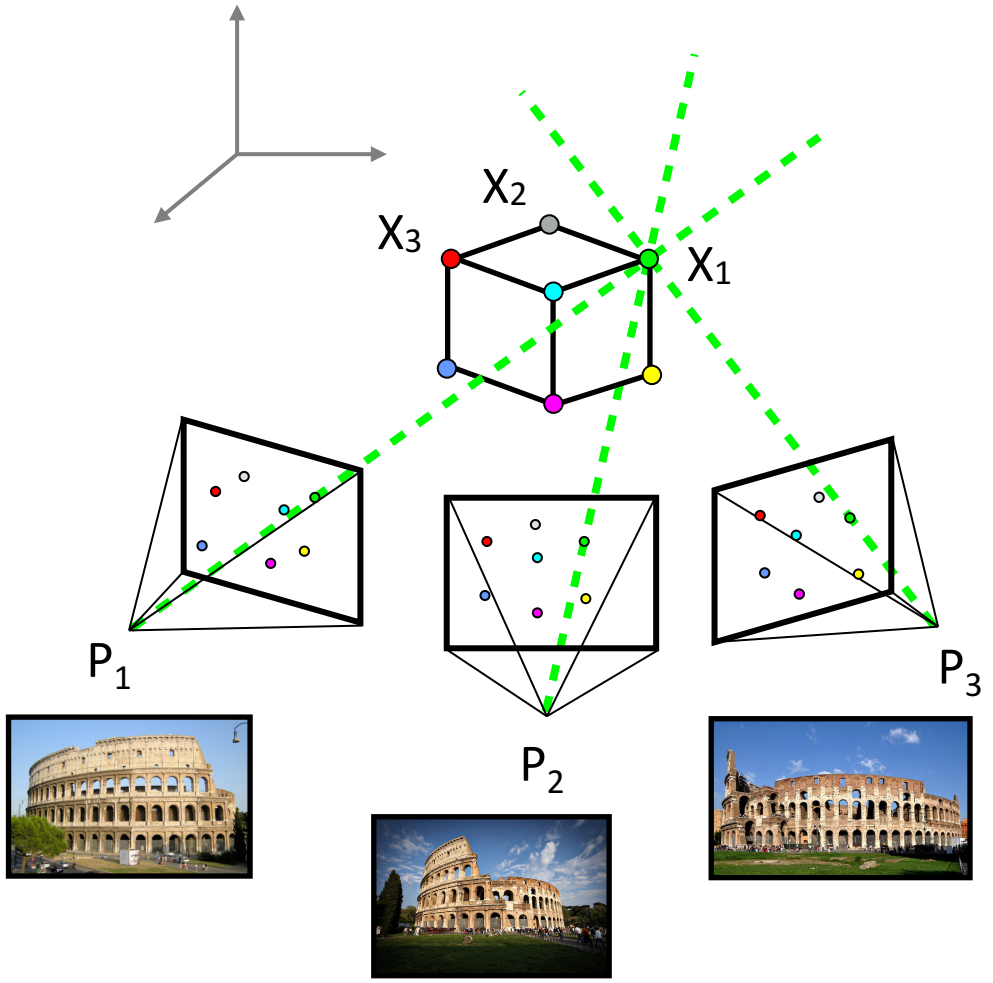
MVS

Dense Model



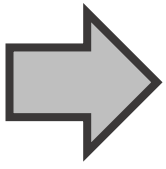
Structure-from-Motion

- Joint estimation of …
  - Structure $\mathbf{X}_i$
  - Cameras $\mathbf{P}_j$
- … from motion, i.e.
  - images at different viewpoints



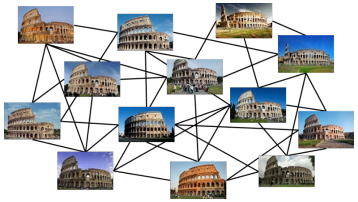
Data Association

Unstructured Images

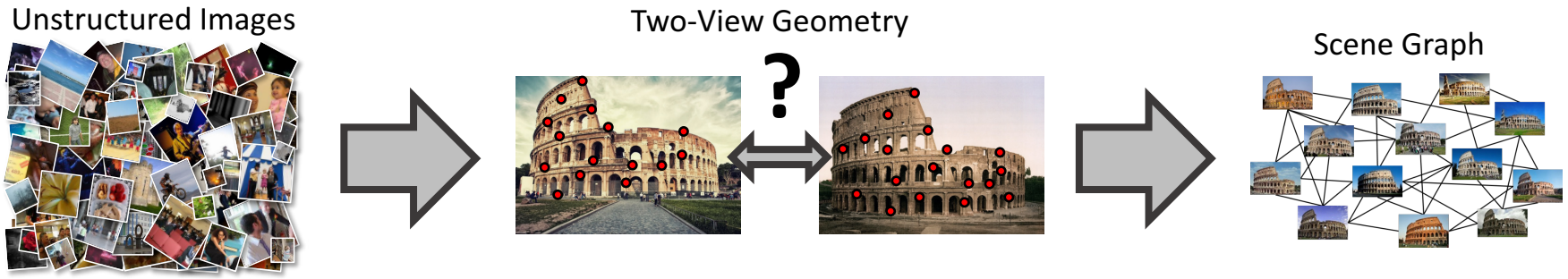


Connected Components

Scene Graph

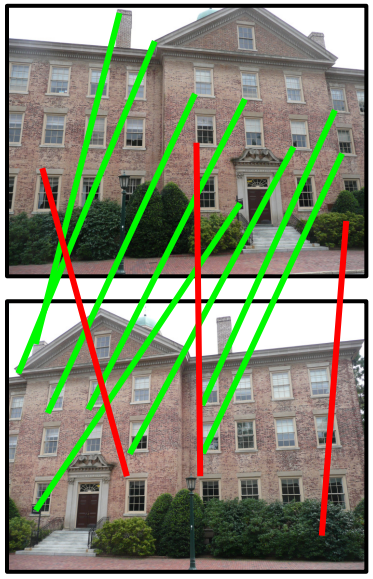


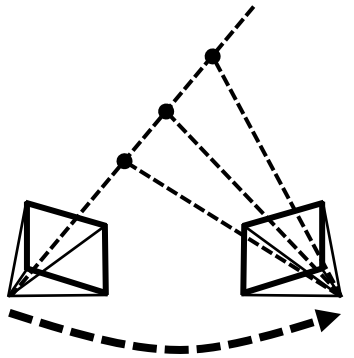
Data Association



Data Association

1. Feature Extraction → 2. Feature Matching → 3. Geometric Verification

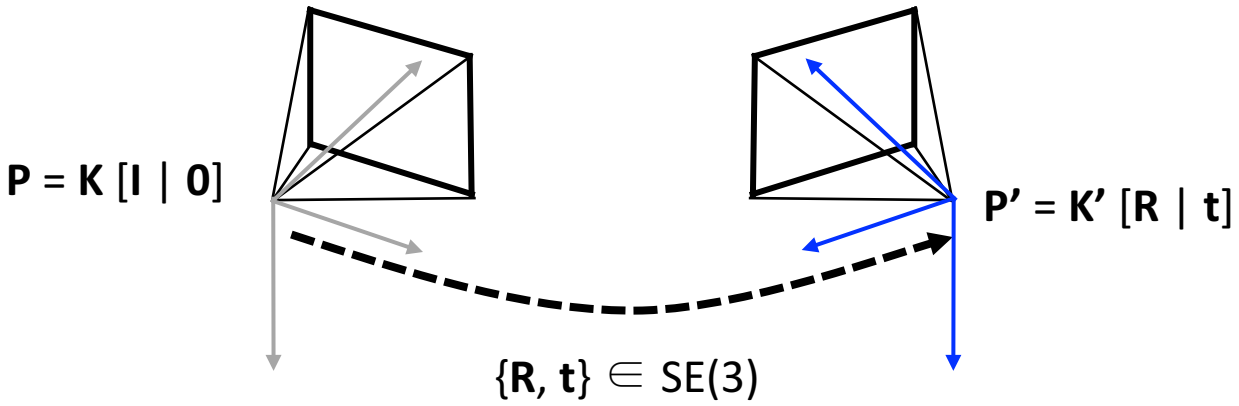




Robust Two-View Geometry Estimation

Two-View Geometry

- Relative camera geometry: **P**, **P'**

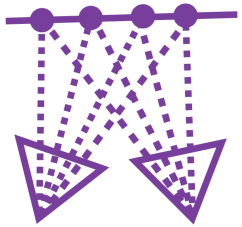


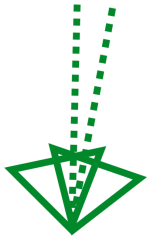
Scale of **t** unknown!

Two-View Geometry

- Model selection

General	Planar	Panoramic
• Fundamental matrix F (*uncalibrated*) • Essential matrix E (*calibrated*)	• Homography H	• Homography H
• 7 correspondences • 5 correspondences	• 4 correspondences	• 4 correspondences

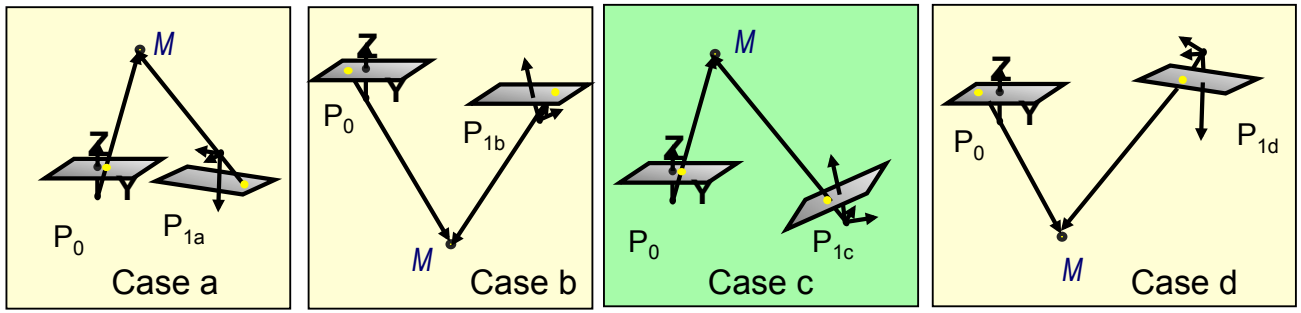




Hartley and Zisserman 2004, “Multiple View Geometry”

Two-View Geometry

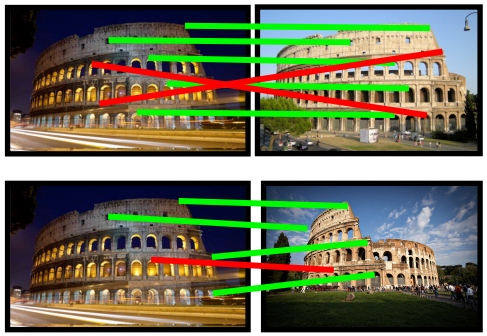
- Robust estimation using RANSAC
- Model selection (**F**, **E**, **H**)
  - Inlier maximization
  - GRIC criterion Torr 1998, “An assessment of information criteria for motion model selection"
  - QDEGSAC Frahm and Pollefeys 2006, “RANSAC for (quasi-) degenerate data (QDEGSAC)”
- Decompose **E**, **H** to {**R**, **t**}

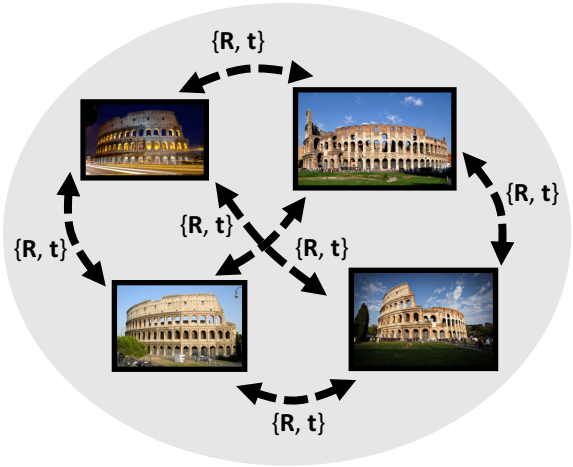


Scene Graph

Unstructured Images

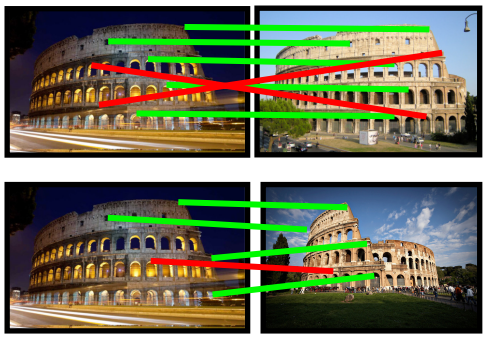
Inlier/outlier correspondences

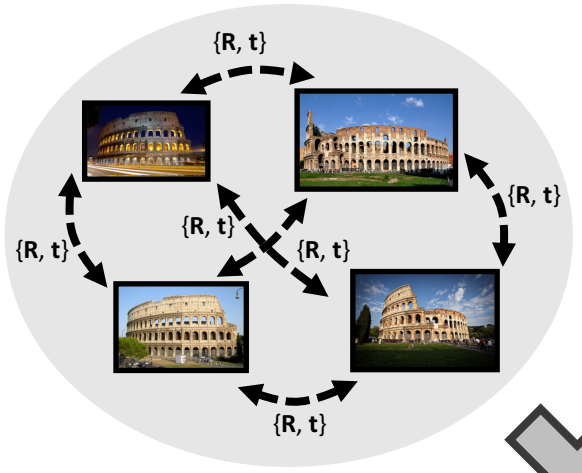


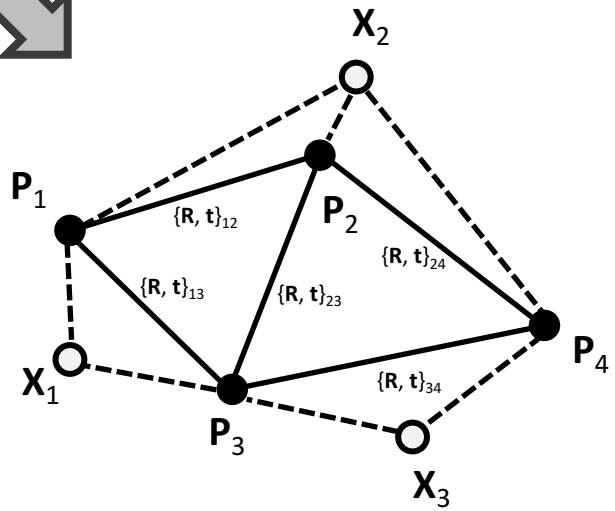


Scene Graph

Inlier/outlier correspondences

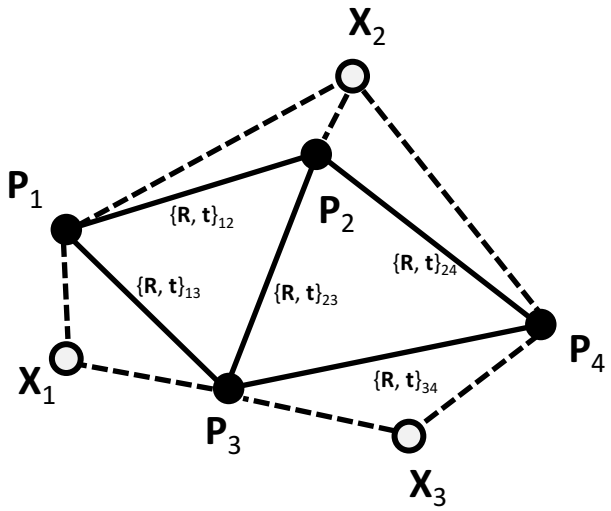


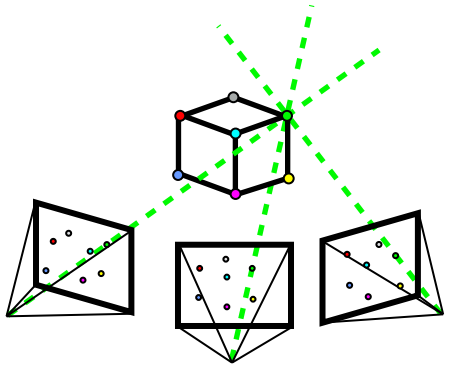




Structure-from-Motion

- From relative to absolute cameras and structure

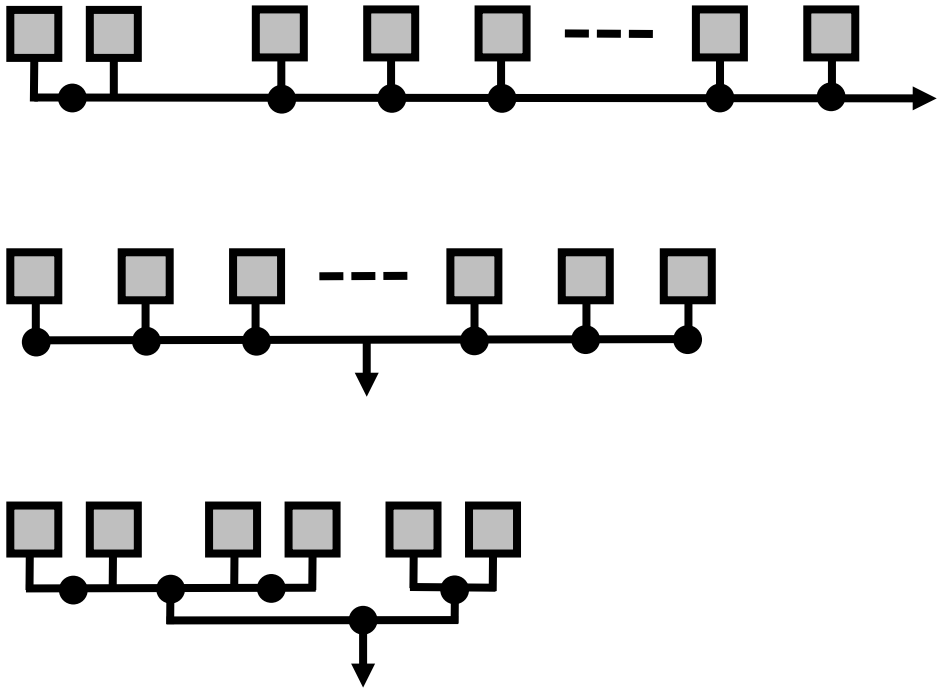




Scale of **t** unknown!
Outlier correspondences!
Outlier image pairs!

Structure-from-Motion

- 3 paradigms
  - Incremental
  - Global
  - Hierarchical



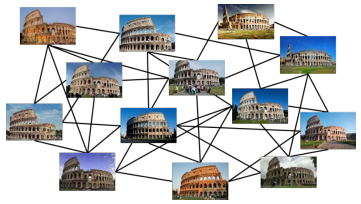
Structure-from-Motion

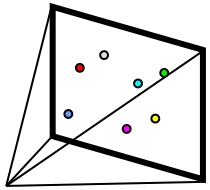
- Comparison

Method	Efficiency	Robustness	Accuracy
Incremental	-	++	+
Global	+	+	+
Hierarchical	++	-	-

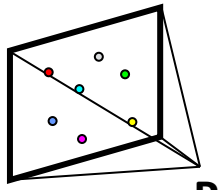
Incremental SfM

- Initialization
  1. Choose two non-panoramic views ($\|t\| \neq 0$)





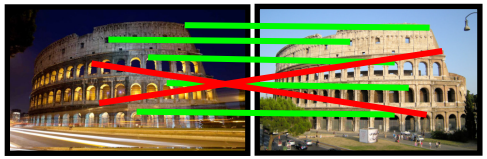
P = **K** [**I** | **0**]

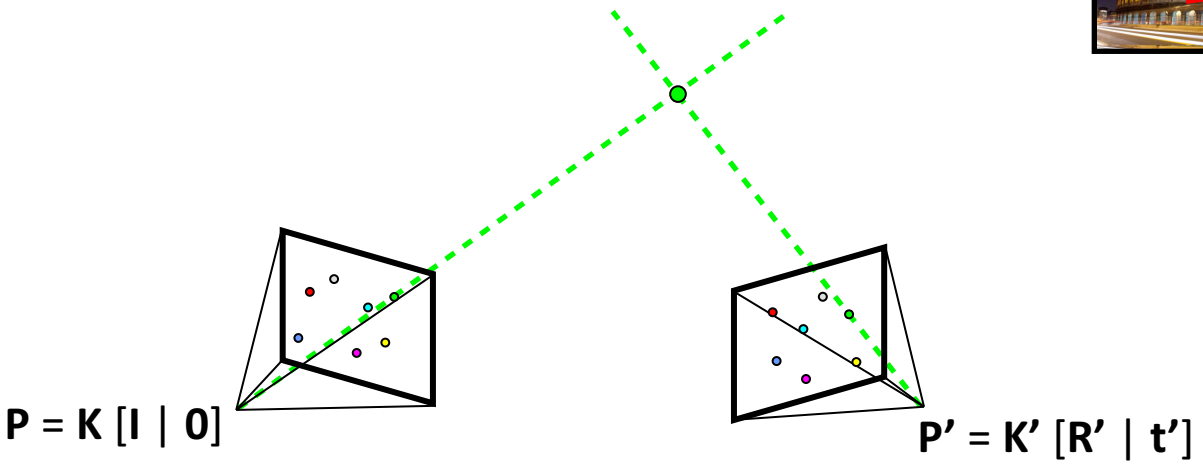


P’ = **K’** [**R’** | **t’**]

Incremental SfM

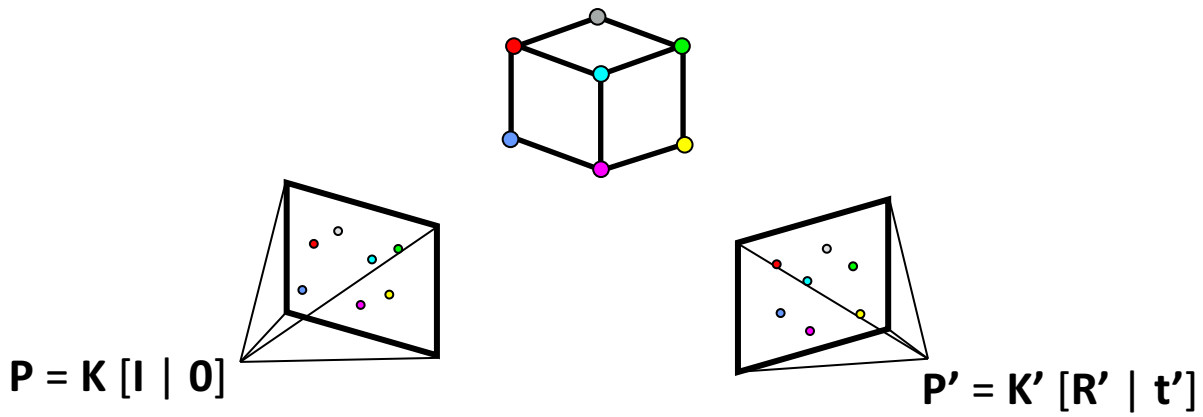
- Initialization
  1. Choose two non-panoramic views ($\|t\| \neq 0$)
  2. Triangulate inlier correspondences





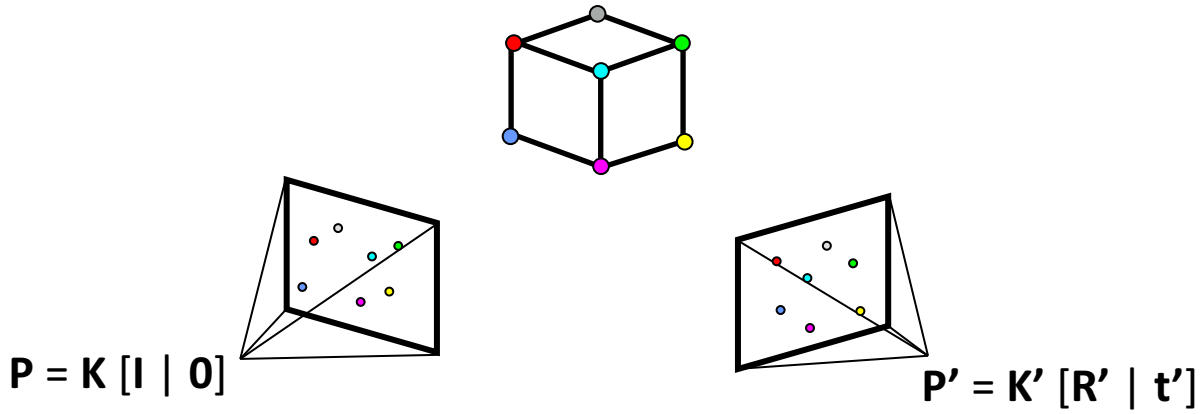
Incremental SfM

- Initialization
  1. Choose two non-panoramic views ($\|t\| = 1$)
  2. Triangulate inlier correspondences



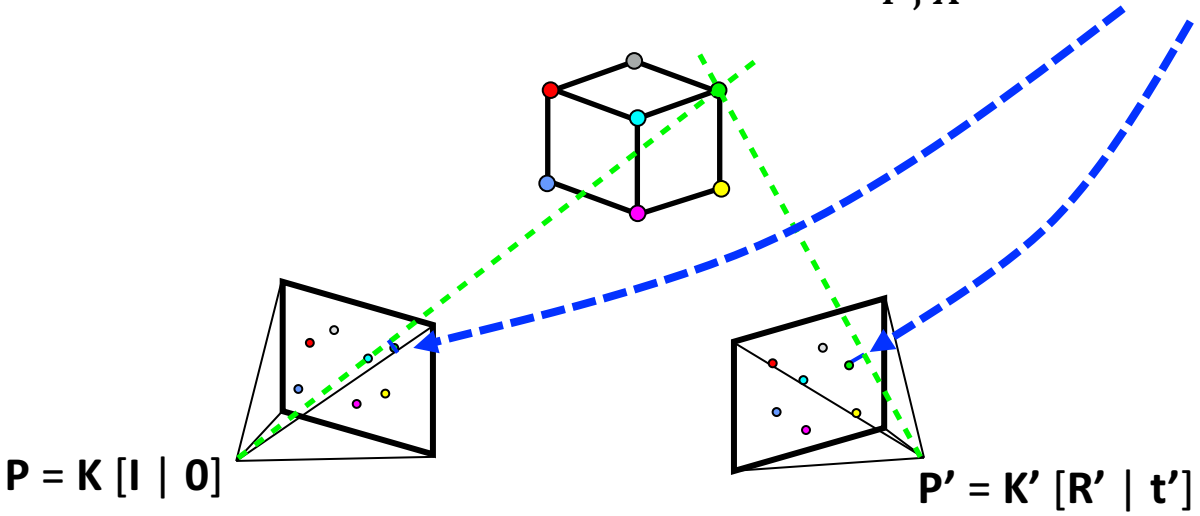
Incremental SfM

- Initialization
  1. Choose two non-panoramic views ($\|t\| = 1$)
  2. Triangulate inlier correspondences
  3. Bundle adjustment



Incremental SfM

- Bundle adjustment
  - Non-linear refinement of structure and motion
  - Minimize reprojection error: $\min_{P,X} \|x - \pi(P,X)\|$

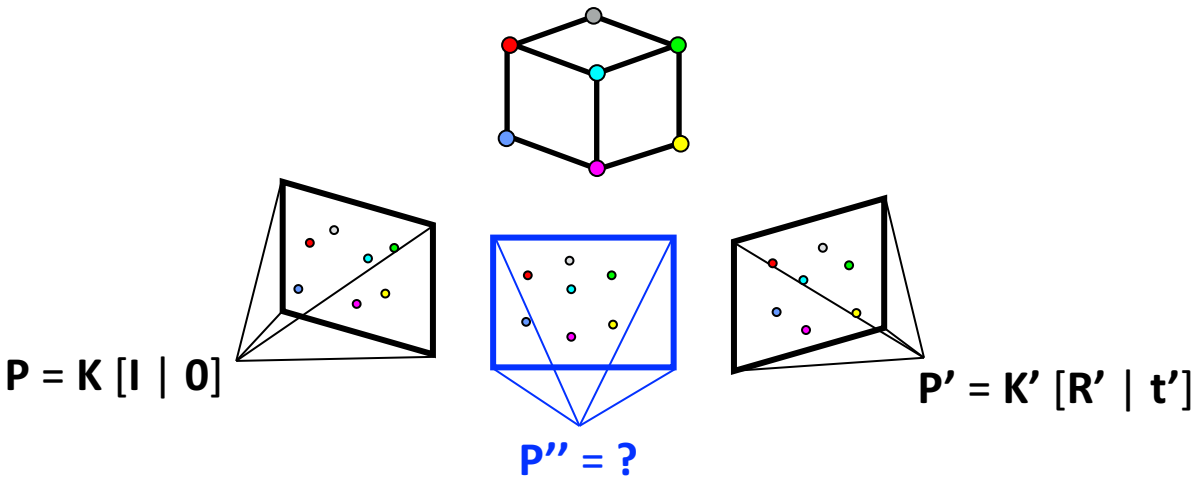


Ceres-Solver, http://ceres-solver.org/

Triggs et al., "Bundle Adjustment – A Modern Synthesis"

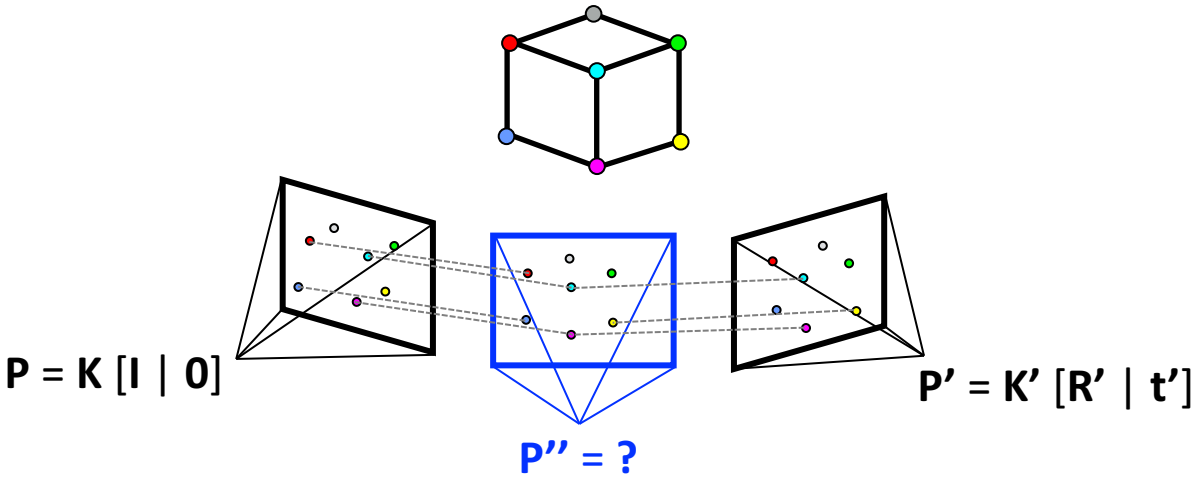
Incremental SfM

- Absolute camera registration



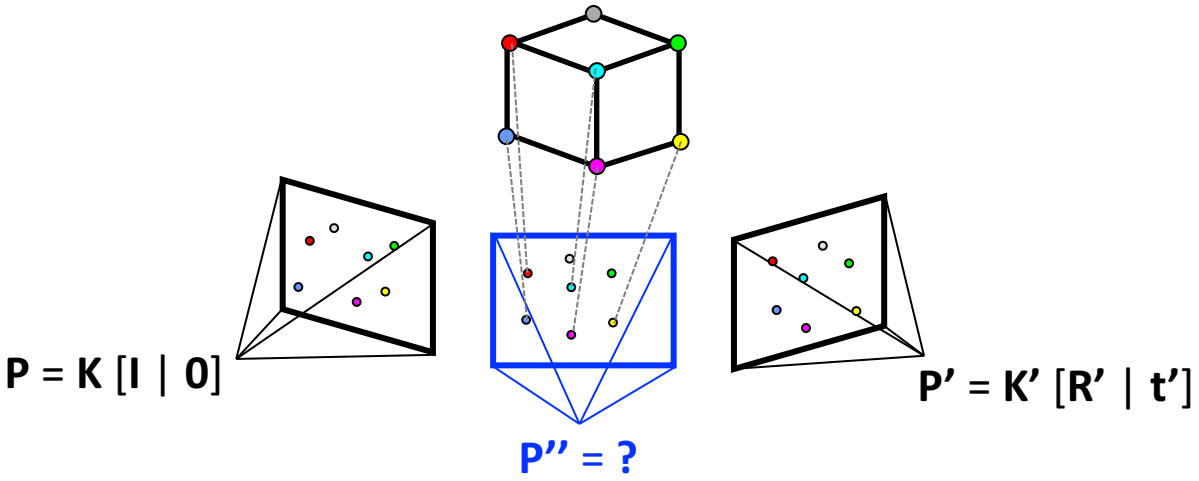
Incremental SfM

- Absolute camera registration
  1. Find 2D-3D correspondences



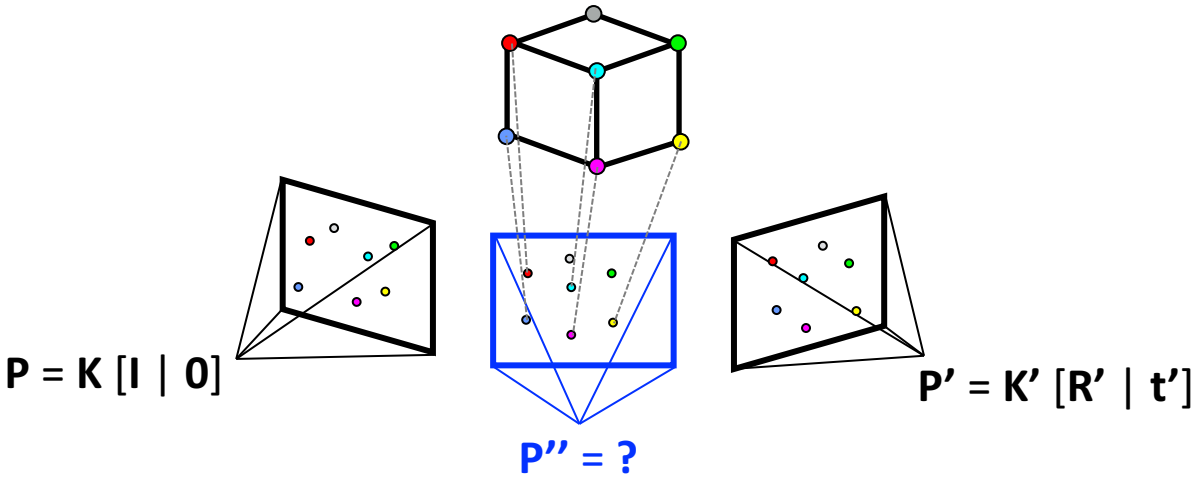
Incremental SfM

- Absolute camera registration
  1. Find 2D-3D correspondences



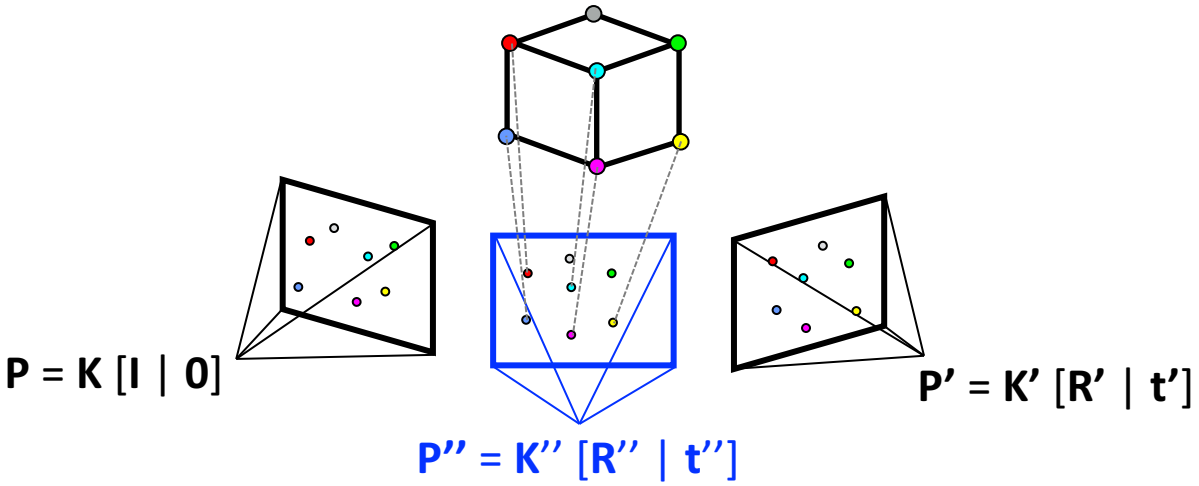
Incremental SfM

- Absolute camera registration
  1. Find 2D-3D correspondences
  2. Solve Perspective-n-Point problem



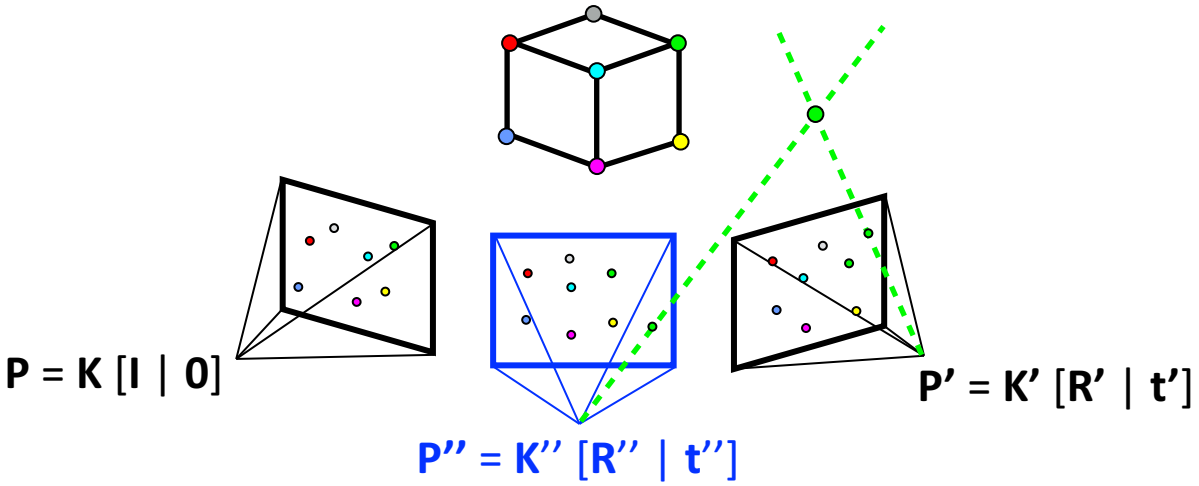
Incremental SfM

- Absolute camera registration
  1. Find 2D-3D correspondences
  2. Solve Perspective-n-Point problem



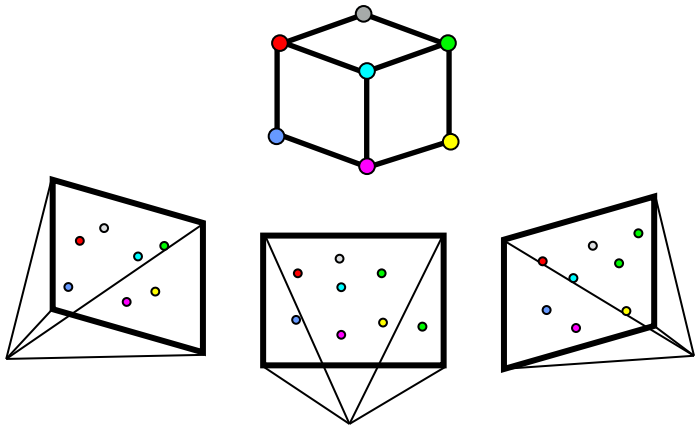
Incremental SfM

- Absolute camera registration
  1. Find 2D-3D correspondences
  2. Solve Perspective-n-Point problem
  3. Triangulate new points



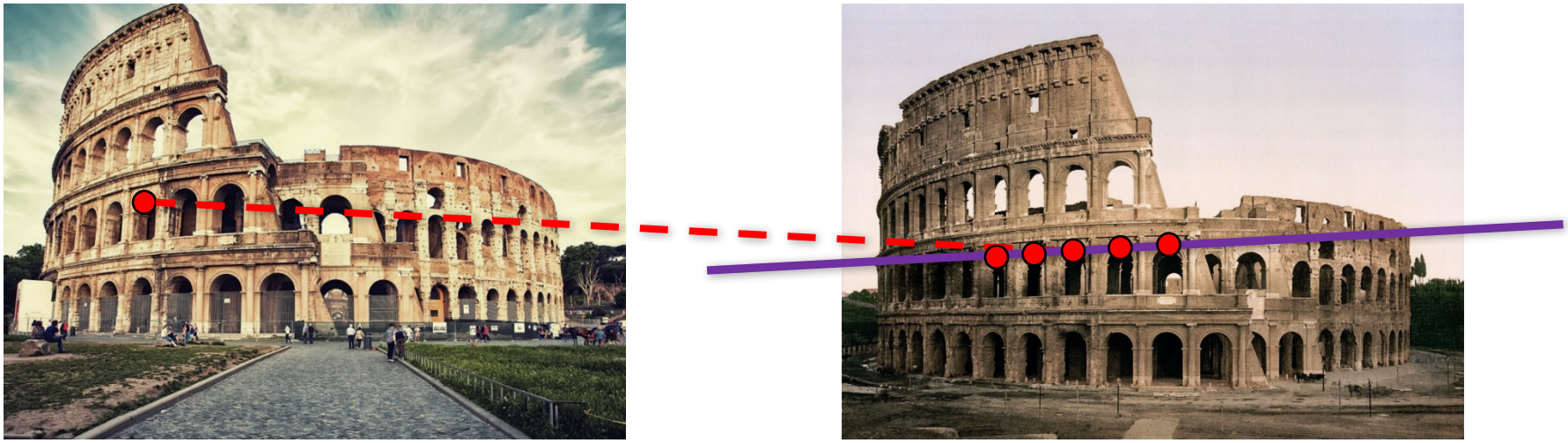
Incremental SfM

- Bundle adjustment $\min_{P,X} \|x - \pi(P,X)\|$



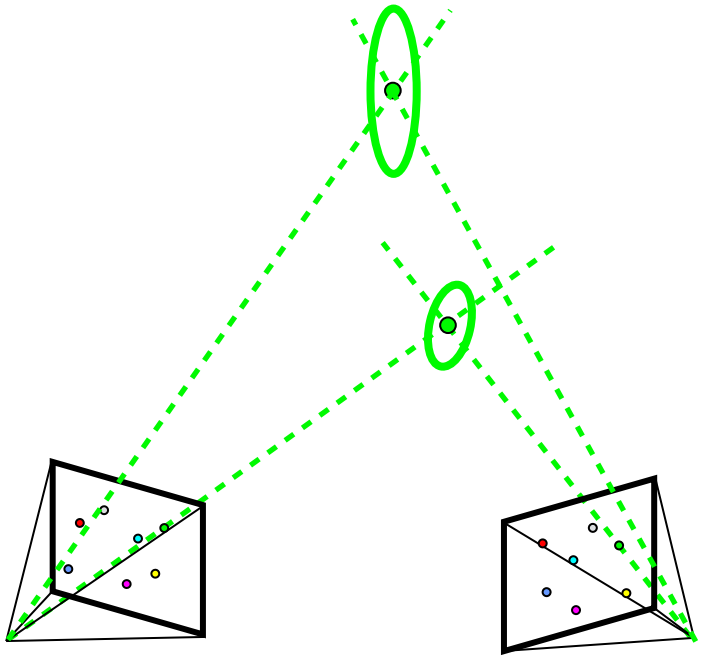
Incremental SfM

- Outlier filtering
  - Remove points with large reprojection error

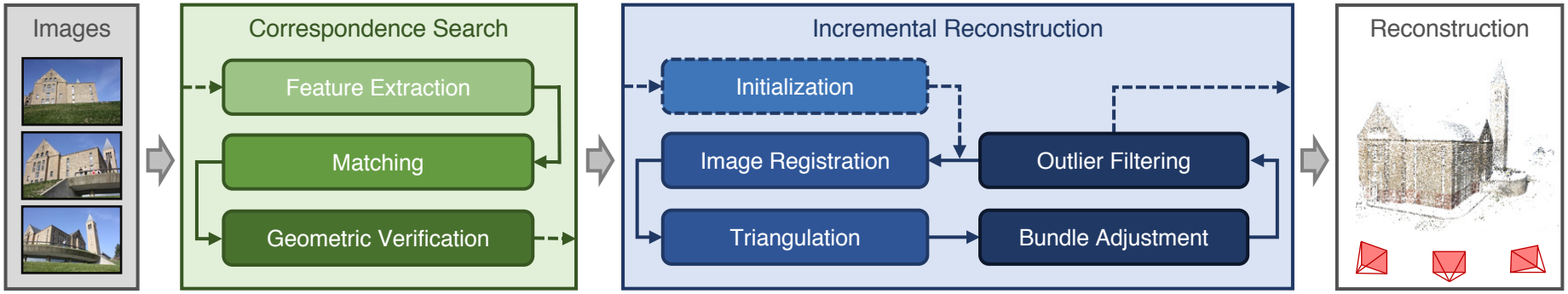


Incremental SfM

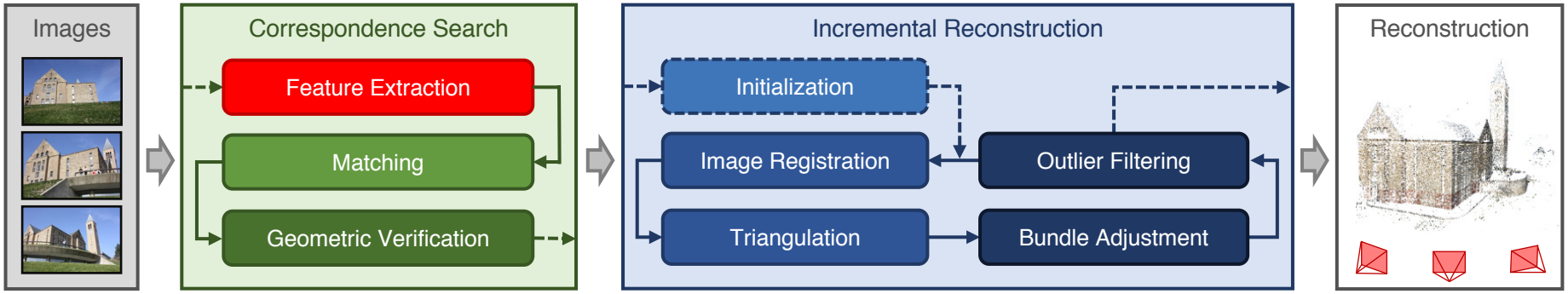
- Outlier filtering
  - Remove points with large reprojection error
  - Remove points at “infinity”



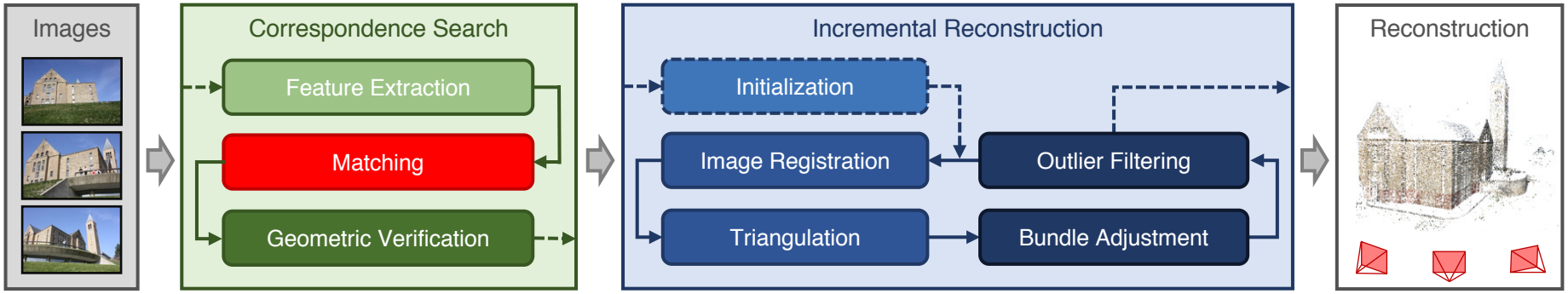
Incremental SfM

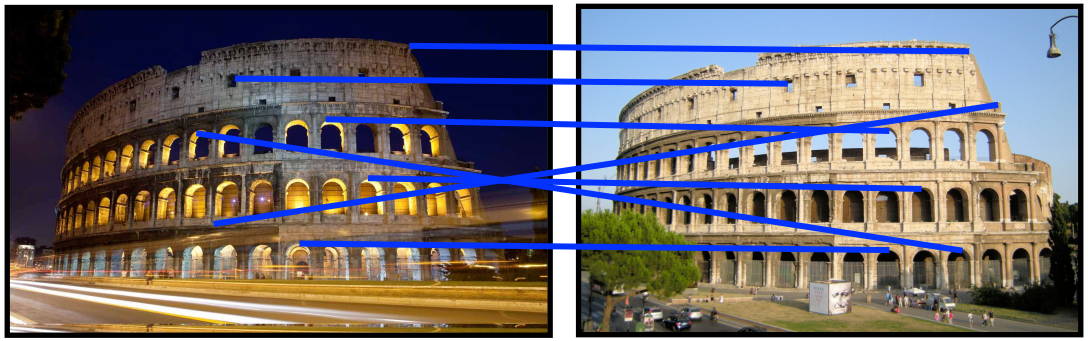


Incremental SfM

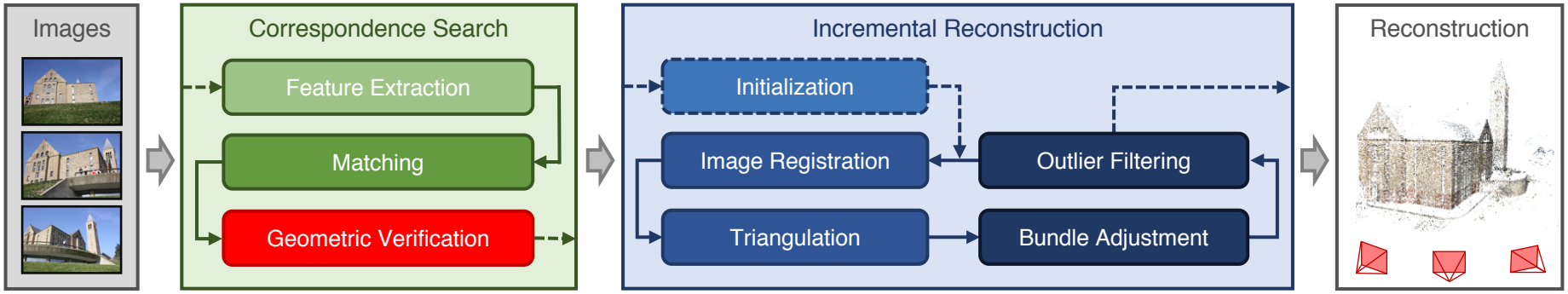


Incremental SfM

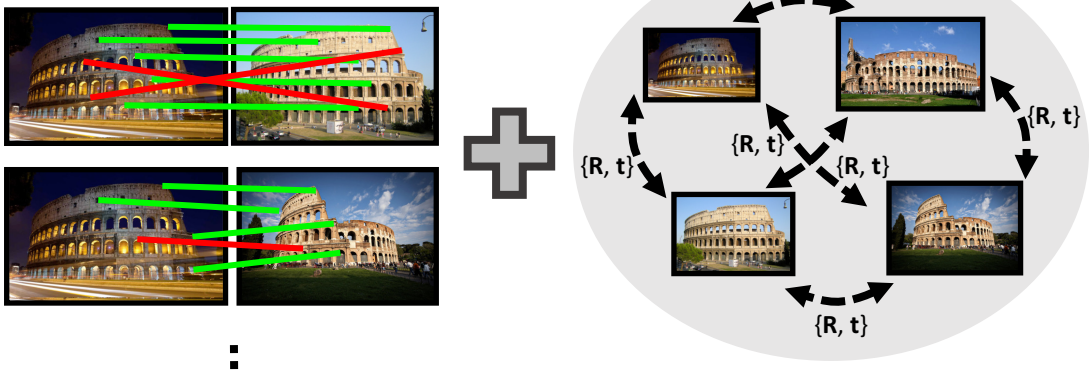




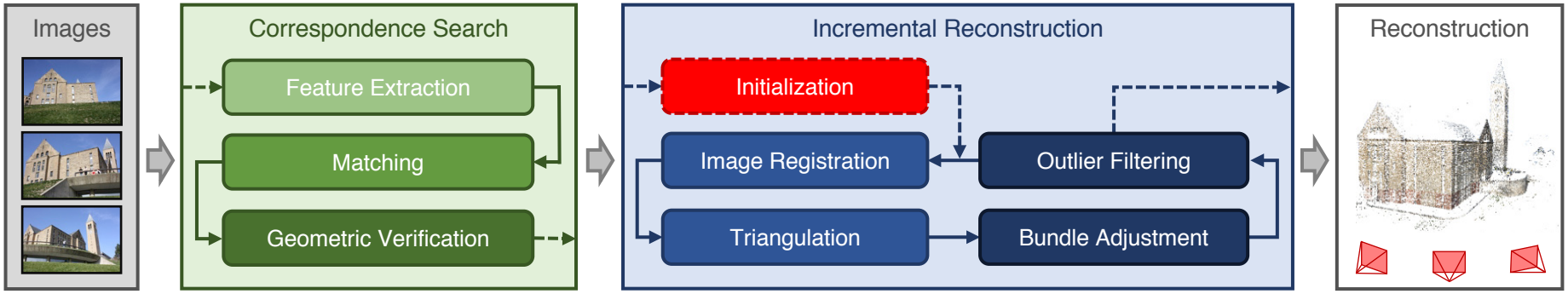
Incremental SfM

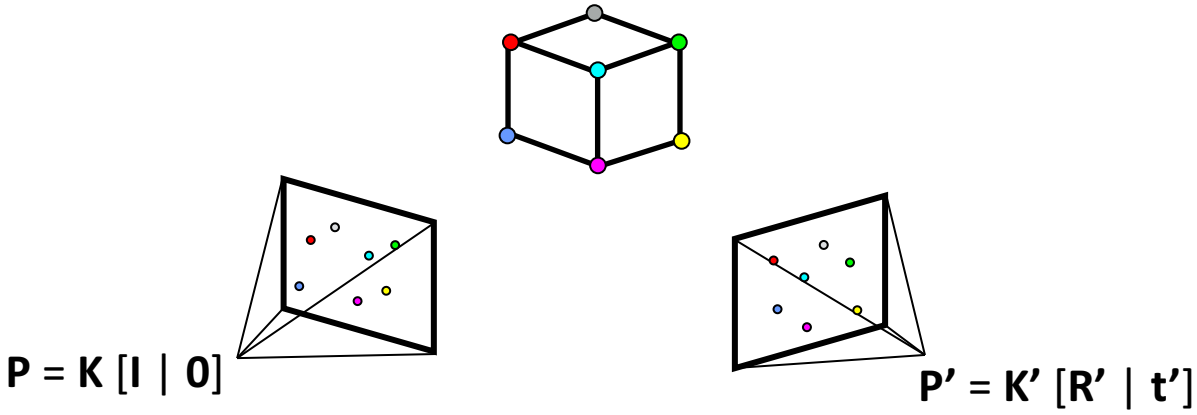


Inlier/outlier correspondences

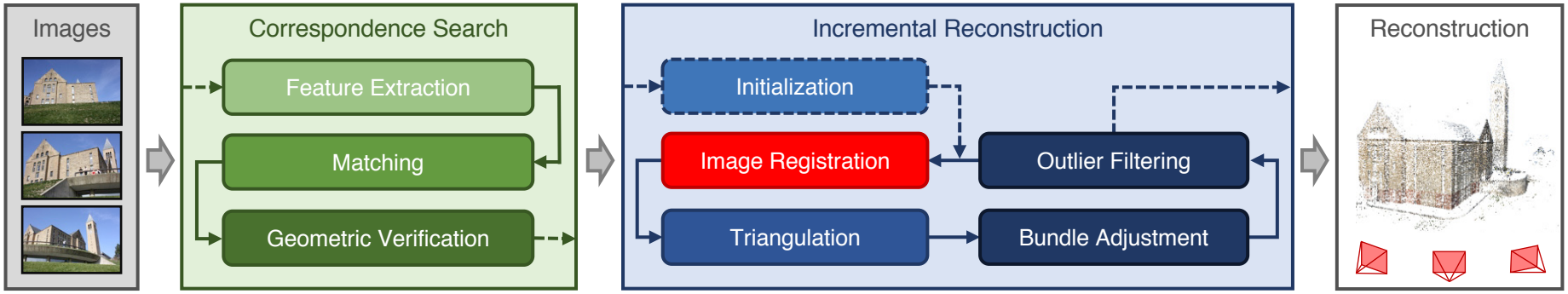


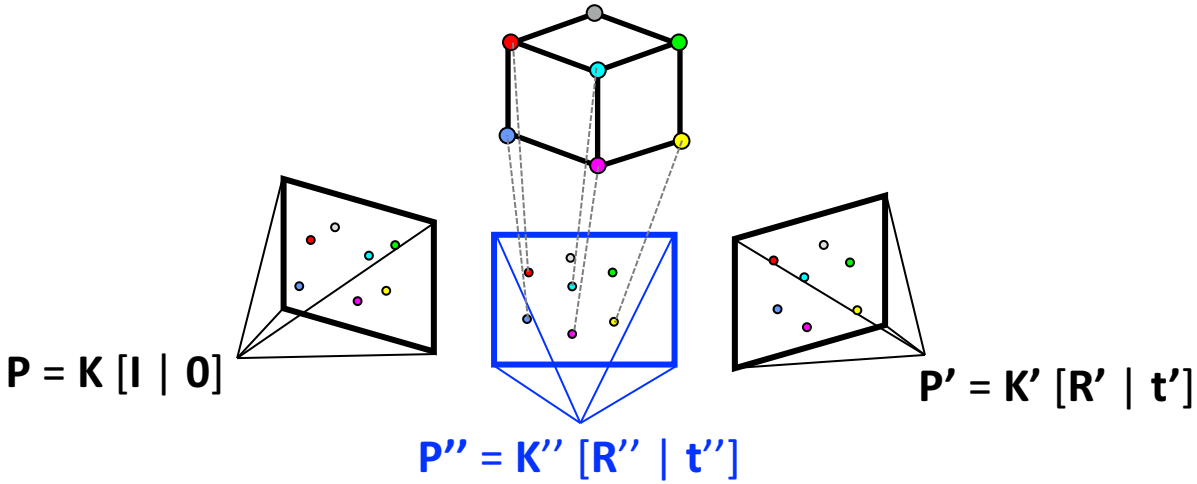
Incremental SfM



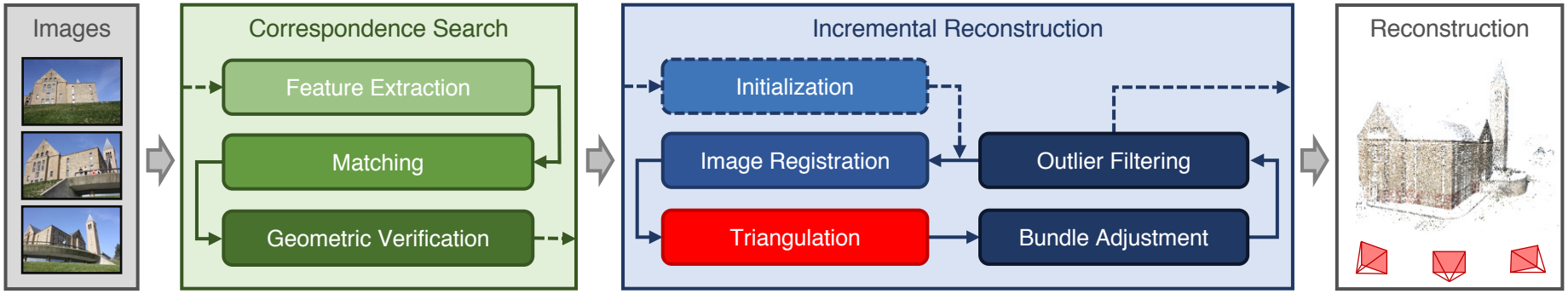


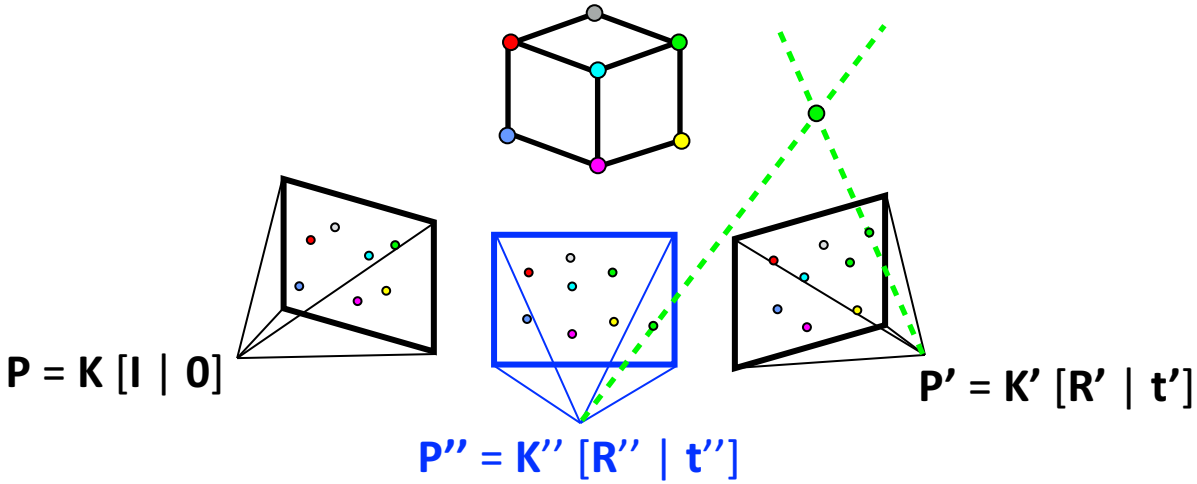
Incremental SfM



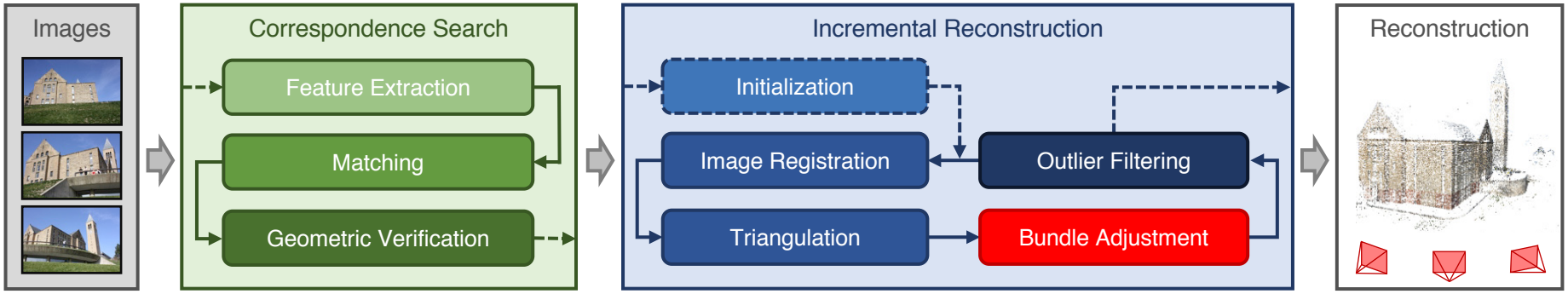


Incremental SfM



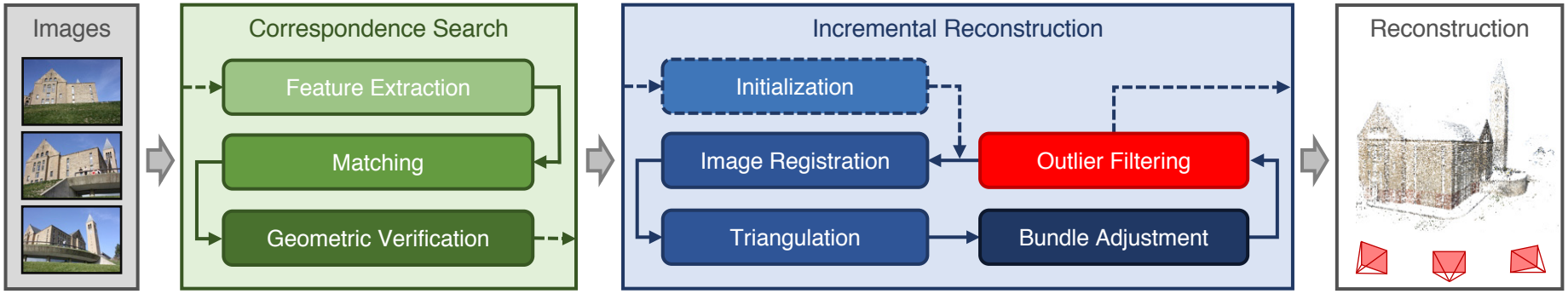


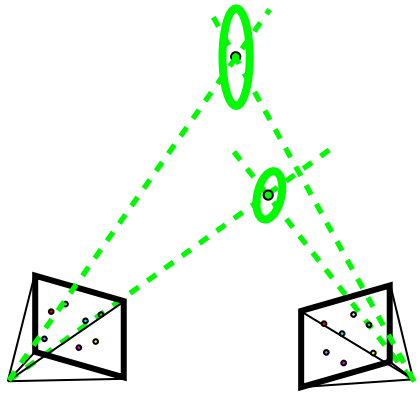
Incremental SfM



$$\min_{P,X} \|x - \pi(P,X)\|$$

Incremental SfM





Monocular SLAM

SfM vs SLAM: Similarities

- Solve for camera poses and 3D scene points given images
- Correspondence, registration, outlier rejection, and bundle adjustment are core problems

SfM vs SLAM: differences

SfM	SLAM
• Input is unordered set of images	• Input is stream of images, stereo, or depth and sometimes IMU
• Focus is on precision, with aim to produce a good 3D model	• Focus is on speed and robustness, with aim to localize camera or robot
• Offline, one-time process	• Online process, possibly with relocalization
• Published mainly in vision conferences	• Published mainly in robotics conferences
• Complex	• Very complex

ORB-SLAM

ORB-SLAM: a Versatile and Accurate Monocular SLAM System

Raúl Mur-Artal*, J. M. M. Montiel, *Member, IEEE,* and Juan D. Tardós, *Member, IEEE,*

***Abstract*—This paper presents ORB-SLAM, a feature-based monocular SLAM system that operates in real time, in small and large, indoor and outdoor environments. The system is robust to severe motion clutter, allows wide baseline loop closing and relocalization, and includes full automatic initialization. Building on excellent algorithms of recent years, we designed from scratch a novel system that uses the same features for all SLAM tasks: tracking, mapping, relocalization, and loop closing. A survival of the fittest strategy that selects the points and keyframes of the reconstruction leads to excellent robustness and generates a compact and trackable map that only grows if the scene content changes, allowing lifelong operation. We present an exhaustive evaluation in 27 sequences from the most popular datasets. ORB-**

ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM

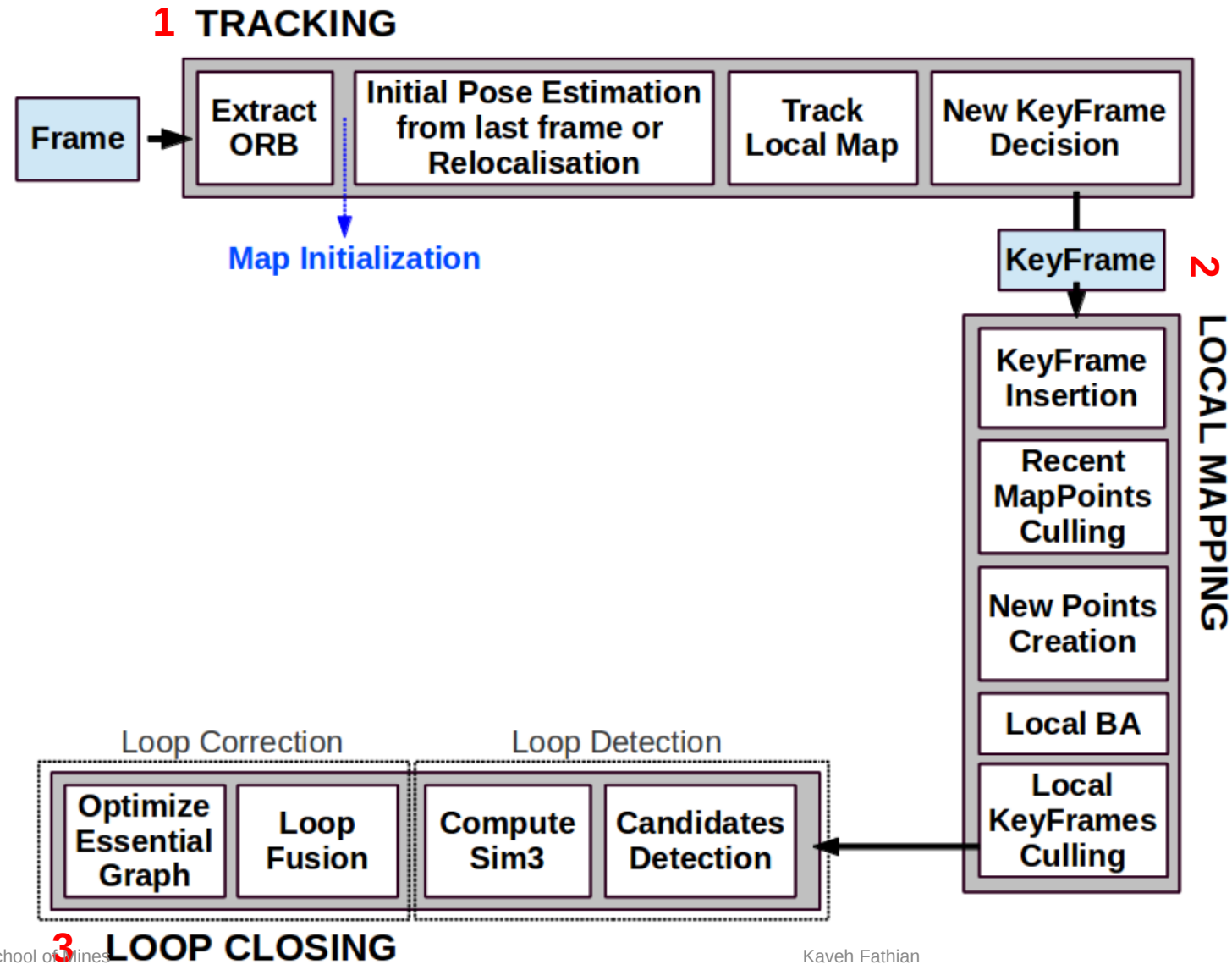
Carlos Campos*, Richard Elvira*, Juan J. Gómez Rodríguez, José M.M. Montiel and Juan D. Tardós

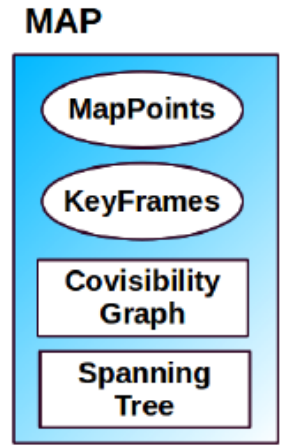
***Abstract*—This paper presents ORB-SLAM3, the first system able to perform visual, visual-inertial and multi-map SLAM with monocular, stereo and RGB-D cameras, using pin-hole and fisheye lens models.**

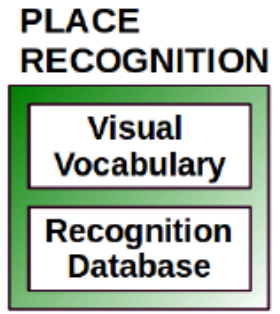
The first main novelty is a feature-based tightly-integrated visual-inertial SLAM system that fully relies on Maximum-a-Posteriori (MAP) estimation, even during the IMU initialization phase. The result is a system that operates robustly in real time,

on-board a mobile agent to build a map of the environment and compute in real-time the pose of the agent in that map. In contrast, VO systems put their focus on computing the agent's ego-motion, not on building a map. The big advantage of a SLAM map is that it allows matching and using in BA previous observations performing three types of data association (extending the terminology used in [1]):

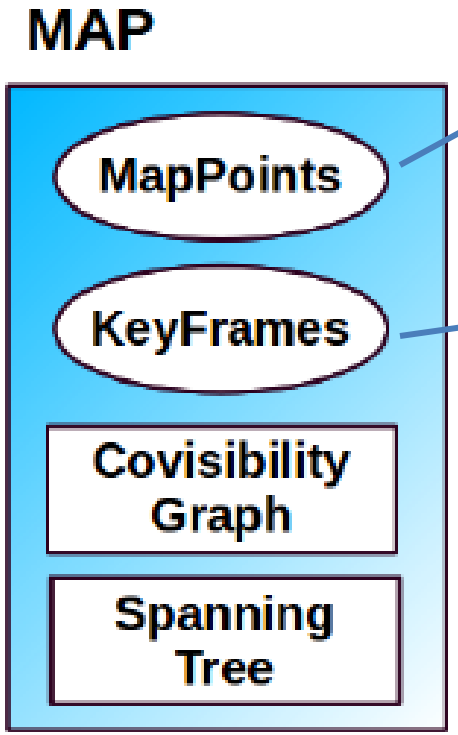
ORB-SLAM: three parallel threads





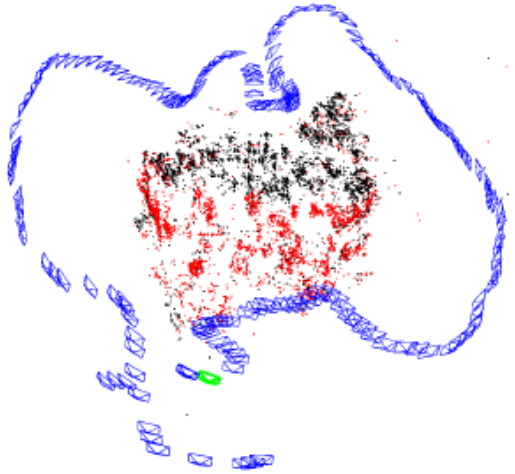


ORB-SLAM data

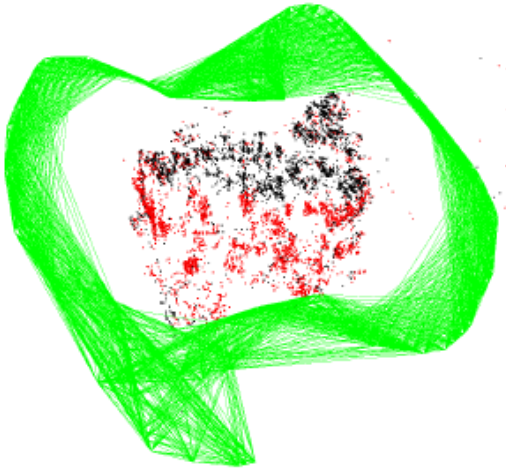


- 3D position ($\mathbf{X}_{w,i}$)
- average viewing direction $\mathbf{n}_i$
- centroid ORB descriptor $\mathbf{D}_i$
- Observable distance range

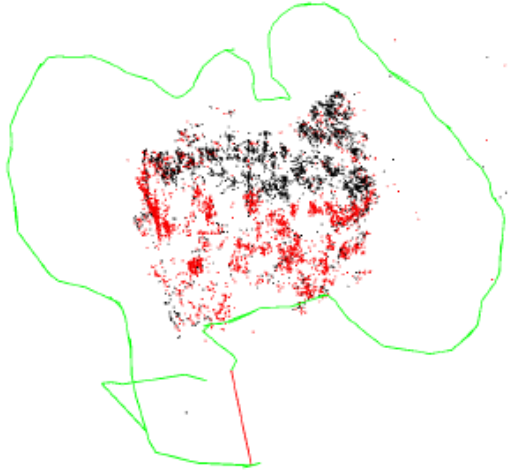
- Camera pose $\mathbf{T}_{iw}$
- All ORB features



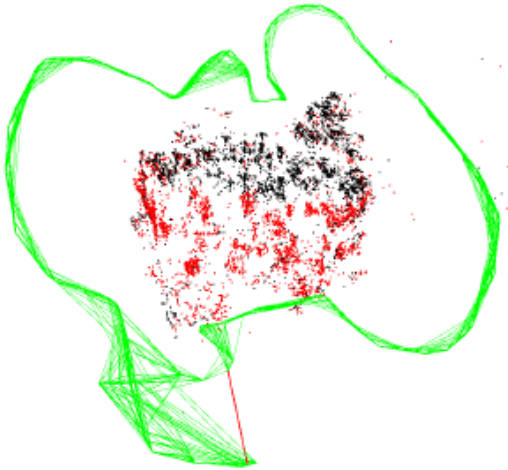
(a) KeyFrames (blue), Current Camera (green), MapPoints (black, red), Current Local MapPoints (red)



(b) Covisibility Graph

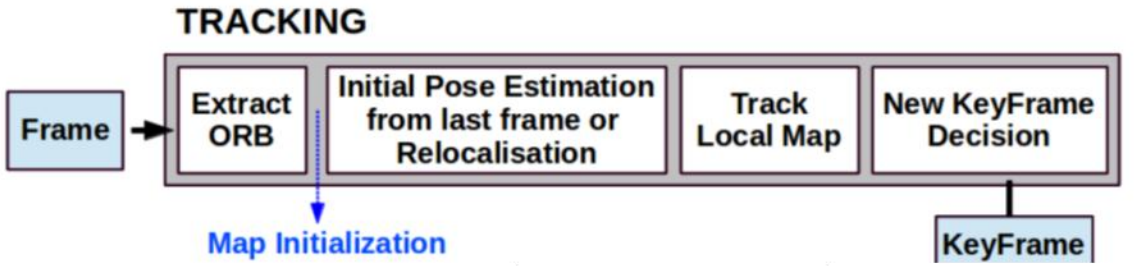


(c) Spanning Tree (green) and Loop Closure (red)



(d) Essential Graph

Tracking Overview



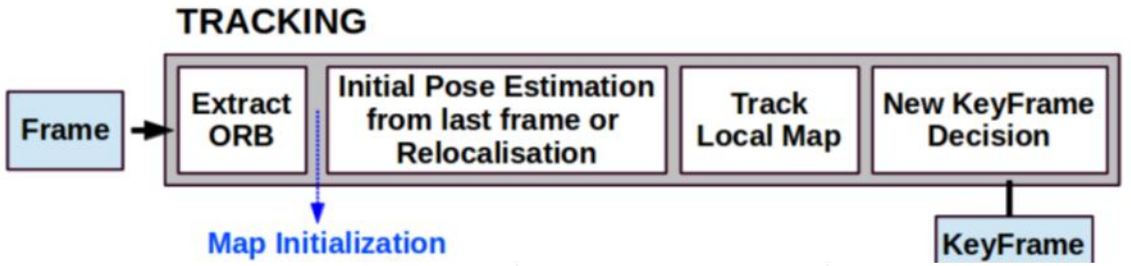
Goal: Achieve fast and robust matching of each frame based on points observed in last frame

For each new frame

1. Extract features
2. Localize to previous frame (if possible) or keyframes
3. Find more matching points
4. Potentially add this frame to set of keyframes

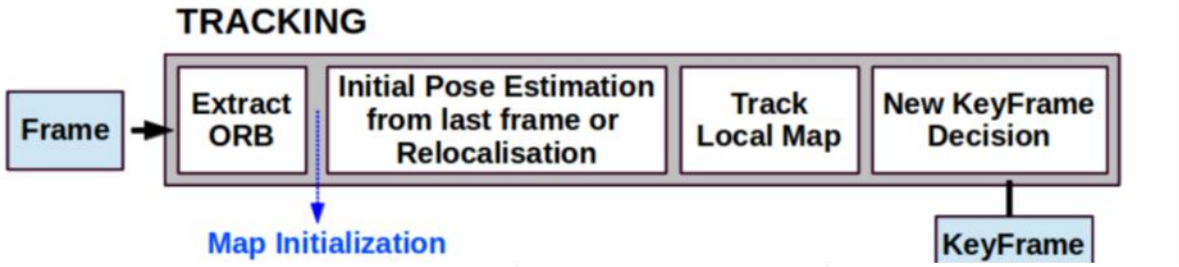
This is “visual odometry” and could also be augmented with IMU (visual-inertial odometry)

Tracking: Map Initialization



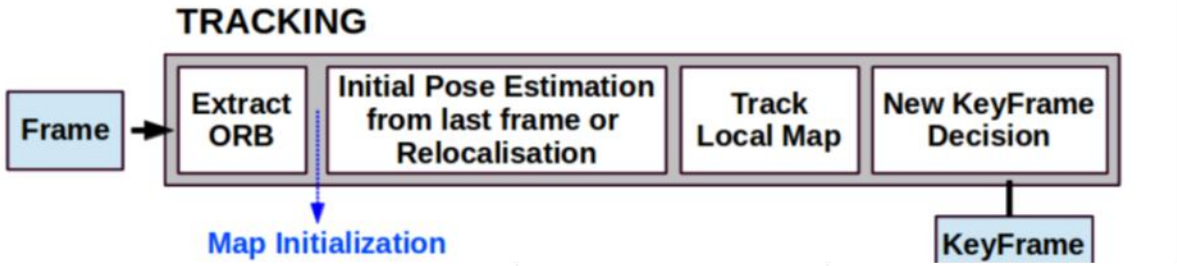
- Extract ORB features
- Find correspondences in two frames (current and reference)
- Check whether points are mostly explained by homography H or fundamental matrix F
  - If F: solve for F and compute E using intrinsic matrix K
  - If H: check for valid planar solution
  - Get a new reference frame if well-conditioned F or H cannot be found
- Perform bundle adjustment (BA) on two frames and mapped points

Tracking: ORB + Initial Pose



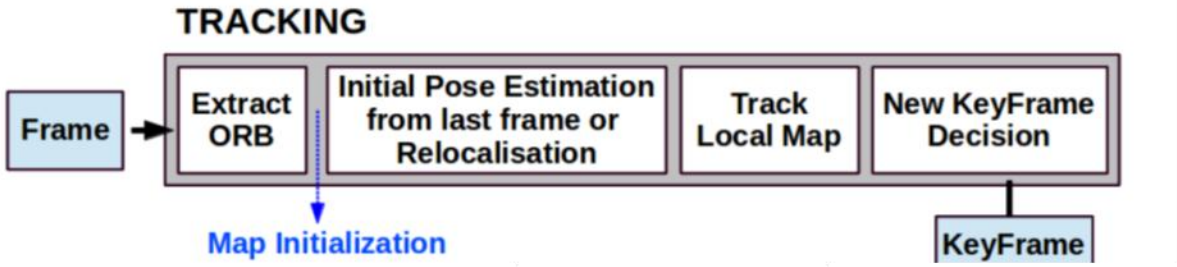
- FAST corners + ORB features
  - 1000-2000 corners
  - Distributed across 8 scales and a grid of cells
- Try to initialize pose using previous frame
  - Predict positions of previously observed map points into current frame based on constant velocity motion estimate
  - Perform wider search if not enough points found
- Else, perform relocalization
  - Find candidate matches among existing keyframes with bag of words search
  - Use RANSAC and PnP to optimize pose and then perform guided search of for more map points

Tracking: Track Local Map



- For keyframes that share map points and their neighbors, get all map points
- Find more map point correspondences
  1. Project each to current frame, check if in bounds
  2. Check that current view angle is within 60 deg of avg for point
  3. Check that current distance is within point range
  4. Find best matching ORB feature at similar position/scale, and associate
- Optimize camera pose wrt associated points

Tracking: Keyframe Decision



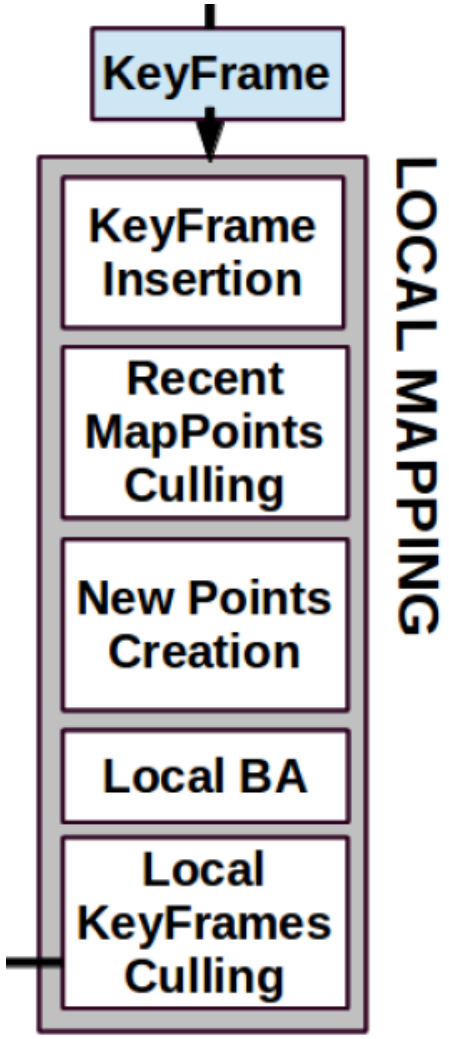
Add current frame as new keyframe K_i if all conditions are met:

1. More than 20 frames since last global relocalization
2. Ready for new: Local mapping idle, or more than 20 frames since last keyframe insertion
3. Good tracking: Current frame tracks at least 50 points
4. Not redundant: Current frame tracks less than 90% of points from most similar keyframe

Local Mapping Overview

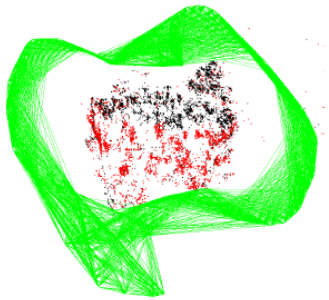
Goal: Jointly refine points and poses of recently viewed parts of the scene to reduce drift

1. Update graphs and maps with new keyframe K_i
2. Optimize nearby keyframes and points
3. Remove bad points and redundant keyframes

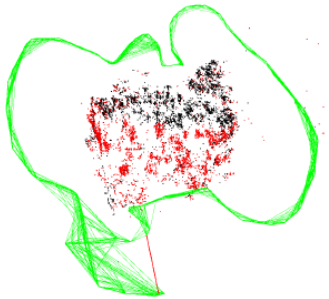


Local Mapping: Keyframe Insertion

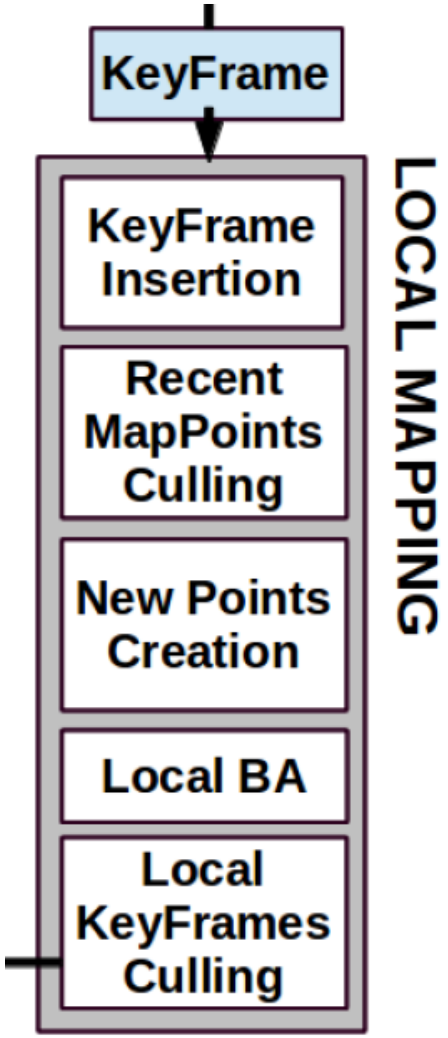
- Update covisibility graph (which other keyframes see the same points as K_i)
- Update spanning tree, linking with keyframe that has most points in common with K_i
- Compute bag of words for K_i



(b) Covisibility Graph



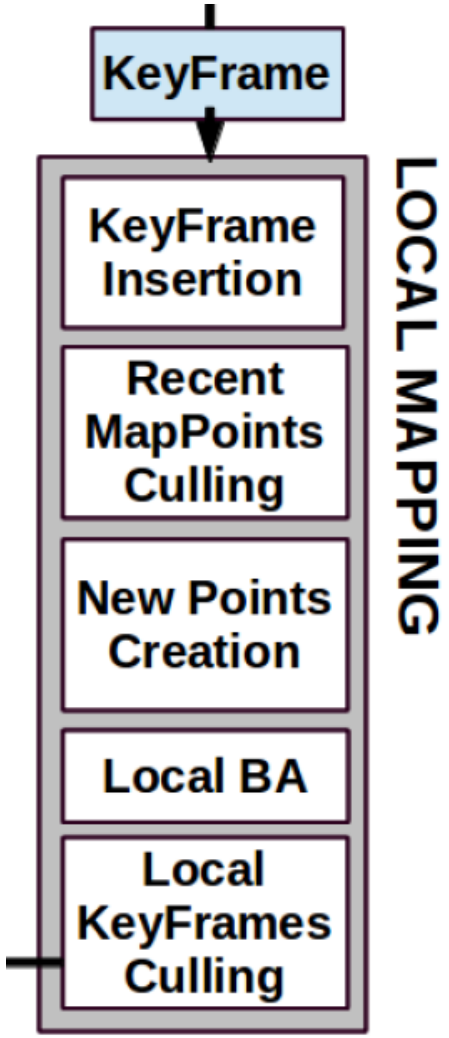
(d) Essential Graph



Local Mapping: Recent Point Culling

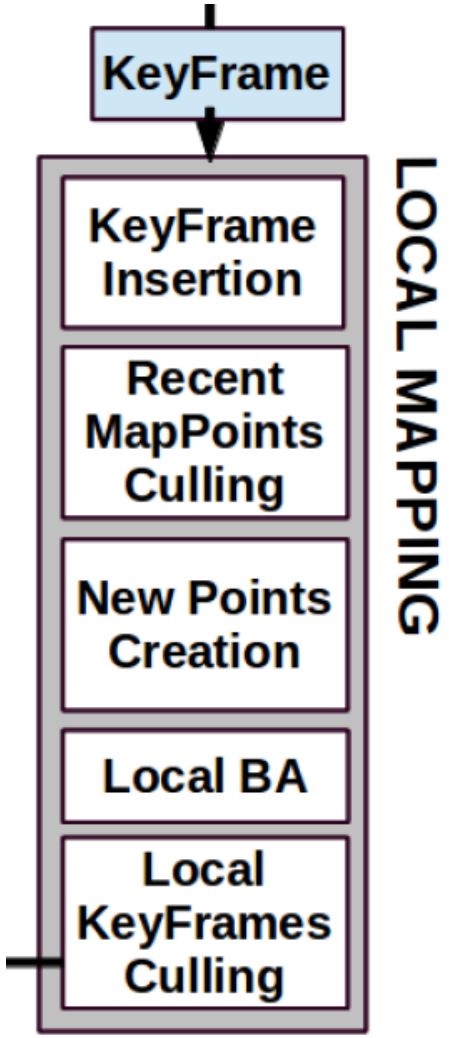
- Points are initially kept if
  - Point is tracked in at least 25% of expected frames
  - Observed in at least three keyframes (after initialization)

- Remove points if not enough keyframes (after keyframes removed), or high reprojection error after Local BA



Local Mapping: Point Creation

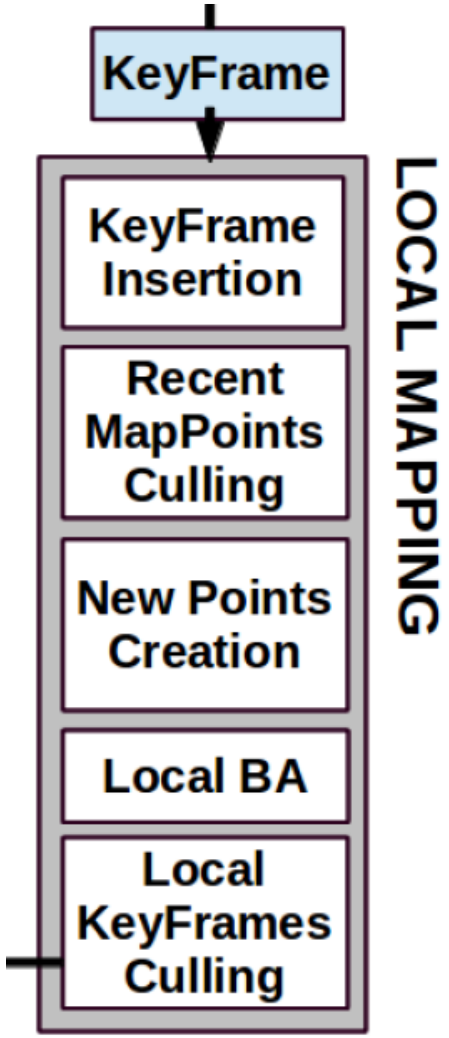
- Attempt to match any unmatched features in K_i to co-visible keyframes
  - Use vocab tree and check epipolar constraint
- Triangulate good matches, check reprojection error, etc.
- Find correspondences in additional connected keyframes, similar to “track local map”



Local Mapping: Local BA

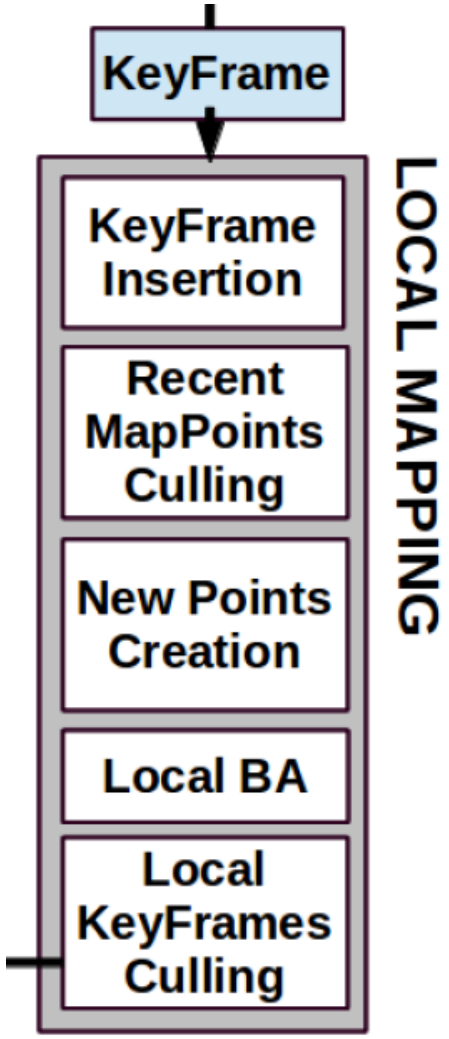
- Optimize new keyframe K_i, those connected in covisibility graph, and points seen by those keyframes

- Discard points that have high reprojection errors at the middle and end of process



Local Mapping: KeyFrames culling

- Discard keyframes if at least 90% of its observed points are also observed by other keyframes at similar or closer distance

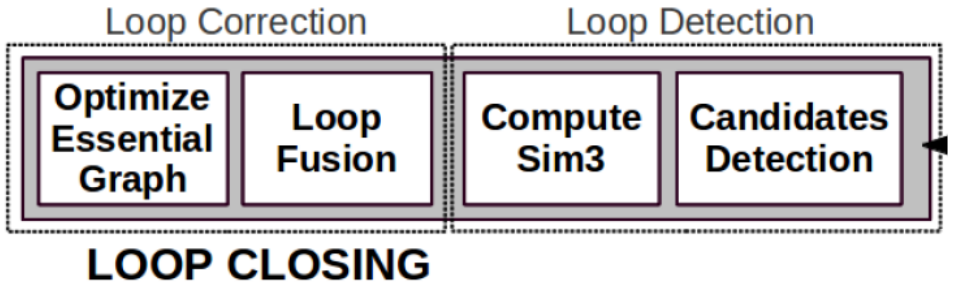


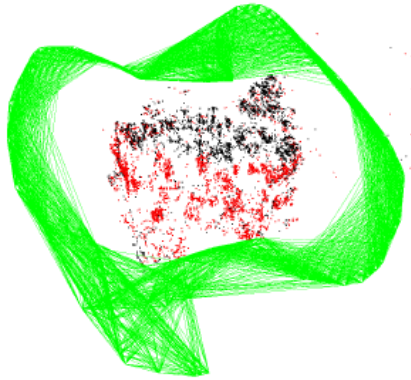
Loop Closing Overview

Goal: Find and optimize over long-range connections to eliminate drift

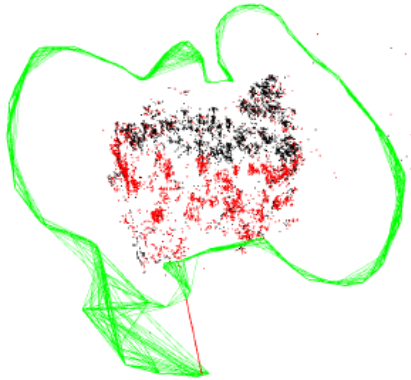
For new keyframe K_i:

1. Vocab tree matching to all non-connected keyframes and get candidates
2. Find matches with candidates and use RANSAC to solve similarity transform
3. Add edges to co-visibility and essential graph and perform graph optimization on essential graph





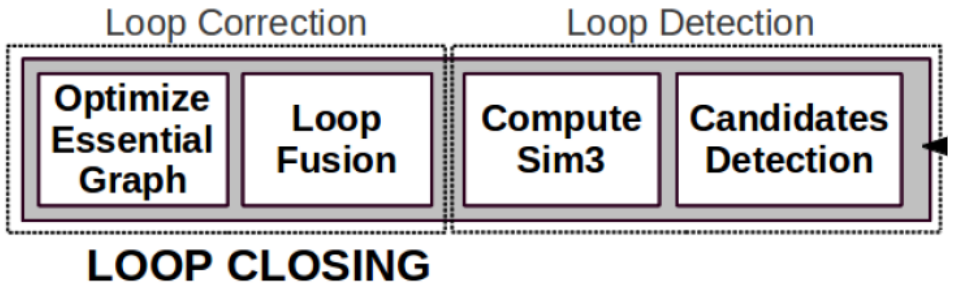
(b) Covisibility Graph

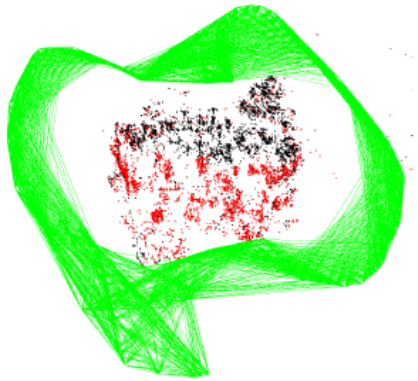


(d) Essential Graph

Loop: Candidate Detection

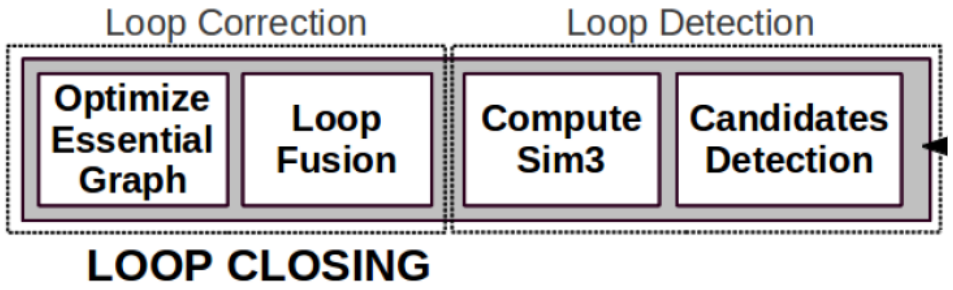
- Find BoW similarity of K_i to neighbors covisibility graph and set threshold S as minimum similarity
- Candidates are keyframes that
  - Are not connected to K_i
  - Have score greater than S
  - Two other connected keyframes in co-visibility graph also have score with K^i greater than S





(b) Covisibility Graph

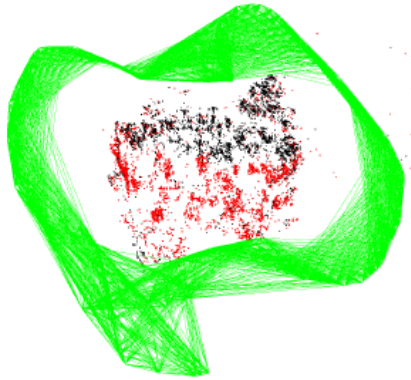
Loop: Similarity Transform



- Find feature matches between K_i and loop keyframe K_l
  - Provides 3D to 3D correspondences since features are linked to 3D points
- Solve for similarity transform with RANSAC
- Optimize camera pose with points fixed and perform guided search for more matches
- Accept loop closure with K_l if there are enough inliers

Loop Closing: Fusion

- Update pose of K_i and its neighbors with similarity transform
- Fuse points with K_i that are also seen by K_l and its neighbors
- Search for additional points to fuse by checking projections and feature similarities
- Update covisibility graph to reflect fused points



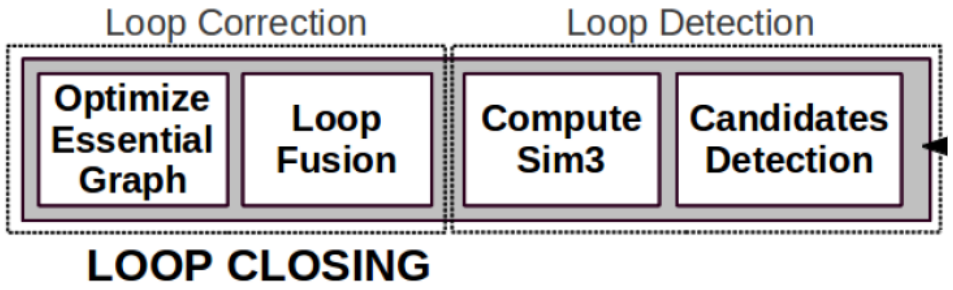
(b) Covisibility Graph

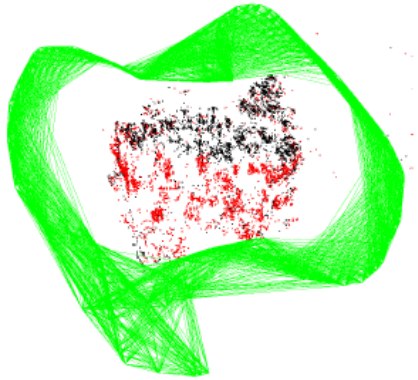
Loop Closing: Optimization

- Essential graph is spanning tree of covisibility graph (keeping strongest edges in tree structure) plus loop closure edges

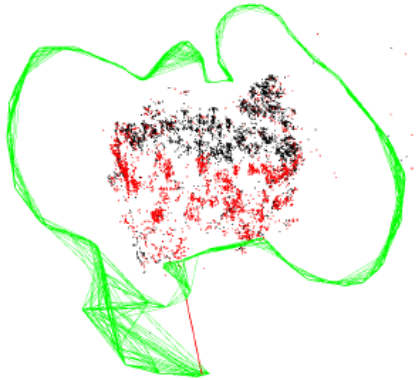
- Pose graph optimization: Solve for pose of each camera that satisfies pairwise similarity transforms (edges in essential graph) as well as possible

- Update map points to be consistent with new poses





(b) Covisibility Graph



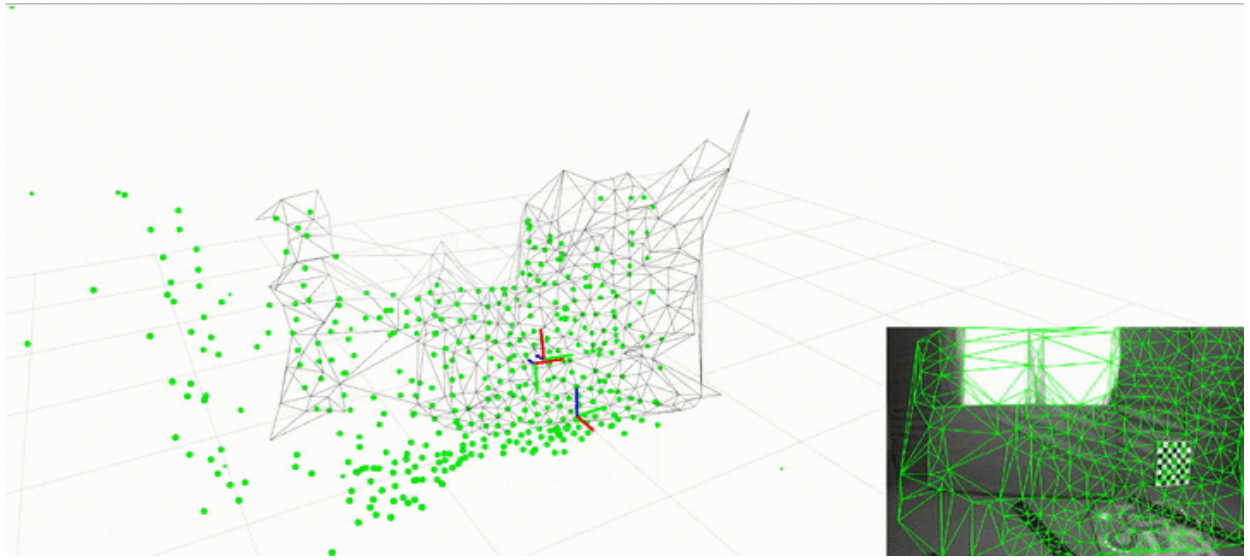
(d) Essential Graph

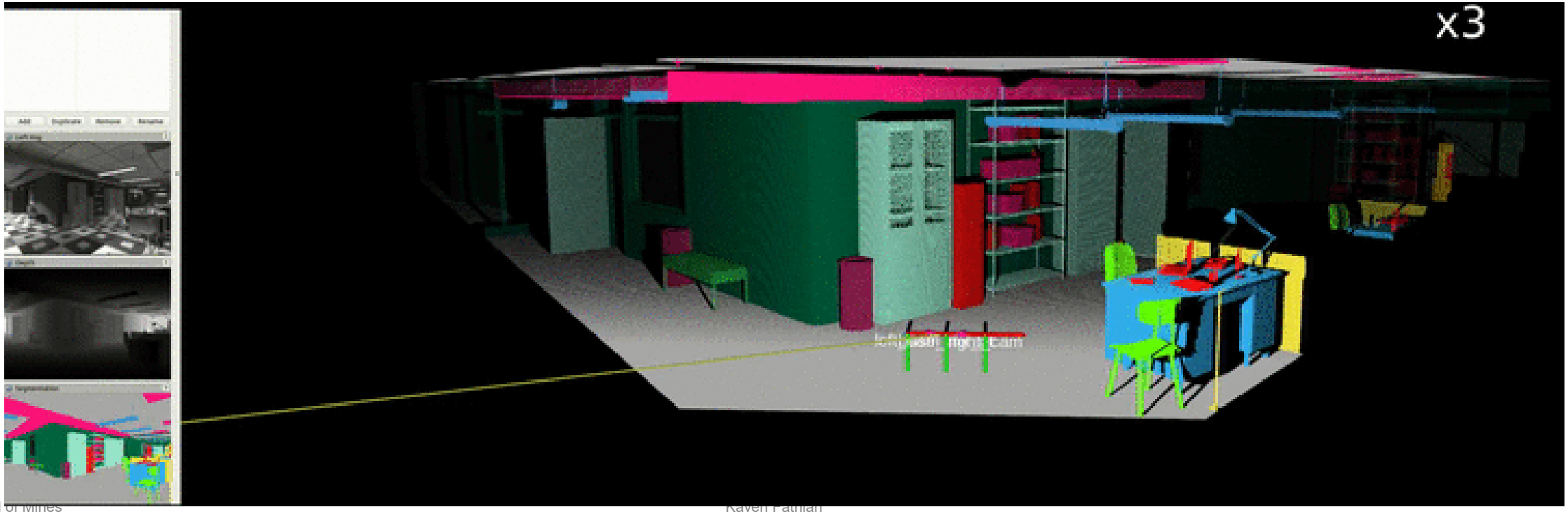
Example

https://www.youtube.com/watch?v=8DISRmsO2YQ

Metric-Semantic SLAM

- Many SOTA SLAM systems produce a semantic understanding of the world
- E.g., Kimera for real-time metric-semantic SLAM
  - Code: https://github.com/MIT-SPARK/Kimera
  - Paper: https://arxiv.org/pdf/1910.02490





Deep-Learning-Based SLAM

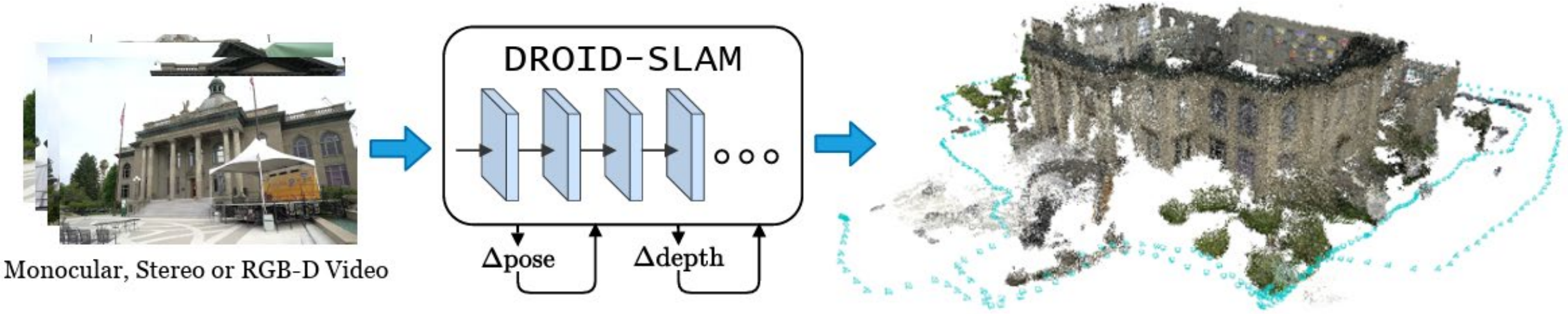
- SLAM can be solve end-to-end using DL
- E.g., DROID-SLAM
  - Code: https://github.com/princeton-vl/DROID-SLAM
  - Paper: https://arxiv.org/pdf/2108.10869

DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras

Zachary Teed **Jia Deng**
Princeton University
{zteed,jiadeng}@princeton.edu

Abstract

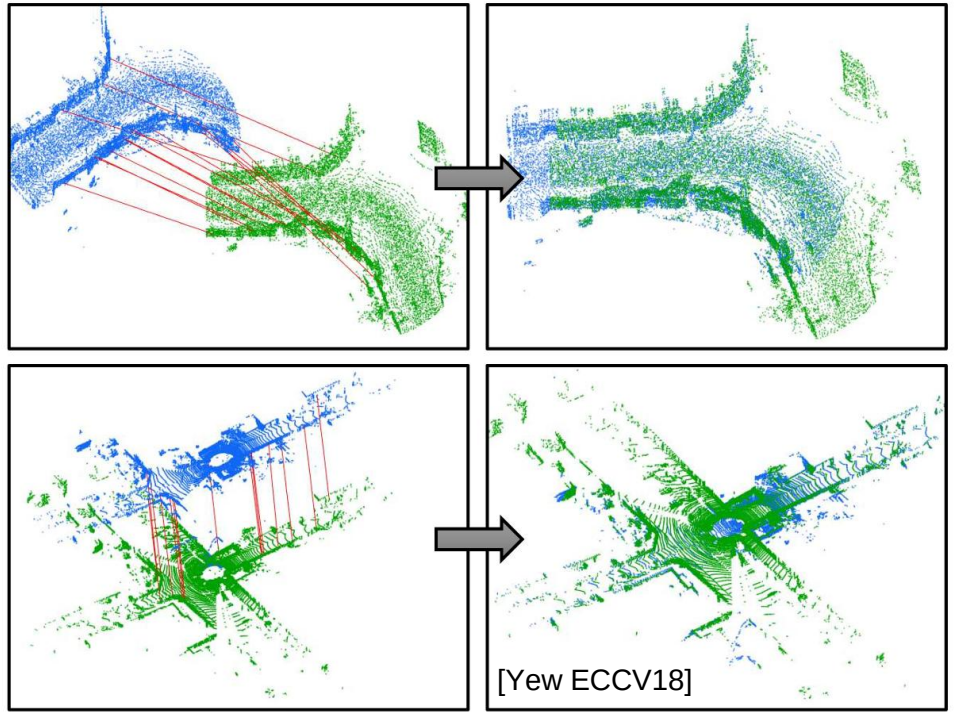
We introduce DROID-SLAM, a new deep learning based SLAM system. DROID-SLAM consists of recurrent iterative updates of camera pose and pixelwise depth through a Dense Bundle Adjustment layer. DROID-SLAM is accurate, achieving large improvements over prior work, and robust, suffering from substantially fewer catastrophic failures. Despite training on monocular video, it can leverage stereo or RGB-D video to achieve improved performance at test time. The URL to our open source code is https://github.com/princeton-vl/DROID-SLAM.

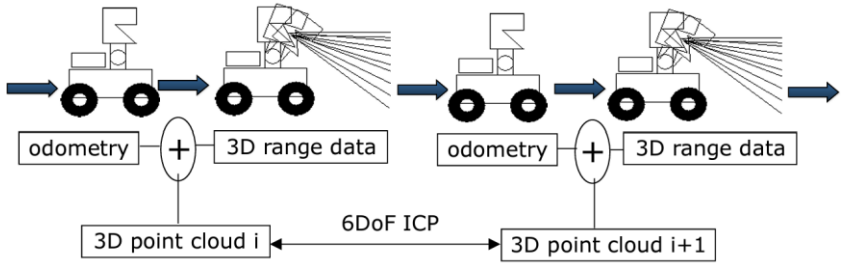


LiDAR SLAM

LiDAR Odometry Review

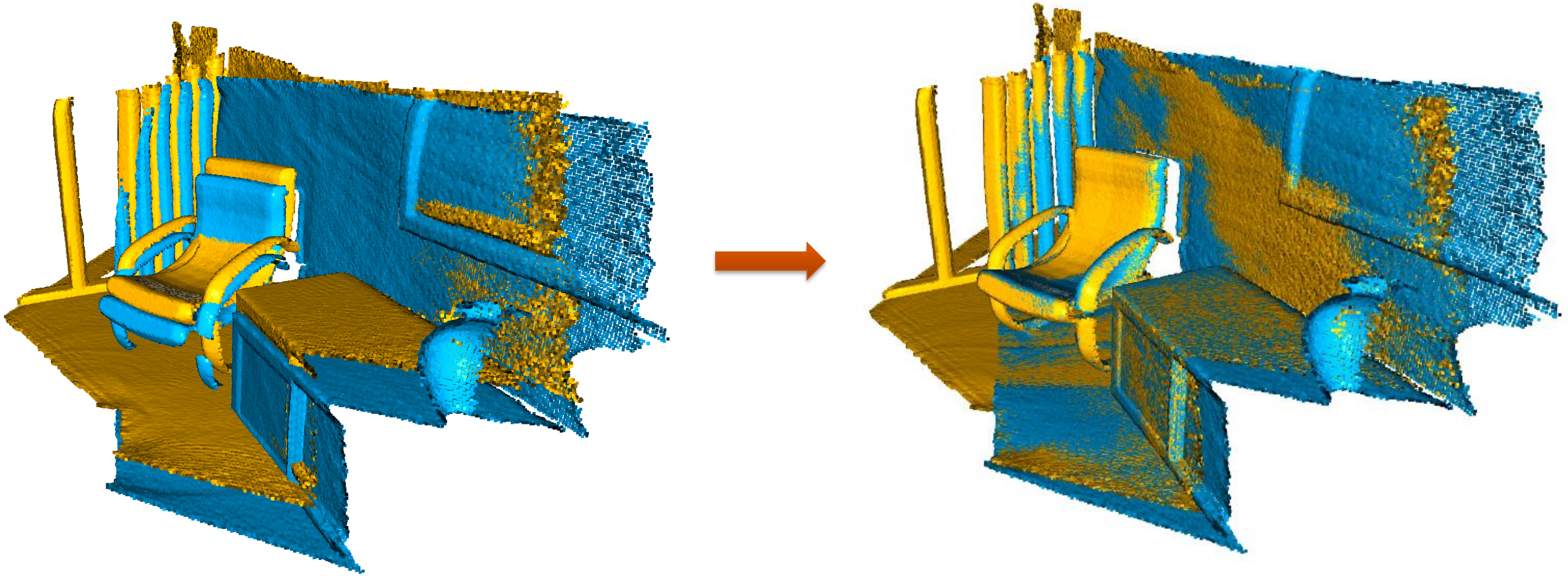
- Lidar odometry requires solving the ***point cloud registration*** problem
- Registration problem: Find the geometric relationship between sensor data and some prior (such as previous sensor data or map).
- Approaches can vary by:
  - Resolution
    - Coarse algorithms look for crude alignments, such as for place recognition.
    - Fine algorithms try to achieve better alignments, but generally need more accurate initial guesses.
  - Data
    - Direct algorithms use the raw observations.
    - Indirect (feature-based) algorithms abstract dense raw data into sparse features.
  - Local vs. global solvers.
  - Real-time vs. offline solvers.





Point Cloud Registration Methods

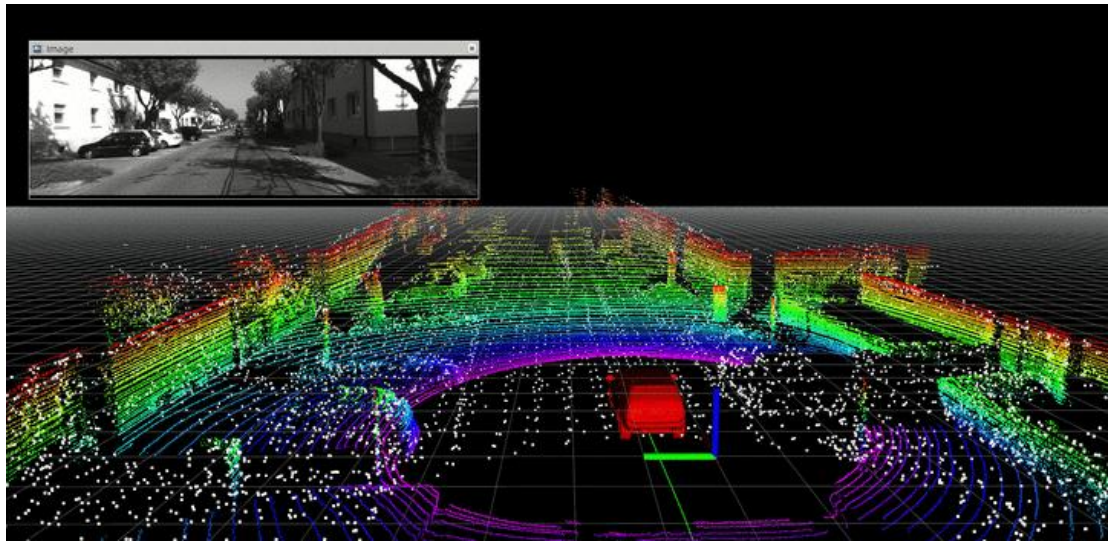
- ICP
- RANSAC
- Etc.

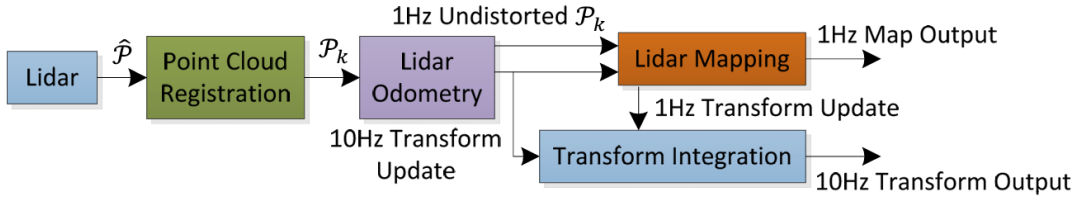


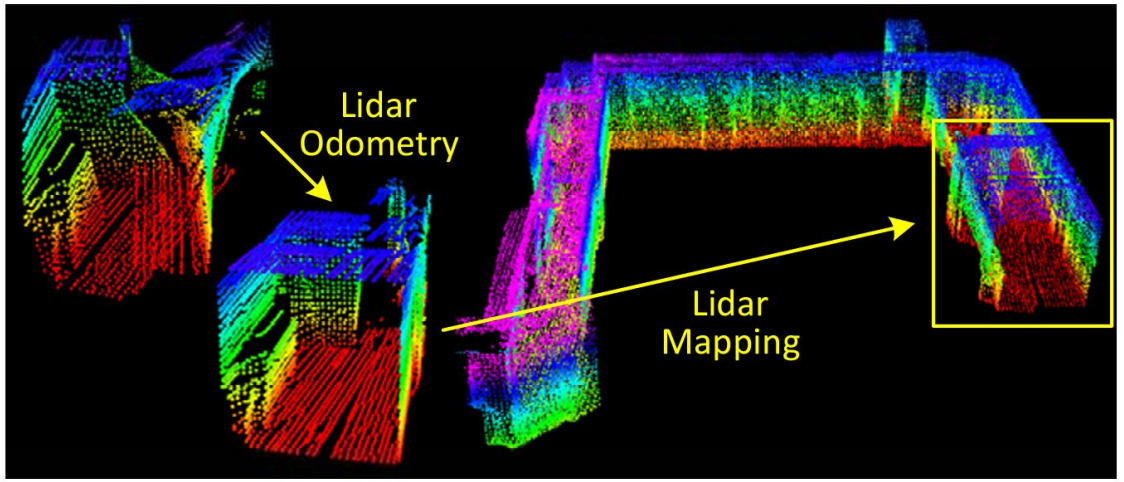
LiDAR Motion Compensation

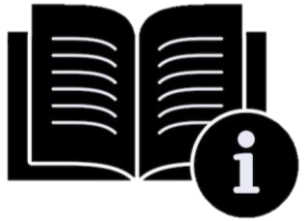
LOAM

- Paper: "LOAM: Lidar Odometry and Mapping in Real-time," RSS, 2014.
- **One of the first** 3D LiDAR systems
- Low-drift and low-computational complexity (real-time) without IMU
- **Compensate for motion** during point cloud scans
- LOAM consists of LiDAR odometry & mapping algorithms
- Code: https://github.com/HKUST-Aerial-Robotics/A-LOAM









LOAM

$$\boldsymbol{X}^L_{(k+1,i)} = \mathbf{R}\tilde{\boldsymbol{X}}^L_{(k+1,i)} + \boldsymbol{T}^L_{(k+1,i)}(1:3), \tag{5}$$

where $\boldsymbol{X}^L_{(k+1,i)}$ is the coordinates of a point i in $\mathcal{E}_{k+1}$ or $\mathcal{H}_{k+1}$, $\tilde{\boldsymbol{X}}^L_{(k+1,i)}$ is the corresponding point in $\tilde{\mathcal{E}}_{k+1}$ or $\tilde{\mathcal{H}}_{k+1}$, $\boldsymbol{T}^L_{(k+1,i)}(a:b)$ is the a-th to b-th entries of $\boldsymbol{T}^L_{(k+1,i)}$, and $\mathbf{R}$ is a rotation matrix defined by the Rodrigues formula [25],

$$\mathbf{R} = e^{\hat{\omega}\theta} = \mathbf{I} + \hat{\omega}\sin\theta + \hat{\omega}^2(1-\cos\theta). \tag{6}$$

In the above equation, θ is the magnitude of the rotation,

$$\theta = ||\boldsymbol{T}^L_{(k+1,i)}(4:6)||, \tag{7}$$

ω is a unit vector representing the rotation direction,

$$\omega = \boldsymbol{T}^L_{(k+1,i)}(4:6)/||\boldsymbol{T}^L_{(k+1,i)}(4:6)||, \tag{8}$$

and $\hat{\omega}$ is the skew symmetric matrix of ω [25].

Recall that (2) and (3) compute the distances between points in $\tilde{\mathcal{E}}_{k+1}$ and $\tilde{\mathcal{H}}_{k+1}$ and their correspondences. Combining (2) and (4)-(8), we can derive a geometric relationship between an edge point in $\mathcal{E}_{k+1}$ and the corresponding edge line,

$$f_{\mathcal{E}}(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1}) = d_{\mathcal{E}},\ i \in \mathcal{E}_{k+1}. \tag{9}$$

Similarly, combining (3) and (4)-(8), we can establish another geometric relationship between a planar point in $\mathcal{H}_{k+1}$ and the corresponding planar patch,

$$f_{\mathcal{H}}(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1}) = d_{\mathcal{H}},\ i \in \mathcal{H}_{k+1}. \tag{10}$$

Finally, we solve the lidar motion with the Levenberg-Marquardt method [26]. Stacking (9) and (10) for each feature point in $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$, we obtain a nonlinear function,

$$\boldsymbol{f}(\boldsymbol{T}^L_{k+1}) = \boldsymbol{d}, \tag{11}$$

where each row of $\boldsymbol{f}$ corresponds to a feature point, and $\boldsymbol{d}$ contains the corresponding distances. We compute the Jacobian matrix of $\boldsymbol{f}$ with respect to $\boldsymbol{T}^L_{k+1}$, denoted as $\mathbf{J}$, where $\mathbf{J} = \partial f/\partial \boldsymbol{T}^L_{k+1}$. Then, (11) can be solved through nonlinear iterations by minimizing $\boldsymbol{d}$ toward zero,

$$\boldsymbol{T}^L_{k+1} \leftarrow \boldsymbol{T}^L_{k+1} - (\mathbf{J}^T\mathbf{J} + \lambda \mathrm{diag}(\mathbf{J}^\mathrm{T}\mathbf{J}))^{-1}\mathbf{J}^\mathrm{T}\boldsymbol{d}. \tag{12}$$

λ is a factor determined by the Levenberg-Marquardt method.

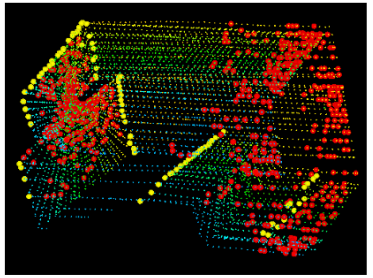


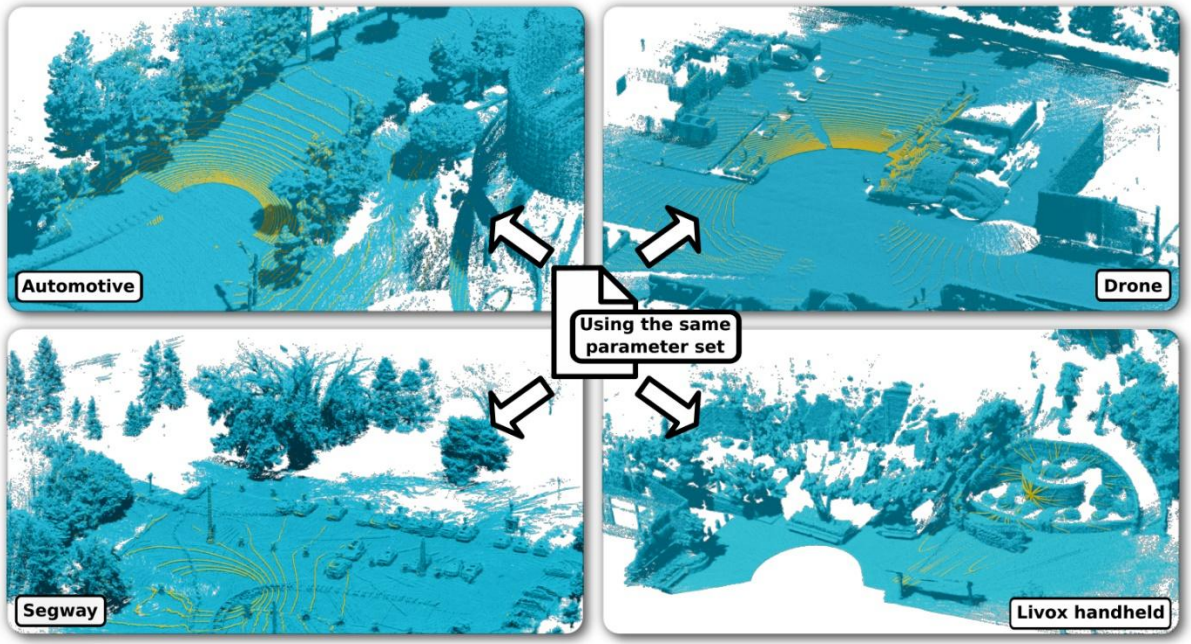
Fig. 5. An example of extracted edge points (yellow) and planar points (red) from lidar cloud taken in a corridor. Meanwhile, the lidar moves toward the wall on the left side of the figure at a speed of 0.5m/s, this results in motion distortion on the wall.

Algorithm 1: Lidar Odometry

input : $\bar{\mathcal{P}}_k$, $\mathcal{P}_{k+1}$, $\boldsymbol{T}^L_{k+1}$ from the last recursion
output : $\bar{\mathcal{P}}_{k+1}$, newly computed $\boldsymbol{T}^L_{k+1}$
begin
  if at the beginning of a sweep **then**
    $\boldsymbol{T}^L_{k+1} \leftarrow \mathbf{0}$;
  end
  Detect edge points and planar points in $\mathcal{P}_{k+1}$, put the points in $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$, respectively;
  for a number of iterations **do**
    for each edge point in $\mathcal{E}_{k+1}$ **do**
      Find an edge line as the correspondence, then compute point to line distance based on (9) and stack the equation to (11);
    end
    for each planar point in $\mathcal{H}_{k+1}$ **do**
      Find a planar patch as the correspondence, then compute point to plane distance based on (10) and stack the equation to (11);
    end
    Compute a bisquare weight for each row of (11);
    Update $\boldsymbol{T}^L_{k+1}$ for a nonlinear iteration based on (12);
    if the nonlinear optimization converges **then**
      Break;
    end
  end
  if at the end of a sweep **then**
    Reproject each point in $\mathcal{P}_{k+1}$ to t_{k+2} and form $\bar{\mathcal{P}}_{k+1}$;
    Return $\boldsymbol{T}^L_{k+1}$ and $\bar{\mathcal{P}}_{k+1}$;
  end
  else
    Return $\boldsymbol{T}^L_{k+1}$;
  end
end

KISS ICP

- Paper: "KISS-ICP: In Defense of Point-to-Point ICP," RAL, 2022
- Based on **point-to-point ICP**
- No need for IMU
- Reduces parameter tuning
- Downsamples points using a voxel grid
- Compensates for motion using the **constant velocity model**
- This assumes that a robot moves with same translational and rotational velocity as in previous timestep
- Code: https://github.com/PRBonn/kiss-icp



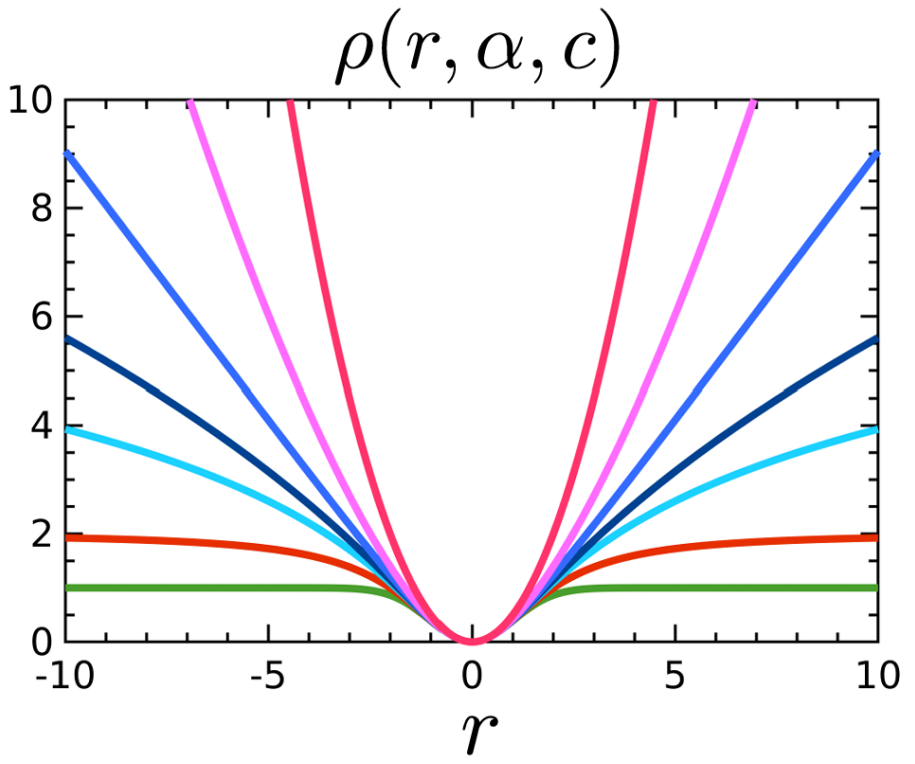
KISS ICP

For each iteration j of ICP, we obtain a set of correspondences between the point cloud $\mathcal{S}$ and the local map $\mathcal{Q} = \{\boldsymbol{q}_i \mid \boldsymbol{q}_i \in \mathbb{R}^3\}$ through nearest neighbor search over the voxel grid (Sec. III-C) considering only correspondences with a point-to-point distance below τ_t. To compute the current pose correction $\Delta\mathsf{T}_{\text{est},j}$, we perform a robust optimization minimizing the sum of point-to-point residuals

$$\Delta\mathsf{T}_{\text{est},j} = \underset{\mathsf{T}}{\operatorname{argmin}} \sum_{(s,q)\in\mathcal{C}(\tau_t)} \rho(\|\mathsf{T}\boldsymbol{s} - \boldsymbol{q}\|_2), \tag{12}$$

where $\mathcal{C}(\tau_t)$ is the set of nearest neighbor correspondences with a distance smaller than τ_t and ρ is the Geman-McClure robust kernel, i.e., an M-estimator with a strong outlier rejection property, given by

$$\rho(e) = \frac{e^2/2}{\underbrace{\sigma_t/3}_{\kappa_t} + e^2}, \tag{13}$$

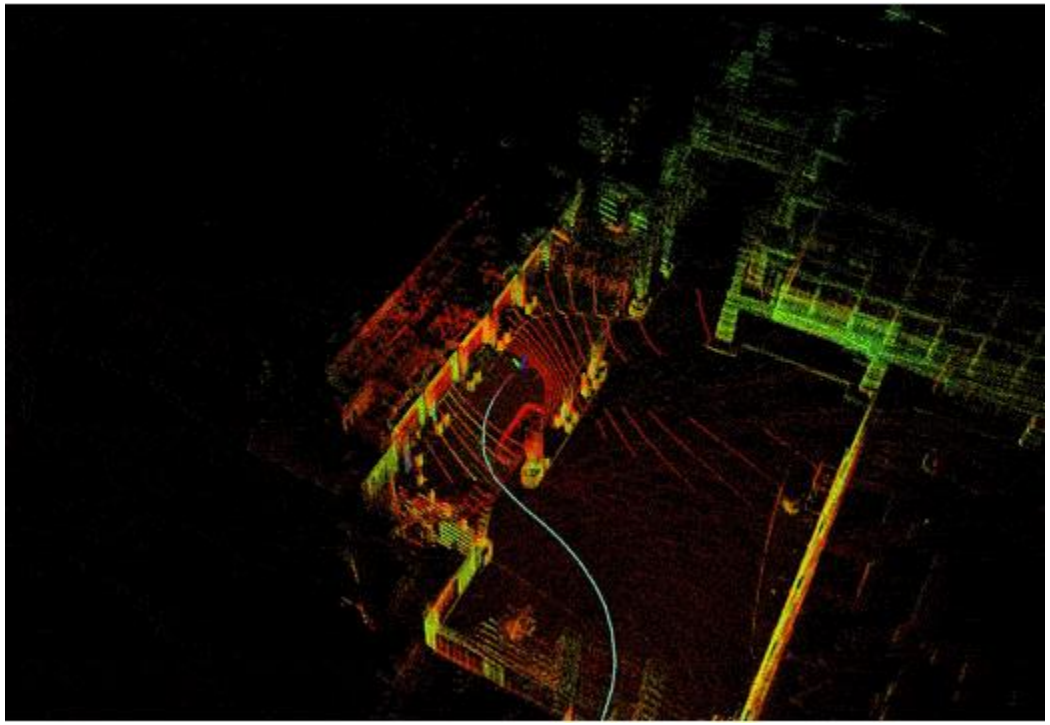


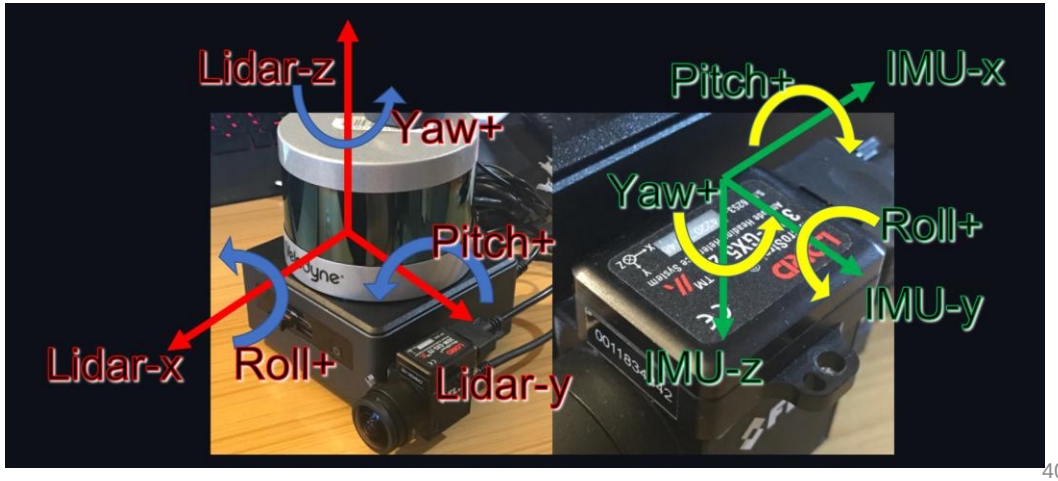
https://arxiv.org/pdf/2004.14938.pdf

LiDAR-IMU SLAM

LIO-SAM

- Paper: "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," IROS 2020
- Estimated motion from **IMU pre-integration de-skews** point clouds and produces an initial guess for lidar odometry optimization
- Formulates LiDAR-inertial odometry as a **factor graph**
- Code: https://github.com/TixiaoShan/LIO-SAM





LIO-SAM

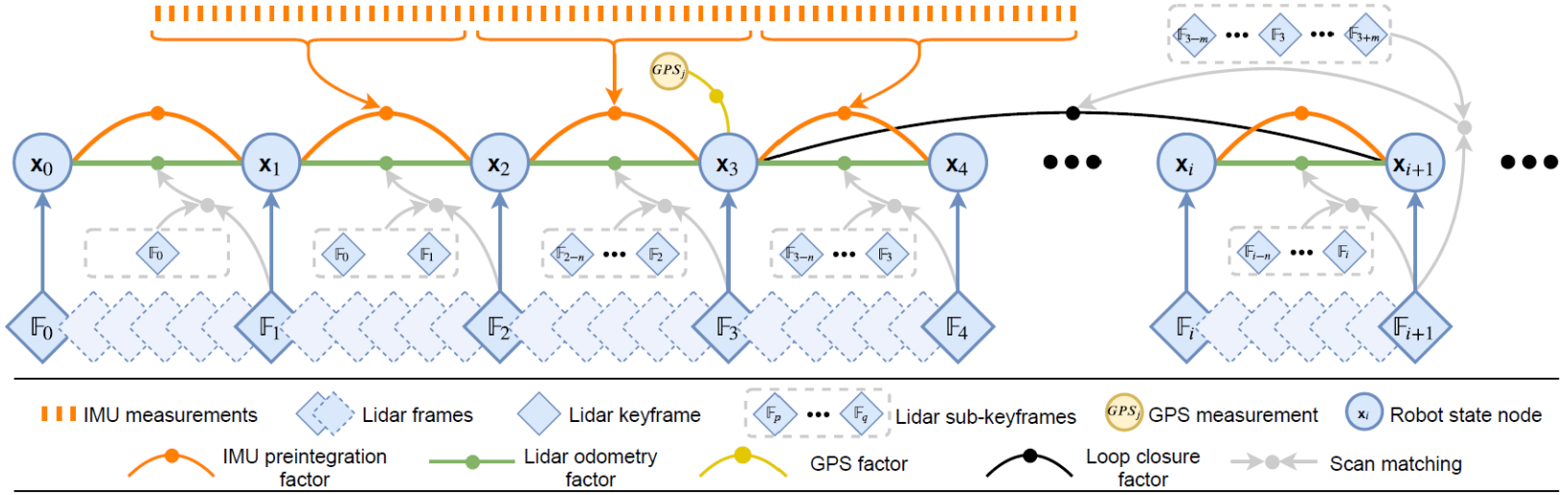
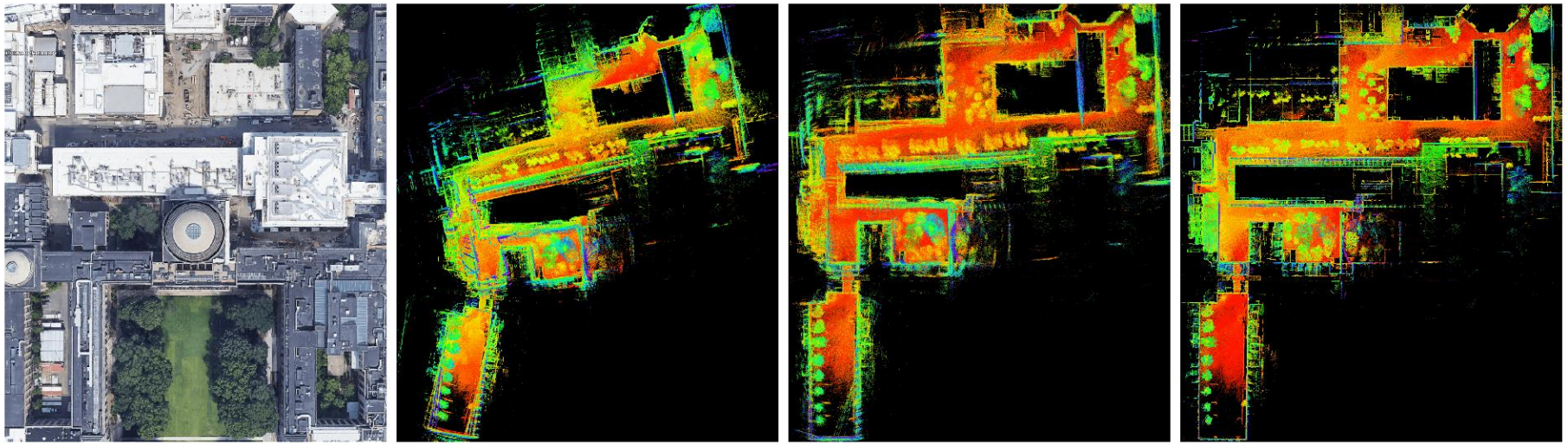


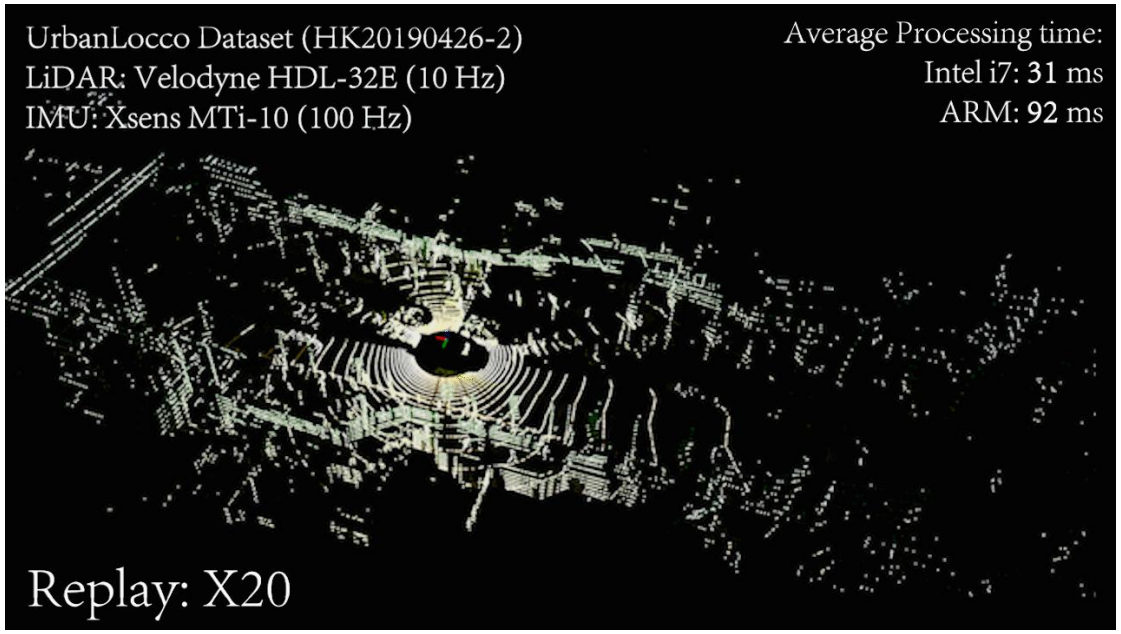
Fig. 1: The system structure of LIO-SAM. The system receives input from a 3D lidar, an IMU and optionally a GPS. Four types of factors are introduced to construct the factor graph.: (a) IMU preintegration factor, (b) lidar odometry factor, (c) GPS factor, and (d) loop closure factor. The generation of these factors is discussed in Section III.



(a) Google Earth (b) LOAM (c) LIOM (d) LIO-SAM

Fast-LIO2

- Paper: "FAST-LIO2: Fast Direct LiDAR-inertial Odometry," TRO, 2022
- Based on Kalman filter estimation
- Incremental k-d tree data structure for computational efficiency
- Code: https://github.com/hku-mars/FAST_LIO



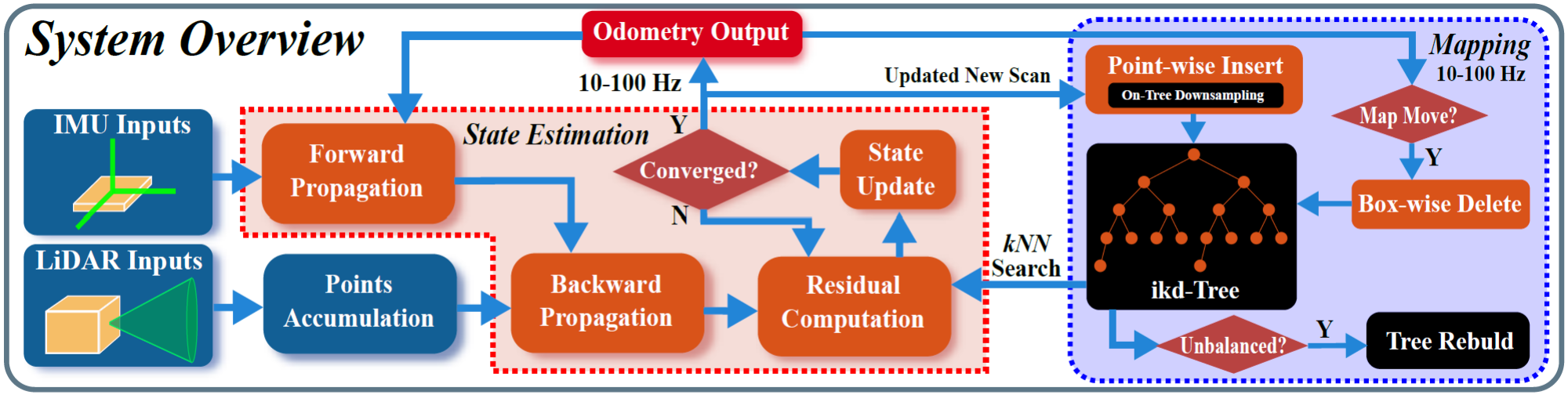


Fig. 1. System overview of FAST-LIO2.