

Robotic Mapping & Localization

Kaveh Fathian

Assistant Professor

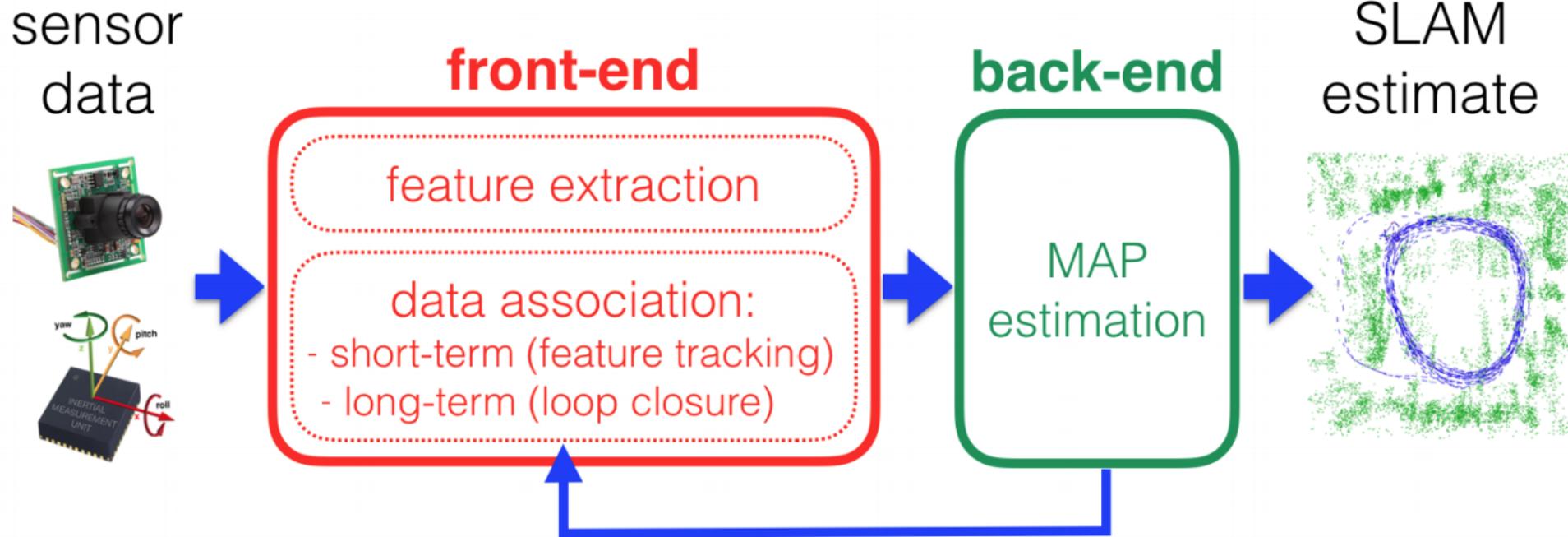
Computer Science Department

Colorado School of Mines

Lec14: SLAM Backend

*Courtesy of Maani Ghaffari, Ryan Eustice, Frank Dellaert, Michael Kaess, Cyrill Stachniss, Pranav Ganti, Luca Carlone

SLAM Components



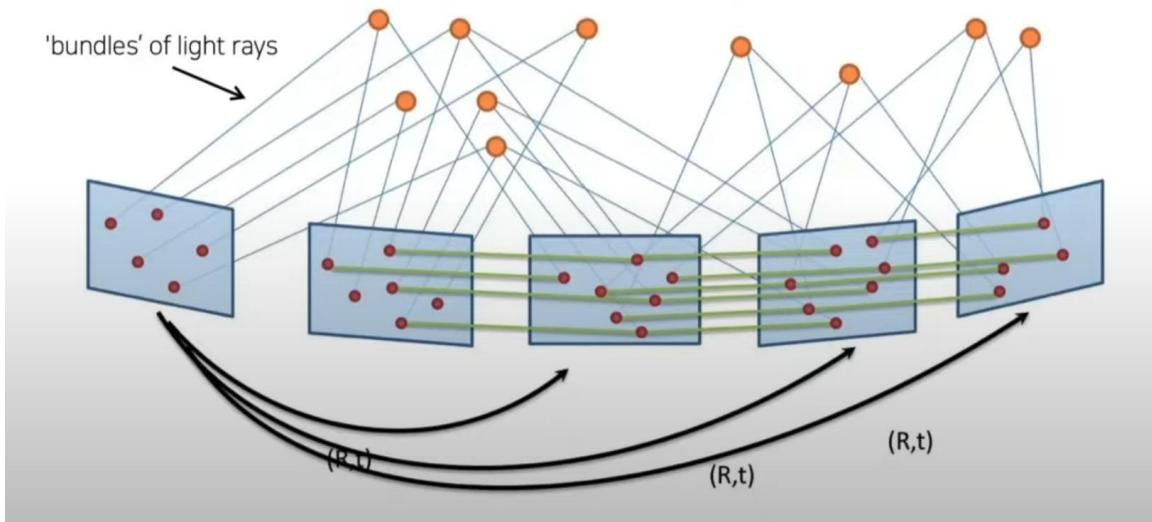
[Carloni, Visual Nav, MIT]

Standard SLAM pipeline consists of

- Sensors
- Front-end algorithms (real-time & online processing of sensor data, trajectory, map)
- Back-end algorithms (optimization & refinement of map, trajectory.)

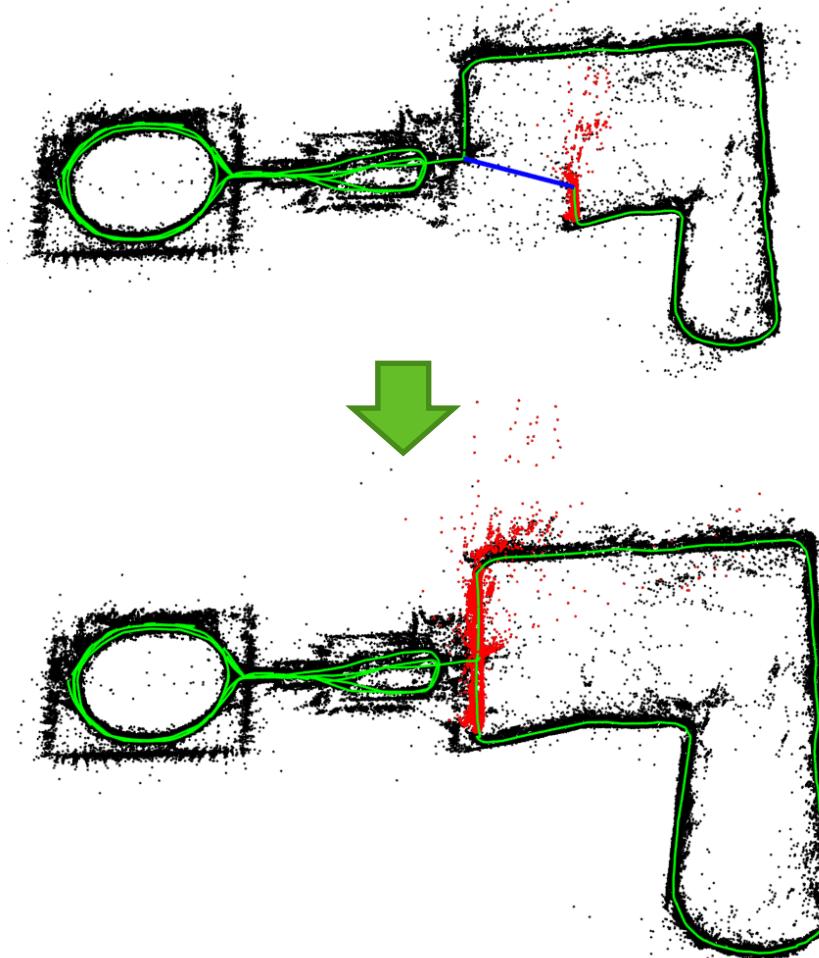
Loop Closure

Front-end or short-term
(feature tracking)



<https://youtu.be/z4ldKGh12ok>

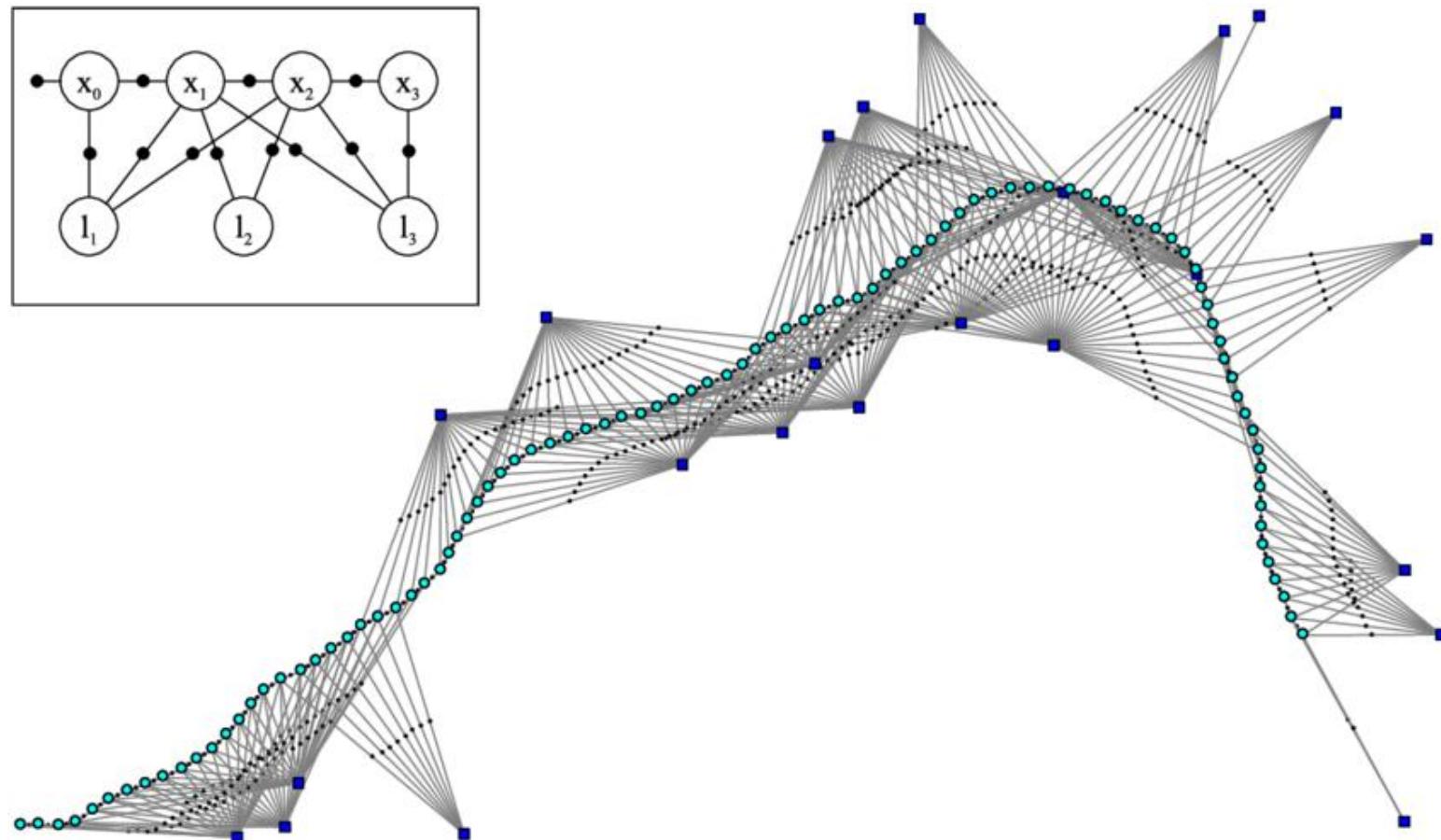
Back-end or long-term
(loop closure detection)



Back-End

SLAM Back-end deals with

- Optimization/refinement of map & trajectory (for **geometric consistency**)
- Finding loop closures correcting drift



Geometric Consistency

Example: Suppose we compute odometry estimates between t_1, t_2, t_3



Geometric Consistency

Example: Suppose we compute odometry estimates between t_1, t_2, t_3

- Estimates can be geometrically inconsistent (always are in practice!)



Geometric Consistency

Example: Suppose we compute odometry estimates between t_1, t_2, t_3

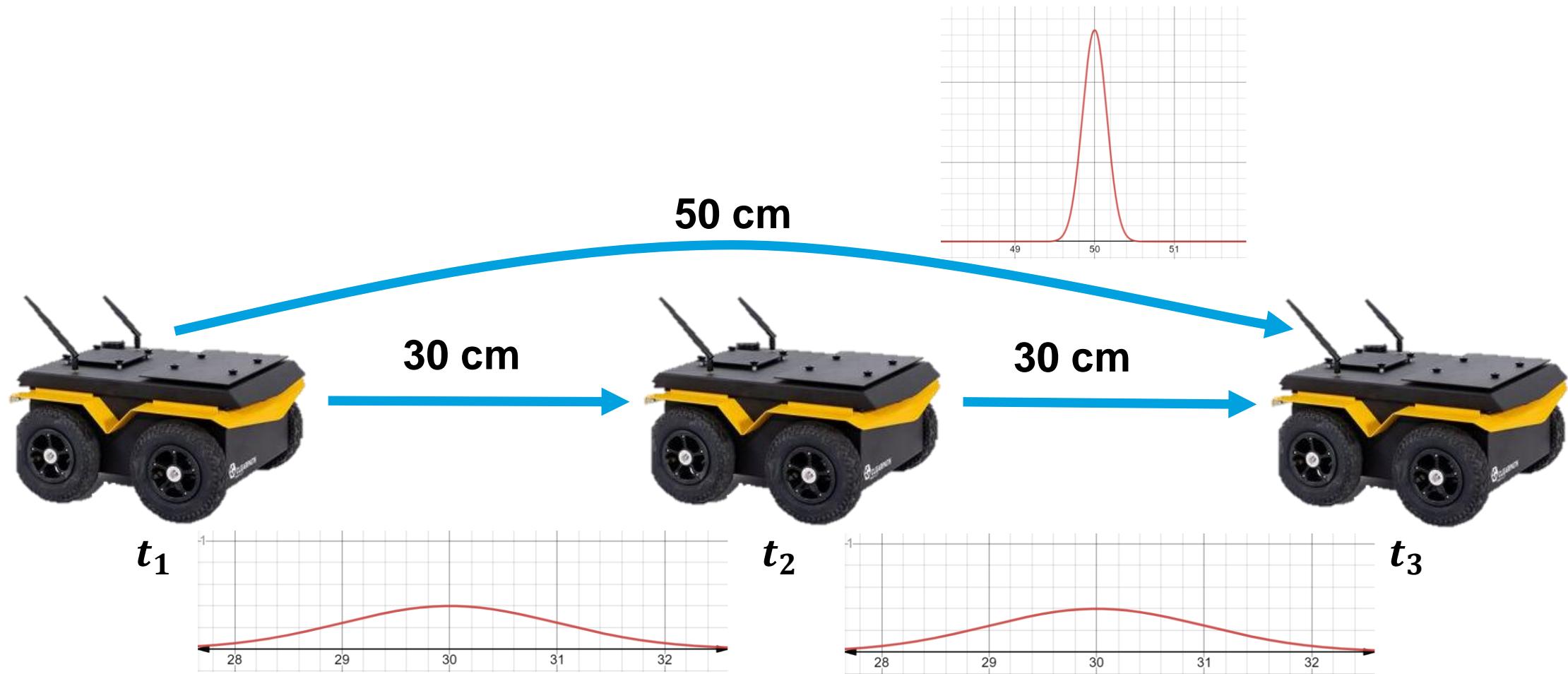
- Estimates can be geometrically inconsistent (always are in practice!)
- We can take “average” of the estimates to obtain consistent results



Geometric Consistency

Example: Suppose we compute odometry estimates between t_1, t_2, t_3

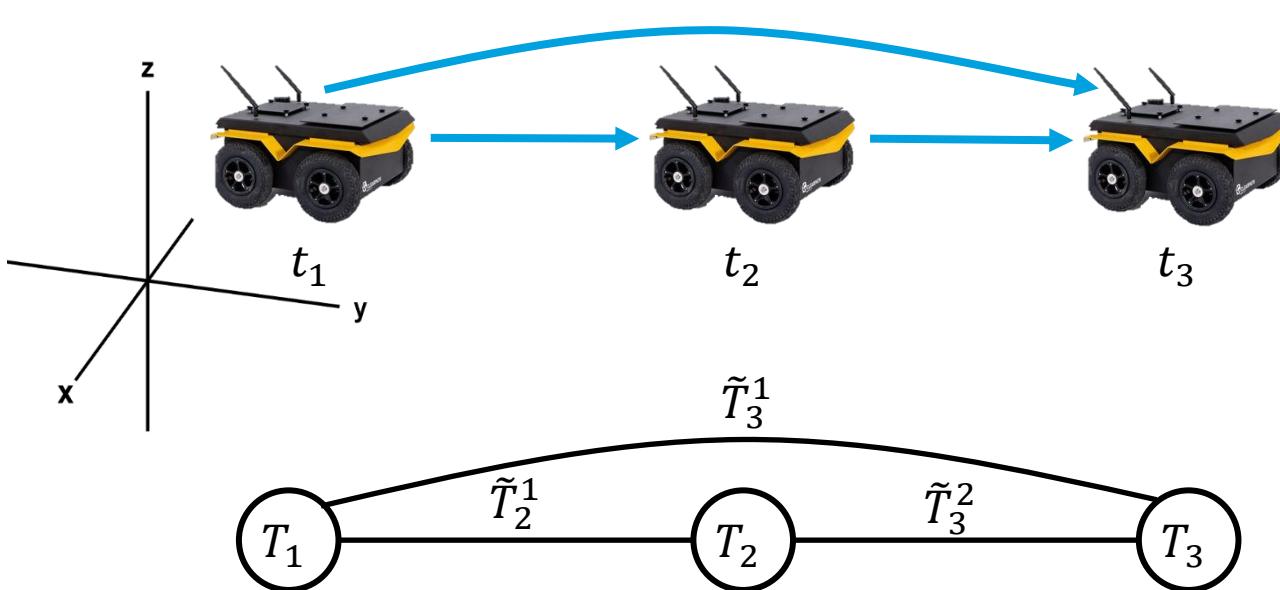
- Estimates can be geometrically inconsistent (always are in practice!)
- We can take “average” of the estimates to obtain consistent results
- What if we have a **probability** for correctness of estimates?



Factor Graph Optimization

Factor graph: Mathematical framework for solving estimation/inference problems & obtain consistency

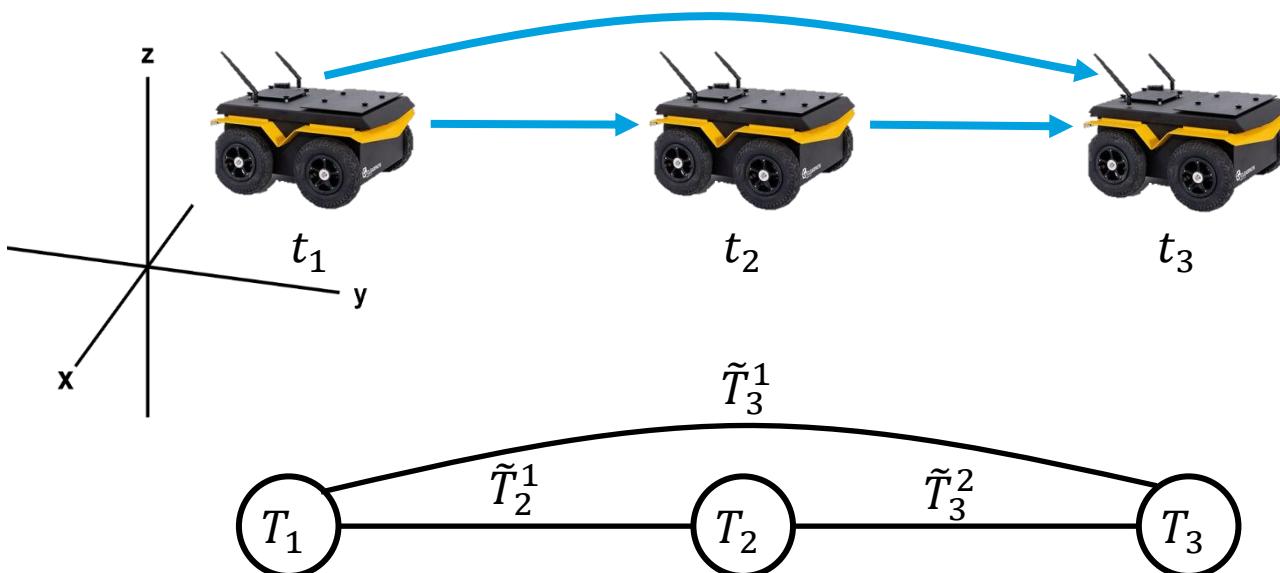
- Problem representation as a graph
- **Nodes/Vertices:** are optimization variables. Can be of different types (continuous or discrete, poses or landmarks, etc.)
- **Factors/Edges:** represent relationship/constraints between variables (e.g. relative pose estimates, sensor measurements)



Factor Graph Optimization

Factor graph: Mathematical framework for solving estimation/inference problems & obtain consistency

- Problem representation as a graph
- **Nodes/Vertices:** are optimization variables. Can be of different types (continuous or discrete, poses or landmarks, etc.)
- **Factors/Edges:** represent relationship/constraints between variables (e.g. relative pose estimates, sensor measurements)



Pose/factor graph optimization:

$$\min_{T_t \in \text{SE}(3)} \sum_{(i,j) \in \varepsilon} \| T_i^{-1} T_j - \tilde{T}_{ij} \|_{\Sigma_{ij}}^2$$

Solvers:

- GTSAM (<https://gtsam.org/>)
- G2o (<https://openslam-org.github.io/g2o.html>)
- Ceres (<http://ceres-solver.org/>)

Graph-Based SLAM

Three Main SLAM Paradigms

Kalman
filter

Particle
filter

Graph-
based



**least squares
approach to SLAM**

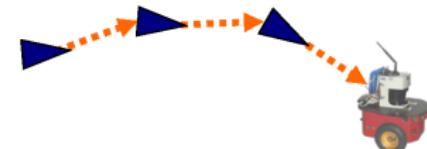
Least Squares in General

- Approach for computing a solution for an **overdetermined system**
- “More equations than unknowns”
- Minimizes the **sum of the squared errors** in the equations
- Standard approach to a large set of problems

Today: Application to SLAM

Graph-Based SLAM

- Constraints connect the poses of the robot while it is moving
- Constraints are inherently uncertain



Robot pose

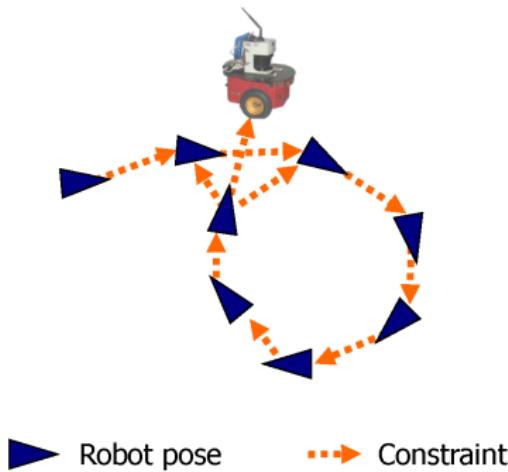


Constraint

Courtesy: C. Stachniss

Graph-Based SLAM

- Observing previously seen areas generates constraints between non-successive poses



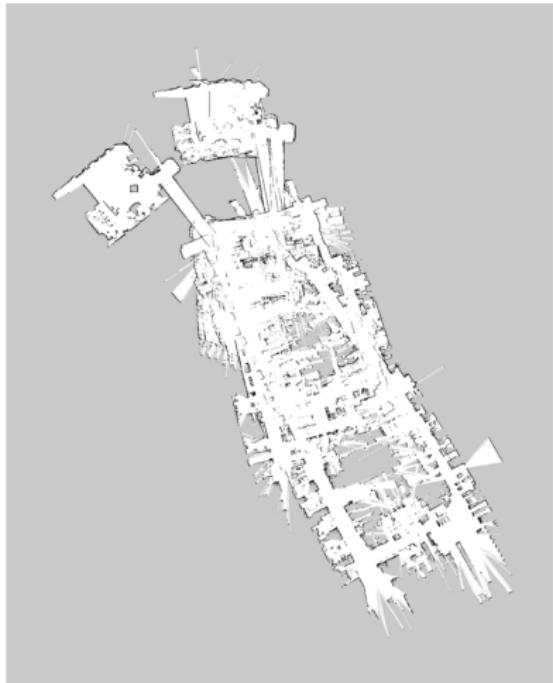
Courtesy: C. Stachniss

Idea of Graph-Based SLAM

- Use a **graph** to represent the problem
- Every **node** in the graph corresponds to a pose of the robot during mapping
- Every **edge** between two nodes corresponds to a spatial constraint between them
- **Graph-Based SLAM:** Build the graph and find a node configuration that minimizes the error introduced by the constraints

Graph-Based SLAM in a Nutshell

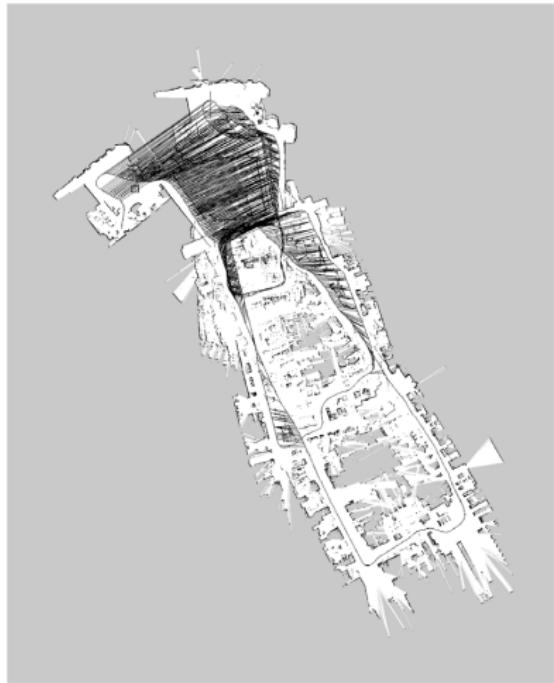
- Every node in the graph corresponds to a robot position and a laser measurement
- An edge between two nodes represents a spatial constraint between the nodes



KUKA Halle 22, courtesy of P. Pfaff

Graph-Based SLAM in a Nutshell

- Every node in the graph corresponds to a robot position and a laser measurement
- An edge between two nodes represents a spatial constraint between the nodes

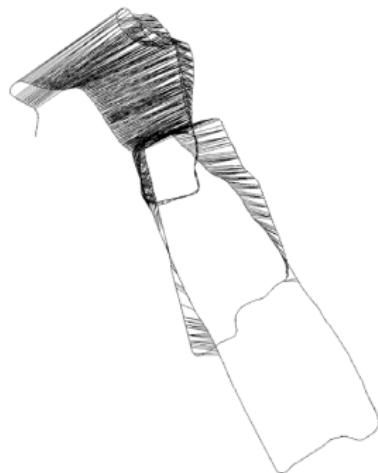


KUKA Halle 22, courtesy of P. Pfaff

Courtesy: C. Stachniss

Graph-Based SLAM in a Nutshell

- Once we have the graph, we determine the most likely map by correcting the nodes



Courtesy: C. Stachniss

Graph-Based SLAM in a Nutshell

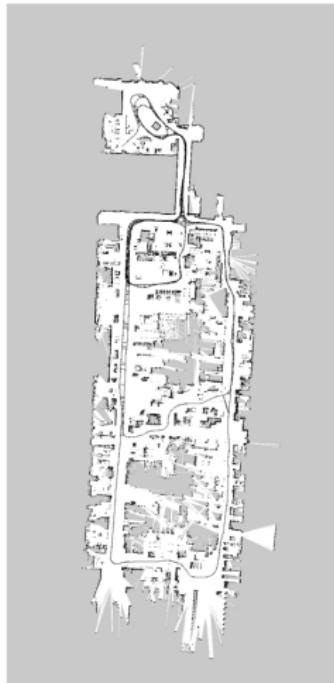
- Once we have the graph, we determine the most likely map by correcting the nodes
... like this



Courtesy: C. Stachniss

Graph-Based SLAM in a Nutshell

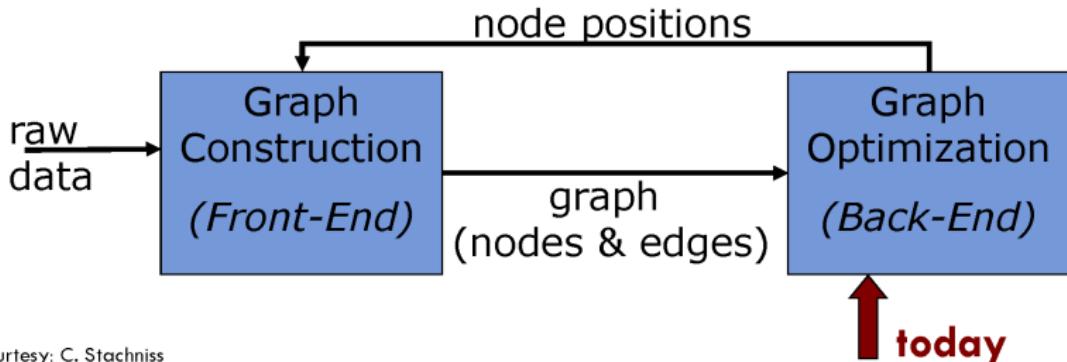
- Once we have the graph, we determine the most likely map by correcting the nodes
... like this
- Then, we can render a map based on the known poses



Courtesy: C. Stachniss

The Overall SLAM System

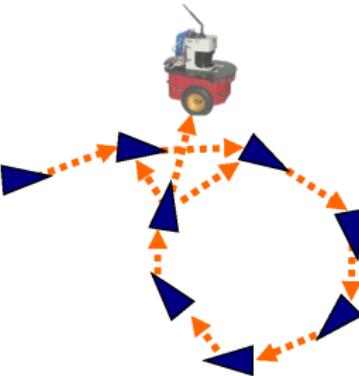
- Interplay of front-end and back-end
- Map helps to determine constraints by reducing the search space
- Topic today: optimization



Courtesy: C. Stachniss

The Graph

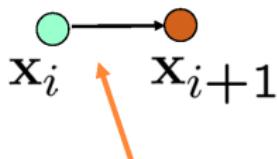
- It consists of n nodes $\mathbf{x} = \mathbf{x}_{1:n}$
- Each \mathbf{x}_i is a 2D or 3D transformation (the pose of the robot at time t_i)
- A constraint/edge exists between the nodes \mathbf{x}_i and \mathbf{x}_j if...



Courtesy: C. Stachniss

Create an Edge If... (1)

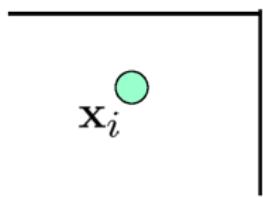
- ...the robot moves from \mathbf{x}_i to \mathbf{x}_{i+1}
- Edge corresponds to odometry



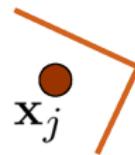
The edge represents the
odometry measurement

Create an Edge If... (2)

- ...the robot observes the same part of the environment from \mathbf{x}_i and from \mathbf{x}_j



Measurement from \mathbf{x}_i

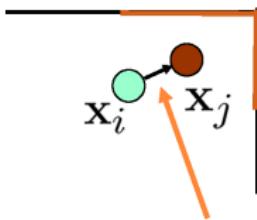


Measurement from \mathbf{x}_j

Courtesy: C. Stachniss

Create an Edge If... (2)

- ...the robot observes the same part of the environment from \mathbf{x}_i and from \mathbf{x}_j
- Construct a **virtual measurement** about the position of \mathbf{x}_j seen from \mathbf{x}_i



Edge represents the position of x_j seen from x_i based on the **observation**

Transformations

- Transformations can be expressed using **homogenous coordinates**
- Odometry-Based edge

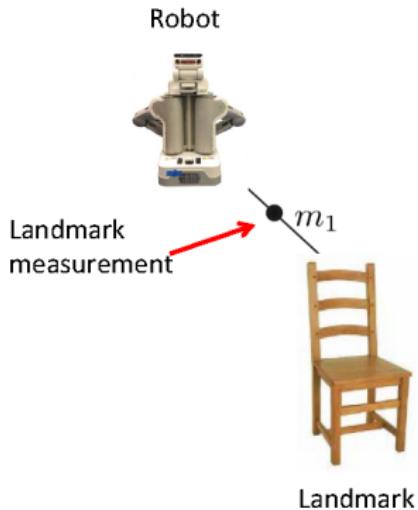
$$(\mathbf{X}_i^{-1} \mathbf{X}_{i+1})$$

- Observation-Based edge

$$(\mathbf{X}_i^{-1} \mathbf{X}_j)$$

How node i sees node j

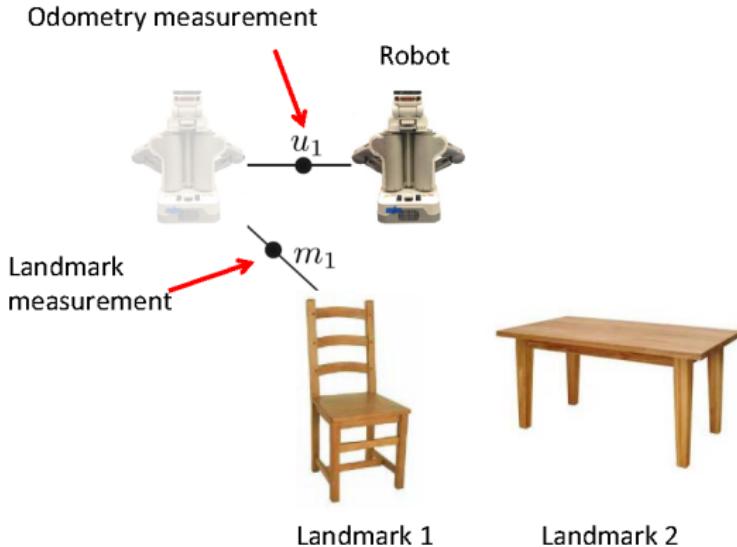
The SLAM Problem ($t=0$)



Onboard sensors:

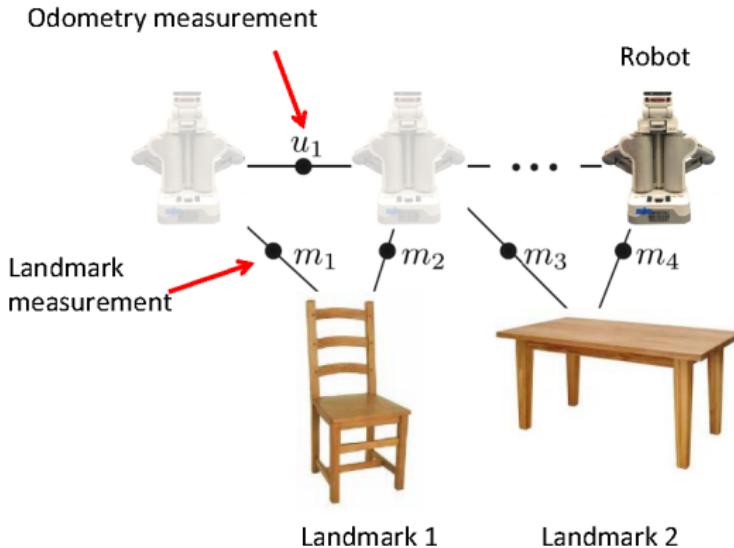
- Wheel odometry
- Inertial measurement unit (gyro, accelerometer)
- Sonar
- Laser range finder
- Camera
- RGB-D sensors

The SLAM Problem ($t=1$)



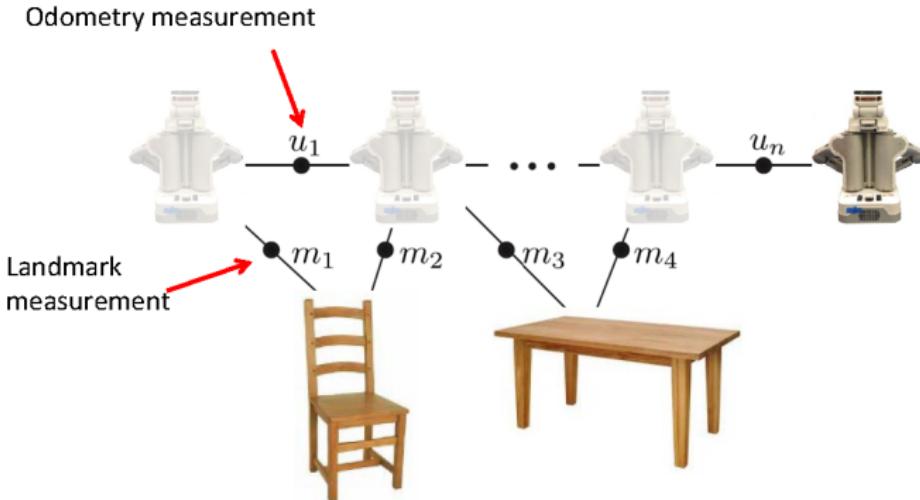
Courtesy: M. Kaess

The SLAM Problem ($t=n-1$)



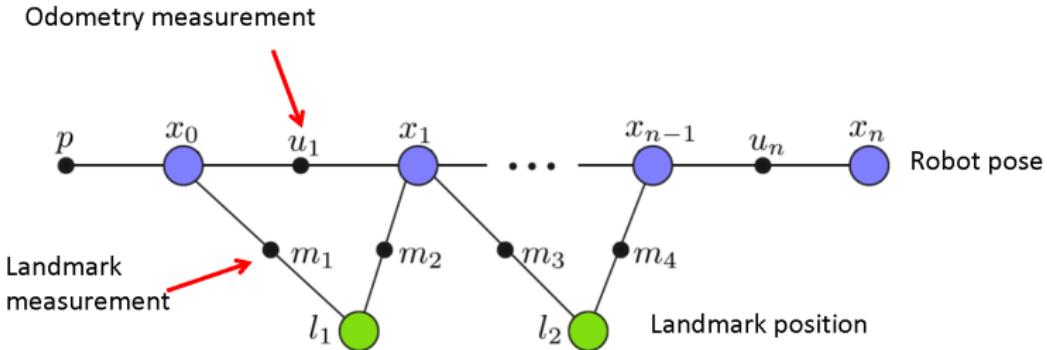
Courtesy: M. Kaess

The SLAM Problem ($t=n$)



Courtesy: M. Kaess

Factor Graph Representation

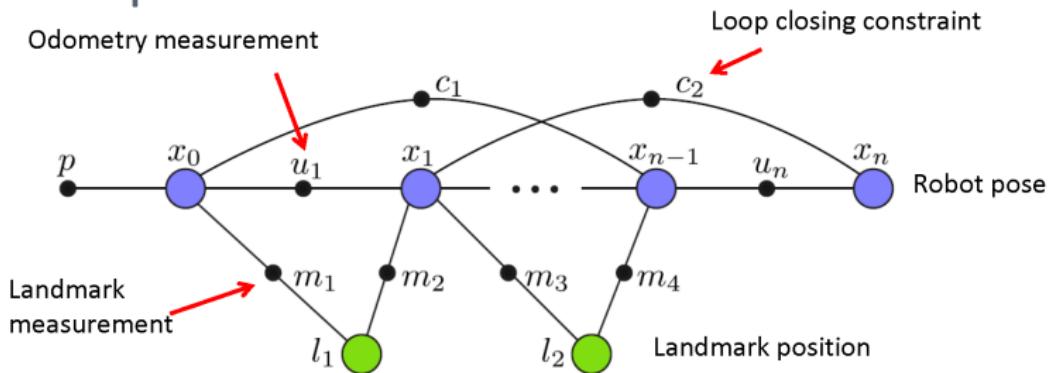


Bipartite graph with ***variable nodes*** and ***factor nodes***



Courtesy: M. Kaess

Factor Graph Representation: Pose Graph

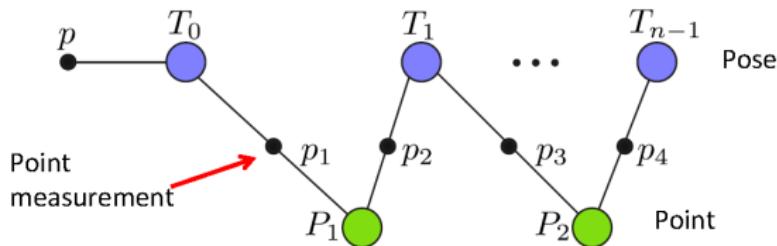


Bipartite graph with ***variable nodes*** and ***factor nodes***



Courtesy: M. Kaess

Factor Graph Representation: Bundle Adjust.

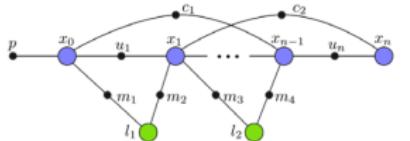


Bipartite graph with ***variable nodes*** and ***factor nodes***



Courtesy: M. Kaess

Nonlinear Least-Squares

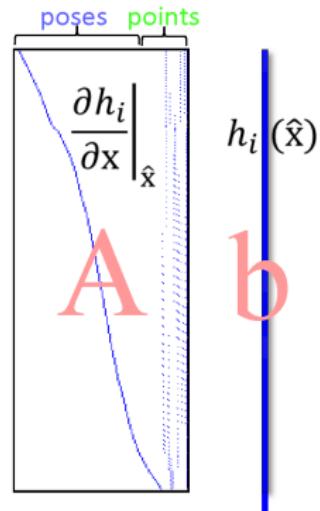


$$\operatorname{argmin}_x \sum_i \|h_i(x)\|_{\Sigma}^2$$

Repeatedly solve linearized system (GN)

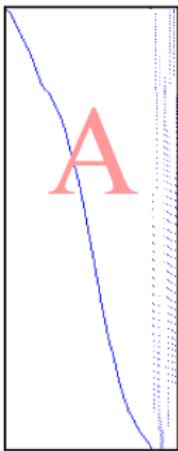
$$\operatorname{argmin}_x \|Ax - b\|^2$$

$$A = \begin{bmatrix} F_{11} & G_{11} & & \\ F_{12} & & G_{12} & \\ F_{13} & & & G_{13} \\ F_{21} & G_{21} & & \\ F_{22} & & G_{22} & \\ F_{23} & & & G_{23} \end{bmatrix}, \quad x = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix}, \quad b = \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \\ b_{16} \end{bmatrix}$$



Solving the Linear Least-Squares System

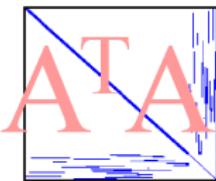
Solve: $\operatorname{argmin}_x \|Ax - b\|^2$



Measurement Jacobian

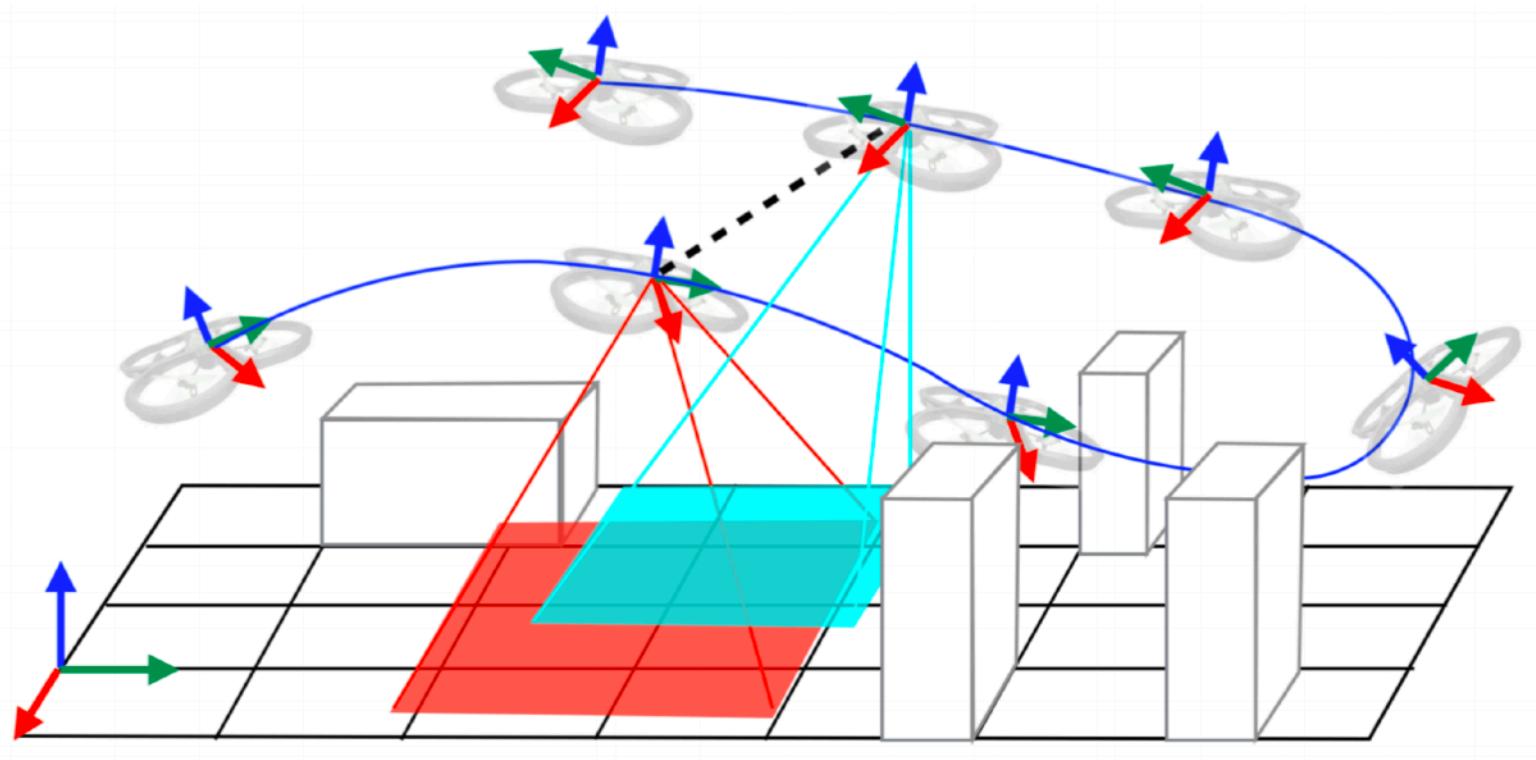
Normal equations

$$A^T A x = A^T b$$



Information matrix

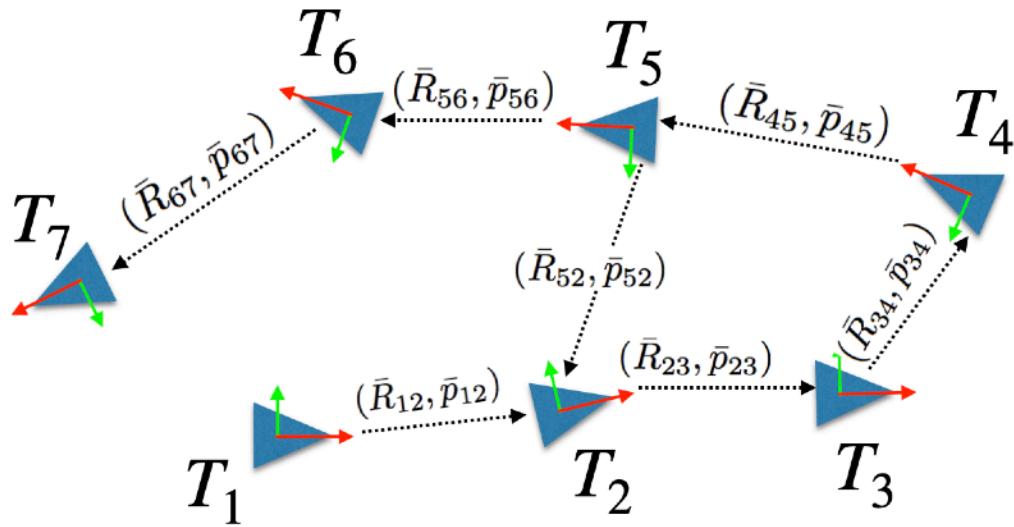
Pose Graph Optimization



- **Measurements:** odometry + loop closures (relative poses)
- **Variables:** robot poses

Pose Graph Optimization

pose
graph

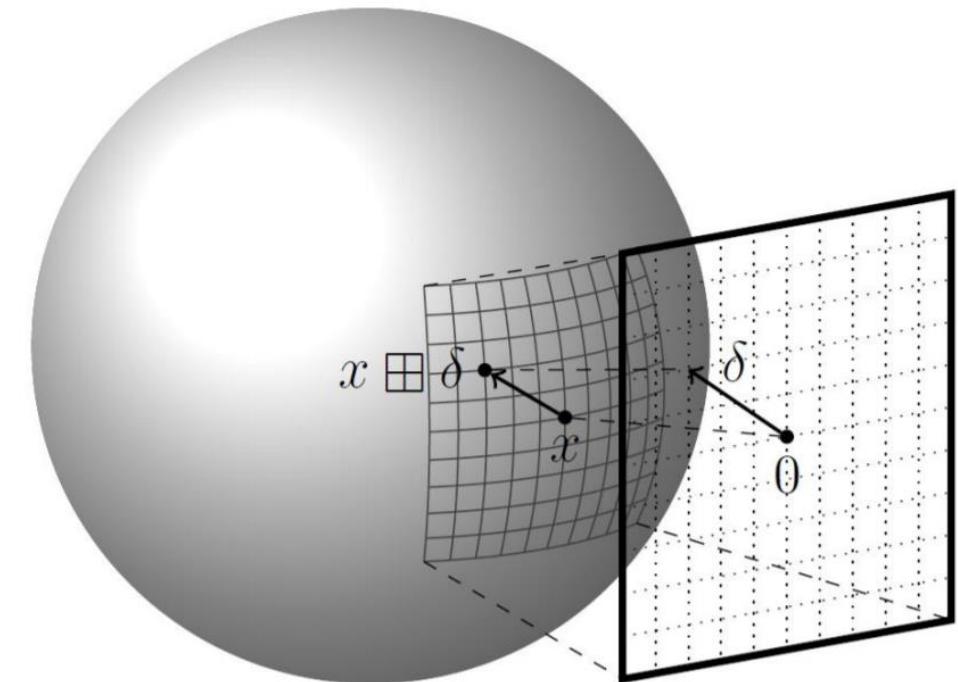


pose graph
optimization

$$\min_{\mathbf{T}_t, t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}} \|(\mathbf{T}_i^{-1} \mathbf{T}_j) \boxminus \bar{\mathbf{T}}_j^i\|_{\Sigma_{ij}}^2$$

NLLS ON SMOOTH MANIFOLDS

- Our previous least-squares formulation assumed the parameters are Euclidean
 - **Not true** for SLAM
 - **Pose** estimates are SE(3)
- Box-plus and box-minus
 - $\boxplus : \mathcal{S} \times \mathbb{R}^n \rightarrow \mathcal{S}$,
 - $\boxminus : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^n$



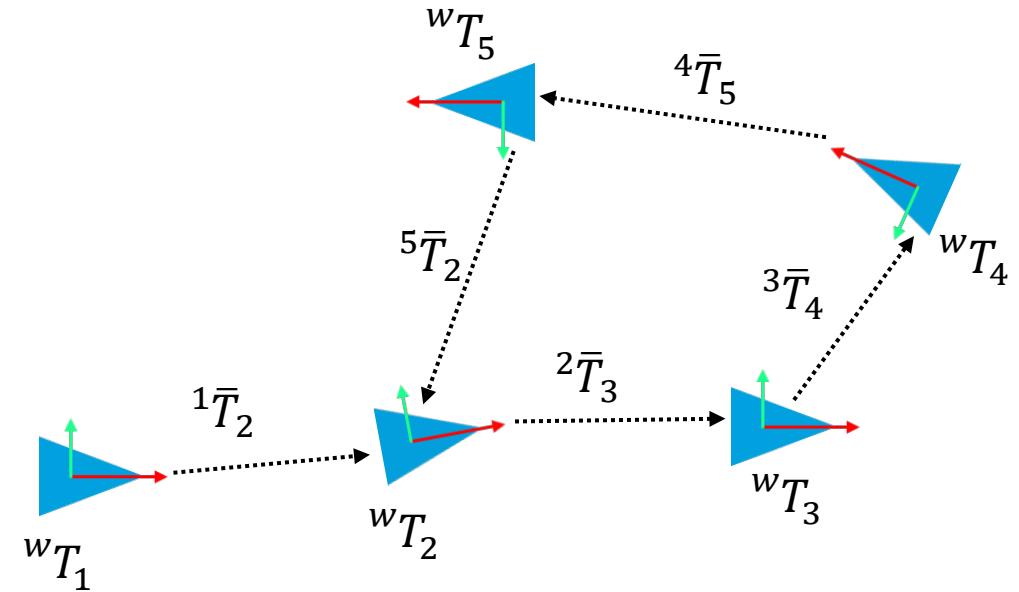
Example

$$\min_{T_t, t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}} \| (T_i^{-1} T_j) \boxminus \bar{T}_j^i \|_{\Sigma_{ij}}^2$$



$$\min_{{}^w T_1, \dots, {}^w T_5 \in \text{SE}(3)} \|({}^w T_1^{-1} {}^w T_2) - {}^1 \bar{T}_2\| + \|({}^w T_2^{-1} {}^w T_3) - {}^2 \bar{T}_3\| + \dots \|({}^w T_5^{-1} {}^w T_2) - {}^5 \bar{T}_2\|$$

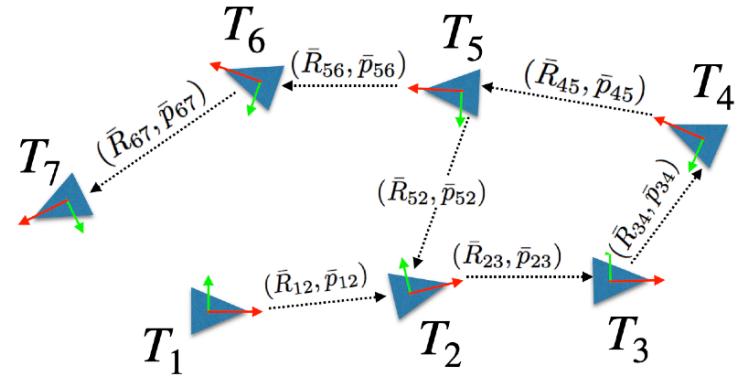
Note: $T = \begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix} \in \text{SE}(3)$ is the homogeneous transformation matrix



Pose Graph Optimization: Sparsity

Jacobian \mathbf{J}

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
\bar{T}_{12}	■	■					
\bar{T}_{23}		■	■				
\bar{T}_{34}			■	■			
\bar{T}_{45}				■	■		
\bar{T}_{56}					■	■	
\bar{T}_{67}		■				■	
\bar{T}_{52}			■				

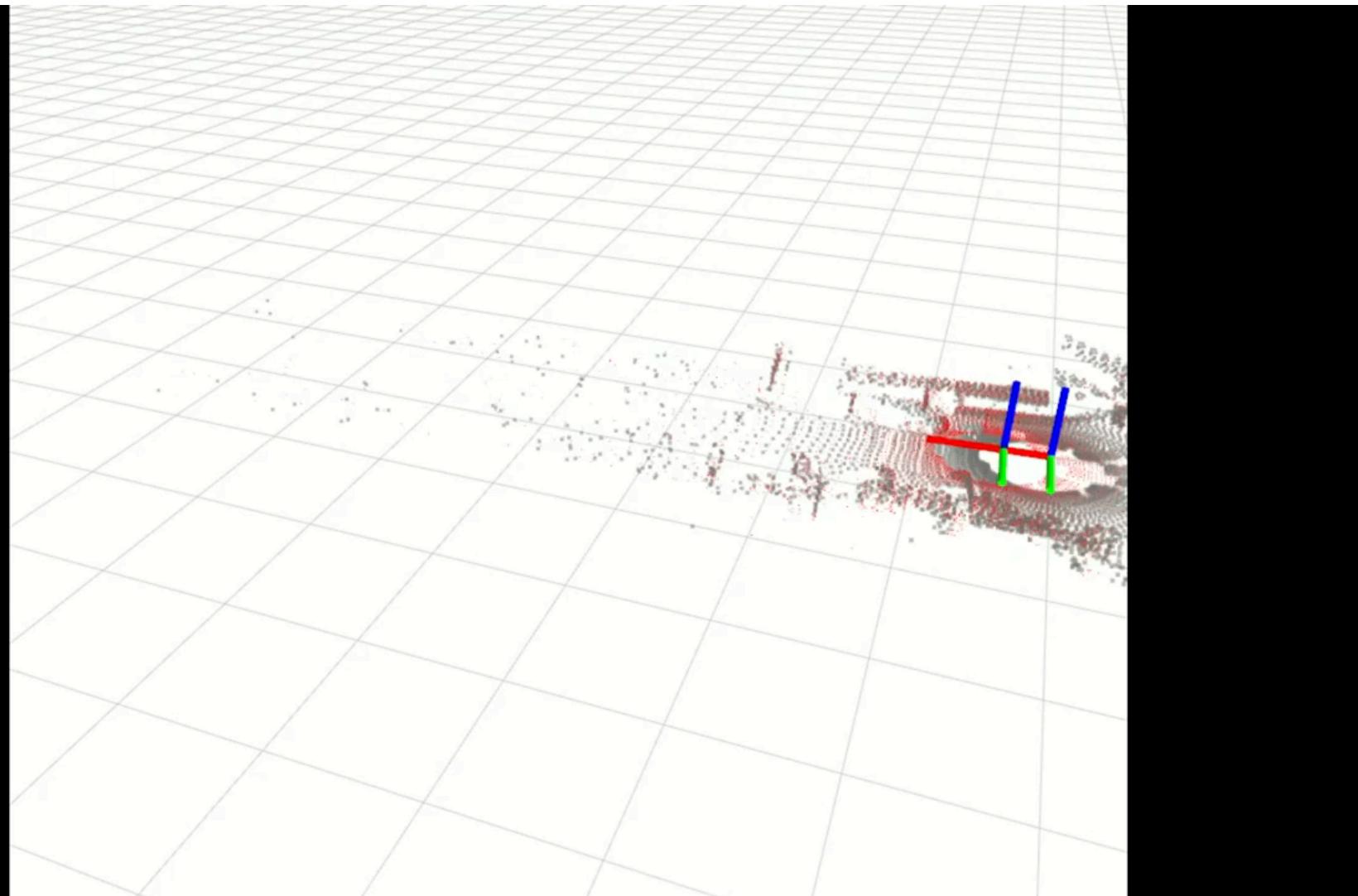


Hessian $\mathbf{J}^\top \mathbf{J}$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
T_1	■						
T_2		■					
T_3			■				
T_4				■			
T_5					■		
T_6						■	
T_7							■

a.k.a.
Information
Matrix of
the estimate

Pose Graph Optimization: Example

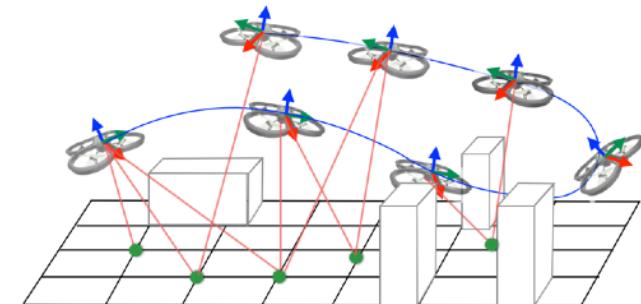
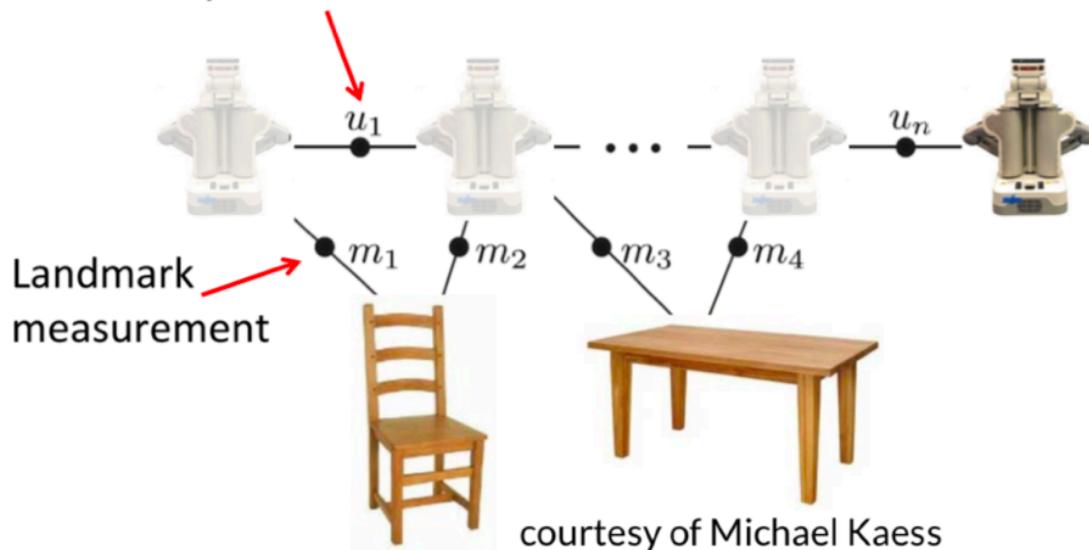


https://www.youtube.com/watch?v=KYvOqUB_0dg

Landmark-based SLAM

- ▶ Sequence of robot (camera) poses $\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_t \in \text{SE}(d)$
- ▶ Robot measures the relative pose between \mathbf{T}_i and \mathbf{T}_{i+1} (odometry)
- ▶ Robot measures the environment (e.g., point landmarks $\mathbf{p}_i \in \mathbb{R}^d$)

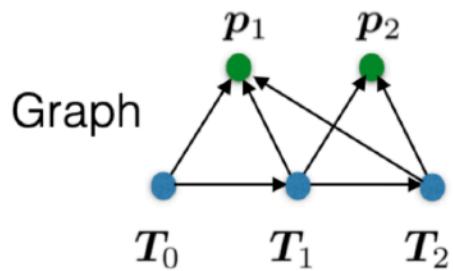
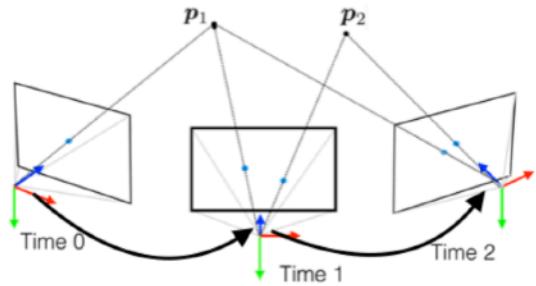
Odometry measurement



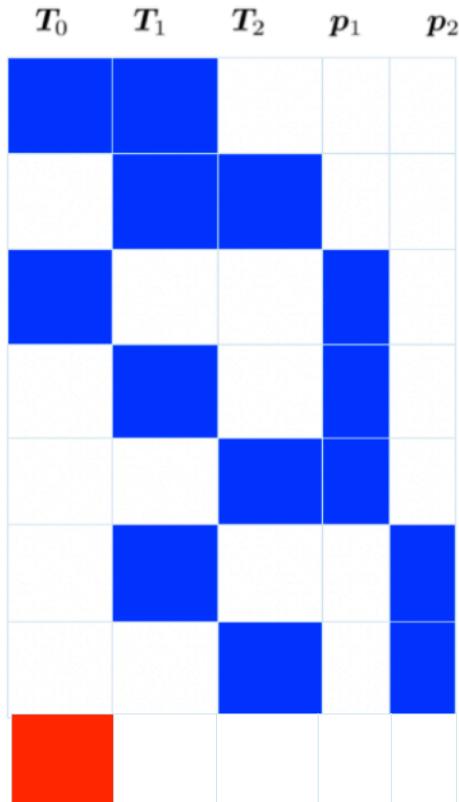
- **Measurements:** odometry + measurements of (projection, range, position, or others) of external landmarks
- **Variables:** robot poses and landmark positions

Landmark-based SLAM: Sparsity

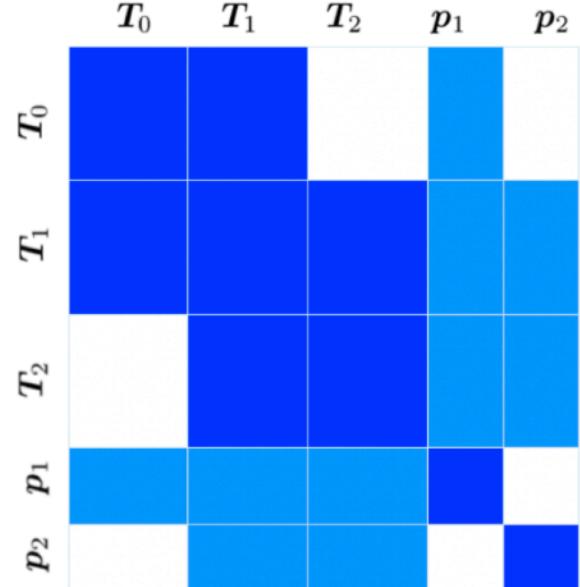
$$\min_{\substack{\mathbf{T}_t, t=1,\dots,n \\ \mathbf{l}_k, k=1,\dots,K}} \sum_{t=1,\dots,n-1} \|(\mathbf{T}_t^{-1} \mathbf{T}_{t+1}) \boxminus \bar{\mathbf{T}}_{t+1}^t\|_{\Sigma_o}^2 + \sum_{k=1,\dots,K} \sum_{t \in \mathcal{S}_k} \|\bar{\mathbf{y}}_{k,t} - h_i(\mathbf{T}_t, \mathbf{l}_k)\|_{\Sigma_l}^2$$



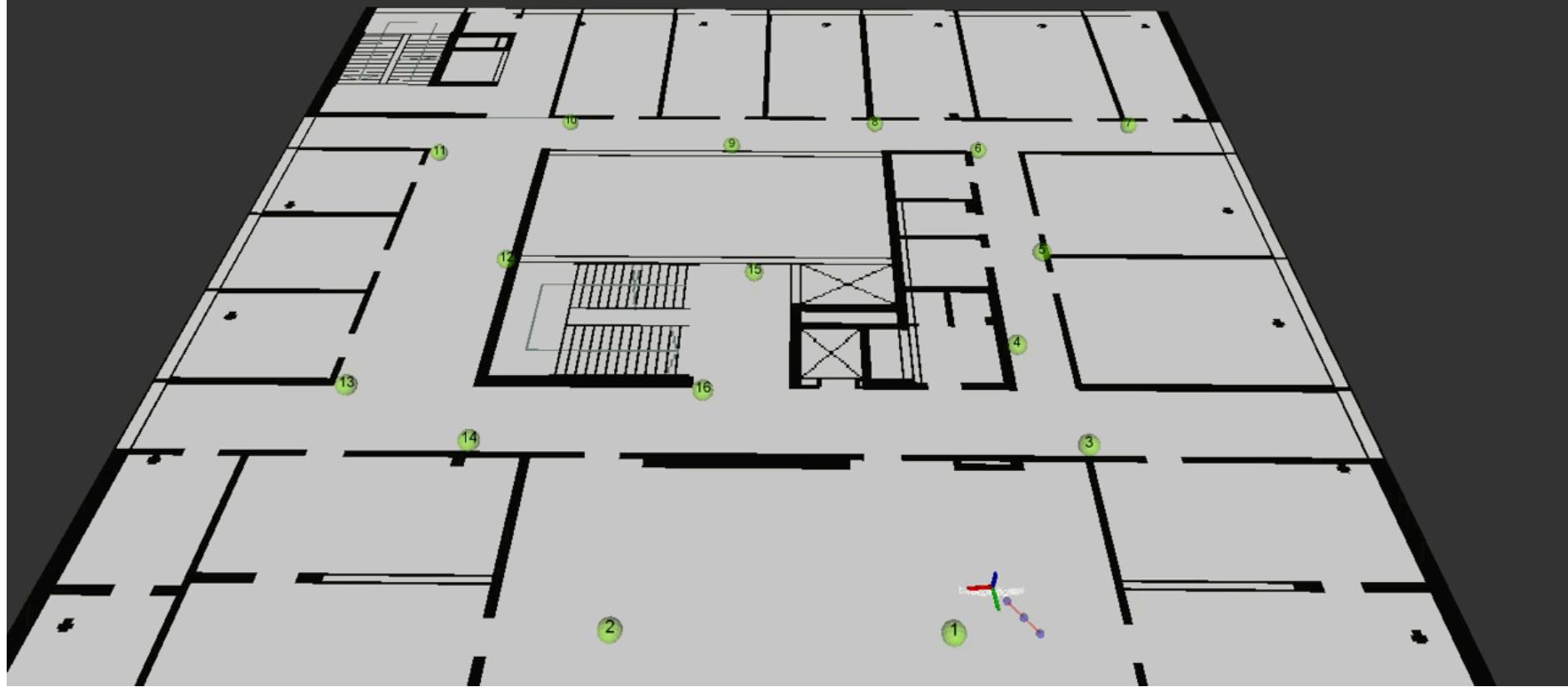
Jacobian \mathbf{J}



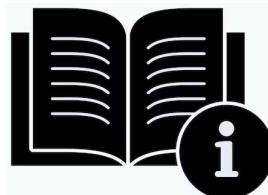
Hessian $\mathbf{J}^\top \mathbf{J}$



Landmark-based SLAM: Example



https://www.youtube.com/watch?v=OdJ042prg_M



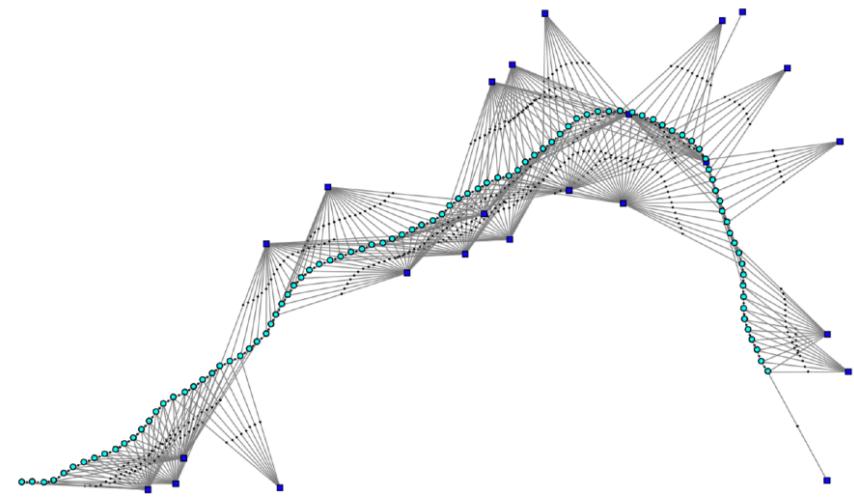
Smoothing and Filtering

MAP or Full smoothing (estimate entire trajectory and map)

- ▶ **Many** variables but
- ▶ Information matrix $\mathbf{J}^\top \Sigma^{-1} \mathbf{J}$ is **sparse**

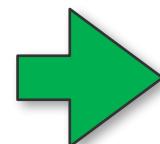
Fixed-lag smoothing (estimate only variables in a time window)

- ▶ Use Schur complement to marginalize out old states (hence **less** variables)
- ▶ Information matrix after Schur complement is **denser**



Filtering (estimate only current pose and landmarks)

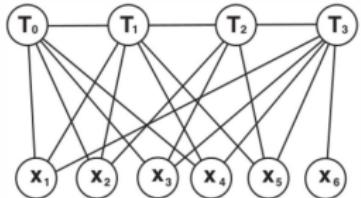
- ▶ Use Schur complement to marginalize out **ALL** old states (hence **few variables**)
- ▶ Information matrix after Schur complement is typically **dense**



Kalman filter,
Extended Kalman Filter



Full Bundle Adjustment



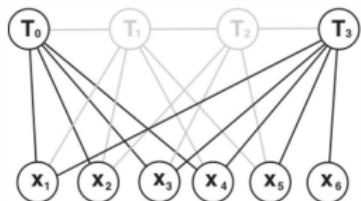
From Strasdat et al, 2011 IVC “Visual SLAM: Why filter?”

- Graph grows with time:
 - Have to solve a sequence of increasingly larger BA problems
 - Will become too expensive even for sparse Cholesky

F. Dellaert and M. Kaess, “Square Root SAM: Simultaneous localization and mapping via square root information smoothing,” IJRR 2006



Keyframe Bundle Adjustment



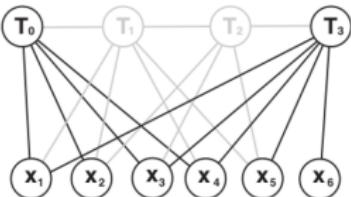
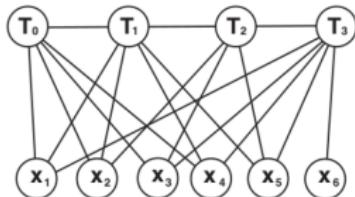
- Drop subset of poses to reduce density/complexity
- Only retain “keyframes” necessary for good map

- Complexity still grows with time, just slower



Incremental Solver

- Back to full BA and keyframes:



- New information is added to the graph
- Older information does not change
- Can be exploited to obtain an efficient solution!



iSAM

Solving a growing system:

- ❑ Exact/batch (quickly gets expensive)
- ❑ Approximations
- ❑ Incremental Smoothing and Mapping (iSAM)

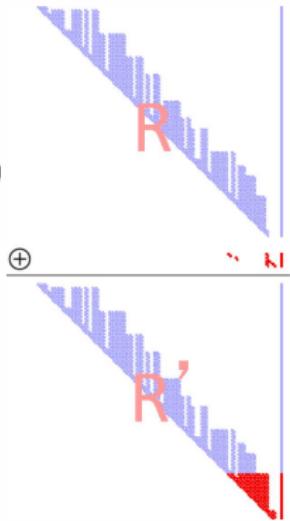
Key idea:

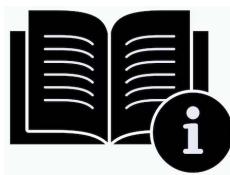
- ❑ Append to existing matrix factorization
- ❑ “Repair” using Givens rotations

Periodic batch steps for

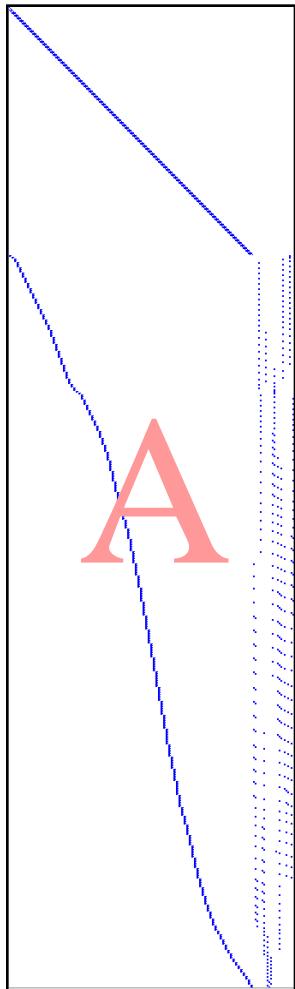
- ❑ Relinearization
- ❑ Variable reordering (to keep sparsity)

New measurements ->





Solving the Linear Least-Squares System

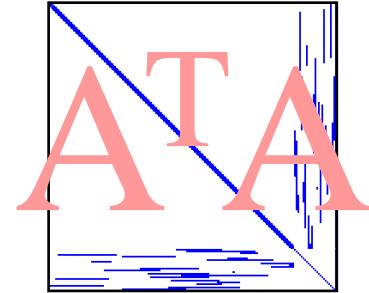


Measurement Jacobian

Solve: $\operatorname{argmin}_{\theta} \|A\theta - b\|^2$

Normal equations

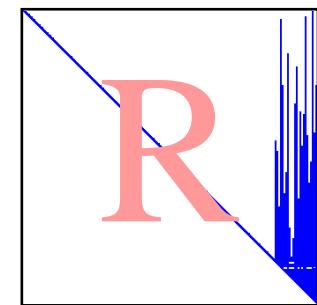
$$A^T A \theta = A^T b$$



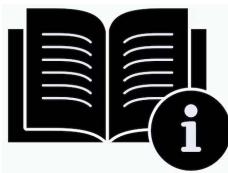
Information matrix

Matrix factorization

$$A^T A = R^T R$$

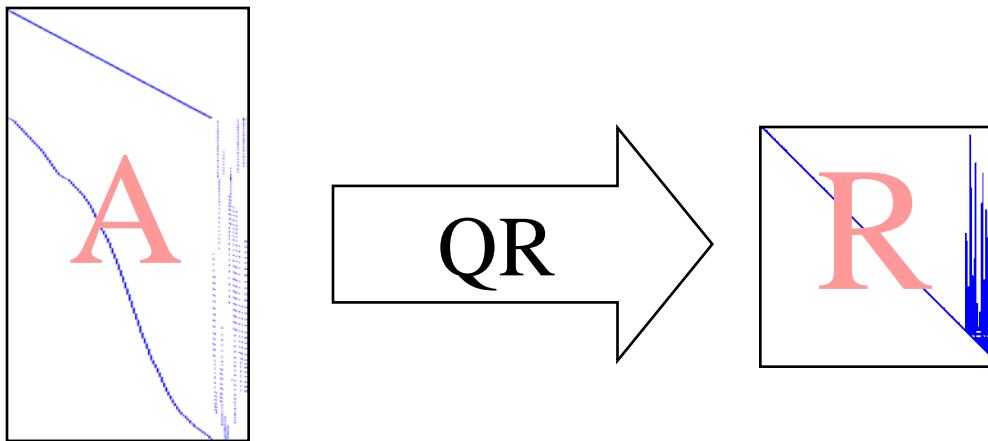


Square root information matrix

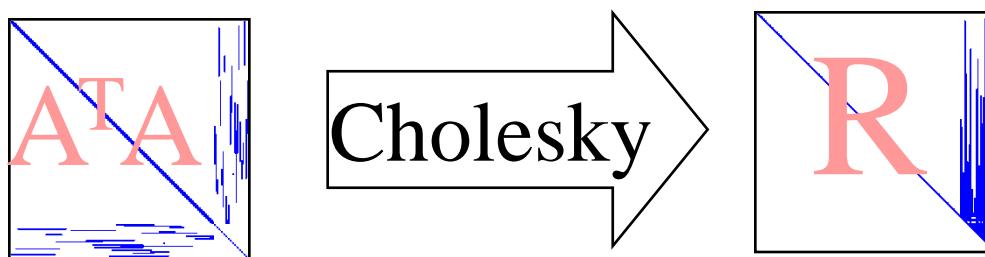


Matrix – Square Root Factorization

- QR on A: Numerically Stable

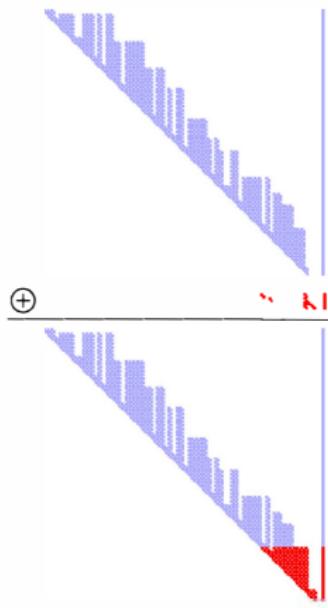


- Cholesky on $A^T A$: Faster





Factor Updates with Givens Rotations



Old R factor

New rows

New R factor:

Triangulated using
Givens Rotations

Constant time!

Zeroing entries with Givens
Rotations:

$$\begin{array}{c} \text{row } k \\ \text{row } i \end{array} \xrightarrow{\text{Givens}} \begin{matrix} 1 & c & s \\ -s & 1 & c \end{matrix} \cdot \begin{matrix} x \\ R \end{matrix} = \begin{matrix} 0 \\ R^* \end{matrix}$$

Numerically stable !



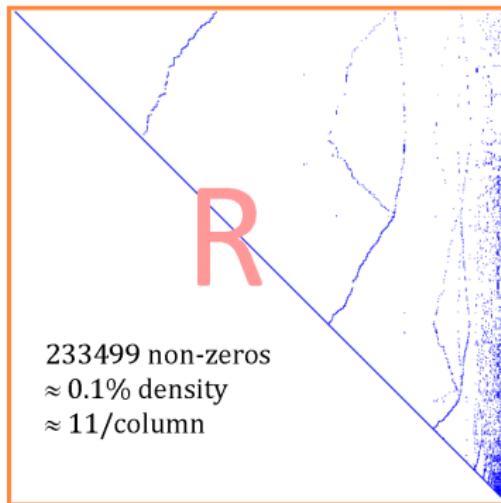
iSAM

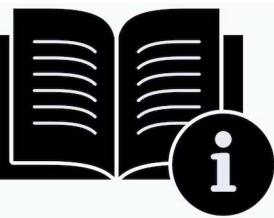
Example from real sequence:

Square root inf. matrix 

Side length: 21000 variables

Dense: 1.7GB, sparse: 1MB



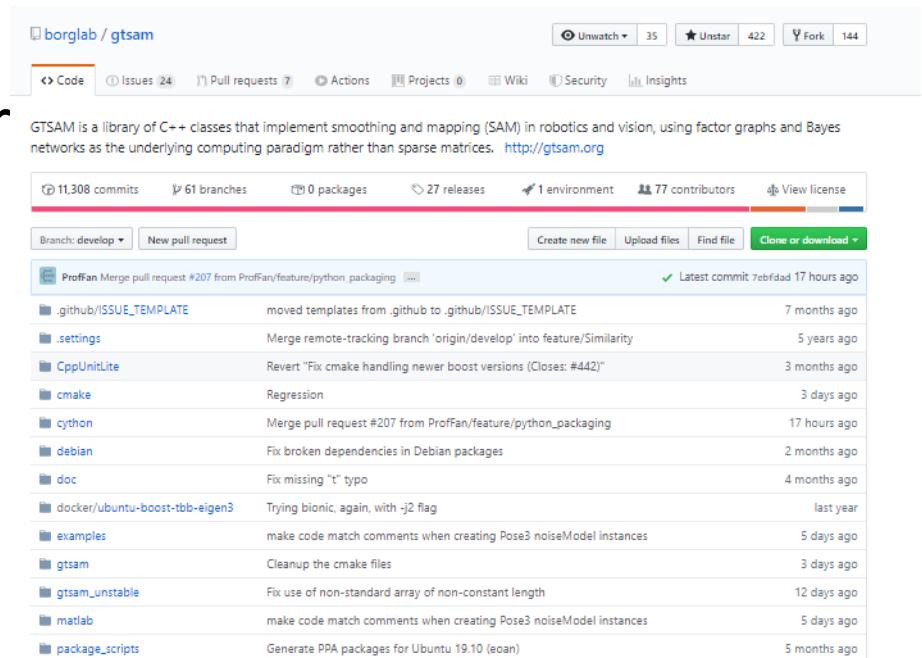


GTSAM

GTSAM

- C++ library.
- Frank Dellaert, Georgia Tech “Institute for Robotics and Intelligent Machines”.
- Tailored to SLAM but coded as a general purpose library.
- Matlab and Python bindings.

<https://github.com/borglab/gtsam/>



GTSAM

- Main C++ classes:
 - NonlinearFactorGraph : The Factor graph.
 - FactorXXX: Factors.
 - Arguments:
 - N Keys. Can be built with “Symbol” (e.g. “X1”, “V2”).
 - Observed value.
 - Noise model, e.g. “noiseModel::Diagonal::Sigmas()”
 - Values : “Variant” container for initial and final values of all keys.
Associative container: Key → Value.

GTSAM: PlanarSLAMExample.cpp

```
// As this is a planar SLAM example, we will use Pose2 variables (x, y, theta) to represent
// the robot positions and Point2 variables (x, y) to represent the landmark coordinates.
#include <gtsam/geometry/Pose2.h>
#include <gtsam/geometry/Point2.h>

// Each variable in the system (poses and landmarks) must be identified with a unique key.
// We can either use simple integer keys (1, 2, 3, ...) or symbols (X1, X2, L1).
// Here we will use Symbols
#include <gtsam/inference/Symbol.h>

// In GTSAM, measurement functions are represented as 'factors'. Several common factors
// have been provided with the library for solving robotics/SLAM/Bundle Adjustment problems.
// Here we will use a RangeBearing factor for the range-bearing measurements to identified
// landmarks, and Between factors for the relative motion described by odometry measurements.
// Also, we will initialize the robot at the origin using a Prior factor.
#include <gtsam/slam/PriorFactor.h>
#include <gtsam/slam/BetweenFactor.h>
#include <gtsam/sam/BearingRangeFactor.h>

// When the factors are created, we will add them to a Factor Graph. As the factors we are using
// are nonlinear factors, we will need a Nonlinear Factor Graph.
#include <gtsam/nonlinear/NonlinearFactorGraph.h>
```

GTSAM: PlanarSLAMExample.cpp

```
// Finally, once all of the factors have been added to our factor graph, we will want to
// solve/optimize to graph to find the best (Maximum A Posteriori) set of variable values.
// GTSAM includes several nonlinear optimizers to perform this step. Here we will use the
// common Levenberg-Marquardt solver
#include <gtsam/nonlinear/LevenbergMarquardtOptimizer.h>

// Once the optimized values have been calculated, we can also calculate the marginal covariance
// of desired variables
#include <gtsam/nonlinear/Marginals.h>

// The nonlinear solvers within GTSAM are iterative solvers, meaning they linearize the
// nonlinear functions around an initial linearization point, then solve the linear system
// to update the linearization point. This happens repeatedly until the solver converges
// to a consistent set of variable values. This requires us to specify an initial guess
// for each variable, held in a Values container.
#include <gtsam/nonlinear/Values.h>

using namespace std;
using namespace gtsam;

int main(int argc, char** argv) {

    // Create a factor graph
    NonlinearFactorGraph graph;

    // Create the keys we need for this simple example
    static Symbol x1('x',1), x2('x',2), x3('x',3);
    static Symbol l1('l',1), l2('l',2);
```

GTSAM: PlanarSLAMExample.cpp

```
// Add a prior on pose x1 at the origin. A prior factor consists of a mean and a noise model (covariance
Pose2 prior(0.0, 0.0, 0.0); // prior mean is at origin
noiseModel::Diagonal::shared_ptr priorNoise = noiseModel::Diagonal::Sigmas(Vector3(0.3, 0.3, 0.1)); // 3
graph.emplace_shared<PriorFactor<Pose2>>(x1, prior, priorNoise); // add directly to graph

// Add two odometry factors
Pose2 odometry(2.0, 0.0, 0.0); // create a measurement for both factors (the same in this case)
noiseModel::Diagonal::shared_ptr odometryNoise = noiseModel::Diagonal::Sigmas(Vector3(0.2, 0.2, 0.1)); /
graph.emplace_shared<BetweenFactor<Pose2>>(x1, x2, odometry, odometryNoise);
graph.emplace_shared<BetweenFactor<Pose2>>(x2, x3, odometry, odometryNoise);

// Add Range-Bearing measurements to two different landmarks
// create a noise model for the landmark measurements
noiseModel::Diagonal::shared_ptr measurementNoise = noiseModel::Diagonal::Sigmas(Vector2(0.1, 0.2)); //
// create the measurement values - indices are (pose id, landmark id)
Rot2 bearing11 = Rot2::fromDegrees(45),
bearing21 = Rot2::fromDegrees(90),
bearing32 = Rot2::fromDegrees(90);
double range11 = std::sqrt(4.0+4.0),
range21 = 2.0,
range32 = 2.0;

// Add Bearing-Range factors
graph.emplace_shared<BearingRangeFactor<Pose2, Point2>>(x1, 11, bearing11, range11, measurementNoise);
graph.emplace_shared<BearingRangeFactor<Pose2, Point2>>(x2, 11, bearing21, range21, measurementNoise);
graph.emplace_shared<BearingRangeFactor<Pose2, Point2>>(x3, 12, bearing32, range32, measurementNoise);

// Print
graph.print("Factor Graph:\n");
```

GTSAM: PlanarSLAMExample.cpp

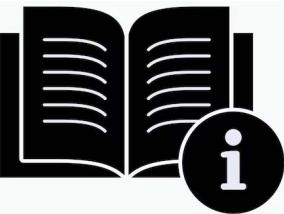
```
// Create (deliberately inaccurate) initial estimate
Values initialEstimate;
initialEstimate.insert(x1, Pose2(0.5, 0.0, 0.2));
initialEstimate.insert(x2, Pose2(2.3, 0.1,-0.2));
initialEstimate.insert(x3, Pose2(4.1, 0.1, 0.1));
initialEstimate.insert(l1, Point2(1.8, 2.1));
initialEstimate.insert(l2, Point2(4.1, 1.8));

// Print
initialEstimate.print("Initial Estimate:\n");

// Optimize using Levenberg-Marquardt optimization. The optimizer
// accepts an optional set of configuration parameters, controlling
// things like convergence criteria, the type of linear system solver
// to use, and the amount of information displayed during optimization.
// Here we will use the default set of parameters. See the
// documentation for the full set of parameters.
LevenbergMarquardtOptimizer optimizer(graph, initialEstimate);
Values result = optimizer.optimize();
result.print("Final Result:\n");

// Calculate and print marginal covariances for all variables
Marginals marginals(graph, result);
print(marginals.marginalCovariance(x1), "x1 covariance");
print(marginals.marginalCovariance(x2), "x2 covariance");
print(marginals.marginalCovariance(x3), "x3 covariance");
print(marginals.marginalCovariance(l1), "l1 covariance");
print(marginals.marginalCovariance(l2), "l2 covariance");

return 0;
}
```



Supplementary Notes

Lecture 23: Simultaneous Localization and Mapping I

Lecturer: Luca Carlone

Scribes: -

Disclaimer: These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor(s).

This lecture focuses on Simultaneous Localization and Mapping (SLAM) and discusses:

- the typical structure of a SLAM system, including a SLAM front-end and a SLAM back-end,
- common formulations for the SLAM back-end, including landmark-based SLAM and pose graph optimization,
- and the sparsity structure of the resulting iterative solvers, which enables fast optimization.

The presentation is based on the survey [1], while here we mostly focus on different back-end formulations. The interested reader is referred to [1] for a more comprehensive overview of the state of the art and open problems. Another excellent reference on the topic is [3], which provides a broader review of the mathematical aspects.

23.1 SLAM Front-end and Back-end

Simultaneous Localization and Mapping (SLAM) is the problem of estimating the pose (or trajectory) of the robot and a representation of the environment using sensor data and without prior knowledge of the environment. In this sense, SLAM is an estimation problem where one wants to compute a variable of interest \mathbf{x} (describing the state of the robot and a parametrization of the environment) using sensor data.

In previous lectures, we have introduced Maximum Likelihood estimation (MLE) and Maximum a Posteriori (MAP) estimation as two general estimation frameworks that allow framing an estimation problem in terms of mathematical optimization. The main ingredient of MLE and MAP estimation consists in the definition of measurement models that describe the likelihood of taking a measurement \mathbf{y}_i from a state \mathbf{x} :

$$\mathbf{y}_i = h_i(\mathbf{x}, \boldsymbol{\epsilon}_i) \quad (23.1)$$

where $h_i(\cdot)$ is a known function (the “measurement model”) and $\boldsymbol{\epsilon}_i$ is a random variable describing measurement noise.¹

Now the main issue is that the data produced by key sensors used for robot navigation is difficult to directly describe as in (23.1). How can we describe the intensity of every pixel in an image with a function as in (23.1)? how can we describe every point in a dense point cloud produced by a lidar with a measurement model? Even if the design of a measurement model were viable, it would lead to extremely complex estimation problems. For instance, in order to describe the intensity of every pixel in the image as in (23.1), one would have to include as part of the state \mathbf{x} the position and intensity of the light source, a parametrization of the geometry of the portion of the environment seen by the camera (e.g., a 3D mesh), the material properties that impact

¹In MAP one also needs to define suitable priors on \mathbf{x} , which can also be written in a form akin to (23.1) and often have simple expressions, so we will focus on how to build measurement models in the rest of these notes.

the reflection of light, among other aspects. This would induce several issues: (i) it would lead to a very high-dimensional \mathbf{x} (again, we would need to include all the variables of interest in \mathbf{x}), (ii) it would lead to a very large set of measurements (e.g., one measurement per pixel), which would prevent the optimization problem to be solved efficiently; furthermore, many of the quantities in \mathbf{x} may not even be observable (e.g., can we uniquely identify material properties from images?), which implies that the resulting problem would have an infinite number of solutions.

To alleviate this problem, the common approach in computer vision and robotics (and in many other fields) is to extract “intermediate representations” that are easier to describe mathematically in the form (23.1). This creates a natural split in common SLAM architectures (Fig. 23.1): the sensor data is first passed to a set of algorithms (the “SLAM front-end”) in charge of extracting such intermediate representations, and then the resulting intermediate representations are framed as in (23.1) and passed to a MLE/MAP estimator (the “SLAM back-end”). While the notion of “intermediate representations” is still vague, the next section shows that several computer vision techniques we have seen in VNAV can be already understood as a SLAM front-end. For instance, point features can be understood as intermediate representations of images that can be more easily described mathematically.

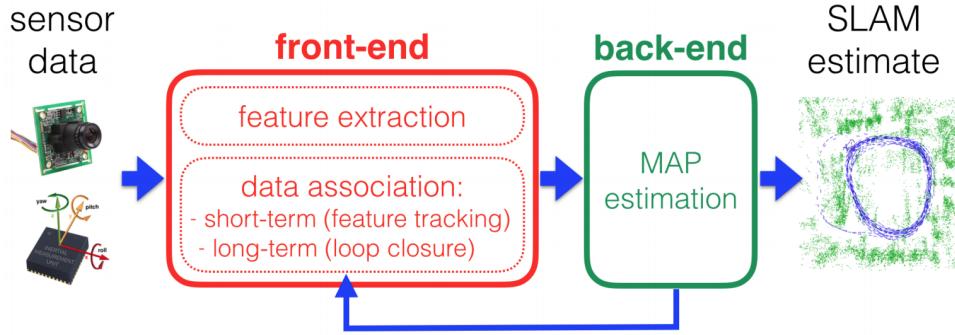


Figure 23.1: Front-end and back-end in a typical SLAM system.

23.2 Examples of SLAM Front-ends

We have already seen many algorithms that abstract the sensor data into intermediate representations that are easier to describe mathematically:

- 2-view geometry and visual odometry abstract a pair (or a subset) of images into motion estimates. In other words, they convert the raw sensor data (the images) into an intermediate representation, the relative pose $\bar{\mathbf{T}}_{t+1}^t$ of the camera between consecutive images.
- 3D registration abstracts a pair of point clouds into a relative pose between the corresponding observation poses. In other words, they convert the raw sensor data (two consecutive point clouds) into an intermediate representation, e.g., the relative pose $\bar{\mathbf{T}}_{t+1}^t$ of the lidar between consecutive scans.
- feature detection and tracking/matching: abstracts image pixels into a sparse set of pixels corresponding to the projection of 3D points \mathbf{l}_k^w onto the camera image.²
- place recognition identifies that two images, say i and j , are observing the same scene, from which (using 2-view geometry) we can compute the relative pose of the corresponding cameras $\bar{\mathbf{T}}_j^i$.

²In the symbol \mathbf{l}_k^w , k is the index of the point, w denotes that the corresponding vector is expressed in the world frame.

These are only few examples of typical SLAM front-ends. In general, the choice of front-end depends on the sensor (e.g., we may need to use a different front-end for feature detection in a lidar scan, compared to a camera), and on the intermediate representation we desire as output of the front-end. For instance, while we talked about point feature detection, research efforts have been devoted to the development of front-end extracting other representations, including lines, planes, object poses, and other geometric primitives.

23.3 SLAM Back-end Formulations

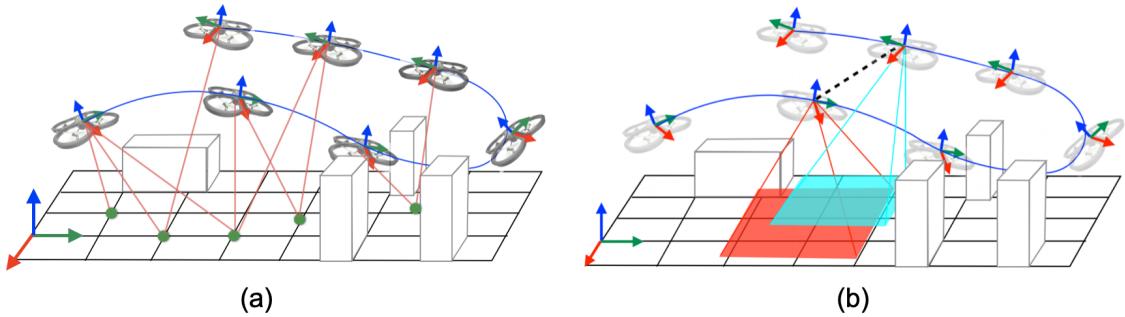


Figure 23.2: (a) Landmark-based SLAM. (b) Pose Graph Optimization (PGO).

So far, we have seen that the SLAM front-end produces an intermediate representation of the sensor data that can be expressed mathematically in a form akin to (23.1). Then the back-end computes an estimate given these intermediate representations using MLE or MAP estimation. In the following we review common choices of intermediate representations that lead to popular back-end formulations, summarized in Fig. 23.2.

23.3.1 Landmark-based SLAM

In landmark-based SLAM, the front-end produces odometry information (e.g., from visual odometry, visual-inertial odometry, or even wheel odometry for wheeled robots) and also produces detections of external *landmarks*. The most common case is the one in which landmarks are 3D points, but, as mentioned above, landmarks can be also more complex geometric primitives (e.g., lines, planes, objects). These intermediate representations can be easily described as in (23.1). The odometry $\bar{\mathbf{T}}_{t+1}^t$ describes the change in the pose between time t and $t + 1$ and can be described using the following measurement model:

$$\bar{\mathbf{T}}_{t+1}^t = (\mathbf{T}_t^w)^{-1} (\mathbf{T}_{t+1}^w) \mathbf{T}_{\epsilon_o} \quad (23.2)$$

where \mathbf{T}_{ϵ_o} is a random variable describing the odometry measurement noise with covariance Σ_o . Similarly, the measurements of a point landmark \mathbf{l}_k^w at time t can be written as:

$$\bar{\mathbf{y}}_{k,t} = h_i(\mathbf{T}_t^w, \mathbf{l}_k^w) + \epsilon_l \quad (23.3)$$

where $\bar{\mathbf{y}}_{k,t}$ is the measurement and ϵ_l is the measurement noise with covariance Σ_l . For a monocular camera, $h_i(\cdot)$ would simply be the projection of the 3D point \mathbf{l}_k^w onto the camera at pose \mathbf{T}_t^w . From now on, we will assume that all quantities (poses, points) are expressed in the world frame, and we will omit the superscript “w” to simplify the notation.

As we have seen when discussing MLE/MAP estimation, and assuming the noise is Gaussian (or wrapped Gaussian for the pose \mathbf{T}_{ϵ_o}), the measurement models (23.8)-(23.3) lead to the following estimator:

$$\min_{\mathbf{T}_t, t=1,\dots,n \atop \mathbf{l}_k, k=1,\dots,K} \sum_{t=1,\dots,n-1} \|\log([\mathbf{T}_t^{-1} \mathbf{T}_{t+1}]^{-1} \bar{\mathbf{T}}_{t+1}^t)^\vee\|_{\Sigma_o}^2 + \sum_{k=1,\dots,K} \sum_{t \in \mathcal{S}_k} \|\bar{\mathbf{y}}_{k,t} - h_i(\mathbf{T}_t, \mathbf{l}_k)\|_{\Sigma_l}^2 \quad (23.4)$$

where \mathcal{S}_k is the subset of poses that observe landmark k , and for a vector \mathbf{v} , we have defined the weighted squared norm $\|\mathbf{v}\|_{\Sigma}^2 = \mathbf{v}^T \Sigma^{-1} \mathbf{v}$. Using the shorthand notation $\mathbf{T}_1 \boxminus \mathbf{T}_2 \triangleq \log(\mathbf{T}_1^{-1} \mathbf{T}_2)^\vee$ (which computes a tangent-space representation of the relative pose between two poses \mathbf{T}_1 and \mathbf{T}_2), the previous expression can be written more succinctly as:

$$\min_{\mathbf{T}_t, t=1,\dots,n \atop \mathbf{l}_k, k=1,\dots,K} \sum_{t=1,\dots,n-1} \|(\mathbf{T}_t^{-1} \mathbf{T}_{t+1}) \boxminus \bar{\mathbf{T}}_{t+1}^t\|_{\Sigma_o}^2 + \sum_{k=1,\dots,K} \sum_{t \in \mathcal{S}_k} \|\bar{\mathbf{y}}_{k,t} - h_i(\mathbf{T}_t, \mathbf{l}_k)\|_{\Sigma_l}^2 \quad (23.5)$$

If the rotation noise is assumed to follow a Langevin distribution, the estimator becomes:

$$\min_{\mathbf{T}_t, t=1,\dots,n \atop \mathbf{l}_k, k=1,\dots,K} \sum_{t=1,\dots,n-1} \|\mathbf{T}_t^{-1} \mathbf{T}_{t+1} - \bar{\mathbf{T}}_{t+1}^t\|_{\Sigma_o}^2 + \sum_{k=1,\dots,K} \sum_{t \in \mathcal{S}_k} \|\bar{\mathbf{y}}_{k,t} - h_i(\mathbf{T}_t, \mathbf{l}_k)\|_{\Sigma_l}^2 \quad (23.6)$$

where for a pose $\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ and the covariance $\Sigma = \begin{bmatrix} \sigma_R^2 \mathbf{I}_3 & \\ & \Sigma_t \end{bmatrix}$, we have defined the weighted squared matrix norm $\|\mathbf{M}\|_{\Sigma}^2 = \mathbf{t}^T \Sigma_t^{-1} \mathbf{t} + \frac{1}{\sigma_R^2} \|\mathbf{R}\|_F^2$. Note that in the case of Langevin noise, the rotation noise is assumed *isotropic*, meaning that the covariance is a multiple of the identity matrix, $\sigma_R^2 \mathbf{I}_3$.

So far we have kept complete generality with respect to the measurement model $h_i(\cdot)$. For instance, the estimators (23.5)-(23.6), with a suitable choice of $h_i(\cdot)$, can represent a visual-SLAM problem (where $h_i(\cdot)$ is a projection model) or a range-SLAM problem using radio beacons (where $h_i(\cdot)$ measures the range between the robot and radio beacons at locations \mathbf{l}_k in the environment).

SLAM vs. Structure from Motion. When the measurements are produced by a monocular camera, we obtain the following visual-SLAM back-end:

$$\min_{\mathbf{T}_t, t=1,\dots,n \atop \mathbf{l}_k, k=1,\dots,K} \sum_{t=1,\dots,n-1} \|\mathbf{T}_t^{-1} \mathbf{T}_{t+1} - \bar{\mathbf{T}}_{t+1}^t\|_{\Sigma_o}^2 + \sum_{k=1,\dots,K} \sum_{t \in \mathcal{S}_k} \|\bar{\mathbf{y}}_{k,t} - \pi(\mathbf{T}_t, \mathbf{l}_k)\|_{\Sigma_l}^2 \quad (23.7)$$

where π is the perspective projection function. Comparing this expression with the one we discussed in the visual odometry lecture, it becomes clear that the visual-SLAM problem is very related to *bundle adjustment* in Structure from Motion (SfM). The key differences are as follows:

- the SLAM problem typically has the odometry terms, since the data is collecting by a single robot moving in the environment (in SfM, the poses correspond to different cameras picturing the same scene, e.g., multiple tourists taking a picture of a monument);
- in SLAM, the data is presented incrementally: at each time stamp the robot collects new data and has to solve the problem again, while in SfM the problem is solved only one time on the entire batch of data;
- in SLAM, the problem has to be solved in close-to-real-time, while in SfM the reconstruction may be allowed to require a few hours.
- in SLAM, the images are collected by the robot over time, which makes feature tracking possible/easier. On the other hand, feature matching is the approach of choice when finding correspondences in SfM.

23.3.2 Pose Graph Optimization

Pose Graph Optimization (PGO), also called pose-based SLAM and pose-graph SLAM, assumes a different intermediate representation from the front-end. In particular, in PGO, the front-end produces odometry information (e.g., same as in the previous case), but then uses place recognition to compute relative pose measurements between the current pose and a previous observation pose; such a relative pose is called a “loop closure” and can be established when two non-consecutive poses observe the same place.

The odometry $\bar{\mathbf{T}}_{t+1}^t$ describes the change in the pose between time t and $t + 1$ and can be described as in (23.8). The loop closure measurements can be also described as relative pose measurements between two poses \mathbf{T}_i and \mathbf{T}_j

$$\bar{\mathbf{T}}_j^i = \mathbf{T}_i^{-1} \mathbf{T}_j \mathbf{T}_{\epsilon_{lc}} \quad (23.8)$$

Assuming the noise follows a wrapped Gaussian distribution, the PGO estimator becomes:

$$\min_{\mathbf{T}_t, t=1, \dots, n} \sum_{t=1, \dots, n-1} \|(\mathbf{T}_t^{-1} \mathbf{T}_{t+1}) \boxminus \bar{\mathbf{T}}_{t+1}^t\|_{\Sigma_o}^2 + \sum_{(i,j) \in \mathcal{E}_{lc}} \|(\mathbf{T}_i^{-1} \mathbf{T}_j) \boxminus \bar{\mathbf{T}}_j^i\|_{\Sigma_{lc}}^2 \quad (23.9)$$

where \mathcal{E}_{lc} is the set of pose pairs (i, j) for which we have a loop closure between pose i and j .

Since the two terms in (23.9) have the same expression, we can write them together as:

$$\min_{\mathbf{T}_t, t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}} \|(\mathbf{T}_i^{-1} \mathbf{T}_j) \boxminus \bar{\mathbf{T}}_j^i\|_{\Sigma_{ij}}^2 \quad (23.10)$$

where $\mathcal{E} = (1, 2) \cup (2, 3) \cup \dots \cup (n - 1, n) \cup \mathcal{E}_{lc}$ and collects pairs of poses for which we have odometry measurements or loop closures. This is the cost implemented when using `betweenFactor <Pose3>` in GTSAM.

Assuming that the rotation noise follows a Langevin distribution, it is possible to reformulate the PGO problem using the chordal distance [4]:

$$\min_{\mathbf{T}_t \in \text{SE}(3), t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}} \|\mathbf{T}_i^{-1} \mathbf{T}_j - \bar{\mathbf{T}}_j^i\|_{\Sigma_{ij}}^2 \quad (23.11)$$

which is often more convenient due to the quadratic nature of the cost function. This is the cost implemented when using `betweenChordalFactor` in GTSAM.

While we presented PGO as the result of a robot using an odometry-and-place-recognition front-end, the PGO model also applies to other SLAM problems.

Multi-robot SLAM as PGO. Assume that we have two robots α and β performing collaborative SLAM. The robots exchange images to establish loop closure between the corresponding images. Mathematically, the problem can be still formulated as a variant of PGO (in this case, using the chordal distance):

$$\min_{\mathbf{T}_{\alpha,t}, \mathbf{T}_{\beta,t}, t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}_{\alpha}} \|\mathbf{T}_{\alpha,i}^{-1} \mathbf{T}_{\alpha,j} - \bar{\mathbf{T}}_{\alpha,j}^{\alpha,i}\|_{\Sigma_{ij}}^2 + \sum_{(i,j) \in \mathcal{E}_{\beta}} \|\mathbf{T}_{\beta,i}^{-1} \mathbf{T}_{\beta,j} - \bar{\mathbf{T}}_{\beta,j}^{\beta,i}\|_{\Sigma_{ij}}^2 + \sum_{(i,j) \in \mathcal{E}_{\alpha\beta}} \|\mathbf{T}_{\alpha,i}^{-1} \mathbf{T}_{\beta,j} - \bar{\mathbf{T}}_{\beta,j}^{\alpha,i}\|_{\Sigma_{ij}}^2 \quad (23.12)$$

where $\mathcal{E}_{\alpha\beta}$ is the set of loop closures (sometimes referred to as inter-robot loop closures [2]) between the robots. Note that the structure of problem (23.12) is essentially identical to (23.11) but the problem simultaneously estimates the poses of both robots. The formulation can be easily extended to an arbitrary number of robots.

Object-based SLAM as PGO. We can also use PGO as a back-end for the case where the SLAM front-end produces odometry measurements as well as measures the relative pose of external objects. In that case, PGO can be used to simultaneously estimate the pose of the robot and the poses of the external objects.

23.3.3 Some Remarks

Robustness and Outliers. One important observation is that to derive our MLE/MAP estimator we have assumed Gaussian noise (of Langevin noise, which can be understood as the equivalent of a Gaussian distribution on the group of rotations). However, the front-end can sometime make mistakes (e.g., feature tracking/matching might fail to detect the correct pixels in the images, or place recognition may match images of different places). These incorrect measurements produced by the front-end, often referred to as *outliers*, have large errors and violate the assumption of Gaussian noise, inducing large errors in the estimation. This observation stresses the importance of two aspects:

- it is important to vet the measurements produced by the front-end before passing them to the back-end. This motivates the use of RANSAC (or similar techniques) in the front-end that have the goal of removing outliers;
- it is important to consider more general back-ends having cost functions that are robust to the presence of outliers. The cost functions lead to the theory of M-estimation (where “M” stands for “Maximum Likelihood type” estimator), which we are going to discuss later in VNAV. See [5] for a discussion.

Priors. In this section we have only considered optimization problems where each term in the objective corresponds to a measurement (these are Maximum Likelihood estimators). In practice, it is customary to also add priors (e.g., a `priorFactor` in GTSAM), leading to MAP estimation. Such priors are used to constrain the problem and force it to have a unique solution. For instance, since relative poses are preserved when rigidly rotating a trajectory, all the optimization problems above have infinite solutions corresponding to arbitrary rigid transformations of the robot trajectory (and possibly of the landmarks). To avoid this ambiguity it is common to add a prior that fixes one of the poses (typically the first pose) to an arbitrary value (typically the identity pose), thus “anchoring” the trajectory. Note that this ambiguity does not arise when we have sensors measuring absolute quantities (e.g., a GPS).

23.4 Leveraging Sparsity to Speed-Up Computation

In this section we derive the expression of the Gauss-Newton (GN) update for a Pose Graph Optimization (PGO) problem. Then, we comment on the fact that the Normal equations arising from GN are sparse and have a precise sparsity pattern (and the same holds for landmark-based SLAM). This enables the use of fast sparse linear solvers that allow solving problems with thousands of variables in seconds.

23.4.1 Pose Graph Optimization: Normal Equations and Sparsity

Let us start from the PGO formulation we introduced in eq. (23.10) (where we make the Ξ explicit):

$$\min_{\mathbf{T}_t, t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}} \|\log([\mathbf{T}_i^{-1} \mathbf{T}_j]^{-1} \bar{\mathbf{T}}_j^i)^\vee\|_{\Sigma_{ij}}^2 \quad (23.13)$$

Let us now derive the expression of a Gauss-Newton (GN) update for eq. (23.13), assuming that we are given an initial guess $\hat{\mathbf{T}}_t, t = 1, \dots, n$ for the poses in the problem.

As we mentioned in the optimization lectures, we first reformulate the problem into one defined in the tangent space at $\hat{\mathbf{T}}_t, t = 1, \dots, n$ by introducing a retraction $\mathcal{R}(\cdot)$ that transforms a vector in the tangent

space to a 3D pose:

$$\min_{\delta_t, t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}} \|\log([(T_i \cdot \mathcal{R}(\delta_i))^{-1} (T_j \cdot \mathcal{R}(\delta_j))]^{-1} T_j^i)^\vee\|_{\Sigma_{ij}}^2 \quad (23.14)$$

Now we note that for a vector \mathbf{x} and a covariance \mathbf{P} , $\|\mathbf{x}\|_{\mathbf{P}}^2 = \mathbf{x}^\top \mathbf{P}^{-1} \mathbf{x} = (\mathbf{P}^{-\frac{1}{2}} \mathbf{x})^\top (\mathbf{P}^{-\frac{1}{2}} \mathbf{x}) = \|(\mathbf{P}^{-\frac{1}{2}} \mathbf{x})\|^2$, hence we can rewrite (23.15) as:

$$\min_{\delta_t, t=1, \dots, n} \sum_{(i,j) \in \mathcal{E}} \|\Sigma_{ij}^{-\frac{1}{2}} \log([(T_i \cdot \mathcal{R}(\delta_i))^{-1} (T_j \cdot \mathcal{R}(\delta_j))]^{-1} T_j^i)^\vee\|^2 = \min_{\delta_t, t=1, \dots, n} \|r_{ij}(\delta_i, \delta_j)\|^2 \quad (23.15)$$

where on the right-hand-side we simply noted that the term inside the squared norm is a function of δ_i and δ_j ; we denoted this function as $r_{ij} : \mathbb{R}^6 \times \mathbb{R}^6 \mapsto \mathbb{R}^6$, which is often called the (*whitened*) *residual error*.

Now we take a Taylor expansion of the terms inside the norm in (23.15):

$$\min_{\delta \in \mathbb{R}^{6n}} \sum_{(i,j) \in \mathcal{E}} \|\mathbf{J}_{ij} \delta - \mathbf{b}_{ij}\|^2 \quad (23.16)$$

where we defined:

- the vector δ , which includes all the unknown $\delta_t \in \mathbb{R}^6, t = 1, \dots, n$. Mathematically:

$$\delta = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_n \end{bmatrix} \in \mathbb{R}^{6n} \quad (23.17)$$

- the matrix \mathbf{J}_{ij} , which is the *Jacobian* of r_{ij} with respect to δ :

$$\mathbf{J}_{ij} = \left[\begin{array}{cccc} (\frac{\partial r_{ij}}{\partial \delta_1})^\top & (\frac{\partial r_{ij}}{\partial \delta_2})^\top & \dots & (\frac{\partial r_{ij}}{\partial \delta_n})^\top \end{array} \right] \in \mathbb{R}^{6 \times 6n} \quad (23.18)$$

Since r_{ij} only depends on δ_i and δ_j , the Jacobian \mathbf{J}_{ij} is sparse:

$$\mathbf{J}_{ij} = \left[\begin{array}{cccccc} \mathbf{0} & \dots & \mathbf{0} & (\frac{\partial r_{ij}}{\partial \delta_i})^\top & \mathbf{0} & \dots & \mathbf{0} & (\frac{\partial r_{ij}}{\partial \delta_j})^\top & \mathbf{0} \end{array} \right] \in \mathbb{R}^{6 \times 6n} \quad (23.19)$$

- the vector $\mathbf{b}_{ij} = -r_{ij}(\mathbf{0}, \mathbf{0})$ (note that $r_{ij}(\mathbf{0}, \mathbf{0})$ is the residual error evaluated at the linearization point in our Taylor expansion).

Recalling that the squared norm of a vector is the sum of the squared norms of its sub-vectors, we succinctly rewrite (23.16) as:

$$\min_{\delta \in \mathbb{R}^{6n}} \|\mathbf{J}\delta - \mathbf{b}\|^2 \quad (23.20)$$

where \mathbf{J} is a tall matrix stacking the Jacobians \mathbf{J}_{ij} by rows, and \mathbf{b} is a vector stacking all the \mathbf{b}_{ij} . As we have seen in previous lecture, the linear least squares problem (23.20) can be solved in closed form as:

$$\delta^* = (\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \mathbf{b} \quad (23.21)$$

where the matrix $\mathbf{J}^\top \mathbf{J}$ is the Hessian of the problem. In practice, the inverse is never computed, but one rather uses linear solvers to solve the linear system

$$(\mathbf{J}^\top \mathbf{J})\delta^* = \mathbf{J}^\top \mathbf{b} \quad (23.22)$$

Graph	Transpose Incidence Matrix	Laplacian Matrix
	<p style="text-align: center;">Transpose Incidence Matrix</p> $\bar{T} = \begin{bmatrix} T_0 & T_1 & T_2 & T_3 & T_4 \\ \bar{T}_{01} & -1 & 1 & 0 & 0 \\ \bar{T}_{12} & 0 & -1 & 1 & 0 \\ \bar{T}_{23} & 0 & 0 & -1 & 1 \\ \bar{T}_{34} & 0 & 0 & 0 & -1 \\ \bar{T}_{41} & 1 & 0 & 0 & -1 \end{bmatrix}$	<p style="text-align: center;">Laplacian Matrix</p> $T = \begin{bmatrix} T_0 & T_1 & T_2 & T_3 & T_4 \\ T_0 & 1 & -1 & 0 & 0 \\ T_1 & -1 & 3 & -1 & 0 \\ T_2 & 0 & -1 & 2 & -1 \\ T_3 & 0 & 0 & -1 & 2 \\ T_4 & 0 & 0 & 0 & 2 \end{bmatrix}$

Figure 23.3: Transpose Incidence matrix and Laplacian matrix of a toy graph. The transpose incidence matrix is defined as follows: each row of the transpose incidence matrix describes an edge from node i to node j and has 2 non-zero elements, a -1 in position i and a $+1$ in position j . The Laplacian matrix is defined as follows: the i -th diagonal term of the matrix contains the node degree (number of edges incident on i) and an off-diagonal entry in position (i, j) is equal to -1 if there is an edge between node i to node j , or is zero otherwise. Calling L the Laplacian matrix and A the transpose incidence matrix, it holds $L = A^T A$.

which is often referred to as the *Normal equations*. A key observation, which directly follows from the structure of the Jacobians in (23.19), is that the matrix J (and hence the Hessian $J^T J$) is very sparse (i.e., it has many entries equal to zero), which allows the use of very fast and scalable *sparse linear solvers*.

Sparsity of Jacobian and Hessian. Let us define a graph \mathcal{G} , where the nodes in \mathcal{G} are the (indices of) the unknown poses in our pose graph, and the edges describe measurements between a pair of poses. For instance an edge (i, j) denotes that we have a measurement of the pose between pose i and pose j (either via odometry or loop closure). Now, since the Jacobian J is stacking the Jacobian J_{ij} of each measurement by rows and the only nonzero blocks in J_{ij} correspond to the variables involved in measurement (i, j) , it is easy to see that the blocks of the Jacobian matrix J have the same sparsity pattern of the *transpose incidence matrix* of the graph \mathcal{G} (Fig. 23.3). Similarly, the blocks of the Hessian matrix can be seen to have the same sparsity pattern of the *Laplacian matrix* of \mathcal{G} (Fig. 23.3).

23.4.2 Landmark-based SLAM: Sparsity

For landmark-based SLAM problems, the derivation of the Normal equations in the Gauss-Newton update is similar to the one in the previous section (but includes landmarks which already live in Euclidean space), and again leads to sparse linear system. Fig. 23.4 and Fig. 23.5 show the sparsity patterns of the corresponding Jacobian and Hessian matrices in two toy problems. Fig. 23.4 considers a bundle adjustment problem with 3 cameras and 2 landmarks. Fig. 23.5 considers a similar visual SLAM problem where one is also given odometry measurements between consecutive camera poses.

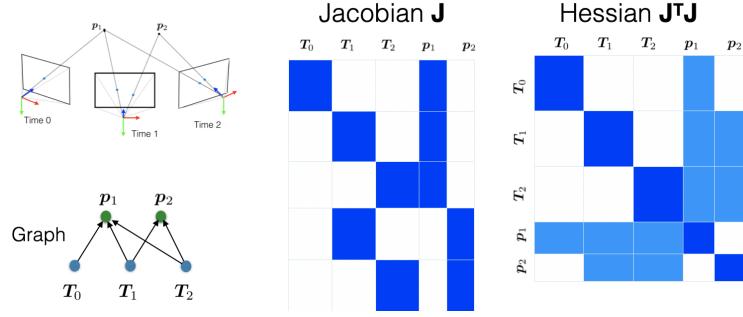


Figure 23.4: Bundle adjustment problem with 3 cameras and 2 landmarks. Sparsity patterns of the Jacobian and Hessian matrices.

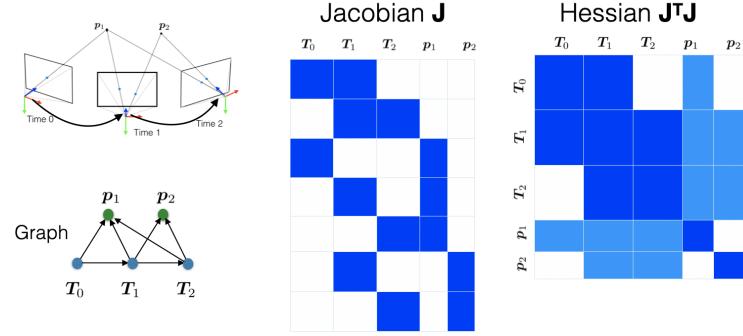


Figure 23.5: Visual SLAM problem with 3 cameras and 2 landmarks (with odometry). Sparsity patterns of the Jacobian and Hessian matrices.

References

- [1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J.J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robotics*, 32(6):1309–1332, 2016. arxiv preprint: 1606.05830, ([pdf](#)).
- [2] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H.I. Christensen, and F. Dellaert. Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models, accepted. *Intl. J. of Robotics Research*, 2017. arxiv preprint: 1702.03435, ([pdf](#)) ([web](#)) ([code](#)) ([code](#)) ([code](#)) ([video](#)) ([video](#)) ([video](#)).
- [3] F. Dellaert and M. Kaess. Factor graphs for robot perception. *Foundations and Trends in Robotics*, 6(1-2):1–139, 2017.
- [4] D.M. Rosen, L. Carlone, A.S. Bandeira, and J.J. Leonard. SE-Sync: A certifiably correct algorithm for synchronization over the Special Euclidean group. In *Intl. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, San Francisco, CA, December 2016. extended arxiv preprint: 1611.00128, ([pdf](#)) ([pdf](#)) ([code](#)).
- [5] H. Yang, P. Antonante, V. Tzoumas, and L. Carlone. Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):1127–1134, 2020. arXiv preprint arXiv:1909.08605 (with supplemental material), ([pdf](#)), .

References

- **Slides:** MIT's Visual Navigation course, Luca Carlone
(<https://vnav.mit.edu/>)
- **Book:** Factor Graphs for Robot Perception, Frank Dellaert
(<https://www.cs.cmu.edu/~kaess/pub/Dellaert17fnt.pdf>)
- **Paper:** M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM : Incremental smoothing and mapping," IEEE Transactions on Robotics, 2008
(<https://www.cs.cmu.edu/~kaess/pub/Kaess08tro.pdf>)
- **Library:** GTSAM: Georgia Tech Smoothing and Mapping Library
(<https://github.com/borglab/gtsam>)
- **Slides:** A brief introduction to Factor Graphs, José Luis Blanco Claraco
(https://ingmec.ual.es/~jlblanco/papers/2020-introduction-factor-graphs_JLBlanco.pdf)
- **Slides:** Factor graphs and nonlinear least squares problems on manifolds, Pranav Ganti
(http://wavelab.uwaterloo.ca/slam/2017-SLAM/Lecture6-factor_graphs_and_nlls_on_manifolds/Factor%20Graphs%20and%20NLLS%20on%20Manifolds.pdf)