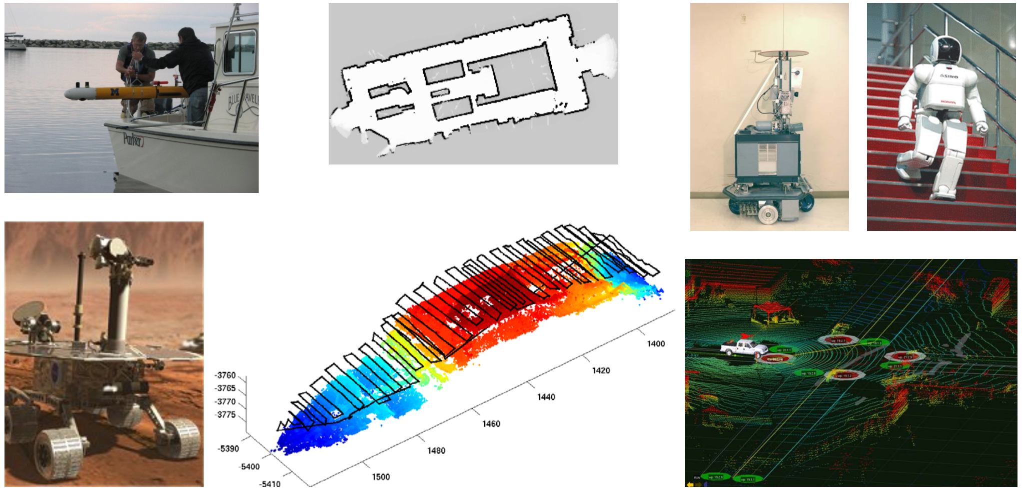
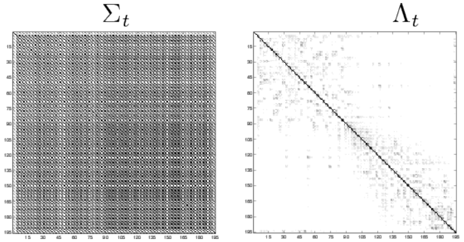


L15. Pose-Graph SLAM

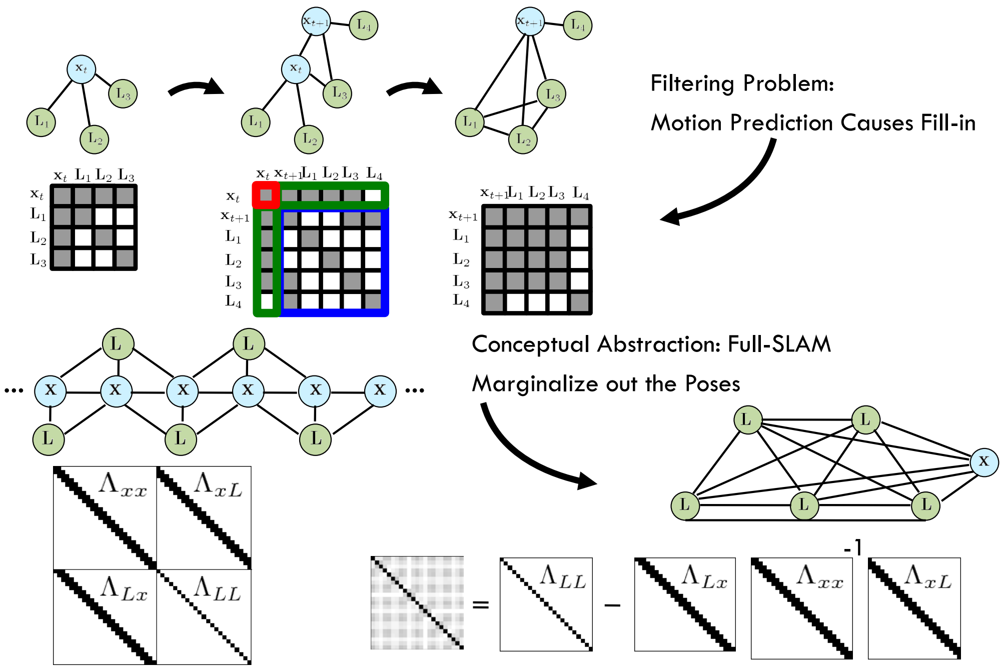
NA568 Mobile Robotics: Methods & Algorithms

Today's Topic



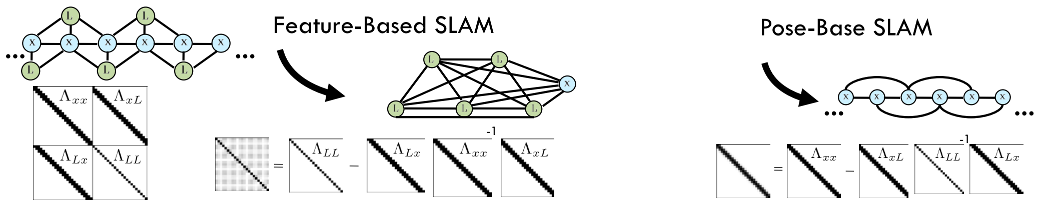
- Nonlinear Least Squares
- Pose-Graph SLAM
- Incremental Smoothing and Mapping

Feature-Based SLAM



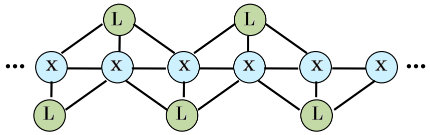
Feature-Based SLAM

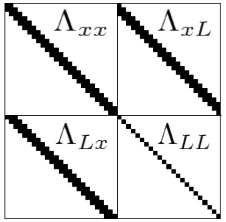
- In feature-based SLAM, the information matrix fills in unless we enforce approximations to make it sparse
  - This is because we are continually marginalizing out the robot trajectory from the state representation
- What if we were to marginalize out the landmarks instead?



Pose-Graph SLAM

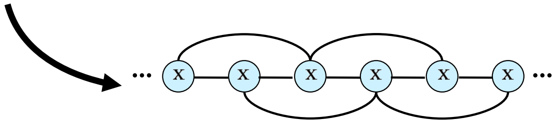
- Feature-based SLAM requires approximations to enforce sparsity.
- Furthermore, this approximation is non-trivial.

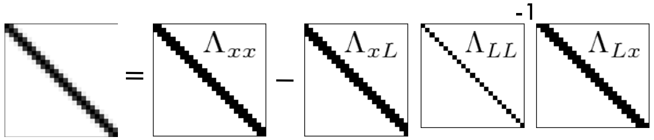




Conceptual Abstraction: Full-SLAM

Marginalize out the Landmarks





Key idea: marginalizing out the landmarks preserves locality

Three Main SLAM Paradigms

Kalman filter	Particle filter	Graph-based

least squares approach to SLAM

Courtesy: C. Stachniss

Least Squares in General

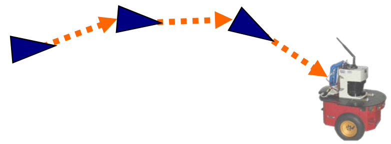
- Approach for computing a solution for an **overdetermined system**
- "More equations than unknowns"
- Minimizes the **sum of the squared errors** in the equations
- Standard approach to a large set of problems

Today: Application to SLAM

Courtesy: C. Stachniss

Graph-Based SLAM

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

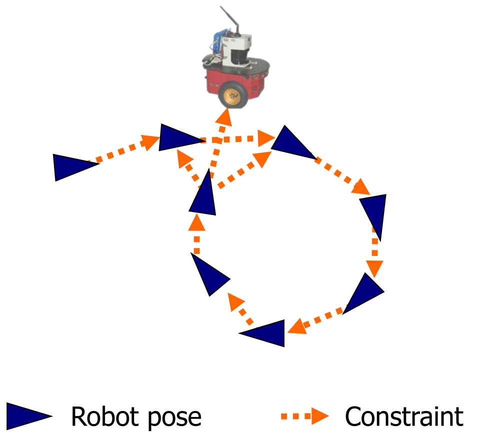


Robot pose Constraint

Courtesy: C. Stachniss

Graph-Based SLAM

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



Courtesy: C. Stachniss

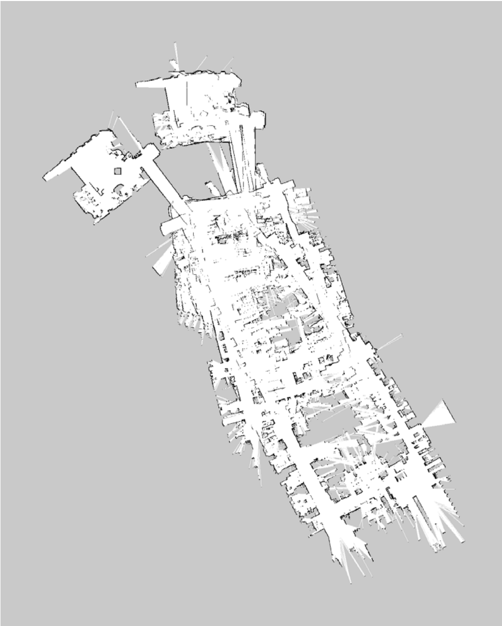
Idea of Graph-Based SLAM

- 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

Courtesy: C. Stachniss

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

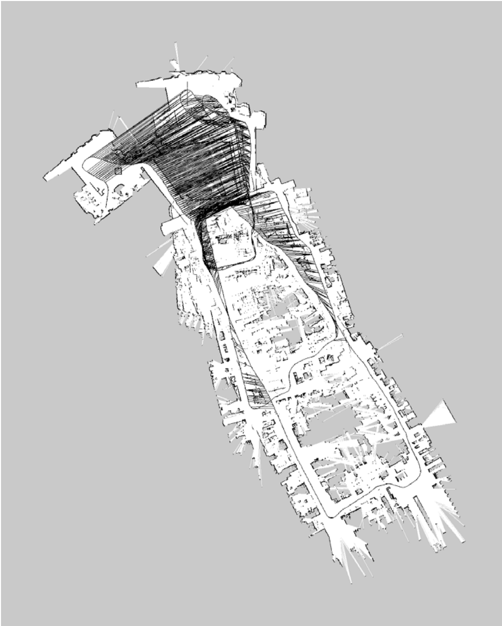


KUKA Halle 22, courtesy of P. Pfaff

Courtesy: C. Stachniss

Graph-Based SLAM in a Nutshell

- 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

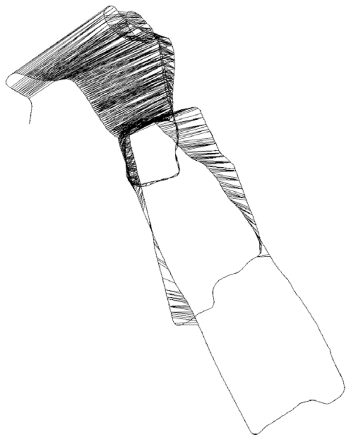


KUKA Halle 22, courtesy of P. Pfaff

Courtesy: C. Stachniss

Graph-Based SLAM in a Nutshell

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

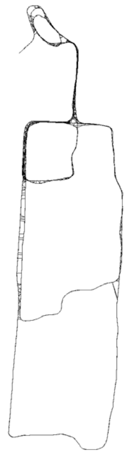


Courtesy: C. Stachniss

Graph-Based SLAM in a Nutshell

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

  … like this

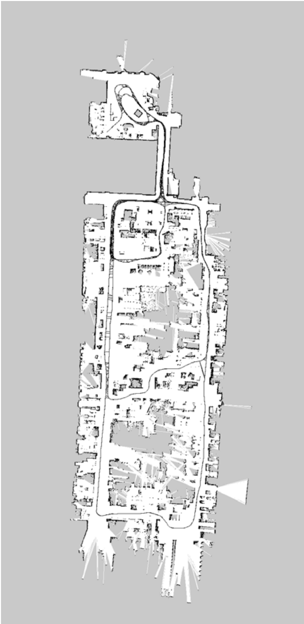


Courtesy: C. Stachniss

Graph-Based SLAM in a Nutshell

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

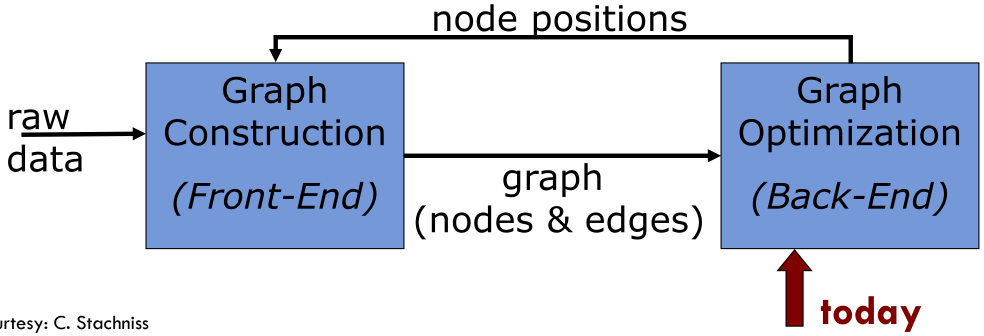
  … like this
- Then, we can render a map based on the known poses



Courtesy: C. Stachniss

The Overall SLAM System

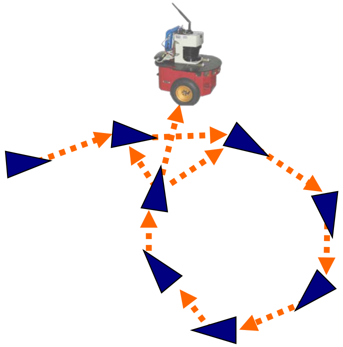
- Interplay of front-end and back-end
- Map helps to determine constraints by reducing the search space
- Topic today: optimization



Courtesy: C. Stachniss

The Graph

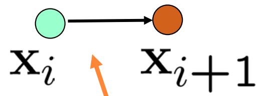
- It consists of n nodes $\mathbf{x} = \mathbf{x}_{1:n}$
- Each $\mathbf{x}_i$ is a 2D or 3D transformation (the pose of the robot at time t_i)
- A constraint/edge exists between the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ if…



Courtesy: C. Stachniss

Create an Edge If… (1)

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

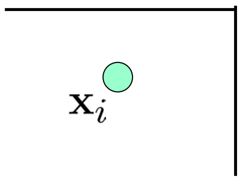


The edge represents the **odometry** measurement

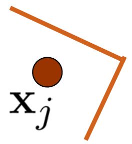
Courtesy: C. Stachniss

Create an Edge If… (2)

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



Measurement from $\mathbf{x}_i$

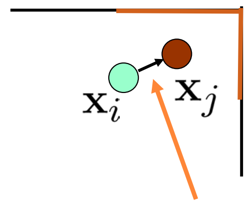


Measurement from $\mathbf{x}_j$

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 $\mathbf{x}_j$ seen from $\mathbf{x}_i$ based on the **observation**

Courtesy: C. Stachniss

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

Courtesy: C. Stachniss

Simultaneous Localization and Mapping

Given a **single camera** feed,
estimate the 3D **position of the camera** and
the 3D **positions of all landmark** points in the world

Courtesy: M. Kaess

Visual SLAM: Why Filter?

Image and Vision Computing 30 (2012) 65–77

Contents lists available at SciVerse ScienceDirect

Image and Vision Computing

journal homepage: www.elsevier.com/locate/imavis

Editors Choice Article

Visual SLAM: Why filter?

Hauke Strasdat [a,*], J.M.M. Montiel [b], Andrew J. Davison [a]

[a] Department of Computing, Imperial College London, UK
[b] Instituto de Investigacion en Ingeniera de Aragon (I3A), Universidad de Zaragoza, Spain

ARTICLE INFO

Article history:
Received 2 August 2011
Received in revised form 13 December 2011
Accepted 17 February 2012

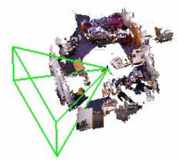
Keywords:
SLAM
Structure from motion
Bundle adjustment
EKF
Information filter
Monocular vision
Stereo vision

ABSTRACT

While the most accurate solution to off-line structure from motion (SFM) problems is undoubtedly to extract as much correspondence information as possible and perform batch optimisation, sequential methods suitable for live video streams must approximate this to fit within fixed computational bounds. Two quite different approaches to real-time SFM – also called visual SLAM (simultaneous localisation and mapping) – have proven successful, but they sparsify the problem in different ways. Filtering methods marginalise out past poses and summarise the information gained over time with a probability distribution. Keyframe methods retain the optimisation approach of global bundle adjustment, but computationally must select only a small number of past frames to process.
In this paper we perform a rigorous analysis of the relative advantages of filtering and sparse bundle adjustment for sequential visual SLAM. In a series of Monte Carlo experiments we investigate the accuracy and cost of visual SLAM. We measure accuracy in terms of entropy reduction as well as root mean square error (RMSE), and analyse the efficiency of bundle adjustment versus filtering using combined cost/accuracy measures. In our analysis, we consider both SLAM using a stereo rig and monocular SLAM as well as various different scenes and motion patterns. For all these scenarios, we conclude that keyframe bundle adjustment outperforms filtering, since it gives the most accuracy per unit of computing time.

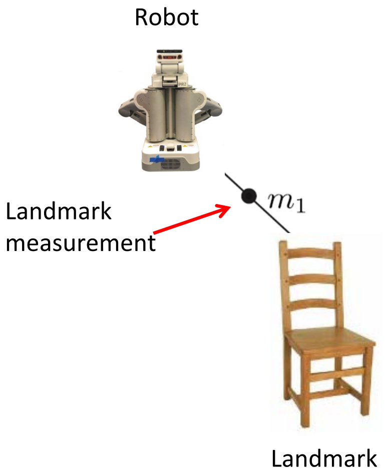
Courtesy: M. Kaess

Visual SLAM



Courtesy: M. Kaess

The SLAM Problem (t=0)

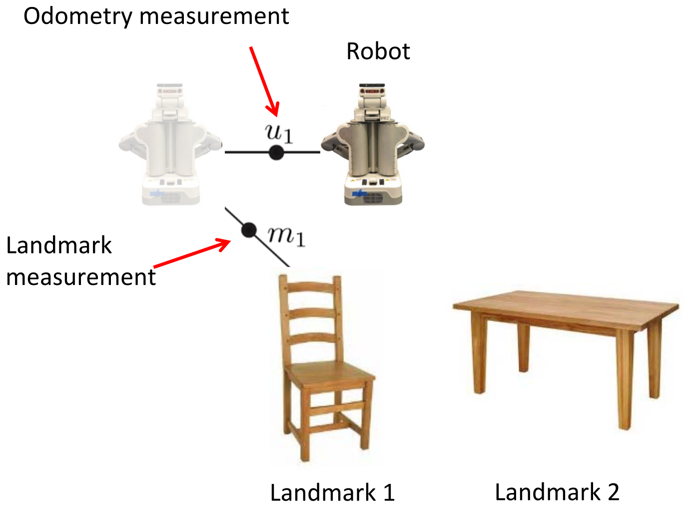


Onboard sensors:

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

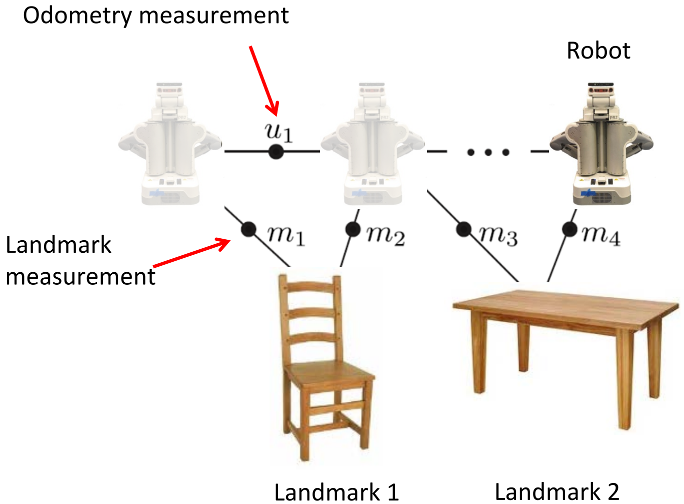
Courtesy: M. Kaess

The SLAM Problem (t=1)



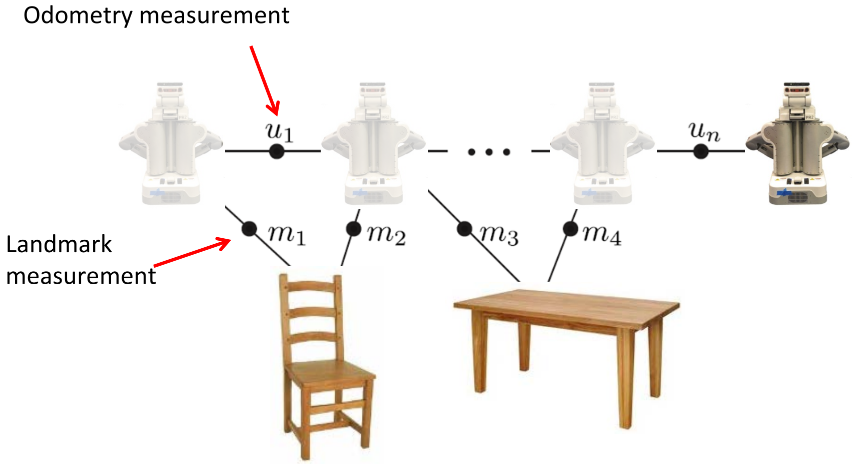
Courtesy: M. Kaess

The SLAM Problem (t=n-1)



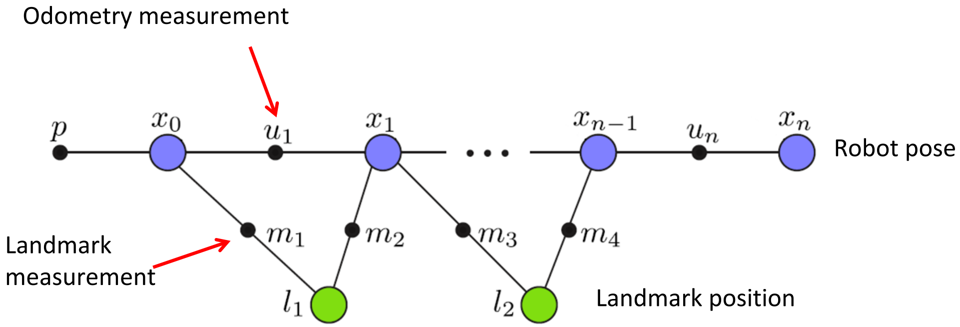
Courtesy: M. Kaess

The SLAM Problem (t=n)



Courtesy: M. Kaess

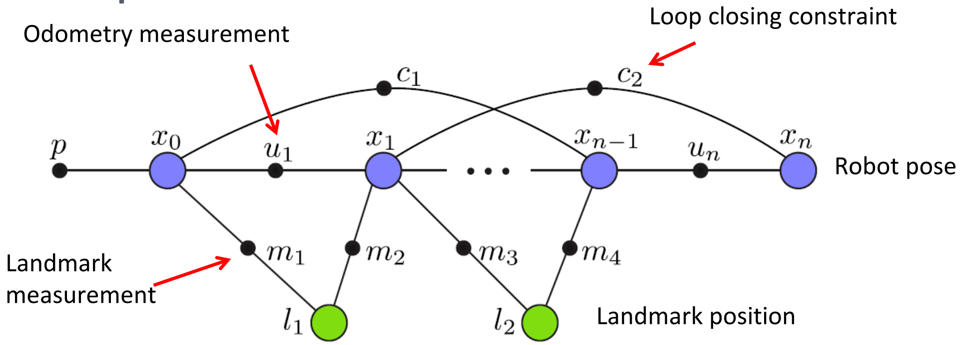
Factor Graph Representation



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

Courtesy: M. Kaess

Factor Graph Representation: Pose Graph

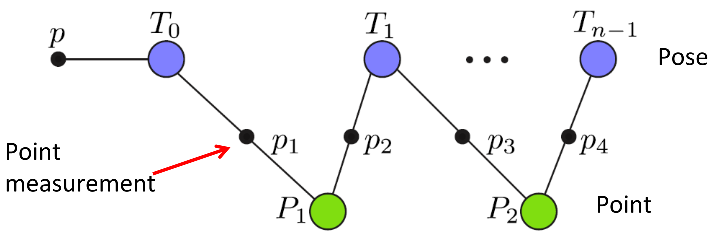


Bipartite graph with ***variable nodes*** and ***factor nodes***

Courtesy: M. Kaess

[Dellaert and Kaess, IJRR 06]

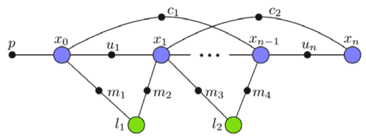
Factor Graph Representation: Bundle Adjust.



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

Courtesy: M. Kaess

Nonlinear Least-Squares



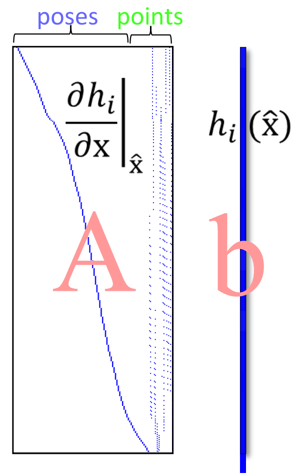
Gaussian noise

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

Repeatedly solve linearized system (GN)

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

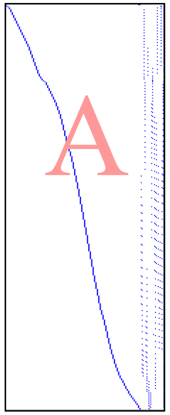
$$A = \begin{bmatrix} F_{11} & & G_{11} & & \\ F_{12} & & & G_{12} & \\ F_{13} & & & & G_{13} \\ & F_{21} & G_{21} & & \\ & F_{22} & & G_{22} & \\ & F_{23} & & & G_{23} \end{bmatrix}, \quad x = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix}, \quad b = \begin{bmatrix}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



Courtesy: M. Kaess

Solving the Linear Least-Squares System

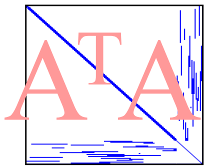
Solve: $\text{argmin}_x \|Ax - b\|^2$



Measurement Jacobian

Normal equations

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



Information matrix

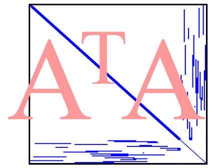
Courtesy: M. Kaess

Solving the Linear Least-Squares System

- Can we simply invert $A^T A$ to solve for x?

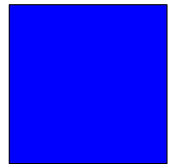
Normal equations

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



Information matrix

- Yes, but we shouldn't…
  The inverse of $A^T A$ is dense $\rightarrow O(n^3)$
- Can do much better by taking advantage of sparsity!

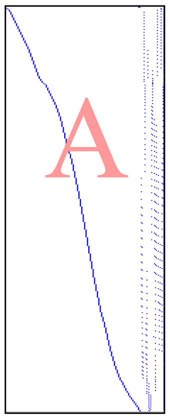


Courtesy: M. Kaess

Solving the Linear Least-Squares System

[Dellaert and Kaess, IJRR 06]

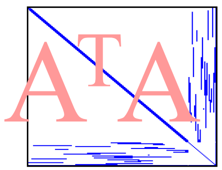
Solve: $\text{argmin}_x \|Ax - b\|^2$



Measurement Jacobian

Normal equations

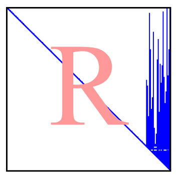
$$A^T A x = A^T b$$



Information matrix

Matrix factorization

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

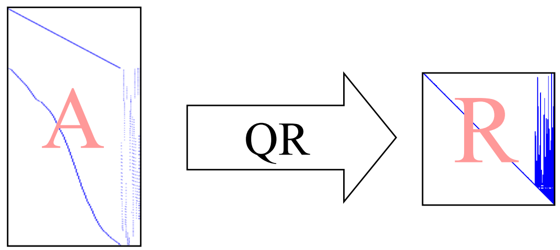


Square root information matrix

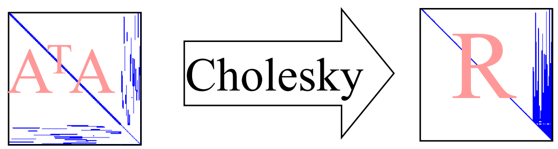
Courtesy: M. Kaess

Matrix – Square Root Factorization

- QR on A: Numerically More Stable



- Cholesky on $A^T A$: Faster

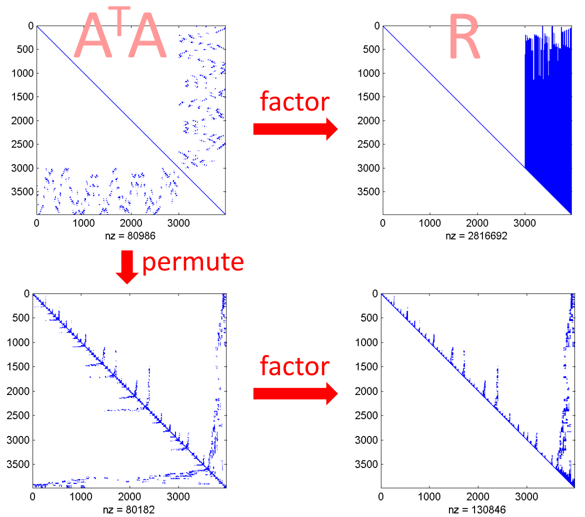


Courtesy: M. Kaess

[Dellaert and Kaess, IJRR 06]

Retaining Sparsity: Variable Ordering

Fill-in depends on elimination order:



Default ordering (poses, landmarks)

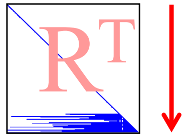
Ordering based on COLAMD heuristic [Davis04] (best order: NP hard)

Courtesy: M. Kaess

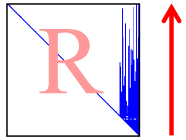
Solving by Backsubstitution

After factorization: $R^T R\,\boldsymbol{x} = A^T\boldsymbol{b}$

- Forward substitution
  $R^T\boldsymbol{y} = A^T\boldsymbol{b}$, solve for $\boldsymbol{y}$

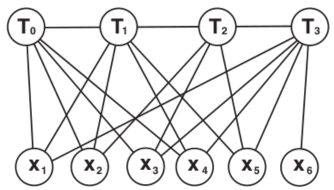


- Backsubstition
  $R\,\boldsymbol{x} = \boldsymbol{y}$, solve for $\boldsymbol{x}$



Courtesy: M. Kaess

Full Bundle Adjustment



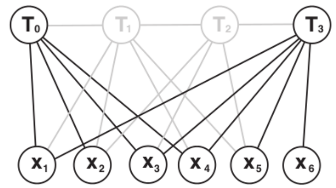
From Strasdat et al, 2011 IVC "Visual SLAM: Why filter?"

- 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

Courtesy: M. Kaess

Keyframe Bundle Adjustment

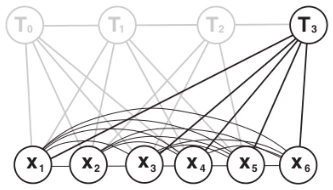


- Drop subset of poses to reduce density/complexity
- Only retain "keyframes" necessary for good map

- Complexity still grows with time, just slower

Courtesy: M. Kaess

Filter

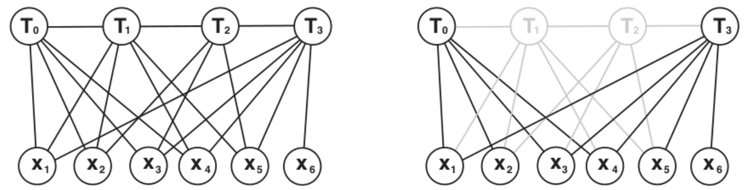


- Keyframe idea not applicable: map would fall apart
- Instead, marginalize out previous poses
  - Extended Kalman Filter (EKF)
- Problems when used for Visual SLAM:
  - All points become fully connected $\rightarrow$ expensive
  - Relinearization not possible $\rightarrow$ inconsistent

Courtesy: M. Kaess

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!

Courtesy: M. Kaess

Incremental Smoothing and Mapping (iSAM)

Courtesy: M. Kaess

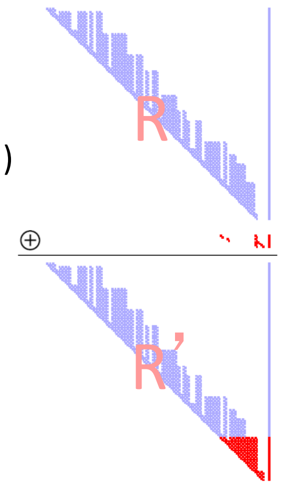
[Kaess et al., TRO 08]

iSAM

Solving a growing system:

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

New measurements ->



Key idea:

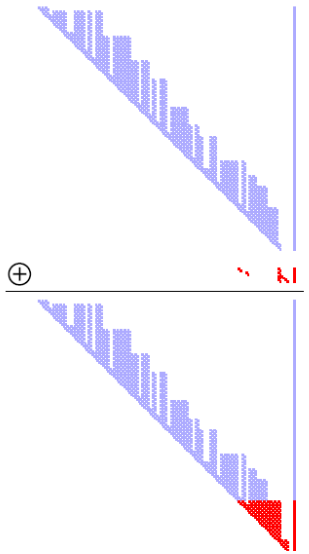
- Append to existing matrix factorization
- “Repair” using Givens rotations

Periodic batch steps for

- Relinearization
- Variable reordering (to keep sparsity)

Courtesy: M. Kaess

Factor Updates with Givens Rotations



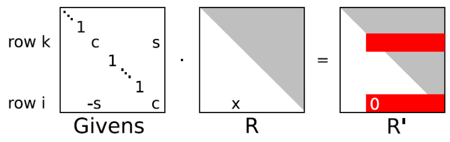
Old R factor

New rows

New R factor: Triangulated using Givens Rotations

Constant time!

Zeroing entries with Givens Rotations:



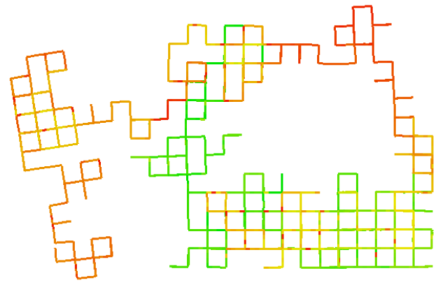
Numerically stable !

Courtesy: M. Kaess

Variable Reordering – Constrained COLAMD

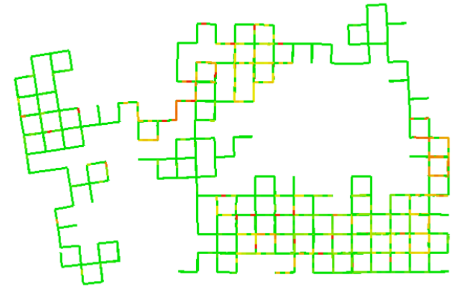
Greedy approach

Arbitrary placement of newest variable



Constrained Ordering

Newest variables forced to the end



Number of affected variables:

low high

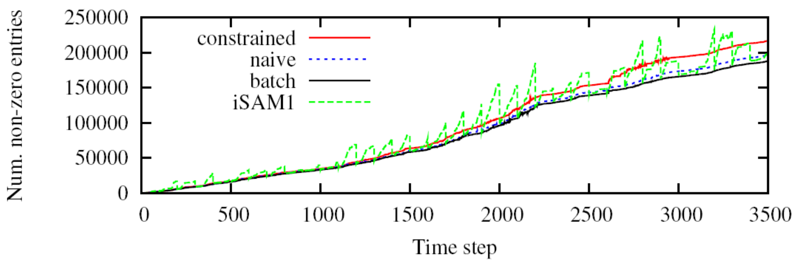
Much cheaper!

Courtesy: M. Kaess

[Kaess et al., TRO 08]

Variable Reordering – Fill-in

Incremental ordering still yields good overall ordering



- Only slightly more fill-in than batch COLAMD ordering
- Constrained ordering is worse than naïve/greedy:
  - Suboptimal ordering because of partial constraint, but cheaper to update!

Courtesy: M. Kaess

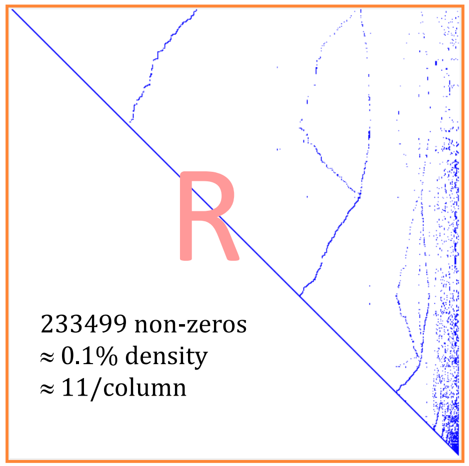
iSAM

Example from real sequence:

Square root inf. matrix

Side length: 21000 variables

Dense: 1.7GB, sparse: 1MB



Courtesy: M. Kaess

Next Lecture

- FastSLAM