Advanced Techniques for Mobile Robotics

TORO – SLAM with Gradient Descent

Wolfram Burgard, Cyrill Stachniss,

Kai Arras, Maren Bennewitz

Graph-based SLAM

- SLAM = simultaneous localization and mapping
- 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 the spatial constraints between them
- **Goal:** Find a configuration of the nodes that minimize the error introduced by the constraints

Topics Today

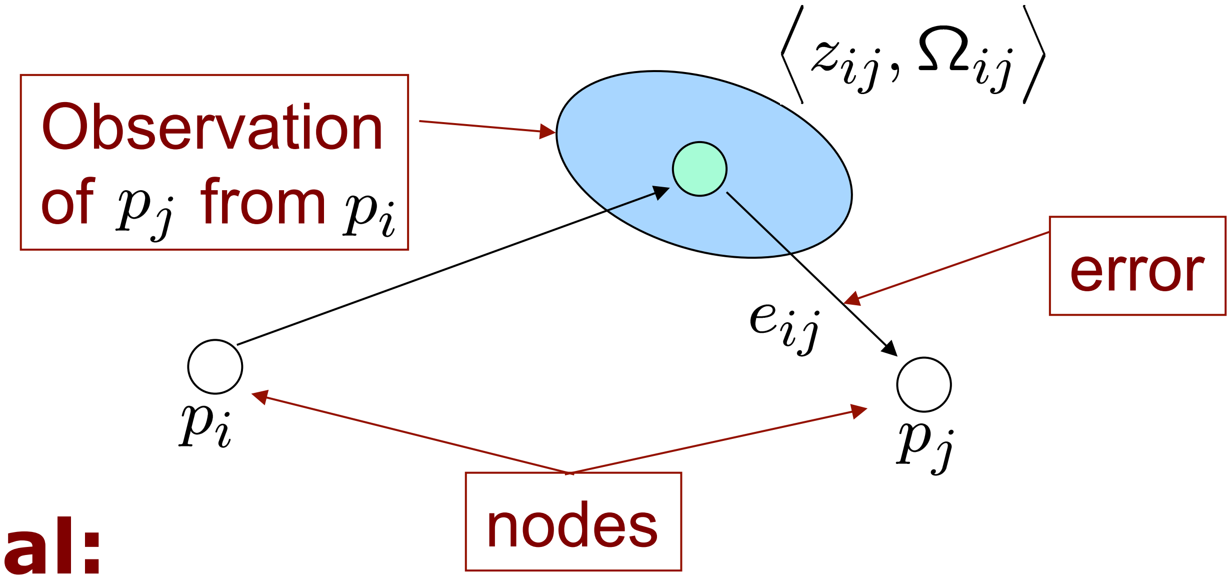
- Estimate the Gaussian posterior about the poses of the robot using gradient descent

Two Parts:

- Estimate the means via gradient descent (maximum likelihood map)
- Estimate the covariance matrices via belief propagation and covariance intersection

Problem Formulation

- The problem can be described by a graph



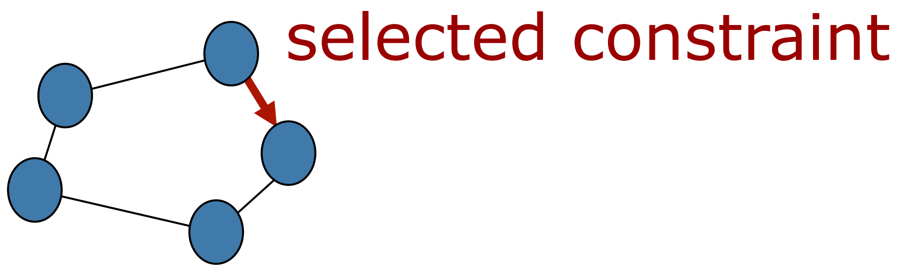
Goal:

- Find the assignment of poses to the nodes of the graph which minimizes the negative log likelihood of the observations:

$$\hat{\mathbf{p}} = \operatorname{argmin} \sum_{ij} e_{ij}^T \Omega_{ij} e_{ij}$$

Stochastic Gradient Descent

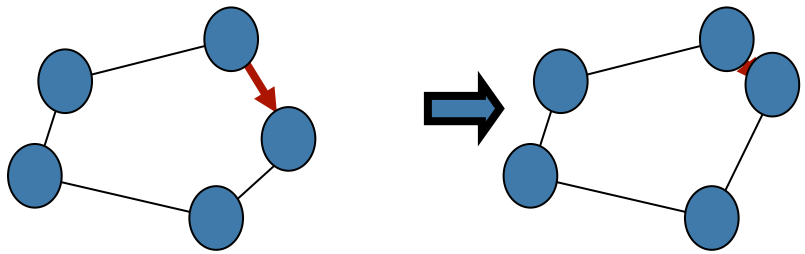
- Minimize the error individually for each constraint (decomposition of the problem into sub-problems)
- Solve one step of each sub-problem
- Solutions might be contradictory
- The magnitude of the correction decreases with each iteration
- Learning rate to achieve convergence



[First introduced in the SLAM community by Olson et al., '06]

Stochastic Gradient Descent

- Minimize the error individually for each constraint (decomposition of the problem into sub-problems)
- Solve one step of each sub-problem
- Solutions might be contradictory
- The magnitude of the correction decreases with each iteration
- Learning rate to achieve convergence

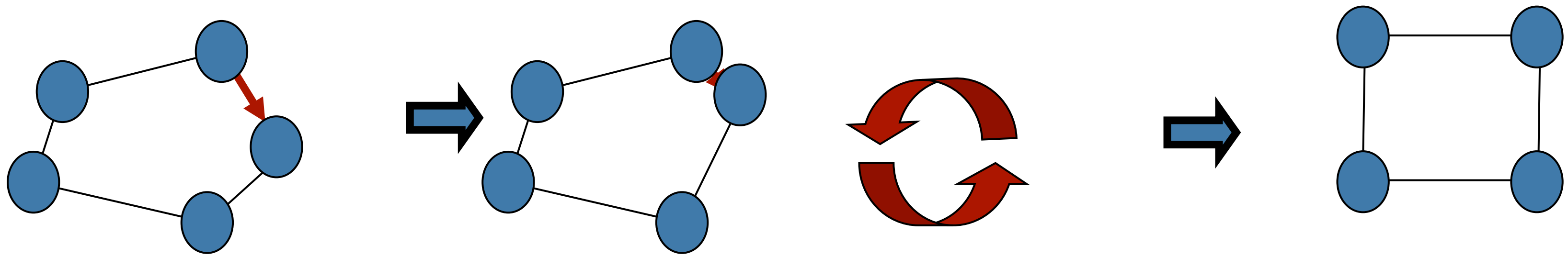


distribute the error over a set of involved nodes

[First introduced in the SLAM community by Olson et al., ’06]

Stochastic Gradient Descent

- Minimize the error individually for each constraint (decomposition of the problem into sub-problems)
- Solve one step of each sub-problem
- Solutions might be contradictory
- The magnitude of the correction decreases with each iteration
- Learning rate to achieve convergence



[First introduced in the SLAM community by Olson et al., ’06]

Preconditioned SGD

- Minimize the error individually for each constraint (decomposition of the problem into sub-problems)
- Solve one step of each sub-problem
- Solutions might be contradictory
- A solution is found when an equilibrium is reached
- Update rule for a single constraint:

$$\underbrace{\mathbf{x}^{t+1}}_{\text{Current solution}} = \overbrace{\mathbf{x}^{t}}^{\text{Previous solution}} + \underbrace{\lambda}_{\text{Learning rate}} \cdot \overbrace{\mathbf{H}^{-1}}^{\text{Hessian}} \underbrace{J_{ij}^{T}}_{\text{Jacobian}} \overbrace{\Omega_{ij}}^{\text{Information matrix}} \underbrace{r_{ij}}_{\text{residual}}$$

[First introduced in the SLAM community by Olson et al., ’06]

Node Parameterization

- How to represent the nodes in the graph?
- Impact on which parts need to be updated for a single constraint update?
- This are to the "sub-problems" in SGD
- Transform the problem into a different space so that:
  - the structure of the problem is exploited
  - the calculations become fast and easy

parameters

poses

$$\mathbf{x} = g(\mathbf{p}) \leftrightarrow \mathbf{p} = g^{-1}(\mathbf{x})$$

Mapping function

parameters

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{i,j} e'_{ij}(\mathbf{x})^T \Omega_{ij} e'_{ij}(\mathbf{x})$$

transformed problem

Parameterization of Olson

- Incremental parameterization:

$$x_i = p_i - p_{i-1}$$

parameters

poses

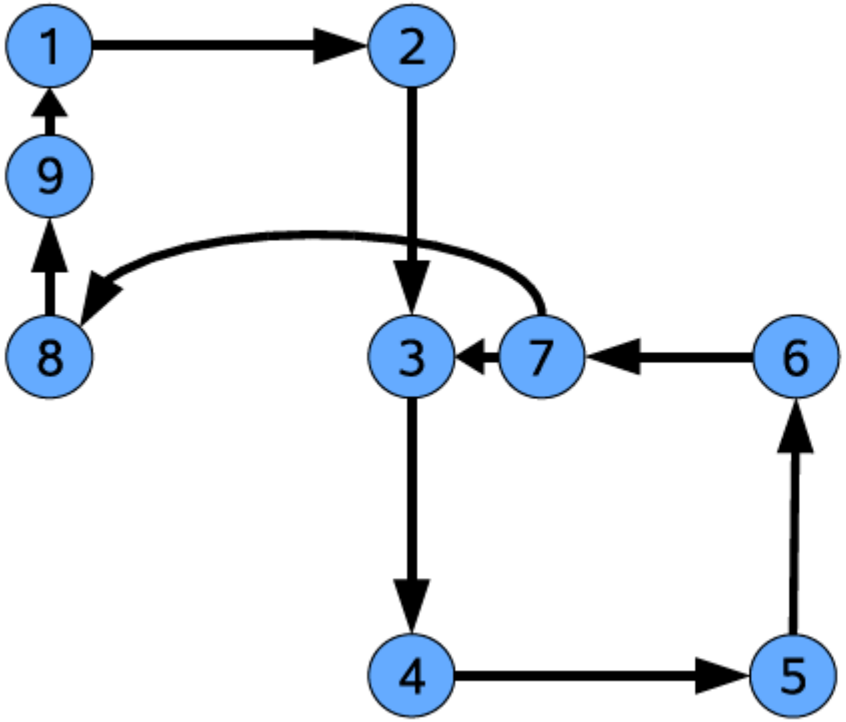
- Results directly from the trajectory takes by the robot
- Problem: for optimizing a constraint between the nodes *i* and *k*, one needs to updates the nodes *j* = *i*, …, *k* ignoring the topology of the environment

Alternative Parameterization

- Exploit the topology of the space to compute the parameterization
- Idea: “Loops should be one sub-problem”
- Such a parameterization can be extracted from the graph topology itself

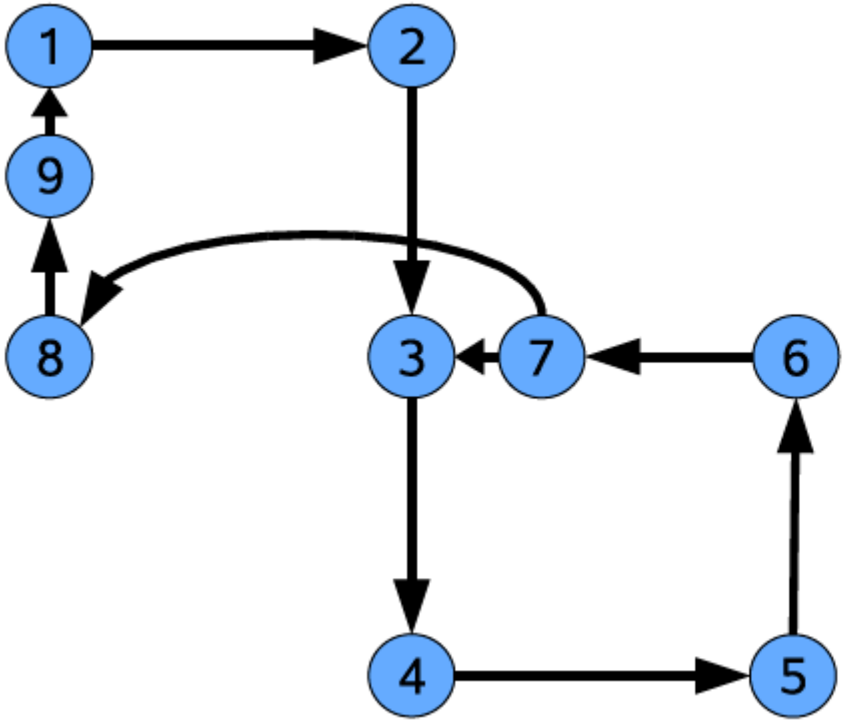
Tree Parameterization

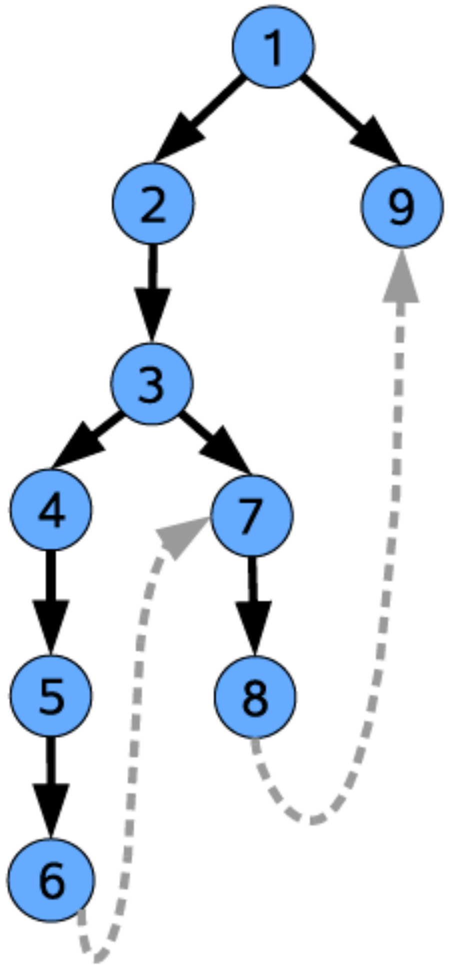
- How should such a problem decomposition look like?



Tree Parameterization

- Use a spanning tree!



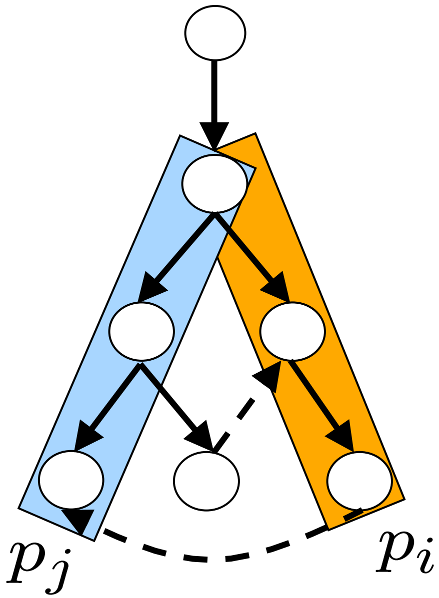


Tree Parameterization

- Construct a spanning tree from the graph
- The mapping function between the poses and the parameters is:

$$x_i = p_i \ominus p_{\mathsf{parent}(i)} \qquad\qquad X_i = P^{-1}_{\mathsf{parent}(i)} P_i$$

- Error of a constraint in the new parameterization

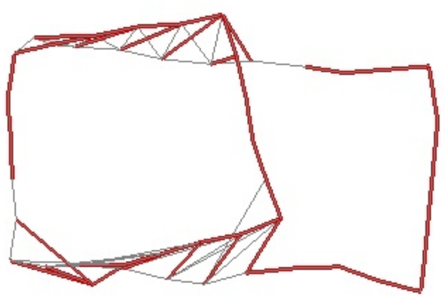


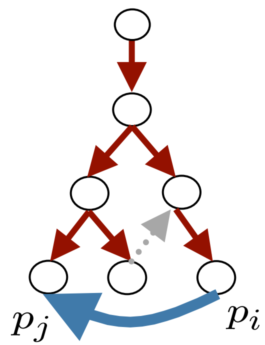
$$E_{ij} = \Delta_{ij}^{-1} \cdot \mathsf{UpChain}^{-1} \cdot \mathsf{DownChain}$$

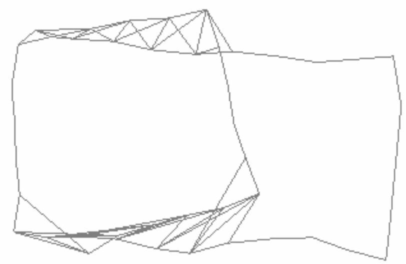
Only variables along the path of a constraint are involved in the update

Stochastic Gradient Descent using the Tree Parameterization

- The tree parameterization leads to several smaller problems which are either:
  - constraints on the tree (“open loop”)
  - constraints not in the tree (“a loop closure”)
- Each SGD equation independently solves one sub-problem at a time
- The solutions are integrated via the learning rate







Computation of the Update Step

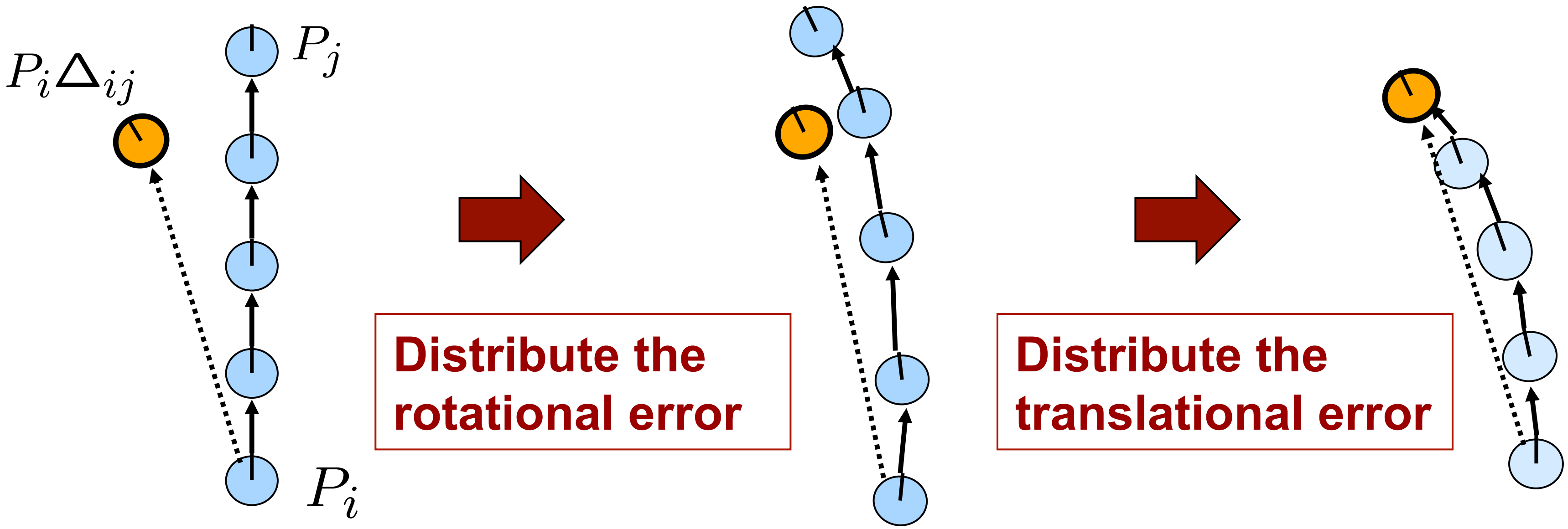
- 3D rotations lead to a nonlinear system
  - Update the poses directly according to the SGD equation may lead to poor convergence
  - This increases with the connectivity of the graph
- Key idea in the SGD update:

$$\mathbf{\Delta x} = \lambda \cdot \mathbf{H}^{-1} J_{ij}^T \Omega_{ij} r_{ij}$$

Idea: distribute a fraction of the residual along the parameters so that the error of that constraint is reduced

Computation of the Update Step

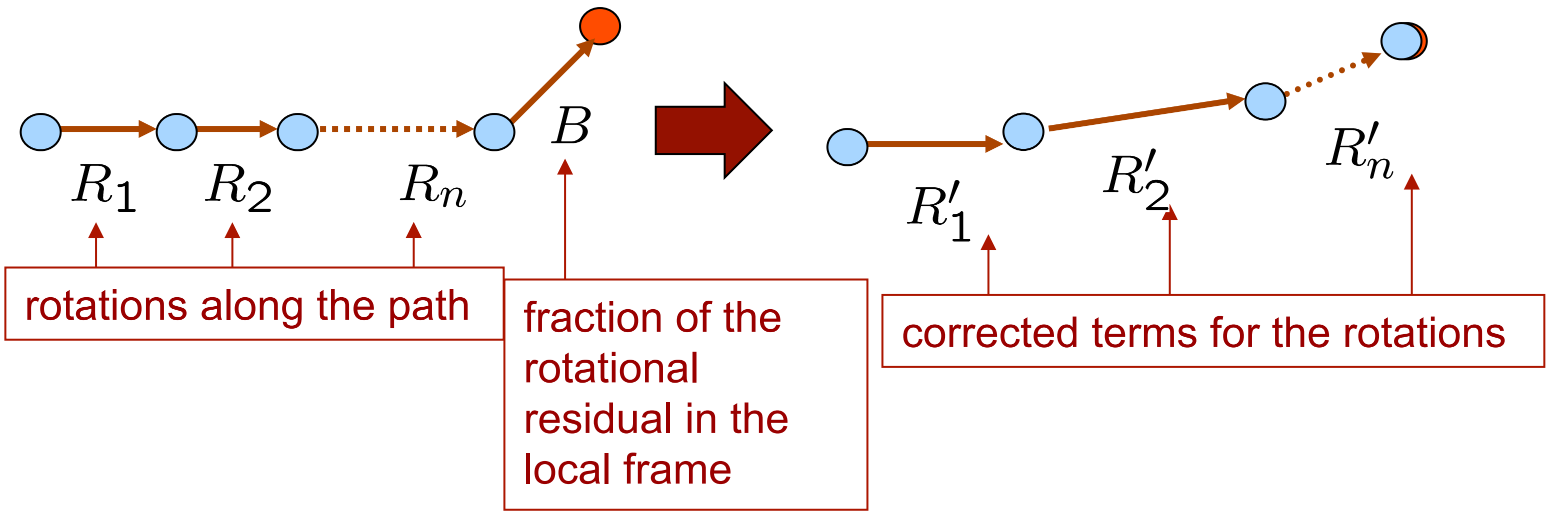
Alternative update in the "spirit" of the SGD: Smoothly deform the path along the constraints so that the error is reduced



Distribution of the Rotational Error

- In 3D, the rotational error cannot be simply added to the parameters because the rotations are not commutative
- Find a set of **incremental** rotations so that the following equality holds:

$$R_1 R_2 \cdots R_n B = R'_1 R'_2 \cdots R'_n$$



Distributing the Rotational Residual

- Assume that the first node is the reference frame
- We want a correcting rotation with **a single axis**
- Let A_i be the orientation of the i-th node in the global reference frame

$$A'_n = A_n B = Q A_n$$

with a decomposition of the rotational residual into a chain of incremental rotations obtained by spherical linear interpolation (slerp)

$$\begin{aligned} Q &= Q_1 Q_2 \cdots Q_n \\ Q_k &= \mathsf{slerp}(Q, u_{k-1})^T \mathsf{slerp}(Q, u_k) \qquad u \in [0 \ldots \lambda] \end{aligned}$$

- Slerp has been designed for 3d animation: constant speed motion along a circle arc

What is the SLERP?

- SLERP = Spherical LinEar inteRPolation
- Introduced by Ken Shoemake for interpolations in 3D animations
- Constant speed motion along a circle arc with unit radius
- Properties:

$$
\begin{aligned}
\mathcal{R}' &:= \mathbf{slerp}(\mathcal{R}, u) \\
\mathbf{axisOf}(\mathcal{R}') &= \mathbf{axisOf}(\mathcal{R}) \\
\mathbf{angleOf}(\mathcal{R}') &= u \cdot \mathbf{angleOf}(\mathcal{R})
\end{aligned}
$$

Distributing the Rotational Residual

- Given the Q_k, we obtain

$$A'_k = Q_1 \dots Q_k = Q_{1:k} A_k$$

- as well as

$$R'_k = A'^T_{k-1} A'_k$$

- and can then solve:

$$\begin{aligned} R'_1 &= Q_1 R_1 \\ R'_2 &= (Q_1 R_1)^T Q_{1:2} R_{1:2} = R_1^T Q_1^T Q_1 Q_2 R_1 R_2 \\ &\vdots \\ R'_k &= [(R_{1:k-1})^T Q_k R_{1:k-1}] R_k \end{aligned}$$

Distributing the Rotational Residual

- Resulting update rule

$$R'_k \;=\; (R_{1:k-1})^T Q_k R_{1:k}$$

- It can be shown that the change in each rotational residual is bounded by

$$\Delta r'_{k,k-1} \leq |\mathsf{angleOf}(Q_k)|$$

- This bounds a potentially introduced error at node k when correcting a chain of poses including k

How to Determine u_k?

- The values of u_k describe the relative distribution of the error along the chain

$$Q_k = \text{slerp}(Q, u_{k-1})^T \text{slerp}(Q, u_k) \qquad u \in [0 \ldots \lambda]$$

- Here, we need to consider the uncertainty of the constraints

$$u_k = \min\left(1, \lambda|\mathcal{P}_{ij}|\right) \left[\sum_{m \in \mathcal{P}_{ij} \wedge m \leq k} d_m^{-1}\right] \left[\sum_{m \in \mathcal{P}_{ij}} d_m^{-1}\right]^{-1}$$

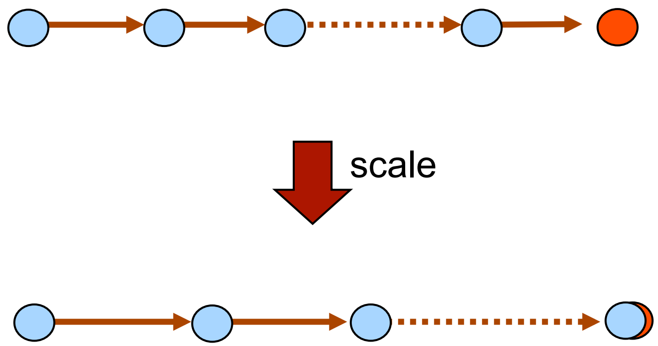
$$d_m = \sum_{\langle l,m \rangle} \min\left[\text{eigen}(\Omega_{lm})\right]$$

all constraints connecting m

- This assumes roughly spherical covariances!

Distributing the Translational Error

- That is trivial
- Just scale the x, y, z dimension



Summary of the Algorithm

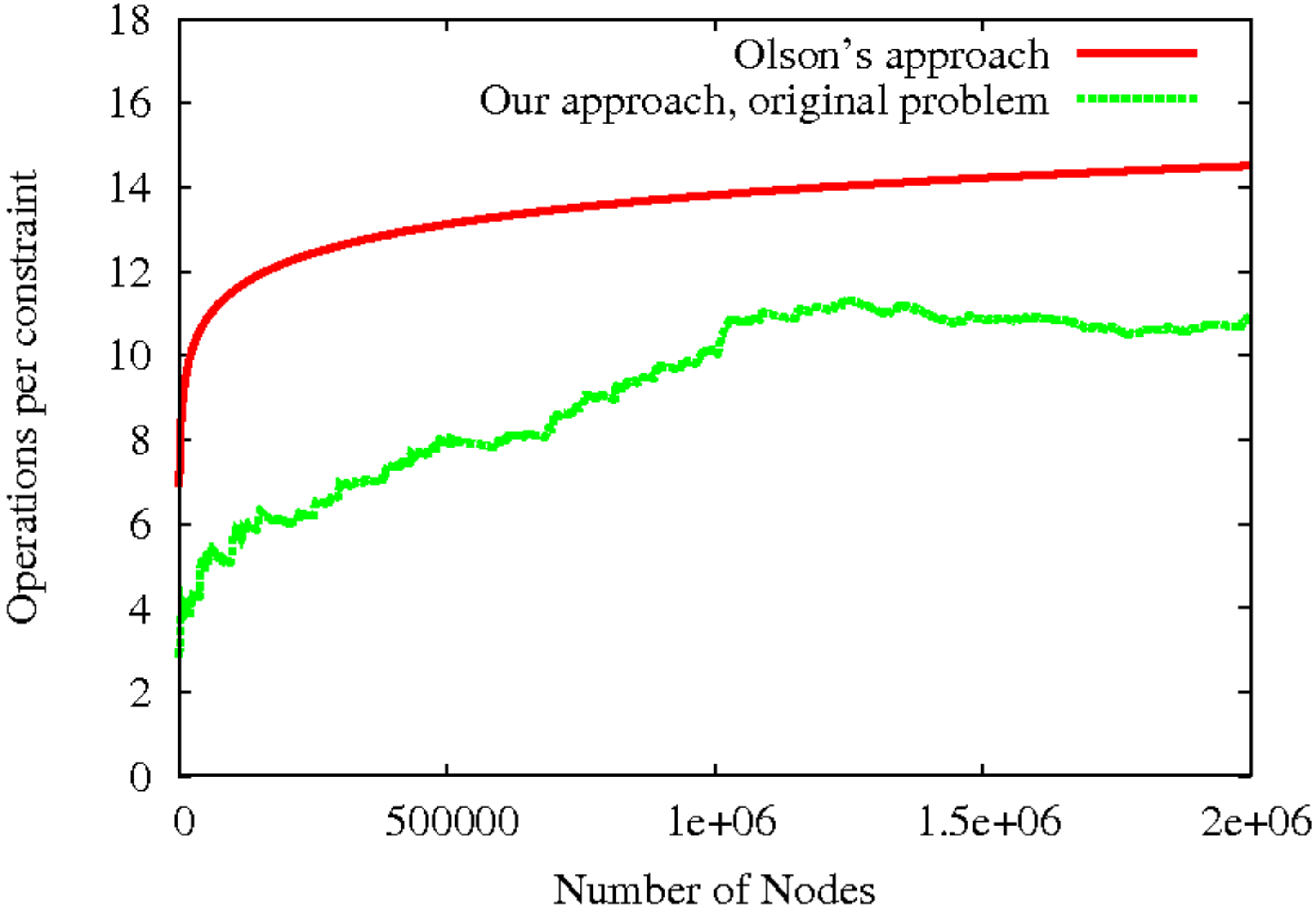
- Decompose the problem according to the tree parameterization
- Loop:
  - Select a constraint
    - Randomly
    - Alternative: sample inverse proportional to the number of nodes involved in the update
  - Compute the nodes involved in the update
    - Nodes according to the parameterization tree
  - Reduce the error for this sub-problem
    - Reduce the rotational error (slerp)
    - Reduce the translational error

Complexity

- In each iteration, the approach considers all constraints
- Each constraint optimization step requires to update a set of nodes (on average: the average “path length according to the tree)
- This results in a complexity per iteration of

$$\mathcal{O}(\underset{\substack{\uparrow \\ \text{\#constrains}}}{M} \cdot \underset{\substack{\uparrow \\ \text{avg. path length} \\ \text{(parameterization tree)}}}{l})$$

Cost of a Constraint Update



$$\approx \mathcal{O}(M \cdot \log(N))$$

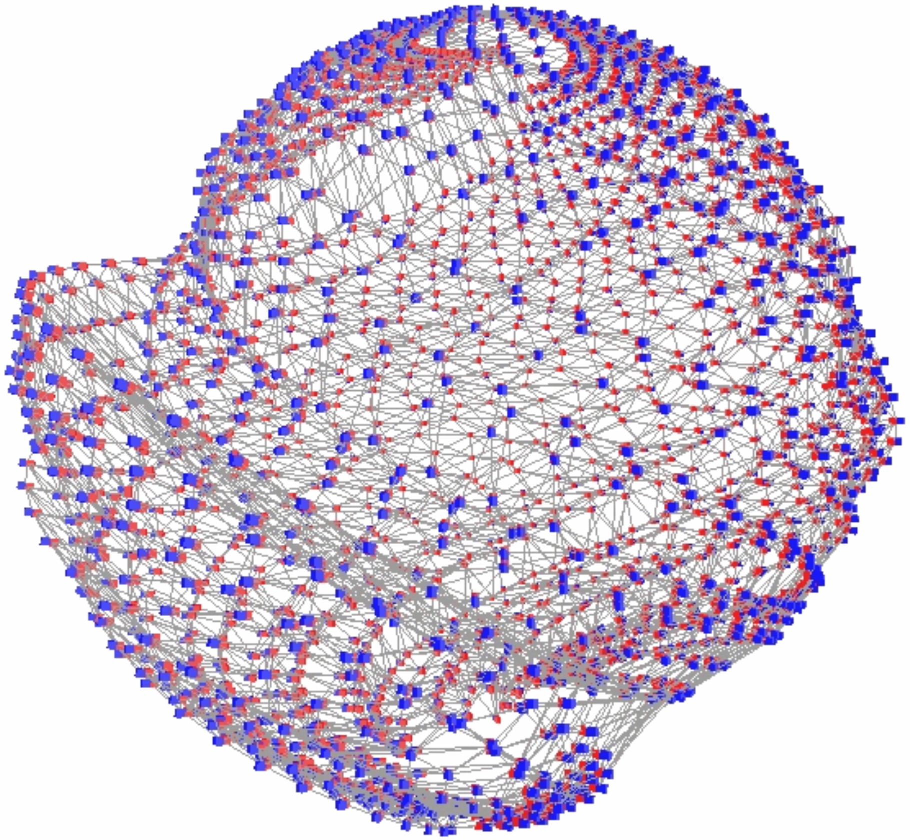
Node Reduction

- Complexity grows with the length of the trajectory
- Bad for life-long learning
- Idea: Combine constraints between nodes if the robot is well-localized

$$
\begin{aligned}
\Omega_{ij} &= \Omega_{ij}^{(1)} + \Omega_{ij}^{(2)} \\
\delta_{ij} &= \Omega_{ij}^{-1}(\Omega_{ij}^{(1)}\delta_{ij}^{(1)} + \Omega_{ij}^{(2)}\delta_{ij}^{(2)})
\end{aligned}
$$

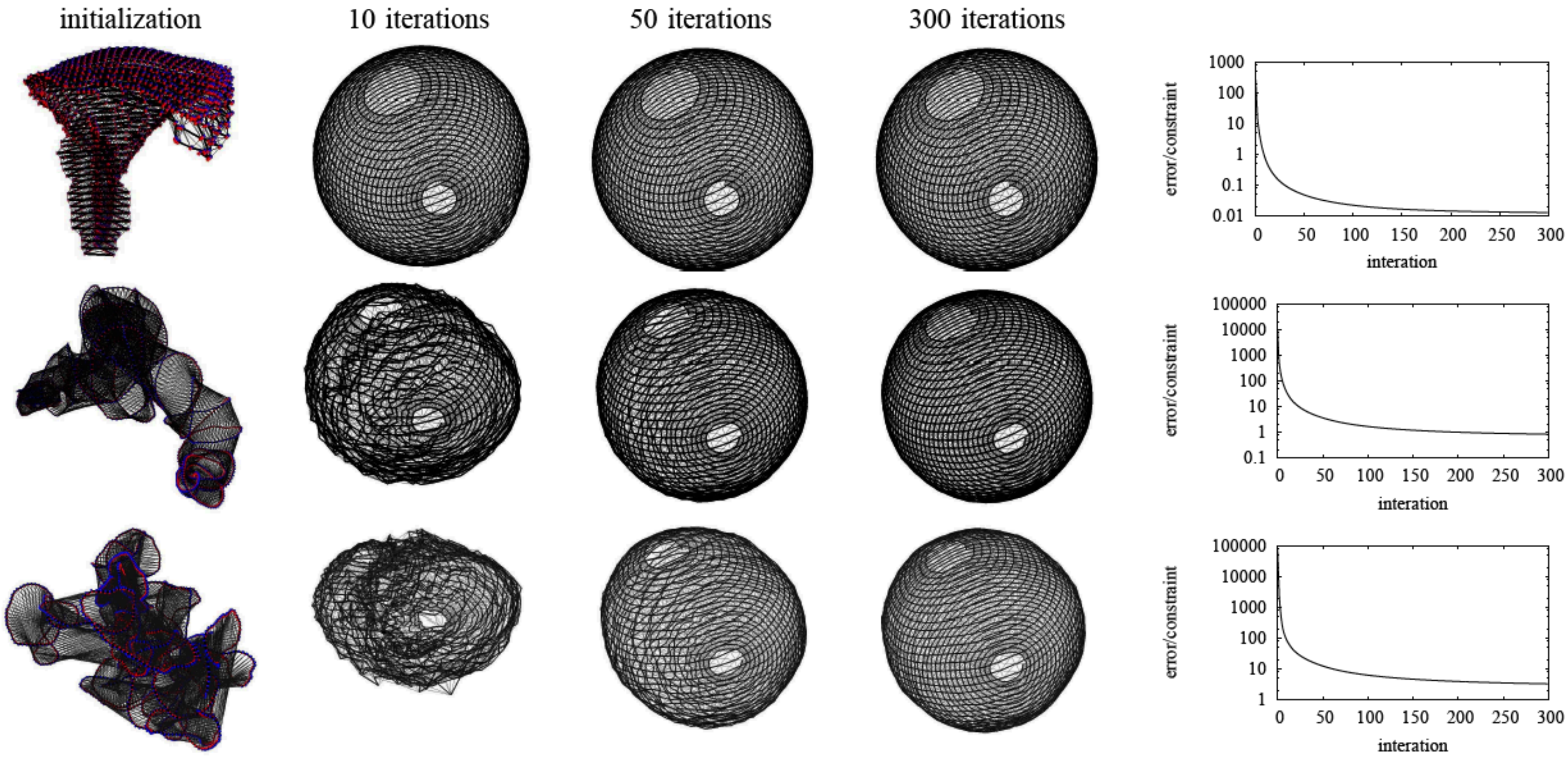
- Similar to adding rigid constraints
- Complexity depends only on the size if the environment, not the length of the trajectory

Simulated Experiment

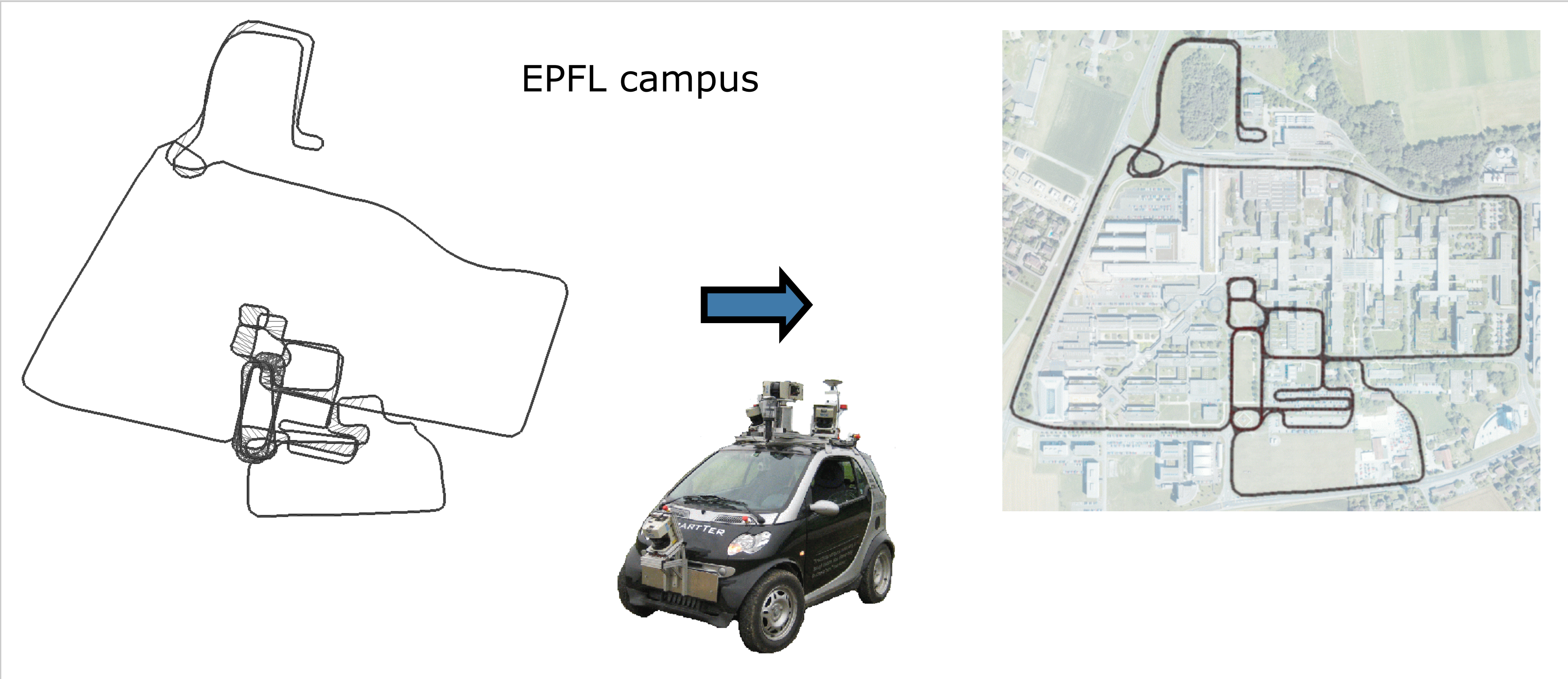


- Highly connected graph
- Poor initial guess
- 2200 nodes
- 8600 constraints

Spheres with Different Noise

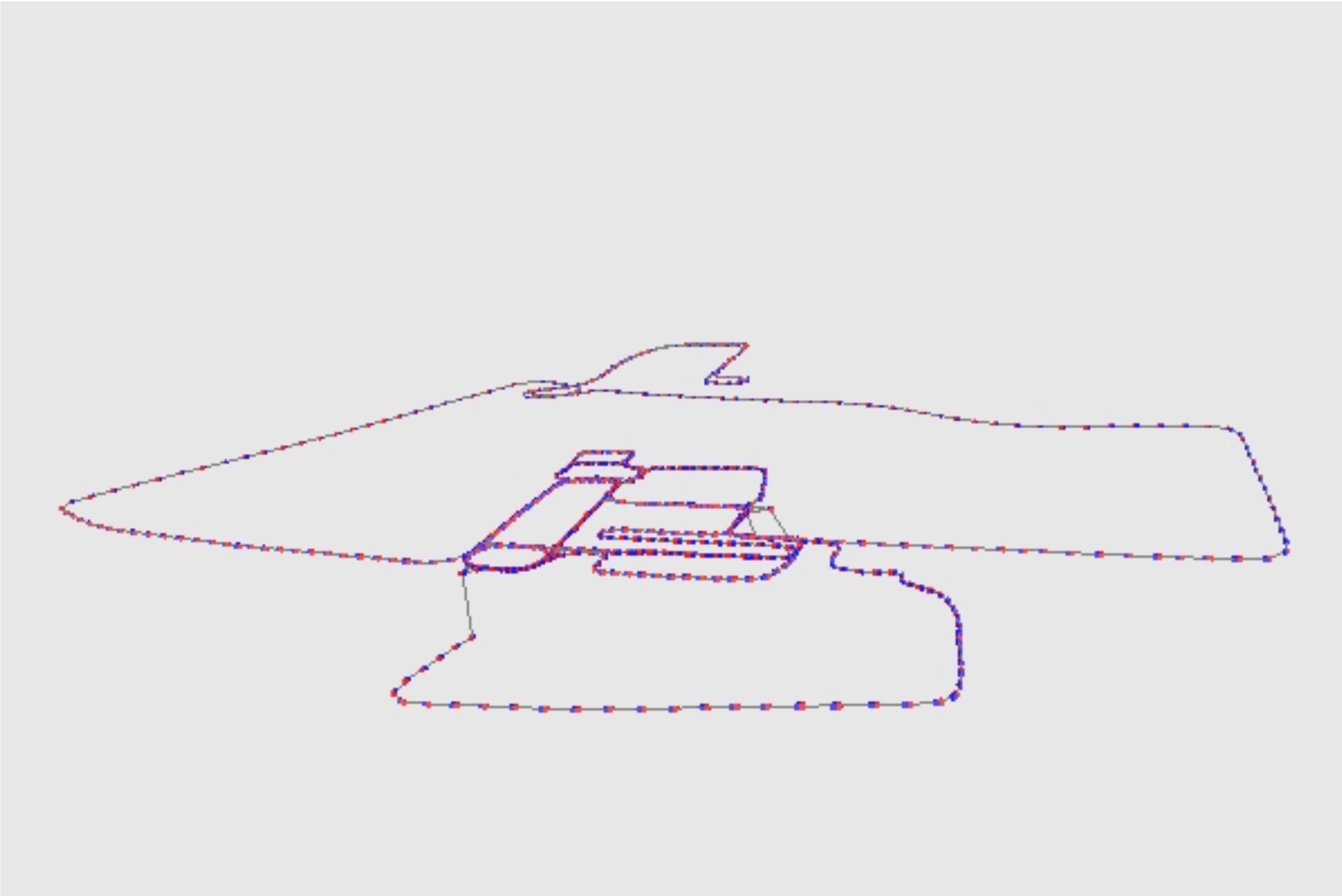


Mapping the EPFL Campus



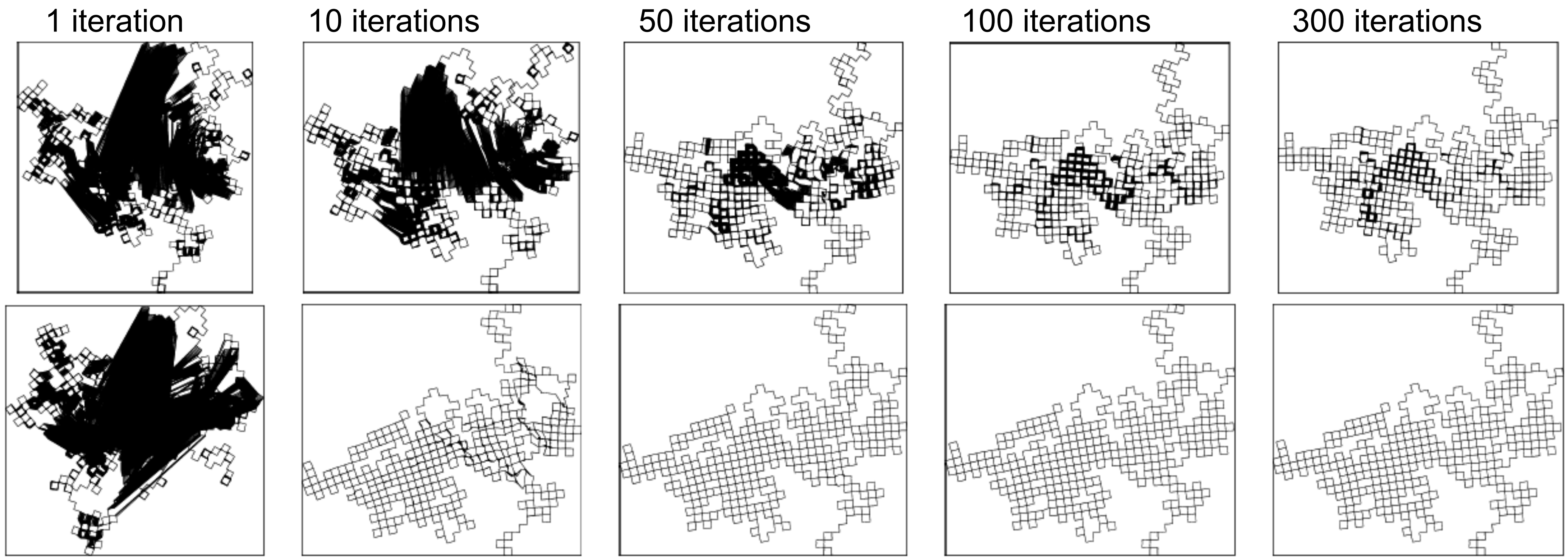
- 10km long trajectory with 3D laser scans
- Not easily tractable by most standard optimizers

Mapping the EPFL Campus



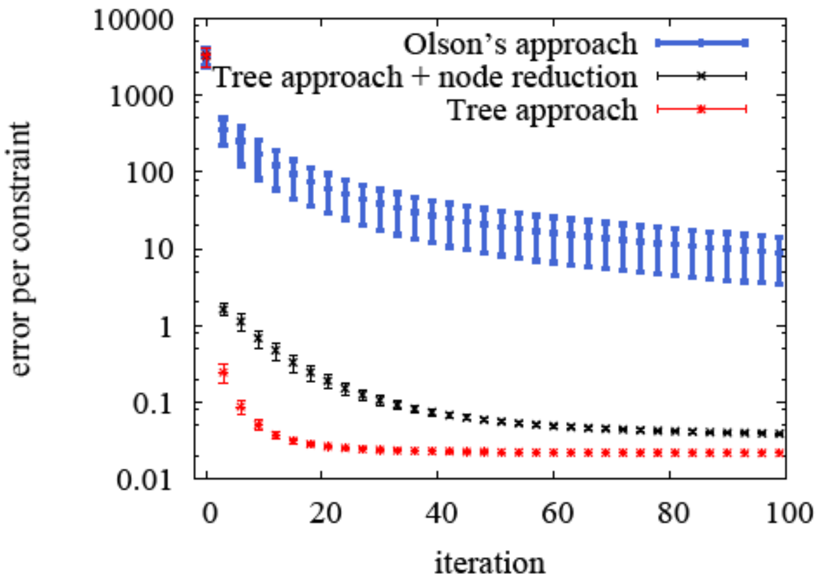
TORO vs. Olson's Approach

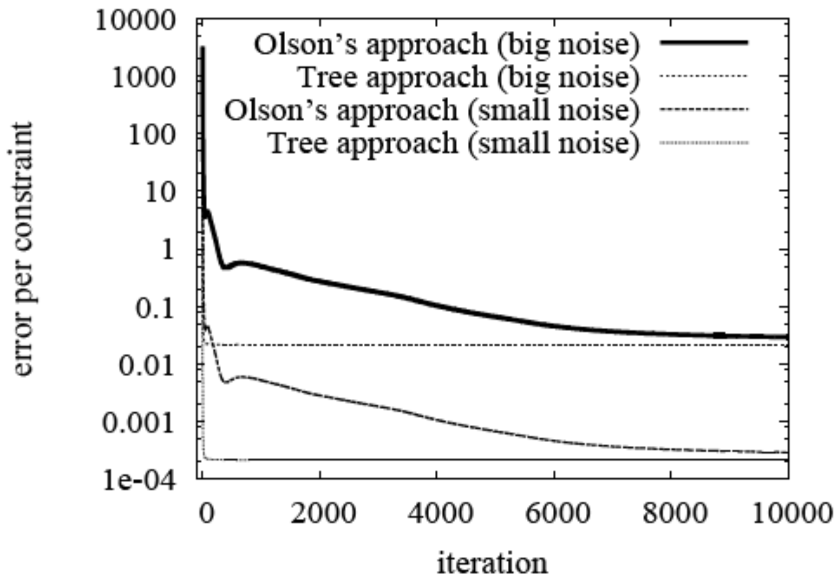
Olson's approach

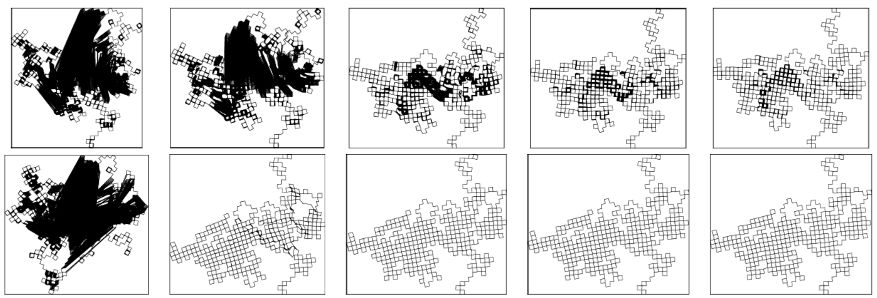


TORO

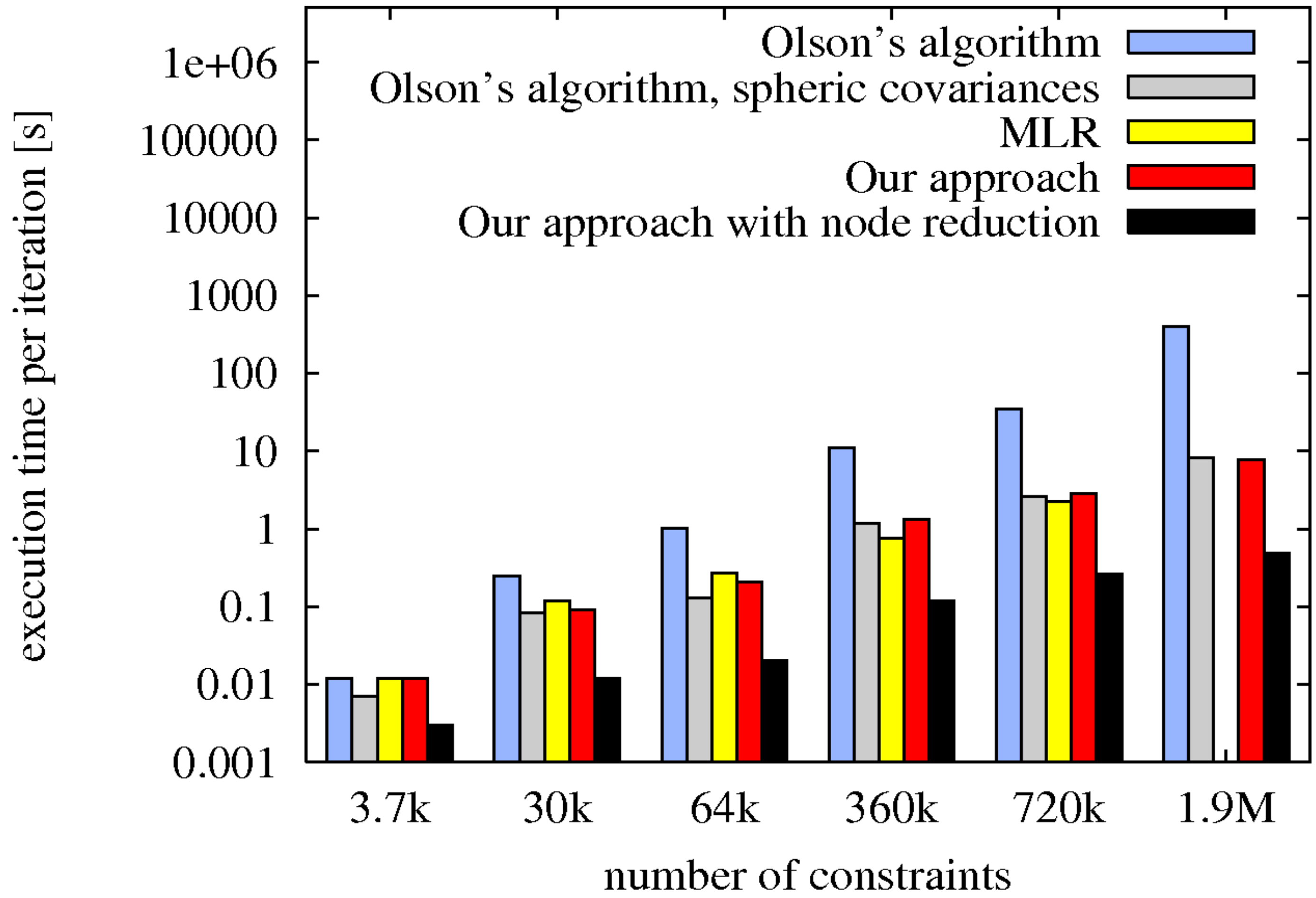
TORO vs. Olson's Approach





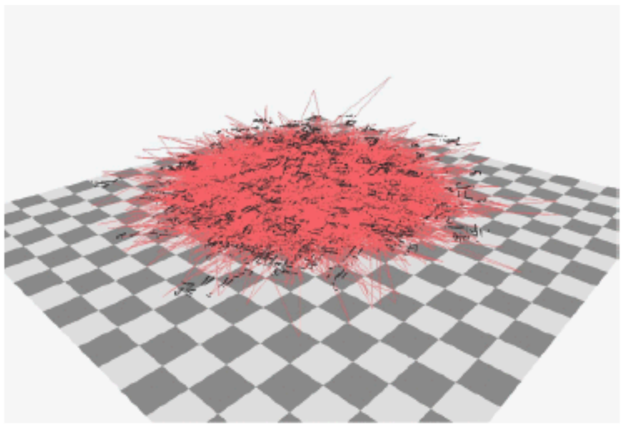


Time Comparison (2D)

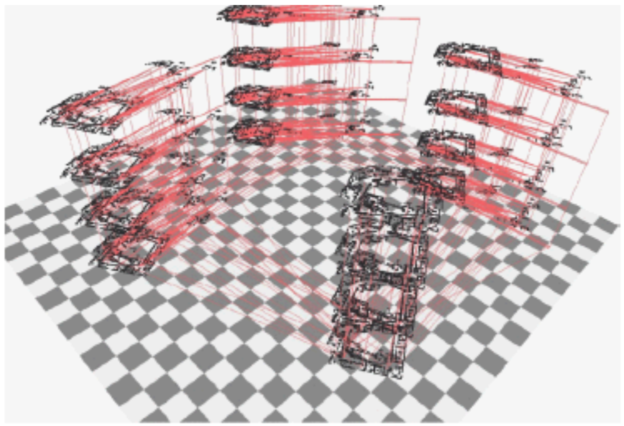


Robust to the Initial Guess

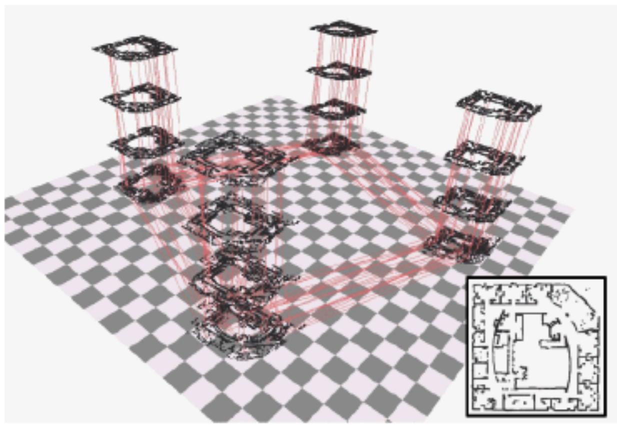
- Random initial guess
- Intel datatset as the basis for 16 floors distributed over 4 towers



initial configuration



intermediate result



final result (50 iterations)

TORO Summary

- Robust to bad initial configurations
- Efficient technique for ML map estimation
- Works in 2D and 3D
- Scales up to millions of constraints
- Available at OpenSLAM.org
  http://www.openslam.org/toro.html

Drawbacks of TORO

- The slerp-based update rule optimizes rotations and translations separately.
- It assume **roughly spherical covariance ellipses**.

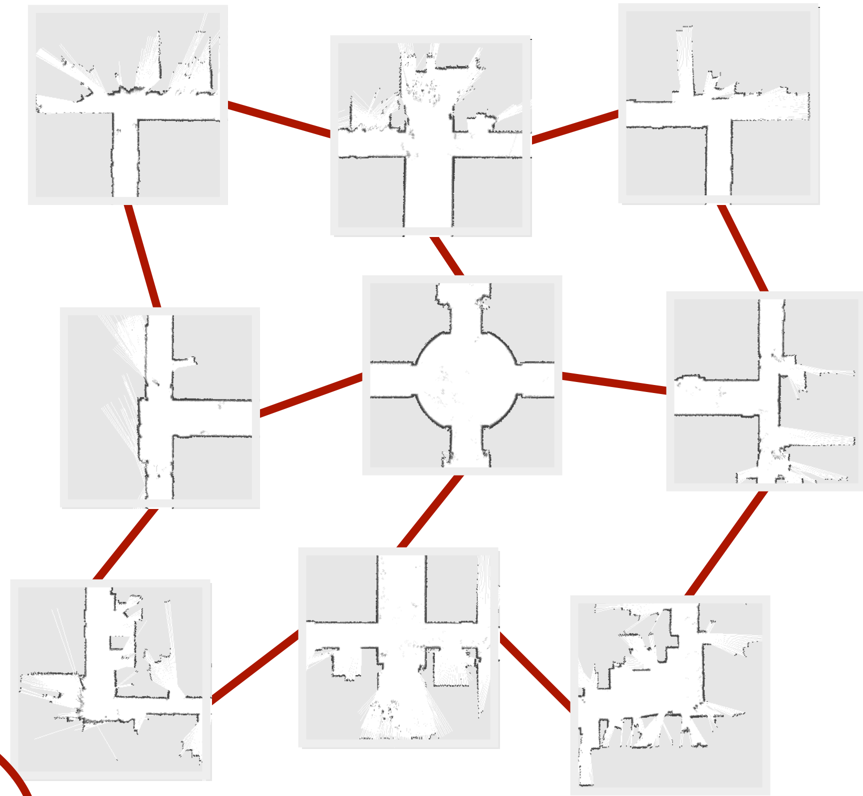
- It is a maximum likelihood technique.
No covariance estimates!
- Approach of Tipaldi et al. accurately estimates the covariances after convergence [Tipaldi et al., 2007]

Data Association

- TORO computes the mean of the distribution given the data associations
- To determine the data associations, we need the uncertainty about the nodes' poses
- Approaches to compute the uncertainties:
  - Matrix inversion
  - Loopy belief propagation
  - Belief propagation on a spanning tree
  - **Loopy intersection propagation**

Graphical SLAM as a GMRF

- Factor the distribution
  - **local** potentials
  - **pairwise** potentials



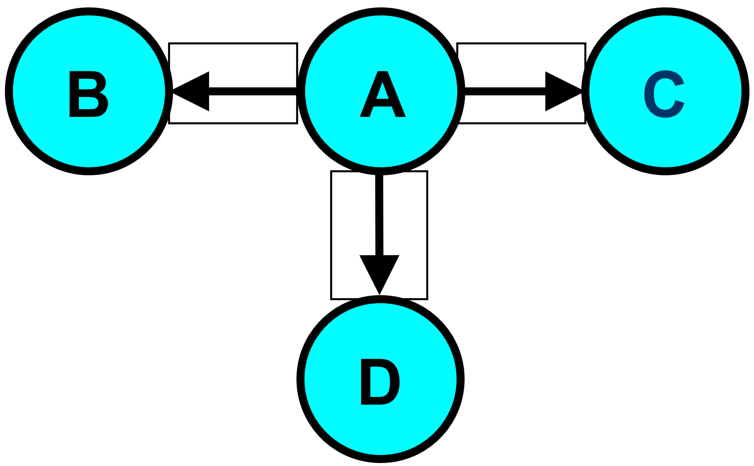
$$p(x) = \frac{1}{Z} \prod_{i=1}^{n} \phi_i(x_i) \prod_{j=i+1}^{n} \underbrace{\phi_{i,j}(x_i, x_j)}_{\text{Gaussian in canonical form}}$$

Belief Propagation

- Inference by local message passing
- Iterative process
  - **Collect** messages

$$\cancel{\mathbf{m}_i^{(t)} = \boldsymbol{\eta}_i + \sum_{j \in \mathcal{N}_i} \mathbf{m}_{ji}^{(t-1)}}$$

$$\mathrm{M}_i^{(t)} = \Omega_i + \sum_{j \in \mathcal{N}_i} \mathrm{M}_{ji}^{(t-1)}$$



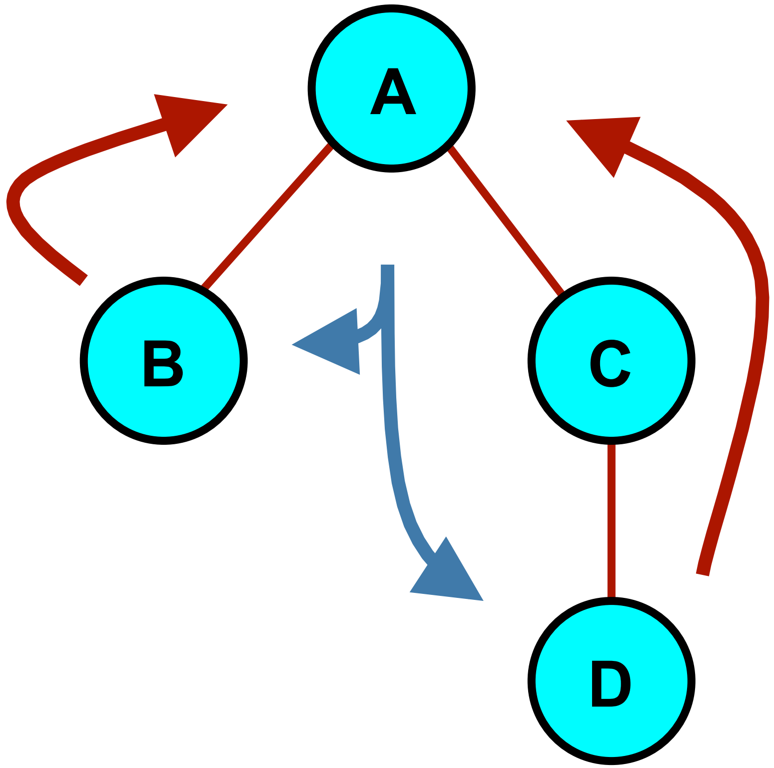
  - **Send** messages

$$\cancel{\mathbf{m}_{ij}^{(t)} = \boldsymbol{\eta}_{ij}^{j} - \Omega_{ij}^{[ji]} \left(\Omega_{ij}^{[ii]} + \mathrm{M}_i^{(t)} - \mathrm{M}_{ji}^{(t-1)} \right)^{-1} \left(\boldsymbol{\eta}_{ij}^{i} + \mathbf{m}_i^{(t)} - \mathbf{m}_{ji}^{(t-1)} \right)}$$

$$\mathrm{M}_{ij}^{(t)} = \boxed{\Omega_{ij}^{[jj]} - \Omega_{ij}^{[ji]} \left(\Omega_{ij}^{[ii]} + \mathrm{M}_i^{(t)} - \mathrm{M}_{ji}^{(t-1)} \right)^{-1} \Omega_{ij}^{[ij]}}$$

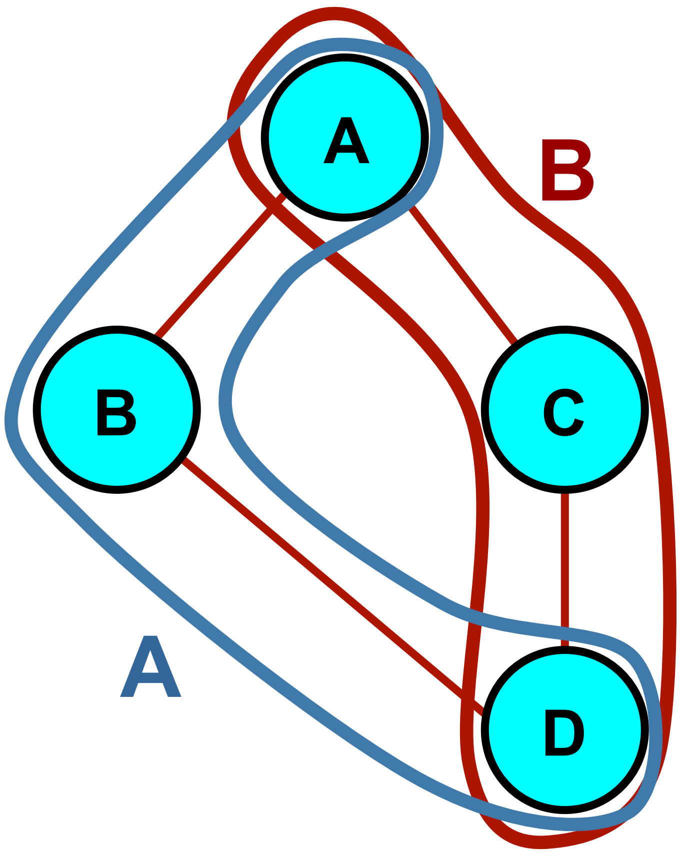
Ignore the math!

Belief Propagation - Trees



- Exact inference
- Message passing
- Two iterations
  - From leaves to root: **variable elimination**
  - From root to leaves: **back substitution**

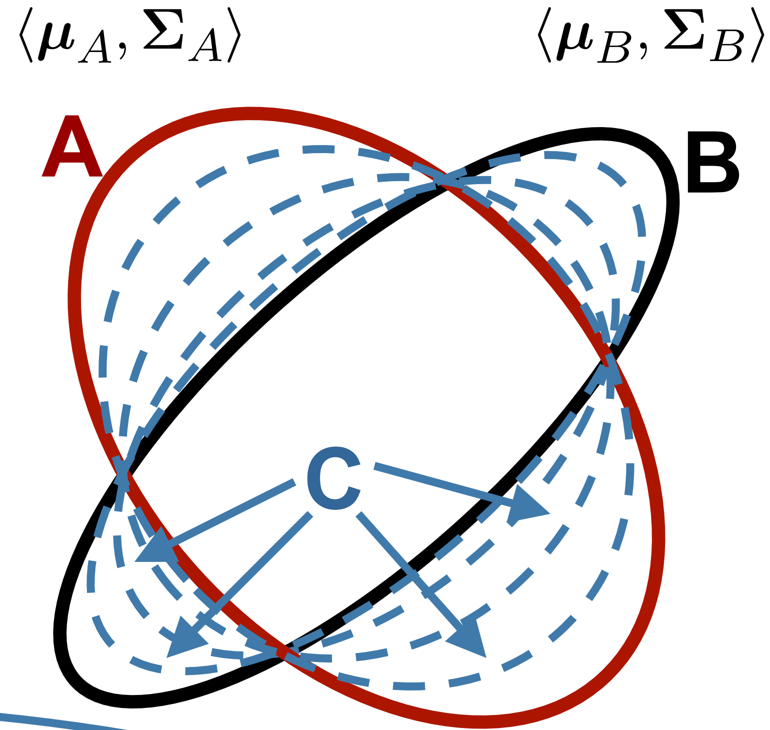
Belief Propagation - Loops



- Approximation
- Multiple paths
- Overconfidence
  - Correlations between path **A** and path **B**
- How to integrate information at **D**?

Covariance Intersection

- Fusion rule for unknown correlations
- Combine **A** and **B** to obtain **C**



$$\begin{aligned} \Sigma_C &= (\omega \Sigma_A^{-1} + (1-\omega)\Sigma_B^{-1})^{-1} \\ \boldsymbol{\mu}_C &= \Sigma_C(\omega \Sigma_A^{-1}\boldsymbol{\mu}_A + (1-\omega)\Sigma_B^{-1}\boldsymbol{\mu}_B) \end{aligned}$$

Loopy Intersection Propagation

Key ideas

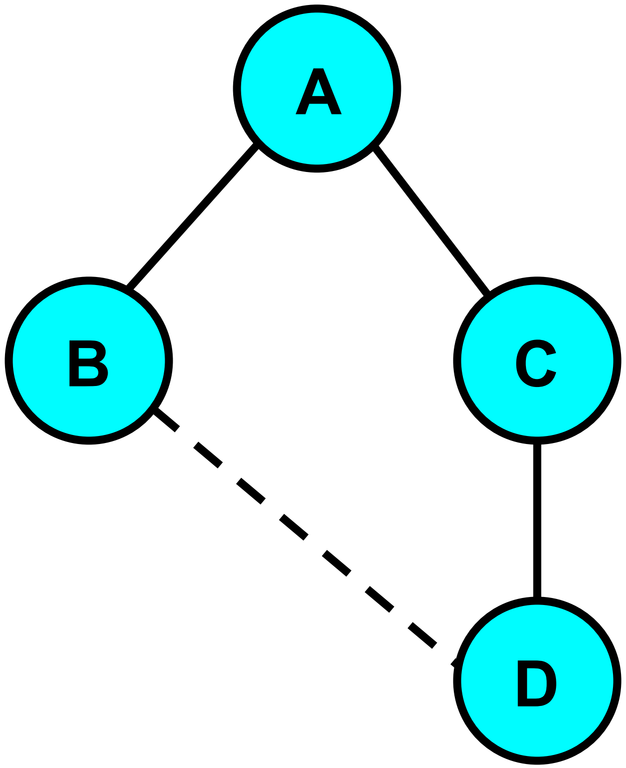
- Exact inference on a spanning tree computed via **cutting matrices**
- Augment the tree with information coming from loops within **local potentials** (priors)
- Apply belief propagation

Approximation via Cutting Matrix

- Removal as matrix subtraction

$$\hat{\Omega} = \Omega - \mathrm{K}$$

- Regular cutting matrix
- Cut all off-tree edges



$$\mathrm{K}_{BD} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \Omega_{BD}^{[BB]} & 0 & \Omega_{BD}^{[BD]} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \Omega_{BD}^{[DB]} & 0 & \Omega_{BD}^{[DD]} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fusing Loops with Spanning Trees

- Estimate **A** and **B**

$$
\begin{aligned}
\mathbf{E}_{BD}^{[D]} &= \boxed{\Omega_{BD}^{[BB]} - \Omega_{BD}^{[BD]}(\mathrm{M}_D + \Omega_{BD}^{[DD]})^{-1}\Omega_{BD}^{[DB]}} \\
\mathbf{E}_{BD}^{[B]} &= \boxed{\Omega_{BD}^{[DD]} - \Omega_{BD}^{[DB]}(\mathrm{M}_B + \Omega_{BD}^{[BB]})^{-1}\Omega_{BD}^{[BD]}}
\end{aligned}
$$

Ignore the math!

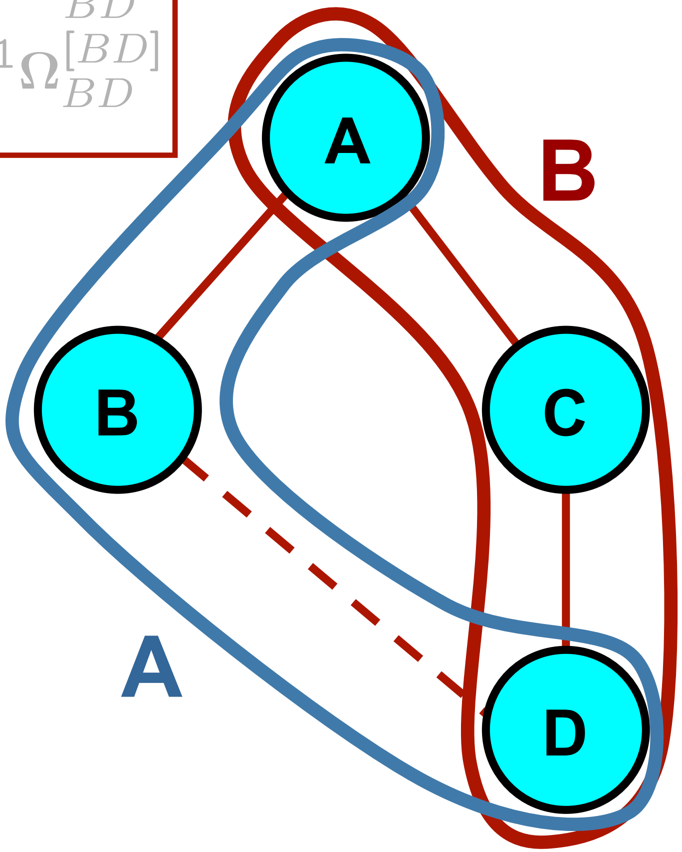
- Fuse the estimates

$$
\begin{aligned}
\hat{\mathrm{M}}_B &= \boxed{\omega_B \mathrm{M}_B + (1-\omega_B)\mathrm{E}_{BD}^{[B]}} \\
\hat{\mathrm{M}}_D &= \boxed{\omega_D \mathrm{M}_D + (1-\omega_D)\mathrm{E}_{BD}^{[D]}}
\end{aligned}
$$

Covariance Intersection!

- Compute the priors

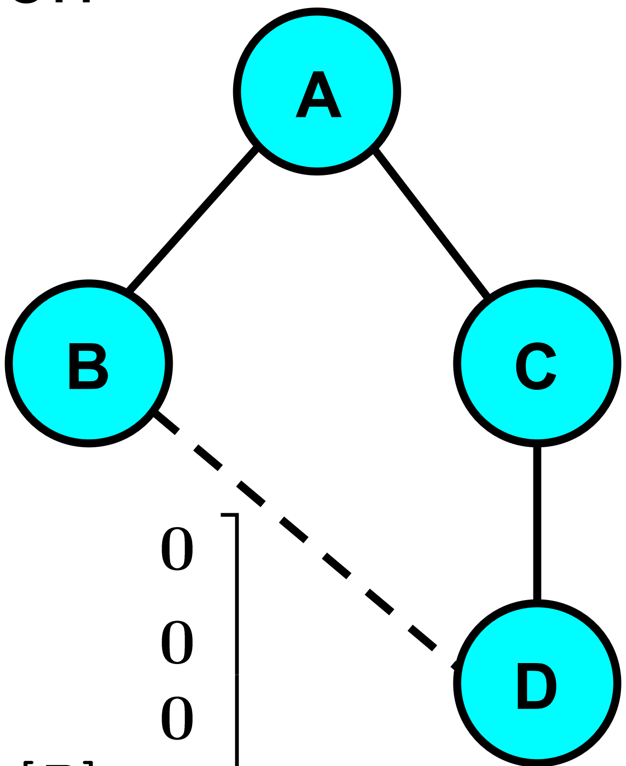
$$\mathbf{P}_{ij}^{[k]} = \hat{\mathrm{M}}_k - \mathrm{M}_k$$



Remove Edge and Add Priors

- Removal of the edge and adding priors realized as a matrix subtraction

$$\hat{\mathbf{\Omega}} = \mathbf{\Omega} - \mathbf{K}$$

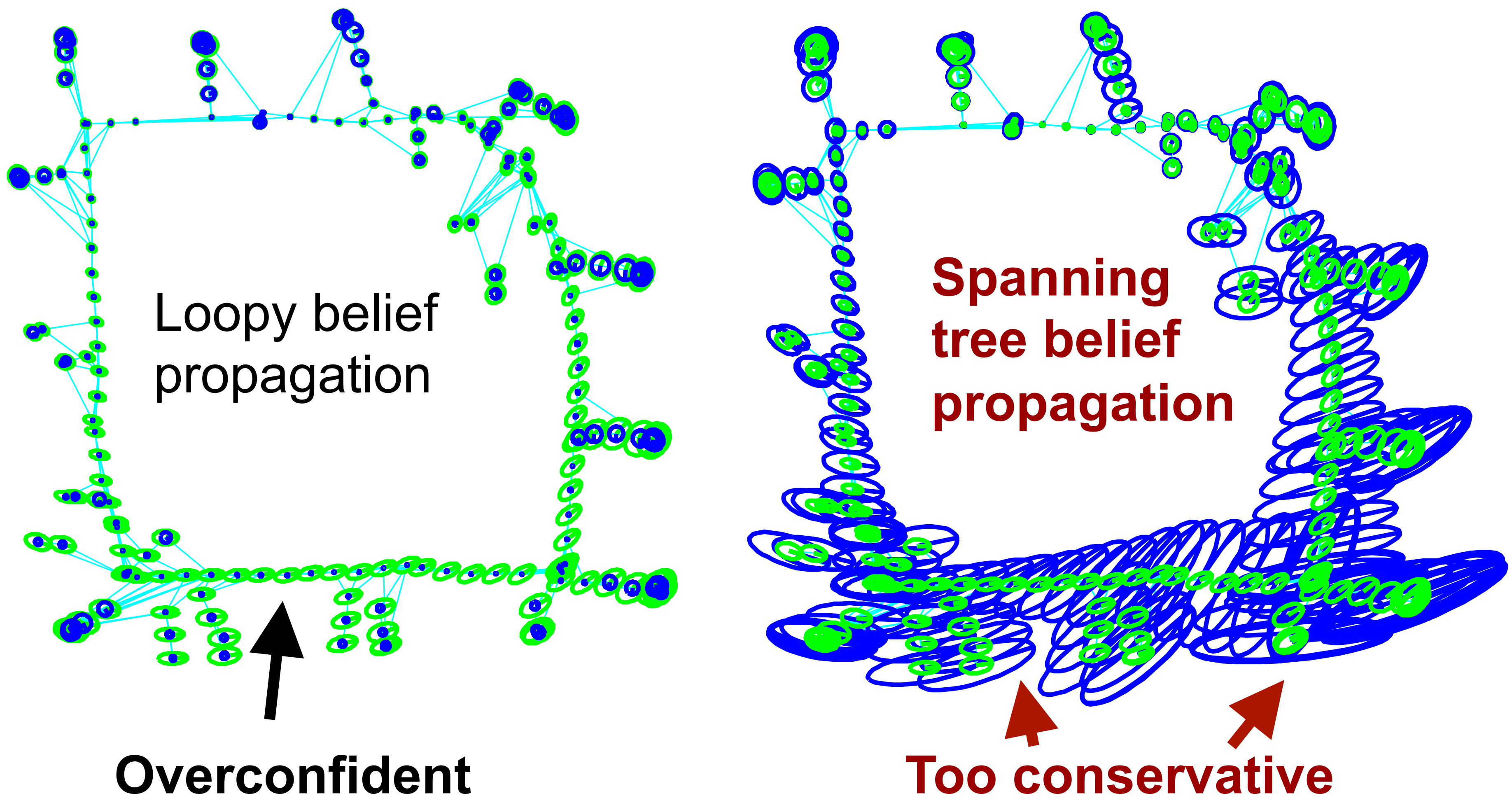


$$\mathbf{K}_{BD} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Omega}_{BD}^{[BB]} - \mathbf{P}_{BD}^{[B]} & \mathbf{0} & \mathbf{\Omega}_{BD}^{[BD]} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Omega}_{BD}^{[DB]} & \mathbf{0} & \mathbf{\Omega}_{BD}^{[DD]} - \mathbf{P}_{BD}^{[D]} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

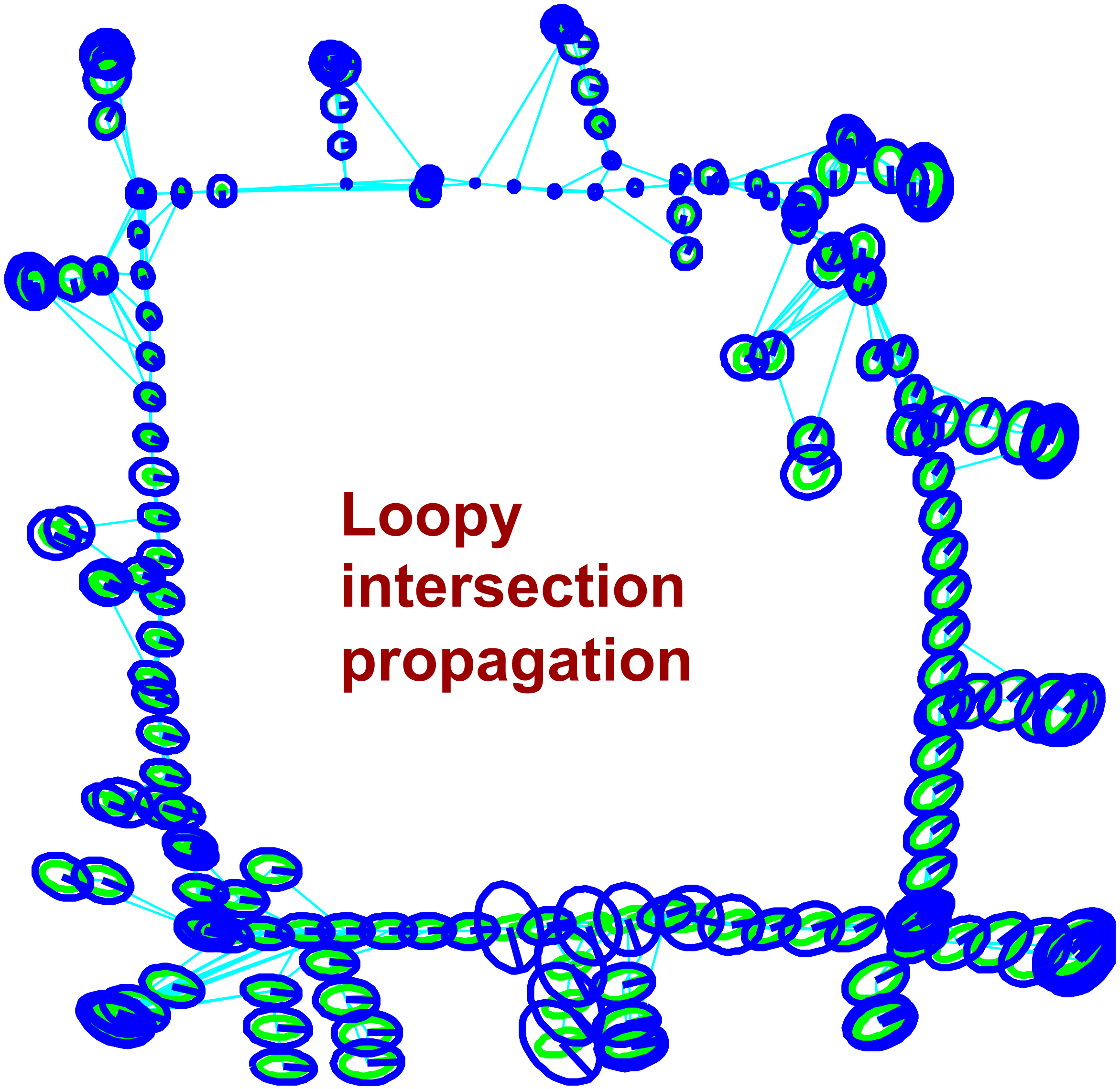
LIP – Algorithm

1. Compute a spanning tree
2. Run belief propagation on the tree
3. For every off-tree edge
    1. compute the off-tree estimates,
    2. compute the new priors, and
    3. delete the edge
4. Re-run belief propagation

Results


Loopy belief propagation
Overconfident
Spanning tree belief propagation
Too conservative

Results



Conclusions

- TORO - Efficient maximum likelihood algorithm for 2D and 3D graphs of poses
- No covariance estimates!
- Approach for recovering the covariance matrices via belief propagation and covariance intersection
  - Linear time complexity
  - Tight estimates
  - Generally conservative (not guaranteed!)