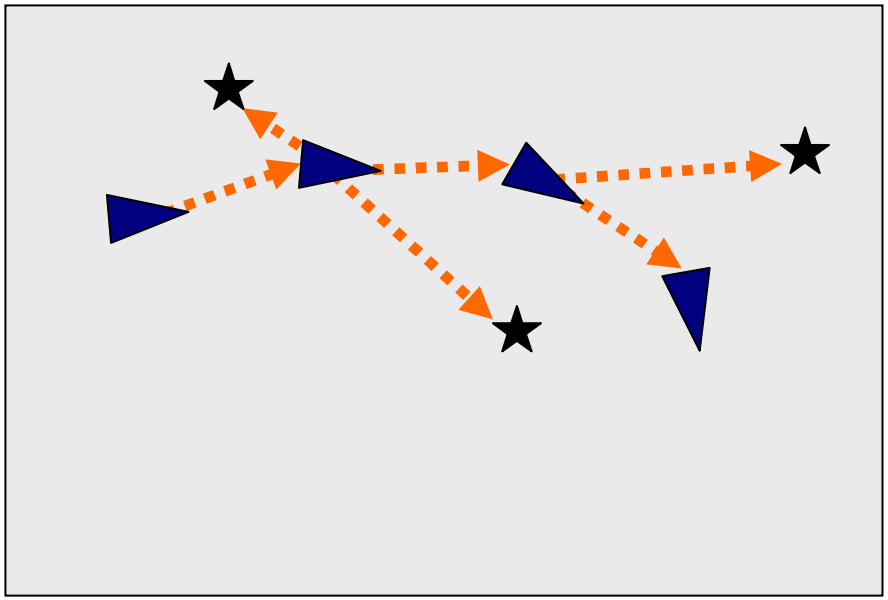
Robotics 2
TORO - Efficient Constraint Network Optimization for SLAM

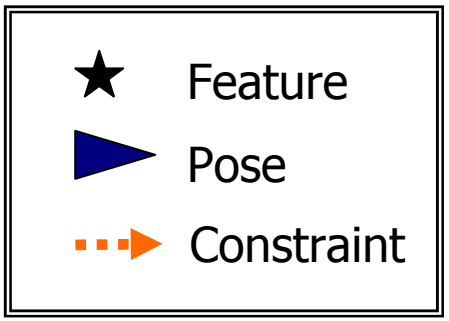
Giorgio Grisetti, Cyrill Stachniss,

Kai Arras, Wolfram Burgard

Robot Mapping

- Constraints connect the poses of the robot while it is moving (odometry).



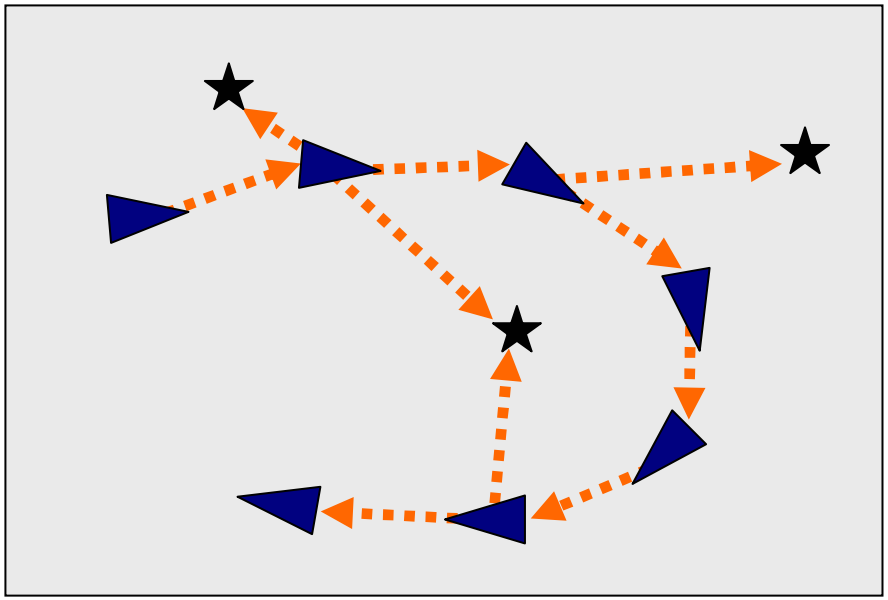


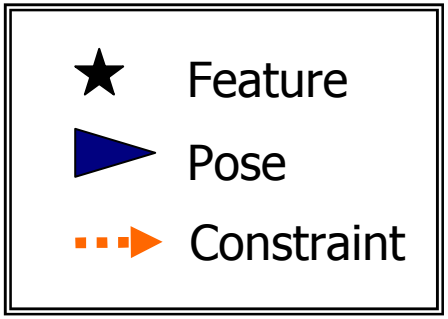
State: x, y, yaw

or x, y, z, roll, pitch, yaw

Robot Mapping

- Re-observing features defines constraints between non-successive poses.





State: x, y, yaw

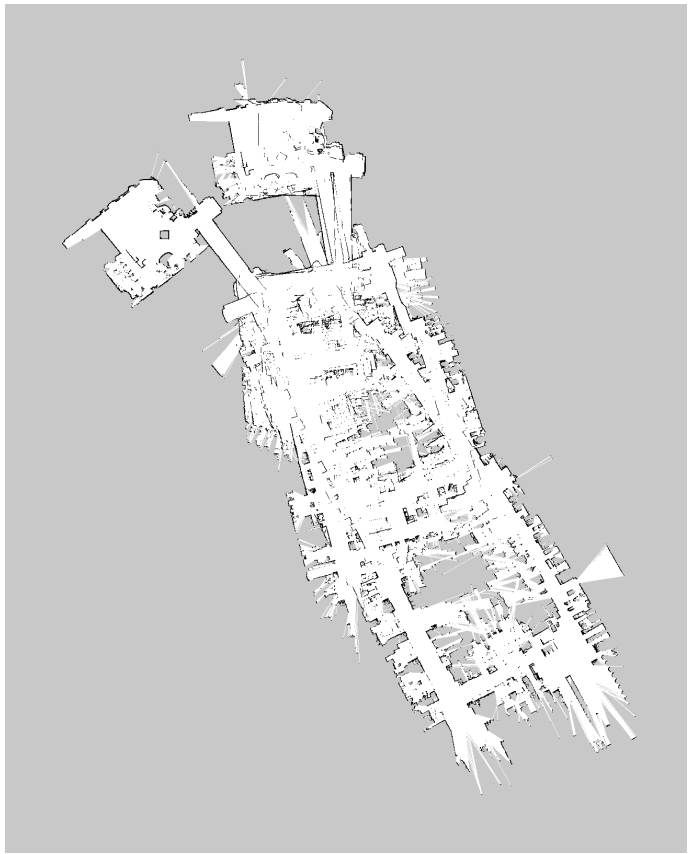
or x, y, z, roll, pitch, yaw

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

Problem Summary

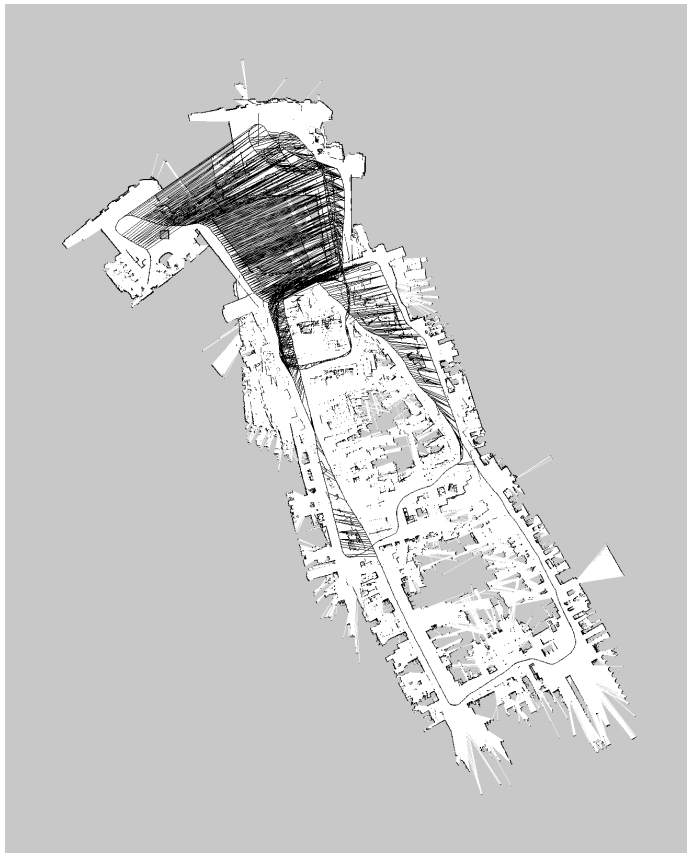
- **Goal:** Find the arrangement of the nodes that satisfies the constraints best



An initial configuration (KUKA production hall 22)

Problem Summary

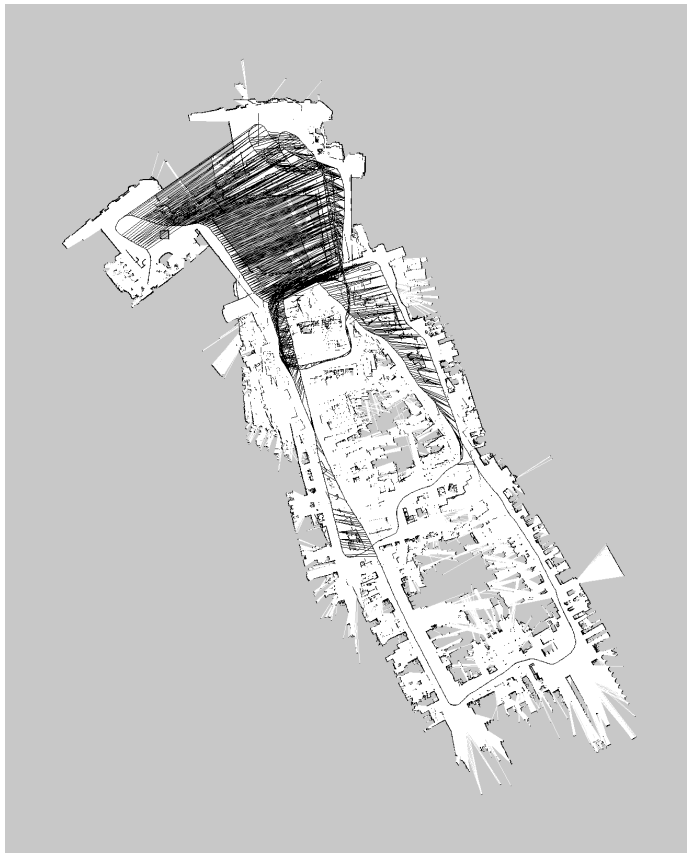
- **Goal:** Find the arrangement of the nodes that satisfies the constraints best



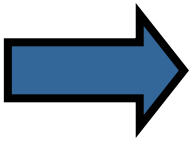
An initial configuration (KUKA production hall 22)

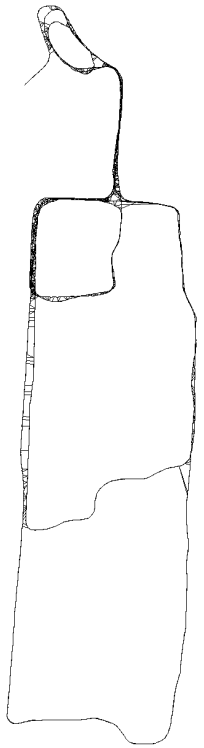
Problem Summary

- **Goal:** Find the arrangement of the nodes that satisfies the constraints best



An initial configuration

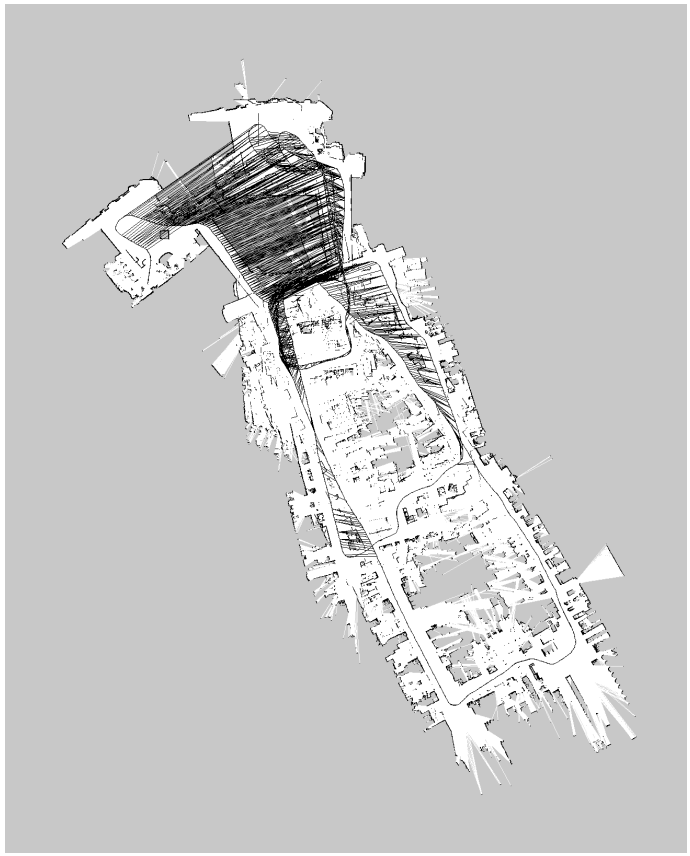




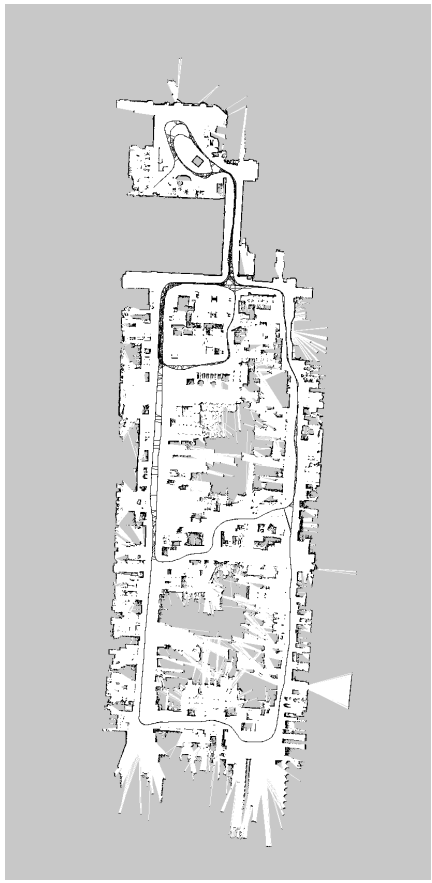
Maximum likelihood configuration

Problem Summary

- **Goal:** Find the arrangement of the nodes that satisfies the constraints best

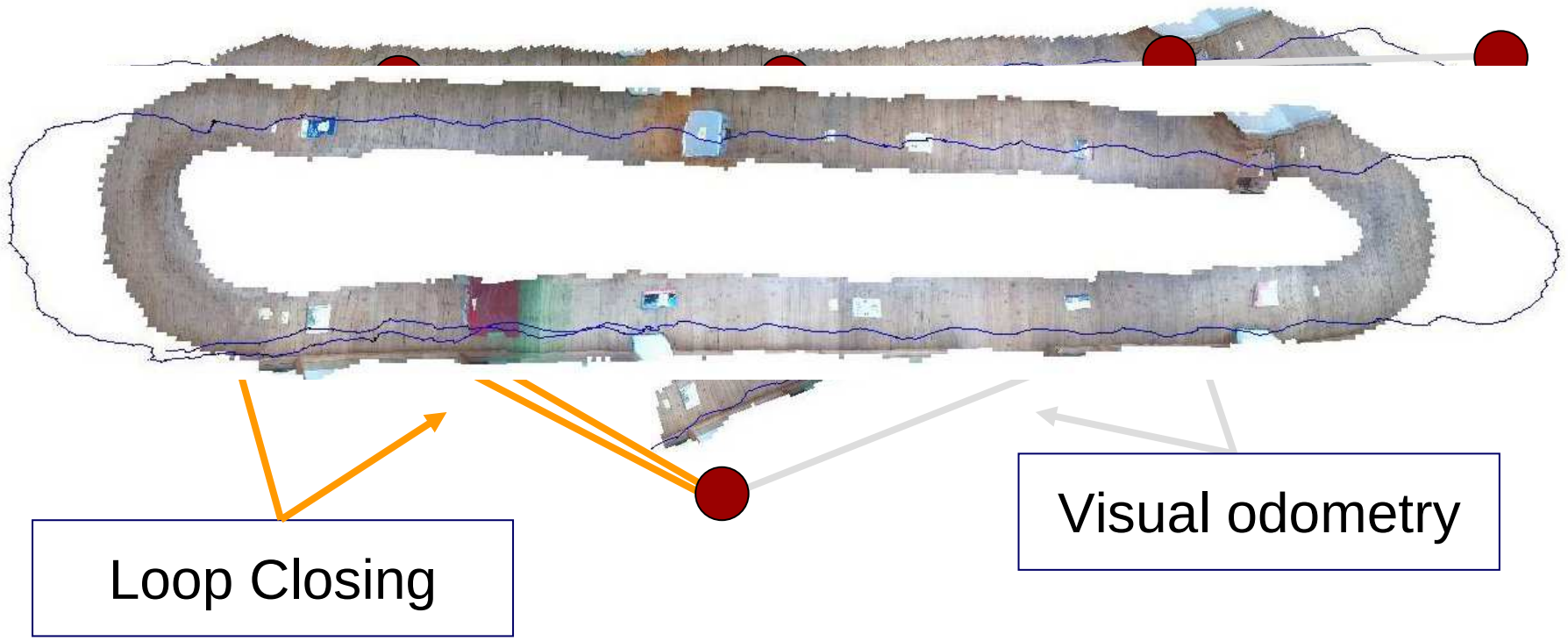


An initial configuration



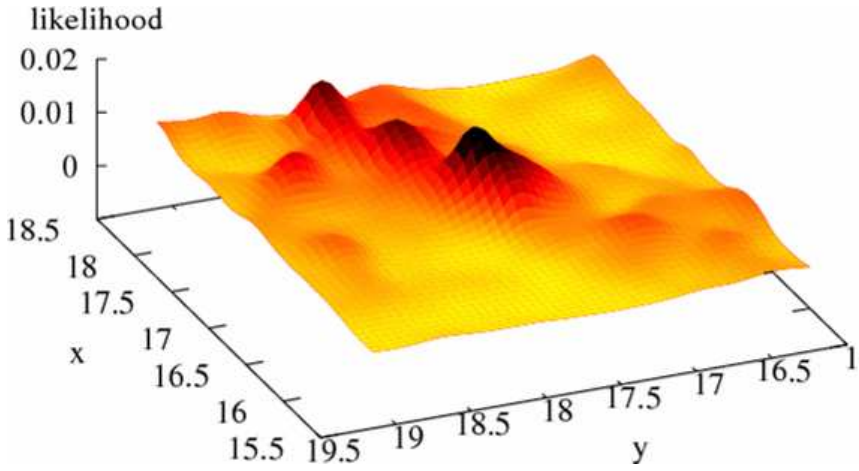
Maximum likelihood configuration

Graph-based Visual SLAM



Why is This Hard?

- Huge number of unknowns
  - |x| = 10^3 is a *small* problem
- Non-linear constraints
  - Function of the orientation of the robot
- Bad initial estimates
  - Odometry (if at all)
- Local minima
  - in the observation likelihood $p(z \mid x)$
- Possibility to incremental computation the solution



Topics Today

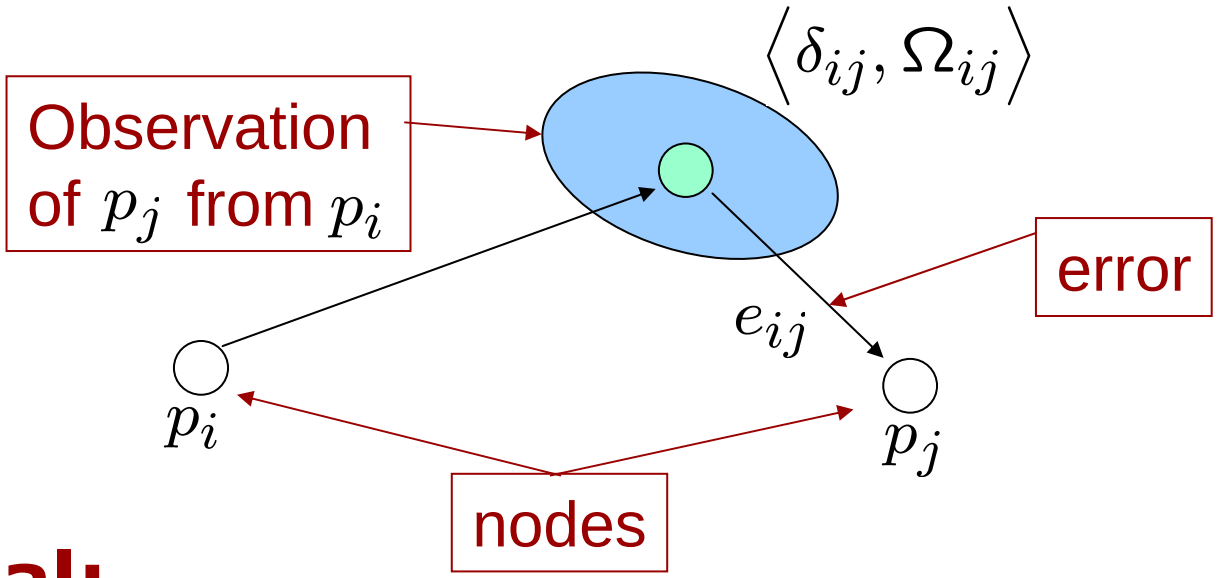
- Estimate the Gaussian posterior about the poses of the robot (full SLAM)

Three Parts:

- Estimate the means via non-linear optimization (maximum likelihood map)
- Estimate the covariance matrices via belief propagation and covariance intersection
- Examples of how to obtain constraints

Problem Formulation

- The problem can be described by a graph



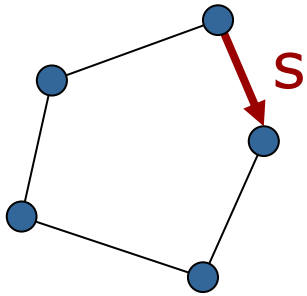
Goal:

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

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

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

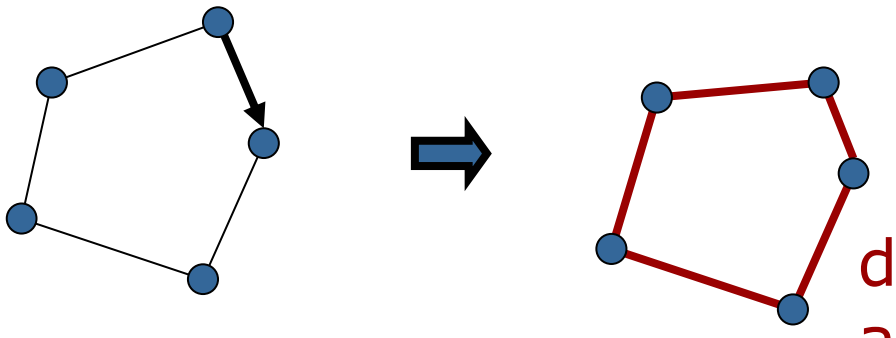


selected constraint

[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

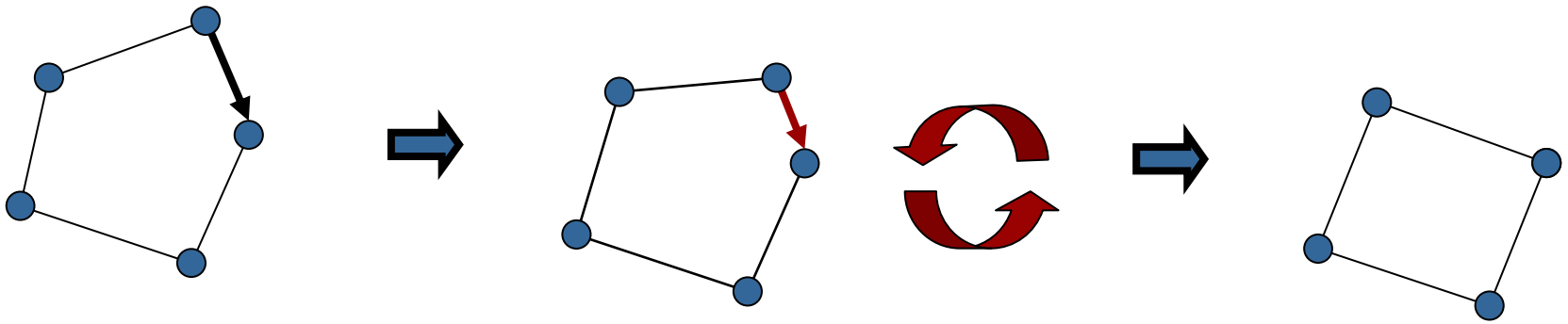


distribute the error over a set of involved nodes

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

Stochastic Gradient Descent

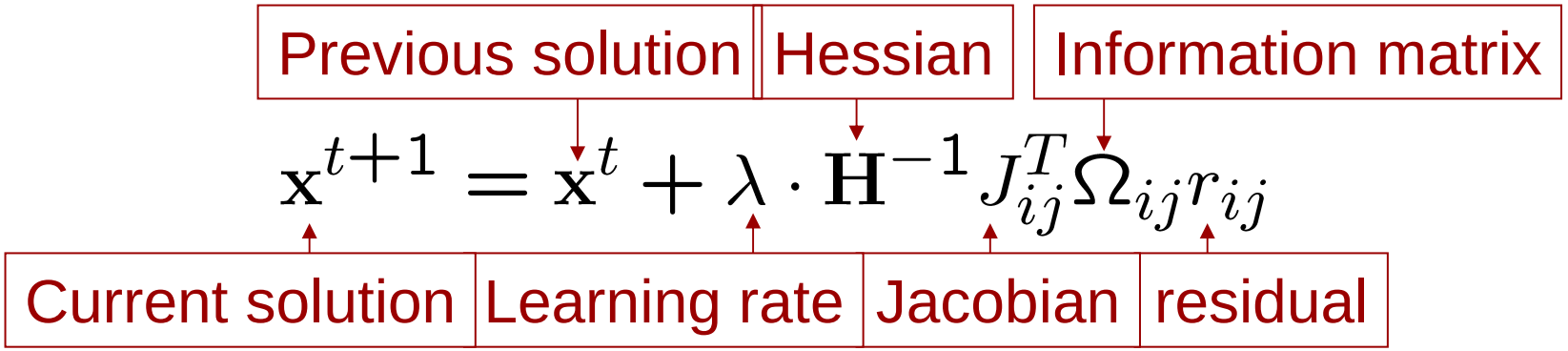
- Minimize the error individually for each constraint (decomposition of the problem into sub-problems)
- Solve one step of each sub-problem
- Solutions might be contradictory
- The magnitude of the correction decreases with each iteration
- Learning rate to achieve convergence



[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:



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

Node Parameterization

- How to represent the nodes in the graph?
- **Key question: 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 easier and faster.

parameters

poses

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

Mapping function

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

transformed problem

Parameterization of Olson

- Incremental parameterization:

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

parameters | poses

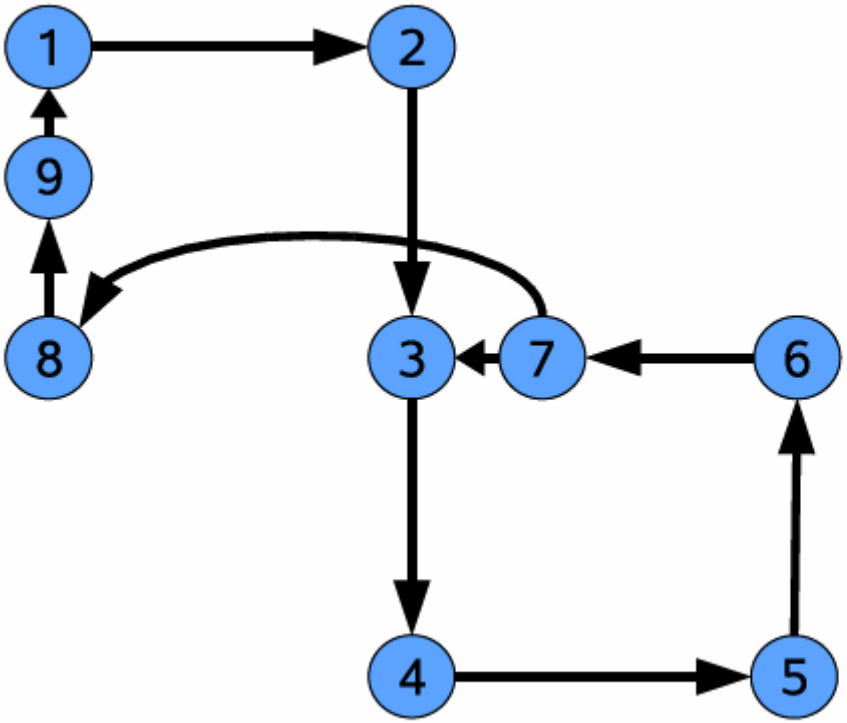
- Results directly from the trajectory takes by the robot
- Problem: for optimizing a constraint between node *i* and *k*, one needs to updates all *j* = *i*, …, *k* nodes ignoring the topology of the environment

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

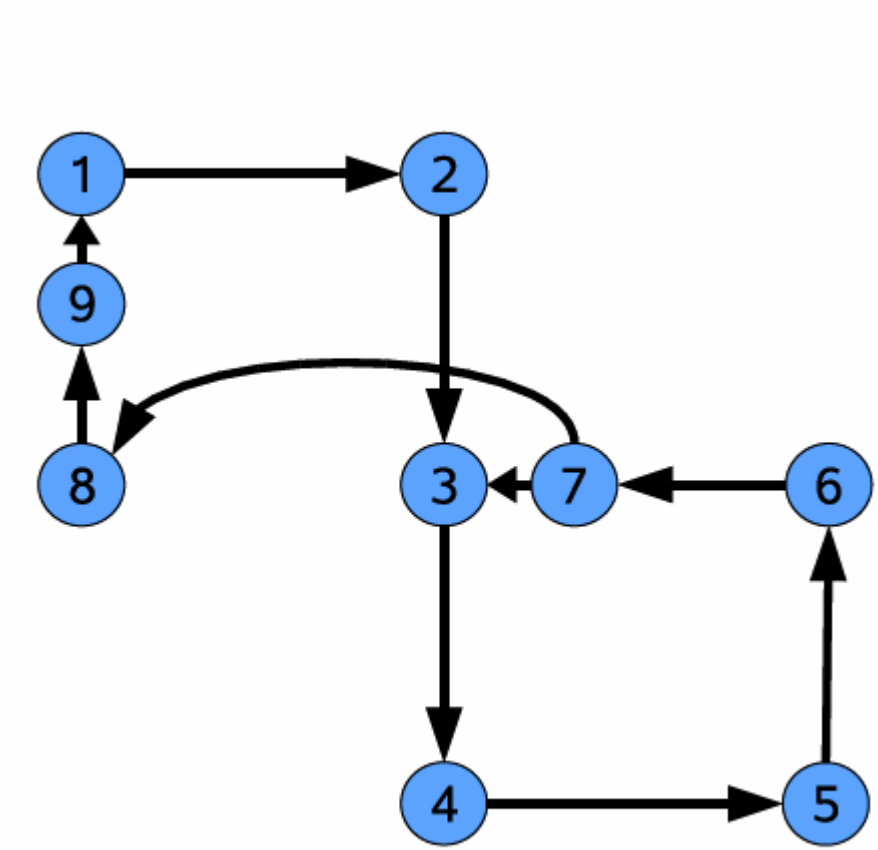
Tree Parameterization

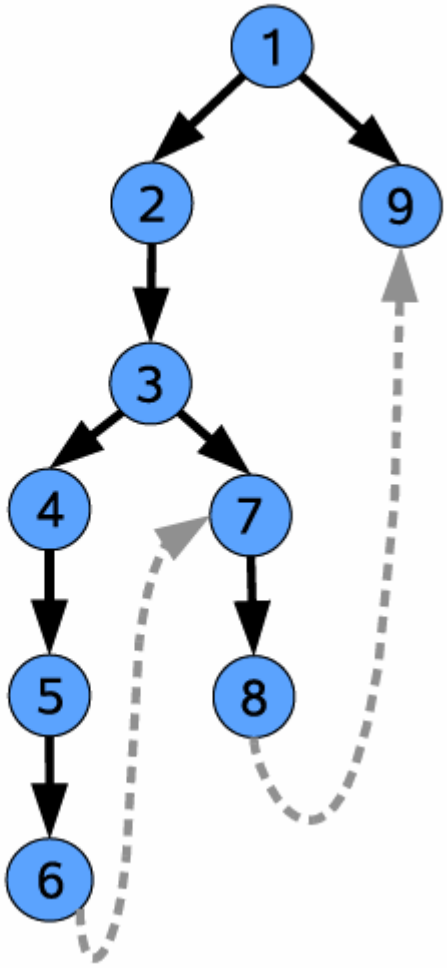
- How should such a problem decomposition look like?



Tree Parameterization

- Use a spanning tree!



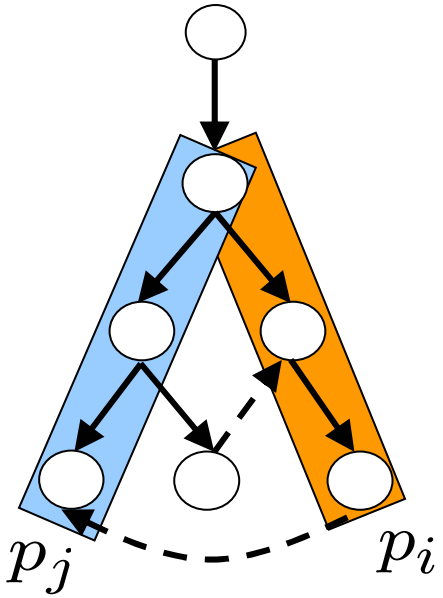


Tree Parameterization

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

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

- Error of a constraint in the new parameterization.

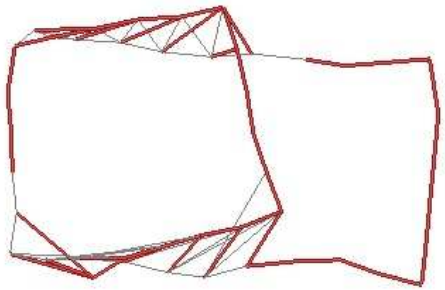


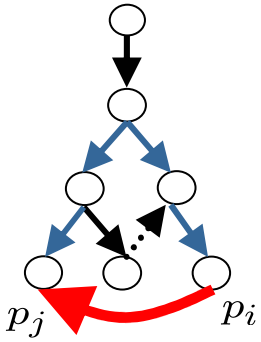
$$E_{ij} = \Delta_{ij}^{-1} \cdot \text{UpChain}^{-1} \cdot \text{DownChain}$$

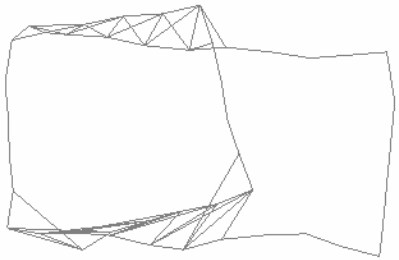
Only variables in the path of a constraint are involved in the update.

Stochastic Gradient Descent using the Tree Parameterization

- Using a tree parameterization, we decompose the problem in many small sub-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

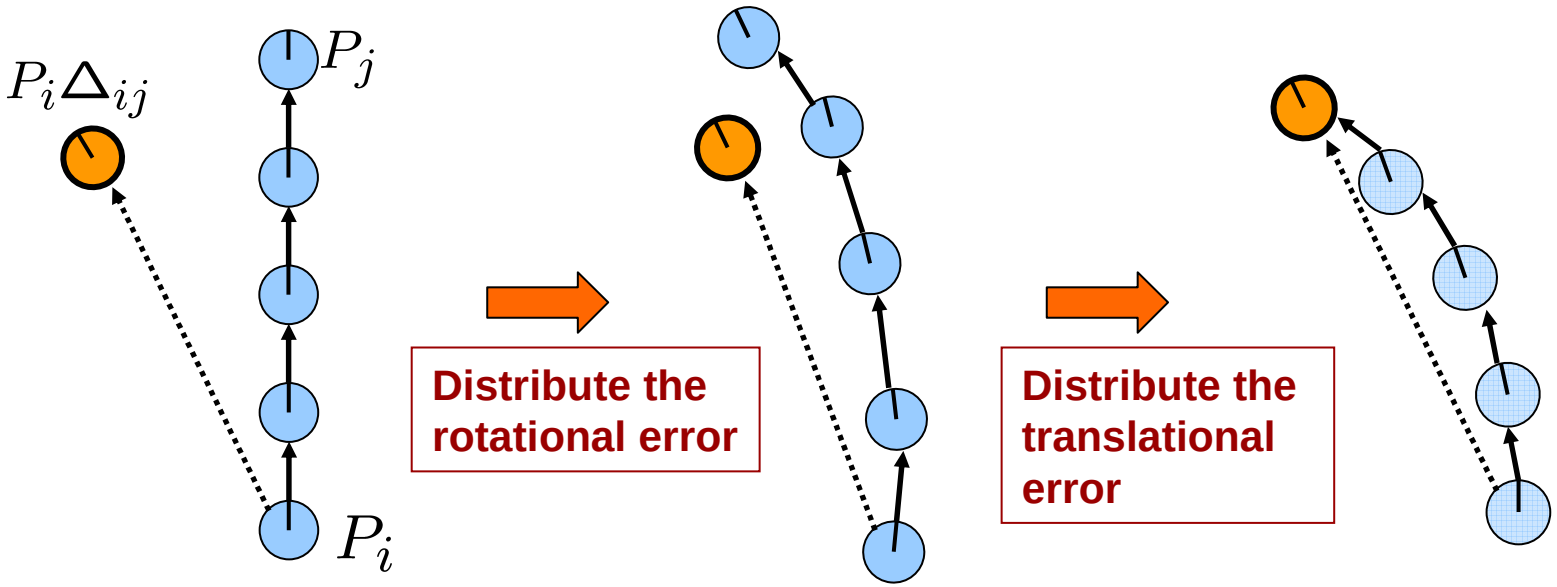
- 3D rotations lead to a highly nonlinear system.
  - Update the poses directly according to the SGD equation may lead to poor convergence.
  - This effect increases with the connectivity of the graph.
- Key idea in the SGD update:

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

Distribute a fraction of the residual along the parameters so that the error of that constraint is reduced.

Computation of the Update Step

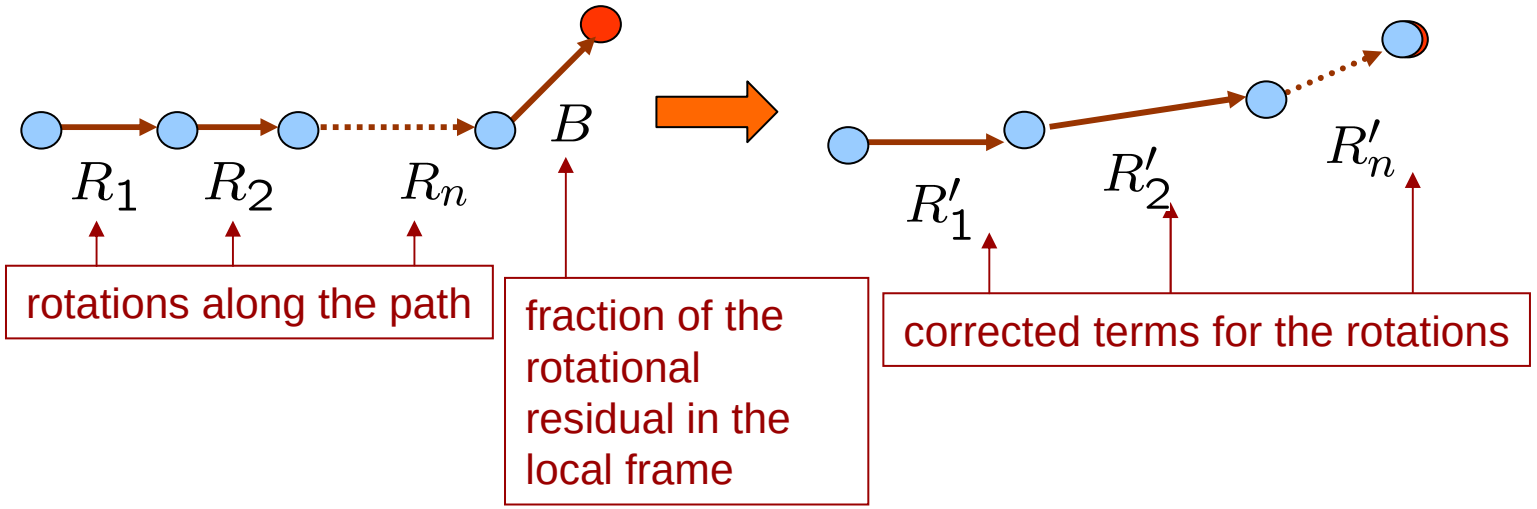
Update in the "spirit" of the SGD:
Smoothly deform the path along the constraints so that the error is reduced.



Distribution of the Rotational Error

- In 3D the rotational error cannot be simply added to the parameters because the rotations are not commutative.
- Our goal is to 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)

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

- 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

- To obtain the rotations, we recursively solve:

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

- With this update rule, it can be shown that the change in each rotational residual is bounded

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

- 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

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

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

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

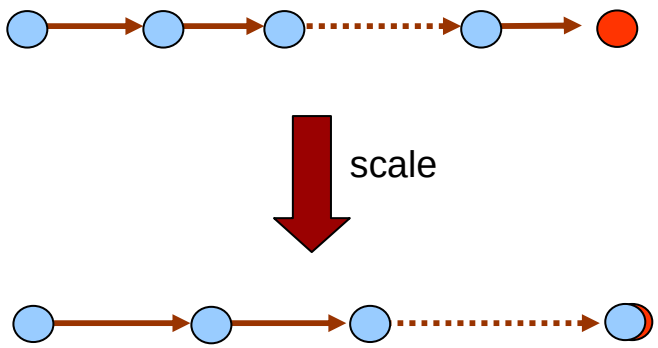
$$d_m = \sum_{\langle l,m \rangle} \min\left[\text{eigen}(\Omega_{lm})\right]$$

all constraints connecting m

- This assumes roughly spherical covariances!

Distributing the Translation Part of the Residual

- That is trivial
- Just scale the x, y, z dimension (with covariances)



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)

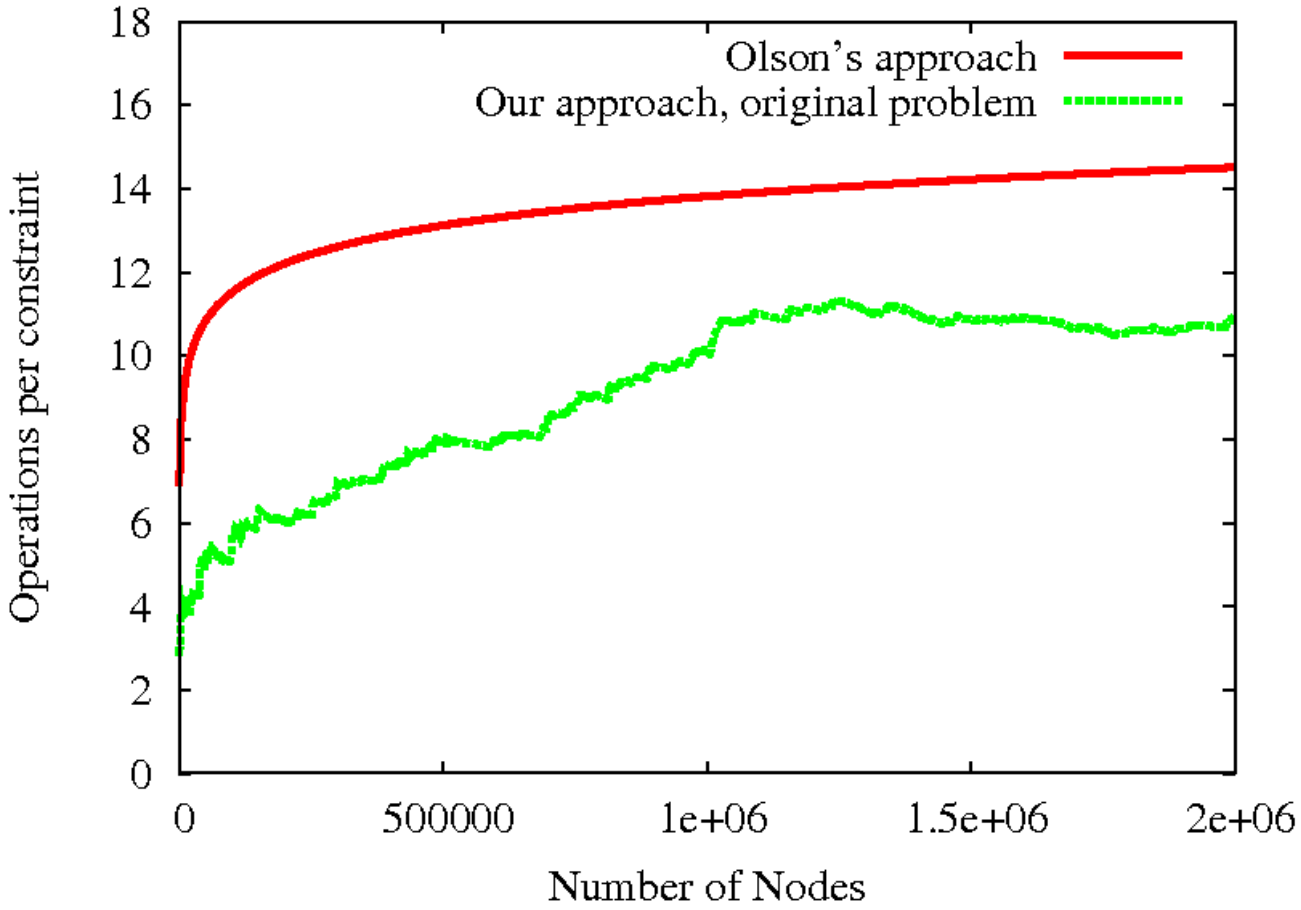
- This results in a complexity per iteration of

$$\mathcal{O}(M \cdot l)$$

#constrains

avg. path length
(parameterization tree)

Cost of a Constraint Update



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

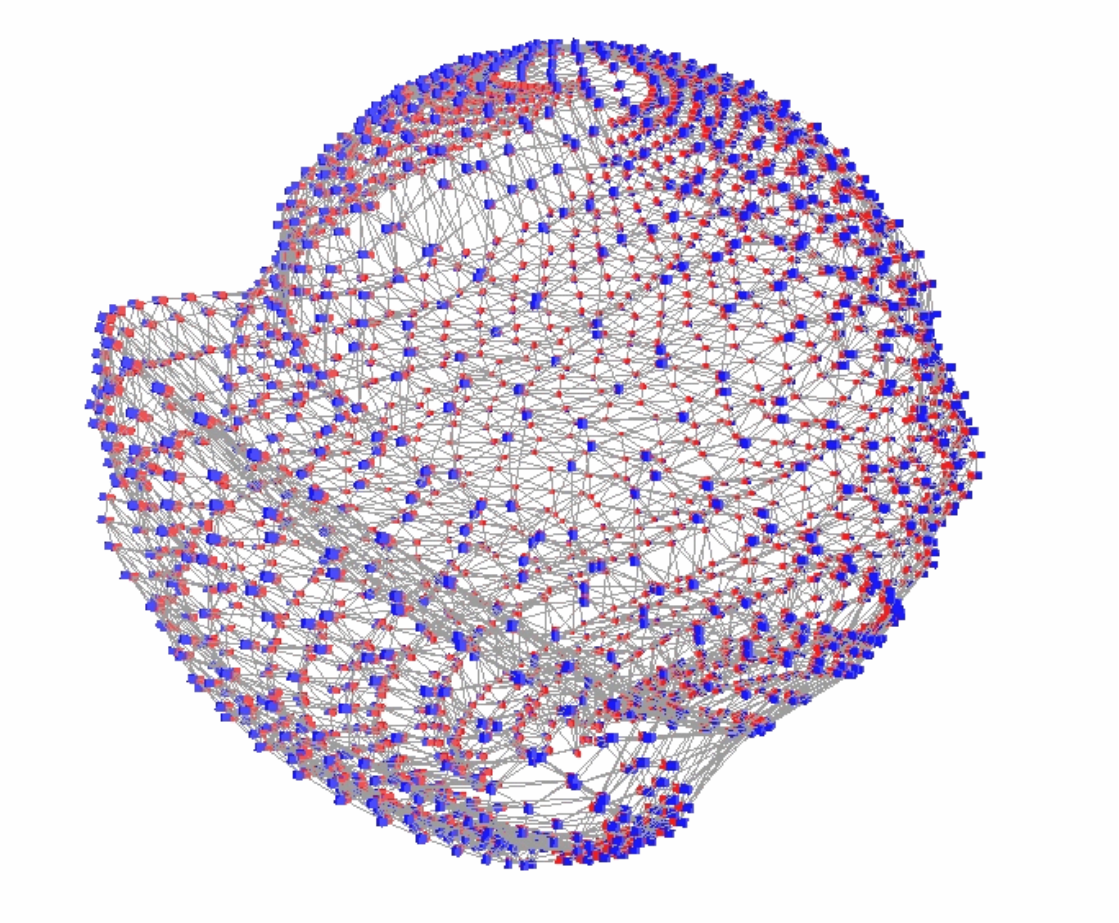
Node Reduction

- The complexity grows with the length of the trajectory
- That is bad for life-long learning!
- Idea: Combine constraints between nodes if the robot is well-localized

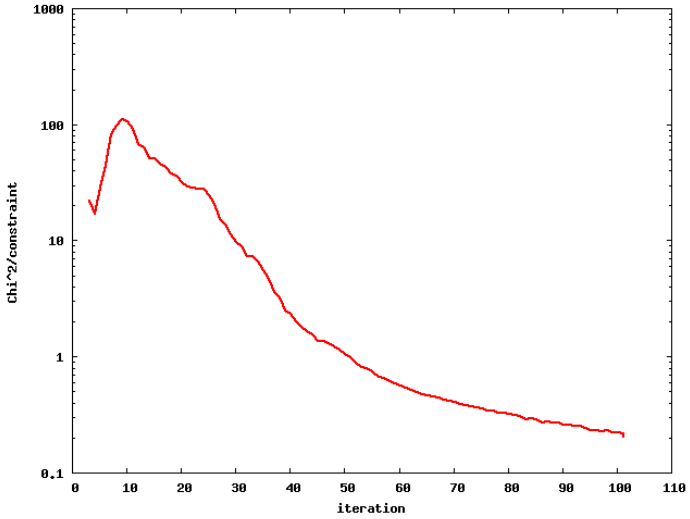
$$\begin{aligned} \Omega_{ij} &= \Omega_{ij}^{(1)} + \Omega_{ij}^{(2)} \\ \delta_{ij} &= \Omega_{ij}^{-1}(\Omega_{ij}^{(1)}\delta_{ij}^{(1)} + \Omega_{ij}^{(2)}\delta_{ij}^{(2)}) \end{aligned}$$

- This is similar to adding a rigid constraints
- Valid approximation if the robot is well localized
- Complexity depends only on the size if the environment, not the length of the trajectory

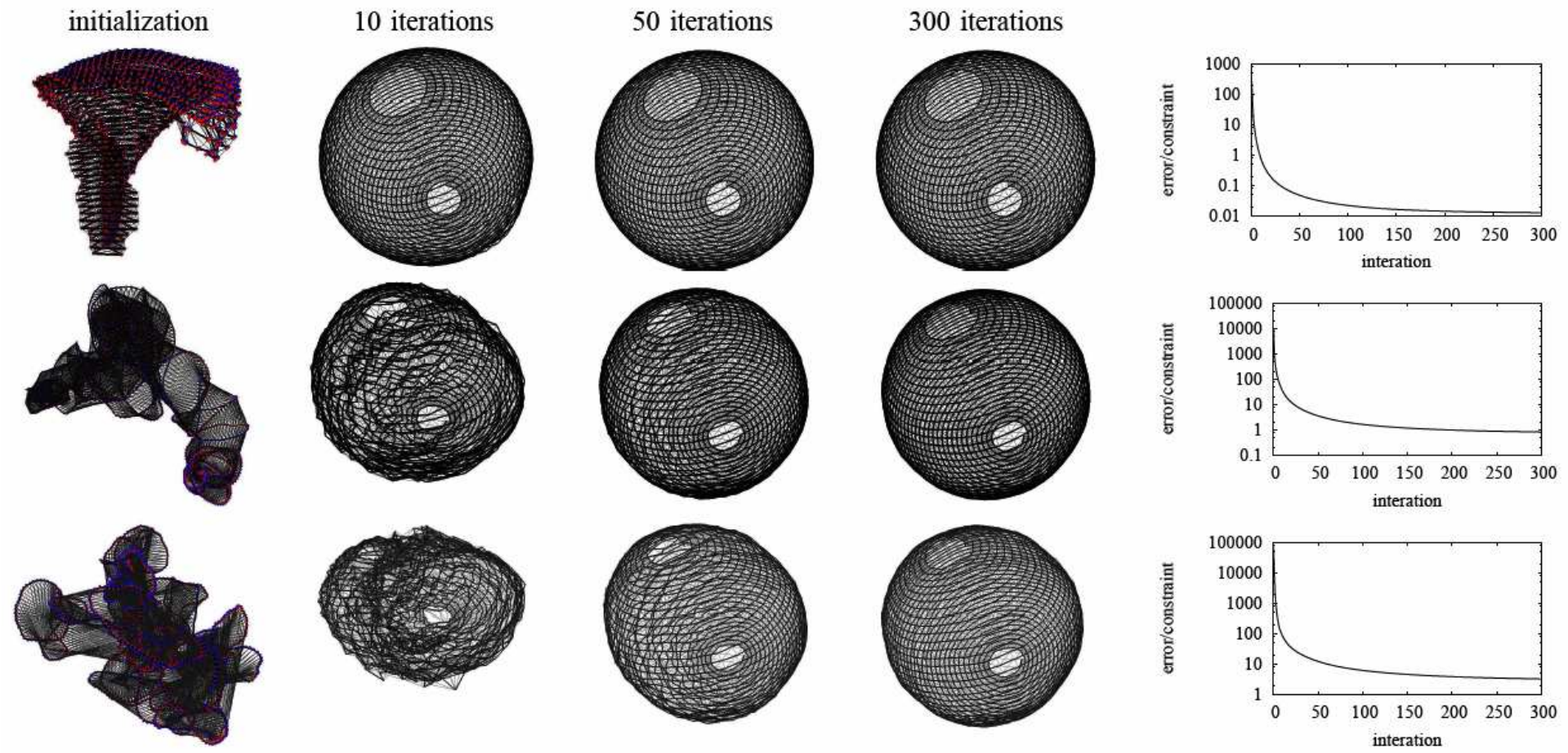
Simulated Experiment



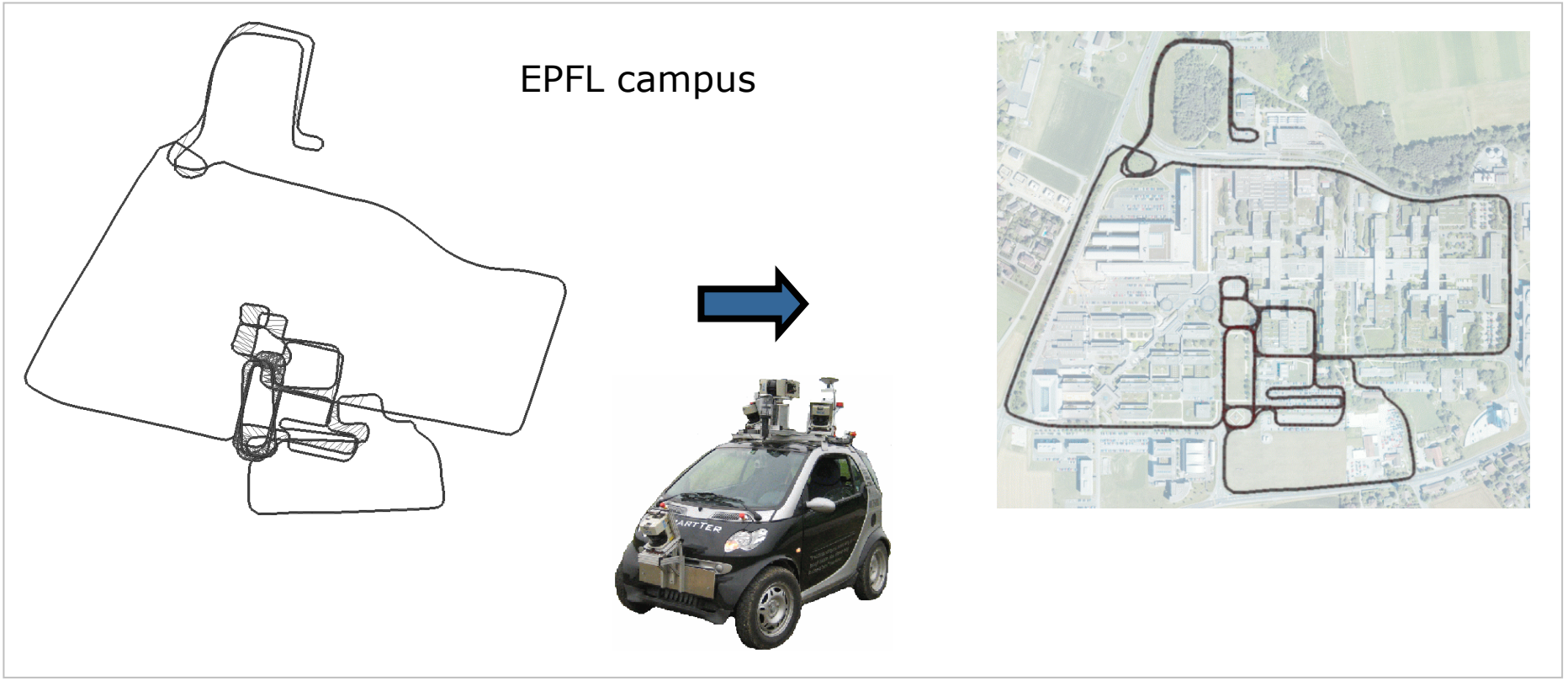
- Highly connected graph
- Poor initial guess
- LU & friends fail
- 2200 nodes
- 8600 constraints



Spheres with Different Noise

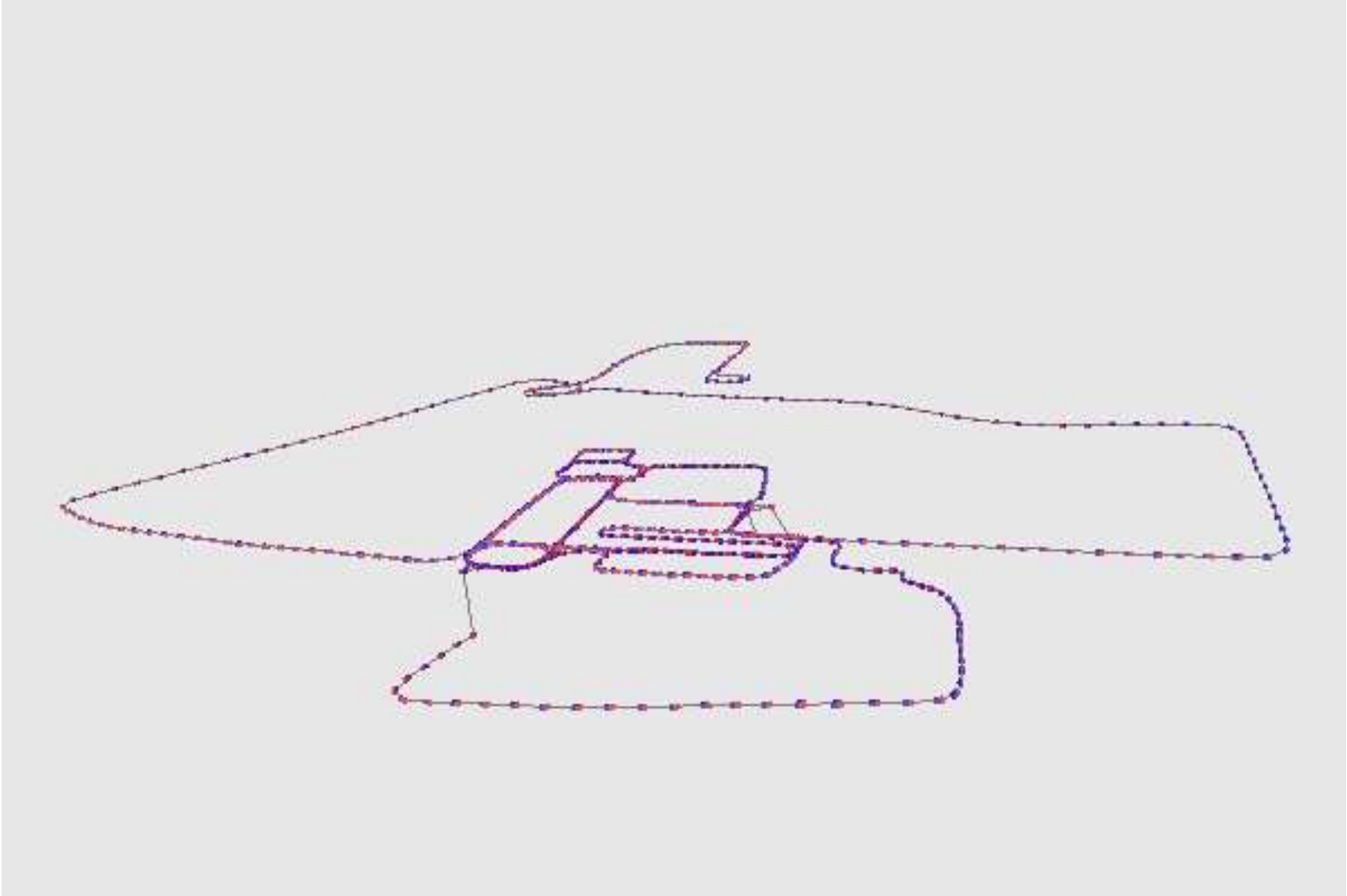


Mapping the EPFL Campus



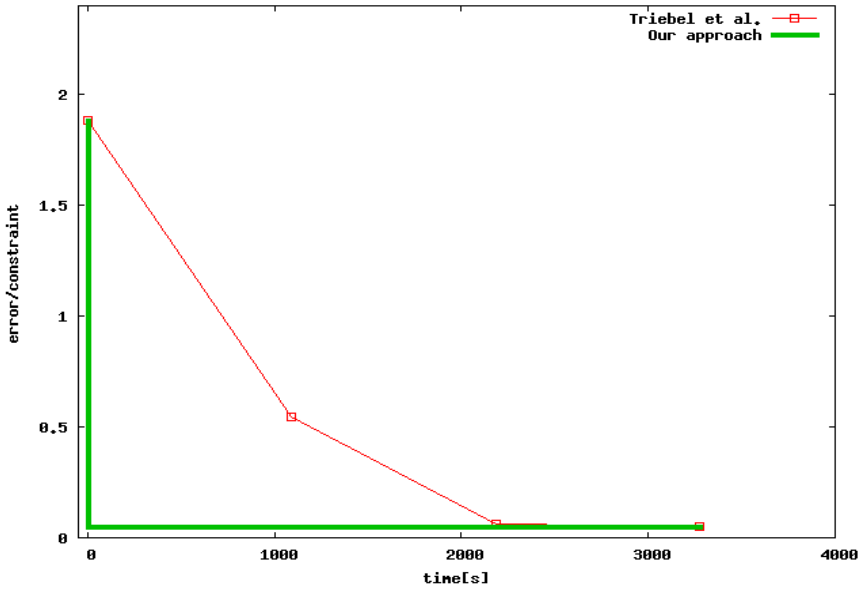
- 10km long trajectory and 3D lasers recorded with a car
- Problem not easily tractable by most standard optimizers

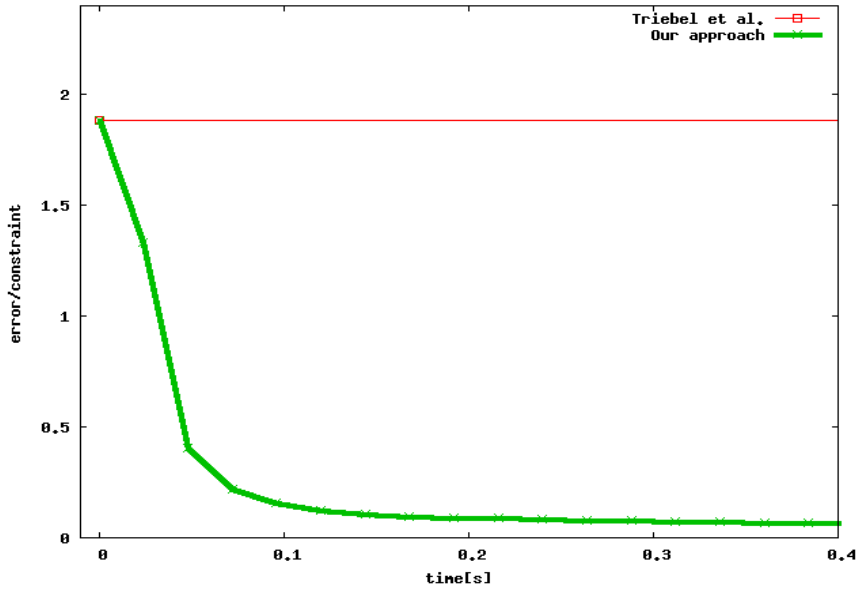
Mapping the EPFL Campus



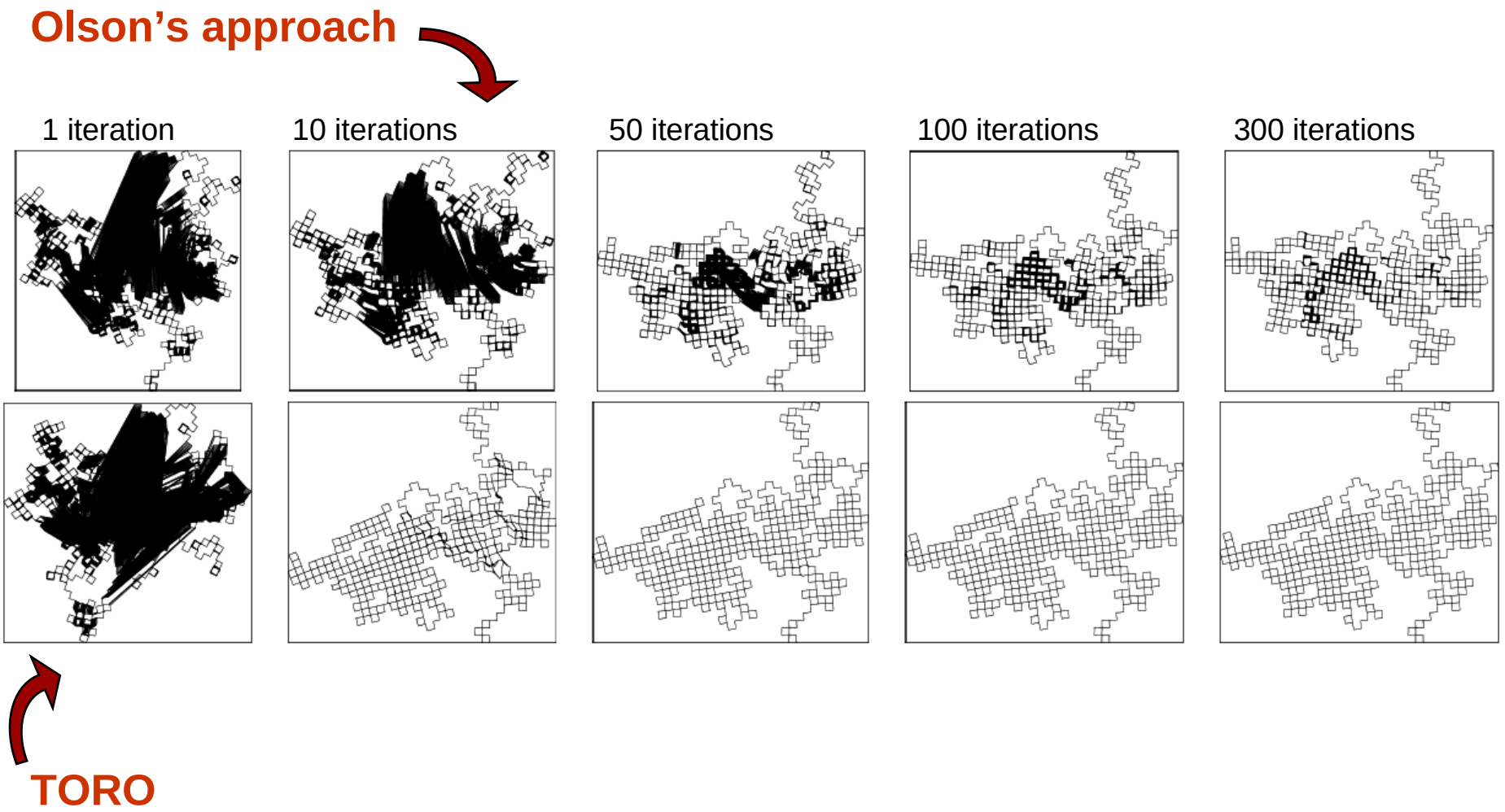
Comparison with Standard Approaches (LU Decomposition)

- Tractable subset of the EPFL dataset
- Optimization carried out in less than one second.
- The approach is so fast that in typical applications one can run it while incrementally constructing the graph.

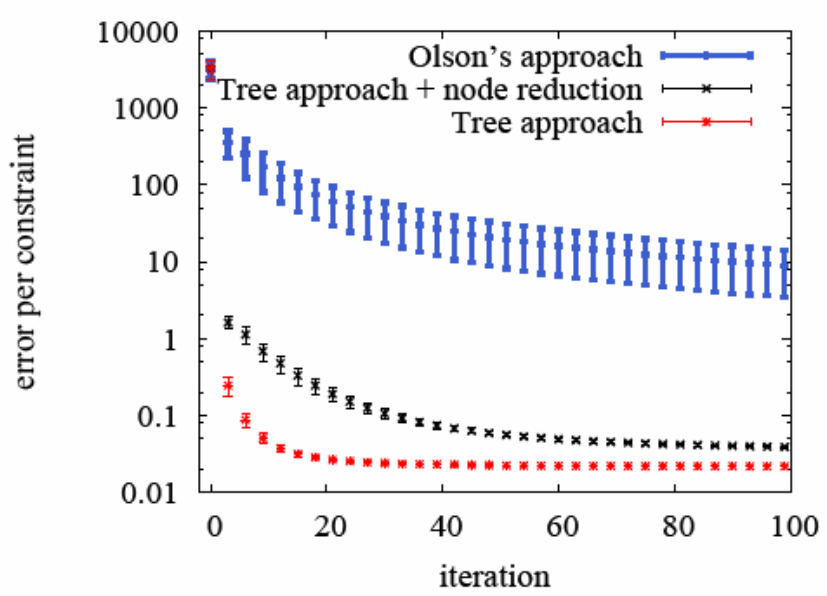


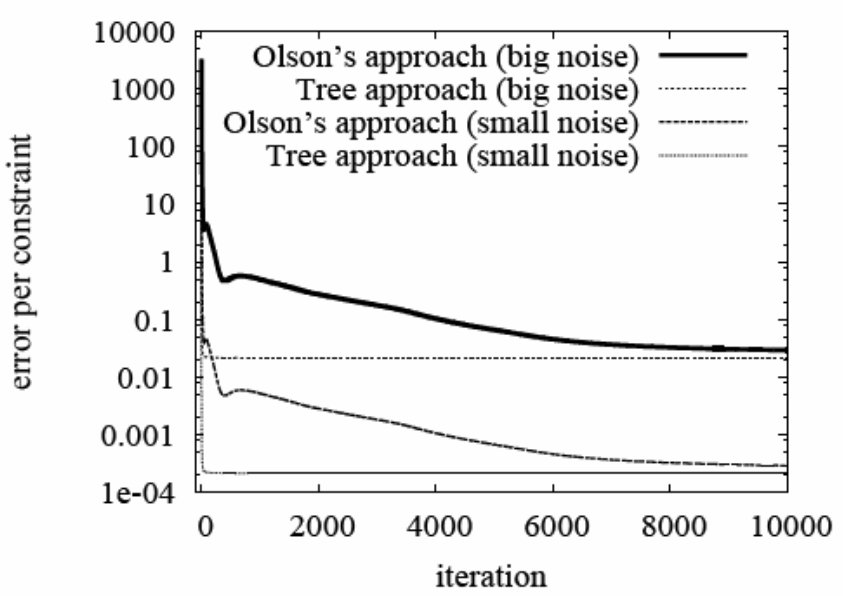


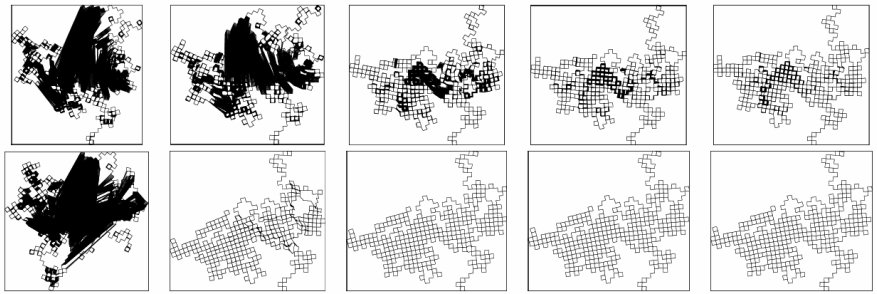
TORO vs. Olson's Approach



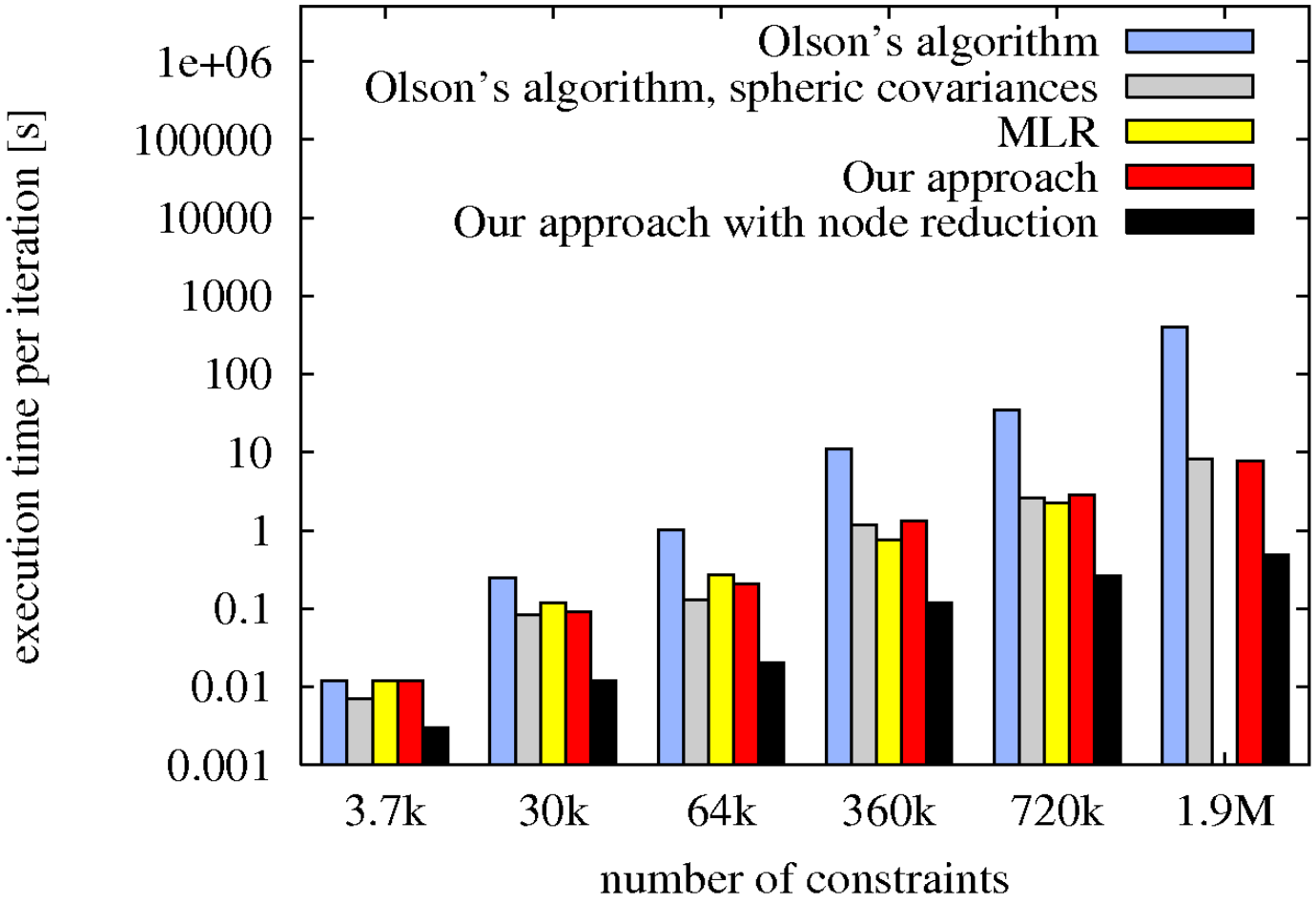
TORO vs. Olson's Approach





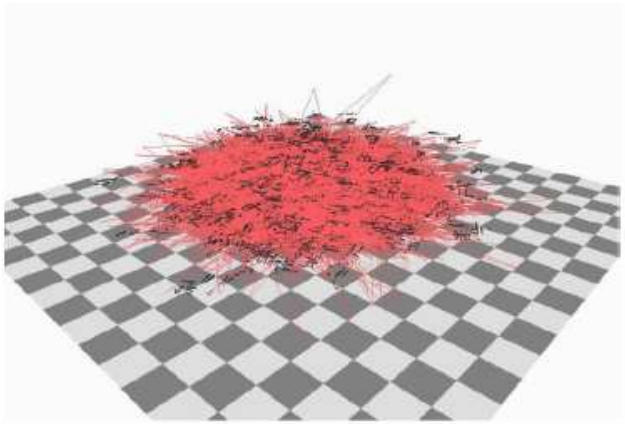


Time Comparison (2D)

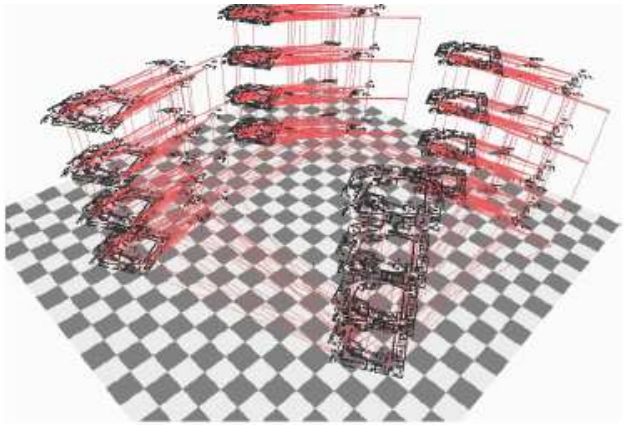


Robust to the Initial Guess

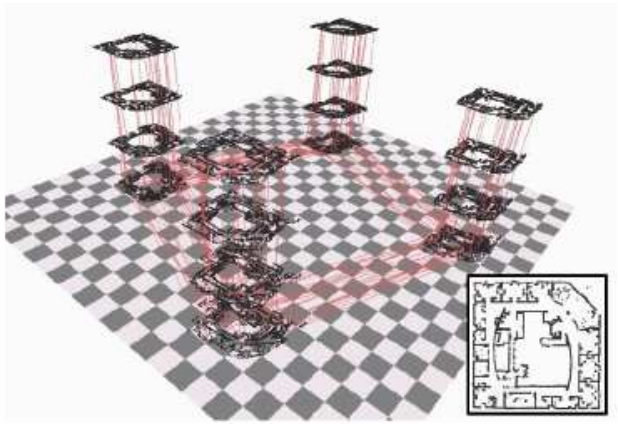
- Random initial guess
- Intel datatset as the basis for 16 floors distributed over 4 towers



initial configuration



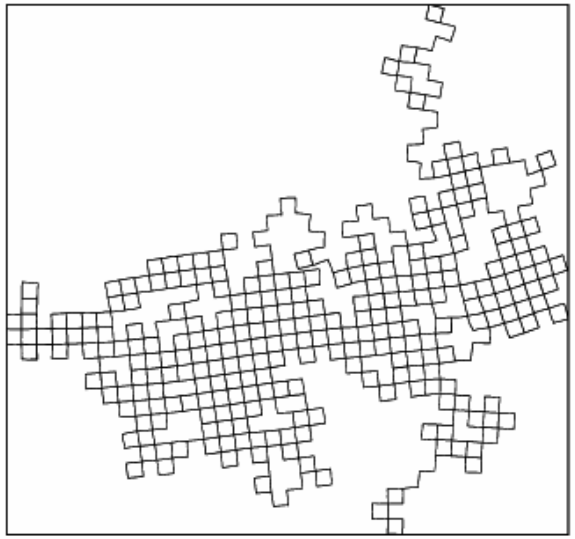
intermediate result



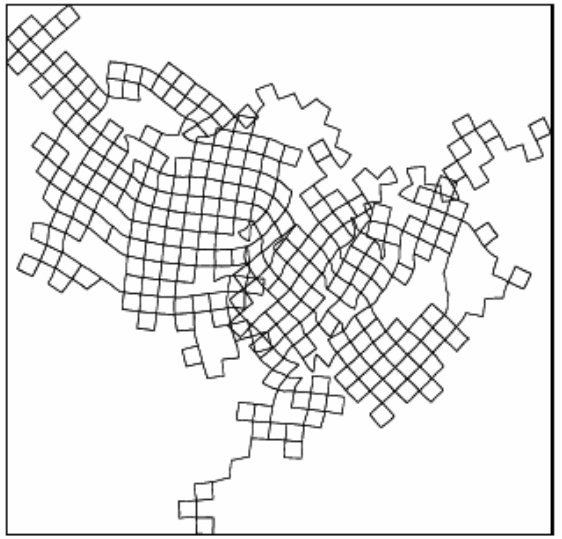
final result (50 iterations)

Informal Comparison to MLR

- More robust under the initial guess
- Much faster than MLR
- If MLR converges, it gives slightly lower error values



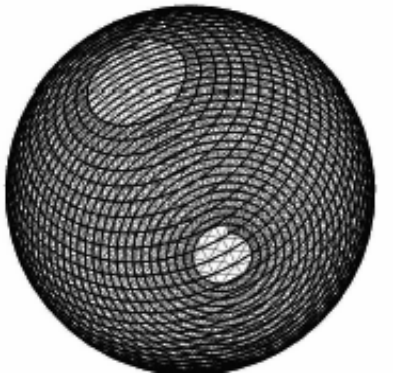
with a good initial guess



with a bad initial guess

Informal Comparison to (i)SAM

- More stable than SAM
- Results similar to iSAM
- Faster than SAM/iSAM



noise level	SAM (batch)	SAM (incremental)	Our method (batch)
$\sigma = 0.05$	119 s	not tested (see batch)	20 s (100 iterations)
$\sigma = 0.1$	diverged	270 s (optimized each 100 nodes)	40 s (200 iterations)
$\sigma = 0.2$	diverged	510 s (optimized each 50 nodes)	50 s (250 iterations)

TORO Summary

- Extremely robust to bad initial network configurations
- One of the most efficient techniques for ML map estimates that is available (~01/2008)
- Works in 2D and 3D
- Scales up to millions of constraints
- Allows for multi-vehicle mapping (with minor modifications)
- Used by different research groups/companies worldwide (U. Oxford, ETH Zurich, La Sapienza Rome, Willow Garage, KUKA, …)
- Download from 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]

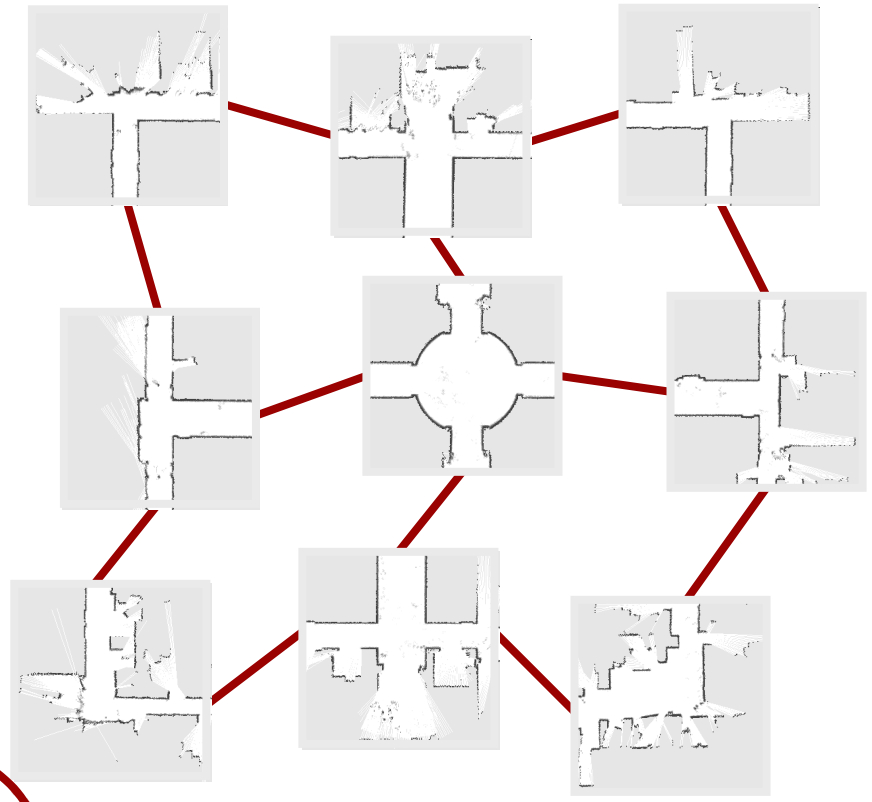
Data Association

- So far we explained how to compute the mean of the distribution given the data associations.
- However, to determine the data associations, we need to know the covariance matrices of the nodes.
- Standard approaches include:
  - Matrix inversion
  - Loopy belief propagation
  - Belief propagation on a spanning tree
  - **Loopy intersection propagation**

[Tipaldi et al., 07]

Graphical SLAM as a GMRF

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



$$p(x) = \frac{1}{Z} \prod_{i=1}^{n} \phi_i(x_i) \prod_{j=i+1}^{n} \phi_{i,j}(x_i, x_j)$$

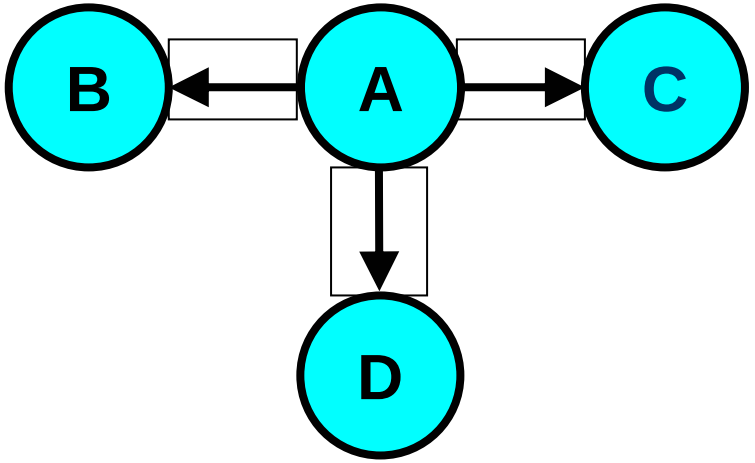
Gaussian in canonical form

Belief Propagation

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

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

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



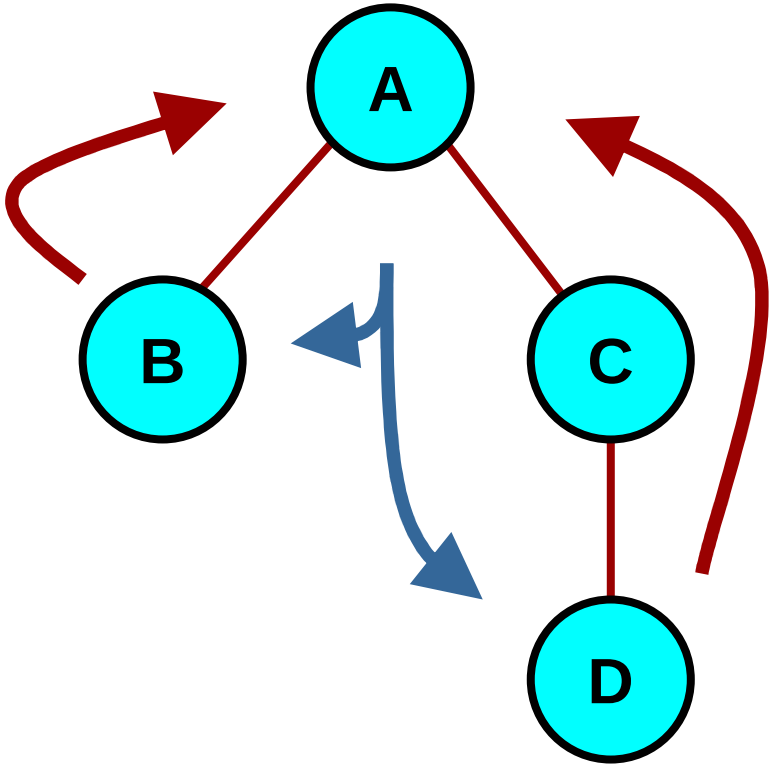
  - **Send** messages

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

$$\mathbf{M}_{ij}^{(t)} = \boldsymbol{\Omega}_{ij}^{[ji]} - \boldsymbol{\Omega}_{ij}^{[ji]} \left(\boldsymbol{\Omega}_{ij}^{[ii]} + \mathbf{M}_i^{(t)} - \mathbf{M}_{ji}^{(t-1)} \right)^{-1} \boldsymbol{\Omega}_{ij}^{[ij]}$$

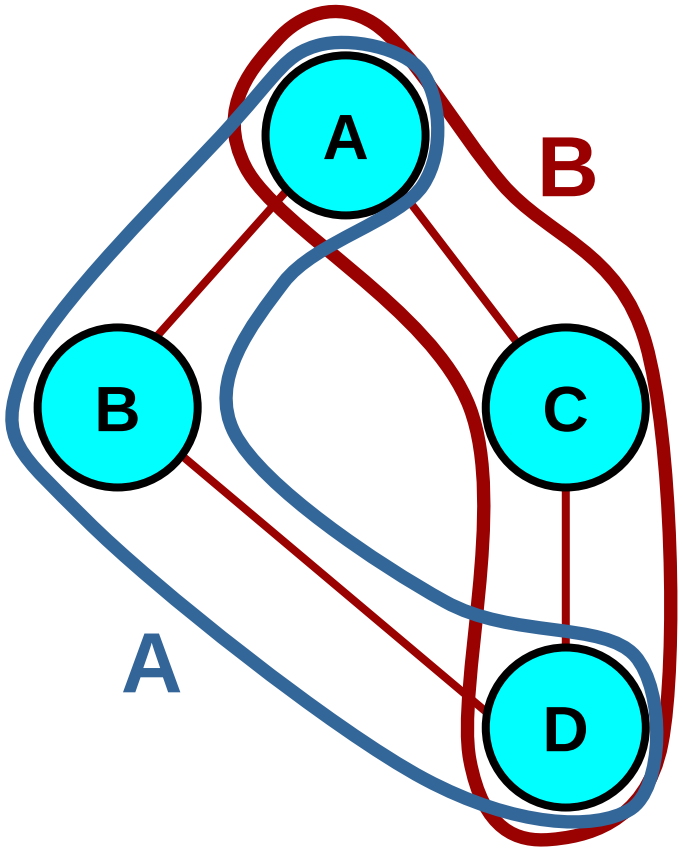
Ignore the math!

Belief Propagation - Trees



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

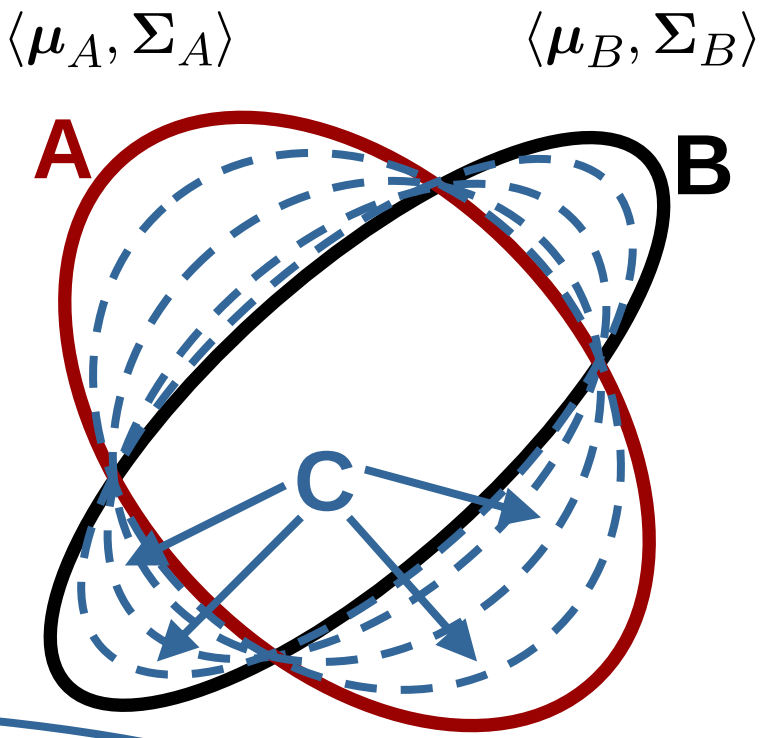
Belief Propagation – loops



- 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**



$$\Sigma_C = (\omega\Sigma_A^{-1} + (1-\omega)\Sigma_B^{-1})^{-1}$$
$$\mu_C = \Sigma_C(\omega\Sigma_A^{-1}\mu_A + (1-\omega)\Sigma_B^{-1}\mu_B)$$

Loopy Intersection Propagation

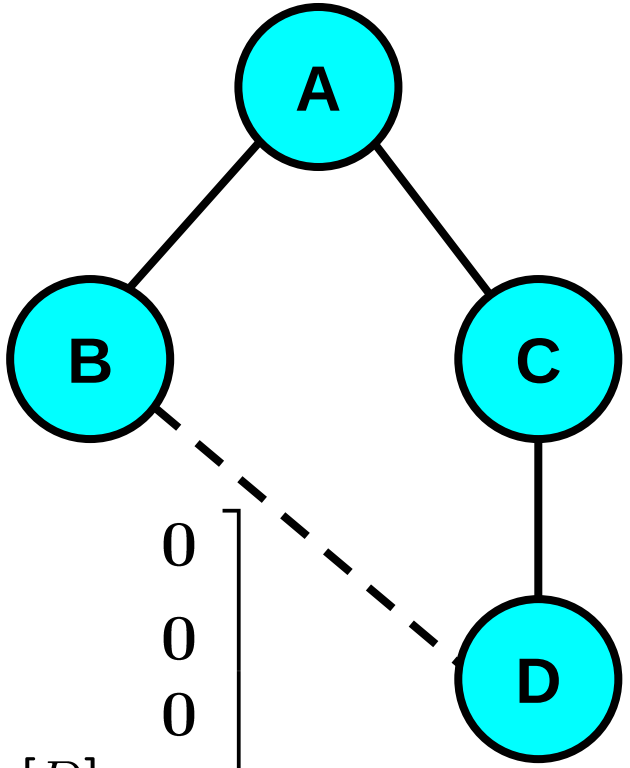
- **Key ideas**
  - Exact inference on a spanning tree of the graph
  - Augment the tree with information coming from loops
- **How**
  - Approximation by means of **cutting matrices**
  - Loop information within **local potentials** (priors)

Approximation via Cutting Matrix

- Removal as matrix subtraction

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

- Regular cutting matrix
- Cut our all off-tree edges



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

Fusing Loops with Spanning Trees

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

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

Ignore the math!

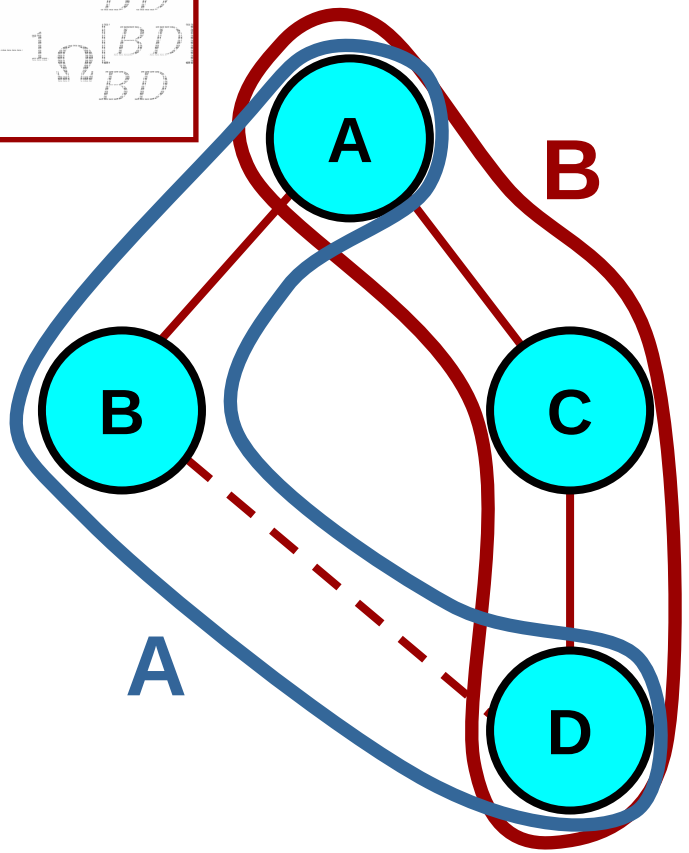
- Fuse the estimates

$$\hat{\mathbf{M}}_B = \omega_B \mathbf{M}_B + (1-\omega_B)\mathbf{E}_{BD}^{[B]}$$
$$\hat{\mathbf{M}}_D = \omega_D \mathbf{M}_D + (1-\omega_D)\mathbf{E}_{BD}^{[D]}$$

Ignore the math!

- Compute the priors

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



LIP – Algorithm

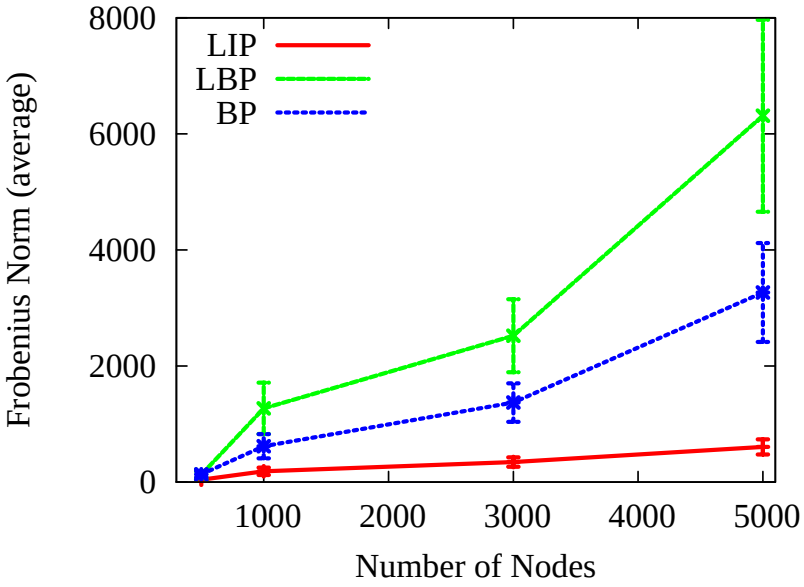
1. Compute a spanning tree
2. Run belief propagation on the tree
3. For every off-tree edge
    1. compute the off-tree estimates,
    2. compute the new priors, and
    3. delete the edge
4. Re-run belief propagation

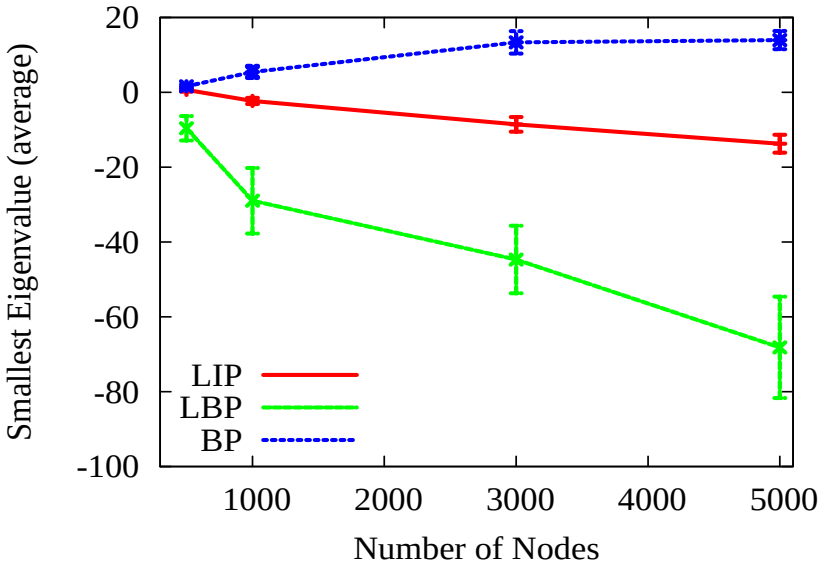
Experiments – Setup & Metrics

- Simulated data
  - Randomly generated networks of different sizes
- Real data
  - Graph extracted from Intel and ACES dataset from radish
- Approximation error
  - Frobenius norm
- Conservativeness
  - Smallest eigenvalue of matrix difference

Experiments – Simulated Data

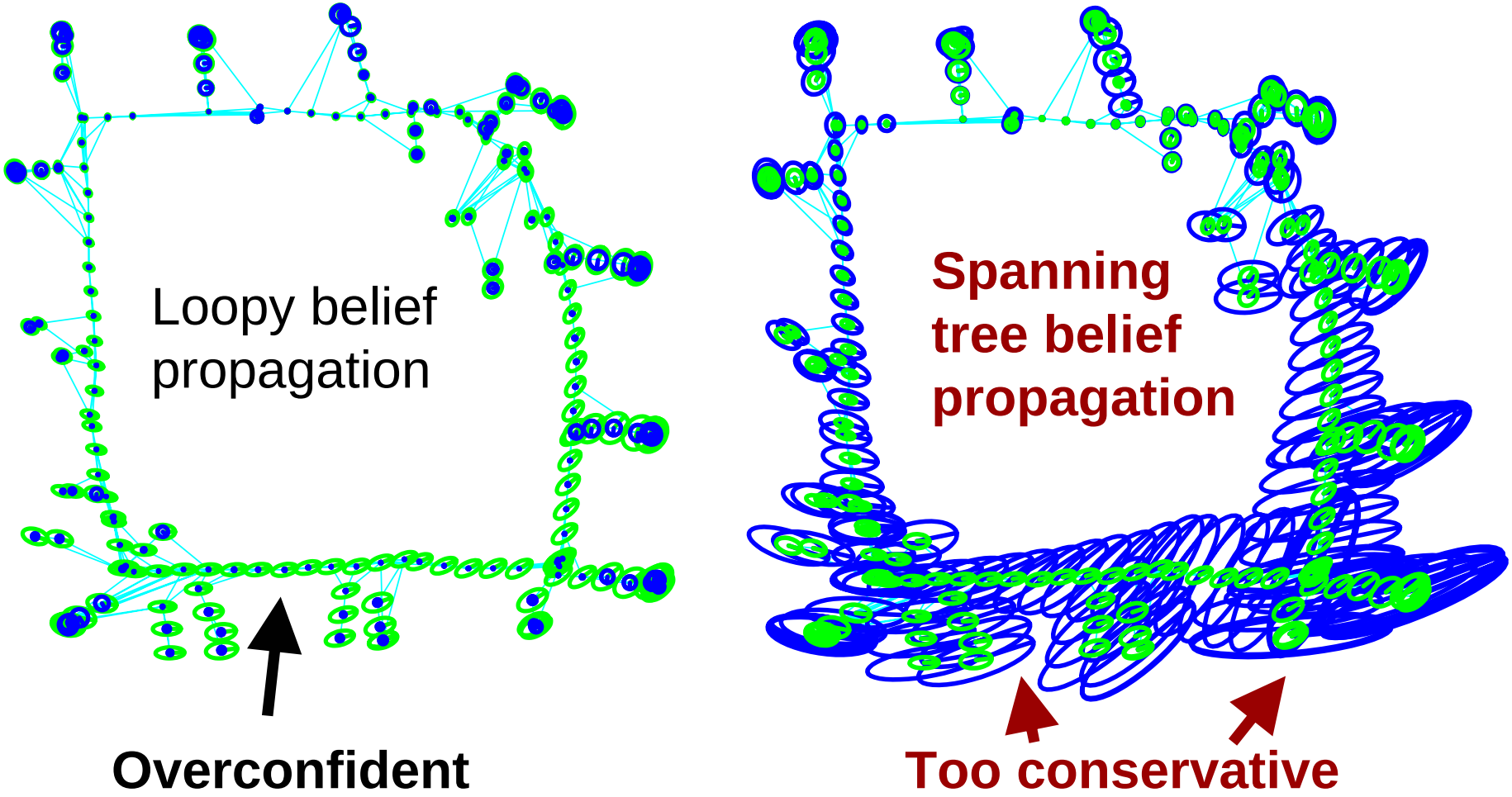
Approximation error





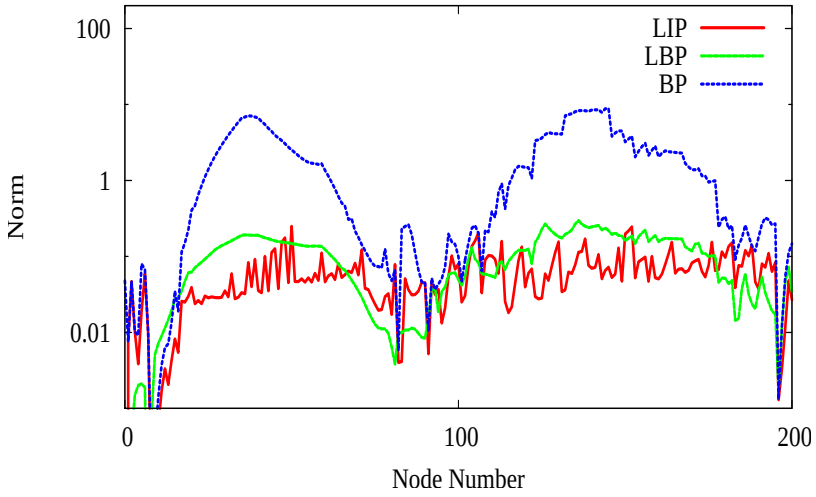
Conservativeness

Experiments – Real Data (Intel)

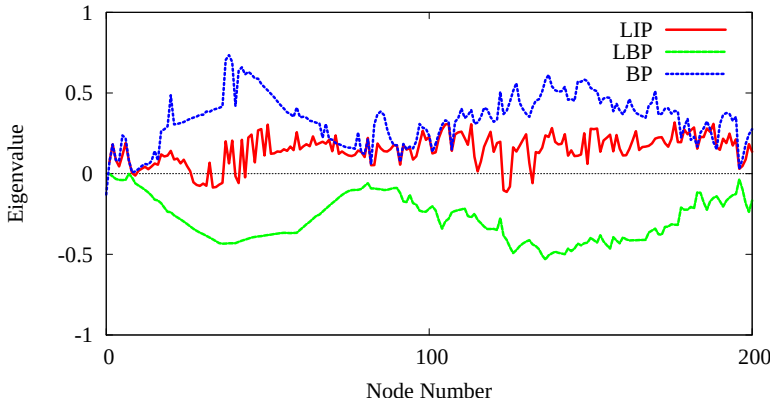


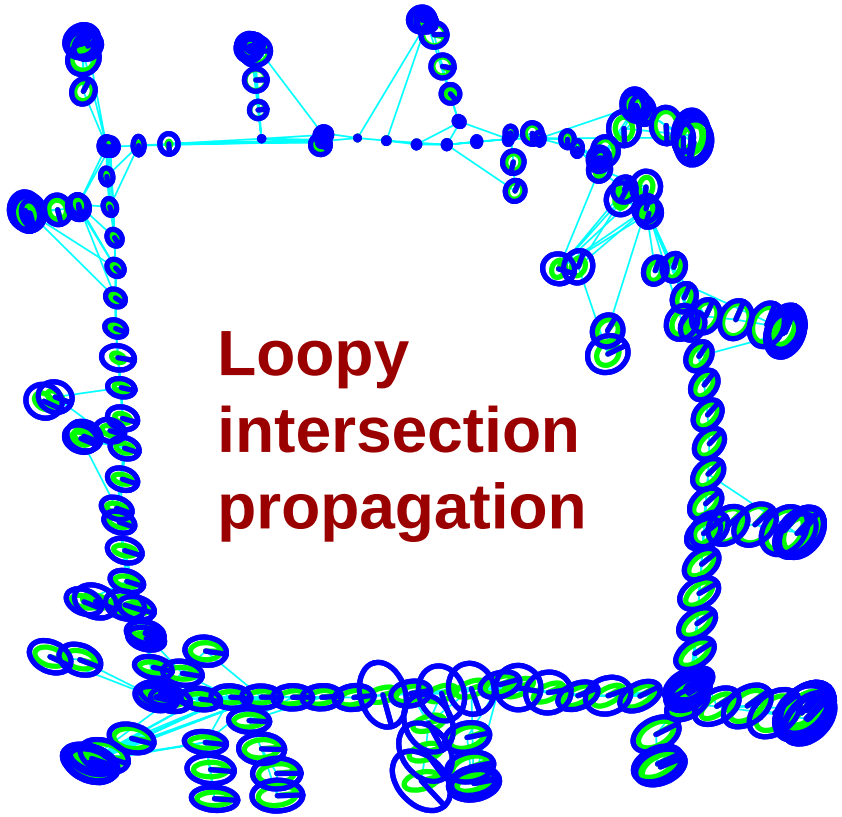
Experiments – Real Data (Intel)

Approximation Error



Conservativeness





Conclusions – So far...

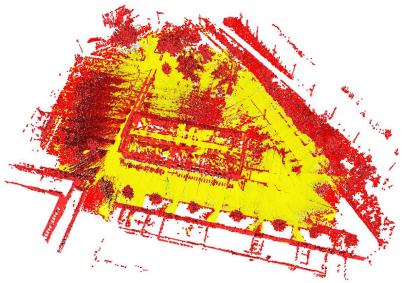
- TORO - Efficient maximum likelihood algorithm for 2D and 3D graphs of poses
- But no covariance estimates!
- Approach for recovering the covariance matrices via belief propagation and covariance intersection
  - Linear time complexity
  - Tight estimates
  - Generally conservative (not guaranteed!)

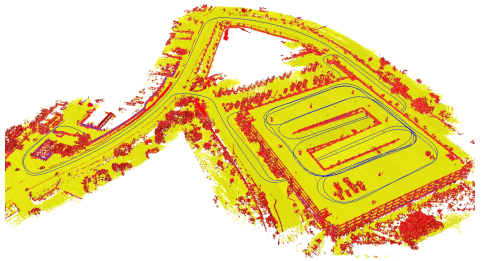
SLAM Front-end and Back-end

- So far, we talked only about the “SLAM back-end” (how to optimize a network)
- For real world applications, the “front-end” is important as well
- How to interpret the sensor data. This encodes
  - Data association problems
  - Extraction of spatial relations/constraints

- The remainder of this talk will gives examples for SLAM front-ends

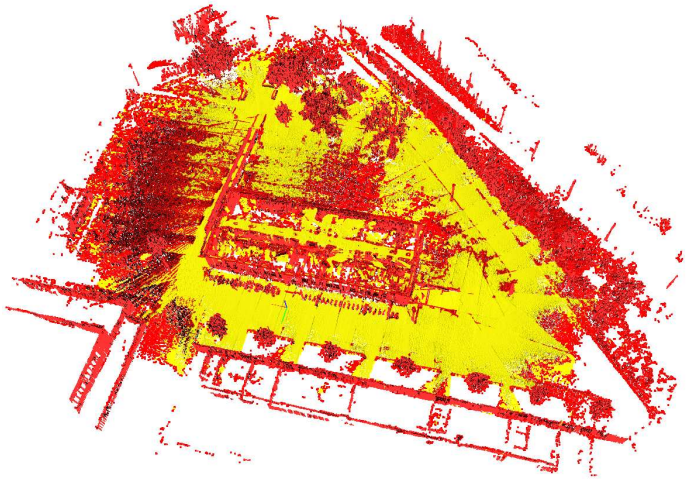
Three Applications Scenarios that use TORO to Build Maps





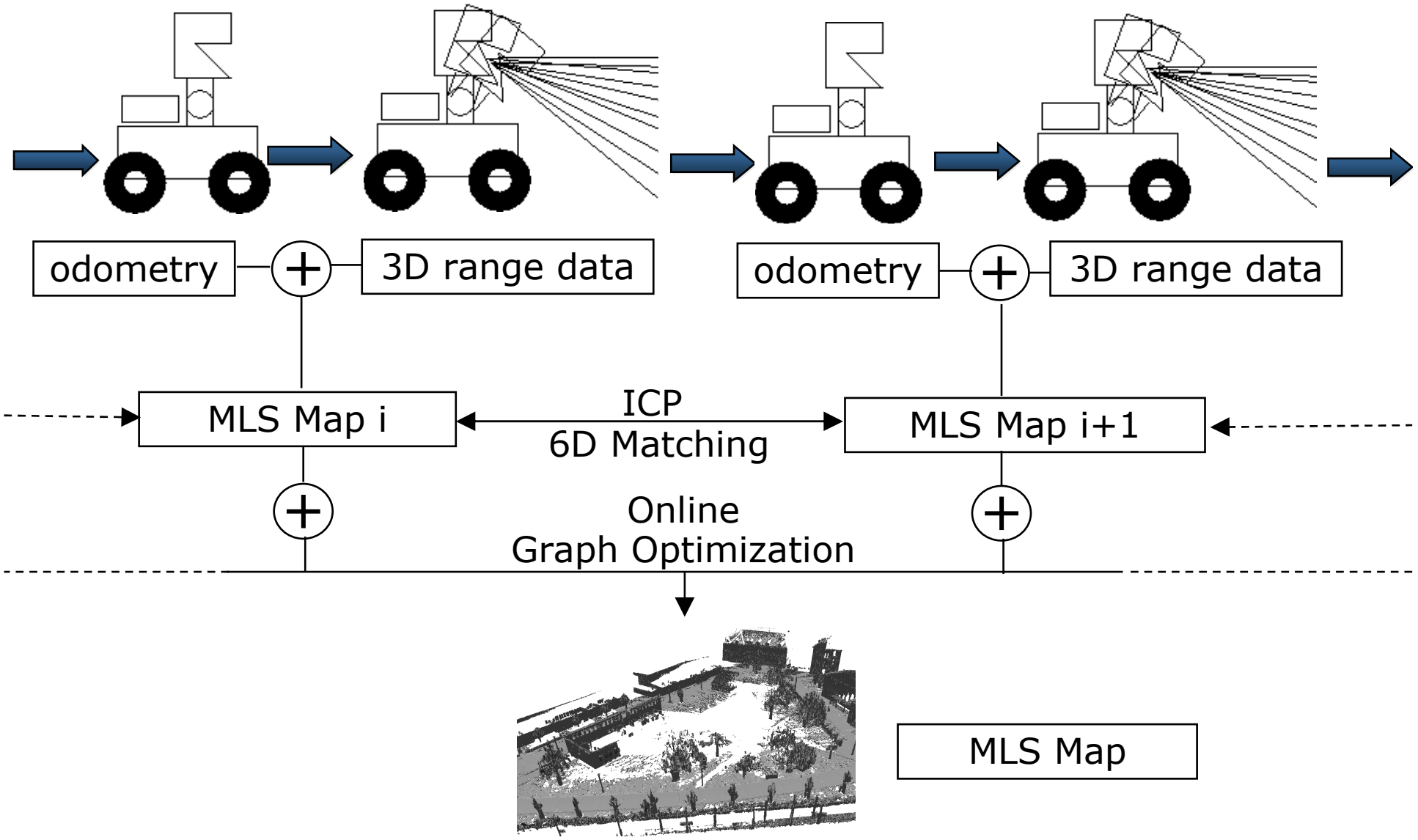
Learning 3D Maps with Laser Data

- Laser range data on a pan-tilt-unit
- Robot that provides odometry

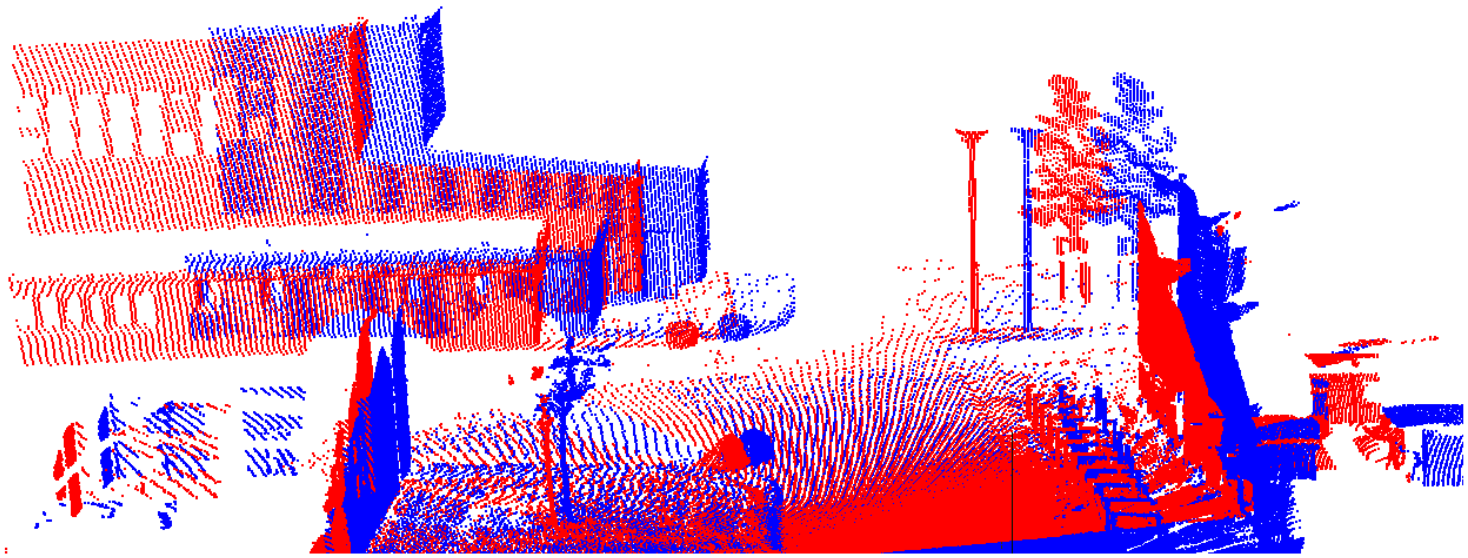


[Kuemmerle et al.]

Incremental 6D SLAM



3D SLAM: Aligning Consecutive Maps



3D SLAM: Aligning Consecutive Maps

- Given that $\mathbf{u}_{i_c}$ and $\mathbf{u}'_{j_c}$ are corresponding points.
- Try to find the parameters R and t which minimize the sum of the squared error e(R,t)

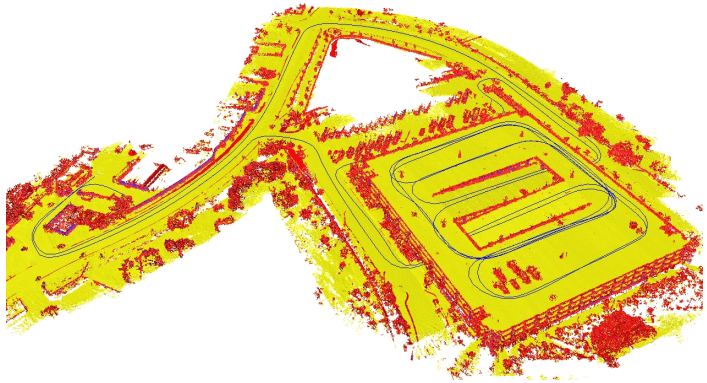
$$e(R,\mathbf{t}) = \underbrace{\sum_{c=1}^{C_1} d_v(\mathbf{u}_{i_c}, \mathbf{u}'_{j_c})}_{\text{vertical objects}} + \underbrace{\sum_{c=1}^{C_2} d(\mathbf{v}_{i_c}, \mathbf{v}'_{j_c})}_{\text{traversable}} + \underbrace{\sum_{c=1}^{C_3} d(\mathbf{w}_{i_c}, \mathbf{w}'_{j_c})}_{\text{non-traversable}}$$

Aerial Image of the Scene

Online Estimated MLS Map

Learning 3D Maps with Laser Data from a Car

- Car robot with a Velodyne 3D laser range scanner
- Use resulting map for autonomous driving

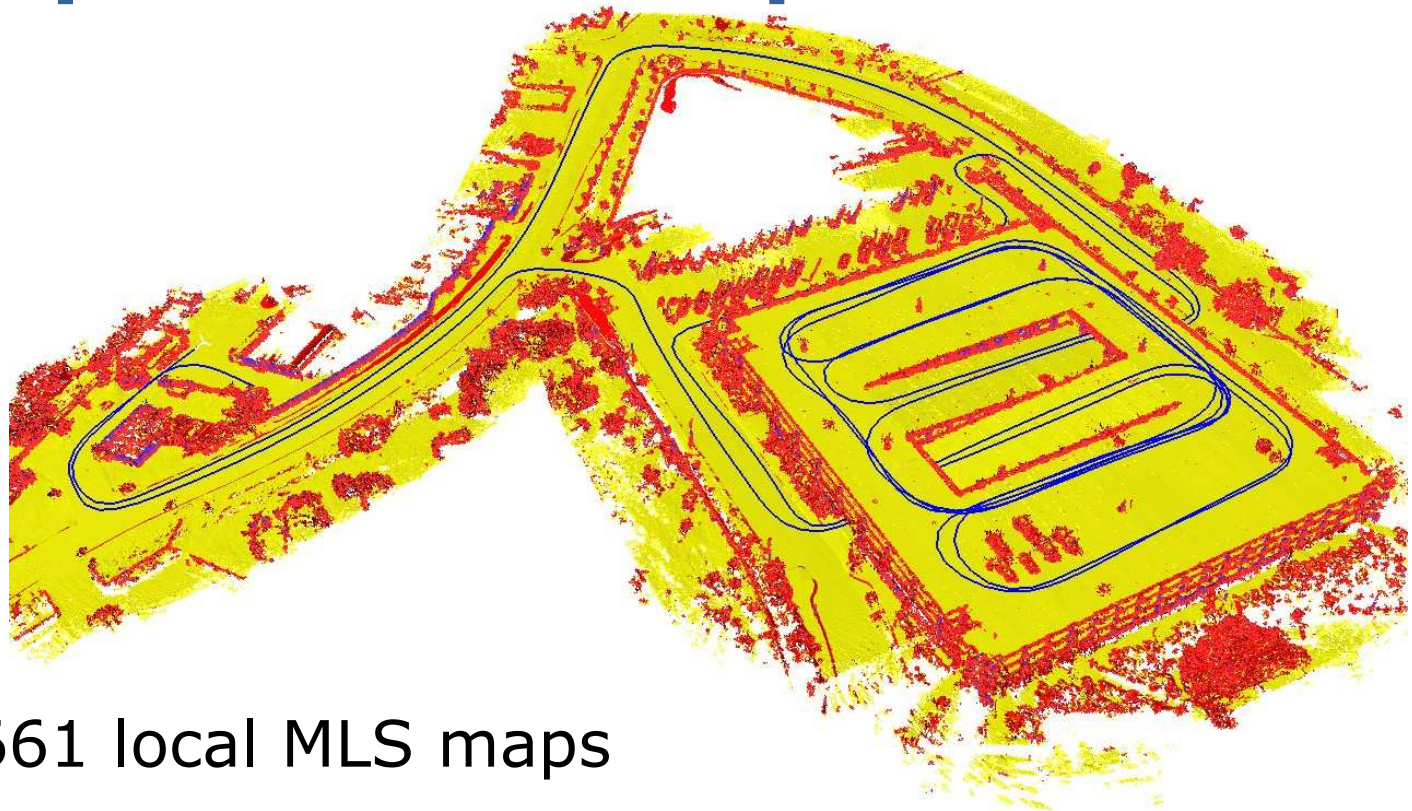


[Kuemmerle et al.]

Learning Large 3D Maps for Driving

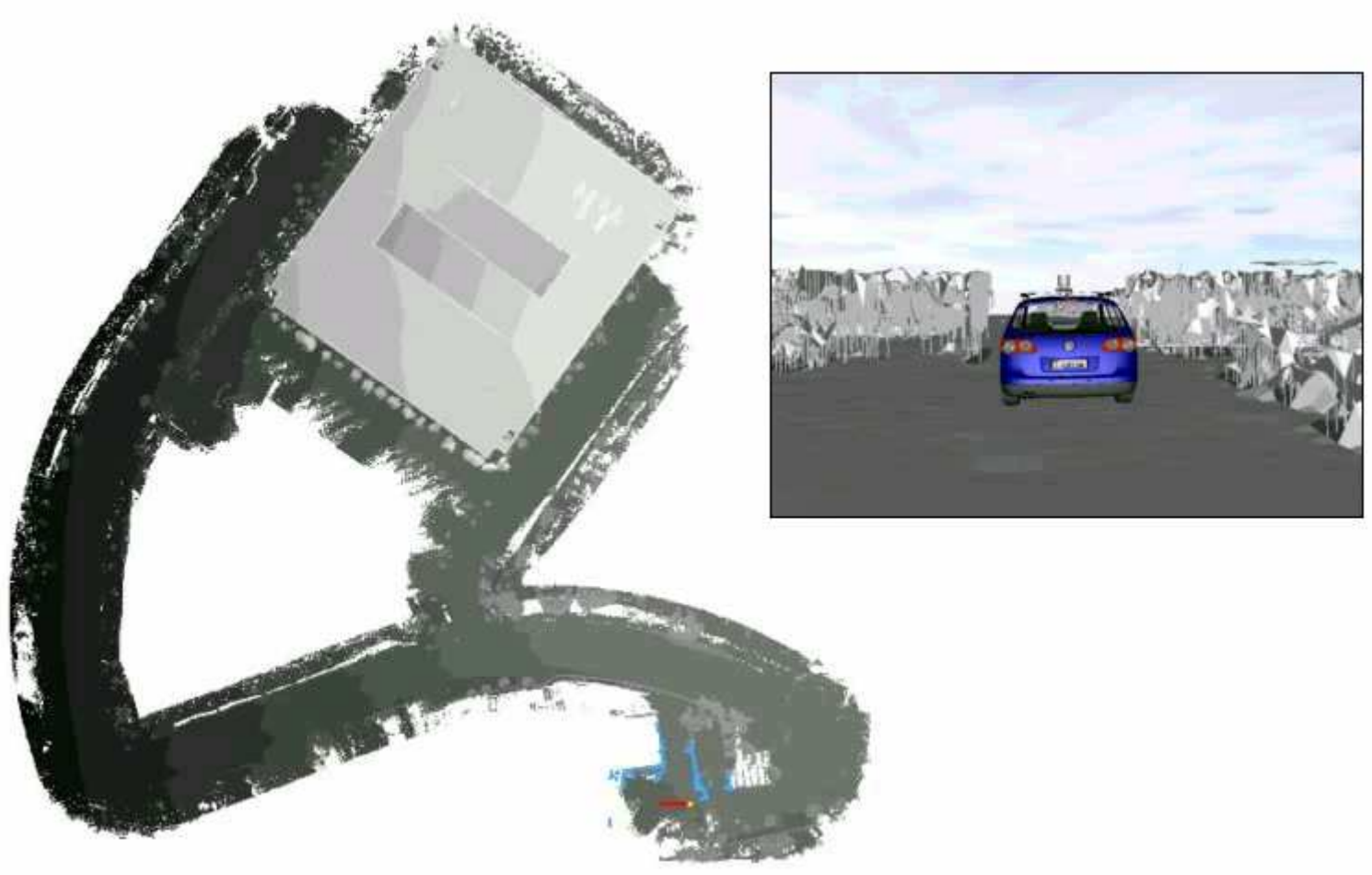
- Parking lot at Stanford University
- Nested loops, trajectory of ~7,000m

Learning Large Scale MLS Maps with Multiple Nested Loops



- 1661 local MLS maps
- nested loops with a total length of ~7,000m
- cell size of 20cm x 20cm.
- requires 118 MB of memory for storage

Application: Car Localization



Learning Maps for Aerial Vehicles

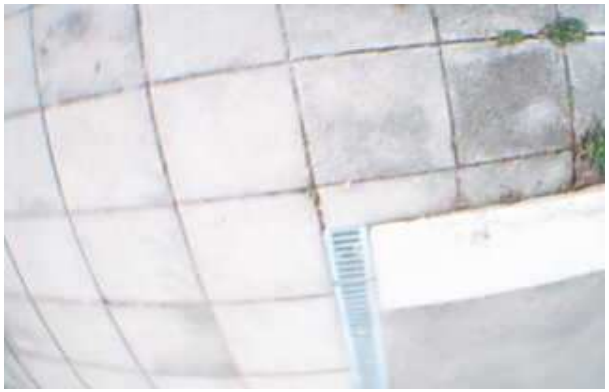
- Learn a map using
  - a flying vehicle

  - Camera(s)

  - An intertial measurement unit

[Steder et al., 08]

Examples of Camera Images



Concrete/Stone

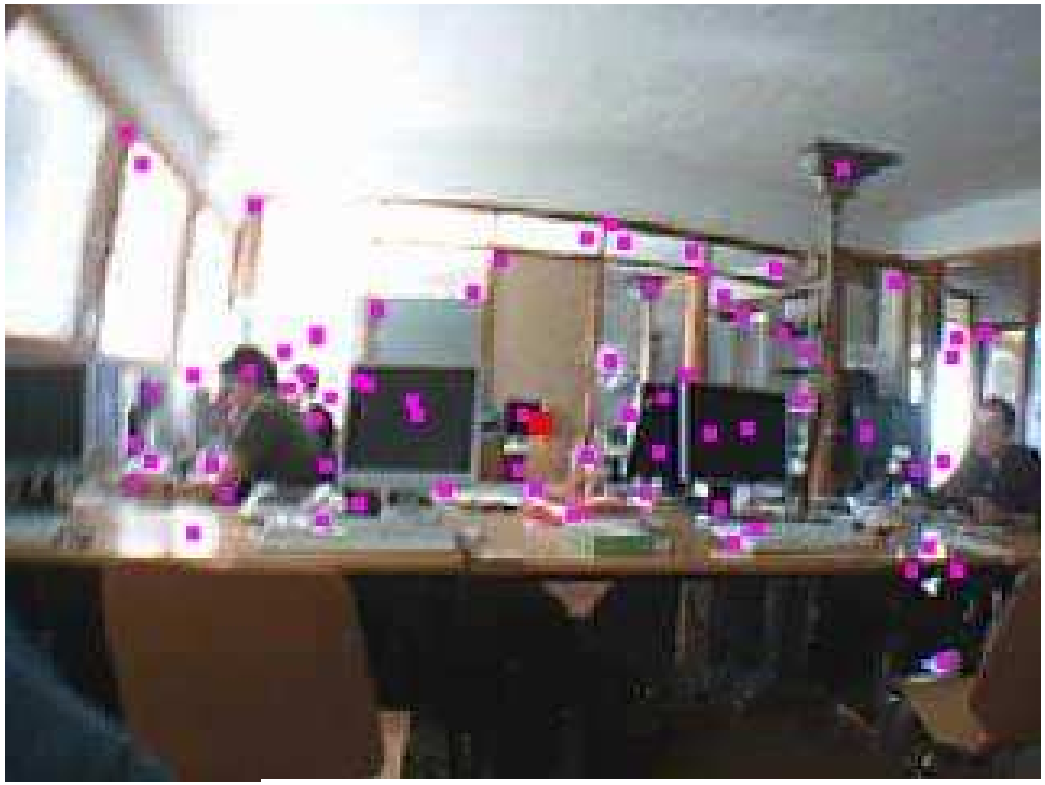
Intersections concrete/grass

Grass

Wooden floor (indoor)

Feature Extraction: SURF

- Provide a description vector and an orientation
- Rotation and scale invariance



[Bay et al., 06]

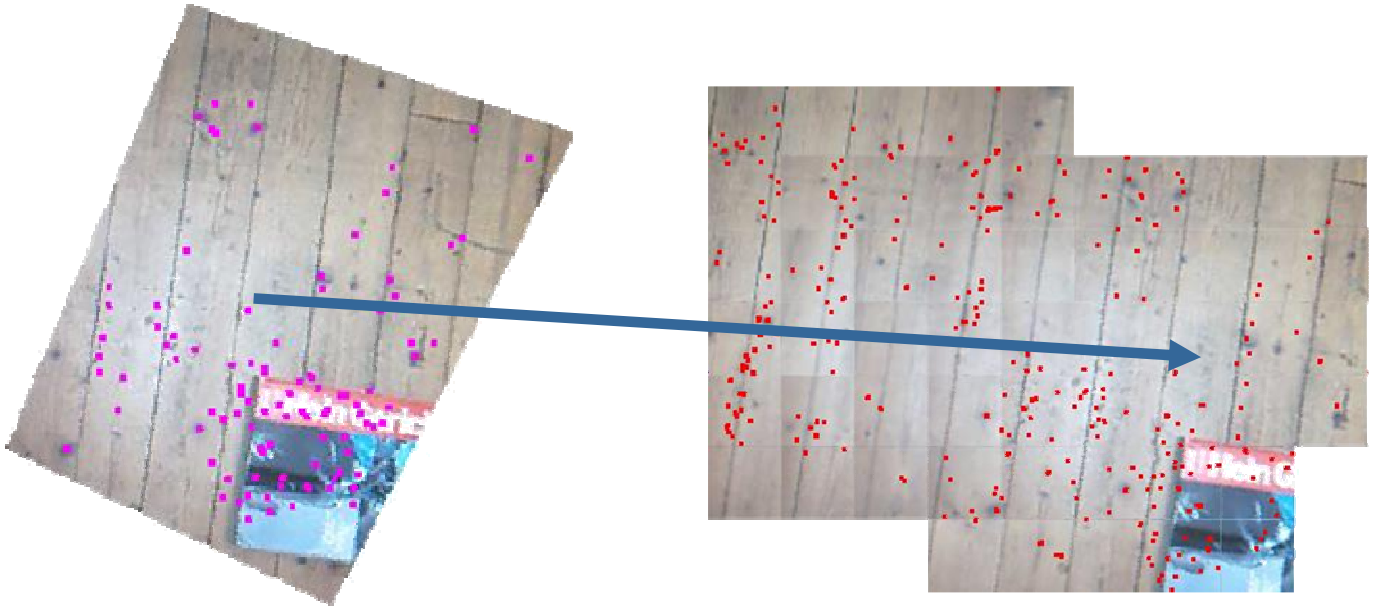
Determining the Camera Pose

Wanted: x, y, z, φ, θ, Ψ (roll, pitch, yaw)

- IMU determines roll and pitch accurately enough
- x, y, z and the heading (yaw) have to be calculated based on the camera images

⇨ 3D position of **two** image features is sufficient to determine the camera pose.

Building a Map



Features in image Features in map

Match features to extract the position

Camera Pose Estimation

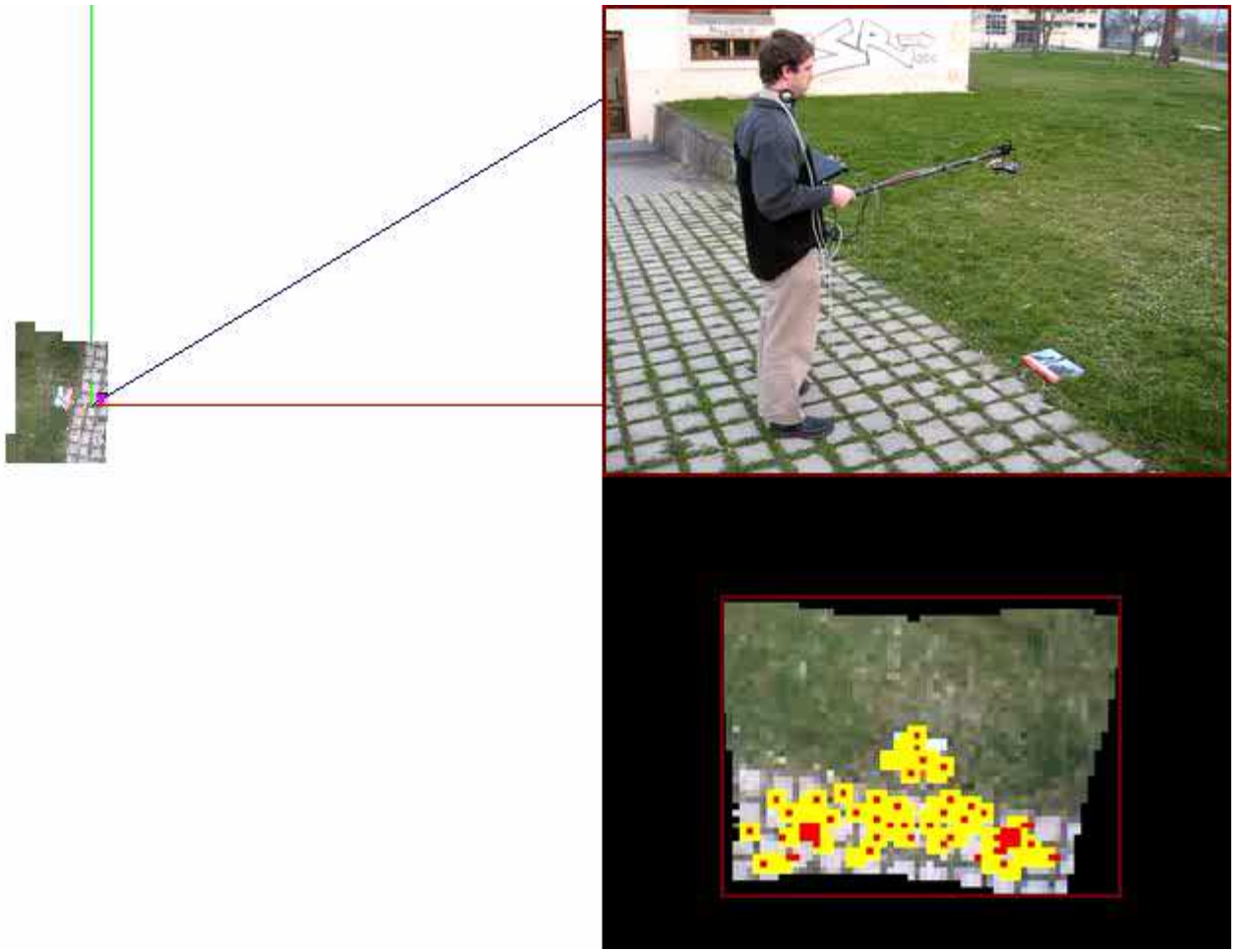
1. Find possible matches.
2. Use the descriptor distance to order matches.
   - Use two matches to calculate the camera position. Start with the best.
   - Re-project all features accordingly to get a quality value about this pose.
   - Repeat until satisfactory pose is found.
3. Update map.

Similar to RANSAC

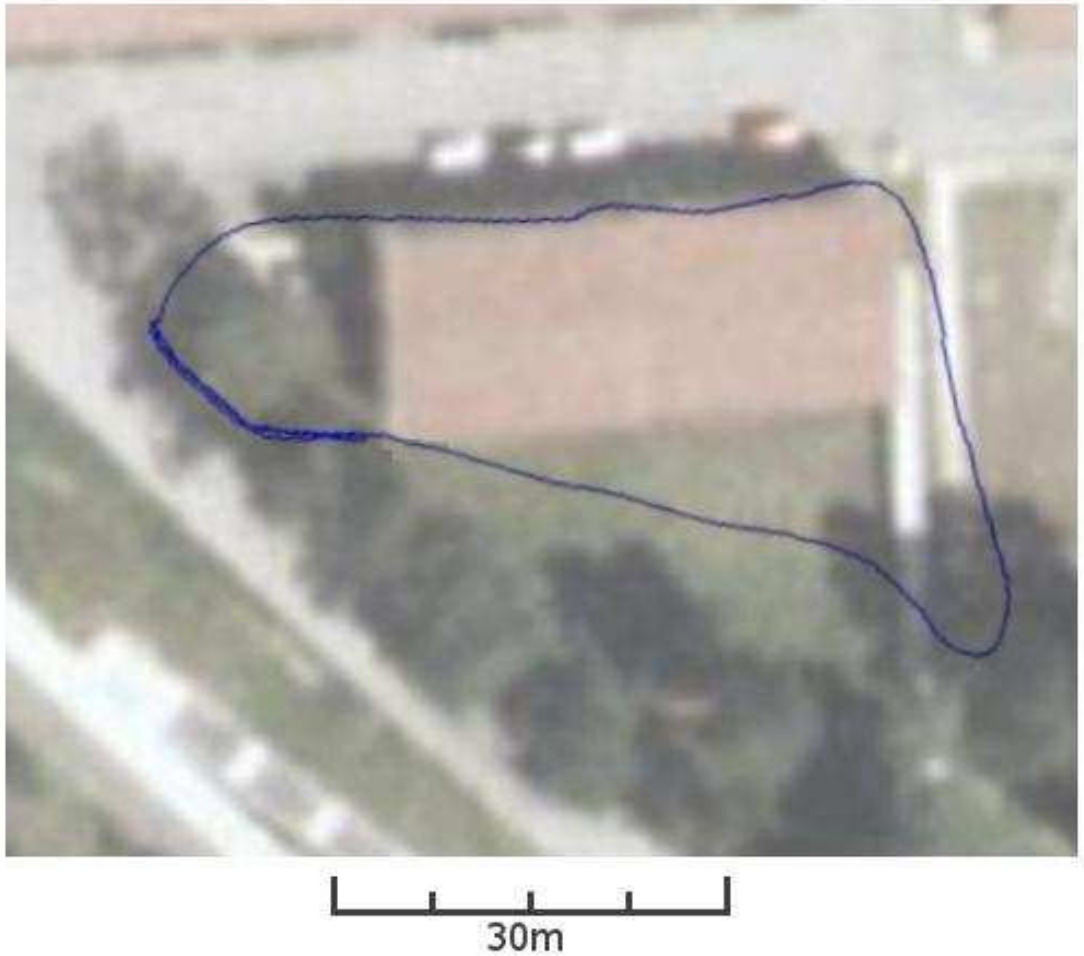
Finding Edges in the Graph

- **Visual odometry:** Compare with temporal close features (e.g., the last 5 frames).
- **Localization:** Compare with features from the map in a given region around the odometry position (local search).
- **Loop closing:** One or more reference features are compared to all map features. Hits result in a localization attempt in that area.

Outdoor Experiment

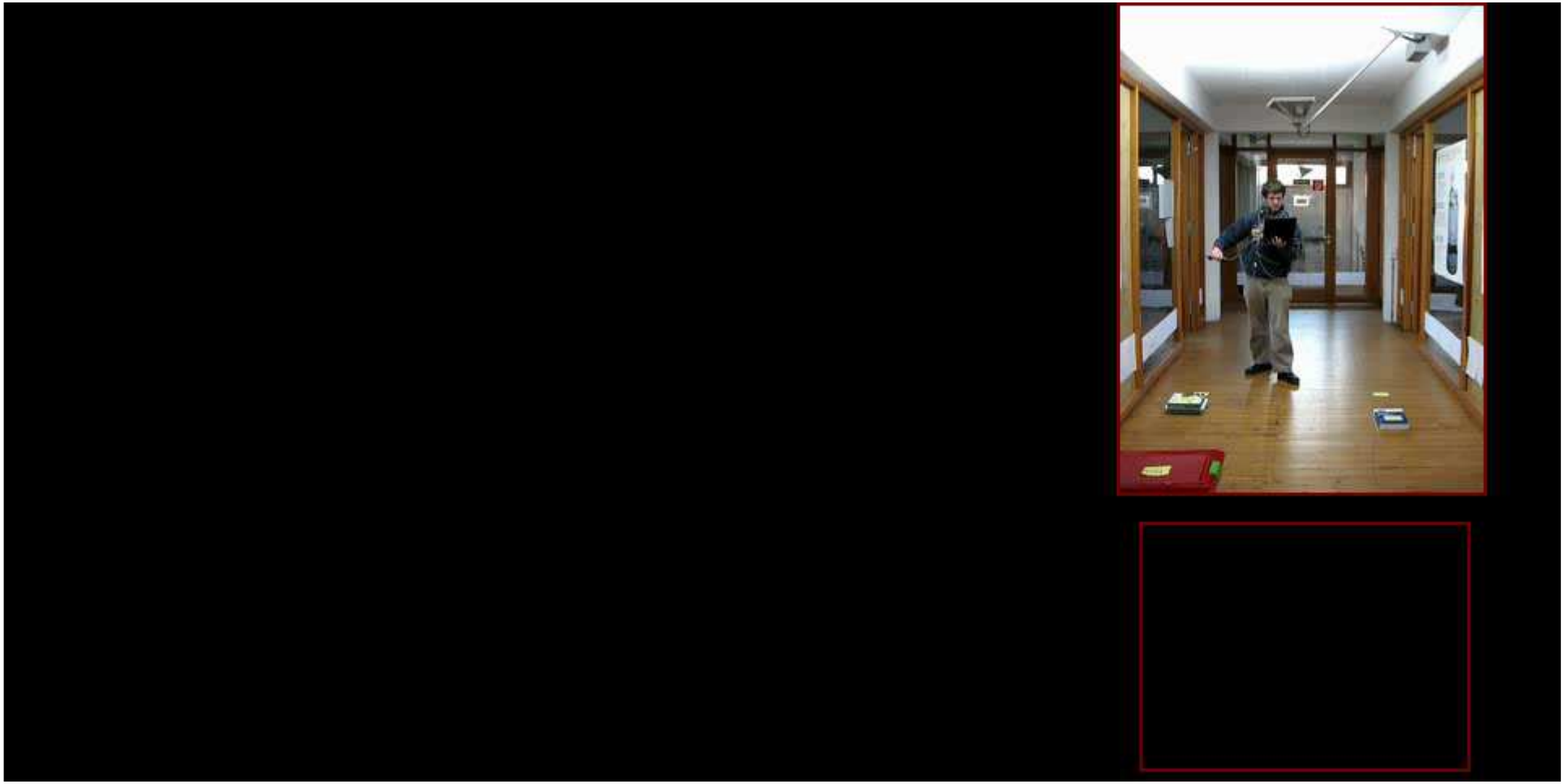


Resulting Trajectory

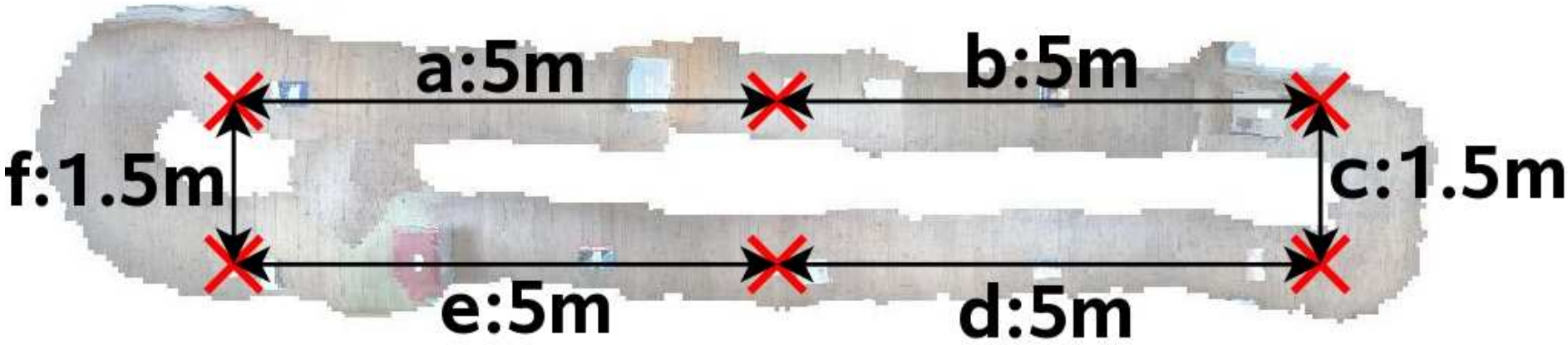


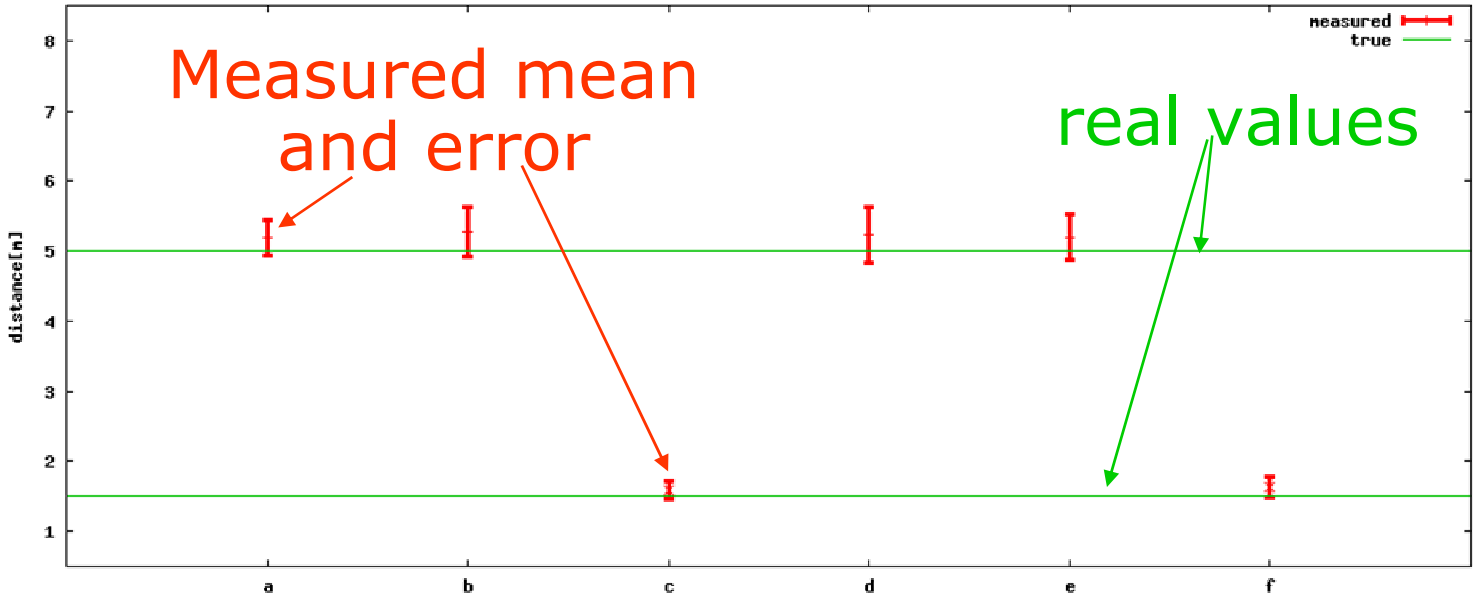
- Length (Google Earth): 188m
- Determined length: 208m

Indoor Experiment



Ground Truth





Result of 10 Loops

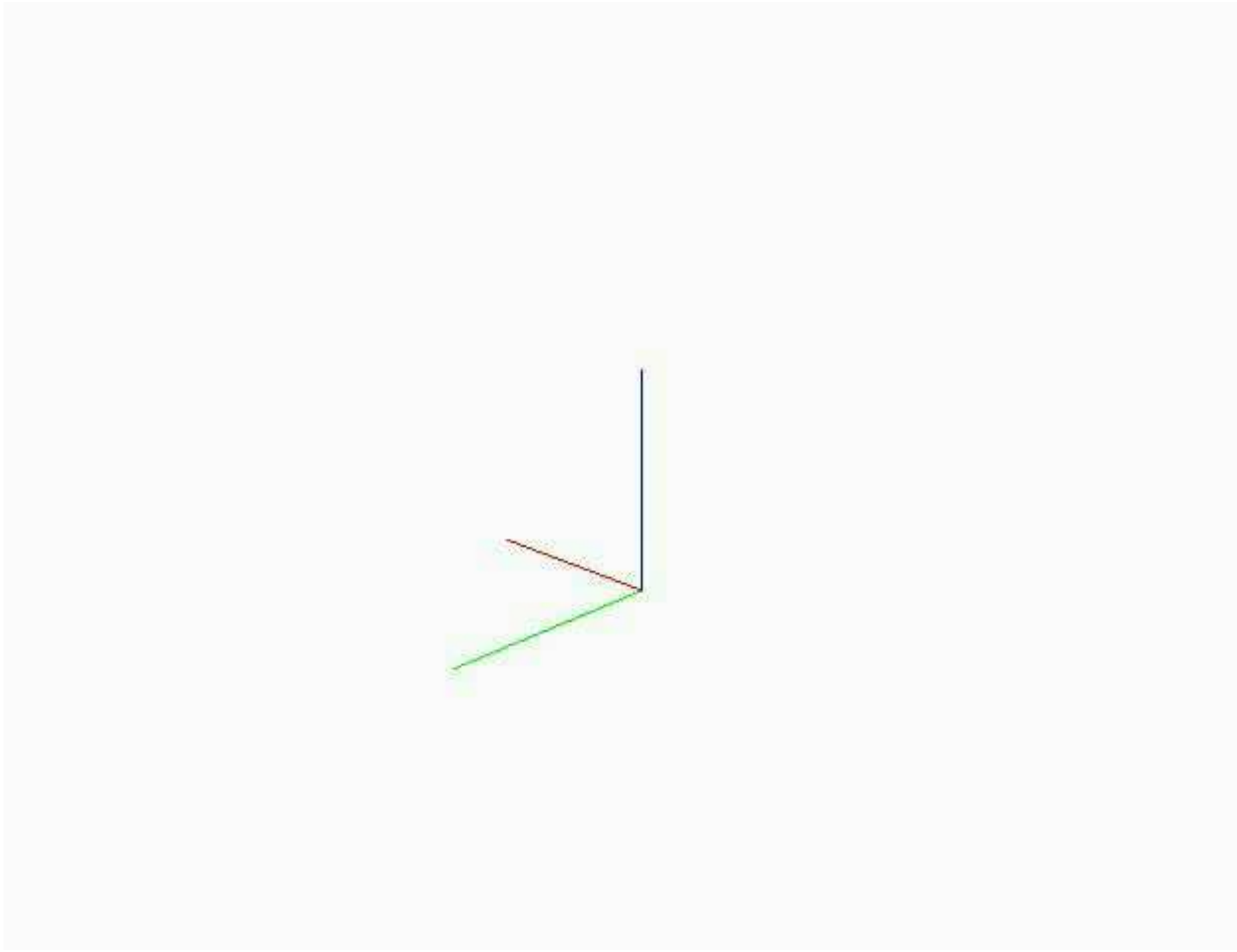
Real trajectory length:	23.00m
Average length in map:	24.11m
Standard deviation:	1.32m
Average error:	5.2%
Error in object heights:	16.9%

Sources of errors:

- Sensor noise
- Inaccuracy in camera calibration
- Wrong feature matches

Experiment with a Blimp

Experiment with a Helicopter



Conclusions

- Highly efficient approach for optimizing 2D and 3D pose graphs
- Orders of magnitude faster than standard nonlinear optimization approaches
- Covariance estimates can be recovered by means of belief propagation with covariance intersection
- Application examples (standard wheeled robots, autonomous cars, flying vehicles)