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:  $x, y, \text{yaw}$

or  $x, y, z, \text{roll}, \text{pitch}, \text{yaw}$
Robot Mapping

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

State: $x, y, \text{yaw}$

or $x, y, z, \text{roll}, \text{pitch}, \text{yaw}$
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

Loop Closing

Visual odometry
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 $p(z | x)$

- 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:

\[ \hat{p} = \text{argmin} \sum_{ij} e_{ij}^T \Omega_{ij} e_{ij} \]
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

[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]
### 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:

\[ x^{t+1} = x^t + \lambda \cdot H^{-1} J^T \Omega_i r_k \]

[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.

\[
x = g(p) \iff p = g^{-1}(x)
\]

\[
x^* = \arg\min_x \sum_{i,j} e'_{ij}(x)^T \Omega_{ij} e'_{ij}(x)
\]
Parameterization of Olson

- Incremental parameterization:

\[ x_i = p_i - p_{i-1} \]

- Results directly from the trajectory taken by the robot

- Problem: for optimizing a constraint between node \( i \) and \( k \), one needs to update 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:

\[ x_i = p_i \ominus p_{\text{parent}(i)} \quad \quad X_i = P_{\text{parent}(i)}^{-1} P_i \]

- 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.
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:
  \[ \Delta x = \lambda \cdot H^{-1} J_{ij}^T \Omega_{ij} r_{ij} \]

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.

\[ P_i \Delta_{ij} \]

Distribute the rotational error

Distribute the translational error
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

$$A'_n = A_nB = QA_n$$

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

$$Q = Q_1Q_2\cdots Q_n$$
$$Q_k = \text{slerp}(Q, u_{k-1})^T\text{slerp}(Q, u_k) \quad u \in [0\ldots\lambda]$$

- 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:

\[ \mathcal{R}' := \text{slerp}(\mathcal{R}, u) \]
\[ \text{axisOf}(\mathcal{R}') = \text{axisOf}(\mathcal{R}) \]
\[ \text{angleOf}(\mathcal{R}') = u \cdot \text{angleOf}(\mathcal{R}) \]
Distributing the Rotational Residual

- To obtain the rotations, we recursively solve:

\[
\begin{align*}
R_1' &= Q_1 R_1 \\
R_2' &= (Q_1 R_1)^T Q_{1:2} R_{1:2} \\
&\vdots \\
R_k' &= [(R_{1:k-1})^T Q_k R_{1:k-1}] R_k
\end{align*}
\]

- 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

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

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

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

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

- 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 $O(M \cdot l)$.

  \[ O(M \cdot l) \]

  - #constrains
  - avg. path length (parameterization tree)
Cost of a Constraint Update

\[ \approx O(M \cdot \log(N)) \]
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

\[
\begin{align*}
\Omega_{ij} &= \Omega_{ij}^{(1)} + \Omega_{ij}^{(2)} \\
\delta_{ij} &= \Omega_{ij}^{-1}(\Omega_{ij}^{(1)} \delta_{ij}^{(1)} + \Omega_{ij}^{(2)} \delta_{ij}^{(2)})
\end{align*}
\]

- 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

Olson’s approach

1 iteration

10 iterations

50 iterations

100 iterations

300 iterations

TORO
TORO vs. Olson’s Approach

[Graphs showing error per constraint against iteration for different approaches.]
Time Comparison (2D)

![Graph showing time comparison for different methods with varying number of constraints.](image)
Robust to the Initial Guess

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

<table>
<thead>
<tr>
<th>noise level</th>
<th>SAM (batch)</th>
<th>SAM (incremental)</th>
<th>Our method (batch)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma = 0.05 )</td>
<td>119 s</td>
<td>not tested (see batch)</td>
<td>20 s (100 iterations)</td>
</tr>
<tr>
<td>( \sigma = 0.1 )</td>
<td>diverged</td>
<td>270 s (optimized each 100 nodes)</td>
<td>40 s (200 iterations)</td>
</tr>
<tr>
<td>( \sigma = 0.2 )</td>
<td>diverged</td>
<td>510 s (optimized each 50 nodes)</td>
<td>50 s (250 iterations)</td>
</tr>
</tbody>
</table>
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
    \[ m_i^{(t)} = \eta_i + \sum_{j \in \mathcal{N}_i} m_j^{(t-1)} \]
    \[ M_i^{(t)} = \Omega_i + \sum_{j \in \mathcal{N}_i} M_j^{(t-1)} \]
  - **Send** messages
    \[ m_{ij}^{(t)} = \eta_{ij} + \Omega_{ij} [ji] \left( \Omega_{ij} [ii] + M_i^{(t)} M_j^{(t-1)} \right)^{-1} (\eta_{ij} + m_i^{(t)} m_j^{(t-1)}) \]
    \[ M_{ij}^{(t)} = \Omega_{ij}^{[ji]} - \Omega_{ij}^{[ii]} \left( \Omega_{ij}^{[ii]} M_j^{(t-1)} \right)^{-1} \Omega_{ij}^{[ii]} \]

*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**
  \[ \hat{\Omega} = \Omega - K \]
- **Regular cutting matrix**
- **Cut our all off-tree edges**

\[
K_{BD} = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & \Omega_{BD}^{[BB]} - P_{BD}^{[B]} & 0 & 0 \\
0 & 0 & \Omega_{BD}^{[BD]} & 0 \\
0 & \Omega_{BD}^{[DB]} & 0 & \Omega_{BD}^{[DD]} - P_{BD}^{[D]}
\end{bmatrix}
\]
Fusing Loops with Spanning Trees

- Estimate A and B

\[
\begin{align*}
E^{[D]}_{BD} &= \Omega_{BD}^{[BB]} - \Omega_{BD}^{[BD]} (M_B + \Omega_{BD}^{[DD]})^{-1} \Omega_{BD}^{[DB]} \\
E^{[B]}_{BD} &= \Omega_{BD}^{[DD]} - \Omega_{BD}^{[DB]} (M_B + \Omega_{BD}^{[BB]})^{-1} \Omega_{BD}^{[BD]} \end{align*}
\]

- Fuse the estimates

\[
\begin{align*}
\hat{M}_B &= \omega_B M_B + (1 - \omega_B) E^{[B]}_{BD} \\
\hat{M}_D &= \omega_D M_D + (1 - \omega_D) E^{[D]}_{BD} \end{align*}
\]

- Compute the priors

\[
P_{i,j}^{[k]} = \hat{M}_k - M_k
\]
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)

Loopy belief propagation
- Overconfident

Spanning tree belief propagation
- Too conservative
Experiments – Real Data (Intel)

Approximation Error

Conservativeness

Loopy intersection propagation
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 give 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

1. odometry + 3D range data → MLS Map i → ICP → MLS Map i+1
2. odometry + 3D range data
3. Online Graph Optimization

MLS Map
3D SLAM: Aligning Consecutive Maps
3D SLAM: Aligning Consecutive Maps

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

\[
e(R, t) = \sum_{c=1}^{C_1} d_v(u_{ic}, u'_{jc}) + \sum_{c=1}^{C_2} d(v_{ic}, v'_{jc}) + \sum_{c=1}^{C_3} d(w_{ic}, w'_{jc})
\]

- \( d_v \): vertical objects
- \( d \): traversable
- \( d \): 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 inertial 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**: \( x, y, z, \varphi, \theta, \Psi \) (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

Measured mean and error

real values
Result of 10 Loops

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Real trajectory length:</td>
<td>23.00m</td>
</tr>
<tr>
<td>Average length in map:</td>
<td>24.11m</td>
</tr>
<tr>
<td>Standard deviation:</td>
<td>1.32m</td>
</tr>
<tr>
<td>Average error:</td>
<td>5.2%</td>
</tr>
<tr>
<td>Error in object heights:</td>
<td>16.9%</td>
</tr>
</tbody>
</table>

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)