Photogrammetry & Robotics Lab

Graph-Based SLAM
A Least Squares Approach to SLAM using Pose Graphs

Cyrill Stachniss

Traditional SLAM Paradigms

Kalman filter  Particle filter  Graph-based

least squares approach to SLAM

Least Squares in General

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

Today: Application to SLAM

Graph-Based SLAM

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

Robot pose  Constraint
Graph-Based SLAM

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

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 minimize the error introduced by the constraints

Graph-Based SLAM in a Nutshell

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

KUKA Halle 22, courtesy of P. Pfaff
Graph-Based SLAM in a Nutshell

- 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

The Overall SLAM System

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

- It consists of $n$ nodes $x = x_1:n$
- Each $x_i$ is a pose of the robot at time $t_i$
- A constraint/edge exists between the nodes $x_i$ and $x_j$ if...

Create an Edge If... (1)

- the robot moves from $x_i$ to $x_{i+1}$
- Edge corresponds to odometry

Create an Edge If... (2)

- the robot observes the same part of the environment from $x_i$ and from $x_j$
- Construct a virtual measurement about the position of $x_j$ seen from $x_i$

Create an Edge If... (2)

- the robot observes the same part of the environment from $x_i$ and from $x_j$
- Construct a virtual measurement about the position of $x_j$ seen from $x_i$
Transformations

- Transformations can be expressed using **homogenous coordinates**
- Odometry-Based edge
  \[(X^{-1}_iX_{i+1})\]
- Observation-Based edge
  \[(X^{-1}_iX_j)\]
  How node i sees node j

Homogenous Coordinates

- H.C. are a system of coordinates used in projective geometry
- Projective geometry is an alternative representation of geometric objects and transformations
- **A single matrix can represent affine transformations and projective transformations**

Homogenous Coordinates

- N-dim space expressed in N+1 dim
- 4 dim. for modeling the 3D space
- To HC: \((x, y, z)^T \rightarrow (x, y, z, 1)^T\)
- Backwards: \((x, y, z, w)^T \rightarrow (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T\)
- Vector in HC: \(v = (x, y, z, w)^T\)
- Translation:
  \[T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}\]
- Rotation:
  \[R = \begin{pmatrix} R_{3D} & 0 \\ 0 & 1 \end{pmatrix}\]

The Edge Information Matrices

- Observations are affected by noise
- Information matrix \(\Omega_{ij}\) for each edge to encode its uncertainty
- The “bigger” \(\Omega_{ij}\) the more the edge “matters” in the optimization

Questions

- How do the information matrices look like in case of scan-matching vs. odometry?
- How will these matrices look like when moving in a long, featureless corridor?
**Pose Graph**

Nodes according to the graph

The graph error observation of \( x_j \) from \( x_i \)

\[ e_{ij}(x_i, x_j) \]

- **Goal:**
  \[ x^* = \arg\min_x \sum_{ij} e_{ij}^T(x) \Omega_{ij} e_{ij}(x) \]

**Least Squares SLAM**

- This error function looks suitable for least squares error minimization

  \[ x^* = \arg\min_x \sum_{ij} e_{ij}^T(x) \Omega_{ij} e_{ij}(x) = \arg\min_x \sum_k e_k^T(x) \Omega_k e_k(x) \]

**Question:**
- What is the state vector?
Least Squares SLAM

- This error function looks suitable for least squares error minimization
  \[ x^* = \arg\min_x \sum_k e_k^T(x)\Omega_k e_k(x) \]

Question:

- What is the state vector?
  \[ x^T = (x_1^T \ x_2^T \ \cdots \ x_n^T) \]
- Specify the error function!

Gauss-Newton: The Overall Error Minimization Procedure

- Define the error function
- Linearize the error function
- Compute its derivative
- Set the derivative to zero
- Solve the linear system
- Iterate this procedure until convergence

The Error Function

- Error function for a single constraint
  \[ e_{ij}(x_i, x_j) = t2v(Z_{ij}^{-1}(X_i^{-1}X_j)) \]

- Error as a function of the whole state vector
  \[ e_{ij}(x) = t2v(Z_{ij}^{-1}(X_i^{-1}X_j)) \]

- Error takes a value of zero if
  \[ Z_{ij} = (X_i^{-1}X_j) \]

Linearizing the Error Function

- We can approximate the error functions around an initial guess \( x \) via Taylor expansion
  \[ e_{ij}(x + \Delta x) \approx e_{ij}(x) + J_{ij}\Delta x \]
  with \( J_{ij} = \frac{\partial e_{ij}(x)}{\partial x} \)
Derivative of the Error Function

- Does one error term $e_{ij}(x)$ depend on all state variables?
  
  No, only on $x_i$ and $x_j$

- Is there any consequence on the structure of the Jacobian?
  
  Yes, it will be non-zero only in the rows corresponding to $x_i$ and $x_j$

$$\frac{\partial e_{ij}(x)}{\partial x} = \begin{pmatrix} 0 & \cdots & \frac{\partial e_{ij}(x_i)}{\partial x_i} & \cdots & \frac{\partial e_{ij}(x_j)}{\partial x_j} & \cdots & 0 \end{pmatrix}$$

$$J_{ij} = \begin{pmatrix} 0 & \cdots & A_{ij} & \cdots & B_{ij} & \cdots & 0 \end{pmatrix}$$
**Jacobians and Sparsity**

- Error $e_{ij}(x)$ depends only on the two parameter blocks $x_i$ and $x_j$

\[ e_{ij}(x) = e_{ij}(x_i, x_j) \]

- The Jacobian will be zero everywhere except in the columns of $x_i$ and $x_j$

\[
J_{ij} = \begin{bmatrix}
0 & \frac{\partial e(x_i)}{\partial x_i} & 0 & \frac{\partial e(x_j)}{\partial x_j} & 0 & \cdots & 0
\end{bmatrix}
\]

**Consequences of the Sparsity**

- We need to compute the coefficient vector $b$ and matrix $H$:

\[
b^T = \sum_{ij} b_{ij}^T = \sum_{ij} e_{ij}^T \Omega_{ij} J_{ij}
\]

\[
H = \sum_{ij} H_{ij} = \sum_{ij} J_{ij}^T \Omega_{ij} J_{ij}
\]

- The sparse structure of $J_{ij}$ will result in a sparse structure of $H$

- This structure reflects the adjacency matrix of the graph

**Illustration of the Structure**

- $b_{ij} = J_{ij}^T \Omega_{ij} e_{ij}$

- Non-zero only at $x_i$ and $x_j$

- Non-zero on the main diagonal at $x_i$ and $x_j$
Illustration of the Structure

\[ b_{ij} = J_{ij}^T \Omega_{ij} e_{ij} \]

Non-zero only at \( x_i \) and \( x_j \)

\[ H_{ij} = J_{ij}^T \Omega_{ij} J_{ij} \]

Non-zero on the main diagonal at \( x_i \) and \( x_j \)

... and at the blocks \( i,j \)

Consequences of the Sparsity

- An edge contributes to the linear system via \( b_{ij} \) and \( H_{ij} \)
- The coefficient vector is:

\[
\begin{align*}
    b_{ij}^T &= e_{ij}^T \Omega_{ij} J_{ij} \\
              &= e_{ij}^T \Omega_{ij} \left( \begin{array}{cccc}
                      0 & \cdots & A_{ij} & \cdots & B_{ij} & \cdots & 0 \\
                      \end{array} \right) \\
              &= \left( \begin{array}{cccc}
                      0 & \cdots & e_{ij}^T \Omega_{ij} A_{ij} & \cdots & e_{ij}^T \Omega_{ij} B_{ij} & \cdots & 0 \\
                      \end{array} \right)
\end{align*}
\]

- It is non-zero only at the indices corresponding to \( x_i \) and \( x_j \)

Illustration of the Structure

\[ b = \sum_{ij} b_{ij} \]

\[ H = \sum_{ij} H_{ij} \]

Consequences of the Sparsity

- The coefficient matrix of an edge is:

\[
H_{ij} = J_{ij}^T \Omega_{ij} J_{ij}
= \left( \begin{array}{ccc}
    A_{ij}^T & \cdots & A_{ij}^T \Omega_{ij} B_{ij} \\
    \vdots & & \vdots \\
    B_{ij}^T & \cdots & B_{ij}^T \Omega_{ij} B_{ij}
\end{array} \right)
\]

- Non-zero only in the blocks relating \( i,j \)
Sparsity Summary

- An edge $i_j$ contributes only to the $i^{th}$ and the $j^{th}$ block of $b_{ij}$
- to the blocks $i_i$, $j_j$, $i_j$ and $j_i$ of $H_{ij}$
- Resulting system is sparse
- System can be computed by summing up the contribution of each edge
- Efficient solvers can be used
  - Sparse Cholesky decomposition
  - Conjugate gradients
  - ... many others

The Linear System

- Vector of the states increments:
  $$\Delta x^T = (\Delta x_1^T \Delta x_2^T \ldots \Delta x_n^T)$$
- Coefficient vector:
  $$b^T = (b_1^T b_2^T \ldots b_n^T)$$
- Normal equation matrix:
  $$H = \begin{pmatrix}
  \bar{H}^{11} & \bar{H}^{12} & \cdots & \bar{H}^{1n} \\
  \bar{H}^{21} & \bar{H}^{22} & \cdots & \bar{H}^{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  \bar{H}^{n1} & \bar{H}^{n2} & \cdots & \bar{H}^{nn}
\end{pmatrix}$$

Building the Linear System

For each constraint:
- Compute error $e_{ij} = t2v(Z_{ij}^{-1}(X_i^{-1}X_j))$
- Compute the blocks of the Jacobian:
  $$A_{ij} = \frac{\partial e(x_i, x_j)}{\partial x_i} \quad B_{ij} = \frac{\partial e(x_i, x_j)}{\partial x_j}$$
- Update the coefficient vector:
  $$\tilde{b}_i^T + = e_{ij}T \Omega_{ij} A_{ij} \quad \tilde{b}_j^T + = e_{ij}^T \Omega_{ij} B_{ij}$$
- Update the normal equation matrix:
  $$\bar{H}^{ii} + = A_{ij}^T \Omega_{ij} A_{ij} \quad \bar{H}^{ij} + = A_{ij}^T \Omega_{ij} B_{ij}$$
  $$\bar{H}^{ji} + = B_{ij}^T \Omega_{ij} A_{ij} \quad \bar{H}^{jj} + = B_{ij}^T \Omega_{ij} B_{ij}$$

Algorithm

1: optimize(x):
2: while (!converged)
3:   (H, b) = buildLinearSystem(x)
4:   $\Delta x = \text{solveSparse}(H \Delta x = -b)$
5:   $x = x + \Delta x$
6: end
7: return x
Example on the Blackboard

Trivial 1D Example

- Two nodes and one observation
  \[ \mathbf{x} = (x_1, x_2)^T = (0, 0) \]
  \[ z_{12} = 1 \]
  \[ \Omega = 2 \]
  \[ e_{12} = z_{12} - (x_2 - x_1) = 1 - (0 - 0) = 1 \]
  \[ J_{12} = (1 - 1) \]
  \[ b_{12}^T = e_{12}^T \Omega_{12} J_{12} = (2 - 2) \]
  \[ H_{12} = J_{12}^T \Omega J_{12} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} \]
  \[ \Delta \mathbf{x} = -H_{12}^{-1} b_{12} \] **BUT** \[ \text{det}(H) = 0 \] ???

What Went Wrong?

- The constraint specifies a **relative constraint** between both nodes
- Any poses for the nodes would be fine as long as their relative coordinates fit
- **One node needs to be “fixed”**

\[ H = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \text{ constraint that sets } dx_1 = 0 \]

\[ \Delta \mathbf{x} = -H_{12}^{-1} b_{12} \]

\[ \Delta \mathbf{x} = (0, 1)^T \]

Role of the Prior

- We saw that the matrix \( H \) has not full rank (after adding the constraints)
- The global frame had not been fixed
- Fixing the global reference frame is strongly related to the prior \( p(x_0) \)
- A Gaussian estimate about \( x_0 \) results in an additional constraint
- E.g., first pose in the origin:
  \[ e(x_0) = t2v(X_0) \]
Real World Example

Fixing a Subset of Variables

- Assume that the value of certain variables during the optimization is known a priori
- We may want to optimize all others and keep these fixed
- How?

- If a variable is not optimized, it should disappear from the linear system
- Construct the full system
- Suppress the rows and the columns corresponding to the variables to fix
Why Can We Simply Suppress the Rows and Columns of the Corresponding Variables?

\[ p(\alpha, \beta) = \mathcal{N}(\begin{bmatrix} \mu_\alpha \\ \mu_\beta \end{bmatrix}, \begin{bmatrix} \Sigma_{\alpha\alpha} & \Sigma_{\alpha\beta} \\ \Sigma_{\beta\alpha} & \Sigma_{\beta\beta} \end{bmatrix}) = \mathcal{N}^{-1}(\begin{bmatrix} \eta_\alpha \\ \eta_\beta \end{bmatrix}, \begin{bmatrix} \Lambda_{\alpha\alpha} & \Lambda_{\alpha\beta} \\ \Lambda_{\beta\alpha} & \Lambda_{\beta\beta} \end{bmatrix}) \]

<table>
<thead>
<tr>
<th>MARGINALIZATION</th>
<th>CONDITIONING</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p(\alpha) = \int p(\alpha, \beta) d\beta )</td>
<td>( p(\alpha</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\mu &= \mu_{\alpha} \\
\Sigma &= \Sigma_{\alpha\alpha} \\
\mu' &= \mu_{\alpha} + \Sigma_{\alpha\beta} \Sigma_{\beta\beta}^{-1} (\beta - \mu_{\beta}) \\
\Sigma' &= \Sigma_{\alpha\alpha} - \Sigma_{\alpha\beta} \Sigma_{\beta\beta}^{-1} \Sigma_{\beta\alpha} \\
\eta &= \eta_{\alpha} - \Lambda_{\alpha\beta} \Lambda_{\beta\beta}^{-1} \eta_{\beta} \\
\eta' &= \eta_{\alpha} - \Lambda_{\alpha\beta} \beta \\
\Lambda' &= \Lambda_{\alpha\alpha}
\end{align*}
\]

Relative Uncertainty

To determine the relative uncertainty between \( x_i \) and \( x_j \):

- Construct the full matrix \( \mathbf{H} \)
- Suppress the rows and the columns of \( x_i \) (= do not optimize/fix this variable)
- Compute the block \( j, j \) of the inverse
- This block will contain the covariance matrix of \( x_j \) w.r.t. \( x_i \), which has been fixed

Uncertainty

- \( \mathbf{H} \) represents the information matrix given the linearization point
- Inverting \( \mathbf{H} \) gives the (dense) covariance matrix
- The diagonal blocks of the covariance matrix represent the uncertainties of the corresponding variables

Example

![Robot Example](image)
Conclusions

- The back-end part of the SLAM problem can be effectively solved with Gauss-Newton
- The $H$ matrix is typically sparse
- This sparsity allows for efficiently solving the linear system
- One of the state-of-the-art solutions for computing maps

Literature

Least Squares SLAM

- Grisetti, Kümmerle, Stachniss, Burgard: “A Tutorial on Graph-based SLAM”, 2010