Graph SLAM
Limitations of EKF-SLAM

- Time complexity - $O(n^2)$
- Memory complexity - $O(n^2)$

So this becomes difficult for EKF SLAM:

Large City Navigation Scenario (3.3 kilometers) from the DARPA ASPN project
Graph SLAM Representation

- Let’s say that we have a robot moving through space

- Each of the **states** can be represented as a variable node in a graph.
- The **action** can be represented as the a constraint denoted by an edge.
Each of the landmarks can be represented as a node in a graph.

The measurement can be represented as the a constraint denoted by an edge.
Factor Graph Intuition

- Think of each of the constraints as springs
- The stiffness of the string will be the uncertainty.
Victoria park dataset

- iSAM paper which uses factor graphs solves the complete problem including data association problem in 7.7 mins, the sequence is 26 min long
- 3.3 times faster than real-time on a laptop computer

Nebot Et Al
SLAM is basically figuring out the state given the measurements

So what we want is:

\[ p(X|Z) \]

\[ p(X|Z) = \frac{p(X,Z)}{P(Z)} \]

To get that we want:

\[ p(X,Z) \]
Toy problem: Introducing dynamics

$x_1, x_2$ is the state of the robot

There are no observation

\[
p(X, Z) = p(x_1, x_2) = p(x_2|x_1)p(x_1)
\]
Toy problem: Introducing Measurement

\[ x_1 \text{ is the state of the robot} \]
\[ z_1 \text{ is the observation} \]

\[ p(X, Z) = p(x_1, z_1) = p(z_1|x_1)p(x_1) \]
Toy problem: Dynamics and Measurement

\( x_1, x_2 \) is the state of the robot
\( z_1 \) is the observation

\[
p(X, Z) = p(x_1, x_2, z_1) = p(x_2, z_1|x_1)P(x_1) \\
= p(x_2|x_1)p(z_1|x_1)p(x_1)
\]
Toy problem: Including landmarks

$x_1, x_2$ is the state of the robot
$z_1$ is the observation

$$p(X, Z) = p(x_1, l_1, z_1) = p(z_1 | x_1, l_1)p(x_1)p(l_1)$$
Toy problem:

$x_1, x_2, x_3$ are the states of the robot
$z_i$ is the observation
$l_i$ is the landmark

Similarly we get

$$p(X, Z) = p(x_1, x_2, x_3, l_1, l_2, z_1, z_2, z_3, z_4)$$

$$p(X, Z) = p(x_1)p(x_2|x_1)p(x_3|x_2)$$

$$\times p(l_1)p(l_2)$$

$$\times p(z_1|x_1)$$

$$\times p(z_2|x_1, l_1)p(z_3|x_2, l_1)p(z_4|x_3, l_2).$$
Gaussian Assumption

We will make an assumption that each of the probabilities are gaussians.

\[
p(X, Z) = p(x_1)p(x_2|x_1)p(x_3|x_2)
\times p(l_1)p(l_2)
\times p(z_1|x_1)
\times p(z_2|x_1, l_1)p(z_3|x_2, l_1)p(z_4|x_3, l_2).
\]
Maximum a Posteriori Inference

For SLAM we want:  \( p(X|Z) \)

We want this \( x \)
Maximum a Posteriori Inference

\[ X^{MAP} = \arg \max_X p(X|Z) \]

We know that:
\[ p(X|Z) = \frac{p(X,Z)}{P(Z)} \]

Therefore:
\[ X^{MAP} = \arg \max_X (p(X, Z)) \]
Maximum a Posteriori Inference

Remember that we had done this:

\[ p(X, Z) = p(x_1)p(x_2|x_1)p(x_3|x_2) \]
\[ \times p(l_1)p(l_2) \]
\[ \times p(z_1|x_1) \]
\[ \times p(z_2|x_1, l_1)p(z_3|x_2, l_1)p(z_4|x_3, l_2). \]

\[ X^{MAP} = \operatorname{argmax}_X \left( \begin{array}{c} p(x_1)p(x_2|x_1)p(x_3|x_2) \\ \times p(l_1)p(l_2) \\ \times p(z_1|x_1) \\ \times p(z_2|x_1, l_1)p(z_3|x_2, l_1)p(z_4|x_3, l_2). \end{array} \right) \]
Introducing factor graphs

- Each of the probability can be represented as a factor

\[
\begin{align*}
= p(x_1) p(x_2|x_1) p(x_3|x_2) \\
& \times p(l_1) p(l_2) \\
& \times p(z_1|x_1) \\
& \times p(z_2|x_1, l_1) p(z_3|x_2, l_1) p(z_4|x_3, l_2).
\end{align*}
\]

\[
= \phi_1(x_1) \phi_2(x_2, x_1) \phi_3(x_3, x_2) \\
& \times \phi_4(l_1) \phi_5(l_2) \\
& \times \phi_6(x_1) \\
& \times \phi_7(x_1, l_1) \phi_8(x_2, l_1) \phi_9(x_3, l_2),
\]
Introducing factor graphs

\[
X^{MAP} = \arg\max_X \phi(X) \\
= \arg\max_X \prod_i \phi_i(X_i)
\]
Expanding the factors

\[ \phi(x, l) = p(z|x, l) = \mathcal{N}(z; h(x, l), R) = \frac{1}{\sqrt{|2\pi R|}} \exp \left\{ -\frac{1}{2} \| h(x, l) - z \|^2_R \right\} \]

- Remember the Gaussian Assumption:

\[ \mathcal{N}(\theta; \mu, \Sigma) = \frac{1}{\sqrt{|2\pi \Sigma|}} \exp \left\{ -\frac{1}{2} \| \theta - \mu \|^2_{\Sigma} \right\}, \]

where \( \mu \in \mathbb{R}^n \) is the mean, \( \Sigma \) is an \( n \times n \) covariance matrix, and

\[ \| \theta - \mu \|^2_{\Sigma} \triangleq (\theta - \mu)^\top \Sigma^{-1} (\theta - \mu) \]
Expanding the factors

\[ \phi(x_{t+1}, x_t) = p(x_{t+1} | x_t, u_t) = \]
\[
\frac{1}{\sqrt{|2\pi Q|}} \exp \left\{-\frac{1}{2} \| g(x_t, u_t) - x_{t+1} \|^2_Q \right\}
\]

- Remember the Gaussian Assumption:

\[ \mathcal{N}(\theta; \mu, \Sigma) = \frac{1}{\sqrt{|2\pi \Sigma|}} \exp \left\{-\frac{1}{2} \| \theta - \mu \|^2_\Sigma \right\}, \]

where \( \mu \in \mathbb{R}^n \) is the mean, \( \Sigma \) is an \( n \times n \) covariance matrix, and

\[ \| \theta - \mu \|^2_\Sigma \stackrel{\Delta}{=} (\theta - \mu)^\top \Sigma^{-1} (\theta - \mu) \]
Expanding the factors

\[ X^{MAP} = \arg\max_X \prod_i \phi_i(X_i) \]

\[ = \arg\max_X \left( \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||h(x_1, l_1) - z_2|| \right\} \right) \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||h(x_2, l_1) - z_3|| \right\} \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||h(x_3, l_2) - z_4|| \right\} \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||g(x_1, u_1) - x_2|| \right\} \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||g(x_2, u_2) - x_3|| \right\} \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||f_1(x_1) - x_1|| \right\} \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||f_2(x_1) - x_1|| \right\} \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||o(l_1) - l_1|| \right\} \]

\[ \frac{1}{\sqrt{2\pi R}} \exp \left\{ -\frac{1}{2} ||o(l_2) - l_2|| \right\} \]
MAP

So, each of the factors can be represented as:

\[ \phi_i(X_i) \propto \exp \left\{ -\frac{1}{2} \left\| h_i(X_i) - z_i \right\|_{\Sigma_i}^2 \right\} , \]

Can be rewritten as follows:

\[ X^{MAP} = \arg\min_X \sum_i \left\| h_i(X_i) - z_i \right\|_{\Sigma_i}^2 . \]

We know that:

\[ X^{MAP} = \arg\max_X \phi(X) \]

\[ = \arg\max_X \prod_i \phi_i(X_i) \]
Linearization

using a simple Taylor expansion, we get:

\[ h_i(X_i) = h_i(X_i^0 + \Delta_i) \approx h_i(X_i^0) + H_i \Delta_i, \]

- \( H \) is the measurement Jacobian, which is written as:

\[ H_i \triangleq \left. \frac{\partial h_i(X_i)}{\partial X_i} \right|_{X_i^0}, \]

- \( \Delta \) is the state update vector, which is written as:

\[ \Delta_i \triangleq X_i - X_i^0 \]
Linearization

\[
\Delta^* = \text{argmin}_\Delta \sum_i \left\| h_i(X^0_i) + H_i \Delta_i - z_i \right\|_{\Sigma_i}^2
\]

\[
= \text{argmin}_\Delta \sum_i \left\| H_i \Delta_i - \left\{ z_i - h_i(X^0_i) \right\} \right\|_{\Sigma_i}^2,
\]

- Rewrite the Mahalanobis norm as follows:

\[
\| e \|_\Sigma^2 \overset{\Delta}{=} e^T \Sigma^{-1} e = \left( \Sigma^{-1/2} e \right)^T \left( \Sigma^{-1/2} e \right) = \| \Sigma^{-1/2} e \|_2^2.
\]

\[
\Delta^* = \text{argmin}_\Delta \sum_i \left\| \Sigma_i^{1/2} H_i \Delta_i - \Sigma_i^{1/2} \left\{ z_i - h_i(X^0_i) \right\} \right\|_2^2
\]
Converting to Least Squares

Let:

\[ A_i = \Sigma_i^{-1/2} H_i \]
\[ b_i = \Sigma_i^{-1/2} \left( z_i - h_i(X_i^0) \right) . \]

We finally arrive at the form:

\[ \Delta^* = \arg\min_{\Delta} \sum_i \| A_i \Delta_i - b_i \|_2^2 \]
\[ = \arg\min_{\Delta} \| A \Delta - b \|_2^2 , \]
Solving the Least Squares

\[
\Delta^* = \arg\min_{\Delta} \sum_i \|A_i \Delta_i - b_i\|^2_2 \\
= \arg\min_{\Delta} \|A\Delta - b\|^2_2 \\
\]

\[
\|A\Delta - b\|^2_2 = (A\Delta - b)^T (A\Delta - b) \\
\]

\[
\|A\Delta - b\|^2_2 = \Delta^T A^T A\Delta - 2\Delta^T A^T b + b^T b \\
\]

In order to minimize the error we do:

\[
\frac{\partial \|A\Delta - b\|}{\partial \Delta} = 0 \quad \Rightarrow \quad A^T A\Delta = A^T b 
\]
The Measurement Jacobian

- Each factor represents a constraint between two variables.
- Therefore, the measurement Jacobian is a sparse matrix.
- As $A$ is sparse, the $A^TA$ is also sparse.
- We can use sparse methods which are fast.
Methods for solving the least-squares problem

\[ A^T A \Delta = A^T b \]

Calculating \((A^T A)^{-1}\) is a bad idea \(\rightarrow\) \(O(n^3)\)

\[ \Delta = (A^T A)^{-1} A^T b \]

Use the Cholesky decomposition

\[ A^T A = R^T R \]

- For sparse matrices - \(O(m^{1.5})\) to \(O(m^2)\)

\[ R^T R \Delta = A^T b \]

- Finally use the forward substitution and backward substitution to solve
Graph SLAM Summary

- Full SLAM technique
- Graph SLAM leads to sparse matrices
- Suited for Large scale SLAM.
- Batch optimization of multiple sensor measurements.
SLAM Least-Squares Example

Localyze robot and door based on 1D range measurements

\[ t = 0 \]
\[ d_0 = 2m \]
\[ t = 1 \]
\[ d_1 = 1m \]
\[ \text{door} \]
\[ t = 2 \]
\[ d_2 = -1m \]

\[ u_1 = 1m \]
\[ u_2 = 2m \]

Measurements: distance to the door, signed

Factor graph:

Borrowed from Prof. Michael Keass
Least Square Example

We know that:

\[
\theta^* = \arg\min_{\theta} \sum_i \left\| h_i(\theta) - z_i \right\|^2_{\Sigma_i}
\]

\[
= \arg\min_{\theta} \left( \left\| h_p(x_0) - p \right\|^2_{\Sigma_p} + \left\| h_u(x_0, x_1) - u_1 \right\|^2_{\Sigma_u} + \left\| h_u(x_1, x_2) - u_2 \right\|^2_{\Sigma_u} + \left\| h_d(x_0, l) - d_0 \right\|^2_{\Sigma_d} + \left\| h_d(x_1, l) - d_1 \right\|^2_{\Sigma_d} + \left\| h_d(x_2, l) - d_2 \right\|^2_{\Sigma_d} \right)
\]
Least Square Example

\[ h_u(x_a, x_b) = x_b - x_a \quad \sigma_u = 0.1m \]
\[ h_d(x, l) = l - x \quad \sigma_d = 0.01m \]
\[ h_p(x) = x \quad \sigma_p = 0.1m \]

\[ \| h(x_0, x_1) - u_1 \|_{\Sigma_u}^2 = \| \frac{x_1 - x_0}{\sigma_u} - \frac{u_1}{\sigma_u} \|_2^2 \]
\[ = \| -10x_0 + 10x_1 - 10 \|_2^2 \]
Least Square Example

\[ A = \begin{bmatrix}
10 & 0 & 0 & 0 \\
-10 & 10 & 0 & 0 \\
0 & -10 & 0 & 0 \\
-100 & 0 & 0 & 100 \\
0 & -100 & 0 & 100 \\
0 & 0 & -100 & 100 \\
\end{bmatrix} \quad b = \begin{bmatrix}
0 \\
10 \\
20 \\
200 \\
100 \\
-100 \\
\end{bmatrix} \]

- Solve the following least squares problem
SLAM Least-Squares Example

Localize robot and door based on 1D range measurements

Matrix A:
Each row corresponds to a factor
Each column to a variable
A is sparse!
Sparse Factorization Example

Example from real sequence:
Square root inf. matrix
Side length: 21000 variables
Dense: 1.7GB, sparse: 1MB

233499 non-zeros
\approx 0.1\% density
\approx 11/column

Borrowed from Prof. Michael Keass
Graph-based SLAM - Intel, 2011
Modern SLAM

A typical SLAM system...
Visual Inertial SLAM - Options

Front End
- RGB Cameras:
  - Direct methods
  - Indirect methods
- Feature Tracking:
  - KLT tracker
- IMU:
  - IMU preintegration
- Loop Closures

Back End
- Smoothing and Mapping
- Filtering
Visual Inertial SLAM - Front end

- Extracts relevant features from the sensor data.
# Front end - Direct vs Indirect methods

<table>
<thead>
<tr>
<th>Indirect Methods</th>
<th>Direct Methods</th>
</tr>
</thead>
</table>
| ● Feature-based approaches are quite mature, with a long history of success  
  ● System depends on the availability of features in the environment, the reliance on detection and matching thresholds.  
  ● E.g ORB-SLAM | ● System works with the raw pixel information and dense-direct methods exploit all the information in the image.  
  ● Can outperform feature-based methods in scenes with poor texture, defocus, and motion blur.  
  ● Require high computing power (GPUs) for real-time performance.  
  ● E.g. DSO-SLAM |

**Hybrid Methods: SVO**
- The algorithm uses sparse model-based image alignment for motion estimation
- The algorithm uses point-features for BA
Direct vs Indirect methods

https://youtu.be/C6-xwSOOdqQ
## Back-end and comparisons

<table>
<thead>
<tr>
<th>Smoothing and Mapping</th>
<th>Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Enables an insightful visualization of the problem.</td>
<td>● Proven to be less accurate and efficient compared to smoothing methods</td>
</tr>
<tr>
<td>● Factor graphs can model complex inference problems</td>
<td>● Some of the SLAM systems based on EKF have also been demonstrated to attain state-of-the-art performance. E.g. Multistate Constraint Kalman Filter.</td>
</tr>
<tr>
<td>● The connectivity of the factor graph in turn influences the sparsity of the resulting SLAM problem</td>
<td></td>
</tr>
</tbody>
</table>
Backend examples...
Comparison of Monocular Visual-Inertial Odometry

Algorithms being compared

- **MSCKF**: An Extended Kalman Filter (EKF)-based algorithm for real-time vision-aided inertial navigation [2007].
- **Open Keyframe-based Visual-Inertial SLAM (OKVIS)** utilizes non-linear optimization on a sliding window of keyframe poses.
- **ROVIO**: Visual-Inertial state estimator based on an extended Kalman Filter (EKF), which proposed several novelties.
- **VINS-Mono**: A nonlinear-optimization-based sliding window estimator using pre-integrated IMU factors.
- **SVO+GTSAM**: SVO in front end paired with a full-smoothing backend performing online factor graph optimization using iSAM2.
Comparison of Translation Error

![Comparison of Translation Error](image_url)
Comparison of Yaw Errors
Algorithm Efficiency

![Chart showing algorithm efficiency with CPU and memory usage vs. RMSE error]
Deep Learning for SLAM
TartanVO

<table>
<thead>
<tr>
<th></th>
<th>Seq.</th>
<th>MH-04</th>
<th>MH-05</th>
<th>VR1-02</th>
<th>VR1-03</th>
<th>VR2-02</th>
<th>VR2-03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry-based *</td>
<td>SVO [46]</td>
<td>1.36</td>
<td>0.51</td>
<td>0.47</td>
<td>x</td>
<td>0.47</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>ORB-SLAM [3]</td>
<td><strong>0.20</strong></td>
<td>0.19</td>
<td>x</td>
<td>x</td>
<td><strong>0.07</strong></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>DSO [5]</td>
<td>0.25</td>
<td><strong>0.11</strong></td>
<td>0.11</td>
<td>0.93</td>
<td>0.13</td>
<td>1.16</td>
</tr>
<tr>
<td></td>
<td>LSD-SLAM [2]</td>
<td>2.13</td>
<td>0.85</td>
<td>1.11</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Learning-based †</td>
<td>TartanVO (ours)</td>
<td>0.74</td>
<td>0.68</td>
<td>0.45</td>
<td><strong>0.64</strong></td>
<td>0.67</td>
<td><strong>1.04</strong></td>
</tr>
</tbody>
</table>

* These results are from [46]. † Other learning-based methods [36] did not report numerical results.
TartanVO

however, geometry-based VO methods are not robust enough in difficult cases