Lecture 28, Mar 18, 2026

Batch SLAM

- Filtering-based SLAM keeps only the most recent pose, leading to issues over long-term
  - Filters lock in their decisions; since we don't do global optimization, new information doesn't always allow us to correct for earlier mistakes, e.g. large loop closures
    * EKF-SLAM needs uncertainties to be kept perfects to backpropagate the loop closure uncertainty
    * Particle filters need lots of particles to keep track of the growing uncertainty and map distribution
  - By marginalizing out previous states (i.e. removing $\boldsymbol{x}_{k-1}$), EKF-SLAM destroys the sparse arrowhead structure of the information matrix as all landmark poses start being correlated
- In batch SLAM, we keep previous poses around and perform global optimization upon loop closure, similar to bundle adjustment
  - We build a graph using odometry measurements and landmark observations, with nodes representing poses/landmarks and edges representing measurements
  - The SLAM problem reduces to finding the configuration of the poses that minimizes a cost
  - Analogous to a network of springs where the stiffness of each spring is the confidence of the measurement, and finding the minimum energy configuration
- Consider landmarks with location $\boldsymbol{m} = \begin{bmatrix} x \\ y \end{bmatrix}$, initial pose $\boldsymbol{x}_0 = \boldsymbol{0}$, with odometry measurements $\boldsymbol{u} = \begin{bmatrix} u \\ \omega \end{bmatrix}$ and landmark measurements $\boldsymbol{y} = \begin{bmatrix} r \\ \phi \end{bmatrix}$; we want to find robot poses $\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$
  - The full state consists of stacking all the poses and landmark locations into $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_k & \boldsymbol{m}_{1,1} & \cdots & \boldsymbol{m}_{k,l} \end{bmatrix}^T$
  - Consider a motion model $\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k), \boldsymbol{w}_k \sim (\boldsymbol{0}, \boldsymbol{Q})$ and an observation model $\boldsymbol{y}_{kl} = \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{m}_l, \boldsymbol{n}_k), \boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R})$
    * Often we use a unicycle model and a range and bearing to landmarks:
    * $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + T \begin{bmatrix} \cos\theta_{k-1} & 0 \\ \sin\theta_{k-1} & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{u}_k + \boldsymbol{w}_k$
    * $\boldsymbol{y}_{kl} = \begin{bmatrix} \|\boldsymbol{x}_l - \boldsymbol{x}_k\| \\ \operatorname{atan2}(y_l - y_k, x_l - x_k) \end{bmatrix} + \boldsymbol{n}_k$
  - We form a motion model error $\boldsymbol{e}_k = \boldsymbol{x}_k - \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{0}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q})$ and observation model error $\boldsymbol{e}_{kl} = \boldsymbol{y}_{kl} - \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{m}_l, \boldsymbol{0}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R})$
  - Define a quadratic loss: $J = \frac{1}{2} \sum_{k=1}^{K} \boldsymbol{e}_k^T \boldsymbol{Q}^{-1} \boldsymbol{e}_k + \frac{1}{2} \sum_{k=1}^{K} \sum_{l=1}^{L} \boldsymbol{e}_{kl}^T \boldsymbol{R}^{-1} \boldsymbol{e}_{kl}$
    * This is equivalent to finding the $\boldsymbol{z}$ that maximizes the likelihood of the measurements, assuming independence of measurements with each other and inputs, and using Gaussians
- To optimize the cost, we linearize the cost for a small step $\delta\boldsymbol{x}$, find the optimal step, and repeat
  - $$\begin{aligned} \boldsymbol{e}_k &= \boldsymbol{x}_k - \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{0}) \\ &= \bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k - \boldsymbol{f}(\bar{\boldsymbol{x}}_{k-1} + \delta\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{0}) \\ &= \bar{\boldsymbol{x}}_k - \boldsymbol{f}(\bar{\boldsymbol{x}}_{k-1} + \delta\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{0}) + \delta\boldsymbol{x}_k - \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \delta\boldsymbol{x}_{k-1} \\ &= \bar{\boldsymbol{e}}_k + \delta\boldsymbol{x}_k - \boldsymbol{F}_{k-1} \delta\boldsymbol{x}_{k-1} \end{aligned}$$
  - $$\begin{aligned} \boldsymbol{e}_{kl} &= \boldsymbol{y}_k - \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{m}_l, \boldsymbol{0}) \\ &= \boldsymbol{y}_k - \boldsymbol{g}(\bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \bar{\boldsymbol{m}}_l + \delta\boldsymbol{m}_l, \boldsymbol{0}) \\ &= \bar{\boldsymbol{y}}_k - \boldsymbol{g}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{m}}_l, \boldsymbol{0}) - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \delta\boldsymbol{x}_k - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{m}} \boldsymbol{m}_l \\ &= \bar{\boldsymbol{e}}_{kl} - \boldsymbol{G}_{x,kl} \delta\boldsymbol{x}_k - \boldsymbol{G}_{m,kl} \delta\boldsymbol{m}_l \end{aligned}$$

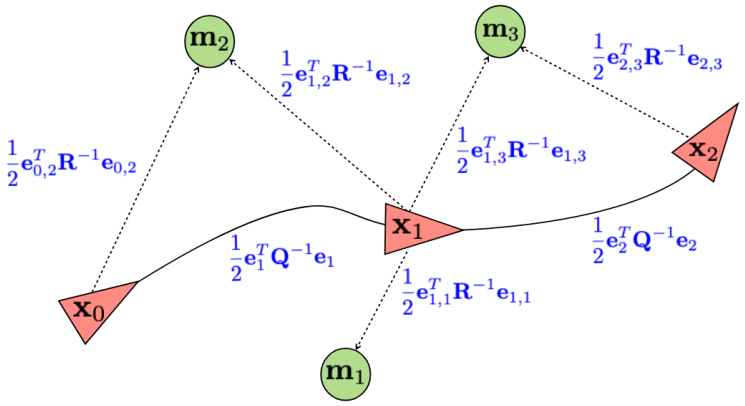

Figure 1: Cost terms for a simple batch SLAM problem.

- Perturbed cost: $J = \frac{1}{2}\sum_{k=1}^{K}\left(\bar{\boldsymbol{e}}_k + \delta\boldsymbol{x}_k - \boldsymbol{F}_{k-1}\delta\boldsymbol{x}_{k-1}\right)^T \boldsymbol{Q}^{-1}\left(\bar{\boldsymbol{e}}_k + \delta\boldsymbol{x}_k - \boldsymbol{F}_{k-1}\delta\boldsymbol{x}_{k-1}\right)$
  $+ \frac{1}{2}\sum_{k=1}^{K}\sum_{l=1}^{L}\left(\bar{\boldsymbol{e}}_{kl} - \boldsymbol{G}_{x,kl}\delta\boldsymbol{x}_k - \boldsymbol{G}_{m,kl}\delta\boldsymbol{m}_l\right)^T \boldsymbol{R}^{-1}\left(\bar{\boldsymbol{e}}_{kl} - \boldsymbol{G}_{x,kl}\delta\boldsymbol{x}_k - \boldsymbol{G}_{m,kl}\delta\boldsymbol{m}_l\right)$
  $= \frac{1}{2}(\bar{\boldsymbol{e}} + \boldsymbol{E}\delta\boldsymbol{z})^T\boldsymbol{T}^{-1}(\bar{\boldsymbol{e}} + \boldsymbol{E}\delta\boldsymbol{z})$
  - Here $\boldsymbol{T}$ is a large block diagonal matrix with the individual covariance matrices $\boldsymbol{Q}, \boldsymbol{R}$ repeated as many times as necessary along the diagonal
- To get the optimal update we solve $\boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E}\delta\boldsymbol{z}^* = -\boldsymbol{E}^T\boldsymbol{T}^{-1}\bar{\boldsymbol{e}}$; starting with an initial guess (odometry and first observed landmark locations), we solve for $\delta\boldsymbol{z}^*$ and update $\bar{\boldsymbol{z}} \leftarrow \bar{\boldsymbol{z}} + \delta\boldsymbol{z}^*$ and iterate until $\delta\boldsymbol{z}^*$ is sufficiently small
- Since this method assumes locally quadratic costs, in practice we use the following techniques to improve robustness:
  1. *Levenberg-Marquardt* iteration: $(\boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E} + \mu\,\mathrm{diag}(\boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E}))\delta\boldsymbol{z}^* = -\boldsymbol{E}^T\boldsymbol{T}^{-1}\bar{\boldsymbol{e}}$ where μ is a tunable parameter and $\mathrm{diag}(\boldsymbol{A})$ means a diagonal matrix obtained by setting all off-diagonal terms of $\boldsymbol{A}$ to 0
     - Helps when $\boldsymbol{A}$ is noninvertible and prevents large jumps for highly nonlinear functions
  2. Line search: After obtaining $\delta\boldsymbol{z}^*$, do a line search along its direction to make sure we don't go too far; do $\bar{\boldsymbol{z}} \leftarrow \bar{\boldsymbol{z}} + \alpha\delta\boldsymbol{z}$ where we pick the largest $\alpha \in [0, 1]$ that still decreases J
- $\boldsymbol{A} = \boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E}$ is the precision or inverse covariance of the final estimate of $\bar{\boldsymbol{z}}$; it is sparse with an arrowhead structure, so the Schur complement can be used for inversion

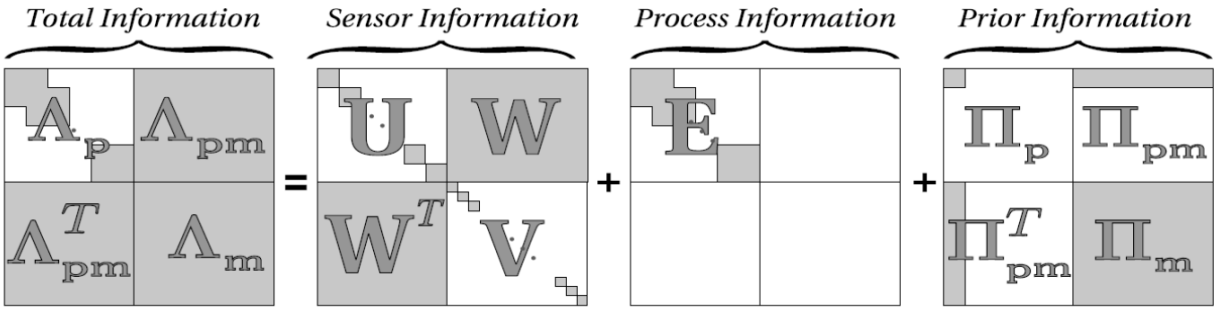

Figure 2: Structure of the inverse covariance matrix. With no prior, we obtain the sparse arrowhead structure.

- Running batch SLAM online requires changes to limit the growth of the problem

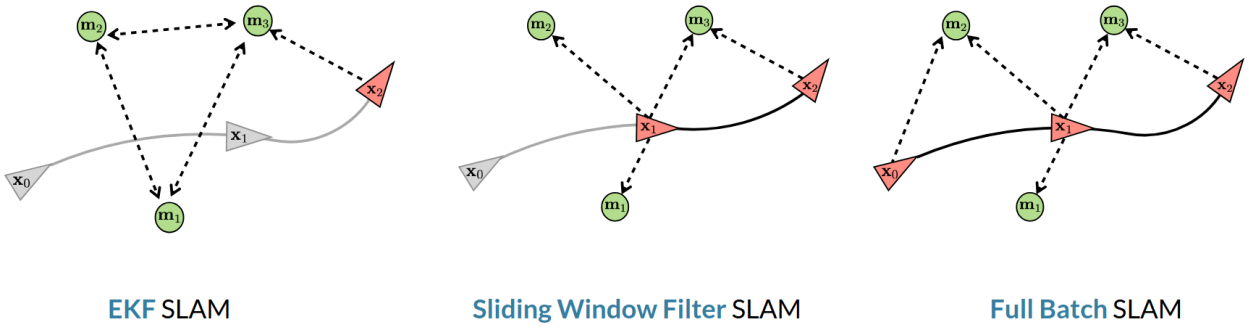

Figure 3: Sliding window filter SLAM is a hybrid between EKF-SLAM and full batch SLAM.

- Sliding window filter: marginalize out old poses and invisible landmarks beyond the sliding window, but keeps poses within the window
  * This is similar to a hybrid between EKF-SLAM/filtering based methods and batch SLAM
  * However loop closure is still a problem
- Adaptive relative bundle adjustment: represent poses relatively, and use an adaptive local window of active landmarks and poses for optimization
  * This takes advantage of the fact that in a large scene, typically most of the map changes very little, so most of it does not need to be optimized over
  * Landmark poses are represented relative to the robot pose that observed them, and robot poses are represented relative to the last pose
    - This removes the need for a single privileged global coordinate frame, which makes optimization costly for very large problems, since with a global frame a small change in the pose of the first steps leads to huge swings in the poses of subsequent steps
  * Only landmarks and robot poses within the active window are optimized
  * The active window adapts based on which poses have seen the current landmarks and the current error
    - On loop closure, it starts by activating the most recent poses, and activates more as needed until the error drops below a threshold
    - This avoids having to optimize over every landmark and pose for a loop closure