Introduction to Mobile Robotics

Lecture 7: SLAM

Zhi Yan

ENSTA, Institut Polytechnique de Paris

Recap: Why We Are Here

- The "sense-think-act" loop.
- To "think" (e.g., plan a path from A to B), a robot needs two things:
  1. A map of the world ("What does the world look like?").
  2. Its pose within that map ("Where am I?").

The problem:

For a robot in a new environment, the answer to both questions is: "I don't know."

ENSTA IP PARIS

The Failure of Odometry

- Recap: Odometry (or "Dead Reckoning") integrates velocity estimates (v, ω) from encoders/IMU over time:

$$P_k = P_{k-1} + \Delta t \cdot f(v, \omega)$$

  where f represents a kinematic model function (cf. Lecture 5).

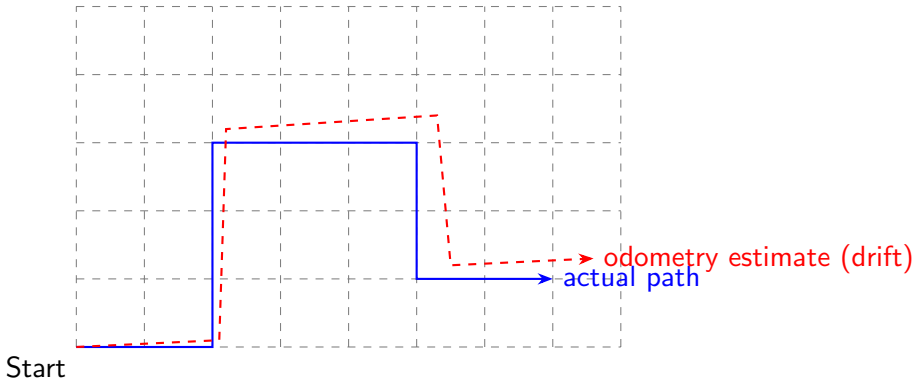
- This is an **open-loop** estimate: It only looks at *internal* (proprioceptive) sensors (cf. Lecture 6).

- **Sources of error:**
  - Wheel slippage on smooth floors.
  - Bumps or uneven ground.
  - Imperfect kinematic model parameters.
  - etc.

ENSTA IP PARIS

The Unavoidable Truth: Drift

- These small, random errors are *integrated* (summed up) over time.
- The error in pose grows non-linearly with time/distance $\implies$ **drift**.



Odometry is reliable for short-term (e.g., seconds), but guaranteed to fail over long-term (e.g., minutes).

ENSTA IP PARIS

SLAM: The Chicken-and-Egg Problem

Definition:

SLAM (Simultaneous Localization and Mapping) is the process by which a robot builds a map of an unknown environment while simultaneously estimating its own position within that map.

The paradox:

- **Mapping requires localization:** To place a new sensor scan (S_k) on the map, you must know the robot's exact pose. A bad pose estimate creates a blurry or duplicated map.
- **Localization requires a map:** To find your pose, you must match your current sensor scan against an existing map ("I see a wall on my left, so I must be in the hallway.").

SLAM:

Solving for both unknowns at the same time.

ENSTA IP PARIS

How to Represent a Map?

1. Feature/Landmark-based map
   - Map is a set of landmark positions: $M = \{L_1, L_2, ..., L_N\}$.
   - Landmarks can be corners, trees, or visual features (e.g., in V-SLAM).
   - Used in classic EKF-SLAM and most Visual SLAM systems.

2. Occupancy grid map
   - Discretize the world into a grid of cells (e.g., 5cm x 5cm).
   - Each cell m_i stores the probability $P(m_i = \text{occupied})$.
   - This is the standard for 2D LiDAR SLAM (like `slam_toolbox`).

ENSTA IP PARIS

Occupancy Grids: The Math

We want to compute $P(m_i|z_{1:k}, x_{1:k})$: probability a cell is occupied given all sensor readings z and poses x.

$\Longrightarrow$ This is hard to compute directly.

Solution: log-odds representation:

- Define the *odds* of an event as: $\text{odds}(A) = \frac{P(A)}{1-P(A)}$
- Define the *log-odds* as: $l_i = \log \frac{P(m_i)}{1-P(m_i)}$
- This maps probability $P \in [0, 1]$ to log-odds $l \in [-\infty, +\infty]$.

Why? A complex Bayesian update becomes simple addition:

$$l_k = l_{k-1} + \text{InverseSensorModel}(z_k, x_k)$$

$\Longrightarrow$ This is extremely fast and efficient!

ENSTA IP PARIS

Occupancy Grids in ROS 2

A YAML file contains metadata about the occupancy grid (cf. Practical work 7):

- `image: my_map.pgm`: The `.pgm` image file (Portable GrayMap).
- `mode: trinary`: The map uses three values (not just two).
- `resolution: 0.050000`: Each pixel is 0.05 meters (5cm).
- `origin: [-1.000000, -1.000000, 0.000000]`: The real-world (x, y, θ) coordinate of the map's bottom-left pixel.
- `negate: 0`: The white/black free/occupied semantics shouldn't be reversed.
- `occupied_thresh: 0.65`: Probability $> 65\% \rightarrow$ Occupied (black).
- `free_thresh: 0.25`: Probability $< 25\% \rightarrow$ Free (white).

Question:

What about values between 0.25 and 0.65?

ENSTA IP PARIS

Occupancy Grids in ROS 2

A YAML file contains metadata about the occupancy grid (cf. Practical work 7):

- `image: my_map.pgm`: The `.pgm` image file (Portable GrayMap).
- `mode: trinary`: The map uses three values (not just two).
- `resolution: 0.050000`: Each pixel is 0.05 meters (5cm).
- `origin: [-1.000000, -1.000000, 0.000000]`: The real-world (x, y, θ) coordinate of the map's bottom-left pixel.
- `negate: 0`: The white/black free/occupied semantics shouldn't be reversed.
- `occupied_thresh: 0.65`: Probability $> 65\% \rightarrow$ Occupied (black).
- `free_thresh: 0.25`: Probability $< 25\% \rightarrow$ Free (white).

Question:

What about values between 0.25 and 0.65?

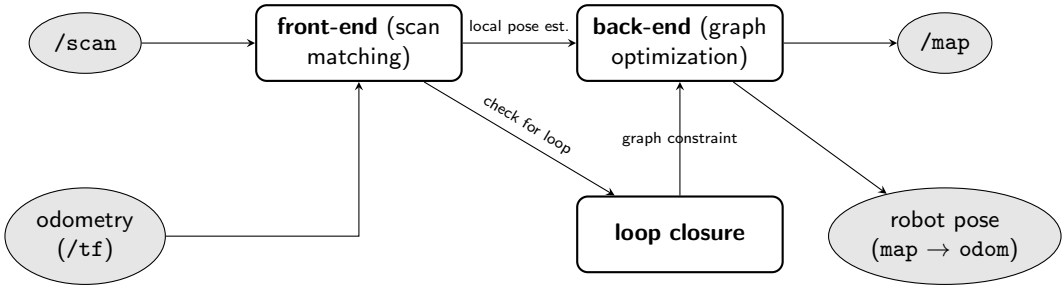
Answer:

That is "unknown" (gray).

ENSTA IP PARIS

The Modern SLAM Pipeline

A modern SLAM system is split into two parts:



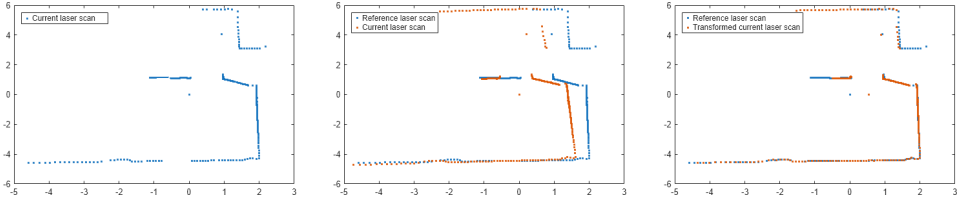
- **Front-end:** Runs fast (e.g., 10 Hz).
  $\Longrightarrow$ Manages sensor data and estimates motion *frame-to-frame*.
- **Back-end:** Runs slower (e.g., 1 Hz).
  $\Longrightarrow$ Optimizes the *entire history* of poses to ensure global consistency.

ENSTA IP PARIS

The Front-End's Job: Scan Matching

The Goal/Question:

Given a new scan S_k (from LiDAR) and a reference (either the previous scan S_{k-1} or the map M built so far), what is the **transformation** T (i.e., the robot's movement $\Delta x, \Delta y, \Delta\theta$) that best aligns S_k with the reference?



Left: map/scan S_{k-1} (blue). Center: new scan S_k (orange) with odometry guess. Right: S_k (orange) after alignment[1].

This provides a motion estimate based on what the robot *sees*,
not just what its wheels *did*. $\Longrightarrow$ The **correction** (an "odometry constraint").

[1] https://www.mathworks.com/help/nav/ug/estimate-robot-pose-with-scan-matching.html

ENSTA IP PARIS

Iterative Closest Point (ICP)

ICP is the most famous scan-matching algorithm.

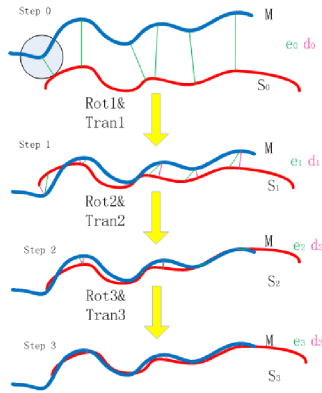
Goal:
Find the transformation T that minimizes the distance between two point clouds (the new scan and the reference).

icp-animation.gif

ENSTA
IP PARIS

ICP: The Algorithm

1. **Initial guess:** Use odometry as the initial guess for T.
2. **Associate:** For each point in the new scan S_k, find its *closest* corresponding point in the reference scan S_{k-1} (or map M).
3. **Optimize:** Calculate the transformation T (rotation + translation) that minimizes the sum of squared distances between these associated pairs.
4. **Iterate:** Apply the new T to S_k. Go back to step 2.
5. **Converge:** Stop when the change in T is very small or after N iterations.



ENSTA
IP PARIS

ICP: Challenges & Variants

Challenges:

- Sensitive to the initial guess (e.g., odometry).
- Difficulty in handling ambiguous/featureless places (e.g., a long and empty corridor).

Variants:

- Point-to-Point ICP: The classic algorithm we described.
- Point-to-Plane ICP: Minimizes the distance from a point to the *plane* of its neighbors.
- And many others (e.g., Generalized-ICP, NDT).

ENSTA
IP PARIS

The Front-End Isn't Enough

- The front-end (ICP) is fast, but it only compares S_k to S_{k-1} (or a local map).
- It's just a "chain" of measurements, and each link in the chain may have a tiny error.
- $error(S_1 \to S_2) + error(S_2 \to S_3) + ... + error(S_{99} \to S_{100})$
- The drift **still accumulates**, just much slower than odometry.
- We need a way to correct the *entire* chain.

ENSTA
IP PARIS

The Pose Graph

The **pose graph** is the back-end's main data structure, including:

- **Nodes (P_i):** Represent **robot poses** (x, y, θ) at different moments in time.
- **Edges ($T_{i \to j}$):** Represent **spatial constraints** (movements) between poses.

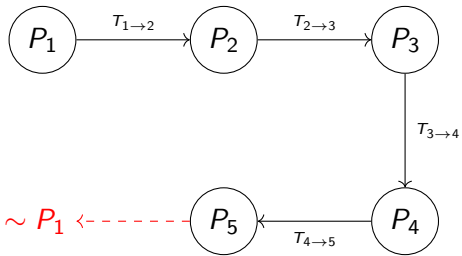
Types of edges:

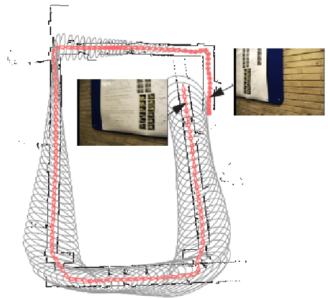
- *Odometry/Scan-match constraints* (from front-end): "The robot moved by $T_{1 \to 2}$ between P_1 and P_2."
  $\Longrightarrow$ Has some uncertainty.
- *Loop closure constraints*: "The robot recognized that P_5 is in the same place as P_1!"
  $\Longrightarrow$ Has very high certainty.

ENSTA IP PARIS

Visualizing the Graph (Before Loop Closure)

The graph is built by the front-end, containing all the accumulated drift:





A map built (Ho and Newman 2007) with only a front-end. Note the visible drift where the hallways fail to align.

ENSTA IP PARIS

Loop Closure

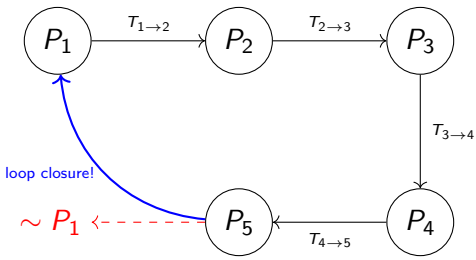
Loop closure is the most critical step in SLAM:

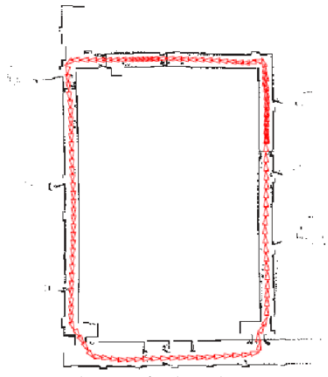
- It is the act of **recognizing a previously visited place**.
- This is a **Place Recognition** problem.
- **How?** The system compares the current scan S_k to *all* old scans ($S_1 ... S_{k-1}$), not just the last one.
- This is done using "Scan Descriptors" (e.g., histograms, FPFH) that are fast to compare.
- When a match is found (e.g., $S_k \approx S_j$), a new **loop closure constraint** (edge) is added to the graph.

ENSTA
IP PARIS

Visualizing Loop Closure

This new edge creates **tension** in the graph:





The map after loop closure.

The odometry chain $P_1 \to \cdots \to P_5$ says the robot is at P_5, but the loop closure says P_5 is at P_1!

ENSTA IP PARIS

The Back-End's Job: Graph Optimization

- The Back-End is a **Global Optimizer**.
- Its job: Find the set of poses $P_1 ... P_n$ that **minimizes the error** of *all* constraints in the graph *simultaneously*.
- This is a massive **Non-Linear Least-Squares** problem.
- We are finding the "most likely" map and trajectory given *all* the evidence.

ENSTA
IP PARIS

The "Springs" Analogy

- Think of the graph as a system of springs:
- **Nodes:** Connectors.
- **Edges:** Springs.
- **Odometry edges:** Weak, soft, "stretched" springs (low certainty).
- **Loop closure edges:** Very strong, stiff springs (high certainty).

When the optimizer runs, it "lets go" of the system: The strong loop-closure spring "snaps" the graph into its lowest-energy, most consistent state.

ENSTA
IP PARIS

The Three Key Coordinate Frames

Understanding SLAM in ROS 2 requires understanding three coordinate frames:

- `/base_link`
  - The robot's frame. $\implies$ Always at $(0, 0)$ relative to the robot's center.
  - This is where the `/scan` data originates.
- `/odom`
  - The odometry frame. This is the drifting, open-loop estimate from encoders/IMU.
  - It is **continuous and smooth**, but becomes **inaccurate** over time.
- `/map`
  - The SLAM frame. This is the globally consistent, optimized frame.
  - It is **accurate**, but it is **not smooth** (it "jumps" during corrections).

ENSTA IP PARIS

The map → odom Transform

The TF tree is structured like this:

$$\texttt{map} \xrightarrow{\text{published by SLAM}} \texttt{odom} \xrightarrow{\text{published by Odometry}} \texttt{base_link}$$

- The robot's odometry publishes the odom → base_link transform.
- The slam_toolbox node publishes the map → odom transform.
  - The **correction**.
  - When the front-end makes a small correction, this transform "jumps" slightly.
  - When the back-end makes a big correction, this transform "jumps" significantly.

ENSTA IP PARIS

Post-lecture Reading: SLAM Algorithm Families

SLAM Family 1: Filter-based (EKF-SLAM)

- The "classic" approach.
- Core idea: Maintain one large state vector and one massive covariance matrix.
- State vector: $[\underbrace{x_v, y_v, \theta_v}_{\text{robot pose}} | \underbrace{x_1, y_1, ...x_N, y_N}_{\text{all N landmark positions}}]$
- Covariance matrix: An $(N+3) \times (N+3)$ matrix of all uncertainties.

Pros:

- Well-understood, proven.
- Provides full uncertainty.

Cons:

- Computationally scales $O(N^2)$ with N landmarks.
- Fails for large maps.
- Prone to failure on bad data association.

ENSTA IP PARIS

SLAM Family 2: Particle Filters (PF-SLAM)

- Example: `gmapping` (the ROS 1 standard)
- Core idea: Uses a **Rao-Blackwellized Particle Filter**.
- Each particle represents a hypothesis of the robot's trajectory.
- Each particle maintains its own individual map.
- Particles with bad maps (that don't match sensor data) "die" (low probability).
- Particles with good maps "survive" and are "resampled" (copied).

Pros:

- Very robust to unknown initial pose.
- Handles non-Gaussian noise well.

Cons:

- Computationally heavy (many particles).
- Map quality degrades over large areas.
- Loop closure is difficult and not as robust as graph methods.

ENSTA IP PARIS

SLAM Family 3: Graph-based (Graph-SLAM)

- Examples: `slam_toolbox` (the ROS 2 standard)
- Core idea: The pose graph optimization we just discussed.
- Separates the problem into front-end (local scan-matching) and back-end (global pose graph optimization) .

Pros:

- **The modern standard.**
- **Scales extremely well** for large maps.
- Produces **globally consistent** maps.
- Robust loop closure.

Cons:

- Optimization can be a complex batch process.
- Can be more complex to configure.

ENSTA IP PARIS

SLAM Family 4: Visual SLAM (V-SLAM)

- Examples: `ORB-SLAM3`, `VINS-Mono`
- Core idea: Uses **cameras** (mono, stereo, RGB-D) instead of LiDAR.
- "Landmarks" are visual features (e.g., corners, blobs).

Pros:

- Cheap, passive sensors (cameras).
- Provides very rich 3D data.
- Excellent for relocalization (place recognition).

Cons:

- Needs good light.
- Struggles on texture-less surfaces (e.g., white walls).
- Scale ambiguity (with mono camera): Is it a small object nearby or a big one far away?

ENSTA IP PARIS

Summary of SLAM Approaches

Method	Idea	Pros	Cons
Filter-based (e.g., EKF-SLAM)	maintain one large state vector & covariance	fast (small maps) proven	scales poorly $O(N^2)$, prone to bad data association
Particle-based (e.g., `gmapping`)	many trajectory "hypotheses" (particles)	robust to noise	computationally heavy, loop closure is weak
Graph-based (e.g., `slam_toolbox`)	optimize a graph of robot poses (nodes) & constraints (edges)	scales well, globally consistent, modern standard	more complex, batch optimization
Visual SLAM (e.g., ORB-SLAM)	use camera features as landmarks	rich 3D data, cheap cameras	needs light, no texture = fail

ENSTA IP PARIS

Summary

- **Odometry drifts:** Unavoidable, so we have to use exteroceptive sensors (LiDAR, camera) to correct it.
- **SLAM** solves this by localizing and mapping simultaneously.
- **Front-end (e.g., ICP):** Provides fast, local correction (scan-matching), but drifts over time.
- **Back-end (e.g., graph optimization):** Provides slower, global correction by optimizing the entire history with loop closure.

ENSTA
IP PARIS

Questions?

Next: Practical Work 7 - `slam_toolbox`

ENSTA IP PARIS