Introduction to Mobile Robotics

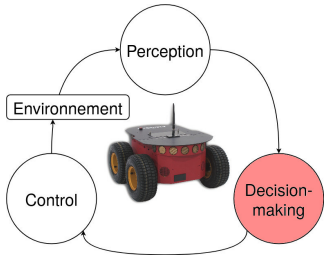
Lecture 9: Planning

Zhi Yan

ENSTA, Institut Polytechnique de Paris

Where Are We?

The "sense-think-act" paradigm: Planning is the core of "think" and a formalized procedure for decision-making.



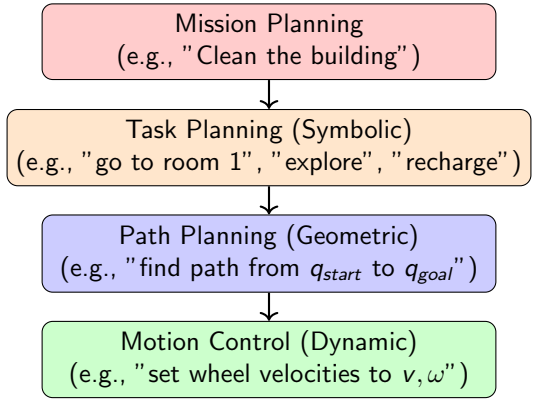
"Think" involves answering two key questions:

- **What to do?** $\Longrightarrow$ **Task Planning** (e.g., "go to kitchen", "pick up cup")
- **How to get there?** $\Longrightarrow$ **Path Planning** (e.g., "find the optimal route from my current pose to the kitchen")

ENSTA IP PARIS

The Planning Hierarchy

Planning is not one single problem, but a hierarchy of problems, from abstract to concrete.



Today's focus: Path Planning (Level 3) and Task Planning (Level 2).

ENSTA IP PARIS

What is a Path?

Definition:

A **path** is a continuous sequence of configurations (poses) that connects a start configuration q_{start} to a goal configuration q_{goal}, while avoiding collisions with obstacles.

- $q = (x, y, \theta)$ for a differential drive robot.
- $q = (x, y, \theta, j_1, j_2, ...j_N)$ for a mobile manipulator.



The core problem of path planning:

Find a function $\tau(s) : [0, 1] \rightarrow \mathcal{C}$ such that:

- $\tau(0) = q_{start}$
- $\tau(1) = q_{goal}$
- $\forall s \in [0, 1], \tau(s) \in \mathcal{C}_{free}$ (is collision-free)

ENSTA IP PARIS

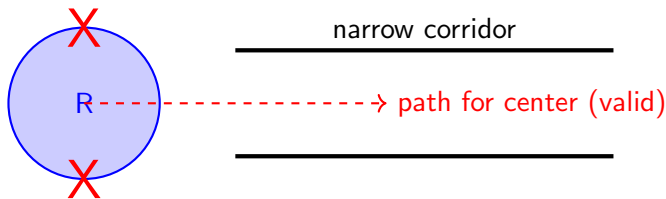
Configuration Space

Why robots are not just points.

The Problem: Robots Have Volume

Planning for a single point (the robot's center) is easy, **but the robot has a body.**

A path for the center might be valid, but the robot's body might collide.



We must plan for the *entire robot*, not just its center!

Solution: The Configuration Space

Definition:
The **Configuration Space (or C-Space)** $\mathcal{C}$ of a robot is the set of all possible configurations (poses) it can have.

- For a point robot in 2D, $\mathcal{C}$ is just (x, y).
- For a circular robot in 2D, $\mathcal{C}$ is (x, y).
- For a square robot in 2D, $\mathcal{C}$ is (x, y, θ). $\implies$ The C-Space is 3D!
- For a 6-DOF robot arm, $\mathcal{C}$ is 6-dimensional $(\theta_1, ..., \theta_6)$.

The "trick": We pre-process the map by "inflating" all obstacles by the robot's shape.
$\implies$ This transforms the **robot into a single point** and the **obstacles into** $\mathcal{C}_{obs}$.

ENSTA IP PARIS

Visualizing C-Space: The Minkowski Sum

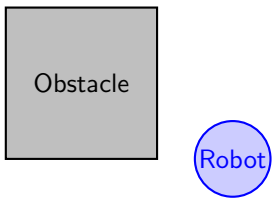
This "inflation" process is formally known as a **Minkowski Sum**:

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$$

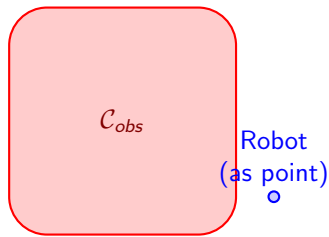
We "drag" the robot's shape (centered at its origin) along the boundary of every obstacle:

$$\mathcal{C}_{obs} = \mathcal{O} \oplus (-\mathcal{R}) = \{\mathbf{o} - \mathbf{r} \mid \mathbf{o} \in \mathcal{O}, \mathbf{r} \in \mathcal{R}\}$$

Workspace (Real World)



Configuration Space (C-Space)

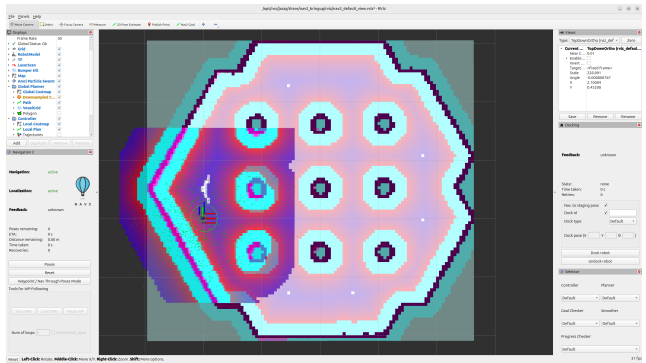


Now, any path planning algorithm just has to find a path for the **point** without touching the $\mathcal{C}_{obs}$.

ENSTA IP PARIS

C-Space in ROS: The Costmap

The **costmap** in Nav2 is a 2D representation of the C-Space. $\Longrightarrow$ You have already seen this!



Planners in ROS do not plan on the raw map but the costmap.

ENSTA IP PARIS

C-Space in ROS: The Costmap

The **costmap** is an occupancy grid with multiple "layers":

- **Static layer**: Occupancy grid map (black).
- **Obstacle layer**: Raw sensor data (red).
- **Inflation layer**: The $\mathcal{C}_{obs}$ (cyan/purple).

The "inflation_radius" parameter in Nav2 defines the radius to inflate costmap around lethal obstacles:

$$\text{inflation_radius} = \text{robot_radius} + \textit{SafetyBuffer}$$

ENSTA IP PARIS

Global Path Planning

Finding the best path from A to B on a known map.

Global vs. Local Planning

Navigation is a two-part planning problem[1]:

Global Planner

- Has the **entire map** (costmap).
- Finds an optimal path from q_{start} to q_{goal}.
- Assumes the world is static.
- **Slow**, might only runs once (or when the env. / goal changes).
- *Example: Dijkstra, A**

Local Planner

- Has a **small window** of sensor data around the robot.
- Tries to follow the Global Path.
- Reacts to **dynamic obstacles** (e.g., a person walking by).
- **Fast**, runs many times per second (e.g., 20 Hz).
- *Example: DWA, TEB*

Today's focus: Global Planner.

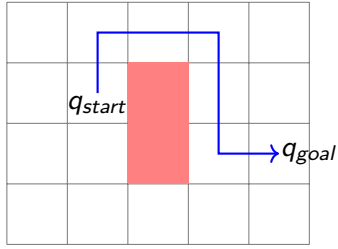
[1] https://docs.nav2.org/setup_guides/algorithm/select_algorithm.html

ENSTA IP PARIS

Discretizing the World: Grids and Graphs

Continuous C-Space is infinitely complex... We'd discretize it:

- Lay a grid over the C-Space and each cell is a "node" in a graph.
- Cells in $\mathcal{C}_{free}$ are valid nodes, while cells in $\mathcal{C}_{obs}$ are removed.
- An "edge" connects a node to its neighbors (4-way or 8-way).



The problem is then transformed from "find a path in continuous space" to **"find a path in a graph"**.

ENSTA IP PARIS

Algorithm 1: Dijkstra's Algorithm (NavFn)

- The classic, foundational graph-search algorithm.
- **How it works**:
  1. Start at q_{start} with cost $= 0$.
  2. Create an expanding "wavefront" of costs.
  3. Visit the lowest-cost unvisited node.
  4. Update the costs of its neighbors.
  5. Repeat until the "wave" hits q_{goal}.
- **Guarantees** finding the absolute shortest path.
- **Problem**: It explores equally in all directions, even those going away from the goal (`Dijkstras_progress_animation.gif`). $\Longrightarrow$ Costly!
- **In ROS 2**: This is the `NavFn` planner.

ENSTA IP PARIS

Algorithm 2: A* (A-Star)

- A "smart" version of Dijkstra.
- **Idea**: Use a **heuristic** to "pull" the search towards the goal.
- **Heuristic** $h(n)$: An estimate of the cost from node n to the goal.
- A common, simple heuristic is **Euclidean distance**:

$$h(n) = \sqrt{(n_x - goal_x)^2 + (n_y - goal_y)^2}$$

- This heuristic is admissible as it never overestimates the true cost, which guarantees A* still finds the shortest path.

ENSTA IP PARIS

A* (A-Star): The Formula

- A* selects the next node to visit based on a new cost function $f(n)$:

$$f(n) = g(n) + h(n)$$

  - $f(n)$: The total estimated cost of the best path through n.
  - $g(n)$: The **actual cost** from q_{start} to n. $\implies$ This is what Dijkstra calculates!
  - $h(n)$: The **heuristic (estimated) cost** from n to q_{goal}.

- A* explores the most promising nodes first, making it **faster** than Dijkstra (`Astar_progress_animation.gif`).
- **In ROS 2**: The `SmacPlanner` is a modern, highly optimized A* planner.

ENSTA IP PARIS

Algorithm 3: Rapidly-exploring Random Trees (RRT)

- **Problem**: Grids (like in A*) suffer from the "curse of dimensionality".
- A 6-DOF arm on a 100x100x100 grid is $100^6 = 10^{12}$ nodes!
- **Solution**: sampling-based methods – Don't build a full graph, sample it.
- **RRT algorithm**:
  1. Start a "tree" at q_{start}.
  2. Pick a random point q_{rand} in the C-Space.
  3. Find the nearest node q_{near} in your tree to q_{rand}.
  4. Grow the tree a small step from q_{near} towards q_{rand} to create q_{new}.
  5. If the path to q_{new} is collision-free, add q_{new} to the tree.
  6. Repeat.

ENSTA IP PARIS

RRT Visualized

RRT can find a path very quickly, but cannot guarantee finding the optimal one.

`RRT_progress_animation.gif` `RRT_connect_progress_animation.gif`

Variant: RRT-Connect grows two trees (one from start, one from goal) and tries to connect them.

ENSTA IP PARIS

Summary: Global Planners

Algorithm	**Method**	**Pros**	**Cons**
Dijkstra (`NavFn`)	grid search (wavefront)	guaranteed optimal	slow, costly
A* (`SmacPlanner`)	grid search (heuristic)	guaranteed optimal, fast, cost-effective	slower in high-dim
RRT (`OMPL`)	sampling (random tree)	very fast in high-dim C-space	not optimal (probabilistically complete)

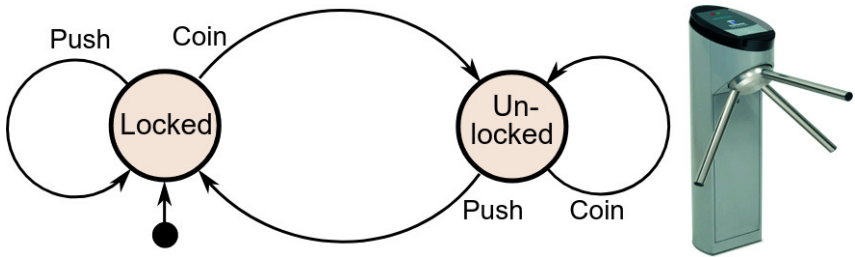
ENSTA IP PARIS

Task Planning

"What to do?"

The Old Way: Finite-State Machine (FSM)

For years, robots were controlled by **Finite-State Machine (FSM)**.



The problem with FSM:

- **Fragile**: If an error occurs (e.g. the state is neither "Locked" nor "Unlocked"), the robot will get stuck.
- **Not scalable**: For a complex task (e.g., play football), the web of states and transitions becomes an unmanageable "spaghetti code".

ENSTA IP PARIS

The New Way: Behavior Trees (BTs)

- Adopted from the video game industry (e.g., AI for enemies).
- Used by default in Nav2 for ROS 2.
- **Key idea**: A BT is a **tree** of tasks, not a graph of states.
- It runs at a high frequency (e.g., 10 Hz). The entire tree is "ticked" from the root.
- A node can return one of three statuses:
  - **Success**
  - **Failure**
  - **Running** (e.g., "I'm still following the path, I'm not done yet.")

ENSTA IP PARIS

BT Node Types

A BT is composed of **nodes** that are combined to create complex logic.

Leaf nodes (the "work")

- **Action**: "Do something" (e.g., `ComputePath`, `FollowPath`, `SpinRobot`)
- **Condition**: "Check something" (e.g., `IsBatteryLow?`, `IsPathValid?`)

Control nodes (the "logic")

- **Sequence** $\rightarrow$: Ticks children one by one. Fails if any child fails.
- **Fallback** ? **(or "Selector")**: Ticks children until one succeeds.
- **Parallel** $\rightrightarrows$: Ticks all children simultaneously.

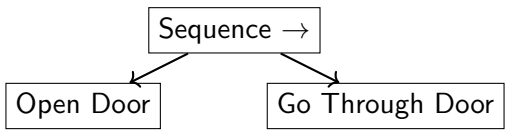
A powerful BT creation and debugging tool: Groot (`https://www.behaviortree.dev/groot`).

ENSTA IP PARIS

Control Node: Sequence $\rightarrow$

Logic: "Do all of these tasks in order."

- Ticks child 1.
- If child 1 returns **Success**, tick child 2.
- If child 2 returns **Success**, tick child 3.
- ...
- If child N returns **Success**, the Sequence returns **Success**.
- If *any* child returns **Failure**, the Sequence stops and returns **Failure**.
- If *any* child returns **Running**, the Sequence stops and returns **Running**.

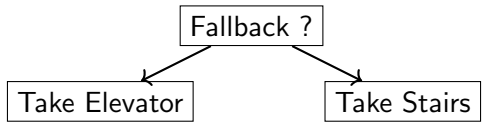


ENSTA IP PARIS

Control Node: Fallback ?

Logic: "Try these tasks in order until one works."

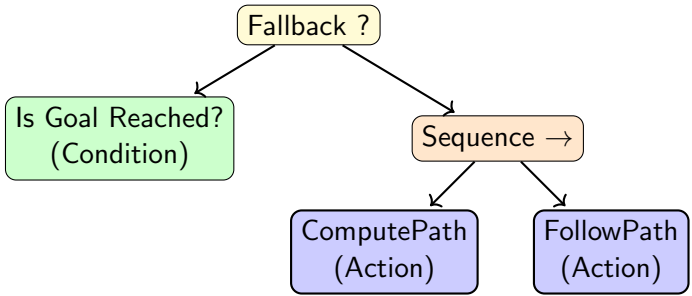
- Ticks child 1.
- If child 1 returns **Failure**, tick child 2.
- If child 2 returns **Failure**, tick child 3.
- ...
- If *any* child returns **Success**, the Fallback stops and returns **Success**.
- If *all* children return **Failure**, the Fallback returns **Failure**.
- If *any* child returns **Running**, the Fallback stops and returns **Running**.



ENSTA IP PARIS

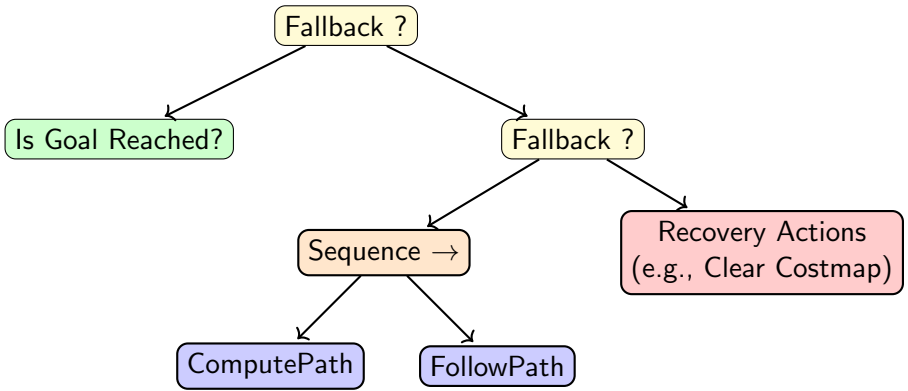
Example: A Simple Nav2 Behavior Tree

This is the logic for “go to a goal”.



Example: Adding Recovery

What if `FollowPath` fails (e.g., an obstacle appears)?



ENSTA IP PARIS

Summary

- Planning is the "think" step, divided into **Task and Path Planning**.
- **Configuration Space (C-Space)** is the key concept that simplifies planning by "inflating" obstacles and "shrinking" the robot to a point.
- **Global Path Planners** find optimal paths on a grid.
  - **Dijkstra**: Simple, but costly.
  - **A***: "Smarter" and faster by using a heuristic ($f = g + h$).
- **Task Planners** (Behavior Trees) provide a robust and modular way to define complex robot behaviors.

ENSTA IP PARIS

Questions?

Next: Practical Work 9 - Nav2 Behavior Tree

ENSTA IP PARIS