Introduction to Mobile Robotics

Lecture 10: Navigation

Zhi Yan

ENSTA, Institut Polytechnique de Paris

Recap: The Planning Hierarchy

In Lecture 9, we discussed **Global Path Planning**: finding a path from A to B on a static map.

- **Global Planner:** "I need to go through the hallway, then turn left."
- **Output:** A static list of waypoints (x, y) from Start to Goal.
- *Analogy:* Google Maps finding a route from Paris to Lyon.

The problem:

- This path is just a geometric line but not a control command.
- It doesn't tell the robot **how** to move along it, nor does it account for a person walking in front of the robot.

Today's topic: obstacle avoidance, local planners, and dynamic environments.

ENSTA IP PARIS

Global vs. Local Planners

Global Planner (The Strategist)

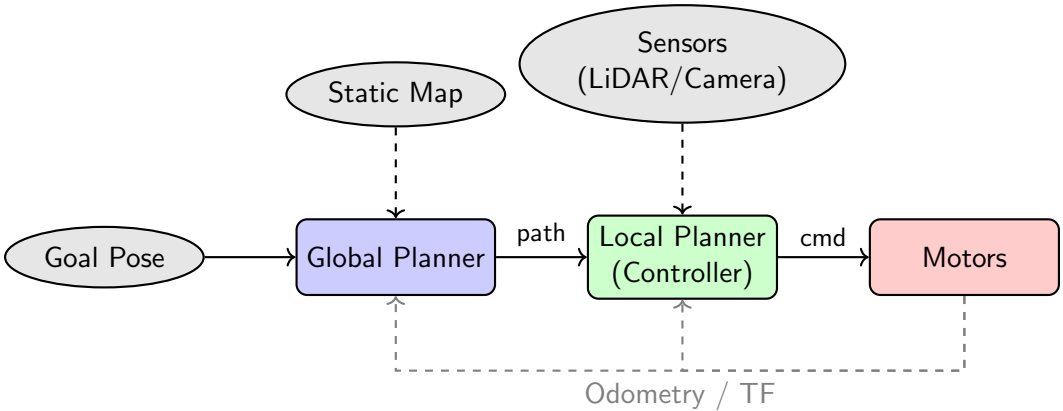
- **Scope:** The entire map.
- **Frequency:** Low (e.g., 1 Hz).
- **Input:** Static Map + Goal.
- **Algorithm:** A*, Dijkstra, RRT.
- **Goal:** Find the optimal route.

Local Planner (The Pilot)

- **Scope:** Immediate surroundings (e.g., 3m radius).
- **Frequency:** High (e.g., 20 Hz).
- **Input:** Global Path + LiDAR/Camera.
- **Algorithm:** DWA, TEB, MPC.
- **Goal:** Follow the path safely and smoothly.

ENSTA
IP PARIS

The Navigation Pipeline



The Local Planner (or **Controller** in Nav2):

- Output: Velocity commands (v, ω) sent to `/cmd_vel`.
- *Analogy:* The human driver turning the steering wheel and pressing the pedals.

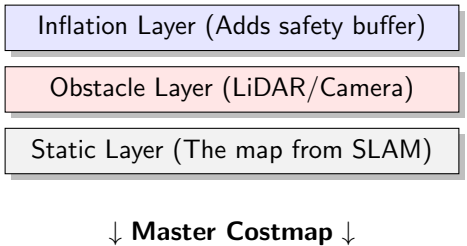
ENSTA IP PARIS

Costmaps in Depth

The real “map” for robot navigation.

ENSTA IP PARIS

Recall: The Costmap



The "Master Costmap" is the final output after all layers are combined.

Master Costmap

In the costmap, the value of each cell ranges from 0 to 254:

- **0 (FREE_SPACE):** Free space, the robot can pass.
- **253 (INSCRIBED_INFLATED_OBSTACLE):** The obstacle inflation zone calculated based on the robot's footprint, the robot should not enter.
- **254 (LETHAL_OBSTACLE):** Obstacle.

255 (NO_INFORMATION): Unknown area.

ENSTA IP PARIS

Master Costmap

Layer combination methods:

1. Initialize the Master Costmap using 255.
2. Overwrite the Master Costmap with the **Static Layer**.
3. Compare the values at corresponding cells in the **Obstacle Layer** and the Master Costmap, keeping the larger one:

$$C_{master} = \max(C_{master}, C_{layer})$$

4. The **Inflation Layer** checks all cells with 254 in the Master Costmap, calculates the decay curves around them, and adds the cost values.

ENSTA
IP PARIS

Two Costmaps for Two Planners

Just as we have two planners, we maintain two costmaps:

Global Costmap

- Covers the **entire environment**.
- Mostly static (walls, furniture).
- Used by the Global Planner to calculate long-term paths.
- Updates slowly.

Local Costmap

- A small **rolling window** (e.g., $4m \times 4m$) centered on the robot.
- Moves with the robot.
- Highly dynamic: shows people, moving obstacles, and recent sensor data.
- Used by the Local Planner to avoid collisions.

ENSTA IP PARIS

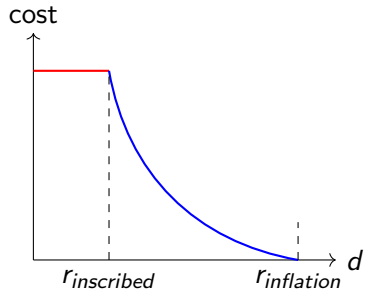
The Inflation Layer

We don't just mark obstacles, we create a gradient of danger.

The mathematical model:

Let d be the distance from the nearest lethal obstacle.

- If $d \leq r_{inscribed}$: $Cost = 253$
- If $r_{inscribed} < d \leq r_{inflation}$: $Cost = \lfloor \exp(-k \cdot (d - r_{inscribed})) \cdot 252 \rfloor$
- If $d > r_{inflation}$: $Cost = 0$



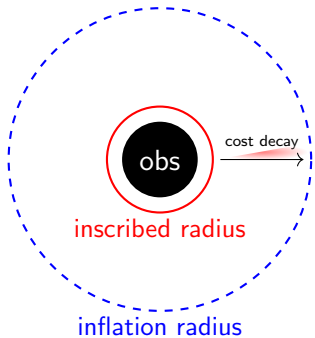
The **Scaling Factor** k determines how aggressive the robot is: A high k means costs drop quickly (robot gets closer to walls).

ENSTA IP PARIS

The Inflation Layer

In Nav2 (e.g., `nav2_params.yaml`):

- k: `cost_scaling_factor`
- $r_{inflation}$: `inflation_radius`
- $r_{inscribed}$: `footprint` (high priority) or `robot_radius` (low priority)



The Robot Footprint

Nav2 supports two types of footprints:

Circular Footprint

- Defined by a single radius r.
- Computationally **very fast** (distance check).
- The robot can rotate in place anywhere it fits.
- *Good for:* TurtleBot, Roomba.

Polygonal Footprint

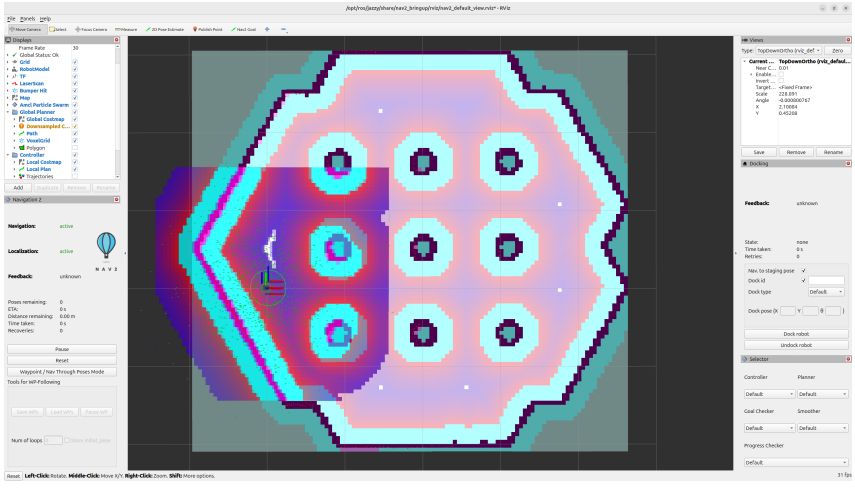
- Defined by a list of points $[(x_1, y_1), \dots]$.
- Computationally **expensive** (requires polygon intersection checks).
- Essential for asymmetric robots (long, rectangular).
- *Good for:* Cars, logistics robots.

Warning: Using a circular footprint for a rectangular robot will either cause collisions (if radius too small) or refuse to pass narrow doors (if radius too big).

ENSTA IP PARIS

The Inflation Layer

Let's take another look at this screenshot:

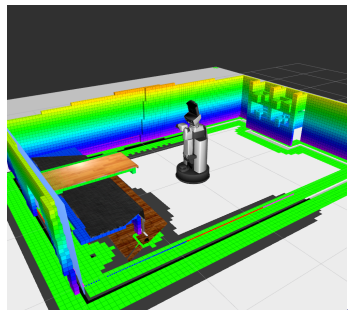


Optional: The Voxel Layer (3D)

So far we have been considering 2D layers, but tables have legs, and overhangs exist.

The Voxel Layer (with RGB-D camera)

- Maintains a 3D grid of cubes (voxels).
- Ray-tracing clears voxels between the sensor and the hit point.
- Columns of voxels are projected down to 2D for the costmap.
- **Marking:** Adding an obstacle.
- **Clearing:** Removing an obstacle (ray-tracing).



ENSTA
IP PARIS

The Local Planner

How a robot follows a path and avoids obstacles.

The Job of the Local Planner

The Local Planner / Controller has a difficult optimization problem to solve in real time:

Find a velocity command (v, ω) that:

1. minimizes distance to the **global path**,
2. minimizes distance to the **goal**,
3. maximizes distance from **obstacles** (safety),
4. respects **kinematic constraints** (max speed, max acceleration),
5. is **smooth** (no jerky movements).

ENSTA
IP PARIS

Algorithm 1: Dynamic Window Approach (DWA)

The classic algorithm for differential drive robots (e.g., TurtleBot).

Core Idea: Don't plan in position space (x, y) but in **velocity space** (v, ω).

Step 1: The Search Space (Dynamic Window). We only consider velocities (v, ω) that are:

1. Reachable: Within the next Δt given acceleration limits:

$$v \in [v_{curr} - a_{max}\Delta t, v_{curr} + a_{max}\Delta t]$$

2. Safe: The robot must be able to stop before hitting an obstacle:

$$v \leq \sqrt{2 \cdot dist(obs) \cdot a_{max}}$$

ENSTA IP PARIS

DWA: Trajectory Generation & Scoring

Step 2: Predict. For each sample (v, ω), simulate the robot's trajectory for a short time (`sim_time` in Nav2, e.g., 2 seconds).

Step 3: Score. Evaluate each trajectory based on:

$$\mathit{Cost} = \alpha \cdot \text{PathDist} + \beta \cdot \text{GoalDist} + \gamma \cdot \text{OccDist}$$

- PathDist: Distance between the trajectory and the global path (how well it follows the plan).
- GoalDist: Euclidean distance to the local goal from the end of the trajectory (progress).
- OccDist: The maximum obstacle cost (0-254) encountered along the trajectory (safety).

Step 4: Select. Pick the lowest cost trajectory and execute the velocity.

ENSTA IP PARIS

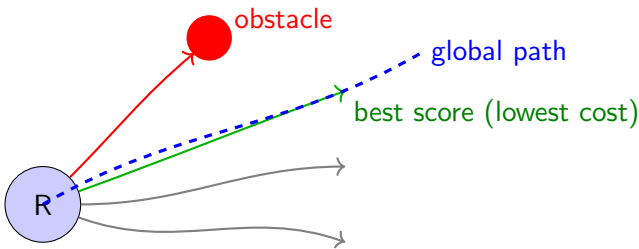
DWA: The Weights in The Cost Function

$$Cost = \alpha \cdot \text{PathDist} + \beta \cdot \text{GoalDist} + \gamma \cdot \text{OccDist}$$

- High $\alpha \implies$ Robot sticks to the global path strictly, like a train on tracks.
- High $\beta \implies$ Robot prioritizes moving toward the goal, potentially "cutting corners".
- High $\gamma \implies$ Robot maintains a large safety margin from obstacles (very cautious).

ENSTA IP PARIS

DWA Visualized



DWA simulates many futures and picks the best one. $\Longrightarrow$ A **sampling-based** controller!

Algorithm 2: Timed Elastic Band (TEB)

An optimization-based approach (very popular in ROS).

- Instead of sampling velocities, TEB deforms the initial Global Path.
- It treats the path as a rubber band (elastic).
- **Forces:**
  - **Internal forces:** Keep the path short and smooth (contract the band).
  - **External forces:** Push the band away from obstacles.

Pros:

- Handles complex maneuvers (e.g., parking, reversing).
- Can utilize the full kinematics of car-like robots.

Cons:

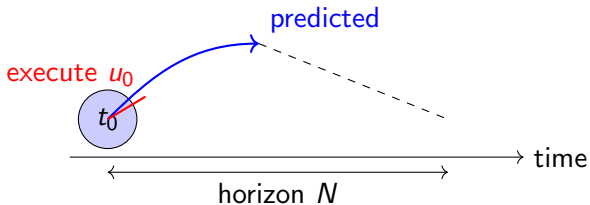
- Computationally heavier than DWA.
- Can get stuck in local minima (oscillations).

ENSTA IP PARIS

Algorithm 3: Model Predictive Control (MPC)

The state-of-the-art in control theory (used in SpaceX rockets, self-driving cars).

1. **Look ahead:** Predict the system behavior for the next N steps (horizon).
2. **Optimize:** Find the optimal sequence of control inputs $\{u_0, u_1, \ldots, u_{N-1}\}$ that minimizes cost.
3. **Execute:** Send **only the first** command u_0 to the motors.
4. **Repeat:** Shift the horizon forward and re-calculate.



ENSTA IP PARIS

MPC: The Cost Function

The mathematical model:

We minimize the cost function J:

$$J = \sum_{k=0}^{N} \underbrace{(x_k - x_{ref})^T Q (x_k - x_{ref})}_{\text{state error}} + \underbrace{u_k^T R u_k}_{\text{control effort}}$$

w.r.t. constraints:

- **Model Dynamics:** $x_{k+1} = f(x_k, u_k)$ (where will the robot be?)
- **Input Limits:** $v \leq v_{max}, \omega \leq \omega_{max}$ (motor limits)
- **State Limits:** Distance to Obstacle $> r_{robot}$ (safety)

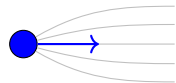
The magic is in the model $f(x, u)$: If it is accurate, the control is perfect.

ENSTA IP PARIS

MPC in Nav2

Classic MPC solves gradients. Nav2 uses a **sampling-based** MPC called "Model Predictive Path Integral (MPPI)":

- Generate thousands of random noisy trajectories on the GPU/CPU.
- Evaluate cost for each trajectory (collision, path following).
- Calculate the weighted average of the trajectories.
- Can handle non-linear costmaps easily (no gradients needed!).



thousands of samples

ENSTA IP PARIS

Comparison: Which Controller to Choose?

Algorithm	**DWA**	**TEB**	**MPC**
Method	velocity sampling	trajectory optimization	receding horizon
Robot	differential	car-like	all
Compute	low	medium	high
Pros	simple, stable, real-time[1]	smooth trajectory, conforms to dynamics	overcomes local minima, adaptable to complex models
Cons	prone to local optima, poor foresight	numerous parameters, high computational cost	relies on HPC (e.g., GPU), sampling jitte
ROS 1	**standard**	popular	rare / research
ROS 2	available (DWB)	available	**standard (MPPI)**

[1] Suitable for deployment on low-power hardware.

ENSTA IP PARIS

The Art of Tuning

Tuning a local planner is an art. Some take-home messages:

- **Robot oscillates (wiggles):**
  - Cause: `GoalDist` weight is too high compared to `PathDist`.
  - Fix: Increase `PathDist` and reduce `GoalDist`.
- **Robot hits walls when turning:**
  - Cause: `sim_time` is too short (robot is "shortsighted").
  - Fix: Increase `sim_time` (e.g., from 1.7s to 3.0s).
- **Robot moves too slowly:**
  - Cause: `cost_scaling_factor` in costmap is too low (walls feel "scary" from far away).
  - Fix: Increase `cost_scaling_factor` so cost drops faster.

ENSTA IP PARIS

Dynamic Environment

Robots are developed for humans.

The Dynamic Problem

The world is not static: things move (people, pets, cars).

The problem with costmaps:

If a person is walking at 1.5 m/s, marking their *current* position as an obstacle is not enough. $\Longrightarrow$ By the time the robot "sees" them, the person might have moved.

TODO 1: Trajectory prediction (physics/filter based)

- Use a Kalman Filter (with e.g., constant velocity model) to track moving obstacles.
- **Spatiotemporal Costmap:** The costmap layers project obstacle positions into the future time t.
- Nav2 can use sensor filters to track dynamic objects (but full prediction requires advanced plugins).

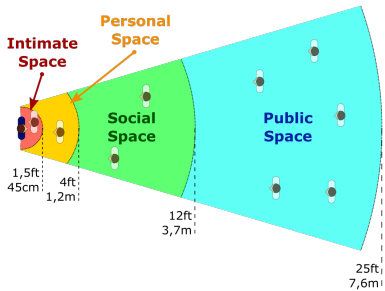
ENSTA IP PARIS

Social Navigation

TODO 2: Social norms

Robots operating around humans must follow unwritten social rules:

- **Proxemics (Hall's distances):**
  - Intimate: < 0.45 m (never enter)
  - Personal: 0.45 m - 1.2 m (ask permission)
  - Social: 1.2 m - 3.7 m (safe interaction)
- **Passing rules:** Pass on the right (or left, depending on country), not cut in front of moving person, etc.
- **Group integrity:** Do not cut through a group of two people talking.



ENSTA IP PARIS

The Oscillation Problem

Scenario:

1. Robot wants to go north, but the path is blocked by a person.
2. Local planner sees space to the left, so robot steers left.
3. Global planner recalculates, says "right is shorter", so robot steers right.
4. Robot **wiggles back and forth (oscillates)** without moving forward.

Solution:

If the robot hasn't moved X meters in Y seconds, trigger a **recovery**.

ENSTA IP PARIS

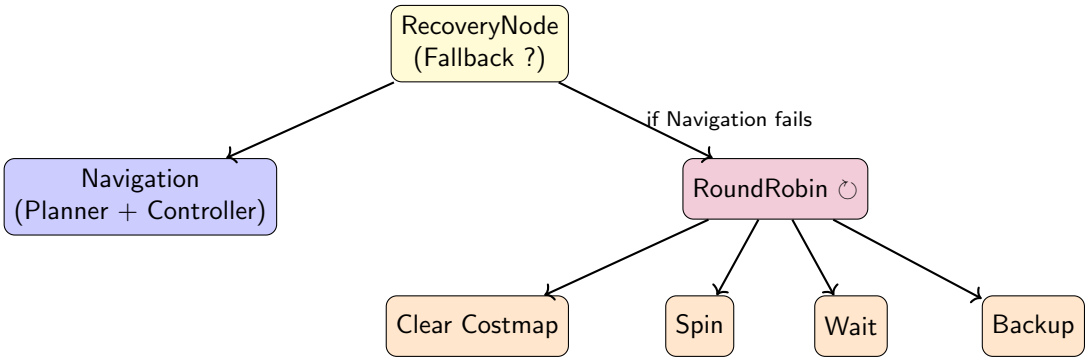
Recovery Behaviors

Nav2 uses a specific Behavior Tree branch for recoveries. If the Controller fails, it attempts these actions in sequence:

1. **Clear Costmap:** Clear the obstacle layer in the costmap (maybe the obstacle is gone?).
2. **Spin:** Rotate in place 360° to update the map with new sensor data.
3. **Wait:** Stop and wait for a few seconds for the dynamic environment to change (e.g., person walks away).
4. **Backup:** Drive backwards slowly (blindly for a few centimeters!) to unstuck wheels.

ENSTA IP PARIS

Recovery Tree Visualized



- **RoundRobin** picks the *next* recovery action in the list.
- *Example:* Fail → Clear → Retry → Fail → Spin → Retry → Fail → Backup...

Summary: The Full Stack

1. **Sensors** (LiDAR/camera) update the **Local Costmap**.
2. **Global Planner** (Dijkstra/A*) plans a path on the **Global Costmap**.
3. **Local Planner** (DWA/TEB/MPC) optimizes velocities (v, ω) to follow the path safely.
4. If Controller fails $\rightarrow$ **Recoveries**.

This architecture is the industry standard for mobile robotics today.

ENSTA IP PARIS

Questions?

Next: Practical Work 10 - Nav2 Controller

ENSTA IP PARIS