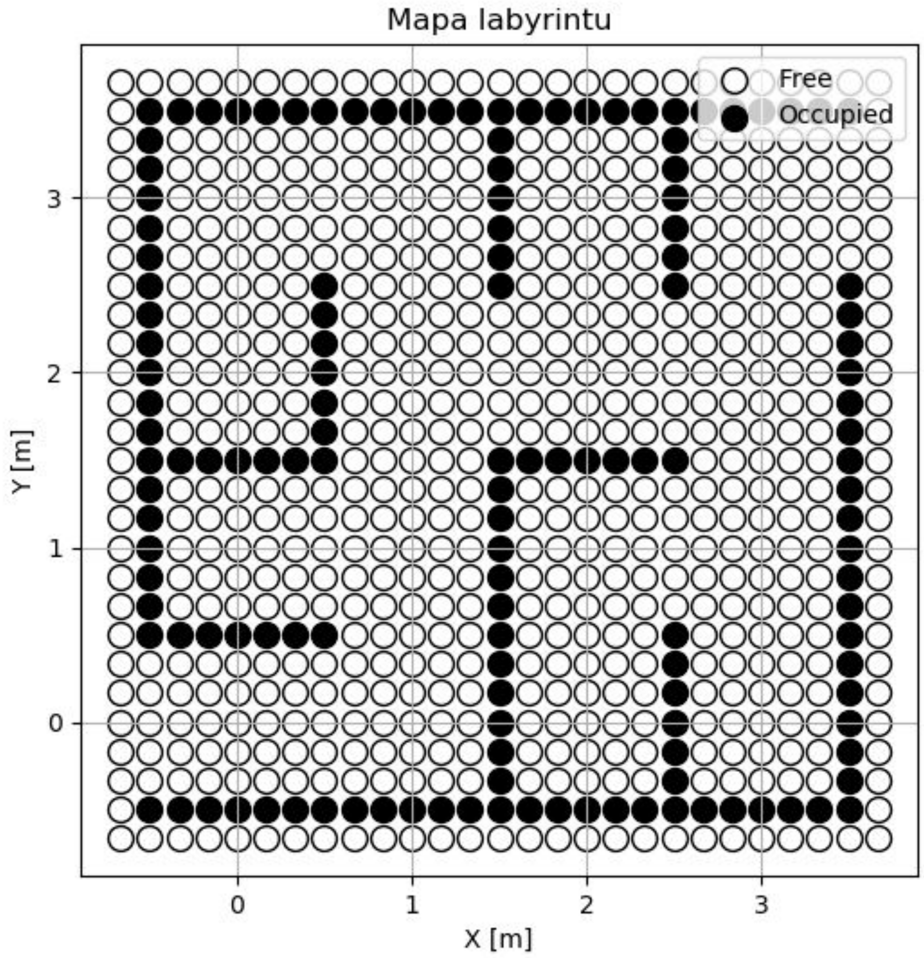
Robot Trajectory

Magnus Thomsen, Jan Kai Marek

Global Planner

Map Visualization



Implementing Dijkstra

Dijkstra's Algorithm

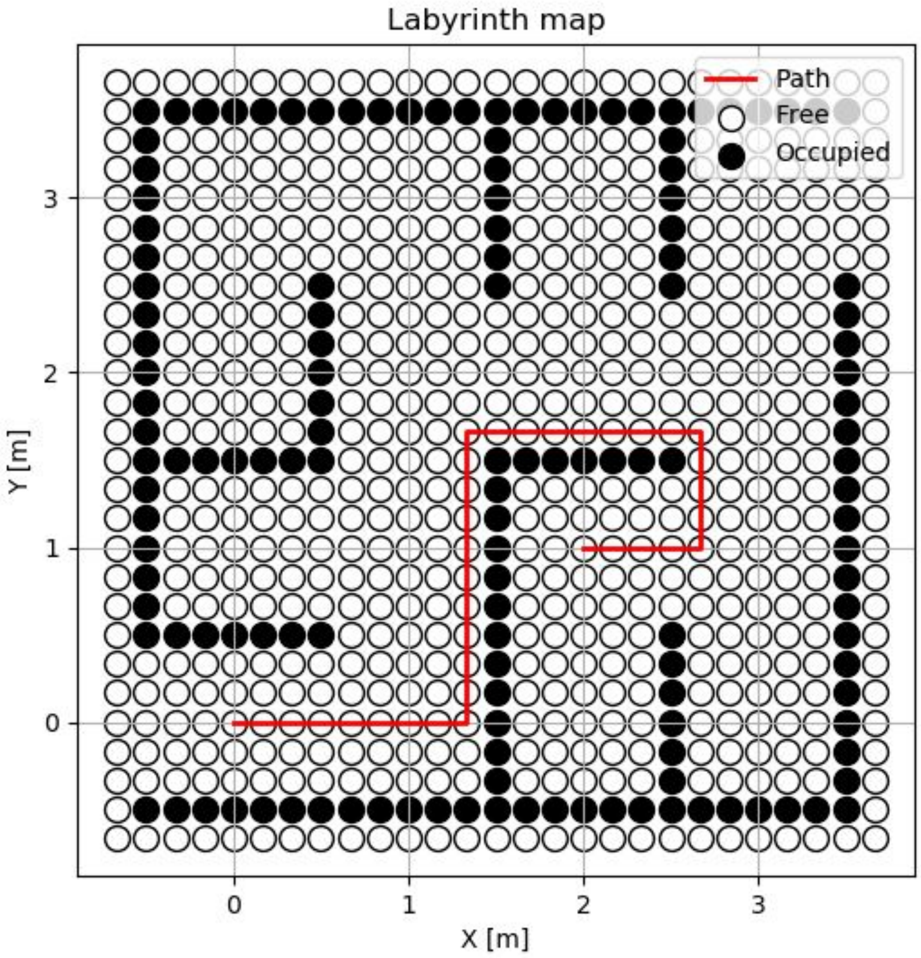
What it's good at

- Finds the **shortest path by total cost**
- Works with **weighted graphs** (weights ≥ 0)

How it thinks

> "From what I know so far, which node can I reach **cheapest** next?"

It uses a **priority queue** to always expand the currently cheapest path.



Implementing Costmap

What is a costmap?

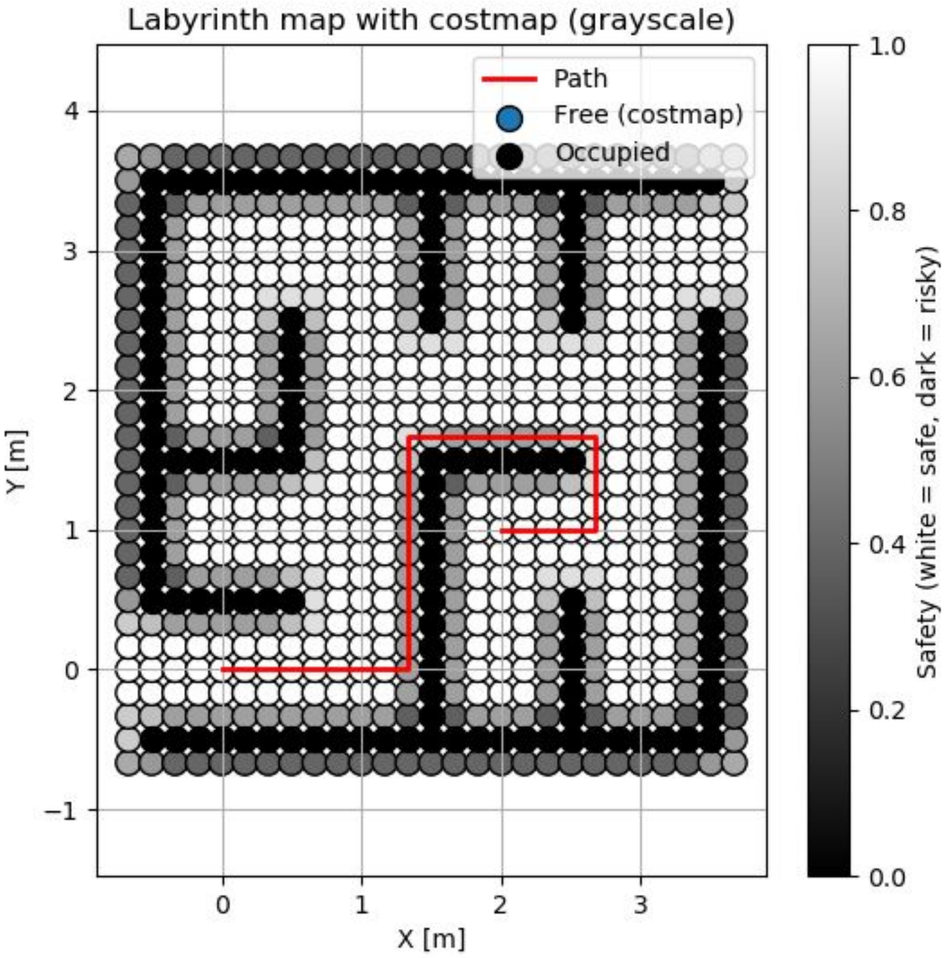
A **costmap** is a data structure (usually a grid or graph) where **each position has a cost** representing how *good or bad* it is to move through that position.

Instead of:

> “Can I go there?”

You ask:

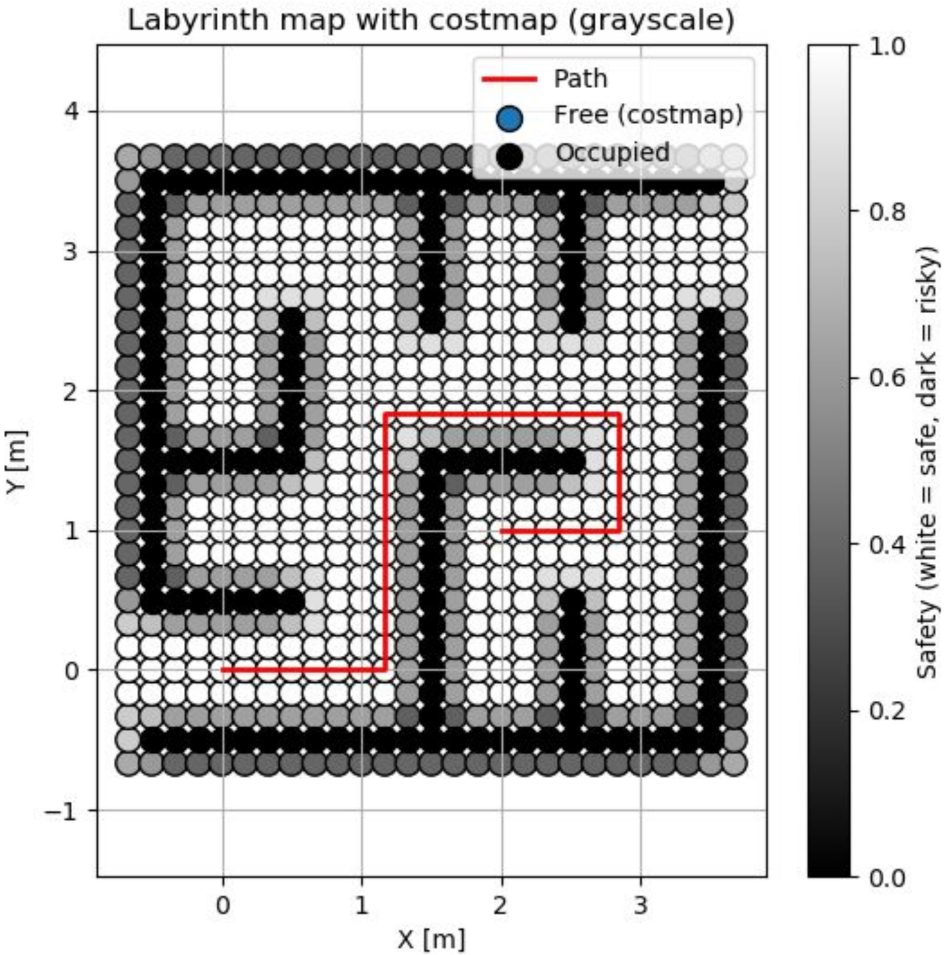
> “How **expensive** is it to go there?”



Dijkstra Costmap Extension

Dijkstra + costmap:

- Optimizes **what you actually care about**
- Produces:
  - smooth paths
  - safe clearance from obstacles
  - energy-efficient routes



Dijkstra Costmap Extension

Why not A*?

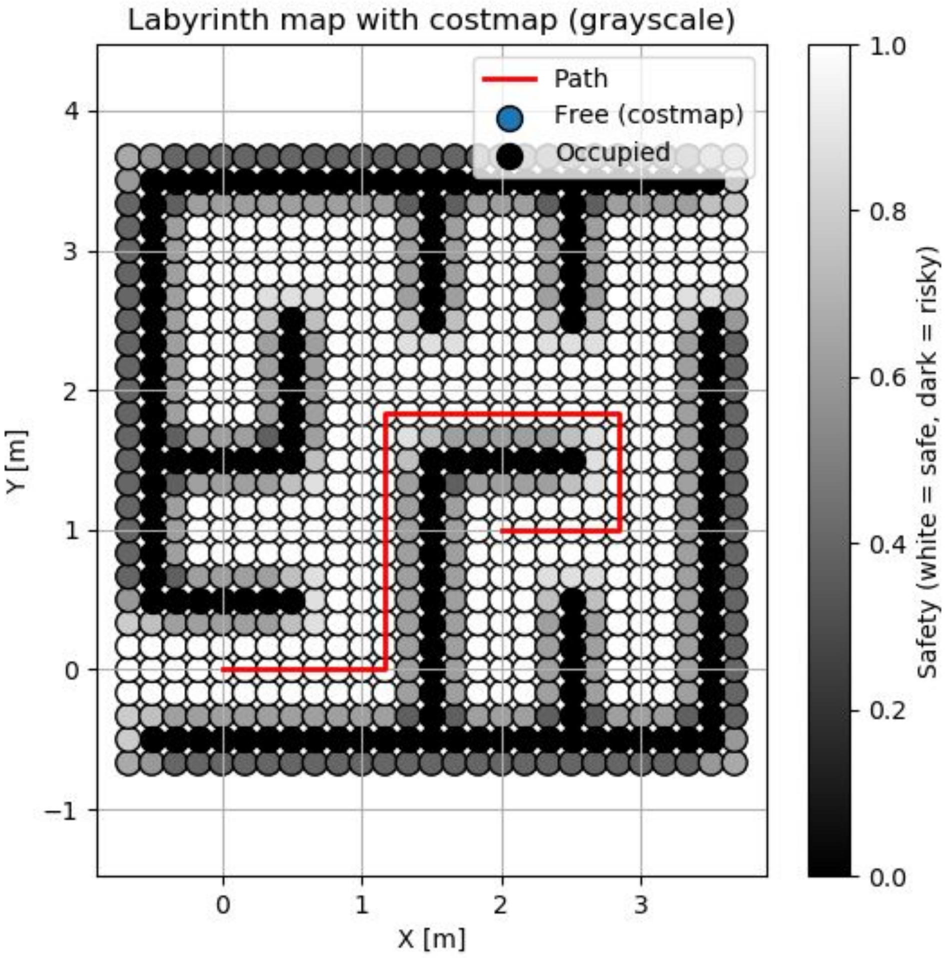
Imagine searching for a café in a city

Dijkstra

- Checks streets in *all directions*
- Guarantees cheapest route
- But wastes time exploring irrelevant areas

A*

- Strongly prefers streets **toward the café**
- Ignores streets going the wrong way
- Still guarantees the cheapest route



Implementing Line-of-sight Smoothing

What line-of-sight smoothing does

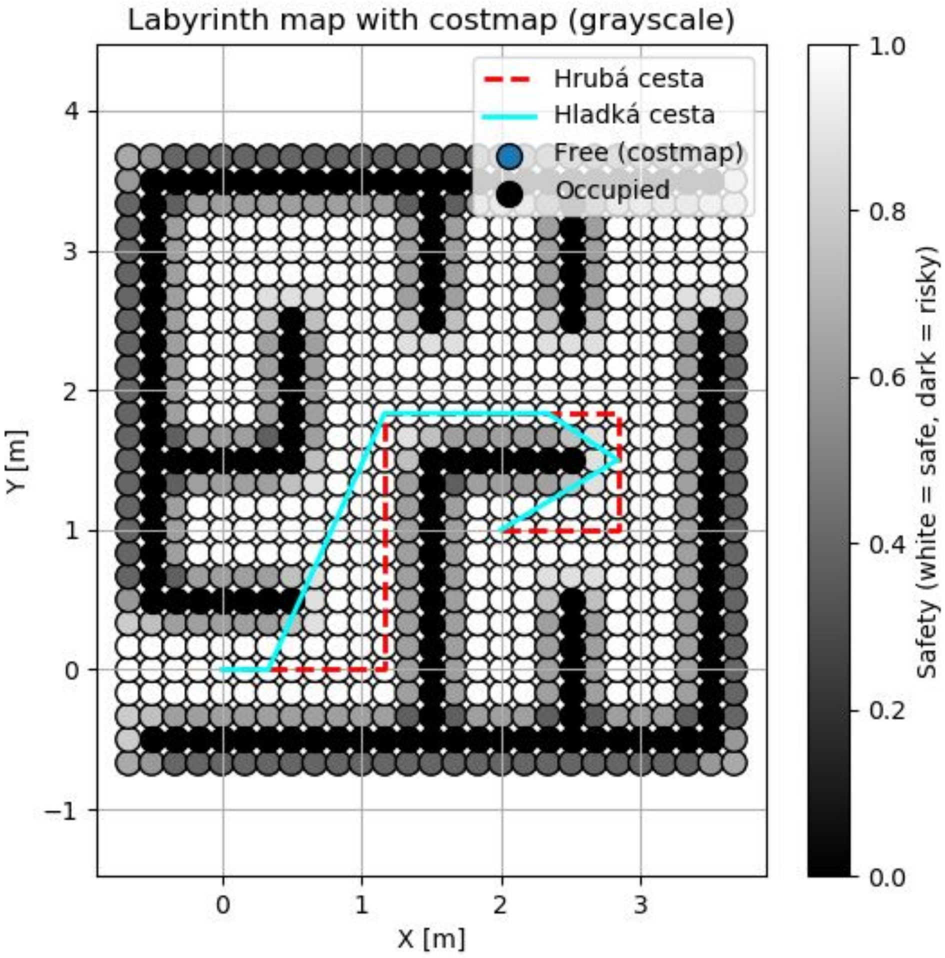
LoS smoothing asks a simple question:

> "Can I go **straight** from point A to point C without hitting an obstacle?"

If yes:

- delete point B
- keep the straight segment

Repeat until no more shortcuts are possible.



Implementing Line-of-sight Smoothing

What line-of-sight smoothing does

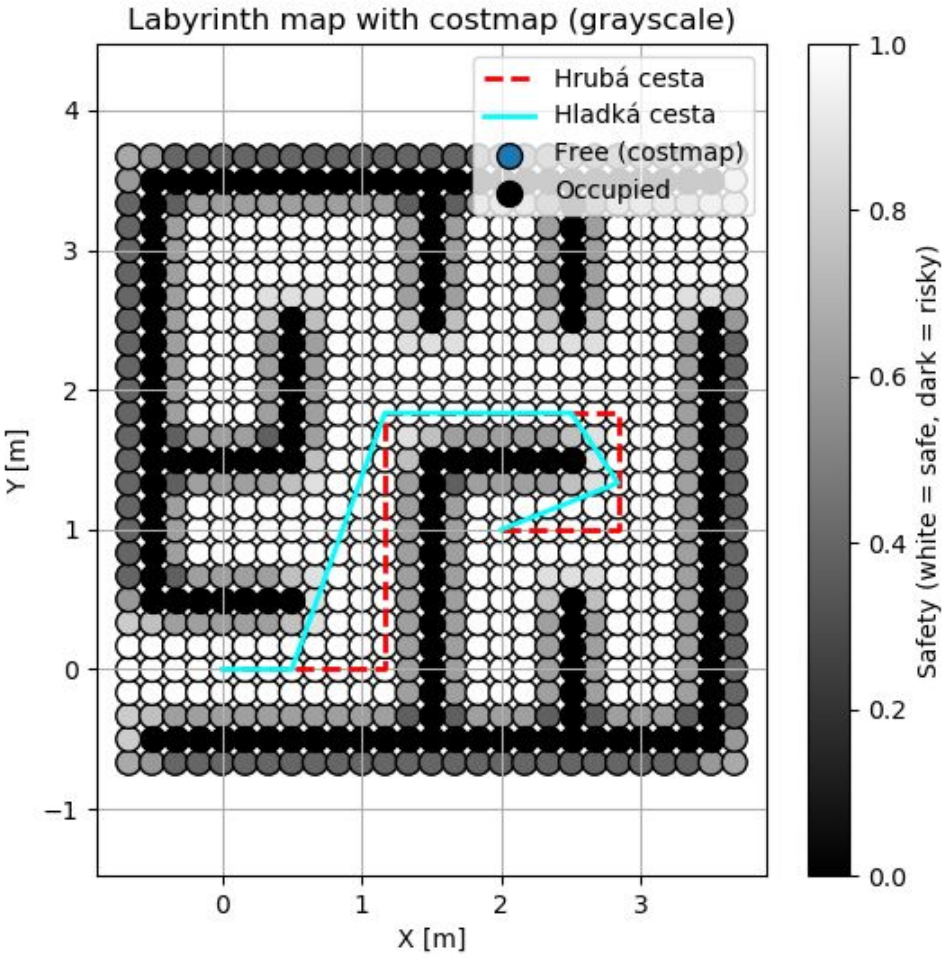
LoS smoothing asks a simple question:

> “Can I go **straight** from point A to point C without hitting an obstacle?”

If yes:

- delete point B
- keep the straight segment

Repeat until no more shortcuts are possible.



Implementing Line-of-sight Smoothing

What line-of-sight smoothing does

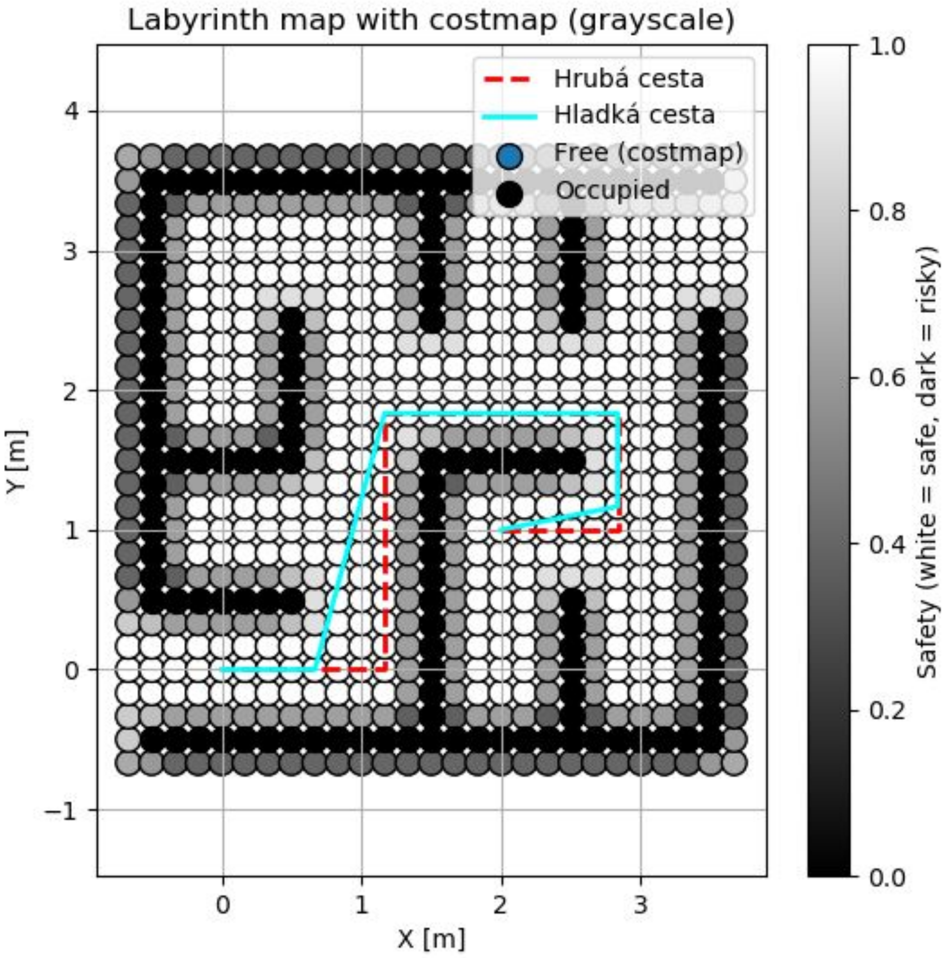
LoS smoothing asks a simple question:

> “Can I go **straight** from point A to point C without hitting an obstacle?”

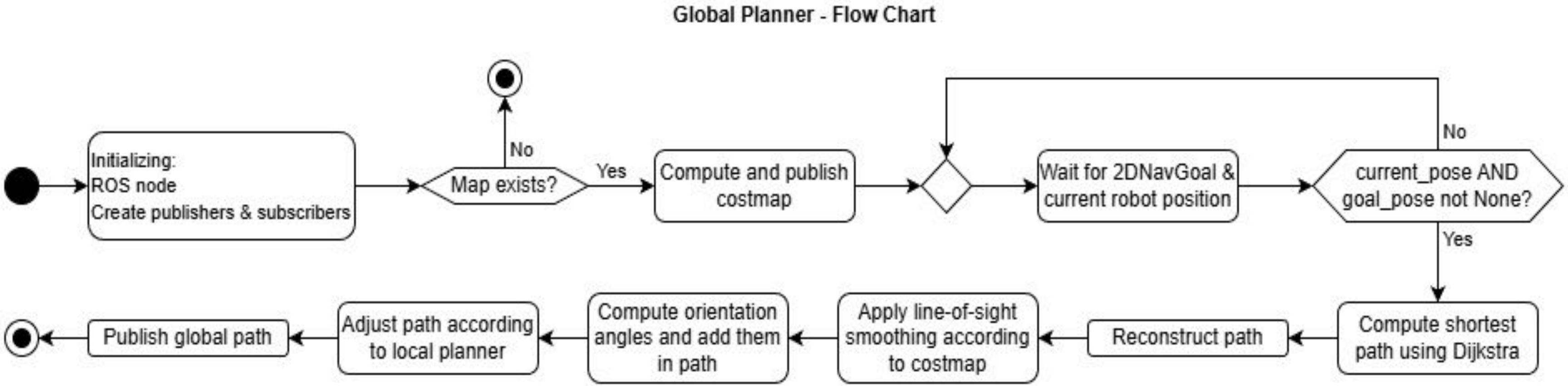
If yes:

- delete point B
- keep the straight segment

Repeat until no more shortcuts are possible.



Global Planner


Global Planner - Flow Chart
Initializing:
ROS node
Create publishers & subscribers
Map exists?
No
Yes
Compute and publish costmap
Wait for 2DNavGoal & current robot position
current_pose AND goal_pose not None?
No
Yes
Compute shortest path using Dijkstra
Reconstruct path
Apply line-of-sight smoothing according to costmap
Compute orientation angles and add them in path
Adjust path according to local planner
Publish global path

Local Planner

Local Planner

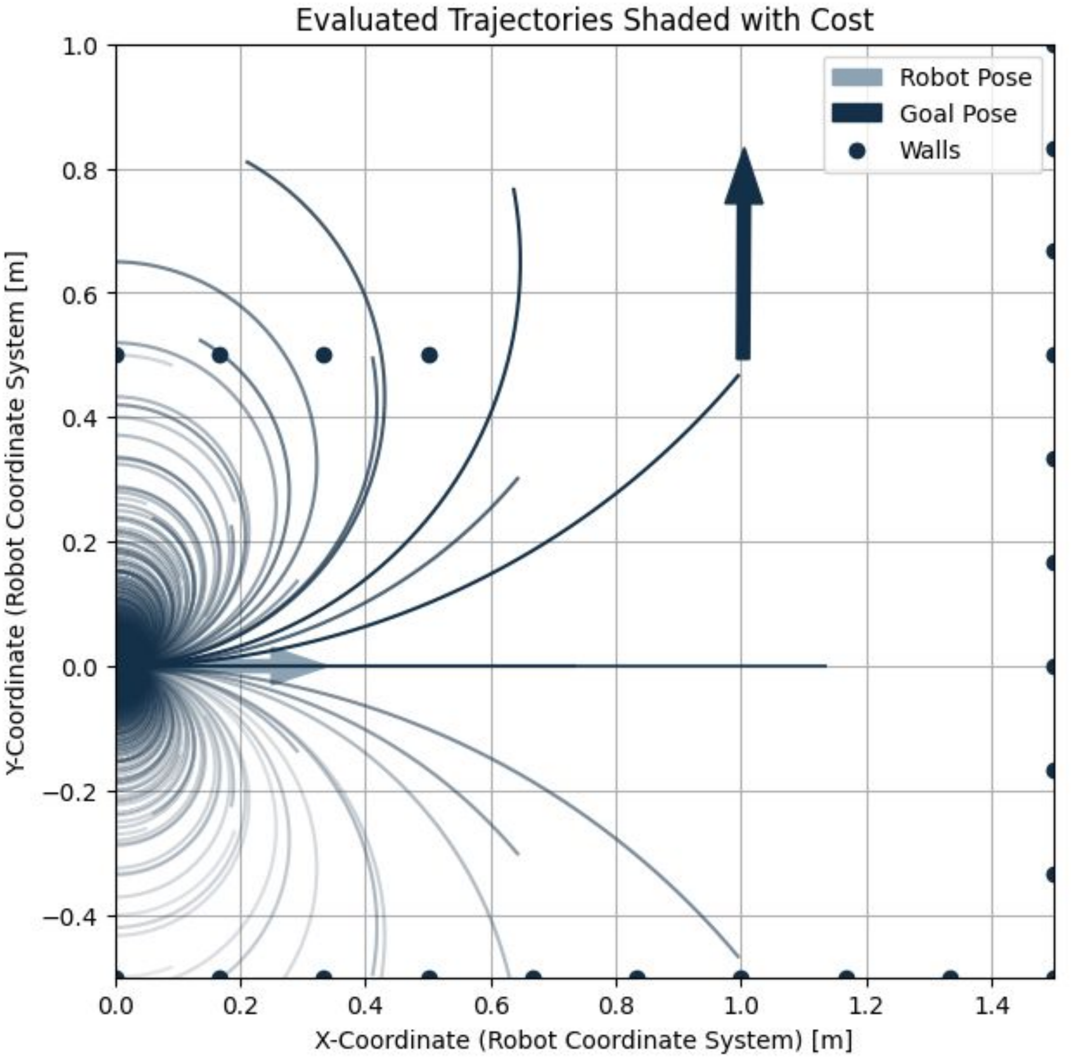
- The **global planner** gives a rough route made of points.
- That route **cannot be followed directly** by a real robot.
- A robot has **limits**: it can’t turn or accelerate instantly.
- The robot must **move smoothly and react in real time**.
- So instead of jumping between points, the robot must decide:

“What speed and direction should I move *right now*?”

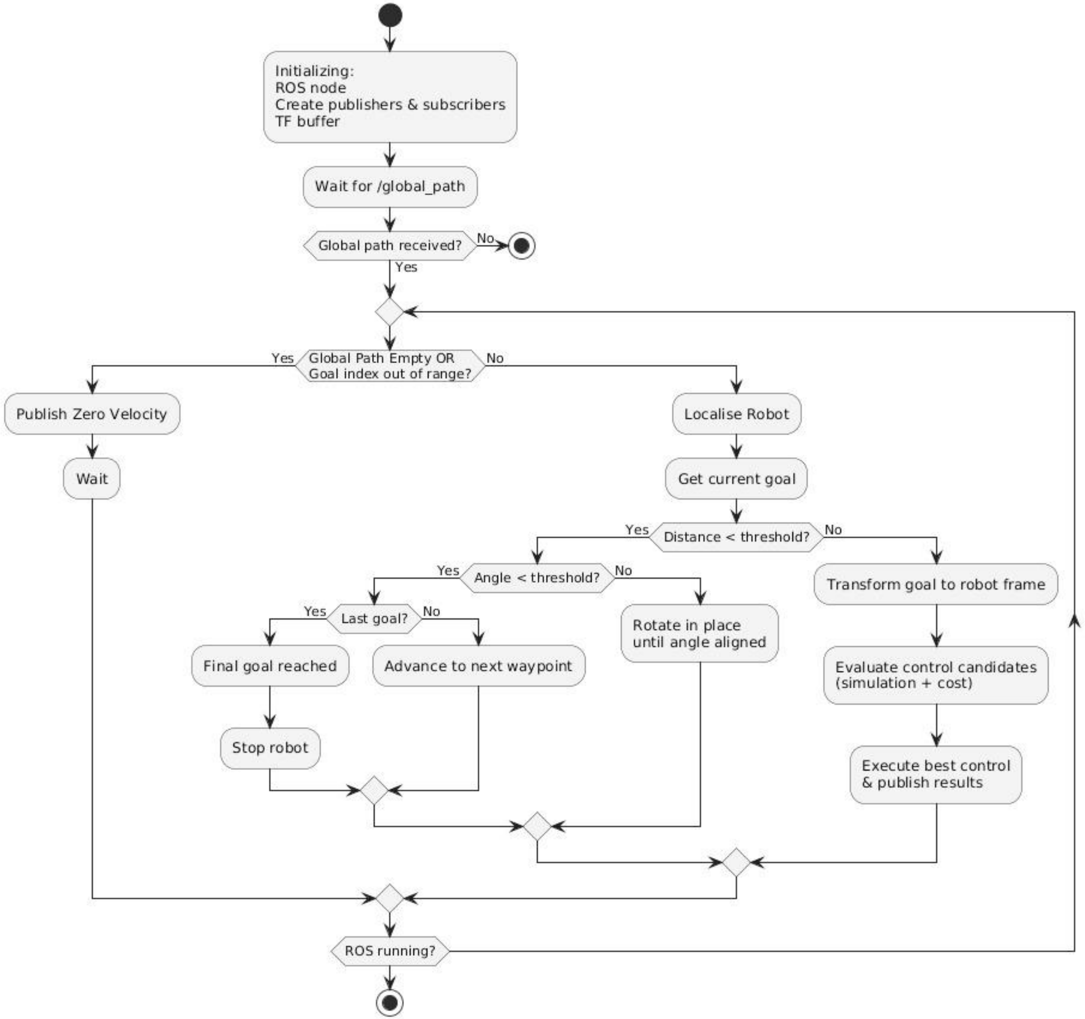
The **local planner** turns the rough path into **actual movements** the robot can physically execute.

Local Planner Approach

- The robot **tries several possible velocity commands**.
- For each command, it **simulates how the robot would move** for a short time.
- Each simulated motion creates a **trajectory**.
- Every trajectory is **scored using a cost function** (safety, smoothness, path following).
- The robot **chooses the trajectory with the lowest cost** and executes it.



Local Planner - Flow Chart



Initializing:
ROS node
Create publishers & subscribers
TF buffer

Wait for /global_path

Local Planner - Problems

01

Not turning enough before reaching goal
Reached first goal but had a hard time reaching second.

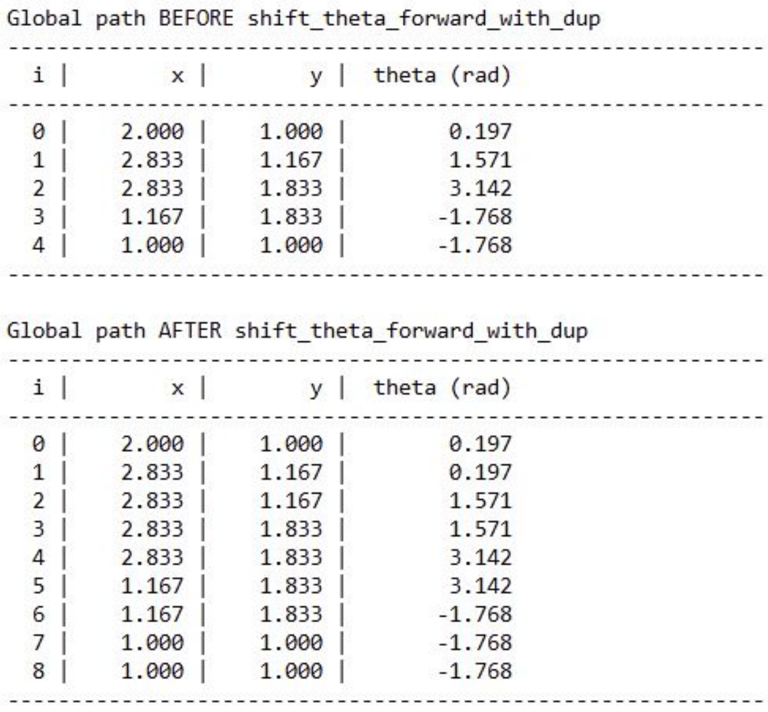
Solution:
When reaching the goal the robot turns in place until it is within an angle threshold before continuing to next goal.

02

Turned too much too early, instead of going straight.
Missing the goals completely.

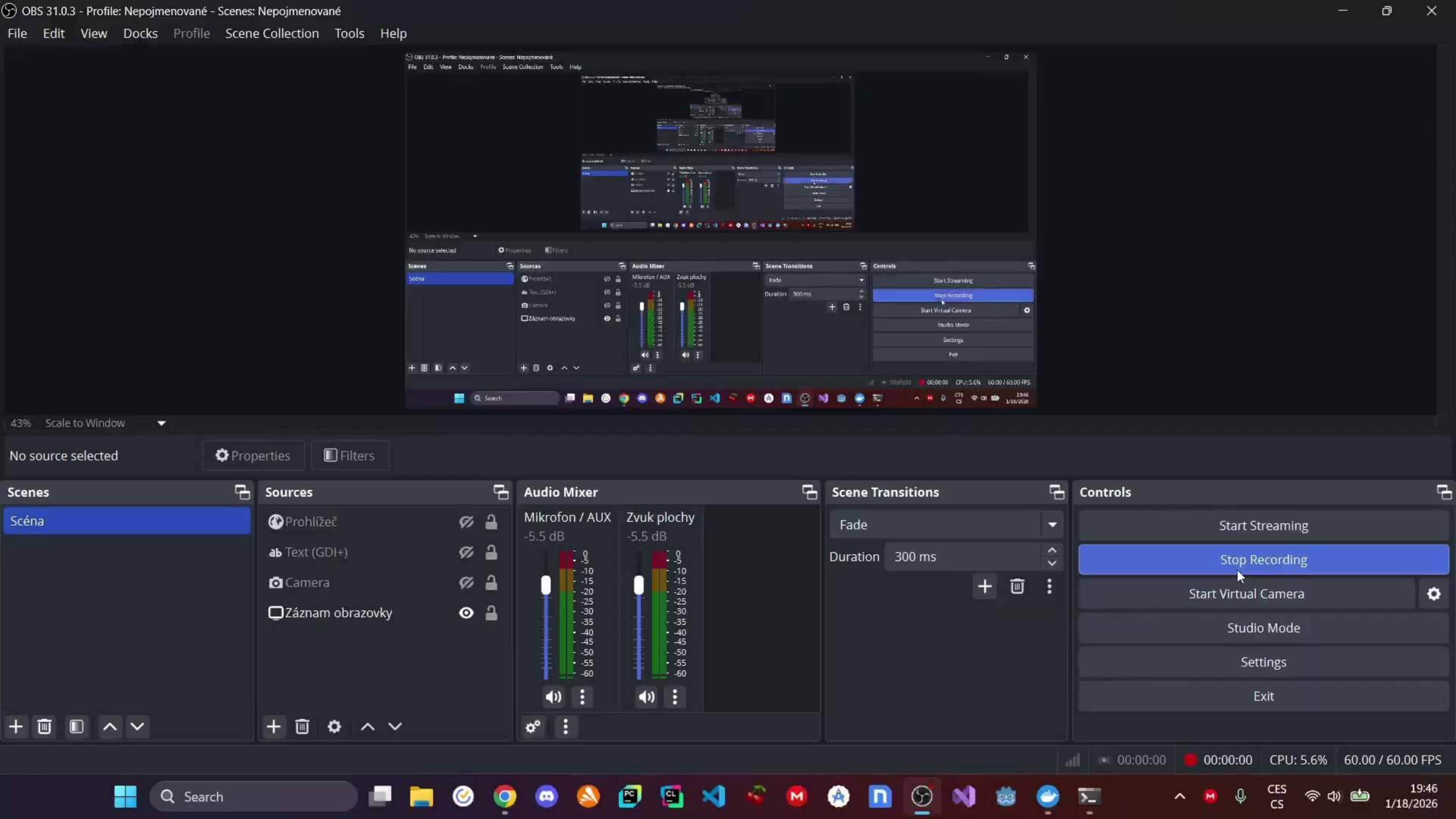
Solution:
Reconstruct global path with rotations one goal before the next one, ensuring correct “future” rotation.

Shifting Global Path



```
Global path BEFORE shift_theta_forward_with_dup
------------------------------------------------------------
  i |        x |        y |  theta (rad)
------------------------------------------------------------
  0 |    2.000 |    1.000 |        0.197
  1 |    2.833 |    1.167 |        1.571
  2 |    2.833 |    1.833 |        3.142
  3 |    1.167 |    1.833 |       -1.768
  4 |    1.000 |    1.000 |       -1.768
------------------------------------------------------------

Global path AFTER shift_theta_forward_with_dup
------------------------------------------------------------
  i |        x |        y |  theta (rad)
------------------------------------------------------------
  0 |    2.000 |    1.000 |        0.197
  1 |    2.833 |    1.167 |        0.197
  2 |    2.833 |    1.167 |        1.571
  3 |    2.833 |    1.833 |        1.571
  4 |    2.833 |    1.833 |        3.142
  5 |    1.167 |    1.833 |        3.142
  6 |    1.167 |    1.833 |       -1.768
  7 |    1.000 |    1.000 |       -1.768
  8 |    1.000 |    1.000 |       -1.768
------------------------------------------------------------
```



Fade

Duration 300 ms

00:00:00 00:00:00 CPU: 5.6% 60.00 / 60.00 FPS

Thank you