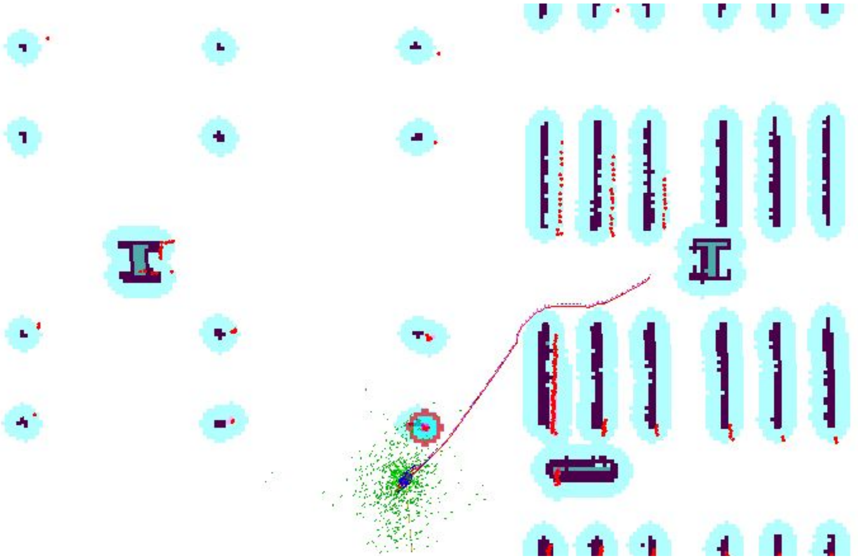
Mapping & Navigation
How do we do them in ROS 2?



UNIVERSITY OF EASTERN FINLAND

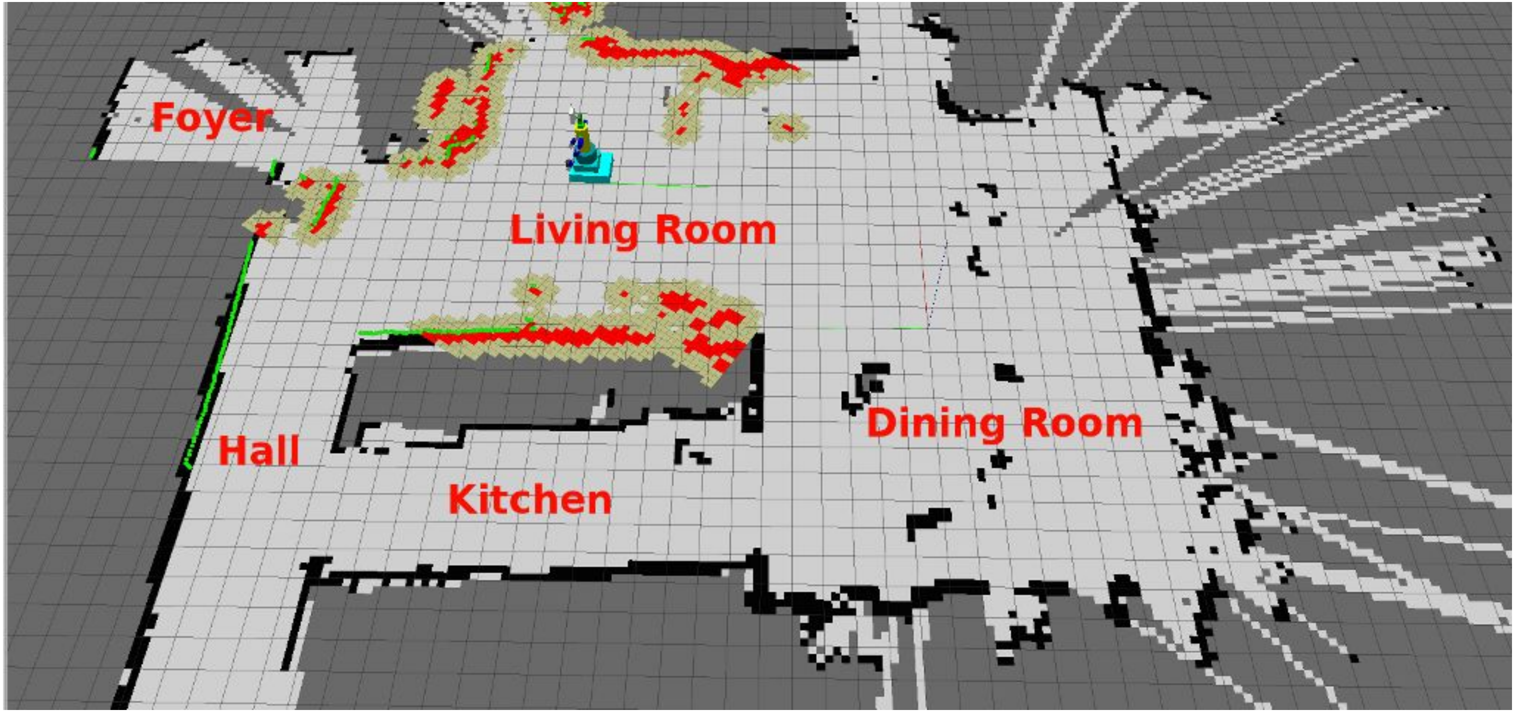
Henki Robotics

George-Cosmin Porusniuc
Co-Founder & Robotics Consultant @ Henki Robotics

During this lecture…

1. Mapping. SLAM.
    a. What is mapping. Why is it important? SLAM.
    b. Important terms
        - Laser Scan
        - Odometry
        - Pose graph
        - Scan-matching
        - Occupancy grid
    c. The mapping process
    d. slam-toolbox library
    e. Mapping examples
    f. Real-world challenges
2. (Autonomous?) Navigation
    a. What is it?
    b. Planning vs Controlling
    c. Nav2 library
    d. Real-world challenges

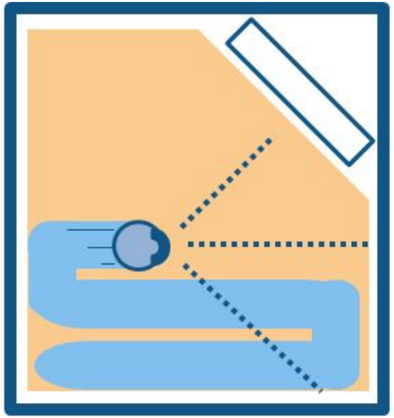
Mapping. SLAM



What is mapping? Why is it important? SLAM.

- **Mapping** - process of creating an internal representation of the outside environment by making use of the data gathered from observation sources (sensors)
- Without mapping the robot would have no internal representation of the outside world, and would not be able to take intelligent actions that would interact with the environment in a meaningful way.
- **SLAM - Simultaneous Localization and Mapping**
- **SLAM** is the process of creating or updating a map of an unknown environment while simultaneously keeping track of the robot's location within it.

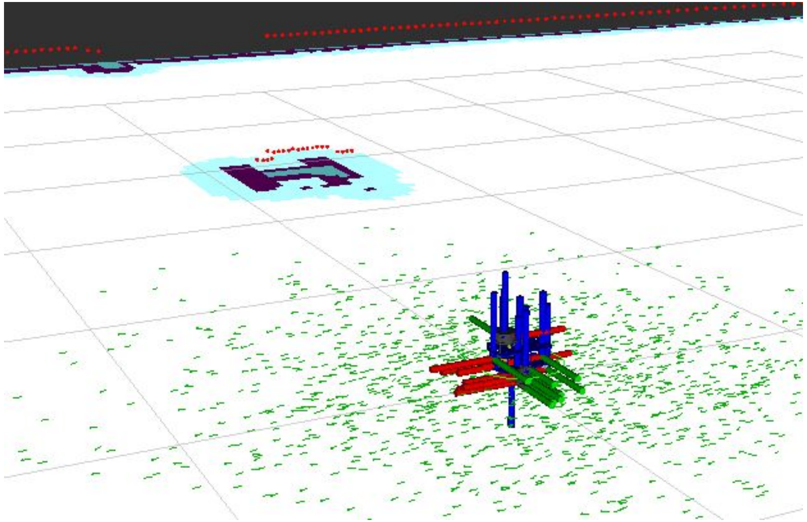
Without SLAM:
Cleaning a room randomly.



With SLAM:
Cleaning while understanding the room's layout.

Laser scan

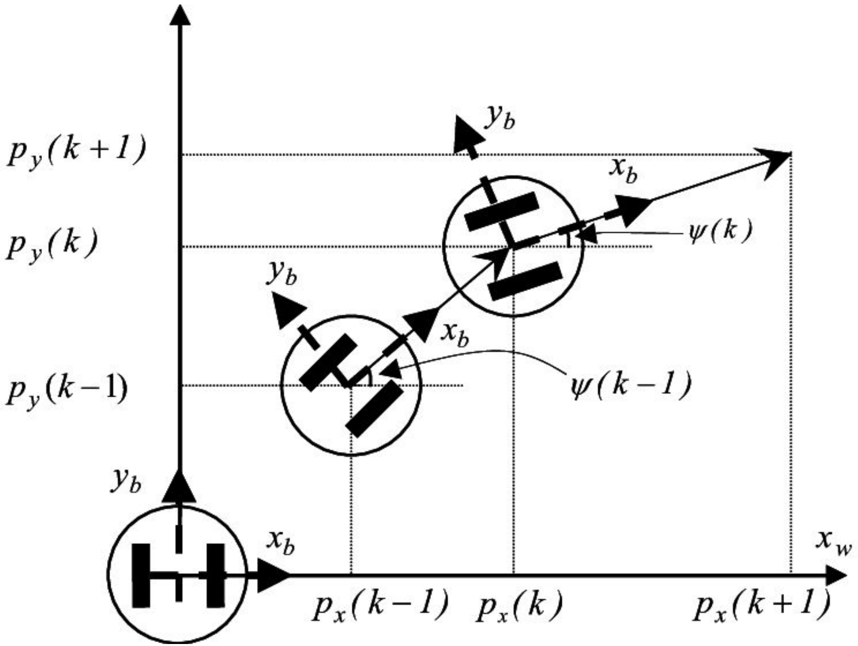
- It is an array of distance measurements visualized on a 2-dimensional plane.
- It is usually and best obtained from a laser-based sensor (LIDAR)
  - Can also be obtained with other depth-measuring sensors (depth cameras)



Odometry

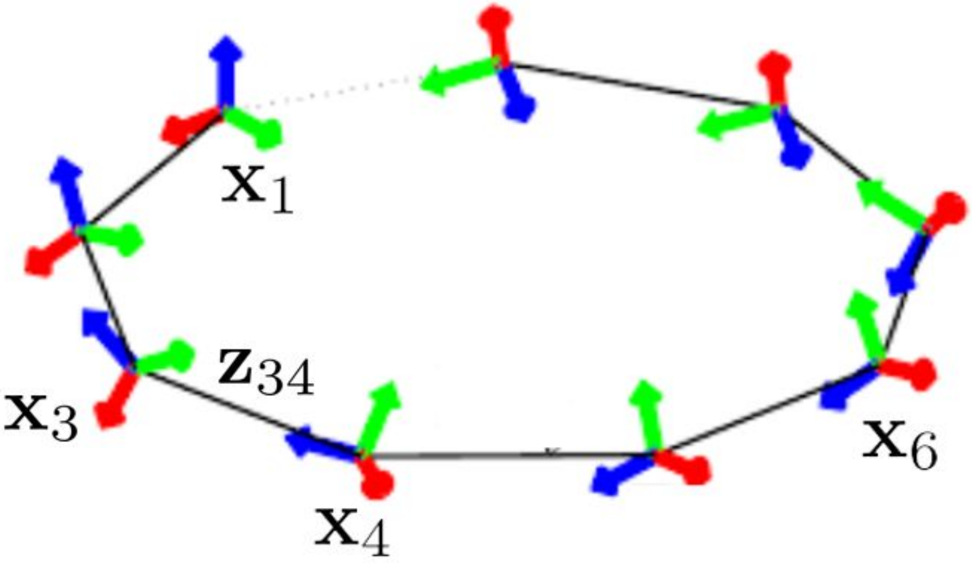
Recap:

- The estimation of a robot's position and orientation (**pose**) over time, using data gathered from the robot's sensors
- Keeps track of the robot's location in the environment, relative to a starting point - **pose estimation**
- Multiple types of odometry: wheel-based, visual, inertial, etc.



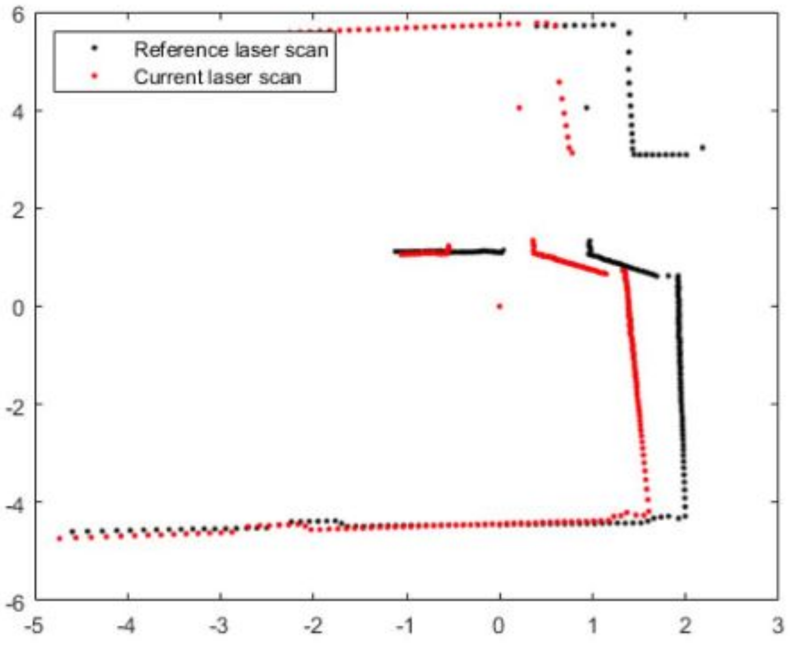
Pose graph

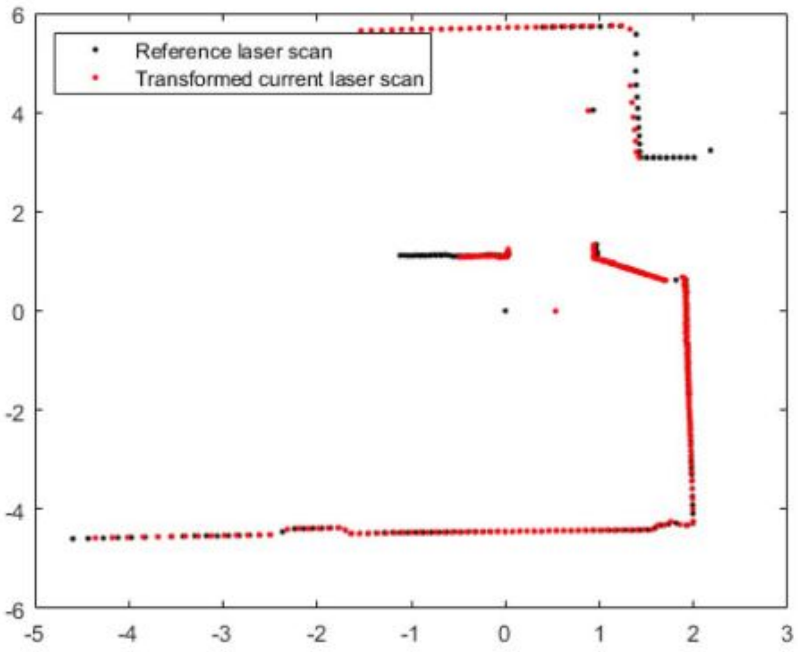
- It is an abstract **structure** that **correlates** the **poses** of the robot (estimated using odometry), **with the laser scan readings**, during each mapping step.



Scan-matching

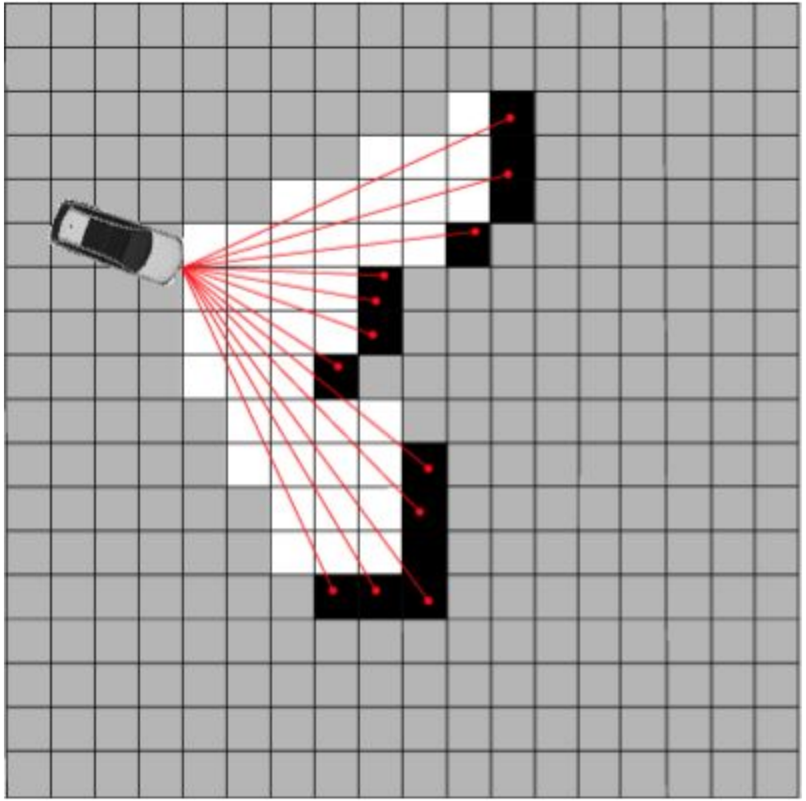
- Using just odometry data while building maps is not a good approach, as movement is inherently noisy.
- The sensors that are fixed on the robot body are often times more precise than the estimated odometry
- **Scan-matching**: the incremental alignment of the currently built map, with the current laser scan received from the lidar





Occupancy grid

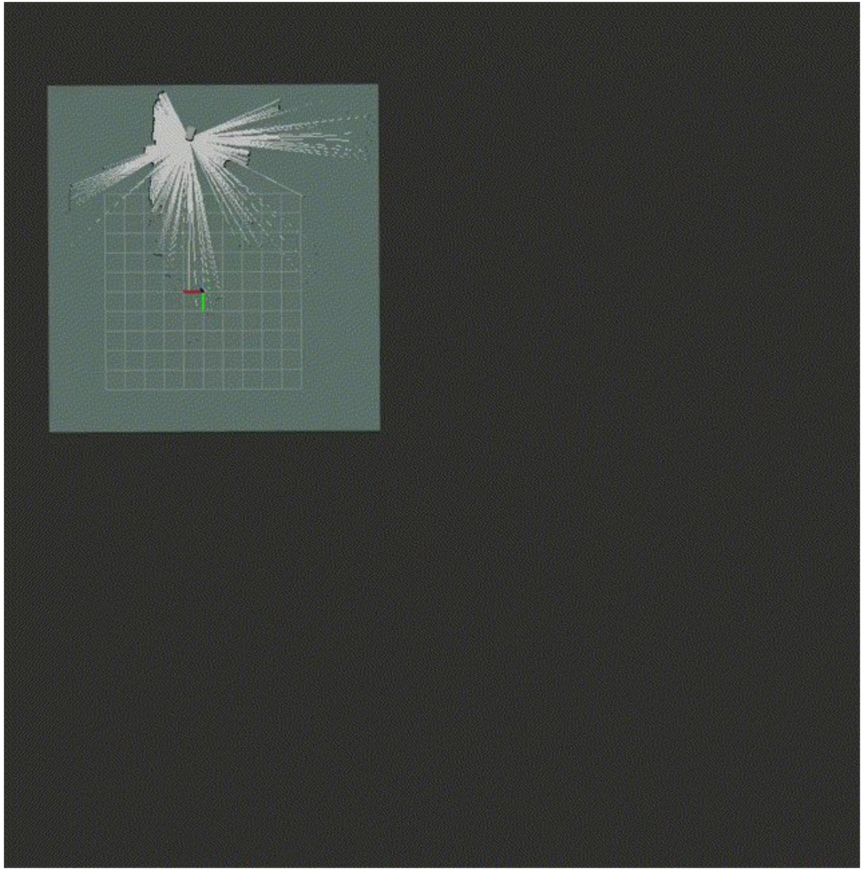
- A 2D grid-based representation of the outside world.
- It is built with a certain cell resolution, based on the size of the robot and the SLAM configuration.
- Easy to understand - each cell holds a value. The value tells whether that cell is free, occupied or unknown.
    - Values between [-1, 100].
    - Most commonly the range of values is discrete: {-1, 0, 100}
        - -1 -> unknown space
        - 0 -> free space
        - 100 -> occupied space (obstacle)



The mapping process

At each timestep:

1. Receive new scan from LIDAR
2. Receive odometry from sensors. Estimate robot pose.
3. Save the current scan and estimated pose to the pose graph.
4. Perform scan-matching
5. Update the built occupancy grid based on the current pose and scan
6. As we visit previous locations, we must use the pose graph and senor reading to perform **loop closure detection**

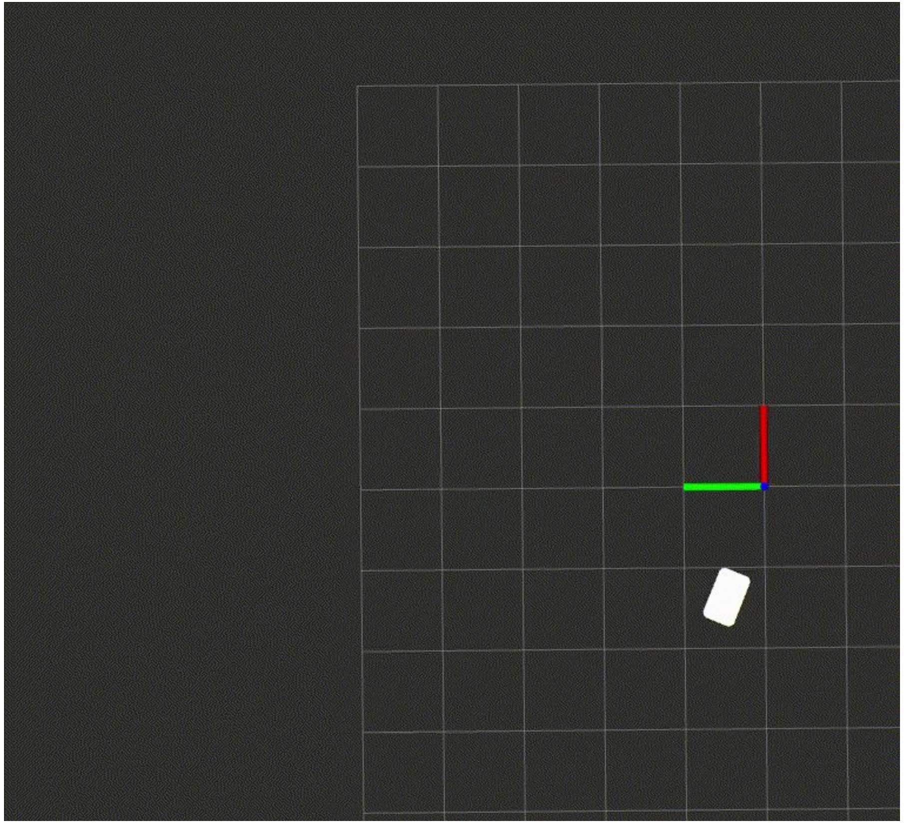


Mapping in ROS 2 - slam-toolbox

- Slam Toolbox is a set of tools and capabilities for 2D SLAM
- One of the most popular 2D SLAM packages in ROS 2

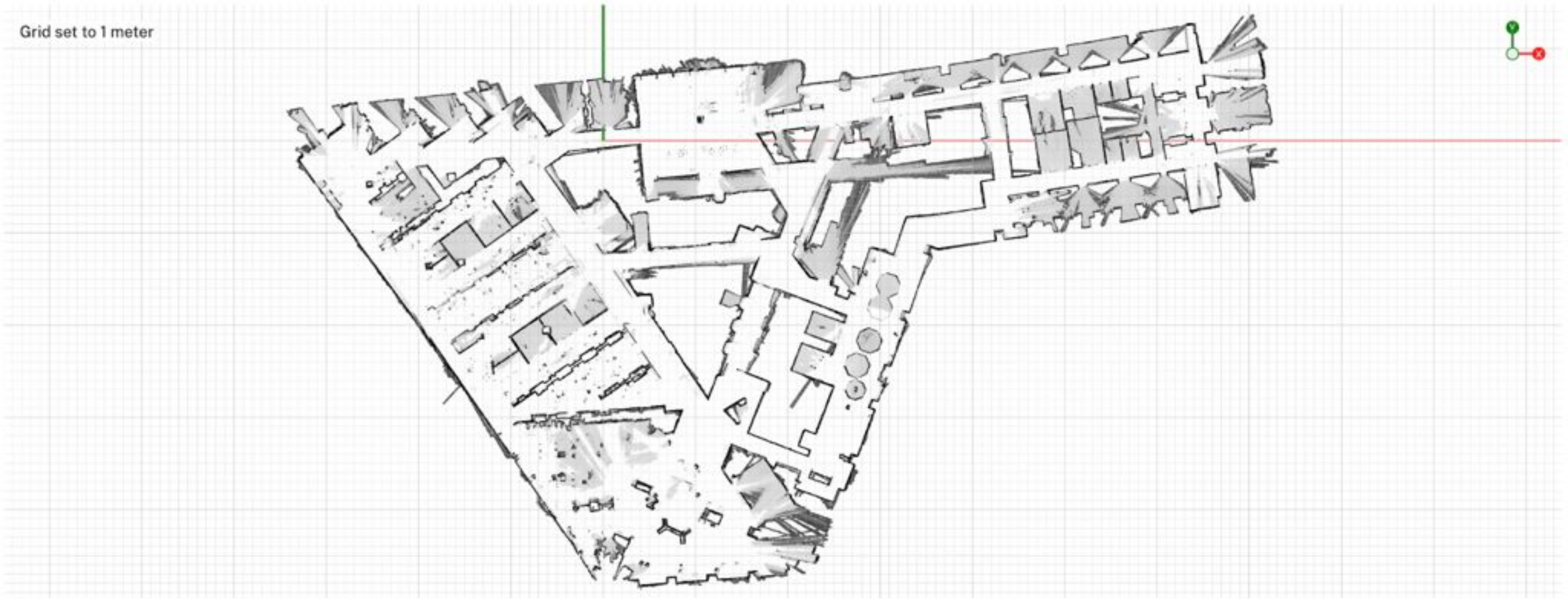
Provides:

- Ordinary point-and-shoot 2D SLAM mobile robotics folks expect (start, map, save pgm file) with some nice built in utilities like saving maps
- Continuing to refine, remap, or continue mapping a saved (serialized) pose-graph at any time
- life-long mapping
- RVIZ plugin for interacting with the tools
- …everything you need for your 2D SLAM robotics needs!



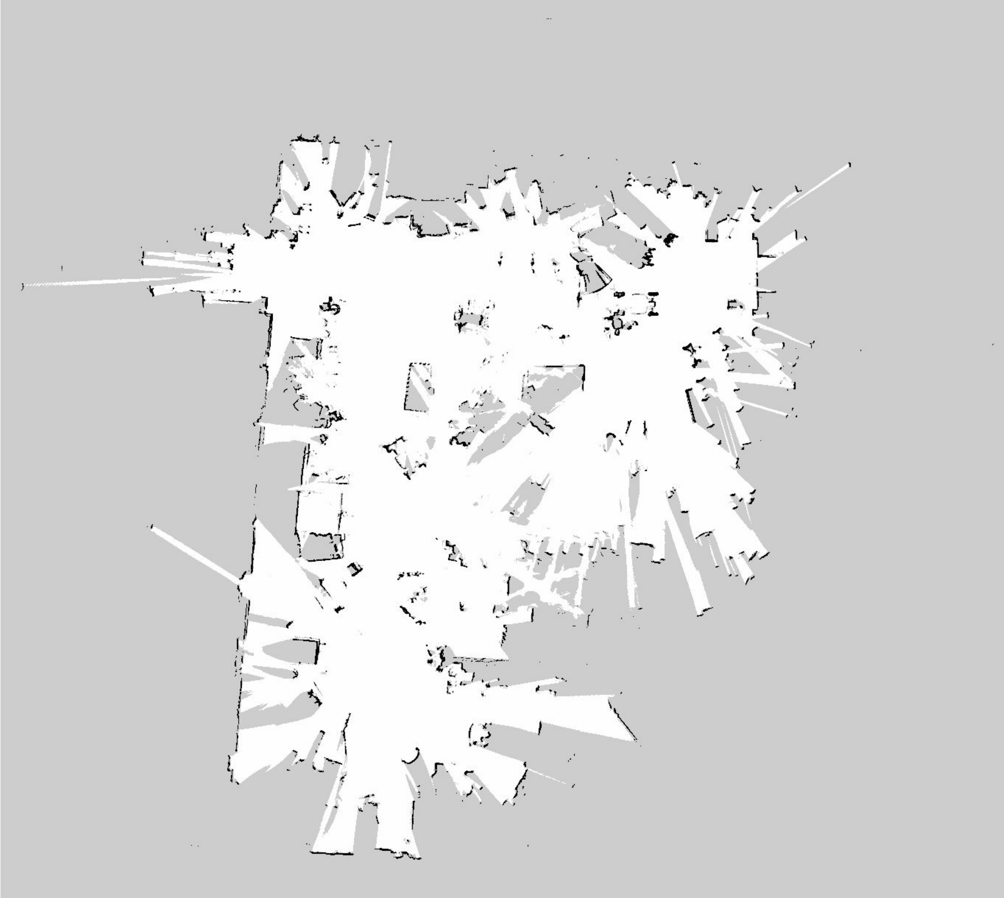
https://github.com/SteveMacenski/slam_toolbox

Good map example. Why is this map good?



https://docs.viam.com/services/slam/

Bad map example. Why? How did it end up like this?

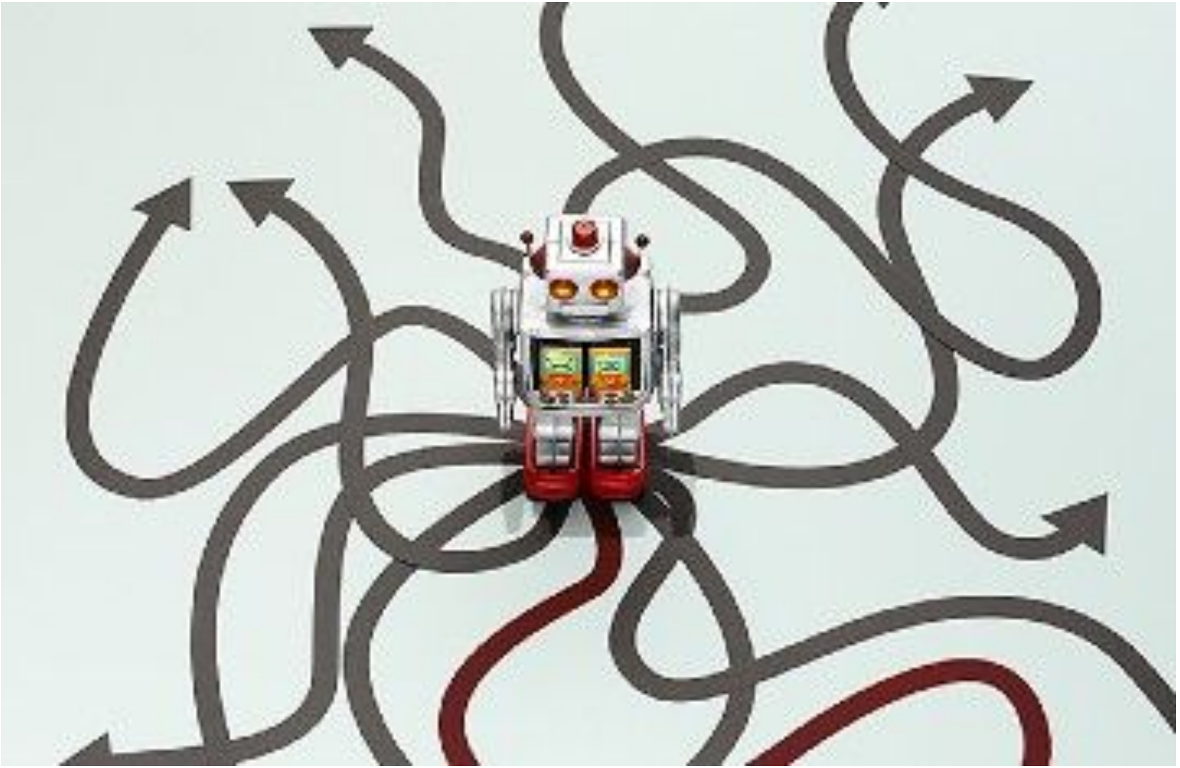


https://github.com/cartographer-project/cartographer_ros/issues/639

Challenges of mapping in the real world

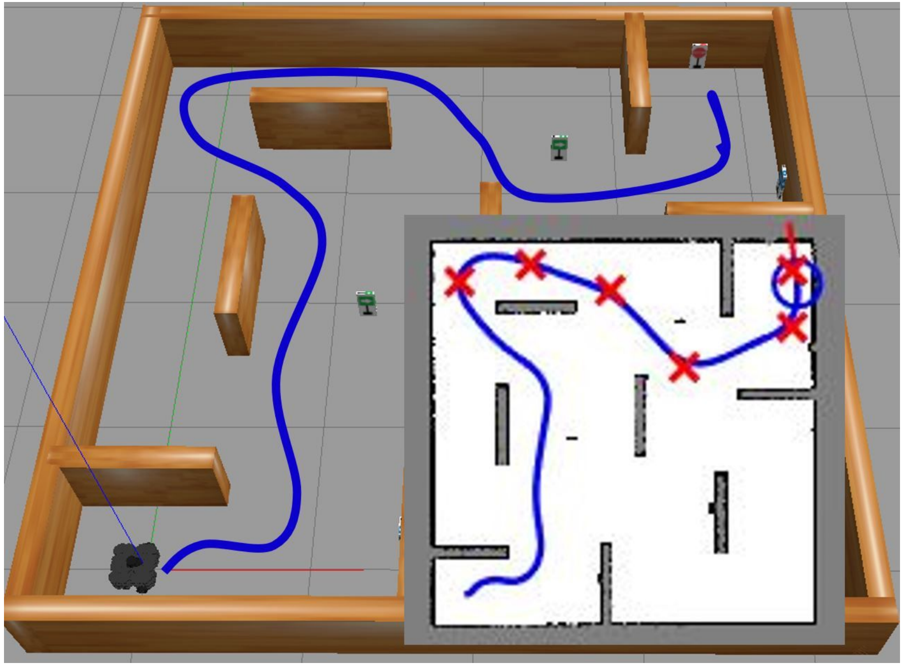
- Dynamic Environments
  - Moving objects: people, vehicles, etc.
  - Non-static features: doors, trees
- Sensor limitations
  - Noisy data
  - Sensor range
  - Bad calibration
- Large environments
  - Memory and CPU constraints
  - Loop closure detection
  - Bad real-time performance
- Feature-Sparse Areas (hallways, parking lots, open fields)
- Feature-Rich Areas (overcrowded areas)
- Localization drift

(Autonomous?) Navigation



What is (autonomous) navigation?

- The process of moving a robot autonomously between 2 locations in 3D space.
- Done by planning a path, and then executing that path without any human intervention.
- The planning is done by a **planner** component
- The execution of the plan is done by a **controller** component



Planning VS Controlling

Planning

- Reading information from the sensors and other components and building a path. Also checking if there is any feasible path between the 2 locations.

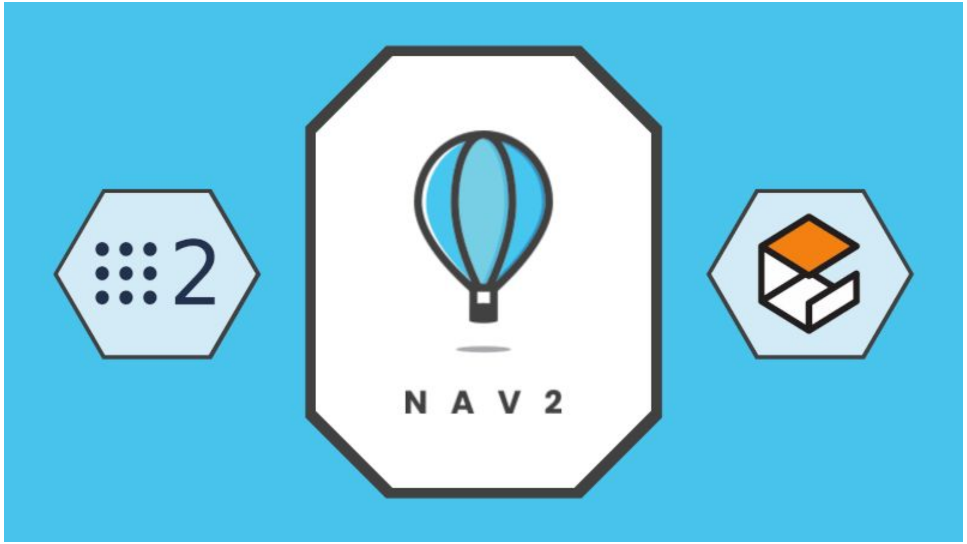
Controlling

- Sending commands to the robot actuators so that the planned path can be followed as closely as possible.
- Also known as “local planning”

https://clementmihailescu.github.io/Pathfinding-Visualizer/#

Navigation in ROS 2

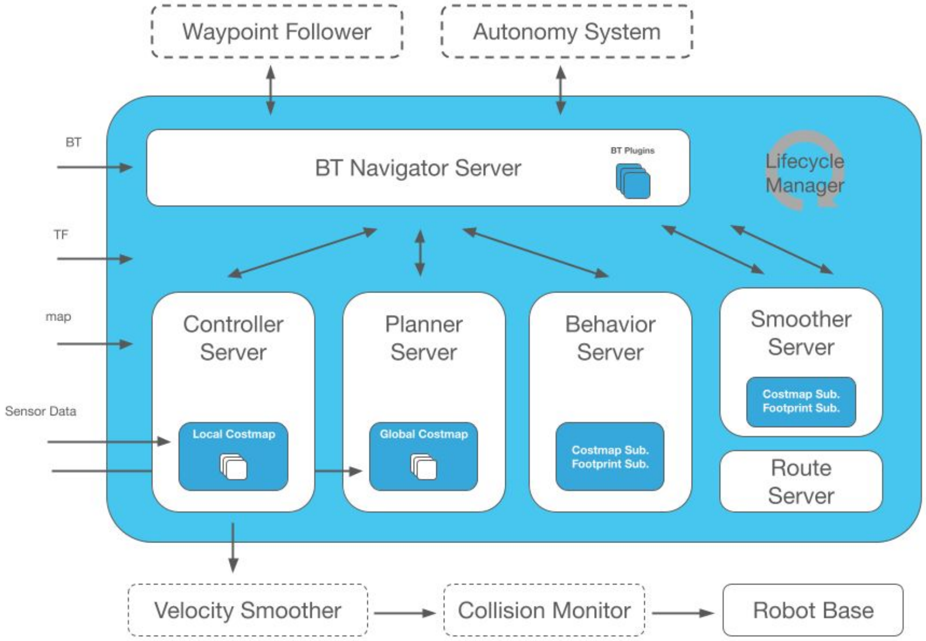
The Nav2 library



Contents

Nav2 Introduction

- The most popular autonomous driving package in ROS 2. Used by tens of companies at the moment!
- provides **perception**, **planning**, **control**, **localization**, **visualization**, and **much more** to build highly reliable autonomous systems
- Nav2 uses **behavior trees** to create customized and intelligent navigation behavior
- https://www.youtube.com/watch?v=CYaN43TJANc&t=7s



What is needed to integrate Nav2?

- Your robot must have
  - Odometry from the wheels
  - Correctly configured TFs (tf2 library)
  - Sensors (Lidar, Camera)
  - Correctly calculated robot footprint
- A map (occupancy grid) must be built beforehand using a SLAM package
- Nav2 parameter configuration

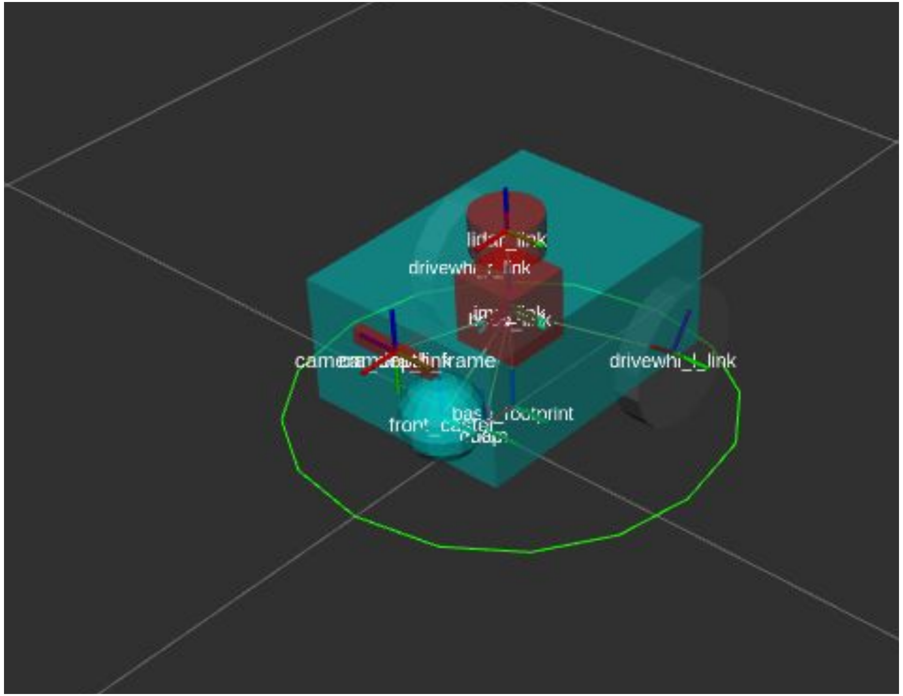


Image source: https://docs.nav2.org/setup_guides/footprint/setup_footprint.html

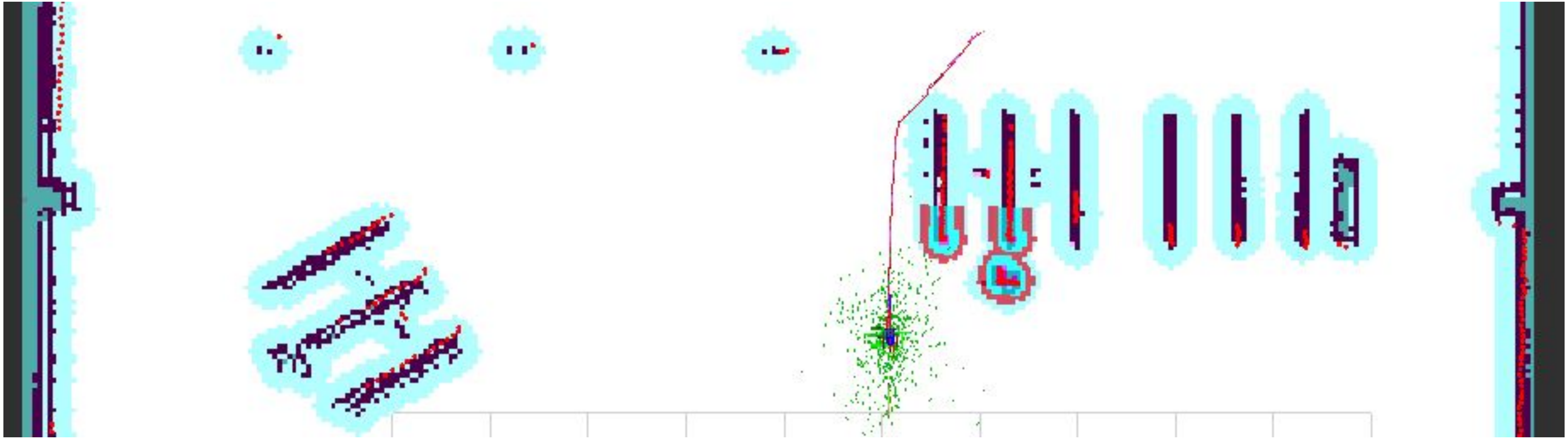
Environmental representation system in Nav2

- **3 maps are used**

Static map

Global Costmap

Local Costmap



Environmental representation - Static Map

Purpose: Provides a pre-built map of the environment for navigation. Does not adapt to dynamic changes. It is built once during SLAM.

Features: Represents fixed, unchanging obstacles (e.g., walls, furniture). It is used as a base layer for costmaps.

Format: 2D Occupancy Grid represented as a 1-D array

- Can have varying resolution. Usually the resolution is defined in the map metadata ".yaml" file. Resolution defines the area covered by one pixel of the map. Can be, for example, 5 square cm, less, or more, depending on the robot.
- Values between [-1, 100].
- Most commonly the range of values is discrete: {-1, 0, 100}
    - -1 -> unknown space
    - 0 -> free space
    - 100 -> occupied space (obstacle)

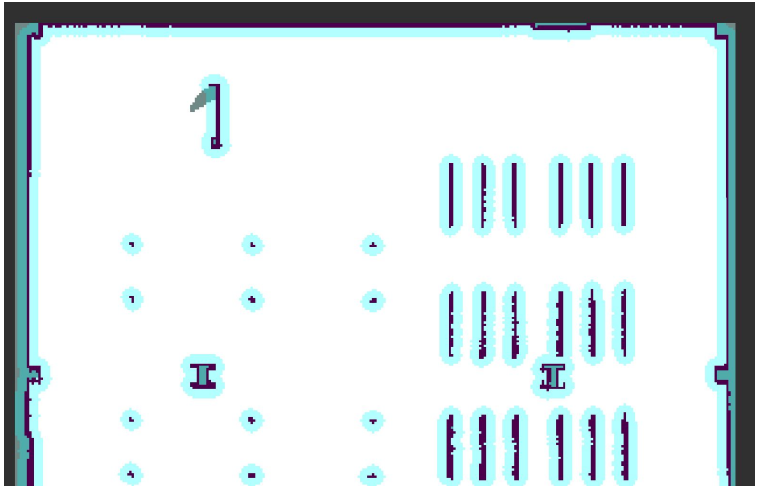
Environmental representation - Global CostMap

Purpose: Provides a long-term, global view for *path planning* over the entire environment

Features:

- Static Layer: Includes fixed obstacles from pre-built maps.
- Dynamic Layer: Incorporates real-time sensor data.
- Inflation Layer: Adds safety margins around obstacles.

Dynamic Updates: Reacts to changes in the environment using live sensor data (Laser, LIDAR, camera). Acts as a “memory” of dynamic obstacles.

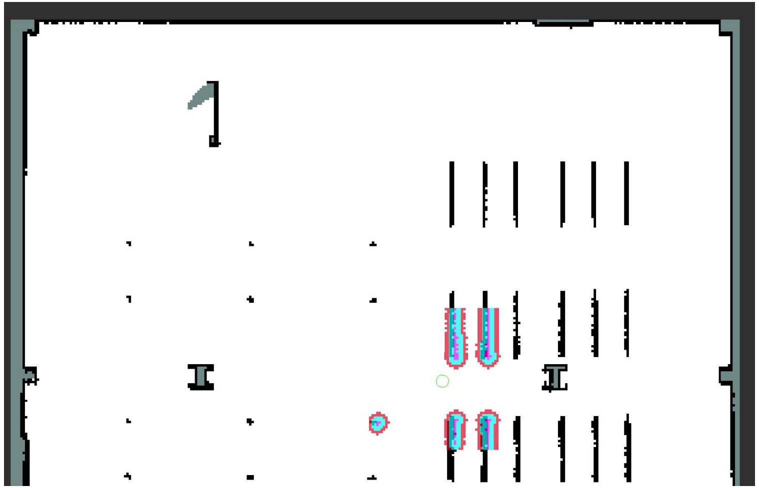


Environmental representation - Local CostMap

Purpose: Offers a short-term, detailed view of the robot's immediate surroundings for real-time **obstacle avoidance**.

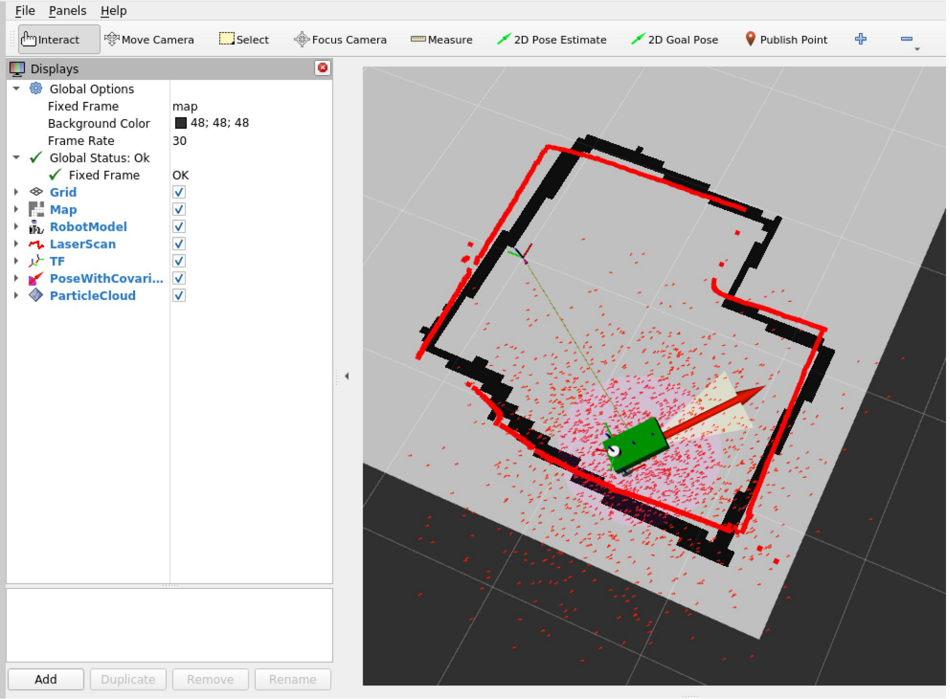
Features:

- Focuses on short-term obstacle avoidance.
- Continuously updates with live sensor data (e.g., lidar, cameras).
- Works in the robot's coordinate frame (base_link).



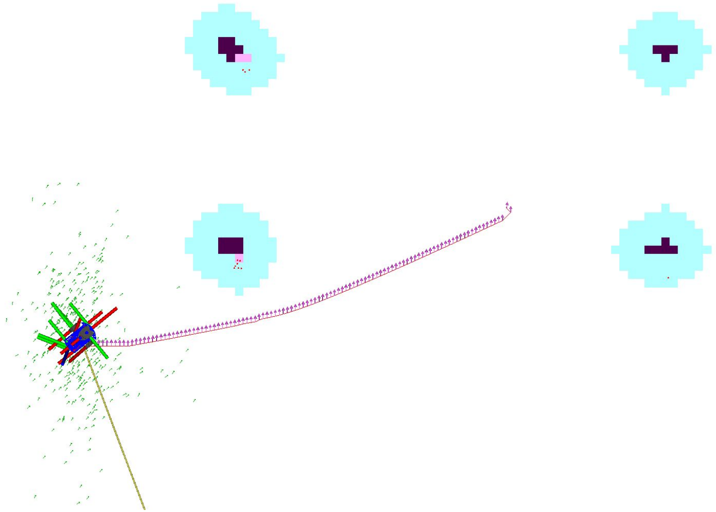
Localization - AMCL

- Adaptive Monte Carlo Localization (AMCL)
- probabilistic localization module which estimates the position and orientation (i.e. Pose) of a robot in a given known map using a 2D laser scanner (scan matching 💡)
- AMCL implements the server for taking a static map and localizing the robot within it, in the context of Nav2



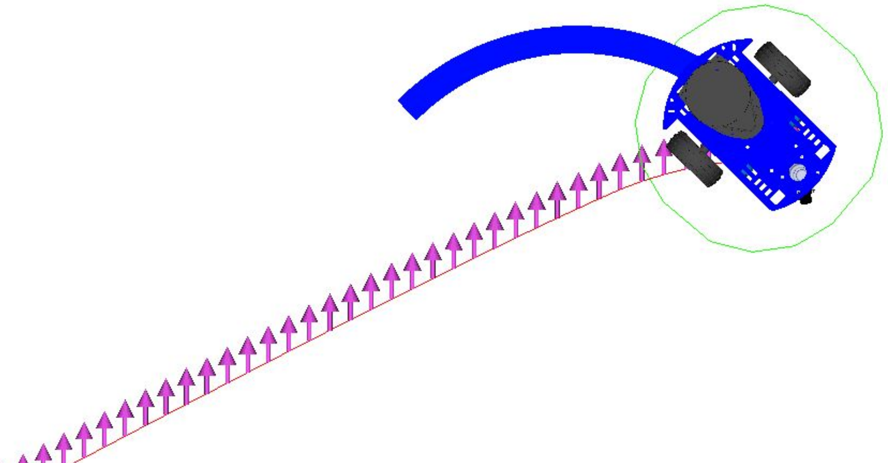
Path planning in Nav2 - Global Path Planner

- Computes a collision-free path from the robot's current position to the goal
- Uses the **global costmap** to plan paths around the obstacles
- Common algorithms: A*, Dijkstra, and NavFn.



Path planning in Nav2 - Local Path Planner

- Also known as **Controller.** Follows the globally computed path by moving the robot.
- Uses the **local costmap** to avoid dynamically detected obstacles
- Makes sure that the robot's movements are feasible:
    - Ensures that the velocity commands do not exceed the hardware limits
    - Smoothens the trajectories; Shortcuts for sharp corners
    - Drive-system constraints: For example differential, holonomic, ackermann drive systems



Behavior Trees

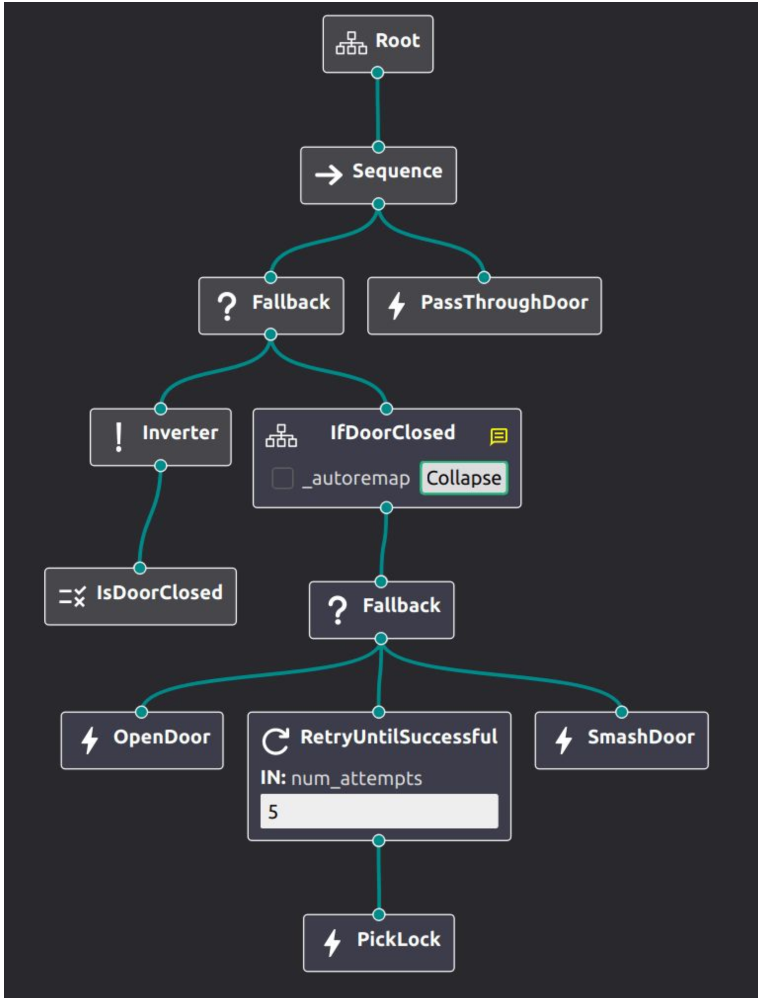
Purpose: Orchestrate robot actions and decision-making in a structured and flexible way.

Key Features:

- **Modular**: Breaks behaviors into reusable, independent components.
- **Hierarchical**: Combines tasks in a tree structure for clear logic flow.
- **Reactive**: Dynamically adjusts based on the robot's state and environment.

Advantages: Easy to debug and extend. Supports parallel and conditional execution of tasks.

Example: A robot encountering a closed door and deciding on what to do.



Other useful Nav2 features

- Autonomous Docking
- Collision Monitor
- Map Filters
    - Keepout zones
    - Speed restricted areas

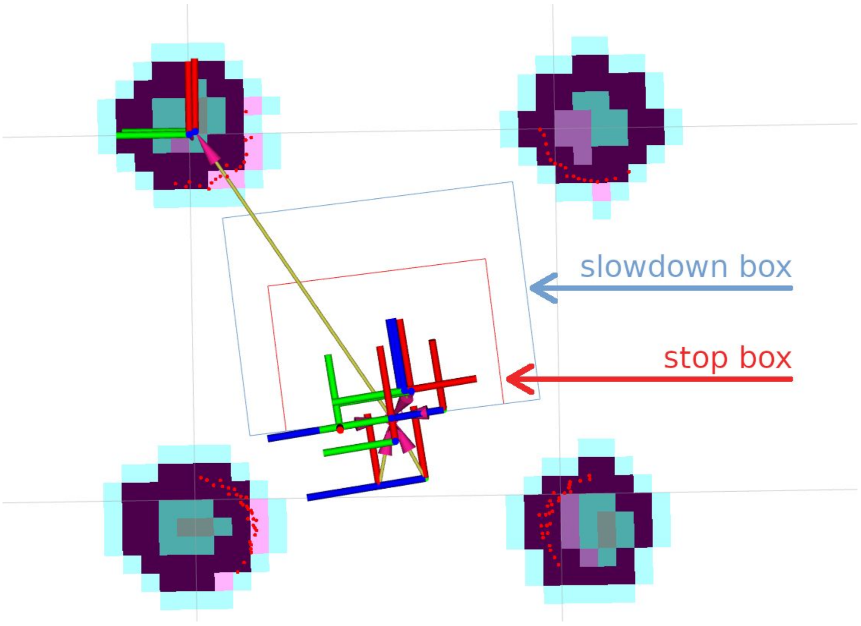


Image source: https://docs.nav2.org/tutorials/docs/using_collision_monitor.html

Real-world robot navigation challenges

- Dynamic/Unstructured environments
  - Moving obstacles
  - Environment changes: weather, ground surface. Sometimes there are no walls in areas where robots should navigate.
  - Unpredictable behaviour - humans
- SLAM complexity
  - Feature-poor areas are hard to map, maps are sometimes of a bad quality
- Sensor limitations
  - Lighting, reflections
  - Bad calibrations
- Odometry drift
- Hardware constraints
  - Battery life

Thank you! 👋