

Easy ::: Nav

An open-source framework
for navigating everywhere

Prof. Dr. Francisco Martín Rico

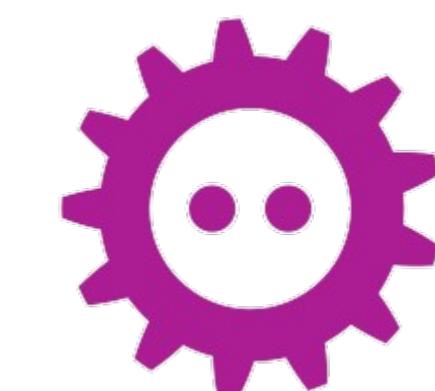
francisco.rico@urjc.es

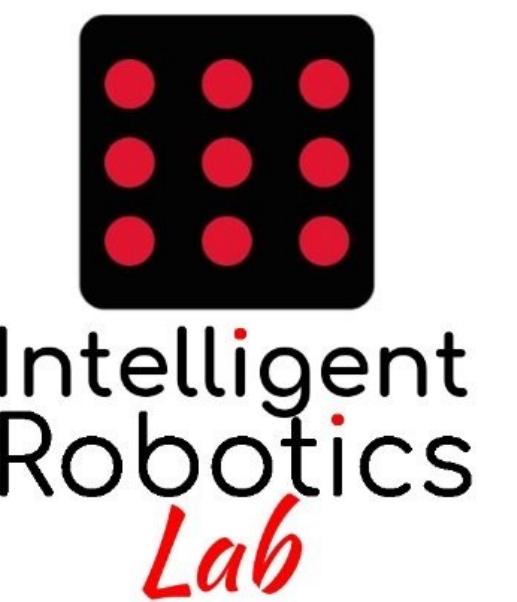
@fmrico

Dr. Francisco Miguel Moreno

franciscom.moreno@urjc.es

@butakus





[Overview](#) [Repositories 161](#) [Projects](#) [Packages](#) [Stars 22](#)

Francisco Martín Rico
fmrico

[Follow](#)

Doctor Engineer on Robotics. Full Professor at Rey Juan Carlos University. Leader of the Intelligent Robotics Lab

586 followers · 14 following

Rey Juan Carlos University
Madrid
www.gsyc.es/~fmartin
[@fmrico](https://twitter.com/fmrico)

Achievements

x3

Contribution activity

January 2026 · 2026 · 2025

1,446 contributions in the last year

Learn how we count contributions

Less More

[Overview](#) [Repositories 36](#) [Projects](#) [Packages](#) [Stars](#)

Francisco Miguel Moreno
Butakus

[Follow](#)

15 followers · 4 following

Spain

Achievements

x2

Highlights

PRO

Organizations

Block or Report

405 contributions in the last year

2026 · 2025 · 2024 · 2023 · 2022 · 2021 · 2020 · 2019

Learn how we count contributions

Less More

Activity overview

Contributed to **Butakus/pikaball-revamped**, **EasyNavigation/EasyNavigation**, **Butakus/ros2-env** and 26 other repositories

4% Code review

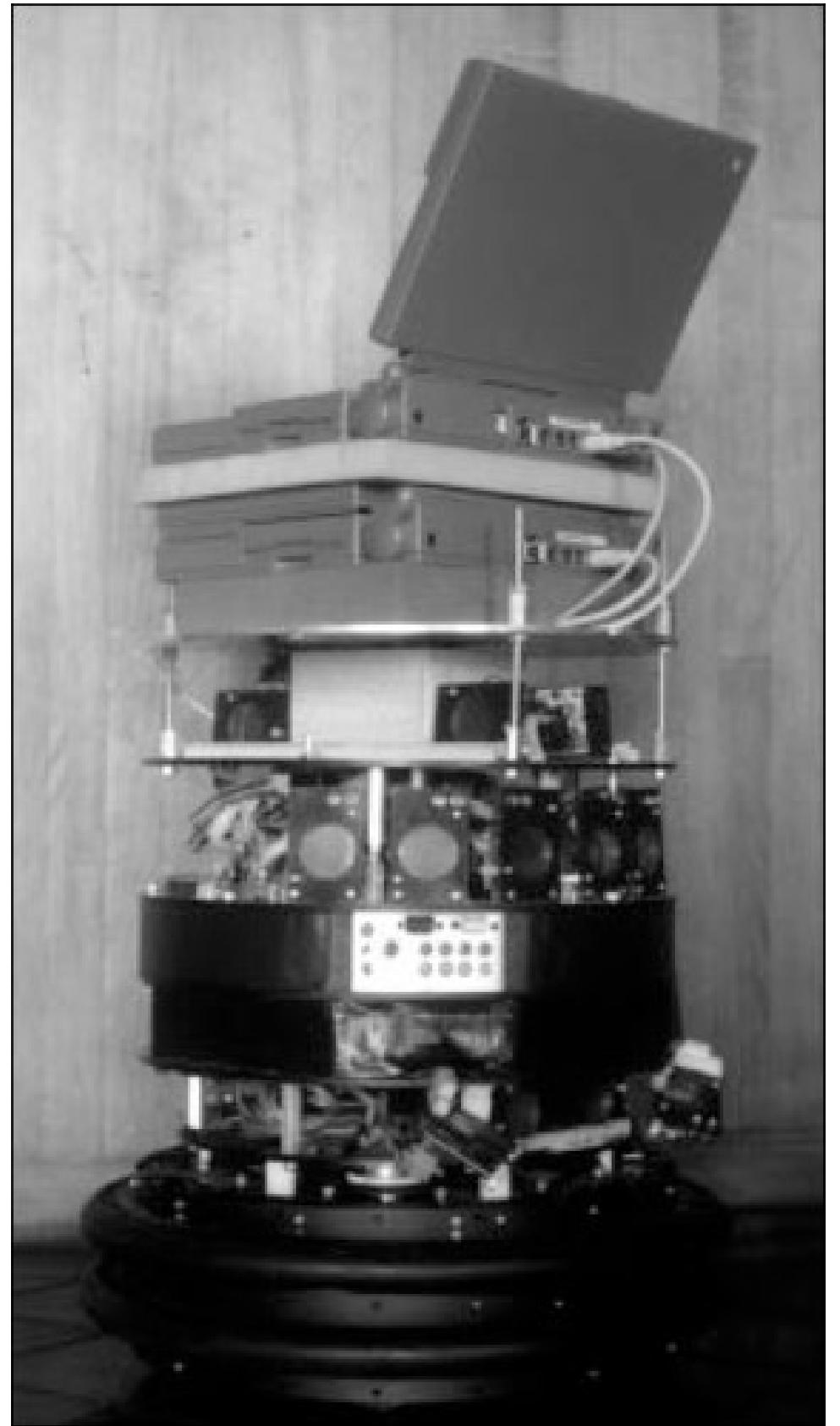
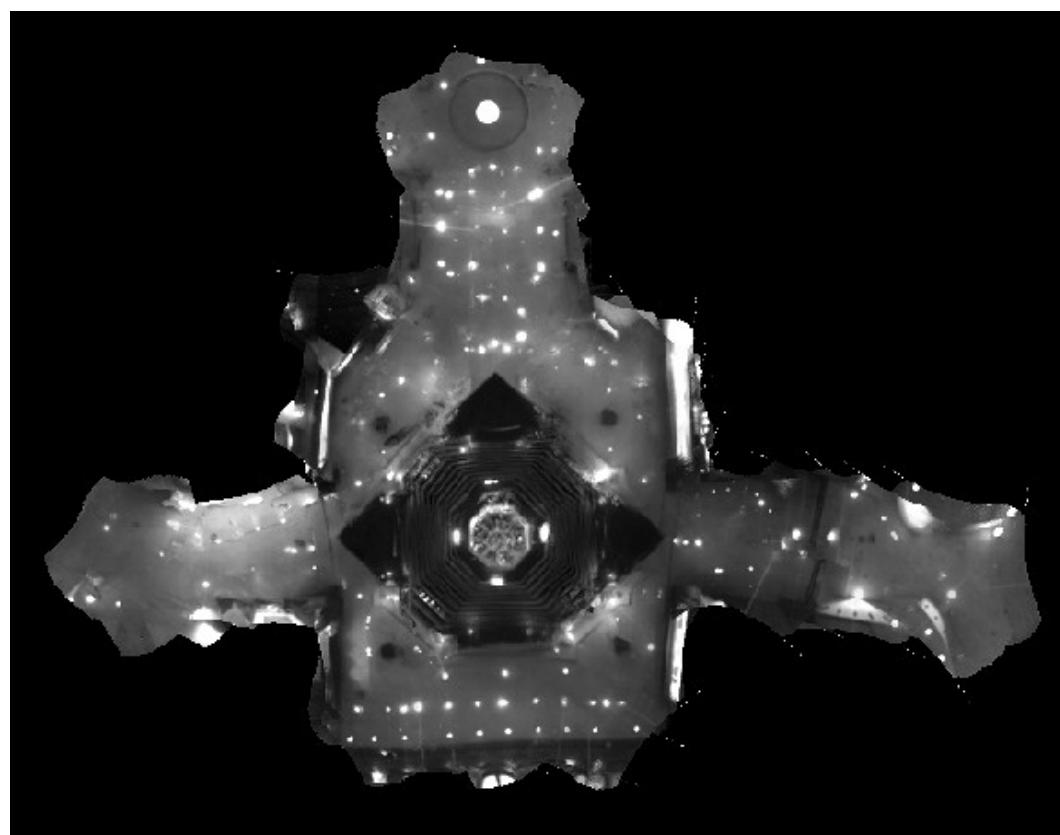
Introduction

Some background...

Introduction

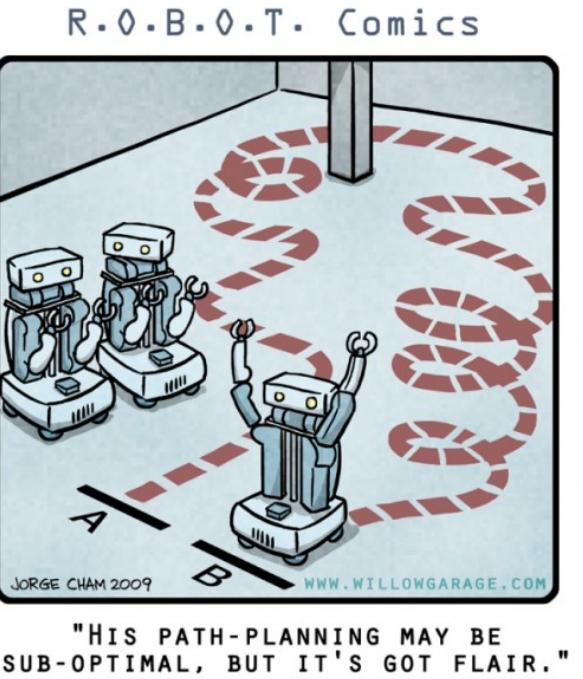
Robot navigation

- It is a fundamental skill for a mobile robot
- A very active field since more than half a century ago
- ¿Is it solved?
- Lack of standards → Re-implementation from scratch
- ROS



Introduction

ROS 1: Move Base



The Office Marathon: Robust Navigation in an Indoor Office Environment

Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, Kurt Konolige

ICRA 2010

The Office Marathon: Robust Navigation in an Indoor Office Environment

Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, Kurt Konolige

Willow Garage Inc., USA

{eitan,berger,tfoote,gerkey,konolige}@willowgarage.com

Abstract— This paper describes a navigation system that allowed a robot to complete 26.2 miles of autonomous navigation in a real office environment. We present the methods required to achieve this level of robustness, including an efficient Voxel-based 3D mapping algorithm that explicitly models unknown space. We also provide an open-source implementation of the algorithms used, as well as simulated environments in which our results can be verified.

I. INTRODUCTION

We study the problem of robust navigation for indoor mobile robots. Within this well-studied domain, our area of interest is robots that inhabit unmodified office-like environments that are designed for and shared with people. We want our robots to avoid all obstacles that they might encounter, yet still drive through the tightest spaces that they can physically fit. We believe that reliable navigation of this kind is a necessary prerequisite for any useful task that an indoor robot might perform.

While many robots have been shown to navigate in office-like environments, existing approaches invariably require some modification of the environment, or do not allow the robot to negotiate tight spaces. Most indoor robots rely on a planar or quasi-planar obstacle sensor, such as a laser rangefinder or sonar array. Because vertical structure dominates man-made environments, these sensors are positioned on the robot to detect obstacles along a horizontal slice of the world. The result is a robot that can easily avoid walls but will miss chair legs, door handles, table tops, feet, etc. Collisions are avoided by either modifying the environment (e.g., removing chairs, covering tables with floor-length tablecloths), or adding artificial padding (e.g., inflate obstacles by the maximum expected length of a person's foot), which prevents



Fig. 1. The PR2 avoiding a table after passing through a narrow doorway.

through the tightest spaces that the robot can fit. At the core of our work is an efficient technique for constructing, updating, and accessing a high-precision three-dimensional Voxel Grid. This structure encodes the robot's knowledge about its environment, classifying space as free, occupied, or unknown. Using this grid, the robot is able to plan and execute safe motions in close proximity to obstacles.

Through extensive experimentation, we have established that our approach is safe and robust. During endurance runs with the PR2, we regularly left a robot running unattended overnight in our office building, which was not modified to accommodate the robot. The software described in this paper is available under an open-source license,¹ and we encourage others to experiment with and use our code.



[Link to video]

Introduction

ROS 2: Nav2

- Main author: **Steven Macenski**
- Move Base stack reimplementations
- An example of good practices in ROS 2
- More algorithms, plugins, components...
- Contributions: 59 collaborators, Intel, Samsung

The screenshot shows the ROS 2 Navigation website. The left sidebar includes links for Getting Started, Build and Install, Navigation Concepts, Tutorials, Configuration Guide, Navigation Plugins, Migration Guides, Getting Involved, About and Contact, and Projects for 2020 Summer Student Program. The main content area displays a video feed of a white mobile robot navigating through a hallway with people. Below the video, there's an "Overview" section with text about the Navigation 2 project and its goals. At the bottom, there's a note about the expected inputs to Navigation2 (Nav2).

Navigation 2
latest
ROS
Getting Started
Build and Install
Navigation Concepts
Tutorials
Configuration Guide
Navigation Plugins
Migration Guides
Getting Involved
About and Contact
Projects for 2020 Summer Student Program

ROS 2 Navigation

Sensor Data

Amcl

Map Server

TF Transforms

map

map

nav_msgs/msg/OccupancyGrid

Waypoint Follower

NavigateToPose

Behavior Tree Plugins

Behavior Tree

XML

CP

FP

Recovery Server

Recovery Plugins

Costmap Sub. Footprint Sub.

Controller Server

Control Plugins

Local Costmap

Planner Server

Planning Plugins

Global Costmap

Sensor Data

sensor_msgs/msg/PointCloud2

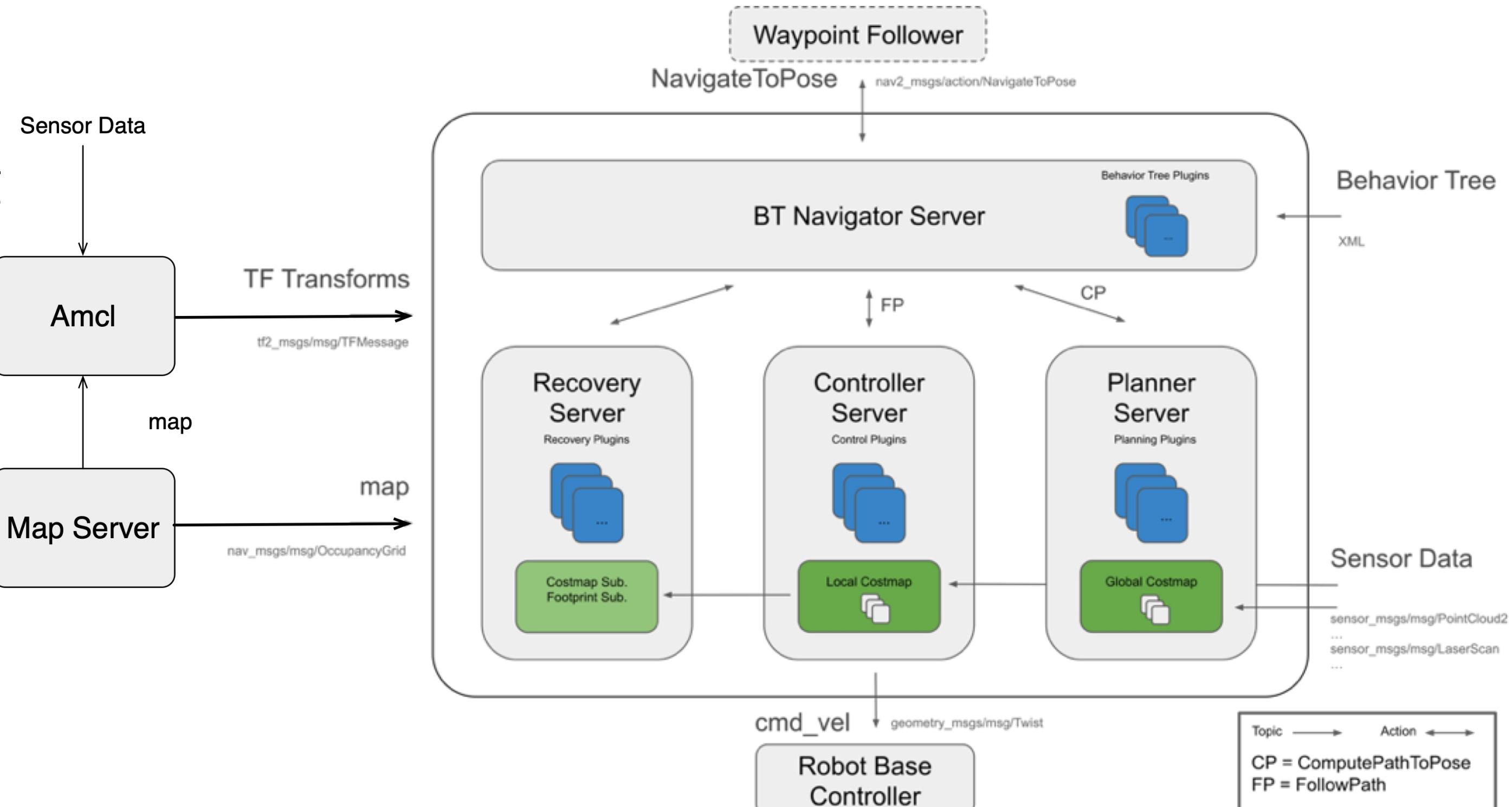
...
sensor_msgs/msg/LaserScan
...

cmd_vel

geometry_msgs/msg/Twist

Robot Base Controller

Topic — Action
CP = ComputePathToPose
FP = FollowPath



Introduction

ROS 2: Nav2

THE MARATHON 2

A NAVIGATION SYSTEM

STEVE MACENSKI, FRANCISCO MARTIN, RUFFIN WHITE,
JONATHAN GINES

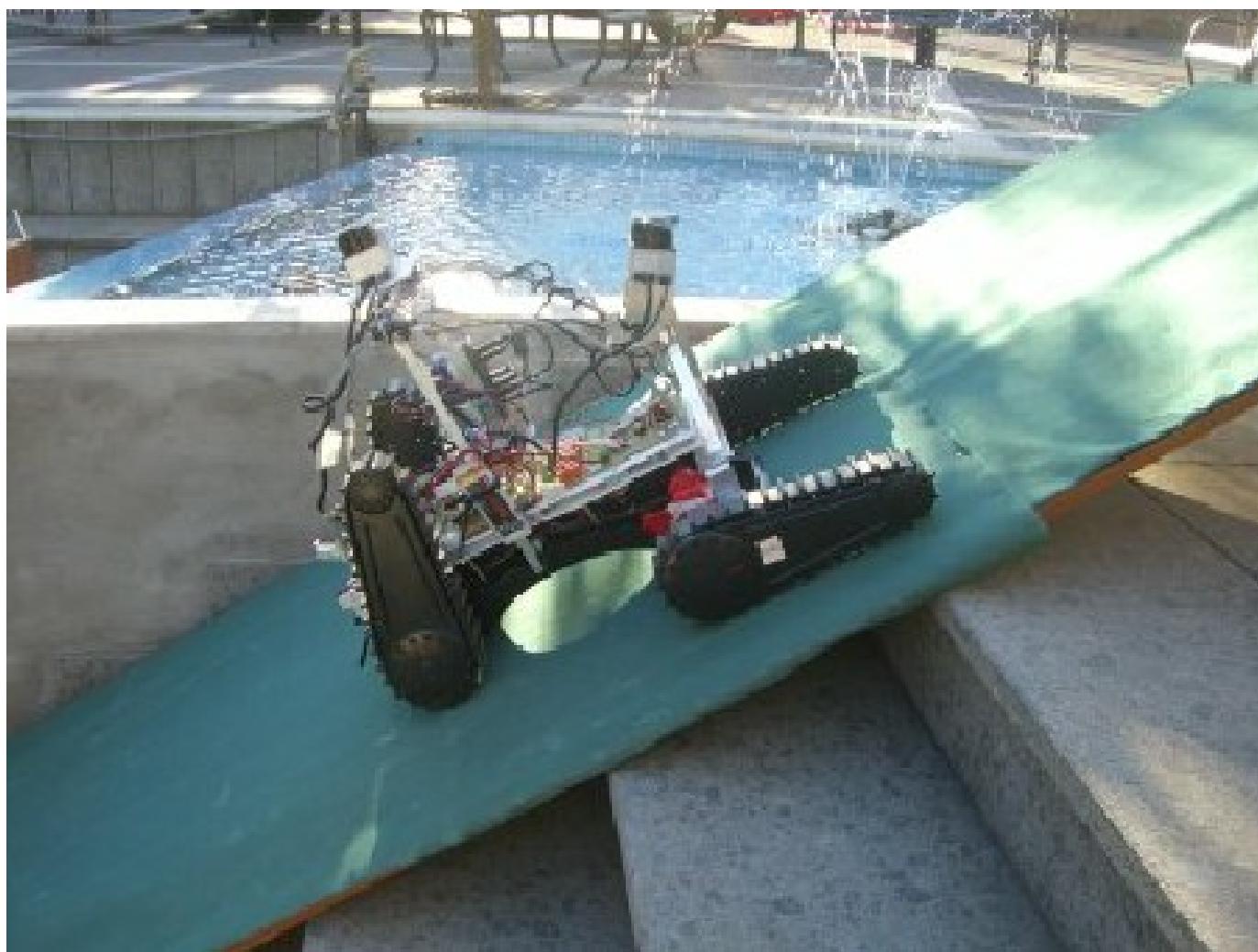
IROS 2020

[Link to video]

EasyNav
Beyond Nav2...

EasyNav

Motivation



EasyNav

¿Why?

- We work with some outdoor robots, and hacking Nav2 is not always an option.
- We wanted to offer multiple options to the community with the ability to adapt to different scenarios and restrictions.
- We believe in free and open source. Open source is about the freedom to choose. By providing alternatives, we strengthen the entire ROS 2 community.
- EasyNav does not aim to replace Nav2, which we admire and which served as an inspiration. Our goal is to offer flexibility and to satisfy specific requirements in areas where Nav2 would be less appropriate.

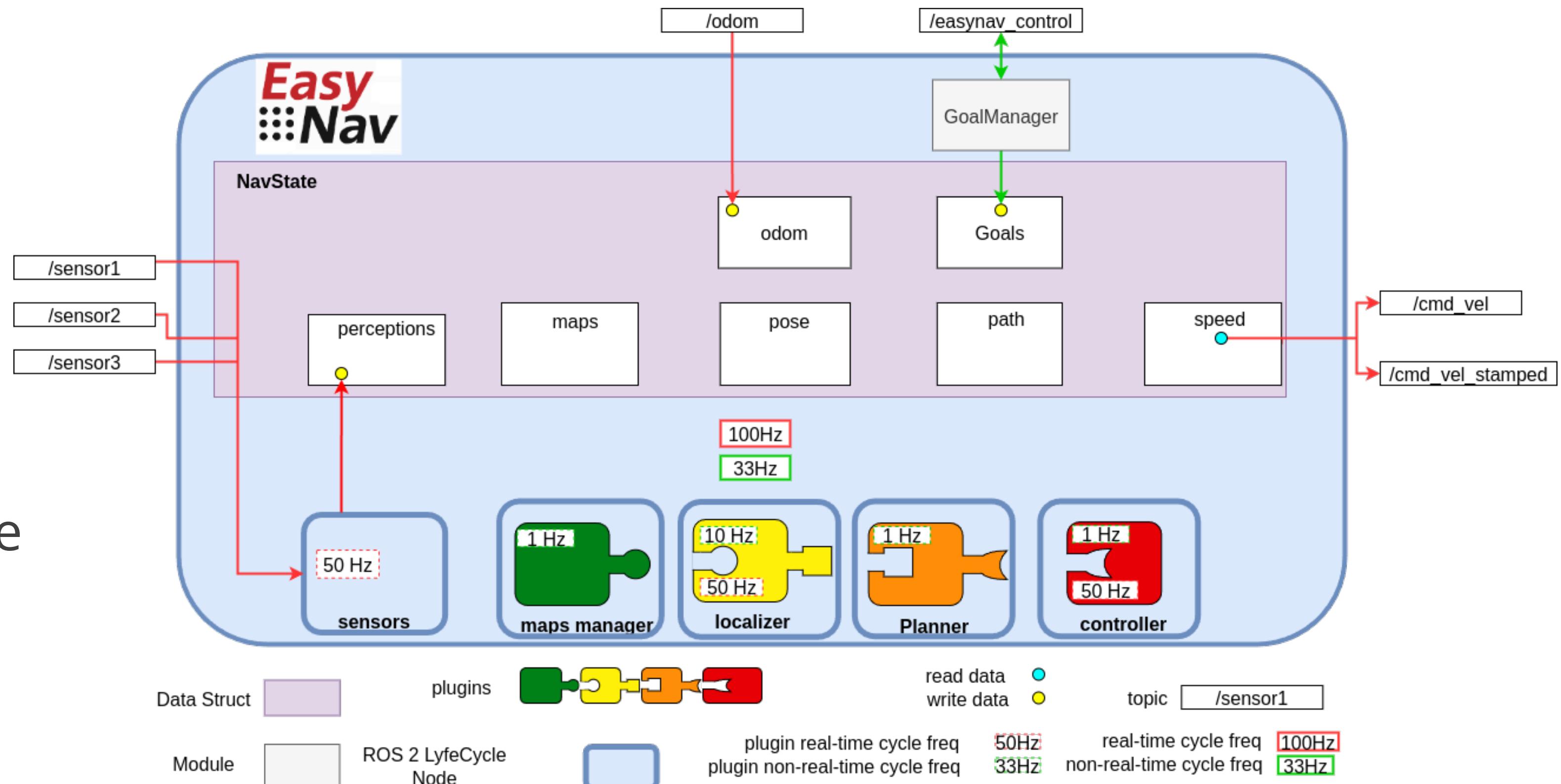


EasyNav
In action

Navigating with the **Costmap Stack**

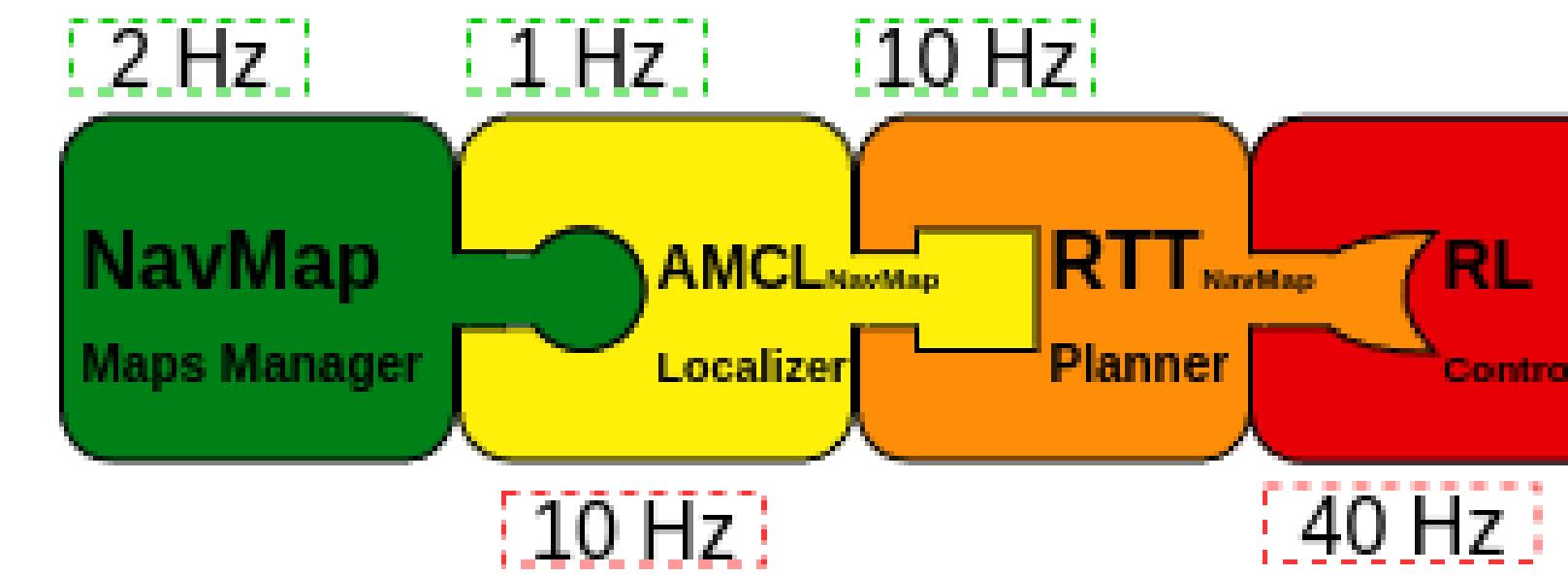
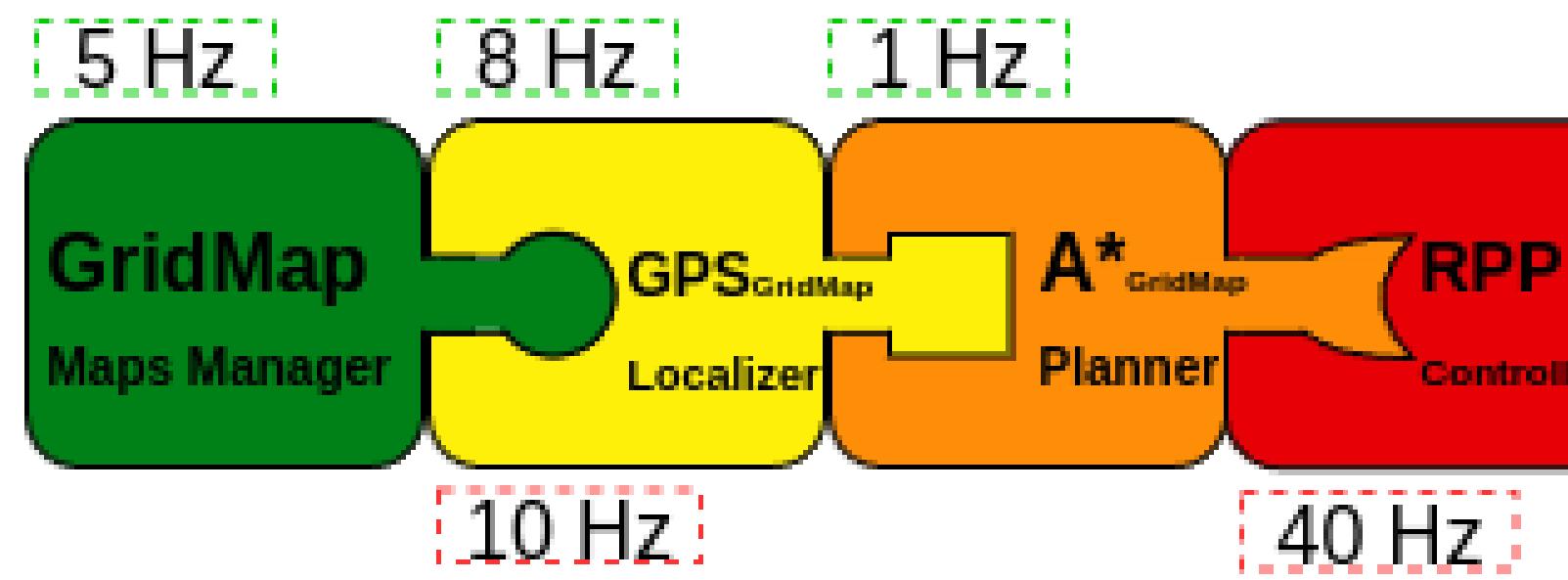
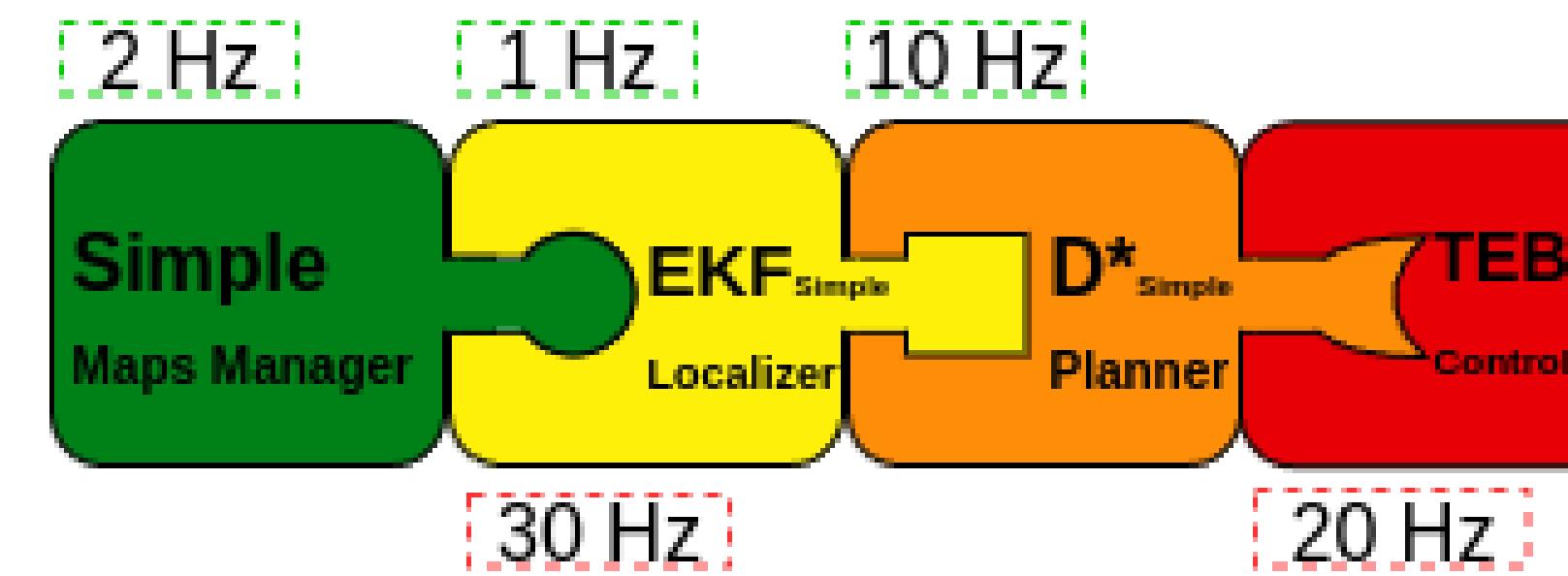
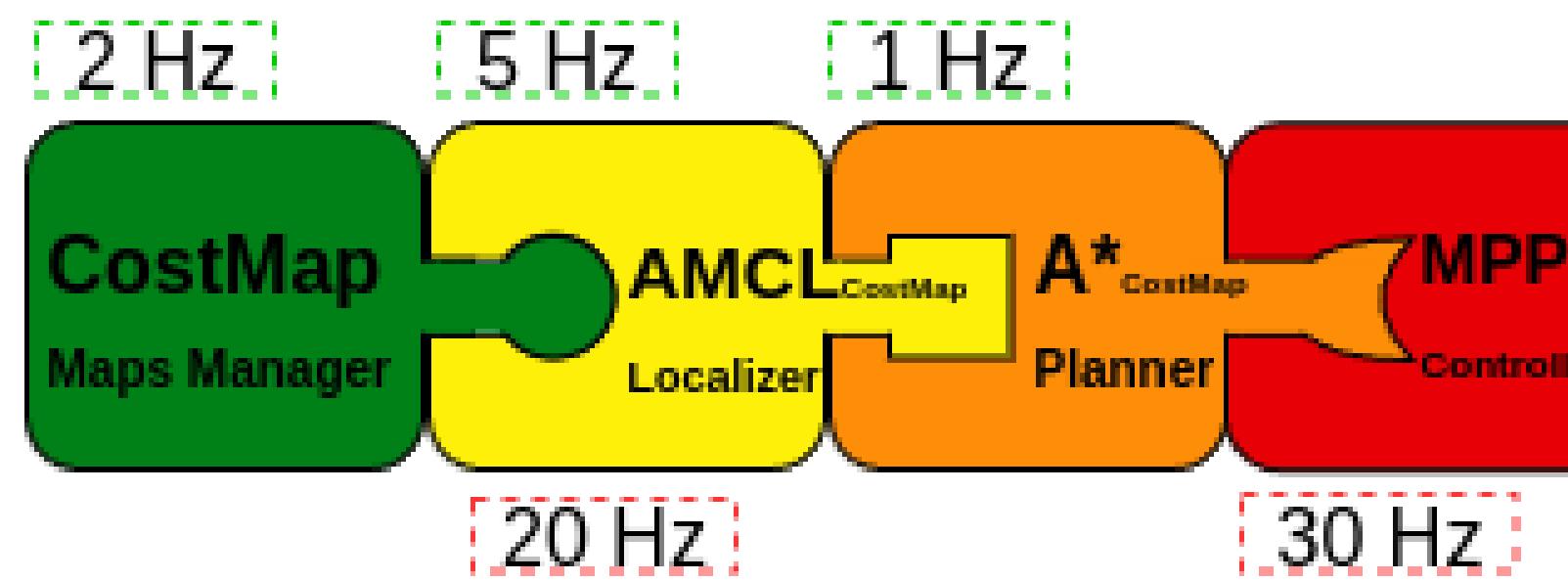
EasyNav Design

- Independent from representation
- Real-Time execution
- Light and easy to use:
 - Single binary executable
 - Single config file
- Modular, plugin-based architecture and reusable navigation stacks.



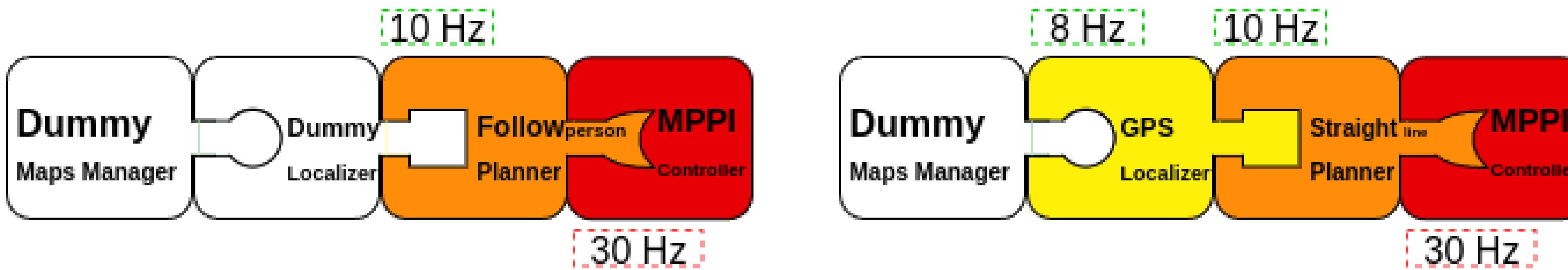
EasyNav

Plugin composition



EasyNav

Plugin composition



EasyNav

Plugin composition

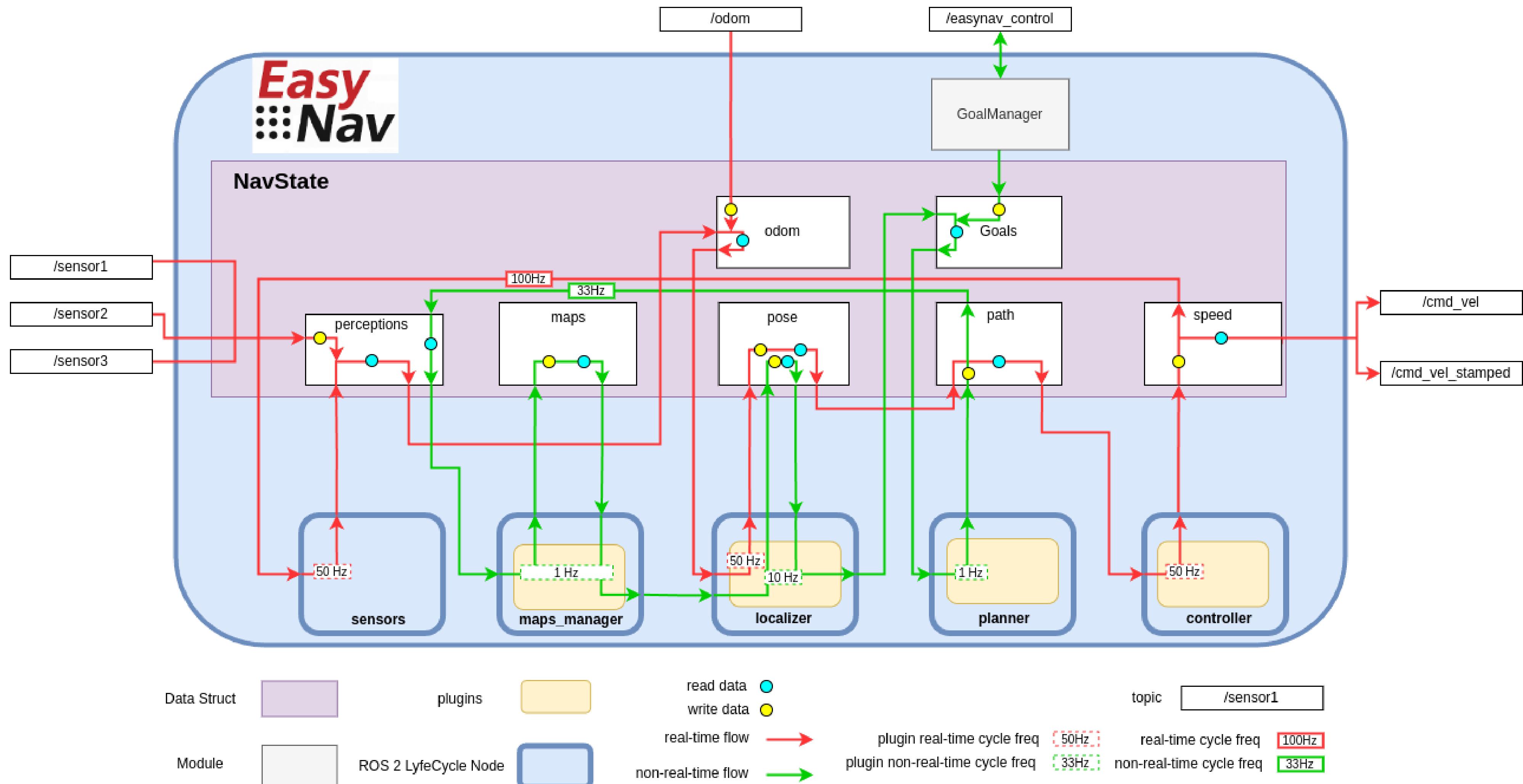
```
controller_node:  
  ros_parameters:  
    use_sim_time: true  
    controller_types: [simple]  
    simple:  
      rt_freq: 30.0  
      plugin: easynav_simple_controller/SimpleController  
      max_linear_speed: 0.6  
      max_angular_speed: 1.0  
      look_ahead_dist: 0.2  
      k_rot: 0.5  
  
localizer_node:  
  ros_parameters:  
    use_sim_time: true  
    localizer_types: [simple]  
    simple:  
      rt_freq: 50.0  
      freq: 5.0  
      reseed_freq: 1.0  
      plugin: easynav_simple_localizer/AMCLLocalizer  
      num_particles: 100  
      noise_translation: 0.05  
      noise_rotation: 0.1  
      noise_translation_to_rotation: 0.1  
    initial_pose:  
      x: 0.0  
      y: 0.0  
      yaw: 0.0  
      std_dev_xy: 0.1  
      std_dev_yaw: 0.01
```

```
maps_manager_node:  
  ros_parameters:  
    use_sim_time: true  
    map_types: [simple]  
    simple:  
      freq: 10.0  
      plugin: easynav_simple_maps_manager/SimpleMapsManager  
      package: easynav_indoor_testcase  
      map_path_file: maps/home.map  
  
planner_node:  
  ros_parameters:  
    use_sim_time: true  
    planner_types: [simple]  
    simple:  
      freq: 0.5  
      plugin: easynav_simple_planner/SimplePlanner  
      robot_radius: 0.3  
  
sensors_node:  
  ros_parameters:  
    use_sim_time: true  
    forget_time: 0.5  
    sensors: [laser1]  
    perception_default_frame: odom  
    laser1:  
      topic: /scan_raw  
      type: sensor_msgs/msg/LaserScan  
      group: points  
  
system_node:  
  ros_parameters:  
    use_sim_time: true  
    position_tolerance: 0.1  
    angle_tolerance: 0.05
```

EasyNav

Execution model

- Single Process
- 1 Thread RT
 - Sensor Update
 - Pose predict
 - Control
- 1 Thread no-RT
 - Maps Update
 - Pose correct
 - Path Planner
- 1 TF Buffer RT
 - Shared TF buffer



EasyNav

Composing map representations

- The MapsManager module supports multiple concurrent representations
- Different plugins may require / use different environment representations
- The representations may work together for a common task
- The plugins for the MapsManager implementations may also include layers or filters.

```
maps_manager_node:  
  ros_parameters:  
    use_sim_time: true  
    map_types: [bonxai, navmap]  
    bonxai:  
      freq: 10.0  
      plugin: easynav_bonxai_maps_manager/BonxaiMapsManager  
      package: easynav_indoor_testcase  
      bonxai_path_file: maps/excavation_urjc.pcd  
    navmap:  
      freq: 10.0  
      plugin: easynav_navmap_maps_manager/NavMapMapsManager  
      package: easynav_indoor_testcase  
      navmap_path_file: maps/excavation_urjc.navmap  
      filters: [obstacles, inflation]  
      obstacles:  
        plugin: easynav_navmap_maps_manager/NavMapMapsManager/ObstaclesFilter  
      inflation:  
        plugin: easynav_navmap_maps_manager/NavMapMapsManager/InflationFilter  
        inflation_radius: 5.0  
        cost_scaling_factor: 1.0
```

EasyNav
In action

Navigating with the **NavMap Stack**

EasyNav

Tools: TUI and CLI

The screenshot shows the EasyNav TUI interface running in a terminal window titled "fmrico@argo: ~/ros/ros2/easynav_ws". The interface is divided into several sections:

- Status Tab (Active):** Displays "Navigation Status" with "Navigation Control" details:
 - Type: FEEDBACK
 - Message: Current pose: pos=(2.953, 0.762, 0.000), quat=(0.000, 0.000, 0.948, 0.317)
 - Navigation time: 3.225s
 - ETA: 0.000s
 - Distance covered: 0.000 m
 - Distance to goal: 3.028 m
- Commanding Tab:** Displays "Goal Info" with:
 - Status: ACTIVE
 - Position: distance=3.021 m / tol=0.100 m
 - Angle: distance=0.061 rad / tol=0.050 rad
 - Goals remaining: 1
 - First goal: x=-0.044, y=1.198, z=0.000, yaw=2.402 rad
- Twist Tab:** Displays "Twist" information:
 - linear : x=0.186, y=0.000, z=0.000
 - angular: x=0.000, y=0.000, z=0.876
- NavState Tab:** Displays navigation state details:

```
cmd_vel = [0x771c1002d340] : Twist with (0.178672, 0, 0) (0, 0, 0.883066)
path = [0x5609b6076a10] : Path with 16 poses and length3.24853 m.
goals = [0x5609b6055300] : Goals 1 with :
    --> (-0.044216, 1.198157)

robot_pose = [0x771be4010620] : Odometry with pose: (x: 2.93773, y: 0.751773, yaw: 2.49647)
map.static = [0x5609b6073b90] : SimpleMap of (383 x 231) with resolution 0.050000
points = [0x771be4015b50] : PointPerception 1 with:
    [0x5609b6009a40] --> 360 points in frame with ts 158.8

map.dynamic = [0x5609b6073c10] : SimpleMap of (383 x 231) with resolution 0.050000
navigation_state = [0x5609b5fb7580] : State ACTIVE
```
- Time stats Tab:** Displays execution times for various functions:

function name	execution time (ms)	elapsed (ms)	frequency (Hz)
LocalizerMethodBase::internal_update	1.699 ± 0.369	235.942 ± 18.254	4.26 ± 0.28
MapsManagerBase::internal_update	1.179 ± 0.276	134.818 ± 12.644	7.47 ± 0.54
PlannerMethodBase::internal_update	0.353 ± 0.672	2312.212 ± 363.949	0.44 ± 0.04
ControllerMethodBase::internal_update_rt	0.026 ± 0.010	19.789 ± 9.173	62.77 ± 29.25
LocalizerMethodBase::internal_update_rt	0.515 ± 0.087	28.244 ± 7.568	39.54 ± 18.61
SystemNode::system_cycle	0.694 ± 0.823	33.704 ± 5.848	30.15 ± 3.32
SystemNode::system_cycle_rt	0.298 ± 0.287	10.000 ± 0.091	100.01 ± 0.91
SimpleMapsManager::update	1.165 ± 0.268	134.818 ± 12.644	7.47 ± 0.54

At the bottom of the interface, there are tabs for "Status" (active), "Commanding", and "Twisting". The status bar at the bottom shows "q Salir 1 Tab Status 2 Tab Commanding" and "p palette".

EasyNav

Tools: TUI and CLI

```
ros2 easynav -h  
ros2 easynav plugins -h  
ros2 easynav nav-state -h  
# ... etc.
```

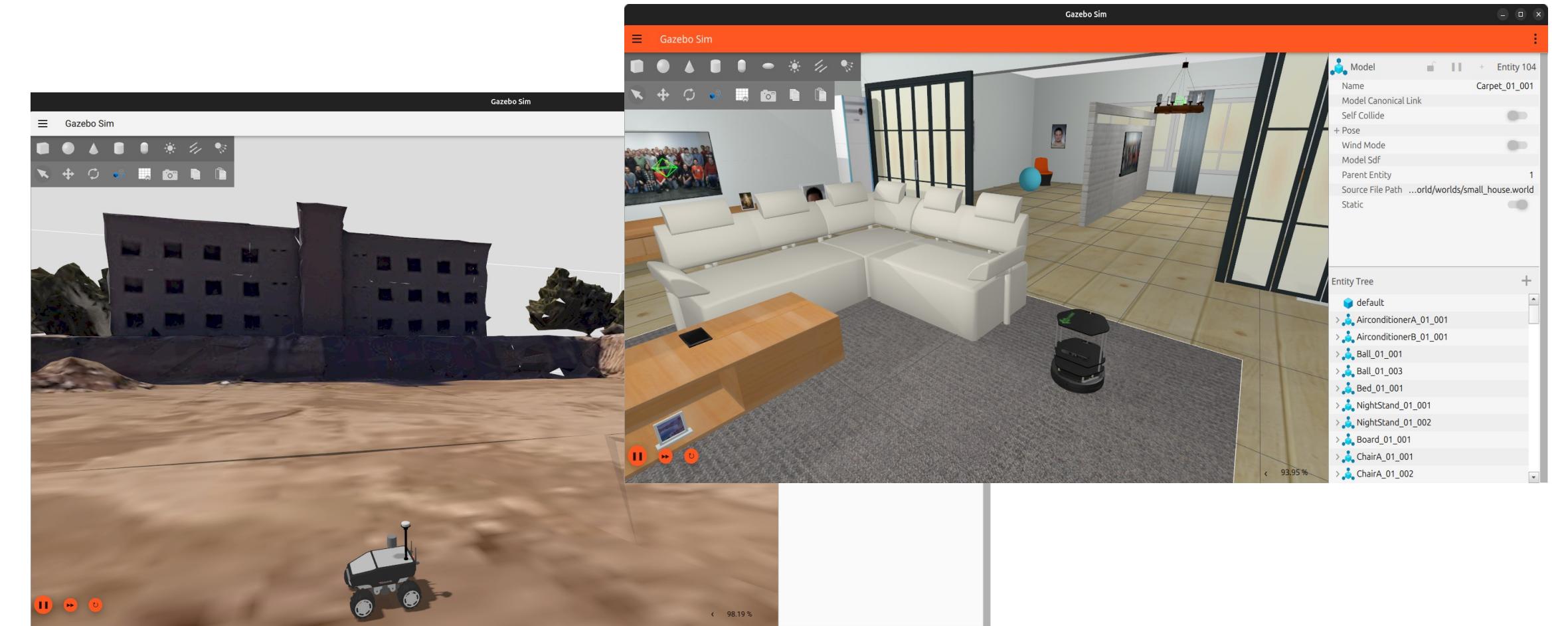
```
ros2 easynav plugins [--mapsmanager] [--localizer] [--planner] [--controller]  
[--costmap-filters] [--navmap-filters]  
[--grep SUBSTR] [--show-lib] [--show-xml]  
[--json] [--pretty] [--debug]
```

```
# Quick inventory of installed EasyNav plugins  
ros2 easynav plugins  
  
# Inspect planner plugins and search for "astar"  
ros2 easynav plugins --planner --grep astar  
  
# Monitor NavState for 45 seconds  
ros2 easynav nav-state --duration 45  
  
# Watch goal manager events while sending goals from RViz  
ros2 easynav goal-info --duration 60  
  
# Observe control loop values  
ros2 easynav navigation-control --duration 30  
  
# Tail robot velocities  
ros2 easynav twist --duration 10  
  
# Render periodic time-stats with screen refresh  
ros2 easynav timetats --duration 20
```

EasyNav

Repositories

- Code hosted on github.com/EasyNavigation
- GPLv3 license
- CI pipelines for multiple distros
- Binary releases for ROS 2 Jazzy and Kilted
- Simulation playgrounds in Gazebo with different worlds and robots



Repository	Doxxygen	Rolling	Kilted	Jazzy	Humble
EasyNavigation	Doxygen Deployment passing	rolling passing	kilted passing	jazzy passing	humble passing
NavMap	Doxygen Deployment passing	rolling passing	kilted passing	jazzy passing	humble passing
easynav_plugins	Doxygen Deployment passing	rolling passing	kilted passing	jazzy passing	humble passing
yaets	Doxygen Deployment passing	rolling passing	kilted passing	jazzy passing	humble passing

EasyNav

Documentation

The screenshot shows the 'HowTos and Practical Guides' section of the EasyNav documentation. At the top, there's a navigation bar with the EasyNav logo, a search bar, and a 'latest' button. Below the navigation bar, the page title is 'HowTos and Practical Guides'. A sub-section title 'On this page' is followed by a list of categories: Overview, Simple Stack, Costmap Stack (with sub-items: Mapping with the Costmap Stack, Mapping with SLAM Toolbox and EasyNav, Deploying EasyNav on a Real iCreate3 Robot), GridMap Stack, Bonxai Stack, NavMap Stack, Controllers, Behaviors, and General. A sidebar on the left contains links for Build & Install, Getting Started, EasyNav Plugins, HowTos and Practical Guides (which is expanded to show the current category), Developers Guide, and About and Contact.

The screenshot shows the 'GoalManagerClient Class Reference' page from the Easy Navigation documentation. The header includes the Easy Navigation logo, a search bar, and navigation links for Main Page, Packages, Classes, and Files. The page title is 'GoalManagerClient Class Reference'. It describes the class as a client-side interface for interacting with GoalManager. Below the description is a code snippet for the class definition:

```
#include <GoalManagerClient.hpp>
```

The page then lists 'Public Types' and 'Public Member Functions' with their descriptions. The 'Public Types' section shows an enum class 'State' with values: IDLE, SENT_GOAL, SENT_PREEPMT, ACCEPTED_AND_NAVIGATING, NAVIGATION_FINISHED, NAVIGATION_REJECTED, NAVIGATION_FAILED, NAVIGATION_CANCELLED, and ERROR. The 'Public Member Functions' section lists methods like cancel(), get_feedback(), get_last_control(), get_result(), get_state(), and send_goal().

<https://easynavigation.github.io>

EasyNav

Documentation

EasyNavigation

Planners

Path-planning plugins implementing A*, costmap-based, and NavMap-based methods.

Package	Description	Documentation
easynav_costmap_planner	A* planner over <code>Costmap2D</code> .	easynav_costmap_planner README
easynav_simple_planner	Simple A* planner for <code>SimpleMap</code> .	easynav_simple_planner README
easynav_navmap_planner	A* planner operating on a NavMap mesh.	easynav_navmap_planner README

Controllers

Motion controllers for trajectory tracking and reactive behaviors.

Package	Description Documentation	
easynav_vff_controller	Vector Field Force (VFF) reactive controller.	easynav_vff_controller README
easynav_mppi_controller	Model Predictive Path Integral (MPPI) controller.	easynav_mppi_controller README
easynav_simple_controller	Simple proportional controller for testing.	easynav_simple_controller README
easynav_serest_controller	SeReST (Safe Reactive Steering) controller.	easynav_serest_controller README

Maps Managers

Map-management plugins that provide, update, and store different environment representations.

Package	Description	Documentation
easynav_navmap_maps_manager	Manages NavMap mesh layers.	easynav_navmap_maps_manager README
easynav_bonxai_maps_manager	Manages Bonxai probabilistic voxel maps.	easynav_bonxai_maps_manager README
easynav_octomap_maps_manager	Manages OctoMap 3D occupancy trees.	easynav_octomap_maps_manager README
easynav_costmap_maps_manager	Manages <code>Costmap2D</code> layers with filters.	easynav_costmap_maps_manager README
easynav_simple_maps_manager	Minimal example map manager for <code>SimpleMap</code> .	easynav_simple_maps_manager README

Localizers

Localization plugins based on different map types and sensors.

Package	Description	Documentation
easynav_gps_localizer	GPS-based localizer for outdoor navigation.	easynav_gps_localizer README
easynav_simple_localizer	Basic localizer for <code>SimpleMap</code> -based setups.	easynav_simple_localizer README
easynav_navmap_localizer	AMCL-like localizer operating on NavMap meshes.	easynav_navmap_localizer README
easynav_costmap_localizer	AMCL-like localizer using <code>Costmap2D</code> .	easynav_costmap_localizer README

easynav_navmap_planner

ROS 2 kilted ROS 2 rolling

Description

A* path planner over a NavMap triangular surface/layer. Consumes NavMap and goals from NavState and publishes a `nav_msgs/Path`.

Authors and Maintainers

- Authors: Intelligent Robotics Lab
- Maintainers: Francisco Martín Rico fmrico@gmail.com

Supported ROS 2 Distributions

Distribution	Status
kilted	<small>Killed</small> <small>Supported</small>
rolling	<small>Rolling</small> <small>Supported</small>

Plugin (pluginlib)

- Plugin Name: `easynav_navmap_planner/AStarPlanner`
- Type: `easynav::navmap::AStarPlanner`
- Base Class: `easynav::PlannerMethodBase`
- Library: `easynav::navmap::planner`
- Description: A* path planner over a NavMap triangular surface/layer. Consumes NavMap and goals from NavState and publishes a `nav_msgs/Path`.

Parameters

All parameters are declared under the plugin namespace, i.e., `/<node_fqn>/easynav_navmap_planner/AStarPlanner/...`

Name	Type	Default	Description
<code><plugin>.layer</code>	string	<code>"inflated_obstacles"</code>	NavMap layer name to read costs from (e.g., <code>inflated_obstacles</code>).
<code><plugin>.cost_factor</code>	double	2.0	Scaling factor applied to cell/triangle costs.
<code><plugin>.inflation_penalty</code>	double	5.0	Extra penalty near inflated/inscribed regions to keep paths away from obstacles.
<code><plugin>.continuous_replan</code>	bool	true	Replan continuously as NavState updates (true) or plan once per request (false).

Interfaces (Topics and Services)

Publications

Direction	Topic	Type	Purpose	QoS
Publisher	<code><node_fqn>/<plugin>/path</code>	<code>nav_msgs/msg/Path</code>	Publishes the computed A* path.	<code>depth=10</code>

This plugin does not create subscriptions or services directly; it reads inputs from `NavState`.

NavState Keys

Key	Type	Access	Notes
<code>goals</code>	<code>nav_msgs::msg::Goals</code>	Read	Planner targets.
<code>map</code>	<code>navmap::NavMap</code>	Read	NavMap (reads the specified layer).
<code>robot_pose</code>	<code>nav_msgs::msg::Odometry</code>	Read	Start pose for path planning.
<code>path</code>	<code>nav_msgs::msg::Path</code>	Write	Output path to follow.

TF Frames

This plugin does not perform TF lookups directly; frame consistency is assumed between NavMap, robot pose, and published path.

License

GPL-3.0-only

Conclusions

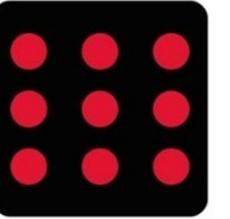
Where are we?

Where are we going?

Conclusions

- We have developed a navigation system which is robust, easy to use and well designed
- It is light and deterministic
- It adapts to problems in which using Nav2 requires hacks
- Many plugins and navigation stacks already operative: Costmaps, gridmaps, Bonxai, NavMap...
- We want to build community around it
- Still lacking robust and effective controllers

Easy
:::
Nav



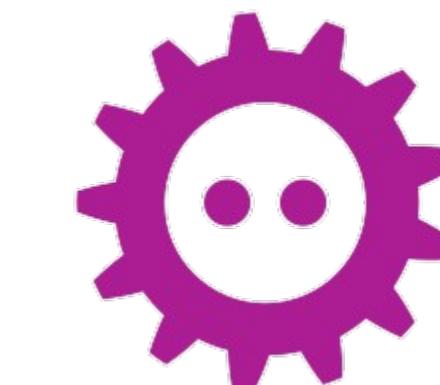
Easy Nav

An open-source framework
for navigating everywhere

¿Questions?

Prof. Dr. Francisco Martín Rico
francisco.rico@urjc.es

Dr. Francisco Miguel Moreno
franciscom.moreno@urjc.es



FOSDEM