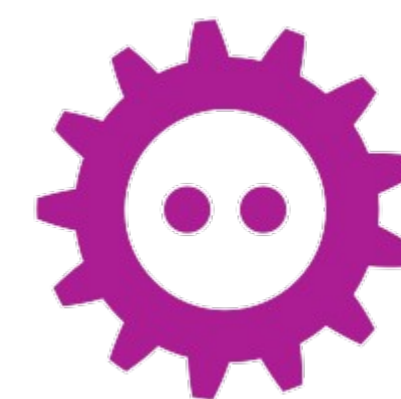


Easy Nav

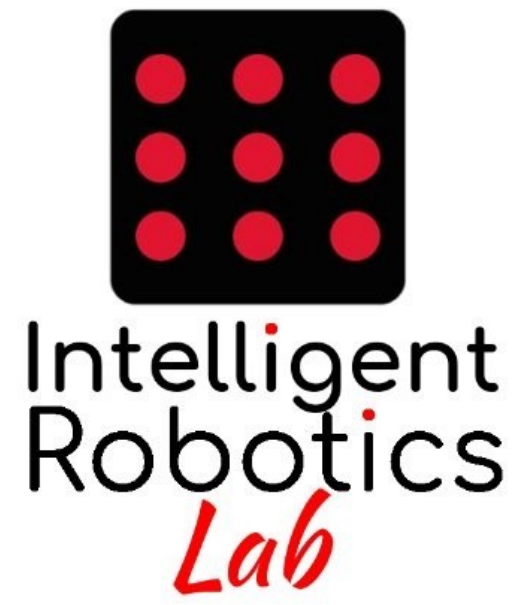
An open-source framework
for navigating everywhere

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

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



FOSDEM



Francisco Martín Rico
fmrico

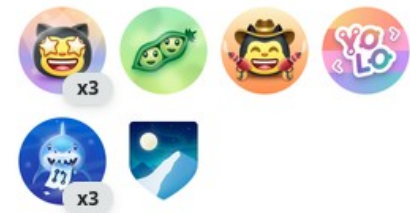
Follow

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.gsys.es/~fmartin
@fmrico

Achievements



Overview Repositories 161 Projects Packages Stars 22

Pinned

PlanSys2/ros2_planning_system Public
This repo contains a PDDL-based planning system for ROS2.
C++ 462 106

cascade_lifecycle Public
Managed nodes (or lifecycle nodes, LN) package provides a mechanism to create LifecycleNode activation trees
C++ 82 13

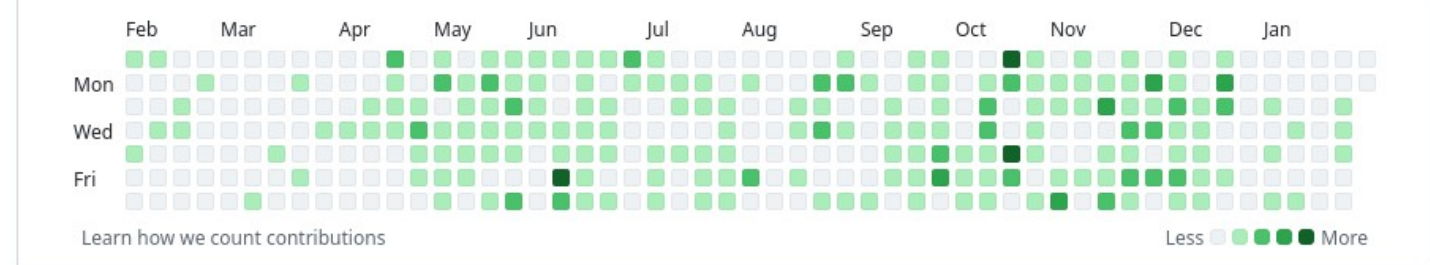
rclicpp Public
Forked from ros2/rclicpp
rclicpp (ROS Client Library for C++)
C++

ros2_dotnet Public
Forked from ros2-dotnet/ros2_dotnet
.NET bindings for ROS2
C# 1

navigation2 Public
Forked from ros-navigation/navigation2
ROS2 Navigation
C++ 1 1

popf Public
The POPF planner from KCL planning group with some modifications to make it work with "modern" compilers...
C++ 22 8

1,446 contributions in the last year



Contribution activity

January 2026

2026

2025



Francisco Miguel Moreno
Butakus

Follow

15 followers · 4 following

Spain

Achievements



Highlights

PRO

Organizations



Block or Report

Overview Repositories 36 Projects Packages Stars

Pinned

ros2-env Public
A zsh plugin to manage ROS 2 workspaces
Shell 16 3

EasyNavigation Public
Forked from EasyNavigation/EasyNavigation
C++

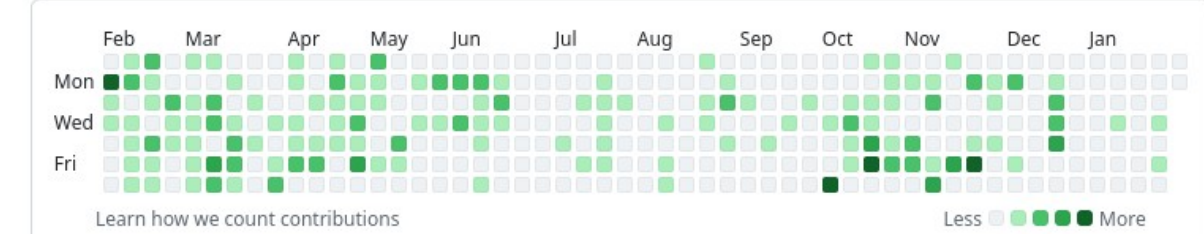
landmark_placement_optimization Public
Python 1

pikaball-revamped Public
A Revamped version of the Pikachu Volleyball game - Murcia LAN Party Edition
C++ 2 1

ArduComm Public
A data link layer protocol and library for advanced serial communication between a host computer and 8-bit Arduino boards.
Python 1

Isi-uc3m/hypergrid Public
Hypergrid is a ROS package for building local maps from multiple sensors in the blink of an eye
C++ 19 8

405 contributions in the last year



@EasyNavigation @ros @IntelligentRoboticsLa... More

Activity overview

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

4% Code review

2026

2025

2024

2023

2022

2021

2020

2019

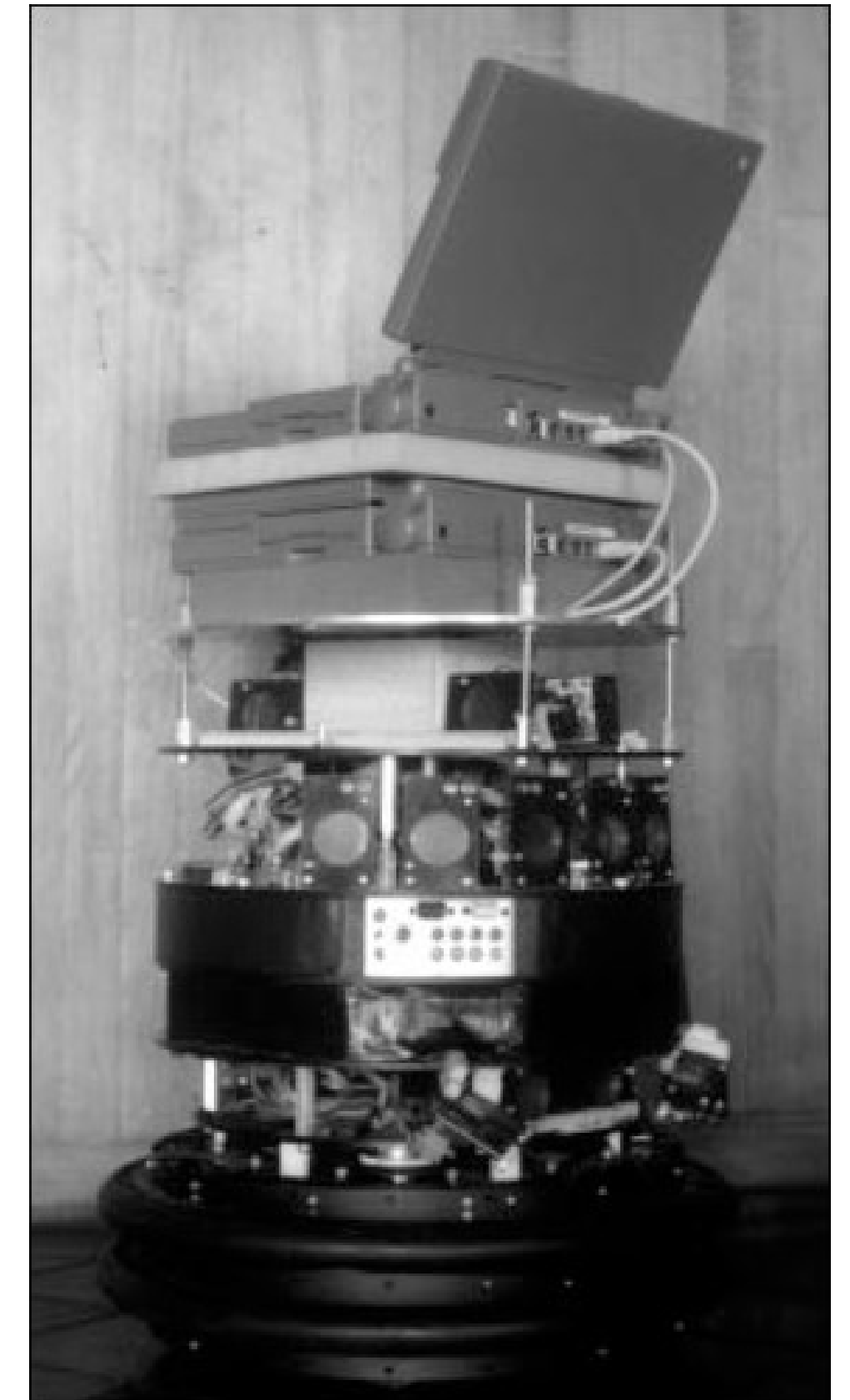
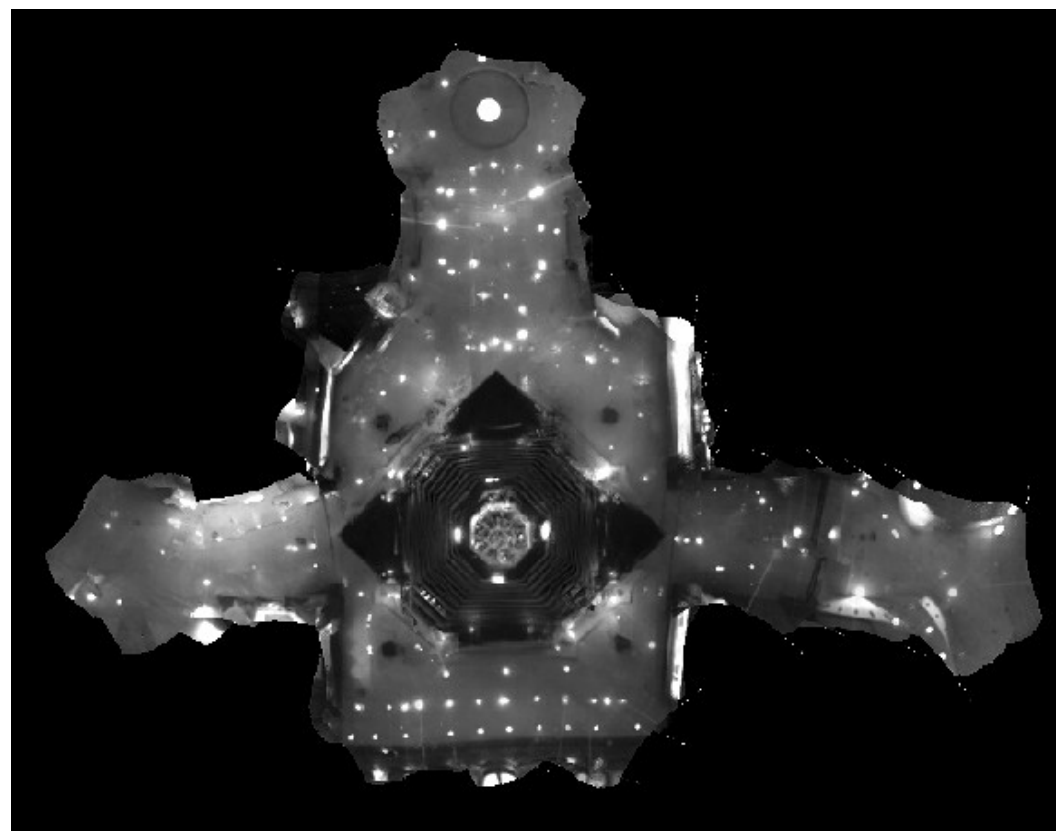
Introduction

Some background...

Introduction

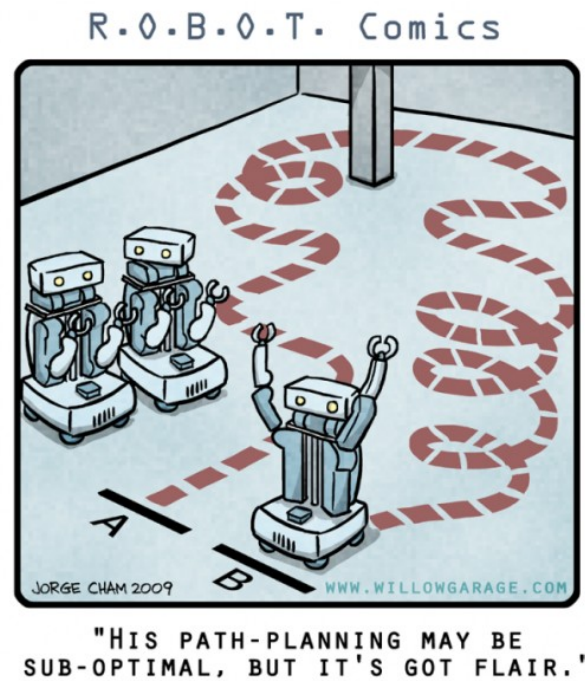
Robot navigation

- 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 range-finder 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 reimplementation
- An example of good practices in ROS 2
- More algorithms, plugins, components...
- Contributions: 59 collaborators, Intel, Samsung

Navigation 2

ROS

latest

Search docs

- 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

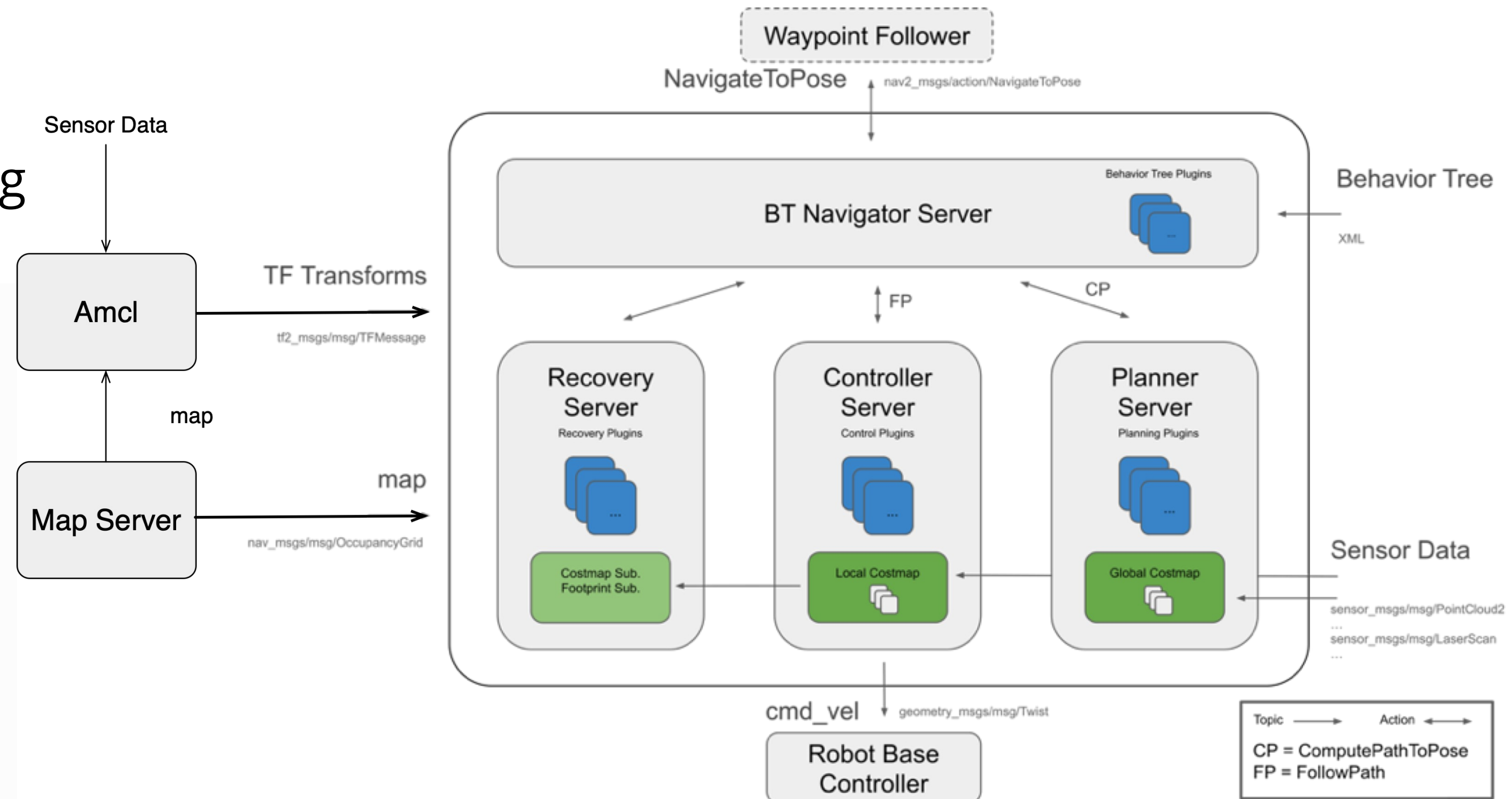
ROS 2 Navigation

Overview

The Navigation 2 project is the spiritual successor of the ROS Navigation Stack. This project seeks to find a safe way to have a mobile robot move from point A to point B. This will complete dynamic path planning, compute velocities for motors, avoid obstacles, and structure recovery behaviors. To learn more about this project see [About and Contact](#).

Navigation 2 uses behavior trees to call modular servers to complete an action. An action can be to compute a path, control effort, recovery, or any other navigation related action. These are each separate nodes that communicate with the behavior tree (BT) over a ROS action server. The diagram below will give you a good first-look at the structure of Navigation 2. Note: It is possible to have multiple plugins for controllers, planners, and recoveries in each of their servers with matching BT plugins. This can be used to create contextual navigation behaviors. If you would like to see a comparison between this project and ROS (1) Navigation, see [ROS to ROS 2 Navigation](#).

The expected inputs to Navigation2 (Nav2) are TF transformations conforming to REP-105, a map source if utilizing the Static Costmap Layer, a BT XML file, and any relevant sensor data sources. It will then provide valid velocity commands for the motors of a holonomic or non-holonomic robot to follow. We currently support holonomic and differential-drive base types but plan to support Ackermann (car-like) robots as well in the near future.



Introduction

ROS 2: Nav2

THE MARATHON 2

A NAVIGATION SYSTEM

STEVE MACENSKI, FRANCISCO MARTIN, RUFFIN WHITE,
JONATHAN GINES

IROS 2020

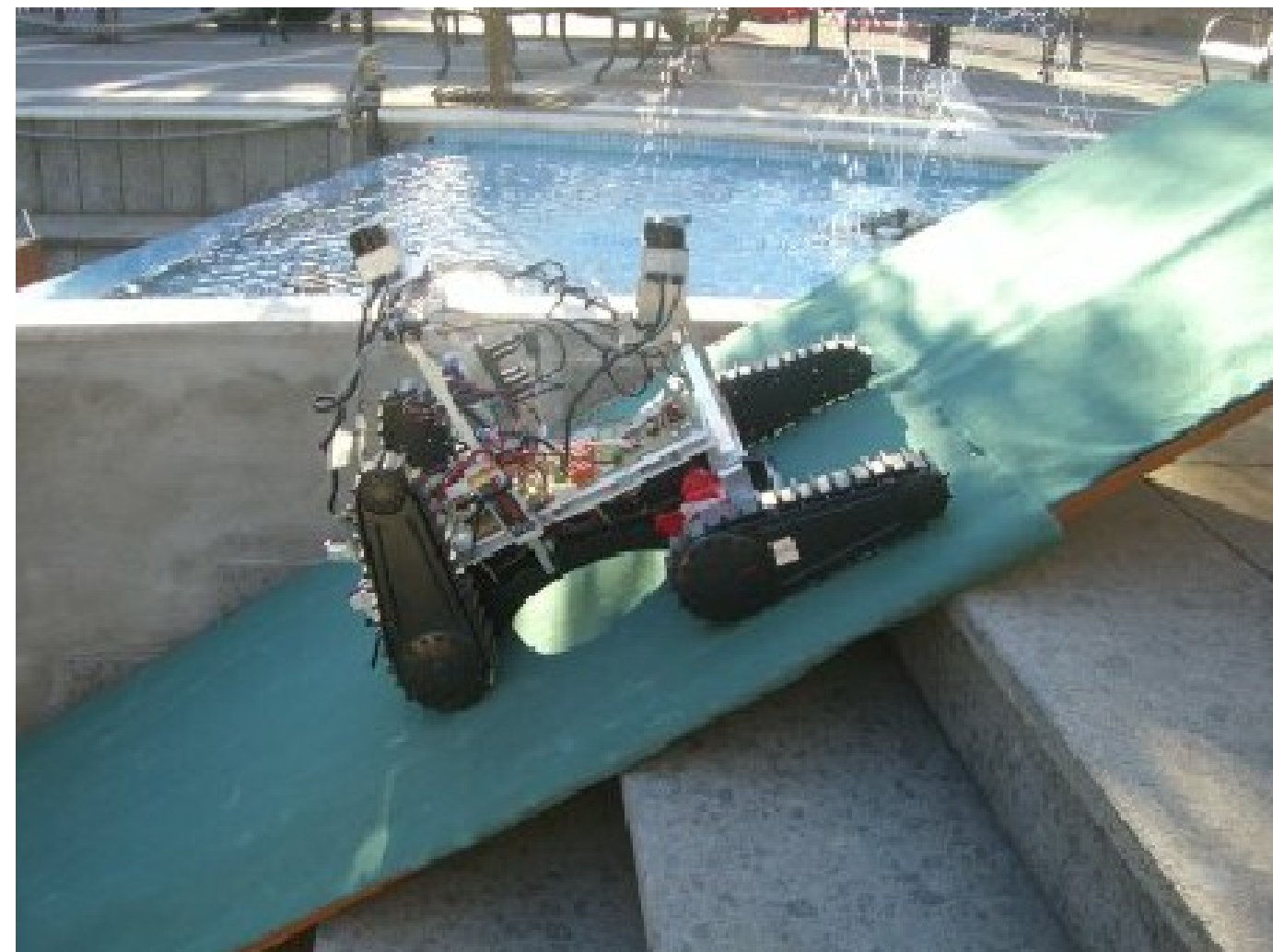
[[Link to video](#)]

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.

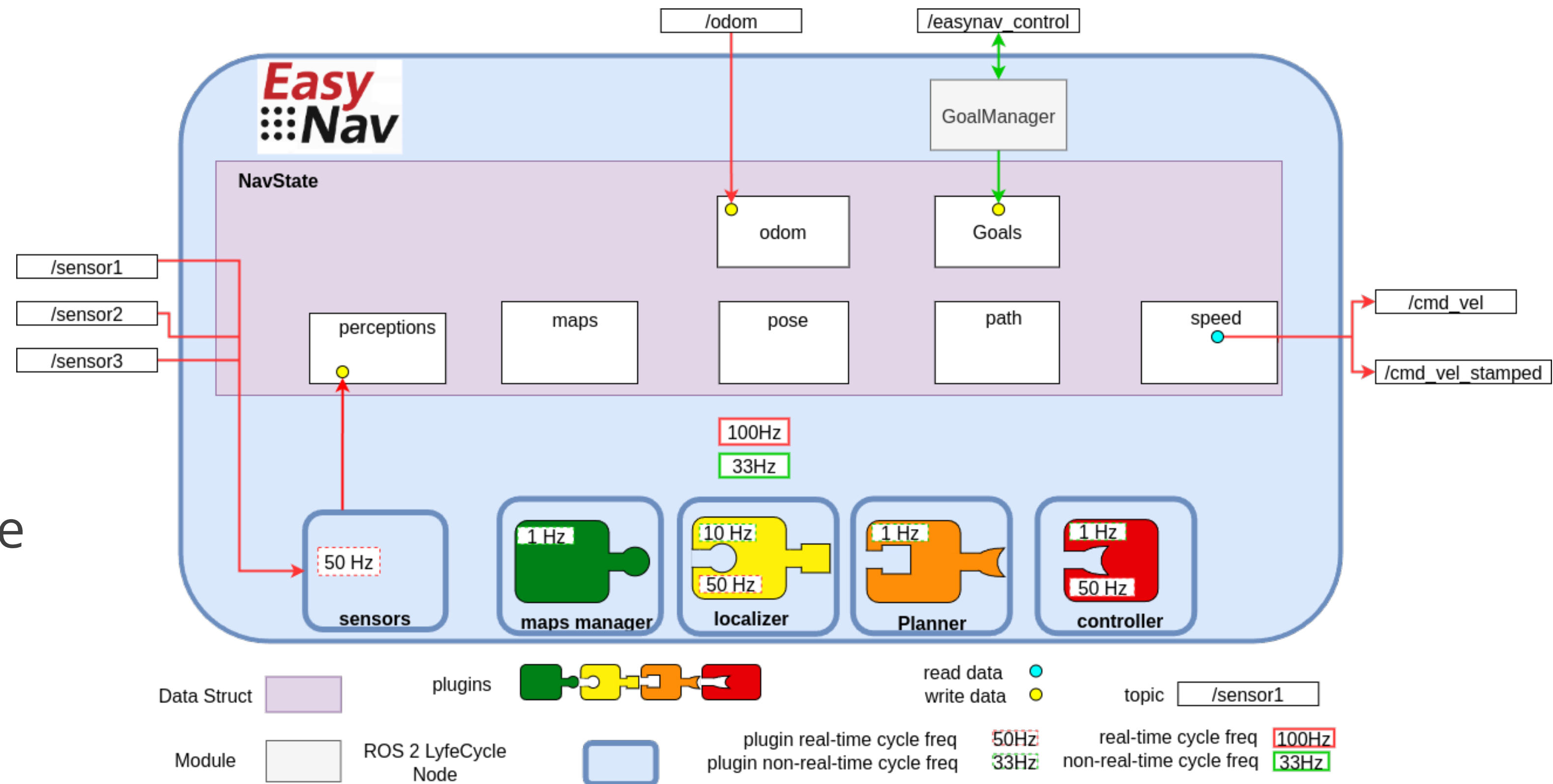


EasyNav
In action

Navigating with the
Costmap Stack

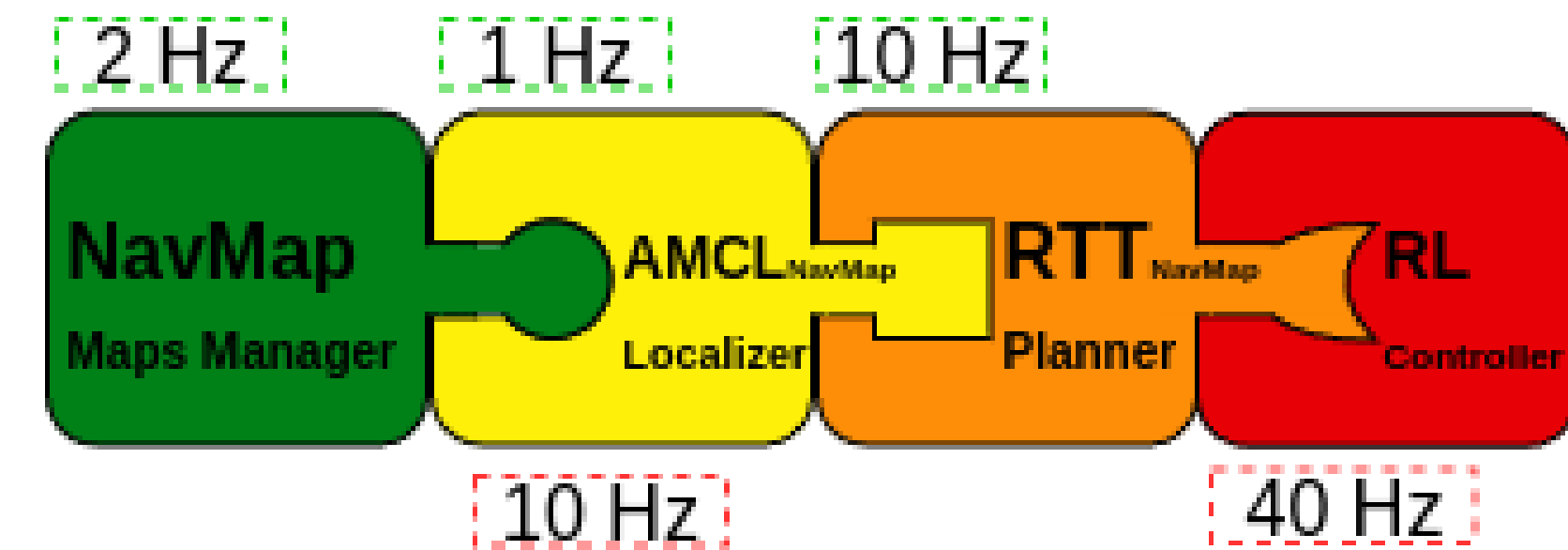
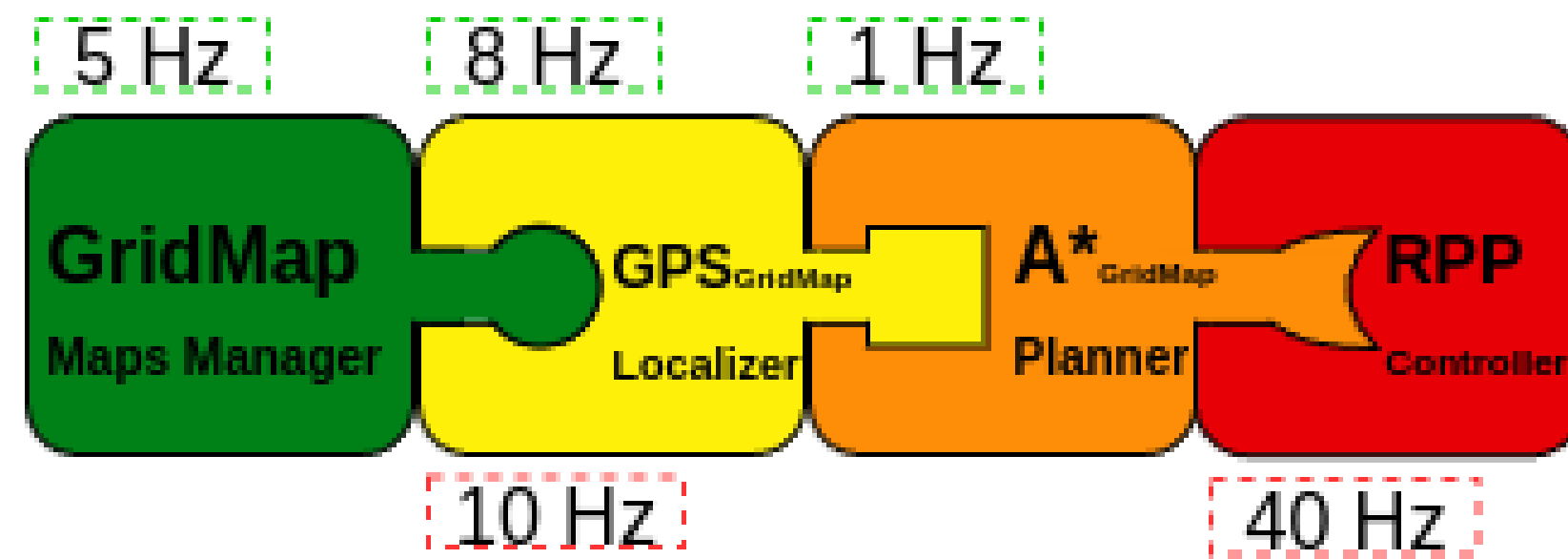
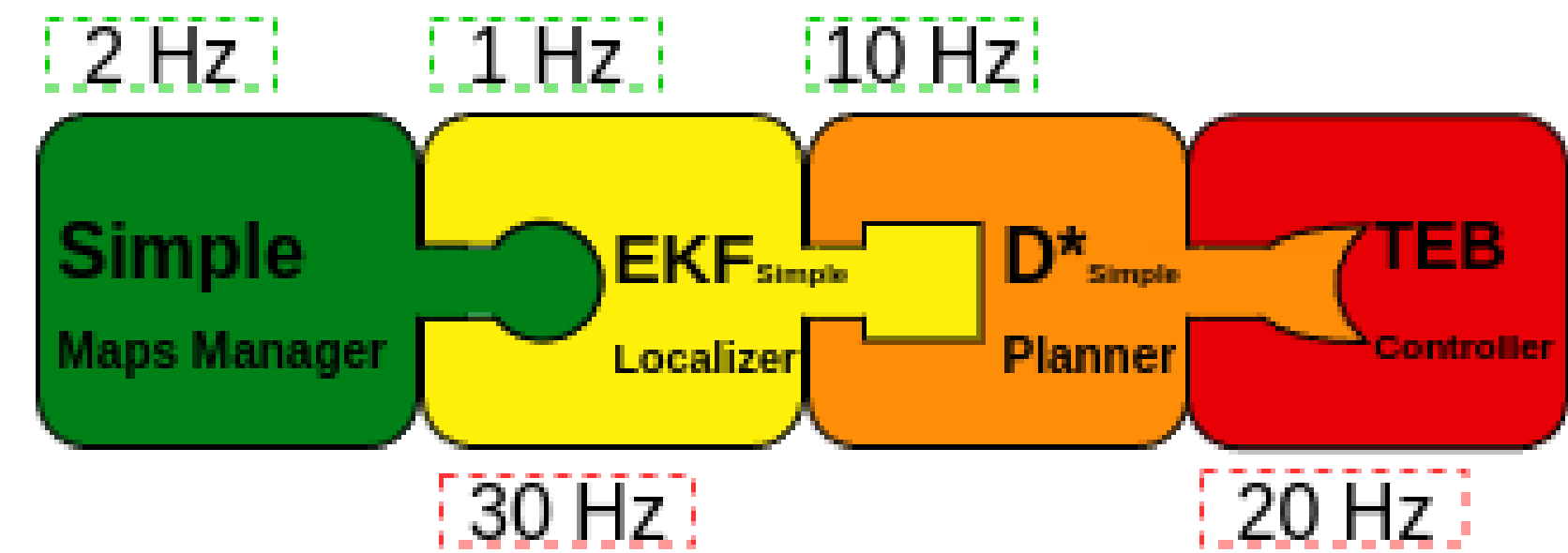
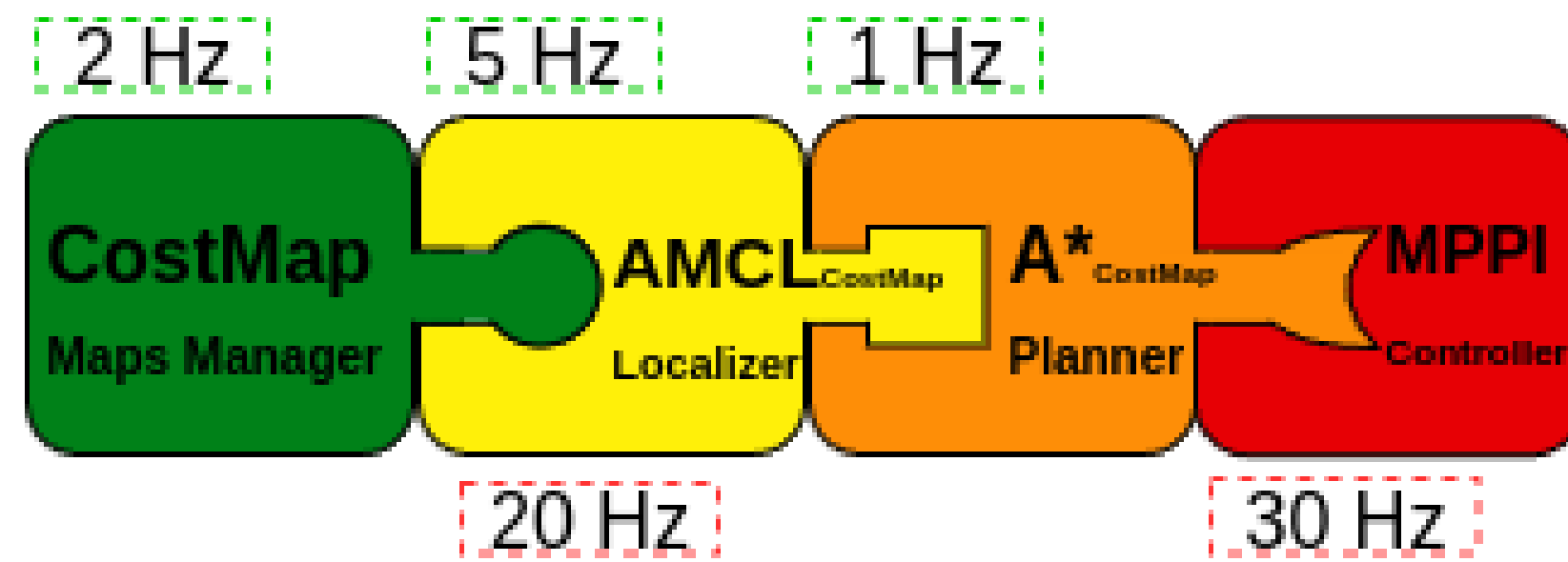
EasyNav Design

- 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.



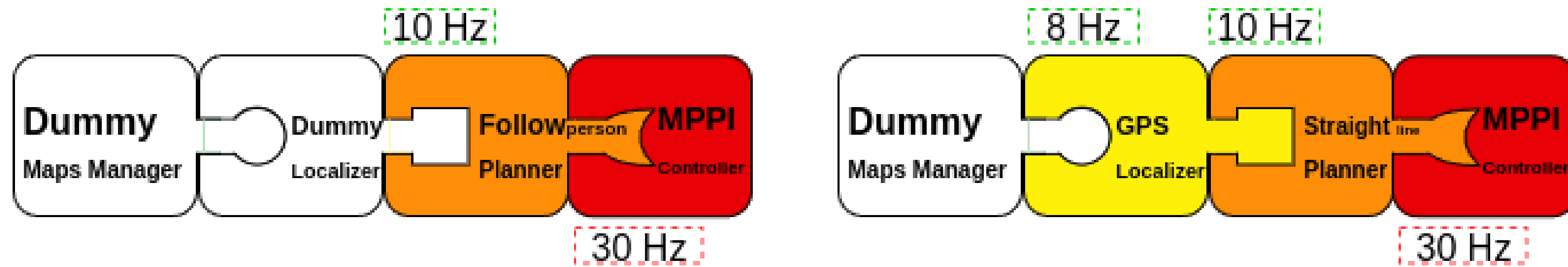
EasyNav

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

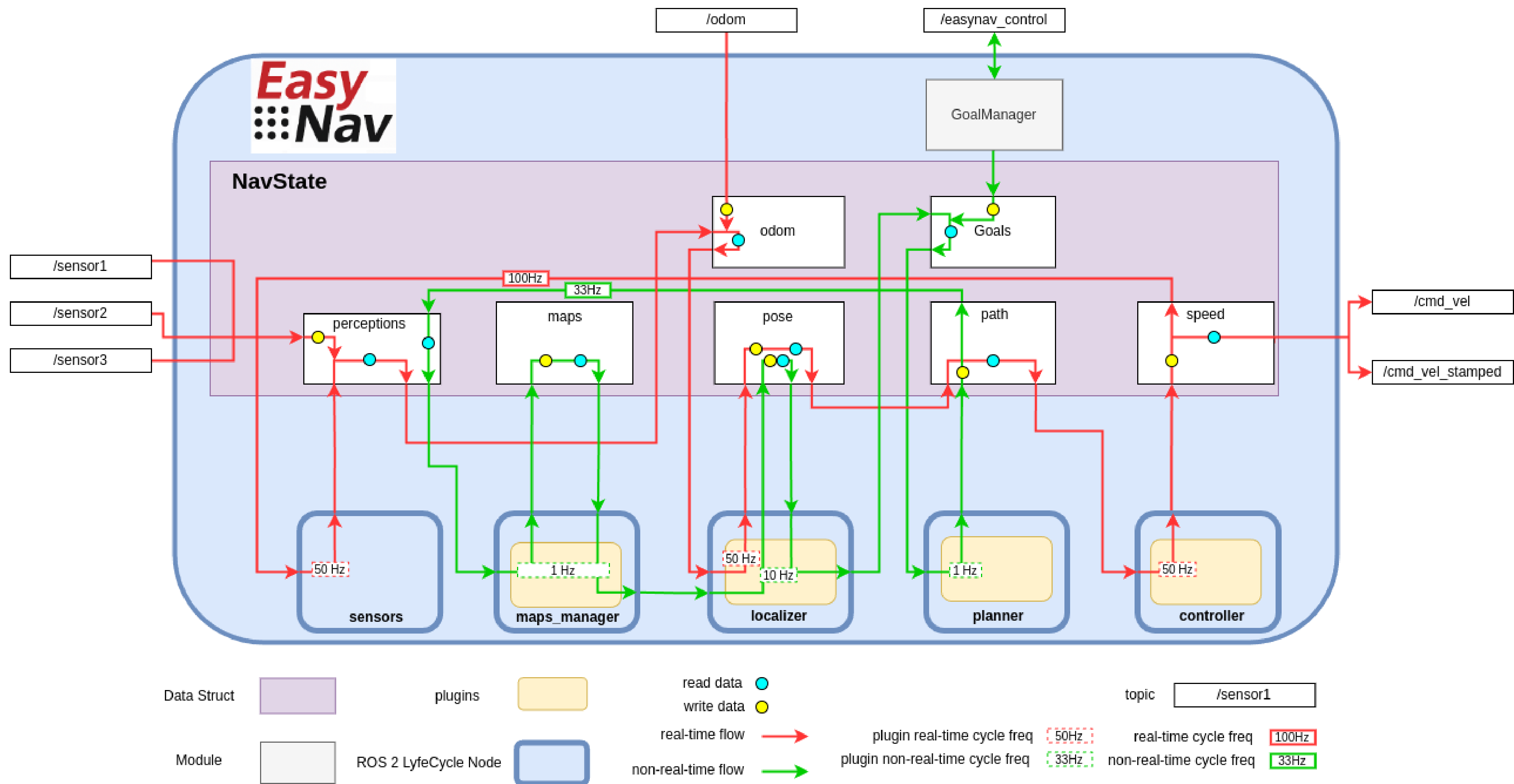
sensors_node:
  ros__parameters:
    use_sim_time: true
    forget_time: 0.5
    sensors: [laser1]
    perception_default_frame: odom
  laser1:
    topic: /scan_raw
    type: sensor_msgs/msg/LaserScan
    group: points

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 displays the EasyNav TUI interface, which is a terminal-based application. The window title is "fmrico@argo: ~/ros/ros2/easynav_ws". The interface is divided into several sections:

- Status:** Commanding
- Navigation Status:**
 - Navigation Control:**
 - 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
 - Goal Info:**
 - 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:**
 - Twist:
 - linear : x=0.186, y=0.000, z=0.000
 - angular: x=0.000, y=0.000, z=0.876
- NavState:**
 - cmd_vel = [0x771c1002d340] : Twist with (0.178672, 0, 0) (0, 0, 0.883066)
 - path = [0x5609b6076a10] : Path with 16 poses and length 3.24853 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:**

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

The bottom of the window shows a status bar with "q Salir", "1 Tab Status", "2 Tab Commanding", and a "palette" button.

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

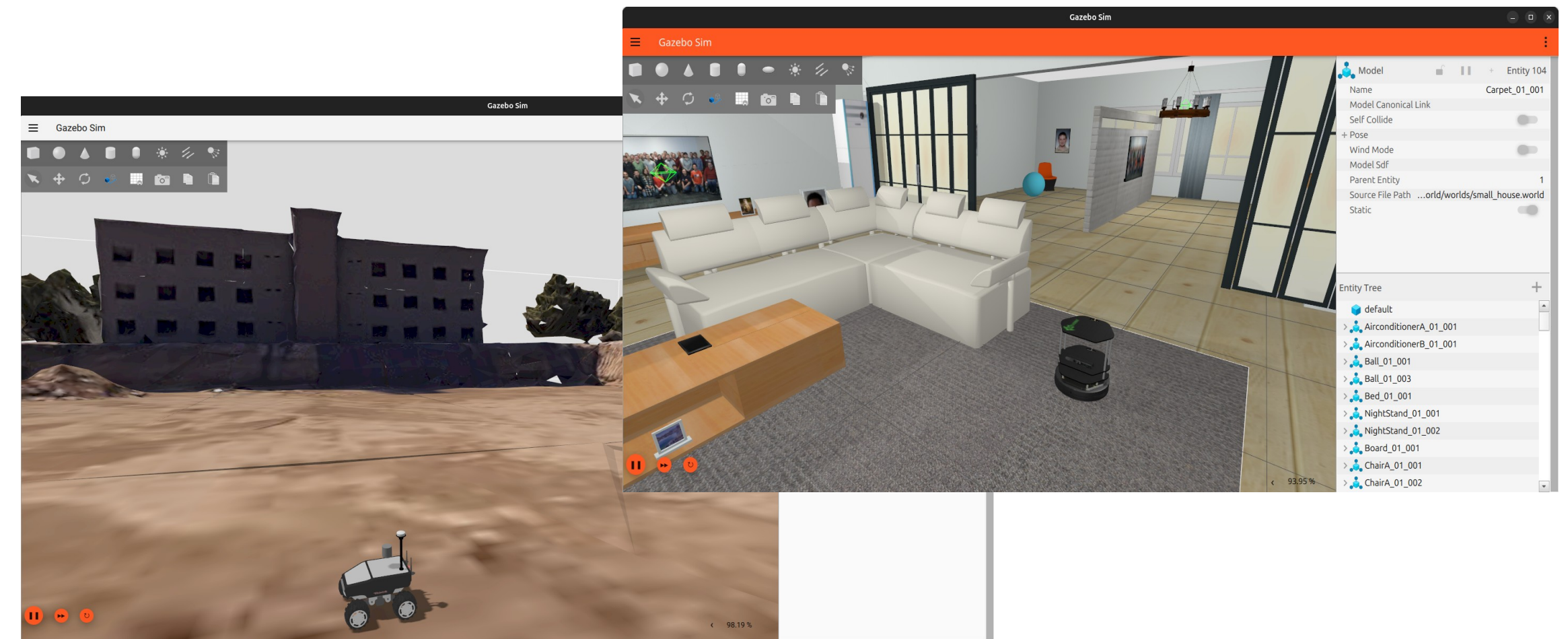
# Observe control loop values
ros2 easynav navigation-control --duration 30

# 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	Doxygen	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

EasyNav

latest

Search docs

Build & Install

Getting Started

EasyNav Plugins

HowTos and Practical Guides

Overview

Simple Stack

Costmap Stack

- 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

General

Developers Guide

About and Contact

🏠 / HowTos and Practical Guides-

Edit

HowTos and Practical Guides

This section contains a curated collection of **HowTos** and short guides for EasyNavigation (EasyNav). Each document provides step-by-step instructions to accomplish specific tasks – from setting up costmaps to configuring controllers and running navigation examples.

If you are new to EasyNav, we recommend starting with [Getting Started](#) and [Build & Install](#) before diving into these guides.

On this page

- Overview
- Simple Stack
- Costmap Stack
- GridMap Stack
- Bonxai Stack
- NavMap Stack
- Controllers
- Behaviors
- General

📖 Overview

The HowTos are grouped by category:

- Simple navigation** – basic examples using the Simple stack.
- Costmap navigation** – using 2D costmaps for mapping and planning.
- GridMap navigation** – elevation-aware mapping and path planning.
- Bonxai navigation** – building and using probabilistic Bonxai maps.
- NavMap navigation** – surface-based mapping and 3D navigation.
- Controllers** – configuring and tuning controllers for different robots.
- Behaviors** – using EasyNav externally from any application or behavior.
- General** – ways to do something independent of a specific stack.

Easy Navigation

Main Page

Packages ▾

Classes ▾

Files ▾

Search

easynav

GoalManagerClient

Public Types | Public Member Functions | List of all members

GoalManagerClient Class Reference

Client-side interface for interacting with **GoalManager**. More...

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

Public Types

```
enum class State {
    IDLE , SENT_GOAL , SENT_PREEMPT , ACCEPTED_AND_NAVIGATING ,
    NAVIGATION_FINISHED , NAVIGATION_REJECTED , NAVIGATION_FAILED , NAVIGATION_CANCELLED ,
    ERROR
}
Internal state of the client-side goal manager. More...
```

Public Member Functions

	void	cancel ()	Cancel the current goal.
const easynav_interfaces::msg::NavigationControl &		get_feedback () const	Get the most recent feedback received.
const easynav_interfaces::msg::NavigationControl &		get_last_control () const	Get the last control message sent or received.
const easynav_interfaces::msg::NavigationControl &		get_result () const	Get the last result message received.
	State	get_state () const	Get the current internal state.
		GoalManagerClient (rclcpp::Node::SharedPtr node)	Constructor.
	void	reset ()	Reset internal client state.
	void	send_goal (const geometry_msgs::msg::PoseStamped &goal)	

https://easynavigation.github.io

EasyNav

Documentation

EasyNavigation



latest

Search docs

Build & Install

Getting Started

EasyNav Plugins

Supported ROS 2 versions

Repository overview

Planners

Controllers

Maps Managers

Localizers

License

HowTos and Practical Guides

Developers Guide

About and Contact

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 [killed](#) [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
killed	killed supported
rolling	rolling supported

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_name>/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	<code>2.0</code>	Scaling factor applied to cell/triangle costs.
<code><plugin>.inflation_penalty</code>	double	<code>5.0</code>	Extra penalty near inflated/inscribed regions to keep paths away from obstacles.
<code><plugin>.continuous_replan</code>	bool	<code>true</code>	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_name>/<plugin>/path</code>	<code>nav_msgs/msg/Path</code>	Publishes the computed A* path.	depth=10

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 <code>layer</code>).
<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 determinist
- 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



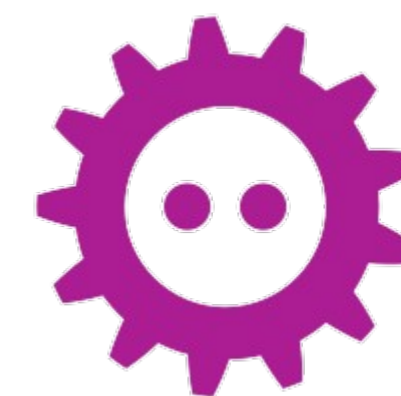
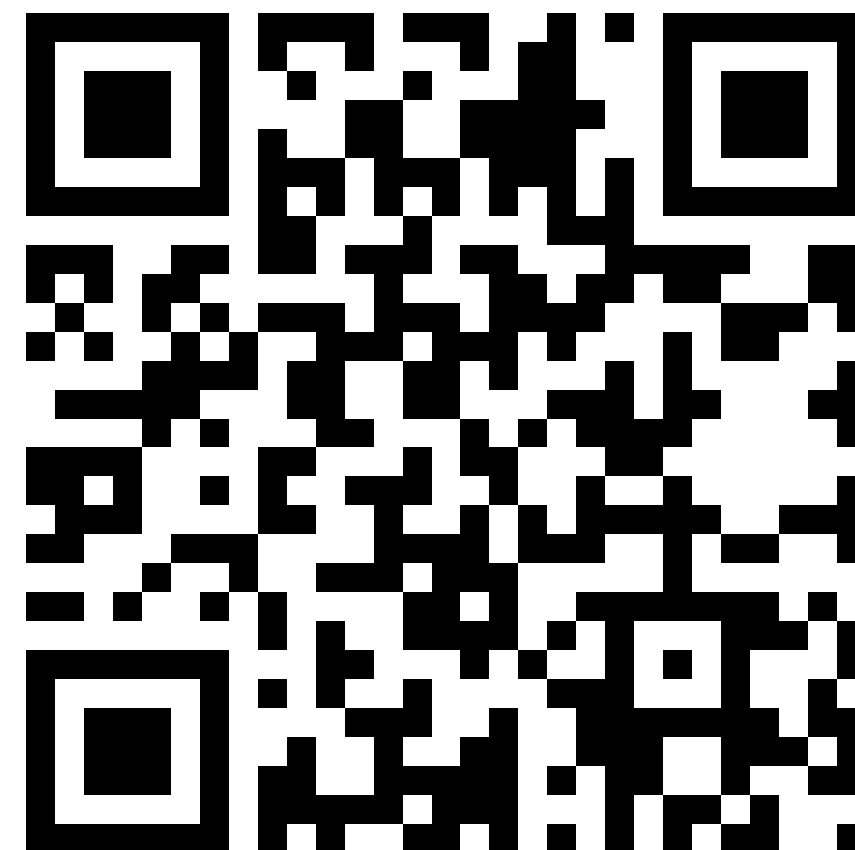
Easy Nav

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