Workshop:Navegación de Robots

Francisco Martín Rico, Francisco Miguel Moreno Olivo, Juan Carlos Manzanares Serrano, José Miguel Guerrero Hernández, Juan Sebastián Cely Gutiérrez, Esther Aguado, and Francisco José Romero Ramírez, Juan Diego Peña

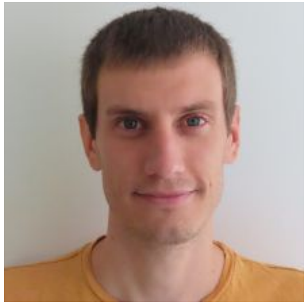
CORESENSE

Intelligent Robotics Lab

Universidad Rey Juan Carlos

Autores

Francisco Martín Rico
(URJC)



Francisco Miguel Moreno Olivo
(URJC)

Esther Aguado
(URJC)

Juan Sebastián Cely Gutiérrez
(URJC)

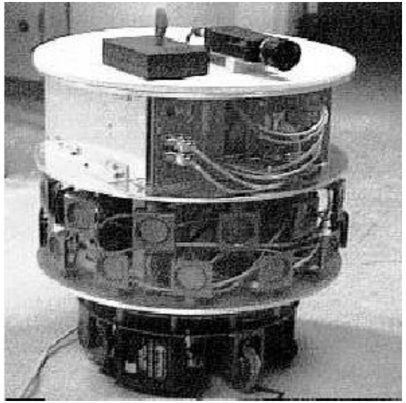
Juan Diego Peña
(URJC)

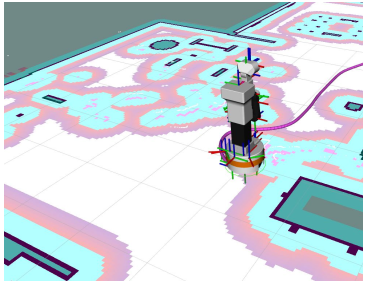
José Miguel Guerrero Hernández
(URJC)

Contenido

Introducción

- Objetivo esencial: moverse de manera autónoma
- Tema de investigación por más de 50 años
- Todavía con muchos retos por resolver





Intelligent Robotics Lab

Introducción

- Mapeo 3D basado en vóxeles
- Localización y Mapeo Simultáneos (SLAM)
- Pruebas en un entorno real (26 millas)
- Una de las primeras demostraciones de una navegación robusta usando ROS (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

Introducción

- ROS -> ROS 2
- Uso de Behavior Trees
- Sistema totalmente modular y extensible por plugins.

The Marathon 2: A Navigation System

Steve Macenski
R&D Innovations
Samsung Research
s.macenski@samsung.com

Francisco Martín
Intelligent Robotics Lab
Rey Juan Carlos University
francisco.rico@urjc.es

Ruffin White
Contextual Robotics Institute
UC San Diego
rwhitema@eng.ucsd.edu

Jonatan Gins Clavero
Intelligent Robotics Lab
Rey Juan Carlos University
jonatan.gines@urjc.es

***Abstract*—Developments in mobile robot navigation have enabled robots to operate in warehouses, retail stores, and on sidewalks around pedestrians. Various navigation solutions have been proposed, though few as widely adopted as ROS (Robot Operating System) Navigation. 10 years on, it is still one of the most popular navigation solutions[1]. Yet, ROS Navigation has failed to keep up with modern trends. We propose the new navigation solution, *Navigation2*, which builds on the successful legacy of ROS Navigation. Navigation2 uses a behavior tree for navigator task orchestration and employs new methods designed for dynamic environments applicable to a wider variety of modern sensors. It is built on top of ROS2, a secure message passing framework suitable for safety critical applications and program lifecycle management. We present experiments in a campus setting utilizing Navigation2 to operate safely alongside students over a marathon as an extension of the experiment proposed in Eppstein et al. [1]. The Navigation2 system is freely available at https://github.com/ros-planning/navigation2 with a rich community and instructions.**

***Index Terms*—Service Robots; Software, Middleware and Programming Environments; Behaviour-Based Systems**

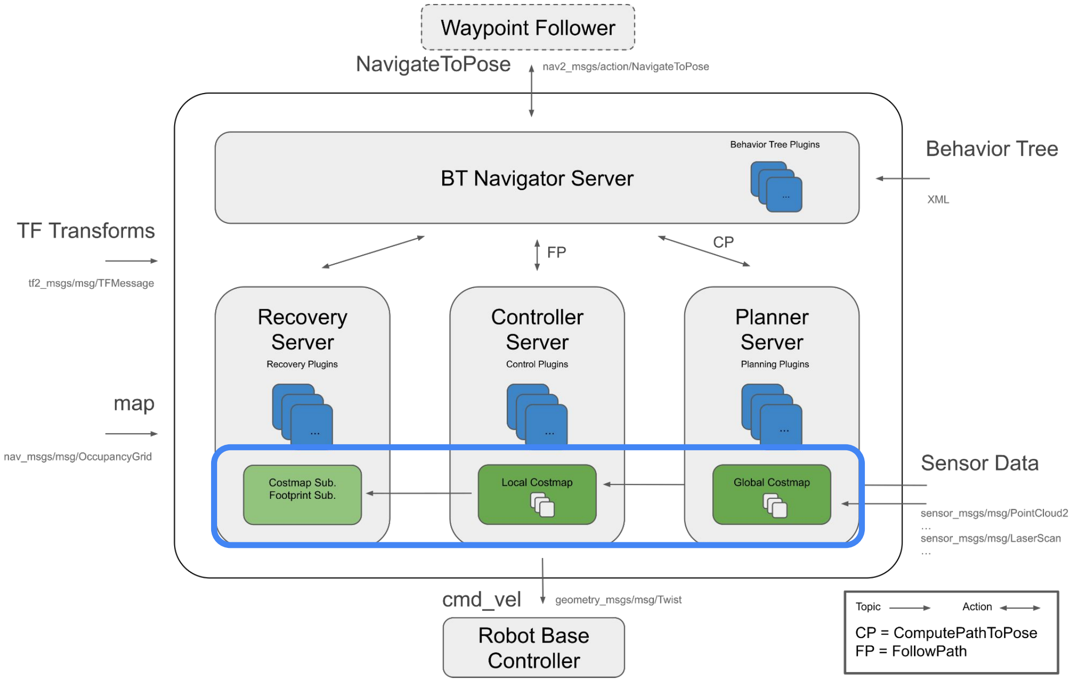
I. Introduction

Many mobile robot navigation frameworks and systems

(a) Tiago [12] (b) RB-1 [13]

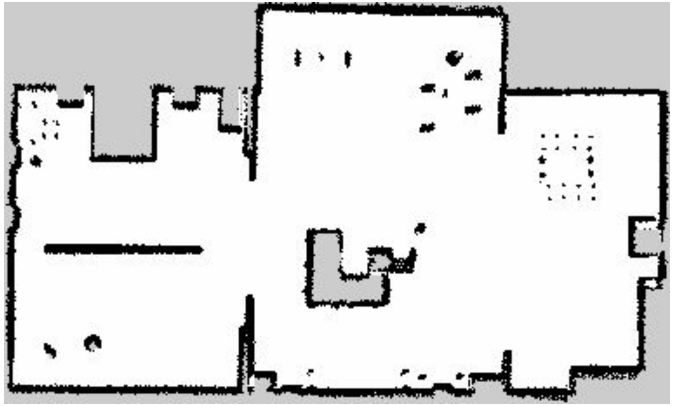
Fig. 1: Robots used for the marathon experiments.

Introducción



Representación de mapas

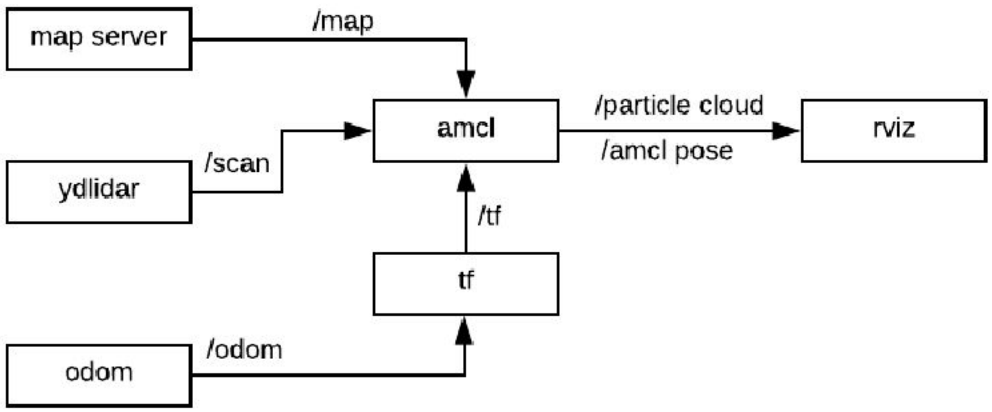
```yaml
image: aws_house.pgm
mode: trinary
resolution: 0.05
origin: [-9.62, -5.83, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```



```
std_msgs/Header header
nav_msgs/MapMetaData info
int8[] data
```

Localización

- Posición del robot en el mapa
- Se usa el sistema de TFs
- Nav2 utiliza una implementación del algoritmo AMCL

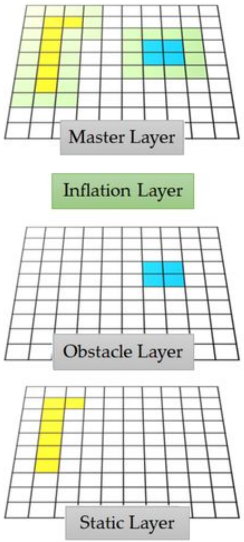


Localización

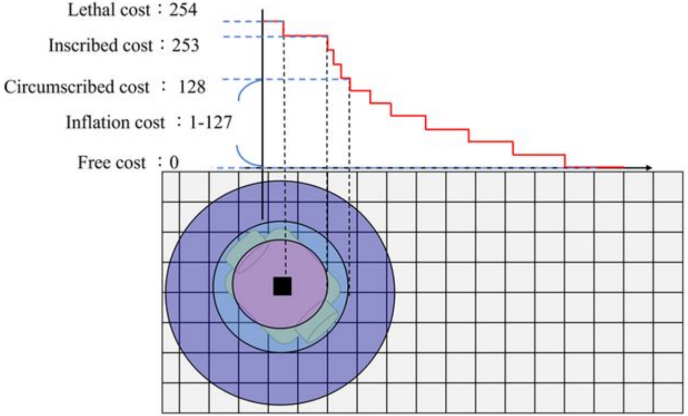
```yaml
amcl:
  ros__parameters:
    use_sim_time: true
    alpha1: 0.1
    alpha2: 0.1
    alpha3: 0.1
    alpha4: 0.1
    base_frame_id: base_footprint
    beam_skip_distance: 0.5
    beam_skip_error_threshold: 0.9
    beam_skip_threshold: 0.3
    do_beamskip: false
    global_frame_id: map
    lambda_short: 0.1
    laser_likelihood_max_dist: 2.0
    laser_max_range: 100.0
    laser_min_range: -1.0
    laser_model_type: "likelihood_field"
    max_beams: 60
```

```yaml
    min_particles: 500
    odom_frame_id: odom
    pf_err: 0.05
    pf_z: 0.99
    recovery_alpha_fast: 0.0
    recovery_alpha_slow: 0.0
    resample_interval: 1
    robot_model_type: "nav2_amcl::DifferentialMotionModel"
    save_pose_rate: 0.5
    sigma_hit: 0.2
    tf_broadcast: true
    transform_tolerance: 1.0
    update_min_a: 0.2
    update_min_d: 0.25
    z_hit: 0.5
    z_max: 0.05
    z_rand: 0.5
    z_short: 0.05
    scan_topic: /scan_raw
```

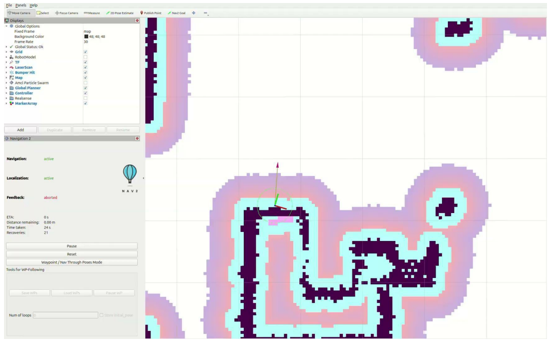
Costmaps



Master Layer
Inflation Layer
Obstacle Layer
Static Layer



Lethal cost : 254
Inscribed cost : 253
Circumscribed cost : 128
Inflation cost : 1-127
Free cost : 0



Costmaps

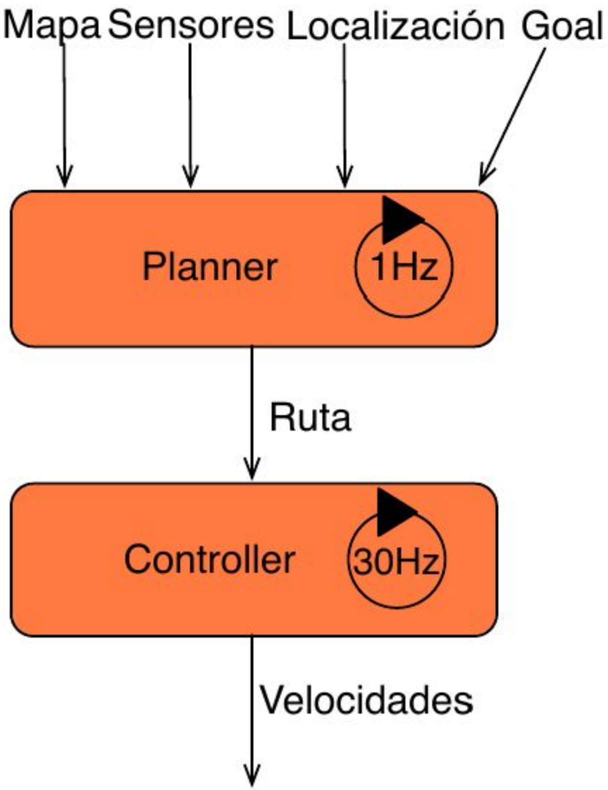
```yaml
global_costmap:
  global_costmap:
    ros__parameters:
      update_frequency: 1.0
      publish_frequency: 1.0
      global_frame: map
      robot_base_frame: base_footprint
      use_sim_time: true
      robot_radius: 0.18
      resolution: 0.05
      track_unknown_space: true
      plugins: ["static_layer", "obstacle_layer", "inflation_layer"]
      obstacle_layer:
        plugin: "nav2_costmap_2d::ObstacleLayer"
        enabled: True
        observation_sources: scan
        scan:
          topic: /scan_raw
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
          raytrace_max_range: 3.0
          raytrace_min_range: 0.0
          obstacle_max_range: 2.5
          obstacle_min_range: 0.0
      static_layer:
        plugin: "nav2_costmap_2d::StaticLayer"
        map_subscribe_transient_local: True
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.55
      always_send_full_costmap: True
```

```yaml
local_costmap:
  local_costmap:
    ros__parameters:
      update_frequency: 5.0
      publish_frequency: 2.0
      global_frame: odom
      robot_base_frame: base_footprint
      use_sim_time: true
      rolling_window: true
      width: 3
      height: 3
      resolution: 0.05
      robot_radius: 0.275
      plugins: ["voxel_layer", "inflation_layer"]
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.55
      voxel_layer:
        plugin: "nav2_costmap_2d::VoxelLayer"
        enabled: True
        publish_voxel_map: True
        origin_z: 0.0
        z_resolution: 0.05
        z_voxels: 16
        max_obstacle_height: 2.0
        mark_threshold: 0
        observation_sources: scan
        scan:
          topic: /scan_raw
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
          raytrace_max_range: 3.0
          raytrace_min_range: 0.0
          obstacle_max_range: 2.5
          obstacle_min_range: 0.0
      static_layer:
```

Costmaps

- Voxel Layer
- Range Layer
- Static Layer
- Inflation Layer
- Obstacle Layer
- Spatio-Temporal Voxel Layer
- Non-Persistent Voxel Layer

Planificación de trayectorias



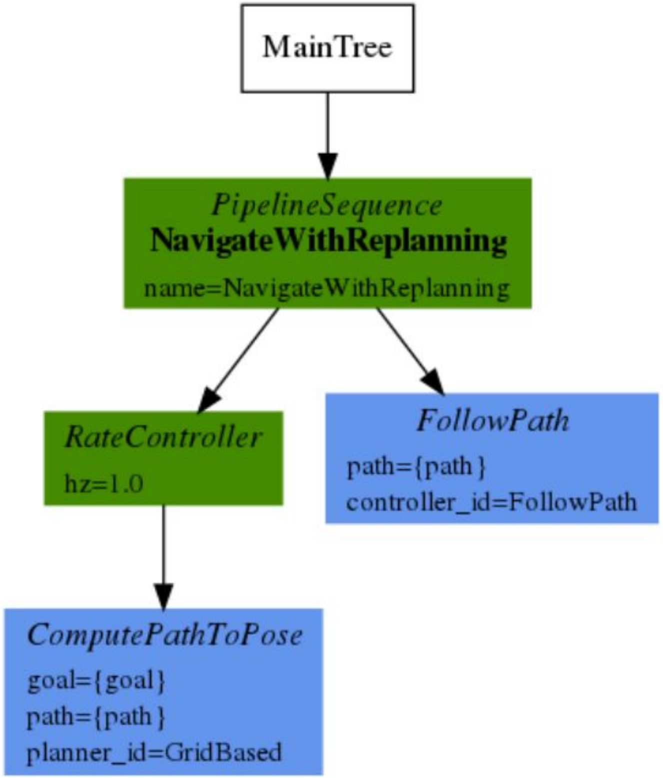

```yaml
planner_server:
  ros__parameters:
    expected_planner_frequency: 20.0
    use_sim_time: true
    planner_plugins: ["GridBased"]
    GridBased:
      plugin: "nav2_navfn_planner::NavfnPlanner"
      tolerance: 0.5
      use_astar: false
      allow_unknown: true
```

Controlador

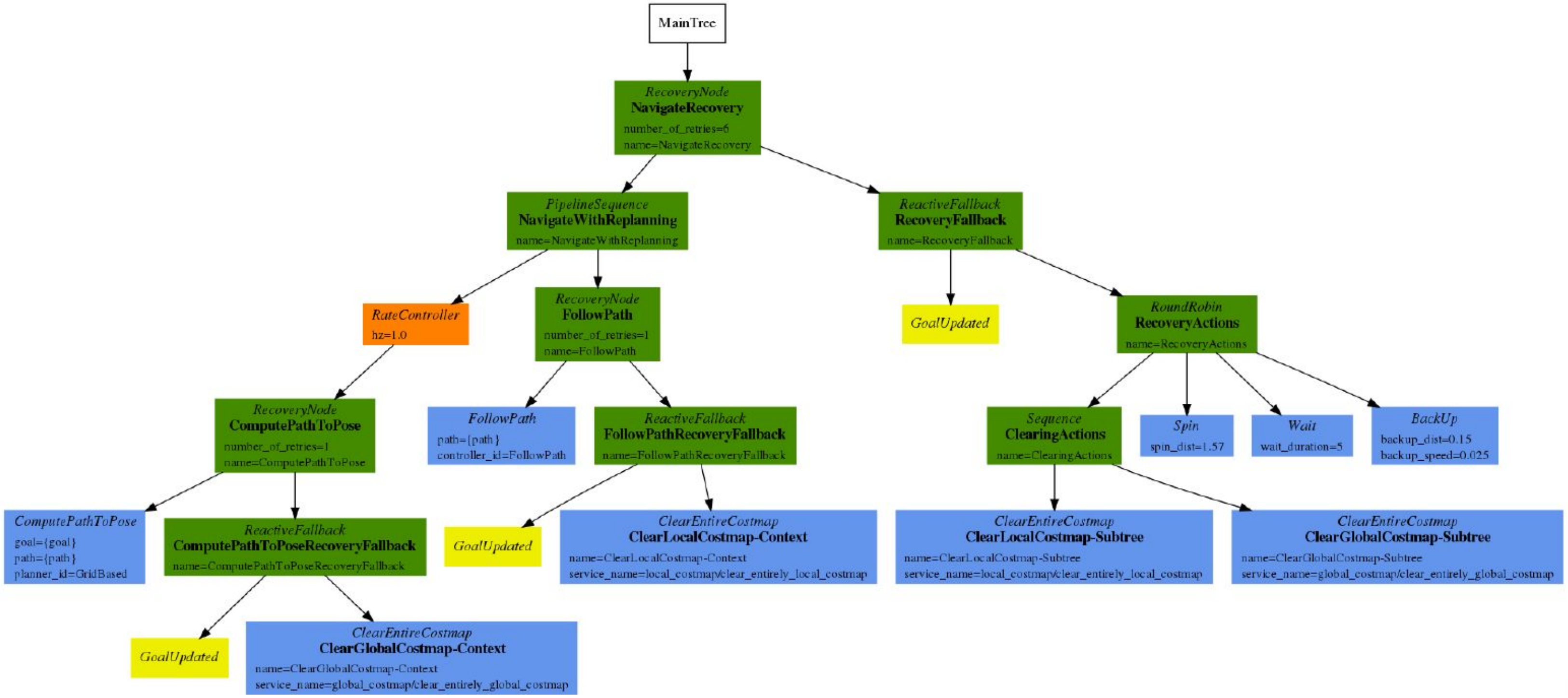
- Siguen la trayectoria global o usando el local costmap

Plugin Name	Creator	Description	Drivetrain support
DWB Controller	David Lu!!	A highly configurable DWA implementation with plugin interfaces	Differential, Omnidirectional, Legged
TEB Controller	Christoph Rösmann	A MPC-like controller suitable for ackermann, differential, and holonomic robots.	**Ackermann**, Legged, Omnidirectional, Differential
Regulated Pure Pursuit	Steve Macenski	A service / industrial robot variation on the pure pursuit algorithm with adaptive features.	**Ackermann**, Legged, Differential
MPPI Controller	Steve Macenski Aleksei Budyakov	A predictive MPC controller with modular & custom cost functions that can accomplish many tasks.	Differential, Omni, **Ackermann**
Rotation Shim Controller	Steve Macenski	A "shim" controller to rotate to path heading before passing to main controller for tracking.	Differential, Omni, model rotate in place
Graceful Controller	Alberto Tudela	A controller based on a pose-following control law to generate smooth trajectories.	Differential, Omni, Legged
Vector Pursuit Controller	Black Coffee Robotics	A controller based on the vector pursuit algorithm useful for high speed accurate path tracking.	Differential, Ackermann, Legged,

Orquestación de la navegación



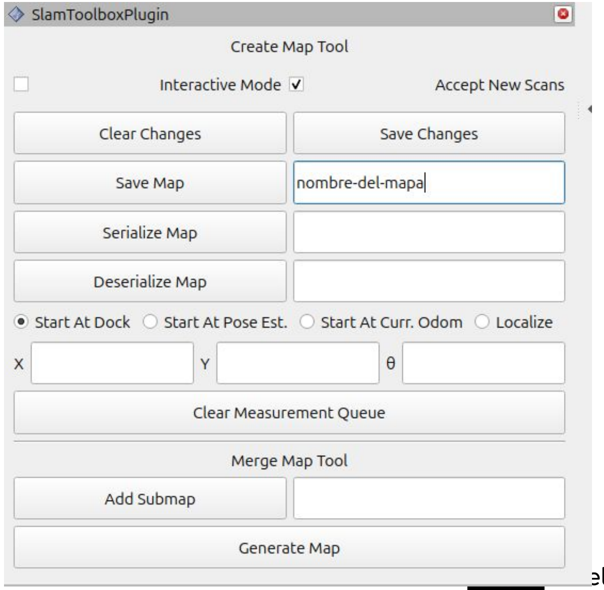
Orquestación de la navegación



Generación de mapas con slam toolbox

https://github.com/EasyNavigation/roscon2025_workshop/blob/main/exercises/nav2/slam.md

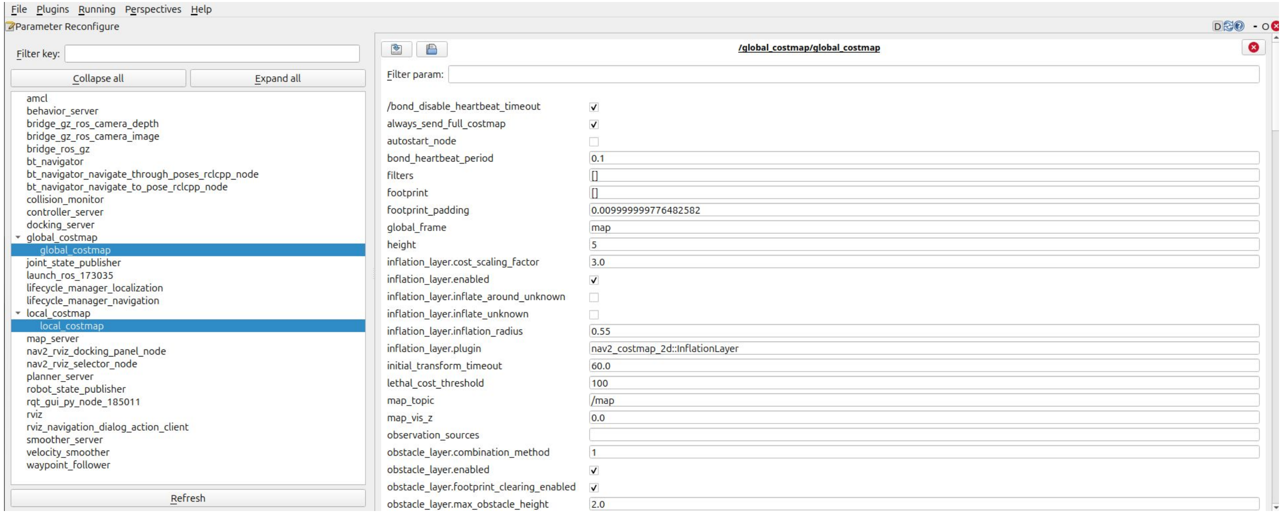
1. ros2 launch nav2_playground playground_kobuki.launch.py
2. ros2 launch nav2_playground slam_launch.py
3. ros2 run teleop_twist_keyboard teleop_twist_keyboard
4. ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name:
 data: 'nombre-del-mapa'"



Configuración básica

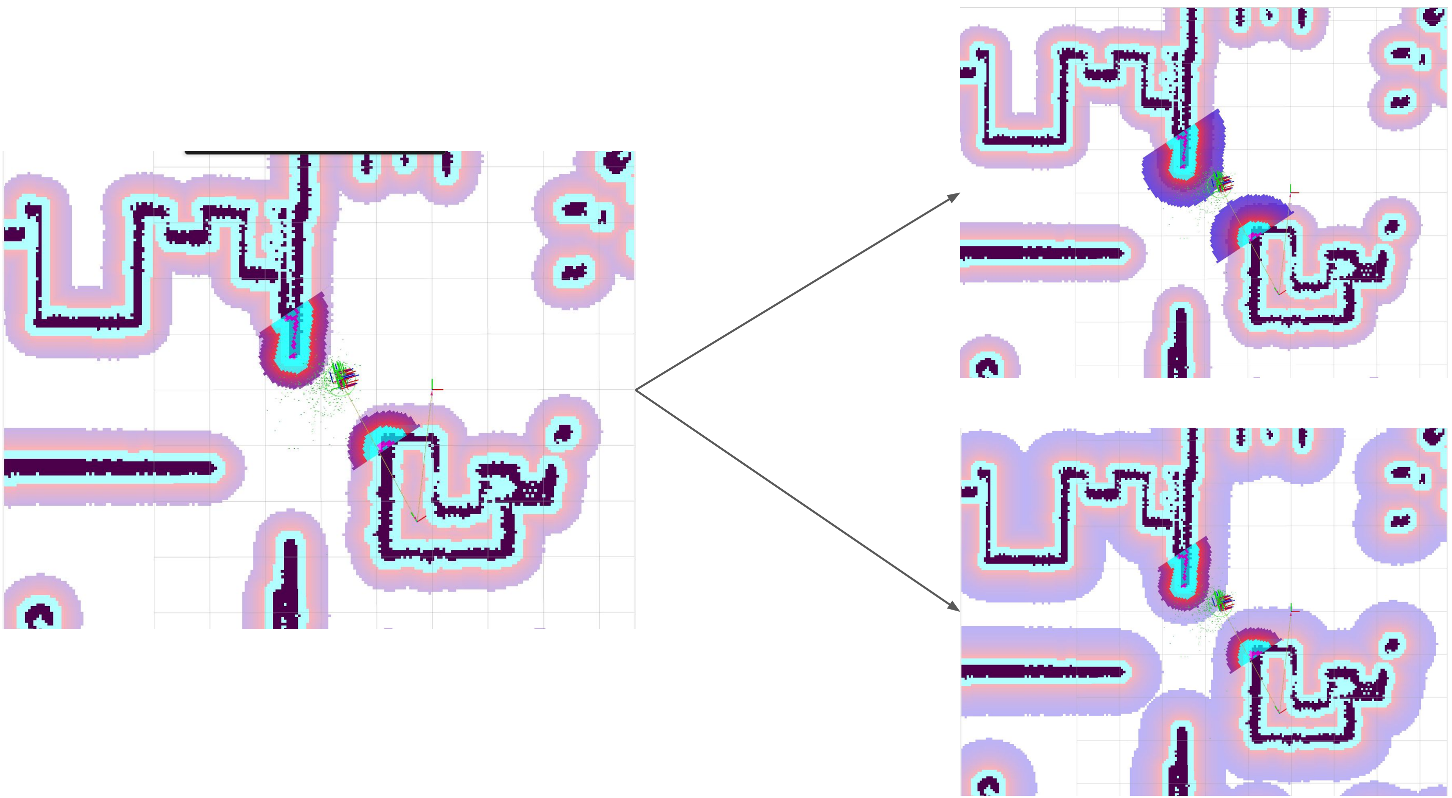
https://github.com/EasyNavigation/roscon2025_workshop

1. ros2 launch nav2_playground navigation_launch.py map:=<path-to-generated-yaml>
2. rqt



Configuración básica

https://github.com/EasyNavigation/roscon2025_workshop



Intelligent Robotics Lab

Patrullaje

https://github.com/EasyNavigation/roscon2025_workshop/blob/main/exercises/nav2/patrolling_exercise.md

```cpp
case PatrolState::IDLE:
  if (!initialized_) {
    RCLCPP_INFO(get_logger(), "Initializing patrolling node");

    // Create Nav2 action client
    nav_client_ = rclcpp_action::create_client<NavigateToPose>(
      this,
      "navigate_to_pose");

    initialize();
    initialized_ = true;

    if (waypoints_.empty()) {
      RCLCPP_ERROR(get_logger(), "Cannot start patrol with no waypoints");
      state_ = PatrolState::ERROR;
      break;
    }
  }

  // Wait for action server
  if (!nav_client_->wait_for_action_server(std::chrono::seconds(0))) {
    RCLCPP_INFO_THROTTLE(
      get_logger(), *get_clock(), 5000,
      "Waiting for navigate_to_pose action server...");
    break;
  }

  RCLCPP_INFO(get_logger(), "Starting patrol with %zu waypoints", waypoints_.size());
  current_waypoint_index_ = 0;
  state_ = PatrolState::SENDING_GOAL;
  break;
```

```yaml
/**:
  ros__parameters:
    frame_id: "map"
    waypoints: ["wp1", "wp2", "wp3", "wp4"]
    wp1: [0.0, 0.0, 0.0]
    wp2: [2.0, 0.0, 1.57]
    wp3: [2.0, 2.0, 3.14]
    wp4: [0.0, 2.0, -1.57]
```

Patrullaje

https://github.com/EasyNavigation/roscon2025_workshop/blob/main/exercises/nav2/patrolling_exercise.md

```
case PatrolState::SENDING_GOAL:
  {
    if (current_waypoint_index_ >= waypoints_.size()) {
      RCLCPP_INFO(get_logger(), "All waypoints visited");
      state_ = PatrolState::FINISHED;
      return;
    }

    auto goal_msg = NavigateToPose::Goal();
    goal_msg.pose = waypoints_[current_waypoint_index_];
    goal_msg.pose.header.stamp = now();

    RCLCPP_INFO(
    get_logger(), "Sending goal %zu/%zu: [%.2f, %.2f]",
    current_waypoint_index_ + 1, waypoints_.size(),
    goal_msg.pose.pose.position.x,
    goal_msg.pose.pose.position.y);

    current_future_goal_handle_ = nav_client_->async_send_goal(goal_msg);
    state_ = PatrolState::NAVIGATING;
    break;
  }
```

Patrullaje

https://github.com/EasyNavigation/roscon2025_workshop/blob/main/exercises/nav2/patrolling_exercise.md

```cpp
case PatrolState::NAVIGATING:
  {
    if (!current_future_goal_handle_.valid()) {
      RCLCPP_ERROR(get_logger(), "No goal handle available");
      state_ = PatrolState::ERROR;
      break;
    }

    current_goal_handle_ = current_future_goal_handle_.get();
    if (!current_goal_handle_) {
      RCLCPP_ERROR(get_logger(), "Goal was not accepted by the action server");
      state_ = PatrolState::ERROR;
      break;
    }

    auto status = current_goal_handle_->get_status();

    switch (status) {
      case action_msgs::msg::GoalStatus::STATUS_ACCEPTED:
        RCLCPP_INFO_THROTTLE(
        get_logger(), *get_clock(), 2000,
        "Goal accepted, waiting to start execution...");
        break;

      case action_msgs::msg::GoalStatus::STATUS_EXECUTING:
        RCLCPP_INFO_THROTTLE(
        get_logger(), *get_clock(), 2000,
        "Navigating to waypoint %zu/%zu...",
        current_waypoint_index_ + 1, waypoints_.size());
        break;

      case action_msgs::msg::GoalStatus::STATUS_SUCCEEDED:
        RCLCPP_INFO(
        get_logger(), "Successfully reached waypoint %zu/%zu",
        current_waypoint_index_ + 1, waypoints_.size());
        current_waypoint_index_++;
        current_goal_handle_.reset();
        state_ = PatrolState::SENDING_GOAL;
        break;

      default:
        RCLCPP_ERROR(
        get_logger(), "Unexpected goal status: %d", status);
        state_ = PatrolState::ERROR;
        break;
    }
    break;
  }
```

Intelligent Robotics Lab

Patrullaje

https://github.com/EasyNavigation/roscon2025_workshop/blob/main/exercises/nav2/patrolling_exercise.md

```
case PatrolState::FINISHED:
  RCLCPP_INFO(get_logger(), "Patrol cycle completed. Restarting from first waypoint.");
  current_waypoint_index_ = 0;
  state_ = PatrolState::SENDING_GOAL;
  break;
```

Patrullaje: Ejercicio 1

https://github.com/EasyNavigation/roscon2025_workshop/blob/main/exercises/nav2/patrolling_exercise.md

```cpp
case PatrolState::DO_SOMETHING_AT_WAYPOINT:
  {
  // Implement what the robot should do when it reaches a waypoint.
  // Ideas:
  // After completing the task, you should:
  //   1. Increment current_waypoint_index_
  //   2. Transition to PatrolState::SENDING_GOAL
  //
  // YOUR CODE HERE

    break;
  }
```

```python
elif self.state == PatrolState.DO_SOMETHING_AT_WAYPOINT:
    # Implement what the robot should do when it reaches a waypoint.
    #
    # After completing the task, you should:
    #   1. Increment self.current_waypoint_index
    #   2. Transition to PatrolState.SENDING_GOAL
    # YOUR CODE HERE
    pass
```

Patrolling usando NavigateThroughPoses

https://github.com/EasyNavigation/roscon2025_workshop

```cpp
class WaypointClientNode : public rclcpp::Node {
public:
  using FollowWaypoints = nav2_msgs::action::FollowWaypoints;
  using GoalHandleFollowWaypoints = rclcpp_action::ClientGoalHandle<FollowWaypoints>;

  WaypointClientNode()
  : Node("waypoint_client_node")
  {
    declare_parameter<std::string>("frame_id", "map");
    declare_parameter<std::vector<std::string>>("waypoints", std::vector<std::string>{});

    load_waypoints_from_params();
    action_client_ = rclcpp_action::create_client<FollowWaypoints>(
      this,
      "follow_waypoints");

    if (waypoints_.empty()) {
      RCLCPP_WARN(get_logger(), "No waypoints loaded from parameters.");
      return;
    }

    send_goal();
  }
```

Patrolling usando NavigateThroughPoses

https://github.com/EasyNavigation/roscon2025_workshop

```cpp
#include "nav2_playground/plugins/print_pick_task_executor.hpp"
#include <pluginlib/class_list_macros.hpp>
#include <thread>
#include <chrono>

namespace nav2_playground
{

void PrintPickTaskExecutor::initialize(const rclcpp_lifecycle::LifecycleNode::WeakPtr & parent, const std::string & plugin_name)
{
  parent_weak_ = parent;
  plugin_name_ = plugin_name;
  auto parent_locked = parent_weak_.lock();
  if (!parent_locked) {
    RCLCPP_ERROR(logger_, "[%s] Failed to lock parent lifecycle node", plugin_name_.c_str());
    return;
  }
  logger_ = parent_locked->get_logger();
  parent_locked->declare_parameter(plugin_name_ + ".simulated_delay_ms", simulated_delay_ms_);
  parent_locked->get_parameter(plugin_name_ + ".simulated_delay_ms", simulated_delay_ms_);
  RCLCPP_INFO(logger_, "[%s] Initialized with simulated delay %d ms", plugin_name_.c_str(), simulated_delay_ms_);
}

bool PrintPickTaskExecutor::processAtWaypoint(const geometry_msgs::msg::PoseStamped & curr_pose, const int & curr_waypoint_index)
{
  (void)curr_pose;
  RCLCPP_INFO(logger_, "[%s] Performing PICK operation at waypoint %d", plugin_name_.c_str(), curr_waypoint_index);
  std::this_thread::sleep_for(std::chrono::milliseconds(simulated_delay_ms_));
  RCLCPP_INFO(logger_, "[%s] PICK complete at waypoint %d", plugin_name_.c_str(), curr_waypoint_index);
  return true;  //
}

}  // namespace nav2_playground

PLUGINLIB_EXPORT_CLASS(nav2_playground::PrintPickTaskExecutor, nav2_core::WaypointTaskExecutor)
```

Patrolling usando NavigateThroughPoses

https://github.com/EasyNavigation/roscon2025_workshop

```yaml
waypoint_follower:
  ros__parameters:
    use_sim_time: true
    stop_on_failure: false
    waypoint_task_executor_plugin: "wait_at_waypoint"
    wait_at_waypoint:
      plugin: "nav2_waypoint_follower::WaitAtWaypoint"
      enabled: True
      waypoint_pause_duration: 0
    print_pick_task:
      plugin: "nav2_playground::PrintPickTaskExecutor"
      simulated_delay_ms: 1200
```

Patrullaje

https://github.com/EasyNavigation/roscon2025_workshop

```
ros2 launch nav2_playground patrol_launch.py
```

Intelligent Robotics Lab