ROS NAVIGATION

Paloma de la Puente

INDUSTRIALES
ETSII | UPM

Content

- Overview
- ROS SLAM
- ROS localization
- ROS navigation

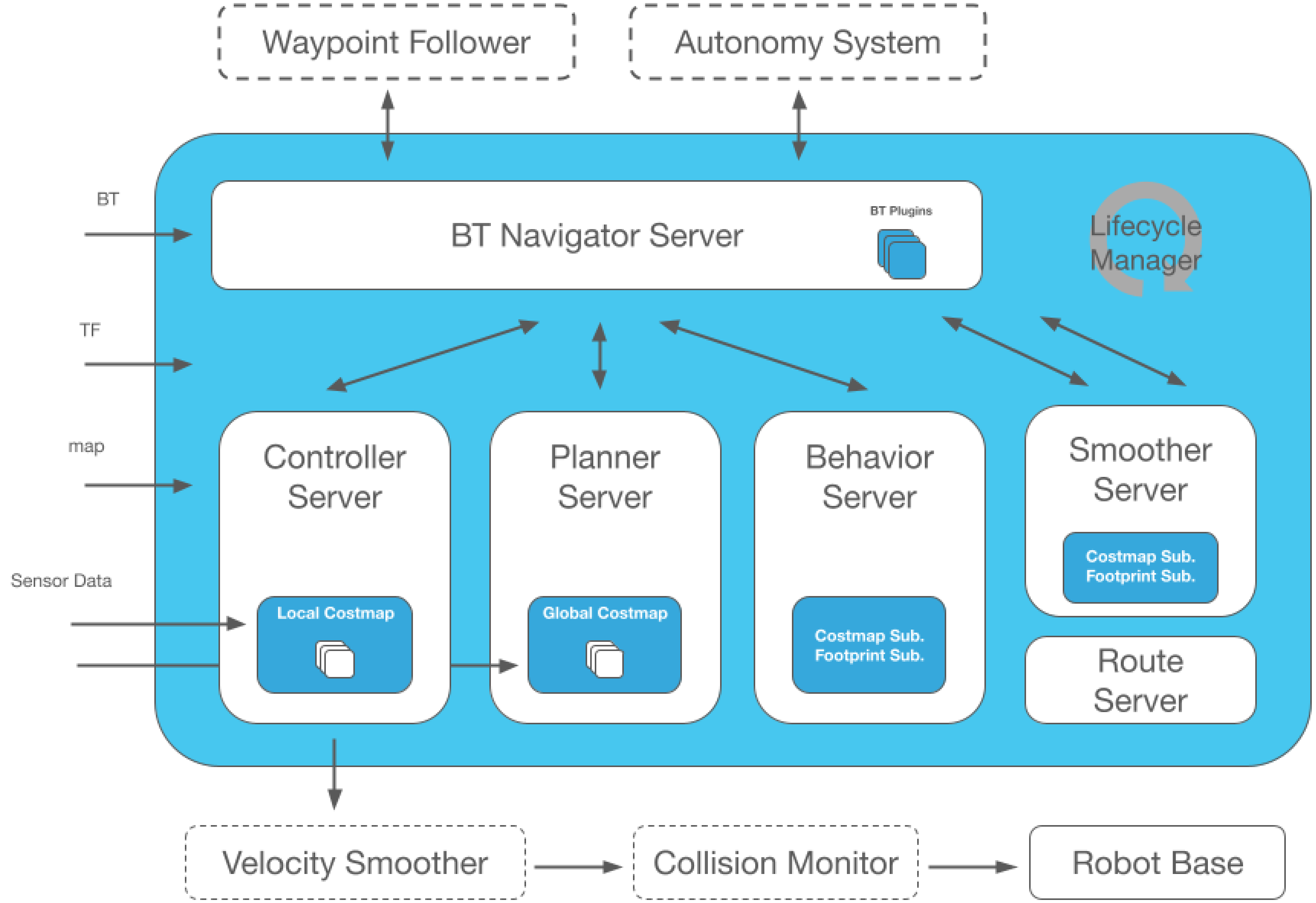
ROS NAV2

OVERVIEW

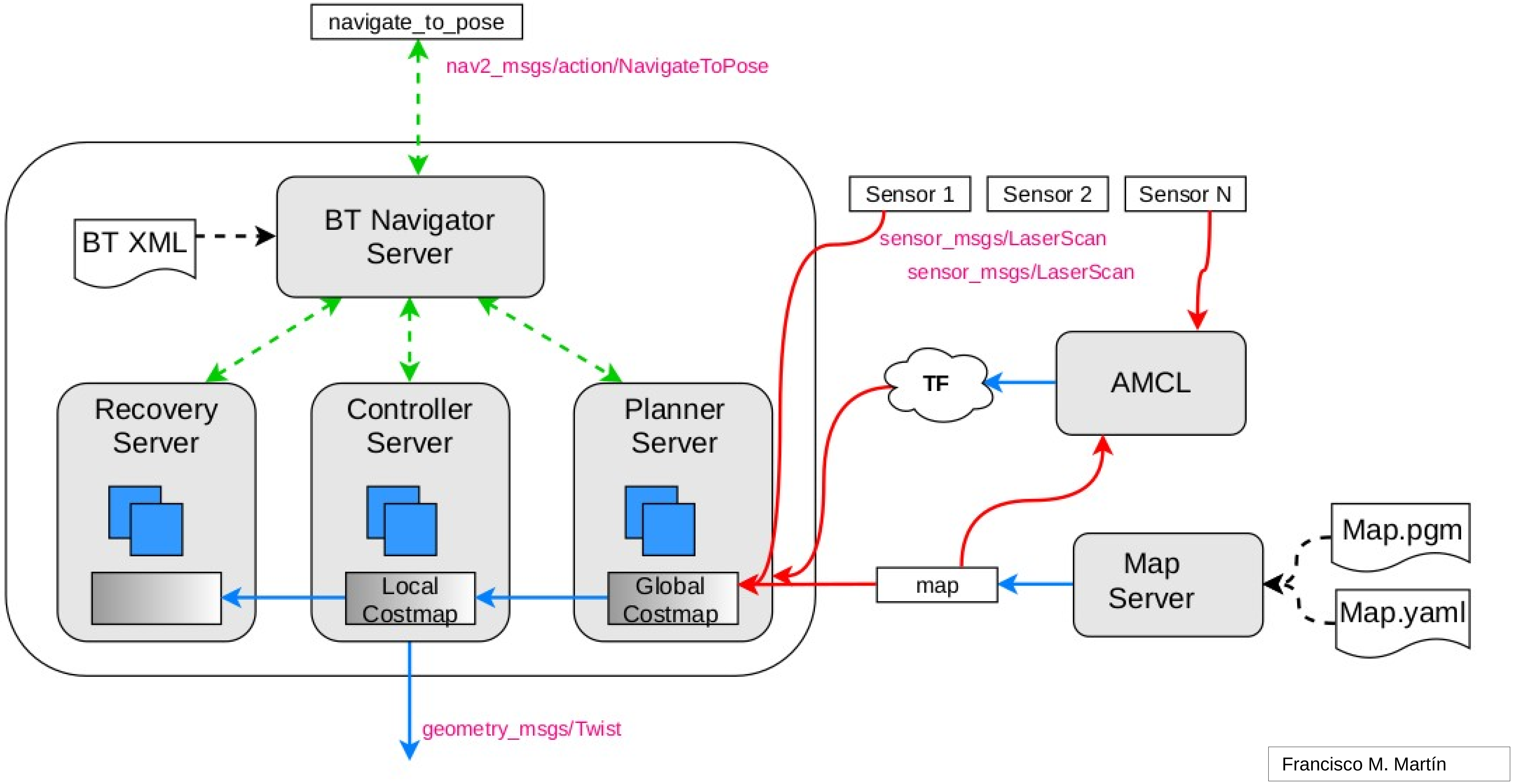
Overview

- Main components:
  - SLAM: slam_toolbox, Cartographer
  - map_server
  - localization: amcl, robot_localization
  - Nav2
    - global_planners
    - smoothers
    - controllers
    - recovery behaviours

Overview


Waypoint Follower
Autonomy System
BT
TF
map
Sensor Data
BT Navigator Server
BT Plugins
Lifecycle
Manager
Controller
Server
Local Costmap
Planner
Server
Global Costmap
Behavior
Server
Costmap Sub.
Footprint Sub.
Smoother
Server
Costmap Sub.
Footprint Sub.
Route
Server
Velocity Smoother
Collision Monitor
Robot Base

Overview

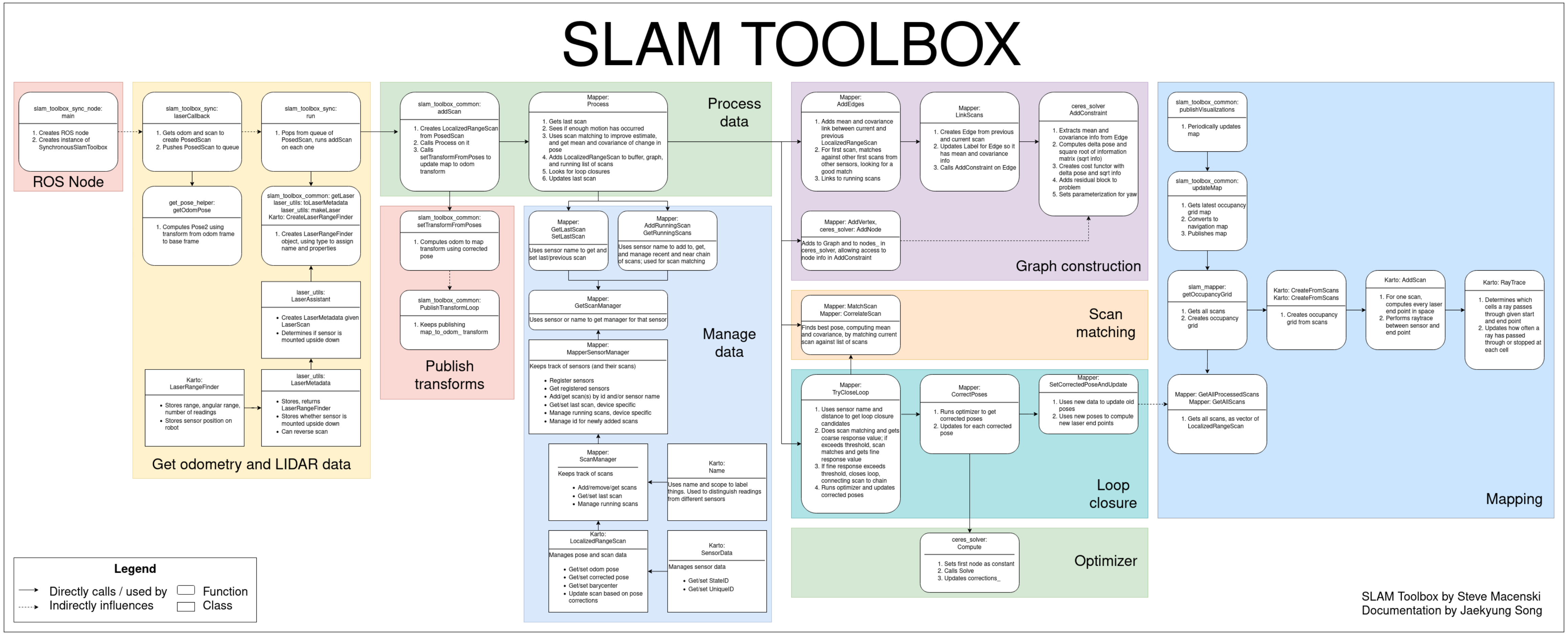

navigate_to_pose
nav2_msgs/action/NavigateToPose
BT XML
BT Navigator Server
Sensor 1
Sensor 2
Sensor N
sensor_msgs/LaserScan
sensor_msgs/LaserScan
Recovery Server
Controller Server
Planner Server
TF
AMCL
Local Costmap
Global Costmap
map
Map Server
Map.pgm
Map.yaml
geometry_msgs/Twist
Francisco M. Martín

ROS NAV2

SLAM

SLAM

- SLAM toolbox



Macenski, S., Jambrecic I., "SLAM Toolbox: SLAM for the dynamic world", Journal of Open Source Software, 6(61), 2783, 2021.

Macenski, S., "On Use of SLAM Toolbox, A fresh(er) look at mapping and localization for the dynamic world", ROSCon 2019.

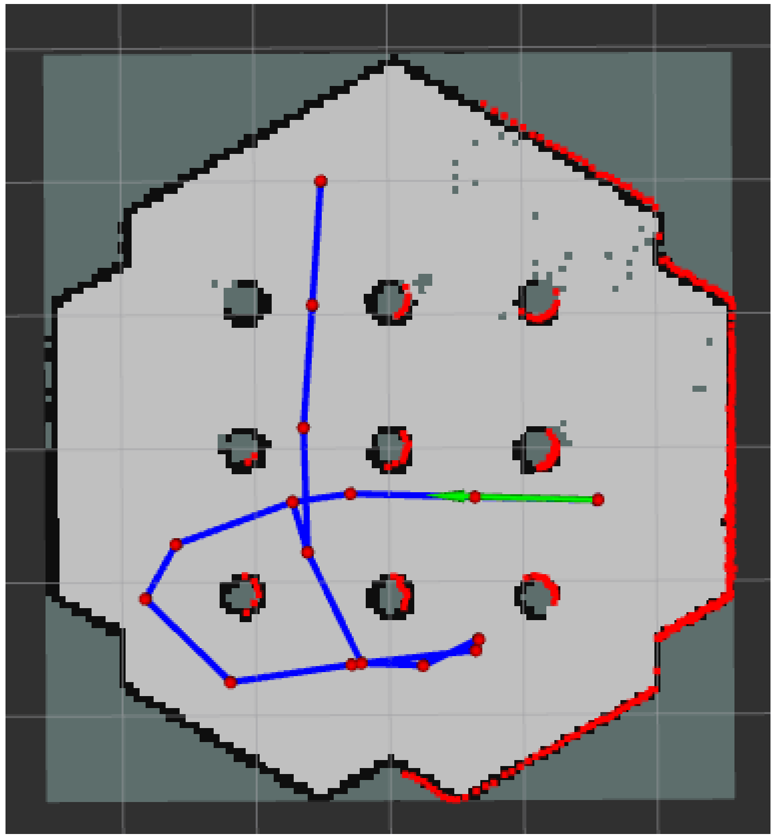
https://github.com/SteveMacenski/slam_toolbox/

SLAM

$sudo apt-get install ros-humble-slam-toolbox

Reference launch file at:
https://github.com/SteveMacenski/slam_toolbox/blob/ros2/launch/online_sync_launch.py

```python
import os

from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory


def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time')
    slam_params_file = LaunchConfiguration('slam_params_file')

    declare_use_sim_time_argument = DeclareLaunchArgument(
        'use_sim_time',
        default_value='true',
        description='Use simulation/Gazebo clock')
    declare_slam_params_file_cmd = DeclareLaunchArgument(
        'slam_params_file',
        #default_value=os.path.join(get_package_share_directory("slam_toolbox"),'config', 'mapper_params_online_sync.yaml'),
        default_value='mapping_params.yaml',
        description='Full path to the ROS2 parameters file to use for the slam_toolbox node')

    start_sync_slam_toolbox_node = Node(
        parameters=[
          slam_params_file,
          {'use_sim_time': use_sim_time}
        ],
        package='slam_toolbox',
        executable='sync_slam_toolbox_node',
        name='slam_toolbox',
        output='screen')

    ld = LaunchDescription()

    ld.add_action(declare_use_sim_time_argument)
    ld.add_action(declare_slam_params_file_cmd)
    ld.add_action(start_sync_slam_toolbox_node)

    return ld
```



SLAM

```yaml
slam_toolbox:
  ros__parameters:

    # Plugin params
    solver_plugin: solver_plugins::CeresSolver
    ceres_linear_solver: SPARSE_NORMAL_CHOLESKY
    ceres_preconditioner: SCHUR_JACOBI
    ceres_trust_strategy: LEVENBERG_MARQUARDT
    ceres_dogleg_type: TRADITIONAL_DOGLEG
    ceres_loss_function: None

    # ROS Parameters
    odom_frame: odom
    map_frame: map
    base_frame: base_link
    scan_topic: /base_scan
    mode: mapping

    debug_logging: false
    throttle_scans: 1
    transform_publish_period: 0.02 #if 0 never publishes odometry
    map_update_interval: 5.0
    resolution: 0.05
    max_laser_range: 20.0 #for rastering images
    minimum_time_interval: 0.5
    transform_timeout: 0.2
    tf_buffer_duration: 30.
    stack_size_to_use: 40000000 #// program needs a larger stack size to serialize large maps
    enable_interactive_mode: true

    # General Parameters
    use_scan_matching: true
    use_scan_barycenter: true
    minimum_travel_distance: 0.5
    minimum_travel_heading: 0.5
    scan_buffer_size: 10
    scan_buffer_maximum_scan_distance: 10.0
    link_match_minimum_response_fine: 0.1
    link_scan_maximum_distance: 1.5
    loop_search_maximum_distance: 3.0
    do_loop_closing: true
    loop_match_minimum_chain_size: 10
```

Exercise

1. Simulate a robot with a laser scan, e.g. run the turtlebot simulation

    $ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py

2. Run your slam_toolbox launch file in a terminal

    $ros2 launch SLAM_launch.py

3. In another terminal, run the map server in order to save a version of the map when it is ready.

    $ ros2 run nav2_map_server map_saver_cli -f map

4. Run Rviz and configure the visualizations

    $ ros2 run rviz2 rviz2

5. Run rqt and teleoperate the robot around to build a map

    $ rqt

Tools

LOCALIZATION

Localization

```python
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():

    amcl_yaml = 'amcl_params.yaml'
    map_file = 'maps/map.yaml'

    return LaunchDescription([
        Node(
            package='nav2_map_server',
            executable='map_server',
            name='map_server',
            output='screen',
            parameters=[{'use_sim_time': True},
                        {'yaml_filename':map_file}]
        ),

        Node(
            package='nav2_amcl',
            executable='amcl',
            name='amcl',
            output='screen',
            parameters=[amcl_yaml]
        ),

        Node(
            package='nav2_lifecycle_manager',
            executable='lifecycle_manager',
            name='lifecycle_manager_localization',
            output='screen',
            parameters=[{'use_sim_time': True},
                        {'autostart': True},
                        {'node_names': ['map_server', 'amcl']}]
        )
    ])

```

Localization

```
amcl:
  ros__parameters:
    use_sim_time: True
    alpha1: 0.2
    alpha2: 0.2
    alpha3: 0.2
    alpha4: 0.2
    alpha5: 0.2
    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
    max_particles: 8000
    min_particles: 200
    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: "differential"
    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

```

Exercise

1. Run the simulation you used for the SLAM exercise.

   $ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py

2. Run the map server to publish the map that you created, together with amcl

   $ros2 launch amcl_launch_test.py

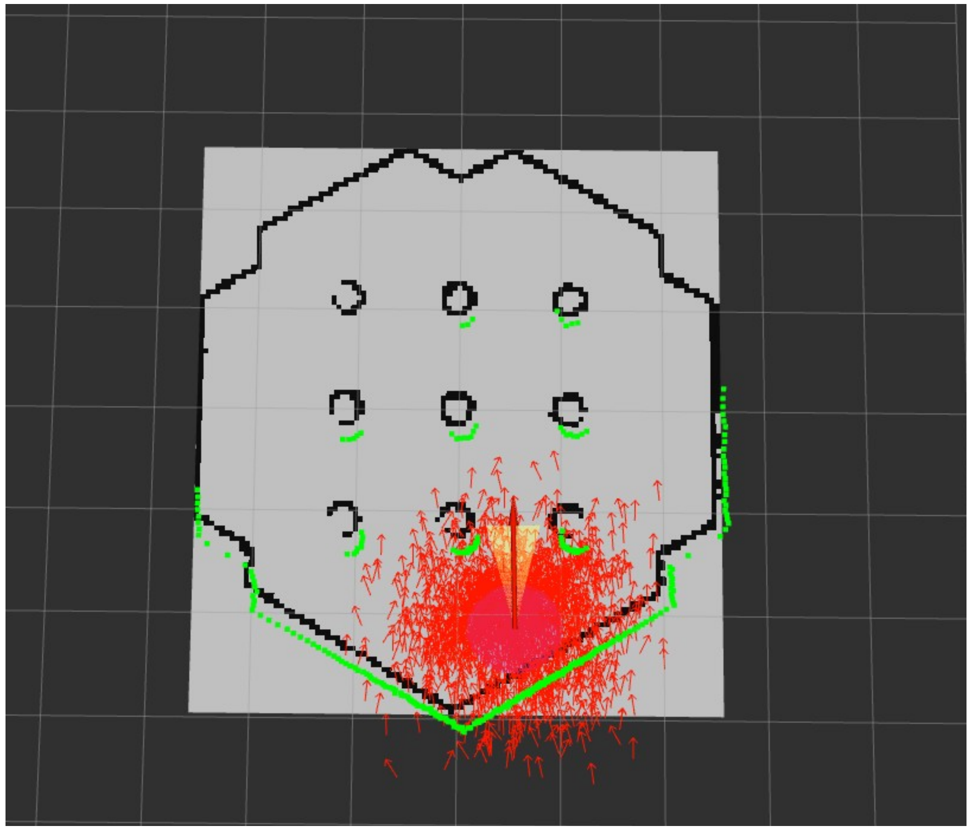
3. Run Rviz and configure the visualizations. You will need to change the **/particle_cloud** reliability policy to “**Best effort**” and the /**map** durability policy to “**Transient local**”. Alternatively, you may use the provided configuration file:

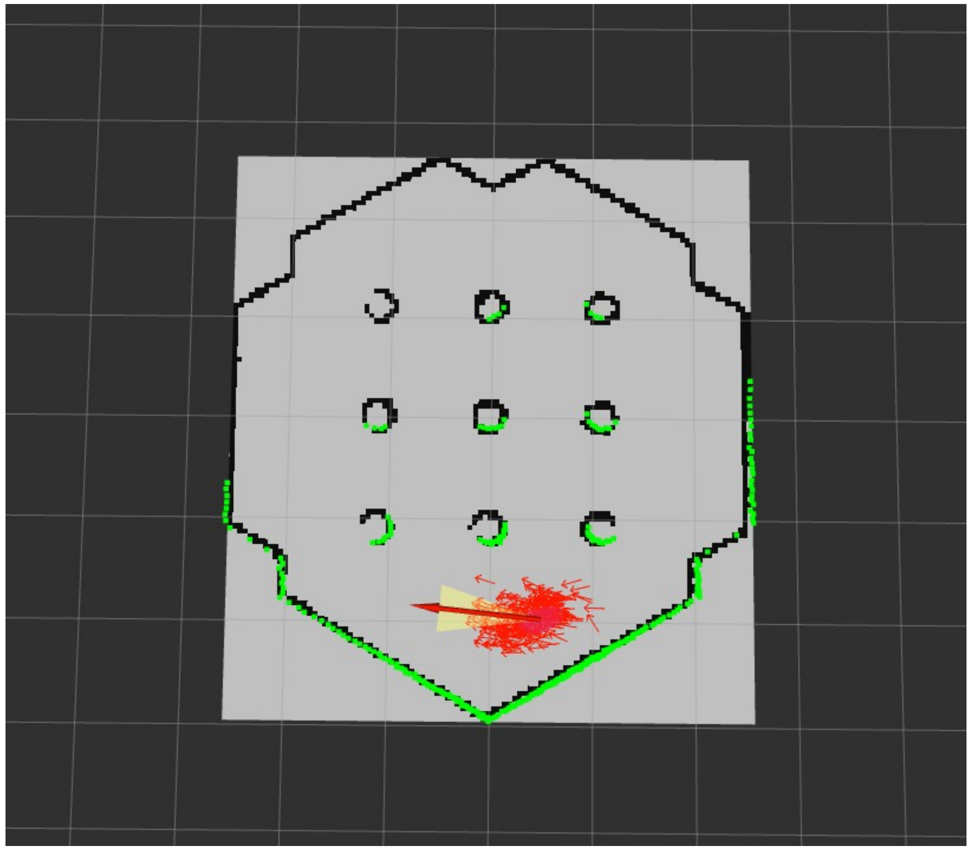
   $ros2 run rviz2 rviz2 -d amcl_rviz.rviz

4. Publish initial pose and covariance in RVIZ or from another node. Set a slightly wrong initial pose in Rviz, with high covariance, in a location with enough references

5. Rotate the robot and observe if the particles converge. It is nice to see if the laser scan matches the map

Exercise





Tools

NAV2

Nav2

```
declare_use_sim_time_cmd = DeclareLaunchArgument(
    'use_sim_time',
    default_value='false',
    description='Use simulation (Gazebo) clock if true')

declare_params_file_cmd = DeclareLaunchArgument(
    'params_file',
    default_value='my_nav2_params.yaml',
    description='Full path to the ROS2 parameters file to use for all launched nodes')

declare_autostart_cmd = DeclareLaunchArgument(
    'autostart', default_value='true',
    description='Automatically startup the nav2 stack')


declare_container_name_cmd = DeclareLaunchArgument(
    'container_name', default_value='nav2_container',
    description='the name of conatiner that nodes will load in if use composition')

declare_use_respawn_cmd = DeclareLaunchArgument(
    'use_respawn', default_value='False',
    description='Whether to respawn if a node crashes. Applied when composition is disabled.')

declare_log_level_cmd = DeclareLaunchArgument(
    'log_level', default_value='info',
    description='log level')

load_nodes = GroupAction(
    actions=[
        Node(
            package='nav2_controller',
            executable='controller_server',
            output='screen',
            respawn=use_respawn,
            respawn_delay=2.0,
            parameters=[configured_params],
            arguments=['--ros-args', '--log-level', log_level],
            remappings=remappings + [('cmd_vel', 'cmd_vel_nav')]),
        Node(
            package='nav2_smoother',
            executable='smoother_server',
            name='smoother_server',
            output='screen',
```

Nav2

```yaml
      - nav2_is_battery_charging_condition_bt_node

bt_navigator_navigate_through_poses_rclcpp_node:
  ros__parameters:
    use_sim_time: True

bt_navigator_navigate_to_pose_rclcpp_node:
  ros__parameters:
    use_sim_time: True

controller_server:
  ros__parameters:
    use_sim_time: True
    controller_frequency: 20.0
    min_x_velocity_threshold: 0.001
    min_y_velocity_threshold: 0.5
    min_theta_velocity_threshold: 0.001
    failure_tolerance: 0.3
    progress_checker_plugin: "progress_checker"
    goal_checker_plugins: ["general_goal_checker"] # "precise_goal_checker"
    controller_plugins: ["FollowPath"]

    # Progress checker parameters
    progress_checker:
      plugin: "nav2_controller::SimpleProgressChecker"
      required_movement_radius: 0.5
      movement_time_allowance: 10.0

    general_goal_checker:
      stateful: True
      plugin: "nav2_controller::SimpleGoalChecker"
      xy_goal_tolerance: 0.25
      yaw_goal_tolerance: 0.25
    # DWB parameters
    FollowPath:
      plugin: "dwb_core::DWBLocalPlanner"
      debug_trajectory_details: True
      min_vel_x: 0.0
      min_vel_y: 0.0
      max_vel_x: 0.26
      max_vel_y: 0.0
      max_vel_theta: 1.0
      min_speed_xy: 0.0
```

```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

smoother_server:
  ros__parameters:
    use_sim_time: True
    smoother_plugins: ["simple_smoother"]
    simple_smoother:
      plugin: "nav2_smoother::SimpleSmoother"
      tolerance: 1.0e-10
      max_its: 1000
      do_refinement: True

behavior_server:
  ros__parameters:
    costmap_topic: local_costmap/costmap_raw
    footprint_topic: local_costmap/published_footprint
    cycle_frequency: 10.0
    behavior_plugins: ["spin", "backup", "drive_on_heading", "assisted_teleop", "wait"]
    spin:
      plugin: "nav2_behaviors/Spin"
    backup:
      plugin: "nav2_behaviors/BackUp"
    drive_on_heading:
      plugin: "nav2_behaviors/DriveOnHeading"
    wait:
      plugin: "nav2_behaviors/Wait"
    assisted_teleop:
      plugin: "nav2_behaviors/AssistedTeleop"
    global_frame: odom
    robot_base_frame: base_link
    transform_tolerance: 0.1
```

Exercise

1. Run the simulation you used for the SLAM exercise.

   $ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py

2. Run the map server, amcl and nav2

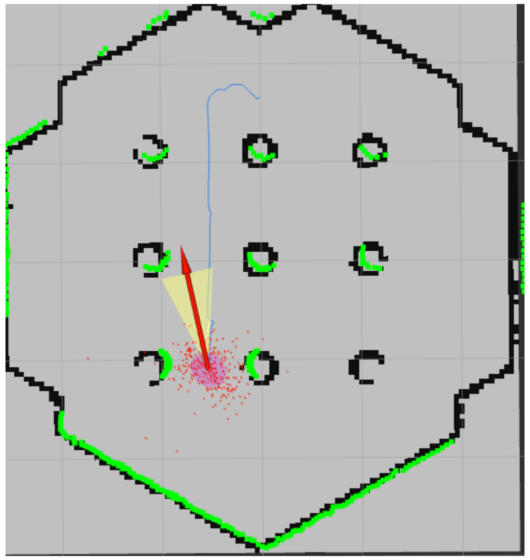
   $ros2 launch amcl_launch_test.py

   $ros2 launch nav2_launch_test.py

3. Run Rviz and configure the visualizations

   $ros2 run rviz2 rviz2

4. Publish initial pose and covariance in RVIZ or from another node. Publish a target pose. Visualize the paths.



ROS NAVIGATION

End of lesson