12. Navigation and obstacle avoidance

- 12. Navigation and obstacle avoidance
  - 12.1. Operation and use
    - 12.1.1. Start
    - 12.1.2. Use
    - 12.1.3. Dynamic parameter adjustment
    - 12.1.4. Node graph
    - 12.1.5, tf coordinate system
  - 12.2, navigation
    - 12.2.1. Introduction
    - 12.2.2. Setting tf
  - 12.3, move_base
    - 12.3.1. Introduction
    - 12.3.2, move_base communication mechanism
      - 1) Action
      - 2) topic
      - 3) servies
      - 4) Parameter configuration
    - 12.3.3. Recovery Behavior
      - 1 Introduction
      - 2) Related function packages
  - 12.4, costmap_params
    - 12.4.1, costmap_common
    - 12.4.2, global_costmap
    - 12.4.3, local_costmap
    - 12.4.4, costmap_2D
      - 1 Introduction
      - 2) topic
      - 3) Parameter configuration
      - 4) Layer specifications
      - 5)obstacle layer
      - 6) **Inflation** layer
  - 12.5, planner_params
    - 12.5.1, global_planner
    - 12.5.2, local_planner
      - 1) dwa_local_planner
      - 2) teb_local_planner

navigation: http://wiki.ros.org/navigation/

navigation/Tutorials: http://wiki.ros.org/navigation/Tutorials/RobotSetup

costmap_2d: http://wiki.ros.org/costmap_2d

nav_core: http://wiki.ros.org/nav_core

global_planner: http://wiki.ros.org/global_planner

dwa_local_planner: http://wiki.ros.org/dwa_local_planner

teb_local_planner: http://wiki.ros.org/teb_local_planner

move_base: http://wiki.ros.org/move_base

12.1. Operation and use

Note: [R2] on the remote control handle has the function of canceling the target point.

According to different models, you only need to set the purchased model in [.bashrc], X1 (normal four-wheel drive) X3 (Mailun) X3 as an example

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

Open the [.bashrc] file

```
sudo vim .bashrc
```

Find the [ROBOT_TYPE] parameters and modify the corresponding car model

```
export ROBOT_TYPE=X3 # ROBOT_TYPE: X1 X3 X3plus R2 X7
```

12.1.1. Start

Start the driver (robot side). For ease of operation, this section takes [mono + laser + yahboomcar] as an example.

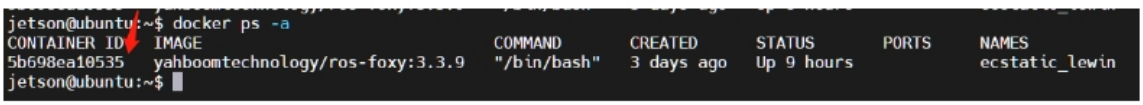
```
roslaunch yahboomcar_nav laser_bringup.launch # laser + yahboomcar
roslaunch yahboomcar_nav laser_usb_bringup.launch # mono + laser + yahboomcar
roslaunch yahboomcar_nav laser_astrapro_bringup.launch # Astra + laser +
yahboomcar
```

Start the navigation obstacle avoidance function (robot side), set parameters and modify the launch file according to needs.

<PI5 needs to open another terminal to enter the same docker container

1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

```
docker ps -a
```



2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```

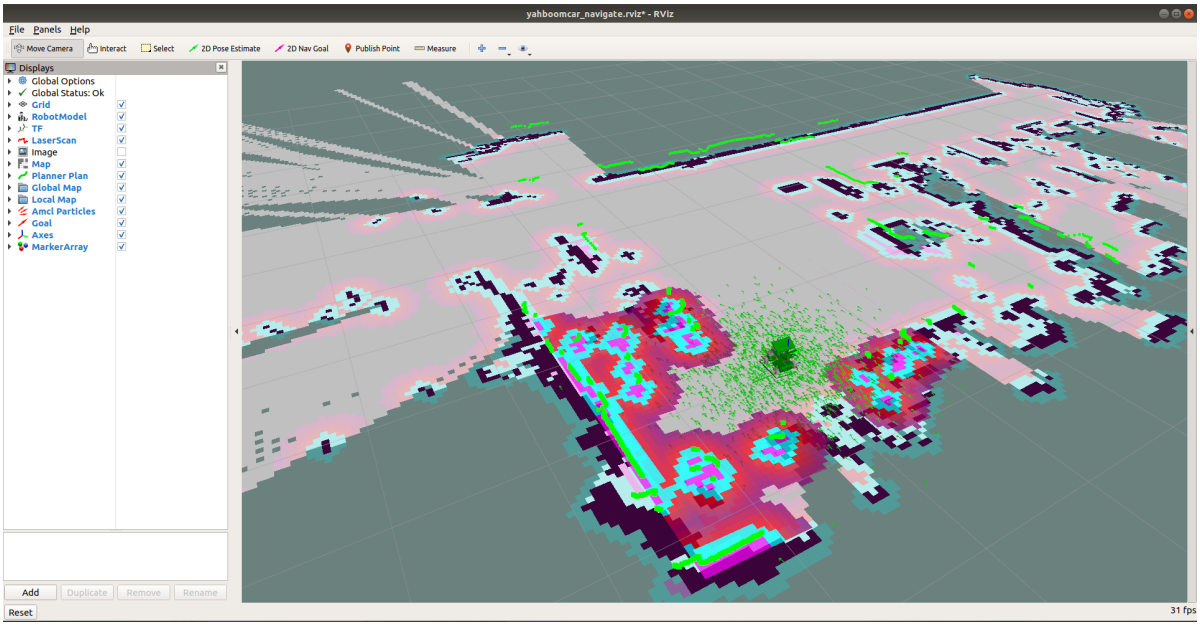
After successfully entering the container, you can open countless terminals to enter the container.

```
roslaunch yahboomcar_nav yahboomcar_navigation.launch use_rviz:=false map:=house
```

- [use_rviz] parameter: whether to open rviz.
- [map] parameters: map name, map to be loaded.

Turn on the visual interface (virtual machine side)

```
roslaunch yahboomcar_nav view_navigate.launch
```



12.1.2. Use

1. Single point navigation

- Use the [2D Pose Estimate] of the [rviz] tool to set the initial pose until the position of the car in the simulation is consistent with the position of the actual car.
- Click [2D Nav Goal] of the [rviz] tool, and then select a target point on the map where there are no obstacles. Release the mouse to start navigation. Only one target point can be selected, and it will stop when it is reached.

2. Multi-point navigation

- Same as the first step of single-point navigation, first set the initial pose of the car.
- Click [Publish Point] of the [rviz] tool, and then select the target point on the map where there are no obstacles. Release the mouse to start navigation. You can click [Publish Point] again, and then select the point, and the robot will click on it. Cruising between points.
- When using the [2D Pose Estimate] tool of the [rviz] tool to set the initial pose of the car, the multi-point navigation function is automatically canceled.

3. Parameter configuration

According to different models, you only need to set the purchased model in [.bashrc], X1 (normal four-wheel drive) X3 (Mailun) X3 as an example

Open the [.bashrc] file

```
sudo vim .bashrc
```

Find the [ROBOT_TYPE] parameters and modify the corresponding car model

```
export ROBOT_TYPE=X3 # ROBOT_TYPE: X1 X3 X3plus R2 X7
```

Looking at the yahboomcar_navigation.launch file, we can see that the navigation parameters are modified in the move_base.launch file under the yahboomcar_nav function package.

```
<launch>
     <!-- Whether to open rviz || Whether to open rviz -->
     <arg name="use_rviz" default="true"/>
     <!-- Map name || Map name my_map-->
     <arg name="map" default="my_map"/>
     <!-- MarkerArray node> -->
     <node name='send_mark' pkg="yahboomcar_nav" type="send_mark.py"/>
     <!-- Load map || Load map -->
     <node name="map_server" pkg="map_server" type="map_server" args="$(find
yahboomcar_nav)/maps/$(arg map).yaml"/>
     <!-- AMCL adaptive Monte Carlo positioning -->
     <include file="$(find yahboomcar_nav)/launch/library/amcl.launch"/>
     <!-- Mobile APP node -->
     <include file="$(find yahboomcar_nav)/launch/library/app.launch"/>
     <!-- Navigation core component move_base -->
     <include file="$(find yahboomcar_nav)/launch/library/move_base.launch"/>
     <!-- RVIZ -->
     <include file="$(find yahboomcar_nav)/launch/view/view_navigate.launch"
if="$(arg use_rviz)"/>
</launch>
```

Find the move_base.launch file and open the example file as follows. It can be modified and replaced according to your needs. At this time, [DWA Planner] is selected and the [DWA] file is loaded.

```
<launch>
     <arg name="robot_type" value="$(env ROBOT_TYPE)" doc="robot_type
[X1,X3,X3plus,R2,X7]"/>
     <!-- Arguments -->
     <arg name="move_forward_only" default="false"/>
     <!-- move_base -->
     <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
          <rosparam file="$(find
yahboomcar_nav)/param/common/global_costmap_params.yaml" command="load"/>
          <rosparam file="$(find
yahboomcar_nav)/param/common/local_costmap_params.yaml" command="load"/>
          <rosparam file="$(find
yahboomcar_nav)/param/common/move_base_params.yaml" command="load"/>
          <rosparam file="$(find
yahboomcar_nav)/param/common/costmap_common_params_$(arg robot_type).yaml"
command="load"
               ns="global_costmap"/>
          <rosparam file="$(find
yahboomcar_nav)/param/common/costmap_common_params_$(arg robot_type).yaml"
command="load"
               ns="local_costmap"/>
```

```
        <rosparam file="$(find
yahboomcar_nav)/param/common/dwa_local_planner_params_$(arg robot_type).yaml"
command="load"/>
        <param name="base_local_planner" type="string"
value="dwa_local_planner/DWAPlannerROS" if="$(eval arg('robot_type') == 'X3')"/>
        <!-- <param name="base_local_planner" type="string"
value="teb_local_planner/TebLocalPlannerROS"/>-->
        <param name="DWAPlannerROS/min_vel_x" value="0.0" if="$(arg
move_forward_only)"/>
        <remap from="cmd_vel" to="cmd_vel"/>
        <remap from="odom" to="odom"/>
    </node>
</launch>
```

Note: When using the DWA planner, the difference between an omnidirectional car and a differential car lies in whether the speed in the Y direction is 0. There are clear comments in it, which can be modified according to the actual situation.

Enter the dwa_local_planner_params.yaml file under the yahboomcar_nav function package. Some parameters are as follows:

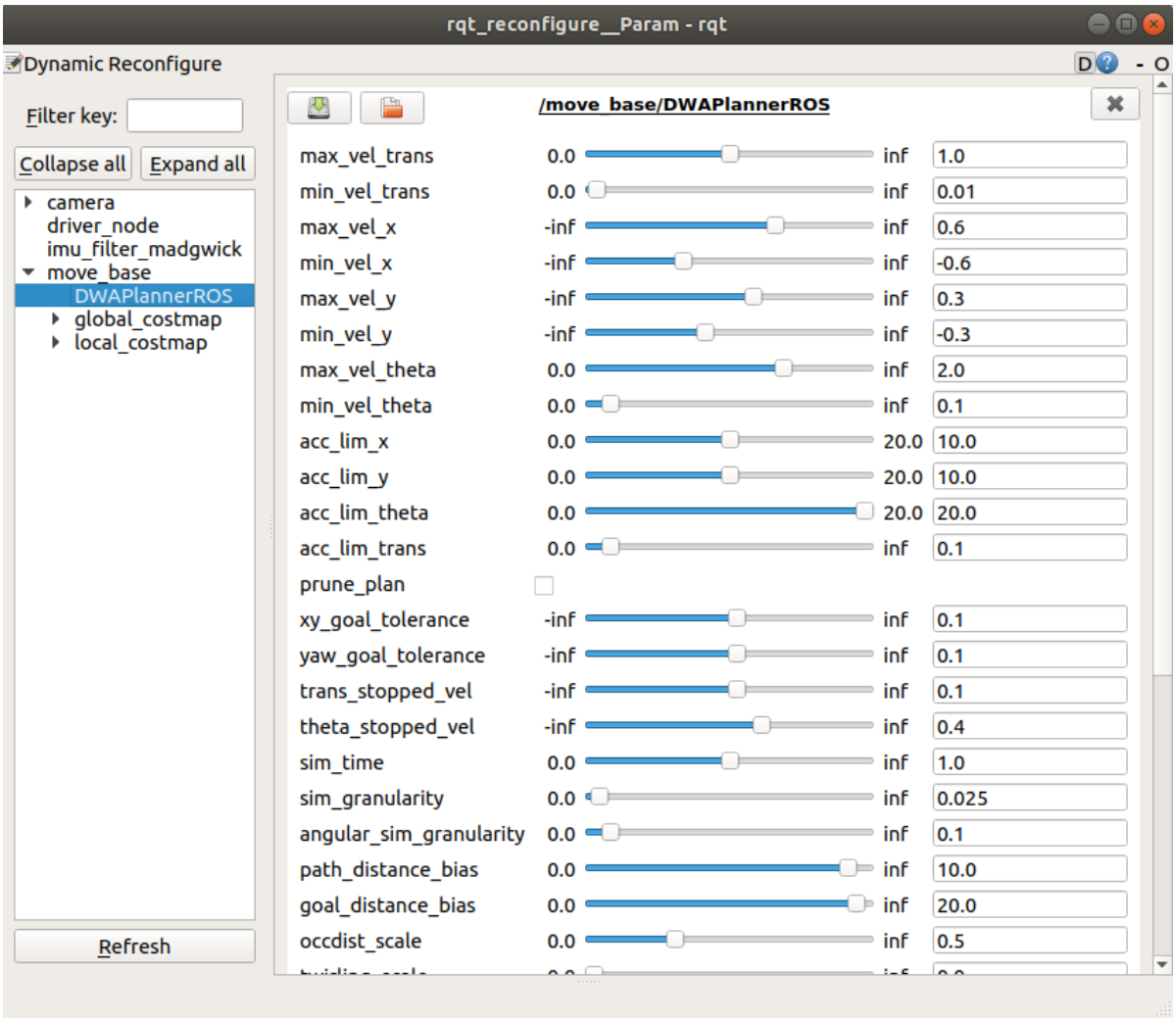
```
DWAPlannerROS:
   # Robot Configuration Parameters
   # Absolute value of maximum linear velocity in x direction, unit:
meters/second
   # The maximum y velocity for the robot in m/s
   max_vel_x: 0.6
   # The absolute value of the minimum linear velocity in the x direction, a
negative number means it can be retreated, unit: meters/second
   # The minimum x velocity for the robot in m/s, negative for backwards motion.
   min_vel_x: -0.6
   # Absolute value of the maximum linear velocity in the y direction, unit:
meters/second. The differential robot is 0
   # The maximum y velocity for the robot in m/s
   max_vel_y: 0.3
   # The absolute value of the minimum linear velocity in the y direction, unit:
meters/second. The differential robot is 0
   # The minimum y velocity for the robot in m/s
   min_vel_y: -0.3
... ...
   #The ultimate acceleration of the robot in the x direction, the unit is
meters/sec^2
   # The x acceleration limit of the robot in meters/sec^2
   acc_lim_x: 10.0
   # The ultimate acceleration of the robot in the y direction, which is 0 for
differential robots
   # The y acceleration limit of the robot in meters/sec^2
   acc_lim_y: 10.0
... ..
```

Other parameter files can be opened, combined with comments and courseware, and modified according to your own needs.

12.1.3. Dynamic parameter adjustment

After starting the navigation function, open the dynamic parameter adjustment tool, adjust according to your own needs, and observe the robot's motion status until the effect is optimal. Record the current parameters and modify them to the dwa_local_planner_params.yaml file under the yahboomcar_nav function package.

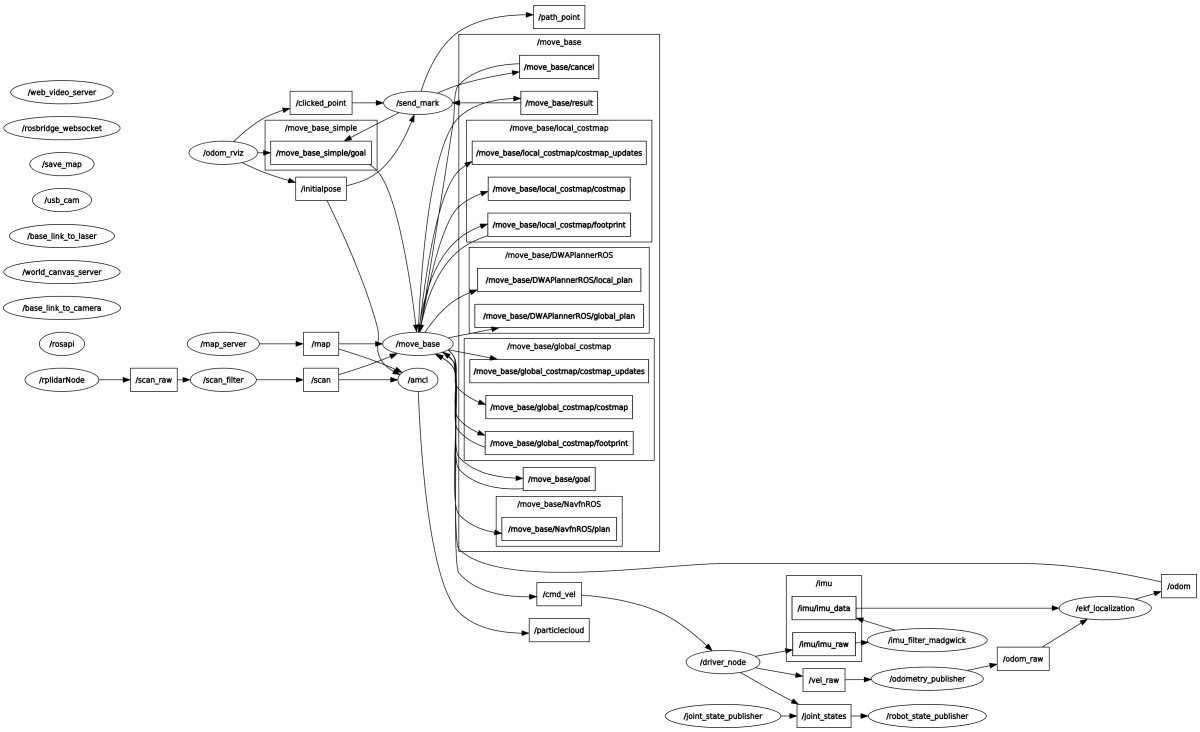
```
rosrun rqt_reconfigure rqt_reconfigure
```



12.1.4. Node graph

Take starting [mono + laser + yahboomcar] in section [1.1] as an example to observe the communication between nodes.

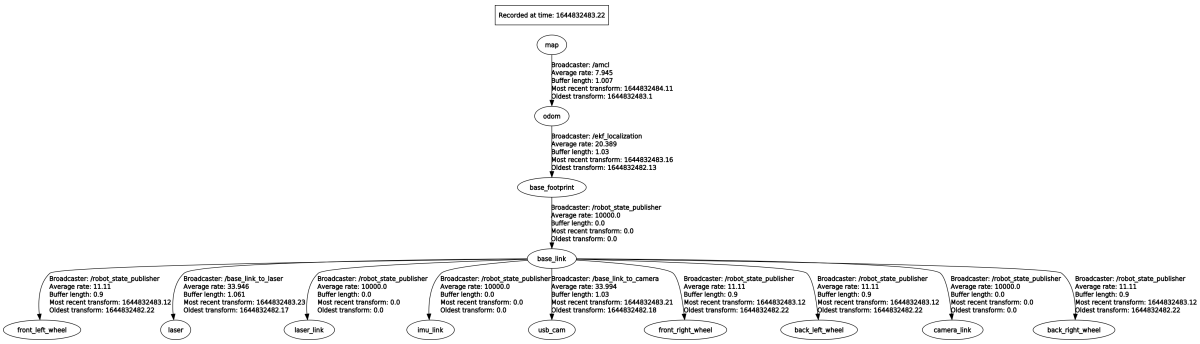
```
rosrun rqt_graph rqt_graph
```



12.1.5, tf coordinate system

The conversion relationship between coordinates.

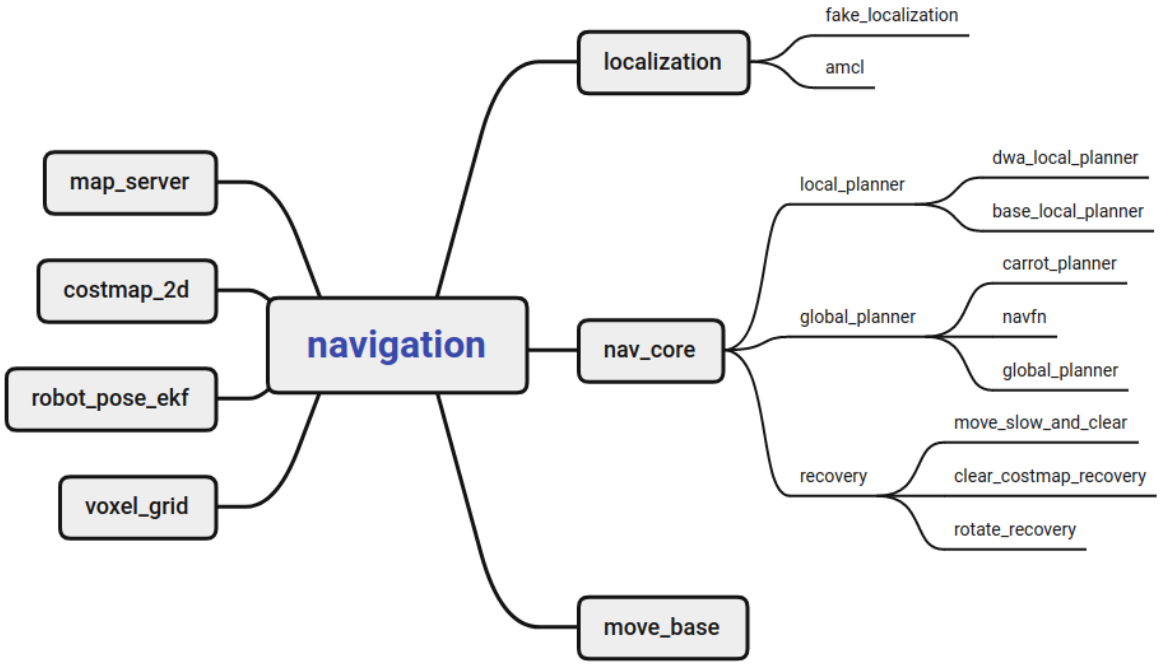
```
rosrun rqt_tf_tree rqt_tf_tree
```



12.2, navigation

12.2.1. Introduction

Navigation is the two-dimensional navigation and obstacle avoidance function package of ROS. Simply put, it calculates safe and reliable robot speed control instructions through the navigation algorithm based on the information flow from the input odometer and other sensors and the global position of the robot.

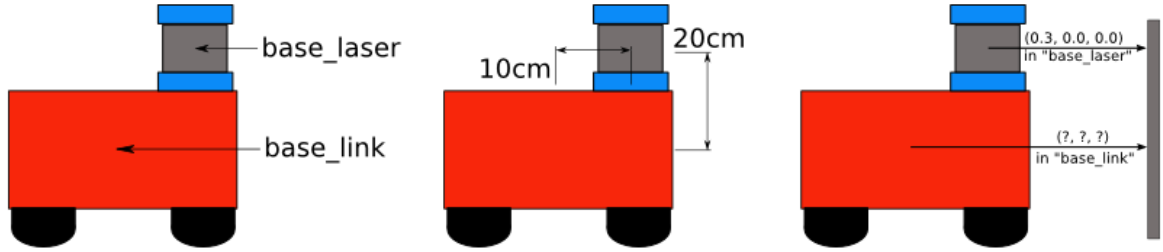


navigation main nodes and configuration

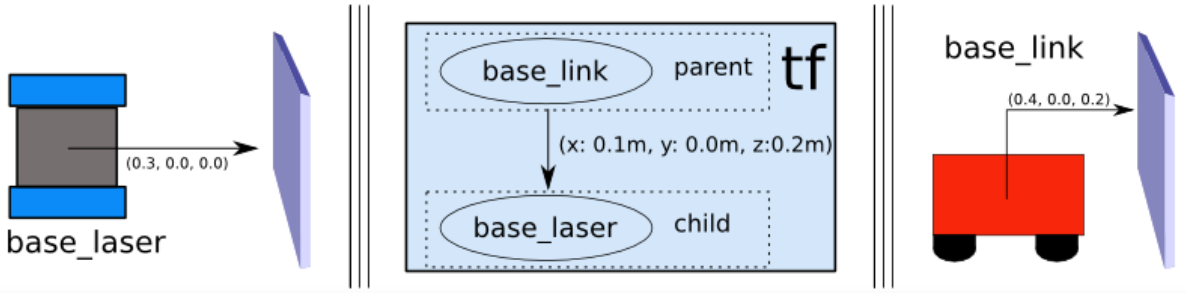
- move_base: The final execution mechanism of navigation obstacle avoidance control. move_base subscribes to the navigation target move_base_simple/goal and publishes the motion control signal cmd_vel in real time. Various navigation algorithm modules in move_base are called in the form of plug-ins.
- global_planner: used for global path planning.
- local_planner: used for local path planning.
- global_costmap: The global cost map is used to describe global environment information.
- local_costmap: Local cost map is used to describe local environment information.
- recovery_behaviors: The recovery strategy is used for the robot to automatically escape and recover after encountering obstacles.
- amcl: Use particle filter algorithm to realize the global positioning of the robot and provide global position information for robot navigation.
- map_server: The map obtained by calling SLAM mapping provides environmental map information for navigation.
- costmap_2d: can produce cost maps and provide various related functions.
- robot_pose_ekf: Extended Kalman filter, the input is any two or three of odometry, IMU, and VO, and the output is a fused pose.
- fake_localization: Generally used for simulation.
- nav_core: There are only three files in it, which correspond to the general interface definitions of global path planning, local path planning, and recovery_action. The specific function implementation is in each corresponding planner function package.
- You also need to provide tf information, odometer odom information, and lidar scan information related to the robot model.

12.2.2. Setting tf

Navigation functionality requires the robot to use tf to publish information about the relationships between coordinate systems. For example: lidar



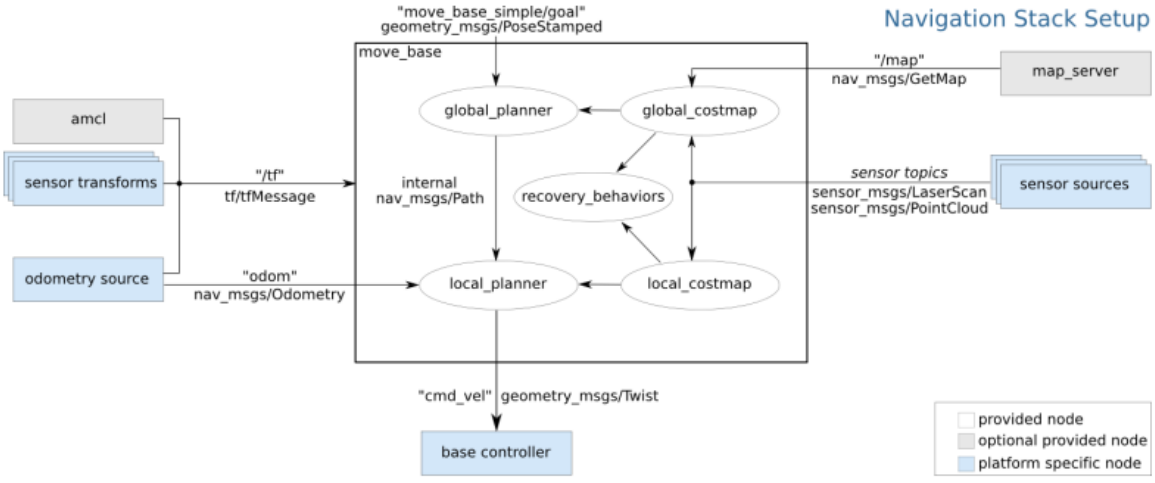
Suppose we know that the lidar is mounted 10 cm and 20 cm above the center point of the mobile base. This gives us the translation offset that relates the "base_link" frame to the "base_laser" frame. Specifically, we know that to get data from the "base_link" coordinate system to the "base_laser" coordinate system, we have to apply a translation of (x: 0.1m, y: 0.0m, z: 0.2m) and get the data from the "base_laser" frame To the "base_link" frame we have to apply the opposite translation (x: -0.1m, y: 0.0m, z: -0.20m).



12.3, move_base

12.3.1. Introduction

move_base provides configuration, operation, and interactive interfaces for ROS navigation.



Implementing the robot navigation function must be configured in a specific way, as shown above:

- White components are required components that have been implemented,
- Gray components are optional components that have been implemented,
- Blue components must be created for each robot platform.

12.3.2, move_base communication mechanism

1) Action

The move_base node provides an implementation of SimpleActionServer that receives targets containing geometry_msgs/PoseStamped messages. You can communicate directly with the move_base node via ROS, but if you care about tracking the status of the target, it is recommended to use SimpleActionClient to send the target to move_base. (See actionlib documentation for more information)

Name	Type	Description
move_base/goal	move_base_msgs/MoveBaseActionGoal	move_base subscribes to the target point to be reached.
move_base/cancel	actionlib_msgs/GoalID	move_base subscribes to requests to cancel a specific goal.
move_base/feedback	move_base_msgs/MoveBaseActionFeedback	Publish the current position containing the chassis.
move_base/status	actionlib_msgs/GoalStatusArray	Publish status information of the process of moving to the target point.
move_base/result	move_base_msgs/MoveBaseActionResult	Publish the final result of the move.

2) topic

Name	Type	Description
move_base_simple/goal	geometry_msgs/PoseStamped	Provides a non-action interface for not paying attention to the execution status of the tracking goal. move_base subscribes to the target point that will be reached.
cmd_vel	geometry_msgs/Twist	Publish the moving speed of the car.

3) servies

Name	Type	Description
make_plan	nav_msgs/GetPlan	Allows an external user to request a plan for a given pose from move_base without causing move_base to execute the plan.
clear_unknown_space	std_srvs/Empty	Allows external users to notify move_base to clear unknown space in the area around the robot. This is useful when the costmaps of move_base are stopped for a long period of time and then restarted at a new location in the environment.
clear_costmaps	std_srvs/Empty	Allows external users to tell move_base to clear barriers in the costmaps used by move_base. This may cause the robot to bump into things and should be used with caution

4) Parameter configuration

move_base_params.yaml

```
#Set the plugin name of the global path planner of move_base
#base_global_planner: "navfn/NavfnROS"
base_global_planner: "global_planner/GlobalPlanner"
#base_global_planner: "carrot_planner/CarrotPlanner"

#Set the plug-in name of the local path planner of move_base
#base_local_planner: "teb_local_planner/TebLocalPlannerROS" # Implement an online
optimized local trajectory planner
base_local_planner: "dwa_local_planner/DWAPlannerROS" # Implement DWA (dynamic
window method) local planning algorithm
# Restore behavior.
recovery_behaviors:
   - name: 'conservative_reset'
     type: 'clear_costmap_recovery/ClearCostmapRecovery'
   #- name: 'aggressive_reset'
   # type: 'clear_costmap_recovery/ClearCostmapRecovery'
   #- name: 'super_reset'
   # type: 'clear_costmap_recovery/ClearCostmapRecovery'
   - name: 'clearing_rotation'
     type: 'rotate_recovery/RotateRecovery'
     #- name: 'move_slow_and_clear'
     #type: 'move_slow_and_clear/MoveSlowAndClear'

# Frequency of sending commands to robot chassis cmd_velcontroller_frequency:
10.0
# The time the path planner waits for a valid control command before executing
the space cleanup operation.
planner_patience: 5.0
```

```
# The time the controller waits for a valid control command before executing the
space cleanup operation
controller_patience: 15.0
# This parameter is only used if the default recovery behavior is used with
move_base.
conservative_reset_dist: 3.0
# Whether to enable move_base recovery behavior to try to clear space.
recovery_behavior_enabled: true
# Whether the robot uses in-situ rotation to clean up the space. This parameter
is only used when using the default recovery behavior.
clearing_rotation_allowed: true
# When move_base enters the inactive state, whether to deactivate the node's
costmap
shutdown_costmaps: false
# The time allowed for flapping before performing the recovery operation, 0 means
never timeout
oscillation_timeout: 10.0
# The robot needs to move this distance to be considered as having no vibration.
Reset timer parameters after moving
oscillation_distance: 0.2
# Global path planner cycle rate.
planner_frequency: 5.0
#The number of planned retries allowed before performing recovery actions. A
value of -1.0 corresponds to unlimited retries.
max_planning_retries: -1.0
```

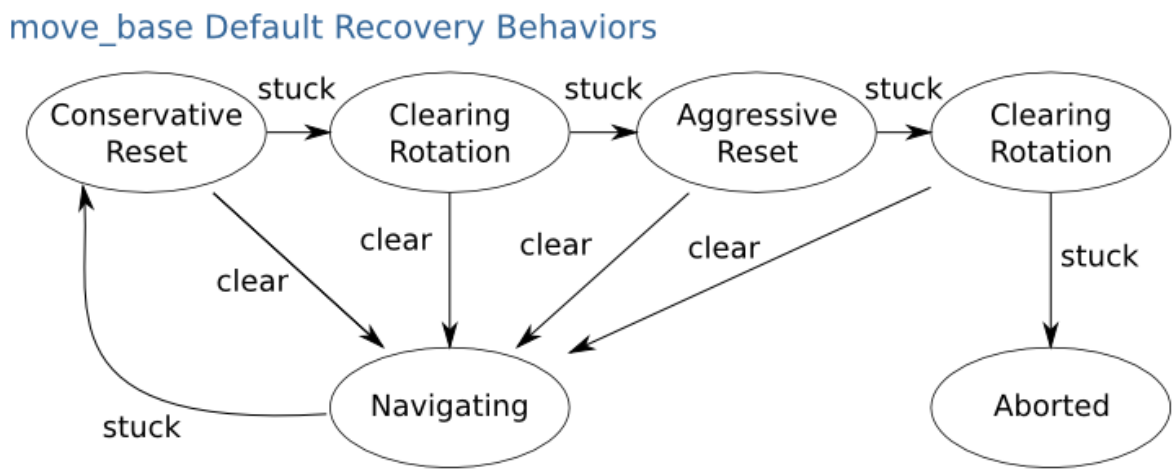
12.3.3. Recovery Behavior

1 Introduction

When ① the global planning fails, ② the robot oscillates, and ③ the local planning fails, it will enter recovery behavior. These recovery behaviors can be configured using the recovery_behaviour parameter and disabled using the recovery_behavior_enabled parameter.

Desired robot behavior

First, obstacles outside the user-specified area are cleared from the robot's map. Next, if possible, the robot will perform a spin to clear space. If this also fails, the robot will clear the map more aggressively, clearing all obstacles outside of a rectangular area within which the robot can rotate in place. This will be followed by another spin in place. If all else fails, the bot considers its goal to be infeasible and notifies the user that it has aborted.

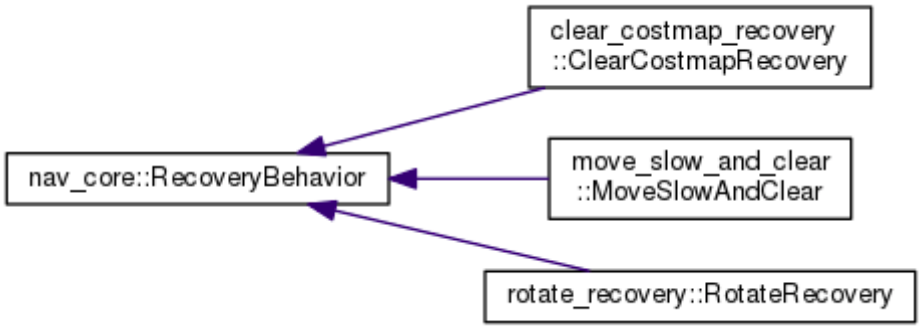


- conservative reset: Conservative recovery.

- clearing rotation: rotation clearing.
- aggressive reset: aggressive recovery.
- aborted: aborted.

2) Related function packages

In the navigation function package collection, there are 3 packages related to the recovery mechanism. They are: clear_costmap_recovery, move_slow_and_clear, rotate_recovery. Three classes are defined in these three packages, all of which inherit the interface specifications in nav_core.



- move_slow_and_clear: It is a simple recovery behavior that clears the information in the cost map , and then limit the speed of the robot. Note that this recovery behavior is not truly safe, the robot may hit objects, it will only happen at a user-specified speed.

Parameters	Type	Default value	Parse
clearing_distance	double	0.5	Obstacles within the robot's clearing distance will be cleared (unit: m).
limited_trans_speed	double	0.25	When performing this recovery behavior, the robot's translation speed will be limited (unit: m/s).
limited_rot_speed	double	0.25	When performing this recovery behavior, the robot's rotational speed will be limited (unit: rad/s).
limited_distance	double	0.3	The distance (unit: m) that the robot must move before the speed limit is released.
planner_namespace	string	"DWAPlannerROS"	The name of the planner whose parameters are to be reconfigured.

- rotate_recovery: Rotation recovery, clearing space by rotating the robot 360 degrees.

Parameters	Type	Default value	Parse
sim_granularity	double	0.017	When checking whether it is safe to rotate in place, the distance between obstacles is checked to be 1 degree (unit: rad) by default.
frequency	double	20.0	The frequency of sending speed commands to the mobile robot (unit: HZ).
TrajectoryPlannerROS/yaw_goal_tolerance	double	0.05	The controller's arc tolerance in yaw/rotation when achieving its goal.
TrajectoryPlannerROS/acc_lim_th	double	3.2	The rotational acceleration limit of the robot (unit: rad/s^2).
TrajectoryPlannerROS/max_rotational_vel	double	1.0	The maximum rotation speed allowed by the base (unit: rad/s).
TrajectoryPlannerROS/min_in_place_rotational_vel	double	0.4	The minimum rotation speed allowed by the base when performing in-place rotation (unit: rad/s).

Note: The TrajectoryPlannerROS parameter is only set when using the base_local_planner::TrajectoryPlannerROS planner; generally no setting is required.

- clear_costmap_recovery: A recovery behavior that restores the cost map used by move_base to a static map outside the user-specified range.

Parameters	Type	Default value	Parse
clearing_distance	double	0.5	The length, centered on the robot, that the obstacle will be removed from the cost map when it reverts to a static map.

12.4, costmap_params

The navigation function uses two cost maps to store obstacle information. One cost map is used for global planning, which means creating globally planned routes throughout the environment, and the other is used for local planning and obstacle avoidance. The two costmaps have some common configurations and some separate configurations. Therefore, the cost map configuration has the following three parts: general configuration, global configuration and local configuration.

12.4.1, costmap_common

Cost map public parameter configuration costmap_common_params.yaml

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
# robot_radius: ir_of_robot
inflation_radius: 0.55
observation_sources: laser_scan_sensor
laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic:
topic_name, marking: true, clearing: true}
```

Parameter analysis

- obstacle_range: The default value is 2.5 meters, which means that the robot will only update information about obstacles within a range of 2.5 meters.
- raytrace_range: The default value is 3.0 meters, meaning the robot will try to clear space beyond 3.0 meters in front of it.
- Footprint: Set the occupied area of the robot. Fill in the "footprint" according to the robot coordinate setting. When the footprint is specified, the center of the robot is considered to be at (0.0, 0.0). Both clockwise and counterclockwise settings are available.
- robot_radius: The occupied area of the robot is a circle, just set the radius directly. Not shareable with footprint.
- inflation_radius: Set the cost map inflation radius. Default is 0.55. This means that the robot will regard a range of 0.55 meters from the obstacle as an obstacle.
- observation_sources: defines a list of sensors that pass information to a cost map separated by spaces.
- laser_scan_sensor: defines each sensor.
    - sensor_frame: The name of the sensor coordinate system.
    - data_type: parameter is set to LaserScan or PointCloud, depending on the message used by the topic.
    - topic: The topic name of the sensor published data.
    - marking: whether to add obstacle information to the cost map.

- clearing: whether to clear obstacle information in the cost map.

12.4.2, global_costmap

Global cost map parameter configuration global_costmap_params.yaml

```
global_costmap:
   global_frame: map
   robot_base_frame: base_link
   update_frequency: 5.0
   static_map: true
```

- global_frame: The coordinate system in which the global cost map operates.
- robot_base_frame: The robot base coordinate system referenced by the global cost map. -update_frequency: Frequency of global cost map cyclic update, unit is Hz.
- static_map: Whether the global cost map should be initialized based on the map provided by map_server. If you are not using an existing map or map server, set the static_map parameter to false.

12.4.3, local_costmap

Local cost map parameter configuration local_costmap_params.yaml

```
local_costmap:
   global_frame: odom
   robot_base_frame: base_link
   update_frequency: 5.0
   publish_frequency: 2.0
   static_map: false
   rolling_window: true
   width: 6.0
   height: 6.0
   resolution: 0.05
```

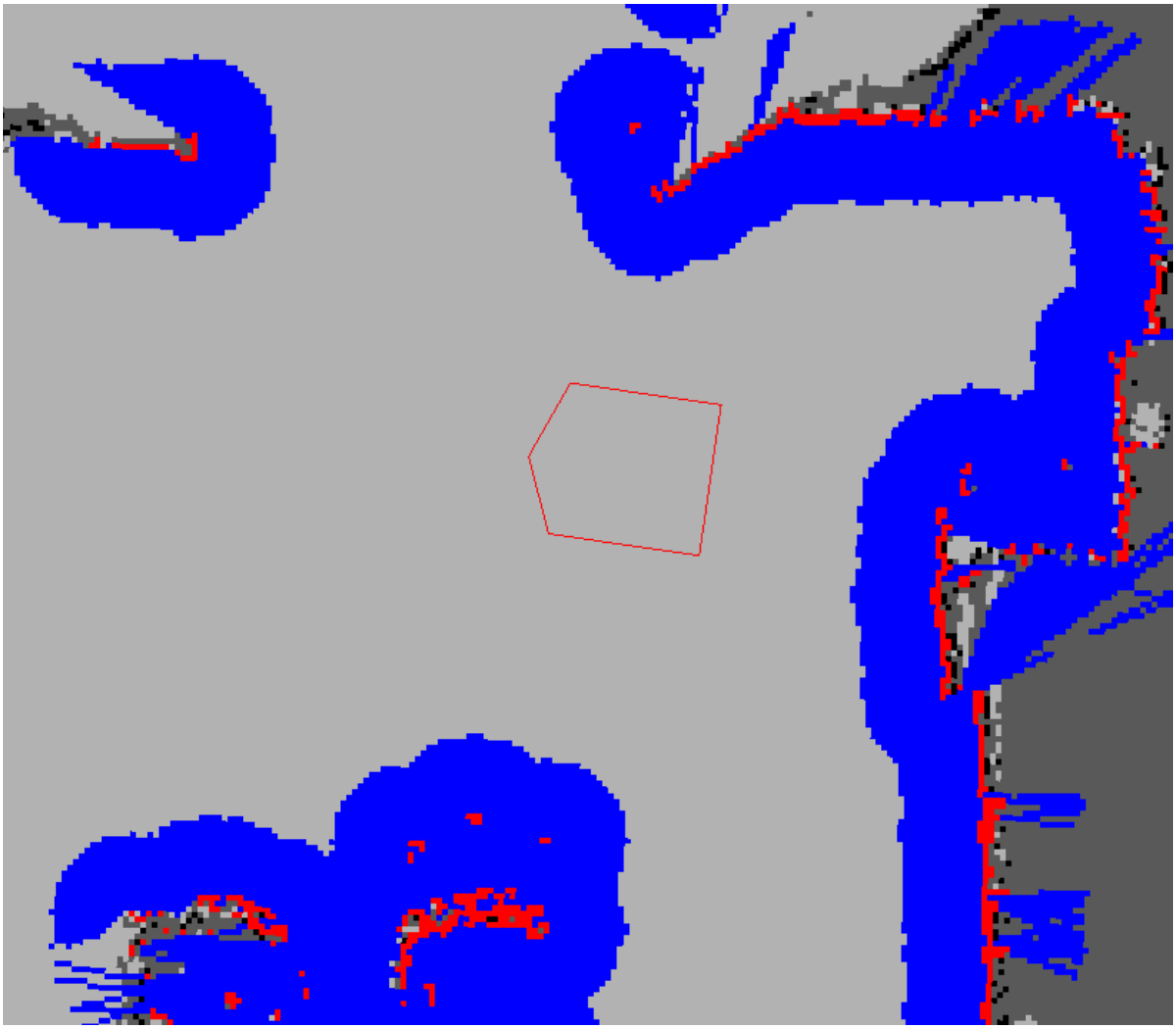
Parameter analysis

- global_frame: The coordinate system in which the local cost map operates.
- robot_base_frame: The robot base coordinate system referenced by the local cost map.
- update_frequency: The frequency at which the local cost map is updated cyclically, in Hz.
- publish_frequency: The rate at which the local cost map publishes visual information, in Hz.
- static_map: Whether the local cost map should be initialized based on the map provided by map_server. If you are not using an existing map or map server, set the static_map parameter to false.
- rolling_window: The (rolling window) parameter set to true means that when the robot is moving, the local cost map will remain centered on the robot.
- width: local cost map width (meters).
- height: height of local cost map (meters).
- resolution: local cost map resolution (meters/unit).

12.4.4, costmap_2D

1 Introduction

The costmap_2d package provides a 2D cost map implementation that uses input sensor data to construct a data 2D or 3D cost map (depending on whether a voxel-based implementation is used) and generates data based on the occupancy grid and user-defined dilation. The radius calculates the cost of the 2D cost map.



- Red represents obstacles in the cost map.
- Blue indicates obstacles with an expanding radius within the robot,
- Red polygons represent the robot's footprints.
- In order for the robot to avoid collisions, the robot's shell is never allowed to intersect with the red cells, and the robot's center point is never allowed to intersect with the blue cells.

2) topic

Name	Type	Description
footprint	geometry_msgs/Polygon	Robot shell specification. This replaces the previous parameter specification for the package outline.
costmap	nav_msgs/OccupancyGrid	costmap

Name	Type	Description
costmap_updates	map_msgs/OccupancyGridUpdate	The update area of the cost map
voxel_grid	costmap_2d/VoxelGrid	voxel grid

3) Parameter configuration

If you do not provide the plugins parameter, the initialization code will assume that your configuration is pre-Hydro and the default namespaces are static_layer, obstacle_layer, and inflation_layer.

plug-in

- plugins: generally use the default.

Coordinate system and tf parameters

- global_frame: the global coordinate system in which the cost map runs.
- robot_base_frame: The coordinate system name of the robot base_link.
- transform_tolerance: Specifies the tolerable transformation (tf) data delay (unit: s).

Rate parameters

- update_frequency: frequency of updating the map (unit: Hz).
- publish_frequency: The frequency (unit: Hz) of publishing the map displaying the information.

Map management parameters

- rolling_window: Whether to use the rolling window version of the cost map. If the static_map parameter is set to true, this parameter must be set to false.
- always_send_full_costmap: If true, every update will publish the full costmap to "/costmap". If false, only changed costmap parts will be published on the "/costmap_updates" topic.

static layer

- width: the width of the map (unit: m).
- height: the height of the map (unit: m).
- resolution: map resolution (unit: m/cell).
- origin_x: x origin of the map in the global frame (unit: m).
- origin_y: The y origin of the map in the global frame (unit: m).

tf transform

global_frame——>robot_base_frame

4) Layer specifications

- Static layer static map layer: Static layers are basically unchanged in the cost map.

  Subscribe to topics

  - map: The cost map will make a service call to map_server to obtain this map.

  parameter

    - unknown_cost_value: This value is read from the map provided by the map server, and its cost will be treated as unknown. A value of zero also causes this parameter to be unused.
    - lethal_cost_threshold: The threshold to consider lethal when reading maps from the map server.
    - map_topic: Specifies the topic that the cost map uses to subscribe to the static map.
    - first_map_only: Only subscribe to the first message on the map topic and ignore all subsequent messages.
    - subscribe_to_updates: In addition to map_topic, also subscribe to map_topic + "_updates".
    - track_unknown_space: If true, unknown values in the map message will be converted directly to the layer. Otherwise, unknown values in the map message will be converted to free space in the layer.
    - use_maximum: only matters if the static layer is not the bottom layer. If true, only the maximum value is written to the main cost chart.
    - trinary_costmap: If true, convert all map message values to NO_INFORMATION/FREE_SPACE/LETHAL_OBSTACLE (three values). If false, the full range of intermediate values is possible.
- Obstacle layer obstacle layer: The obstacle layer tracks obstacles read by sensor data. The collision costmap plugin labels and raytraces obstacles in 2D, while the VoxelCostmapPlugin labels and raytraces obstacles in 3D.
- Inflation layer inflation layer: Add new values (i.e. inflation obstacles) around lethal obstacles so that the cost map represents the robot's configuration space.
- Other layers: Other layers can be implemented and used in cost maps via pluginlib.
    - Social Costmap Layer
    - Range Sensor Layer

5)obstacle layer

The obstacle and voxel layers contain information from sensors in the form of point clouds or laser scans. Barrier layers are tracked in two dimensions, while voxel layers are tracked in three dimensions.

The cost map automatically subscribes to the sensor topic and updates accordingly. Each sensor is used for marking (inserting obstacle information into the cost map) and clearing (removing obstacle information from the cost map). Each time the data is observed, the clear operation performs a ray trace from the sensor origin outward through the mesh. In the voxel layer, the obstacle information in each column is projected down into a 2D map.

Subscribe to topics

Topic name	Type	Analysis
point_cloud_topic	sensor_msgs/PointCloud	Update PointCloud information to the cost map.
point_cloud2_topic	sensor_msgs/PointCloud2	Update PointCloud2 information to cost map

Topic name	Type	Analysis
laser_scan_topic	sensor_msgs/LaserScan	Update LaserScan information to cost map
map	nav_msgs/OccupancyGrid	Cost map with option to initialize from user generated static map

Sensor management parameters

- observation_sources: list of observation source names

  Each source name in an observation source defines a namespace in which parameters can be set:

  - <source_name>/topic: Topics covered by sensor data.
  - <source_name>/sensor_frame: sensor. Can be sensor_msgs/LaserScan, sensor_msgs/PointCloud and sensor_msgs/PointCloud2.
  - <source_name>/observation_persistence: Time to maintain each sensor reading (unit: s). A value of 0.0 will keep only the most recent readings.
  - <source_name>/expected_update_rate: Frequency of sensor readings (unit: s). A value of 0.0 will allow unlimited time between readings.
  - <source_name>/data_type: The data type associated with the topic. Currently, only "PointCloud", "PointCloud2" and "LaserScan" are supported.
  - <source_name>/clearing: Whether this observation should be used to clear free space.
  - <source_name>/marking: Whether this observation should be used to mark obstacles.
  - <source_name>/max_obstacle_height: Maximum height in m for sensor readings to be considered valid. This is usually set slightly higher than the height of the robot.
  - <source_name>/min_obstacle_height: Minimum height (in m) for sensor readings to be considered valid. This is usually set to ground level, but can be set higher depending on the sensor's noise model.High or lower.
  - <source_name>/obstacle_range: The maximum range (unit: m) at which obstacles are inserted into the cost map using sensor data.
  - <source_name>/raytrace_range: Maximum range (in m) of ray tracing obstacles from the map using sensor data.
  - <source_name>/inf_is_valid: Allows input of Inf values in "laser scan" observation information. Inf value converted to laser maximum range

Global filtering parameters: These parameters apply to all sensors.

- max_obstacle_height: The maximum height (unit: m) of any obstacle to be inserted into the cost map. This parameter should be set slightly higher than the height of the robot.
- obstacle_range: The default maximum distance from the robot when inserting obstacles into the cost map (unit: m). This may be overused on a per-sensor basis.
- raytrace_range: Default range (unit: m) for ray tracing obstacles from the map using sensor data. This may be overused on a per-sensor basis.

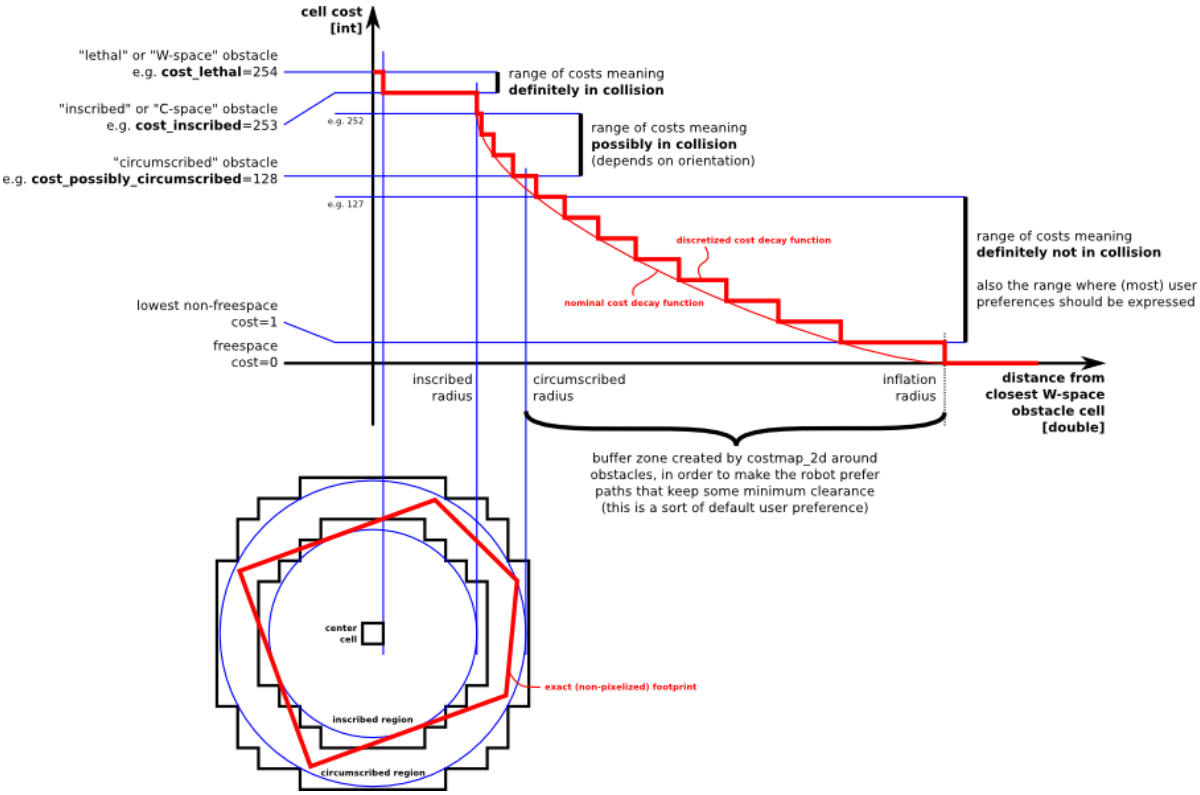
ObstacleCostmapPlugin

- track_unknown_space: If false, each pixel has one of two states: fatal obstacle or free. If true, each pixel has one of three states: fatal obstacle, free, or unknown.
- footprint_clearing_enabled: If true, the robot footprint will clear (mark as free) the space it moves in.
- combination_method: Changes how the barrier layer handles incoming data from layers outside it. Possible values are Override (0), Maximum (1), and None (99).

VoxelCostmapPlugin

- origin_z: The z origin of the map (unit: m).
- z_resolution: z resolution of the map (unit: m/cell).
- z_voxels: number of voxels in each vertical column, raster height is z_resolution * z_voxels
- unknown_threshold: The number of unknown cells in the column that are considered "known"
- mark_threshold: Maximum number of marked cells allowed in a column to be considered "free".
- publish_voxel_map: Whether to publish the underlying voxel raster for visualization purposes.
- footprint_clearing_enabled: If true, the robot footprint will clear (mark as free) the space it moves in.

6) Inflation layer



The inflation cost decreases as the robot's distance from the obstacle increases. Define 5 robot-specific symbols for the cost map's cost value.

- "Lethal" cost: indicates that there is a real obstacle in the cell. If the center of the robot is in this cell, the robot will inevitably collide with the obstacle.

- "Inscribed" cost: This means that the distance between the cell and the obstacle is less than the radius of the robot's inscribed circle. If the center of the robot is located in a cell equal to or higher than the "Inscribed" cost, the robot will inevitably collide with the obstacle.
- "Possibly circumscribed" cost: indicates that the distance of a cell from the obstacle is less than the radius of the robot's circumscribed circle, but greater than the radius of the inscribed circle. If the center of the robot is at a cell equal to or higher than the "Possibly circumscribed" cost, the machine will not necessarily collide with an obstacle, depending on the robot's orientation.
- "Freespace": nothing prevents the robot from going there.
- Unknown ("Unknown"): unknown space.

parameter

- inflation_radius: The radius (unit: m) to which the map inflates the obstacle cost.
- cost_scaling_factor: Scaling factor applied to cost values during inflation.

12.5, planner_params

12.5.1, global_planner

nav_core::BaseGlobalPlanner provides an interface to the global planner used in navigation. All global planners written as move_base node plugins must comply with this interface. Documentation about NavaCys::BaseGoLBalPrimeNe: C++ documentation can be found here: BaseGlobalPlanner documentation.

Global path planning plug-in

- navfn: A global planner based on raster maps that calculates the robot's path when using the navigation function. Implemented dijkstra and A* global planning algorithms. (Plug-in name: "navfn/NavfnROS")
- global_planner: Reimplements Dijkstra and A* global path planning algorithms, which can be regarded as an improved version of navfn. (Plug-in name: "global_planner/GlobalPlanner")
- carrot_planner: A simple global path planner that takes a user-specified target point and tries to move the robot as close to it as possible, even if the target point Located in obstacles. (Plugin name: "carrot_planner/CarrotPlanner")

Global path planning global_planner_params.yaml

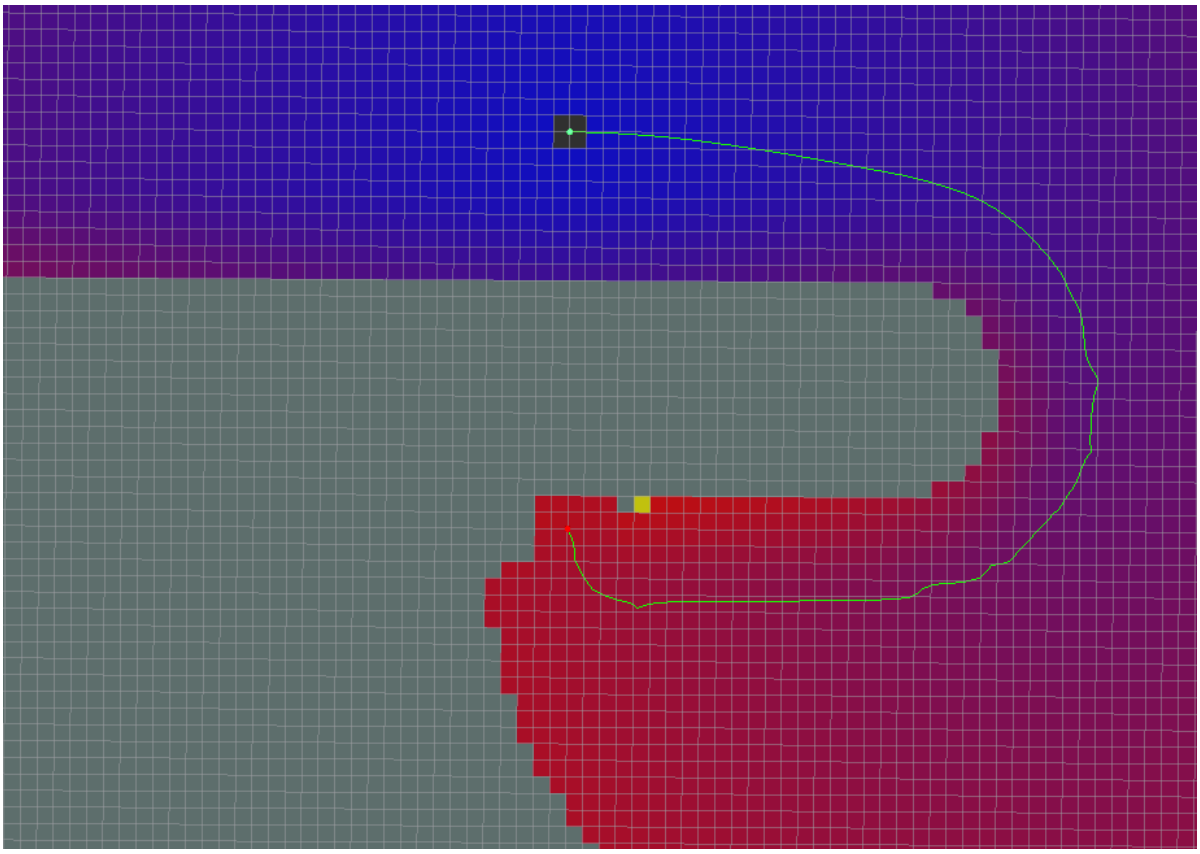
```
GlobalPlanner:
   allow_unknown: false
   default_tolerance: 0.2
   visualize_potential: false
   use_dijkstra: true
   use_quadratic: true
   use_grid_path: false
   old_navfn_behavior: false
   lethal_cost: 253
   neutral_cost: 50
   cost_factor: 3.0
   publish_potential: true
   orientation_mode: 0
   orientation_window_size: 1
```

Parameter analysis

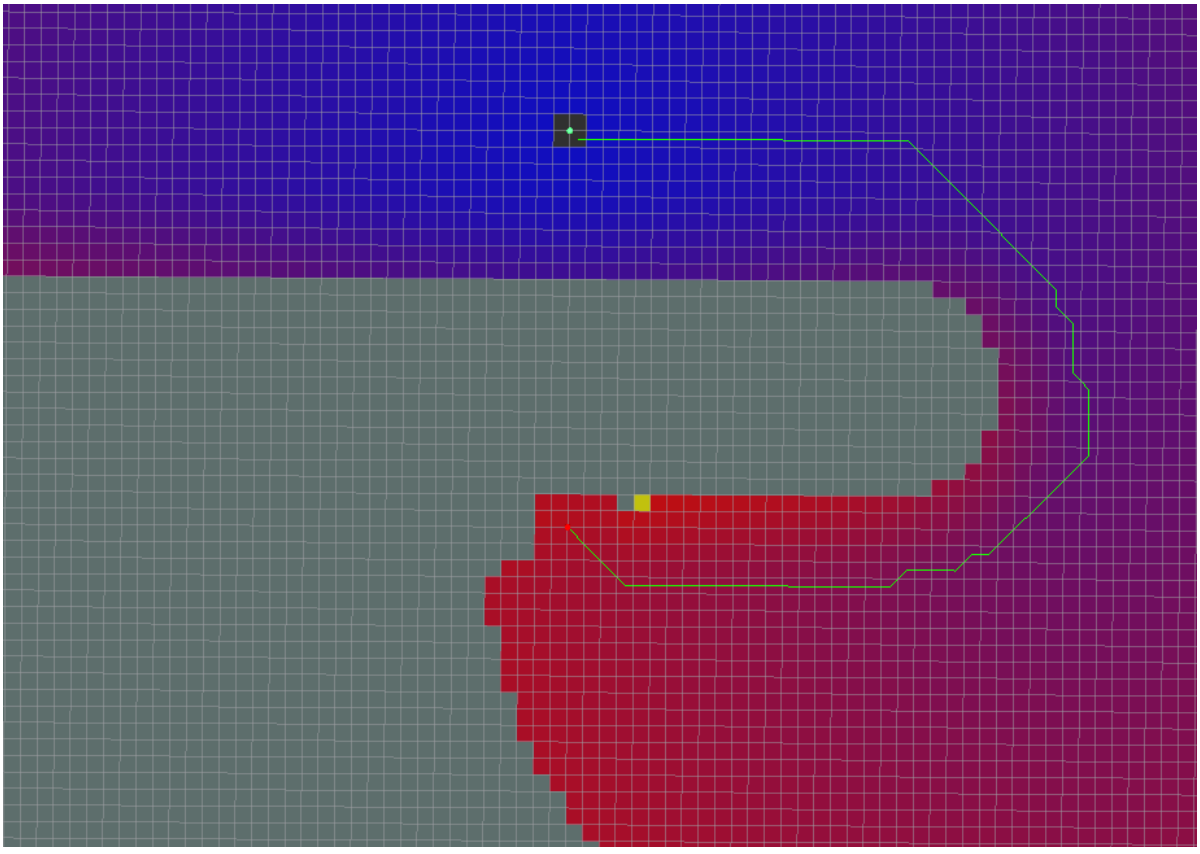
- allow_unknown: Whether to choose to explore unknown areas. It's not enough to just design this parameter to be true. You also need to set `track_unknown_space` in costmap_commons_params.yaml and it must also be set to `true`.
- default_tolerance: When the set destination is occupied by an obstacle, you need to use this parameter as the radius to find the nearest point as the new destination point.
- visualize_potential: Whether to display the possible areas calculated from PointCloud2.
- use_dijkstra: If true, use the dijkstra algorithm. Otherwise, A*.
- use_quadratic: Set to true, the quadratic function will be used to approximate the function, otherwise a simpler calculation method will be used, thus saving hardware computing resources.
- use_grid_path: If true, creates a path along the grid boundary. Otherwise, using gradient descent, the path is smoother.
- old_navfn_behavior: If you want global_planner to have the same effect as the previous navfn version, set it to true, so it is not recommended to set it to true.
  -Lethal_cost: The cost value of the obstacle's lethal area (dynamically configurable).
- neutral_cost: the neutral cost value of the obstacle (dynamically configurable).
- cost_factor: The coefficient by which the cost map is multiplied by each cost value (dynamically configurable).
- publish_potential: Whether to publish possible cost maps (dynamically configurable).
- orientation_mode: Set the orientation of each point. (None=0, Forward=1, Interpolate=2, ForwardThenInterpolate=3, Backward=4, Leftward=5, Rightward=6) (dynamically configurable).
- orientation_window_size: The direction of the window is obtained based on the position integral specified by the orientation method; the default value is 1 (dynamically configurable).
- outline_map: Outline the global cost map with fatal obstacles. For use of non-static (rolling window) global cost maps, this needs to be set to false

Global path planning algorithm renderings

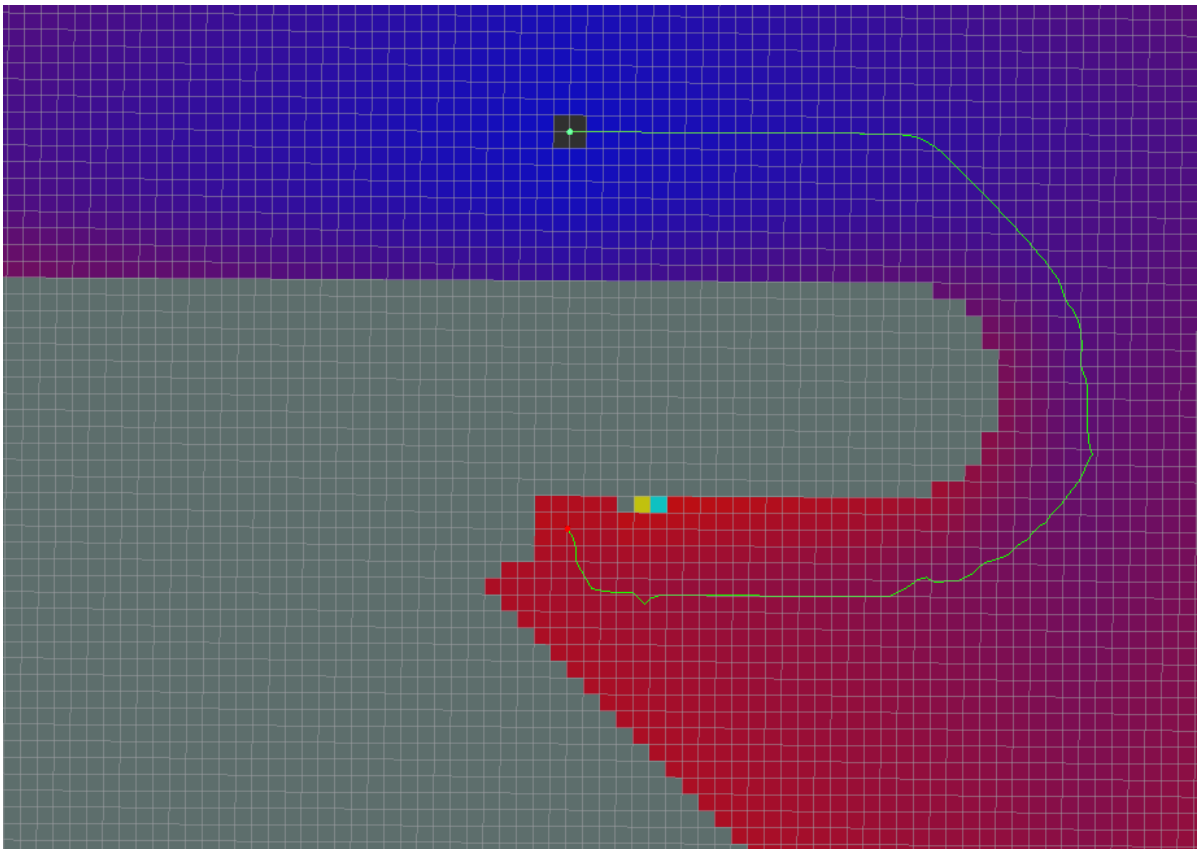
- All parameters are default values



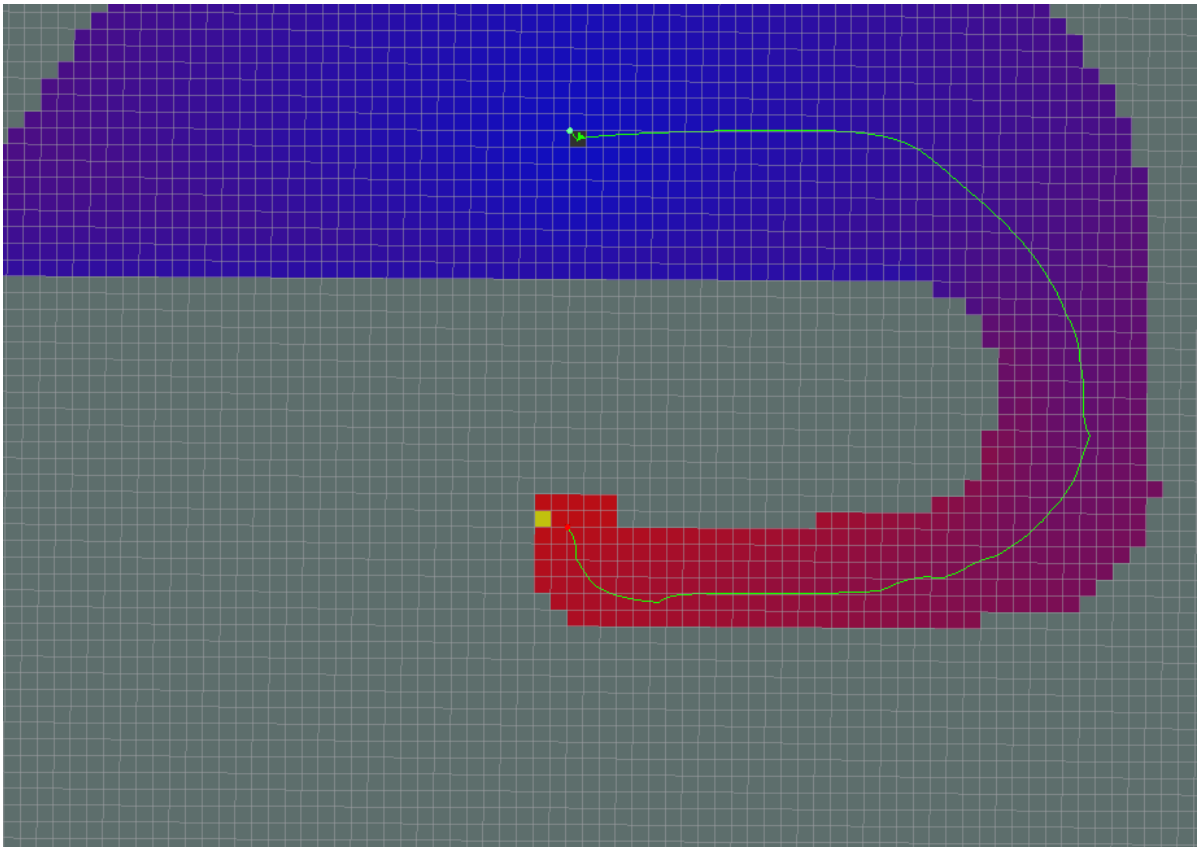
- use_grid_path=True



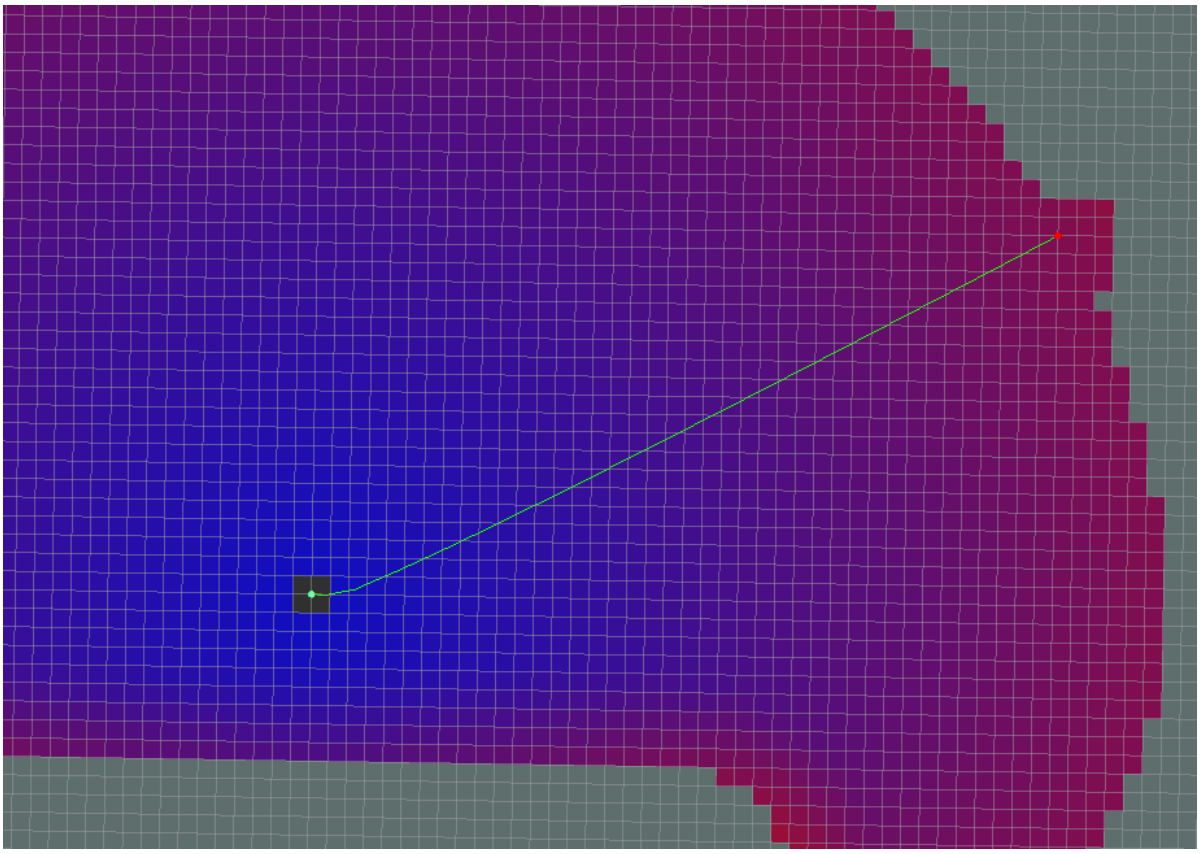
- use_quadratic=False



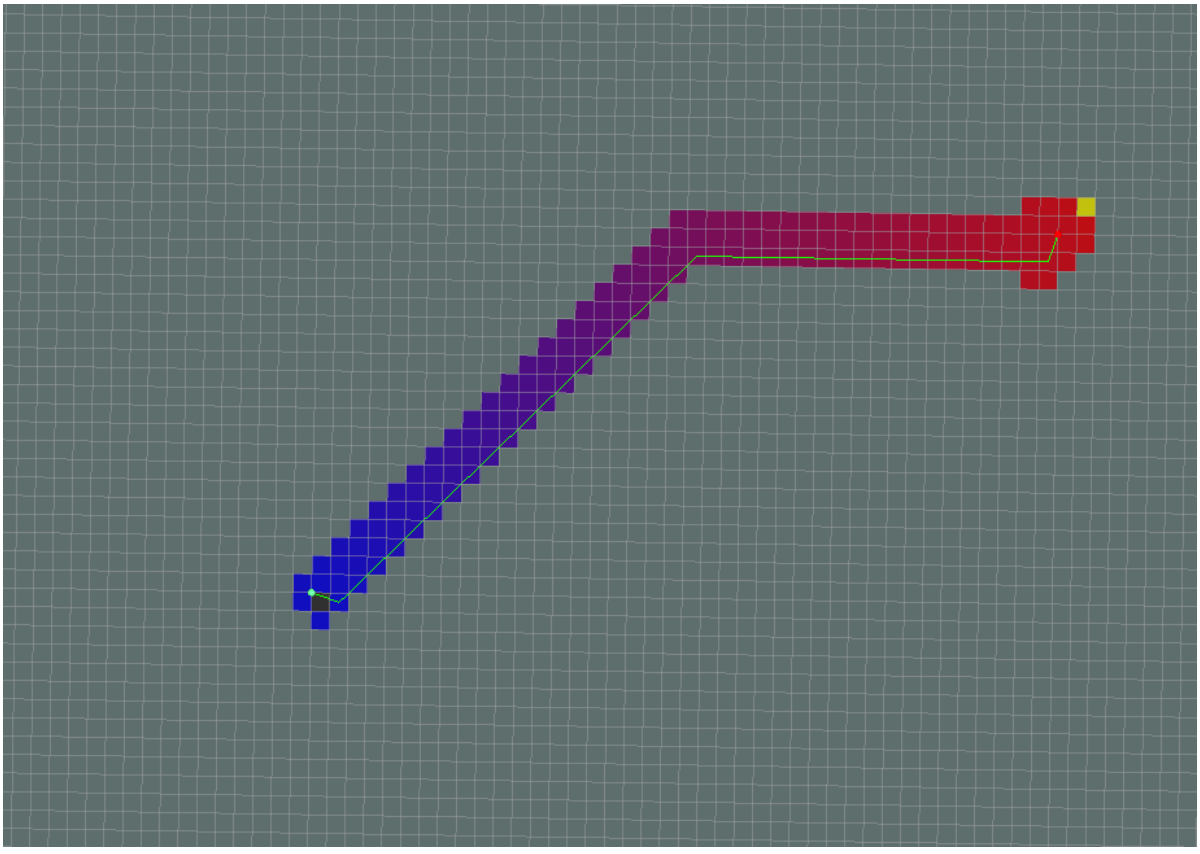
- use_dijkstra=False



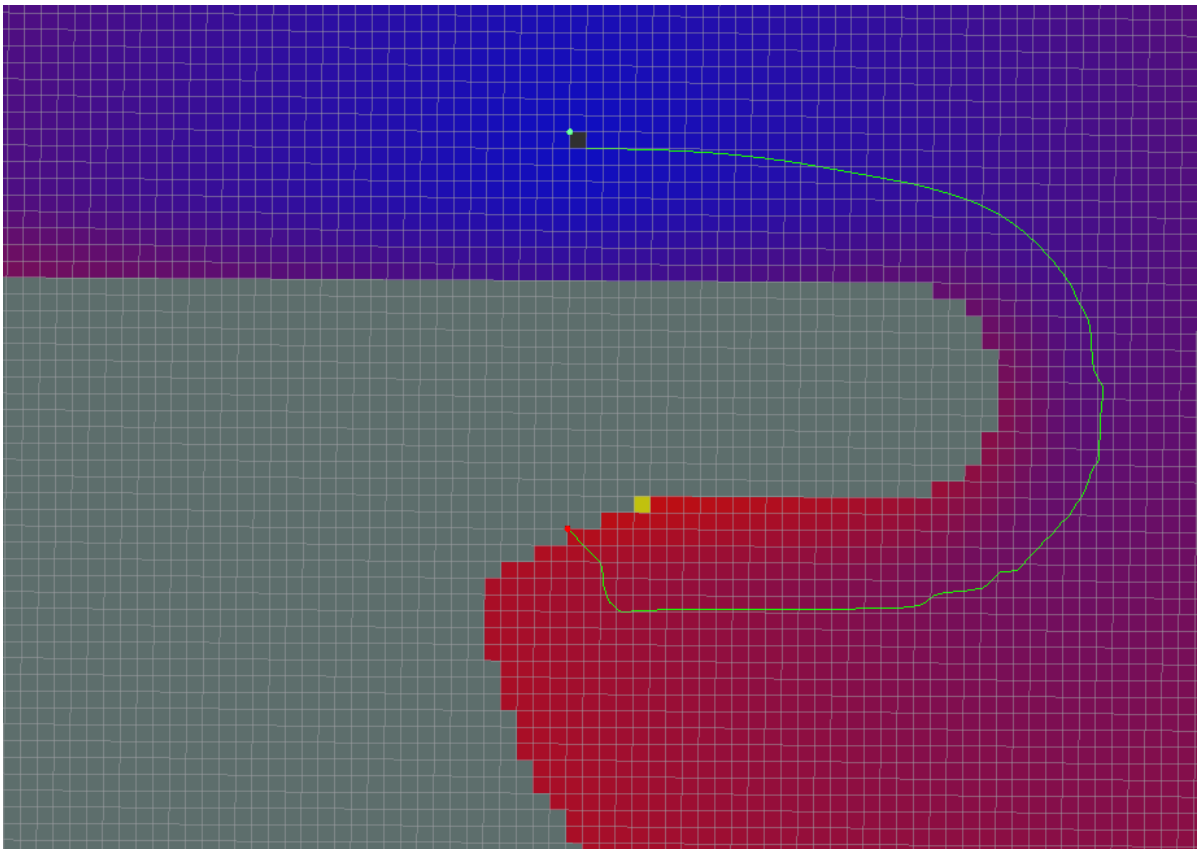
- Dijkstra



-A*



- old_navfn_behavior=True



If it appears at the very beginning:

```
[ERROR] [1611670223.557818434, 295.312000000]: NO PATH!
[ERROR] [1611670223.557951973, 295.312000000]: Failed to get a plan from
potential when a legal potential was found. This shouldn't happen.
```

This situation is related to the direction set by the robot. It is recommended to try the default [navfn] plug-in for global path planning.

12.5.2, local_planner

nav_core::BaseLocalPlanner Provides an interface for local path planners used in navigation. All local path planners written as move_base node plugins must comply with this interface. Documentation on the C++ API of NavaCys::BaseLoCalPrnor can be found here: BaseLocalPlanner documentation.

Local path planning plug-in

- base_local_planner: Implements two local planning algorithms, Trajectory Rollout and DWA.
- dwa_local_planner: Compared with base_local_planner's DWA, the modular DWA implementation has the advantages of a clearer, easier-to-understand interface and more flexible y-axis variables.
- teb_local_planner: Implements the Timed-Elastic-Band method for online trajectory optimization.
- eband_local_planner: Implements the Elastic Band method on the SE2 manifold only works with circular, differential drive, forward drive (not backward), omnidirectional robot.
- mpc_local_planner: Provides several model predictive control approaches embedded in the SE2 manifold

Comparison between TEB and DWA:

Teb will adjust its posture and orientation during the movement. When it reaches the target point, the robot's orientation is usually the target orientation without rotation.

dwa first reaches the target coordinate point, and then rotates in place to the target orientation.

For a two-wheel differential chassis, TEB's adjustment of the direction during movement will make the movement path unsmooth, causing unnecessary retreat when starting and reaching the target point, which is not allowed in some application scenarios. Because retreating may encounter obstacles. It is a more appropriate movement strategy to rotate on the spot to the appropriate direction and then walk away. This is also where teb needs to be optimized according to the scenario.

1) dwa_local_planner

The dwa_local_planner package supports any robot whose chassis can be represented as a convex polygon or a circle. This software package provides a controller to drive robot movement in a flat surface. This controller connects the path planner to the robot. The planner uses the map to create a motion trajectory for the robot from the starting point to the target location, sending the dx, dy, and dtheta velocities to the robot.

The basic idea of DWA algorithm

- Discrete sampling in robot control space (dx, dy, dtheta)
- For each sampled speed, perform a forward simulation from the current state of the robot to predict what will happen if the sampled speed is applied for a (short) period of time.
- Evaluate (score) each trajectory produced by the forward simulation, using metrics containing the following characteristics: proximity to obstacles, proximity to target, proximity to global path, and speed. Illegal trajectories (trajectories that collide with obstacles) are discarded.
- Select the trajectory with the highest score and send the associated velocity to the mobile robot.
- Clean data and repeat.

A number of ROS parameters can be set to customize the behavior of dwa_local_planner::DWAPlannerROS. These parameters are divided into several categories: robot configuration, target tolerance, forward simulation, trajectory scoring, oscillation prevention, and global planning. These parameters can be debugged using the dynamic_reconfigure tool to facilitate tuning of the local path planner in a running system.

```
DWAPlannerROS:
# Robot Configuration Parameters
   acc_lim_x: 2.5
   acc_lim_y: 2.5
   acc_lim_th: 3.2
   max_vel_trans: 0.55
   min_vel_trans: 0.1
   max_vel_x: 0.55
   min_vel_x: 0.0
   max_vel_y: 0.1
   min_vel_y: -0.1
   max_rot_vel: 1.0
   min_rot_vel: 0.4
#Goal Tolerance Parameters
   yaw_goal_tolerance: 0.05
   xy_goal_tolerance: 0.10
```

```
    latch_xy_goal_tolerance: false
# Forward Simulation Parameters
    sim_time: 2.0
    sim_granularity: 0.025
    vx_samples: 6
    vy_samples: 1
    vth_samples: 20
    controller_frequency: 5.0
# Trajectory Scoring Parameters
    path_distance_bias: 90.0 # 32.0
    goal_distance_bias: 24.0 # 24.0
    occdist_scale: 0.3 # 0.01
    forward_point_distance: 0.325 # 0.325
    stop_time_buffer: 0.2 # 0.2
    scaling_speed: 0.20 # 0.25
    max_scaling_factor: 0.2 # 0.2
    publish_cost_grid: false
#OscillationPreventionParameters
    oscillation_reset_dist: 0.05 # default 0.05
#GlobalPlanParameters
    prune_plan: false
```

Robot configuration parameters

- acc_lim_x: x acceleration limit of the robot (unit: m/s^2).
- acc_lim_y: The absolute value of acceleration in the y direction (unit: m/s^2). Note: This value only needs to be configured for robots that move in all directions.
- acc_lim_th: The absolute value of rotational acceleration (unit: rad/s^2).
- max_vel_trans: The absolute value of the maximum translation speed (unit: m/s).
- min_vel_trans: The absolute value of the minimum translation speed (unit: m/s).
- max_vel_x: The absolute value of the maximum speed in the x direction (unit: m/s).
- min_vel_x: absolute value of the minimum value in the x direction (unit: m/s). If it is a negative value, it means it can go back.
- max_vel_y: The absolute value of the maximum velocity in the y direction (unit: m/s).
- min_vel_y: The absolute value of the minimum velocity in the y direction (unit: m/s).
- max_rot_vel: The absolute value of the maximum rotation speed (unit: rad/s).
- min_rot_vel: The absolute value of the minimum rotation speed (unit: rad/s).

target tolerance parameters

- yaw_goal_tolerance: Allowable error in deflection angle when reaching the target point (unit: rad).
- xy_goal_tolerance: The tolerance allowed within the x&y distance when reaching the target point (unit: m).
- latch_xy_goal_tolerance: Set to true, if it reaches the fault tolerance distance, the robot will rotate in place, even if the rotation runs outside the fault tolerance distance.

Forward simulation parameters

- sim_time: the time of forward simulation trajectory (unit: s).

- sim_granularity: the step size between points on a given trajectory (unit: m).
- vx_samples: The number of sampling points in the x-direction velocity space.
- vy_samples: The number of sampling points in the y-direction velocity space.
- vth_samples: The number of velocity space sampling points in the rotation direction.
- controller_frequency: The frequency at which this controller is called (unit: Hz).

Trajectory scoring parameters

- path_distance_bias: Defines the weight of the controller's proximity to a given path.
- goal_distance_bias: The weight that defines the proximity of the controller to the local target point.
- occdist_scale: Defines the weight of the controller to avoid obstacles.
- forward_point_distance: The distance from the center point of the robot to where the extra scoring point is placed (unit: m).
- stop_time_buffer: The length of time the robot must stop in advance before collision (unit: s).
- scaling_speed: The speed at which the scaling robot chassis is started (unit: m/s).
- max_scaling_factor: The maximum scaling parameter of the robot chassis.
- publish_cost_grid: Whether to publish the planner's cost grid when planning the path. If set to true, sensor_msgs/PointCloud2 type messages will be published on the ~/cost_cloud topic.

Anti-vibration parameters

- oscillation_reset_dist: How far the robot moves before the oscillation mark is reset (unit: m).

Global planning parameters

- prune_plan: When the robot moves forward, whether to clear the trajectory 1m behind it.

2) teb_local_planner

teb_local_planner is an optimization-based local trajectory planner. Supports differential models and car-like models. This software package implements an online optimal local trajectory planner for mobile robot navigation and control, effectively obtaining optimal trajectories by solving sparse scalarized multi-objective optimization problems. Users can provide weights to the optimization problem to specify behavior in case of conflicting goals.

The teb_local_planner package allows users to set parameters to customize the behavior. These parameters are divided into several categories: robot configuration, target tolerance, trajectory configuration, obstacles, optimization, planning in unique topologies and other parameters. Some of these were chosen to meet basic local planner requirements. Many (but not all) parameters can be modified at runtime using rqt_reconfigure.

Local path planner teb_local_planner_params.yaml

```
TebLocalPlannerROS:
#MiscellaneousParameters
  map_frame: odom
  odom_topic: odom
#Robot
  acc_lim_x: 0.5
  acc_lim_theta: 0.5
  max_vel_x: 0.4
```

```
  max_vel_x_backwards: 0.2
  max_vel_theta: 0.3
  min_turning_radius: 0.0
  footprint_model:
    type: "point"
#GoalTolerance
  xy_goal_tolerance: 0.2
  yaw_goal_tolerance: 0.1
  free_goal_vel: False
# Trajectory
  dt_ref: 0.3
  dt_hysteresis: 0.1
  min_samples: 3
  global_plan_overwrite_orientation: True
  allow_init_with_backwards_motion: False
  max_global_plan_lookahead_dist: 3.0
  global_plan_viapoint_sep: -1
  global_plan_prune_distance: 1
  exact_arc_length: False
  feasibility_check_no_poses: 5
  publish_feedback: False
#Obstacles
  min_obstacle_dist: 0.25
  inflation_dist: 0.6
  include_costmap_obstacles: True
  costmap_obstacles_behind_robot_dist: 1.5
  obstacle_poses_affected: 15
  dynamic_obstacle_inflation_dist: 0.6
  include_dynamic_obstacles: True
  costmap_converter_plugin: ""
  costmap_converter_spin_thread: True
  costmap_converter_rate: 5
#Optimization
  no_inner_iterations: 5
  no_outer_iterations: 4
  optimization_activate: True
  optimization_verbose: False
  penalty_epsilon: 0.1
  obstacle_cost_exponent: 4
  weight_max_vel_x: 2
  weight_max_vel_theta: 1
  weight_acc_lim_x: 1
  weight_acc_lim_theta: 1
  weight_kinematics_nh: 1000
  weight_kinematics_forward_drive: 1
  weight_kinematics_turning_radius: 1
  weight_optimaltime: 1 # must be > 0
  weight_shortest_path: 0
  weight_obstacle: 100
  weight_inflation: 0.2
  weight_dynamic_obstacle: 10
  weight_dynamic_obstacle_inflation: 0.2
  weight_viapoint: 1
  weight_adapt_factor: 2
# Parallel Planning
 enable_homotopy_class_planning: True
```

```
 enable_multithreading: True
 max_number_classes: 4
 selection_cost_hysteresis: 1.0
 selection_prefer_initial_plan: 0.9
 selection_obst_cost_scale: 100.0
 selection_alternative_time_cost: False
 roadmap_graph_no_samples: 15
 roadmap_graph_area_width: 5
 roadmap_graph_area_length_scale: 1.0
 h_signature_prescaler: 0.5
 h_signature_threshold: 0.1
 obstacle_heading_threshold: 0.45
 switching_blocking_period: 0.0
 viapoints_all_candidates: True
 delete_detours_backwards: True
 max_ratio_detours_duration_best_duration: 3.0
 visualize_hc_graph: False
 visualize_with_time_as_z_axis_scale: False
# Recovery
 shrink_horizon_backup: True
 shrink_horizon_min_duration: 10
 oscillation_recovery: True
 oscillation_v_eps: 0.1
 oscillation_omega_eps: 0.1
 oscillation_recovery_min_duration: 10
 oscillation_filter_duration: 10
```

Robot configuration

- acc_lim_x: The maximum translational acceleration of the robot (unit: m/s^2).
- acc_lim_theta: The maximum angular acceleration of the robot (unit: rad/s^2).
- max_vel_x: The maximum translation speed of the robot (unit: m/s).
- max_vel_x_backwards: The maximum absolute translation speed of the robot when traveling backwards (unit: m/s).
- max_vel_theta: The maximum angular velocity of the robot (unit: rad/s).

  The following parameters are only relevant to carlike robots
- min_turning_radius: The minimum turning radius of the carlike robot (for differential robots, set to zero).
- wheelbase: distance between rear axle and front axle. For rear-wheel robots, this value may be negative (only required if /cmd_angle_instead_rotvel is set to true).
- cmd_angle_instead_rotvel: Use the corresponding steering angle [-pi/2, pi/2] to replace the rotation speed in the command speed information.

  The following parameters are only relevant for complete robots: New Parameters in ROS Dynamics
- max_vel_y: Maximum strafing speed of the robot (should be zero for non-omnidirectional robots!).
- acc_lim_y: The maximum strafing acceleration of the robot.

  The following parameters are relevant to the chassis model used for optimization
- footprint_model:

type: "point" :

Parameter [footprint_model]

Specifies the robot schematic model type used for optimization. The different types are "point", "circular", "line", "two_circles", "polygon". The type of model significantly affects the required computational time.

- footprint_model/radius: This parameter is only relevant for the "circular" type. It contains the radius of the circle. The center of the circle is located on the axis of rotation of the robot.
- footprint_model/line_start: This parameter is only relevant for the "line" type. It contains the starting coordinates of the line segment.
- footprint_model/line_end: This parameter is only relevant for the "line" type. It contains the coordinates of the endpoints of the line segment.
- footprint_model/front_offset: This parameter is only relevant for the ''two_circles'' type. It describes how much the center of the front circle moves along the robot's x-axis. Assume that the robot's rotation axis is located at [0,0].
- footprint_model/front_radius: This parameter is only relevant for the ''two_circles'' type. . It contains the radius of the front circle.
- footprint_model/rear_offset: This parameter is only relevant for the ''two_circles'' type. It describes how much the center of the back circle moves along the robot's negative x-axis. Assume that the robot's rotation axis is located at [0,0].
- footprint_model/rear_radius: This parameter is only relevant for the ''two_circles'' type. It contains the radius of the back circle.
- footprint_model/vertices: This parameter is only relevant for the "polygon" type. It contains a list of polygon vertices (the 2D coordinates of each vertex). Polygons are always closed: don't repeat the first vertex at the end.
- is_footprint_dynamic: If true, the footprint is updated before checking track feasibility.

target tolerance

- yaw_goal_tolerance: Allowable error in deflection angle when reaching the target point (unit: rad).
- xy_goal_tolerance: The tolerance allowed within the x&y distance when reaching the target point (unit: m).
- free_goal_vel: Remove the target speed constraint so that the robot can reach the target at maximum speed.

Track configuration

- dt_ref: Desired trajectory time resolution.
- dt_hysteresis: Hysteresis automatically sized based on current time resolution, usually recommended to be around 10% of dt_ref.
- max_samples: Minimum number of samples (should always be greater than 2).
- global_plan_overwrite_orientation: Overrides the orientation of local subgoals provided by the global planner (as they usually only provide 2D paths)
- global_plan_viapoint_sep: If positive, via points are extracted from the global plane (path following mode). This value determines the resolution of the reference path (minimum separation between each consecutive via point along the global plane, if negative: disabled).

- max_global_plan_lookahead_dist: Specifies the maximum length (cumulative Euclidean distance) of the global plan subset considered during optimization. The actual length is determined by the size of the local cost map and the logical connection of this maximum bound. Set to zero or negative to disable this limit.
- force_reinit_new_goal_dist: Reinitialize the trajectory if the previous goal's update interval exceeds the specified number of meters (skip warm start).
- feasibility_check_no_poses: Specifies which poses in the prediction plan should be checked for feasibility each sampling interval.
- publish_feedback: Publish planner feedback with full trajectory and list of active obstacles (should only be enabled when evaluating or debugging)
- shrink_horizon_backup: Allows the planner to temporarily shrink the scope (50%) when it automatically detects a problem (e.g. infeasibility).
- allow_init_with_backwards_motion: If true, the base trajectory may be initialized with backwards motion if the target is behind the starting point in the local cost map (this is only recommended if the robot is equipped with rear sensors).
- exact_arc_length: If true, the planner uses exact arc length (->increased cpu time) in velocity, acceleration and steering rate calculations, otherwise uses Euclidean approximation.
- shrink_horizon_min_duration: Specifies the minimum duration for shrinking the horizon in case an infeasible trajectory is detected.

obstacle

- min_obstacle_dist: Minimum desired distance from obstacles (unit: m).
- include_costmap_obstacles: Specifies whether obstacles for local cost maps should be considered.
- costmap_obstacles_behind_robot_dist: Local cost map obstacles (unit: m) that limit the occupation considered when planning behind the robot.
- obstacle_poses_affected: Each obstacle position is appended to the nearest pose on the trajectory to maintain distance.
- inflation_dist: Buffer around non-zero penalty cost obstacles (should be larger than the minimum obstacle distance in order to take effect).
- include_dynamic_obstacles: If this parameter is set to true, the motion of obstacles with non-zero velocity is predicted and taken into account during the optimization process by the constant velocity model.
- legacy_obstacle_association: Modified strategy for connecting trajectory poses with obstacles for optimization.

The following parameters are only relevant when the costmap_converter plug-in is required:

- costmap_converter_plugin: ""

  Define the plugin name to convert cost map units to points/lines/polygons. Set an empty string to disable the conversion so that all cells are treated as point obstacles.
- costmap_converter_spin_thread: If set to true, costmap_converter will call its callback queue in a different thread
- costmap_converter_rate: Defines the frequency at which the costmap_converter plugin processes the current cost map (unit: Hz).

optimization

- no_inner_iterations: The actual number of solver iterations called on each outer loop iteration.
- no_outer_iterations: Each outer loop iteration automatically resizes the trajectory according to the required time resolution dt_ref and calls the internal optimizer (no inner iterations are performed).
- penalty_epsilon: Add a small safety margin to the penalty function of the hard-constrained approximation.
- weight_max_vel_x: Optimization weight that satisfies the maximum allowed translation speed.
- weight_max_vel_theta: Optimization weight that satisfies the maximum allowed angular velocity.
- weight_acc_lim_x: Optimization weight that satisfies the maximum allowed translation acceleration.
- weight_acc_lim_theta: Optimization weight that satisfies the maximum allowed angular acceleration.
- weight_kinematics_nh: Optimized weights that satisfy non-holonomic kinematics.
- weight_kinematics_forward_drive: Optimized weight used to force the robot to only choose the forward direction (positive translation speed).
- weight_kinematics_turning_radius: Optimized weight to implement minimum turning radius (only for carlike robots).
- weight_optimaltime: shorten the optimization weight of the trajectory with respect to transition/execution time
- weight_obstacle: Optimized weight to maintain minimum distance from obstacles.
- weight_viapoint: Optimization weight used to minimize the distance to the via point (respectively reference path).
- weight_inflation: Optimization weight of inflation penalty (should be smaller).
- weight_adapt_factor: Some special weights (current weights) are scaled repeatedly by this factor in each outer TEB iteration.

planning

- enable_homotopy_class_planning: Activate parallel planning in unique topologies.
- enable_multithreading: Activate multithreading to plan each trajectory in different threads.
- max_number_classes: Specifies the maximum number of different trajectories considered.
- selection_cost_hysteresis: Specifies how much trajectory cost a new candidate trajectory must have before it can be selected.
- selection_obst_cost_scale: Additional expansion of obstacle cost conditions.
- selection_viapoint_cost_scale: Additional expansion of via point cost conditions.
- selection_alternative_time_cost: If true, the time cost (sum of squared time differences) will be replaced by the total transition time.
- roadmap_graph_no_samples: Specifies the number of samples generated to create a roadmap.

- roadmap_graph_area_width: Random keypoints/waypoints sampled in a rectangular area between start and target (unit: m).
- obstacle_heading_threshold: Specifies the value of the scalar product between obstacle headings and goal headings so that they (obstacles) are taken into account when exploring.
- visualize_hc_graph: Visualize graphs created to explore unique trajectories.
- viapoints_all_candidates: If true, all traces of different topologies will be connected to the via point set, otherwise only traces sharing the same topology as the initial/global plan will be connected to it.
- switching_blocking_period: Specifies the duration (unit: s) that needs to expire before switching to a new equivalence class is allowed.