8. Navigation and obstacle avoidance

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

8.1、Operation and use

Note: The [SWB] mid-range of the aircraft remote control has the [Emergency Stop] function for this gameplay

- To start control, you need to first turn the SWB button to the upper position (control command mode) to release the remote control.

8.1.1 Startup

Turn off the self-starting chassis service

```
sudo supervisorctl stop ChassisServer
```

Start chassis driver command (robot side)

```
sudo supervisorctl restart LaserServer #start/stop Turn on/off radar service
(indoor version)
roslaunch scout_bringup scout_mini_robot_base.launch # laser + yahboom
```

- It is necessary to ensure that the radar starts normally. If you run `rostopic echo /scan`

and the print is empty and the data cannot be obtained, the startup is abnormal. Please restart the radar service command.

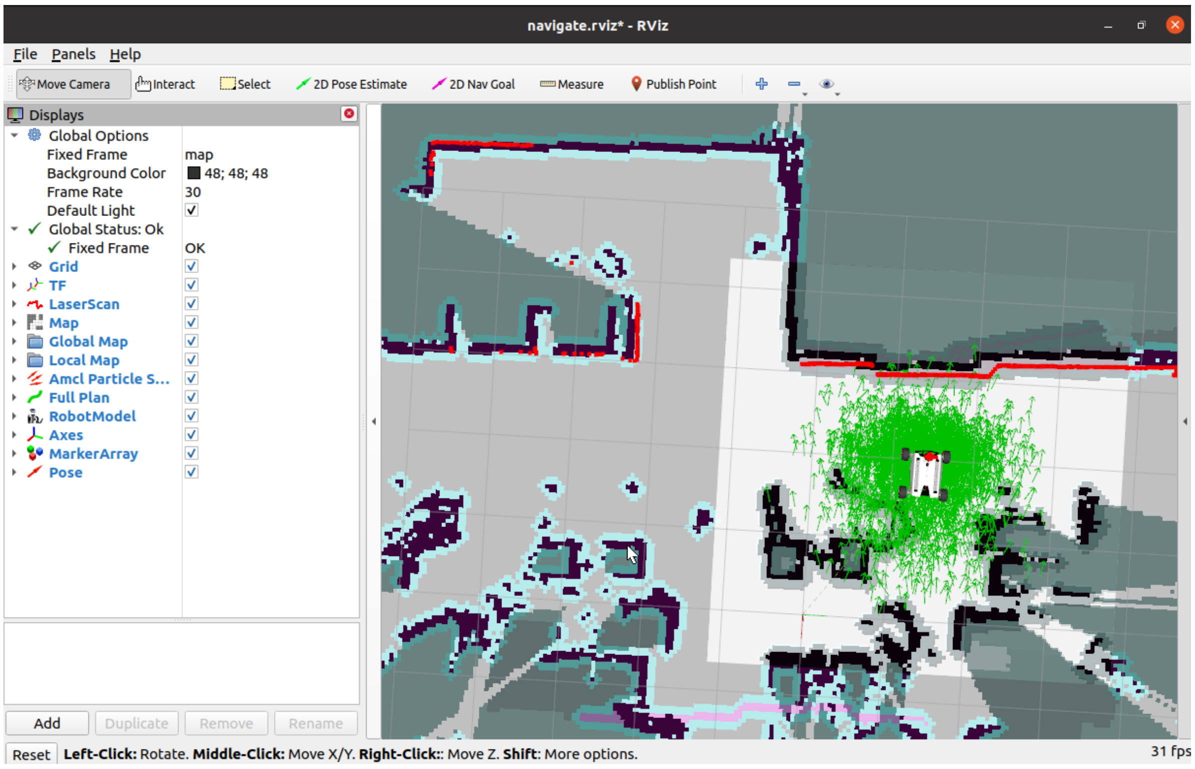
Start the navigation obstacle avoidance function (robot side). You can set parameters according to your needs and modify the launch file.

```
roslaunch yahboom_navrobo_nav yahboom_navrobo_navigation.launch use_rviz:=false
map:=my_map
```

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

Open the visualization interface (robo side)

```
roslaunch yahboom_navrobo_nav view_navigate.launch
```



```
Global Options
    Fixed Frame          map
    Background Color     48; 48; 48
    Frame Rate           30
    Default Light        ✓
✓ Global Status: Ok
    ✓ Fixed Frame        OK
Grid                     ✓
TF                       ✓
LaserScan                ✓
Map                      ✓
Global Map               ✓
Local Map                ✓
Amcl Particle S...       ✓
Full Plan                ✓
RobotModel               ✓
Axes                     ✓
MarkerArray              ✓
Pose                     ✓
```

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click::** Move Z. **Shift:** More options.

8.1.2, 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 actual position of the car.
- Click the [2D Nav Goal] 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 only select one target point and stop when you reach it.

2) Multi-point navigation

- Same as the first step of single-point navigation, first set the initial pose of the car.
- Click the [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. The robot will cruise 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

Looking at the yahboom_navrobo_navigation.launch file, we can see that the navigation parameters are modified in the yahboom_scout_mini_move_base_felx_.launch file under the yahboom_scout_mini_localization function package.

```
<launch>
    <!-- move_base parameters -->
    <arg name="move_forward_only"         default="false" />
    <arg name="odom_topic"                default="odom_wheel" />

    <!-- 是否打开rviz || Whether to open rviz -->
    <arg name="use_rviz" default="true"/>
    <!-- 地图名 || Map name  my_map-->
    <arg name="map" default="my_map"/>
    <!-- MarkerArray node> -->
    <node name='send_mark' pkg="yahboom_navrobo_nav" type="send_mark.py"/>
    <!-- 加载地图 || Load map -->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find
yahboom_navrobo_nav)/maps/$(arg map).yaml"/>
    <!-- AMCL自适应蒙特卡洛定位 -->
    <include file="$(find
yahboom_scout_mini_localization)/launch/yahboom_scout_mini_amcl.launch" />
    <!-- 导航核心组件move_base -->
    <!-- move_base -->
    <include file="$(find
yahboom_scout_mini_localization)/launch/yahboom_scout_mini_move_base_felx_.launc
h" >
        <arg name="move_forward_only" value="$(arg move_forward_only)" />
        <arg name="odom_topic" value="$(arg odom_topic)" />
    </include>

    <!-- RVIZ -->
    <include file="$(find yahboom_navrobo_nav)/launch/view/view_navigate.launch"
if="$(arg use_rviz)"/>
</launch>
```

Find the yahboom_scout_mini_move_base_felx_.launch file and open the sample file as follows. You can modify and replace it according to your needs. At this time, the [DWA Planner] is selected and the [DWA] file is loaded.

```
<launch>
  <!-- Arguments move_base-->
  <arg name="cmd_vel_topic" default="/cmd_vel" />
  <arg name="odom_topic" default="odom" />
  <arg name="move_forward_only" default="false"/>
<!--<rosparam file="$(find
yahboom_scout_mini_localization)/param/global_planner_params.yaml" command="load"
/>

<rosparam file="$(find
yahboom_scout_mini_localization)/param/move_base_params.yaml" command="load" />
-->

  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
clear_params="true" output="screen">
    <param name="base_global_planner" value="graph_planner/GraphPlanner"/>
    <param name="GraphPlanner/planner_name" value="dijkstra"/>
    <rosparam file="$(find sim_env)/config/planner/graph_planner_params.yaml"
command="load"/>
    <param name="base_local_planner" value="dwa_planner/DWAPlanner" />
    <rosparam file="$(find
yahboom_scout_mini_localization)/param/costmap_common_params_yb.yaml"
command="load" ns="global_costmap" />
    <rosparam file="$(find
yahboom_scout_mini_localization)/param/costmap_common_params_yb.yaml"
command="load" ns="local_costmap" />
    <rosparam file="$(find
yahboom_scout_mini_localization)/param/local_costmap_params_yb.yaml"
command="load" />
    <rosparam file="$(find
yahboom_scout_mini_localization)/param/global_costmap_params_yb.yaml"
command="load" />
```

```
    <rosparam file="$(find
yahboom_scout_mini_localization)/param/move_base_params.yaml" command="load" />
    <rosparam file="$(find
yahboom_scout_mini_localization)/param/dwa_local_planner_params.yaml"
command="load" />

    <remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
    <remap from="odom" to="$(arg odom_topic)"/>
    <param name="DWAPlannerROS/min_vel_x" value="0.0" if="$(arg
move_forward_only)" />
  </node>

</launch>
```

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

Enter the dwa_local_planner_params.yaml file under the yahboom_scout_mini_localization function package, and some parameters are as follows:

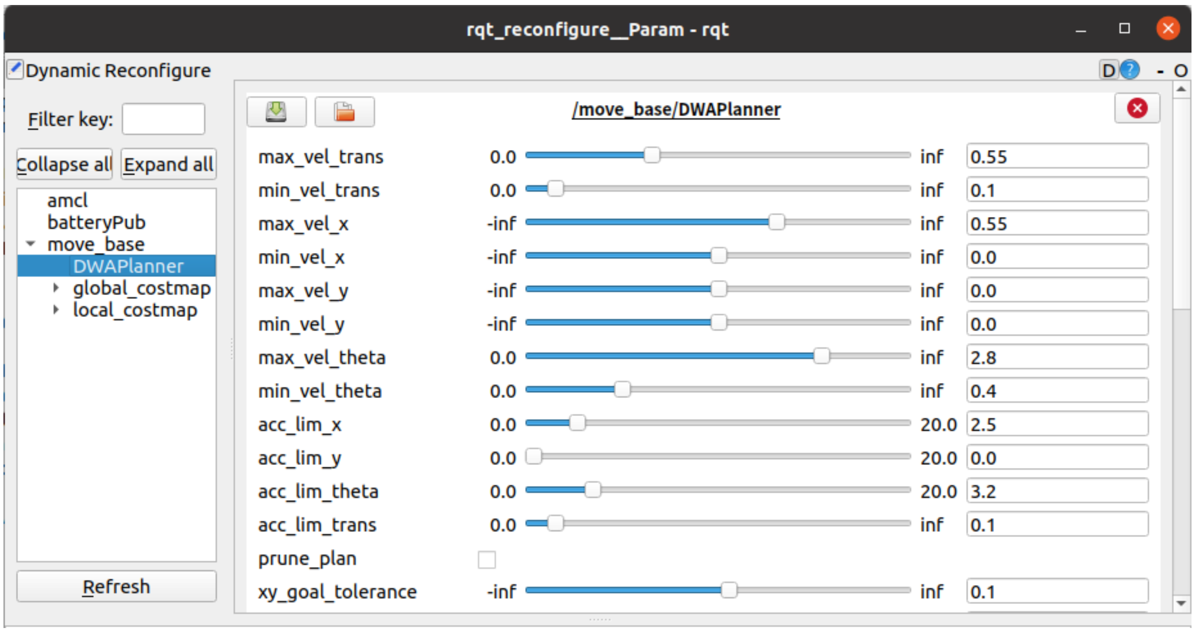
```
DWAPlannerROS:
# Robot Configuration Parameters
  max_vel_x: 0.5 # Absolute value of the maximum linear velocity in the x
direction, unit: m/s
  min_vel_x: 0.0 # Absolute value of the minimum linear velocity in the x
direction, negative numbers mean that the robot can move backwards, unit: m/s
  max_vel_y: 0.0 # Absolute value of the maximum linear velocity in the y
direction, unit: m/s. For differential drive robots, it is 0
  min_vel_y: 0.0 # Absolute value of the minimum linear velocity in the y
direction, unit: m/s. For differential drive robots, it is 0
... ...
  # 机器人在x方向的极限加速度，单位为 meters/sec^2
  # The x acceleration limit of the robot in meters/sec^2
  acc_lim_x: 0.5
  # 机器人在y方向的极限加速度，差速机器人来说是0
  # The y acceleration limit of the robot in meters/sec^2
  acc_lim_y: 0.0
... ..
```

Other parameter files can be opened and modified according to your own needs in combination with annotations and courseware.

8.1.3 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 in the dwa_local_planner_params.yaml file under the yahboom_scout_mini_localization function package.

```
rosrun rqt_reconfigure rqt_reconfigure
```


rqt_reconfigure__Param - rqt

Dynamic Reconfigure

Filter key:

Collapse all | Expand all

- amcl
- batteryPub
- move_base
  - DWAPlanner
  - global_costmap
  - local_costmap

Refresh

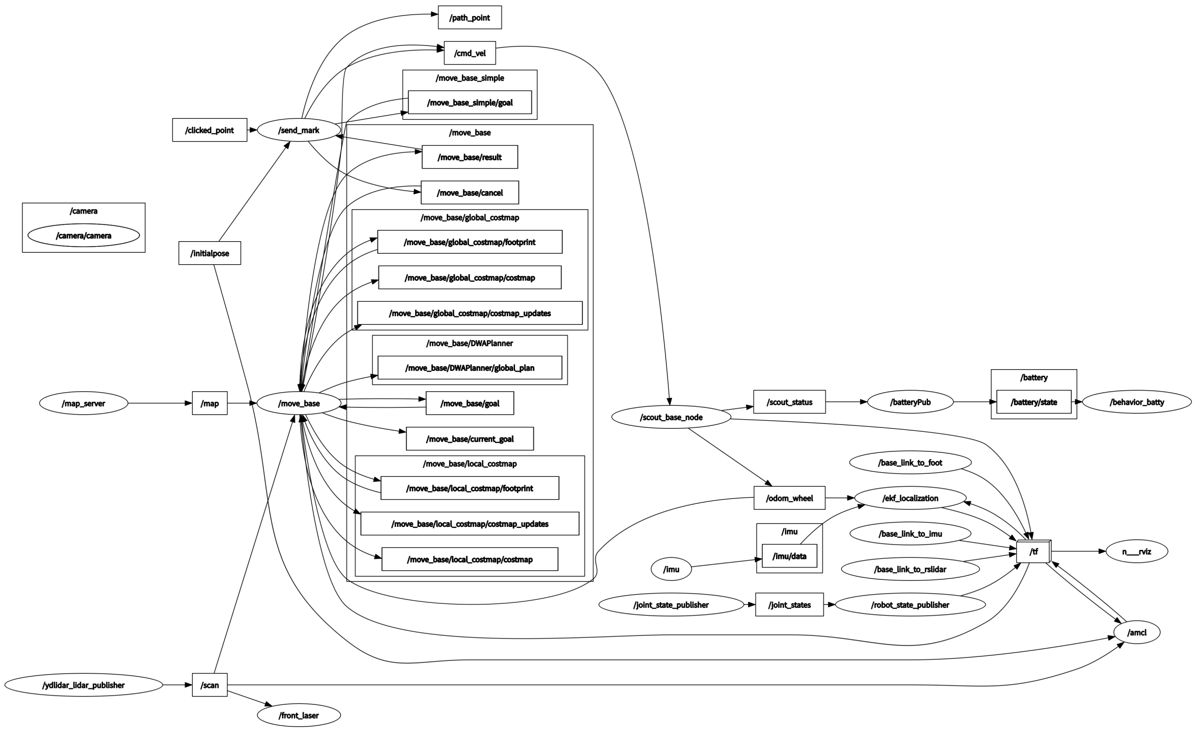
/move_base/DWAPlanner

```
max_vel_trans       0.0   inf   0.55
min_vel_trans       0.0   inf   0.1
max_vel_x          -inf   inf   0.55
min_vel_x          -inf   inf   0.0
max_vel_y          -inf   inf   0.0
min_vel_y          -inf   inf   0.0
max_vel_theta       0.0   inf   2.8
min_vel_theta       0.0   inf   0.4
acc_lim_x           0.0  20.0   2.5
acc_lim_y           0.0  20.0   0.0
acc_lim_theta       0.0  20.0   3.2
acc_lim_trans       0.0   inf   0.1
prune_plan
xy_goal_tolerance  -inf   inf   0.1
```

8.1.4, Node Graph

Take the startup in section [1.1] as an example to observe the communication between nodes.

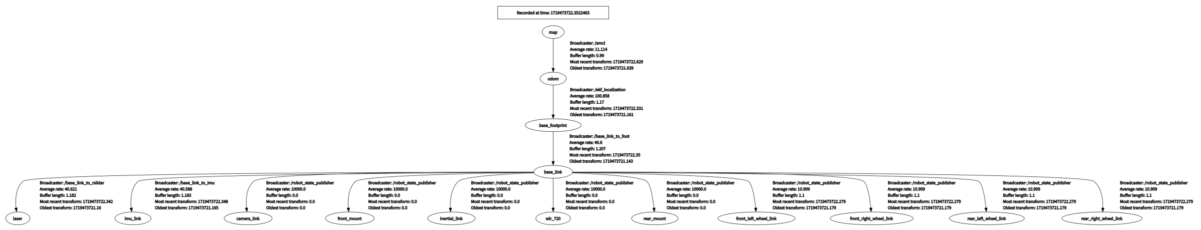
```
rosrun rqt_graph rqt_graph
```



8.1.5, tf coordinate system

The conversion relationship between coordinates.

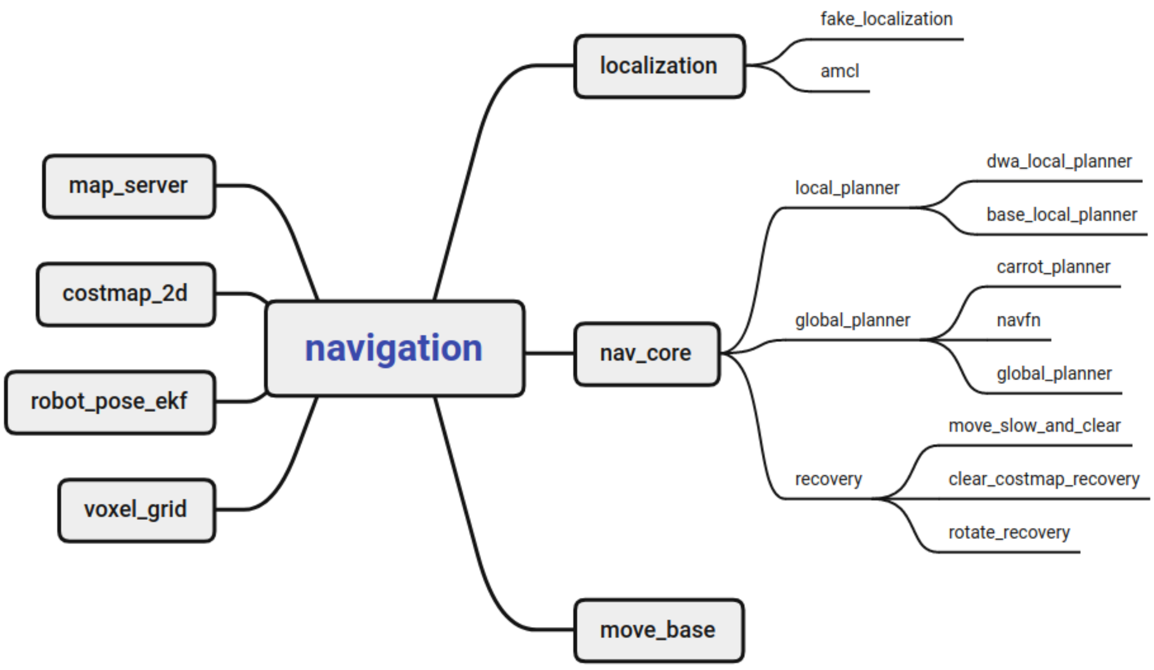
```
rosrun rqt_tf_tree rqt_tf_tree
```


Recorded at time: 1719473722.3522463
map
Broadcaster: /amcl
Average rate: 11.114
Buffer length: 0.99
Most recent transform: 1719473722.629
Oldest transform: 1719473721.639
odom
Broadcaster: /ekf_localization
Average rate: 100.858
Buffer length: 1.17
Most recent transform: 1719473722.331
Oldest transform: 1719473721.161
base_footprint
Broadcaster: /base_link_to_foot
Average rate: 40.6
Buffer length: 1.207
Most recent transform: 1719473722.35
Oldest transform: 1719473721.143
base_link
Broadcaster: /base_link_to_rslidar
Average rate: 40.621
Buffer length: 1.182
Most recent transform: 1719473722.342
Oldest transform: 1719473721.16
Broadcaster: /base_link_to_imu
Average rate: 40.588
Buffer length: 1.183
Most recent transform: 1719473722.348
Oldest transform: 1719473721.165
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10.909
Buffer length: 1.1
Most recent transform: 1719473722.279
Oldest transform: 1719473721.179
laser
imu_link
camera_link
front_mount
inertial_link
wlr_720
rear_mount
front_left_wheel_link
front_right_wheel_link
rear_left_wheel_link
rear_right_wheel_link

8.2, navigation

8.2.1, Introduction

Navigation is a two-dimensional navigation and obstacle avoidance function package of ROS. In simple terms, it is to calculate safe and reliable robot speed control instructions based on the information flow of input sensors such as odometers and the global position of the robot through navigation algorithms.



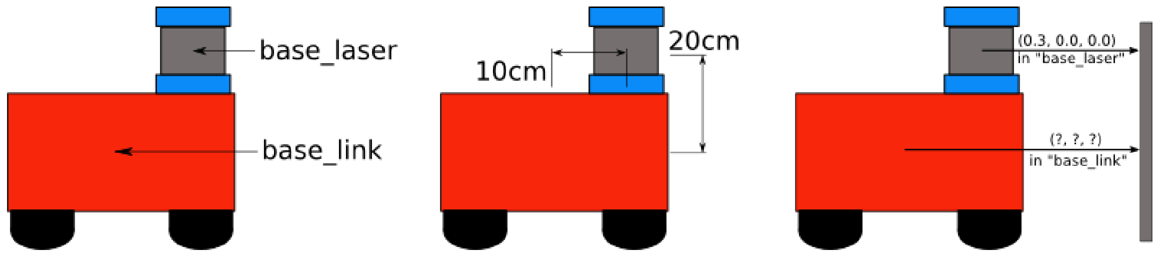
Main nodes and configurations of navigation

- move_base: The final actuator of navigation and 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 environmental information.
- local_costmap: The local cost map is used to describe local environmental information.
- recovery_behaviors: The recovery strategy is used for the robot to automatically escape and recover after encountering an obstacle.
- amcl: Use the particle filter algorithm to achieve 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: It can produce cost maps and provide various related functions.

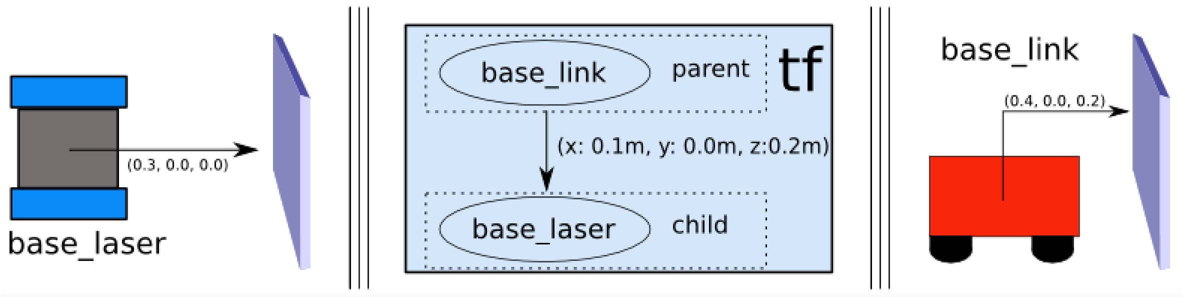
- robot_pose_ekf: Extended Kalman filter, the input is any two or three of the odometer, IMU, and VO, and the output is a fused pose.
- fake_localization: Generally used for simulation.
- nav_core: There are only three files here, corresponding to the general interface definitions of global path planning, local path planning, and recovery_action, and the specific function implementation is in each corresponding planner function package.
- It is also necessary to provide tf information related to the robot model, odometer odom information, and lidar scan information.

8.2.2, set tf

The navigation function requires the robot to use tf to publish information about the relationship between coordinate systems. For example: LiDAR



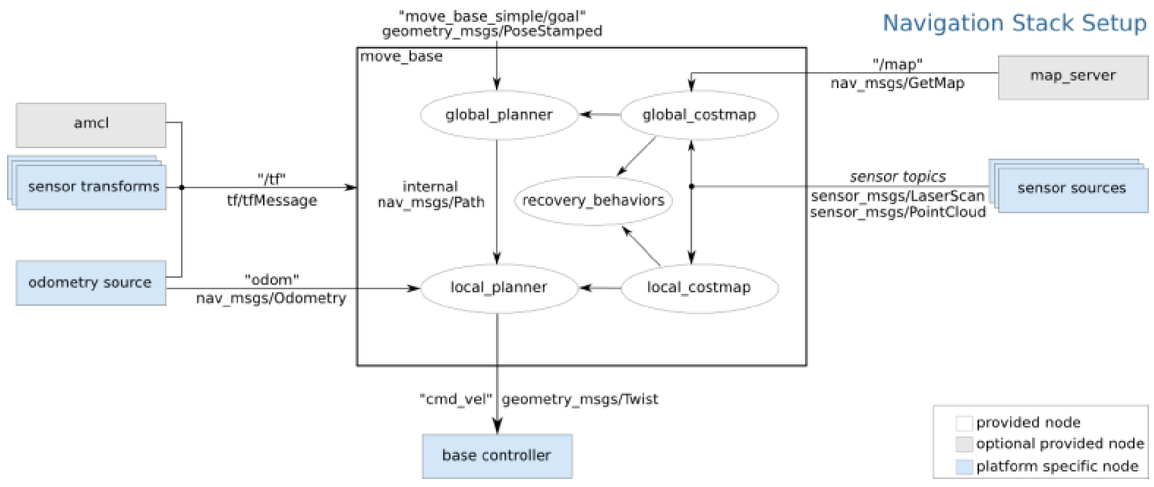
Suppose we know that the LiDAR is mounted 10cm and 20cm above the center point of the mobile base. This gives us the translation offsets that relate the "base_link" frame to the "base_laser" frame. Specifically, we know that to get data from the "base_link" coordinate system to the "base_laser" coordinate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



8.3、move_base

8.3.1, Introduction

move_base provides the configuration, operation, and interaction interfaces for ROS navigation.



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.

8.3.2, move_base communication mechanism

1) Action

The move_base node provides an implementation of SimpleActionServer, which receives targets containing geometry_msgs/PoseStamped messages. You can communicate directly with the move_base node through ROS, but if you care about the state of the tracked target, it is recommended to use SimpleActionClient to send the target to move_base. (For more information, see the actionlib documentation)

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

2) topic

Name	Type	Description
move_base_simple/goal	geometry_msgs/PoseStamped	Provides a non-action interface for execution states that do not care about tracking targets. move_base subscribes to the target point to be reached.
cmd_vel	geometry_msgs/Twist	Publishes the speed of the car.

3) servies

Name	Type	Description
make_plan	nav_msgs/GetPlan	Allows external users 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 move_base's costmaps are stopped for a long period of time and then restarted at a new location in the environment.
clear_costmaps	std_srvs/Empty	Allows an external user to tell move_base to clear obstacles in the costmaps used by move_base. This may cause the robot to hit something, so use it 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 plugin 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 the DWA
(dynamic window method) local planning algorithm
# Recover 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'

# How often to send commands to the robot chassis cmd_vel
controller_frequency: 10.0
# How long the path planner waits for a valid control command before performing a
space clearing operation
planner_patience: 5.0
# How long the controller waits for a valid control command before performing a
space clearing operation
controller_patience: 15.0
# This parameter is only used when the default recovery behavior is used for
move_base.
conservative_reset_dist: 3.0
# Whether to enable the move_base recovery behavior to try to clear the space.
recovery_behavior_enabled: true
# Whether the robot uses the rotation in place movement to clear the space. This
parameter is only used when the default recovery behavior is used.
clearing_rotation_allowed: true
# Whether to deactivate the node's costmap when move_base enters the inactive
state
shutdown_costmaps: false
# The time allowed for oscillation before performing a recovery operation. 0
means never timeout
oscillation_timeout: 10.0
# The robot needs to move this distance to be considered as no oscillation. Reset
the timer parameter after the move is completed
oscillation_distance: 0.2
# Global path planner loop rate.
planner_frequency: 5.0
#The number of planning retries allowed before recovery behavior is performed. A
value of -1.0 corresponds to unlimited retries.
max_planning_retries: -1.0
```

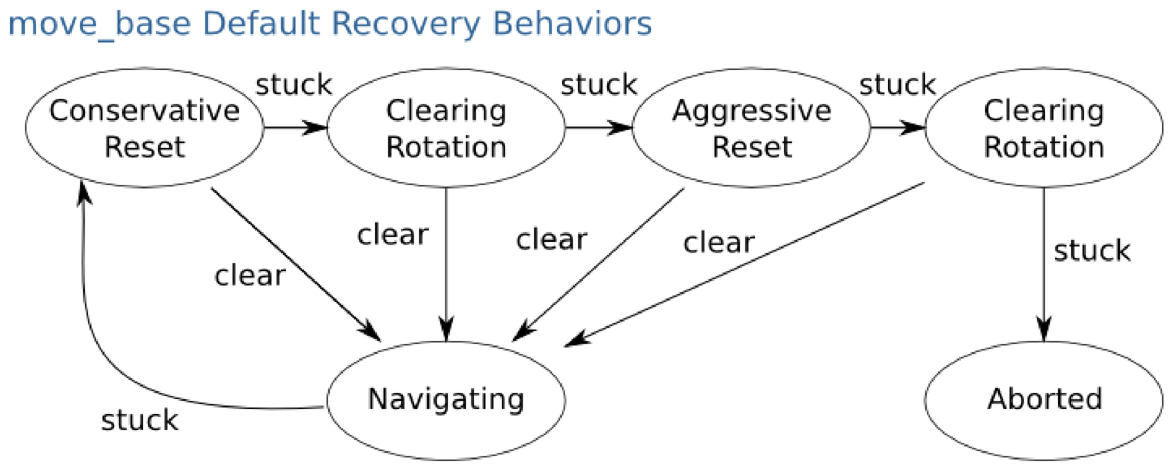
8.3.3、Recovery Behavior

1) Introduction

Recovery behaviors are entered when ① global planning fails, ② the robot oscillates, or ③ local planning fails. These recovery behaviors can be configured using the recovery_behaviour parameter and disabled using the recovery_behavior_enabled parameter.

Expected robot behavior

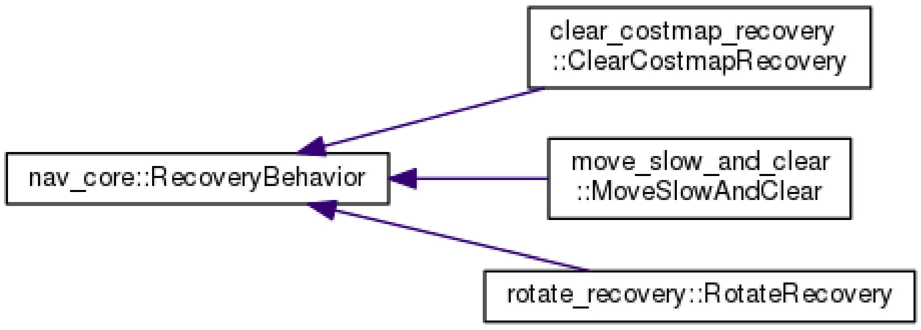
First, obstacles outside the user-specified area are cleared from the robot's map. Next, if possible, the robot will perform an in-place rotation to clear the space. If this also fails, the robot will clear the map more aggressively, clearing all obstacles outside the rectangular area within which the robot can rotate in-place. Another in-place rotation will follow. If all of these fail, the robot will consider its goal to be infeasible and notify 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 set, 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 3 packages, all of which inherit the interface specifications in nav_core.


clear_costmap_recovery ::ClearCostmapRecovery
nav_core::RecoveryBehavior
move_slow_and_clear ::MoveSlowAndClear
rotate_recovery::RotateRecovery

- move_slow_and_clear: is a simple recovery behavior that clears the information in the cost map and then limits the robot's speed. Note that this recovery behavior is not really safe, the robot may hit objects, and it will only occur at the speed specified by the user.

Parameter	Type	Default	Explanation
clearing_distance	double	0.5	Obstacles within the robot's clearing distance are cleared (in meters).
limited_trans_speed	double	0.25	The robot's translation speed will be limited when performing this recovery behavior (in meters per second).
limited_rot_speed	double	0.25	The robot's rotation speed will be limited when performing this recovery behavior (in rad/s).
limited_distance	double	0.3	The distance the robot must move before the speed limit is lifted (in meters).
planner_namespace	string	"DWAPlannerROS"	The name of the planner to reconfigure parameters for.

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

Parameter	Type	Default	Explanation
sim_granularity	double	0.017	The distance between obstacles to check for safety when checking if a rotation in place is safe, defaults to 1 degree (in rad).
frequency	double	20.0	How often to send velocity commands to the mobile robot (in Hz).
TrajectoryPlannerROS/yaw_goal_tolerance	double	0.05	The radian tolerance in yaw/rotation that the controller should have when achieving its goal.
TrajectoryPlannerROS/acc_lim_th	double	3.2	The rotational acceleration limit of the robot (in rad/s^2).
TrajectoryPlannerROS/max_rotational_vel	double	1.0	The maximum rotational velocity allowed for the base (in rad/s).
TrajectoryPlannerROS/min_in_place_rotational_vel	double	0.4	Minimum rotational speed allowed for the base when performing in-place rotations (in rad/s).

Note: TrajectoryPlannerROS parameters are only set when using the base_local_planner::TrajectoryPlannerROS planner; they do not normally need to be set.

- clear_costmap_recovery: A recovery action that reverts the costmap used by move_base to a static map outside of a user-specified range.

Parameter	Type	Default	Explanation
clearing_distance	double	0.5	The length centered on the robot that obstacles are removed from the costmap when they are restored to the static map.

8.4、costmap_params

Navigation uses two costmaps to store information about obstacles. One costmap is used for global planning, which means creating a global planned route in the entire environment, and the other is used for local planning and obstacle avoidance. These two costmaps have some common configurations and some individual configurations. Therefore, the costmap configuration has the following three parts: common configuration, global configuration, and local configuration.

8.4.1、costmap_common

Costmap common parameter configuration costmap_common_params_yb.yaml

```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 the information of obstacles within 2.5 meters.
- raytrace_range: The default value is 3.0 meters, which means that the robot will try to clear the space beyond 3.0 meters in front of it.
- footprint: Set the footprint of the robot. Fill in "footprint" according to the robot coordinate setting. When specifying the footprint, the center of the robot is considered to be at (0.0, 0.0). Both clockwise and counterclockwise settings are possible.
- robot_radius: The footprint of the robot is circular. Just set the radius. It cannot be shared with footprint.
- inflation_radius: Set the inflation radius of the cost map. The default value is 0.55. It means that the robot will consider obstacles within 0.55 meters of the obstacle.
- observation_sources: Defines a list of sensors that pass information to the cost map separated by spaces.
- laser_scan_sensor: Defines each sensor.
- sensor_frame: The name of the sensor coordinate system.
- data_type: The parameter is set to LaserScan or PointCloud, depending on the message used by the topic.
- topic: The name of the topic to which the sensor publishes data.
- marking: Whether to add obstacle information to the costmap.
- clearing: Whether to clear obstacle information from the costmap.

8.4.2、global_costmap

Global costmap parameter configuration global_costmap_params_yb.yaml

```
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 10.0
  publish_frequency: 5
  static_map: true

  transform_tolerance: 0.5
  plugins:
    - {name: static_layer,           type: "costmap_2d::StaticLayer"}
    - {name: obstacle_layer,         type: "costmap_2d::VoxelLayer"}
    - {name: inflation_layer,        type: "costmap_2d::InflationLayer"}
```

-global_frame: The coordinate system in which the global costmap operates.

- robot_base_frame: The coordinate system of the robot base referenced by the global costmap.
- update_frequency: The frequency of the global costmap loop update, in Hz.
- static_map: Whether the global costmap should be initialized based on the map provided by map_server. If not using an existing map or map server, set the static_map parameter to false.

8.4.3、local_costmap

Local costmap parameter configuration local_costmap_params_yb.yaml

```
local_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 4.5
  height: 4.5
  resolution: 0.05
  transform_tolerance: 0.5
  inflation_radius:     0.1
  plugins:
   - {name: static_layer,        type: "costmap_2d::StaticLayer"}
   - {name: obstacle_layer,      type: "costmap_2d::ObstacleLayer"}
   - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}
```

Parameter analysis

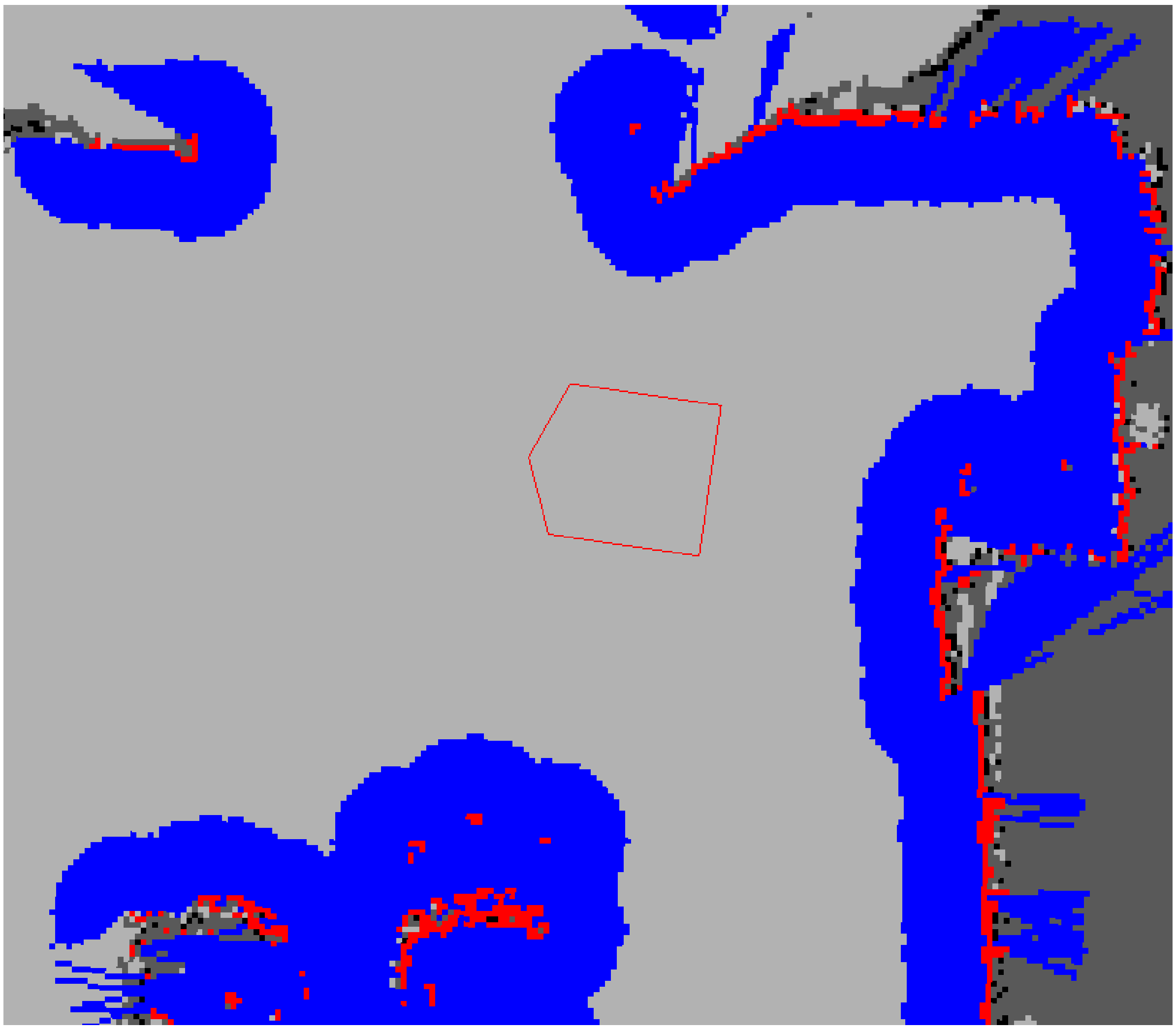
- global_frame: In which coordinate system the local cost map operates.
- robot_base_frame: The robot base coordinate system referenced by the local cost map.
- update_frequency: The frequency of the local cost map loop update, in Hz.
- publish_frequency: The rate at which the local cost map publishes visualization information, in Hz.
- static_map: Whether the local cost map should be initialized based on the map provided by map_server. If no existing map or map server is used, set the static_map parameter to false.

- rolling_window: (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: Local cost map height (meters).
- resolution: Local cost map resolution (meters/unit).

8.4.4、costmap_2D

1) Introduction

The costmap_2d package provides a 2D costmap implementation that takes input sensor data, builds a 2D or 3D costmap (depending on whether a voxel-based implementation is used), and computes the cost of the 2D costmap based on the occupancy grid and the user-defined inflation radius.



- Red represents obstacles in the costmap.
- Blue represents obstacles inscribed in the robot with an inflation radius.
- Red polygons represent the robot's footprint.
- In order for the robot to avoid collisions, the robot's shell must not intersect with red cells, and the robot's center point must not intersect with blue cells.

2) topic

Name	Type	Description
footprint	geometry_msgs/Polygon	Robot shell specification. This replaces the previous parameter specification for the footprint.
costmap	nav_msgs/OccupancyGrid	Costmap
costmap_updates	map_msgs/OccupancyGridUpdate	Update area of costmap
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.

Plugins

- plugins: Generally, the default is sufficient.

Coordinate system and tf parameters

- global_frame: The global coordinate system in which the costmap 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: The frequency of updating the map (unit: Hz).
- publish_frequency: How often to publish the map showing information (in Hz).

Map management parameters

- rolling_window: Whether to use a rolling window version of the costmap. This parameter must be set to false if the static_map parameter is set to true.
- always_send_full_costmap: If true, a full costmap is published to "/costmap" with each update. If false, only the parts of the costmap that have changed are published on the "/costmap_updates" topic.

Static layers

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

tf transformation

global_frame——>robot_base_frame

4) Layer specifications

- 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 the map_server to obtain this map.

Parameters

- unknown_cost_value: The cost value read from the map provided by the map server will be treated as unknown. A value of zero will also cause this parameter to be unused.
- lethal_cost_threshold: The threshold to consider fatal when reading the map 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, ignoring all subsequent messages.
- subscribe_to_updates: In addition to map_topic, subscribe to map_topic + "_updates".
- track_unknown_space: If true, unknown values in map messages are converted directly to the layer. Otherwise, unknown values in map messages are converted to free space in the layer.
- use_maximum: Only important if the static layer is not the bottom layer. If true, only the maximum value is written to the main costmap.
- trinary_costmap: If true, all map message values are converted to NO_INFORMATION/FREE_SPACE/LETHAL_OBSTACLE (three values). If false, the full range of intermediate values may appear.
- obstacle layer obstacle layer: The obstacle layer tracks obstacles read from sensor data. The collision costmap plugin marks and raytraces obstacles in 2D, while VoxelCostmapPlugin marks and raytraces obstacles in 3D.
- Inflation layer: Adds new values around deadly obstacles (i.e. inflates them) so that the costmap represents the robot's configuration space.
- Other layers: Other layers can be implemented and used in the costmap via pluginlib.
- Social Costmap Layer
- Range Sensor Layer

5) obstacle layer

The obstacle layer and voxel layer contain information from sensors in the form of point clouds or laser scans. The obstacle layer tracks in two dimensions, while the voxel layer tracks in three dimensions.

The costmap automatically subscribes to sensor topics and updates accordingly. Each sensor is used to mark (insert obstacle information into the costmap) and clear (remove obstacle information from the costmap). For each observation, the clear operation performs a ray trace from the sensor origin outward through the grid. At the voxel level, the obstacle information in each column is projected down into the 2D map.

Subscribe to Topics

Topic Name	Type	Parsing
point_cloud_topic	sensor_msgs/PointCloud	Update the PointCloud information to the costmap.
point_cloud2_topic	sensor_msgs/PointCloud2	Updates the costmap with PointCloud2 information
laser_scan_topic	sensor_msgs/LaserScan	Updates the costmap with LaserScan information
map	nav_msgs/OccupancyGrid	The costmap has the option to be initialized from a user-generated static map

Sensor management parameters

- observation_sources: List of observation source names

Each source name in observation sources defines a namespace where parameters can be set:

- <source_name>/topic: The topic the sensor data refers to.
- <source_name>/sensor_frame: The sensor. Can be sensor_msgs/LaserScan, sensor_msgs/PointCloud, and sensor_msgs/PointCloud2.
- <source_name>/observation_persistence: How long to keep each sensor reading (in seconds). A value of 0.0 will only keep the most recent reading.
- <source_name>/expected_update_rate: Frequency of sensor readings (in seconds). A value of 0.0 will allow an infinite time interval between readings.
- <source_name>/data_type: 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 meters) for a sensor reading to be considered valid. This is typically set to slightly above the height of the robot.
- <source_name>/min_obstacle_height: Minimum height (in meters) for a sensor reading to be considered valid. This is typically set to ground level, but can be set higher or lower depending on the noise model of the sensor.
- <source_name>/obstacle_range: Maximum range (in meters) for inserting obstacles into the costmap using sensor data.
- <source_name>/raytrace_range: Maximum range (in m) to ray trace obstacles from the map using sensor data.
- <source_name>/inf_is_valid: Allows Inf values to be entered in Laser Scan observation information. Inf values are converted to the laser max range

Global filter parameters: These parameters apply to all sensors.

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

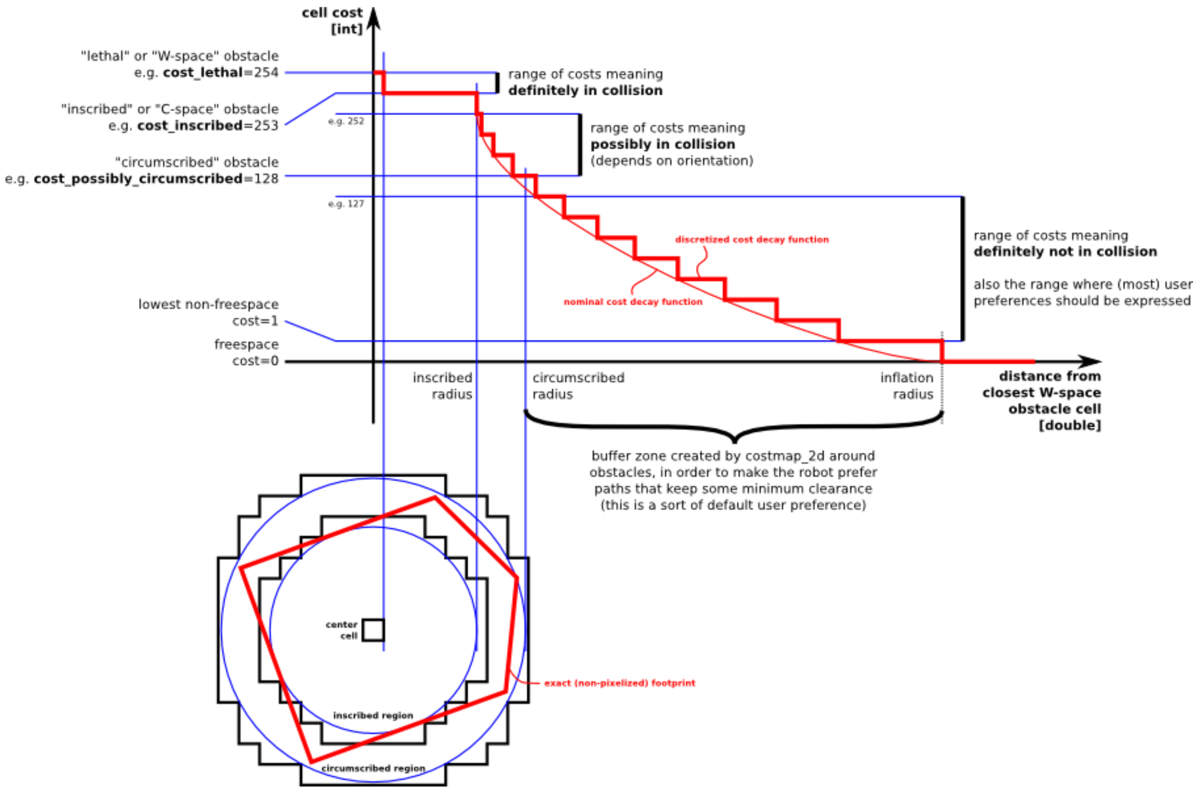
ObstacleCostmapPlugin

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

6) Inflation layer


cell cost [int]
"lethal" or "W-space" obstacle e.g. cost_lethal=254
"inscribed" or "C-space" obstacle e.g. cost_inscribed=253
"circumscribed" obstacle e.g. cost_possibly_circumscribed=128
e.g. 252
e.g. 127
lowest non-freespace cost=1
freespace cost=0
range of costs meaning definitely in collision
range of costs meaning possibly in collision (depends on orientation)
range of costs meaning definitely not in collision
also the range where (most) user preferences should be expressed
discretized cost decay function
nominal cost decay function
inscribed radius
circumscribed radius
inflation radius
distance from closest W-space obstacle cell [double]
buffer zone created by costmap_2d around obstacles, in order to make the robot prefer paths that keep some minimum clearance (this is a sort of default user preference)
center cell
inscribed region
circumscribed region
exact (non-pixelized) footprint

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

- Lethal cost: indicates that there is a real obstacle in the cell. If the robot's center is in the cell, the robot will definitely collide with the obstacle.
- Inscribed cost: indicates that the distance from the cell to the obstacle is less than the radius of the robot's inscribed circle. If the robot's center is in a cell with an "Inscribed" cost equal to or higher than the robot's inscribed circle, the robot will definitely collide with the obstacle.
- Possibly circumscribed cost: indicates that the distance from a cell to the obstacle is less than the robot's circumscribed circle radius, but greater than the inscribed circle radius. If the robot's center is in a cell with an "Possibly circumscribed" cost equal to or higher than the

robot's inscribed circle radius, the robot will not necessarily collide with the obstacle, depending on the robot's orientation.
- Freespace: Nothing can stop the robot from going there.
- Unknown: Unknown space.

Parameters

- inflation_radius: The radius to which the map inflates obstacle cost values (in meters).
- cost_scaling_factor: The scaling factor applied to the cost values during inflation.

8.5, planner_params

8.5.1, global_planner

nav_core::BaseGlobalPlanner provides an interface for global planners used in navigation. All global planners written as move_base node plugins must adhere to this interface. Documentation for navacys::BaseGlobalPlanner: C++ documentation can be found here: BaseGlobalPlanner documentation.

Global Path Planning Plugin

- navfn: A global planner based on a grid map that calculates the robot's path when using navigation functions. Implements the dijkstra and A* global planning algorithms. (Plugin name: "navfn/NavfnROS")
- global_planner: Reimplements the Dijkstra and A* global path planning algorithms, which can be seen as an improved version of navfn. (Plugin 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 is in an obstacle. (Plugin name: "carrot_planner/CarrotPlanner")

Global Path Planning global_planner_params.yaml

```
GlobalPlanner:                                  # Also see:
http://wiki.ros.org/global_planner
  old_navfn_behavior: false                     # Exactly mirror behavior of
navfn, use defaults for other boolean parameters, default false
  use_quadratic: false                           # Use the quadratic
approximation of the potential. Otherwise, use a simpler calculation, default
true
  use_dijkstra: true # Use dijkstra's algorithm. Otherwise, A*, default true
  use_grid_path: false                          # Create a path that follows the
grid boundaries. Otherwise, use a gradient descent method, default false

  allow_unknown: false                           # Allow planner to plan through
unknown space, default true
  default_tolerance: 0.2
  visualize_potential: false
                                                #Needs to have
track_unknown_space: true in the obstacle / voxel layer (in
costmap_commons_param) to work
  planner_window_x: 0.0                         # default 0.0
  planner_window_y: 0.0                         # default 0.0
  default_tolerance: 0.0                        # If goal in obstacle, plan to
the closest point in radius default_tolerance, default 0.0

  publish_scale: 100                            # Scale by which the published
potential gets multiplied, default 100
```

```
  planner_costmap_publish_frequency: 0.0        # default 0.0

  lethal_cost: 253                              # default 253
  neutral_cost: 66                              # default 50
  cost_factor: 0.55                              # Factor to multiply each cost
from costmap by, default 3.0
  publish_potential: true
```

Parameter analysis

- allow_unknown: Whether to explore unknown areas. It is not enough to just set this parameter to true. `track_unknown_space` must also be set to `true` in costmap_commons_params.yaml.
- default_tolerance: When the set destination is occupied by an obstacle, you need to find the nearest point with this parameter as the radius as the new destination.
- visualize_potential: Whether to display the possible area calculated from PointCloud2.
- use_dijkstra: If true, use the dijkstra algorithm. Otherwise, A*.
- use_quadratic: Set to true to use the quadratic function approximation function, otherwise use a simpler calculation method to save hardware computing resources.
- use_grid_path: If true, create a path along the grid boundary. Otherwise, use the gradient descent method, and 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: cost value of the fatal area of the obstacle (dynamically configurable).
- neutral_cost: neutral cost value of the obstacle (dynamically configurable).
- cost_factor: coefficient of the cost map multiplied by each cost value (dynamically configurable).
- publish_potential: whether to publish the potential cost map (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: get the direction of the window to be used according to the position integral specified by the orientation mode; the default value is 1 (dynamically configurable).
- outline_map: outline the global cost map with fatal obstacles. For the use of non-static (scrolling window) global cost map, it needs to be set to false

Global path planning algorithm effect diagram

- All parameters are default values
- use_grid_path=True
- use_quadratic=False
- use_dijkstra=False
- Dijkstra
- A*
- old_navfn_behavior=True

If 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 for the robot. It is recommended to try the default [navfn] plug-in for global path planning.

8.5.2、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 adhere to this interface. Documentation on the C++ API of nav_core::BaseLocalPlanner can be found here: BaseLocalPlanner documentation.

Local Path Planning Plugins

- 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 for circular, differential drive, forward drive (not backward), omnidirectional robots.
- mpc_local_planner： Provides several model predictive control approaches embedded in the SE2 manifold

Comparison between TEB and DWA:

TEB will adjust its position and orientation during movement. When reaching the target point, the robot's orientation is usually the target orientation without rotation.

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

For a two-wheel differential chassis, TEB's orientation adjustment during movement will make the movement path unsmooth, and unnecessary retreat will occur when starting and reaching the target point, which is not allowed in some application scenarios. Because retreating may hit obstacles. Rotating in place to the appropriate orientation and then walking away is a more appropriate movement strategy. 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. The package provides a controller that moves the robot in a plane. The controller connects a path planner to the robot. The planner uses the map to create a trajectory for the robot from a starting point to a goal position, sending the dx, dy, dtheta velocities to the robot.

The basic idea of the DWA algorithm

- Discrete samples (dx, dy, dtheta) in the robot's control space
- For each sampled velocity, perform a forward simulation from the robot's current state to predict what would happen if the sampled velocity was applied for a (short) period of time.
- Evaluate (score) each trajectory produced by the forward simulation, using a metric that includes the following features: proximity to obstacles, proximity to goal, proximity to global path, and velocity. Discard illegal trajectories (trajectories that collide with obstacles).

- Select the trajectory with the highest score and send the associated velocity to the mobile robot.
- Clean the data and repeat.

A large 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 tuned using the dynamic_reconfigure tool to facilitate tuning the local path planner in a running system.

```
DWAPlannerROS:
# Robot Configuration Parameters
  max_vel_x: 0.5 #The absolute value of the maximum linear velocity in the x direction, unit: m/s
  min_vel_x: 0.0 #The absolute value of the minimum linear velocity in the x direction, negative number means you can move backward, unit: m/s
  max_vel_y: 0.0 # #The absolute value of the maximum linear velocity in the y direction, unit: m/s. 0 for differential drive robots
  min_vel_y: 0.0 # #The absolute value of the minimum linear velocity in the y direction, unit: m/s. 0 for differential drive robots
  max_vel_trans: 0.0 # #The absolute value of the robot's maximum translation velocity, unit: m/s
  min_vel_trans: 0.0 # #The absolute value of the robot's minimum translation velocity, unit: m/s. Cannot be zero
  trans_stopped_vel: 0.05 #The translation velocity when the robot is considered to be in the "stopped" state. If the robot's speed is lower than this value, it is considered to have stopped. The unit is m/s
  max_vel_theta: 0.2 #The absolute value of the robot's maximum rotational angular velocity, in rad/s
  min_vel_theta: 0.1 #The absolute value of the robot's minimum rotational angular velocity, in rad/s
  theta_stopped_vel : 0.1 #The rotational velocity of the robot when it is considered to be in the "stopped" state. The unit is rad/s
  acc_lim_x: 0.5 #The robot's limit acceleration in the x direction, in meters/sec^2
  acc_lim_theta: 3.5 #The robot's limit rotational acceleration, in rad/sec^2
  acc_lim_y: 0.0 #The robot's limit acceleration in the y direction, of course, is 0 for differential robots
# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.15 #When reaching the target point, the controller's radian tolerance in yaw/rotation. That is: the allowable error of the deviation angle when reaching the target point, in radians
  xy_goal_tolerance: 0.2 # The tolerance of the controller in the x and y directions when reaching the target point (meters). That is: the distance error with the target point in the xy plane when reaching the target point
# Forward Simulation Parameters
  sim_time: 10 # The time of the forward simulation trajectory, in seconds
  vx_samples: 40 # The number of sampling points in the x-direction velocity space
  vy_samples: 1 # The number of sampling points in the y-direction velocity space. Differential drive robots always have only 1 value (0.0) in the y direction
  vtheta_samples: 80 #Number of velocity space sampling points in the rotation direction
# Trajectory Scoring Parameters
  path_distance_bias: 50.0 #The weight of the controller's proximity to a given path
  goal_distance_bias: 24.0 #The weight of the controller's proximity to a local goal point, also used for speed control
  occdist_scale: 0.5 #The degree to which the controller avoids obstacles
  forward_point_distance: 0.325 #The distance to place an additional scoring point with the robot as the center
  stop_time_buffer: 0.2 #The minimum amount of time the robot must have before a collision occurs. The trajectory used during this time is still considered valid. That is: the length of time the robot must stop in advance to prevent a collision
  scaling_speed: 0.25 #The absolute value of the speed when scaling the robot's footprint, in m/s.
  max_scaling_factor: 0.2 #The maximum scaling factor. max_scaling_factor is the value of the above formula.
# Oscillation Prevention Parameters
  oscillation_reset_dist: 0.05  # default 0.05
# Global Plan Parameters
  prune_plan: false
```

Robot configuration parameters

- acc_lim_x: The robot's x acceleration limit (unit: m/s^2).
- acc_lim_y: The absolute value of the acceleration in the y direction (unit: m/s^2). Note: This value only needs to be configured for omnidirectional robots.
- acc_lim_th: The absolute value of the 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: The absolute value of the minimum speed in the x direction (unit: m/s). If it is a negative value, it means that you can move backward.
- max_vel_y: The absolute value of the maximum speed in the y direction (unit: m/s).
- min_vel_y: The absolute value of the minimum speed 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: the allowable error of the yaw 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 the robot reaches the tolerance distance, it will rotate in place, even if it turns out of the tolerance distance.

Forward simulation parameters

- sim_time: the time to simulate the trajectory forward (unit: s).
- sim_granularity: the step size between each point on a given trajectory (unit: m).
- vx_samples: the number of sampling points in the velocity space in the x direction.
- vy_samples: the number of sampling points in the velocity space in the y direction.
- vth_samples: the number of sampling points in the velocity space in the rotation direction.
- controller_frequency: How often this controller is called (in Hz).

Trajectory scoring parameters

- path_distance_bias: Defines the weight of how close the controller is to a given path.
- goal_distance_bias: Defines the weight of how close the controller is to a local goal point.
- occdist_scale: Defines the weight of how the controller avoids obstacles.

- forward_point_distance: The distance from the robot center point to where the extra scoring point is placed (in meters).
- stop_time_buffer: How long the robot must stop before a collision (in seconds).
- scaling_speed: The speed at which the robot chassis is scaled (in meters/second).
- max_scaling_factor: The maximum scaling factor of the robot chassis.
- publish_cost_grid: Whether to publish the cost grid of the planner when planning a path. If set to true, a sensor_msgs/PointCloud2 type message will be published on the topic ~/cost_cloud.

Isolation parameters

- oscillation_reset_dist: How far the robot has to move before the oscillation flag is reset (in meters).

Global planning parameters

- prune_plan: Whether to clear the track 1m behind the robot when it moves forward.

2) teb_local_planner

teb_local_planner is an optimization-based local trajectory planner. Supports differential models, car-like models. The package implements an online optimal local trajectory planner for mobile robot navigation and control, which efficiently obtains the optimal trajectory by solving a sparse scalarized multi-objective optimization problem. Users can provide weights to the optimization problem to specify the 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 them are selected to conform to the basic local planner. Many (but not all) parameters can be modified at runtime using rqt_reconfigure.

Local Path Planner teb_local_planner_params.yaml

```
TebLocalPlannerROS:
# Miscellaneous Parameters
  odom_topic: /odom
  map_frame: map
# Robot
  max_vel_x: 0.7 #Maximum x forward speed
  max_vel_y: 0.0 #Maximum y forward speed, non-omnidirectional mobile car needs
to be set to 0
  max_vel_x_backwards: 0.3 #Maximum backward speed 0.5
  max_vel_theta: 2 #Maximum steering angular velocity 0.6
  acc_lim_x: 1 #Maximum x acceleration 0.5
  acc_lim_y: 0.0 #Maximum y acceleration, non-omnidirectional mobile car needs
to be set to 0
  acc_lim_theta: 0.5 #Maximum angular acceleration 0.6
# GoalTolerance
  xy_goal_tolerance: 0.2
  yaw_goal_tolerance: 0.1
  free_goal_vel: False
# Trajectory
  teb_autosize: True # Allows changing the time domain length of the trajectory
during optimization
  dt_ref: 1 # Local path planning resolution # minimum 0.01
  dt_hysteresis: 0.1 # The floating range of the time domain resolution allowed
to change, generally about 10% of dt_ref minimum 0.002
```

```
# Obstacles
  inflation_dist: 0.8
  include_costmap_obstacles: True #Whether to predict dynamic obstacles as
velocity models,
  costmap_obstacles_behind_robot_dist: 1.0 #Limit the local costmap obstacles
considered when planning behind the robot
  obstacle_poses_affected: 10 #Obstacle poses affected 0~30
# Optimization
  no_inner_iterations: 2   # 5
  no_outer_iterations: 2   # 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: 1000
  weight_kinematics_turning_radius: 1000
  weight_optimaltime: 0.3 # must be > 0
  weight_shortest_path: 0
  weight_obstacle: 100
  weight_inflation: 0.2
  weight_dynamic_obstacle: 10 # not in use yet
  weight_dynamic_obstacle_inflation: 0.2
  weight_viapoint: 1000
  weight_adapt_factor: 2
 # Homotopy Class Planner
  enable_homotopy_class_planning: False
  enable_multithreading: True
  max_number_classes: 4
  selection_cost_hysteresis: 1.0
  selection_prefer_initial_plan: 0.95
  selection_obst_cost_scale: 1.0
  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: True
# Recovery
  shrink_horizon_backup: True
  shrink_horizon_min_duration: 10
  oscillation_recovery: False
  oscillation_v_eps: 0.1
  oscillation_omega_eps: 0.1
  oscillation_recovery_min_duration: 10
  oscillation_filter_duration: 10
```

Robot configuration

- acc_lim_x: Maximum translational acceleration of the robot (in m/s^2).
- acc_lim_theta: Maximum angular acceleration of the robot (in rad/s^2).
- max_vel_x: Maximum translational velocity of the robot (in m/s).
- max_vel_x_backwards: Maximum absolute translational velocity of the robot when driving backwards (in m/s).
- max_vel_theta: Maximum angular velocity of the robot (in rad/s).

The following parameters are only relevant for carlike robots

- min_turning_radius: Minimum turning radius for carlike robots (set to zero for differential robots).
- wheelbase: Distance between rear and front axles. For rear-wheel robots this value may be negative (only needed if /cmd_angle_instead_rotvel is set to true).
- cmd_angle_instead_rotvel: Replace the rotational speed in the command velocity information with the corresponding steering angle [-pi/2, pi/2].

The following parameters are only relevant for holo robots: New parameters in ROS dynamics

- max_vel_y: Maximum sweep velocity of the robot (should be zero for non-omnidirectional robots!).
- acc_lim_y: Maximum sweep acceleration of the robot.

The following parameters are relevant for the chassis model used for optimization

- footprint_model:

type: "point":

Parameter [footprint_model]

Specifies the type of robot footprint model used for optimization. The different types are "point", "circular", "line", "two_circles", "polygon". The type of model significantly affects the required computation 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 robot's rotation axis.
- 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 end coordinates 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 is shifted along the robot's x-axis. Assume the robot's rotation axis is 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 rear circle is shifted along the robot's negative x-axis. Assume the robot's rotation axis is at [0,0].
- footprint_model/rear_radius: This parameter is only relevant for the ''two_circles'' type. It contains the radius of the rear circle.
- footprint_model/vertices: This parameter is only relevant for the ''polygon'' type. It contains the list of polygon vertices (2D coordinates of each vertex). Polygons are always closed: do not repeat the first vertex at the end.
- is_footprint_dynamic: If true, the footprint is updated before checking the feasibility of the trajectory.

Goal tolerances

- yaw_goal_tolerance: The tolerance allowed in the yaw angle when reaching the goal point (in rad).
- xy_goal_tolerance: The tolerance allowed in the x&y distance when reaching the goal point (in m).
- free_goal_vel: Removes the goal velocity constraint, allowing the robot to reach the goal at maximum velocity.

Trajectory configuration

- dt_ref: The desired time resolution of the trajectory.
- dt_hysteresis: A hysteresis that is automatically sized according to the current time resolution, usually about 10% of dt_ref is recommended.
- max_samples: Minimum number of samples (should always be greater than 2).
- global_plan_overwrite_orientation: Overwrite the orientation of the local subgoals provided by the global planner (as they usually only provide 2D paths)
- global_plan_viapoint_sep: If positive, extract viapoints from the global plane (path following mode). This value determines the resolution of the reference path (minimum spacing 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 when optimizing. The actual length is determined by the size of the local costmap and the logical connection of this maximum bound. Set to zero or negative to deactivate this limit.
- force_reinit_new_goal_dist: reinitialize the trajectory if the update interval of the previous goal exceeds the specified number of meters (skipping warm start).
- feasibility_check_no_poses: specifies which poses in the predicted plan should be checked for feasibility every sampling interval.
- publish_feedback: publishes planner feedback containing the full trajectory and a list of active obstacles (should be enabled only for evaluation or debugging)
- shrink_horizon_backup: allows the planner to temporarily shrink the horizon (by 50%) if a problem (e.g. infeasibility) is detected automatically.
- allow_init_with_backwards_motion: If true, the base trajectory may be initialized with backwards motion if the goal is after the start point in the local costmap (recommended only if the robot is equipped with a rear sensor).
- exact_arc_length: If true, the planner uses exact arc lengths in velocity, acceleration and turn rate calculations (-> increased cpu time), otherwise uses Euclidean approximations.
- shrink_horizon_min_duration: Specifies the minimum duration for shrinking the horizon in case an infeasible trajectory is detected.

Obstacles

- min_obstacle_dist: Minimum desired distance to an obstacle (in m).
- include_costmap_obstacles: Specifies whether obstacles from the local costmap should be considered.
- costmap_obstacles_behind_robot_dist: Limits the occupied local costmap obstacles considered for planning behind the robot (in m).
- obstacle_poses_affected: Each obstacle position is attached to the nearest pose on the trajectory to maintain distance.
- inflation_dist: Buffer around non-zero penalty cost obstacles (should be larger than minimum obstacle distance in order to be effective).
- include_dynamic_obstacles: If this parameter is set to true, the motion of obstacles with non-zero velocity is predicted and considered by the constant velocity model during optimization.

- legacy_obstacle_association: Modified the strategy for associating trajectory poses with obstacles for optimization.

The following parameters are only relevant if the costmap_converter plugin is required:

- costmap_converter_plugin: ""

Defines the name of the plugin to convert costmap cells to points/lines/polygons. Set an empty string to disable conversion so that all cells are considered 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 rate (in Hz) of how often the costmap_converter plugin processes the current costmap.

Optimization

- no_inner_iterations: The actual number of solver iterations called in each outer loop iteration.
- no_outer_iterations: Each outer loop iteration automatically resizes the trajectory according to the desired time resolution dt_ref and calls the inner optimizer (no inner iterations are performed).
- penalty_epsilon: Adds a small safety margin to the penalty function for hard constraint approximations.
- weight_max_vel_x: Optimization weight for satisfying the maximum allowed translational velocity.
- weight_max_vel_theta: Optimization weight for satisfying the maximum allowed angular velocity.
- weight_acc_lim_x: Optimization weight for satisfying the maximum allowed translational acceleration.
- weight_acc_lim_theta: Optimization weight for satisfying the maximum allowed angular acceleration.
- weight_kinematics_nh: Optimization weight for satisfying non-holonomic kinematics.
- weight_kinematics_forward_drive: Optimization weight for forcing the robot to choose only forward directions (positive translational velocity).
- weight_kinematics_turning_radius: Optimization weight for implementing minimum turning radius (only for carlike robots).
- weight_optimaltime: Optimization weight for shortening trajectories with respect to transition/execution time
- weight_obstacle: Optimization weight for keeping minimum distance to obstacles.
- weight_viapoint: Optimization weight for minimizing distance to viapoint (resp. reference path).
- weight_inflation: Optimization weight for inflation penalty (should be small).
- weight_adapt_factor: Some special weights (current weights) are repeatedly scaled by this factor at each outer TEB iteration.

Planning

- enable_homotopy_class_planning: Activate parallel planning in unique topologies.
- enable_multithreading: Activate multithreading to plan each trajectory in a different thread.
- max_number_classes: Specify the maximum number of different trajectories considered.
- selection_cost_hysteresis: Specify how much trajectory cost a new candidate trajectory must have to be selected.
- selection_obst_cost_scale: Additional expansion of obstacle cost condition.
- selection_viapoint_cost_scale: Additional extension of viapoint cost criteria.

- 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 the roadmap.
- roadmap_graph_area_width: Random keypoints/waypoints are sampled in a rectangular area between the start and goal (in m).
- obstacle_heading_threshold: Specifies the value of the scalar product between obstacle headings and goal headings in order to consider them (obstacles) while exploring.
- visualize_hc_graph: Visualize the graph created for exploring unique trajectories.
- viapoints_all_candidates: If true, all trajectories of different topologies will be connected to the viapoints set, otherwise, only trajectories that share the same topology as the initial/global plan will be connected to it.
- switching_blocking_period: Specifies the duration (in s) that needs to expire before switching to a new equivalence class is allowed.