# 8、navigation

Note: This course takes Rosmaster-X3 as an example. Users need to modify according to their own motion model, which is different from the content of handheld laser radar mapping. This mapping adds odom data, so users need to have odom data in their own motion model.

Function package path:~/oradar_ ws/src/yahboomcar_ nav

## 8.1、Start

```
roslaunch yahboomcar_nav laser_bringup.launch
roslaunch yahboomcar_nav yahboomcar_navigation open_rviz:=true map:=house
```

- open_ Rviz parameter: whether to open rviz
- Map: map name, map to load

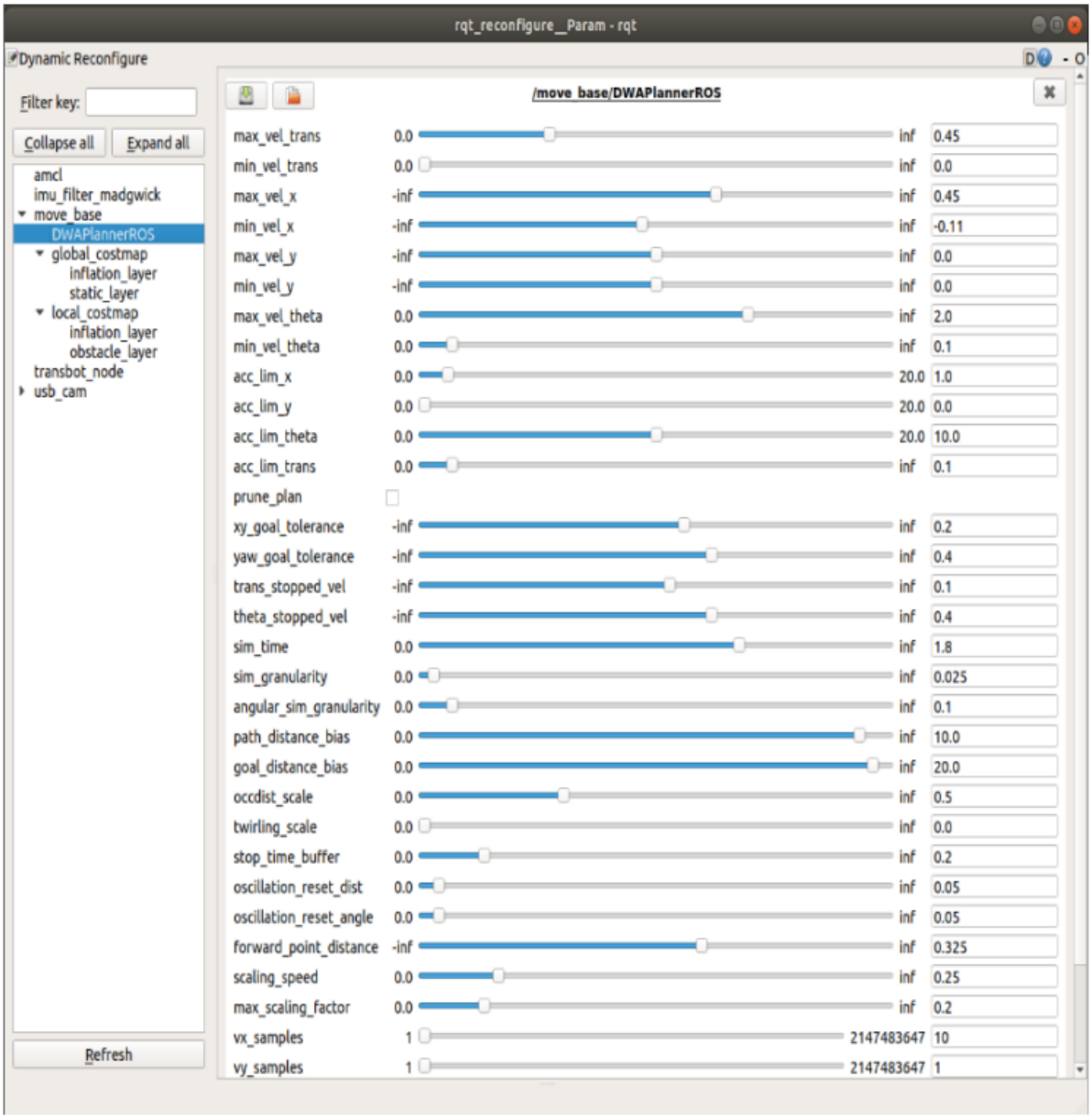
## 8.2、Usage

- Place the robot at the origin. If the radar scanning edge does not coincide with the map, you need to use the [2D PoseEstimate] of the [rviz] tool to set the initial position and pose. If the robot cannot find the position and pose in the map, you also need to set the initial position and pose.
- Click the [2D Nav Goal] of the [rviz] tool, and then select the target point in the place where there is no obstacle on the map. Release the mouse to start navigation. Only one target point can be selected and stop when it is reached.
- Multi-point navigation: click [Publish Point] of [rviz] tool, and then select the target point in the place without obstacles on the map. Release the mouse to start navigation. Click [Publish Point] again, and then select a point. The robot will cruise between points.

## 8.3、Dynamic parameter adjustment

Input following command.

```
rosrun rqt_reconfigure rqt_reconfigure
```



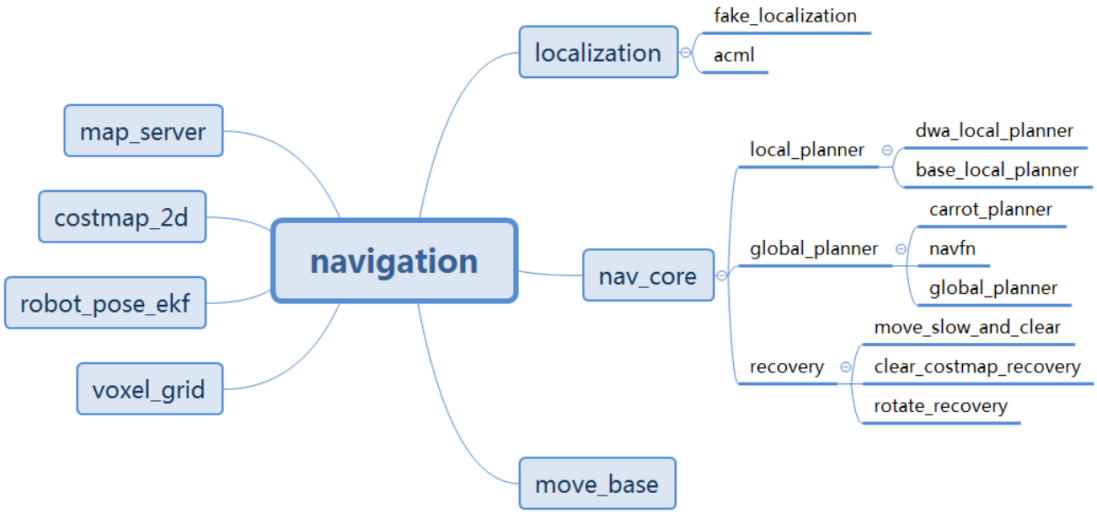

Input following command to view the node relationship and TF tree.

```
rqt_graph
rosrun rqt_tf_tree rqt_tf_tree
```

# 8.3、navigation

## 8.3.1、Introduction

Navigation is the 2D navigation obstacle avoidance function package of ROS. In short, it is to calculate the safe and reliable robot speed control command through the navigation algorithm according to the information flow of the input odometer and other sensors and the global position of the robot.

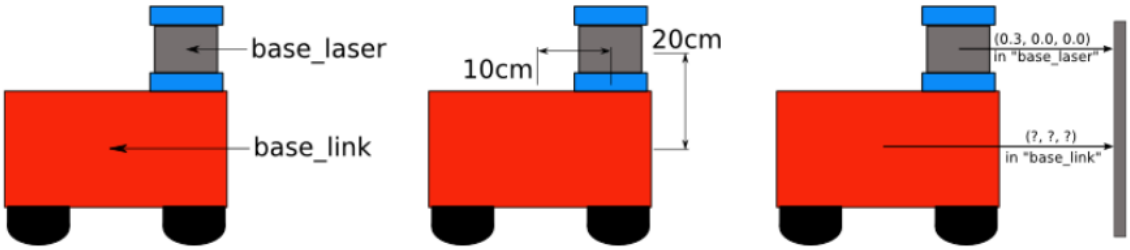



Main nodes and configurations of navigation:

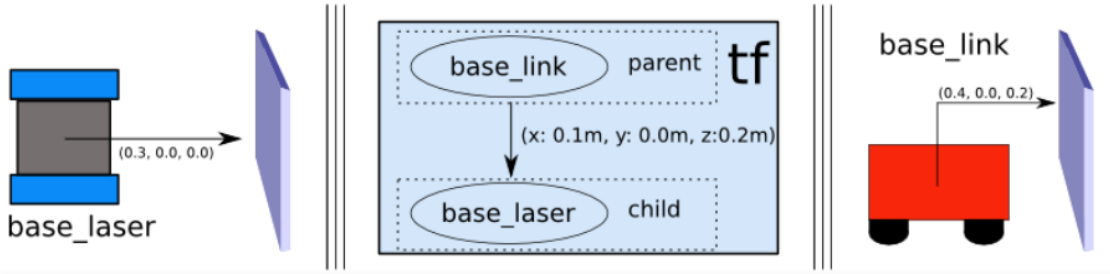
- move_ Base: the final actuator of navigation obstacle avoidance control, move_ Base subscription navigation target move_ base_ Simple/goal, and real-time release of motion control signal cmd_ vel， move_ Various navigation algorithm modules in base are called in the form of plug-ins.
- global_ Planner: used for global path planning.
- local_ Planner: used for local path planning.
- global_ Costmap: The global cost map is used to describe the 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: the particle filter algorithm is used to realize the global positioning of the robot and provide the global position information for the robot navigation.
- map_ Server: The map obtained by calling SLAM mapping provides environment 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 odometer, IMU and VO, and the output is a fused pose.
- nav_ Core: There are only three files in it, which correspond to global path planning, local path planning and recovery_ The general interface definition of action and the specific function implementation are in each corresponding planner function package.
- TF information related to robot model, odometer odom information, laser radar information scan.

### 8.3.2、Set TF

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



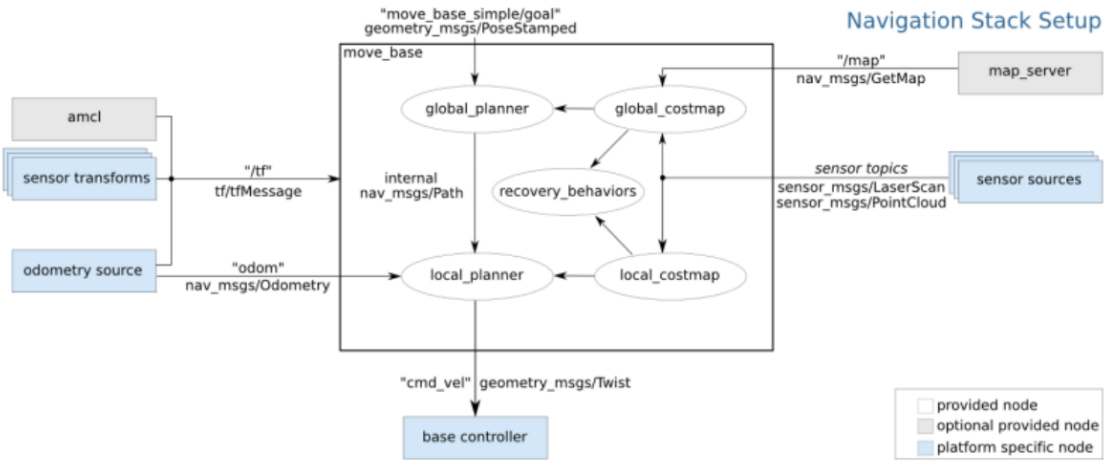

Suppose we know that the lidar is mounted 10 cm and 20 cm above the center point of the mobile base. This gives us the translation offset to associate the "base_link" frame with the "base_laser" frame. Specifically, we know that to get data from the "base_link" coordinate system to the "base_laser" coordinate 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).




## 8.4、move_base

### 8.4.1、Introduction

move_base provides the configuration, operation and interaction interface of 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,
  - Grey components are optional components that have been implemented,
  - Blue components must be created for each robot platform.

### 8.4.2、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 state of the target, it is recommended to use SimpleActionClient to send the target to the move_base.(See actionlib documentation )

| name | type | illustrate |
| --- | --- | --- |
| 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 cancel requests for a specific target. |
| move_base/feedback | move_base_msgs/MoveBaseActionFeedback | The post contains the current position of the chassis. |
| move_base/status | actionlib_msgs/GoalStatusArray | Publishes status information for the move to the target point process. |
| move_base/result | move_base_msgs/MoveBaseActionResult | Post the final result of the move. |

#### 2) topic

| name | type | illustrate |
| --- | --- | --- |
| move_base_simple/goal | geometry_msgs/PoseStamped | Provides a non-action interface that does not care about the execution state of the tracking target. move_base subscribes to the target point to be reached. |
| cmd_vel | geometry_msgs/Twist | Publishes the speed of the car's movement. |

**3) tableware**

| name | type | illustrate |
| --- | --- | --- |
| 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 spaces 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 external users to tell move_base to clear barriers in the costmap 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

```
# 建议差速车开启DWA导航算法，非差速车开启TEB导航算法。
# 设置move_base的全局路径规划器的插件名称
# The name of the plugin for the global planner to use with move_base, see
pluginlib documentation for more details on plugins.
#base_global_planner: "navfn/NavfnROS"
#base_global_planner: "global_planner/GlobalPlanner"
#base_global_planner: "carrot_planner/CarrotPlanner"

# 设置move_base的局部路径规划器的插件名称
# The name of the plugin for the local planner to use with move_base see
pluginlib documentation for more details on plugins.
#base_local_planner: "teb_local_planner/TebLocalPlannerROS"
#base_local_planner: "dwa_local_planner/DWAPlannerROS"

# 恢复行为。
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'


# 向机器人底盘cmd_vel发送命令的频率
```

```yaml
# The rate in Hz at which to run the control loop and send velocity commands to
the base.
controller_frequency: 10.0
# 空间清理操作执行前，路径规划器等待有效控制命令的时间
# How long the planner will wait in seconds in an attempt to find a valid plan
before space-clearing operations are performed.
planner_patience: 5.0
# 空间清理操作执行前，控制器等待有效控制命令的时间
# How long the controller will wait in seconds without receiving a valid control
before space-clearing operations are performed.
controller_patience: 5.0
# 仅当默认恢复行为用于 move_base 时才使用此参数。
# The distance away from the robot in meters beyond which obstacles will be
cleared from the costmap when attempting to clear space in the map.
conservative_reset_dist: 5.0
# 是否启用move_base恢复行为以尝试清除空间。
# Whether or not to enable the move_base recovery behaviors to attempt to clear
out space.
recovery_behavior_enabled: true
# 机器人是否采用原地旋转的运动方式清理空间,此参数仅在使用默认恢复行为时使用。
# Determines whether or not the robot will attempt an in-place rotation when
attempting to clear out space.
clearing_rotation_allowed: true
# 当move_base进入inactive状态时，是否停用节点的costmap
# Determines whether or not to shutdown the costmaps of the node when move_base
is in an inactive state
shutdown_costmaps: false
# 执行恢复操作之前允许震荡的时间，0代表永不超时
# How long in seconds to allow for oscillation before executing recovery
behaviors.
oscillation_timeout: 10.0
# 机器人需要移动该距离才可当做没有震荡。移动完毕后重置定时器参数
# How far in meters the robot must move to be considered not to be oscillating.
oscillation_distance: 0.3
# 全局路径规划器循环速率。
# The rate in Hz at which to run the global planning loop.
planner_frequency: 10.0 #0.0
# 在执行恢复行为之前允许计划重试的次数。值-1.0对应于无限次重试。
# How many times to allow for planning retries before executing recovery
behaviors.
max_planning_retries: -1.0

conservative_reset:
  reset_distance: 1.0
  #layer_names: [static_layer, obstacle_layer, inflation_layer]
  layer_names: [obstacle_layer]

aggressive_reset:
  reset_distance: 3.0
  #layer_names: [static_layer, obstacle_layer, inflation_layer]
  layer_names: [obstacle_layer]

super_reset:
  reset_distance: 5.0
  #layer_names: [static_layer, obstacle_layer, inflation_layer]
```

```
  layer_names: [obstacle_layer]

move_slow_and_clear:
  clearing_distance: 0.1
  limited_trans_speed: 0.1
  limited_rot_speed: 0.4
  limited_distance: 0.3
```

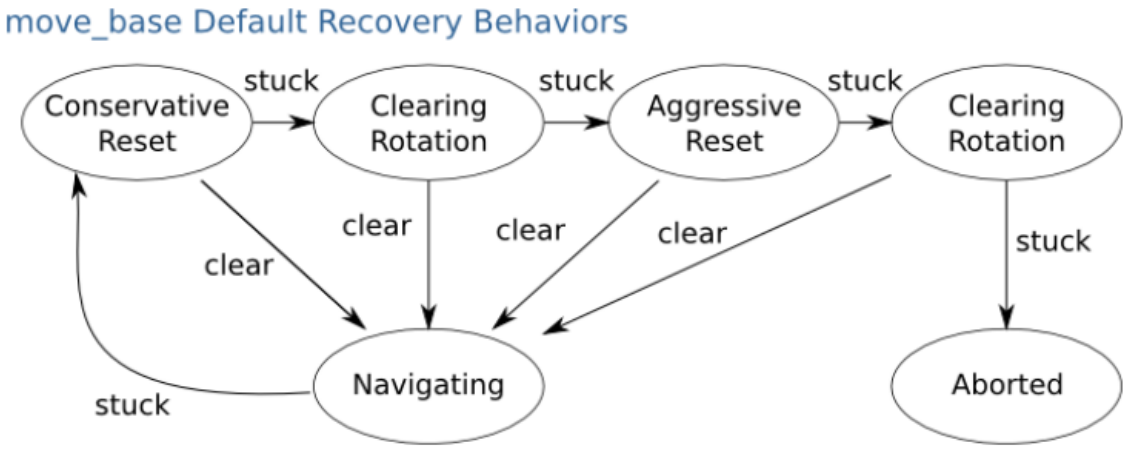
## 8.5、Recovery Behavior

### 1) Introduction

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

Desired Robot Behavior

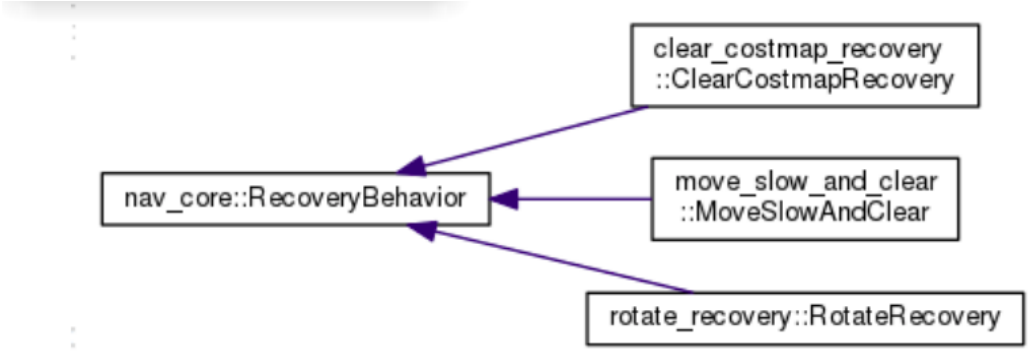
First, obstacles outside the user-specified area are cleared from the robot's map. Next, if possible, the robot will perform a spin in place to clear the space. If this also fails, the robot will more aggressively clear the map, clearing all obstacles outside the rectangular area where the robot can rotate in place. Another in-place spin will follow. If all of these fail, the bot considers its goal unfeasible and informs 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 set of navigation function packs, there are 3 packs 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 specification in nav_core.




- move_slow_and_clear : is a simple recovery behavior that clears the information in the costmap and then limits the speed of the robot. Note that this recovery behavior is not really safe, the robot may hit the object, it will only happen at the speed specified by the user.

| parameter | type | Defaults | 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 this recovery behavior is performed, the rotational speed of the robot will be limited(unit: rad/s). |
| limited_distance | double | 0.3 | The distance(unit: m) that the robot must move before releasing the speed limit. |
| 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.

| parameter | type | Defaults | Parse |
| --- | --- | --- | --- |
| sim_granularity | double | 0.017 | When checking whether it is safe to rotate in place, the distance between checking obstacles is 1 degree by default(unit: rad). |
| frequency | double | 20.0 | The frequency(unit: Hz) of sending speed commands to the mobile robot. |
| TrajectoryPlannerROS/yaw_goal_tolerance | double | 0.05 | The tolerance in radians for the controller in yaw/rotation to achieve 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(unit: rad/s) allowed by the base when performing an in-position rotation. |

Note: The TrajectoryPlannerROS parameter is only set when using the base_local_planner::TrajectoryPlannerROS planner; generally it is not required.

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

| parameter | type | Defaults | Parse |
| --- | --- | --- | --- |
| clearing_distance | double | 0.5 | The length centered on the robot that the obstacle will be removed from the costmap when it reverts to a static map. |

## 8.6、costmap_params

The navigation function uses two costmaps to store information about obstacles. One costmap is used for global planning, which means creating global planning routes across the entire environment, and the other is used for local planning and obstacle avoidance. The two costmaps have some common configuration and some individual configuration. Therefore, the costmap configuration has the following three parts: general configuration, global configuration and local configuration.

### 8.6.1、costmap_common

Costmap public parameter configuration costmap_common_params.yaml

```
obstacle_range: 3.0
raytrace_range: 3.5

footprint: [[-0.117, -0.1], [-0.117, 0.1], [0.117, 0.1], [0.117, -0.1]]
#robot_radius: 0.105

# 膨胀半径
inflation_radius: 0.3
cost_scaling_factor: 3.0

map_type: costmap

obstacle_layer:
  # 使能障碍层
  enabled:              true
  # 最大障碍物高度
  max_obstacle_height:  2.0
  min_obstacle_height:  0.0
  combination_method:   1
  # true needed for disabling global path planning through unknown space
  track_unknown_space:  true
  # 机器人更新代价地图中的障碍物距离基坐标系的阈值。2.0
  # The robot updates the threshold of the obstacle distance from the base
coordinate system in the cost map.
  obstacle_range: 3.0
  # 机器人清除代价地图中的障碍物距离基坐标系的阈值。3.0
  # The robot clears obstacles in the cost map from the threshold of the base
coordinate system.
  raytrace_range: 3.5
  publish_voxel_map: false
  observation_sources:  scan
  scan:
    sensor_frame: laser
    data_type: LaserScan
    topic: "scan"
    marking: true
    clearing: true
    expected_update_rate: 0

#cost_scaling_factor and inflation_radius were now moved to the inflation_layer
ns
inflation_layer:
```

```
  enabled:              true
  # exponential rate at which the obstacle cost drops off
  cost_scaling_factor:  1.0

static_layer:
  enabled:              true
  map_topic:            "/map"
```

#### 8.6.2、loca_costmap

Local costmap parameter configuration local_costmap_params.yaml

```
local_costmap:
  # Coordinate frame and tf parameters
  # The "global_frame" parameter defines what coordinate frame the costmap
should run in, in this case, we'll choose the /map frame.
  global_frame: odom
  # 机器人的基坐标系
  # The "robot_base_frame" parameter defines the coordinate frame the costmap
should reference for the base of the robot.
  robot_base_frame: base_footprint
  # 指定可容忍的转换（tf）数据延迟（以秒为单位）。
  # Specifies the delay in transform (tf) data that is tolerable in seconds.
  transform_tolerance: 0.5
  # 代价地图更新的频率（以Hz为单位）。
  # The "update_frequency" parameter determines the frequency, in Hz, at which
the costmap will run its update loop.
  update_frequency: 10.0
  # 代价地图发布可视化信息的速率（以Hz为单位）。
  # The rate at which the cost map publishes visual information
  publish_frequency: 10.0
  # 如果为true，则由map_server提供的地图服务来进行代价地图的初始化，否则为false。
  # If true, the map service provided by the map server will initialize the cost
map, otherwise it will be false.
  static_map: false
  # 是否使用滚动窗口版本的costmap。如果static_map参数设置为true，则该参数必须设置为false。
  # Whether or not to use a rolling window version of the costmap.
  # If the static_map parameter is set to true, this parameter must be set to
false.
  rolling_window: true
  # 代价地图宽度、高度、分辨率（米/单元格）
  # The "width," "height," and "resolution" parameters set the width (meters),
height (meters), and resolution (meters/cell) of the costmap.
  width: 2.5
  height: 2.5
  resolution: 0.05
```

#### 8.6.3、global_costmap

Global costmap parameter configuration global_costmap_params.yaml

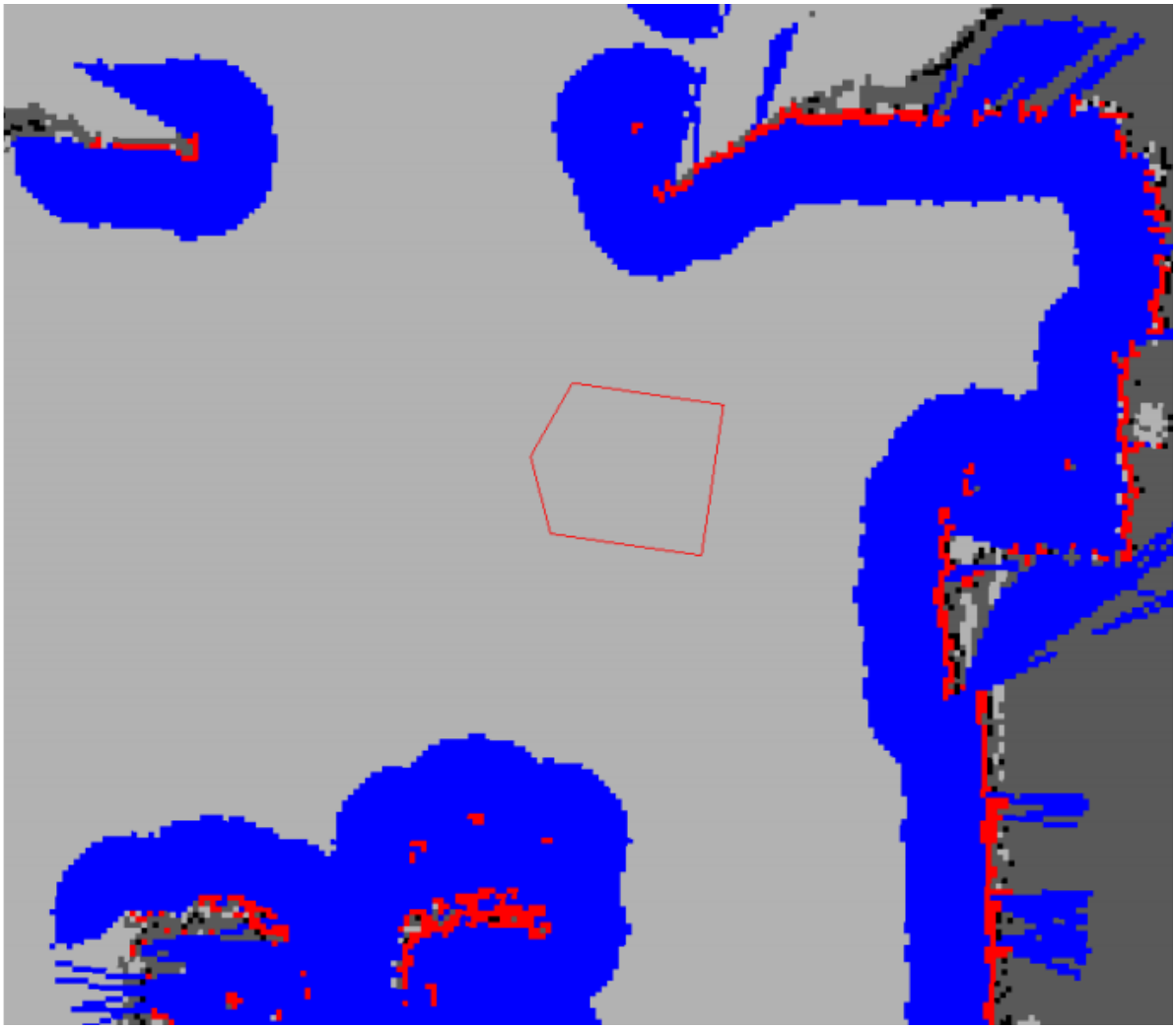
```
global_costmap:
  # Coordinate frame and tf parameters
```

```
  # The "global_frame" parameter defines what coordinate frame the costmap
should run in, in this case, we'll choose the /map frame.
  global_frame: map
  # 机器人的基坐标系
  # The "robot_base_frame" parameter defines the coordinate frame the costmap
should reference for the base of the robot.
  robot_base_frame: base_footprint
  # 指定可容忍的转换（tf）数据延迟（以秒为单位）。
  # Specifies the delay in transform (tf) data that is tolerable in seconds.
  transform_tolerance: 0.5
  # 代价地图更新的频率（以Hz为单位）,数值越大CPU负担越重，通常设定在1.0到5.0之间。
  # The "update_frequency" parameter determines the frequency, in Hz, at which
the costmap will run its update loop.
  update_frequency: 10.0
  # 代价地图发布可视化信息的速率（以Hz为单位）。
  # The rate at which the cost map publishes visual information
  publish_frequency: 10.0
  # 如果为true，则由map_server提供的地图服务来进行代价地图的初始化，否则为false。
  # If true, the map service provided by the map server will initialize the cost
map, otherwise it will be false.
  static_map: true
  # 代价地图分辨率（米/单元格）
  # Cost map resolution (m/cell)
  resolution: 0.05
  # 比例因子
  cost_scaling_factor: 10.0
  # 膨胀半径
  inflation_radius: 0.02
```

#### 8.6.4、costmap_2D

**1 Introduction**

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



- Red represents obstacles in the costmap.
- Blue represents obstacles with expanded radius inscribed in the robot,
- The red polygons represent the robot's footprint.
- In order for the robot to avoid collisions, the robot's shell must never intersect the red cells, and the robot's center point must never intersect the blue cells.

**2)topic**

| name | type | illustrate |
| --- | --- | --- |
| footprint | geometry_msgs/Polygon | Robot enclosure specification. This replaces the previous parameter specification for the package outline. |
| costmap | nav_msgs/OccupancyGrid | cost map |
| costmap_updates | map_msgs/OccupancyGridUpdate | Update area of the costmap |
| voxel_grid | costmap_2d/VoxelGrid | voxel grid |

**3) Parameter configuration**

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

plugin

- plugins: generally use the default.

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 transform(tf) data delay(unit: s).

Rate parameter

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

Map management parameters

- rolling_window: Whether to use the rolling window version of the costmap. If the static_map parameter is set to true, this parameter must be set to false.
- always_send_full_costmap: If true, the full costmap will be published to "/costmap" on every update. If false, only the 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: The 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_fram——>robot_base_frame

4) Layer Specifications

- Static layer static map layer : The static layer is basically unchanged in the cost map.

  Subscribe to topics

  - map: The costmap will make a service call to map_server to get 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: Consider the lethal threshold when reading maps from the map server.
  - map_topic: Specifies the topic that the costmap 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, also subscribe to map_topic + "_updates".
  - track_unknown_space: If true, unknown values in map messages will be converted directly to the layer. Otherwise, unknown values in the map message are 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 will be written to the main costmap.

    - 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 may appear.
- Obstacle layer : The obstacle layer tracks the obstacles read by the sensor data. The collision costmap plugin labels and raytraces obstacles in 2D, while the VoxelCostmapPlugin labels and raytraces obstacles in 3D.
- Inflation layer : Add new values around lethal obstacles(i.e. inflate obstacles) so that the costmap represents the robot's configuration space.
- Other layers: Other layers can be implemented and used in the costmap through pluginlib .
    - Social Costmap Layer
    - Range Sensor Layer

5）、obstacle layer

Obstacle layers and voxel layers contain information from sensors in the form of point clouds or laser scans. Barrier layers track in 2D, while voxel layers track in 3D.

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

Subscribe to topics

| topic name | type | Parse |
| --- | --- | --- |
| point_cloud_topic | sensor_msgs/PointCloud | Update PointCloud information to costmap. |
| point_cloud2_topic | sensor_msgs/PointCloud2 | Update PointCloud2 information to costmap |
| laser_scan_topic | sensor_msgs/LaserScan | Update LaserScan information to costmap |
| map | nav_msgs/OccupancyGrid | The costmap has the option to initialize from a user-generated static map |

传感器管理参数：

- observation_sources: list of observation source names

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

    - <source_name>/topic: The topic covered by the sensor data.
    - <source_name>/sensor_frame: Sensor. Can be sensor_msgs/LaserScan, sensor_msgs/PointCloud, and sensor_msgs/PointCloud2.

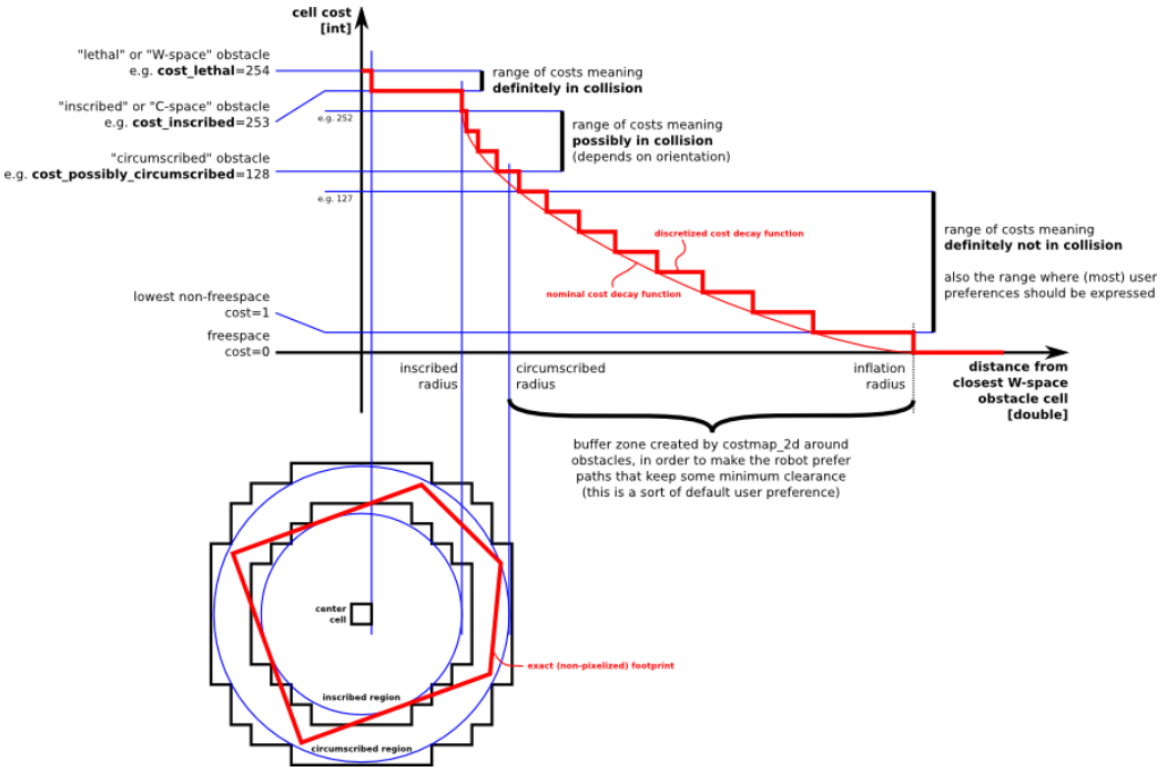
- <source_name>/observation_persistence: The time(unit: s) to hold each sensor reading. A value of 0.0 will keep only the most recent reading.
- <source_name>/expected_update_rate: Frequency of sensor readings(unit: s). A value of 0.0 will allow infinite time between readings.
- <source_name>/data_type: 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: The maximum height(unit: m) of sensor readings that are considered valid. This is usually set slightly above the height of the robot.
- <source_name>/min_obstacle_height: Minimum height(unit: m) for sensor readings to be considered valid. This is usually 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(unit: m) to insert obstacles into the costmap using sensor data.
- <source_name>/raytrace_range: Maximum range(unit: m) to raytrace obstacles from the map using sensor data.
- <source_name>/inf_is_valid: Allows the entry of Inf values in "Laser Scan" observations. Conversion of Inf values to laser maximum range

observation_sources: list of observation source names

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

- <source_name>/topic: The topic covered by the sensor data.
- <source_name>/sensor_frame: Sensor. Can be sensor_msgs/LaserScan, sensor_msgs/PointCloud, and sensor_msgs/PointCloud2.
- <source_name>/observation_persistence: The time(unit: s) to hold each sensor reading. A value of 0.0 will keep only the most recent reading.
- <source_name>/expected_update_rate: Frequency of sensor readings(unit: s). A value of 0.0 will allow infinite 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: The maximum height(unit: m) of sensor readings that are considered valid. This is usually set slightly above the height of the robot.
- <source_name>/min_obstacle_height: Minimum height(unit: m) for sensor readings to be considered valid. This is usually 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(unit: m) to insert obstacles into the costmap using sensor data.
- <source_name>/raytrace_range: Maximum range(unit: m) to raytrace obstacles from the map using sensor data.
- <source_name>/inf_is_valid: Allows the entry of Inf values in "Laser Scan" observations. Conversion of Inf values to laser maximum range

6) Inflation Layer




- The inflation cost decreases as the robot's distance from the obstacle increases. Define 5 specific symbols related to bots for the cost value of the costmap.
    - Fatal("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("Inscribed" cost): Indicates that the distance between the cell and the obstacle is less than the radius of the inscribed circle of the robot. If the center of the robot is at or above the "Inscribed" cost cell, the robot is bound to collide with the obstacle.
    - Possibly circumscribed("Possibly circumscribed" cost): Indicates that the distance of a cell from the obstacle is less than the radius of the circumscribed circle of the robot, but greater than the radius of the circumscribed circle. If the center of the robot is in a cell equal to or higher than the "Possibly circumscribed" cost, the machine will not necessarily collide with the obstacle, depending on the orientation of the robot.
    - Freespace("Freespace"): Nothing prevents the robot from going there.
    - Unknown("Unknown"): Unknown space.

  parameter

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

## 8.7、planner_params

### 8.7.1、global_planner

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

Global Path Planning Plugin

- [navfn ]: A grid map-based global planner that calculates the robot's path when using the navigation function. Dijkstra and A* global planning algorithms are implemented.(Plugin

name: "navfn/NavfnROS")

- global_planner : Reimplemented 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

```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 parsing
    - allow_unknown: Whether to choose to explore unknown areas. It is not enough to just design this parameter to be true, but also set it in costmap_commons_params.yaml `track_unknown_space` must also be set to `true`.
    - default_tolerance: When the set destination is occupied by obstacles, 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 area 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, which saves hardware computing resources.
    - use_grid_path: If true, creates a path along the grid boundaries. Otherwise, using gradient descent, the path is smoother.
    - old_navfn_behavior: If you want global_planner to be the same 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 lethal area of the obstacle(dynamically configurable).
    - neutral_cost: The neutral cost of the obstacle(dynamically configurable).
    - cost_factor: The factor by which the costmap is multiplied by each cost value(dynamically configurable).
    - publish_potential: Whether to publish a possible costmap(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 orientation of the used window is obtained according to the position integral specified by the orientation method; the default value is 1(can be dynamically configured).

- outline_map: Outline the global costmap with deadly obstacles. For non-static(rolling window) global costmap usage, it needs to be set to false

Global path planning algorithm renderings

- All parameters are default

#### 8.7.2、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 NavaCys::BaseLoCalPrnor's C++ API can be found here: BaseLocalPlanner documentation .

Local path planning plugin

- base_local_planner : Implements two local planning algorithms, Trajectory Rollout and DWA.
- dwa_local_planner : Compared to the DWA of base_local_planner, the modular DWA implementation has the advantage of a cleaner, 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 of TEB and DWA:

teb will adjust its orientation during the movement. When it reaches the target point, usually the orientation of the robot is also the orientation of the target and does not need to rotate.

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

For a two-wheel differential chassis, adjusting the orientation of the teb during movement will make the movement path unsmooth, and unnecessary backwards will occur when starting and reaching the target point, which is not allowed in some application scenarios. Because backing up may encounter obstacles. Rotating in place to a suitable orientation and then walking away is a more appropriate exercise strategy. This is also where teb needs to be optimized according to the scene.

**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 package provides a controller that drives the robot to move in a plane. 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 position, sending the dx, dy, dtheta velocities to the robot.

The basic idea of DWA algorithm

- Discrete sampling in robot control space(dx, dy, dtheta)
- For each sampling velocity, perform a forward simulation from the current state of the robot to predict what would happen if the sampling velocity were applied for a(short) period of time.
- Evaluate(score) each trajectory produced by the simulation ahead, using a metric that includes the following characteristics: approaching obstacle, approaching target,

approaching global path, and speed. Illegal trajectories(trajectories that collide with obstacles) are discarded.
- The trajectory with the highest score is selected and the associated velocity is sent to the mobile robot.
- Clean the data and repeat.

A number of ROS parameters can be set to customize the behavior of dwa_local_planner::DWAPlannerROS. These parameters fall 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 tune the local path planner in a running system.

```
DWAPlannerROS:
  # Robot Configuration Parameters
  # x方向最大线速度绝对值，单位:米/秒
  # The maximum y velocity for the robot in m/s
  max_vel_x: 0.6
  # x方向最小线速度绝对值，负数代表可后退，单位:米/秒
  # The minimum x velocity for the robot in m/s, negative for backwards motion.
  min_vel_x: -0.6
  # y方向最大线速度绝对值，单位:米/秒。差速机器人为0 0.6
  # The maximum y velocity for the robot in m/s
  max_vel_y: 0.3
  # y方向最小线速度绝对值，单位:米/秒。差速驱动机器人为0 0.6
  # The minimum y velocity for the robot in m/s
  min_vel_y: -0.3
  # 机器人最大旋转速度的绝对值，单位为 rad/s 2.0
  # The absolute value of the maximum rotational velocity for the robot in rad/s
  max_rot_vel: 5.0
  # 机器人最小旋转速度的绝对值，单位为 rad/s 2.0
  # The absolute value of the minimum rotational velocity for the robot in rad/s
  min_rot_vel: 5.0
  # 机器人最大平移速度的绝对值，单位为 m/s
  # The absolute value of the maximum translational velocity for the robot in
m/s
  max_vel_trans: 1.0
  # 机器人最小平移速度的绝对值，单位为 m/s 不可为零
  # The absolute value of the minimum translational velocity for the robot in
m/s
  min_vel_trans: 0.01
  # 机器人被认属于“停止”状态时的平移速度。单位为 m/s
  # The translation speed when the robot is considered to be in the "stop" state
in m/s.
  trans_stopped_vel: 0.1
  # 机器人的最大旋转角速度的绝对值，单位为 rad/s
  # The maximum rotational velocity limit of the robot in radians/sec^2
  max_vel_theta: 2.0
  # 器人的最小旋转角速度的绝对值，单位为 rad/s
  # The minimum rotational velocity limit of the robot in radians/sec^2
  min_vel_theta: 0.1
  # 机器人被认属于“停止”状态时的旋转速度。单位为 rad/s
  # The rotation speed of the robot when it is considered to be in the "stopped"
state in m/s.
  theta_stopped_vel: 0.4
  # 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: 10.0
  # 机器人在y方向的极限加速度，差速机器人来说是0 10
  # The y acceleration limit of the robot in meters/sec^2
  acc_lim_y: 10.0
  # 机器人的极限旋转加速度，单位为 rad/sec^2
  # The rotational acceleration limit of the robot in radians/sec^2
  acc_lim_theta: 20.0

  # Goal Tolerance Parameters目标距离公差参数
  # 到达目标点时偏行角/旋转时的弧度允许的误差，单位弧度
  # The tolerance in radians for the controller in yaw/rotation when achieving
its goal
  yaw_goal_tolerance: 0.1
  # 到达目标点时,在xy平面内与目标点的距离误差，单位:m.
  # The tolerance in meters for the controller in the x & y distance when
achieving a goal
  xy_goal_tolerance: 0.1
  # 设置为true时表示：如果到达容错距离内,机器人就会原地旋转；即使转动是会跑出容错距离外。
  # If goal tolerance is latched, if the robot ever reaches the goal xy location
it will simply rotate in place, even if it ends up outside the goal tolerance
while it is doing so.
  latch_xy_goal_tolerance: false
  # Forward Simulation Parameters前向模拟参数
  # 前向模拟轨迹的时间，单位为s(seconds)
  # The amount of time to forward-simulate trajectories in seconds
  sim_time: 1.0
  # x方向速度空间的采样点数
  # The number of samples to use when exploring the x velocity space
  vx_samples: 10
  # y方向速度空间采样点数.。差分驱动机器人y方向永远只有1个值（0.0）
  # The number of samples to use when exploring the y velocity space
  vy_samples: 0
  # 旋转方向的速度空间采样点数
  # The number of samples to use when exploring the theta velocity space
  vtheta_samples: 10
  # 以 Hz 为单位调用此控制器的频率。
  # The frequency at which this controller will be called in Hz.
  controller_frequency: 10.0

  # Trajectory Scoring Parameters
  # 控制器与给定路径接近程度的权重
  # The weighting for how much the controller should stay close to the path it
was given
  path_distance_bias: 10.0
  # 控制器与局部目标点的接近程度的权重，也用于速度控制
  # The weighting for how much the controller should attempt to reach its local
goal, also controls speed
  goal_distance_bias: 20.0
  # 控制器躲避障碍物的程度
  # The weighting for how much the controller should attempt to avoid obstacles
  occdist_scale: 0.5
  # 以机器人为中心，额外放置一个计分点的距离
  # The distance from the center point of the robot to place an additional
scoring point, in meters
```

```
  forward_point_distance: 0.325
  # 机器人在碰撞发生前必须拥有的最少时间量。该时间内所采用的轨迹仍视为有效。
  # The amount of time that the robot must stop before a collision in order for
a trajectory to be considered valid in seconds
  stop_time_buffer: 0.1
  # 开始缩放机器人足迹时的速度的绝对值，单位为m/s。
  # The absolute value of the velocity at which to start scaling the robot's
footprint,
  scaling_speed: 0.25
  # 最大缩放因子。max_scaling_factor为上式的值的大小。
  # The maximum factor to scale the robot's footprint by
  max_scaling_factor: 0.2

  # Oscillation Prevention Parameters
  # 机器人必须运动多少米远后才能复位震荡标记(机器人运动多远距离才会重置振荡标记)
  # How far the robot must travel in meters before oscillation flags are reset
  oscillation_reset_dist: 0.05
  oscillation_reset_angle: 0.05

  # Debugging调试参数
  # 将规划的轨迹在RVIZ上进行可视化
  publish_traj_pc: true
  # 将代价值进行可视化显示
  publish_cost_grid_pc: true
  # 全局参考坐标系
  global_frame_id: /map
```