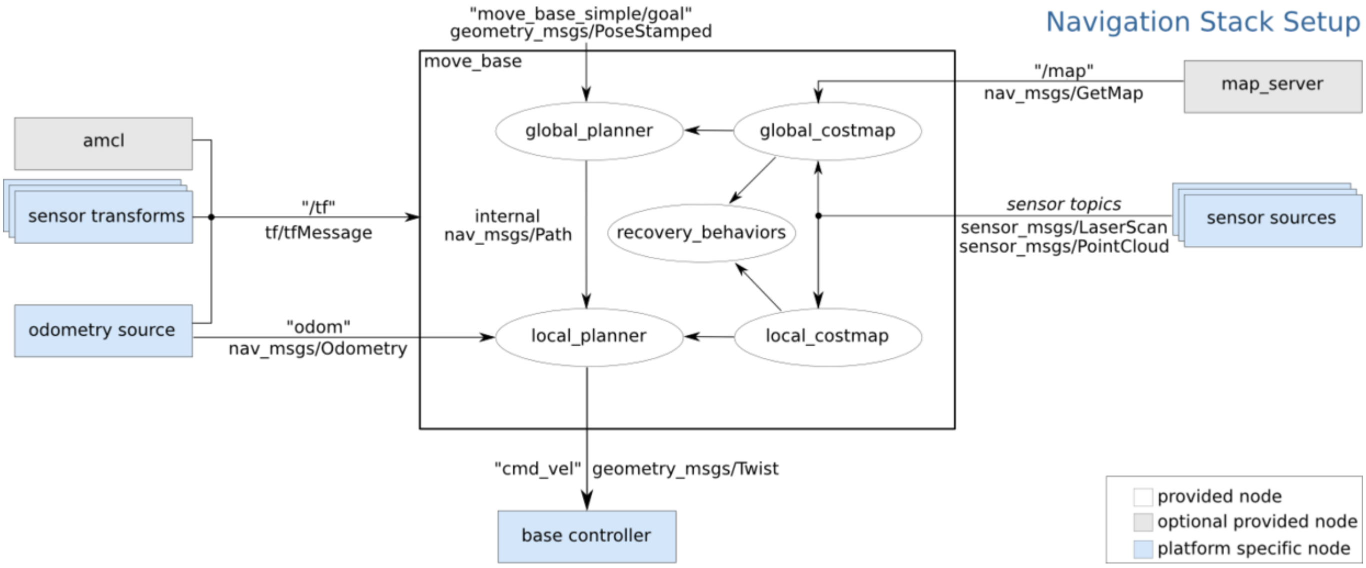
1.配置导航参数

前文(13.move_base介绍(1))讲了`move_base`的简单的基础,本文将详细分析下如何配置move_base参数

最简单的配置

再次引用该图



`map_server`和`amcl`都是不必须的，我们就首先配置一个没有map的`move_base`

`fake_move_base_with_out_map.launch`

```
<launch>
  <include file="$(find pibot_bringup)/launch/robot.launch"/>
  <param name="use_sim_time" value="false" />
  <include file="$(find
pibot_nav)/launch/include/move_base_with_out_map.launch.xml" />
</launch>
```

- **`robot.launch`为`Pibot`的驱动，其他底盘可替换为自己的驱动**
- `move_base_with_out_map.launch.xml`

```
<launch>
  <arg name="model" default="$(env PIBOT_MODEL)" doc="model type [apollo, zeus]"/>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen" clear_params="true">
    <rosparam file="$(find pibot_nav)/params/costmap_common_params_$(arg
model).yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find pibot_nav)/params/costmap_common_params_$(arg
model).yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find pibot_nav)/params/local_costmap_params_withoutmap.yaml"
command="load" />
    <rosparam file="$(find
pibot_nav)/params/global_costmap_params_withoutmap.yaml" command="load" />
    <rosparam file="$(find pibot_nav)/params/dwa_local_planner_params_$(arg
model).yaml" command="load" />
    <rosparam file="$(find pibot_nav)/params/move_base_params.yaml" command="load"
/>
    <rosparam file="$(find pibot_nav)/params/global_planner_params.yaml"
command="load" />
  </node>
</launch>
```

- 配置文件详情

1.costmap_common_params_apollo.yaml

```
max_obstacle_height: 0.60  # assume something like an arm is mounted on top of the
robot

# Obstacle Cost Shaping (http://wiki.ros.org/costmap_2d/hydro/inflation)
#robot_radius: 0.16  # distance a circular robot should be clear of the obstacle
(kobuki: 0.18)
footprint: [[0.10, -0.07], [0.10, 0.18], [-0.10, 0.18], [-0.10, -0.07]]
# footprint: [[x0, y0], [x1, y1], ... [xn, yn]]  # if the robot is not circular

map_type: voxel

obstacle_layer:
  enabled:              true
  max_obstacle_height:  0.6
  origin_z:             0.0
  z_resolution:         0.2
  z_voxels:             2
  unknown_threshold:    15
  mark_threshold:       0
  combination_method:   1
  track_unknown_space:  true    #true needed for disabling global path planning
through unknown space
  obstacle_range: 2.5
```

```
  raytrace_range: 3.0
  origin_z: 0.0
  z_resolution: 0.2
  z_voxels: 2
  publish_voxel_map: false
  observation_sources:  scan
  scan:
    data_type: LaserScan
    topic: scan
    inf_is_valid: true
    marking: true
    clearing: true
    min_obstacle_height: 0.05
    max_obstacle_height: 0.35
  #bump:
    #data_type: PointCloud2
    #topic: mobile_base/sensors/bumper_pointcloud
    #marking: true
    #clearing: false
    #min_obstacle_height: 0.0
    #max_obstacle_height: 0.15
  # for debugging only, let's you see the entire voxel grid

#cost_scaling_factor and inflation_radius were now moved to the inflation_layer ns
inflation_layer:
  cost_scaling_factor:  2.5  # exponential rate at which the obstacle cost drops off (default: 10)
  inflation_radius:     1.2  # max. distance from an obstacle at which costs are incurred for planning paths.

static_layer:
  enabled:              false
```

2.`local_costmap_params_withoutmap.yaml`

```
local_costmap:
   global_frame: /odom
   robot_base_frame: /base_link
   update_frequency: 1.0
   publish_frequency: 2.0
   static_map: false
   rolling_window: true
   width: 4
   height: 4
   resolution: 0.05
   transform_tolerance: 0.5
   plugins:
    - {name: obstacle_layer,      type: "costmap_2d::VoxelLayer"}
    - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}
```

3.`global_costmap_params_withoutmap.yaml`

```
global_costmap:
   global_frame: /map
   robot_base_frame: /base_link
   update_frequency: 1.0
   publish_frequency: 0.5
   static_map: false
   rolling_window: true
   width: 12
   height: 12
   resolution: 0.05
   transform_tolerance: 0.5
   plugins:
     - {name: obstacle_layer,          type: "costmap_2d::VoxelLayer"}
     - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}
```

4.`dwa_local_planner_params_apollo.yaml`

```
DWAPlannerROS:

# Robot Configuration Parameters - Kobuki
  max_vel_x: 0.25
  min_vel_x: 0.05

  max_vel_y: 0
  min_vel_y: 0

  max_trans_vel: 0.35 # choose slightly less than the base's capability
  min_trans_vel: 0.001  # this is the min trans velocity when there is negligible
rotational velocity
  trans_stopped_vel: 0.05

  # Warning!
  #   do not set min_trans_vel to 0.0 otherwise dwa will always think
translational velocities
  #   are non-negligible and small in place rotational velocities will be created.

  max_rot_vel: 0.6  # choose slightly less than the base's capability
  min_rot_vel: 0.4  # this is the min angular velocity when there is negligible
translational velocity
  rot_stopped_vel: 0.1

  acc_lim_x: 1 # maximum is theoretically 2.0, but we
  acc_lim_theta: 1.5
  acc_lim_y: 0      # diff drive robot

# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.2
  xy_goal_tolerance: 0.15
```

```
  latch_xy_goal_tolerance: false

# Forward Simulation Parameters
  sim_time: 2.0       # 1.7
  vx_samples: 10       # 3
  vy_samples: 1       # diff drive robot, there is only one sample
  vtheta_samples: 20  # 20

# Trajectory Scoring Parameters
  path_distance_bias: 32.0      # 32.0   - weighting for how much it should stick
to the global path plan
  goal_distance_bias: 24.0      # 24.0   - wighting for how much it should attempt
to reach its goal
  occdist_scale: 0.4            # 0.01   - weighting for how much the controller
should avoid obstacles
  forward_point_distance: 0.325 # 0.325  - how far along to place an additional
scoring point
  stop_time_buffer: 0.2         # 0.2    - amount of time a robot must stop in
before colliding for a valid traj.
  scaling_speed: 0.25           # 0.25   - absolute velocity at which to start
scaling the robot's footprint
  max_scaling_factor: 0.2       # 0.2    - how much to scale the robot's footprint
when at speed.

# Oscillation Prevention Parameters
  oscillation_reset_dist: 0.05  # 0.05   - how far to travel before resetting
oscillation flags

# Debugging
  publish_traj_pc : true
  publish_cost_grid_pc: true
  global_frame_id: odom


# Differential-drive robot configuration - necessary?
#   holonomic_robot: false
```

5.`move_base.yaml`

```
# Move base node parameters. For full documentation of the parameters in this
file, please see
#
#   http://www.ros.org/wiki/move_base
#
shutdown_costmaps: false

controller_frequency: 5.0
controller_patience: 3.0
```

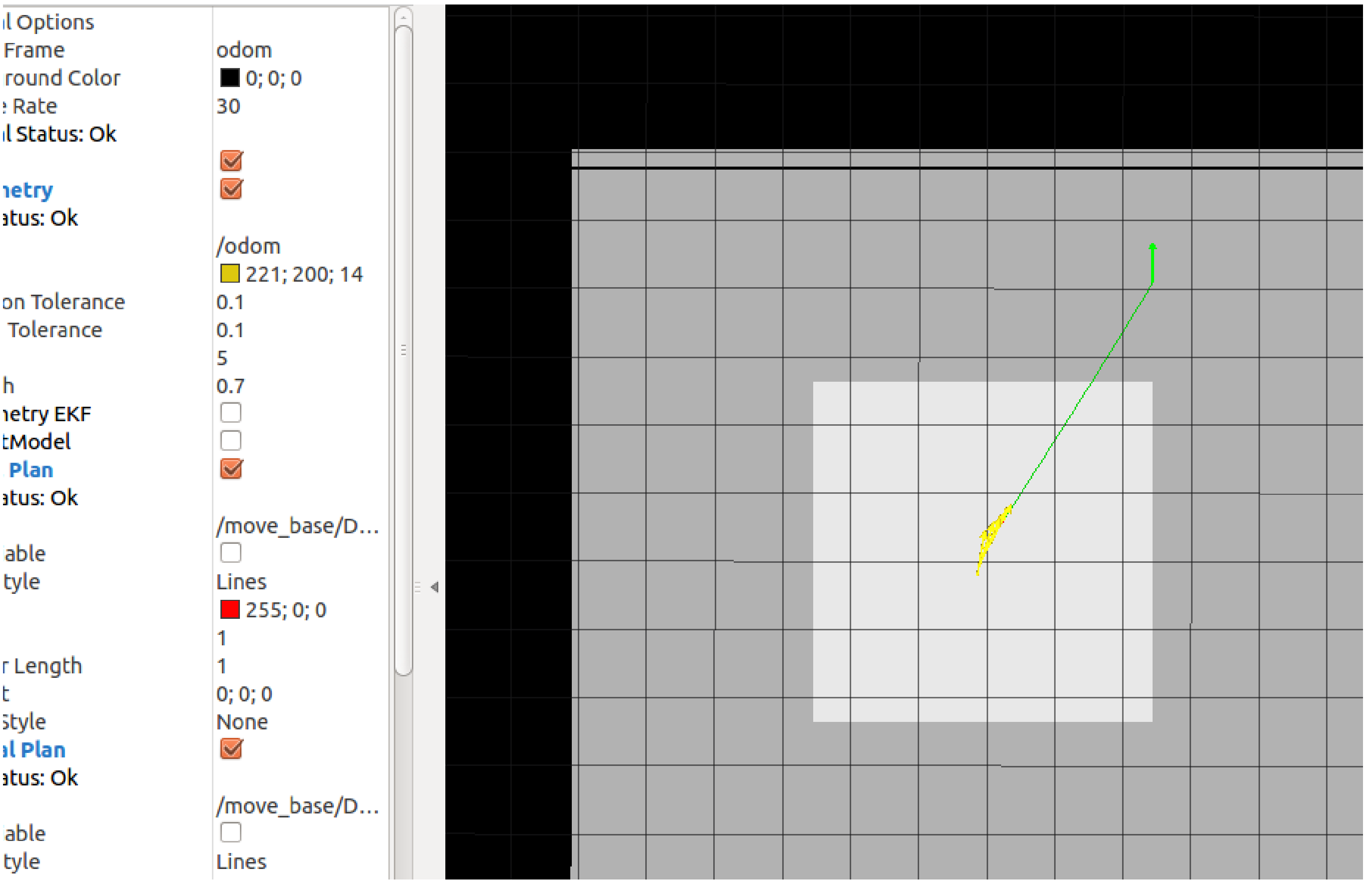
```
planner_frequency: 1.0
planner_patience: 5.0

oscillation_timeout: 10.0
oscillation_distance: 0.2

# local planner - default is trajectory rollout
base_local_planner: "dwa_local_planner/DWAPlannerROS"

base_global_planner: global_planner/GlobalPlanner #"navfn/NavfnROS" #alternatives:
, carrot_planner/CarrotPlanner
```

6.`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: true                           # 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: true                           # Allow planner to plan through
unknown space, default true
                                                #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: 0.5                        # 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                       # Publish Potential Costmap (this
is not like the navfn pointcloud2 potential), default true
```

运行结果

运行`roslaunch pibot_navigation fake_move_base_with_out_map.launch` `roslaunch pibot_navigation view_nav_with_out_map.launch` 选择`2D Nav Goal`导航可以看到



配置分析

可以看到`move_base`配置项较多，涉及到`cost_map`及`planner`,分别又包括`local_cost_map`、`global_cost_map`和`local_planner`、`global_planner` 首先看根配置(简单说就是没有前面的namespace的)，除了`move_base.yaml`,其他文件都是二级配置项

> **common_cost_map中在move_base_with_out_map.launch.xml都已经指定了namespace**
> ```
> <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen" clear_params="true">
>   <rosparam file="$(find pibot_nav)/params/costmap_common_params_$(arg model).yaml" command="load" ns="global_costmap" />
>   <rosparam file="$(find pibot_nav)/params/costmap_common_params_$(arg model).yaml" command="load" ns="local_costmap" />
>   <rosparam file="$(find pibot_nav)/params/local_costmap_params_withoutmap.yaml" command="load" />
>   <rosparam file="$(find pibot_nav)/params/global_costmap_params_withoutmap.yaml" command="load" />
>   <rosparam file="$(find pibot_nav)/params/dwa_local_planner_params_$(arg model).yaml" command="load" />
>   <rosparam file="$(find pibot_nav)/params/move_base_params.yaml" command="load" />
>   <rosparam file="$(find pibot_nav)/params/global_planner_params.yaml" command="load" />
> </node>
> ```

####`move_base` 根配置

- `shutdown_costmaps` 当move_base在不活动状态时，是不是要关掉move_base node的 costmap

> 查看源码可知move_base空闲时`shutdown_costmaps`为`true`会关掉`cost_map`,激活是会重新开启
>
> ```
> void MoveBase::resetState(){
>   // Disable the planner thread
>   boost::unique_lock<boost::recursive_mutex> lock(planner_mutex_);
>   runPlanner_ = false;
>   lock.unlock();
>
>   // Reset statemachine
>   state_ = PLANNING;
>   recovery_index_ = 0;
>   recovery_trigger_ = PLANNING_R;
>   publishZeroVelocity();
>
>   //if we shutdown our costmaps when we're deactivated... we'll do that now
>   if(shutdown_costmaps_){
>     ROS_DEBUG_NAMED("move_base","Stopping costmaps");
>     planner_costmap_ros_->stop();
>     controller_costmap_ros_->stop();
>   }
> }
> ```
>
> ```
> void MoveBase::executeCb(const move_base_msgs::MoveBaseGoalConstPtr& move_base_goal)
> {
>   if(! isQuaternionValid(move_base_goal->target_pose.pose.orientation)){
>     as_->setAborted(move_base_msgs::MoveBaseResult(), "Aborting on goal because it was sent with a
>     return;
>   }
>
>   geometry_msgs::PoseStamped goal = goalToGlobalFrame(move_base_goal->target_pose);
>
>   //we have a goal so start the planner
>   boost::unique_lock<boost::recursive_mutex> lock(planner_mutex_);
>   planner_goal_ = goal;
>   runPlanner_ = true;
>   planner_cond_.notify_one();
>   lock.unlock();
>
>   current_goal_pub_.publish(goal);
>   std::vector<geometry_msgs::PoseStamped> global_plan;
>
>   ros::Rate r(controller_frequency_);
>   if(shutdown_costmaps_){
>     ROS_DEBUG_NAMED("move_base","Starting up costmaps that were shut down previously");
>     planner_costmap_ros_->start();
>     controller_costmap_ros_->start();
>   }
> ```
>
> 默认`false`

- `controller_frequency` 规划频率，太大会占用CPU 这里我们设置为3， 好点的处理器可以设置稍高
- `controller_patience`

> ~`controller_patience` (double, default: 15.0)
>
> How long the controller will wait in seconds without receiving a valid control before space-clearing operations are performed.
>
> 算了还是直接看源码吧

```
if(tc_->computeVelocityCommands(cmd_vel)){
  ROS_DEBUG_NAMED( "move_base", "Got a valid command from the local planner: %.3lf, %.3lf, %.3lf",
                cmd_vel.linear.x, cmd_vel.linear.y, cmd_vel.angular.z );
  last_valid_control_ = ros::Time::now();
  //make sure that we send the velocity command to the base
  vel_pub_.publish(cmd_vel);
  if(recovery_trigger_ == CONTROLLING_R)
    recovery_index_ = 0;
}
else {
  ROS_DEBUG_NAMED("move_base", "The local planner could not find a valid plan.");
  ros::Time attempt_end = last_valid_control_ + ros::Duration(controller_patience_);

  //check if we've tried to find a valid control for longer than our time limit
  if(ros::Time::now() > attempt_end){
    //we'll move into our obstacle clearing mode
    publishZeroVelocity();
    state_ = CLEARING;
    recovery_trigger_ = CONTROLLING_R;
  }
```

> 计算速度失败就判断有没有超时，超时就切换状态

- `planner_frequency`

> ~planner_frequency (double, default: 0.0)
>
> The rate in Hz at which to run the global planning loop. If the frequency is set to 0.0, the global planner will only run when a new goal is received or the local planner reports that its path is blocked. **New in navigation 1.6.0**
>
> 容易理解这个是全局路径规划的频率；如果为0即只规划一次

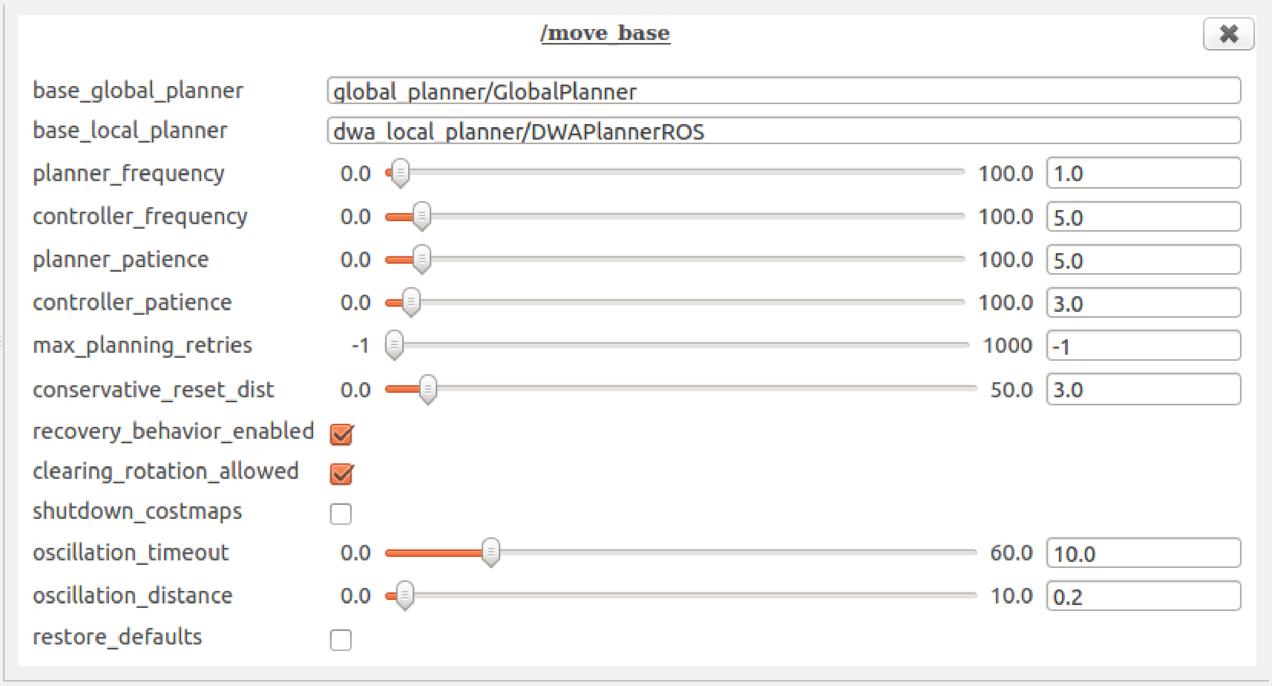
- `planner_patience` 容易理解，规划路径的最大容忍时间
- `oscillation_timeout`&`oscillation_distance`

> 陷在方圆`oscillation_distance`达`oscillation_timeout`之久，认定机器人在震荡，从而做异常处理（应该容易理解吧）

```
if(oscillation_timeout_ > 0.0 &&
    last_oscillation_reset_ + ros::Duration(oscillation_timeout_) < ros::Time::now())
{
  publishZeroVelocity();
  state_ = CLEARING;
  recovery_trigger_ = OSCILLATION_R;
}
```

- `base_local_planner` & `base_global_planner`

> 最为重要的2个参数，直接指定使用哪种局部规划和全局规划， 具体类分别继承与实现nav_core::BaseLocalPlanner和`nav_core::BaseGlobalPlanner`接口

> `rosrun rqt_reconfigure rqt_reconfigure` 查看move_base的参数
>
> 
>
>
> 可以看到还有几个参数，一并看下

- `max_planning_retries` 最大规划路径的重试次数 `-1`标识无限次
- `recovery_behavior_enabled` 是否启用恢复机制
- `clearing_rotation_allowed` 是否启用旋转的恢复，当然是在`recovery_behavior_enabled` 为`true`的基础上的
- `recovery_behaviors` 一系列的恢复机制，同`base_local_planner` & `base_global_planner` 具体类继承于`nav_core::RecoveryBehavior`
- `conservative_reset_dist` 清除机制的参数， 决定清除多远外的障碍