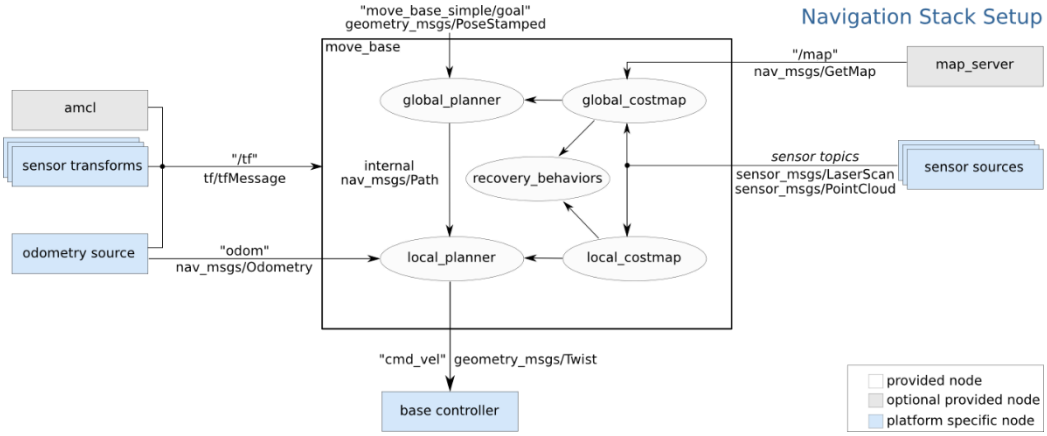
第二十二课 Move_base 导航框架

1. 前言

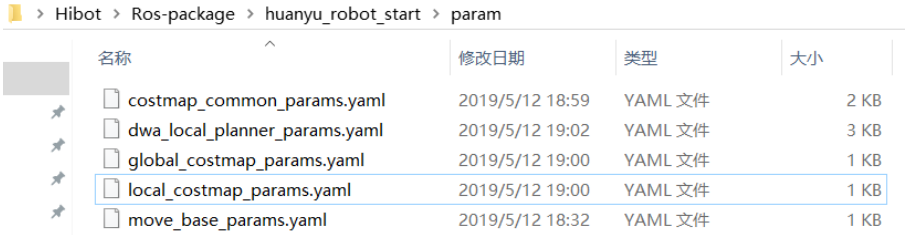
本节课程重点理解 Move_Base 的整个框架，move_base 包提供了一个动作的实现（参见 actionlib 包），在地图上给定一个目标，move_base 将会规划出路径并使机器人避开障碍物从而到达目标。move_base 节点其实相当于 ros navigation 整体框架的一个指挥官，它统筹了各个 navigation 节点发过来的信息，在下面的系统框图中，浅蓝色的项目是必须要输入的，之前的工作已经完成了这些项目的需求。



move_base 节点将全局导航和局部导航链接在一起以完成其导航任务。全局导航用于建立到地图上最终目标或一个远距离目标的路径，局部导航用于建立到近距离目标和为了临时躲避障碍物的路径，例如机器人四周一个 4x4 m 的方形窗口。

2. 配置文件

move_base 节点支持任何遵循 nav_core 包中指定的 nav_core :: BaseGlobalPlanner 接口的全局规划以及任何遵循 nav_core 包中指定的 nav_core :: BaseLocalPlanner 接口的局部规划。move_base 节点还维护两个代价地图，分别为全局代价地图和局部代价地图。在运行 move_base 节点之前需要五个配置文件，这些文建定义了一系列相关参数，包括越过障碍物的代价、机器人的半径、路劲规划时的需要考虑未来多长的路，机器人的的最大速度等等，五个配置文件分别如下：



3. 全局路径规划

在 ROS 的导航中，首先会通过全局路径规划，计算出机器人到目标位置的全局路线。这一功能是 navfn 这个包实现的。navfn 通过 Dijkstra 最优路径的算法，计算 costmap 上的最小花费路径，作为机器人的全局路线。

4. 局部路径规划

本地的实时规划是利用 base_local_planner 包实现的。该包使用 Trajectory

Rollout 和 Dynamic Window approaches 算法计算机器人每个周期内应该行驶的速度和角度（dx, dy, dtheta velocities）。base_local_planner 这个包通过地图数据，通过算法搜索到达目标的多条路经，利用一些评价标准（是否会撞击障碍物，所需要的时间等等）选取最优的路径，并且计算所需要的实时速度和角度。

其中，Trajectory Rollout 和 DWA 算法的主要思路如下：（之前的课程已经有介绍了）

（1） 采样机器人当前的状态（dx,dy,dtheta）;

（2） 针对每个采样的速度，计算机器人以该速度行驶一段时间后的状态，得出一条行驶的路线。

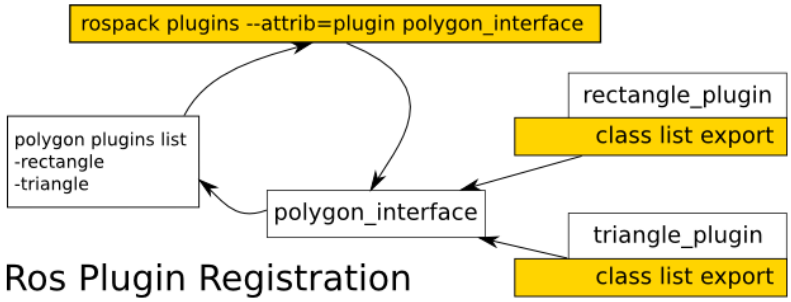
（3） 利用一些评价标准为多条路线打分。

（4） 根据打分，选择最优路径。

（5） 重复上面过程。

5. 插件的工作方式

如果我们想按照自己的思路去写 global 或者 local planner 时，该如何做呢？显然 ROS 有一个标准。你必须按照它提供给你的模板去实现你自己的算法，这些模板就是基类。如果觉得抽象，可以先通过官方的文档来了解插件是如何工作的？



在上图的这个例子中，假设我们想使用一个形状画图模块，ROS 的官方包中已经有了 polygon_interface 这个基类，它已经提供了标准的接口函数，并且有两个子类：矩形插件包 rectangle_plugin package 和三角形插件包 triangle_plugin package。要想使用这两个子类你只要在相应文件中将它们注册为插件，告诉 ROS 我将使用这两个插件就行了。这个图中一个关键的中心点就是 polygon_interface 基类。

6. 参数配置说明

1>. move_base_params.yaml

```
shutdown_costmaps: false

controller_frequency: 3.0
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: "navfn/NavfnROS"
 #alternatives: global_planner/GlobalPlanner, carrot_planner/CarrotPlanner
```

shutdown_costmaps:当 move_base 在不活动状态时,是否关掉 costmap。

controller_frequency:向底盘控制移动话题 cmd_vel 发送命令的频率。

controller_patience:在空间清理操作执行前,控制器花多长时间等有效控制下发。

planner_frequency:全局规划操作的执行频率.如果设置为 0.0,则全局规划器仅在接收

到新的目标点或者局部规划器报告路径堵塞时才会重新执行规划操作。
planner_patience:在空间清理操作执行前,留给规划器多长时间来找出一条有效规划。
oscillation_timeout:执行修复机制前,允许振荡的时长。
oscillation_distance:来回运动在多大距离以上不会被认为是振荡。
base_local_planner:指定用于 move_base 的局部规划器名称。
base_global_planner:指定用于 move_base 的全局规划器插件名称。

2>. dwa_local_planner_params.yaml

```yaml
DWAPlannerROS:

# Robot Configuration Parameters - Kobuki
  max_vel_x: 0.25  # 0.55
  min_vel_x: 0.0

  max_vel_y: 0.0  # diff drive robot
  min_vel_y: 0.0  # diff drive robot

  max_trans_vel: 0.5 # choose slightly less than the base's capability
  min_trans_vel: 0.1  # this is the min trans velocity when there is negligible rotational velocity
  trans_stopped_vel: 0.1

  # 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: 0.7  # choose slightly less than the base's capability
  min_rot_vel: 0.3  # this is the min angular velocity when there is negligible translational velocity
  rot_stopped_vel: 0.4

  acc_lim_x: 0.8 # maximum is theoretically 2.0, but we
  acc_lim_theta: 3.5
  acc_lim_y: 0.0      # diff drive robot
```

```yaml
# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.1  # 0.05
  xy_goal_tolerance: 0.4  # 0.10
  # latch_xy_goal_tolerance: false

# Forward Simulation Parameters
  sim_time: 1.8       # 1.7
  vx_samples: 6       # 3
  vy_samples: 1       # diff drive robot, there is only one sample
  vtheta_samples: 20  # 20

# Trajectory Scoring Parameters
  path_distance_bias: 64.0      # 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: 0.5            # 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_combined
```

acc_lim_x:x 方向的加速度绝对值
acc_lim_y:y 方向的加速度绝对值,该值只有全向移动的机器人才需配置.
acc_lim_th:旋转加速度的绝对值.
max_trans_vel:平移速度最大值绝对值
min_trans_vel:平移速度最小值的绝对值
max_vel_x:x 方向最大速度的绝对值
min_vel_x:x 方向最小值绝对值,如果为负值表示可以后退.
max_vel_y:y 方向最大速度的绝对值.
min_vel_y:y 方向最小速度的绝对值.
max_rot_vel:最大旋转速度的绝对值.
min_rot_vel:最小旋转速度的绝对值.
yaw_goal_tolerance:到达目标点时偏行角允许的误差, 单位弧度.

xy_goal_tolerance:到达目标点时,在 xy 平面内与目标点的距离误差.
sim_time:向前仿真轨迹的时间.
sim_granularity:步长,轨迹上采样点之间的距离,轨迹上点的密集程度.
vx_samples:x 方向速度空间的采样点数.
vy_samples:y 方向速度空间采样点数.
vth_samples:旋转方向的速度空间采样点数.
controller_frequency:发送给底盘控制移动指令的频率.
path_distance_bias:定义控制器与给定路径接近程度的权重.
goal_distance_bias:定义控制器与局部目标点的接近程度的权重.
occdist_scale:定义控制器躲避障碍物的程度.
stop_time_buffer:为防止碰撞,机器人必须提前停止的时间长度.
scaling_speed:启动机器人底盘的速度.
max_scaling_factor:最大缩放参数.

3>. global_costmap_params.yaml

```yaml
global_costmap:
   global_frame: map
   robot_base_frame: base_footprint
   update_frequency: 1.0
   publish_frequency: 0.5
   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:代价地图应该运行的坐标系，将选择/map 坐标系。
robot_base_frame:代价地图应该为机器人的基座的坐标系。应设为 base_footprint。
update_frequency:决定了代价地图更新的频率。数值越大， CPU 负担会越重。
static_map:是否由 map_server 提供的地图服务来进行代价地图的初始化。

4>. local_costmap_params.yaml

```yaml
local_costmap:
   global_frame: odom_combined
   robot_base_frame: base_footprint
   update_frequency: 3.0
   publish_frequency: 1.0
   static_map: false
   rolling_window: true
   width: 2.0
   height: 2.0
   resolution: 0.05
   transform_tolerance: 0.5
   plugins:
    - {name: obstacle_layer,      type: "costmap_2d::ObstacleLayer"}
    - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}
```

rolling_window:设置为 true 意味着当机器人移动时,保持机器人在本地代价地图中心。
width: 滑动地图的宽度（单位米）。
height: 滑动地图的高度（单位米）。
resolution: 滑动地图的分辨率。

7. 总结

本节课程重点介绍了 move_base 的整个框架、全局和局部路径规划的插件配置、导航包

的参数配置。下节课程我们开始使用 HiBot 来测试我们配置的导航功能包。