

ROS理论与实践

—— 第10讲：机器人自主导航



主讲人 胡春旭






机器人博客“古月居”博主

《ROS机器人开发实践》作者

武汉精锋微控科技有限公司 联合创始人

华中科技大学 人工智能与自动化学院 硕士



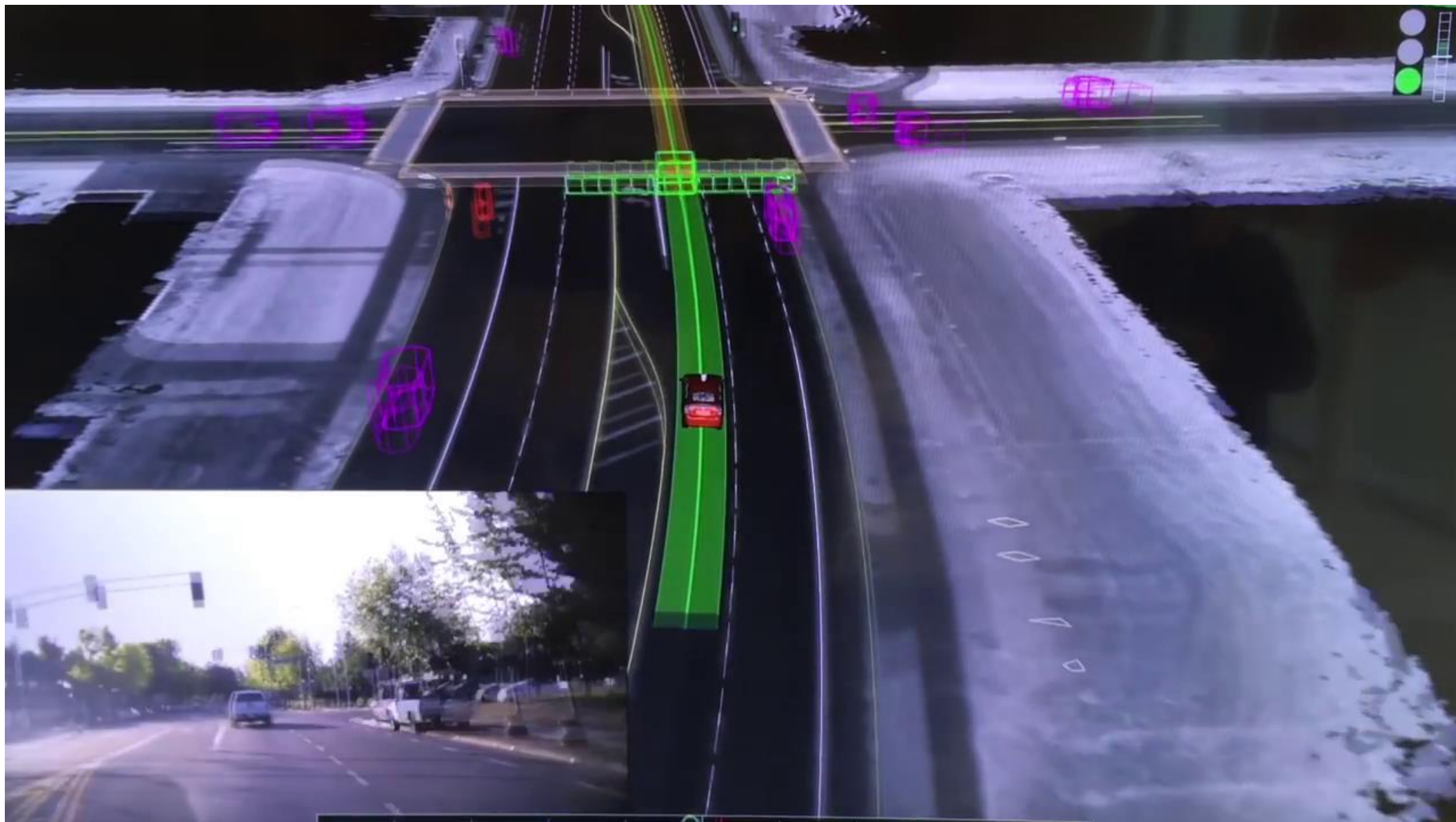
-  1. ROS中的导航框架
-  2. 导航框架中的关键功能包
-  3. 机器人自主导航案例



1. ROS中的导航框架



1. ROS中的导航框架





1. ROS中的导航框架



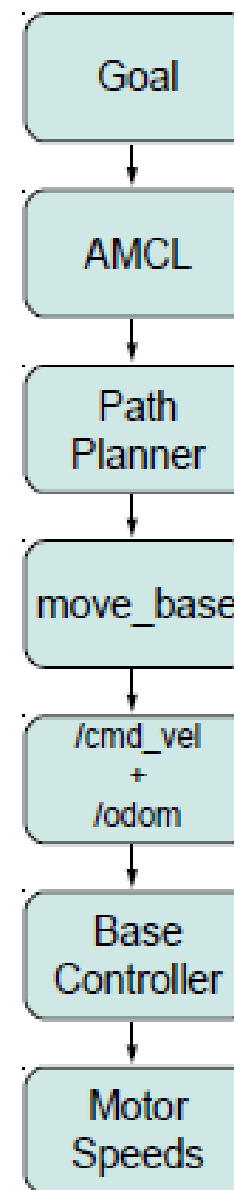
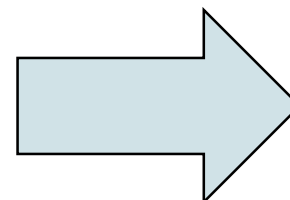


1. ROS中的导航框架



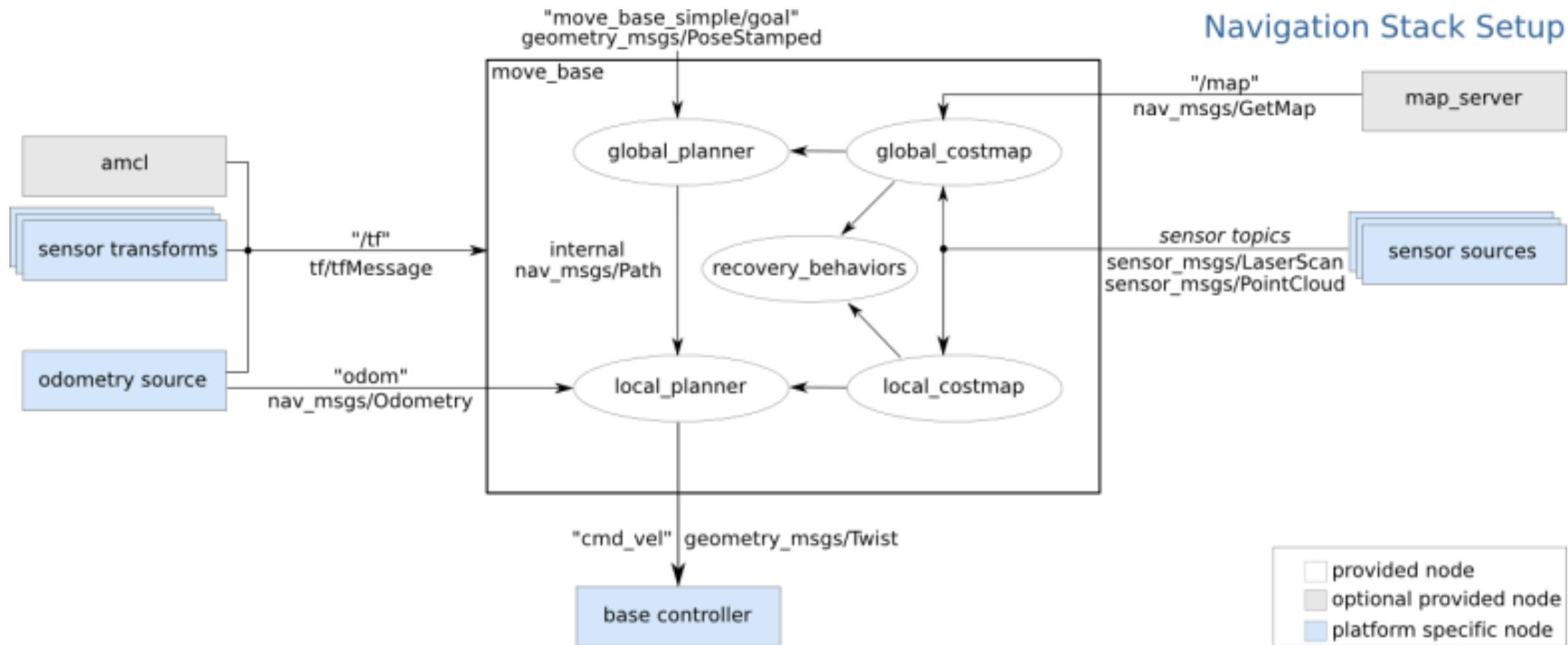


1. ROS中的导航框架





1. ROS中的导航框架



基于move_base的导航框架

(\$ sudo apt-get install ros-melodic-navigation)



1. ROS中的导航框架 —— 硬件约束

(1) 差分轮式机器人，可使用Twist速度指令控制

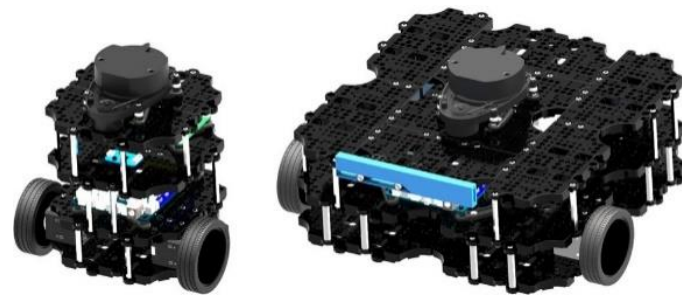
- linear: XYZ 方向上的线速度，单位是 m/s;
- angular: XYZ 方向上的角速度，单位是 rad/s。

```
→ ~ rosmmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
float64 x
float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
float64 z
```

(2) 机器人必须安装激光雷达等测距设备，
可以获取环境深度信息。



(3) 最好使用正方形和圆形的机器人，其他外形的
机器人虽然可以正常使用，但是效果很可能不佳。





2. 导航框架中的关键功能包

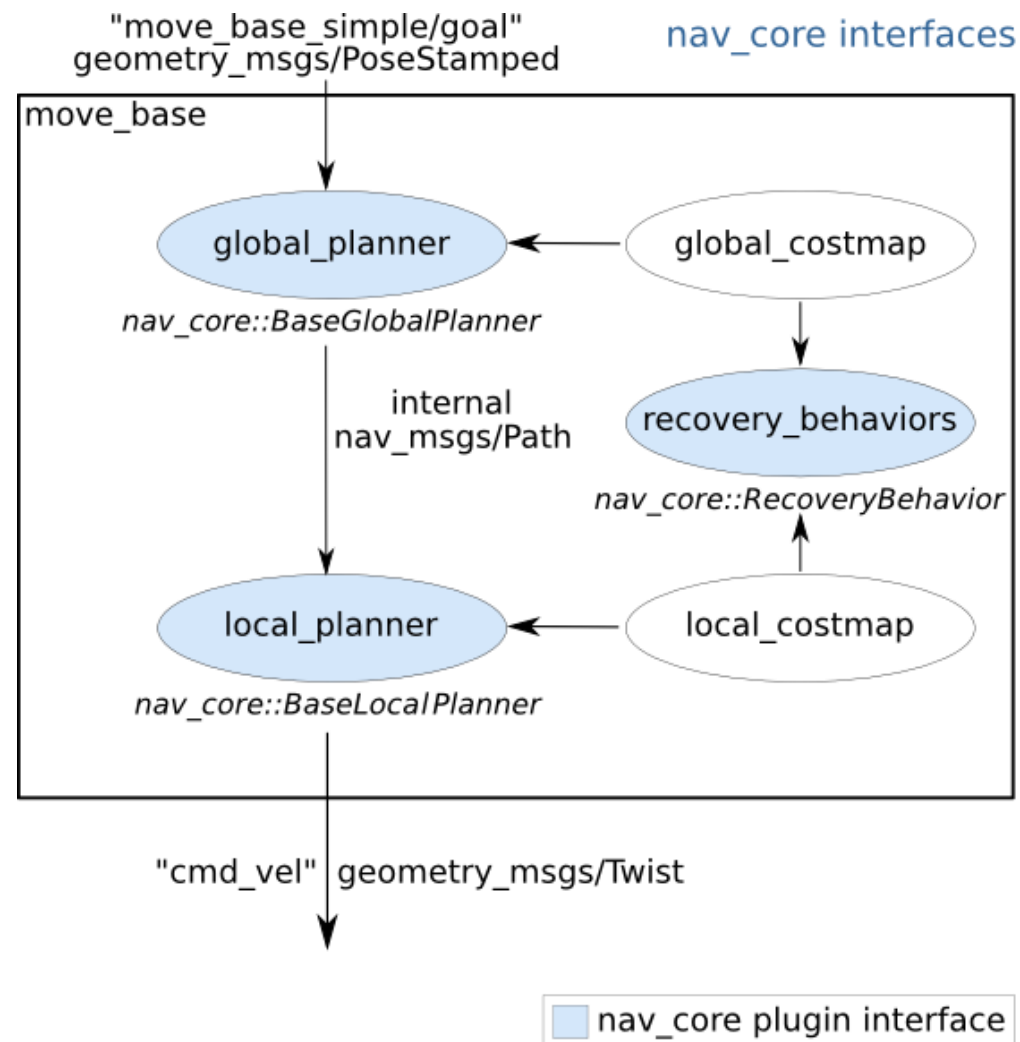
2. 导航框架中的关键功能包 —— move_base

➤ 全局路径规划 (global planner)

- 全局最优路径规划
- Dijkstra或A*算法

➤ 本地实时规划 (local planner)

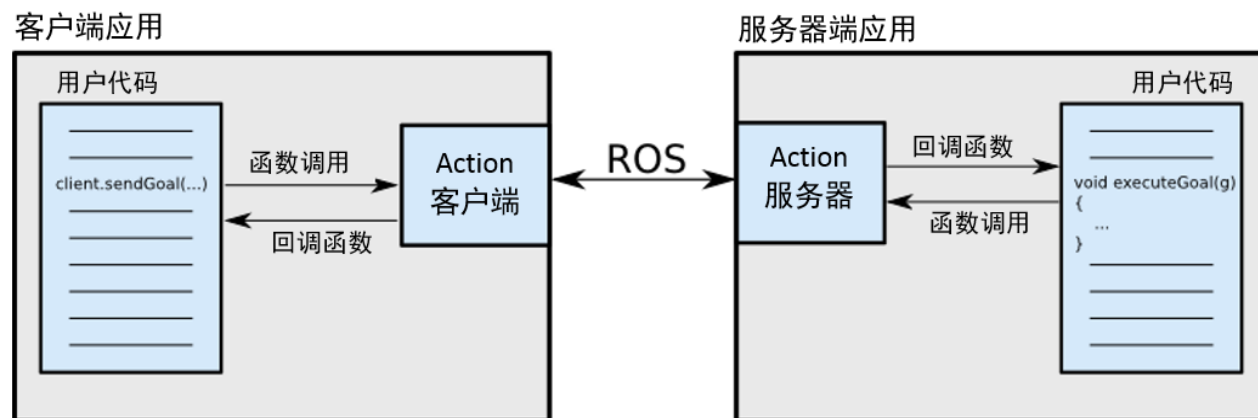
- 规划机器人每个周期内的线速度、角速度，使之尽量符合全局最优路径。
- 实时避障
- Trajectory Rollout 和 Dynamic Window Approaches算法
- 搜索躲避和行进的多条路径，综合各评价标准选取最优路径



2. 导航框架中的关键功能包 —— move_base

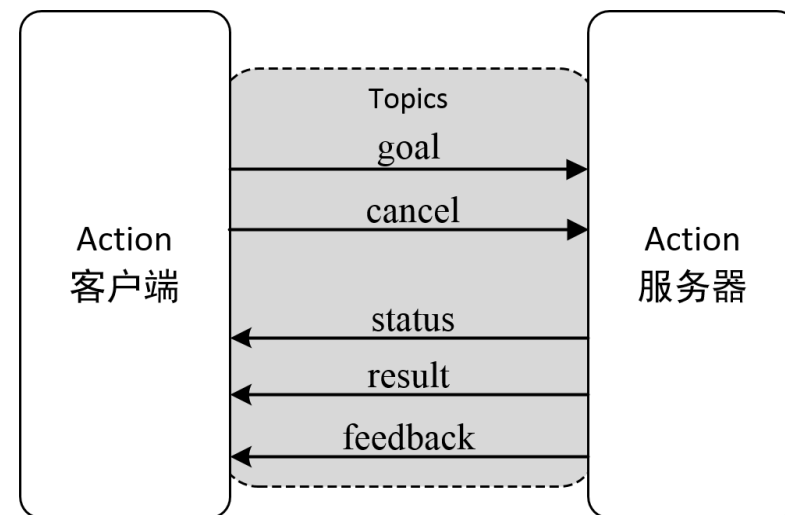
什么是动作（action）

- 一种问答通信机制;
- 带有连续反馈;
- 可以在任务过程中止运行;
- 基于ROS的消息机制实现。



Action的接口

- goal: 发布任务目标;
- cancel: 请求取消任务;
- status: 通知客户端当前的状态;
- feedback: 周期反馈任务运行的监控数据;
- result: 向客户端发送任务的执行结果, 只发布一次。



2. 导航框架中的关键功能包 —— move_base

move_base功能包中的话题和服务

	名称	类型	描述
Action 订阅	move_base/goal	move_base_msgs/ MoveBaseActionGoal	move_base的运动规划目标
	move_base/cancel	actionlib_msgs/GoalID	取消特定目标的请求
Action 发布	move_base/feedback	move_base_msgs/ MoveBaseActionFeedback	反馈信息，含有机器人底盘的坐标
	move_base/status	actionlib_msgs/ GoalStatusArray	发送到move_base的目标状态信息
	move_base/result	move_base_msgs/ MoveBaseActionResult	此处move_base操作的结果为空
Topic 订阅	move_base_simple/goal	geometry_msgs/PoseStamped	为不需要追踪目标执行状态的用户，提供一个非action接口
Topic 发布	cmd_vel	geometry_msgs/Twist	输出到机器人底盘的速度命令
Service	~make_plan	nav_msgs/GetPlan	允许用户从move_base获取给定目标的路径规划，但不会执行该路径规划
	~clear_unknown_space	std_srvs/Empty	允许用户直接清除机器人周围的未知空间。适合于costmap停止很长时间后，在一个全新环境中重新启动时使用
	~clear_costmaps	std_srvs/Empty	允许用户命令move_base节点清除costmap中的障碍。这可能会导致机器人撞上障碍物，请谨慎使用

2. 导航框架中的关键功能包 —— move_base

```
<launch>
```

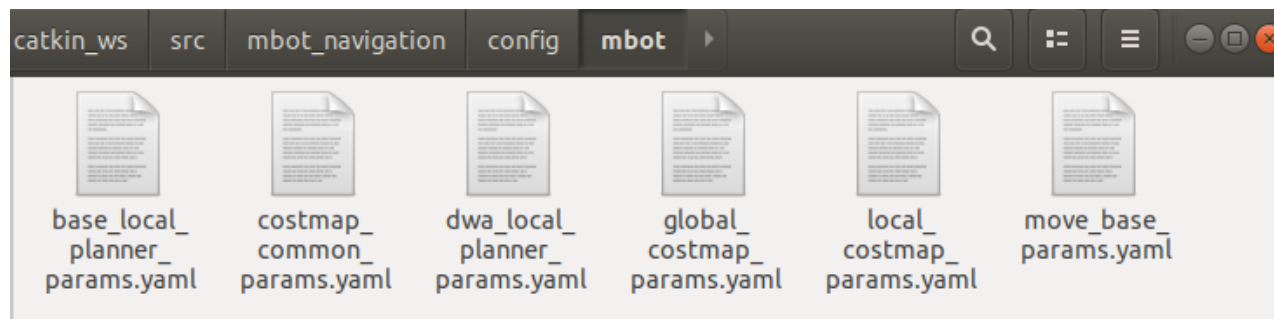
```
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen" clear_params="true">
  <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />

  <rosparam file="$(find mbot_navigation)/config/mbot/costmap_common_params.yaml" command="load" ns="global_costmap" />
  <rosparam file="$(find mbot_navigation)/config/mbot/costmap_common_params.yaml" command="load" ns="local_costmap" />
  <rosparam file="$(find mbot_navigation)/config/mbot/local_costmap_params.yaml" command="load" />
  <rosparam file="$(find mbot_navigation)/config/mbot/global_costmap_params.yaml" command="load" />

  <rosparam file="$(find mbot_navigation)/config/mbot/move_base_params.yaml" command="load" />
  <rosparam file="$(find mbot_navigation)/config/mbot/dwa_local_planner_params.yaml" command="load" />
</node>
```

```
</launch>
```

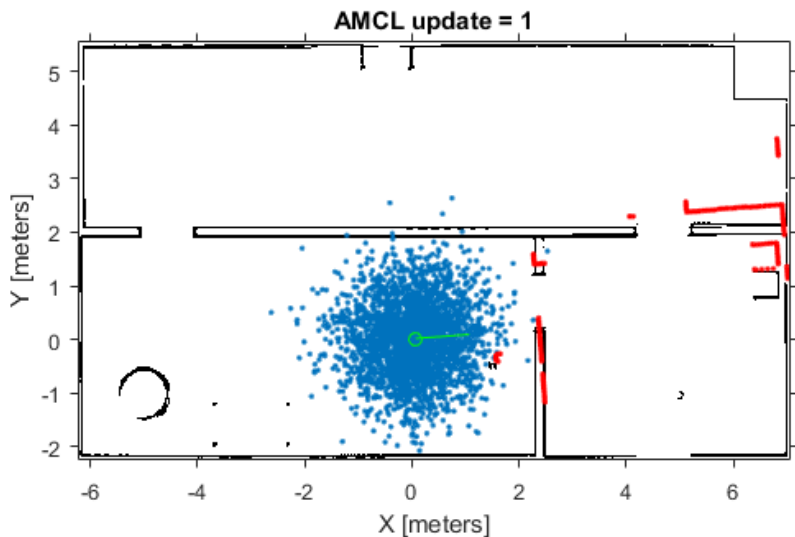
配置move_base节点 mbot_navigation/launch/move_base.launch



2. 导航框架中的关键功能包 —— amcl

amcl功能包中的话题和服务

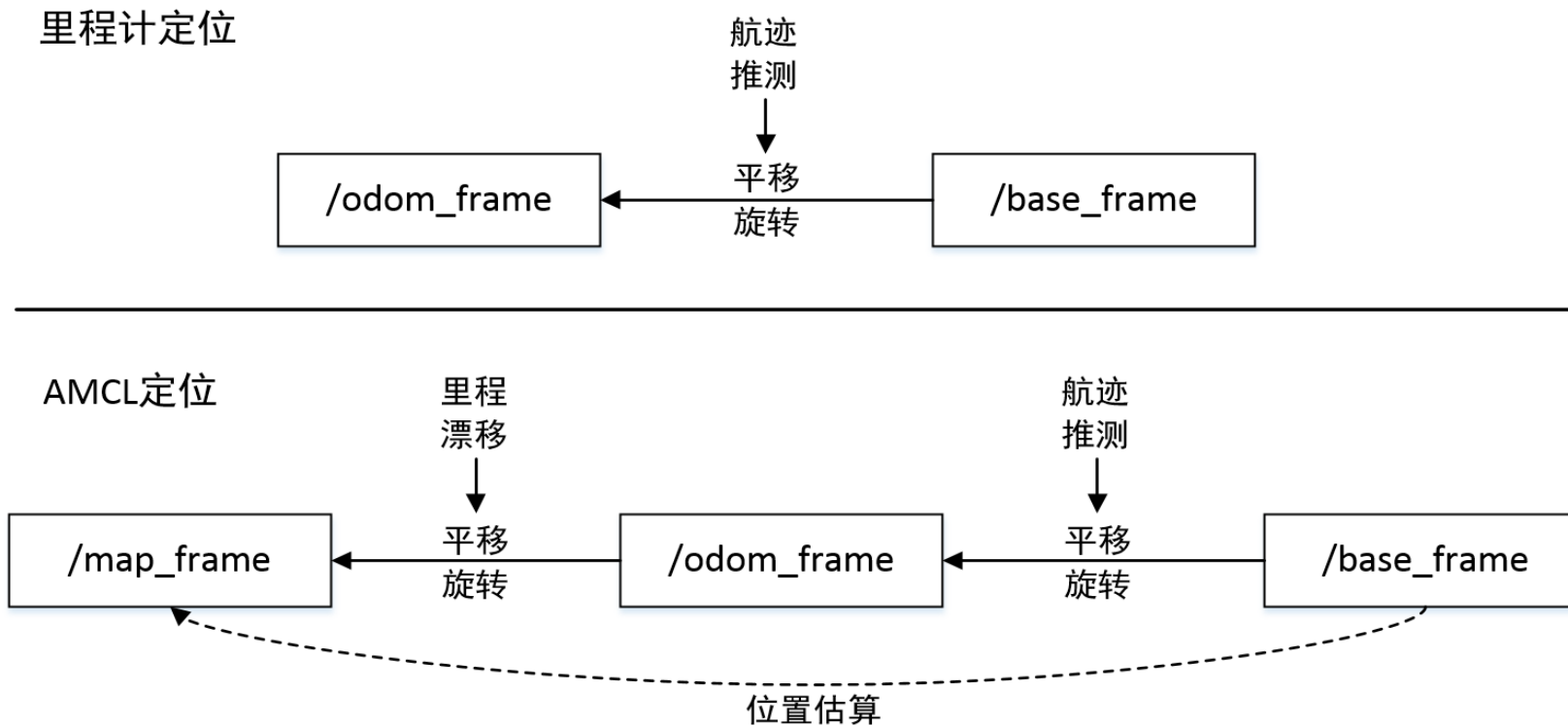
- 蒙特卡罗定位方法
- 二维环境定位
- 针对已有地图使用粒子滤波器跟踪一个机器人的姿态



	名称	类型	描述
Topic 订阅	scan	sensor_msgs/LaserScan	激光雷达数据
	tf	tf/tfMessage	坐标变换信息
	initialpose	geometry_msgs/PoseWithCovarianceStamped	用来初始化粒子滤波器的均值和协方差
	map	nav_msgs/OccupancyGrid	use_map_topic参数设置时，amcl订阅map话题以获取地图数据，用于激光定位
Topic 发布	amcl_pose	geometry_msgs/PoseWithCovarianceStamped	机器人在地图中的位姿估计，带有协方差信息
	particlecloud	geometry_msgs/PoseArray	粒子滤波器维护的位姿估计集合
	tf	tf/tfMessage	发布从odom（可以使用参数~odom_frame_id进行重映射）到map的转换
Service	global_localization	std_srvs/Empty	初始化全局定位，所有粒子被随机撒在地图上的空闲区域
	request_nomotion_update	std_srvs/Empty	手动执行更新并发布更新的粒子
Services Called	static_map	nav_msgs/GetMap	amcl调用该服务获取地图数据



2. 导航框架中的关键功能包 —— amcl



- 里程计定位：只通过里程计的数据来处理/`base`和/`odom`之间的TF转换；
- amcl定位：可以估算机器人在地图坐标系/`map`下的位姿信息，提供/`base`、/`odom`、/`map`之间的TF变换。



2. 导航框架中的关键功能包 —— amcl

```
<launch>
  <arg name="scan_topic" default="scan"/>
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>

  <node pkg="amcl" type="amcl" name="amcl" clear_params="true">
    <param name="min_particles" value="500"/>
    <param name="max_particles" value="3000"/>
    <param name="kld_err" value="0.02"/>
    <param name="update_min_d" value="0.20"/>
    <param name="update_min_a" value="0.20"/>
    <param name="resample_interval" value="1"/>
    <param name="transform_tolerance" value="0.5"/>
    <param name="recovery_alpha_slow" value="0.00"/>
    <param name="recovery_alpha_fast" value="0.00"/>
    <param name="initial_pose_x" value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y" value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a" value="$(arg initial_pose_a)"/>
    <param name="gui_publish_rate" value="50.0"/>

    <remap from="scan" to="$(arg scan_topic)"/>
    <param name="laser_max_range" value="3.5"/>
    <param name="laser_max_beams" value="180"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type" value="likelihood_field"/>

    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha1" value="0.1"/>
    <param name="odom_alpha2" value="0.1"/>
    <param name="odom_alpha3" value="0.1"/>
    <param name="odom_alpha4" value="0.1"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" value="base_footprint"/>
  </node>
</launch>
```

配置amcl节点

mbot_navigation/launch/amcl.launch



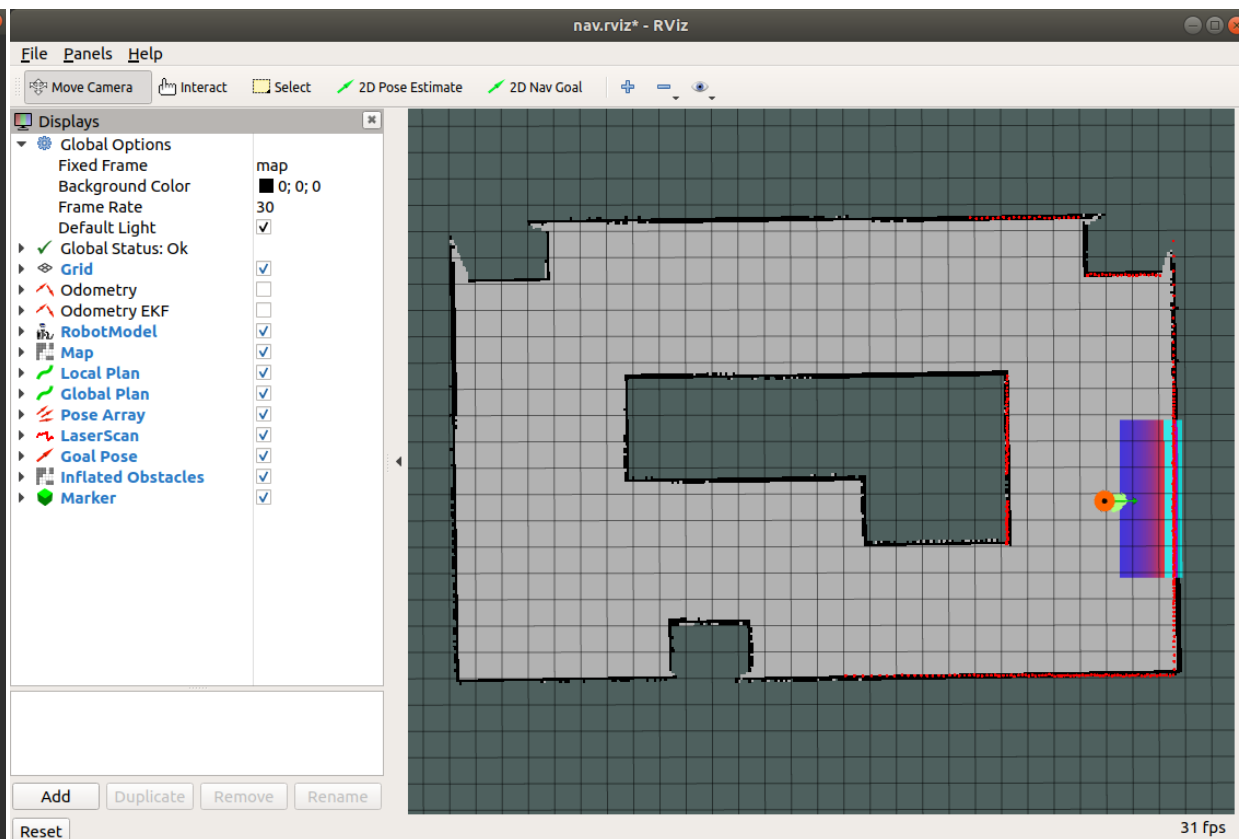
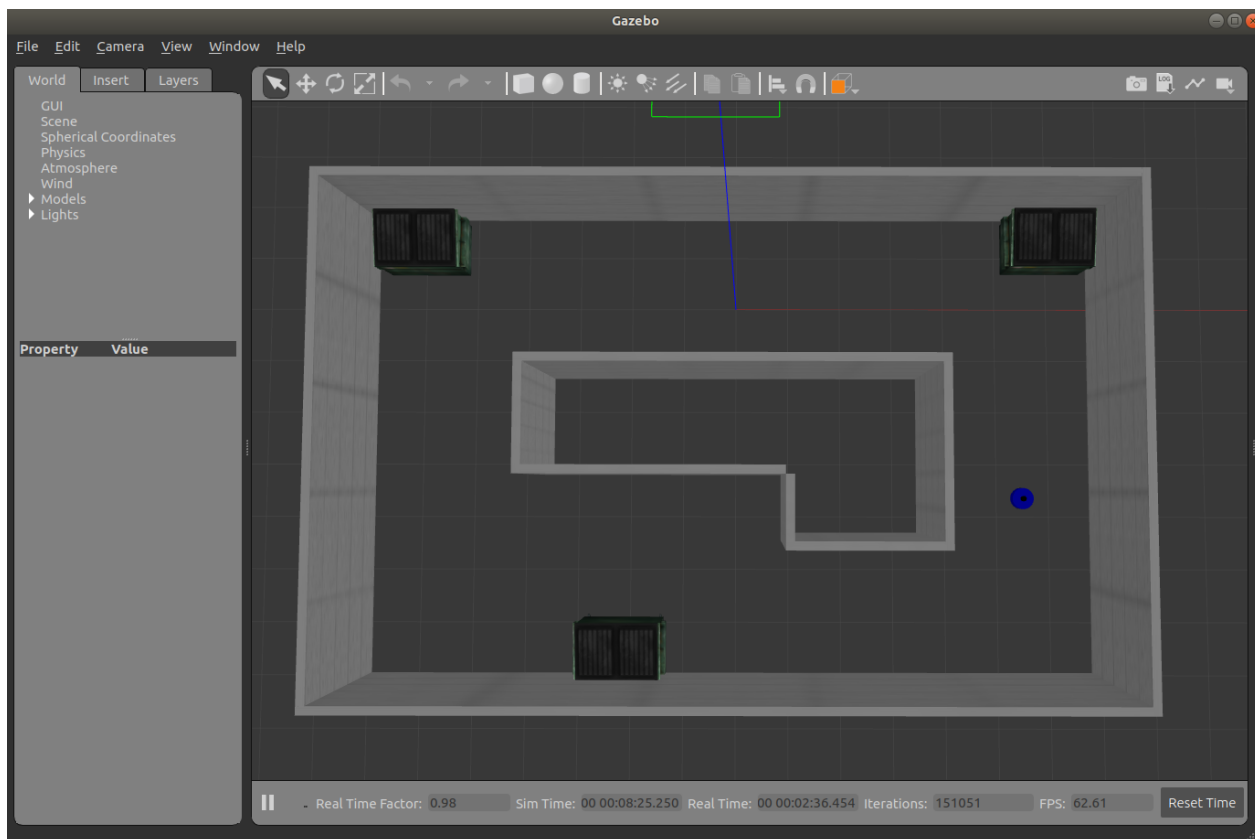
3. 机器人自主导航案例



3. 机器人自主导航案例 —— 导航仿真

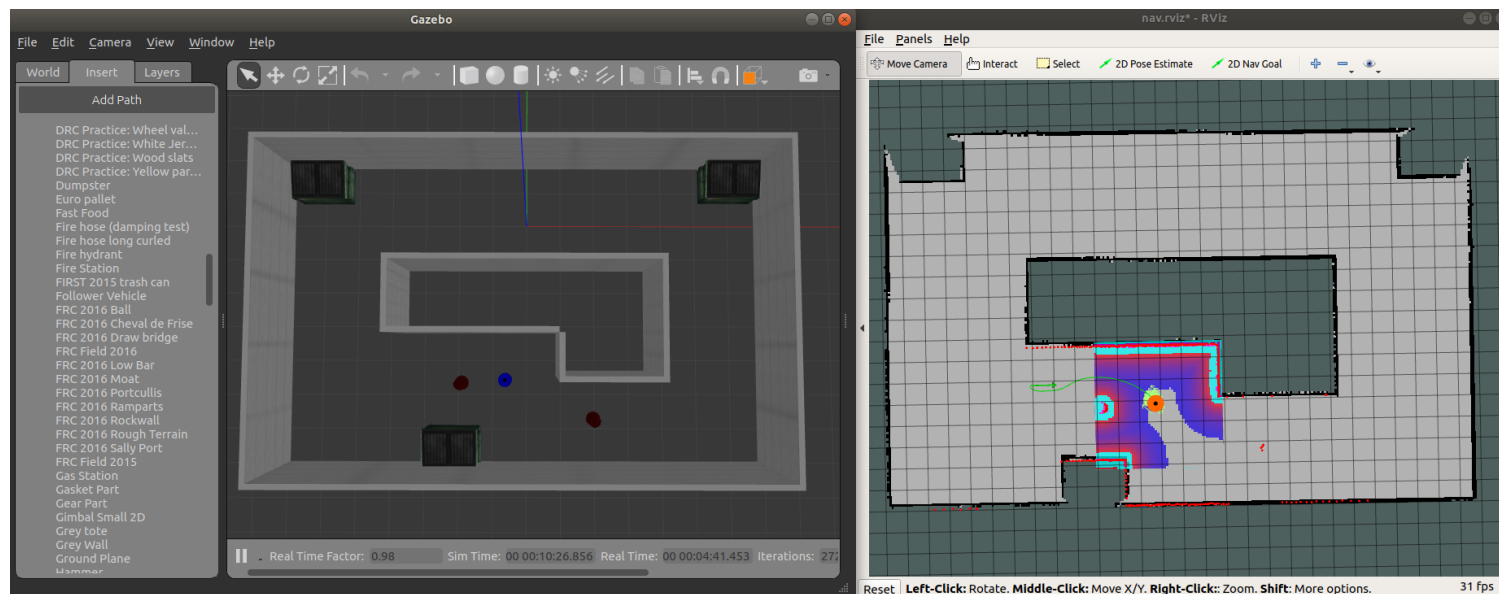
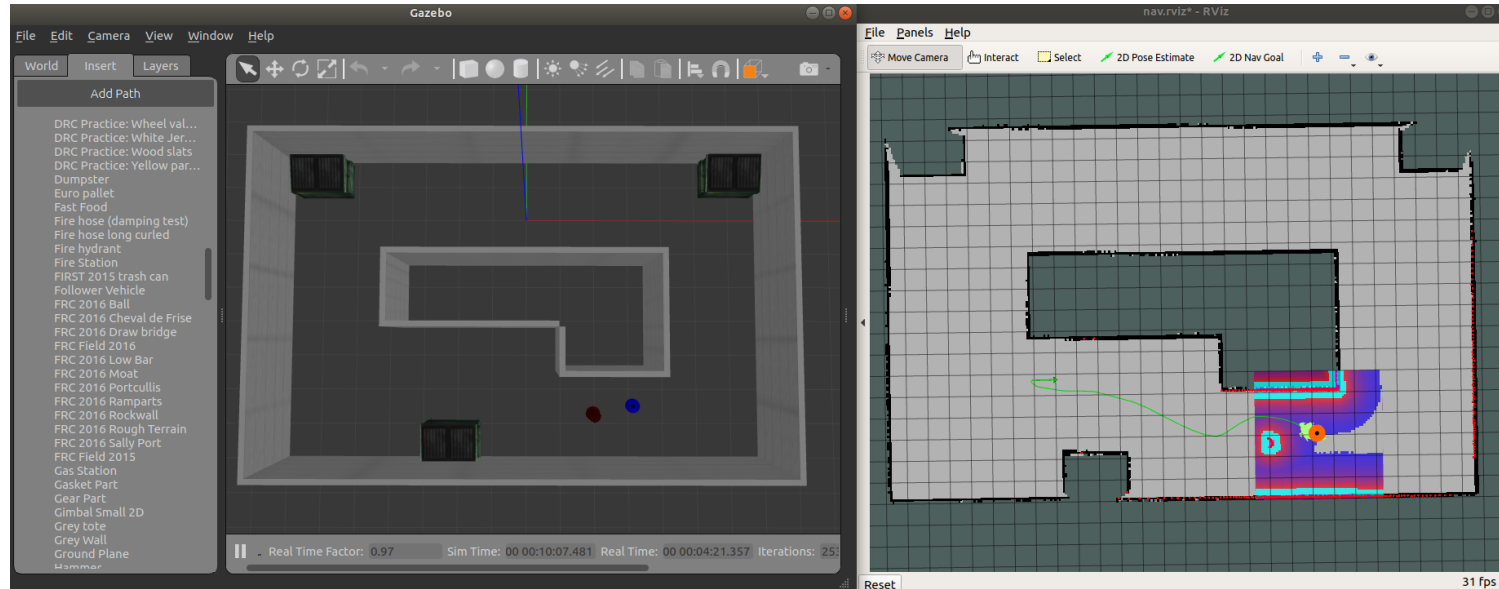
```
$ roslaunch mbot_gazebo mbot_laser_nav_gazebo.launch
```

```
$ roslaunch mbot_navigation nav_cloister_demo.launch
```





3. 机器人自主导航案例 —— 导航仿真



躲避动态出现的障碍物



3. 机器人自主导航案例 —— 程序接口

```
import roslib;
import rospy
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, PoseWithCovarianceStamped, Point, Quaternion, Twist
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal

# 节点初始化
rospy.init_node('move_test', anonymous=True)

# 订阅move_base服务器的消息
move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)

rospy.loginfo("Waiting for move_base action server...")

# 等待连接服务器, 5s等待时间限制
while move_base.wait_for_server(rospy.Duration(5.0)) == 0:
    rospy.loginfo("Connected to move base server")

# 设定目标点
target = Pose(Point(-5.543, -4.779, 0.000), Quaternion(0.000, 0.000, 0.645, 0.764))
goal = MoveBaseGoal()
goal.target_pose.pose = target
goal.target_pose.header.frame_id = 'map'
goal.target_pose.header.stamp = rospy.Time.now()

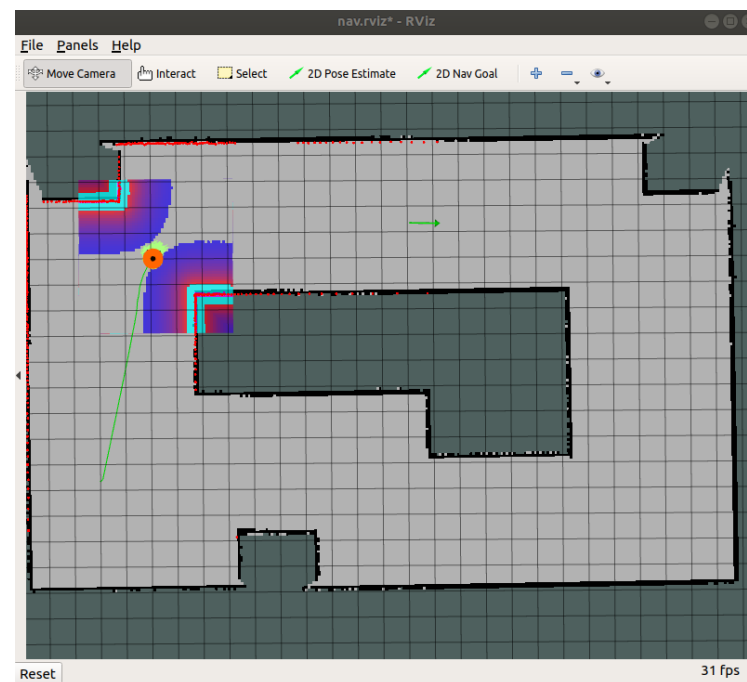
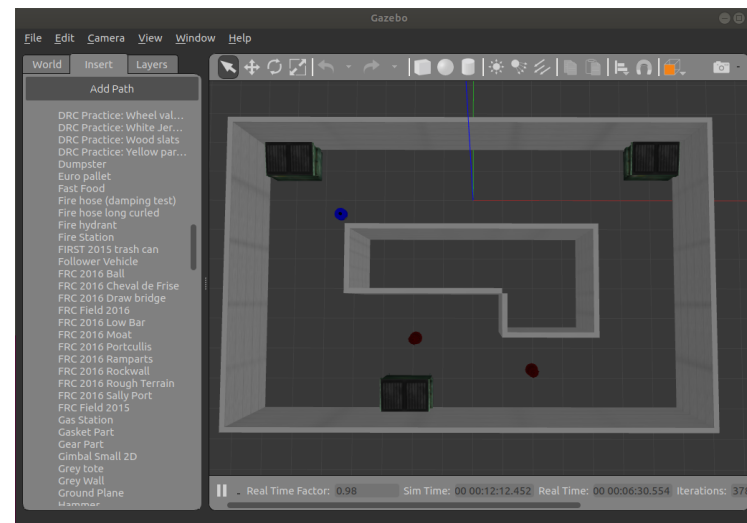
rospy.loginfo("Going to: " + str(target))

# 向目标进发
move_base.send_goal(goal)

# 五分钟时间限制
finished_within_time = move_base.wait_for_result(rospy.Duration(300))

# 查看是否成功到达
if not finished_within_time:
    move_base.cancel_goal()
    rospy.loginfo("Timed out achieving goal")
else:
    state = move_base.get_state()
    if state == GoalStatus.SUCCEEDED:
        rospy.loginfo("Goal succeeded!")
    else:
        rospy.loginfo("Goal failed! ")
```

mbot_navigation/scripts/move_test.py





3. 机器人自主导航案例 —— 程序接口

```
# 设置目标点的位置
# 在rviz中点击 2D Nav Goal 按键, 然后单击地图中一点
# 在终端中就会看到该点的坐标信息
locations = dict()

locations['1'] = Pose(Point(4.589, -0.376, 0.000), Quaternion(0.000, 0.000, -0.447, 0.894))
locations['2'] = Pose(Point(4.231, -6.050, 0.000), Quaternion(0.000, 0.000, -0.847, 0.532))
locations['3'] = Pose(Point(-0.674, -5.244, 0.000), Quaternion(0.000, 0.000, 0.000, 1.000))
locations['4'] = Pose(Point(-5.543, -4.779, 0.000), Quaternion(0.000, 0.000, 0.645, 0.764))
locations['5'] = Pose(Point(-4.701, -0.590, 0.000), Quaternion(0.000, 0.000, 0.340, 0.940))
locations['6'] = Pose(Point(2.924, 0.018, 0.000), Quaternion(0.000, 0.000, 0.000, 1.000))

# 发布控制机器人的消息
self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=5)

# 订阅move_base服务器的消息
self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)

rospy.loginfo("Waiting for move_base action server...")

# 60s等待时间限制
self.move_base.wait_for_server(rospy.Duration(60))
rospy.loginfo("Connected to move base server")

# 保存机器人的在rviz中的初始位置
initial_pose = PoseWithCovarianceStamped()

# 保存成功率、运行时间、和距离的变量
n_locations = len(locations)
n_goals = 0
n_successes = 0
i = n_locations
distance_traveled = 0
start_time = rospy.Time.now()
running_time = 0
location = ""
last_location = ""
```

```
# 设定下一个目标点
self.goal = MoveBaseGoal()
self.goal.target_pose.pose = locations[location]
self.goal.target_pose.header.frame_id = 'map'
self.goal.target_pose.header.stamp = rospy.Time.now()

# 让用户知道下一个位置
rospy.loginfo("Going to: " + str(location))

# 向下一个位置进发
self.move_base.send_goal(self.goal)

# 五分钟时间限制
finished_within_time = self.move_base.wait_for_result(rospy.Duration(300))

# 查看是否成功到达
if not finished_within_time:
    self.move_base.cancel_goal()
    rospy.loginfo("Timed out achieving goal")
else:
    state = self.move_base.get_state()
    if state == GoalStatus.SUCCEEDED:
        rospy.loginfo("Goal succeeded!")
        n_successes += 1
        distance_traveled += distance
        rospy.loginfo("State:" + str(state))
    else:
        rospy.loginfo("Goal failed with error code: " + str(goal_states[state]))

# 运行所用时间
running_time = rospy.Time.now() - start_time
running_time = running_time.secs / 60.0

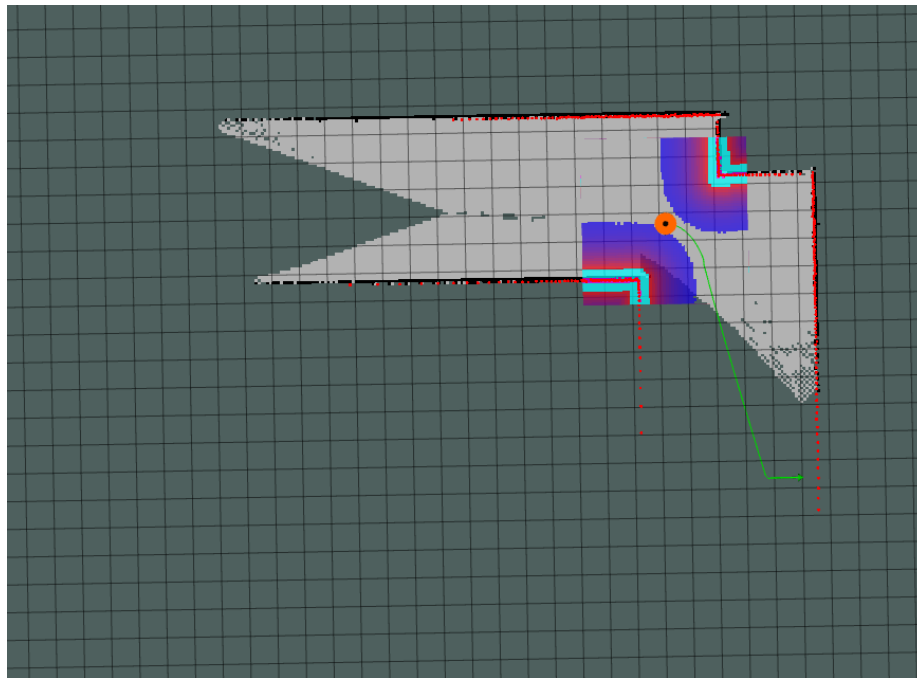
# 输出本次导航的所有信息
rospy.loginfo("Success so far: " + str(n_successes) + "/" +
              str(n_goals) + " = " +
              str(100 * n_successes/n_goals) + "%")
```



3. 机器人自主导航案例 —— move_base + gmapping

```
$ roslaunch mbot_gazebo mbot_laser_nav_gazebo.launch  
$ roslaunch mbot_navigation exploring_slam_demo.launch
```

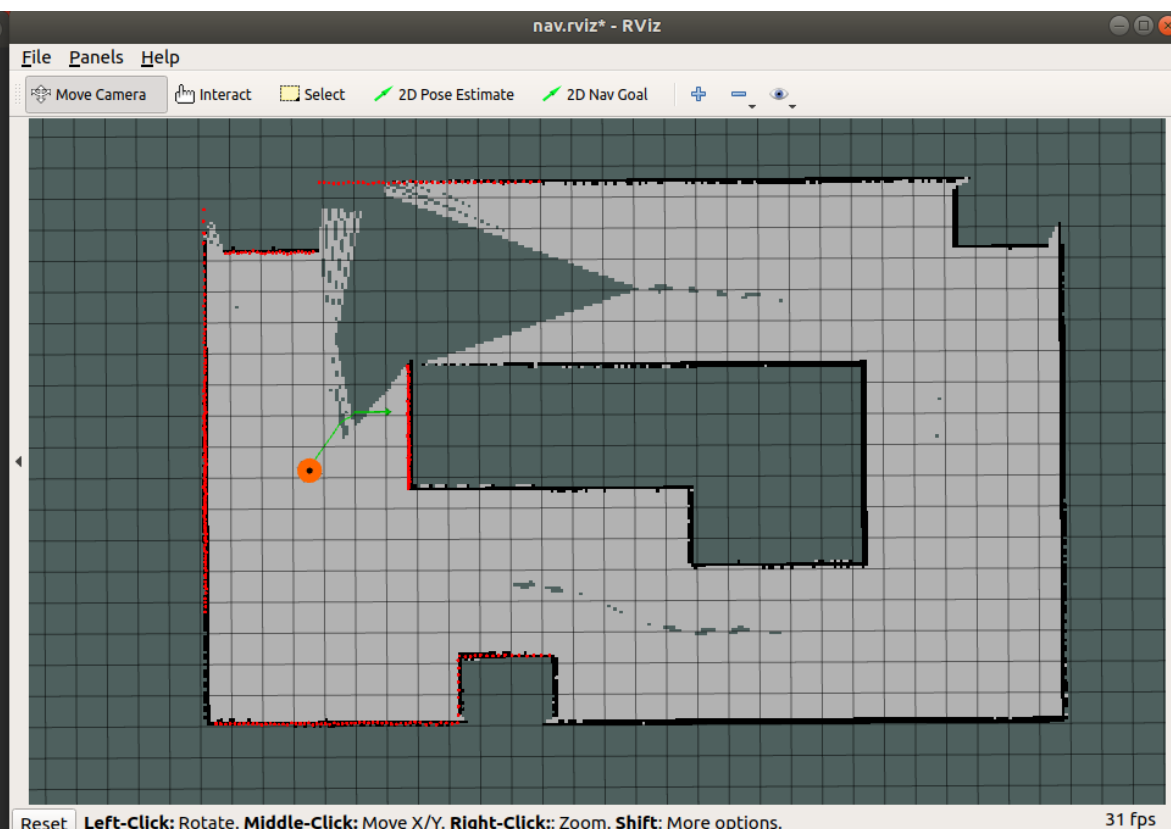
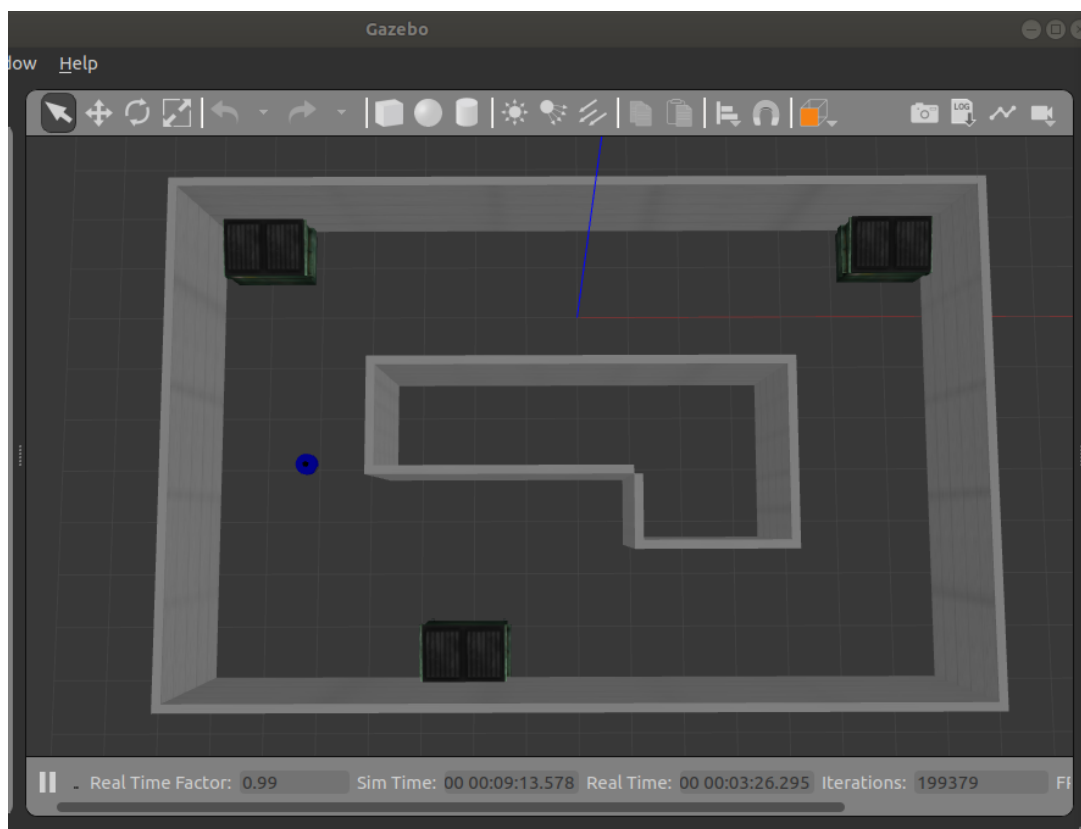
```
<launch>  
  
  <include file="$(find mbot_navigation)/launch/gmapping.launch"/>  
  
  <!-- 运行move_base节点 -->  
  <include file="$(find mbot_navigation)/launch/move_base.launch" />  
  
  <!-- 运行rviz -->  
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find mbot_navigation)/rviz/nav.rviz"/>  
  
</launch>
```





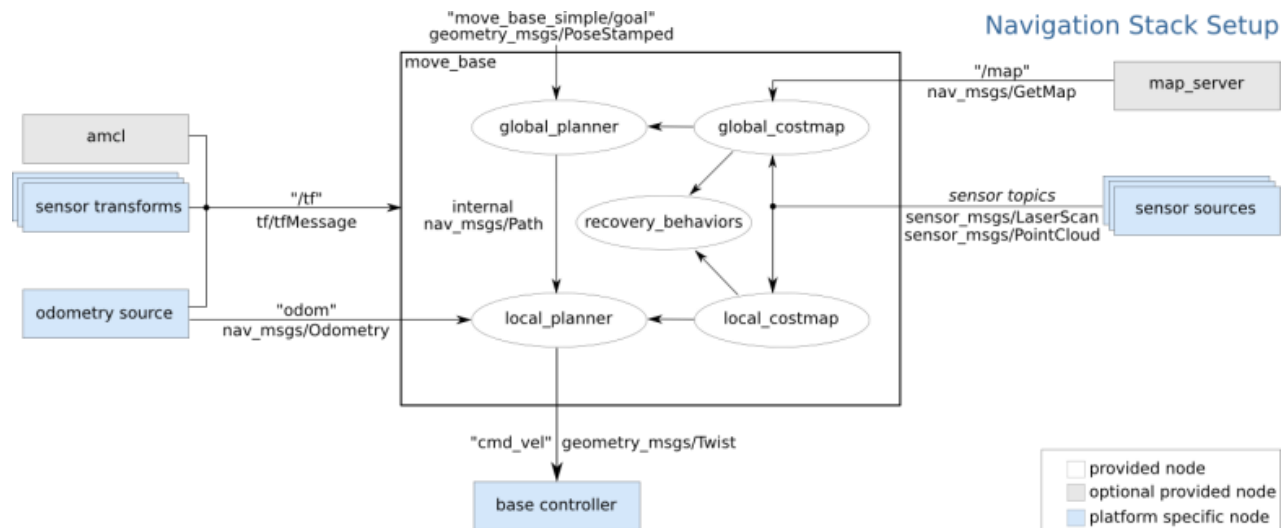
3. 机器人自主导航案例 —— move_base + gmapping

```
$ roslaunch mbot_gazebo mbot_laser_nav_gazebo.launch  
$ roslaunch mbot_navigation exploring_slam_demo.launch  
$ rosrun mbot_navigation exploring_random.py
```





ROS中的导航框架



导航框架中的关键功能包

- move_base
- amcl

机器人自主导航案例

- 导航仿真
- move_base程序接口
- move_base + gmapping



1. 在上讲作业建立完成的地图上，实现基于move_base和amcl功能包的机器人自主导航仿真；
2. 仿照本讲move_base+gmapping例程，实现导航与SLAM的同步仿真。



- ROS Navigation Wiki
<http://wiki.ros.org/navigation>
- ROS探索总结（十九）—— 如何配置机器人的导航功能
<http://www.guyuehome.com/281>
- ROS探索总结（二十）—— 发布导航需要的传感器信息
<http://www.guyuehome.com/326>
- ROS探索总结（二十一）—— 如何发布里程计消息
<http://www.guyuehome.com/332>
- 《ROS机器人开发实践》
第九章

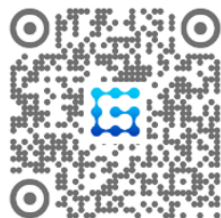




Thank You

怕什么真理无穷，进一寸有一寸的欢喜

更多精彩，欢迎关注



 古月居



 古月春旭