# 机器人自主导航

**实验目的：**

1 了解移动机器人导航功能；

2 学会如何编程控制机器人进行目标点导航；

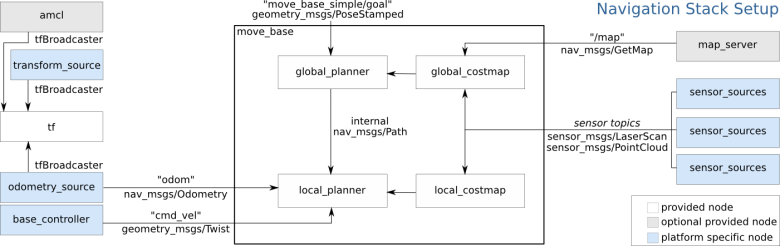
**实验原理：**

1 导航原理

自主导航是智能移动机器人最重要也是最基础的功能之一，虽然SLAM是机器人自主移动的关键，但SLAM并不表示自主导航，SLAM仅仅是解决地图构建和即时定位问题，而自主导航则是解决机器人移动时与环境的自主交互，点到点的的自主移动问题。完整的自主导航应该包含SLAM、路径规划和运动控制。

Navigation是机器人最基本的功能之一，ROS为我们提供了一整套Navigation的解决方案，包括全局与局部的路径规划、代价地图、异常行为恢复、地图服务器等等，这些开源工具包极大地减少了我们开发的工作量，任何一套移动机器人硬件平台经过这套方案就可以快速部署实现。

Navigation Stack的源代码位于https://github.com/ros-planning/navigation，如下为导航框架图：



上图中位于导航功能正中心的是 move\_base 节点，可以理解为一个强大的路径规划器，在实际的导航任务中，你只需要启动这一个node，并且给他提供数据，就可以规划出路径和速度。 move\_base 之所以能做到路径规划，是因为它包含了很多的插件，像图中的白色圆圈 global\_planner 、 local\_planner 、 global\_costmap 、 local\_costmap 、 recovery\_behaviors。这些插件用于负责一些更细微的任务：全局规划、局部规划、全局地图、局部地图、恢复行为。而每一个插件其实也都是一个package，放在Navigation Stack里。

ROS为我们提供了完整的导航框架，我们可以通过接口“move\_base\_simple/goal”给导航框架发送导航目标从而控制机器人自主导航到指定的位置。

2 安装必要的功能包：

由于本镜像缺少局部路径规划插件dwa\_local\_planner,可以使用以下命令进行安装。

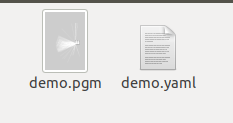
sudo apt-get install ros-melodic-dwa-local-planner

3 导航配置包下载



其中bobac3\_navigation为服务组导航包，oryxbot\_navigation为工业组导航包。选择自己需要的导航包下载并放到自己的工作空间中进行编译。

4 将上个实验中(地图构建)绘制的地图文件以及地图信息文件拷贝到导航包路径下maps文件夹里面。

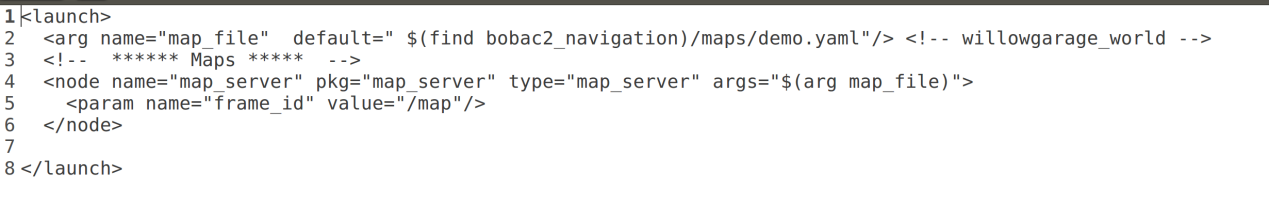


如果你没有绘制地图，请按照地图构建实验绘制场地的地图。

5 修改地图

打开ros\_workspace/src/bobac3\_navigation/launch(服务组)路径下的map\_server.launch，如下图所示：

打开ros\_workspace/src/oryxbot\_navigation/launch(工业组)路径下的map\_server.launch，如下图所示：



将红色框中的地图修改为之前slam画好的地图的名字。

6 运行以下指令启动导航功能包

服务组使用以下命令：

roslaunch bobac3\_navigation demo\_nav\_2d.launch

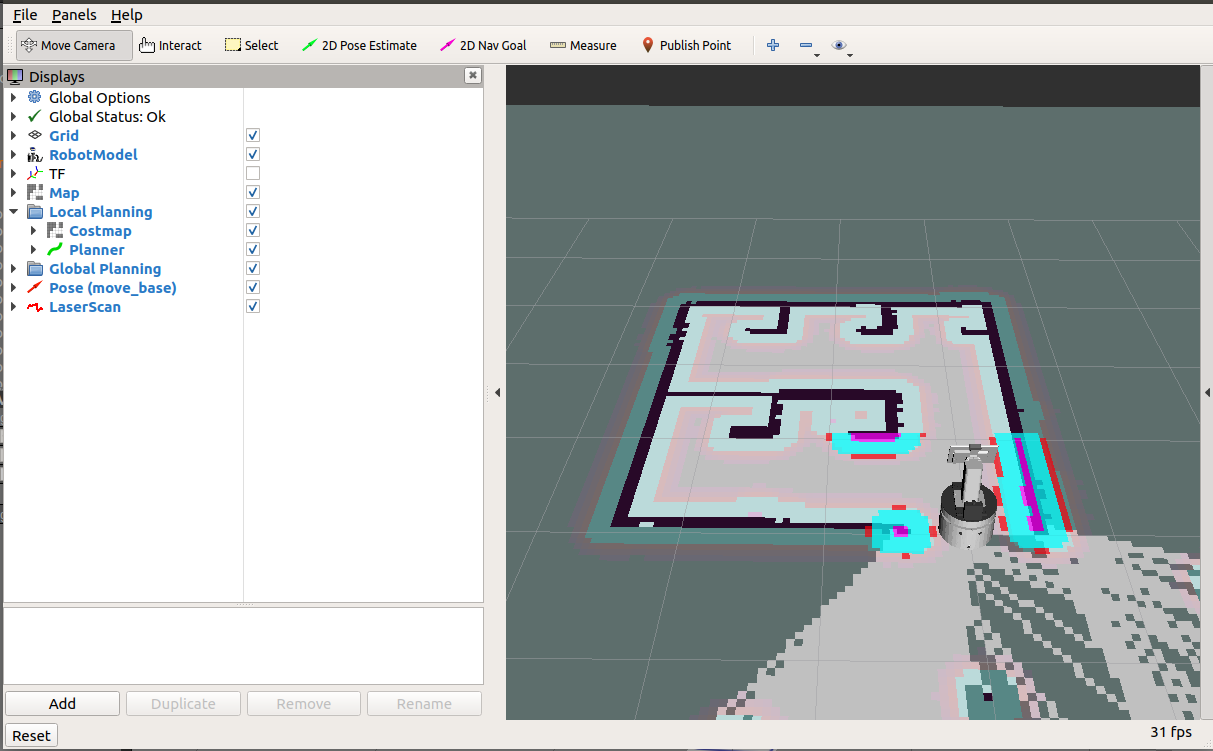
工业组使用以下命令：

roslaunch oryxbot\_navigation demo\_nav\_2d.launch

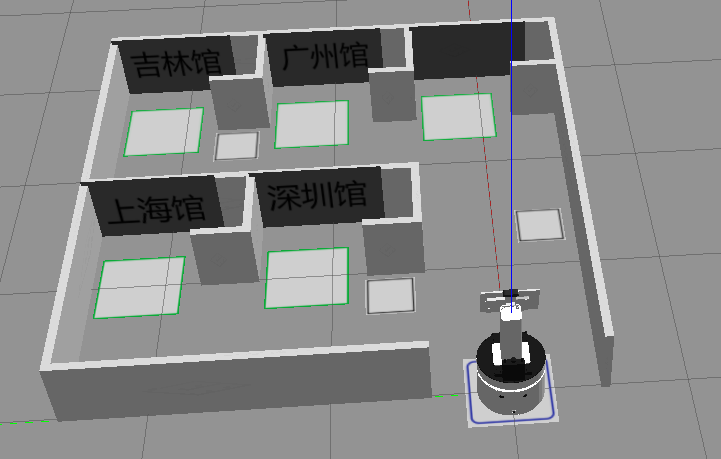
此时会打开rviz和gazebo两个软件，并已加载好地图。

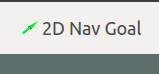
下面以服务组为例说明。

如下图所示为rviz显示画面：



如下图所示为gazebo画面：

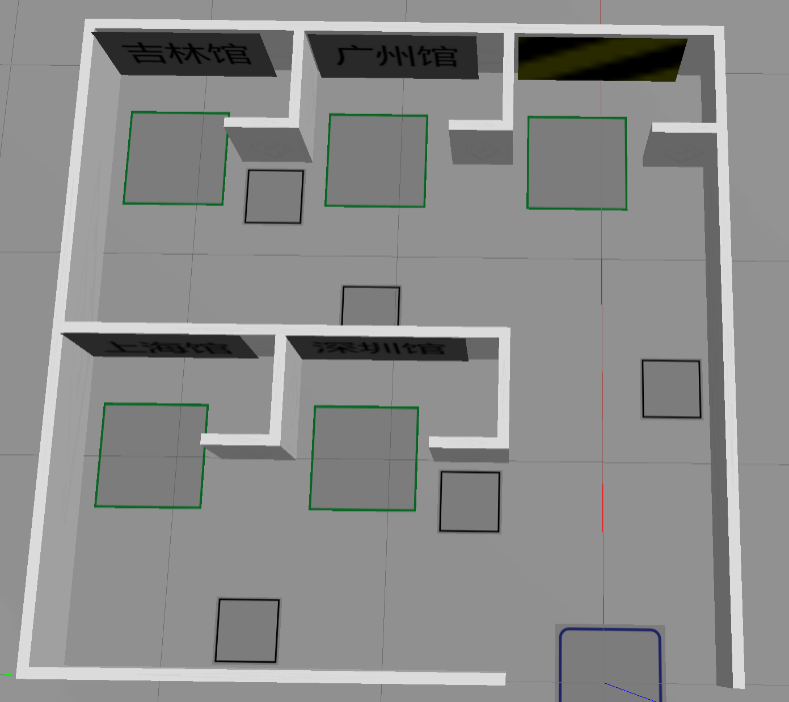


7 使用rviz界面菜单栏中的可以指定机器人导航的目标点，点击“2D Nav Goal”将绿色的箭头放到期望机器人到达的目标点，箭头的位置为期望机器人达到的位置，箭头的方向为机器人到达后的朝向。

8如何获取地图中点的位置

要编写程序控制机器人到达指定的目标点之前，我们需要知道指定位置的坐标，如下图所示，我们来示范如何获取指定位置的坐标。

如下图所示我们来获取红点在地图中的坐标。



8.1 打开机器人导航功能包(**如果已经打开，请忽略！**)

服务组：

roslaunch bobac3\_navigation demo\_nav\_2d.launch

工业组：

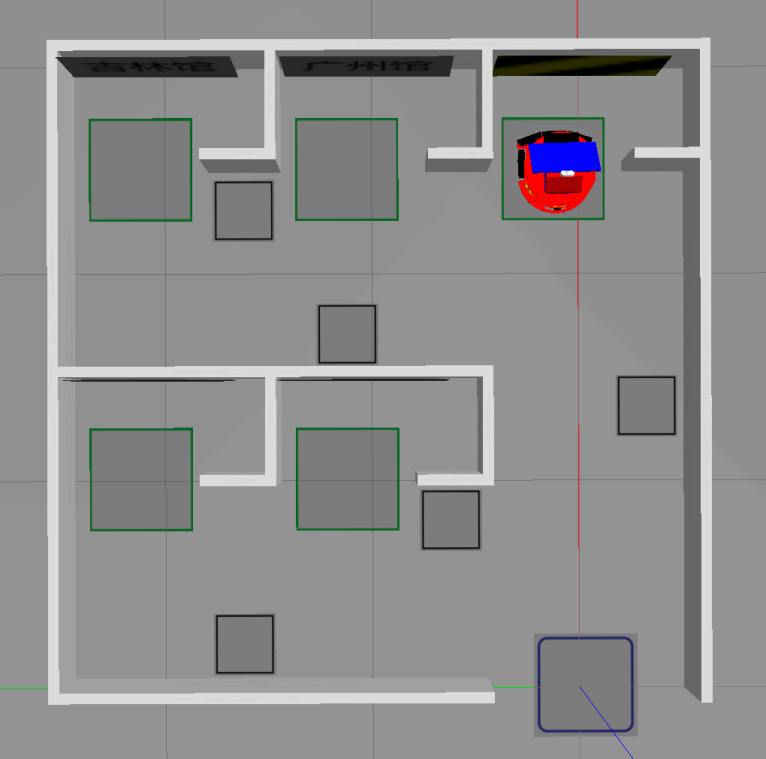
roslaunch oryxbot\_navigation demo\_nav\_2d.launch

8.2 使用rviz中的“2D Nav Goal”将机器人导航目标点附件

8.3 在终端运行以下指令启动键盘控制程序

rosrun teleop\_twist\_keyboard teleop\_twist\_keyboard.py

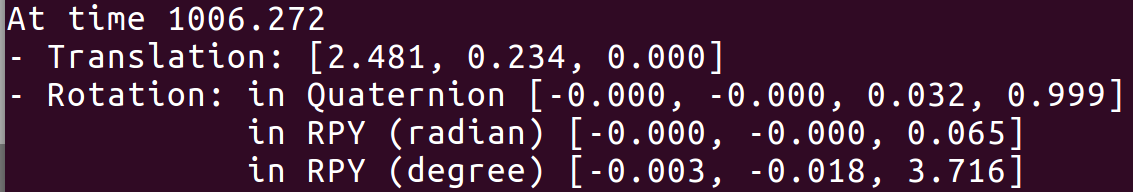
8.4 使用键盘控制将机器人微调控制到指定位置，如下图所示：



8.5 运行以下指令查看机器人在地图中的坐标

rosrun tf tf\_echo /map base\_footprint

如下图所示：



x

y

z

w

(x,y,z,w)即为机器人在地图中的坐标和方位，其中(x,y)为坐标，(z,w)为朝向的四元数表达方式。

**实验步骤：**

**任务一：控制机器人导航到一个位置**

**1创建功能包：**

在终端输入以下命令切换到ros\_workspace\_ws/src目录下：

$ cd ~/ros\_workspace/src

输入以下命令创建功能包nav\_goal

$ catkin\_create\_pkg nav\_goal roscpp rospy actionlib geometry\_msgs move\_base\_msgs

该功能包依赖了roscpp、rospy、actionlib、geometry\_msgs 、move\_base\_msgs等功能包。

**2编写源码**

在目录~/ros\_workspace/src/nav\_goal/src下创建一个源文件命名为：nav\_goal.cpp，打开该文件，输入以下代码。

#include "ros/ros.h"//包含ROS头文件

#include <move\_base\_msgs/MoveBaseAction.h>//包含move\_base action头文件

#include <actionlib/client/simple\_action\_client.h>//包含simple\_aciton客户端头文件

main(int argc,char\*\* argv)

{

ros::init(argc,argv,"nav\_goal");//ros初始化

ros::NodeHandle nh;//定义句柄

actionlib::SimpleActionClient<move\_base\_msgs::MoveBaseAction> ac("move\_base",true);//定义一个acition客户端 ，用于发送机器人要到达的目标

ac.waitForServer();//等待服务开启

move\_base\_msgs::MoveBaseGoal goal;//定义一个存储目标的变量

goal.target\_pose.header.frame\_id = "map";

goal.target\_pose.header.stamp = ros::Time::now();

goal.target\_pose.pose.position.x = 2.477;//目标点x

goal.target\_pose.pose.position.y = 0.151;//目标点y

goal.target\_pose.pose.orientation.z = 0;

goal.target\_pose.pose.orientation.w = 1;//目标点姿态四元数表示

ac.sendGoal(goal);//发送目标

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}//等待机器人执行完成

ROS\_INFO("机器人到达目标点");

return 0;

}

**3修改编译配置文件**

保存以上代码，修改nav\_goal功能包下的CMakeLists.txt文件的第138行附近如下，将nav\_goal.cpp编译为可执行文件。

add\_executable(${PROJECT\_NAME}\_node src/nav\_goal.cpp)

修改nav\_goal功能包下的CMakeLists.txt文件的第151行附近如下，添加链接项。

target\_link\_libraries(${PROJECT\_NAME}\_node

   ${catkin\_LIBRARIES}

 )

**4编译代码**

切到工作空间目录：

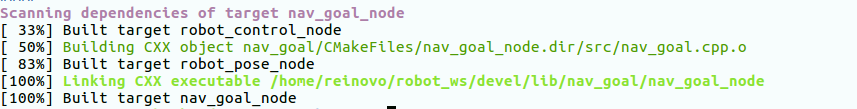
$ cd ~/ros\_workspace/

编译工作空间：

$ catkin\_make

**4编译代码**

成功则如下：



**5运行功能包查看效果**

运行以下命令打开机器人自主导航功能：

服务组：

roslaunch bobac3\_navigation demo\_nav\_2d.launch

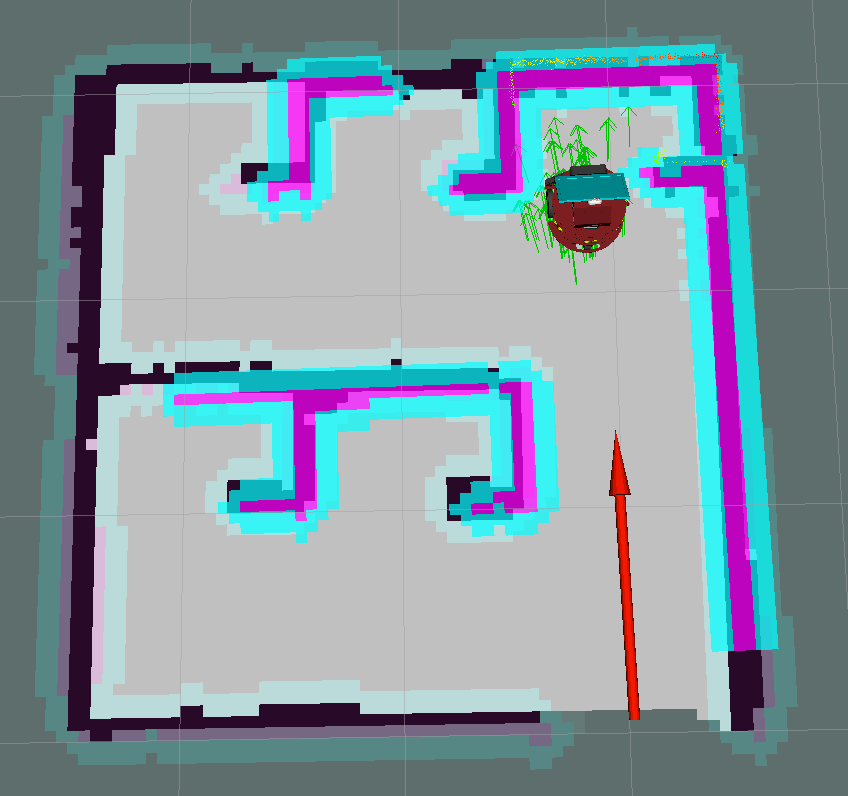
工业组：

roslaunch oryxbot\_navigation demo\_nav\_2d.launch

运行以下命令打开nav\_goal功能包：

$ rosrun nav\_goal nav\_goal\_node

如下图所示机器人已经规划了路线，开始移动。



**任务二：编程实现控制机器人在指定区域内巡航**

接下来我们来实现让机器人在指定的区域自主巡检n次。

**1编写代码**

在目录~/ros\_workspace/src/nav\_goal/src下创建一个源文件命名为：nav\_multigoal.cpp，打开该文件，输入以下代码。

#include"ros/ros.h"//包含ROS头文件

#include <move\_base\_msgs/MoveBaseAction.h>//包含move\_base action头文件

#include <actionlib/client/simple\_action\_client.h>//包含simple\_aciton客户端头文件

main(int argc,char\*\* argv)

{

ros::init(argc,argv,"nav\_goal");

ros::NodeHandle nh;

actionlib::SimpleActionClient<move\_base\_msgs::MoveBaseAction> ac("move\_base",true);//定义一个acition客户端

ac.waitForServer();//等待服务开启

move\_base\_msgs::MoveBaseGoal goal1;//定义第一个目标点 北京馆

goal1.target\_pose.header.frame\_id = "map";

goal1.target\_pose.header.stamp = ros::Time::now();

goal1.target\_pose.pose.position.x = 2.48;//目标点x

goal1.target\_pose.pose.position.y = 0.13;//目标点y

goal1.target\_pose.pose.orientation.z = 0.03;

goal1.target\_pose.pose.orientation.w = 0.99;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal2;//定义第二个目标点 ，广州馆

goal2.target\_pose.header.frame\_id = "map";

goal2.target\_pose.header.stamp = ros::Time::now();

goal2.target\_pose.pose.position.x = 2.48;//目标点x

goal2.target\_pose.pose.position.y = 1.077; //目标点y

goal2.target\_pose.pose.orientation.z = -0.026;

goal2.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal3;//定义第三个目标点 ，吉林馆

goal3.target\_pose.header.frame\_id = "map";

goal3.target\_pose.header.stamp = ros::Time::now();

goal3.target\_pose.pose.position.x = 2.48;//目标点x

goal3.target\_pose.pose.position.y = 2.120; //目标点y

goal3.target\_pose.pose.orientation.z = -0.032;

goal3.target\_pose.pose.orientation.w = 0.999;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal4;//定义第四个目标点 ，深圳馆

goal4.target\_pose.header.frame\_id = "map";

goal4.target\_pose.header.stamp = ros::Time::now();

goal4.target\_pose.pose.position.x =1.026;//目标点x

goal4.target\_pose.pose.position.y = 1.109; //目标点y

goal4.target\_pose.pose.orientation.z = -0.012;

goal4.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal5;//定义第五个目标点 ，上海馆

goal5.target\_pose.header.frame\_id = "map";

goal5.target\_pose.header.stamp = ros::Time::now();

goal5.target\_pose.pose.position.x =1.073;//目标点x

goal5.target\_pose.pose.position.y = 2.120; //目标点y

goal5.target\_pose.pose.orientation.z =0.013;

goal5.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

int Num = 2;//定义巡检的圈数

for(int i=0;i<Num;i++)

{

ac.sendGoal(goal1);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点1");

ac.sendGoal(goal2);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点2");

ac.sendGoal(goal3);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点3");

ac.sendGoal(goal4);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点4");

ac.sendGoal(goal5);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点5");

}

return 0;

}

**2编译代码**

在nav\_goal功能包下的CMakeLists.txt文件夹第139行附近添加如下代码：

add\_executable(nav\_multigoal\_node src/nav\_multigoal.cpp)

在nav\_goal功能包下的CMakeLists.txt文件夹第154行附近添加如下代码：

target\_link\_libraries(nav\_multigoal\_node

${catkin\_LIBRARIES}

)

切到工作空间目录下：

切到工作空间目录：

$ cd ~/ ros\_workspace /

编译工作空间：

$ catkin\_make

**4运行功能包查看效果**

运行以下命令打开机器人自主导航功能：

服务组：

roslaunch bobac3\_navigation demo\_nav\_2d.launch

工业组：

roslaunch oryxbot\_navigation demo\_nav\_2d.launch

运行以下命令打开nav\_goal功能包：

$ rosrun nav\_goal nav\_multigoal\_node

机器人开始移动。

**5 代码改进**：

#include"ros/ros.h"//包含ROS头文件

#include <move\_base\_msgs/MoveBaseAction.h>//包含move\_base action头文件

#include <actionlib/client/simple\_action\_client.h>//包含simple\_aciton客户端头文件

main(int argc,char\*\* argv)

{

ros::init(argc,argv,"nav\_goal");

ros::NodeHandle nh;

actionlib::SimpleActionClient<move\_base\_msgs::MoveBaseAction> ac("move\_base",true);//定义一个acition客户端

ac.waitForServer();//等待服务开启

move\_base\_msgs::MoveBaseGoal goal1\_pre;//定义第一个目标点 北京馆

goal1\_pre.target\_pose.header.frame\_id = "map";

goal1\_pre.target\_pose.header.stamp = ros::Time::now();

goal1\_pre.target\_pose.pose.position.x = 2.28;//目标点x

goal1\_pre.target\_pose.pose.position.y = 0.13;//目标点y

goal1\_pre.target\_pose.pose.orientation.z = 0.03;

goal1\_pre.target\_pose.pose.orientation.w = 0.99;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal1;//定义第一个目标点 北京馆

goal1.target\_pose.header.frame\_id = "map";

goal1.target\_pose.header.stamp = ros::Time::now();

goal1.target\_pose.pose.position.x = 2.48;//目标点x

goal1.target\_pose.pose.position.y = 0.13;//目标点y

goal1.target\_pose.pose.orientation.z = 0.03;

goal1.target\_pose.pose.orientation.w = 0.99;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal2\_pre;//定义第二个目标点 ，广州馆

goal2\_pre.target\_pose.header.frame\_id = "map";

goal2\_pre.target\_pose.header.stamp = ros::Time::now();

goal2\_pre.target\_pose.pose.position.x = 2.28;//目标点x

goal2\_pre.target\_pose.pose.position.y = 1.077; //目标点y

goal2\_pre.target\_pose.pose.orientation.z = -0.026;

goal2\_pre.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal2;//定义第二个目标点 ，广州馆

goal2.target\_pose.header.frame\_id = "map";

goal2.target\_pose.header.stamp = ros::Time::now();

goal2.target\_pose.pose.position.x = 2.48;//目标点x

goal2.target\_pose.pose.position.y = 1.077; //目标点y

goal2.target\_pose.pose.orientation.z = -0.026;

goal2.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal3\_pre;//定义第三个目标点 ，吉林馆

goal3\_pre.target\_pose.header.frame\_id = "map";

goal3\_pre.target\_pose.header.stamp = ros::Time::now();

goal3\_pre.target\_pose.pose.position.x = 2.28;//目标点x

goal3\_pre.target\_pose.pose.position.y = 2.120; //目标点y

goal3\_pre.target\_pose.pose.orientation.z = -0.032;

goal3\_pre.target\_pose.pose.orientation.w = 0.999;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal3;//定义第三个目标点 ，吉林馆

goal3.target\_pose.header.frame\_id = "map";

goal3.target\_pose.header.stamp = ros::Time::now();

goal3.target\_pose.pose.position.x = 2.48;//目标点x

goal3.target\_pose.pose.position.y = 2.120; //目标点y

goal3.target\_pose.pose.orientation.z = -0.032;

goal3.target\_pose.pose.orientation.w = 0.999;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal4\_pre;//定义第四个目标点 ，深圳馆

goal4\_pre.target\_pose.header.frame\_id = "map";

goal4\_pre.target\_pose.header.stamp = ros::Time::now();

goal4\_pre.target\_pose.pose.position.x =0.826;//目标点x

goal4\_pre.target\_pose.pose.position.y = 1.109; //目标点y

goal4\_pre.target\_pose.pose.orientation.z = -0.012;

goal4\_pre.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal4;//定义第四个目标点 ，深圳馆

goal4.target\_pose.header.frame\_id = "map";

goal4.target\_pose.header.stamp = ros::Time::now();

goal4.target\_pose.pose.position.x =1.026;//目标点x

goal4.target\_pose.pose.position.y = 1.109; //目标点y

goal4.target\_pose.pose.orientation.z = -0.012;

goal4.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal5\_pre;//定义第五个目标点 ，上海馆

goal5\_pre.target\_pose.header.frame\_id = "map";

goal5\_pre.target\_pose.header.stamp = ros::Time::now();

goal5\_pre.target\_pose.pose.position.x =0.873;//目标点x

goal5\_pre.target\_pose.pose.position.y = 2.120; //目标点y

goal5\_pre.target\_pose.pose.orientation.z =0.013;

goal5\_pre.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

move\_base\_msgs::MoveBaseGoal goal5;//定义第五个目标点 ，上海馆

goal5.target\_pose.header.frame\_id = "map";

goal5.target\_pose.header.stamp = ros::Time::now();

goal5.target\_pose.pose.position.x =1.073;//目标点x

goal5.target\_pose.pose.position.y = 2.120; //目标点y

goal5.target\_pose.pose.orientation.z =0.013;

goal5.target\_pose.pose.orientation.w = 1.000;//目标点姿态四元数表示

int Num = 2;//定义巡检的圈数

for(int i=0;i<Num;i++)

{

ac.sendGoal(goal1\_pre);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点1");

ac.sendGoal(goal1);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点1");

ac.sendGoal(goal2\_pre);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点1");

ac.sendGoal(goal2);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点2");

ac.sendGoal(goal3\_pre);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点1");

ac.sendGoal(goal3);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点3");

ac.sendGoal(goal4\_pre);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点1");

ac.sendGoal(goal4);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点4");

ac.sendGoal(goal5\_pre);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点1");

ac.sendGoal(goal5);

while(!(ac.getState()==actionlib::SimpleClientGoalState::SUCCEEDED))

{

usleep(1000\*20);

}

ROS\_INFO("机器人到达目标点5");

}

return 0;

}

重新编译代码：

重新运行验证；