重庆大学
CHONGQING UNIVERSITY

机器人操作系统
Robot Operating System

——第六章：机器人自主导航

江　涛、张博文、熊祖明

重庆大学　自动化学院

2023年5月29日

本章目标

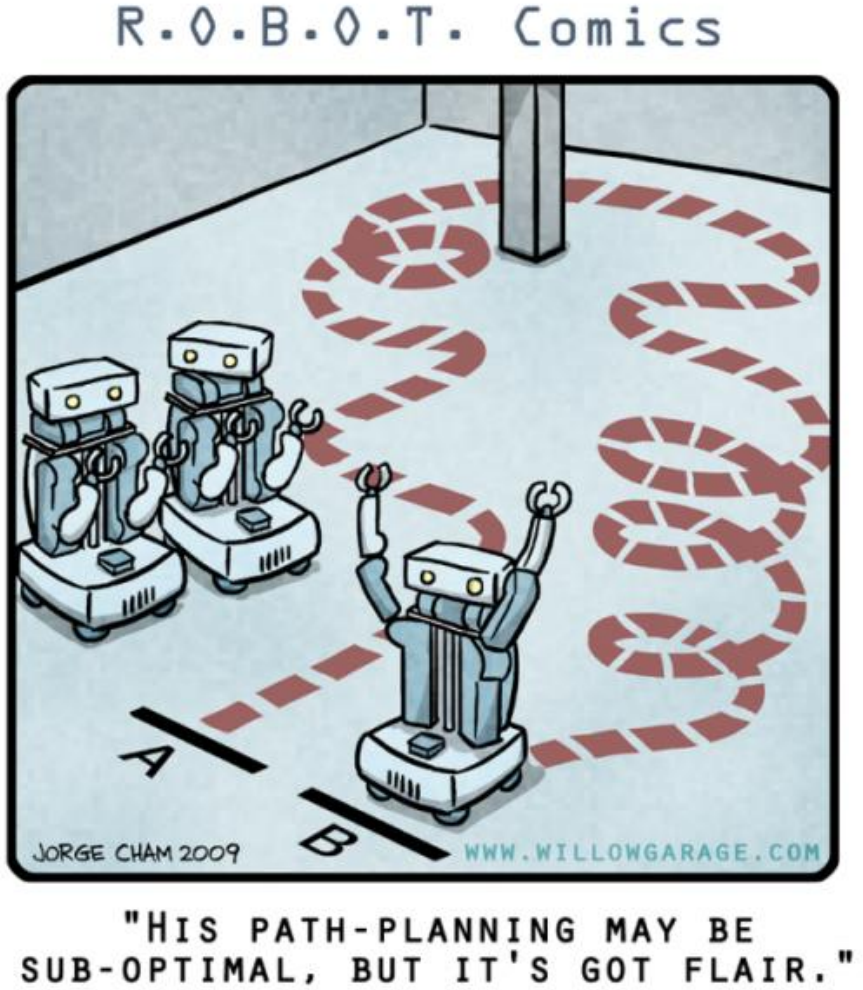
1、掌握机器人导航原理和框架，如：全局导航、局部导航、地图处理

2、掌握ros navigation工具包基本原理和使用方法，包括：启动方式、参数修改、调用流程

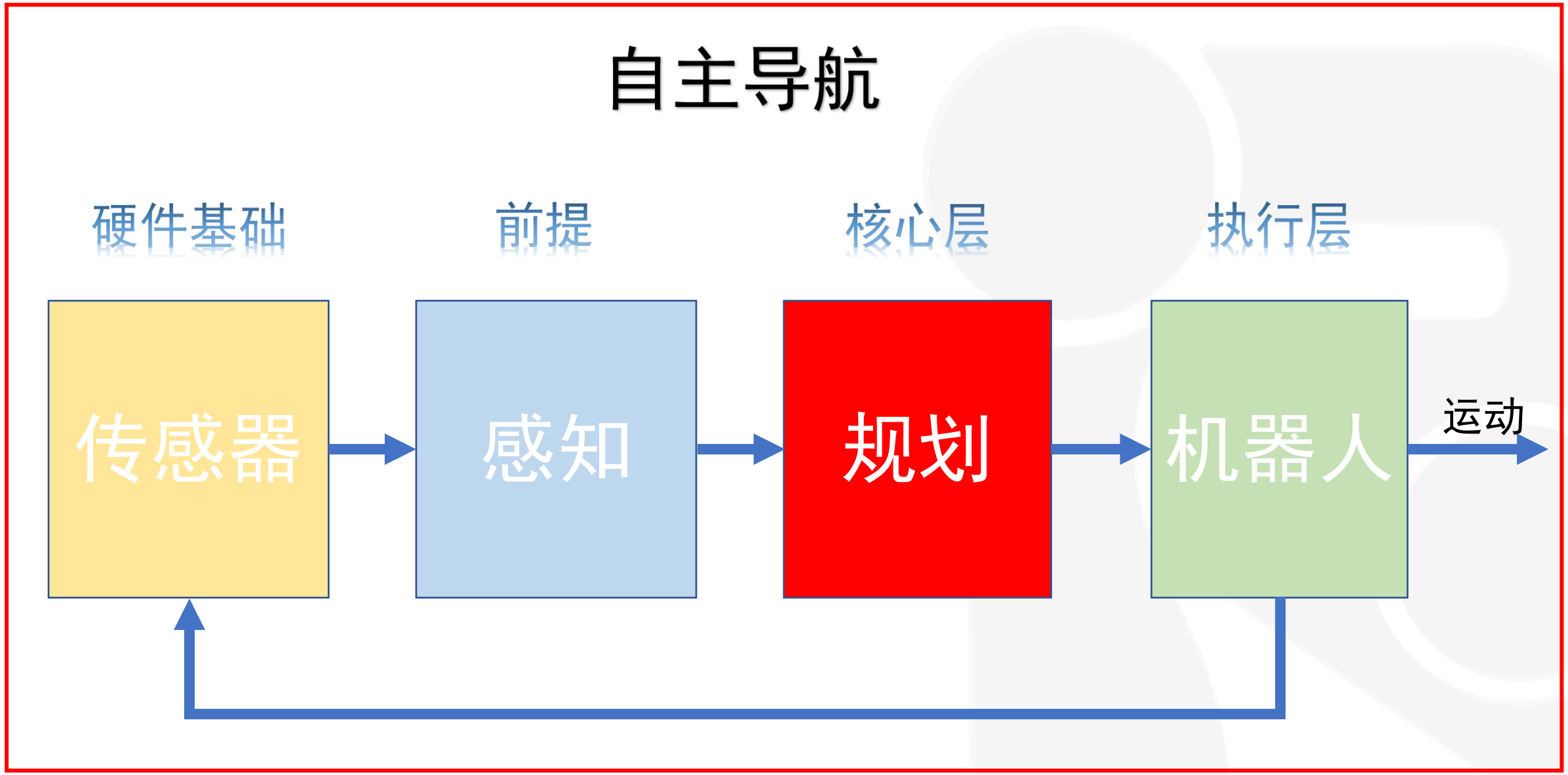
一、原理：定义

- **自主导航**：指机器人可以**自主规划**运动路径，并最终到达指定目标的能力

- 机器人自主从 A 点移动到 B 点的过程

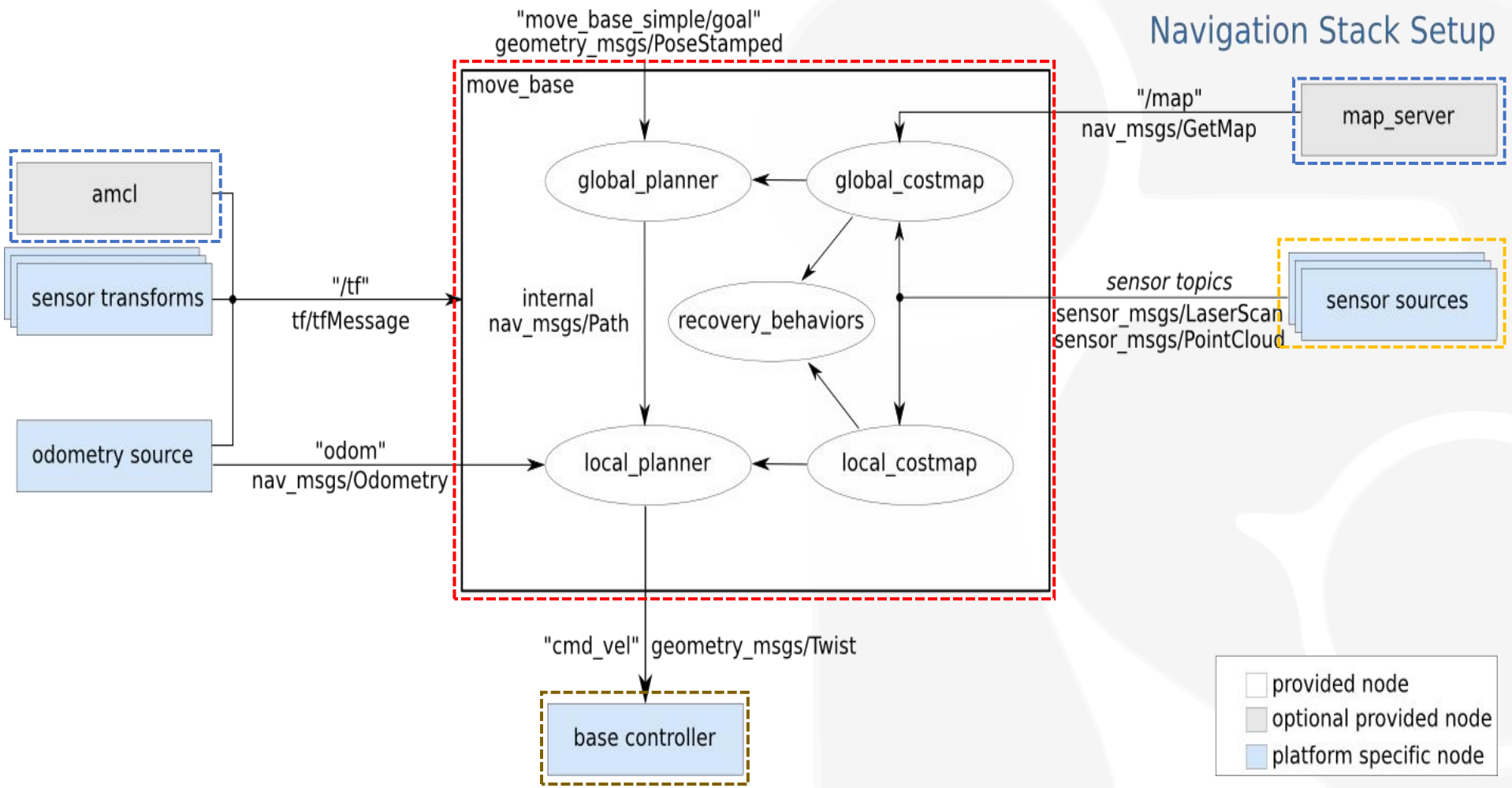


一、原理：定义



二、框架：结构

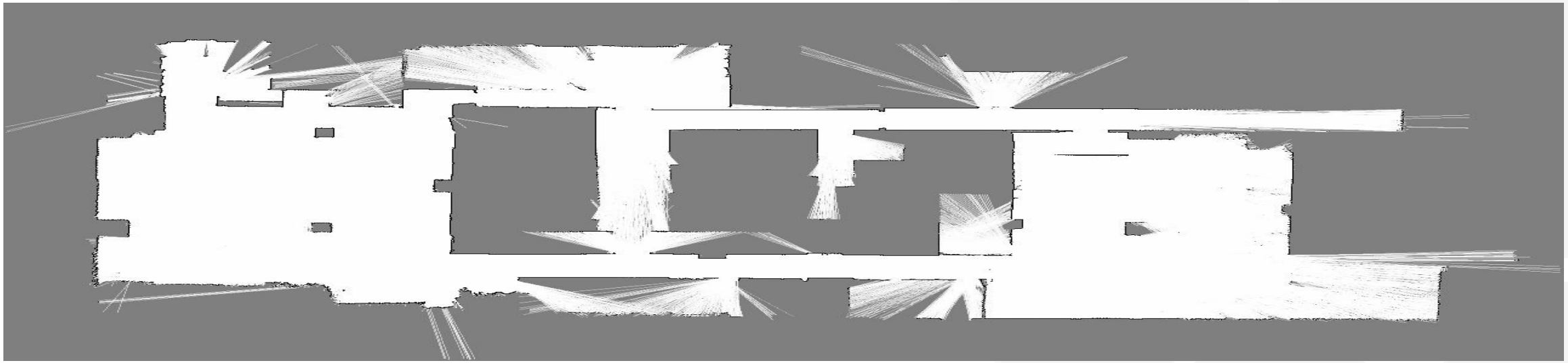
- 环境感知
- 地图
- 定位
- 路径规划
- 运动控制



二、框架：基本要素

➢ 地图

在导航中，根据地图来确定**自身的位置**、**目的地位置**，然后根据地图来**规划**的路线



➢ 自身定位

□ 导航开始和导航过程中，机器人需要确定当前**自身的位置**。

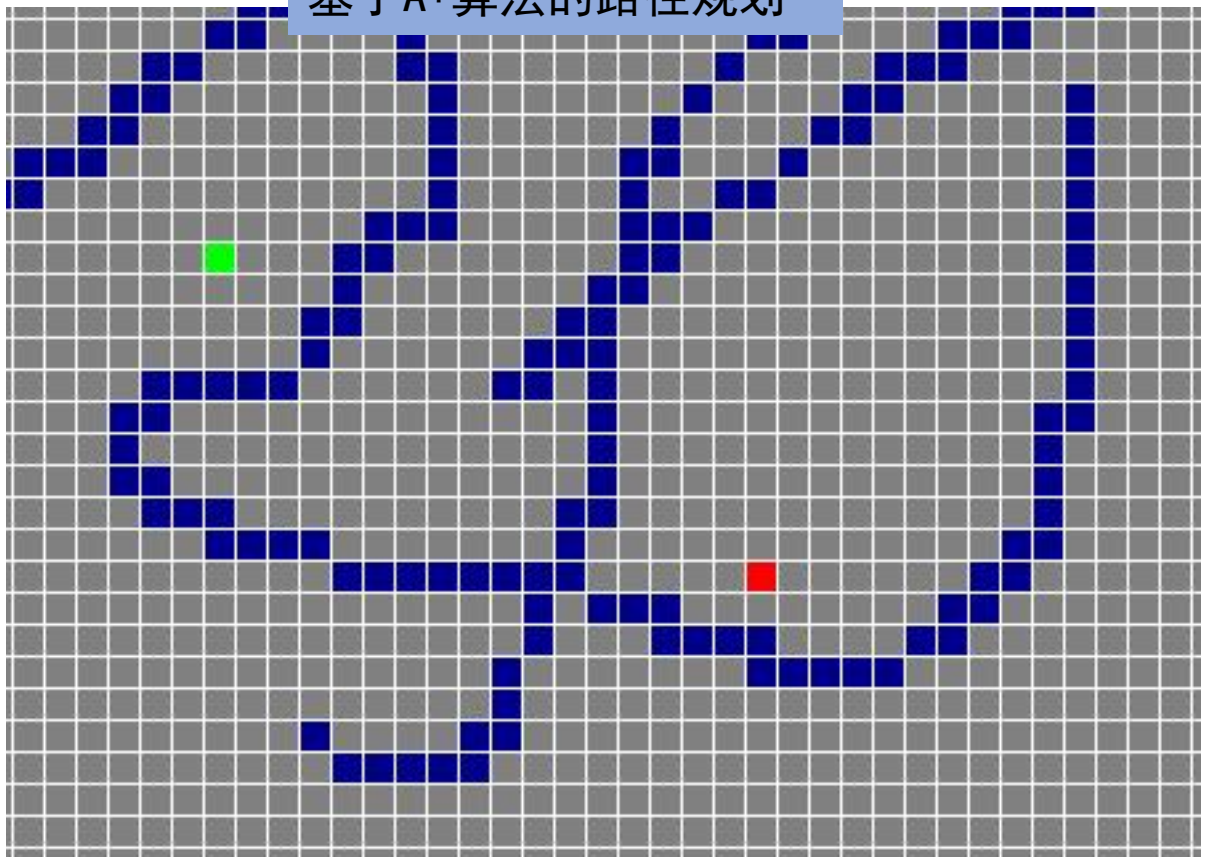
- 室外：可以选择GPS实现自身定位；
- 室内：可以通过 SLAM 实现

二、框架：基本要素

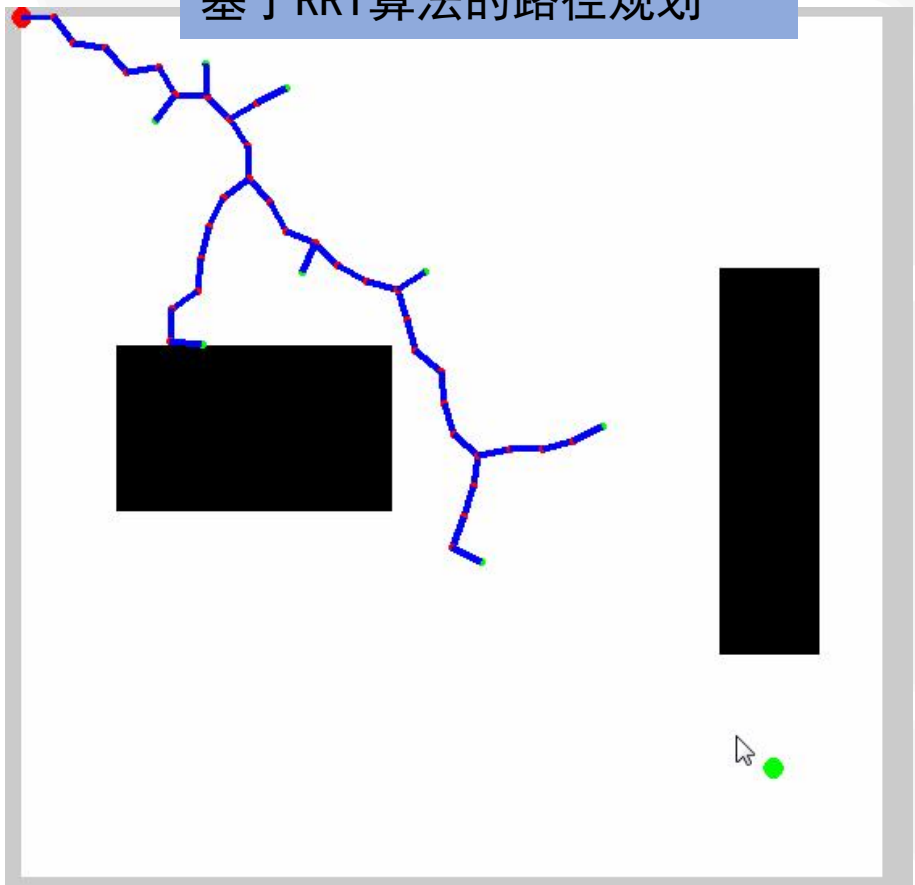
- **路径规划**：在**起点**和**终点**之间寻找一条**连续**的运动轨迹，在**目标代价最优/次优**的同时**避开**环境中的障碍物

  典型算法：Dijkstra算法、启发式搜索 A* 算法、RRT算法等

基于A*算法的路径规划



基于RRT算法的路径规划



二、框架：基本要素

➢ 运动控制

ROS中可以通过话题完成，例如“cmd_vel”发布geometry_msgs/Twist类型的消息，**订阅“cmd_vel”话题，**将该话题上的**速度命令**转换为**电机控制命令**

➢ 环境感知

- 感知周围**环境信息**和获取**自身信息**
- 摄像机（深度）、激光雷达感知外界环境的障碍物信息
- 编码器感知电机的转速信息，生成机器人里程信息

内部传感器
- 编码器
- 陀螺仪
- 加速度计
- 力/扭矩传感器等

外部传感器
- 激光雷达
- 相机
- 惯性测量单元
- GPS
- 毫米波雷达等

三、ros navigation：概述

navigation是ROS的二维导航Package

硬件要求：

（1）仅对**差动轮式机器人**有效，假设机器人可直接使用速度指令控制，速度指令的格式为：x方向速度、y方向速度、速度向量角速度。

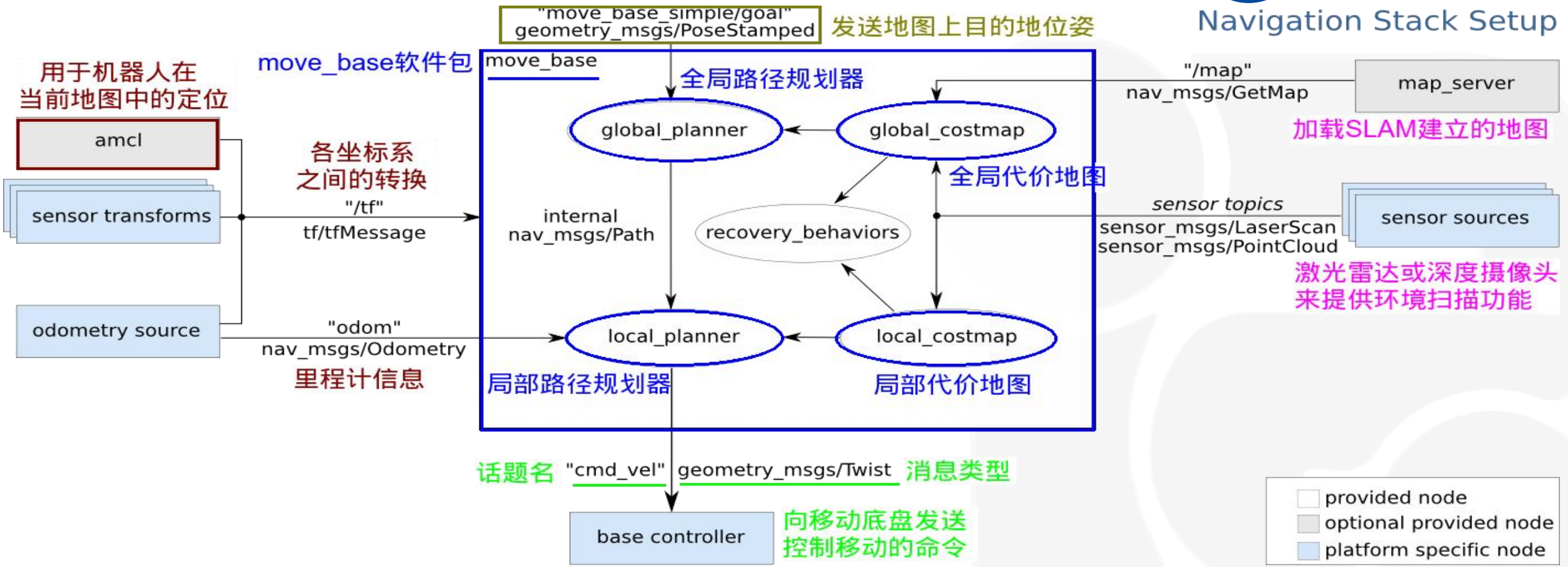
（2）要求机器人须安装有激光雷达等**二维平面测距设备**。

（3）**正方形或者圆形外形**的机器人支持度较好，其他外形的机器人表现不佳。

软件要求：

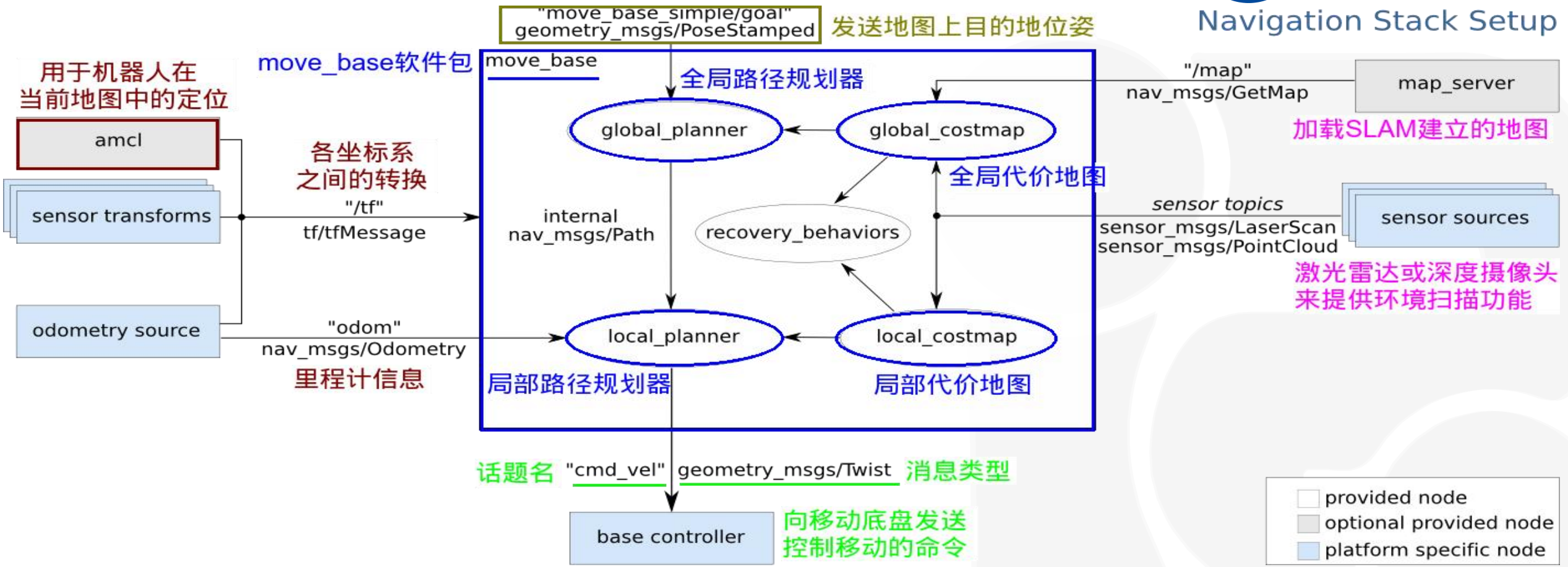
在ROS环境下，安装 navigation 功能包：>>> sudo apt install ros-<ROS版本>-navigation

三、ros navigation：模块介绍



- **global_costmap**：反映全局静态障碍物信息，一般从默认地图中加载，并根据sensor信息更新
- **local_costmap**：反映动态障碍物信息，通过**sensor topics获取**，提高算法实时性
- **sensor transforms**：说明传感器间的**坐标转换关系**

三、ros navigation：模块介绍



- **Odometry**：提供机器人**位姿信息**，包括机器人的速度、角度等，提供给局部规划器来规划路径
- **cmd_vel**：下发给控制模块，控制机器人的运动

三、ros navigation：话题

导航相关话题与解释

	话题	解释
地图相关	nav_msgs/MapMetaData	地图属性：宽度、高度、分辨率等
	nav_msgs/OccupancyGrid	栅格地图
里程计	nav_msgs/Odometry	里程计位姿、速度
坐标变换	tf/tfMessage	坐标系相对关系
定位	geometry_msgs/PoseArray	机器人在地图中的位姿估计集合
目标点	move_base_msgs/MoveBaseActionGoal	目标点位姿
路径规划	nav_msgs/Path	一些列目标点轨迹
传感器相关话题	sensor_msgs/Image	2D图像数据
	sensor_msgs/PointCloud2	点云图像数据（带有深度信息）
	sensor_msgs/LaserScan	激光雷达数据

三、ros navigation：amcl 定位

amcl

- AMCL(adaptive Monte Carlo Localization) 是基于自适应（或KLD采样）蒙特卡洛的2D移动机器人的**概率定位系统**，根据已有地图使用粒子滤波推算机器人位置
- amcl已经被集成到了navigation包。运行 amcl 节点前，需要先加载全局地图

```
<launch>
    <!-- 设置地图的配置文件 -->
    <arg name="map" default="slam.yaml" />
    <!-- 运行地图服务器，并且加载设置的地图-->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find demo02_gazebo)/map/$(arg map)"/>
    <!-- 启动AMCL节点 -->
    <include file="$(find demo02_gazebo)/launch/amcl.launch" />
    <!-- 运行rviz -->
    <node pkg="rviz" type="rviz" name="rviz" />
</launch>
```

三、ros navigation：amcl 定位

运行

1. 先启动 Gazebo 仿真环境

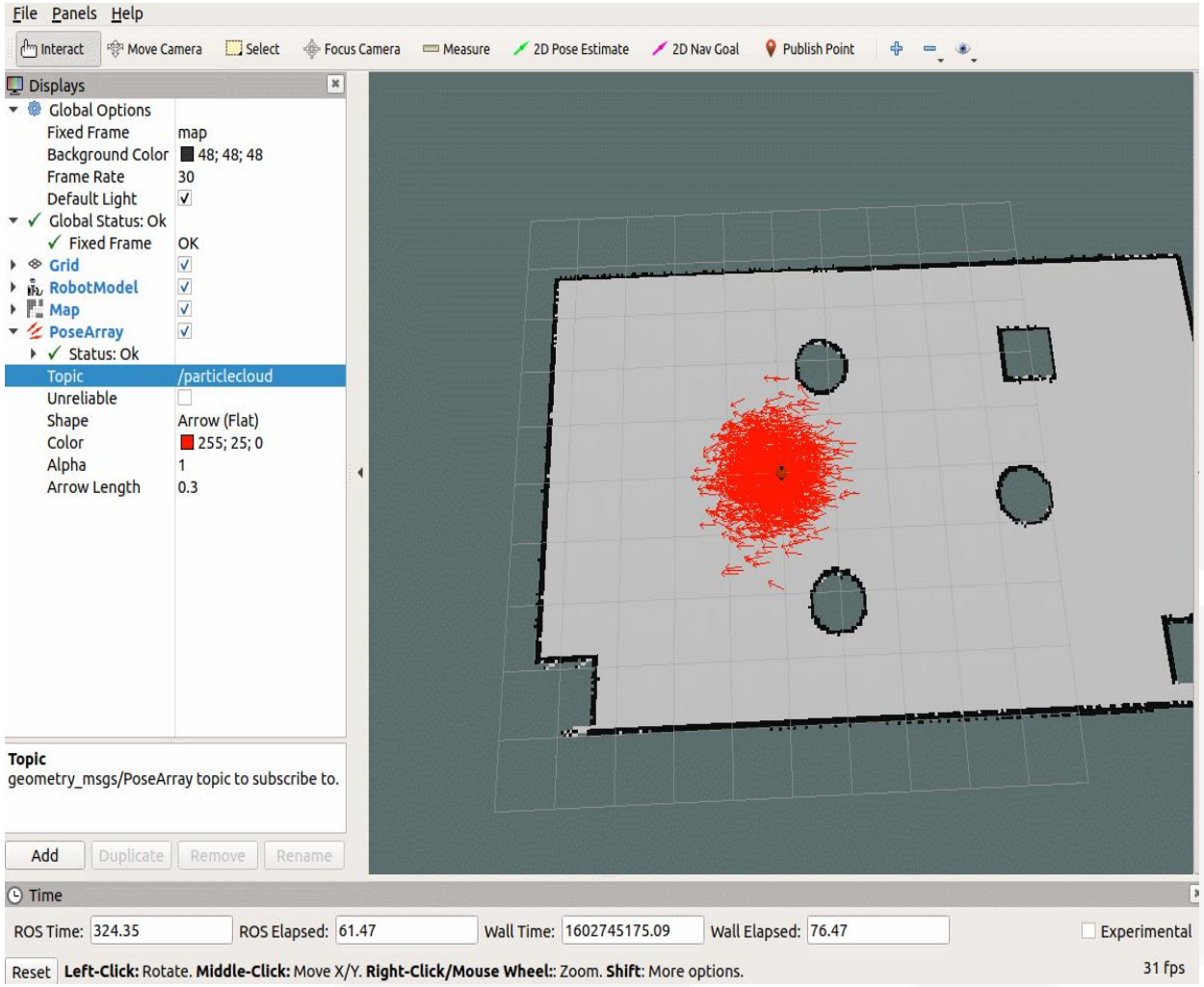
```
>>> roslaunch demo01_urdf_gazebo display_xacro_gazebo_sensor_all.launch
```

2. 启动键盘控制节点

```
>>> rosrun demo01_urdf_gazebo teleop_twist_keyboard
```

3. 启动集成了地图服务、amcl 与 rviz 的 launch 文件

```
>>> roslaunch demo05_navigation nav_amcl.launch
```



三、ros navigation：move_base

move_base=global planner+local planner+costmap

？？因为不同类型机器人大小尺寸、传感器、速度、应用场景不同....最后可能会导致不同的路径规划结果

那么在调用路径规划节点之前，需要配置机器人参数。

具体实现流程如下：

- 1. 编写move_base调用.launch
- 2. 编写move_base配置.yaml
- 3. 集成导航相关的文件.launch
- 4. 运行测试

1. move_base 节点调用 launch 文件模板

```
<launch>

    <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen" clear_params="true">
        <rosparam file="$(find demo02_gazebo)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />
        <rosparam file="$(find demo02_gazebo)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />
        <rosparam file="$(find demo02_gazebo)/param/local_costmap_params.yaml" command="load" />
        <rosparam file="$(find demo02_gazebo)/param/global_costmap_params.yaml" command="load" />
        <rosparam file="$(find demo02_gazebo)/param/base_local_planner_params.yaml" command="load" />
    </node>

</launch>
```

三、ros navigation：路径规划

2. **编写配置文件**：功能包下新建 param 目录，新建并配置以下文件：costmap_common_params.yaml、local_costmap_params.yaml、global_costmap_params.yaml、base_local_planner_params.yaml

base_local_planner_params.yaml

```yaml
TrajectoryPlannerROS:

# Robot Configuration Parameters
  max_vel_x: 0.5 # X 方向最大速度
  min_vel_x: 0.1 # X 方向最小速速

  max_vel_theta:  1.0 #
  min_vel_theta: -1.0
  min_in_place_vel_theta: 1.0

  acc_lim_x: 1.0 # X 加速限制
  acc_lim_y: 0.0 # Y 加速限制
  acc_lim_theta: 0.6 # 角速度加速限制

# Goal Tolerance Parameters，目标公差
  xy_goal_tolerance: 0.10
  yaw_goal_tolerance: 0.05

# Differential-drive robot configuration
# 是否是全向移动机器人
  holonomic_robot: false

# Forward Simulation Parameters，前进模拟参数
  sim_time: 0.8
  vx_samples: 18
  vtheta_samples: 20
  sim_granularity: 0.05
```

costmap_common_params.yaml

```yaml
#机器人几何参，如果机器人是圆形，设置 robot_radius,如果是其他形状设置 footprint
robot_radius: 0.12 #圆形
# footprint: [[-0.12, -0.12], [-0.12, 0.12], [0.12, 0.12], [0.12, -0.12]] #其他形状

obstacle_range: 3.0 # 用于障碍物探测，比如：值为 3.0，意味着检测到距离小于 3 米的障碍物时，就会引入代价地图
raytrace_range: 3.5 # 用于清除障碍物，比如：值为 3.5，意味着清除代价地图中 3.5 米以外的障碍物


#膨胀半径，扩展在碰撞区域以外的代价区域，使得机器人规划路径避开障碍物
inflation_radius: 0.2
#代价比例系数，越大则代价值越小
cost_scaling_factor: 3.0

#地图类型
map_type: costmap
#导航包所需要的传感器
observation_sources: scan
#对传感器的坐标系和数据进行配置。这个也会用于代价地图添加和清除障碍物。例如，你可以用激光雷达传感器用于在代价地图添加障碍物，再添加kinect用于导航和清除障碍物。
scan: {sensor_frame: laser, data_type: LaserScan, topic: scan, marking: true, clearing: true}
```

global_costmap_params.yaml

```yaml
global_costmap:
  global_frame: map #地图坐标系
  robot_base_frame: base_footprint #机器人坐标系
  # 以此实现坐标变换

  update_frequency: 1.0 #代价地图更新频率
  publish_frequency: 1.0 #代价地图的发布频率
  transform_tolerance: 0.5 #等待坐标变换发布信息的超时时间

  static_map: true # 是否使用一个地图或者地图服务器来初始化全局代价地图，
```

local_costmap_params.yaml

```yaml
local_costmap:
  global_frame: map #里程计坐标系
  robot_base_frame: base_footprint #机器人坐标系

  update_frequency: 10.0 #代价地图更新频率
  publish_frequency: 10.0 #代价地图的发布频率
  transform_tolerance: 0.5 #等待坐标变换发布信息的超时时间

  static_map: false  #不需要静态地图，可以提升导航效果
  rolling_window: true #是否使用动态窗口，默认为false，在静态的全局地图中，地图不会变化
  width: 3 # 局部地图宽度 单位是 m
  height: 3 # 局部地图高度 单位是 m
  resolution: 0.05 # 局部地图分辨率 单位是 m，一般与静态地图分辨率保持一致
```

三、ros navigation：路径规划

3. **集成launch 文件**：将地图服务、amcl 、move_base 与 Rviz 在一个launch文件集成*(已知环境导航)*

```
<launch>
    <!-- 设置地图的配置文件 -->
    <arg name="map" default="slam.yaml" />
    <!-- 运行地图服务器，并且加载设置的地图-->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find demo02_gazebo)/map/$(arg map)"/>
    <!-- 启动AMCL节点 -->
    <include file="$(find demo02_gazebo)/launch/amcl.launch" />

    <!-- 运行move_base节点 -->
    <include file="$(find demo02_gazebo)/launch/move_base.launch" />
    <!-- 运行rviz -->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find demo02_gazebo)/config/nav.rviz" />
</launch>
```

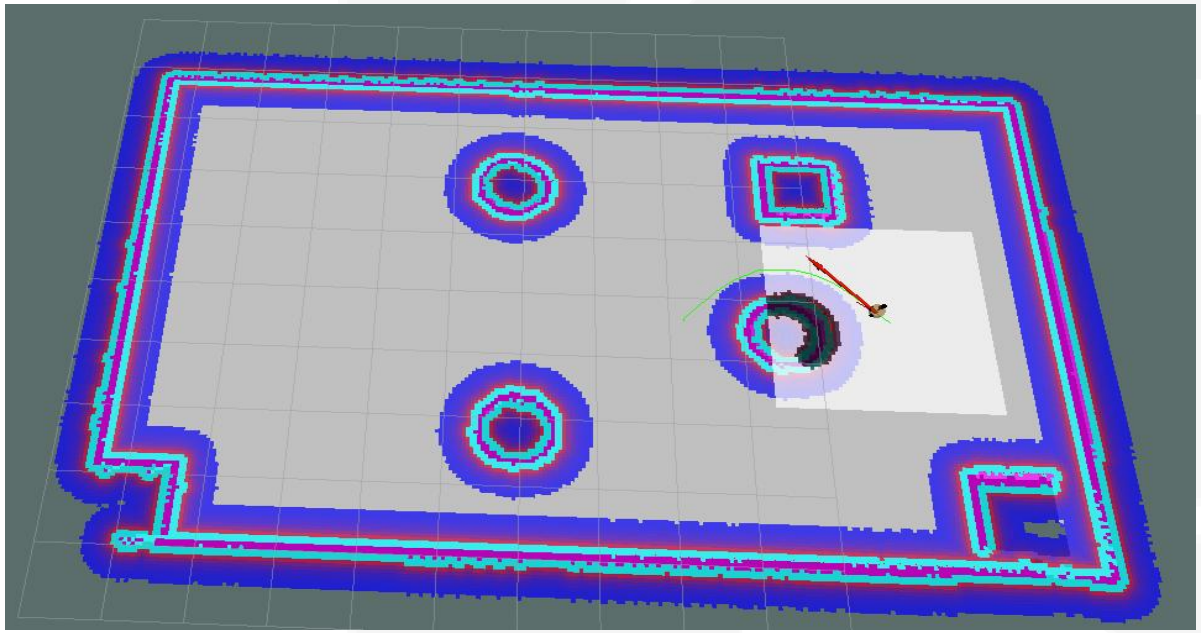
4. **运行测试**

1. 先启动 Gazebo 仿真环境；

>>> roslaunch demo01_urdf_gazebo display_xacro_gazebo_sensor_all.launch

2. 启动导航相关的 launch 文件；

>>> roslaunch demo05_navigation movebase_start_with_map.launch



三、ros navigation：SLAM建图与导航

通过gmapping实时建图与定位，结合move_base功能模块实现(未知环境)自主导航

步骤如下:

1. 编写launch文件，集成SLAM与move_base相关节点
2. 执行launch文件并测试

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

三、ros navigation：SLAM建图与导航

重庆大学 CHONGQING UNIVERSITY

运行测试：

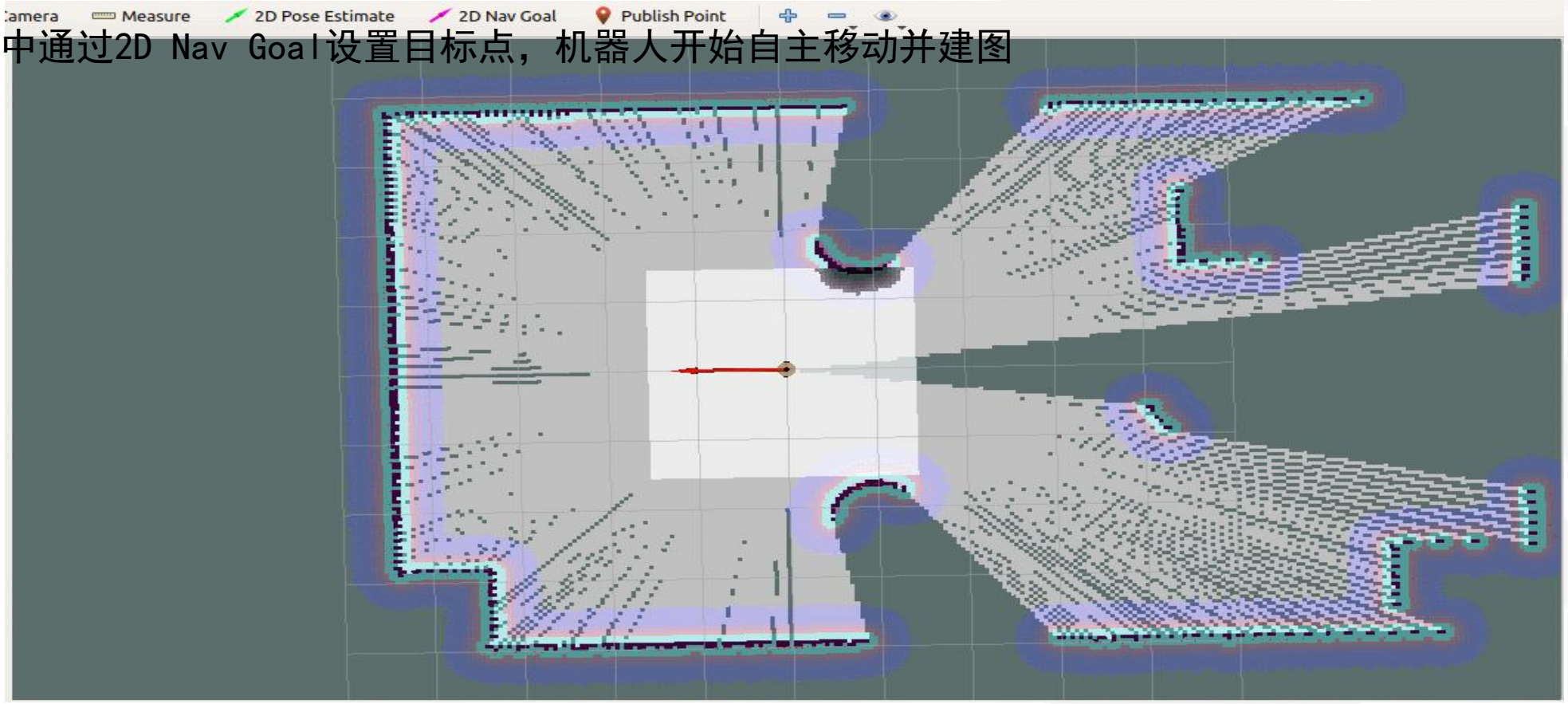
1. 运行gazebo仿真环境

```
>>> roslaunch demo01_urdf_gazebo
display_xacro_gazebo_sensor_all.launch
```

2. 然后执行launch文件

```
>>> roslaunch demo05_navigation
movebase_start_no_map.launch
```

3. 在rviz中通过2D Nav Goal设置目标点，机器人开始自主移动并建图



感谢聆听！