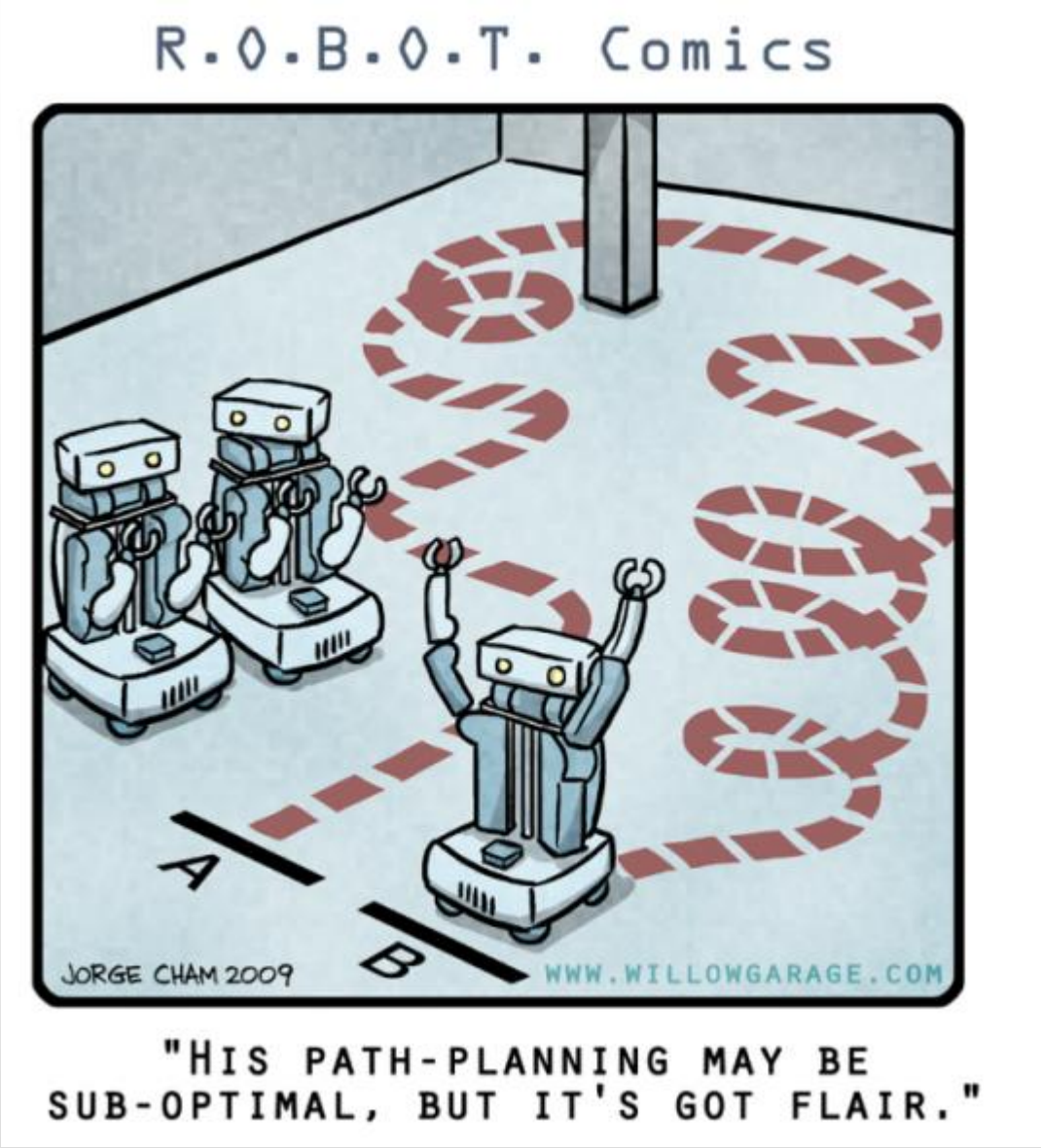
03 导航实例

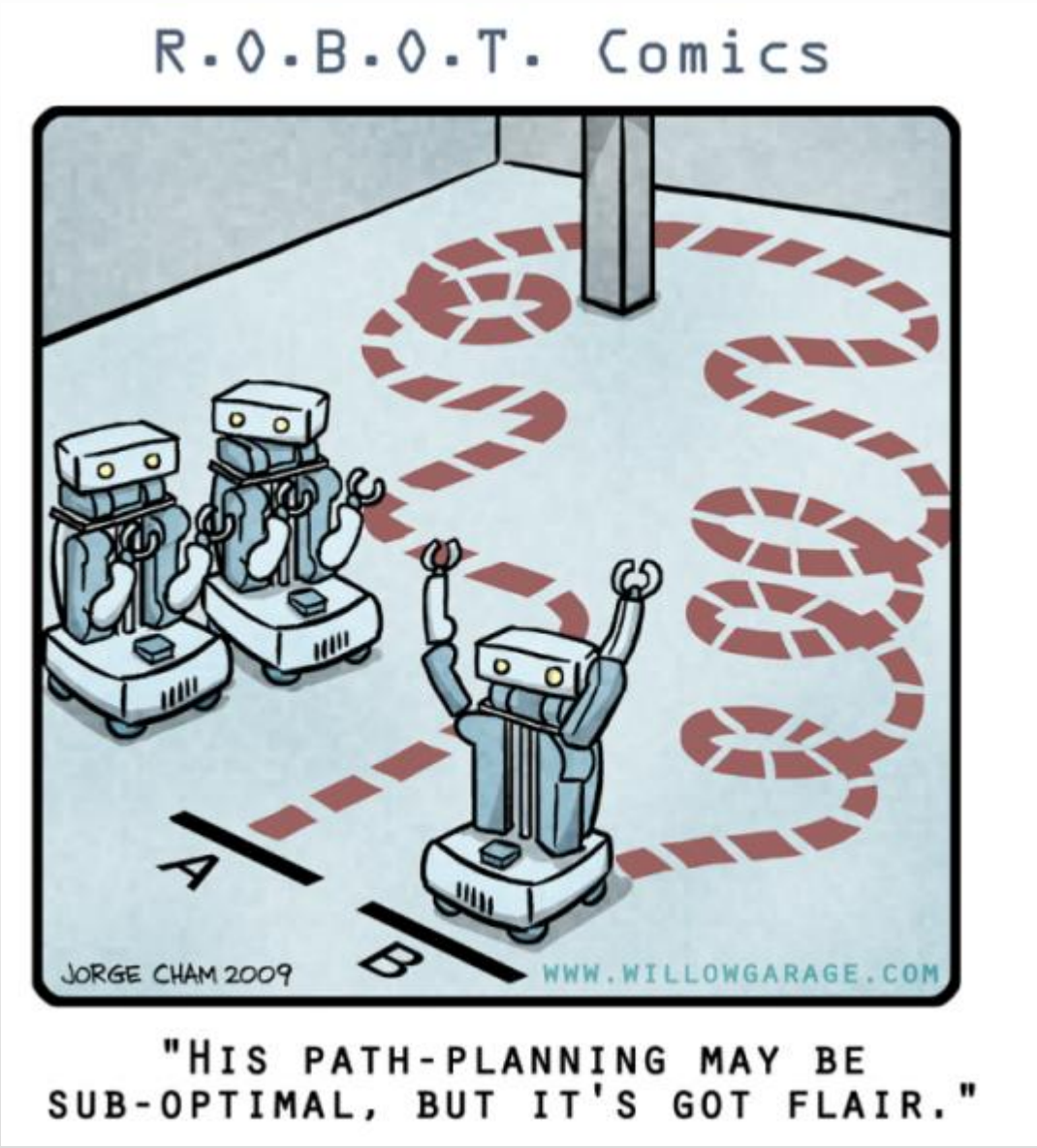
自主导航

从表面上看，自主导航就是解决从地点A到地点B的问题，但是实现起来非常复杂。自主导航是一个非常大的课题，解决方案也是五花八门，而且各方案之间没有明显的理论界限。目前最流行的就是SLAM导航方案，而SLAM导航方案由建图、定位和路径规划三大基本问题组成，这三大问题互相嵌套又组成新的问题，也就是SLAM问题、导航问题、探索问题等。

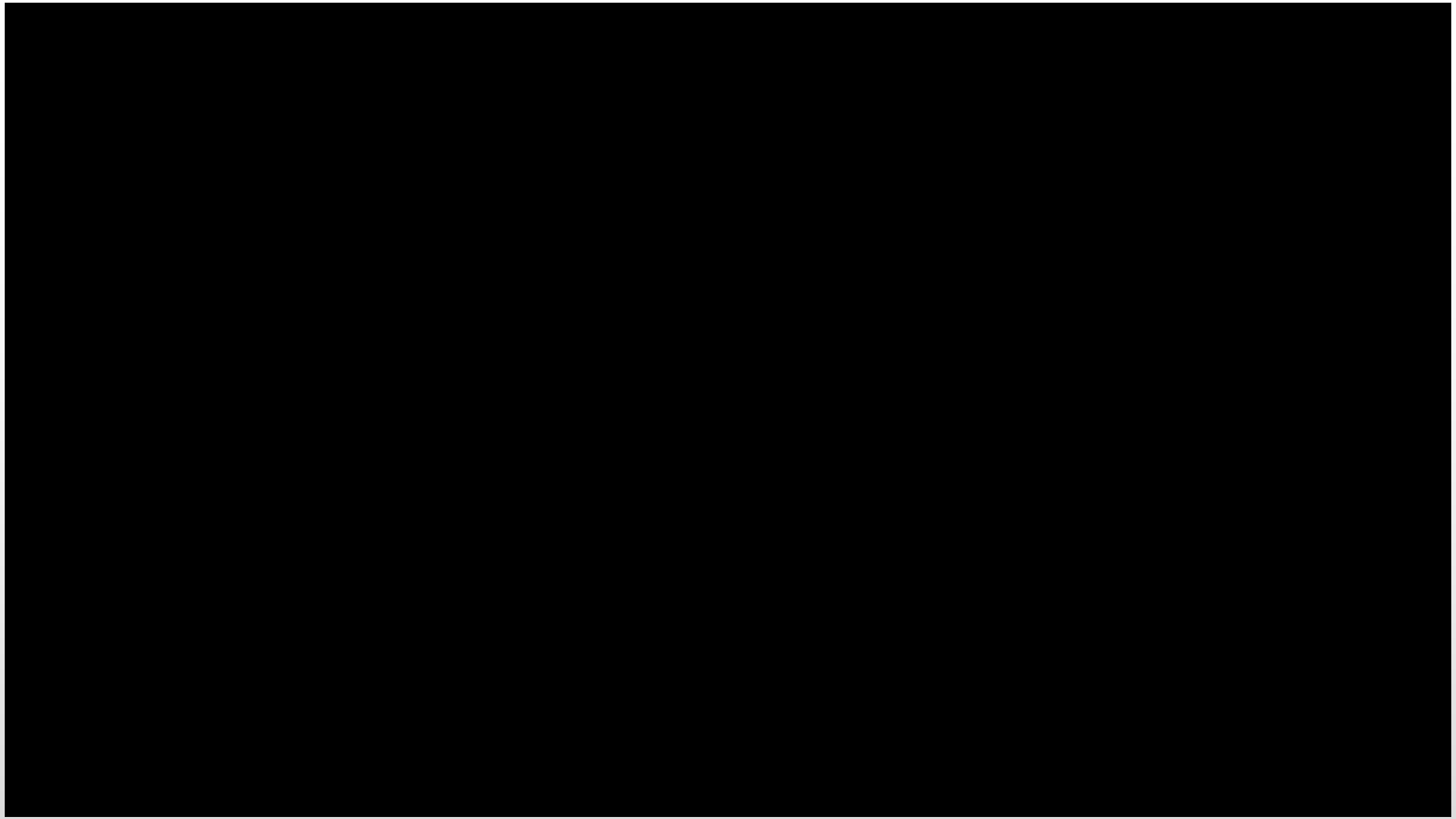


自主导航

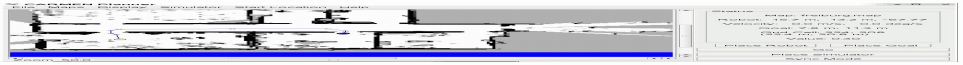
- 自主导航问题的本质我们可以从右图来描述，就是从地点A移动到地点B的问题。当想机器人下达移动到地点B的命令后，机器人这是就会提出三个问题，“我在哪”、“我将到哪”、“我如何去”。
- 目前自主导航主要针对的是机器人、无人机、无人驾驶汽车等无人操控的对象。



Navigation



Navigation

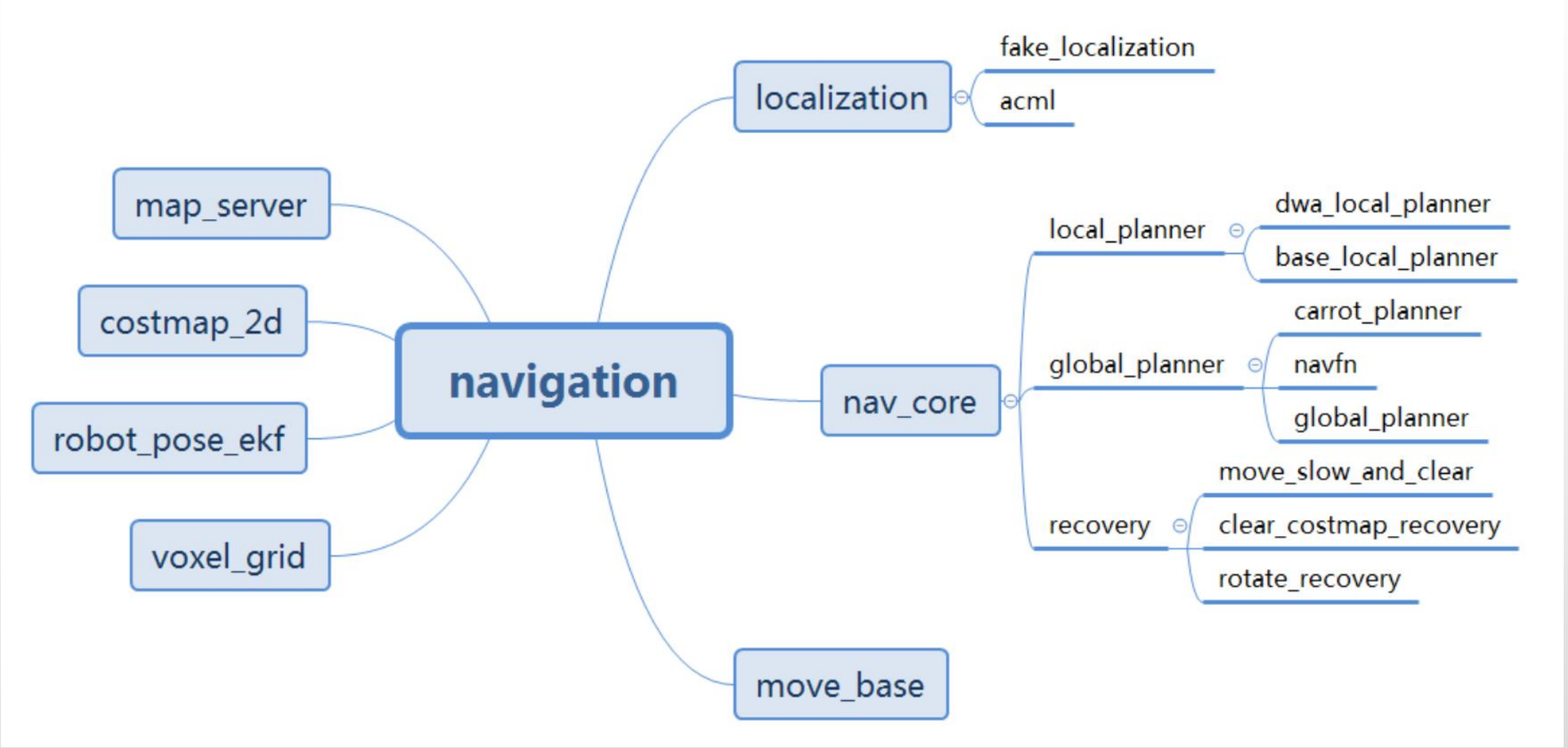


导航（Navigation）是指在地图中，已知机器人当前坐标，当在地图中给定合理的目标点时，能自动规划出一条合理的路径，并控制机器人沿着该路径行走并能自动避开静态和动态的障碍物。实现这样的功能需要有以下几个要点：

- 一张完整的高精度地图
- 生成相应的代价地图
- 机器人在地图中的起始位置，并能在移动中定位其位置
- 在地图上给定一个可到达的目标点位姿
- 全局路径规划
- 局部路径规划

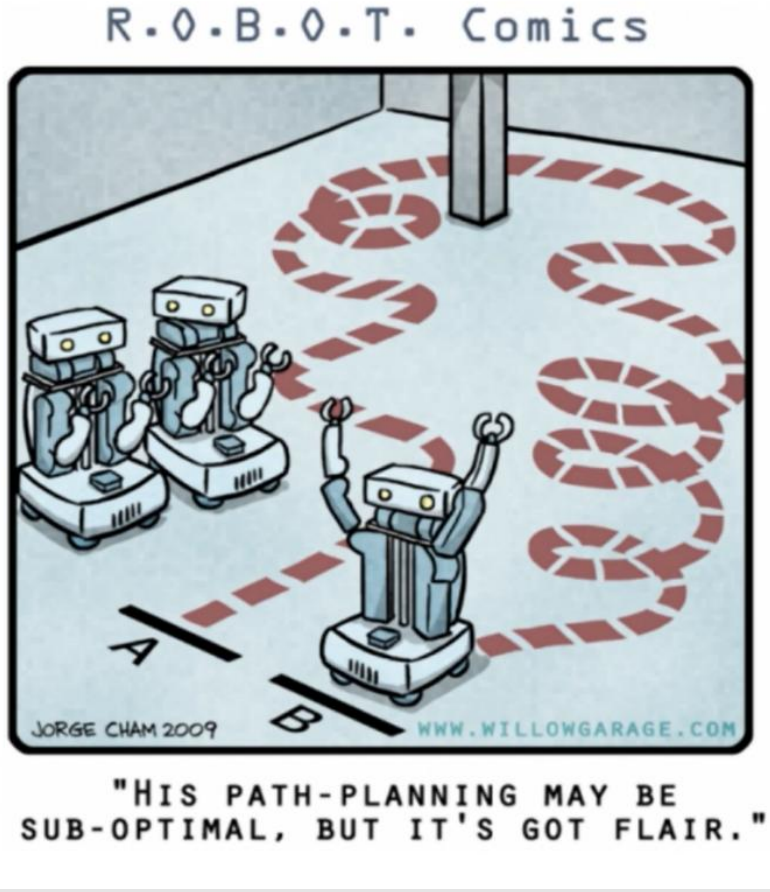
ROS Navigation

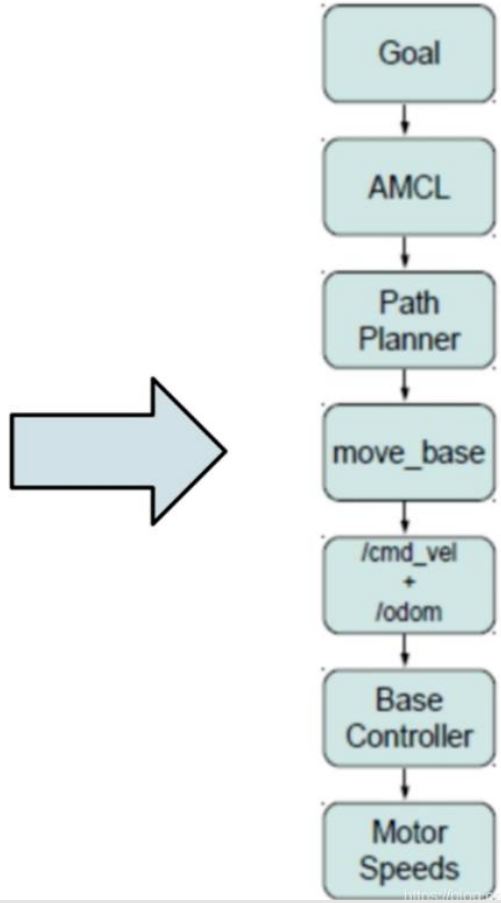
Navigation是ROS的二维导航功能包，简单来说，就是根据输入的里程计等传感器的信息流和机器人的全局位置，通过导航算法，计算得出安全可靠的机器人速度控制指令。在ROS中的代码包如下图



ROS Navigation

首先要知道导航中的目标点（Goal），然后 AMCL 会获取当前坐标点，Path Planner即为路径规划，Move_base 主要解决的是本地优化（Path Planner相当于全局规划，move_base相当于本地规划），在ROS中可以采用Move_base功能包，该功能包包含有全局规划与本地规划。然后Move_base会发布一个速度指令给底层驱动器，完成导航功能。

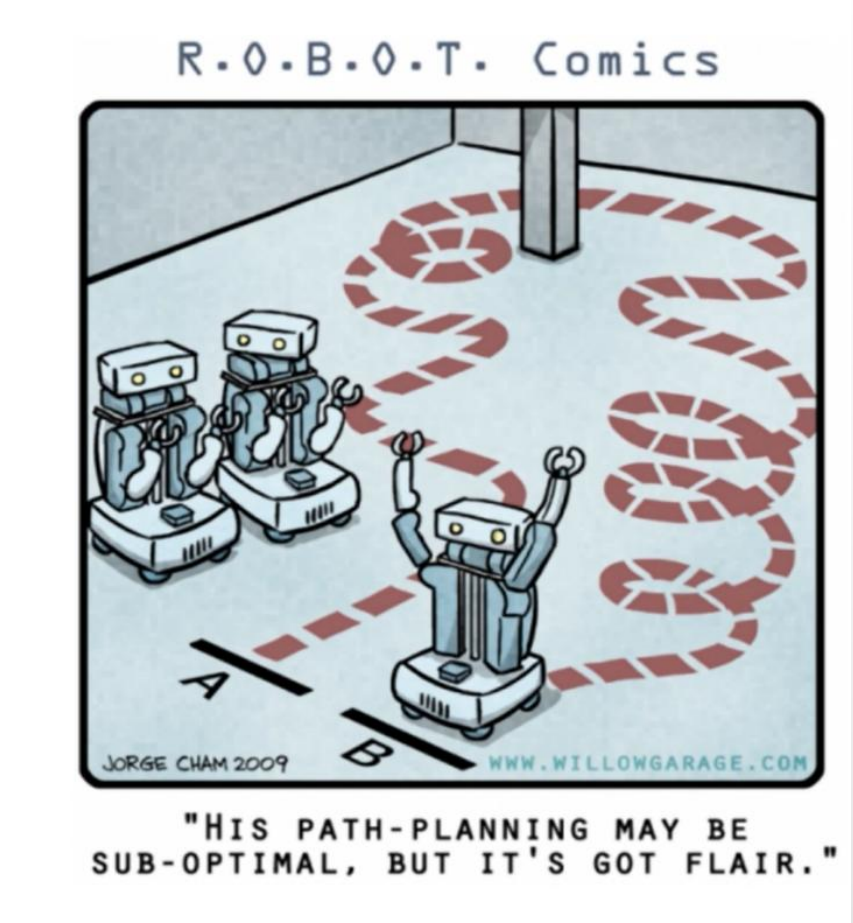




ROS Navigation

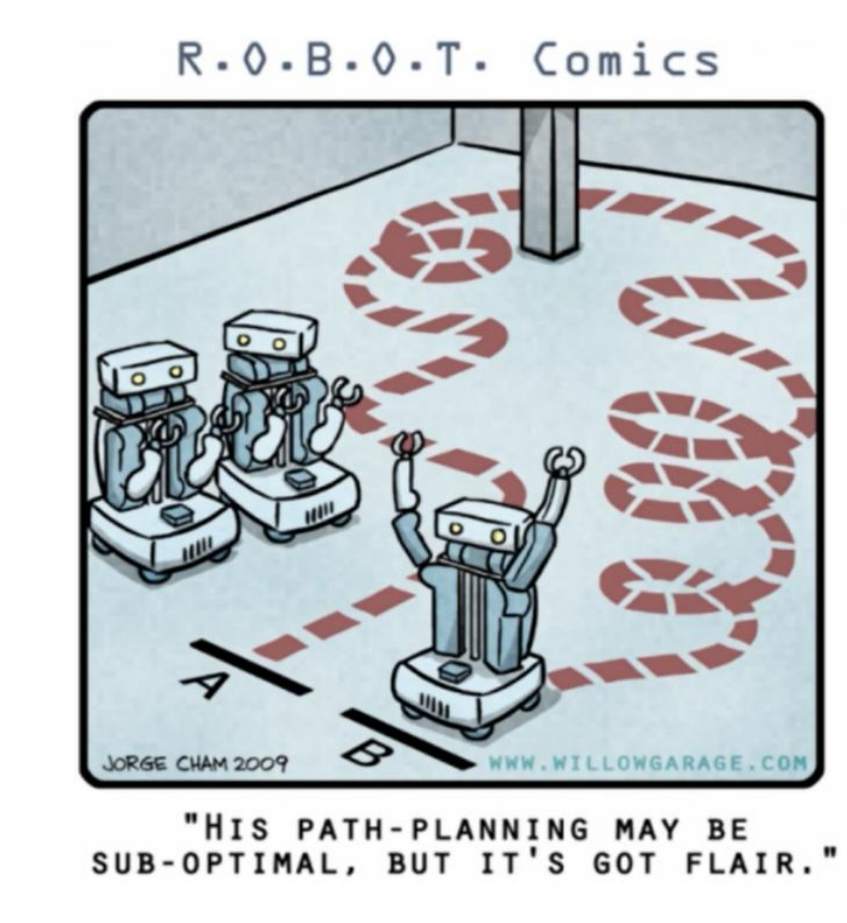
机器人的导航模块（Navigation Stack）是用来解决什么问题的呢?

顾名思义，它解决的自然是在一个已知的、动态的环境里，实现让机器人本体从A地到B地的自主运动。在这个过程中，实现对物理世界的建模、路径规划、避障、移动等等各种功能。



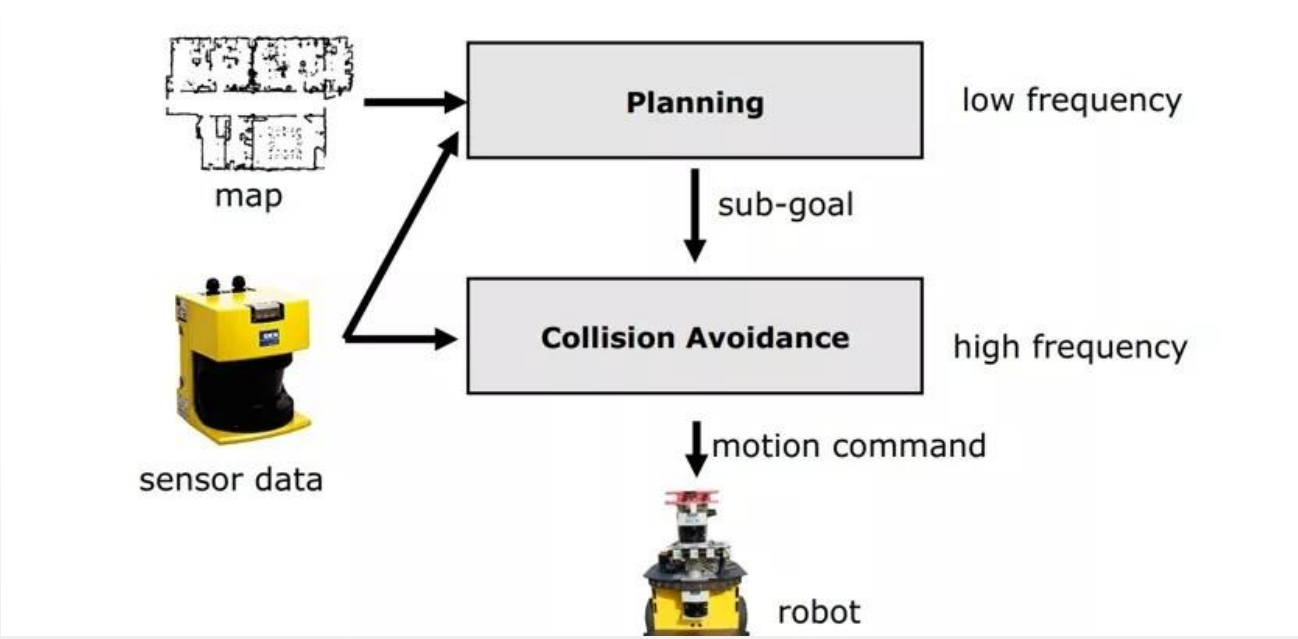
ROS Navigation

导航就是解决一个有先决条件、有动力学限制的几何问题的一系列过程！



ROS Navigation

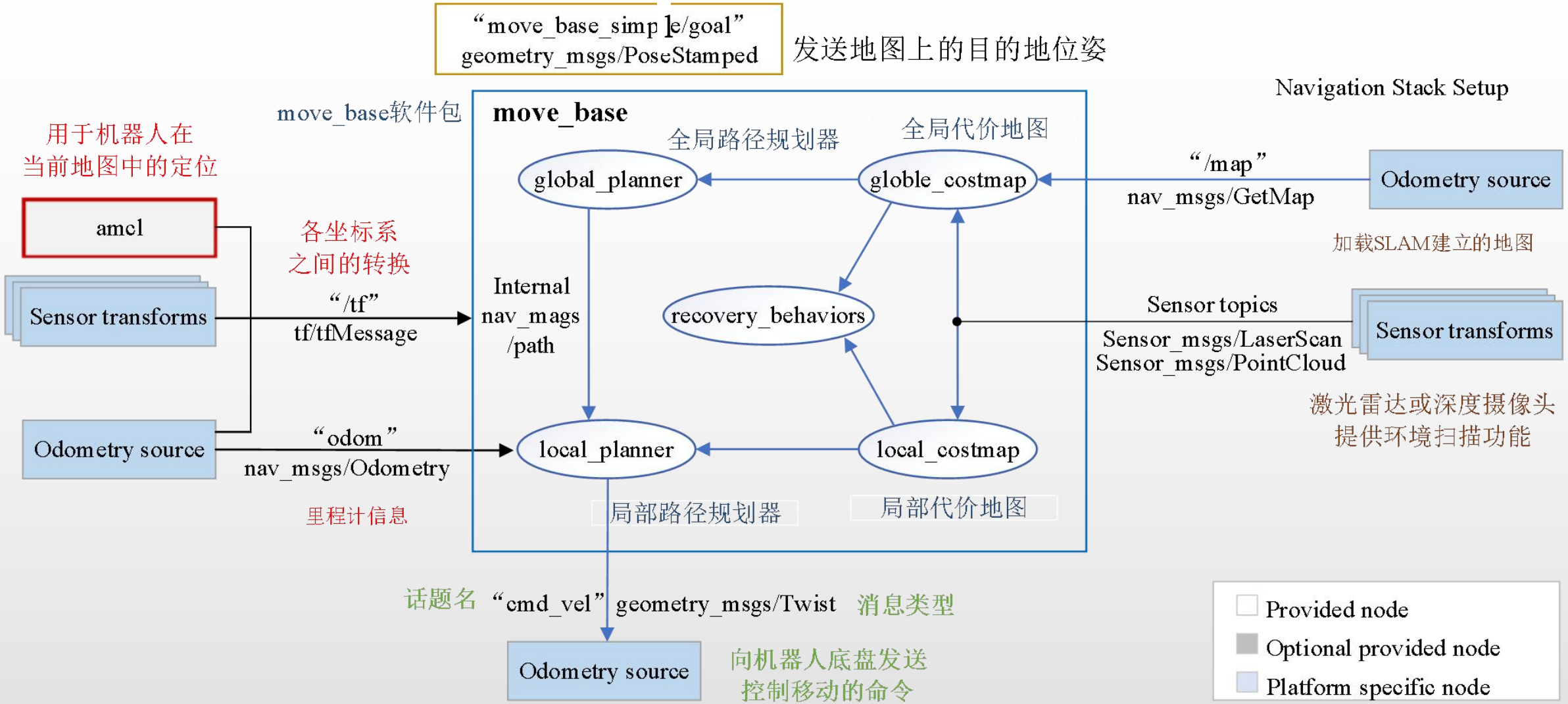
对于一个特定的导航需求，首先有传感技术与SLAM模块共同决定一个世界模型，然后通过路径规划算法获得加权之后的最优路径，再次导入机器人模型做避障规划，最后实现对机器人硬件的移动控制。



ROS Navigation

- **map_server：**地图服务器，主要功能是保存地图和导入地图。
- **costmap_2d：**可以生产代价地图，以及提供各种相关的函数。
- **robot_pose_ekf：**扩展卡尔曼滤波器，输入是里程计、IMU、VO中的任意两个或者三个，输出是一个融合之后的pose。
- **localization：**这里是两个定位用的package。fake_localization一般是仿真用的，amcl才是实际定位用的package。
- **nav_core：**这里面只有三个文件，对应的是全局路径规划、局部路径规划、recovery_action的通用接口定义，具体功能实现则是在各个对应的规划器package里。
- **move_base：**这里实现的是整个导航的流程。什么时候调用全局路径规划、什么时候调用局部路径规划、什么时候调用recovery_action都是这个package管的。就是下图中间方框里做的事情，可以说是整个navigation stack的核心。

ROS Navigation



基于move_base的导航框架

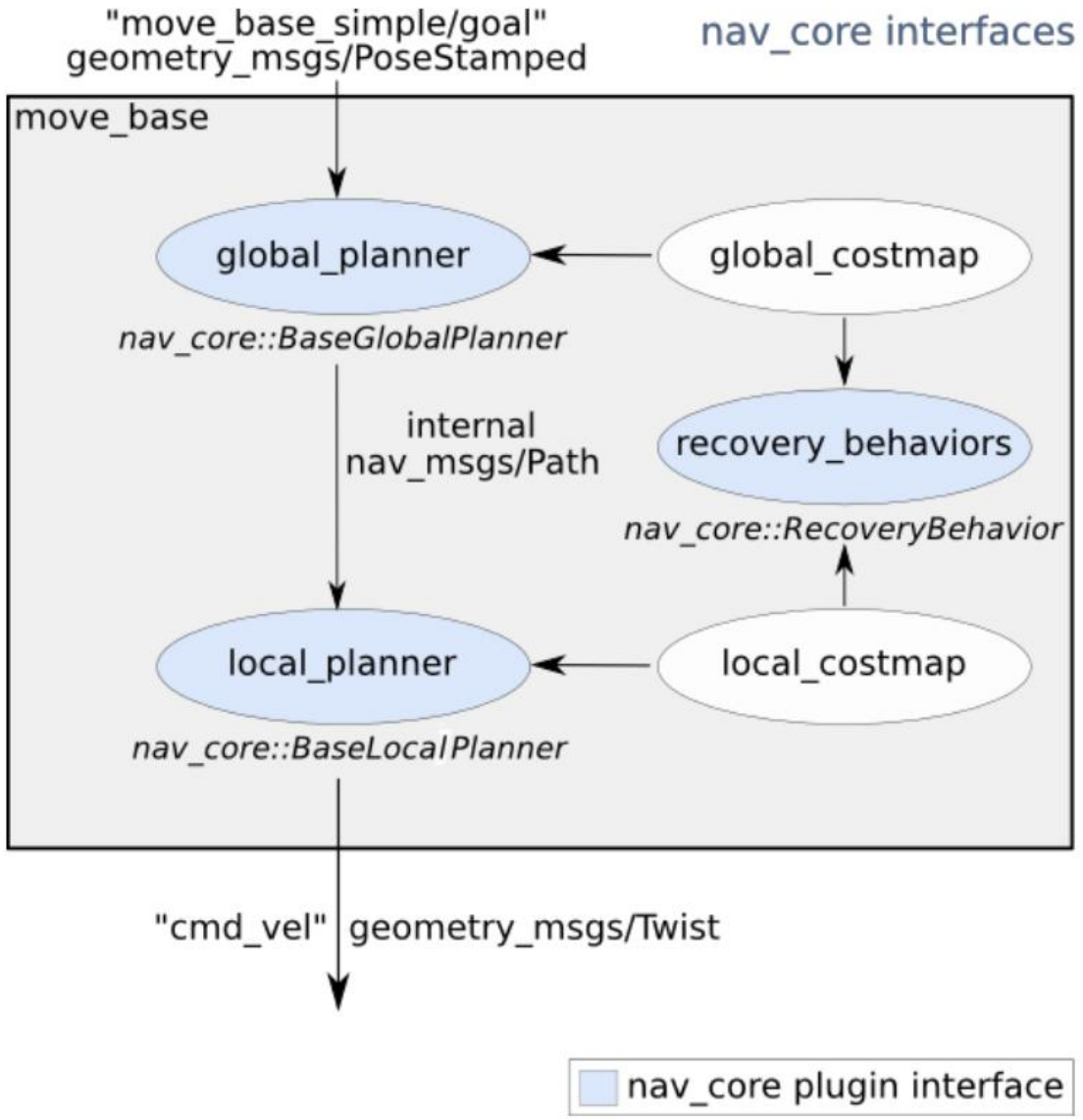
ROS Navigation——move_base

➢ 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经，综合各评价标准选取最优路径



ROS Navigation——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

ROS Navigation——move_base

```
<launch>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen" clear_params="true">
    <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/base_local_planner_params.yaml" command="load" />
  </node>

</launch>
```

配置move_base节点

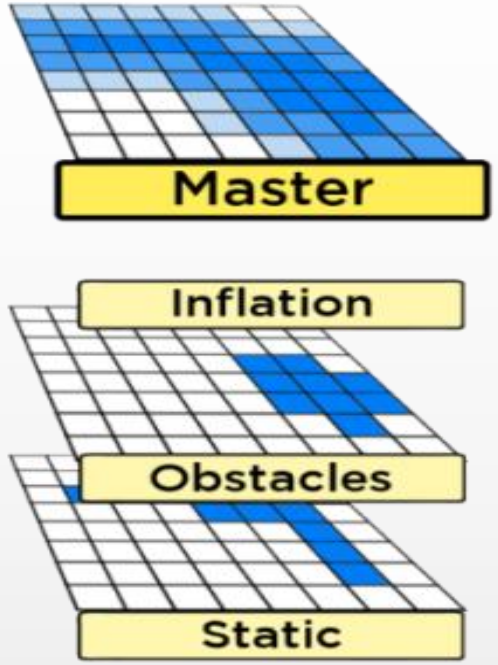
mbot_navigation/launch/move_base.launch

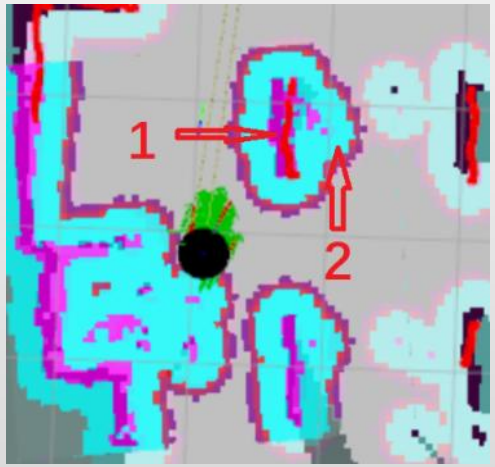
参数含义请参考：http://wiki.ros.org/move_base

ROS Navigation——costmap

使用gmapping构建的地图为全局静态地图，要实现导航避障功能，单靠这一张地图是不够安全的，例如在导航过程中突然出现的障碍物(如：动态障碍物)，因此需要对这张地图进行各种加工修饰，使导航避障更安全，在ROS中是由costmap_2d软件包实现的，该软件包在原始地图上实现了两张新的地图：一个是local_costmap，另外一个是global_costmap，两张costmap一个是为局部路径规划准备的，一个是为全局路径规划准备的。无论是local_costmap还是global_costmap，都可以配置多个图层，包括下面几种：

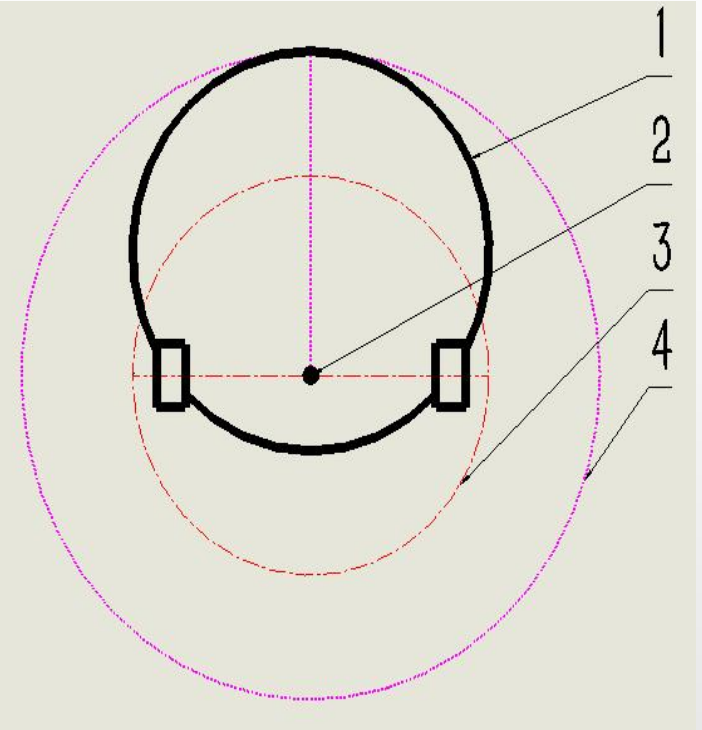
- **Static Map Layer：**静态地图层，基本上不变的地图层，通常是SLAM建立完成的静态地图；
- **Obstacle Map Layer：**障碍地图层，用于动态的记录传感器感知到的障碍物信息；
- **Inflation Layer：**膨胀层，在以上两层地图上进行膨胀（向外扩张），以避免机器人撞上障碍物；
- **Other Layers：**可以通过插件的形式自己实现costmap，目前已有Social Costmap Layer、Range Sensor Layer等开源插件。





ROS Navigation——costmap

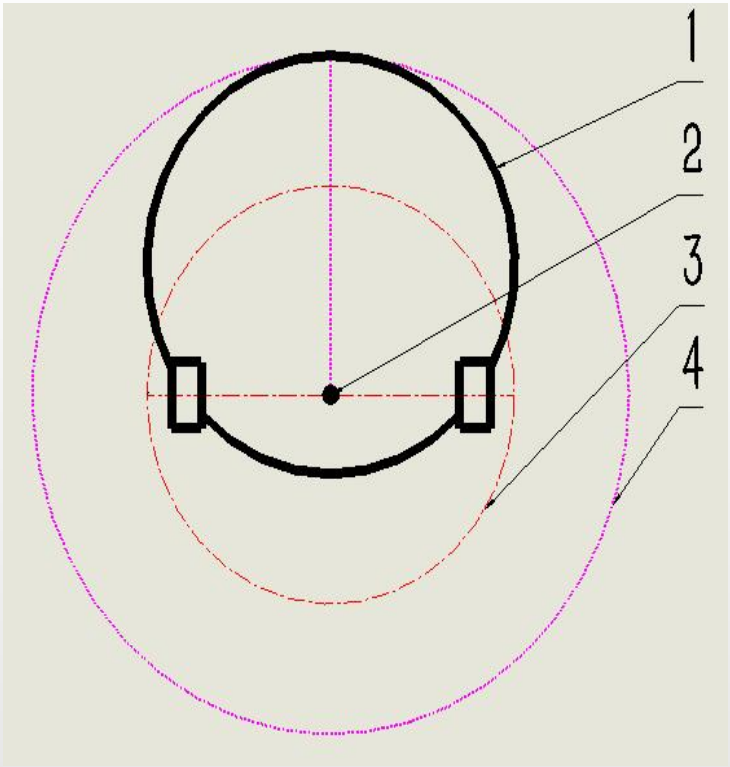
Costmap地图又叫占用栅格图，即将整张地图分成多个小栅格(默认每个小栅格是边长为0.05m的正方型)，每个栅格又分为三种情况：**Occupied被占用（有障碍）**、**Free自由区域（无障碍）**、**Unknown Space未知区域**，划分规则如图所示，图中，“1”为后驱机器人本体，“2”为机器人旋转中心(两轮中心)，“3”为机器人的内切圆，“4”为机器人的外切圆。



ROS Navigation——costmap

当机器人看到障碍物后，会膨胀出一圈安全区域，然后计算栅格与障碍物的距离，根据距离有如下情况：

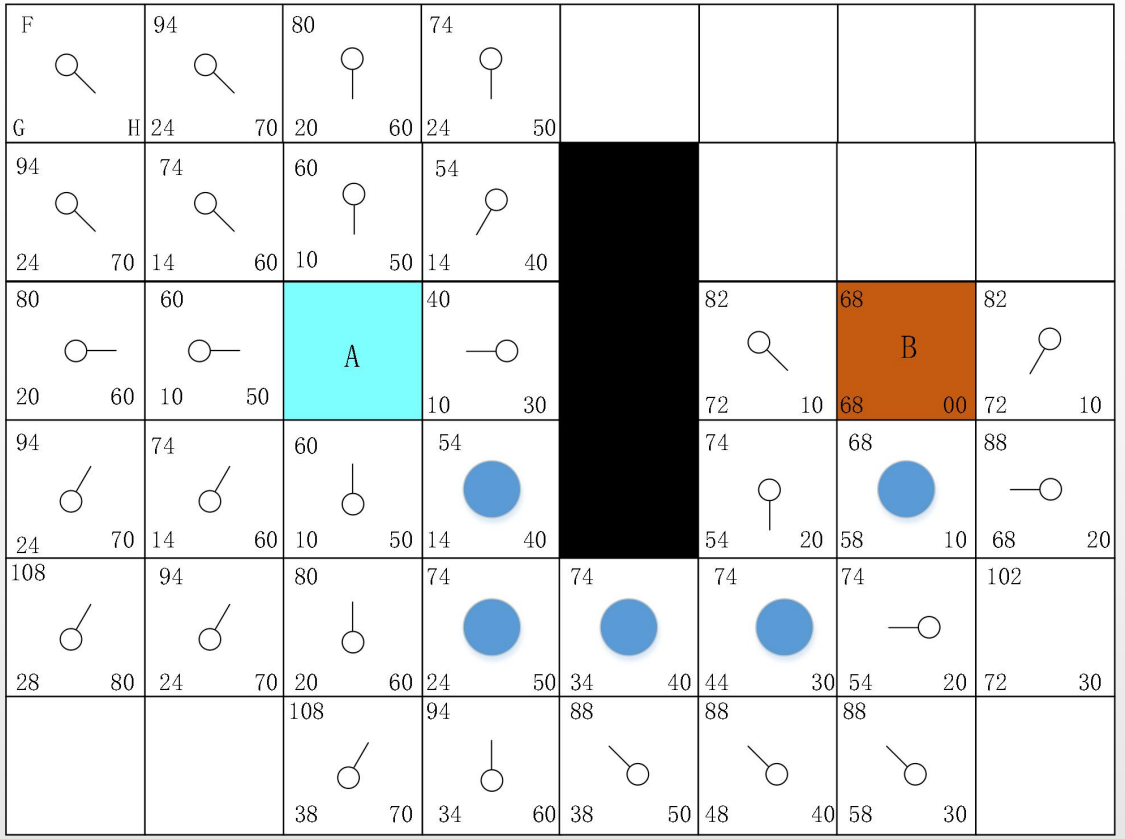
- **小于机器人内切圆半径**，则该栅格代价值为254，表明该栅格是致命的，机器人不允许移动到该栅格，否则会碰到障碍物；
- **大于机器人内切圆半径**，但小于外切圆半径，则该栅格代价值在253~128 之间，距离障碍物越近，代价值越大；
- **大于机器人外切圆半径**，但小于设置的膨胀半径，则该栅格代价值在128~1之间，距离障碍物越近，代价值越大；
- **大于膨胀半径**，则该栅格代价值为0，规划的路径就会往这里走。



ROS Navigation——全局路径规划

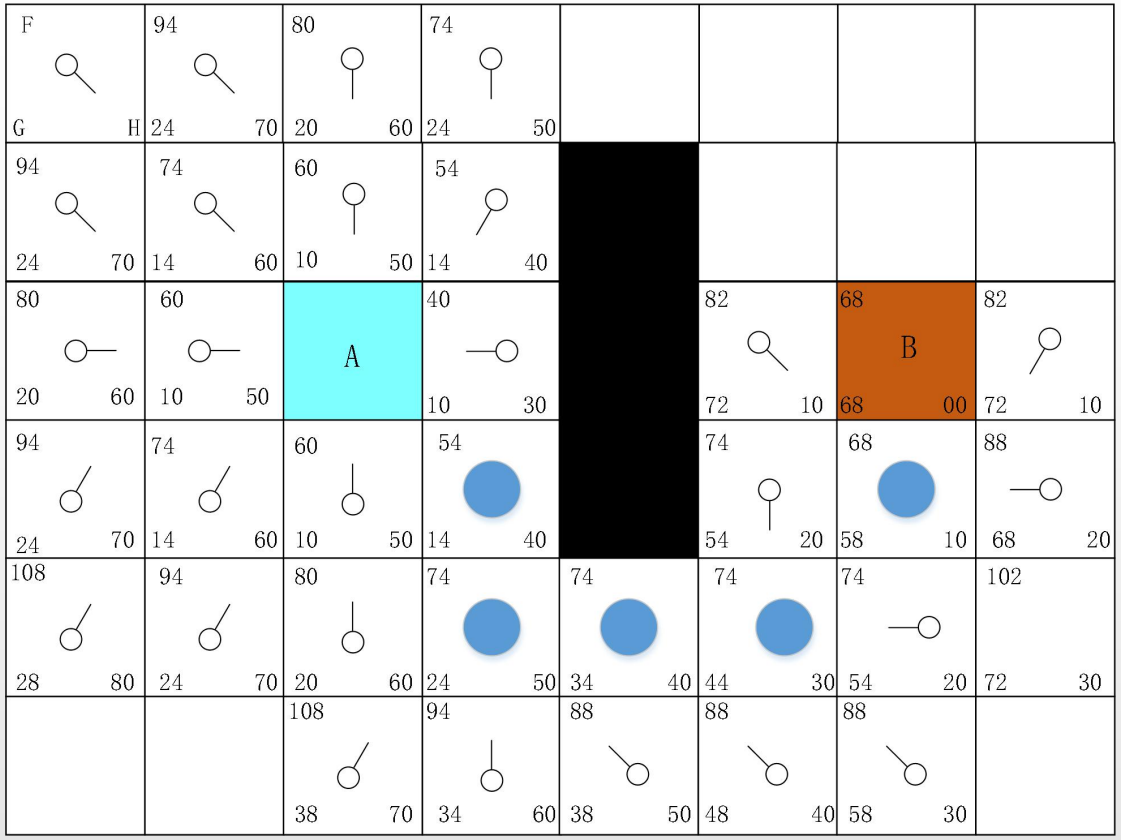
在已知的地图上导航时，先实现机器人当前位置的定位(初始位置设置)，然后在地图上给定目标点，此时可以在当前地点和目标地点之间找到一条**全局路径**，**一般寻找的路径都是找最近的路径**。在ROS中利用dijkstra算法或A*算法实现该功能，这就是全局路径规划。

以A*算法为例：如图所示，起点A，终点B，中间黑色方格为障碍，第一个方格中给出F、G、H三个值标识的位置，其他方格不再标注，其相同的位置代表F、G、H的三个值，圆代表找到的路径。使用A*算法规划路径从起点A到终点B，从起点开始，检查其相邻的方格，然后移动到总代价值最低的栅格，即最优的栅格，逐渐向四周扩展，直至找到目标，



ROS Navigation——全局路径规划

- “G”表示从起点 A 移动到指定方格的移动代价，假设，横向和纵向的移动代价为 10，对角线的移动代价为 14，其在方格左下角标示；
- “H”表示从指定的方格移动到终点 B 的估算成本，有很多方法可以估算 H 值，在本例使用 Manhattan 方法，计算从当前方格横向或纵向移动到达目标所经过的方格数，忽略对角移动，忽略路径中的障碍物，其在方格右下角标示；
- “F”表示总代价值，由 G 和 H相加可得。



ROS Navigation——局部路径规划

机器人在获得目的地信息后，首先经过全局路径规划规划出一条大致可行的路线，然后调用局部路径规划器根据这条路线及costmap的信息规划出合适的局部路径，计算出机器人此时刻最佳的速度指令，发送给机器人运动底盘执行。

在ROS 中常用的局部路径规划算法有

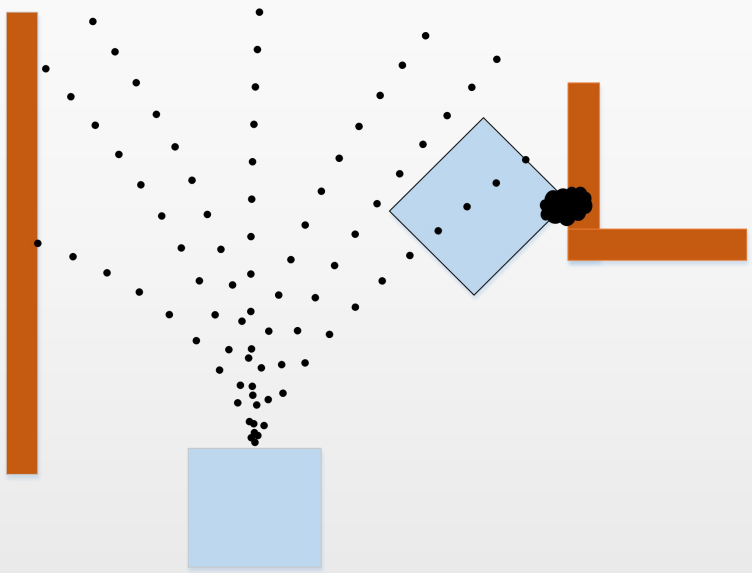
- **dwa_local_planner算法**
- **teb_local_planner算法**

ROS Navigation——局部路径规划

DWA算法

DWA算法全称为Dynamic Window Approach，其原理主要是在速度空间（v,w）中采样多组速度，并模拟这些速度在一定时间内的运动轨迹，通过一个评价函数对这些轨迹打分，最终选择出最优的速度发送给下位机，DWA算法其基本思想如下：

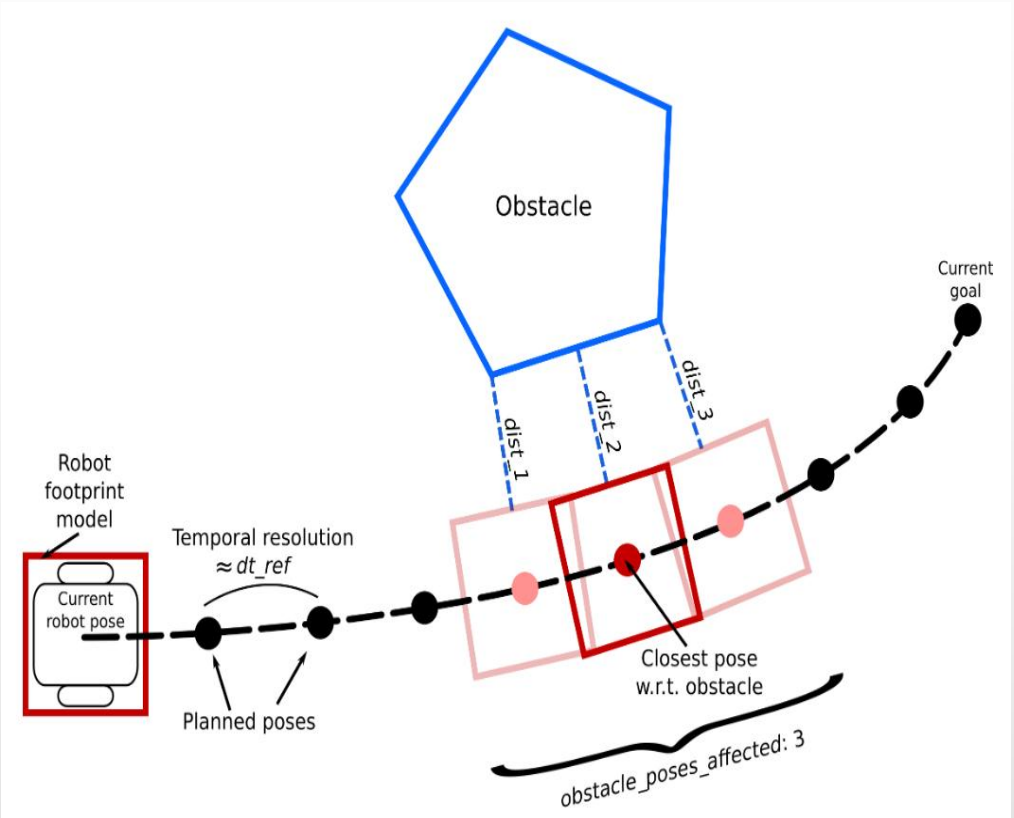
- 在机器人控制空间离散采样(dx, dy, dtheta)；
- 对每一个采样的速度进行前向模拟，看看在当前状态下，使用该采样速度移动一小段时间后会发生什么；
- 评价前向模拟得到的每个轨迹，是否接近障碍物，是否接近目标，是否接近全局路径以及速度等等，舍弃非法路径；
- 选择得分最高的路径，发送对应的速度给底座。



ROS Navigation——局部路径规划

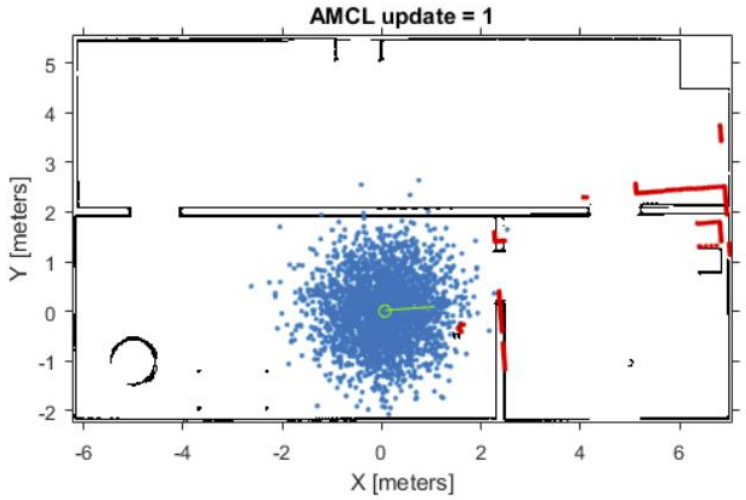
- teb_local_planner其底层算法为TEB（Timed Elastic Band，定时弹性带）算法，该方法根据机器人的轨迹执行时间、与障碍物的分离、在运行时是否符合运动学约束等因素对机器人的轨迹进行局部优化。

- TEB算法先根据全局路径生成初始轨迹，然后根据时间最优，与障碍物分离，满足运动学和动力学约束等条件，转换成最优化问题，最终通过求解sparse scalarized multi objective优化问题有效得到最优轨迹。



ROS Navigation——amcl

- 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

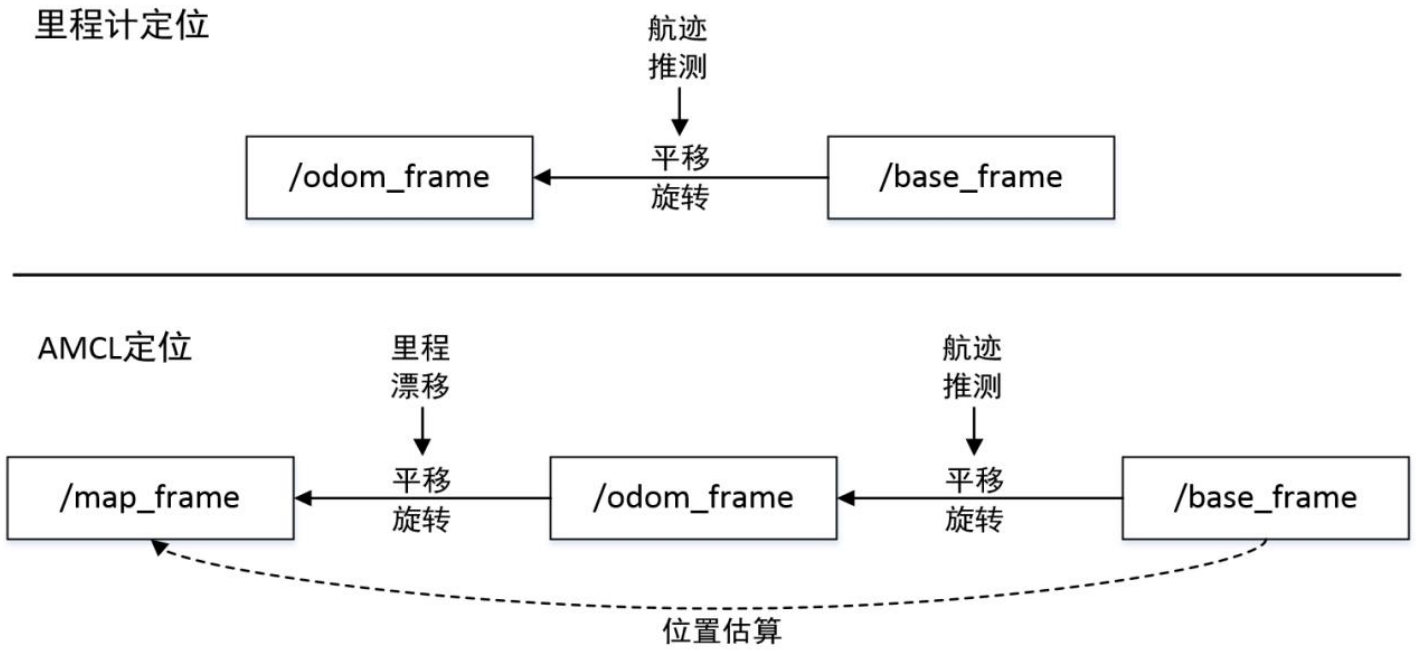


amcl功能包中的话题和服务

	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

具体算法可参考：《概率机器人》

ROS Navigation——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。

ROS Navigation——amcl

```
<launch>
    <arg name="use_map_topic" default="false"/>
    <arg name="scan_topic" default="scan"/>

    <node pkg="amcl" type="amcl" name="amcl" clear_params="true">
        <param name="use_map_topic" value="$(arg use_map_topic)"/>
        <!-- Publish scans from best pose at a max of 10 Hz -->
        <param name="odom_model_type" value="diff"/>
        <param name="odom_alpha5" value="0.1"/>
        <param name="gui_publish_rate" value="10.0"/>
        <param name="laser_max_beams" value="60"/>
        <param name="laser_max_range" value="12.0"/>
        <param name="min_particles" value="500"/>
        <param name="max_particles" value="2000"/>
        <param name="kld_err" value="0.05"/>
        <param name="kld_z" value="0.99"/>
        <param name="odom_alpha1" value="0.2"/>
        <param name="odom_alpha2" value="0.2"/>
        <!-- translation std dev, m -->
        <param name="odom_alpha3" value="0.2"/>
        <param name="odom_alpha4" value="0.2"/>
        <param name="laser_z_hit" value="0.5"/>
        <param name="laser_z_short" value="0.05"/>
        <param name="laser_z_max" value="0.05"/>
        <param name="laser_z_rand" value="0.5"/>
        <param name="laser_sigma_hit" value="0.2"/>
        <param name="laser_lambda_short" value="0.1"/>
        <param name="laser_model_type" value="likelihood_field"/>
        <!-- <param name="laser_model_type" value="beam"/> -->
        <param name="laser_likelihood_max_dist" value="2.0"/>
        <param name="update_min_d" value="0.25"/>
        <param name="update_min_a" value="0.2"/>
        <param name="odom_frame_id" value="odom"/>
        <param name="resample_interval" value="1"/>
        <!-- Increase tolerance because the computer can get quite busy -->
        <param name="transform_tolerance" value="1.0"/>
        <param name="recovery_alpha_slow" value="0.0"/>
        <param name="recovery_alpha_fast" value="0.0"/>
        <remap from="scan" to="$(arg scan_topic)"/>
    </node>
</launch>
```

配置amcl节点

mbot_navigation/launch/amcl.launch

ROS Navigation——arbotix仿真器安装

```
$ sudo apt-get install ros-noeetic-arbotix-*
$ cd ~/catkin_ws/src
$ git clone https://github.com/pirobot/rbx1.git
$ cd rbx1
$ git checkout neoetic-devel-beta
$ ./rbx1-prereq.sh
$ cd ~/catkin_ws
$ catkin_make
$ source ~/catkin_ws/devel/setup.bash
$ rospack profile
```

然后运行

```
$ roslaunch rbx1_bringup fake_turtlebot.launch
$ roslaunch rbx1_nav fake_move_base_blank_map.launch
$ rosrun rviz rviz -d 'rospack find rbx1_nav'/sim.rviz
```

ROS Navigation——arbotix仿真

- 让机器人前进1m

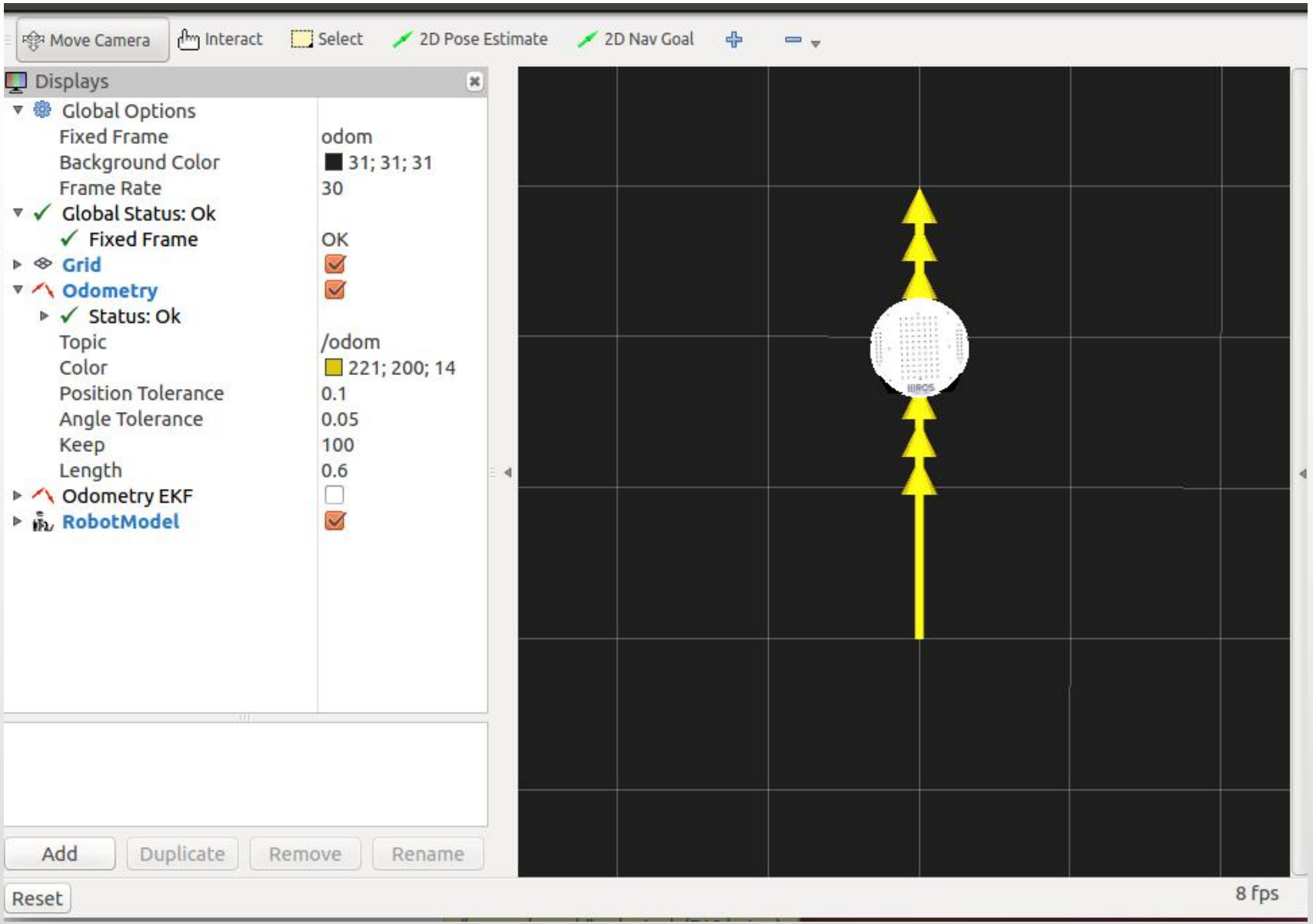
```
$  rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \
'{ header: { frame_id: "base_link" }, pose: { position: { x: 1.0, y: 0, z: 0 },\
 orientation: { x: 0, y: 0, z: 0, w: 1 } } }'
```

- 让机器人后退1m

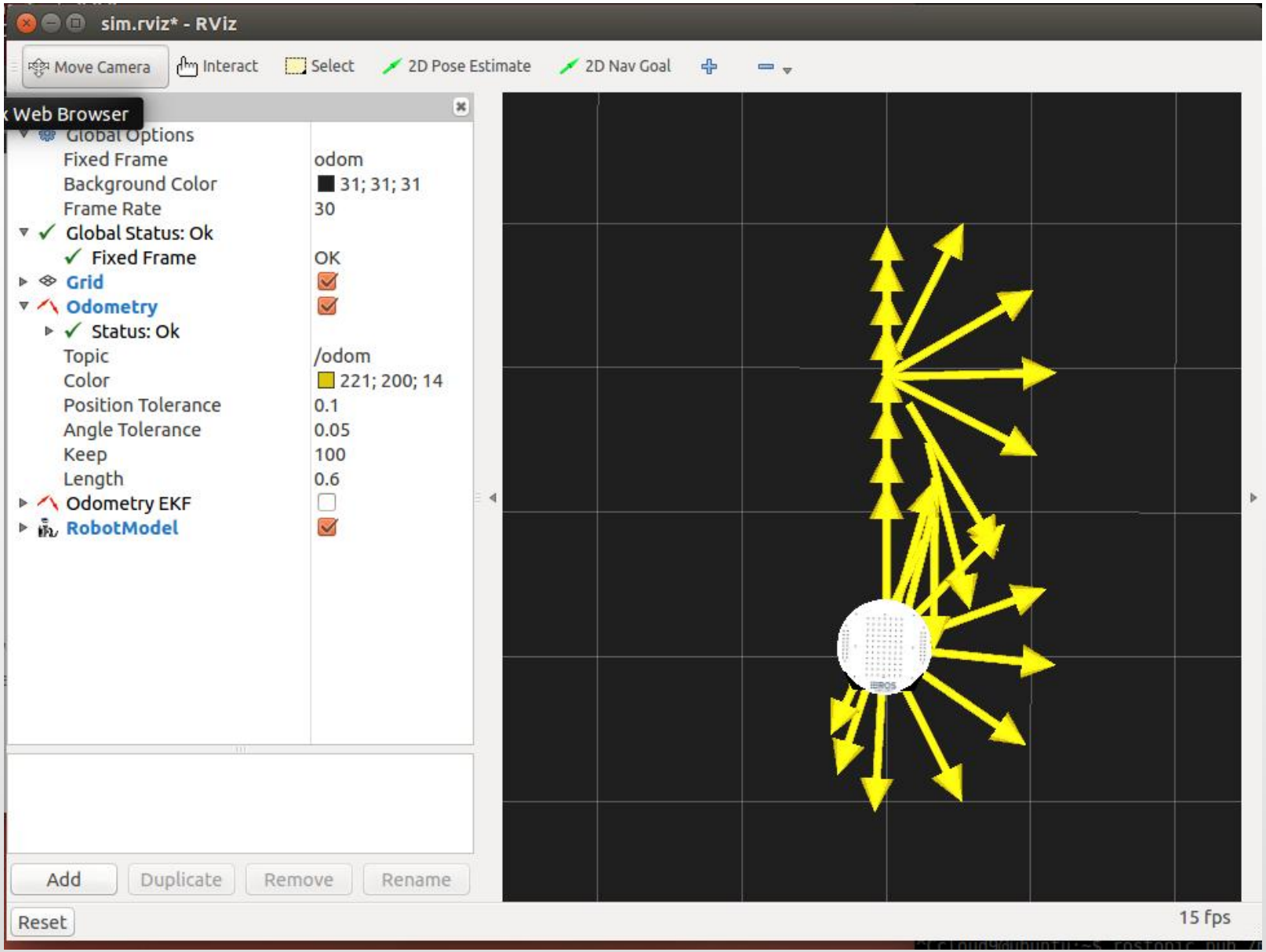
```
$ rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \
'{ header: { frame_id: "map" }, pose: { position: { x: 0, y: 0, z: 0 }, \
orientation: { x: 0, y: 0, z: 0, w: 1 } } }'
```

- 也可以点击rviz上方的2D Nav Goal按键，然后左键选择目标位置，机器人就开始自动导航了。

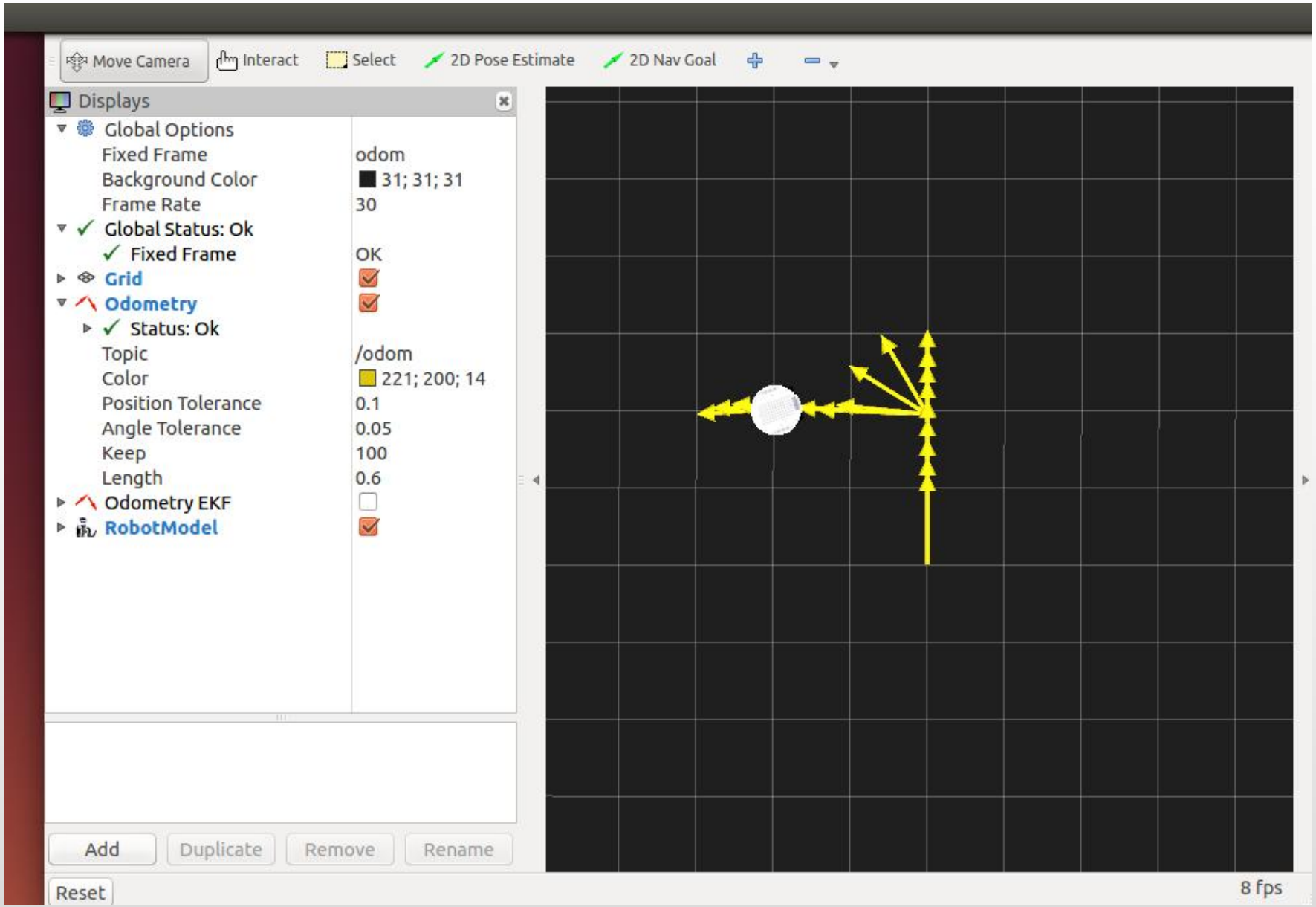
ROS Navigation——arbotix仿真



ROS Navigation——arbotix仿真

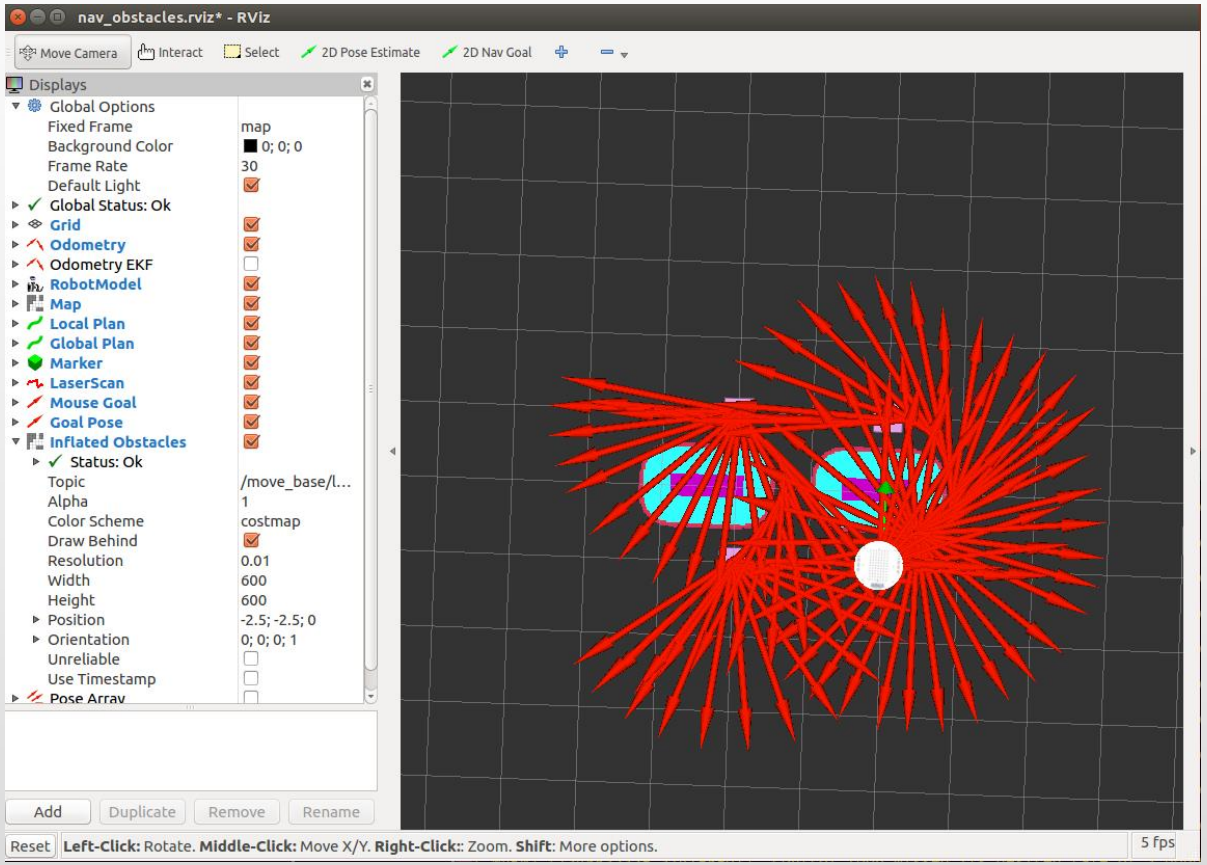


ROS Navigation——arbotix仿真



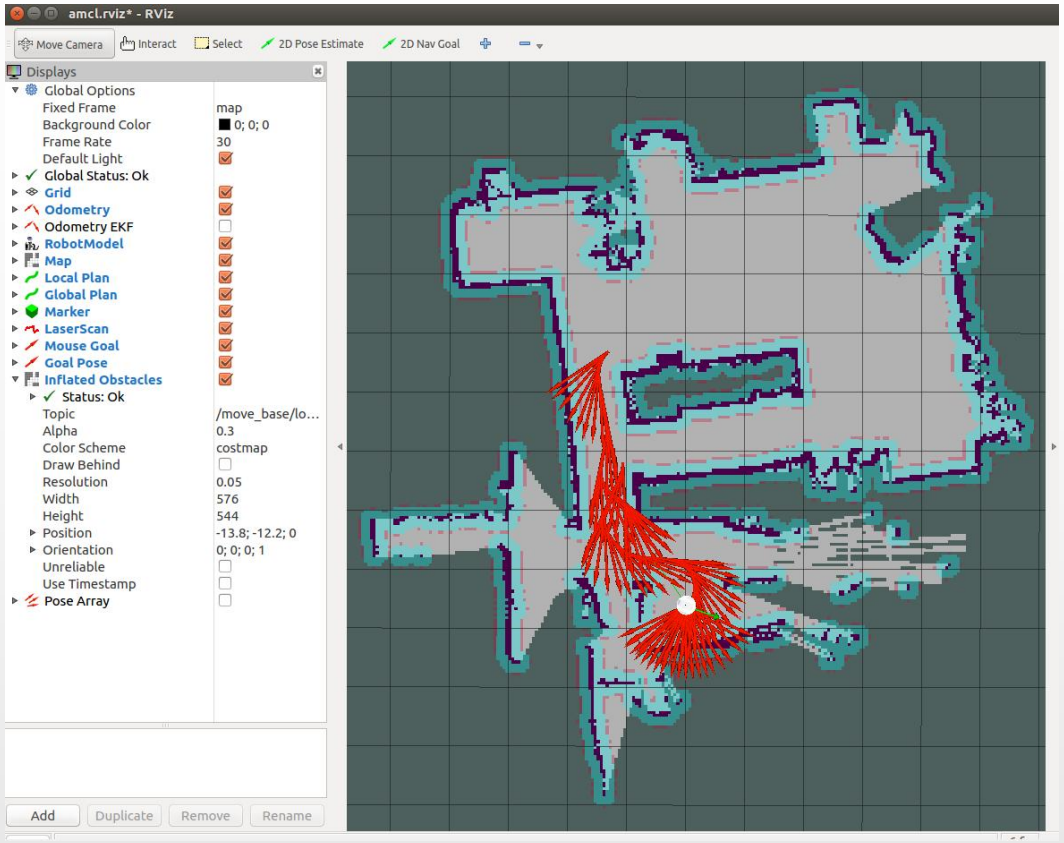
ROS Navigation——arbotix仿真

```
$ roslaunch rbx1_bringup fake_turtlebot.launch
$ roslaunch rbx1_nav fake_move_base_blank_map.launch
$ rosrun rviz rviz -d 'rospack find rbx1_nav'/nav_obstacles.rviz
$ rosrun rbx1_nav move_base_square.py
```



ROS Navigation——arbotix仿真

- 在仿真期间，可以看到细的绿色的线，就是全局规划出来到目标点的位置。机器人实际上只依赖于静态地图和假的里程计。 在图中，紫色是障碍物，周围浅色的椭圆形是根据配置文件中的inflation_radius参数计算出来的安全缓冲区。全局规划的路径基本已经是最短路径了。在仿真的过程中也可以动态重配置这四个配置文件，修改仿真参数。
- base_local_planner_params.yaml
- costmap_common_params.yaml
- global_costmap_params.yaml
- local_costmap_params.yaml



THANKS