11.基于ROS的智能机器人导航应用

方宝富

fangbf@hfut.edu.cn

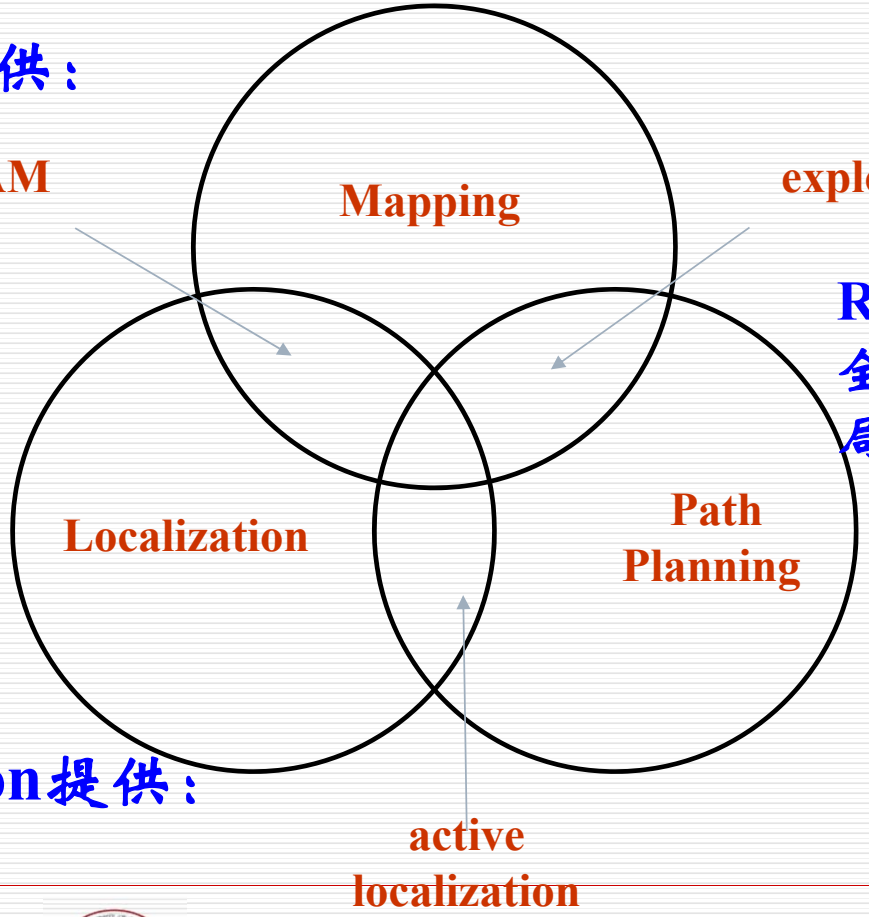
本章提纲

- ROS中的地图
- 几种典型的基于激光的SLAM方法
- Navigation

11.1 ROS中的地图

ROS为SLAM提供：

Gmapping
Karto
Hector
Cartographer



ROS为Path Planning提供：

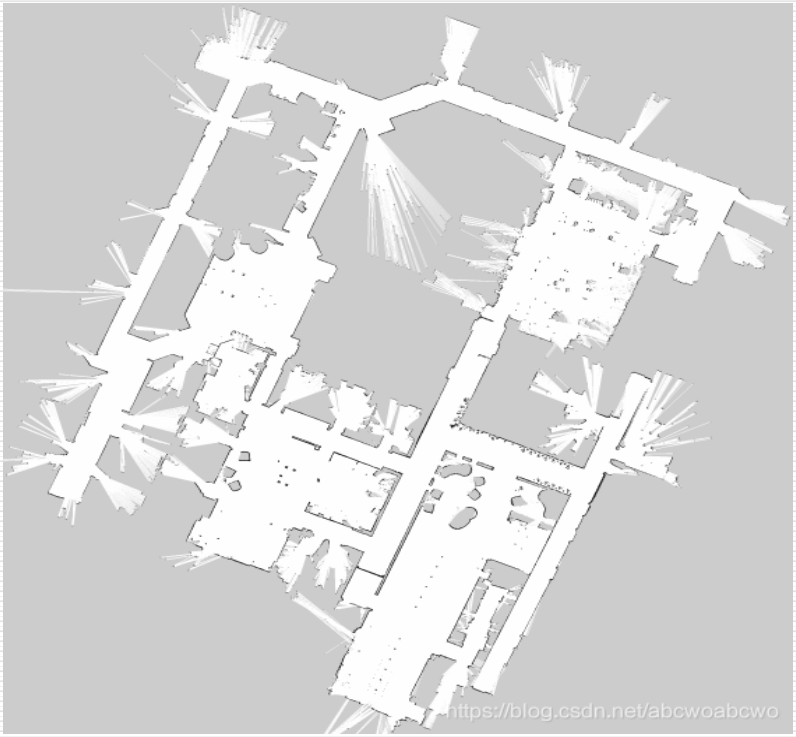
全局路径规划算法：Dijkstra和A*

局部路径规划算法：DWA等

ROS 为localization提供：

AMCL

11.1 ROS中的地图



ROS中的地图很好理解，就是一张普通的灰度图像，通常为pgm格式。这张图像上的黑色像素表示障碍物，白色像素表示可行区域，灰色是未探索的区域。

在SLAM建图的过程中，在RViz里看到一张地图被逐渐建立起来的过程，类似于一块块拼图被拼接成一张完整的地图。

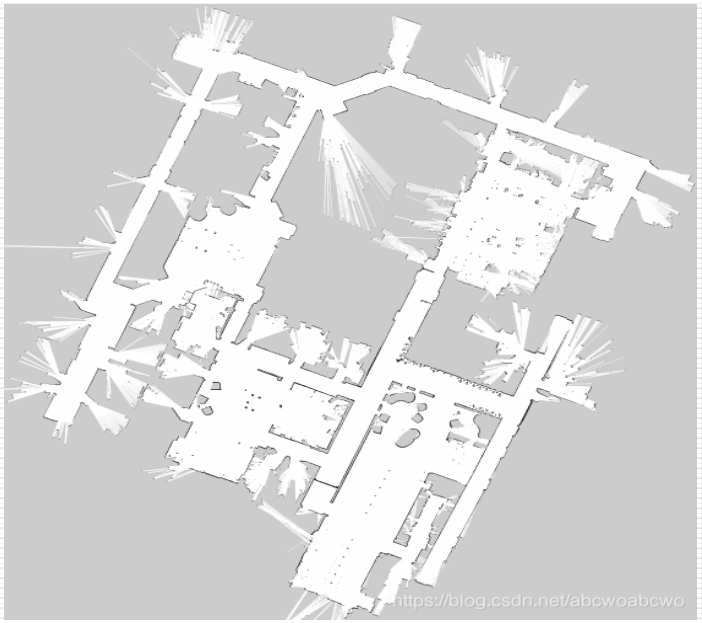
地图在ROS中是以Topic的形式呈现，其Topic名称为/map，消息类型是nav_msgs/OccupancyGrid。

11.1 ROS中的地图

Map

Topic: /map

Type: nav_msgs/OccupancyGrid



nav_msgs/OccupancyGrid.msg

```
std_msgs/Header    header
   uint32    seq
   time      stamp
   string    frame_id          map_frame
nav_msgs/MapMetaData    info
   time    map_load_time
   float32    resolution       分辨率：0.05或者0.025
   uint32     width
   uint32     height           栅格的宽和高
   geometry_msgs/Pose    origin
      geometry_msgs/Point    position
         float64    x
         float64    y
         float64    z
      geometry_msgs/Quaternion    orientation
         float64    x
         float64    y
         float64    z
         float64    w
int8[]    data                 数组大小：栅格的宽*高
```

11.1 ROS中的地图-传感器数据

激光测距仪在ROS中的存储格式：

三维点云数据格式

sensor_msgs/PointCloud2

```
std_msgs/Header header

  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width

sensor_msgs/PointField[] fields
   uint8 INT8=1
   uint8 UINT8=2
   uint8 INT16=3
   uint8 UINT16=4
   uint8 INT32=5
   uint8 UINT32=6
   uint8 FLOAT32=7
   uint8 FLOAT64=8
   string name
   uint32 offset
   uint8 datatype
   uint32 count
bool is_bigendian  // 大小端编码
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

11.1 ROS中的地图-传感器数据

二维激光雷达数据格式

sensor_msgs/LaserScan

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

```
header:
  seq: 3185
  stamp:
    secs: 1511424845
    nsecs: 985671531
  frame_id: laser
angle_min: -2.08007788658
angle_max: 2.07394194603
angle_increment: 0.00613592332229
time_increment: 9.76562514552e-05
scan_time: 0.10000000149
range_min: 0.019999999553
range_max: 5.59999990463
ranges: [0.382999986410141, 0.382999986410141, 0.3889999985694885.....]

intensities: []
```

11.1 ROS中的地图-传感器数据

超声雷达数据格式

rosmsg show
sensor_msgs/LaserEcho

float32[] echoes

11.1 ROS中的地图-传感器数据

IMU 数据格式

sensor_msgs/Imu

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
  float64 x
  float64 y
  float64 z
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
  float64 x
  float64 y
  float64 z
float64[9] linear_acceleration_covariance
```

11.1 ROS中的地图-传感器数据

Odom 数据格式 **nav_msgs/Odometry**

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

本章提纲

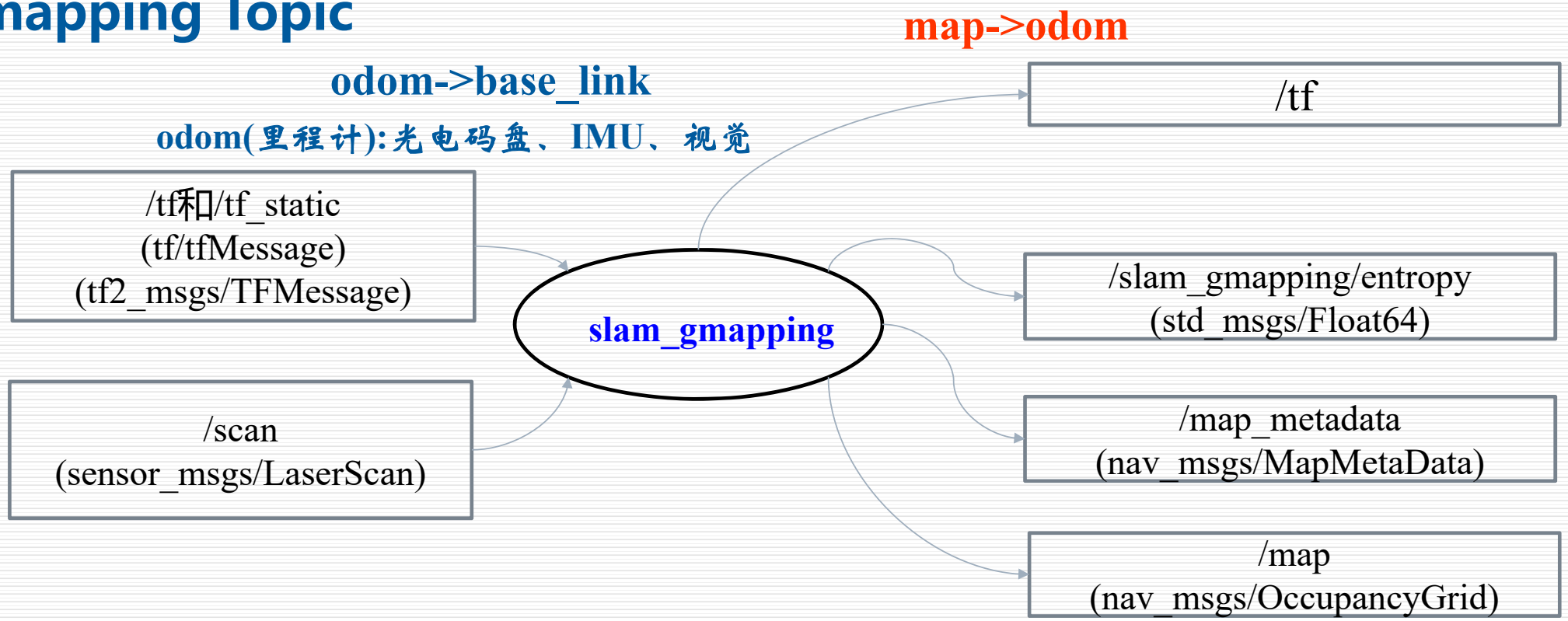
- ROS中的地图
- 几种典型的基于激光的SLAM方法
- Navigation

11.2 几种典型的基于激光的SLAM方法-Gmapping

- Gmapping是一种基于激光的SLAM算法，是移动机器人中使用最多的SLAM算法
- 由Grisetti等人提出是基于 Rao-Blackwellized的粒子滤波的 SLAM方法
- 基于粒子滤波的算法，用许多加权粒子表示路径的后验概率，每个粒子都给出一个重要性因子。
- 需要大量的粒子才能获得比较好的的结果，从而增加算法计算复杂性
- 与PF重采样过程相关的粒子退化耗尽问题也降低了算法的准确性
- 粒子退化问题包括在重采样阶段从样本集粒子中消除大量的粒子

11.2 几种典型的基于激光的SLAM方法-Gmapping

Gmapping Topic



11.2 几种典型的基于激光的SLAM方法-Gmapping

slam_gmapping会从/tf中获得机器人各坐标系的数据及其关系

base_frame与laser_frame之间的tf，即机器人底盘和激光雷达之间的变换；

base_frame与odom_frame之间的tf，即底盘和里程计原点之间的坐标变换；

odom_frame 里程计原点所在的坐标系。

/scan :激光雷达数据，类型为sensor_msgs/LaserScan

11.2 几种典型的基于激光的SLAM方法-Gmapping

（1）/tf： 主要是输出map_frame和odom_frame之间的变换 定位

/slam_gmapping/entropy： std_msgs/Float64类型，机器人位姿估计的分散程度

（2）/map： slam_gmapping建立的地图 建图

/map_metadata： 地图的相关信息

在SLAM场景中，地图是作为SLAM的结果被不断地更新和发布

11.2 几种典型的基于激光的SLAM方法-Gmapping

Gmapping Service

Service: /dynamic_map
Type: nav_msgs/GetMap

nav_msgs/GetMap.srv

service直接call，不需要带参数

```
# Get the map as a nav_msgs/OccupancyGrid
---
nav_msgs/OccupancyGrid map
```

11.2 几种典型的基于激光的SLAM方法-Gmapping

Gmapping Param

~inverted_laser (string, default: “false”)　　#雷达是否反向放置
~throttle_scans (int, default: 1)　　#每多少次scan才处理一次
~base_frame (string, default: “base_link”)　　#底盘坐标系
~map_frame (string, default: “map”)　　#机器人坐标系
~odom_frame (string, default: “odom”)　　#里程计坐标系
~map_update_interval (float, default: 5.0)　　#更新地图周期(秒)
~maxUrange (float, default: 80.0)　　#激光最大可用范围
~particles (int, default: 30)　　#粒子数
~transform_publish_period (float, default: 0.05)　　#tf发布周期
…

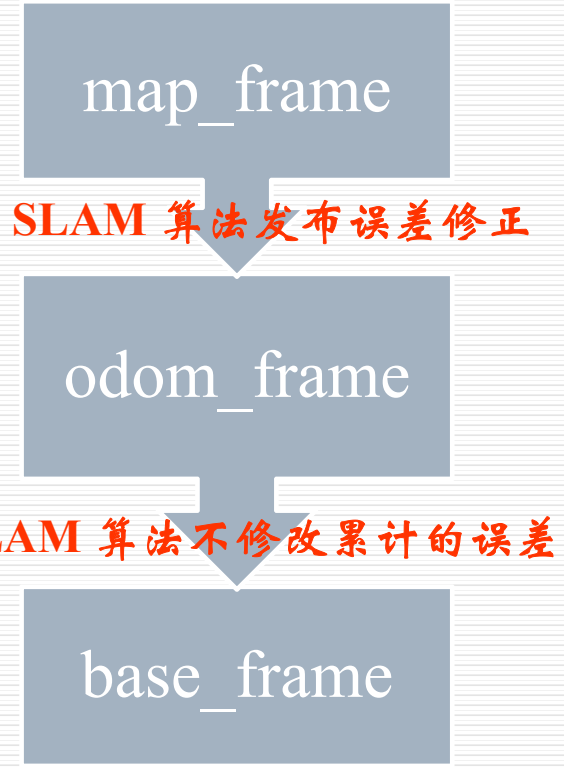
11.2 几种典型的基于激光的SLAM方法-Gmapping

Gmapping的定位修正

/odom: 类型为nav_msgs/Odometry，反映里程计估测的机器人位置、方向、线速度、角速度信息。

/tf: 主要是输出odom_frame和base_frame之间的tf。

里程计都是对机器人的位姿进行估计，存在着累计误差，因此当运动时间较长时，odom_frame和base_frame之间变换的真实值与估计值的误差会越来越大



Gmapping做法：

odom_frame和base_frame的tf数据不做修正

把里程计误差的修正发布到map_frame和odom_frame之间的tf上，即把误差补偿在地图坐标系和里程计原点坐标系之间。通过这种方式来修正定位。

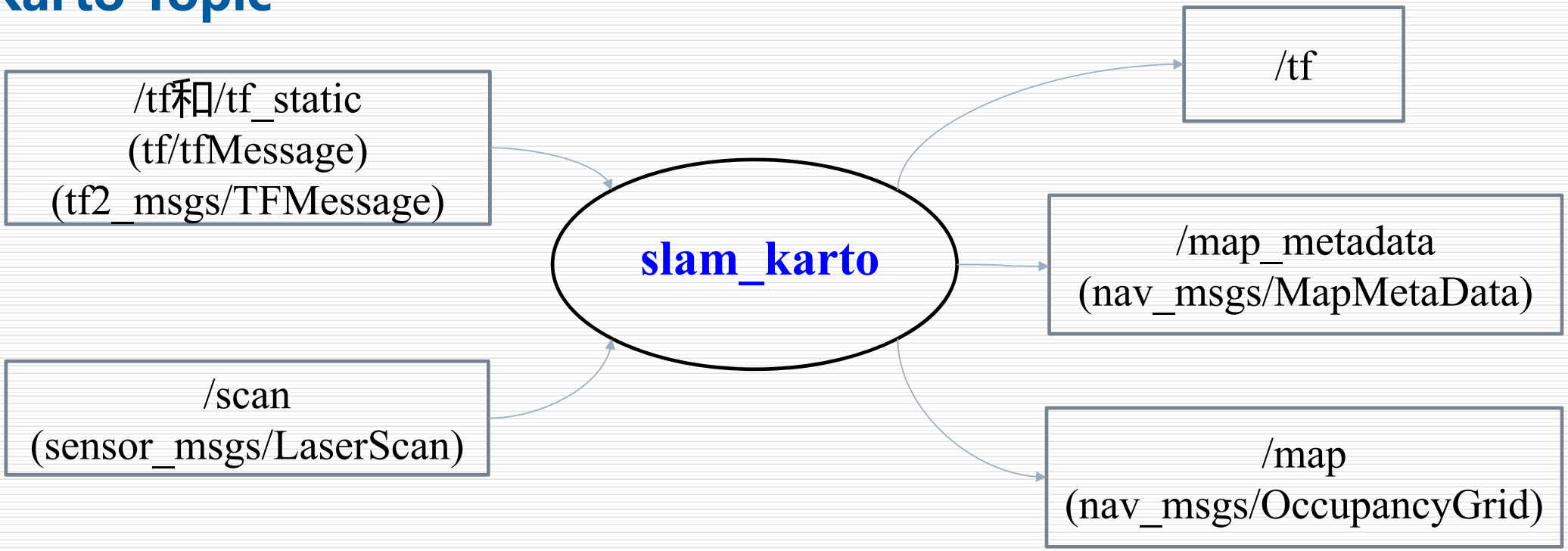
11.2 几种典型的基于激光的SLAM方法-Karto

KartoSLAM是基于图优化的方法，用高度优化和非迭代 cholesky矩阵进行稀疏系统解耦作为解。图优化方法利用图的均值表示地图，每个节点表示机器人轨迹的一个位置点和传感器测量数据集，箭头的指向的连接表示连续机器人位置点的运动，每个新节点加入，地图就会依据空间中的节点箭头的约束进行计算更新。

KartoSLAM的ROS版本，其中采用的稀疏点调整（the Spare Pose Adjustment(SPA)）与扫描匹配和闭环检测相关。landmark越多,内存需求越大,然而图优化方式相比其他方法在大环境下制图优势更大.在某些情况下KartoSLAM更有效,因为他仅包含点的图(robot pose),求得位置后再求map.

11.2 几种典型的基于激光的SLAM方法-Karto

Karto Topic



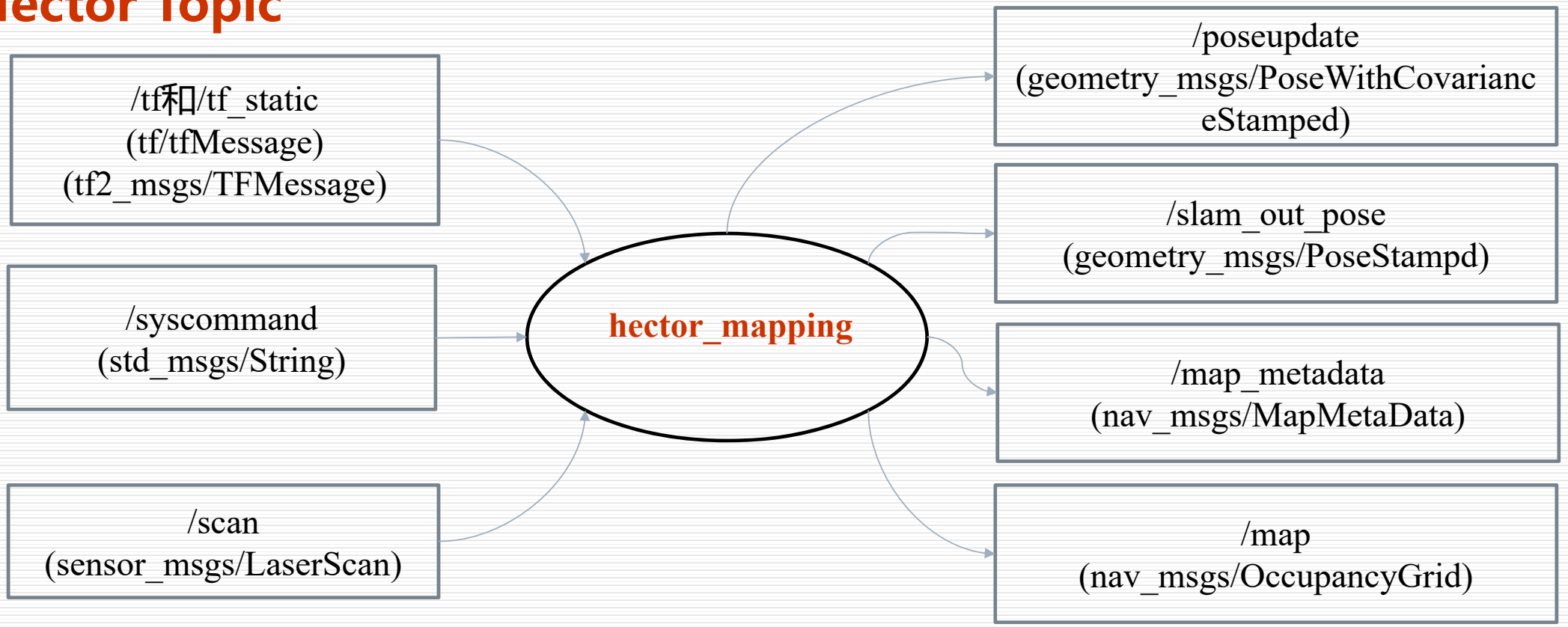
11.2 几种典型的基于激光的SLAM方法-Karto

输入的Topic同样是/tf和/scan，其中/tf里要连通odom_frame与base_frame，还有laser_frame。这里和Gmapping完全一样。

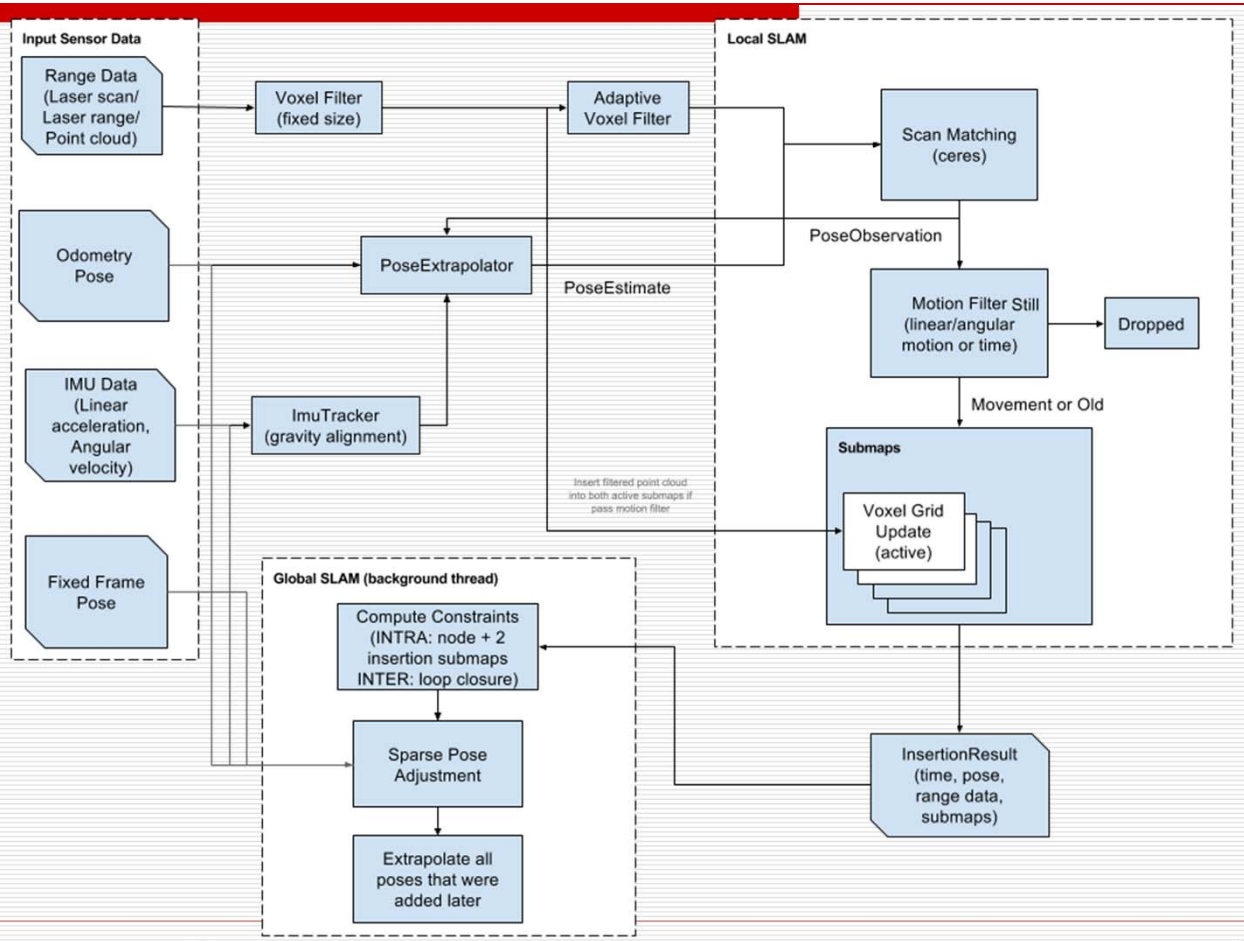
唯一不同的地方是输出，slam_karto的输出少相比slam_gmapping了一个位姿估计的分散程度.

11.2 几种典型的基于激光的SLAM方法- hector

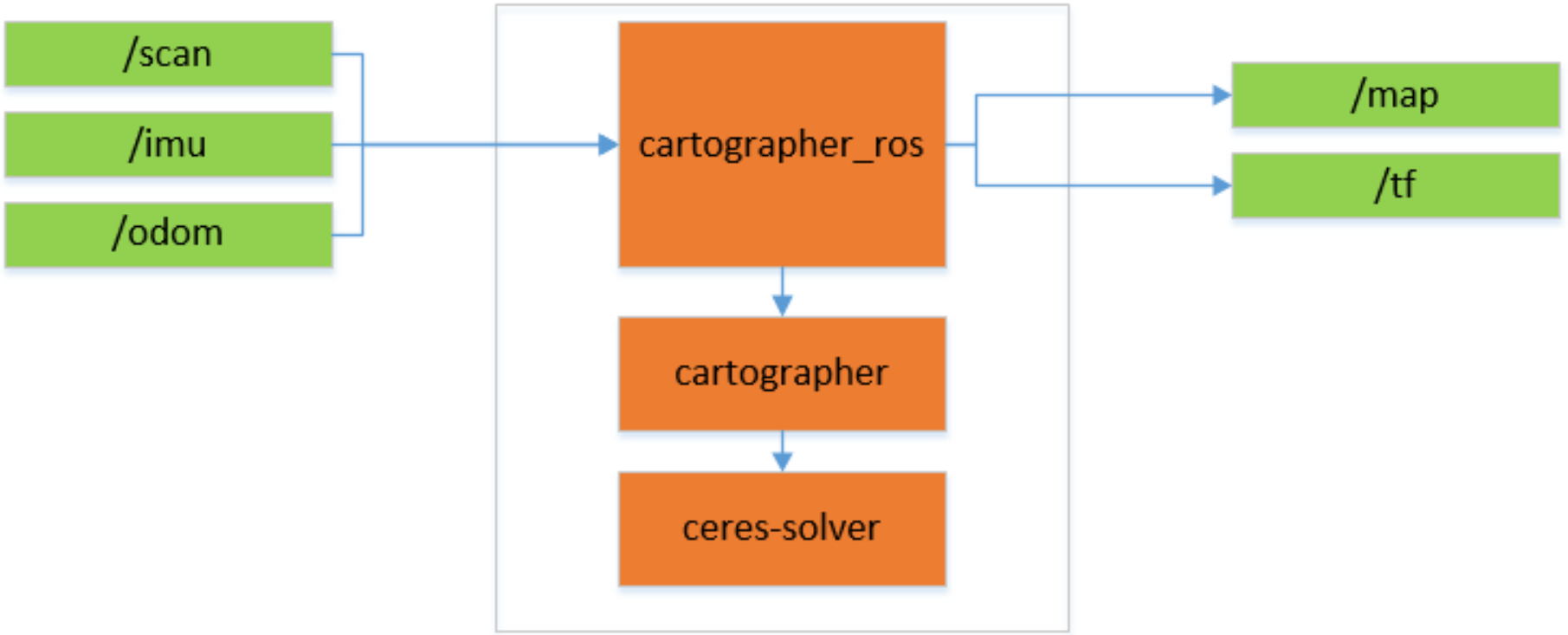
Hector Topic



11.2 几种典型的基于激光的SLAM方法- Cartography



11.2 几种典型的基于激光的SLAM方法- Cartography

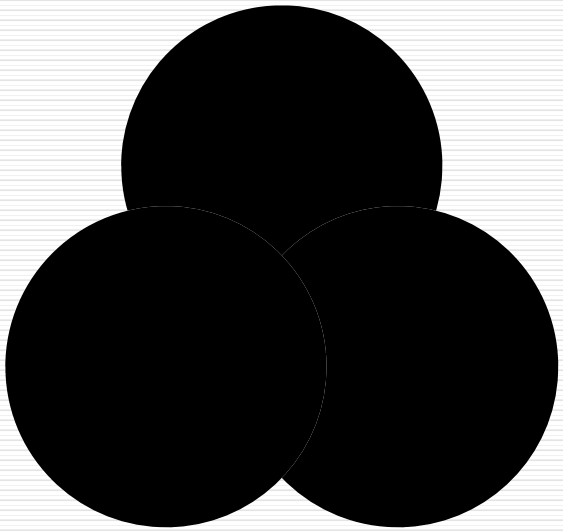


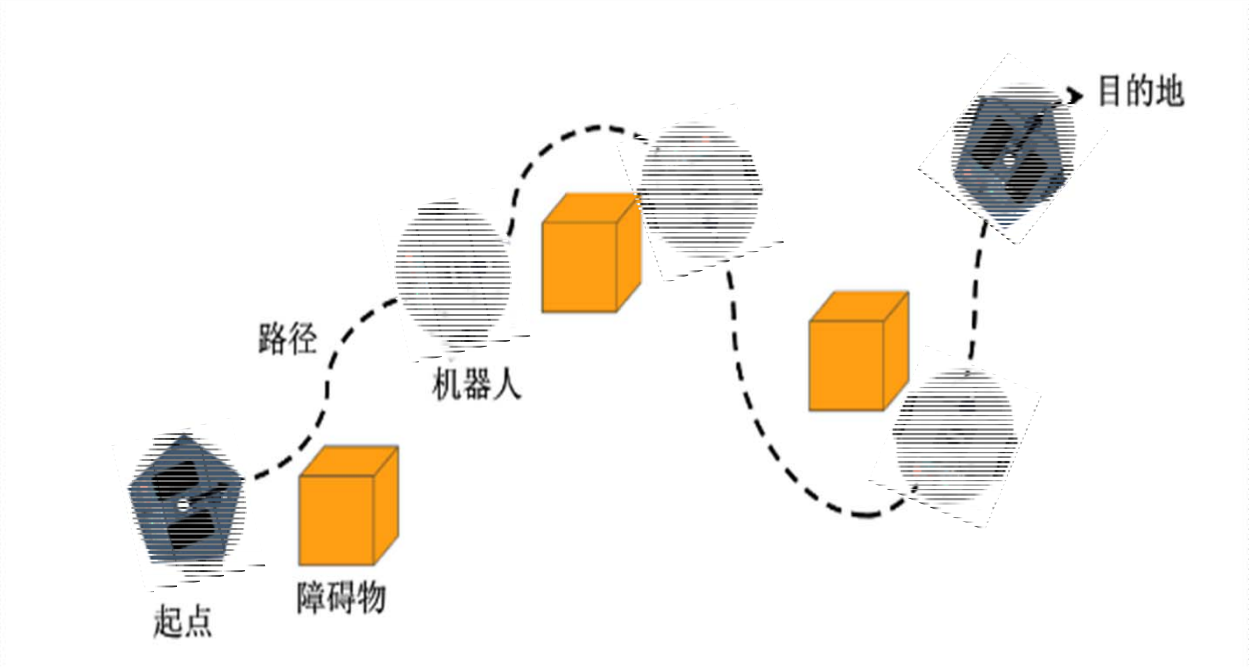
本章提纲

- ROS中的地图
- 几种典型的基于激光的SLAM方法
- Navigation

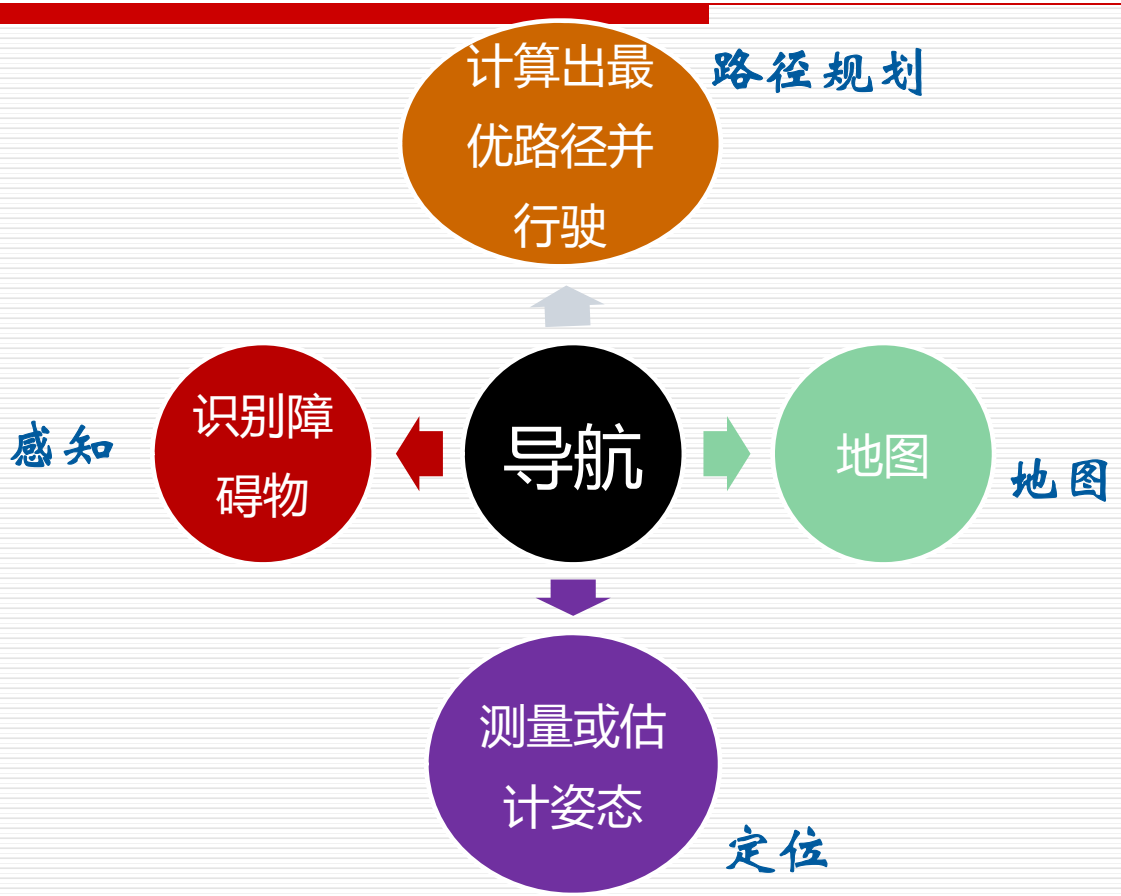
11.3 Navigation

机器人基于地图，从起点A前进到终点B，过程中不发生碰撞并满足自身动力学模型





11.3 Navigation



11.3 Navigation

ROS Navigation Stack（MetaPackage）

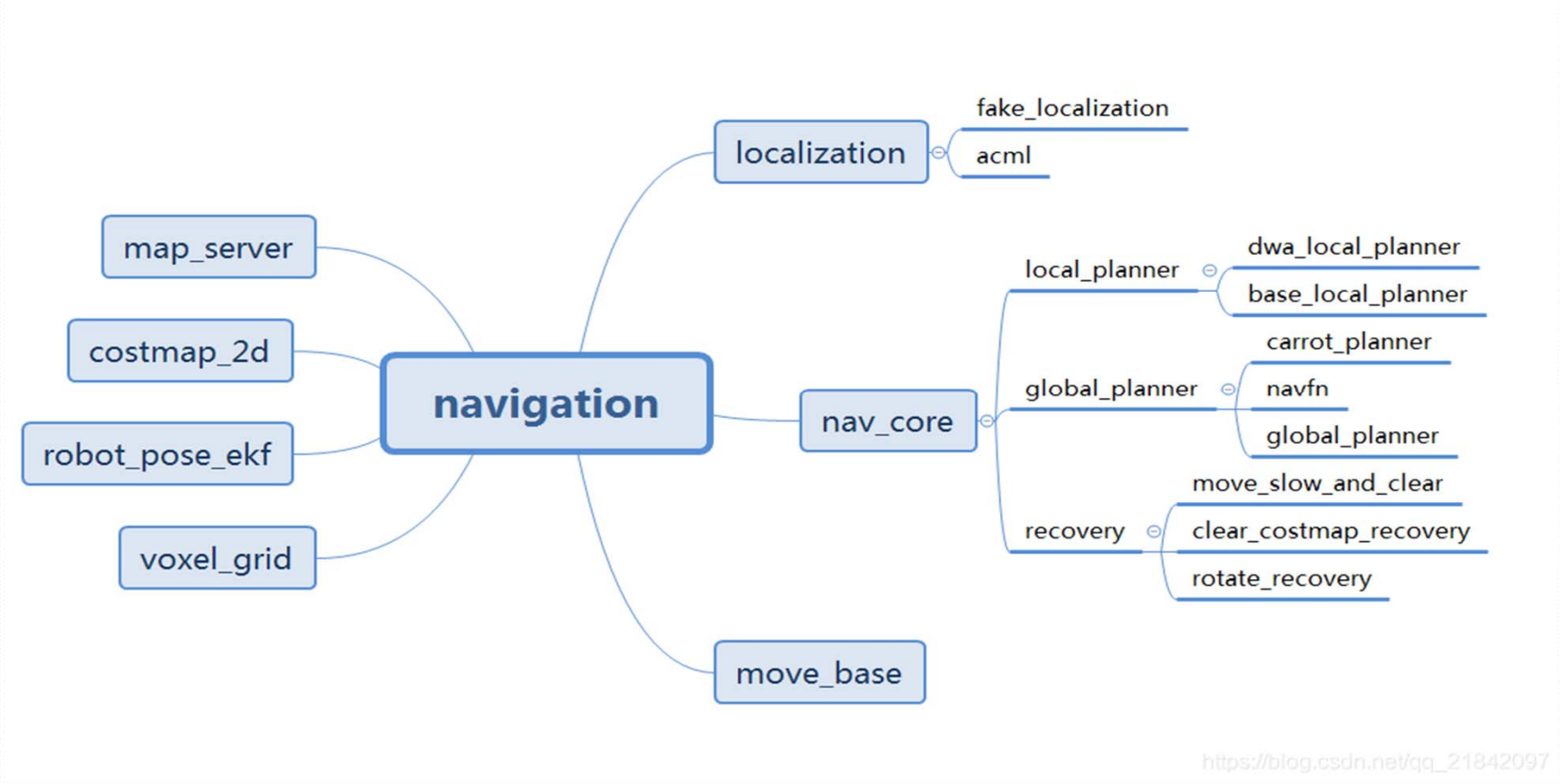
GitHub, Inc. [US] https://github.com/ros-planning/navigation

Branch: melodic-devel | New pull request | Find File | Clone or download

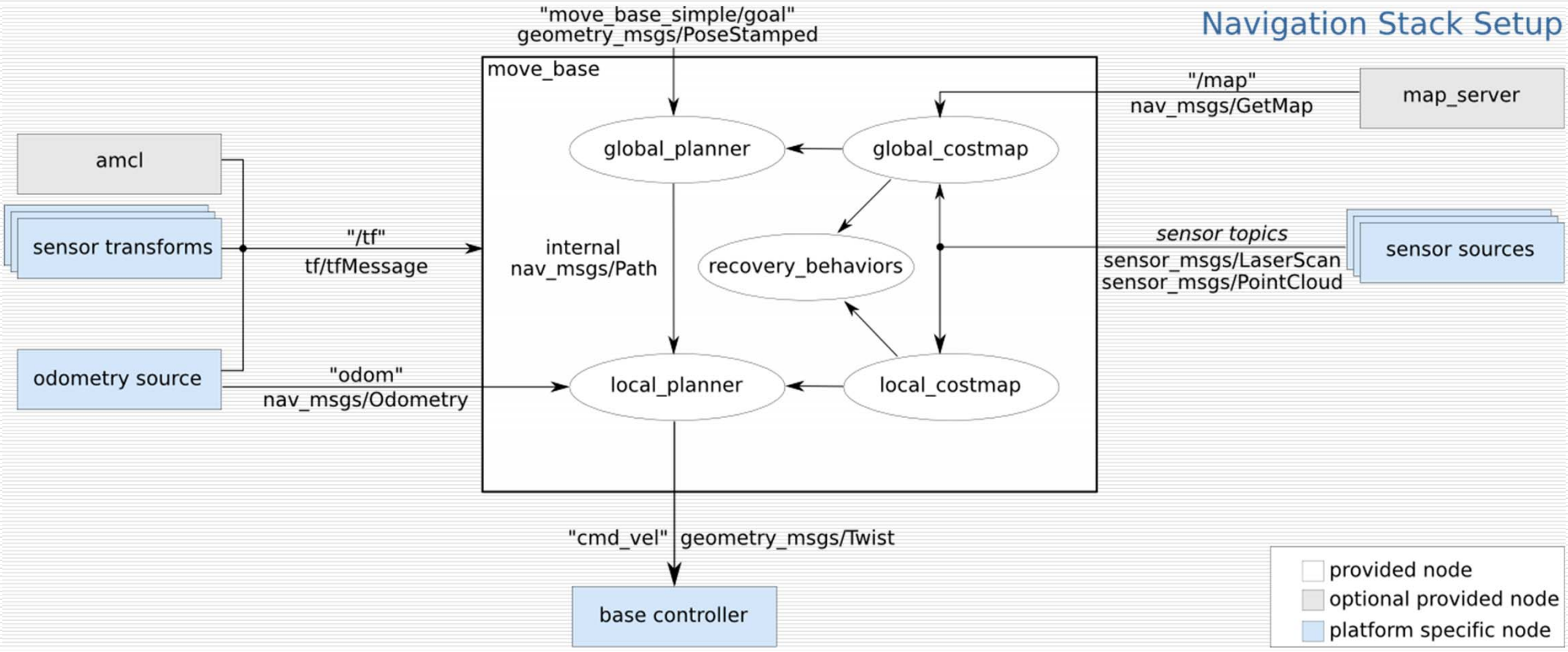
DLu Drop Parameter Magic (#893) — Latest commit d99320a 15 days ago

amcl	revert unrelated changes.	5 months ago
base_local_planner	Provide different negative values for unknown and out-of-map costs (#833	16 days ago
carrot_planner	1.16.2	11 months ago
clear_costmap_recovery	Add force_updating and affected_maps parameters to tailor clear costm...	16 days ago
costmap_2d	Drop Parameter Magic (#893)	15 days ago
dwa_local_planner	Set footprint before in place rotation continuation (#829) (#861)	5 months ago
fake_localization	Remove leading slashes from default frame_id parameters	7 months ago
global_planner	Remove unused visualize_potential (#866)	4 months ago
map_server	map_server Windows build bring up	5 months ago
move_base	Added publishZeroVelocity() before starting planner (#751)	16 days ago
move_slow_and_clear	1.16.2	11 months ago
nav_core	1.16.2	11 months ago
navfn	Remove leading slashes from default frame_id parameters	7 months ago
navigation	1.16.2	11 months ago
rotate_recovery	1.16.2	11 months ago
voxel_grid	1.16.2	11 months ago
.gitignore	ignore eclipse project files.	4 years ago
README.md	update build farm status link	11 months ago

11.3 Navigation

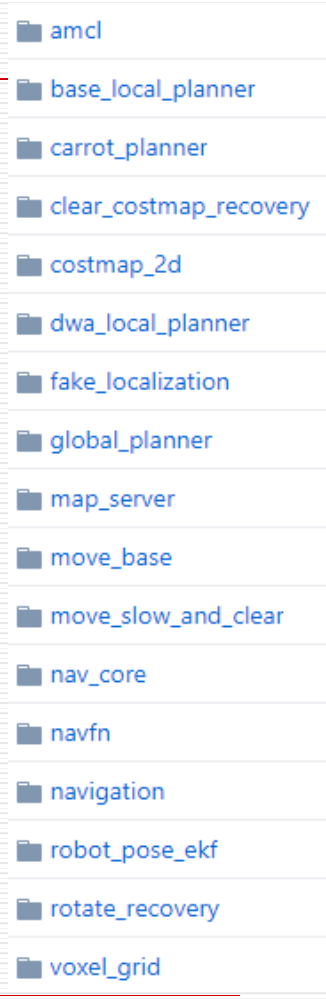


11.3 Navigation

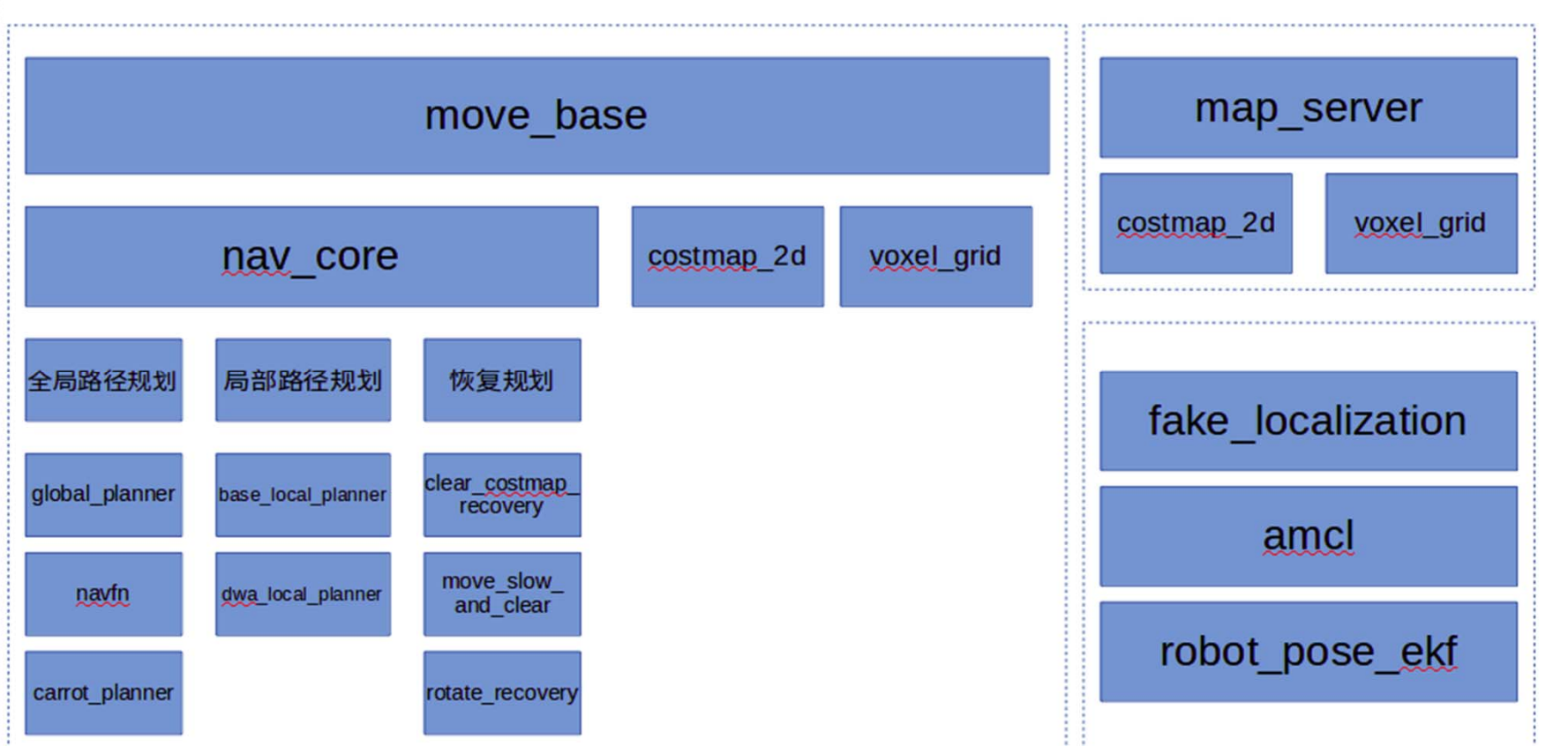


11.3 Navigation

包名	功能	包名	功能
amcl	定位	navfn	实现了Dijkstra和A*全局规划算法
Fake_localization	定位	global_planner	重新实现了Dijkstra和A*全局规划算法
map_server	提供地图	clear_costmap_recovery	实现了清除代价地图的恢复行为
move_base	路径规划节点	rotate_recovery	实现了旋转的恢复行为
nav_core	路径规划接口类	move_slow_and_clear	实现了缓慢移动的恢复行为
base_local_planner	实现了Trajectory Rollout和DWA两种局部规划算法	costmap_2d	二维代价地图
dwa_local_planner	重新实现了DWA局部规划算法	voxel_grid	三维小方块
parrot_planner	实现了较简单的全局规划算法	robot_pose_ekf	机器人位姿的卡尔曼滤波



11.3 Navigation



11.3 Navigation

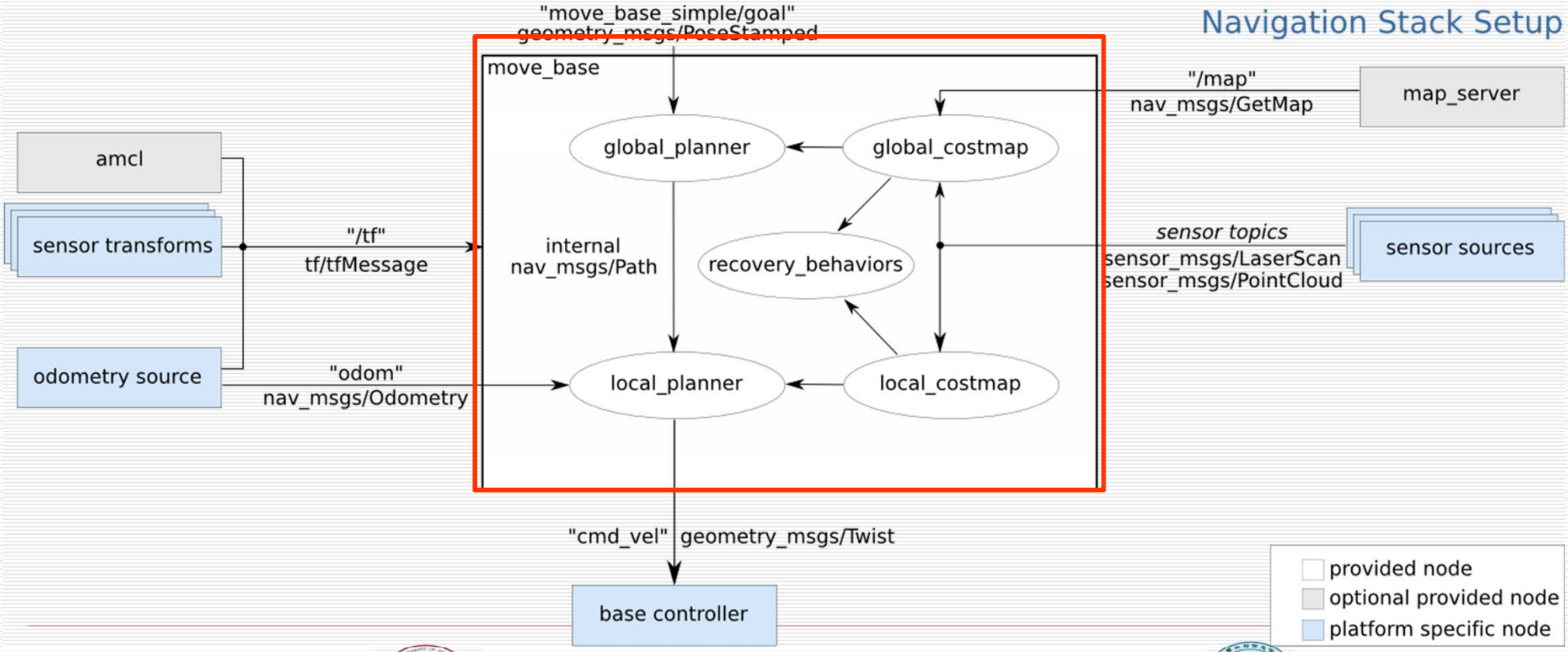
整个navigation stack可以分为三部分：

- 导航方面，move_base为实现逻辑框架，而nav_core提供了全局路径规划器/局部路径规划器通过接口，通过这个接口可以实现按照插件形式来更换算法，另外还提供了两种（二维和三维）栅格地图；
- 地图方面，地图服务器主要是管理地图,用于读写地图并发布地图消息供其他功能包订阅.
- 定位方面fake_localization提供了一个实现，robot_pose_ekf主要是综合传感器信息修正机器人的里程值，而amcl主要是用于提供将机器人的里程计值和地图结合，估算出机器人在地图中位置。

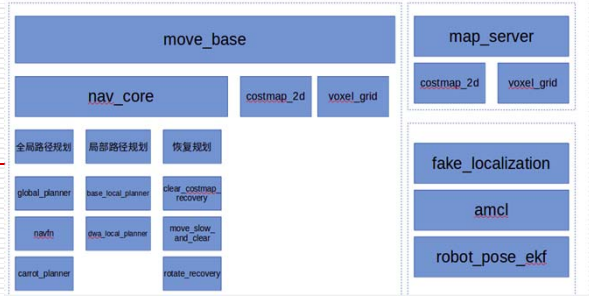
11.3 Navigation- move base模块

move_base

全局规划、局部规划、处理异常行为



11.3 Navigation- move base模块



(1) nav_core

navigation包里有一个nav_core，其定义了核心的几个类应该有的基本功能，在此基础上，nav_core定义了三个接口BaseGlobalPlanner，BaseLocalPlanner和RecoveryBehavior，各自继承和实现了相关功能

- BaseGlobalPlanner

  是全局导航的接口，规定一个功能函数makePlan，给定起始跟目标，输出路径(一系列pose)走过去

- BaseLocalPlanner

  规定了一个核心函数computeVelocityCommands，就是计算局部地图内的下一步控制指令(线速度，角速度)

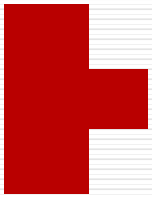
- RecoveryBehavior

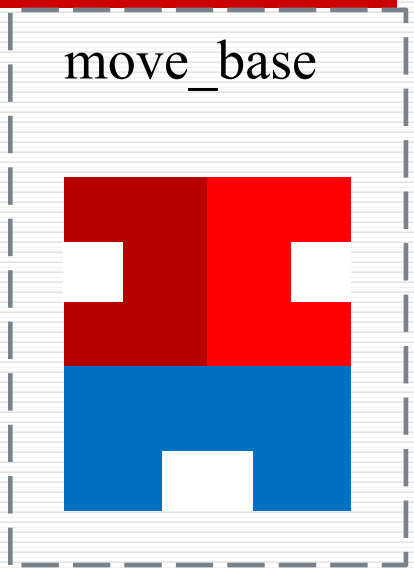
  规定一个runBehavior，在小车卡住情况下执行运动恢复，回到正常的导航状态

11.3 Navigation-move base模块-nav_core

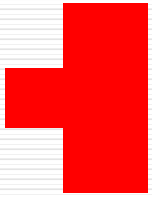
Base Local Planner

base_local_planner
dwa_local_planner





Base Global Planner



parrot_planner
navfn
global_planner

Recovery Behavior

clear_costmap_recovery
rotate_recovery
move_slow_and_clear

11.3 Navigation-move base模块-nav_core

插件

move_base要运行起来，需要选择好插件，包括三种插件

- base_local_planner
- base_global_planner
- recovery_behavior

这三种插件都得指定，否则系统会指定默认值。

11.3 Navigation-move base模块-nav_core

（2）base_local_planner插件

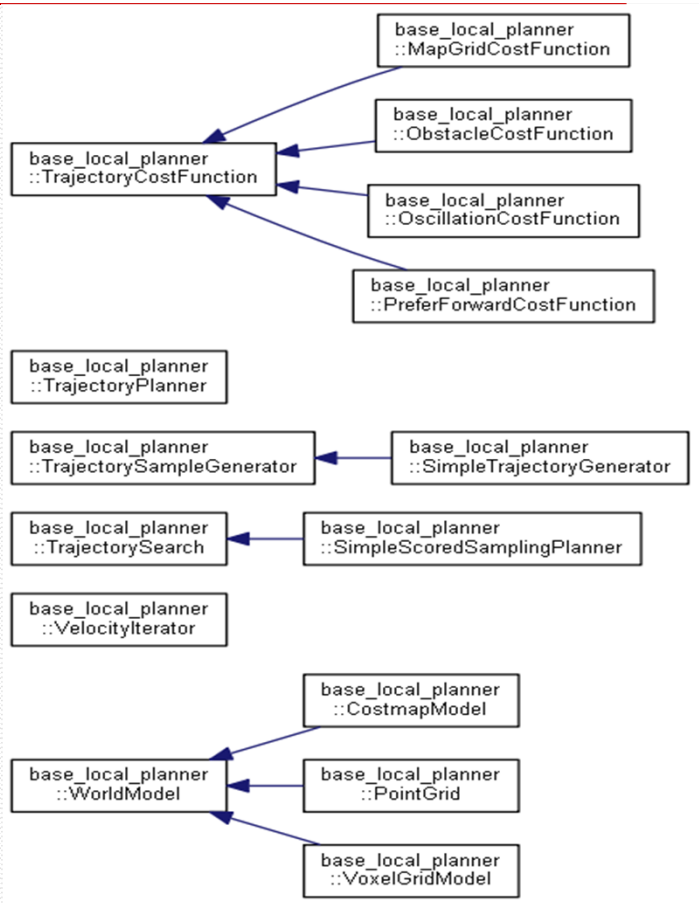
(30个头文件，1万多行)

- **base_local_planner包**

实现了Trajectory Rollout和DWA两种局部规划算法

- **dwa_local_planner包**

实现了DWA局部规划算法，可以看作是base_local_planner的改进版



11.3 Navigation-move base模块-nav_core

local_planner

- base_local_planner包
- Trajectory Rollout和Dynamic Window approaches算法
- 根据地图数据，通过算法搜索到达目标的多条路经，利用评价标准，选取最优的路径

11.3 Navigation-move base模块-nav_core

local_planner

Trajectory Rollout和Dynamic Window Approaches(DWA)
算法的主要思路如下：

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

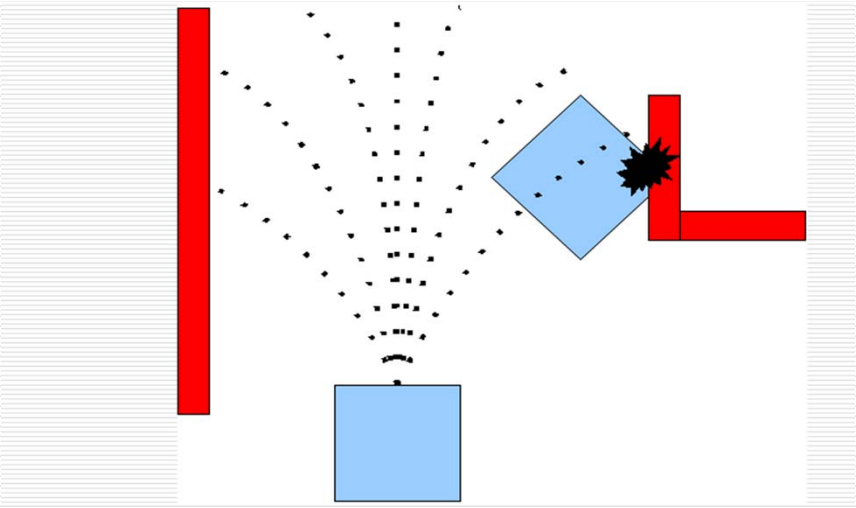
5. 重复上面过程

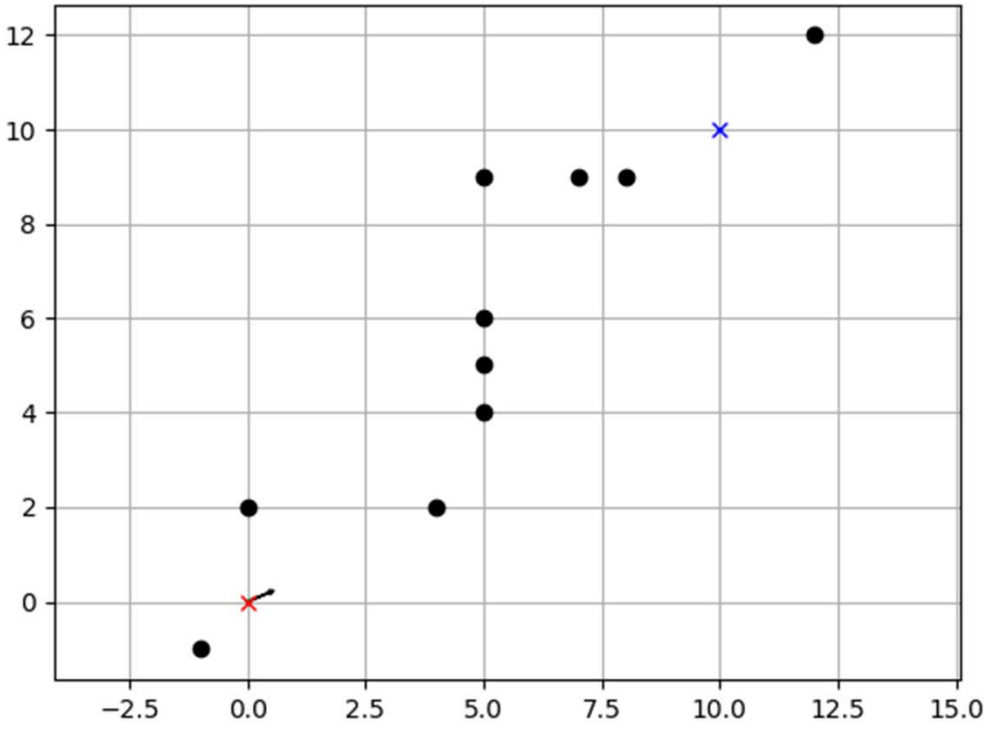
- **TrajectorySampleGenerator产生一系列轨迹**
- **然后TrajectoryCostFunction遍历轨迹打分**
- **TrajectorySearch找到最好的轨迹拿来给小车导航**
- **由于小车不是一个质点，worldModel会检查小车有没有碰到障碍物**

11.3 Navigation-move base模块-nav_core

local_planner

DWA算法演示图





11.3 Navigation-move base模块-nav_core

(3) base_global_planner插件

- **parrot_planner包**

实现了较简单的全局规划算法

- **Navfn包**

实现了Dijkstra和A*全局规划算法

- **global_planner包**

重新实现了Dijkstra和A*全局规划算法,可以看作navfn的改进版

11.3 Navigation-move base模块-nav_core

global_planner

- 计算出机器人到目标位置的全局路线
- navfn或global_planner插件实现
- navfn通过Dijkstra最优路径的算法，计算costmap上的最小花费路径
- global_planner根据给定的目标位置进行总体路径的规划，该实现相比navfn使用更加灵活

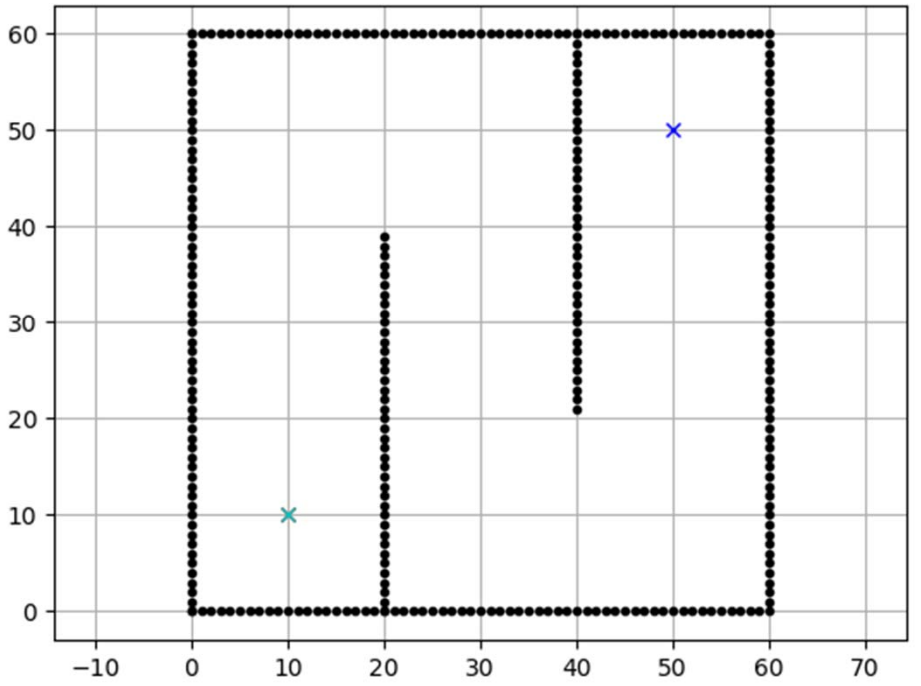
11.3 Navigation-move base模块-nav_core

global_planner Dijkstra

这是一张Dijkstra搜索的动态图

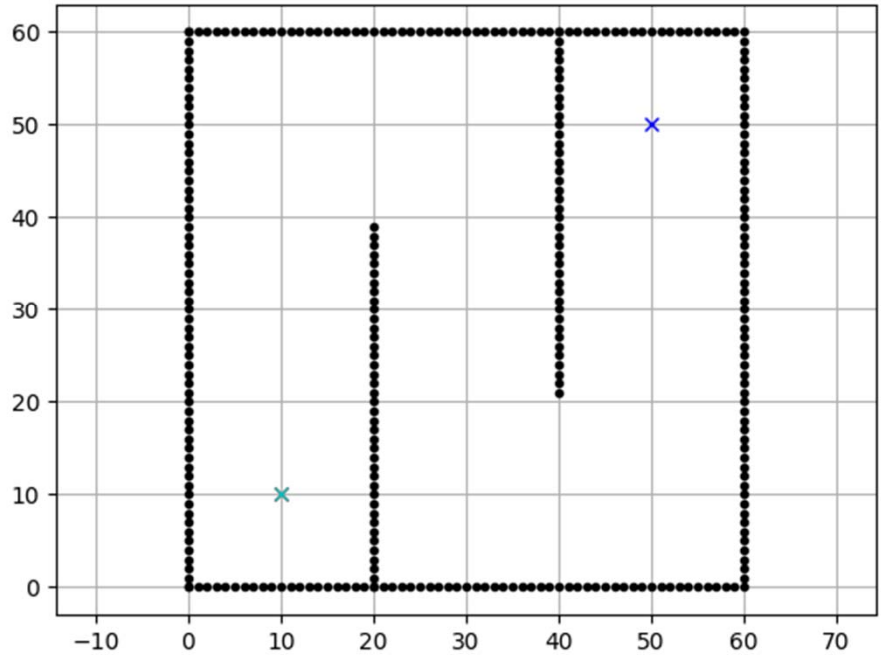
绿色的点是起始点，蓝色点是目标点

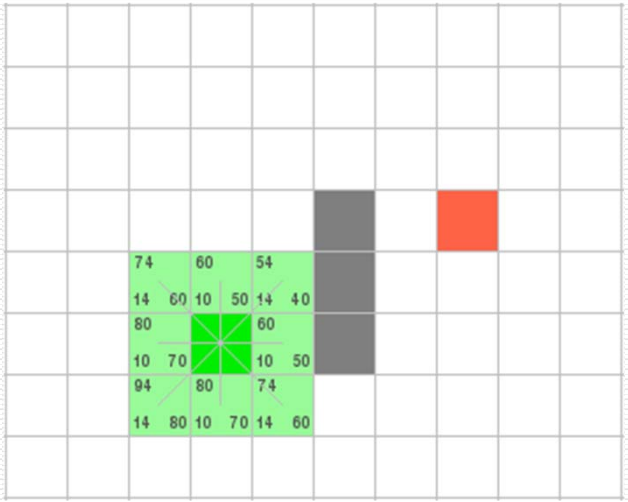
因为Dijkstra 算法是全局遍历，所以会以起始点为中心开始往周围搜索，直到蓝色的目标点被搜索到，停止搜索，然后从蓝色的目标点往绿色的起点提取结果，就可以得到红线连接的路径



11.3 Navigation-move base模块-nav_core

global_planner A*





11.3 Navigation-move base模块--nav_core

(4) recovery_behavior插件

- **clear_costmap_recovery**

  实现了清除代价地图的恢复行为

- **rotate_recovery**

  实现了旋转的恢复行为

- **move_slow_and_clear**

  实现了缓慢移动的恢复行为

11.3 Navigation-move base模块-costmap

costmap是Navigation Stack里的代价地图，它也是move_base插件，本质上是C++的动态链接库，用过catkin_make之后生成.so文件，然后move_base在启动时会通过动态加载的方式调用其中的函数。costmap也是occupancygrid类型，主要用于用于路径规划

地图就是/map这个topic，图片中的一个像素代表了实际的一块区域，用灰度值来表示障碍物存在的可能性。

在实际的导航任务中，光有一张地图是不够的，机器人需要能动态的把障碍物加入，或者清除已经不存在的障碍物，有些时候还要在地图上标出危险区域，为路径规划提供更有用的信息。

11.3 Navigation-move base模块-costmap

代价地图

Costmap是机器人收集传感器信息建立和更新的二维或三维地图

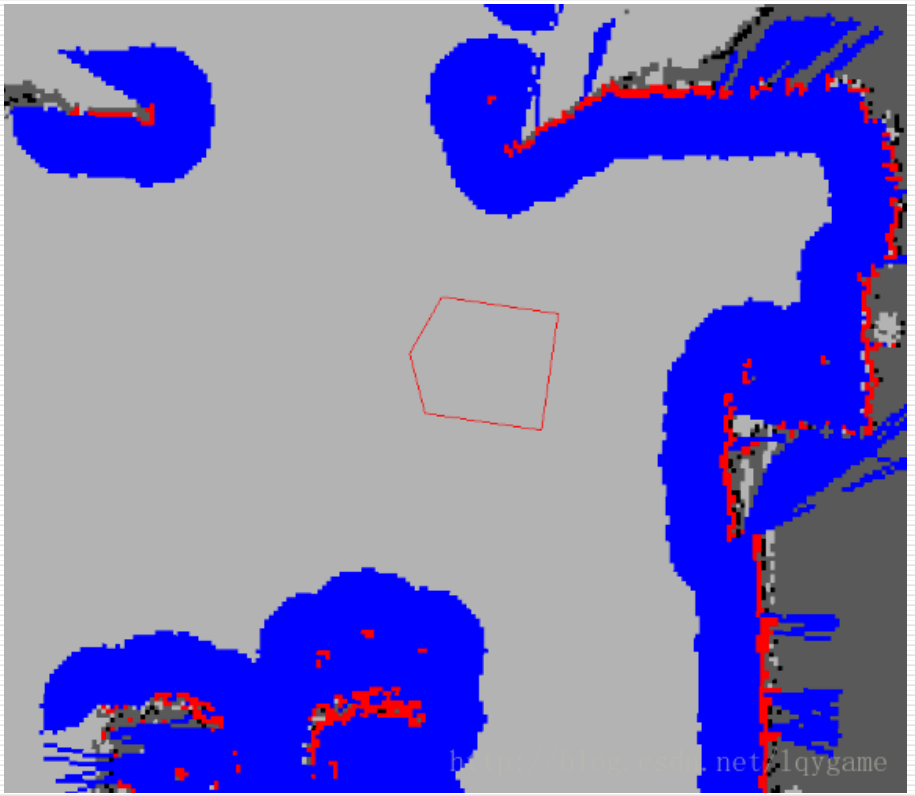
- 红色部分：障碍物
- 蓝色部分：通过机器人内切圆半径膨胀出的障碍
- 红色多边形：footprint(机器人轮廓的垂直投影)

为了避免碰撞，footprint不应该和红色部分有交叉，机器人中心不应该与蓝色部分有交叉。

11.3 Navigation-move base模块-costmap

代价地图

ROS的代价地图（costmap）采用网格（grid）形式，每个网格的值（cell cost）从0~255。分成三种状态：被占用（有障碍）、自由区域（无障碍）、未知区域。



11.3 Navigation-move base模块-costmap

ROS中costmap_2d这个包提供了一个可以配置的结构维护costmap

其中Costmap通过costmap_2d::Costmap2DROS对象利用传感器数据和静态地图中的信息来存储和更新现实世界中障碍物的信息，该对象为用户提供了纯粹的2维索引，这样可以只通过columns查询障碍物。

如一个桌子和一双鞋子在xy平面的相同位置，有不同的Z坐标，在costm_2d::Costmap2DROS目标对应的的costmap中，具有相同的cost值。这旨在帮助规划平面空。

11.3 Navigation-move base模块-costmap

costmap_2d包中**costmap的Layer包括以下几种：**

- Static Map Layer：静态地图层，通常都是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），以避免机器人的外壳会撞上障碍物
- Other Layers：你还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

11.3 Navigation-move base模块-costmap

Costmap

可以代价地图理解为，在/map之上新加的另外几层地图，不仅包含了原始地图信息，还加入了其他辅助信息。

１．首先，代价地图有两张，一张是local_costmap，一张是global_costmap，分别用于局部路径规划器和全局路径规划器，而这两个costmap都默认并且只能选择costmap_2d作为插件。

２．无论是local_costmap还是global_costmap，都可以配置他们的Layer，可以选择多个层次。

11.3 Navigation-move base模块-costmap

地图插件的选择

costmap配置用yaml 来保存，其本质是维护在参数服务器上。

由于costmap通常分为local和global的coastmap，我们习惯把两个代价地图分开

以ROS-Academy-for-Beginners为例，配置写在了param文件夹下的global_costmap_params.yaml和local_costmap_params.yaml里。

11.3 Navigation-move base模块-costmap

global_costmap_params.yaml:

```
global_costmap:
global_frame: /map robot_base_  // 全局frame
frame: /base_footprint          // frame
update_frequency: 2.0           // 更新频率
publish_frequency: 0.5          // 发布频率
static_map: true
rolling_window: false
transform_tolerance: 0.5                    // 容错率
plugins:   // 加载的插件
-   {name: static_layer, type: "costmap_2d::StaticLayer"}
-   {name: voxel_layer, type: "costmap_2d::VoxelLayer"}
-   {name: inflation_layer, type: "costmap_2d::InflationLayer"}
```

11.3 Navigation-move base模块-costmap

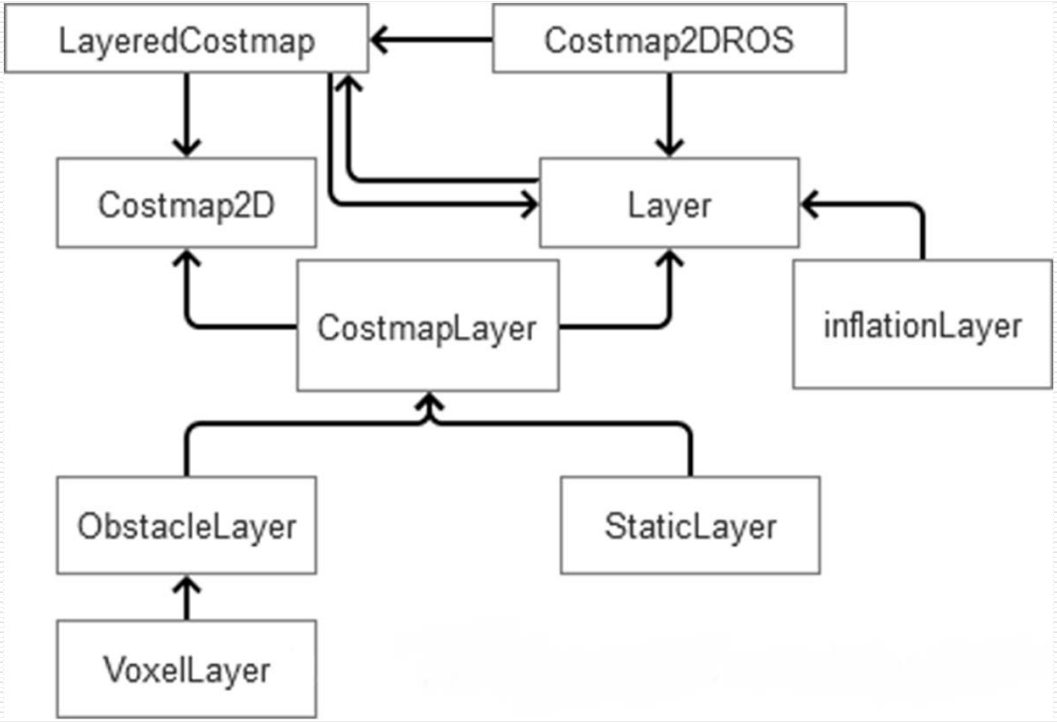
local_costmap_params.yaml

```
local_costmap:
global_frame: /map
robot_base_frame: /base_footprint
update_frequency: 5.0
publish_frequency: 2.0
static_map: false
rolling_window: true
width: 4.0
height: 4.0
resolution: 0.05
origin_x: 5.0
origin_y: 0
transform_tolerance: 0.5
plugins:
- {name: voxel_layer, type: "costmap_2d::VoxelLayer"}
- {name: inflation_layer, type: "costmap_2d::InflationLayer"}
```

11.3 Navigation-move base模块-costmap

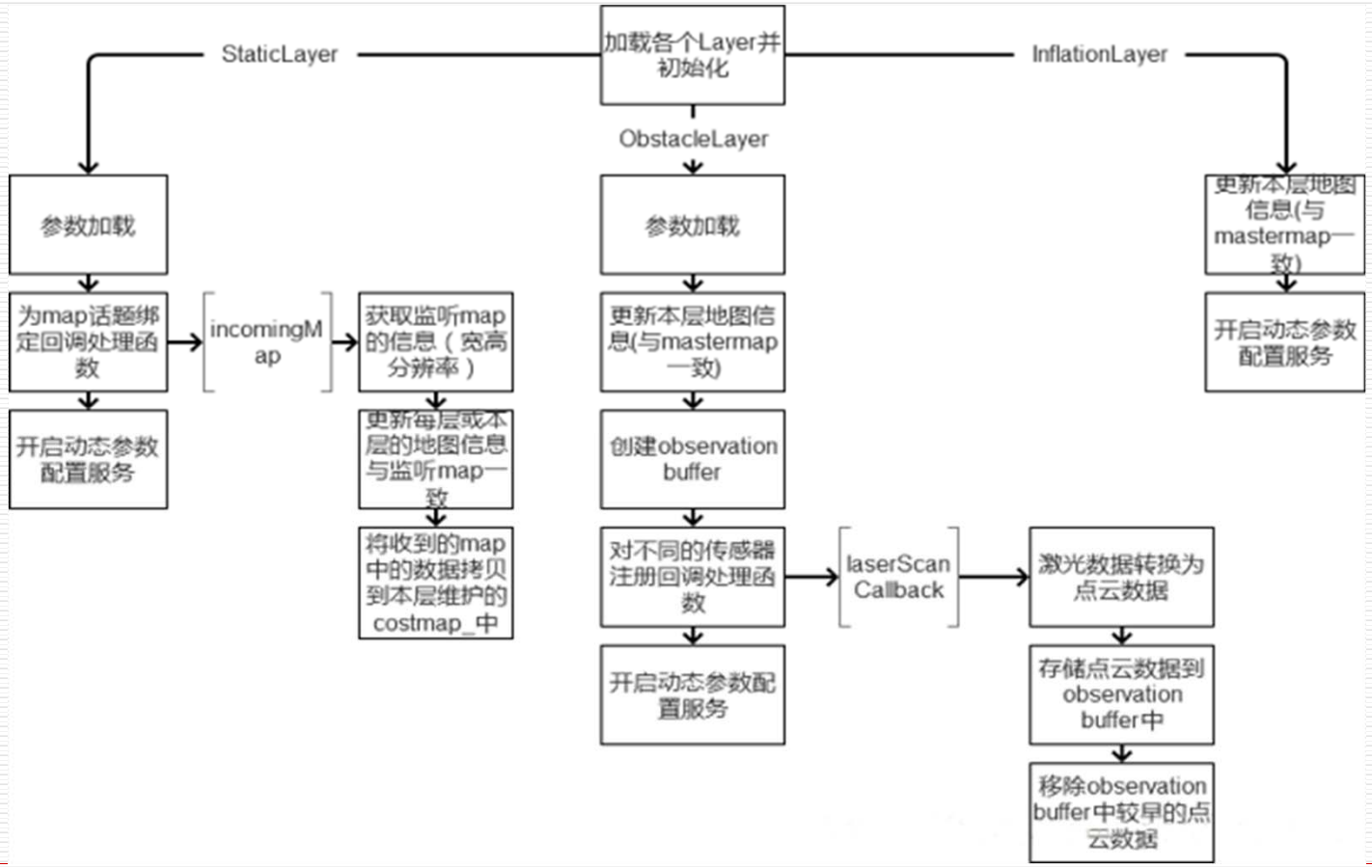
Costmap ROS接口

ROS对costmap进行了复杂的封装，提供给用户的主要接口是Costmap2DROS，而真正的地图信息是储存在各个Layer中。右图可以简要说明Costmap的各种接口的关系

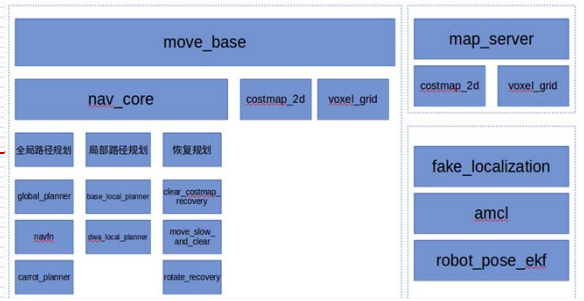


11.3 Navigation-move base模块-costmap

costmap程序架构



11.3 Navigation-mapsever



map_server提供map_server ROS节点，它提供地图数据作为一个ROS服务器，也提供map_saver命令行功能，能动态生成保存到文件中的地图。

简而言之，map_server将地图的数据变成 ros 的可以调用数据，即将已经建立好的地图（SLAM）提供给机器人。

11.3 Navigation-mapsever

map_server

发布topic:
/map (nav_msgs/OccupancyGrid)
/map_metadata (nav_msgs/MapMetaData)

提供service：
/static_map (nav_msgs/GetMap)

设置param：
~frame_id (string, default: "map")

11.3 Navigation-mapsever

map_server地图文件

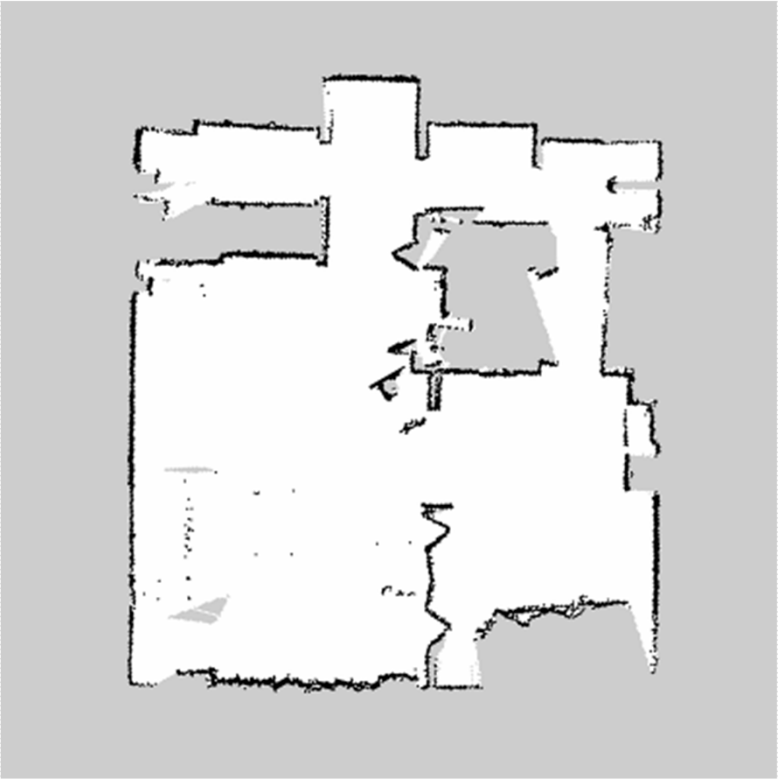
map_server地图文件以两种方式存储：

1、.pgm文件，它编码了地图的占据性情况。

2、.yaml文件，它存储了数据的元数据。

11.3 Navigation-mapsever

my_map.pgm



my_map.yaml

```
image: my_map.pgm
resolution: 0.050000
origin: [-25.000000, -25.000000,-25.000000]
negate: 0  #白/黑  自由/占据
occupied_thresh: 0.65  #高于则视为占据
free_thresh: 0.169  #低于则视为空
```

占据的概率

occ = (255 - color_avg) / 255.0

11.3 Navigation-mapsever

.yaml文件

YAML文件描述了地图元数据，并命名了图像文件。利用图像文件对占用数据进行编码。

```
image:my_map.pgm
resolution:0.050000
origin:[-25.000000, -25.000000, -25.000000]
negate:0 #白/黑  自由/占据
occupied_thresh:0.65 #高于则视为占据
free_thresh:0.169 #低于则视为空
```

- image: 指定包含occupancy data的image文件路径；可以是绝对路径，也可以是相对于YAML文件的对象路径
- resolution: 地图分辨率，单位是meters / pixel
- origin: The 2-D pose of the lower-left pixel in the map, 表示为 (x, y, yaw)，这里yaw是逆时针旋转角度(yaw=0意味着没有旋转)。目前多数系统忽略yaw值
- occupied_thresh: 像素的占用概率比该阈值大被看做完全占用
- free_thresh: 像素的占用概率比该阈值小被看做完全free
- negate:不论白色/黑色，自由/占用，semantics(语义/符号)应该被反转（阈值的解释不受影响）

11.3 Navigation-mapsever

map_server package

map_server package有两个节点：

1.map_server node：读取地图信息，并作为ROS service 为其余节点提供地图数据

2.map_saver node：保存现有扫描到的地图信息

11.3 Navigation-mapsever

map_server node

map_server是一个ROS node，可以从磁盘读取地图并使用ROS service提供地图。

目前实现的map_server可将地图中的颜色值转化成三种占用值： free (0), occupied (100), unknown (-1).未来可用0~100之间的不同值指示占用度。

map_server发布的topic：

1、map_metadata (nav_msgs/MapMetaData)

通过这个锁存topic来接受地图元数据(map metadata)

2、map (nav_msgs/OccupancyGrid)

通过这个锁存topic接收地图。

11.3 Navigation-mapsever

map_server node

MapMetaData.msg

```
# MapMetaData.msg包含有关OccupancyGrid特征的基本信息

# 加载地图的时刻
time map_load_time

# 地图分辨率 [m/cell]
float32 resolution

# 地图宽度 [cells]
uint32 width

# 地图高度 [cells]
uint32 height

# 地图原点 [m, m, rad].  对应真实世界框架的cell (0,0)
geometry_msgs/Pose origin
```

11.3 Navigation-mapsever

map_server node

OccupancyGrid.msg

```
# OccupancyGrid.msg
为二维网格图，其中每个单元格代表占用概率

#地图的MetaData
MapMetaData info

# 地图数据，按行序存储， 以 (0,0)开始
Occupancy

# 概率在[0,100]内, 未知为-1
int8[] data
```

11.3 Navigation-mapsever

map_server node

map_server提供的service：

- static_map（nav_msgs / GetMap）

  通过此服务检索地图。

map_server的parameters：

- ~frame_id（string，default: “map”）

  要在已发布地图的标题中设置的框架。

11.3 Navigation-mapsever

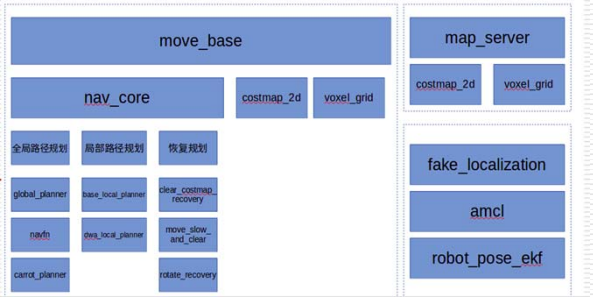
启动map_server，发布地图

```
$rosrun map_server map_server  my_map.yaml
```

保存地图

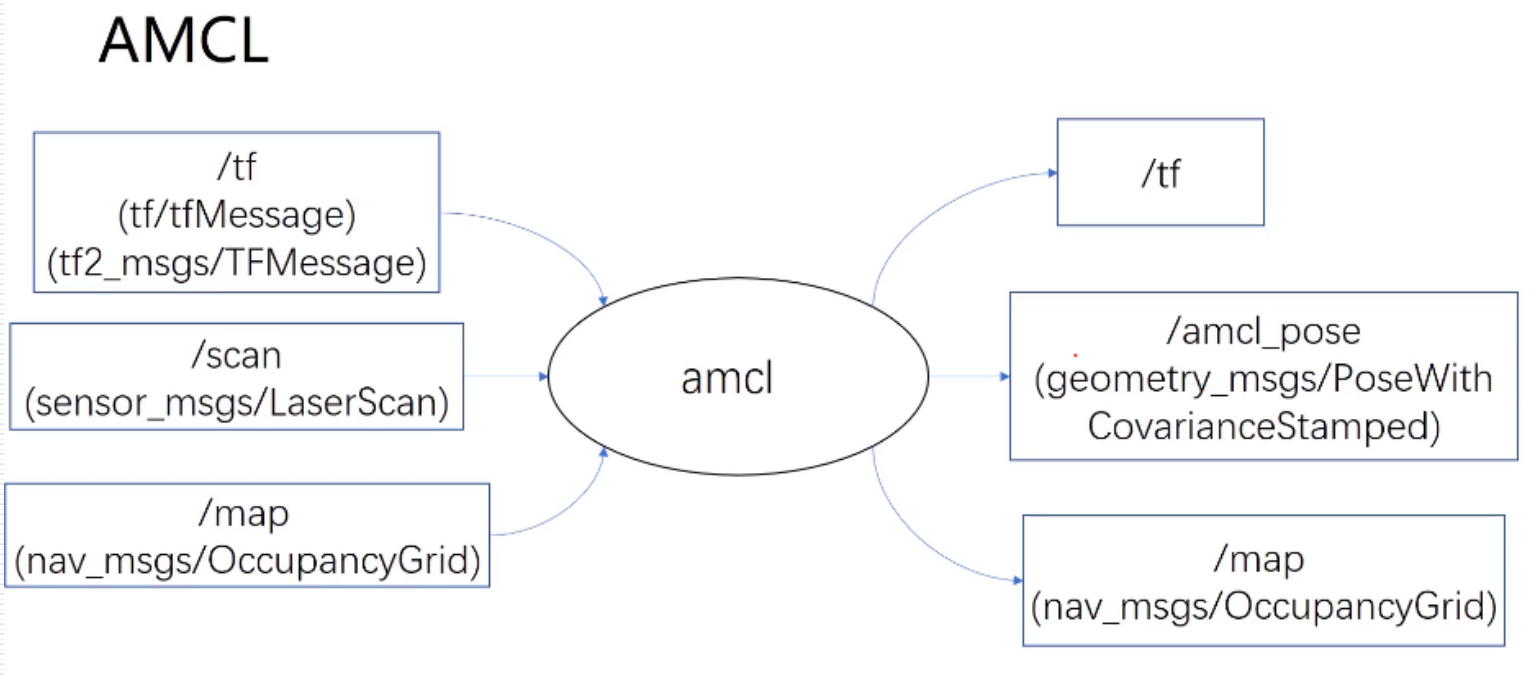
```
$ rosrun map_sever  map_saver  [-f my_map]
```

11.3 Navigation-AMCL



AMCL(adaptive Monte Carlo Localization)自适应蒙特卡洛定位，是机器人在二维移动过程中概率定位系统，采用粒子滤波器来跟踪已经知道的地图中机器人位姿，对于大范围的局部定位问题工作良好。对机器人的定位是非常重要的，因为若无法正确定位机器人当前位置，那么基于错误的起始点来进行后面规划的到达目的地的路径必定也是错误的.

11.3 Navigation-AMCL



11.3 Navigation-AMCL

定位的实现原理：

1、Odometry Localization

纯粹的里程计定位，会根据里程计的信息直接算出odom和base之间的偏移，默认map和odom绑在一起，会有累计误差

2、AMCL Map Localization

通过比对当前的地图和激光雷达实际扫到的地图，来修正自己的位置（Odometry drift）,把漂移补到odom和map两个frame之间

11.3 Navigation-AMCL