ROS 导航功能包调优指南*

原作：郑开宇†
翻译：罗辉武‡

版本：0.01
日期：2019 年 06 月 19 日§

摘　要

ROS 导航功能包适用于实现移动机器人可靠移动。ROS 导航功能包通过处理里程、传感器和环境地图的数据，为机器人运动生成一条安全的路径。最大限度地优化导航功能包的性能需要对参数进行调整，且调参这项工作并不像表面上的那么简单。对其中的概念和推理不熟悉的人很大概率会采用随机尝试的策略，无形中浪费了大量时间。

本文旨在通过调整导航参数的过程指导读者。当读者在设置关键参数的时候需要解答“如何设置”及“为什么这样设置”这些疑问时，本文可以作为一份参考材料。本文假设读者已对导航功能包进行了正确的设置，并准备进行优化。本文亦是我使用 ROS 导航功能包进行工作时的一份工作总结。

目录

*本文是 “ROS Navigation Tuning Guide’’ 的中文翻译。英文原版：http://kaiyuzheng.me/documents/papers/ros_navguide.pdf

†Email: kaiyu_zheng [at] brown [dot] edu
‡Email: huiwu.luo@aliyun.com

§更新：2019 年 04 月 08 日 (增加了 `amcl` 的内容)

1 速度和加速度

本节涉及同步驱动 (synchro-drive) 机器人。机器人的动力学特征 (例如，机器人的速度和加速度) 对于动态窗口法 (Dynamic Window Approach, DWA) 和定时弹性带 (Timed Elastic Band, TEB) 这两种局部规划器是必不可少的内容。在 ROS 的导航功能包中，局部规划器接收里程消息 (“odom" 话题)，输出控制机器人运动的速度指令 (“`cmd_vel`”话题)。

最大/最小速度和加速度是移动基座的两个基本参数。正确设置这两个数值对优化局部规划器的移动行为帮助非常大。在 ROS 的导航中，我们需要知道平移和旋转的速度及加速度。

1.1 获取最大速度

一般可以参考你的移动基座的手册。例如，SCITOS G5 的最大速度为 1.4 m/s[1]。在 ROS 中，你还可以订阅`odom`话题以获取当前里程信息。如果你能手工控制机器人（例如使用操纵杆），则你可以尝试向前移动直到它的速度值达到常数，然后回显 (echo) 里程数据。

平移速度 (m/s) 是机器人沿直线移动时的速度。获取其最大值的方式与上面获取速度的最大值方式相同。*旋转速度*（rad/s）等效于角速度; 它的最大值是机器人在原地位置旋转时的角速度。为获取最大旋转速度，我们可以通过操纵杆控制机器人，在机器人的速度达到恒定值后将机器人旋转 360 度，并对此次运动进行计时。

为安全起见，我们倾向于将最大平移速度和旋转速度设置为低于其实际最大值。

1.2 获取最大加速度

若您的手册没有直接告诉您该值大小，可通过很多方法测量您的移动基座的最大加速度。

在 ROS 中，类似地，我们可以回显包含时间戳的里程数据，记下机器人达到最大平移速度时所花的时间（t_i）。之后我们利用里程计的位置和速度信息 (`nav_msgs/Odometry`消息) 计算此过程中的加速度。多做几条不同的路径，计算其平均值。分别用 t_t 和 t_r 表示机器人从静止达到最大平移速度和最大旋转速度所需要的时间。则最大平移加速度为 $a_{t,max} = \max dv/dt \approx v_{max}/t_t$。同理，最大旋转加速度通过式 $a_{r,max} = \max d\omega/dt \approx \omega_{max}/t_r$ 计算。

[1]此信息来自于MetraLabs 网站。

1.3 设置最小值

设置最小速度不必像上述公式那么正式。对于最小平移速度，我们希望将其设置为较大的负值，因为这样可以让机器人在需要自救时进行回退，虽然在大多数实际情况下它应该前进。对于最小旋转速度，我们希望将其设置为负值（若参数允许的情况下），以便机器人可以在任一方向上旋转。注意，DWA 局部规划器采用的是机器人最小旋转速度的绝对值。

1.4 x,y 方向上的速度

x **velocity** 表示平行于机器人直线运动方向的速度。它与平移速度相同。*y* **velocity** 是垂直于该直线运动方向的速度。在 `teb_local_planner` 中称为“扫射速度”(strafing velocity)。对于非完整类型 (non-holonomic) 的机器人（例如差动轮式机器人），应将 *y* velocity 设置为零。

2 全局规划器

2.1 全局规划器的选择

要在导航堆栈中使用 `move_base` 节点，需要配置一个全局规划器和一个局部规划器。ROS 有三个全局规划器继承自 `nav_core::BaseGlobalPlanner` 接口：`carrot_planner`，`navfn` 和 `global_planner`。

2.1.1 carrot_planner

这是最简单的全局规划器。它检查给定的目标点是否在障碍区，如果目标点在障碍区，则沿机器人和目标点之间连线，在连线上选择一个最接近原始目标点的有效目标点。最终，它将该有效目标点直接传递给局部规划器或内容控制器。所以，该规划器并没有进行任何的规划。它在你需要移动机器人到给定目标点而且目标点无法到达的情况下很有用。对于复杂的环境，该规划器并不实用。

2.1.2 navfn 和 global_planner

`navfn` 采用 Dijkstra 算法在起点和终点之间寻找一条最小代价的路径。`global_planner` 是 `navfn` 的灵活性替代，提供了更多的选项，包括：(1) 支持 A∗, (2) 二次近似的切换, (3) 栅格地图的切换。`navfn` 和 `global_planner` 实现的内容都是基于[1] 的工作。

2.2 全局规划器的参数

通常情况下我们更喜欢使用 `global_planner`，现在来看看它的一些关键参数。注意：并非所有的参数都在 ROS 的官网上罗列出来，但如果你运行 rqt 动态配置程序时你可以看到它们：

```
rosrun rqt_reconfigure rqt_reconfigure
```

可以让 `allow_unknown(true)`，`use_dijkstra(true)`，`use_quadratic(true)`，`use_grid_path(false)`，`old_navfn_behavior(false)` 保留它们的默认参数。当需要在 rviz 可视化候选地图时将 `visualize_potential(false)` 设置为 `true` 会带来很多帮助。

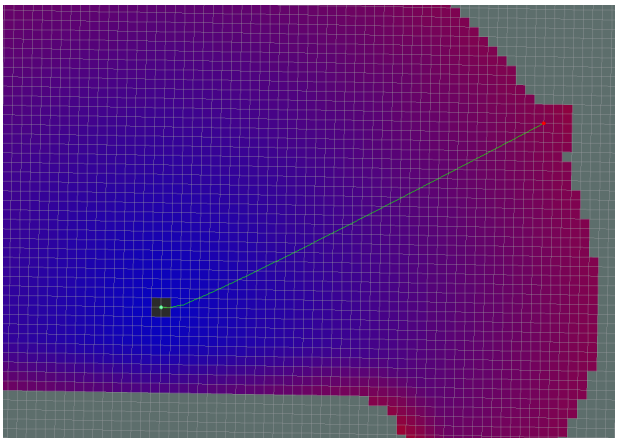

图 1: Dijkstra 路径

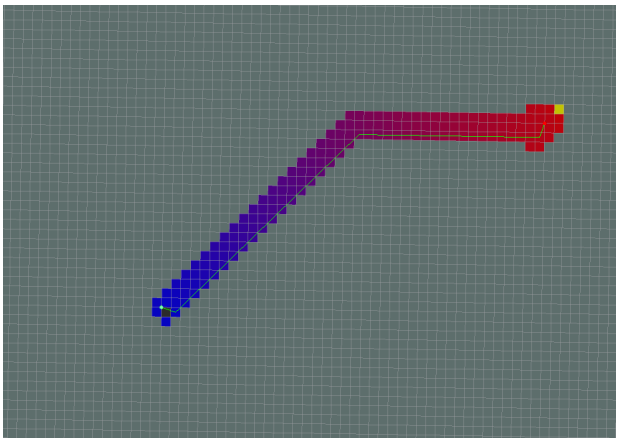

图 2: A* 路径

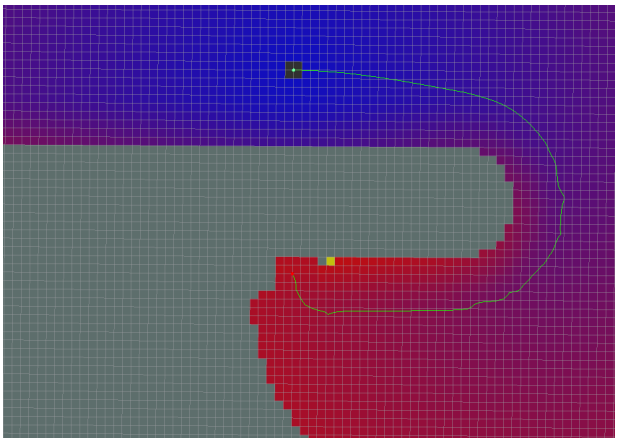

图 3: 标准行为

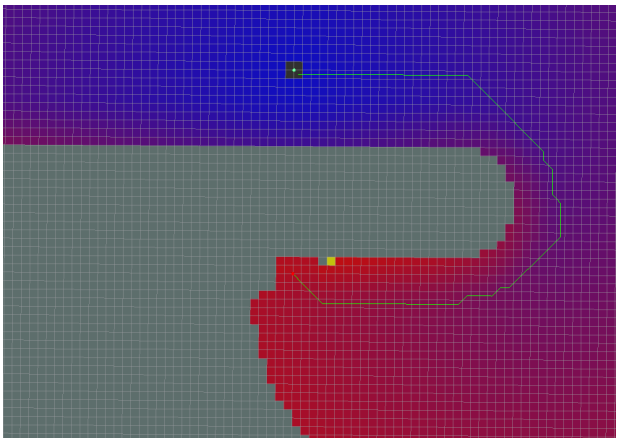

图 4: 栅格路径

除去这三个参数外，还有另外三个未列出的参数对规划全局路径的质量造成很大的影响。它们是 `cost_factor`，`neutral_cost`，`lethal_cost`。实际上，这些参数也存在于 `navfn` 的配置中。源程序[2]有一段描述解释了 `navfn` 是如何计算代价值的。

`navfn` 的代价值设置为：

```
cost = COST_NEUTRAL + COST_FACTOR * costmap_cost_value.
```

传入 costmap 的代价值的范围为 0 到 255。注释中提到：

> 当 `COST_NEUTRAL` 为 50 时，`COST_FACTOR` 需要设置为大约 0.8 以确保输入值均匀分布到 50 到 253 的输出范围内。如果 `COST_FACTOR` 设置为更高的值，则代价值将稳定到障碍物的高度，规划器会不重视狭窄走廊的宽度，使得规划的路线并没有沿着中心下降。

实验情况 该段解释在实验中得到了印证。将 `cost_factor` 设置得太低或太高都会降低路径的质量。这些路径并不会穿过每侧障碍物的中间，并具有相对低较的曲率。`neutral_cost` 的极端值具有相同的效果。对于 `lethal_cost`，将其设置为较低的值可能导致无法生成任何路径，即使可行路径仍非常明显。图

[2] https://github.com/ros-planning/navigation/blob/indigo-devel/navfn/include/navfn/navfn.h

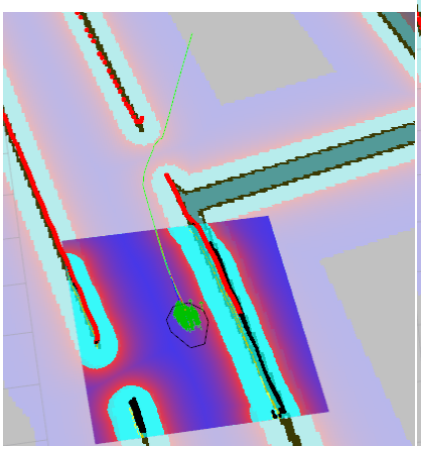

图 5: `cost_factor` = 0.01

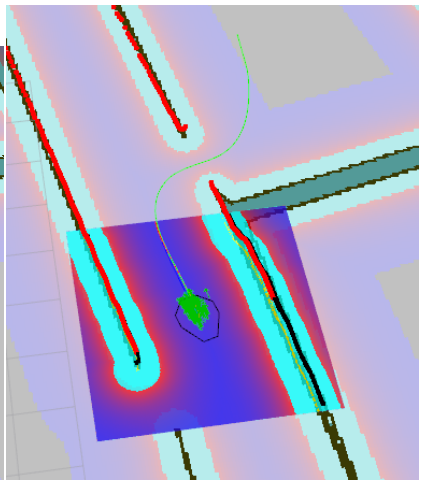

图 6: `cost_factor` = 0.55

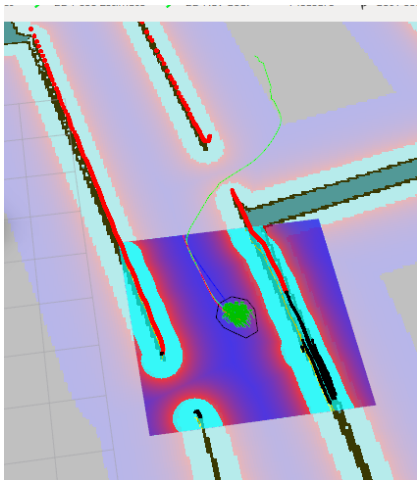

图 7: `cost_factor` = 3.55

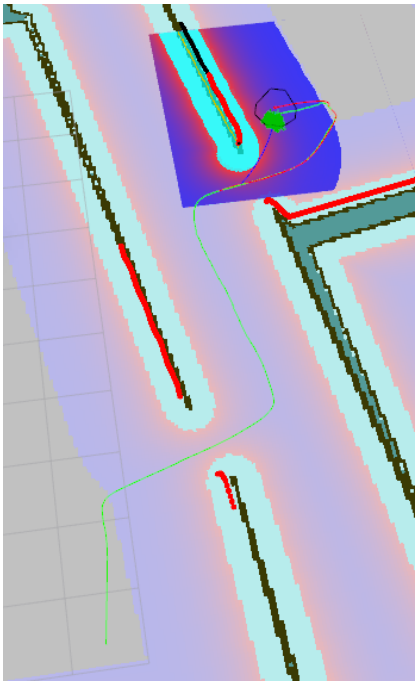

图 8: `neutral_cost` = 1

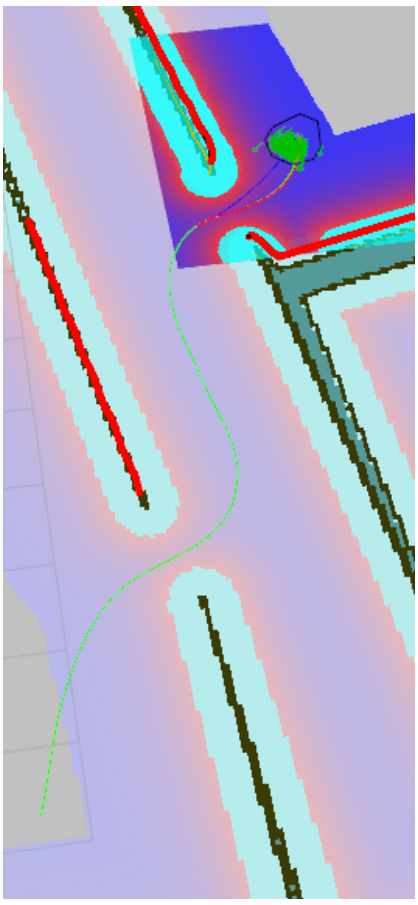

图 9: `neutral_cost` = 66

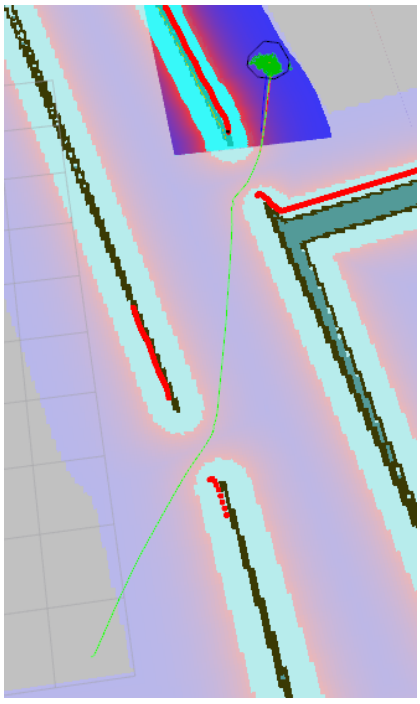

图 10: `neutral_cost` = 233

5–10 显示了 `cost_factor` 和 `neutral_cost` 在全局规划任务上的效果。其中绿线是 `global_planner` 规划的全局路径。

经过几次试验后，我们看到，当 `cost_factor` =0.55, `neutral_cost = 66` , 和 `lethal_cost = 253` 时，生成的全局路径是意向中的路径。

3 局部规划器的选择

继承自 `nav_core::BaseLocalPlanner` 的局部规划器的接口为 `dwa_local_planner`, `eband_local_planner` 和 `teb_local_planner`。它们使用不同的算法生成速度指令。一般而言，首选 `dwa_local_planner`。我们将对它进行详细讨论。以后将提供其他规划器的更多信息。

3.1 DWA 局部规划器

3.1.1 DWA 算法

`dwa_local_planner` 使用动态窗口 (Dynamic Window Approach, DWA) 算法。ROS 的 Wiki 页面提供了该算法的实现总结：

> 1. 在机器人的控制空间 *(dx,dy,dtheta)* 进行离散采样。
> 2. 对于每个采样速度，从机器人的当前状态执行正向模拟，预测使用该采样速度在某个（短）时间段内会发生什么情况。
> 3. 使用含有以下因素的度量来评估（得分）每个前向模拟产生的轨迹：与障碍物的接近度，与目标点的接近度，全局路径的贴合度，及所使用的速度。丢弃不合格的轨迹（与障碍物发生碰撞的轨迹）。
> 4. 选择得分最高的轨迹并将相关的速度指令发送到移动基座。
> 5. 冲洗 (Rinse) 并重复该过程。

DWA 由[2] 提出。根据论文描述，DWA 的目标是产生一对 (v,ω) 对，表示最适合机器人局部条件的圆形轨迹。DWA 通过在下一个时间间隔中搜索速度空间来达到此目标。该空间中的速度被限制为允许的，这意味着机器人必须能够在到达由这些允许的速度决定的圆形轨迹上的最近障碍物之前停止。此外，DWA 将仅考虑动态窗口内的速度，动态窗口被定义为在给定当前平移和旋转速度和加速度的情况下在下一时间间隔内可到达的速度对的集合。DWA 最大化目标函数，其取决于（1）到目标的进展，（2）从障碍物的清除，以及（3）前向速度以产生最佳速度对。

我们看看 ROS Wiki 上的算法概要。第一步对在位于动态窗口内的速度空间中对速度对 (v_x, v_y, ω) 进行采样。第二步基本上是剔除不可用的速度 (即消除不好的轨迹)。第三步是利用目标函数评估得到的速度对，产生*轨迹分数*。第四步和第五步很容易理解：选择对当前状态最好的速度再重新计算。

DWA 规划器依赖于提供障碍信息的代价图。因此，调整局部代价图的参数对于优化 DWA 局部规划器的行为至关重要。下面，我们通过前向模拟、轨迹评分和代价图等看看相关的参数。

3.1.2 DWA 局部规划器：前向模拟

前向模拟是 DWA 算法的第二步。在该步中，局部规划器将在机器人的控制空间中采样速度，检查由这些速度样本表示的圆形轨迹，最后消除不良速度 (轨迹与障碍物相交的速度)。在设置的时间间隔内，每个由参数 `sim_time` (单位为 *s*) 速度被模拟为真实地应用到机器人上。我们可以认为 `sim_time` 是机器人在给定的采样速度下允许运行的时间。

通过实验，我们观察到仿真时间 `sim_time` 越长，计算负荷越大。此外，当仿真时间 `sim_time` 变大后，局部路径规划器产生的路径也会相应变长，这种现象是在合理范围内的。这里有一些关于如何调整仿真时间参数 `sim_time` 的建议。

如何调整 `sim_time` 如果将仿真时间 `sim_time` 的值设置为非常低（例如 ≤ 2.0）将导致性能下降，尤其是当机器人需要通过狭窄的门口、或家具之间的间隙时。这是因为没有足够多的时间进行计算以获得最佳轨迹以便通过狭窄的通道。另一方面，由于使用了 DWA 规划器，所有的轨迹都是简单的圆弧，如果将仿真时间 `sim_time` 设置的非常高（≥5.0），将得到一条很长的缺乏灵活性能的曲线。该问题并不是不可避免的，因为规划器在每个时间间隔后都会主动进行重规划 (由参数 `controller_frequency` (单位为 Hz) 控制)，它可以对结果进行一次较小的调整。对于性能较高的计算机，4.0 秒的计算时间足矣。

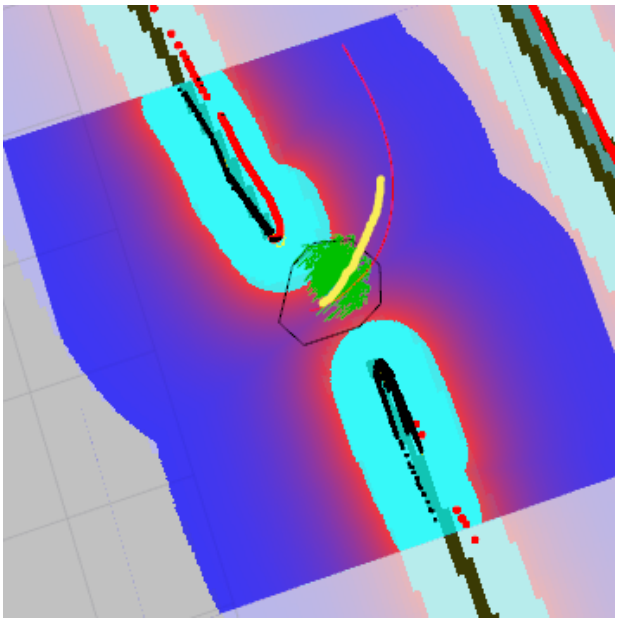

图 11: `sim_time` = 1.5

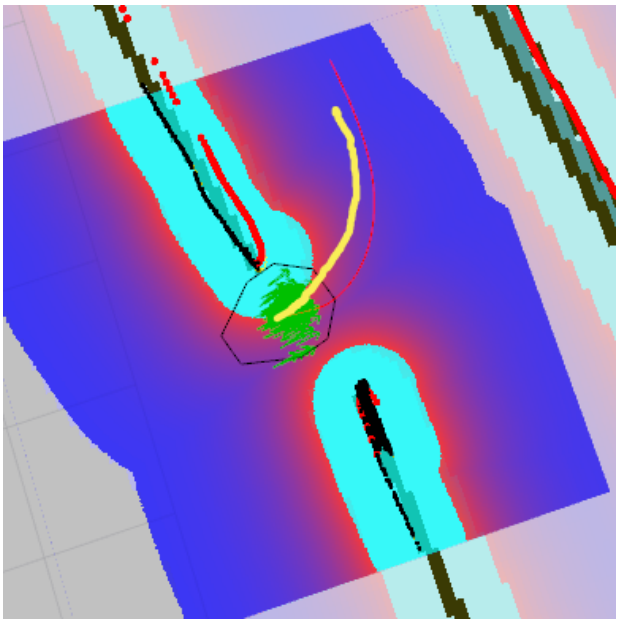

图 12: `sim_time` = 4.0

除了仿真时间，还有几个参数值得注意。

速度采样：在其它几个参数中，`vx_sample` 和 `vy_sample` 分别决定在 x 和 y 方向上进行速度采样时平移部分的量。`vth_sample` 控制速度采样中旋转部分的量。速度采样的数量取决于你的设备的计算能力。多数情况下，我们倾向于设置 `vth_samples` 的值大于平移部分的值，原因在于通常情况下旋转要比直线前进更为复杂。如果将 `max_vel_y` 设置为零，因为没有可用信息所以没必要在 y 方向提取速度样本。我们设置 `vx_sample` = 20, `vth_samples` = 40。

仿真粒度： `sim_granularity` 意为在轨迹上的采样点之间的步长。该参数含义为需要多频繁的检查轨迹上的点（检测它们是否与障碍物相交）。较低的值意味着高频率，当然需要更多的计算性能。对于 turtlebot 这类机器人来说，默认值 0.025 足够胜任。

3.1.3 DWA 局部规划器：轨迹得分

如上所述，DWA 规划器通过最大化目标函数来获得最佳速度对。在 DWA 的论文中，目标函数的值依赖于三部分：到目标点的过程、清除障碍物和前进速度。在 ROS 中，目标函数的计算公式如下：

$$
\begin{aligned}
cost &= \texttt{path\_distance\_bias} * (\text{distance}(m) \text{ to path from the endpoint of the trajectory}) \\
&+ \texttt{goal\_distance\_bias} * (\text{distance}(m) \text{ to local goal from the endpoint of the trajectory}) \\
&+ \texttt{occdist\_scale} * (\text{maximum obstacle cost along the trajectory in obstacle cost (0-254)})
\end{aligned}
$$

目标函数的意义在于获得最小的行走代价。`path_distance_bias` 为局部规划器以多大权重与全局路径保持一致[3]。较大值将使局部规划器倾向于跟踪全局路径。

`goal_distance_bias` 衡量机器人无论走哪条路径应该以多大权重尝试到达目标点。实验显示增加 `goal_distance_bias` 的值会使机器人与全局路径的一致性保持度偏低。

`occdist_scale` 是权衡机器人以多大的权重躲避障碍物。该值过大会导致机器人陷入困境。

在 SCITOS G5 上，我们设置 `path_distance_bias` 为 32，`goal_distance_bias` 为 20，`occdist_scale` 为 0.02，仿真结果良好。

3.1.4 DWA 局部规划器：其他参数

目标距离容差 (Goal distance tolerance)：这些参数很直观，以下为它们在 ROS Wiki 上的描述：

- `yaw_goal_tolerance` (double, 默认值: 0.05，单位：弧度) 到达目标点时，偏航/旋转中控制器的弧度容差 (tolerance)。
- `xy_goal_tolerance` (double, 默认值: 0.10，单位：米) 到达目标点时，控制器在 x & y 方向的距离容差，单位为米。
- `latch_xy_goal_tolerance` (bool, 默认值: false) 目标容差被锁定情况下，机器人到达目标的 xy 位置时会简单旋转到位；如果机器人在目标容差 (goal tolerance) 范围之外进入结束状态时，也会简单旋转到位。

振荡复位（Oscilation reset）：在诸如通过门口情况下，机器人可能会来回振荡，因为其局部规划器正在产生通过两个相反方向的路径。如果机器人保持振荡，导航功能包将让机器人尝试恢复行为。

- `oscillation_reset_dist` (double, 默认值: 0.05，单位：米) 在振荡标志复位前，机器人必须以米为单位行走多远。

4 代价地图参数

如上所述，代价地图参数对于本地规划器（不仅仅是 DWA）是至关重要的。在 ROS 中，代价地图由静态地图层、障碍物图层和膨胀层组成。静态地图层直接给导航堆栈提供静态 SLAM 地图解释。障碍物图层包含 2D 障碍物和 3D 障碍物（体素层）。膨胀层是将障碍物膨胀来计算每个 2D 代价地图单元的代价。

如上所述，调整 *costmap* 代价地图参数对于局部规划器（不仅仅是 DWA）的成功至关重要。在 ROS 中，costmap 由静态地图层 (static map layer)，障碍地图层 (obstacle map layer) 和膨胀层 (inflation layer) 组成。静态地图层对提供给导航功能包的静态 SLAM 地图直接进行解释。障碍层包括有 2D 的障碍物和 3D 的障碍物（体素层）。膨胀层将障碍物进行膨胀以计算每个 2D 代价单元 (cell) 的通行代价。

此外，还存在全局代价地图 (*global costmap*)，和局部代价地图 (*local costmap*)。全局代价地图通过膨胀导航功能包的地图上已存在的障碍物产生。局部代价地图通过将机器人传感器检测到的障碍物进行膨胀产生的。

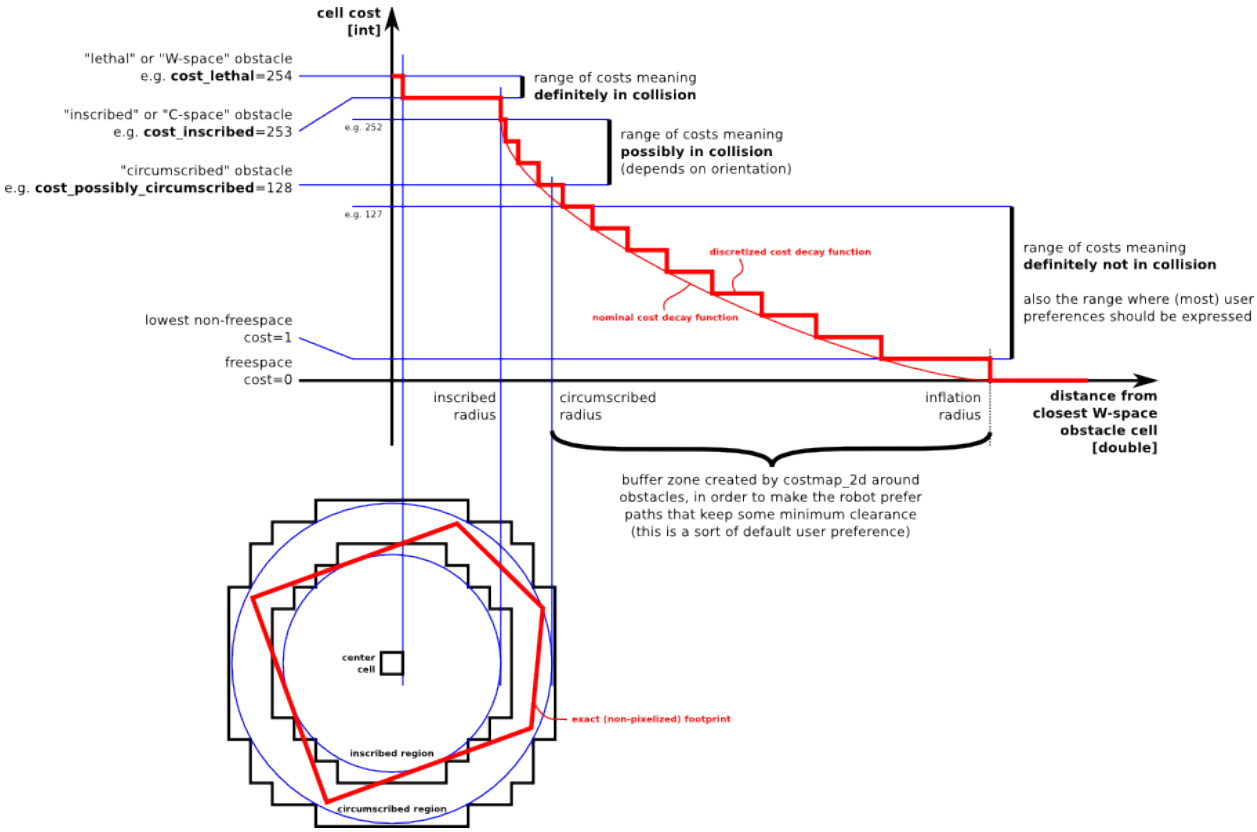

图 13: 膨胀衰减

此外还有很多重要的参数应尽可能设置好。

4.1 足印

足印 (footprint) 是机器人移动本体的轮廓。在 ROS 中，footprint 由二维数组表示 $[x_0, y_0], [x_1, y_1], [x_2, y_2], ...]$，无需重复第一个坐标。footprint 用于计算内切圆和外接圆的半径，以确定适合该机器人的方式对障碍物进行膨胀。为安全起见，我们通常将 footprint 设置为略大于机器人的真实轮廓。

要确定机器人的 footprint，最直接的方法是参考机器人的图纸。此外，您可以手工拍摄其基座俯视图。然后使用 CAD 软件（例如 Solidworks）适当缩放图像，并将鼠标移动到基座轮廓周围并读取其坐标。坐标原点应为机器人的中心。或者，您可以将机器人移动到一张大纸上，然后绘制基座的轮廓。然后选择一些顶点并使用标尺来确定它们的坐标。

4.2 膨胀

膨胀层 (Inflation layer) 由代价值位于 0 到 255 的单元 (cell) 组成。每个单元可能会被占据 (occupied)、无障碍或未知三种情况。图 13 的图表[3]说明了如何计算膨胀衰减曲线的过程。

`inflation_radius` `cost_scaling_factor` 是决定膨胀的主要参数。`inflation_radius` 控制零代价点距离障碍物有多远。`cost_scaling_factor` 与单元 (cell) 的代价值成反比，设置高值将使衰减曲线更为陡峭。

Pronobis 博士认为，最佳代价图的衰减曲线应为斜率相对较低的曲线，这样可使得最佳路径尽可能远离每侧的障碍物。优点是机器人更趋向于在障碍物中间移动。如图 8 和图 9 所示，在具有相同的起点和目标点情况下，当代价图的曲线非常陡峭时，机器人往往会靠近障碍物。在图 14 中，膨胀半径 `inflation_radius` = 0.55，代价比例因子 `cost_scaling_factor` = 5.0；在图 15 中，膨胀半径 `inflation_radius` = 1.75，代价比例因子 `cost_scaling_factor` = 2.58。

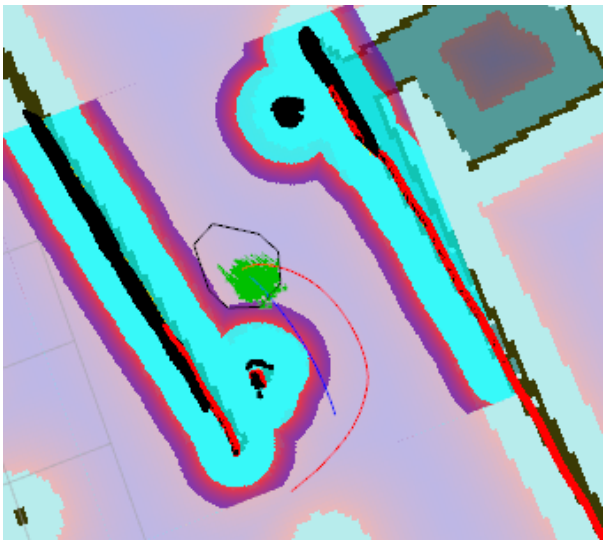

图 14: 陡峭的膨胀曲线

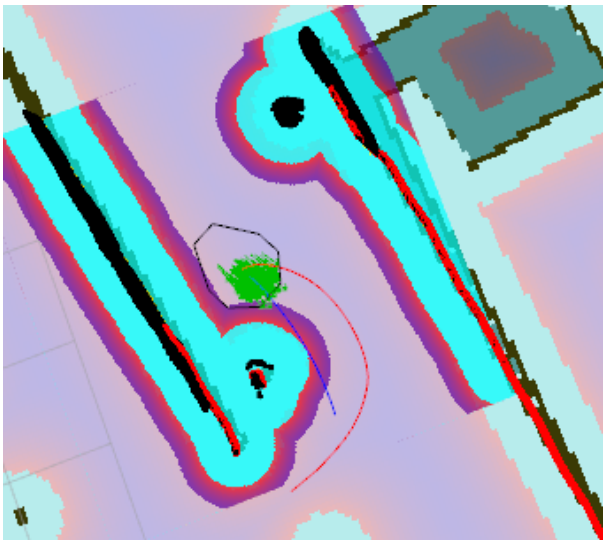

图 15: 温和的膨胀曲线

根据衰变曲线图，我们需设定这两个参数值，以使得膨胀半径几乎覆盖走廊，并且代价值的衰减速度相对中等，这意味着要降低代价的比例因子 `cost_scaling_factor` 的值。

4.3 代价地图的分辨率

可以在局部代价图和全局代价图单独设置此参数。它们影响着计算设备的计算负荷和规划器的路径规划能力。对于低分辨率 (>= 0.05)，在狭窄的通道中，障碍物区域可能会重叠，导致局部规划器无法找到可通行路径。

对于全局代价图的分辨率，只要其与导航功能包的地图的分辨率保持相同即可。如果有足够的计算能力，可以查看激光扫描仪的分辨率，因为当使用 gmapping 建图时，如果激光扫描仪的分辨率低于所需的地图分辨率，则会有很多小的“未知点”，因为激光扫描仪无法覆盖该区域，如图 10 所示。

例如，Hokuyo URG-04LX-UG01 激光扫描仪的分辨率是 0.01mm[4]，因此建立扫描分辨率为 ≤ 0.01

[3]图表来自http://wiki.ros.org/costmap_2d

[4]数据来源：https://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_UG01_spec_en.pdf

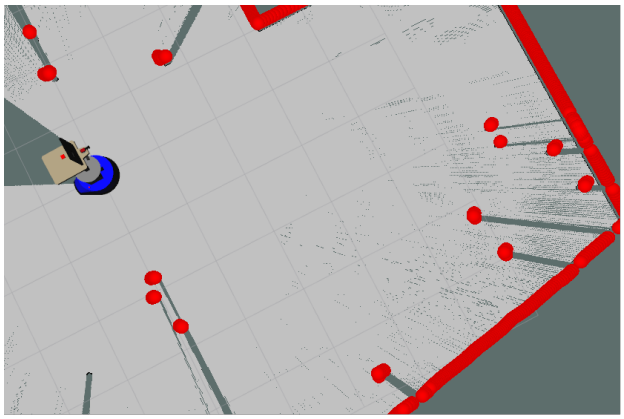

图 16: gmapping 分辨率 = 0.01。注意图像右侧的未知点

的地图需要多旋转机器人几次才能清除未知的点。可以发现，0.02 的分辨率就够用了。

4.4 障碍物层和体素层

这两个图层负责在代价图上标注障碍物。它们可以被称为障碍层 (*obstacle layer*)。根据 ROS wiki 的介绍，障碍层在两个维度上进行跟踪，而体素层在三个维度上进行跟踪。障碍物是根据机器人传感器的数据进行标记（检测）或清除（删除）建立的，同时为代价图提供了订阅的主题 (topics)。

在 ROS 的实现代码中，体素层继承自障碍物层，两者皆为通过解释激光扫描 (laser scans) 或使用 `PointCloud` 和 `PointCloud2` 类型的消息发送数据获取障碍物的信息。此外，体素层需要深度传感器，如 Microsoft Kinect 或 ASUS Xtion。3D 障碍物最终会被膨胀为二维代价图。

体素层如何工作：体素是空间中具有一定相对位置的 3D 立方体（想象为 3D 像素）。它用于与其附近的数据或属性进行关联。例如，它的位置是否为一个障碍。与体素与深度相机相关的 3D 重建已经有很多研究了。已经存在很多通过深度相机获取体素进行在线 3D 重建的研究了。这是其中的一部分。

- KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera
- Real-time 3D Reconstruction at Scale using Voxel Hashing

体素网格 `voxel_grid` 是一个 ROS 包，它提供了一个高效的三维体素网格数据结构的实现，该存储了体素的三种状态：标记状态 (marked)、自由状态 (free)、未知状态 (unknown)。`voxel_grid` 占据了代价地图区域内的体积。在每次更新体素边界期间，体素层根据传感器的数据来标记或移除体素网格中的一些体素。它还执行光线跟踪 (ray tracing) 作用，该内容将在接下来进行讨论。请注意，体素层在更新时，不会重新创建体素网格；仅在更改局部代价图的尺寸大小时体素层才进行更新。

为什么要在障碍物层或体素层进行光线跟踪？光线跟踪以渲染逼真的 3D 图形而闻名，因此读者可能会对为什么将它用于处理障碍物这个问题而倍感困惑。一个重要的原因是可以通过机器人的传感器检测不同类型的障碍物。请观察图 17。理论上，我们还可以知道障碍物是刚性的还是柔性的 (例如草)[5]。

一个关于体素光线跟踪对比多声源追踪的好博客:http://raytracey.blogspot.com/2008/08/voxel-ray-tracing-vs-polygon-ray.html

通过以上的解释，我们来研究一下障碍物层的一些参数[6]。这些参数属于全局滤波参数，对所有的

[5]出处为：*Using Robots in Hazardous Environments*, Boudoin, Habib, pp.370。

[6]个别解释直接拷贝自 ROS Wiki 对 costmap2d 的介绍

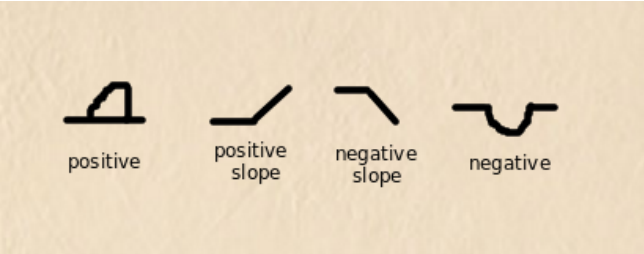

图 17: 通过光线跟踪，激光扫描仪能够识别不同类型的障碍物。

传感器都是适用的。

- `max_obstacle_height`：以米为单位插入代价图中的障碍物的最大高度。该参数应设置为略高于机器人的最大高度。对于体素层，这本质含义是指体素网格的高度。
- `obstacle_range`：插入代价地图的障碍物应距离机器人的最大默认距离，以米为单位。它可以在每个传感器的基础上进行覆盖操作。
- `raytrace_range`：使用传感器数据从地图中光线追踪障碍物的默认范围（以米为单位）。同样，该值可以在每个传感器的基础上被覆盖。

下面的这些参数仅适用于体素层 (VoxelCostmapPlugin)

- `origin_z`：地图的 Z 轴原点，以米为单位
- `z_resolution`：地图的 Z 轴分辨率，以*米/单元格* (meters/cell) 为单位。
- `z_voxels`：每个垂直列中的体素数目，网格的高度为 `z_resolution * z_voxels`。
- `unknown_threshold`：The number of unknown cells allowed in a column considered to be "known" 整列认为是“已知”(“known”) 的时候，含有的未知单元 (“unknown”) 的最大数量。
- `mark_threshold`：整列被认为是“自由”("free") 的时候，含有的标记单元 (“marked”) 的最大数目。

实验观察情况 实验进一步阐明了体素层参数的影响。我们使用 ASUS Xtion Pro 作为深度传感器。我们发现 Xtion 的位置很重要，它决定了“盲区”的范围，即深度传感器看不到任何东西的区域。

此外，当障碍物出现在 Xtion 的范围内时，表示障碍物的体素才会进行更新（标记或清除）。否则，一些体素的信息仍保持不变，在代价地图中的膨胀信息也会保留。

另外，Z 轴分辨率 `z_resolution` 灰顶 Z 轴体素的密度。如果值很高，体素层会很密集。如果值太低（例如 0.01），所有的体素将被放在一起，将不会获得有效的代价图信息。如果将 Z 轴分辨率设置为较高的值，意图是更好地获得障碍物，因此需要增加 Z 轴体素数（该参数控制每个垂直列中的体素数）。如果列中的体素数太多但分辨率不够也是没用的，因为每个垂直列的高度都有限制。图 18-20 显示了不同体素参数设置之间的比较。

另外，Z 轴的分辨率参数 `z_resolution` 控制体素在 z-轴上的密度。如果值很高，则体素层会更密集。如果该值太低（例如 0.01），则所有体素将被放在一起，因此您将无法获得有用的代价图信息。如果将 `z_resolution` 设置为更高的值，您的愿望应该是更好地获得障碍，因此您需要增加 `z_voxels` 参数来控制每个垂直列中的体素数。如果列中的体素太多但分辨率不够会导致设置不起作用，因为每个垂直列的高度都有限制。图 18-20 展示了设置不同的体素层参数进行比较的效果。

图 18: 场景：植物在机器人前面

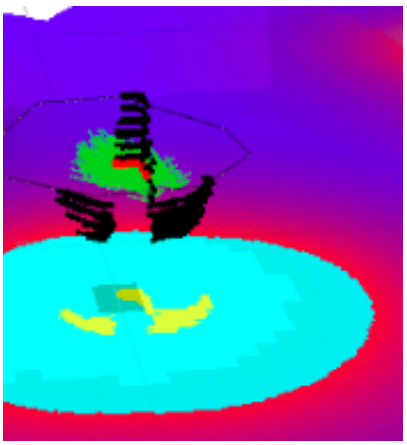

图 19: 较高的 `z_resolution`

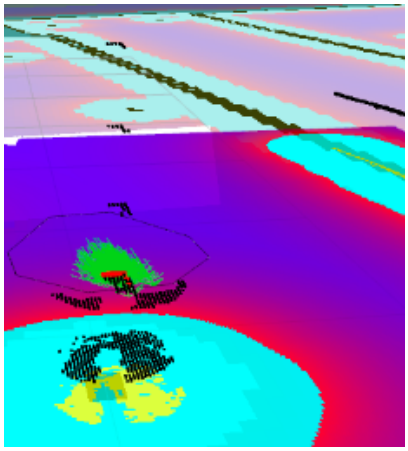

图 20: 较低的 `z_resolution`

5 AMCL

`amcl` 是一个处理机器人定位的 ROS 包。它是 Adaptive Monte Carlo Localization（自适应蒙特卡罗, AMCL）的缩写，也称为粒子滤波 (partical filter) 定位器。这种定位技术的工作原理如下：每个样本都存储一个表示机器人姿态的位置和方向数据。粒子在开始状态时都是随机采样的。当机器人移动时，粒子将根据其当前状态以及机器人的动作使用递归贝叶斯估计 (recursive Bayesian estimation) 进行重新采样。

稍后将提供更多关于调整 AMCL 参数的讨论。请参考 ROS 维基http://wiki.ros.org/amcl以了解更多信息。关于原始 Monte Carlo Localization (蒙特卡罗，MCL) 的算法细节，阅读由 Thrun, Burgard, 和 Fox 所著的《概率机器人》(《Probabilistic Robotics》第八章[4]。

我们现在总结一些可能影响 AMCL 定位质量的问题[7]。我们希望此信息能够使本指南更加完整，并且您觉得有用。

通过实验，我们观察到三个影响 AMCL 定位的问题。正如[4] 中所述，MCL 维护两个概率模型，一个运动模型 (motion) 和一个测量模型 (measurement)。在 ROS 的 `amcl` 中，运动模型对应于里程计的模型，而测量模型对应于激光扫描的模型。通过这种理解，我们分别描述了三个问题归结如下。

5.1 `LaserScan` 的头部结构问题

发布到 `scan` 主题的消息类型为 `sensor_msgs/LaserScan` [8]。此消息包含一个 header 部分，其字段取决于您所使用的具体的激光扫描仪。这些字段包括（以下内容摘自该消息的说明文档）

- `angle_min` (float32) scan 的开始角度 [弧度]
- `angle_max` (float32) scan 的结束角度 [弧度]
- `angle_increment` (float32) 测量的角度间的距离 [弧度]
- `time_increment` (float32) 测量间的时间 [秒] – 如果扫描仪在移动，这将用于插入 3D 点的位置
- `scan_time` (float32) 扫描间的时间 [秒]
- `range_min` (float32) 最小的测量距离 [米]
- `range_max` (float32) 最大的测量距离 [米]

[7]于 2019 年 4 月 8 日增加。此指南于 2017 年 5 月完成，当时未在本指南中报告该部分内容。

[8]请参阅：http://docs.ros.org/melodic/api/sensor_msgs/html/msg/LaserScan.html

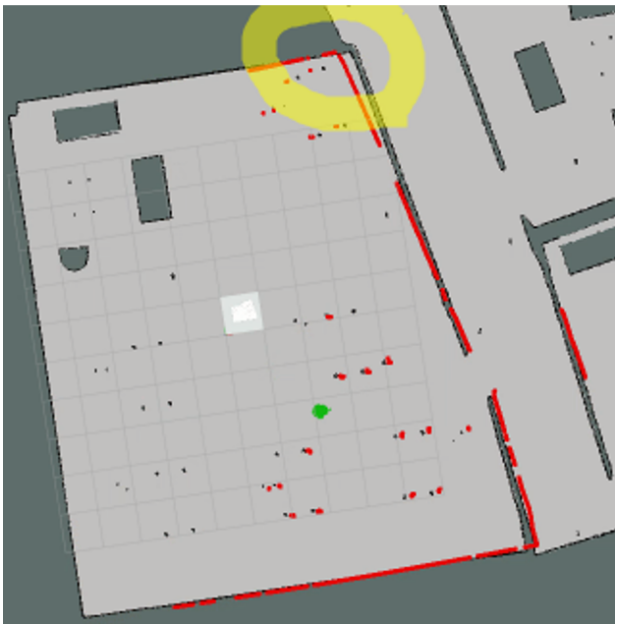

图 21: 当 `LaserScan` 字段不正确时

图 22: 当 `LaserScan` 字段正确时

- `ranges` (float32 []) 测量的距离数据 [米] (注意: 值 < range_min 或 > range_max 应当被丢弃)
- `intensities` (float32 []) 强度数据 [与设备相关的单位]

我们在实验中观察到，如果没有为机器人身上的激光扫描仪正确设置这些参数的值，将影响定位的质量（参见图21和图22。我们使用了两个激光扫描仪，SICK LMS 200 和 SICK LMS 291。我们提供它们的参数如下[9]。

SICK LMS 200:

```
{
"range_min": 0.0,
"range_max": 81.0,
"angle_min": -1.57079637051,
"angle_max": 1.57079637051,
"angle_increment": 0.0174532923847,
"time_increment": 3.70370362361e-05,
"scan_time": 0.0133333336562
}
```

SICK LMS 291:

```
{
"range_min": 0.0,
"range_max": 81.0,
"angle_min": -1.57079637051,
```

[9]对 LMS 200，感谢该 *github* 的问题（https://github.com/smichaud/lidar-snowfall/issues/1）

```
"angle_max": 1.57079637051,
"angle_increment": 0.00872664619235,
"time_increment": 7.40740724722e-05,
"scan_time": 0.0133333336562
}
```

5.2 测量和运动模型的参数问题

`amcl` 包中列出了有关调整激光扫描仪模型（测量）和里程计模型（运动）的参数。有关完整列表及其定义，请参阅 `amcl` 包页面。对这参数的详细讨论需要理解论文[4] 的 MCL 算法，我们在此省略该部分内容。我们提供了一个示例展示如何调整这些参数，并定性描述它们的结果。实际参数应根据所使用的激光扫描仪和机器人作调整。

对于**激光扫描模型**，默认的参数为：

```
{
"laser_z_hit": 0.5,
"laser_sigma_hit": 0.2,
"laser_z_rand" :0.5,
"laser_likelihood_max_dist": 2.0
}
```

为了改善机器人的定位精度，我们增加了 `laser_z_hit` 和 `laser_sigma_hit` 参数来引入更多噪音测量。最后的参数为：

```
{
"laser_z_hit": 0.9,
"laser_sigma_hit": 0.1,
"laser_z_rand" :0.5,
"laser_likelihood_max_dist": 4.0
}
```

机器人的行为如图 23和24所示。显然，在我们的案例中，在测量模型中增加噪声测量有助于定位。

对于**里程计模型**，我们发现我们的里程计非常稳定。因此，我们调整了参数，便于算法假设里程计中的噪音很低：

```
{
"kld_err": 0.01,
"kld_z": 0.99,
"odom_alpha1": 0.005,
"odom_alpha2": 0.005,
```

图 23: 默认测量模型参数

图 24: 调整测量模型参数后（增加噪声）

```
"odom_alpha3": 0.005,
"odom_alpha4": 0.005
}
```

为验证上述参数对运动模型是否有效，我们还尝试了一组受噪声污染的里程模型的建议参数：

```
{
"kld_err": 0.10,
"kld_z": 0.5',
"odom_alpha1": 0.008,
"odom_alpha2": 0.040,
"odom_alpha3": 0.004,
"odom_alpha4": 0.025
}
```

我们观察到，当里程模型噪声较小时，粒子会更加收缩；否则，粒子会更加分散。

5.3 激光扫描仪的平移问题

ROS 中有个 `tf` 变换可以将坐标系从 `laser_link` 转换为 `base_footprint` 或 `base_link`，表示激光扫描仪相对于机器人基座的姿态。如果此变换不正确，则很可能导致机器人的定位行为难以理解。在那种情况下，我们观察到激光传感器扫描环境中的墙壁时其读数有一定的漂移，并且定位产生剧烈变化。这种检验方式足够简单，用以确保 `tf` 变换无误; 通常在机器人的 `urdf` 和 `srdf` 规范中有相应的处理方式。但是，如果您使用的是 `rosbag` 文件，则必须自行发布 `tf` 转换。

6 恢复行为

机器人导航过程中比较头疼的一个问题是机器人可能会被卡住。幸运的是，导航功能包内置了恢复行为。即便如此，有时机器人会耗尽所有可用的恢复行为后保持静止。因此，我们需要一个更强大的解决方案。

恢复行为的种类:ROS 导航功能包有两种恢复行为。它们是`clear_costmap_recovery`和`rotate_recovery`。清除代价地图恢复本质上是将局部代价地图恢复为与全局代价地图等价的状态。旋转恢复是通过旋转 360ř 来恢复。

解救机器人：有时由于旋转故障，旋转恢复无法获得执行。此时，机器人可能会选择放弃，因为它已经尝试了所有的恢复行为--清除代价地图和旋转恢复。我们观察到，在大多数试验中，当机器人选择放弃行为时，实际上仍有很多方法可以解救机器人。为了避免放弃，我们使用 SMACH 来连续尝试不同的恢复行为，通过其他额外的行为，例如设置非常接近机器人的临时目标，并返回到以前访问过的姿态（即退出）。这些方法可以显著提高机器人的耐久性，并且从以前观察到的无望空间中解救出来[10]。

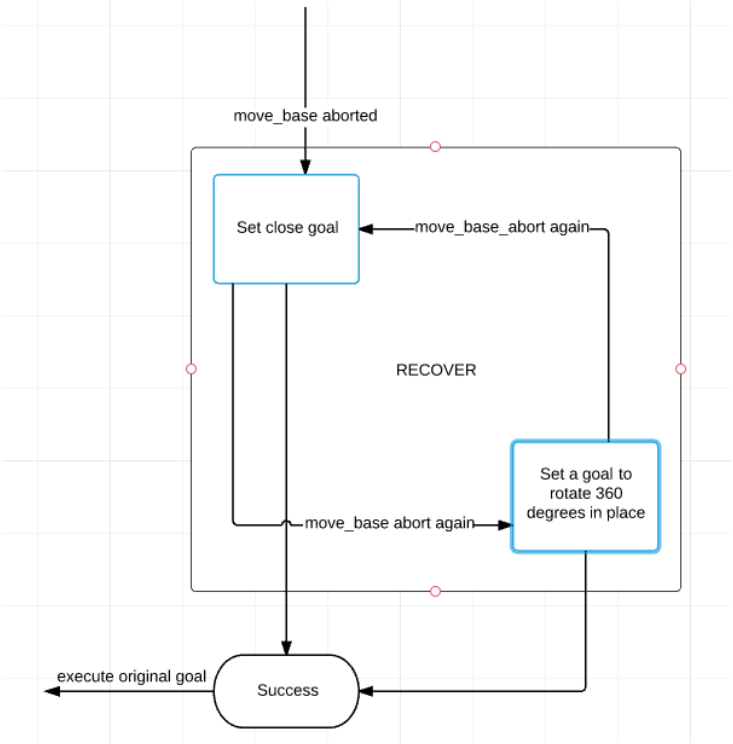

图 25: SMACH 中的简单恢复状态

参数：ROS 的恢复行为的参数一般可以保留为默认值。为了清除代价地图进行恢复，你可以设置一个相对较高的模拟时间`sim_time`，这意味着轨迹很长，你可能需要考虑增加`reset_distance`参数的值，这样可以消除本地代价地图上更大的区域，并且局部规划器有更好的机会寻找一条路径。

[10]这是我在开展移动机器人导航方面的工作时的一个演示视频: https://youtu.be/1-7GNtR6gVk

7 动态重新配置

ROS 导航最灵活的特性之一是动态重新配置，因为不同的参数设置可能对某些情况（如机器人靠近目标点时）更有帮助。即便如此，没有必要进行大量的动态重新配置。

示例：我们在实验中观察到的一种情况是机器人常常会脱离全局路径，即使没有必要这么做或者这样做会带来不好的结果。因此我们增加了路径距离偏差`path_distance_bias`。由于大的路径距离偏差值`path_distance_bias`将使机器人遵循全局路径，但实际上由于容差的原因机器人并不会最终到达目标点，我们需要一种方法让机器人毫不犹豫地达到目标点。此时，我们动态减少路径距离偏差`path_distance_bias`，以便在机器人接近目标点时强调`goal_distance_bias`。毕竟，做更多的实验是为了找到解决问题的方法。

8 问题

1. 陷入困境。在使用 ROS 导航的时候，这个问题经常出现，无论是在仿真还是实际中，机器人都可能陷入困境然后放弃目标
2. 不同方向采用不同的速度。在导航功能包中我们观察到一些奇怪的行为。当目标点设置在相对于 TF 原点的-x 方向时，dwa 局部规划器规划表现不稳定（局部规划的路径存在跳跃行为），而且机器人的移动速度非常缓慢。但是当把目标设置在 +x 方向时，dwa 局部规划器就比较稳定了，并且移动速度很快。

   我在 github 上报告了这个问题：https://github.com/ros-planning/navigation/issues/503。目前还没有人尝试解决它。
3. 实际 VS. 仿真。

   实际运行与仿真运行是有差别的。现实情况中，障碍物有各种各样的形状。例如，实验室有一个垂直的柜子，用以防止门关闭；由于柜子太细，机器人有时无法检测到柜子然后撞上柜子。而且实际中也会存在更复杂的人类活动。
4. 不一致性。

   使用 ROS 导航功能包可能会出现不一致的行为。例如进门时，不同时间里局部规划器会一次又一次地生成差异较小的代价地图，这可能会影响路径规划，特别是在分辨率较低的时候。另外，机器人没有内存时，它不记得上次是怎么从门进入房间的，所以每次尝试进门都需要重新开始尝试。因此，如果机器人进门时采用与以前进门不同的角度，机器人可能会卡住并最终放弃目标。

致谢

希望本指南对你有所帮助。请随意添加你在自己的实验中观察到的更多信息。

【参考文献】

[1] BROCK O, KHATIB O. High-speed navigation using the global dynamic window approach[C]// Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C). Vol. 1. [S.l.]: [s.n.], 1999: 341-346 (引用页: 3).

[2] FOX D, BURGARD W, THRUN S. The dynamic window approach to collision avoidance[J]. IEEE Robotics & Automation Magazine, 1997, 4(1): 23-33 (引用页: 6).

[3] FURRER F, BURRI M, ACHTELIK M, et al. Robot operating system (ros): The complete reference (volume 1)[M]. [S.l.]: by A. Koubaa. Cham: Springer International Publishing, 2016: 595-625 (引用页: 8).

[4] THRUN S, BURGARD W, FOX D. Probabilistic robotics[M]. [S.l.]: MIT press, 2005 (引用页: 13, 15).