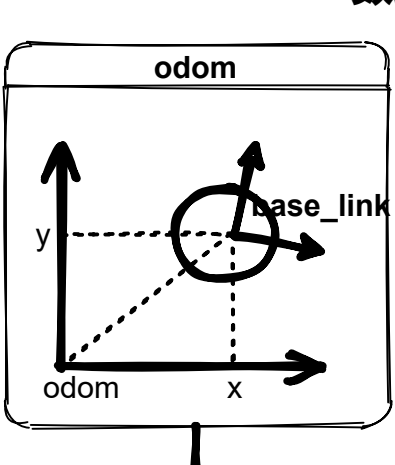
三个部分

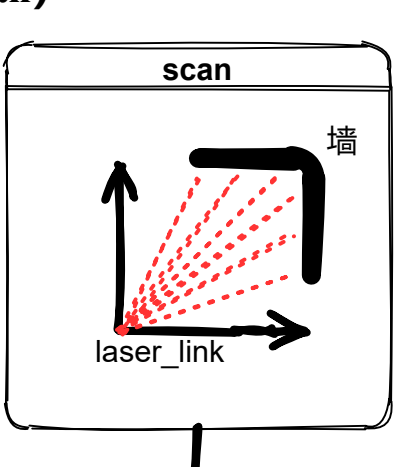
odom scan → SLAM → map tf

数据（odom和scan）



每时每刻机器人在里程计坐标系下的位姿用odom表示

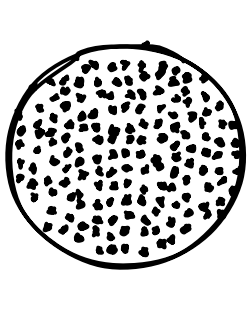
base_link在odom坐标系下的位姿（x,y,theta）



一帧scan的数据表示在当前laser_link坐标系下对周围环境的感知

每帧scan中存储每束激光在当前laser_link下的angle和range

一群粒子



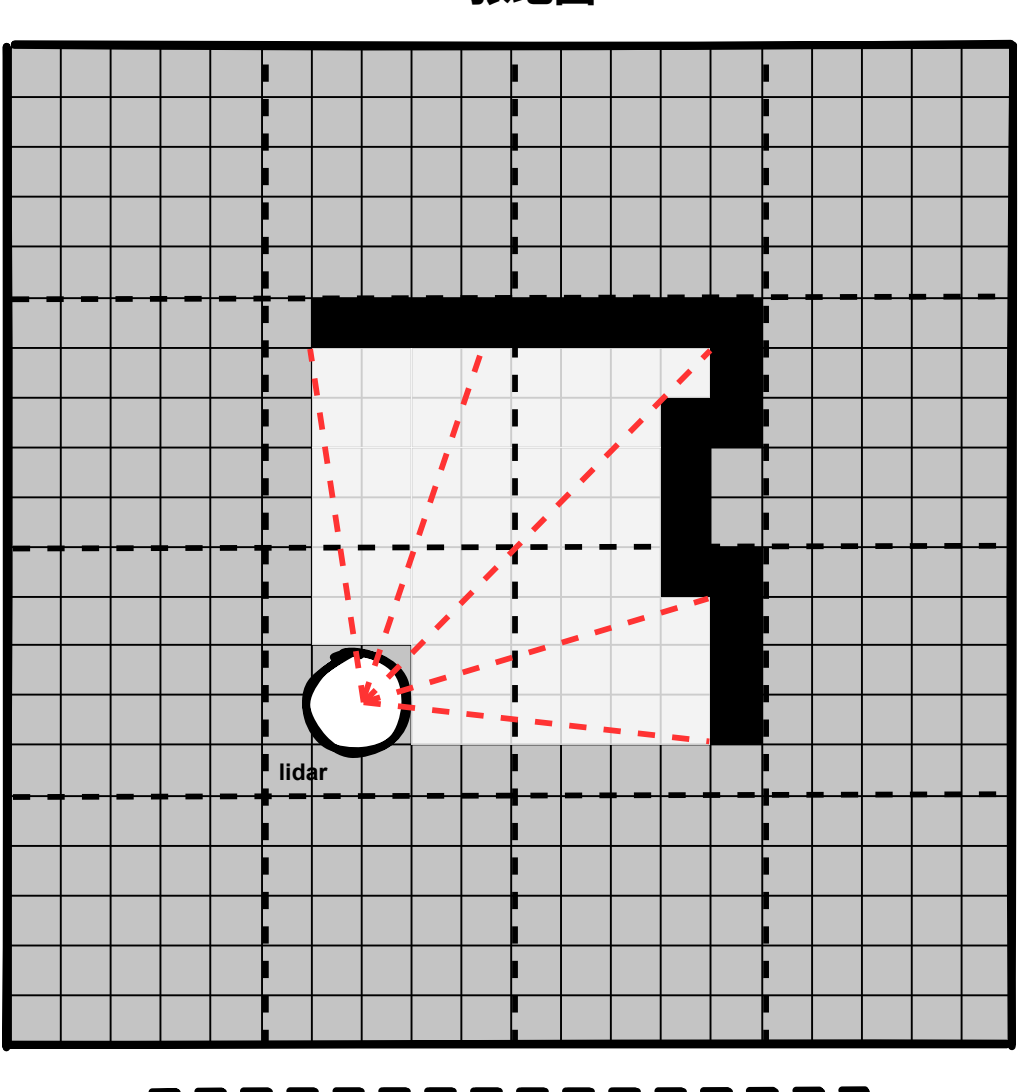
每个粒子包含	
1	该粒子当前map,仅占据部分
2	该粒子的当前最优pose
3	该粒子的权重&累计权重
4	该粒子的整条轨迹

一个机制

当前帧scan在当前位姿pose下与当前map的匹配程度越高，该粒子的权重得分也就越高，pose也就越优。

每一帧scan经过，都要计算该粒子的权重得分，并且累计该粒子的权重，累计权重得分最高的粒子，也就可以绘制最优的地图。

一张地图



图例

地图：一张地图由若干个地图块组成

地图块：一个地图块由若干栅格组成，一般栅格数目固定，栅格的大小表示地图精度

栅格：一个栅格有三种状态，占据、空闲、未知，用不同颜色表示，默认未知

黑色=占据
灰色=未知
白色=空闲

解释说明

每个栅格数据结构	
1	该栅格被访问的次数v
2	该栅格被击中的次数n
3	击中点物理位置的累积

栅格占据概率：n/v

my_slam_gmapping基于RBPF，每个粒子都要存储一个地图，建图范围比较大，极大的占用内存。这里的地图使用两层结构，第一层：每张地图存储若干地图块，第二层：每个地图块都存储固定数目的栅格，通过筛选出含有被击中栅格的地图块，进而对所属的栅格进行内存分配，未被筛选的地图块不进行栅格的内存分配，不用存储栅格的内容，栅格默认未知状态，所以也就节省了不少内存。比如上面地图，只有含击中栅格的地图块申请了内存。

两条主线（my_slam_gmapping）

part_slam

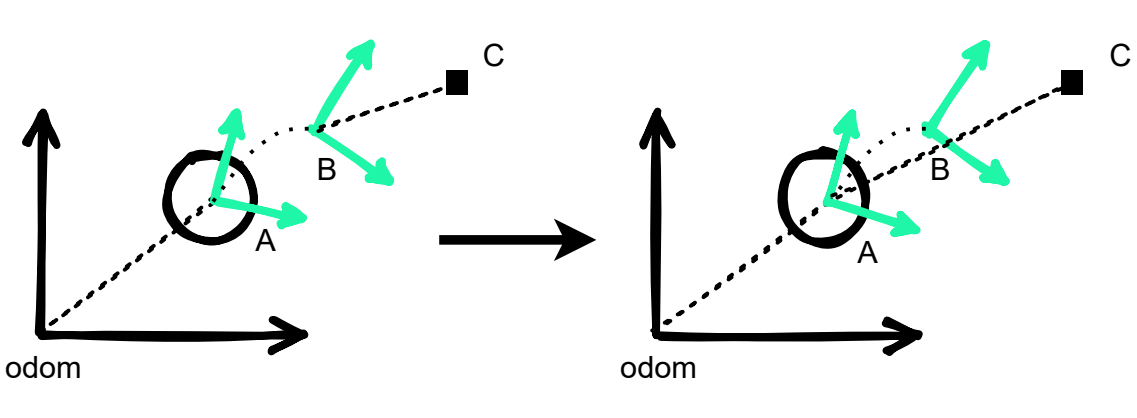
tf scan → 激光雷达运动畸变去除 → 求laser_link在odom下的位姿 → 运动模型传播 → scan_match → 粒子差异权重计算 → 选择性重采样 → 计算地图 →（返回 tf scan）

part_ros

寻找最优粒子 → 占据栅格地图构建 → 发布map →（返回 寻找最优粒子）

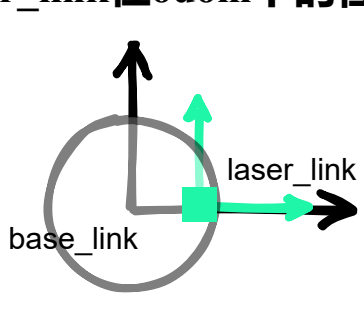
单独线程 → 发布map to odom的tf →（返回 单独线程）

激光雷达运动畸变去除



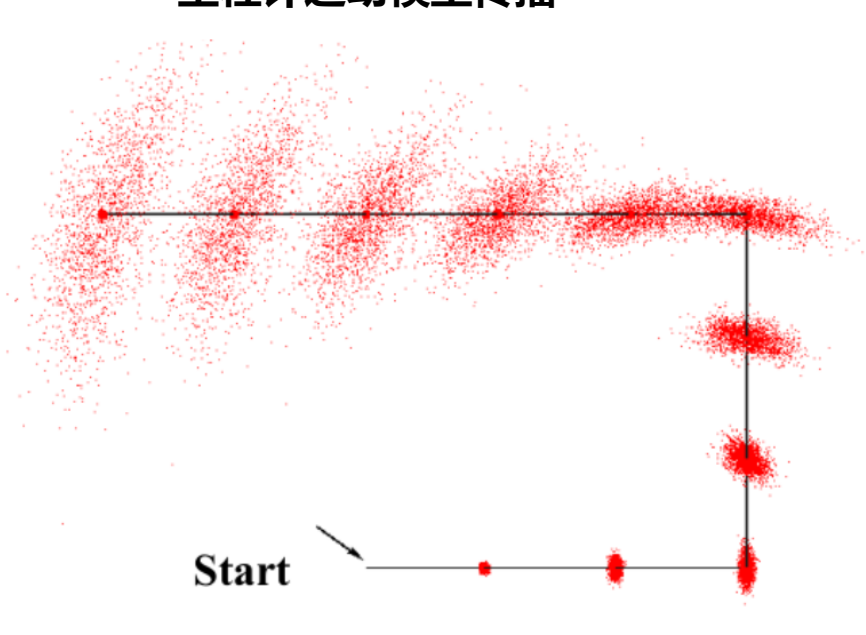
绿色坐标系为laser_link,黑色坐标系为odom；一般激光雷达驱动封装数据时，默认一帧激光雷达数据的所有激光束是在同一位姿下、瞬时获得的，而实际采集的数据却在不同时刻的laser_link上采集的，所以也就产生了激光雷达数据的运动畸变。这里假设A坐标系为一帧激光数据的起始位置，B为运动中的某一束激光数据真实采集时的laser_link，C点为障碍物。该激光束的数据从BC变换到AC的过程就激光雷达运动畸变去除过程。

求laser_link在odom下的位姿



黑色圆圈表示车体，绿色方块表示激光雷达，明显两者几何中心不一致。我们知道odom表示base_link在odom下的位姿，而算法中我们需要laser_link在odom下的位姿，于是我们从tf树上取数据转换即可得到。

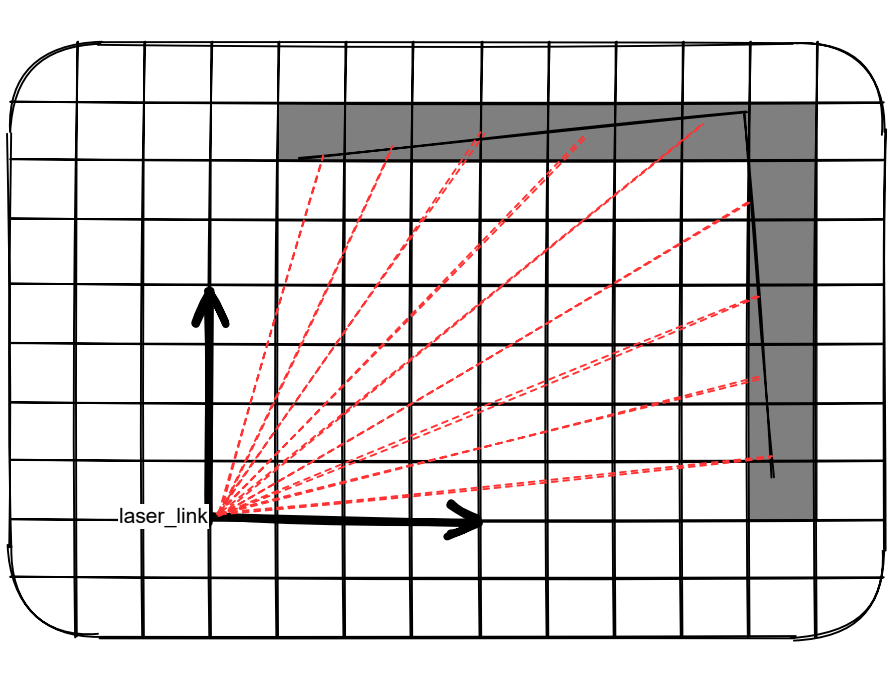
里程计运动模型传播



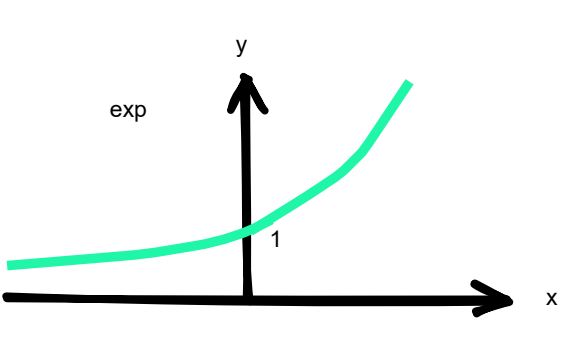
在当前时刻和上一时刻laser_link在odom下的位姿数据下，为每个粒子存储的laser_link在map下的位姿进行传播，此过程加入高斯噪声。若只进行传播，噪声不断累积，粒子所代表的的位姿也就差异越大，粒子也就越离散。如上图所示，加入高斯噪声的目的下面会讲到。

scan_match

scan_match主要有两个功能，第一，为每一个粒子的pose找到一个最优值；第二为每个粒子更新权重&累计权重

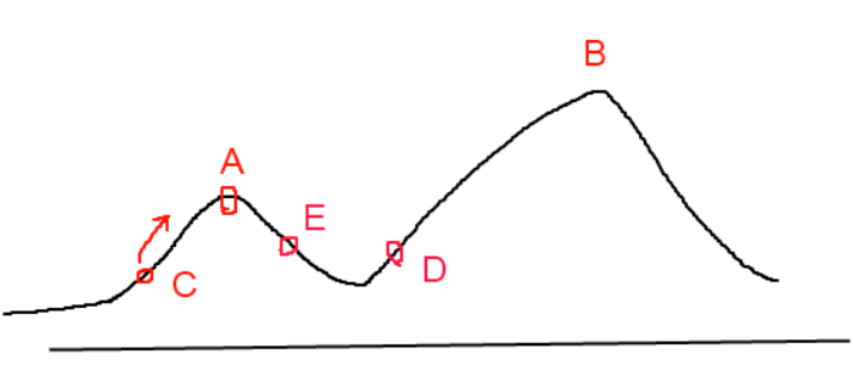


还记得每个被击中的栅格一直累计每次击中时刻击中点的在map下的物理坐标吗（不是栅格坐标，单位m）？在当前时刻当前位姿下，此时每个击中栅格的物理坐标数据，与之前对应栅格累计物理坐标数据的均值求误差距离e。然后误差距离e越小，说明该位姿和当前地图越匹配，如上图所示。



score = exp(-1.0/k*e*e)

这里使用exp函数计算当前位姿每束激光与地图的匹配得分，误差e当做指数的分子，如上式；也就是说误差e越小exp值越向右，当误差为零，exp(0)=1；误差e无穷大时，exp无限接近于0；参数k的大小可以根据误差e的大小分布，尽可能的让exp的取值取满(0,1]的空间。之后对每一个激光束的匹配得分累计，累计得分即为该位姿在该地图的得分，得分越高，越匹配。



之前laser_link在map的位姿运动更新时，加入了高斯噪声，使不同位姿近似高斯分布，所以这里使用爬山算法，评判机制是该位姿与地图匹配得分的大小，位姿启发式的向局部最优值（山顶）改变，当匹配得分下降时，结束位姿探索，此时的位姿即为该粒子的最优位姿。这里对每个粒子都进行爬山算法位姿优化。

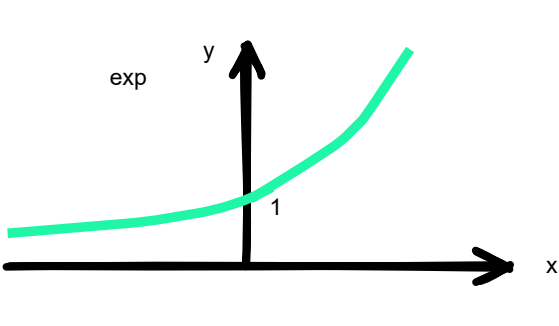
似然得分 = -1.0/k*e*e

粒子权重 = 累计似然得分

粒子累计权重 += 累计似然得分

每个粒子找到最优位姿之后，要为每个粒子计算权重和累计权重；位姿与地图越匹配，误差e越小，似然得分越大，累计似然得分越大，粒子权重也就越大。每次累计粒子权重，累积权重越大，也就说明该粒子和地图越匹配。

粒子差异权重计算



w = exp(1.0/k*(it_w - max_w))

w_all += w

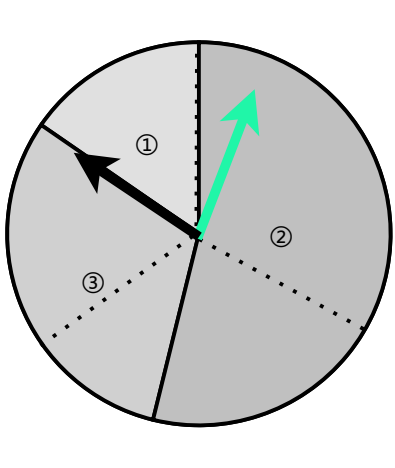
w = w/w_all

neff += w*w

neff = 1/neff

it_w每个粒子当前的权重、max_w所有粒子的最大权重，w每个粒子的差异权重；粒子权重差距越大，权重w越小。累计每个粒子的w，并且归一化每个粒子的w,然后累计得到表示粒子离散程度的neff，并由此判断是否重采样。如上图。

重采样过程



举例说明，如上图，有三个粒子，不同的权重大小用不同灰色的区域表示，如编号，黑色虚线将权重之和均分为三分；绿色箭头是第一次随机抽到的目标权重；

↓

两个箭头都向顺时针旋转，黑色箭头每次增加当前粒子的权重大小；若黑色超过绿色箭头将在绿色箭头区域采样一次，否则不采样，每次采样之后绿色箭头都向前旋转粒子的权重均值个位置。

↓

简单推导一下，在②区域采样两次，在③区域采样一次，在①区域没有采样。这就是重采样过程，优先采样权重大的粒子，舍弃权重小的粒子。

↓

重采样之后，为每个粒子添加新的节点，指向之前的轨迹，并且把粒子权重清零

part_slam的计算地图过程

根据激光束扩展地图大小 → 根据激光束找到需要申请内存的地图块 → 为筛选的地图块的栅格申请内存 → 根据击中栅格更新栅格数据

这里值得说明一下，为了减少计算量，part_slam的地图计算过程与part_ros不同，这里只更新击中栅格的数据，其他栅格不更新，采用默认未知数据。

过渡

从上面的内容，你应该了解了关于my_slam_gmapping的算法核心是在解决哪些问题，以及具体是如何解决的。

↓

从数据形式->粒子结构->评判机制->地图形式->算法主线->运动畸变去除->tf变换->运动模型传播->scan_match机制->粒子差异权重计算->重采样过程->到part_slam的计算地图过程，你若仔细看了，你大致也就明白了。

↓

整个过程，就只有一个目的：稳定实时可靠鲁棒的计算出每一帧最优的位姿，即定位。

↓

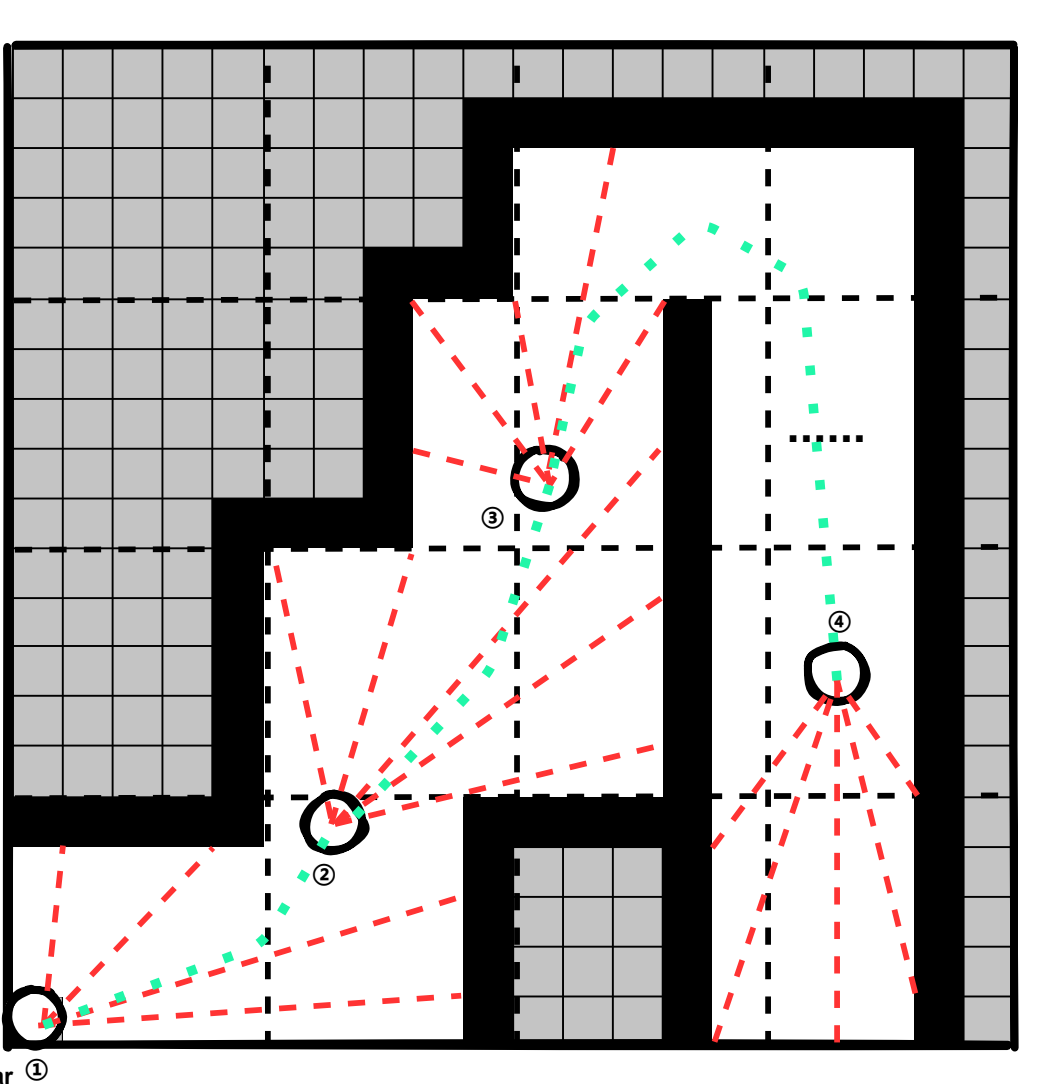
下面就剩下占据栅格地图构建部分了，也就迎来了part_ros。

part_ros占据栅格地图构建

根据粒子权重累计得分，找到最优的粒子，根据其存储的最优轨迹，计算地图。

↓

这里只做一个简单的过程演示，顺序①②③④



红线激光束，绿线机器人轨迹，根据画线算法，找到空闲和击中的栅格序号，分别按照栅格数据结构对栅格进行数据更新，空闲只更新访问次数v，击中更新访问次数v&击中次数n&击中的物理位置，持续多帧计算，栅格的概率=n/v,最后通过占据阈值判断栅格是否被击中，也就区分了白和黑的栅格，灰色是默认状态。

part_ros （map to odom的tf变换）

为了在可视化工具rviz上显示map坐标系下的odom，所以要把map to odom的tf变换发布到tf树上。

↓

这里通过同一时刻lidar在map坐标系的位姿和在odom下的位姿关系，来表述map坐标系和odom坐标系的变换关系。

END

小白学移动机器人
送给
孜孜不倦
为之奋斗
的你