动态环境下的MCL定位导航

原理与代码

1.节点关系

本周任务是在前几周的成果基础上进行整合，主要包括以下几个部分，其交互关系为：

（1）**Particle Filter：**

订阅：

`/laser/scan` 用于评估粒子权重

`/map` 用于构建似然地图

发布：

`/roboPose` 表征当前实际位置

`/mapFlag` 表征是否进行建图

（2）**Mapping:**

订阅：

`/mapFlag` 当接收到信号时开始建图

`/laser/scan` 用于建图

发布：

`/grid_map_mine` 储存当前建图结果

（3）**global Planner:**

订阅：

`/grid_map_mine` 用于构建路径

`/replan` 接收到时进行重归化

发布：

`/global_path` 当前规划的全局路径

（4）**local Planner:**

订阅：

`/laser/scan` 用于动态避障

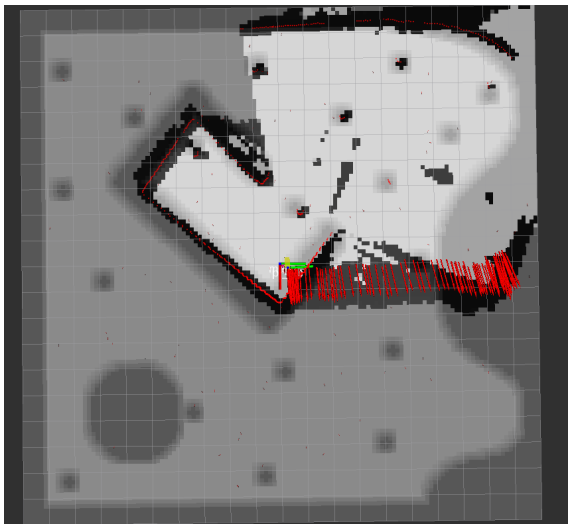
`/global_path` 用于规划轨迹

发布：

`/replan` 当触发阈值时，请求重规划

2. 绑架问题

当粒子滤波定位时，若发生绑架问题则可能会导致错误建图，在现象严重的时候，可能会导致路径搜索失败，如下图：



初始的绑架导致在错误的地方叠加了激光的信息进行建图，导致下方的区域被错误的识别为障碍物，从而隔断了区域，影响之后的建图以及规划，因此需要对当前的定位结果进行一个有效的评判，当且仅当当前定位结果可信时，才进行建图。以下提出两种评判方法：

（1）加权位置评判法

该方法将加权后的定位也视为一个粒子，并采用之前的粒子权重评判法对该粒子进行评估。考虑到在正确定位时，该加权粒子的权重应当具有比较高的水平，因此当且仅当权重大于一定阈值时才向Mapping节点发送MapFlag进行建图，这是对定位结果直接进行判别。

```
double particle_filter::evaluate_location(sensor_msgs::LaserScan input){
    ..
    for(int k = 0; k < laser_num; k++){
        ...
        laser(0) = state(0) + input.ranges[k]*cos(angle) + 10.0;
        laser(1) = state(1) + input.ranges[k]*sin(angle) + 10.0;
        w_laser_sum += LMap(int(laser(0) / map_res), int(laser(1) /
map_res));
        laser_cnt++;
    }
    return w_laser_sum/(1.0*laser_cnt);
}
```

（2）权重MAX评判法

该方法在于分析不同定位情况下的粒子分布状况，进而间接评价当前定位是否准确。当粒子处于绑架状态时，仅有少数几个粒子会处于较高权重区域，而在正确定位的情况下，则会有大量粒子集中于高权重区域。

$$M = \sum_{i=1}^{10} w_{max}$$

当且仅当M大于一定阈值时，才进行重采样

那么对于Mapping，则需要解决同步问题，即在接收到mapFlag时才进行建图，有：

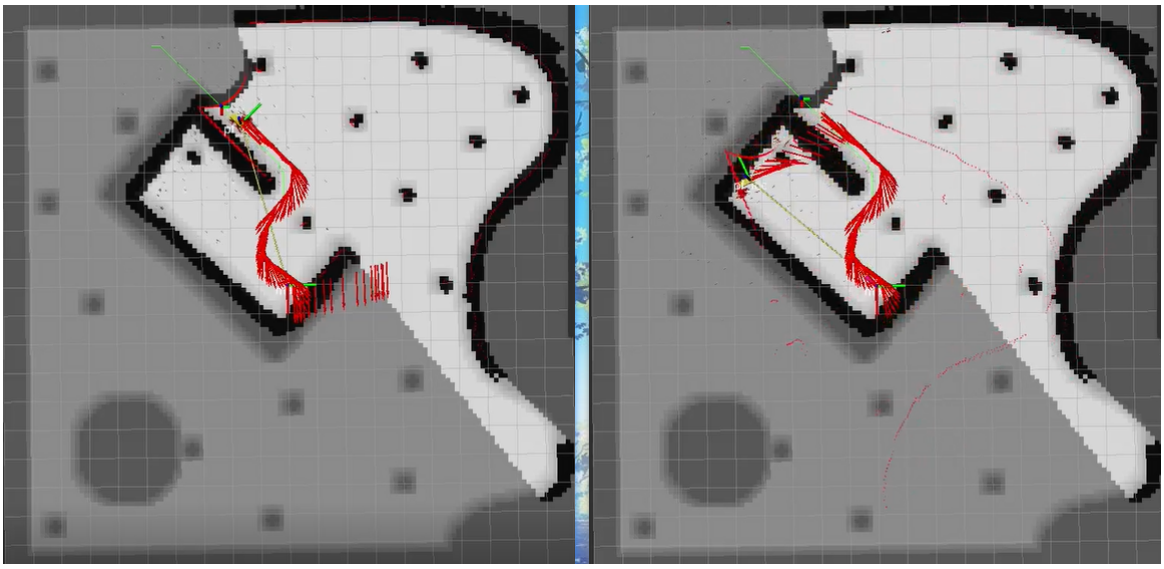
```
laser_sub = n.subscribe("/course_agv/laser/scan", 1,
&mapping::laserCallback, this);
mapFlag_sub = n.subscribe("mapFlag", 1, &mapping::process, this);

void mapping::laserCallback(sensor_msgs::LaserScan input){
    this->laserScan = input;
}

void mapping::process(std_msgs::Float64 mapFlag){
    if(!mapFlag) return;
    ...
}
```

3.误识别问题

在似然地图不更新的时候，还会存在一个较为棘手的问题。在小车较为接近动态障碍物时，由于大量激光点落在动态障碍物上，而这些激光点对应的似然地图为**非占用**，从而导致**粒子权重显著下降**，则PF的定位值很可能会飘到权重更高的错误位置：

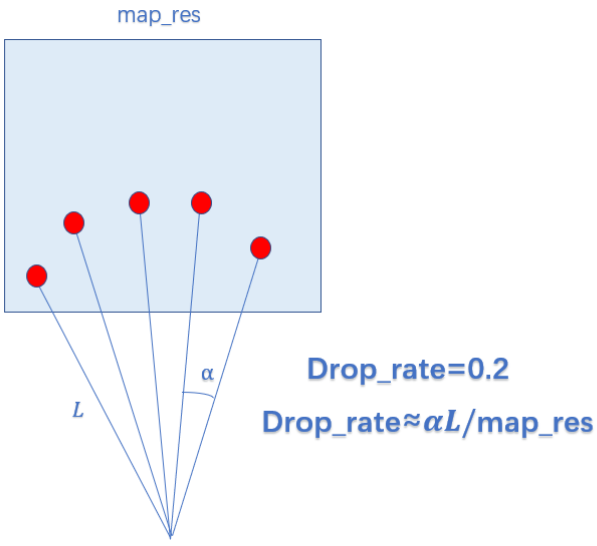


为了解决上述问题，这里引入**random_drop**的方法以改进粒子权重计算。算法的核心思想在于降**低近处激光点对于位置权重计算的贡献**。原因在于：

在**相同位置误差(▲xy,▲theta)**的时候，远处的激光点的偏移会远大于近处的激光点，因此远处的似然值能比近处的似然值更能反应定位的准确与否。

在原思路中，将**激光点**作为主体，对每一个激光点进行计算似然值，此时，远近激光点等地位。

在改进思路中，将**栅格**作为主体，对每一个扫到的栅格取一个激光点反应其属性，此时，近处的激光点会有大量被舍弃。在具体实现中，我们使每一个激光点以一定概率被舍弃，舍弃概率为距离的反比例函数：



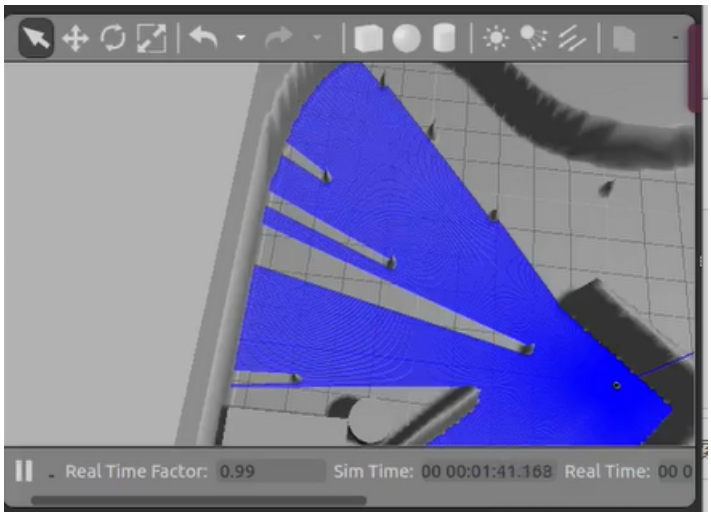
上图中一个栅格内有五个激光点，则每一个的激光点实际的舍弃率为0.2，可以通过图中的公式近似计算

```
for(int k = 0; k < laser_num; k++){
  double drop_rate=2*PI/laser_nums*input.ranges[k]/map_res;
  double sample=rand()%10000/10000.0;
  if (sample>drop_rate) continue; //drop
  ...
}
```

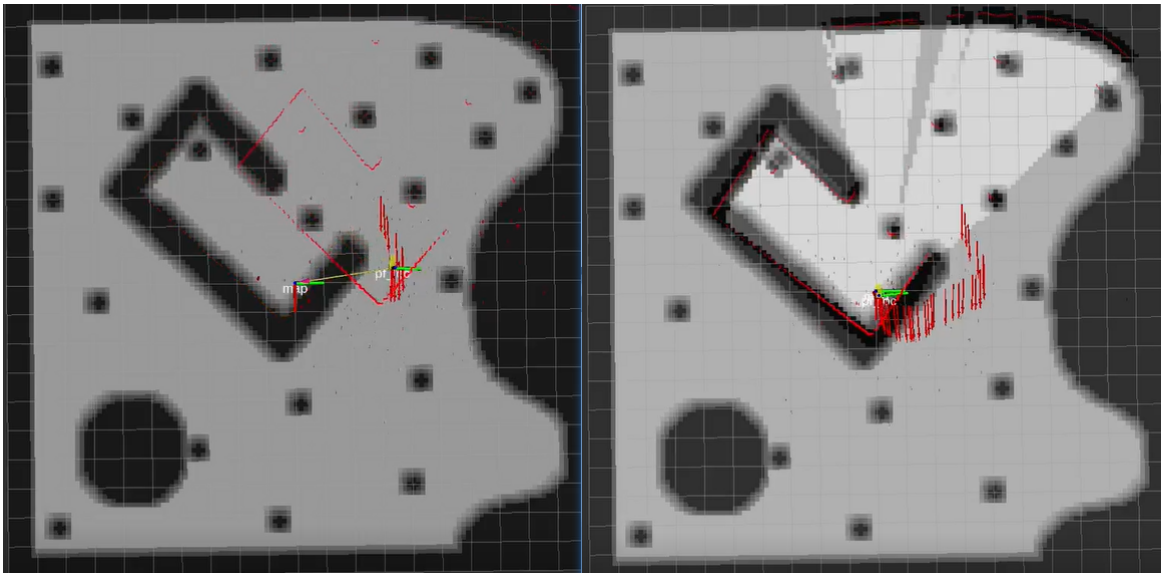
结果与分析

动态建图(见SLAM)：

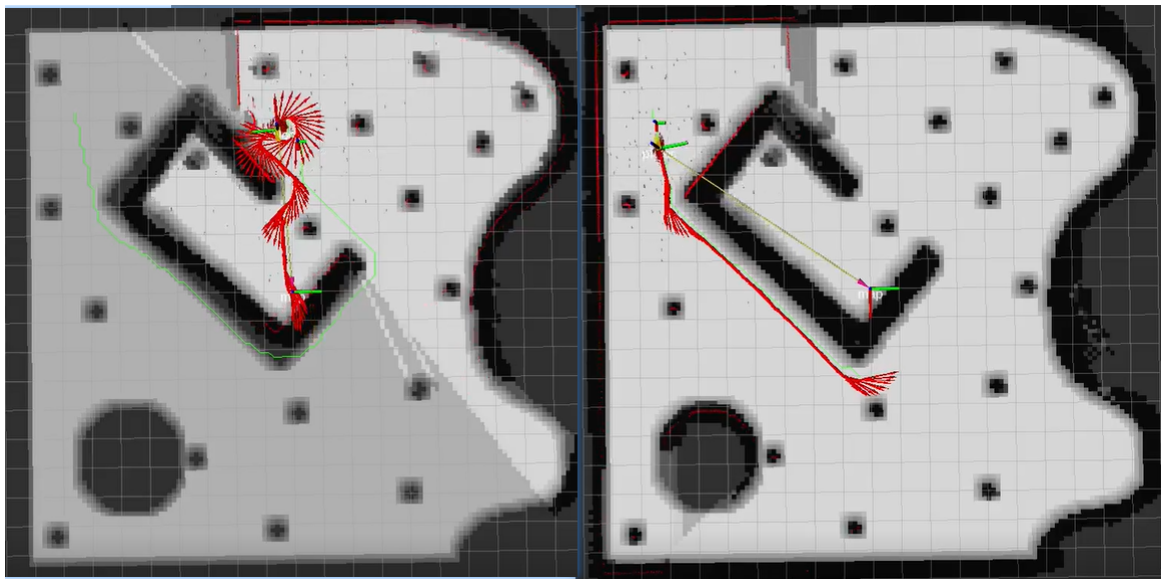
（1）在左上角放置障碍物如下。



（2）在绑架的过程中，当且仅当位置定位准确后才开始建图



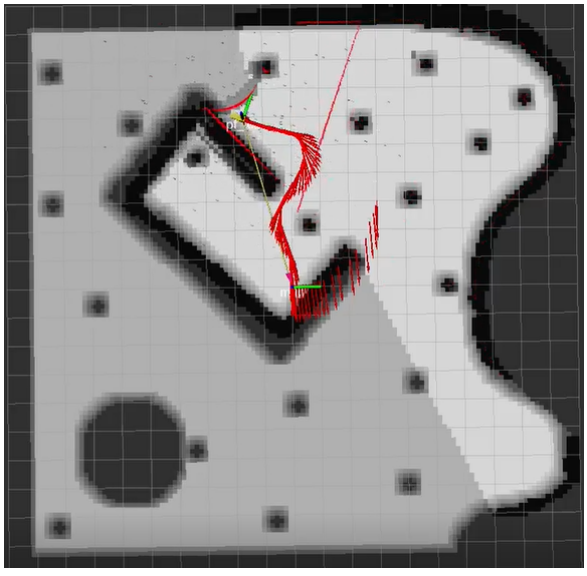
（3）DWA次数达到一定阈值后开始重规划，最终也能顺利完成



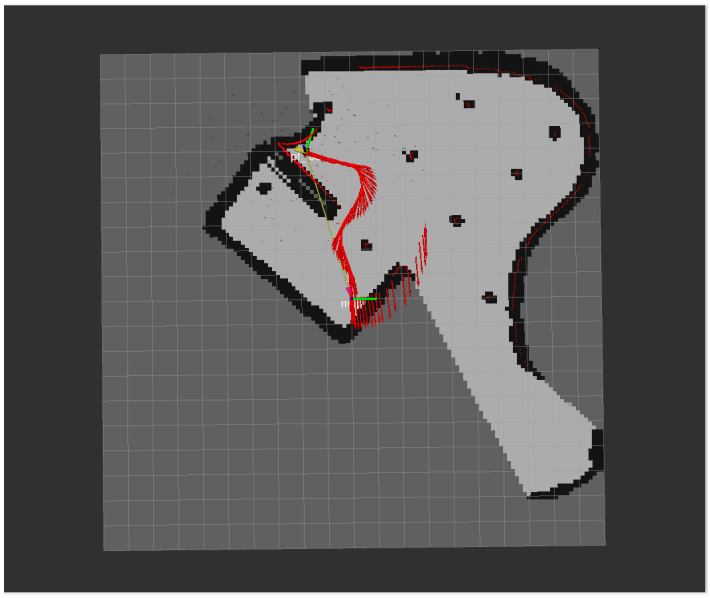
误识别问题(见Location Drift)：

采用random_drop的算法前，抵达动态障碍物前就发生误识别现象。

在采用random_drop的算法后，在**同样的位置停留半分钟，定位准确不漂移。**



实验过程中，建图的效果良好，灰度不一是似然地图叠加造成的，事实上的视频中的深灰色的区域均为建图中视作障碍物的区域：



事实上，PF定位依赖于初始的似然地图，当初始的似然地图与真实环境产生较大的偏差的时候，PF的定位效果并不好，很容易出现误识别的问题，前文提到的random_drop的方法也只是在一定程度上改善这种现象，但是对于动态障碍物很多的环境也是束手无策。

一种很自然的想法是建图的过程中更新似然地图，但是这存在很大的问题：定位的准确衡量是依靠似然地图的，而似然地图又是在定位的基础上更新的，这是一种正反馈，那么在错误的定位下，亦可能会因为停留的时间较久而使得似然地图不断更新最后肯定定位的结果。也就是说，无论定位是否正确，当前的定位都会被视为最优的定位，这样的定位无疑是无用的。当然在初始定位准确的情况下，这种做法可能确实是有效的。

总的来说，在环境较为位置的情况下，PF不是一种很好的选择，可以考虑先通过其他方法探索出比较完整的地图后再运用PF进行定位最终。