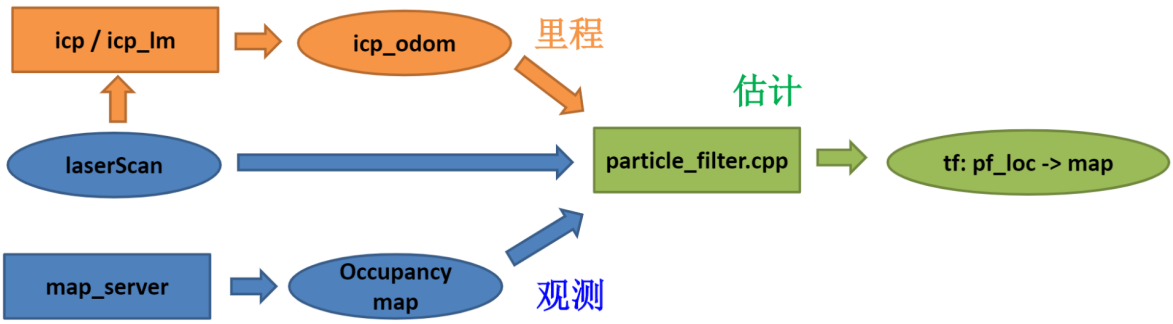
蒙特卡洛定位

Michael Gao

实验任务

- 基于栅格地图，实现粒子滤波定位



实验内容

0.高斯噪声的生成

高斯噪声在蒙特卡洛定位中多处使用，故需要高效得生成不同方差均值的高斯噪声，利用`std`中的函数直接生成：

```
unsigned seed=std::chrono::system_clock::now().time_since_epoch().count();
std::default_random_engine generator(seed);
std::normal_distribution<double> noisex(mean, stdx);
```

1.运动模型

首先读取`icp_odom`的数据，记录下当前时刻与上一时刻的ICP差值，记录`dx,dy,dt`：

```
double nowx = odom.pose.pose.position.x;
double nowy = odom.pose.pose.position.y;
tf::Quaternion quat;
tf::quaternionMsgToTF(odom.pose.pose.orientation, quat);
double roll, pitch, yaw;//定义存储r\p\y的容器
tf::Matrix3x3(quat).getRPY(roll, pitch, yaw);//进行转换
double nowt = yaw;

double dx, dy, dt;
dx = nowx - last_x;
dy = nowy - last_y;
dt = nowt - last_t;
```

在读取到的ICP变化数据中加入高斯噪声，噪声与运动差值成正比:

$$\begin{cases} dx = dx + noise_x * dx \\ dy = dy + noise_y * dy \\ dt = dt + noise_t * dt \end{cases}$$

```
for(int i=0; i<particle_num; i++)
{
    particles[i].id = i;
    particles[i].x += dx + noisex(generator);
    particles[i].y += dy + noisey(generator);
    particles[i].theta += dt + noiset(generator);
}
```

2.观测模型

当接收到一次激光雷达数据时，就进行一次观测，观测的流程如下：

$$\begin{aligned} &\textbf{Algorithm doObservation} \\ &\textbf{for all particles } \mathrm{p}_i \textbf{ do} \\ &\quad \mathrm{Cal\_weight}(\mathrm{p}_i) \\ &\textbf{end} \\ &\mathrm{weightNorm}() \\ &N_{eff} = 1/cov(weight) \\ &\textbf{if } N_{eff} < N_{thres} \\ &\quad \mathrm{resampling}() \\ &\textbf{end} \end{aligned}$$

2.1.构造似然地图

为计算激光点的置信度，首先需要构造似然地图$LikeliMap$，计算激光点落在栅格中的可能性：

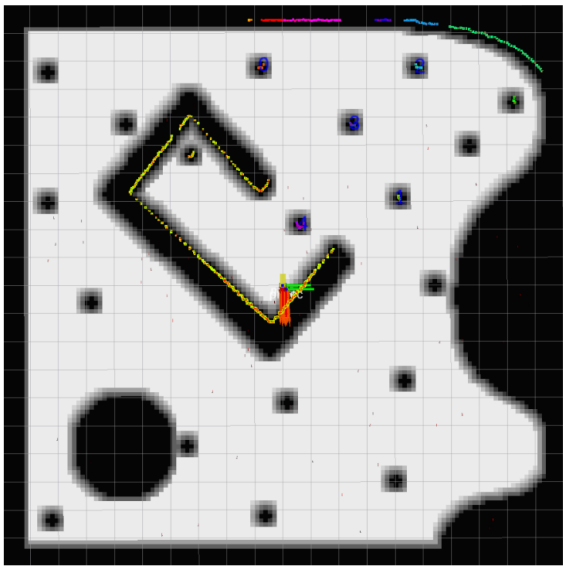
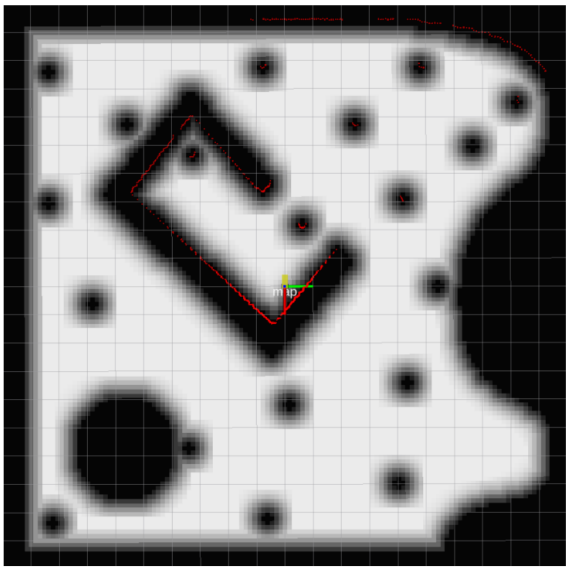
a.高斯MAX扩散地图

对于mapsever发出的原始二值占用栅格地图遍历，对于每个值为100的栅格，将其对应的$LikeliMap$栅格概率赋值为1，并将自身概率向周围的栅格传播，被传播的栅格比较传播过来的概率和自身的概率，取最大值更新：

```
for(int i = 0; i < map_height; i++){
    for(int j = 0; j < map_width; j++){
        if(input.data[j*input.info.width+i] != 0){
            for(int m = i-propag_range; m < i+propag_range; m++){
                for(int n = j-propag_range; n < j+propag_range; n++){
                    if(m<0 || n<0 || m>=map_height || n>=map_width){
                        continue;
                    }
                    likeli_map(m,n) = max(getPGaussian((m-i)*map_res, (n-
j)*map_res), likeli_map(m,n));
                }
            }
        }
    }
}
```

其中，`getPGaussian()`函数接受被传播栅格与中心栅格距离，返回传播的概率值，当距离为0时，传播的值为1.

用这种方式构造出的地图称为高斯MAX扩散地图：

取不同的高斯标准差和不同的扩散距离可以得到不同的似然地图，左图标准差为0.1，扩散距离为3，右图标准差为0.3，扩散距离为6. 后续仿真实验如未说明则使用左图作为似然地图。

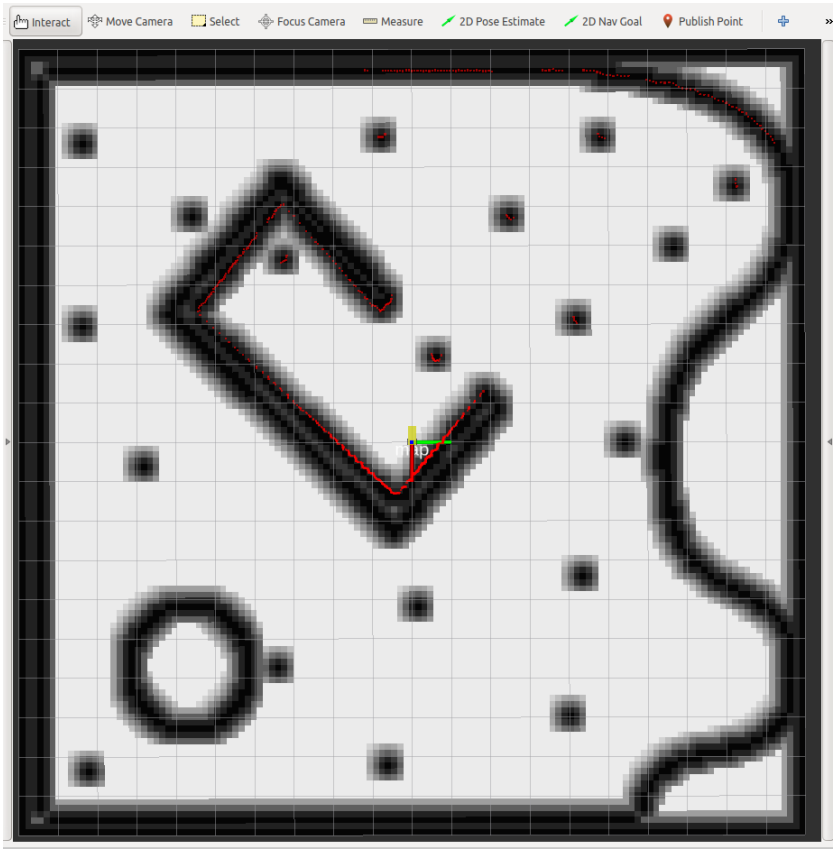
b.<u>边缘高斯地图</u>

在仿真实验中发现，由于激光只可能在障碍物的边缘形成，障碍物内部的点是激光不可能达到的点，然而在似然地图中却又不合理的高置信度，这样会使在错误位置的粒子可能获得不合理的高权值。

故对似然地图构造算法进行调整：当原始二值地图中的值为100的栅格周围的值为100的栅格的数量大于一定值时，认为此栅格在障碍物内部，其对应的$LikeliMap$栅格不执行赋值为1的操作，但仍然接受高斯扩散更新，算法在a部分的基础上改进如下：

```
for(int i = 0; i < map_height; i++){
    for(int j = 0; j < map_width; j++){
        if(input.data[j*input.info.width+i] != 0){
            int black_cnt = 0;
            for(int p = i-1; p <= i+1; p++){
                for(int q = j-1; q <= j+1; q++){
                    if(p<0 || q<0 || p>=map_height || q>=map_width){
                        continue;
                    }
                    if(input.data[q*input.info.width+p] == 100){
                        black_cnt++;
                    }
                }
            }
            if(black_cnt>=7){
                continue;
            }
            ...
                }
            }
        }
    }
}
```

用这种方式构造出的地图称为边缘高斯地图：



2.2.计算粒子权重

对于每个粒子计算其权重，根据每个粒子的位置信息将局部坐标系下的激光点变换到全局坐标系下：

```
double r_angle = input.angle_min + k*input.angle_increment;
g_laser(0) = particles[i].x +
input.ranges[k]*cos(particles[i].theta+r_angle) + 10.0;
g_laser(1) = particles[i].y +
input.ranges[k]*sin(particles[i].theta+r_angle) + 10.0;
```

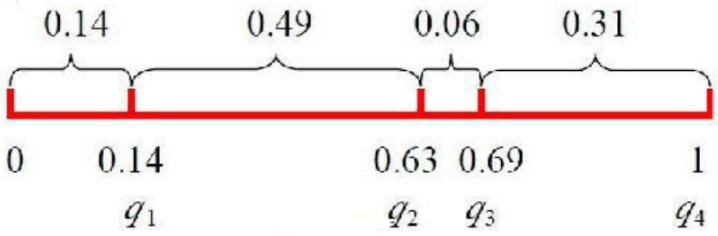
根据激光点落在似然地图的栅格确定此激光点置信度(0-1)，最终将此粒子对应的所有激光点的置信度均值作为粒子权重:

```
p_all += likeli_map(int(g_laser(0)/map_res), int(g_laser(1)/map_res));
p_avg = p_all/(1.0*laser_cnt);
particles[i].weight = p_avg;
```

3.重采样+生成新粒子

重采样步骤中，权值更高的粒子有更大的可能性被重采样，为实现此目标，采用轮盘赌形式：

$$q_i = \sum_{j=1} P(x_j)$$




```
double randw = rand()%1000/1000.0;
w_step.push_back(randw);
sort(w_step.begin(), w_step.end());
vector<double>::iterator it=find(w_step.begin(),w_step.end(),randw);///返回的是地址
int index=std::distance(w_step.begin(), it);///放入迭代器中得到容器中的位置
w_step.erase(it);
```

首先构造如上图所示的权重求和阶梯图存于`vector<double> w_step`中，由于权重已经进行了归一化，故向量最后一个元素值为1.

每次采样时随机生成一个0-1的随机数(此处一定要加时间种子)，加入队列得到其`index`，则该`index`对应的粒子为被采样到的粒子，深复制到新粒子群中。

重采样中设置参数`new_ratio`确定新生成粒子比例，如粒子群数量为100，`new_ratio=0.1`时，新粒子群中将有90个粒子从老粒子群中采样，有10个粒子用生成算法生成。

新粒子在估计的机器人`state`附近以高斯概率随机生成：

```
particle newp;
newp.x = state(0) + noisexy(generator);
newp.y = state(1) + noisexy(generator);
while(newp.x > 10.0 || newp.x < -10.0){
    newp.x = state(0) + noisexy(generator);
}
while(newp.y > 10.0 || newp.y < -10.0){
    newp.y = state(1) + noisexy(generator);
}
newp.theta = state(2) + noiset(generator);
newp.weight=1.0/particle_num;
```

实验结果

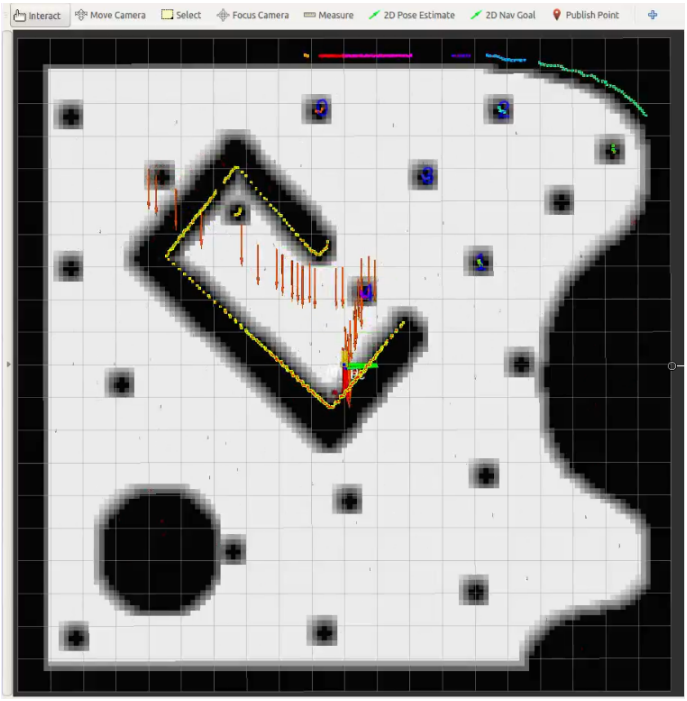
```
roslaunch course_agv_slam_task particle_filter.launch
roslaunch course_agv_slam_task icp_exlm.launch
rosrun course_agv_control keyboard_velocity.py
```

粒子滤波完整实验结果见`/video/pf.mp4`，此仿真采用边缘高斯地图.

视频中$N_{eff} = 8900$，可以明显看到过程中有多次重采样的激发使误差减小.

Case1.机器人绑架

视频结果见`/video/kidnap.mp4`

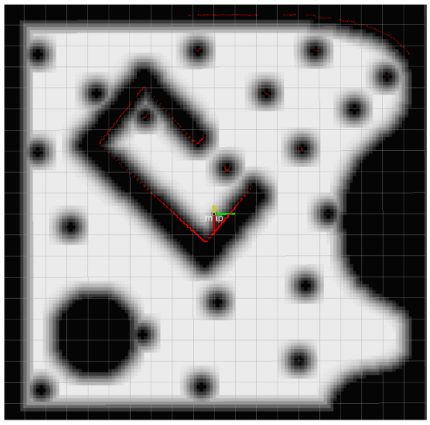


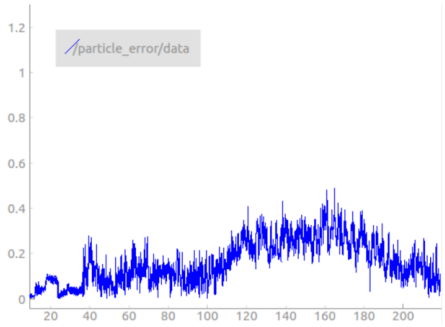
粒子滤波算法全局搜索能力很强，机器人绑架后能够快速重新定位到粒子附近，但是会出现估计位姿在真值附近抖动的情况，最后接近真值的过程较长(视频中间加速处理)，这可能是因为粒子数较少，而为了增强全局搜索能力，新生成粒子噪声较大，很难采到真值点，同时与真值接近的粒子权重本身较高，导致重采样易采到离真值接近的错误粒子。

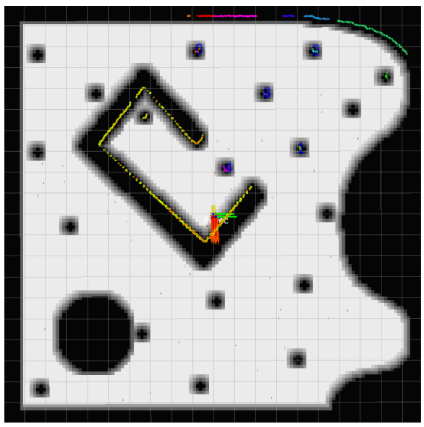
如果提高在机器人附近新增粒子比例并使地图进一步精细化可能可以解决该问题。

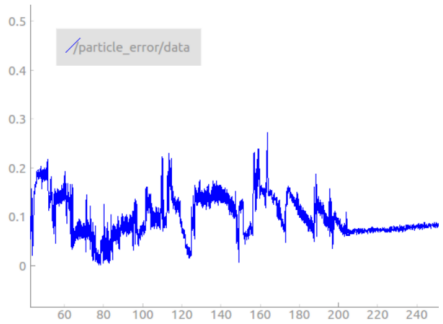
Case2.不同似然地图的比较

不同的高斯MAX扩散地图:



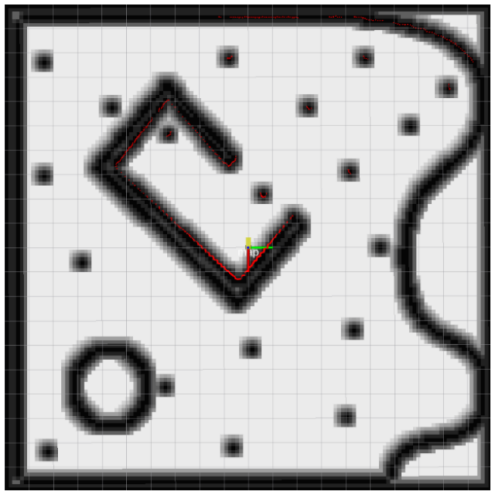






就误差图像来看两者差别不大，更精细的地图会出现误差突然下降的情况，这是因为重采样的新粒子突然采到了距离真值很近的位姿；而更模糊的地图粒子的权重变化较为平滑，故整体误差曲线也较为平滑。

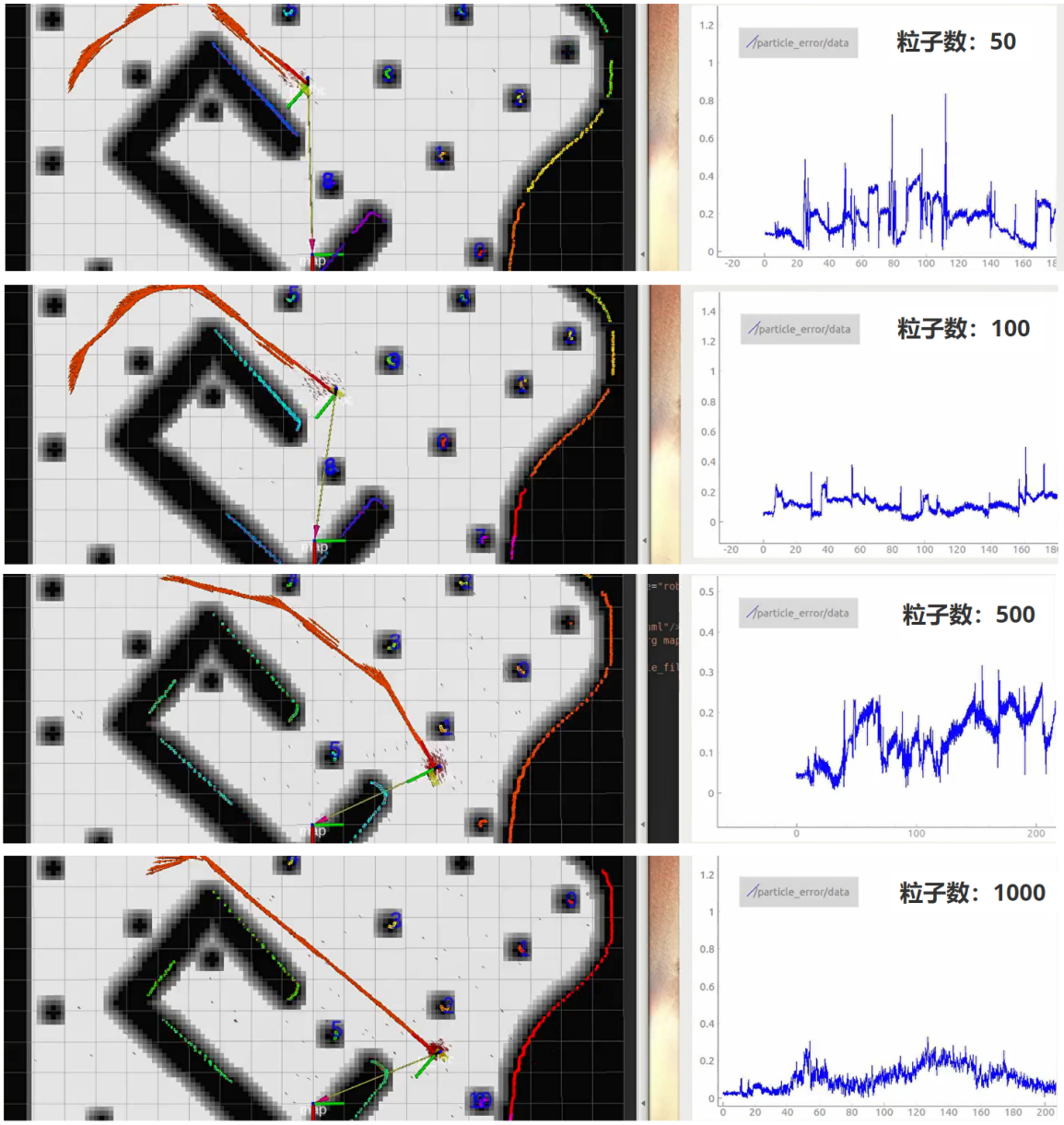
高斯扩散地图与边缘高斯地图：



边缘高斯地图误差减小，说明这种似然地图的构建方式能够减小粒子的激光点被赋予错误置信度的情况。

Case3.不同粒子数的比较

分别设置粒子数为50，100，500，1000，比较小车行驶一圈后的误差，结果如下：



其中每种情况绘图的坐标系稍有不同，看图时需注意。可以发现，随着粒子数增加，整体误差减小，跳变减少。粒子数少时易跳变的原因应是处在真值附近的粒子过少，重采样的新粒子对整体影响较大。

Case4.与EKF的比较

EKF无法解决机器人绑架的问题，但整体精度高于粒子滤波；粒子滤波有很好的全局搜索能力，在定位大幅偏离后还能够收敛到真值：