# 智能机器人概论作业三报告

2100013104 尹骄洋

December $17^{st}$ 2023

## 1 实验目的

- 利用激光及定位数据，计算占有栅格地图（OGM）

## 2 实验原理

### 2.1 激光点坐标从雷达坐标系转换为全局坐标系

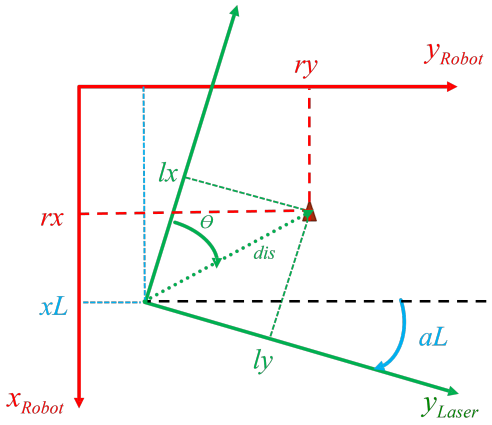


Figure 1: Laser->Robot

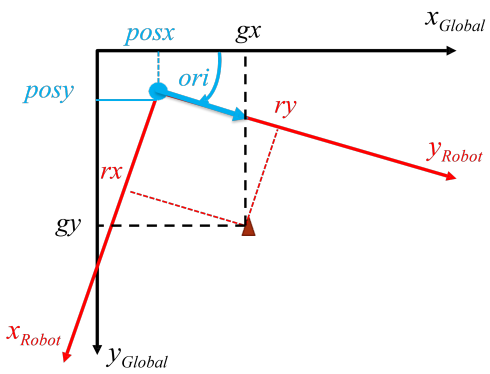


Figure 2: Robot->Global

首先，将激光点坐标从雷达坐标系转换为机器人坐标系，然后从机器人坐标系转换为全局坐标系。对照以上两图进行坐标系转换部分代码的书写。详见实验代码4.1部分。

### 2.2 占有栅格地图法（OGM）

将环境用栅格表示，基于 Binary Bayes Filter 估计每个栅格被障碍物占据的概率。

## 3 实验数据

- 激光扫描数据 a20160801135224.lms1
- 车辆行驶轨迹 a-XW-20160801135224.nav

## 4 实验代码

### 4.1 占有栅格地图法

```
//=========================================================
//-----------------Please edit below---------------------
//=========================================================

//目标：对每条激光束更新全局地图（Occupency Gird Map）
//下列步骤中需要编程的有 3，4，5，8

//1. 分别定义激光点在全局坐标系（GPS），机器人坐标系，
//激光雷达坐标系中的坐标变量
double gx, gy;//激光点在全局坐标系中的位置 单位 m
double rx, ry;//激光点在机器人坐标系中的位置 单位 m
double lx, ly;//激光点在激光雷达坐标系中的位置 单位 m

//2. 若激光点返回测距值为 0，则为无效数据，将其滤除。
if(inputdata_0.front()->data[i] == 0)
    continue;

//3. 计算激光点在激光雷达坐标系下的位置，
//并根据参数 inputparams_0.front()->isReverse 判断是否将 lx 取相反数。
double dis=inputdata_0.front()->data[i]/inputparams_0.front()->unit;
//计算得到单个激光点的距离返回值
```

```
double angle = i*inputparams_0.front()->res*3.14159261314/180.0;
//计算得到当前处理激光束在激光雷达坐标系中角度
lx = dis * cos(angle);
ly = dis * sin(angle);
if (inputparams_0.front()->isReverse)
    lx = -lx;

//4. 进行 激光雷达坐标系-> 机器人坐标系 变换
//xL,yL,aL 定义见课件，其在程序中对应变量为
//inputparams_0.front()->xL, inputparams_0.front()->yL,
//inputparams_0.front()->aL
double laser_aL_rad = inputparams_0.front()->aL*3.14159261314/180.0;
rx = inputparams_0.front()->xL
    - lx * cos(laser_aL_rad) + ly * sin(laser_aL_rad);
ry = inputparams_0.front()->yL
    + lx * sin(laser_aL_rad) + ly * cos(laser_aL_rad);

//5. 进行 机器人坐标系-> 全局坐标系 变换
//机器人航向角 ori 定义见课件，对应变量为 inputdata_1.front()->ori
double orien = inputdata_1.front()->ori;
gx = inputdata_1.front()->x + ry * cos(orien) + rx * sin(orien);
gy = inputdata_1.front()->y + ry * sin(orien) - rx * cos(orien);

//6. 计算激光点在地图中的位置 单位 pixel
int mapx, mapy;
mapx = (gx) / params->mapRes-params->ZeroX;
mapy = (gy) / params->mapRes-params->ZeroY;

//7. 调用函数计算当前激光束途经的栅格点坐标序列（地图中的像素坐标），
//存储在 std::vector<Location>locationVec 中，
//各栅格点坐标按照激光发射方向排列。
int location_mapx, location_mapy;
location_mapx = inputdata_1.front()->x/params->mapRes-params->ZeroX;
location_mapy = inputdata_1.front()->y/params->mapRes-params->ZeroY;
```

```
std::vector<Location> locationVec;
Location startPos(location_mapx, location_mapy);
Location endPos(mapx,mapy);
CalcShortestDistance(startPos, endPos, locationVec);

//8. 根据 OGM 地图生成方法制图。
//地图存储在 vars->map[][] 中，坐标轴为 vars->map[y][x]。
//定义每个栅格 vars->map[y][x] 取值上下界。
double upthres = 300;
double lowthres = -300;
//逐个更新当前激光束途经的栅格点坐标，
//params->logodd_free, params->logodd_occu
//为检测到当前坐标为无障碍/有障碍的更新值。
for (std::vector<Location>::iterator c=locationVec.begin();
    c!=locationVec.end();c++){
    if((*c).x >= 0 && (*c).x < params->mapWidth
        && (*c).y >= 0 && (*c).y < params->mapHeight){
        if (c+1 == locationVec.end())
            vars->map[(*c).y][(*c).x] += params->logodd_occu;
        else
            vars->map[(*c).y][(*c).x] += params->logodd_free;
        // set boundaries for each value
        if(vars->map[(*c).y][(*c).x] > upthres)
            vars->map[(*c).y][(*c).x] = upthres;
        else if(vars->map[(*c).y][(*c).x] < lowthres)
            vars->map[(*c).y][(*c).x] = lowthres;
    }
}
//===================================================
//-----------------Please edit above-----------------
//===================================================
```

### 4.2 栅格投票法

这里只记录绘制地图部分的实现，坐标系转换部分和上面相同。

```
int hits[600][640]={0};
int misses[600][640]={0};
//逐个更新当前激光束途经的栅格点坐标，params->logodd_free，params->logodd_occu
//为检测到当前坐标为无障碍/有障碍的更新值。
for (std::vector<Location>::iterator c=locationVec.begin();
    c!=locationVec.end();c++){
    if((*c).x >= 0 && (*c).x < params->mapWidth
        && (*c).y >= 0 && (*c).y < params->mapHeight){
        if (c+1 == locationVec.end())
            hits[(*c).y][(*c).x] ++;
        else misses[(*c).y][(*c).x]++;
        // limit to [-300,300]
        vars->map[(*c).y][(*c).x] = -300 +
        600*(hits[(*c).y][(*c).x]/(hits[(*c).y][(*c).x]+misses[(*c).y][(*c).x]));
    }

}
```

## 5 地图可视化

最后生成地图如图3。发现需要将 speed 调整为 3 才能输出和示例图片类似的图片，否则生成地图会比较糊，如图4所示。speed 变量的增大可以减慢模拟，地图识别速度放慢，准确度越好。

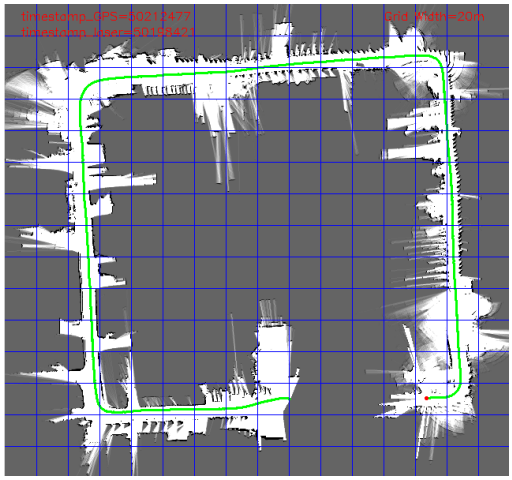

Figure 3: 正确结果

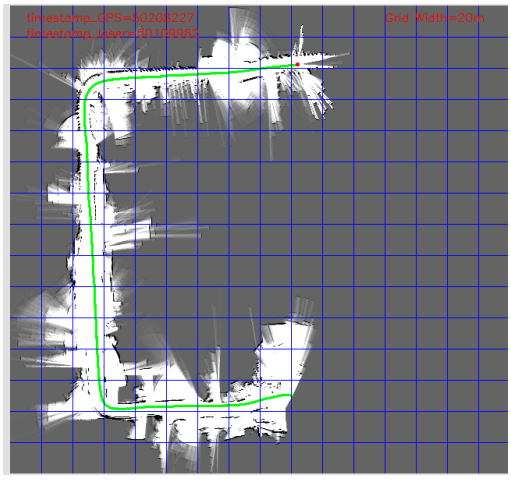

Figure 4: 有问题的结果

## 6 三种方法对比

- 简易投票法5：利用简单的投票策略来确定每个栅格的状态。简单直观，但无法很好地处理传感器数据的噪声和不确定性，任何噪声都可能被识别成障碍物。且简易投票法只能展示出来障碍物边界，缺乏细节。作业二使用简体投票法的可视化地图如图5。
- 占有栅格地图法6：使用概率值来表示每个栅格被占有的可能性，从而可以更好地处理传感器数据的不确定性。相较于简易投票法，它可以展示出楼道的可行走区域，细节更加丰富。
- 栅格投票法7：统计的是每个栅格的反射率（特性），相对来说鲁棒性也会更强，细节也更为丰富。由于侧重的是栅格的反射率，所以在遇到玻璃隔断时，可能呈现结果更像是不透光的墙。而占有栅格地图法则可能将玻璃隔断视为不存在。栅格投票法的实现在4.2给出。


Figure 5: 简易投票法

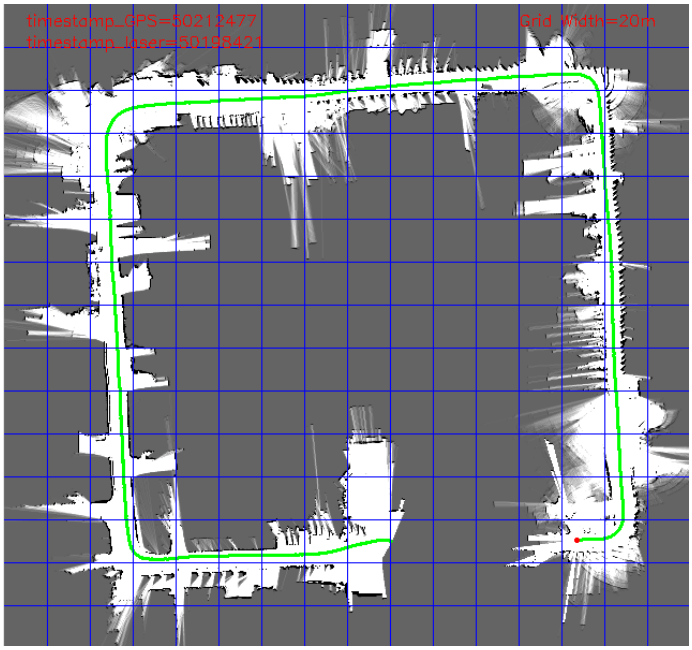


Figure 6: 占有栅格地图法

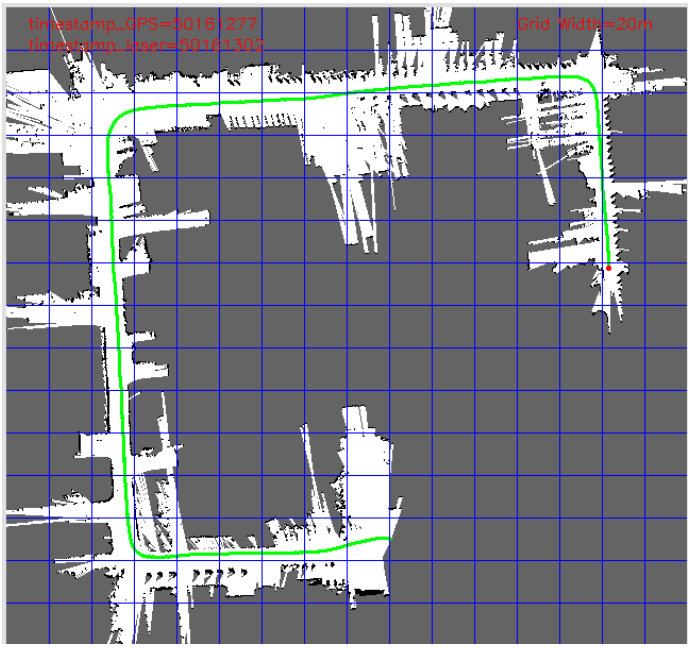


Figure 7: 栅格投票法