Jindong_Shi 2022.05.14

激光 SLAM 第 7 次作业

1. 补充代码,通过覆盖栅格建图算法进行栅格地图构建

代码运行时可以输入 1，2 或 3，来使用三种不同的建图方法:

```
std::cout << "Please enter the mapping method you want to use:" << std::endl;
std::cout << "[1: occupancy grid, 2: count model, 3: TSDF]" << std::endl;
std::cin >> mapping_method;
```

方法 1 为覆盖栅格建图法(occupancy grid)，代码如下:

```
//start of TODO 对对应的map的cell信息进行更新. (1,2,3题内容)
GridIndex pointIndex = ConvertWorld2GridIndex(world_x, world_y);
std::vector<GridIndex> trace_line = TraceLine(robotIndex.x, robotIndex.y, pointIndex.x, pointIndex.y);

if(mapping_method == 1)
{

    // 建图方法一: 使用覆盖栅格建图算法 (使用代码内给定的函数TraceLine)
    // 被激光穿过的栅格
    for(int k = 0; k < trace_line.size(); ++k)
    {
        if(isValidGridIndex(trace_line[k]))
        {
            int linear_index = GridIndexToLinearIndex(trace_line[k]);
            pMap[linear_index] += mapParams.log_free;
            pMap[linear_index] = pMap[linear_index] > mapParams.log_min
                                 ? pMap[linear_index] : mapParams.log_min;
        }
    }

    // 被激光击中的栅格
    if(isValidGridIndex(pointIndex))
    {
        int linear_index = GridIndexToLinearIndex(pointIndex);
        pMap[linear_index] += mapParams.log_occ;
        pMap[linear_index] = pMap[linear_index] < mapParams.log_max
                             ? pMap[linear_index] : mapParams.log_max;
    }
}
```

该建图方法最简单，每次有新的激光帧数据传入的时候，只需要在原有的 Log-Odd 项构建的 pMap 上不断线性累加逆观测模型的值即可:

occupancy_grid_mapping$(\{l_{t-1,i}\}, x_t, z_t)$**:**

1: *for all cells* m_i *do*
2: *if* m_i *in perceptual field of* z_t *then*
3: $l_{t,i} = l_{t-1,i} + \text{inv\_sensor\_model}(m_i, x_t, z_t) - l_0$
4: *else*
5: $l_{t,i} = l_{t-1,i}$
6: *endif*
7: *endfor*
8: *return* $\{l_{t,i}\}$

根据下面我们设定的 log_free 和 log_occ 参数。每次当一个栅格被激光束穿过(即 free 的情况下)，对应的 Log-Odd 项会减小 1，当它被击中(即 occupied 的情况下)， 对应的 Log-Odd 项会增加 2。

```
//每次被击中的log变化值，覆盖栅格建图算法需要的参数
mapParams.log_free = -1;              // 逆观测模型
mapParams.log_occ = 2;

//每个栅格的最大最小值.
mapParams.log_max = 100.0;
mapParams.log_min = 1.0;
```

并且保证其范围在 1 ~100 之间。

2. 将第 1 题代码改为通过计数建图算法进行栅格地图构建

方法 2 为计数建图法，代码如下:

```
else if(mapping_method == 2)
{
    // 建图方法3: 计数法
    // 被激光穿过的栅格
    for(int k = 0; k < trace_line.size(); ++k)
    {
        if(isValidGridIndex(trace_line[k]))
        {
            int linear_index = GridIndexToLinearIndex(trace_line[k]);
            pMapMisses[linear_index] += 1;
        }
    }

    // 被激光击中的栅格
    if(isValidGridIndex(pointIndex))
    {
        int linear_index = GridIndexToLinearIndex(pointIndex);
        pMapHits[linear_index] += 1;
    }

}
```

与方法 1 类似，唯一不同的是，我们这里会累加栅格击中和穿过的次数，据此可以计算出该栅格的占有概率。代码如下:

```
else if(mapping_method == 2)
{
    // 方法2: 计数建图
    for(int k = 0; k < mapParams.width * mapParams.height; ++k)
    {
        if(pMapHits[k] != 0 || pMapMisses[k] != 0)     // 只处理被击中过或被miss过的栅格位置
        {
            double occu_rate = pMapHits[k]/(pMapHits[k] + pMapMisses[k]);
            pMap[k] = static_cast<u_char>(std::ceil(occu_rate * mapParams.log_max));
        }
    }
}
```

由于最后输出的 pMap 的值的范围为 1~100，所以我们这里需要将占有率也转换到 1~100 然后保存到 pMap 中。

3. 将第 1 题代码改为通过 TSDF 建图算法进行栅格地图构建

方法 3 为计数建图法 TSDF 建图法，代码如下:

```
else if(mapping_method == 3)
{
    // 方法3: TSDF建图
    double TSDF, tsdf;              // TSDF: 融合更新后的值    tsdf: 当前值
    double x, y;                    // 世界坐标系下的坐标
    double delta_x, delta_y;
    double dist_xy;                 // 点(x,y)距离传感器原点的距离
    double t = 0.5;                 // 截断距离
    for(int k = 0; k < trace_line.size(); ++k)
    {
        if(isValidGridIndex(trace_line[k]))
        {
            //栅格点需要先转换为世界坐标，才能接下去计算实际距离
            ConvertGridIndex2World(trace_line[k], x, y);
            //计算点(x,y)距离传感器原点的实际距离
            delta_x = x - robotPose(0);
            delta_y = y - robotPose(1);
            dist_xy = std::pow(std::pow(delta_x, 2) + std::pow(delta_y, 2), 0.5);
            tsdf = std::max(-1.0, std::min(1.0, (dist-dist_xy)/t));
            //二维栅格的一维序号
            int pose_num = GridIndexToLinearIndex(trace_line[k]);
            //TSDFi(x)
            pMapTSDF[pose_num] = (pMapW[pose_num]*pMapTSDF[pose_num] + tsdf)/(pMapW[pose_num]+1);
            //Wi(x)
            ++pMapW[pose_num];
        }
    }
}
```

TSDF 法其实是一种将多次观测进行融合更新的方法。
我们会先计算机器人位姿和激光点之间被激光穿过的点的 sdf 值:

$$sdf_i(x) = laser_i(x) - dist_i(x)$$

$laser_i(x)$表示激光测量距离

$dist_i(x)$表示栅格离传感器原点的距离

然后根据截断距离 t 将其截断为 tsdf:

$$tsdf_i(\mathbf{x}) = \max(-1, \min(1, \frac{sdf_i(\mathbf{x})}{t}))$$

然后我们根据权重因子 w 来融合历史激光帧的 TSDF 场和当前激光束的 tsdf 值。我们这里假设每个激光束的权重均为 1，那么融合结果就是简单的算术平均:

$$TSDF_i(\mathbf{x}) = \frac{W_{i-1}(\mathbf{x})TSDF_{i-1}(\mathbf{x}) + w_i(\mathbf{x})tsdf_i(\mathbf{x})}{W_{i-1}(\mathbf{x}) + w_i(\mathbf{x})}$$
$$W_i(\mathbf{x}) = W_{i-1}(\mathbf{x}) + w_i(\mathbf{x})$$

一旦构建好 TSDF 场之后，我们就可以利用其中符号变化的栅格，通过对其进行插值来得到障碍物曲面的精确位置，代码如下:

```
else if(mapping_method == 3)
{
    // 方法3: TSDF建图
    // 使用插值计算障碍物位置
    for(int m = 0; m < mapParams.height-2; m++)     //上下方向(即x方向)上搜索
    {
        for(int n = 0; n < mapParams.width-2; n++)  //左右方向(即y方向)上搜索
        {
            GridIndex index;
            index.SetIndex(m, n);
            int linear_index = GridIndexToLinearIndex(index);       // 遍历到的点的一维搜索序号
            int linear_x = linear_index + mapParams.width;          // 沿着x轴方向向下移动一格
            int linear_y = linear_index + 1;                        // 沿着y轴方向向右移动一格

            //计算 点A 和 点B，用于插值
            double A_x, A_y;                        // 遍历到的点 A = (A_x, A_y)
            ConvertGridIndex2World(index, A_x, A_y);
            double B_x, B_y;                        // 遍历到的点的右下方的点 B = (B_x, B_y)
            index.SetIndex(m+1, n+1);
            ConvertGridIndex2World(index, B_x, B_y);

            double a, bx, by, x, y;
            a = pMapTSDF[linear_index];
            bx = pMapTSDF[linear_x];    // 沿着x轴方向向下移动一格的TSDF值
            by = pMapTSDF[linear_y];    // 沿着y轴方向向下移动一格的TSDF值

            if( a * by < 0)             // 判断y方向上的TSDF值是否存在符号变化
            {
                x = A_x;                                // x方向不插值
                y = interpolation(A_y, B_y, a, by);     // 对 A_y和B_y 根据 a和by 进行插值，得到y方向上的值
                pMap[GridIndexToLinearIndex(ConvertWorld2GridIndex(x,y))] = 100;
            }
            else if( a * bx < 0 )       // 判断x方向上的TSDF值是否存在符号变化
            {
                x = interpolation(A_x, B_x, a, bx);     // 对 A_x和B_x 根据 a和bx 进行插值，得到y方向上的值
                y = A_y;                                // y方向不插值
                pMap[GridIndexToLinearIndex(ConvertWorld2GridIndex(x,y))] = 100;
            }

        }
    }
}
```

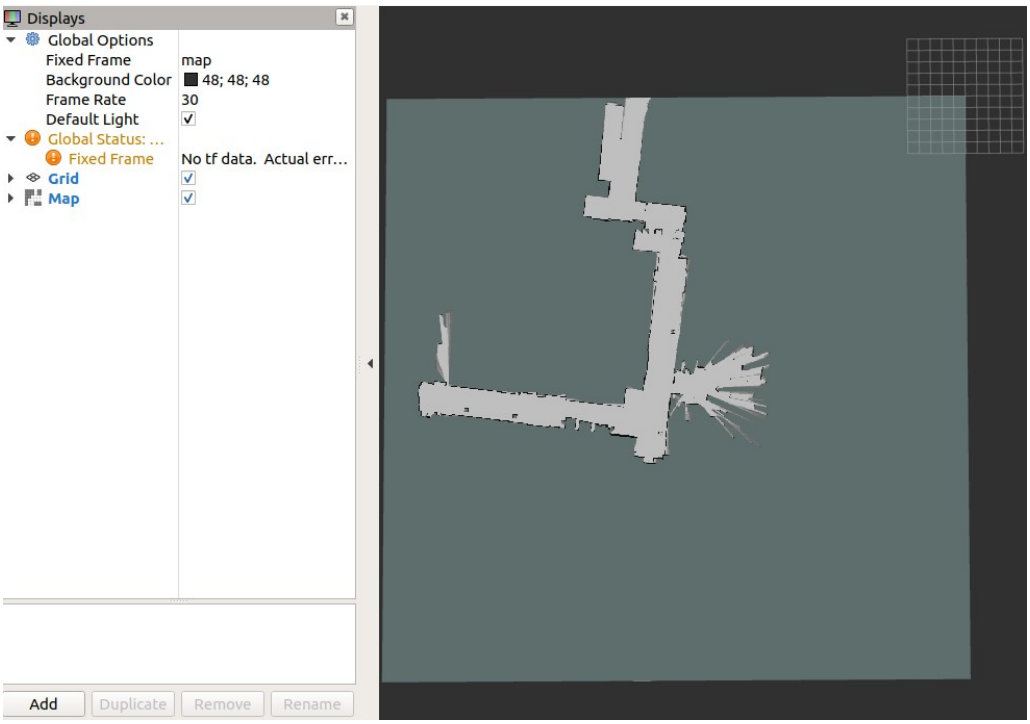
插值函数如下，这里对位置值 A 和 B 根据 TSDF 场值 a 和 b 来进行位置插值。：

```
//插值函数
double interpolation(double A, double B, double a, double b)
{
    double value = (b*A - a*B) / (b-a);
    return value = a == b ? A : value;
}
```

三种方法的建图效果如下所示:

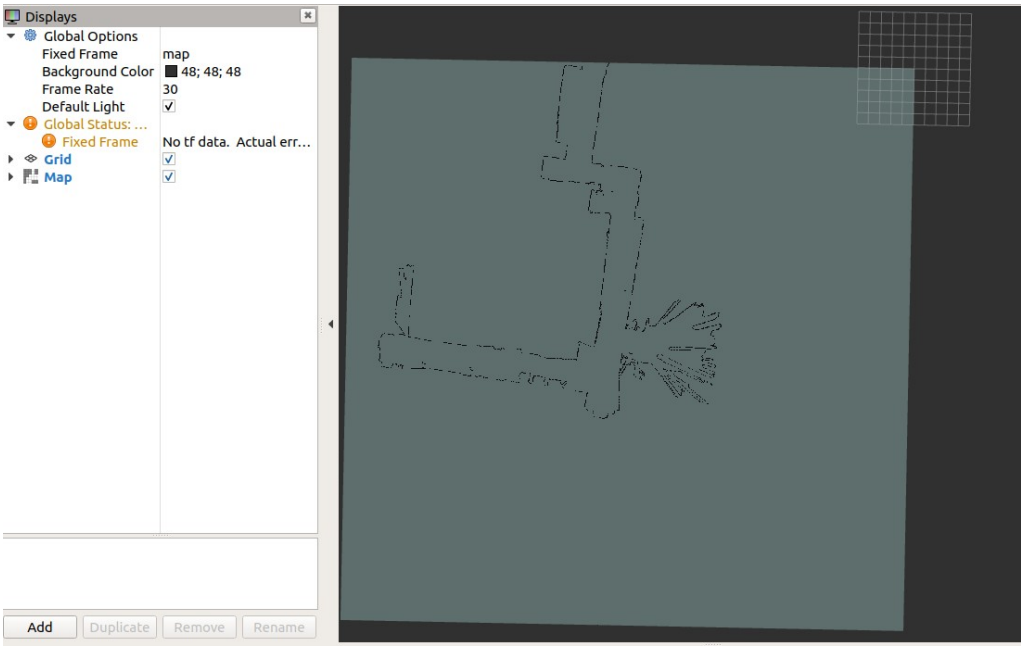
方法 1: 覆盖栅格法

```
iindong@iindong-ThinkPad-X1-Carbon:~/Lidar_SLAM/Exercise/ch7/HW7/O
ccupanyMappingProject$ rosrun occupany_mapping occupany_mapping
Read Pose Good!!!
Read Angle good:1081
XX:1081
Read Laser Scans Good!!!!
**********************************
Pose number: 3700
Angle number at each pose: 1081
Range number at each pose: 1081
**********************************
Please enter the mapping method you want to use:
[1: occupancy grid, 2: count model, 3: TSDF]
1
开始建图，请稍后...
建图完毕
```



方法 2: 计数法

方法 3: TSDF 算法



其中方法 1 和方法 2 建图用时相对较短，方法 3(TSDF)用时最长。

4. 简答题,开放性答案:总结比较课堂所学的 3 种建图算法的优劣

a) 覆盖栅格法(Occupancy Grid)和计数建图法(Count Model)两者较为相似。优点是实现简单，计算用时较少。缺点是对噪声敏感，计算得到的障碍物曲面可能占据多个栅格。

b) TSDF 法使用加权最小二乘对多帧数据进行融合。优点是得到的曲面的位置更为准确，不易受噪声干扰，且构建的障碍物曲面最多只有一个栅格厚度。缺点是实现复杂，建图计算用时较长。