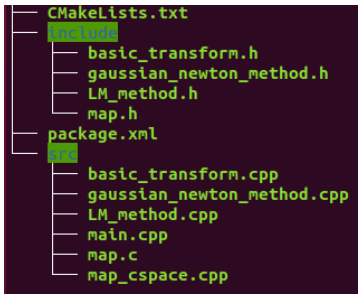
Jindong_Shi 2022.04.21

激光 SLAM 第 5 次作业

1. 补充代码,实现 GN_LM_scanmatcher 模块

src 文件夹内文件罗列如下:



main.cpp 主要实现激光点云数据的读取，通过里程计获取帧间相对位姿的初始值:

```
// 初始解为上一帧激光位姿+运动增量，这里运动增量由轮式里程计增量得来
Eigen::Vector3d deltaPose = nowPose - m_odomPath.back();     // 轮式里程计测得的位姿变化量
deltaPose(2) = GN_NormalizationAngle(deltaPose(2));         // 角度正则化
```

使用上一帧的激光点云数据和估计的位姿来构建 map (似然场):

```
// 使用上一帧激光点云数据生成地图（似然场）
map_t* map = CreateMapFromLaserPoints(m_prevLaserPose,m_prevPts,0.05);     // map resolution = 0.1, 相当于1米是1/0.1=10个栅格

//      --------->              pose_laser  +  T_laser_world *  T_world_odom  *  pose_odom
//                                    |              |              |                |
Eigen::Vector3d finalPose = m_prevLaserPose + R_laser * R_odom.transpose() * deltaPose;
finalPose(2) = GN_NormalizationAngle(finalPose(2));         // 角度正则化
```

调用 GN 或者 LM 算法，来进行帧间匹配:

```
if(optimization_method == 1)
    GaussianNewtonOptimization(map,finalPose,nowPts);          // 参数:（前一帧激光数据构成的似然场，估计位姿作为初始值(亦是返回位姿)，当前帧点云）
else if(optimization_method == 2)
    LevenbergMarquardtOptimization(map,finalPose,nowPts);
else
    std::cout << "No optimization method is selected." << std::endl;
```

最后将估计的位姿保存到路径当中，发布出去。

map.h: 声明了 map (似然场) 和 cell(栅格)的数据结构 map_t 和 map_cell_t。其中 map_t 包含了多个成员变量，其中包含指向多个栅格的指针。并且该头文件还宏定义了世界坐标和地图坐标之间相互的转换关系。

map.c: 定义了函数 map_t *map_alloc( ) 和 map_cell_t *map_get_cell(...)来初始化 map 和 cell。其中最巧妙的地方在于可以利用栅格在地图中的坐标来访问 map 中 cells 对应的内存空间。

```
// 利用地图坐标Index访问Index对应的内存空间
cell = map->cells + MAP_INDEX(map, i, j);
```

map_cspace: 目的主要是计算得到目标 score 函数。简单来说，就是将 map 中被激光点打到的栅格定义为障碍物，该栅格离障碍物的距离即为 0，分数 score 对应的为最大 1。将该障碍物栅格放入优先级队列 Q 当中，使用宽度优先算法，通过不断拓展遍历 Q 当中未被访问过的栅格的上下左右四个方向的栅格，来计算 map 当中各个栅格的 score。这个 score 是高斯分布的，离障碍物的距离 occ_dist 越近，分数越高。如果该栅格离障碍物的距离大于障碍物最大作用距离,不做处理。我认为该算法最精妙的地方在于使用了优先级队列和宽度优先算法，实现了对 map 中栅格的快速不重复的遍历。

basic_transform.cpp: 主要定义了

a) 用激光雷达数据创建势场的函数 map_t* CreateMapFromLaserPoints(…)，

b) 对势场进行插值和计算对应位置梯度的函数 Eigen::Vector3d InterpMapValueWithDerivatives(...)

c) 计算得到 H 和 b 的函数 double ComputeHessianAndb(…)

对于插值和计算梯度的函数 b), 代码如下:

```
Eigen::Vector3d InterpMapValueWithDerivatives(map_t* map, Eigen::Vector2d& coords)
{
    Eigen::Vector3d ans;

    double x, y;            // world coordinate
    x = coords(0);
    y = coords(1);

    double i, j;            // map coordinate
    int i0, j0, i1, j1;
    i = MAP_GXWX_DOUBLE(map, x);
    j = MAP_GXWX_DOUBLE(map, y);
    i0 = MAP_GXWX(map, x);
    j0 = MAP_GXWX(map, y);
    i1 = i0 + 1;
    j1 = j0 + 1;

    double Z1, Z2, Z3, Z4;
    Z1 = map->cells[MAP_INDEX(map, i0, j0)].score;          // 左上角插值点
    Z2 = map->cells[MAP_INDEX(map, i1, j0)].score;          // 右上角插值点
    Z3 = map->cells[MAP_INDEX(map, i1, j1)].score;          // 右下角插值点
    Z4 = map->cells[MAP_INDEX(map, i0, j1)].score;          // 左下角插值点
    double a, b, c, d;
    a = (j - j0)/(j1 - j0);
    b = (j1 - j)/(j1 - j0);
    c = (i - i0)/(i1 - i0);
    d = (i1 - i)/(i1 - i0);

    double L, dL_i, dL_j;
    L = a * (c * Z3 + d * Z4) + b * (c * Z2 + d * Z1);
    dL_i = a * (Z3 - Z4)/(i1 - i0) + b * (Z2 - Z1)/(i1 - i0);
    dL_j = (c * Z3 + d * Z4)/(j1 - j0) - (c * Z2 + d * Z1)/(j1 - j0);

    // 将导数从map转到world
    dL_i = dL_i / map->resolution;
    dL_j = dL_j / map->resolution;

    ans << L, dL_i, dL_j;



    return ans;
}
```

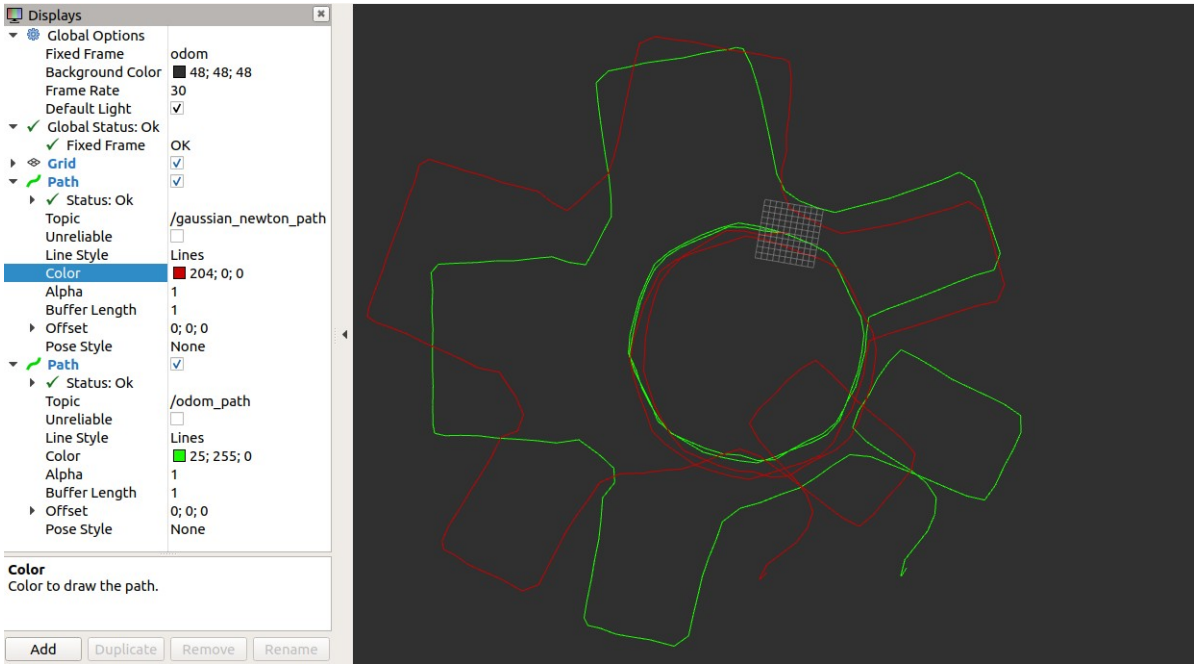
计算 H 和 b 的函数 c)，代码如下:

```
double ComputeHessianAndb(map_t* map, Eigen::Vector3d now_pose,
                          std::vector<Eigen::Vector2d>& laser_pts,
                          Eigen::Matrix3d& H, Eigen::Vector3d& b)
{
    H = Eigen::Matrix3d::Zero();
    b = Eigen::Vector3d::Zero();

    double theta = now_pose(2);
    Eigen::Matrix3d T_laser = GN_V2T(now_pose);                              // now_pose对应的变换矩阵:  T_laser_world
    double currentResidual = 0;                                              // 清空currentResidual

    for(int k = 0; k != laser_pts.size(); ++k)
    {
        double x, y;                                                         // 位姿变换前的2D点云
        x = laser_pts[k](0);
        y = laser_pts[k](1);

        Eigen::Matrix<double, 2, 3> d_Si_T;
        d_Si_T << 1, 0, -std::sin(theta) * x - std::cos(theta) * y,
                  0, 1,  std::cos(theta) * x - std::sin(theta) * y;

        // 将该激光点云 使用估计的位姿 转换到 世界系坐标系下
        Eigen::Vector2d xy_trans = GN_TransPoint(laser_pts[k], T_laser);     // 位姿变换后的2D点云

        Eigen::Vector2d coords(xy_trans(0), xy_trans(1));
        Eigen::Vector3d ans = InterpMapValueWithDerivatives(map, coords);
        double M = ans(0);
        double error = 1 - M;
        Eigen::Matrix<double, 1, 2> dM;
        dM << ans(1), ans(2);
        Eigen::Matrix<double, 1, 3> J = dM * d_Si_T;

        b += J.transpose() * error;
        H += J.transpose() * J;

        currentResidual += error * error;     // 计算代价函数增长情况
    }

    return currentResidual;
}
```

gaussian_newton_method.cpp: 使用了高斯牛顿算法来进行优化。代码如下:

```cpp
void GaussianNewtonOptimization(map_t* map,Eigen::Vector3d& init_pose,std::vector<Eigen::Vector2d>& laser_pts)
{
    int maxIteration = 20;                          // 规定最大迭代次数
    Eigen::Vector3d now_pose = init_pose;           // 使用init_pose初始化now_pose

    int iterationTimes = 0;
    double lastResidual = 0;

    for(int i = 0; i < maxIteration;i++)
    {
        iterationTimes = i;
        double currentResidual = 0;

        Eigen::Matrix3d H;
        Eigen::Vector3d b;

        currentResidual = ComputeHessianAndb(map, now_pose, laser_pts, H, b);

        Eigen::Vector3d delta_pose;

        double lambda = 10;                     //给H加上lambda，保证H具有良好的正定性，减小病态
        H += lambda * Eigen::Matrix3d::Identity();
        delta_pose = H.ldlt().solve(b);         // 由于H可以保证为半正定矩阵，所以可以用ldlt求解
        delta_pose(2) = GN_NormalizationAngle(delta_pose(2));

        if (std::isnan(delta_pose(0)) || std::isnan(delta_pose(1)) || std::isnan(delta_pose(2))) {
            std::cout << "result is nan!" << std::endl;
            break;
        }

        if (i > 0 && currentResidual > lastResidual) {
            // 如果代价函数增长，立刻停止迭代
            break;
        }

        // 使用李代数更新
        now_pose += delta_pose;
        // 更新代价函数
        lastResidual = currentResidual;
    }
    init_pose = now_pose;
    std::cout << "Iterations: " << iterationTimes << " times." << std::endl;
}
```

我们将 map 的 resolution 设为 0.05, 将 sigma 设为 1，得到的轨迹如下，其中绿色为里程计，红色为激光雷达估计的轨迹:



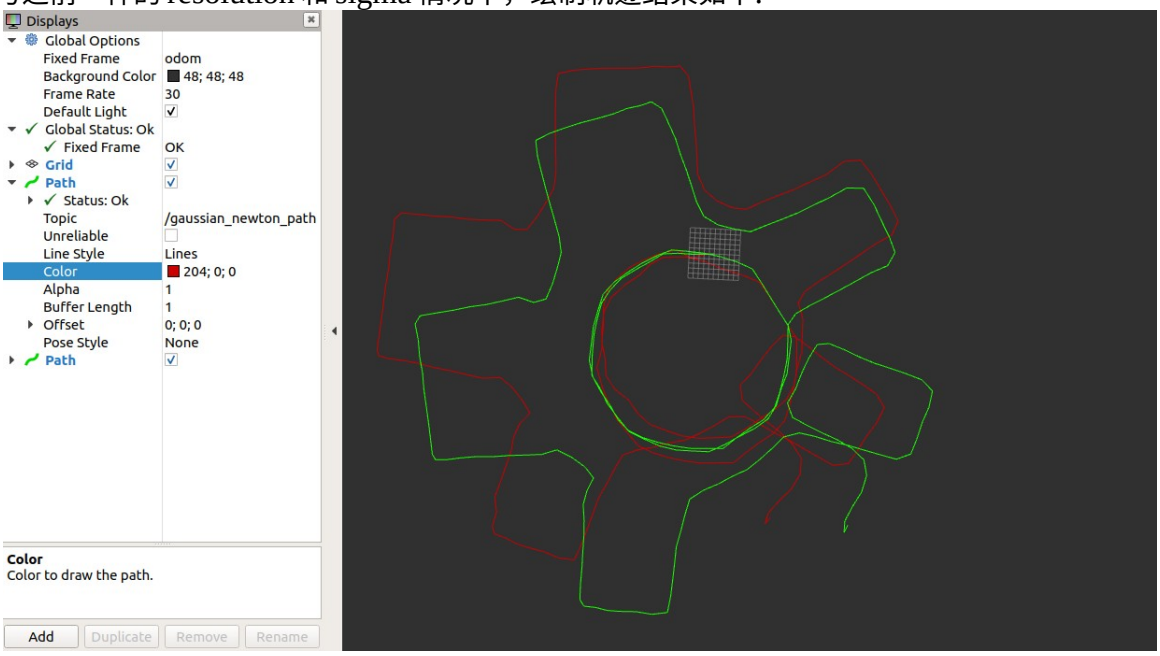
我们发现两者差异还是比较大的，不知道哪个轨迹更为准确。

2. 简答题,开放性答案:提出一种能提升第一题激光匹配轨迹精度的方法,并解释原因;

上述我们使用了高斯牛顿法，高斯牛顿法的缺点之一是 Hessian 矩阵可能奇异或病态，导致算出的位姿时好时坏。为此，在上面的算法中，我给 H 矩阵加上了 lambda, 防止它病态。我们也可以使用 LM 算法，实现优化算法更快更好的收敛。

LM_method.cpp: 实现了 LM 算法。主要借鉴了 VIO 课程中介绍的 LM 算法。

使用与之前一样的 resolution 和 sigma 情况下，绘制轨迹结果如下:



3. 阅读论文 The Normal Distributions Transform: A New Approach to Laser Scan Matching，回答以下问题:(2 分)

(1) NDT 的优化函数(score)是什么?

The mapping according to $\mathbf{p}$ could be considered optimal, if the sum evaluating the normal distributions of all points $\mathbf{x}'_i$ with parameters $\mathbf{\Sigma_i}$ and $\mathbf{q_i}$ is a maximum. We call this sum the `score` of $\mathbf{p}$. It is defined as:

$$\mathtt{score}(\mathbf{p}) = \sum_i \exp\left(\frac{-(\mathbf{x}'_\mathbf{i} - \mathbf{q_i})^\mathbf{t} \mathbf{\Sigma}_\mathbf{i}^{-\mathbf{1}} (\mathbf{x}'_\mathbf{i} - \mathbf{q_i})}{2}\right). \quad (3)$$

令 $q = X_i^{'} - q_i$，则

$$score_i = \exp\left(-\frac{q^T \Sigma_i^{-1} q}{2}\right)$$

根据论文[1]的描述，NDT 的优化函数可描述成以上得分 score 函数的最大化问题，可加上负号转化为一般的最小化问题:

$$\max \sum score_i \quad \Longrightarrow \quad \min \sum -score_i$$

(2) 简述 NDT 根据 score 函数进行优化求解的过程。

根据论文[1]，NDT 优化求解算法流程如下:

where $(t_x, t_y)^t$ describes the translation and ϕ the rotation between the two frames. The goal of the scan alignment is to recover these parameters using the laser scans taken at two positions. The outline of the proposed approach, given two scans (the *first* one and the *second* one), is as follows:

1) Build the NDT of the first scan.
2) Initialize the estimate for the parameters (by zero or by using odometry data).
3) For each sample of the second scan: Map the reconstructed 2D point into the coordinate frame of the first scan according to the parameters.
4) Determine the corresponding normal distributions for each mapped point.
5) The score for the parameters is determined by evaluating the distribution for each mapped point and summing the result.
6) Calculate a new parameter estimate by trying to optimize the score. This is done by performing one step of Newton's Algorithm.
7) Goto 3 until a convergence criterion is met.

由于 NDT 目标函数的 Hessian 比较好求，所以论文中用了普通的加上 lambda 的一步牛顿法来作为优化算法，迭代得到最佳的位姿变换，当然我们也可以使用高斯牛顿，LM，Dogleg 等其他优化算法。

4. 机器人在 XY 方向上进行 CSM 匹配。下图左为机器人在目标区域粗分辨率下 4 个位置的匹配得分,得分越高说明机器人在该位置匹配的越好,下图右为机器人在同一块地图细分辨率下每个位置的匹配得分(右图左上 4 个小格对应左图左上一个大格,其它同理)。如果利用分枝定界方法获取最终细分辨率下机器人的最佳匹配位置,请简述匹配和剪枝流程。

85	99
98	96

左图：机器人在粗分辨率地图下各个位置的匹配得分

41	43	58	24
76	83	87	73
86	95	89	68
70	65	37	15

右图：机器人在细分辨率地图下各个位置的匹配得分（细分辨率下的匹配最高分小于等于相应粗分辨率位置的最高分）

参考文献[4][5]详细描述了分支定界(branch and bound)算法。

Algorithm 2 Generic branch and bound

```
best_score ← −∞
C ← C_0
while C ≠ ∅ do
  Select a node c ∈ C and remove it from the set.
  if c is a leaf node then
    if score(c) > best_score then
      solution ← n
      best_score ← score(c)
    end if
  else
    if score(c) > best_score then
      Branch: Split c into nodes C_c.
      C ← C ∪ C_c
    else
      Bound.
    end if
  end if
end while
return best_score and solution when set.
```

简单来说，分支定界的主要思想就是，把全部可行的解空间不断分割为越来越小的子集（称为分支），并为每个子集内的解的值计算一个下界或上界（称为定界）。在每次分支后，对凡是界限超出已知可行解值那些子集不再做进一步分支。这样，解的许多子集（即搜索树上的许多结点）就可以不予考虑了，从而缩小了搜索范围

以该题为例，

1. 我们从第一层，也就是分辨率最低的顶层开始，通过确定各个 cell 的上界，我们发现右上的 cell 的 score 最大为 99，由于它不是叶子节点(深度为 0)，故对其继续进行划分。

2. 在细分辨率图中，对 99 的根节点继续划分，发现该 cell 的子节点中 87 分为最大，由于它是叶子节点(深度为 0)，所以将其赋给 best_score:

$$best\_score = 87$$

3. 我们回到上一层，也就是粗分辨率下，考虑左上根节点，其上界为 85，低于 best_score, 所以可以将其剪枝掉。

4. 考虑左下节点，上界为 98，高于 best_score，所以继续将其分支，发现其子节点最高分为 95，由于它是叶子节点，而且分数大于 best_score, 所以将其赋给 best_score:

best_score = 95

5. 回到上一层粗分辨率图，考虑右下节点，其 上界为 96，高于 best_score，所以继续将其分支，发现其子节点最高分为 89， 由于它是叶子节点，但是分数低于 best_score，故不考虑它。

6. 至此，顶层的所有格子都遍历完了，所以最后得分为:

best_score = 95

而且该分数对应的具体位置也在细分辨率中得到。

参考文献:

[1] The Normal Distributions Transform: A New Approach to Laser Scan Matching, Peter Biber, Wolfgang Straßer.

[2] Scan registration for autonomous mining vehicles using 3D NDT, Magnusson M, Lilienthal A, Duckett T.

[3] NDT 公式推导及源码解析

https://blog.csdn.net/u013794793/article/details/89306901

[4] Real-Time Loop Closure in 2D LIDAR SLAM, Wolfgang Hess, Damon Kohler, Holger Rapp, Daniel Andor.

[5] Google Cartographer SLAM 原理 (Real-Time Loop Closure in 2D LIDAR SLAM 论文详细解读）

https://blog.csdn.net/weixin_36976685/article/details/84994701

[6] 干货 | 10 分钟带你全面掌握 branch and bound（分支定界）算法

https://cloud.tencent.com/developer/article/1472962

https://cloud.tencent.com/developer/article/1540476