

1. 补充代码，实现 gaussiannewtonscanmatcher 模块；

[代码]:

1. 高斯牛顿迭代优化函数：

```
void GaussianNewtonOptimization(map_t* map, Eigen::Vector3d& init_pose, std::vector<Eigen::Vector2d>& laser_pts)
{
    int maxIteration = 30;
    Eigen::Vector3d now_pose = init_pose;
    Eigen::Matrix3d H;
    Eigen::Vector3d b;
    for (int i = 0; i < maxIteration; i++) {
        //TODO
        ComputeHessianAndb(map, now_pose, laser_pts, H, b);
        // std::cout << H(0, 0) << H(1, 1) << H(2, 2) << std::endl;
        if ((H(0, 0) != 0.0) && (H(1, 1) != 0.0)) {
            Eigen::Vector3d searchDir(H.inverse() * b);

            if (searchDir[2] > 0.2) {
                searchDir[2] = 0.2;
                std::cout << "SearchDir angle change too large\n";
            } else if (searchDir[2] < -0.2) {
                searchDir[2] = -0.2;
                std::cout << "SearchDir angle change too large\n";
            }
            // update
            now_pose += searchDir;
        }
        //END OF TODO
    }
    init_pose = now_pose;
}
```

2. 计算 H 和 b 函数：

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

    //TODO
    Eigen::Matrix3d trans_now;
    double sin_angle = sin(now_pose[2]);
    double cos_angle = cos(now_pose[2]);
    trans_now << cos_angle, -sin_angle, now_pose[0],
```

```

    sin_angle, cos_angle, now_pose[1],
    0, 0, 1;
for (size_t i = 0; i < laser_pts.size(); i++) {
    // 激光雷达坐标系下的激光点的坐标
    const Eigen::Vector2d& pt = laser_pts[i];
    Eigen::Vector3d pt_tmp(pt[0], pt[1], 1);
    // now_pose 是当前待估计的激光雷达在世界坐标系下面的坐标
    // now 此刻激光点云在世界坐标系下面的坐标
    Eigen::Vector3d pt_tmp2 = trans_now * pt_tmp;
    Eigen::Vector2d transformed_pt(pt_tmp2[0], pt_tmp2[1]);
    double cell_x_double, cell_y_double;
    cell_x_double = (transformed_pt[0] - macoordsp->origin_x) / macoordsp->resolution +
        double(macoordsp->size_x / 2);
    cell_y_double = (transformed_pt[1] - macoordsp->origin_y) / macoordsp->resolution +
        double(macoordsp->size_y / 2);
    Eigen::Vector2d coord(cell_x_double, cell_y_double);
    Eigen::Vector3d transformedPointData = InterpMapValueWithDerivatives(macoordsp, coord);
    double funVal = 1.0 - transformedPointData[0];
    // cout << "transformedPointData: " << transformedPointData.transpose() << endl;
    // cout << "funVal: " << funVal << endl;
    b[0] += transformedPointData[1] * funVal;
    b[1] += transformedPointData[2] * funVal;
    double rotDeriv = ((-sin_angle * pt.x() - cos_angle * pt.y()) * transformedPointData[1] +
        (cos_angle * pt.x() - sin_angle * pt.y()) * transformedPointData[2]);
    // cout << "rotDeriv: " << rotDeriv << endl;
    b[2] += rotDeriv * funVal;
    // cout << "H(0,0) " << H(0, 0) << " b[2] " << b[2] << endl;
    H(0, 0) += transformedPointData[1] * transformedPointData[1];
    H(1, 1) += transformedPointData[2] * transformedPointData[2];
    H(2, 2) += rotDeriv * rotDeriv;
    H(0, 1) += transformedPointData[1] * transformedPointData[2];
    H(0, 2) += transformedPointData[1] * rotDeriv;
    H(1, 2) += transformedPointData[2] * rotDeriv;
}
H(1, 0) = H(0, 1);
H(2, 0) = H(0, 2);
H(2, 1) = H(1, 2);
//END OF TODO
}

```

3. 双线性插值函数

```

Eigen::Vector3d InterpMapValueWithDerivatives(map_t* map, Eigen::Vector2d& coords)
{
    Eigen::Vector3d ans;
    //TODO
    // cout << "coords: " << coords.transpose() << endl;
    if (!MAP_VALID(map, coords[0], coords[1])) {
        return Eigen::Vector3d(0.0, 0.0, 0.0);
    }
}

```

```

}

Eigen::Vector2i int_coords(coords.cast<int>());
Eigen::Vector2d factors(coords - int_coords.cast<double>());

int sizeX = map->size_x;
Eigen::Vector4d intensities;
intensities[0] = map->cells[MAP_INDEX(map, int_coords[0], int_coords[1])].score;
intensities[1] = map->cells[MAP_INDEX(map, int_coords[0] + 1, int_coords[1])].score;
intensities[2] = map->cells[MAP_INDEX(map, int_coords[0], int_coords[1] + 1)].score;
intensities[3] = map->cells[MAP_INDEX(map, int_coords[0] + 1, int_coords[1] + 1)].score;
// cout << "intensities: " << intensities.transpose() << endl;
double dx1 = intensities[0] - intensities[1];
double dx2 = intensities[2] - intensities[3];
double dy1 = intensities[0] - intensities[2];
double dy2 = intensities[1] - intensities[3];
double xFacInv = (1.0 - factors[0]);
double yFacInv = (1.0 - factors[1]);
ans = Eigen::Vector3d(
    ((intensities[0] * xFacInv + intensities[1] * factors[0]) * (yFacInv)) +
    ((intensities[2] * xFacInv + intensities[3] * factors[0]) * (factors[1])),
    -((dx1 * yFacInv) + (dx2 * factors[1])) / map->resolution,
    -((dy1 * xFacInv) + (dy2 * factors[0])) / map->resolution);
//END OF TODO
return ans;
}

```

总结：代码中有一处错误困扰了很久，开始代码出来的高斯牛顿 path 十分混乱，后来发现是在一处求导的过程中忘记考虑了 resolution 的影响。

就是这一步：

$$\begin{aligned}
 &x \text{的偏导数:} \\
 \frac{\partial L(x, y)}{\partial x} &= \frac{y - y_0}{y_1 - y_0} (M(P_{11}) - M(P_{01})) \\
 &\quad + \frac{y_1 - y}{y_1 - y_0} (M(P_{10}) - M(P_{00})) \\
 &y \text{的偏导数:} \\
 \frac{\partial L(x, y)}{\partial y} &= \frac{x - x_0}{x_1 - x_0} (M(P_{11}) - M(P_{10})) \\
 &\quad + \frac{x_1 - x}{x_1 - x_0} (M(P_{01}) - M(P_{00}))
 \end{aligned}$$

这里应该是对世界坐标系下面的 xy 进行求导，而代码中开始是对地图坐标系的 coords 进行求导，最终导致出错，中间相差 1/resolution。对应代码中的部分为如下所示：

```

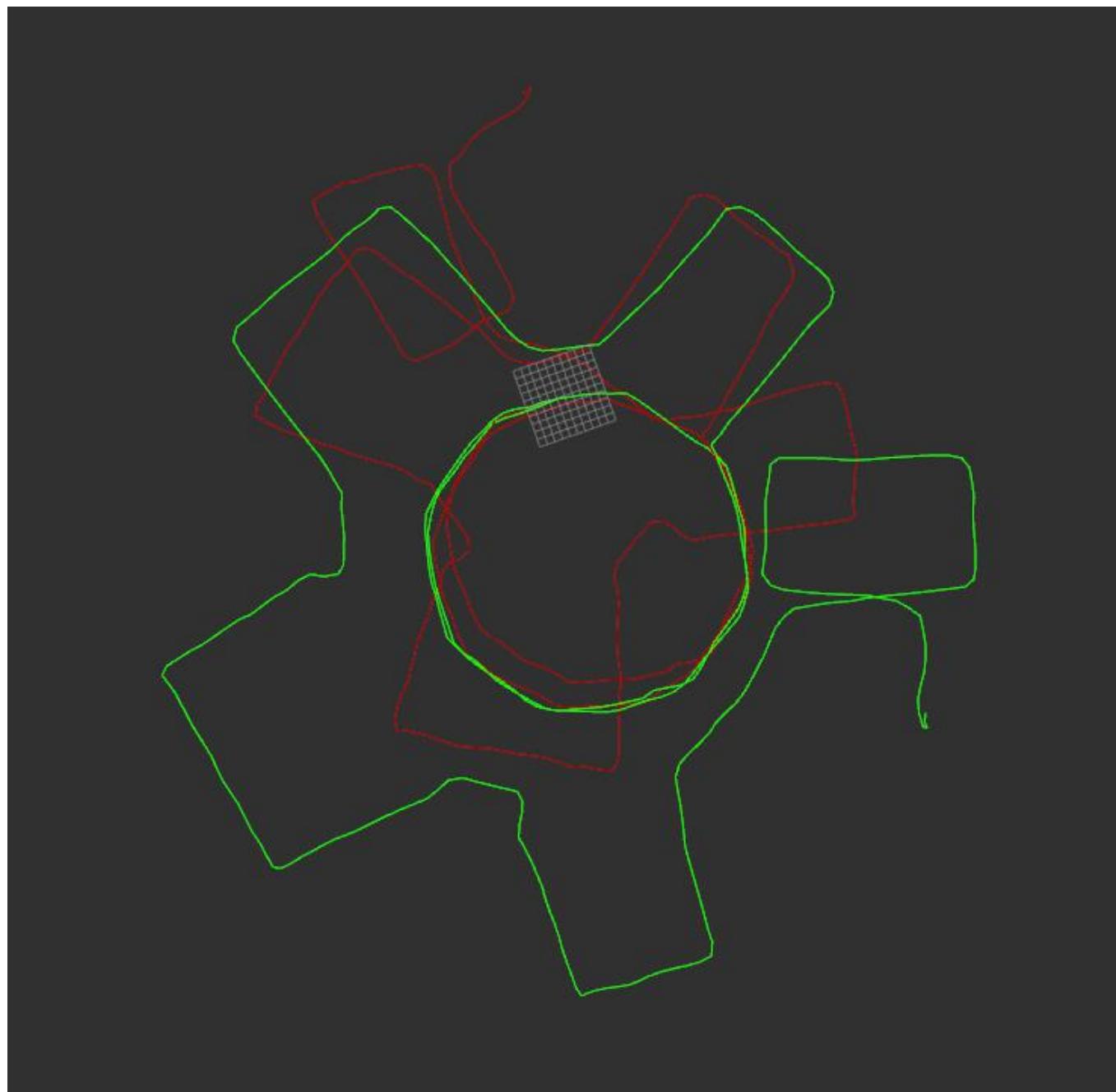
-((dx1 * yFacInv) + (dx2 * factors[1])) / map->resolution,
-((dy1 * xFacInv) + (dy2 * factors[0])) / map->resolution;

```

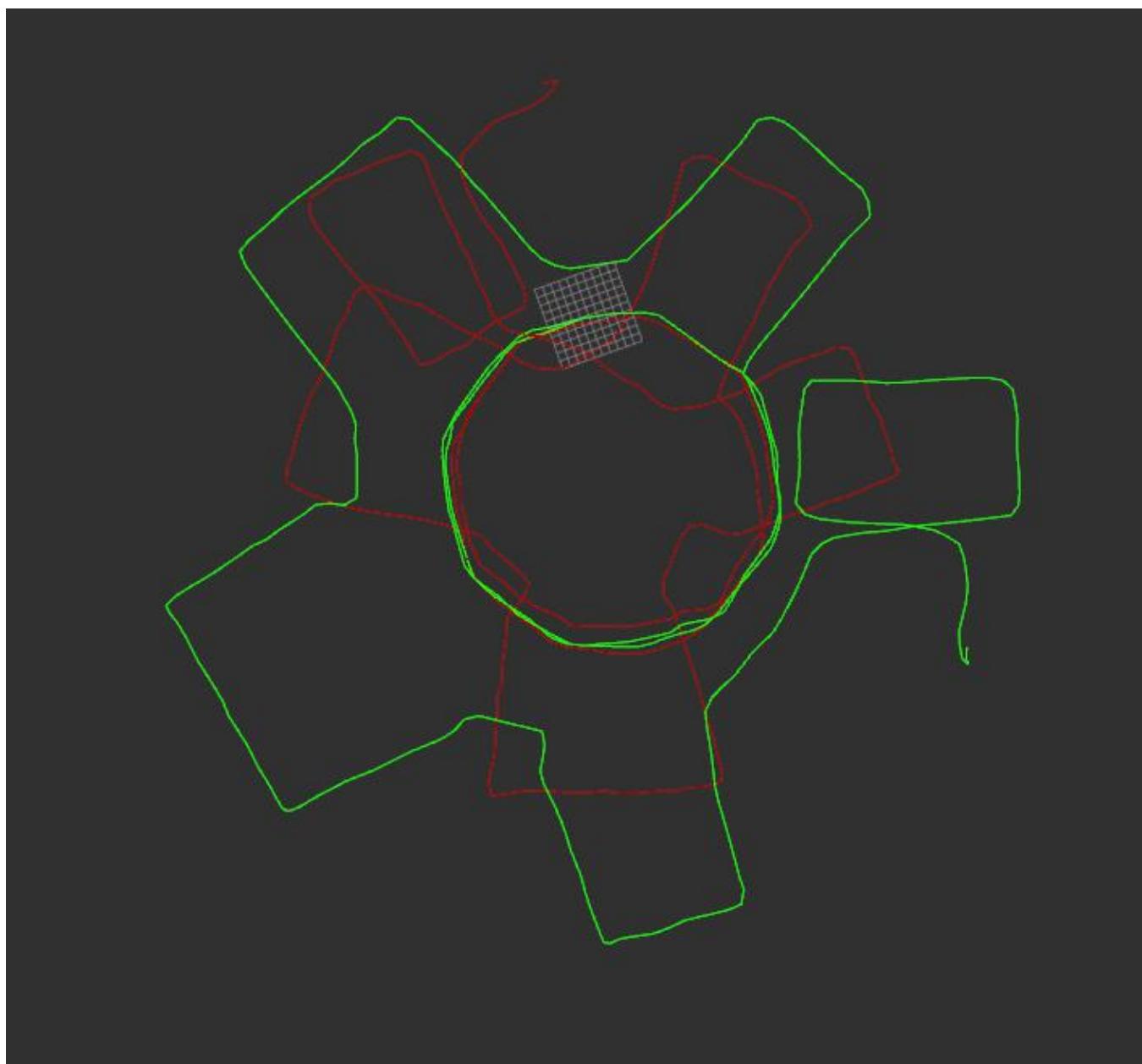
修改之后结果正确，输出如下：（绿色轨迹为里程计坐标系下的激光雷达位置，红色轨迹为激光雷达匹配得到的世界坐标系下的位置）

\

迭代 20 次



迭代 50 次，效果稍微好一点



[补充]:

写这道题目时候开始有一个困惑，就是为什么里程计坐标系下的激光雷达位置可以表示激光雷达的位置，后来发现这里我有一个之前没发现的误区，我感觉群里的讨论有一块是因为一些人也和我一样有这个误区。

就是对于里程计坐标系的理解：

里程计坐标系我开始以为是车上的轮速里程计的那个局部坐标系，所以我开始认为里程计应该是和激光雷达都在车上，他们大致是相对静止的，不应该能够表示激光雷达在世界坐标系下面的位姿。

后来发现里程计坐标系和里程计是两个不一样的概念，在这里总结如下：

坐标系有一些命名规范，里程计坐标系 `odom` 就是按照规范的定义来解释的

1、`base_link` 坐标系

`base_link` 是机器人本体坐标系

2、`odom` 坐标系

`odom` 的坐标系是固定世界坐标系，但是 `odom` 坐标系会随时间推移而漂移，与 `map` 坐标系不同。

这种漂移使 `odom` 坐标系无法用作长期的全局参考。但是，可以确保机器人的位姿在 `odom` 坐标系中是连续的，这意味着 `odom` 坐标系中的机器人的位姿始终以连续平稳的方式变化，而不会出现离散的跳跃，这一点与 `map` 坐标系不同。

一般的，里程计坐标系的数据来源于车轮里程计，视觉里程计或惯性测量单元，以此来计算机器人在里程计坐标系下面的位姿。

里程计坐标系下的机器人位姿可以作为一个短期的参考，但漂移的存在使其不能作为全局的参考。

3、`map` 坐标系

`map` 坐标系是固定的世界坐标系

在 `map` 坐标系下，不应存在漂移，但是机器人在 `map` 坐标系下面的位姿可能会有离散的变化，因为机器人在 `map` 坐标系下面的位姿可能因为回环检测而较大幅度的修改。

一般的，定位模块会根据传感器的观测值不断地重新计算 `map` 坐标系中的机器人姿态，从而消除漂移，但是此时可能会引起离散的跳跃。

map 坐标系是一个长期的全局定位参考，但在位置估计存在离散跳跃，使其不能作为局部的短期位置参考。

4、base_laser 坐标系

激光坐标系，以 base_link 作为母坐标系，是激光相对于底座中心的坐标系，其与 base_link 之间的转换关系是固定的

[补充说明]

1、odom 坐标系和 odom topic 之间的区别

odom 里程计坐标系，这里要区分开和 odom topic 的区别，这是两个概念。

前者是一个坐标系，后者是一个根据编码器（或者视觉等）计算的里程计。

但是两者也有关系，odom topic 转化得位姿矩阵是 odom \rightarrow base_link 的 tf 关系。

2、odom 坐标系和 map 坐标系之间的区别

在机器人刚开始运动的时候，odom 和 map 坐标系是重合的。

但是，随着时间的推移是不重合的，而出现的偏差就是里程计的累积误差

如果 odom 计算没有错误，那么 odom 坐标系和 map 坐标系就是重合的，map \rightarrow odom 的 tf 就是 0

因此，代码里面的 odom 坐标系是一个带有飘移的全局坐标系，不是车上的里程计的局部坐标系。通过做这道题目弥补了自己之前的知识空缺。

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；

解答：第一题里面参考帧构建的地图与当前帧都转换到了世界坐标系下面，当前帧和参考帧构建的地图之间相差了一个帧间转换关系，这个帧间转换关系没有在代码中添加进去进行优化，优化的是当前帧在世界坐标系下面的位置。

因此，可以考虑将当前帧点云转化到世界坐标系下面之后，再利用估计的帧间转移矩阵，将当前帧的点转换到参考帧地图坐标系下，这样二者会更加吻合，得到的势场值会更大，即将帧间转移矩阵也添加进去进行优化会提高匹配效果。

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？
- (2) 简述 NDT 根据 score 函数进行优化求解的过程。

解答：

(1)

优化函数的 score 是一个正态分布表达式：

$$\text{score}(\mathbf{p}) = \sum_i \exp\left(-\frac{(\mathbf{x}'_i - \mathbf{q}_i)^T \Sigma_i^{-1} (\mathbf{x}'_i - \mathbf{q}_i)}{2}\right).$$

里面的 \mathbf{x}'_i 是当前帧转换到参考帧坐标系下面的点， \mathbf{q}_i 和 Σ 是对应的 cell 的均值和协防差。

(2) 优化求解过程：

1 初始化

在空间分配 cell

对于参考帧里面落在对应 cell 里面的激光点，计算属于该 cell 的均值和协防差

2 配准

While(未达到收敛条件) do

Score = 0

\mathbf{g} = 0

\mathbf{H} = 0

for 对于所有属于当前帧的激光点 do

将该激光点转换到参考帧坐标系

寻找属于的 cell

根据该 cell 的均值和协方差计算 score 并加和

更新 \mathbf{g}

$$g_i = \frac{\delta s}{\delta p_i} = \sum_{k=1}^n m d_1 d_2 \vec{x}'_k^T \Sigma_k^{-1} \frac{\delta \vec{x}'_k}{\delta p_i} \exp\left(-\frac{d_2}{2} \vec{x}'_k^T \Sigma_k^{-1} \vec{x}'_k\right)$$

更新 \mathbf{H}

$$H_{ij} = \frac{\delta^2 s}{\delta p_i \delta p_j} = \sum_{k=1}^n d_1 d_2 \exp\left(-\frac{d_2}{2} \vec{x}'_k^T \Sigma_k^{-1} \vec{x}'_k\right) \left(-d_2 (\vec{x}'_k^T \Sigma_k^{-1} \frac{\delta \vec{x}'_k}{\delta p_j}) + \vec{x}'_k^T \Sigma_k^{-1} \frac{\delta^2 \vec{x}'_k}{\delta p_i \delta p_j} + \frac{\delta \vec{x}'_k^T}{\delta p_j} \Sigma_k^{-1} \frac{\delta \vec{x}'_k}{\delta p_i} \right)$$

end for

求解 \mathbf{H} $\Delta \mathbf{p} = -\mathbf{g}$

$\mathbf{p} = \mathbf{p} + \Delta \mathbf{p}$

End while

其中，对于 \mathbf{g} 和 \mathbf{H} 的求解，如下：

首先，确定目标函数是-score，求最小值

$$s = -\exp \frac{-\mathbf{q}^t \Sigma^{-1} \mathbf{q}}{2}.$$

对目标函数 s 求一阶导数：

$$\begin{aligned}\tilde{g}_i &= -\frac{\partial s}{\partial p_i} = -\frac{\partial s}{\partial q} \frac{\partial q}{\partial p_i} \\ &= \mathbf{q}^t \Sigma^{-1} \frac{\partial q}{\partial p_i} \exp \frac{-\mathbf{q}^t \Sigma^{-1} \mathbf{q}}{2}.\end{aligned}$$

这里面 \mathbf{q} 是当前帧转换到参考帧之后去均值的结果，公式如下。

$$\begin{aligned}\mathbf{q} &= \mathbf{x}_i - \mathbf{q}_i = T(\mathbf{x}_i) - \mathbf{q}_i \\ \frac{\partial \mathbf{q}}{\partial T} &= \frac{\partial T(x_i)}{\partial T}\end{aligned}$$

那么对 \mathbf{q} 对 p_i 求偏导数转化为如下所示：（这里 p_i 和 T 是大致等价的，表述的都是参考帧和当前帧之间的转换关系）

$$T(\mathbf{x}_i) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} \mathbf{x}_i \\ t_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \longrightarrow \frac{\partial T(\mathbf{x}_i)}{\partial T} = \begin{pmatrix} 1 & 0 & -x_i \sin \theta - y_i \cos \theta \\ 0 & 1 & x_i \cos \theta - y_i \sin \theta \end{pmatrix}$$

继续对一阶导数求导，得到 \mathbf{H}

$$\begin{aligned}\tilde{H}_{ij} &= -\frac{\partial s}{\partial p_i \partial p_j} = -\exp \frac{-\mathbf{q}^t \Sigma^{-1} \mathbf{q}}{2} \\ &\quad ((-\mathbf{q}^t \Sigma^{-1} \frac{\partial \mathbf{q}}{\partial p_i})(-\mathbf{q}^t \Sigma^{-1} \frac{\partial \mathbf{q}}{\partial p_j}) + \\ &\quad (-\mathbf{q}^t \Sigma^{-1} \frac{\partial^2 \mathbf{q}}{\partial p_i \partial p_j}) + (-\frac{\partial \mathbf{q}^t}{\partial p_j} \Sigma^{-1} \frac{\partial \mathbf{q}}{\partial p_i}))\end{aligned}$$

\mathbf{H} 是一个 6×3 的矩阵，除了 $i=j=3$ ，其余均为 0。

$$\frac{\partial^2 \mathbf{q}}{\partial p_i \partial p_j} = \begin{cases} \begin{pmatrix} -x \cos \phi + y \sin \phi \\ -x \sin \phi - y \cos \phi \end{pmatrix} & i = j = 3 \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{otherwise} \end{cases} \quad (13)$$

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分辨率下机器人的最佳匹配位置，请简述匹配和剪枝流程。 (2 分)

<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">85</td> <td style="width: 50%;">99</td> </tr> <tr> <td>98</td> <td>96</td> </tr> </table>	85	99	98	96	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%;">41</td> <td style="width: 25%;">43</td> <td style="width: 25%;">58</td> <td style="width: 25%;">24</td> </tr> <tr> <td>76</td> <td>83</td> <td>87</td> <td>73</td> </tr> <tr> <td>86</td> <td>95</td> <td>89</td> <td>68</td> </tr> <tr> <td>70</td> <td>65</td> <td>37</td> <td>15</td> </tr> </table>	41	43	58	24	76	83	87	73	86	95	89	68	70	65	37	15
85	99																				
98	96																				
41	43	58	24																		
76	83	87	73																		
86	95	89	68																		
70	65	37	15																		
左图：机器人在粗分辨率地图下各个位置的匹配得分	右图：机器人在细分分辨率地图下各个位置的匹配得分（细分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）																				

流程：

```
Best_score = -无穷
set() = set(85, 99, 98, 96)
```

当 set 不为空，遍历 set 里面的所有节点

1、选择当前 set 最大的 99 节点，发现大于 best_score，将 99 节点进行 split，进入对应细分分辨率的格子，移除 99

2、进入下一分辨率后，发现该分辨率已经是叶子节点，选取最大的节点 87，大于 best_score，则更新 best_score = 87

该分辨率下当前格子其他叶子节点 (58, 24, 73) 都比此刻选取的最大叶子节点 (87) 小，全部 bound 掉

3、返回上一分辨率，选取上一分辨率此刻最大的节点 98，大于 best_score，进入对应的下一分辨率，移除 98

4、下一分辨率是叶子节点，选取最大叶子节点， $95 > best_score$, 更新 $best_score = 95$, 其余叶子节点 bound 掉 (86, 70, 65)

5、返回上一分辨率，选取上一分辨率此刻最大的节点 96，大于 best_score, 进入对应的下一分辨率，移除 96

6、下一分辨率是叶子节点，选取最大叶子节点， $89 < best_score$, 不更新 best_score, 叶子节点全部 bound 掉 (89, 68, 37, 35)

7、返回上一分辨率，选取上一分辨率当前最大节点 85，发现 $85 < best_score$ ，直接 bound 掉 85

此刻 set 等于空，结束，最终 $best_score = 95$, 最佳匹配位置为 95 对应的位置。

