文章复述

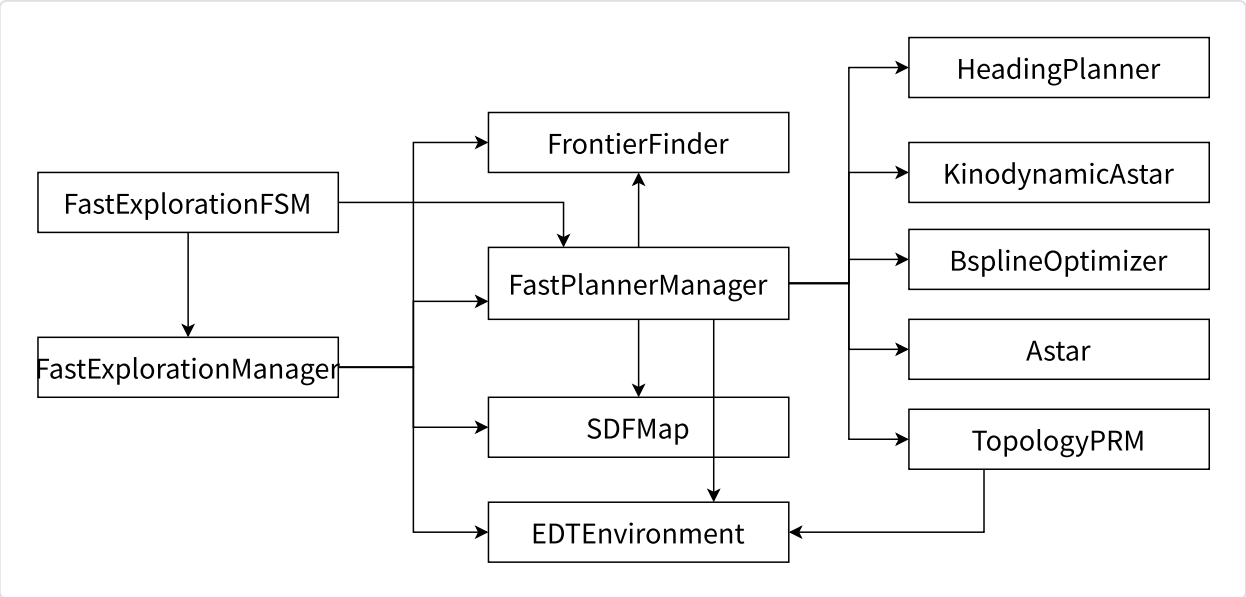
原代码位于https://github.com/HKUST-Aerial-Robotics/FUEL

复现代码位于https://gitee.com/tiemuhua/fuel_refactored

文章中提到的绝大多数内容位于frontier_finder.cpp与graph_node.cpp当中。

下文中提到的函数名以复现代码为准。

类持有关系图



地图的边界

在定位建图模块建立的地图中，点有三种属性：无障碍物、有障碍物、未探索。

其中，未探索的点组成的集合是未探明区域，无障碍物和有障碍物的点组成的集合是已探明区域。

我们定义地图的边界为：无障碍物并且周围有未探索的点组成的集合。

由问题的物理背景可知，已探明区域和有障碍物区域是闭集，未探明区域和无障碍物区域是开集，所以地图的边界是良定义的。

为了绘制未知区域的地图，机器人需要在地图边界上沿边界法向向前探索。

地图边界是一个不规则曲线，甚至有可能是很多条不连通的曲线。

为了更好的探索，我们需要将地图边界进行分割为若干个元边界，元边界内部的点距离都小于某一事先给定的阈值。

每个元边界应当维护如下信息

- 元边界中所有边界点的坐标

- 元边界的中心，即所有边界点坐标的平均数
- 元边界的界限，即一个覆盖了元边界内所有边界点的长方体
- 采样得到的观测点，将在“优化全局路径”一节详细介绍

移除过时边界

位于FrontierFinder::removeOutDatedFrontiers当中。

随着地图的不断更新，地图的边界会不断发生变化，需要定时对其加以维护。

首先我们要遍历之前求出来的每一条元边界，判断其是否符合边界的定义，并且移除不符合边界定义的元边界。

由于每条元边界中的点较多，遍历一遍非常耗时，因此我们希望尽可能地避免遍历元边界中的点。

定位建图模块会给出新探索到的区域的信息，将新探索到的区域视为一个长方体。

如果某个之前找到的元边界不在新探索的区域当中，那么由于该元边界对应的地图并没有发生改变，因此这个元边界一定不需要移除。

我们要先判断新探索区域对应的长方体与元边界界限对应的长方体有无重叠。

如果有重叠，再去遍历元边界中的每个点，判断是否符合边界定义。

如果有大于等于1个点不符合边界定义，那么该元边界就是需要移除的。

如果采用链表来储存元边界数据，需用使用迭代器作为全局变量来储存新加入的元边界的地址，代码可读性相对较差。

因此我们使用数组的数据结构来储存元边界数据。

假设元边界的数量为m，如果我们直接将不符合边界定义的元边界从元边界数组中删除，

那么将所有元边界遍历一遍的时间复杂度将是O(m^2)，这会消耗大量时间。

我们使用两次遍历来加速删除过程。

在第一次遍历的过程中，记录每个元边界是否符合边界的定义，

然后在第二次遍历的过程中，将符合边界定义的元边界加入到一个新的元边界数组当中，最后删除原先的元边界数组。

每次删除的均摊复杂度仍然是O(1)，可以用O(m)的时间复杂度完成遍历。

为了进一步减少时间消耗。我们还可以用移动构造而不是拷贝构造的方式来完成元边界的复制。

这样就避免了在复制元边界时大量的时间开销，进一步提高了问题求解效率。

添加新边界

位于FrontierFinder::searchAndAddFrontiers和FrontierFinder::expandFrontier当中。
expandFrontier函数为DFS过程。

在移除了过时边界之后，我们需要加入新产生的边界。

遍历新探索区域对应的长方体内的每一个点，以当前遍历到的点为根节点做一轮DFS搜索，每轮DFS搜索都会产生一个对应的新边界。

如果当前点的近邻的点满足以下四个条件：

- 高度小于某一事先给定的高度阈值
- 已经探索过且无障碍物
- 在新探索区域对应的长方体中
- 还没有在之前的某轮DFS中被遍历过

那么就将这个点加入到DFS搜索栈当中，并且记录这个点已经被DFS遍历过。

如果某个点已经在之前的DFS搜索中被遍历过，说明这个点要么不是边界点，要么已经在之前的DFS搜索中加入了某个新边界，本轮DFS中就不考虑这个点。

如果搜索到的点满足边界定义，那么就将这个点加入到当前新边界当中。

本轮DFS搜索结束后，如果对应的新边界中的点不是空集，就把对应的新边界加入到当前地图对应的新边界集合当中。

在所有轮的DFS搜索中，长方体中的每个点都会被且只被遍历一次，因此整个遍历的时间复杂度是线性的。

将新边界的分割为元边界

位于FrontierFinder::splitLargeFrontiers和FrontierFinder::splitHorizontally当中。

搜索出的新边界并不是最终的元边界。DFS搜索的过程只能保证每一个搜索出来的新边界都是道路连通的点集。

但是新边界有可能过长，因此我们需要对搜索出的新边界递归地进行二分，将其分割为元边界。

分割过程递归地使用了主成分分析（PCA）算法。

假设我们得到的某个新边界中有n个点，其三维坐标分别为 (x_i, y_i, z_i)

由于机器人基本上是在一个平面内进行活动，z轴方向不会有很大位移，因此我们只考虑

x、y 两个方向的主成分分析。

记每个点的二维坐标 $p_i = (x_i, y_i)$ ，所有二维坐标组成的点集为 $S = p_i | 0 < i \leq n$ ，点集重心的二维坐标为 $\bar{p} = (\bar{x}, \bar{y})$ 。

考虑协方差矩阵： $A = \Sigma_{i=1}^{n} (p_i - \bar{p})(p_i - \bar{p})^T$ 。

对A做特征值分解，由于A是正定对称阵，故必然可以特征值分解，且两个特征值都是非负实数。

记A最大特征值对应的特征向量为 $alpha$ ，则 $alpha$ 所在的直线是点集散布方差最大的方向。

因此，我们根据在 $alpha$ 方向上点相对于重心的坐标，将新边界中的n个点分成两个集合 S_1 和 S_2 ，其中

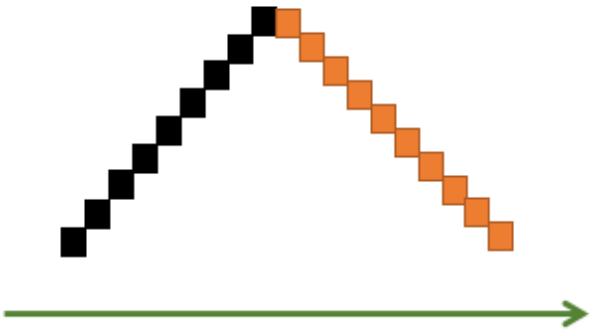
$$S_1 = \{p_i | (p_i - \bar{p})^T \alpha < 0, 0 < i \leq n\}$$

$$S_2 = \{p_i | (p_i - \bar{p})^T \alpha \geq 0, 0 < i \leq n\}$$

然后再对 S_1 和 S_2 进行主成分分解。

递归地进行上述过程，直到二维点集中的每个点到重心的距离均小于阈值。

这样我们就得到了一组元边界，如图所示。



黑色方格和黄色方格分别为分割后的两组元边界，绿色箭头为向量 $alpha$

旅行商问题

我们现在有m个元边界，我们需要将这m个元边界的中心 $c_i, 0 < i \leq m$ 连接起来作为全局路径。

首先我们通过A-STAR算法找到任意两个元边界中心之间的一条粗略路径，用这条粗略路径的长度作为元边界中心之间的距离。

每次移除过时边界和加入新边界时，同时在FrontierFinder::updateFrontierCostMatrix函数中维护边界中心两两之间的距离。

这样，轨迹规划问题就转化为了一个旅行商(TSP)问题，即我们要找到一条遍历所有元边界中心的长度最短的路径。

我们使用了LKH算法求解旅行商问题，获得了元边界之间的路由顺序以及粗略的全局路径。

优化全局路径

旅行商问题生成的最优轨迹是将所有元边界的中心 $c_i, 0 < i \leq m$ 连接起来获得的，这就产生了三个问题。

- 元边界的中心不一定是最优的观测点，可能有其他位置可以更好的观测元边界附近的位置区域
- 元边界中心连接成的轨迹不一定符合机器人动力学约束，其他位置连接成的轨迹可能会更短
- 我们现在只知道目标点的位置（即元边界的中心），但是不知道目标姿态，即不知道机器人运动到目标点之后应该朝哪个方向去观测

为了解决这三个问题，在保证观测效果的同时使得全局路径的长度尽可能短，

我们需要为每条元边界找到一个更合适的观测点及其对应的观测方向，用这些观测点去替代元边界中心，将这些观测点按照上一小节求出的元边界路由顺序连接起来，作为全局路径。

首先在每个元边界中心点附近的一个圆形区域内均匀采样。位于FrontierFinder::sampleViewpoints函数当中。

如果采样点不在障碍物或者未知区域内，则计算采样点可以观测到多少个边界中的点。

假设可以观测到 t 个点，这 t 个点的坐标分别为 $p_i, 0 < i \leq t$ ，当前采样点的坐标为 q_j 。

则当前采样点 q_j 对应的观测方向，为 q_j 朝向 p_i 的平均值。

为了避免在求解反三角函数时出现数值问题，我们先以采样点朝向元边界中心的方向 $r = q_j - c_i$ 作为参考方向，求得相对于参考方向的观测角度为 $\kappa_j^{ref} = \frac{\Sigma_{i=1}^{t} acos((p_i - q_j) \cdot r)}{t}$ 。

然后再加上参考方向相对于世界坐标系的旋转角度，得到绝对观测角度为 $\kappa_j = \kappa_j^{ref} + atan(\frac{r_y}{r_x})$ 。

对于每条元边界，我们只保留观测到边界点个数最多的 N_{vp} 个采样点，默认条件下 N_{vp} =15。位于FrontierFinder::getNViewPoints函数当中。

设m条元边界的路由顺序为 $F_1, F_2, \cdots, F_m$ ，

第i个元边界对应的采样点为 $q_{i,j}, 0 < i \leq m, 0 < j \leq N_{vp}$ 。

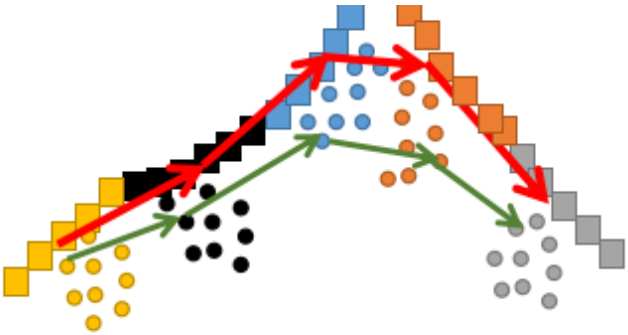
对于 $i = 1, 2, \cdots m-2$ ， $q_{i,j_i}, 0 < j_i \leq N_{vp}$ 与 $q_{i+1,j_{i+1}}, 0 < j_{i+1} \leq N_{vp}$ 构成了 $(m-2) \times (N_{vp}^2)$ 条有向边，

当前机器人坐标点与 $q_{1,j}, 0 < j \leq N_{vp}$ 构成了 N_{vp} 条有向边，

$q_{m-1,j}, 0 < j <= N_{vp}$ 与第m个元边界中心点构成了 N_{vp} 条有向边，

这总共 $(m-2) \times (N_{vp}^2) + 2 \times N_{vp}$ 条有向边构成了一个有向无环图。

于是问题转化成了一个从当前点到最后一个元边界中心点的图搜索最近路径问题，可以通过dijkstra算法进行求解，得到优化后的全局路径，如图所示。代码位于FastExplorationManager::refineLocalTour当中。



方格为边界，圆点为观测点，红色路径为优化前路径，绿色路径为优化后的路径

避免折返

这一部分原论文没有写，CMU的论文https://link.zhihu.com/?target=https%3A//github.com/caochao39/tare_planner更好的解决了这一问题。

在规划路径的时候，经常有在已探明区域内部有未探明区域的情况出现，如红圈部分所示。

无人机经常会优先探索最外侧的区域，扩大已探明区域，然后再返回来去探索已探明区域内部的未探明区域。

为了提高效率，应该尽可能少的去折返，优先探索已探明区域内部的未探明区域，消除空洞之后再去扩张最外侧的边界。

因此我们将元边界分成两类。

第一类，元边界与障碍物构成了一个封闭的二维图形，如红圈所示。

如果能在一定距离内沿着边界或障碍物的边缘能够用A-STAR算法搜索出一条封闭路径，那么就认为这个元边界是第一类边界。

第一类边界在已探明区域的内部，沿边界法向前进很短的距离边界就会撞到障碍物或者闭合消失，基本不会对轨迹有大幅改变。

第二类，元边界位于已探明区域的外部边缘，如绿圈所示。

如果不能在一定距离内用A-STAR搜索出一条位于元边界或者障碍物边界上面的路径，那么认为这个元边界是第二类边界。

第二类边界位于已探明区域的外侧，是开放的，可能会向前探索很远，会对轨迹造成较大影响。

因此，我们将探索过程分成两个旅行商问题，优先探索第一类元边界，然后再探索第二类元边界。