Jindong_Shi 2022.04.05

激光 SLAM 第 3 次作业

1. 补充去除激光雷达运动畸变模块的代码

编写函数 void Lidar_MotionCalibration(...)代码的代码如下所示:

LidarMotionUndistortion.cpp

```
194         *        激光雷达运动畸变去除分段函数;
195         *        在此分段函数中，认为机器人是匀速运动；
196         * @param frame_base_pose       标定完毕之后的基准坐标系
197         * @param frame_start_pose      本分段第一个激光点对应的位姿
198         * @param frame_end_pose        本分段最后一个激光点对应的位姿
199         * @param ranges                激光数据——距离
200         * @param angles                激光数据——角度
201         * @param startIndex            本分段第一个激光点在激光帧中的下标
202         * @param beam_number           本分段的激光点数量
203         */
204        void Lidar_MotionCalibration(
205                tf::Stamped<tf::Pose> frame_base_pose,
206                tf::Stamped<tf::Pose> frame_start_pose,
207                tf::Stamped<tf::Pose> frame_end_pose,
208                std::vector<double>& ranges,
209                std::vector<double>& angles,
210                int startIndex,
211                int& beam_number)
212        {
213            //TODO
214            // 方法一: 使用四元数及四元数插值，使用平移插值
215            tf::Quaternion q_start = frame_start_pose.getRotation();
216            tf::Quaternion q_end = frame_end_pose.getRotation();
217            tf::Vector3 xy_start = tf::Vector3(frame_start_pose.getOrigin().getX(), frame_start_pose.getOrigin().getY(), 1);
218            tf::Vector3 xy_end = tf::Vector3(frame_end_pose.getOrigin().getX(), frame_end_pose.getOrigin().getY(), 1);
```

LidarMotionUndistortion.cpp

```
220            // 对frame_start_pose和frame_end_pose之间的所有点进行线性插值
221            for(int i = 0; i <= beam_number-1; ++i)
222            {
223                double ratio = i/(beam_number-1);
224                // 对平移进行插值: 对 xy_start 和 xy_end 根据 ratio 进行插值
225                tf::Vector3 xy_lerp = xy_start.lerp(xy_end, ratio);
226                // 对旋转进行插值: 对 q_start 和 q_end 根据 ratio 进行插值
227                tf::Quaternion q_slerp = q_start.slerp(q_end, ratio);
228
229                // T_O_lerp 为 当前插值激光点对应的机器人坐标系到里程计坐标系的变换矩阵
230                tf::Transform T_O_lerp;
231                T_O_lerp.setOrigin(xy_lerp);
232                T_O_lerp.setRotation(q_slerp);
233
234                // 从容器 ranges 和 angles 中获得存在运动畸变的位姿
235                double x_distorted = ranges[startIndex + i] * cos(angles[startIndex + i]);
236                double y_distorted = ranges[startIndex + i] * sin(angles[startIndex + i]);
237                double yaw_distorted = angles[startIndex + i];
238
239                // 去畸变:  frame_base_pose.inverse() 相当于 T_O_base.inverse()
240                tf::Vector3 xy_undistorted = frame_base_pose.inverse() * T_O_lerp * tf::Vector3(x_distorted, y_distorted, 1);
241                // 将去除畸变后的位姿转化为激光数据range和angle
242                ranges[startIndex + i] = std::sqrt(xy_undistorted[0] * xy_undistorted[0] + xy_undistorted[1] * xy_undistorted[1]);
243                angles[startIndex + i] = std::atan2(xy_undistorted[1], xy_undistorted[0]);
244            }
```

对上面代码的解析:

我们在 main 函数内创建了一个 tf 变换监听器，通过 tf 树来查询每一个时刻机器人在里程计坐标系下的位姿:

```
tf::TransformListener tf(ros::Duration(10.0));
```

我们又创建一个运动畸变去除器:

```
LidarMotionCalibrator tmpLidarMotionCalib(&tf);
```

在 LidarMotionCalibrator 类内，我们创建了一个 subscriber 用来订阅名为 champion_scan 的 topic，一旦有数据，就立即调用回调函数 ScanCallBack():

```
scan_sub_ = nh_.subscribe("champion_scan", 10,
                          &LidarMotionCalibrator::ScanCallBack, this);
```

在回调函数 ScanCallBack 内，我们记录下每一帧点云的起始和结束时间，并将消息内的点云信息(包括距离和角度)存放到容器 ranges 和 angles 内。并且通过函数 bool getLaserPose()可以计算获得任意 dt 时刻下，激光雷达在里程计坐标系下的位姿。

在回调函数 ScanCallBack 内我们调用函数 void Lidar_Calibration(...)。我们对每一帧点云数据每 5ms 长度进行分段。具体而言，一开始 base_time 为该帧点云起始时间，start_time 也为该帧点云起始时间，mid_time 为刚好长度大于 5ms 的点云段的时间, 我们记录该段点云内的点数为 interp_count。我们通过 getLaserPose(...)函数获得对应的 frame_base_pose, frame_start_pose, frame_mid_pose, 并将其和 interp_count 一起作为参数传给函数 Lidar_MotionCalibration(…), 而完成该函数就是本次的作业。

函数 Lidar_MotionCalibration(…)的目的就是将该段点云内的每个点的位姿转换到该帧点云的初始位姿 frame_base_pose 坐标系下。具体的，我们使用 for 循环，来处理该段点云内的每个点。在每个循环内使用平移插值和四元数插值，获得扫描每个激光点时机器人在里程计坐标系下的位姿:

```
double ratio = i/(beam_number-1);
// 对平移进行插值: 对 xy_start 和 xy_end 根据 ratio 进行插值
tf::Vector3 xy_lerp = xy_start.lerp(xy_end, ratio);
// 对旋转进行插值: 对 q_start 和 q_end 根据 ratio 进行插值
tf::Quaternion q_slerp = q_start.slerp(q_end, ratio);
```

从而得到当前插值激光点对应的机器人坐标系到里程计坐标系的变换矩阵 T_O_lerp:

```
tf::Transform T_O_lerp;
T_O_lerp.setOrigin(xy_lerp);
T_O_lerp.setRotation(q_slerp);
```

可使用如下公式将该激光点位姿转换到该帧点云的初始位姿 frame_base_pose 坐标系下，来去除点云畸变:

```
tf::Vector3 xy_undistorted = frame_base_pose.inverse() * T_O_lerp *
                             tf::Vector3(x_distorted, y_distorted, 1);
```

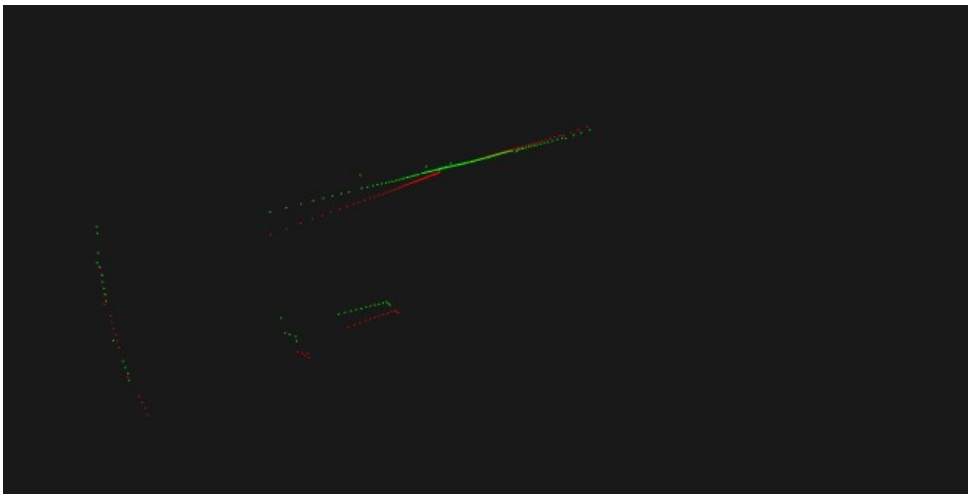
然后将去除畸变后的点云位姿转化为激光数据的 range 和 angle 发送出去。之后，我们令

```
frame_start_pose = frame_mid_pose;
```

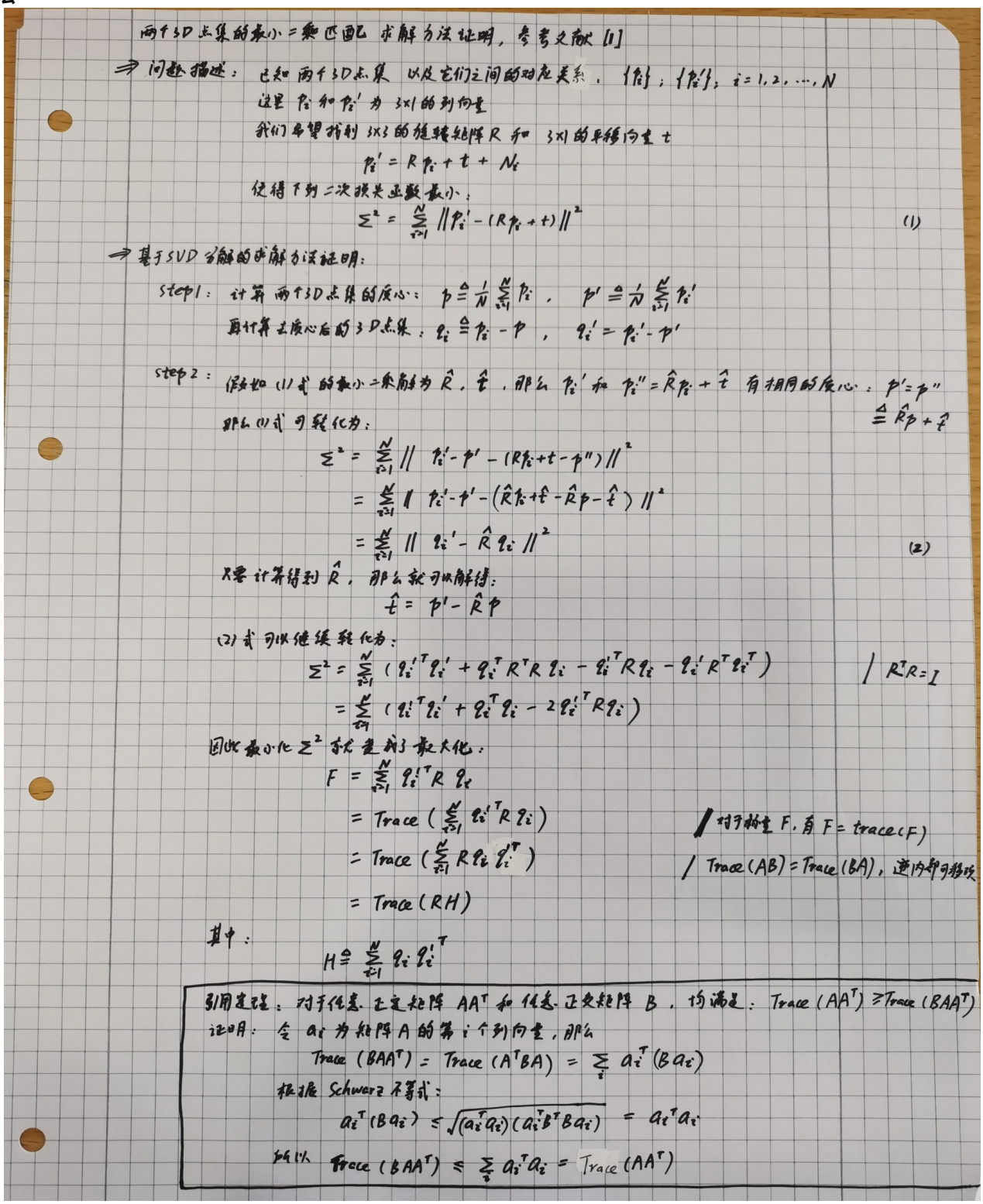
来处理下一段 5ms 长度的点云，直到 end_time。

Jindong_Shi 2022.04.05

点云去畸变的效果如下所示，其中红色为原始存在运动畸变的点云，绿色为去畸变后的点云。很明显绿色曲线看起来更正常, 更平滑一些:



2. 阅读论文 Least-Squares Fitting of Two 3-D Points Sets [1], 推导并证明已知对应点的 ICP 求解方法



两个3D点集的最小二乘匹配 求解方法证明，参考文献 [1]

⇒ 问题描述：已知 两个3D点集 以及它们之间的对应关系，$\{p_i\}, \{p_i'\}, i=1,2,\cdots,N$

这里 p_i 和 p_i' 为 3×1 的列向量

我们希望找到 3×3 的旋转矩阵 R 和 3×1 的平移向量 t

$$p_i' = R p_i + t + N_i$$

使得下列二次损失函数最小：

$$\Sigma^2 = \sum_{i=1}^{N} \| p_i' - (R p_i + t) \|^2 \quad (1)$$

⇒ 基于SVD分解的求解方法证明：

step1：计算 两个3D点集的质心：$p \triangleq \frac{1}{N}\sum_{i=1}^{N} p_i$，$p' \triangleq \frac{1}{N}\sum_{i=1}^{N} p_i'$

再计算去质心后的3D点集：$q_i \triangleq p_i - p$，$q_i' = p_i' - p'$

step2：假如 (1)式 的最小二乘解为 $\hat{R}$，$\hat{t}$，那么 p_i' 和 $p_i'' = \hat{R} p_i + \hat{t}$ 有相同的质心：$p' = p'' \triangleq \hat{R}p + \hat{t}$

那么 (1)式 可转化为：

$$\begin{aligned}\Sigma^2 &= \sum_{i=1}^{N} \| p_i' - p' - (R p_i + t - p'') \|^2 \\ &= \sum_{i=1}^{N} \| p_i' - p' - (\hat{R} p_i + \hat{t} - \hat{R}p - \hat{t}) \|^2 \\ &= \sum_{i=1}^{N} \| q_i' - \hat{R} q_i \|^2 \end{aligned} \quad (2)$$

只要计算得到 $\hat{R}$，那么就可以解得：

$$\hat{t} = p' - \hat{R}p$$

(2)式 可以继续转化为：

$$\begin{aligned}\Sigma^2 &= \sum_{i=1}^{N} (q_i'^T q_i' + q_i^T R^T R q_i - q_i'^T R q_i - q_i' R^T q_i'^T) \quad | \ R^T R = I \\ &= \sum_{i=1}^{N} (q_i'^T q_i' + q_i^T q_i - 2 q_i'^T R q_i)\end{aligned}$$

因此最小化 Σ^2 就是到了最大化：

$$\begin{aligned}F &= \sum_{i=1}^{N} q_i'^T R q_i \\ &= Trace\left(\sum_{i=1}^{N} q_i'^T R q_i \right) \quad | \text{ 对于标量 } F, \text{ 有 } F = trace(F) \\ &= Trace\left(\sum_{i=1}^{N} R q_i q_i'^T \right) \quad | \ Trace(AB) = Trace(BA), \text{ 满足矩阵可移动} \\ &= Trace(RH)\end{aligned}$$

其中：

$$H \triangleq \sum_{i=1}^{N} q_i q_i'^T$$

引用定理：对于任意 正定矩阵 AA^T 和 任意 正交矩阵 B，均满足：$Trace(AA^T) \geq Trace(BAA^T)$

证明：令 a_i 为矩阵 A 的第 i 个列向量，那么

$$Trace(BAA^T) = Trace(A^T B A) = \sum_i a_i^T (B a_i)$$

根据 Schwarz 不等式：

$$a_i^T (B a_i) \leq \sqrt{(a_i^T a_i)(a_i^T B^T B a_i)} = a_i^T a_i$$

所以 $Trace(BAA^T) \leq \sum_i a_i^T a_i = Trace(AA^T)$

通过前述证明的引用定理：$Trace(AA^T) \geq Trace(BAA^T)$

可知：假如我们能够找到一个 R，使得 RH 能转换为 AA^T 的形式，那么在 UV 基础上乘上其他任意一个旋转矩阵（即正交矩阵 B），都不会使得 $Trace(AA^T)$ 变得更大了。因此能使 RH 转换为 AA^T 的 R 就是我们要找的，使 Trace(RH) 最大的旋转矩阵！

Step 3：找到 H 的 SVD 分解：

$$H = U\Lambda V^T$$

Step 4：计算

$$X = VU^T \qquad (X \text{为一个正交矩阵})$$

因为：

$$XH = VU^TU\Lambda V^T = V\Lambda V^T = V\Lambda^{\frac{1}{2}}\left(V\Lambda^{\frac{1}{2}}\right)^T = AA^T$$

所以：

当 $det(X) = +1$，X 满足作为一个旋转矩阵的要求

当 $det(X) = -1$，X 是一个反射旋转矩阵，不满足要求（这种情况几乎不出现）

由 step2 的证明，可知：

当 $det(X) = +1$，且 $R = X = VU^T$ 时，二次投关出数 (1) 最小

所以两个 3D 点集的最小二乘匹配解为：

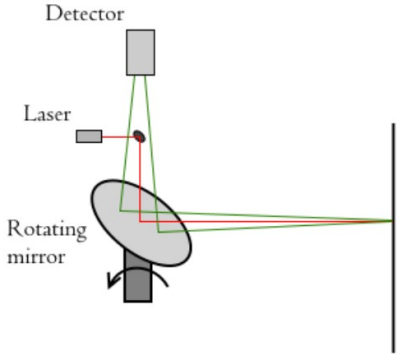
$$\hat{R} = VU^T$$

$$\hat{t} = p' - \hat{R}p$$

3. 阅读论文 Precise indoor localization for mobile laser scanner [2] 前两章,回答问题。

(1) 根据第二章内容,简述激光雷达测距原理。

激光扫描一个平面测距的示意图如下:



该激光扫描仪通过旋转镜面一周，将单个激光发射出的一个激光点对环境扫描一周，来测量一个 2D 平面。而像 Velodyne HDL-64E 可以使用多个激光扫描仪，制造出 64 个激光扫描线，意味着每一帧激光点云可以产生 64 线的激光束。

激光雷达可以通过测量飞行时间(TOF)或者通过测量波形的相位差来测量距离。

a) 通过测量飞行时间(TOF)来测量距离 r:

$$r = \frac{\Delta t \cdot c}{n \cdot 2}.$$

其中 Δt 为飞行时间，c 为常量光速，n 为空气的折射系数。

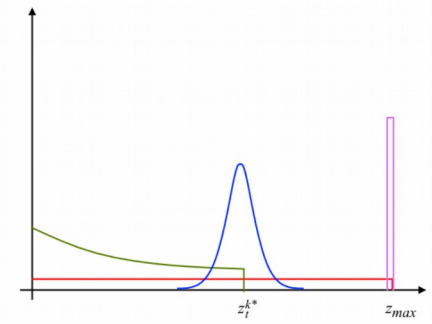
b) 通过测量波形的相位差来测量距离 *r*:

$$\Delta t = \Delta\varphi / (2 * \Pi * f_m)$$

再将 Δt 带入 a)中的式子可以求出距离 *r*。

其中 f_m 为载波的频率，Δφ 为出射和接收光线的相位差。

对于激光雷达测距的的概率模型如下所示:



其中蓝色钟形分布为测量值 z_t 相对于真实值 x_t 的条件概率分布，描述了距离测量的不确定性:

$$p(z_t|x_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t, m)$$

绿色曲线描述了目标物前有动态障碍物的概率分布，可以看到它是一个位于 0 到真实值 x_t 的呈下降趋势的指数分布。

z_{max} 是该激光雷达可测的最大距离，粉红色描述了由于超过该激光可测最大距离而导致测量失败的概率。

红色曲线描述了随机噪声的概率分布，它是一个位于 0 到 z_{max} 的均匀分布。

4. 简答题,开放性答案:设计使用 IMU 去除激光雷达运动畸变的方法并回答问题。

(1) 仅用 IMU 去除运动畸变可能会有哪些不足之处?

(1) IMU 测量线加速度精度太差，即使二次积分得到的位置精度仍然很差，使得无法准确去除激光雷达运动畸变。

(2) 在仅有 IMU 和激光雷达传感器的情况下,你会如何设计运动畸变去除方案(平移+旋转),达到较好的畸变去除效果?

(2) 利用 IMU 去除激光雷达运动畸变的方法和利用轮式里程计去除激光雷达运动畸变的方法类似，就是将一帧点云内的所有点的位姿转换到该帧点云的起始位姿下。

唯一不同在于，IMU 的测量频率极高，IMU 的更新时间可能远小于激光雷达的两个激光点之间测量时间。因此我们可以对一帧点云内的任意一个点直接使用 IMU 里程计的数据来获得扫描该激光点时机器人的位姿，而不必使用线性插值。

我们可以联合使用 VICP 和 IMU(仅使用 IMU 的角度信息)，来达到更好的畸变去除效果。

参考文献:

[1] Least-Squares Fitting of Two 3-D Points Sets, K.S. ARUN, T.S. HUANG, S.D. BLOSTEIN.

[2] Precise indoor localization for mobile laser scanner, Risto Kaijaluoto.