



多传感器融合定位

第3讲 3D激光里程计 II

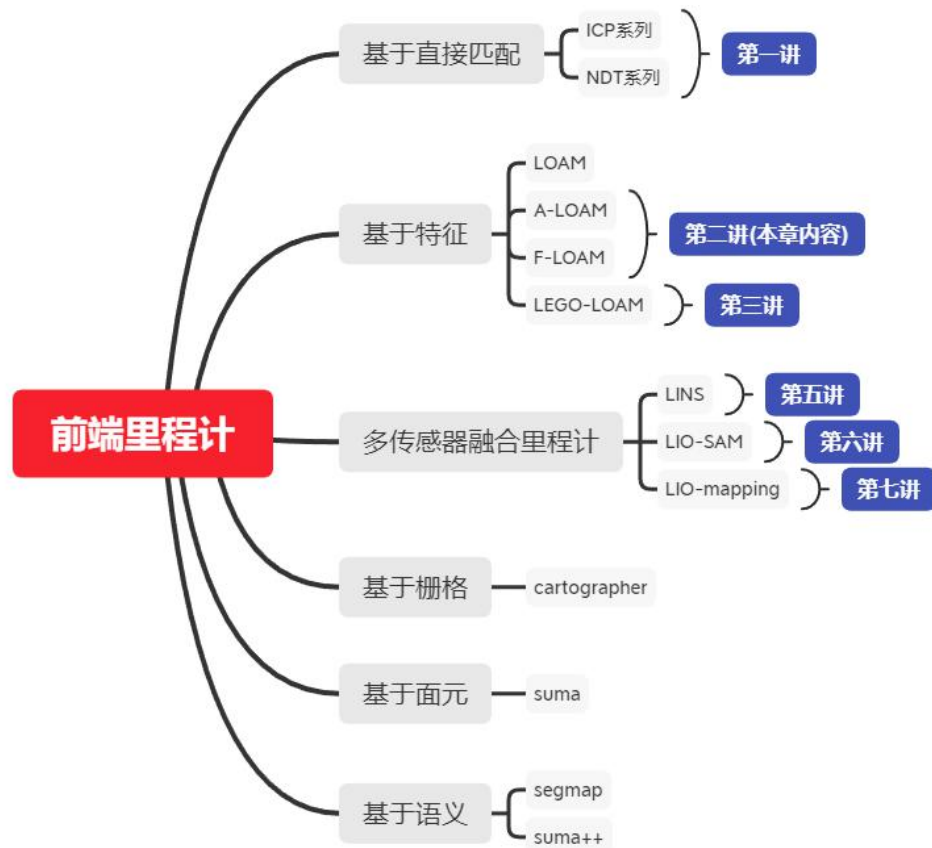
主讲人 任 乾

北京理工大学本硕
自动驾驶从业者





向量基本运算





目录



1. 点线面几何基础



2. 点云线面特征提取



3. 基于线面特征的位姿优化



4. 位姿优化代码实现



5. 相关开源里程计



向量基本运算

1. 向量运算及其几何意义

1) 内积定义

内积，又叫数量积，是向量的点乘。

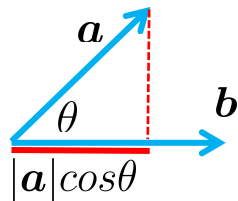
$$\mathbf{a} = (x_1, y_1, z_1)$$

$$\mathbf{b} = (x_2, y_2, z_2)$$

$$\mathbf{a} \bullet \mathbf{b} = x_1x_2 + y_1y_2 + z_1z_2$$

2) 内积几何意义

$$\mathbf{a} \bullet \mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$



当 \mathbf{b} 为单位向量时，内积就是 \mathbf{a} 在 \mathbf{b} 上的投影分量。



向量基本运算

1. 向量运算及其几何意义

3) 外积定义

外积，又叫叉积、向量积，是向量的叉乘。

$$\mathbf{a} = (x_1, y_1, z_1)$$

$$\mathbf{b} = (x_2, y_2, z_2)$$

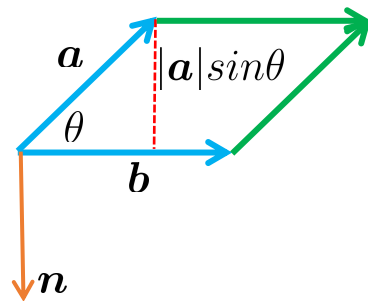
$$\mathbf{a} \times \mathbf{b}$$

$$= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

$$= (y_1 z_2 - y_2 z_1) \mathbf{i} \\ - (x_1 z_2 - x_2 z_1) \mathbf{j} \\ + (x_1 y_2 - x_2 y_1) \mathbf{k}$$

4) 外积几何意义

$$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin \theta$$



外积模长等于 \mathbf{a} 和 \mathbf{b} 组成的平行四边形的面积，

外积的方向满足右手定则， \mathbf{a} 和 \mathbf{b} 张成平面的

$$\text{单位法向量为: } \mathbf{n} = \frac{\mathbf{a} \times \mathbf{b}}{|\mathbf{a} \times \mathbf{b}|}$$



向量基本运算

1. 向量运算及其几何意义

5) 内积微分性质

从内积的定义出发，有

$$\frac{\partial \mathbf{a} \bullet \mathbf{b}}{\partial \mathbf{a}} = \mathbf{b}$$

证明：

$$\frac{\partial \mathbf{a} \bullet \mathbf{b}}{\partial x_1} = \frac{\partial (x_1 x_2 + y_1 y_2 + z_1 z_2)}{\partial x_1} = x_2$$

$$\frac{\partial \mathbf{a} \bullet \mathbf{b}}{\partial y_1} = \frac{\partial (x_1 x_2 + y_1 y_2 + z_1 z_2)}{\partial y_1} = y_2$$

$$\frac{\partial \mathbf{a} \bullet \mathbf{b}}{\partial z_1} = \frac{\partial (x_1 x_2 + y_1 y_2 + z_1 z_2)}{\partial z_1} = z_2$$

6) 外积微分性质

根据外积的定义，有

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]^{\wedge} \mathbf{b}$$

其中 \mathbf{a}^{\wedge} 为 \mathbf{a} 的反对称矩阵

$$\mathbf{a}^{\wedge} = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix}$$

请各位自行证明：

$$\mathbf{a}^{\wedge} \mathbf{b} = -\mathbf{b}^{\wedge} \mathbf{a}$$

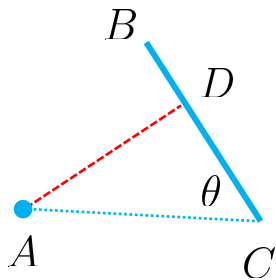
$$\frac{\partial \mathbf{a}^{\wedge} \mathbf{b}}{\partial \mathbf{a}} = -\frac{\mathbf{b}^{\wedge} \partial \mathbf{a}}{\partial \mathbf{a}} = -\mathbf{b}^{\wedge}$$



向量基本运算

2. 线面特征运算

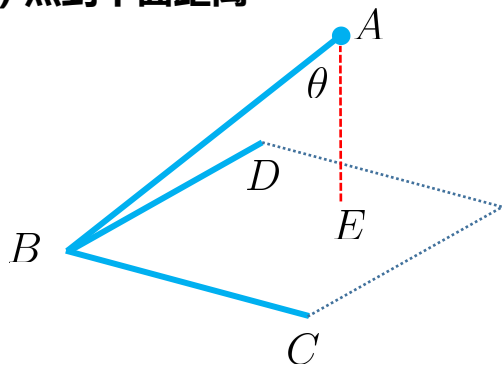
1) 点到直线距离



点 A 到直线 BC 的距离为

$$|\overrightarrow{AD}| = |\overrightarrow{CA}| \sin \theta = \frac{|\overrightarrow{CA} \times \overrightarrow{CB}|}{|\overrightarrow{CB}|}$$

2) 点到平面距离



平面 BCD 的单位法向量为 $\mathbf{n} = \frac{\overrightarrow{BC} \times \overrightarrow{BD}}{|\overrightarrow{BC} \times \overrightarrow{BD}|}$

点 A 到平面 BCD 的距离为

$$|\overrightarrow{AE}| = |\overrightarrow{AB}| \cos \theta = \overrightarrow{AB} \cdot \mathbf{n}$$



目录



1. 点线面几何基础



2. 点云线面特征提取



3. 基于线面特征的位姿优化



4. 位姿优化代码实现



5. 相关开源里程计



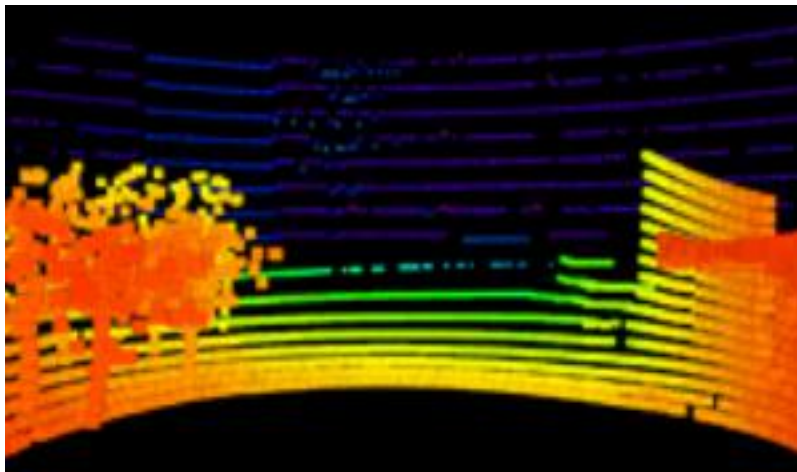
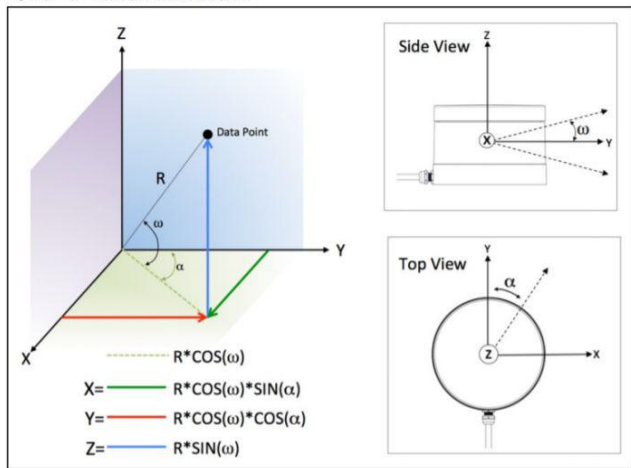
点云线面特征提取

1. 按线数分割

根据激光点坐标，可计算该束激光相比于雷达水平面的倾角 $\omega = \arctan \frac{z}{\sqrt{x^2+y^2}}$

根据倾角和雷达内参(各扫描线的设计倾角)，可知雷达属于哪条激光束。

Figure 9-1 VLP-16 Sensor Coordinate System



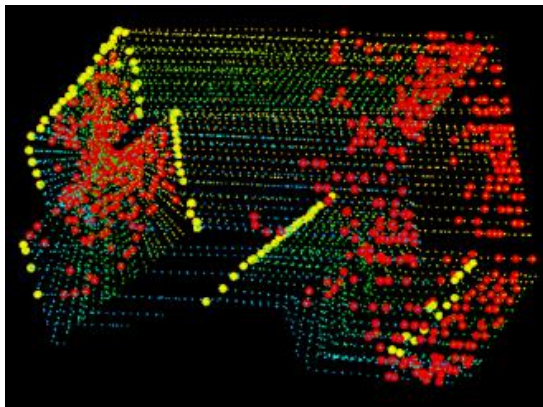


点云线面特征提取

2. 计算曲率

根据前后各5个点与当前点的长度(长度指激光点到雷达的距离), 计算曲率大小。

$$c = \frac{1}{\|X\|} \left\| \sum_i (X - X_i) \right\|$$



3. 按曲率大小筛选特征点

共分4类:

- a. 曲率特别大的点(sharp)
- b. 曲率大的点(less_sharp)
- c. 曲率特别小的点(flat)
- d. 曲率小的点(less_flat)

实际使用时:

- a. sharp 为 “点到直线” 中的 “点”
- b. sharp 和 less_sharp 为 “点到直线” 中的直线
- c. flat 为 “点到平面” 中的 “点”
- d. flat 和 less_flat 为 “点到平面” 中的 “平面”

参考文献: LOAM: Lidar Odometry and Mapping in Real-time Ji Zhang and Sanjiv Singh

推荐博客: <https://blog.csdn.net/robinvista/article/details/104379087>



目录



1. 点线面几何基础



2. 点云线面特征提取



3. 基于线面特征的位姿优化



4. ceres基础知识



5. 相关开源里程计



基于线面特征的位姿优化

1. 帧间关联

1) 点云位姿转换

第 $k+1$ 帧与第 k 帧的相对位姿为

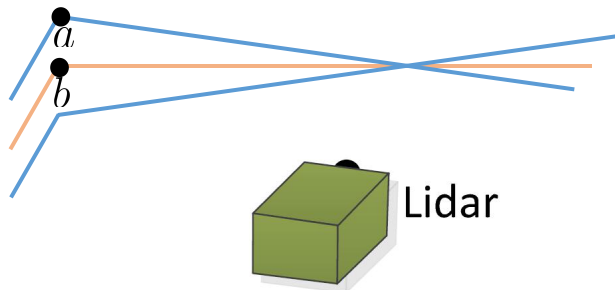
$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

第 $k+1$ 帧中的点 p_i 转到第 k 帧坐标系

$$\tilde{p}_i = R p_i + t$$

2) 线特征关联

当 p_i 为 sharp 时，在上一帧中搜索离 \tilde{p}_i 最近的线特征点，并在相邻线上再找一个线特征点，组成直线。



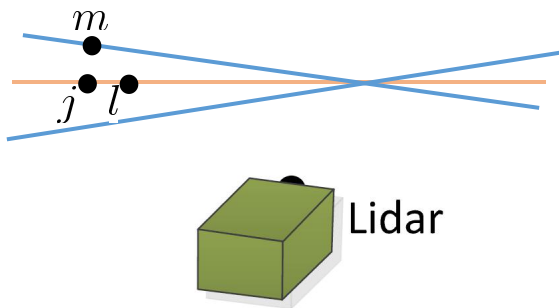


基于线面特征的位姿优化

1. 帧间关联

3) 面特征关联

当 p_i 为 flat 时，在上一帧中搜索离 \tilde{p}_i 最近的面特征点，并在相邻线上找两个面特征点，组成平面。

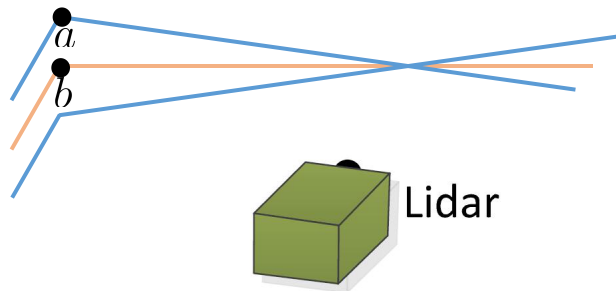




基于线面特征的位姿优化

2. 残差函数

1) 线特征



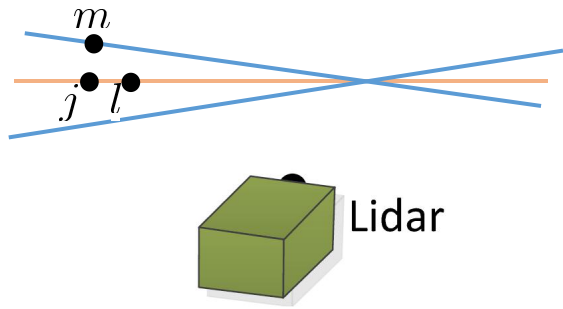
点到直线的距离

$$d_{\mathcal{E}} = \frac{|(\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)|}{|p_a - p_b|}$$

在实际代码中，使用的是矢量形式

$$d_{\mathcal{E}} = \frac{(\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)}{|p_a - p_b|}$$

2) 面特征



点到平面的距离

$$d_{\mathcal{H}} = \left| (\tilde{p}_i - p_j) \cdot \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|} \right|$$



基于线面特征的位姿优化

3. 位姿优化

按第一章介绍凸优化基础，只要求得残差关于待求变量的雅可比，便可采用高斯牛顿等进行优化。

1) 线特征残差雅可比

$$J_{\mathcal{E}} = \frac{\partial d_{\mathcal{E}}}{\partial T} = \frac{\partial d_{\mathcal{E}}}{\partial \tilde{p}_i} \frac{\partial \tilde{p}_i}{\partial T}$$

等号右边第二项与李代数相关，此处直接给出结论，推导过程见《视觉SLAM十四讲》第 4.3 节。

对平移的雅可比： $\frac{\partial \tilde{p}_i}{\partial t} = I$

对旋转的雅可比： $\frac{\partial \tilde{p}_i}{\partial R} = -(Rp_i)^{\wedge}$

等号右边第一项可以根据外积的微分性质，推导得到：

$$\begin{aligned} \frac{\partial d_{\mathcal{E}}}{\partial \tilde{p}_i} &= \frac{1}{|p_a - p_b|} \left(\frac{\partial (\tilde{p}_i - p_b)^{\wedge} (\tilde{p}_i - p_a)}{\partial \tilde{p}_i} + \frac{(\tilde{p}_i - p_b)^{\wedge} \partial (\tilde{p}_i - p_a)}{\partial \tilde{p}_i} \right) \\ &= \frac{1}{|p_a - p_b|} \left(-(\tilde{p}_i - p_a)^{\wedge} + (\tilde{p}_i - p_b)^{\wedge} \right) \\ &= \frac{(p_a - p_b)^{\wedge}}{|p_a - p_b|} \end{aligned}$$



基于线面特征的位姿优化

3. 位姿优化

2) 面特征残差雅可比

$$J_{\mathcal{H}} = \frac{\partial d_{\mathcal{H}}}{\partial T} = \frac{\partial d_{\mathcal{H}}}{\partial \tilde{p}_i} \frac{\partial \tilde{p}_i}{\partial T}$$

等号右边第二项与线特征的一致。

若令

$$X = (\tilde{p}_i - p_j) \bullet \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|}$$

则有

$$\frac{\partial d_{\mathcal{H}}}{\partial \tilde{p}_i} = \frac{\partial |X|}{\partial \tilde{p}_i} = \frac{\partial |X|}{\partial X} \frac{\partial X}{\partial \tilde{p}_i} = \frac{X}{|X|} \frac{\partial X}{\partial \tilde{p}_i}$$

对于等号右边第二项，根据内积的微分性质，有

$$\frac{\partial X}{\partial \tilde{p}_i} = \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|}$$

物理意义上，它代表的是平面的单位法向量。



目录



1. 点线面几何基础



2. 点云线面特征提取



3. 基于线面特征的位姿优化



4. 位姿优化代码实现



5. 相关开源里程计



位姿优化代码实现

1. ceres 基础知识

1) 基本概念

优化任务一般可以表示成如下形式：

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2} \sum_i \rho_i \left(\|f_i(x_{i_1}, \dots, x_{i_k})\|^2 \right) \\ \text{s.t.} \quad & l_j \leq x_j \leq u_j \end{aligned}$$

其中

- a. $\rho_i \left(\|f_i(x_{i_1}, \dots, x_{i_k})\|^2 \right)$ 称为残差块，即 ResidualBlock;
- b. $f_i(\cdot)$ 称为代价函数，对应之前讲的残差函数，即 CostFunction;
- c. $[x_{i_1}, \dots, x_{i_k}]$ 这一系列参数称为参数块，即 ParameterBlock;
- d. $\rho_i(\cdot)$ 称为损失函数，即 LossFunction。



位姿优化代码实现

2) 自动求导与解析求导

a. 自动求导

以一个简单的例子来说明该问题，假设代价函数为 $f(x) = 10 - x$

则首先编写 CostFunctor 的代码如下：

```
struct CostFunctor {  
    template <typename T>  
    bool operator()(const T* const x, T* residual) const {  
        residual[0] = T(10.0) - x[0];  
        return true;  
    }  
};
```



位姿优化代码实现

随后可直接构建 ceres 优化问题

```
int main(int argc, char** argv) {
    google::InitGoogleLogging(argv[0]);

    // The variable to solve for with its initial value.
    double initial_x = 5.0;
    double x = initial_x;

    // Build the problem.
    Problem problem;

    // Set up the only cost function (also known as residual). This uses
    // auto-differentiation to obtain the derivative (jacobian).
    CostFunction* cost_function =
        new AutoDiffCostFunction<CostFunctor, 1, 1>(new CostFunctor);
    problem.AddResidualBlock(cost_function, NULL, &x);
}
```

此处的 AutoDiffCostFunction即代表当前模式为自动求导，它使用 CostFunctor中的残差公式自动求解出导数，而不需要手动给出导数形式。



位姿优化代码实现

b. 解析求导

解析求导的含义就是直接给出导数的解析形式，而不是ceres去推导。

```
class QuadraticCostFunction : public ceres::SizedCostFunction<1, 1> {  
    // 定义一个CostFunction或 SizedCostFunction（如果参数和残差在编译时就已知了）的子类。  
    public:  
        virtual ~QuadraticCostFunction() {}  
        virtual bool Evaluate(double const* const* parameters,  
                               // 输入参数数组  
                               double* residuals,  
                               // 输出残差数组  
                               double** jacobians) const {  
            // 输出雅可比行列式  
            const double x = parameters[0][0];  
            residuals[0] = 10 - x;  
  
            // Compute the Jacobian if asked for.  
            if (jacobians != NULL && jacobians[0] != NULL) {  
                jacobians[0][0] = -1;  
            }  
            return true;  
        }  
};
```

- 第一个参数为ResidualBlock维数
- 第二个参数为第一个ParameterBlock维数
- 当有多个ParameterBlock时，此处参数就不只两个

- 给出雅可比时，ceres会直接使用该雅可比
- 不给出雅可比时，ceres就会自动去求导



位姿优化代码实现

随后可构建优化问题

```
int main(int argc, char** argv) {  
    google::InitGoogleLogging(argv[0]);  
  
    // 初始化待优化变量  
    double x = 0.5;  
    const double initial_x = x;  
  
    // 构建问题  
    Problem problem;  
  
    // 设置残差函数  
    CostFunction* cost_function = new QuadraticCostFunction;  
    problem.AddResidualBlock(cost_function, NULL, &x);  
}
```

这种使用方式，就是vio/llo中使用ceres构建优化问题的方式



位姿优化代码实现

c. 自动求导与解析求导的对比

- 自动求导实现方便，但效率会比解析求导低 (比较 A-LOAM 和 F-LOAM)；
- 实际使用中，能够自动求导且效率没有形成障碍的，优先使用自动求导；
- 除这两种方法外，还有数值求导 (SLAM问题中不常见，不过多介绍)。



位姿优化代码实现

2. 自动求导实现位姿优化(A-LOAM)

1) 线特征

```
LidarEdgeFactor(Eigen::Vector3d curr_point_, Eigen::Vector3d last_point_a_,
                Eigen::Vector3d last_point_b_, double s_)
: curr_point(curr_point_), last_point_a(last_point_a_), last_point_b(last_point_b_), s(s_) {}
template <typename T>
bool operator()(const T *q, const T *t, T *residual) const
{
    Eigen::Matrix<T, 3, 1> cp{T(curr_point.x()), T(curr_point.y()), T(curr_point.z())};
    Eigen::Matrix<T, 3, 1> lpa{T(last_point_a.x()), T(last_point_a.y()), T(last_point_a.z())};
    Eigen::Matrix<T, 3, 1> lpb{T(last_point_b.x()), T(last_point_b.y()), T(last_point_b.z())};

    //Eigen::Quaternion<T> q_last_curr{q[3], T(s) * q[0], T(s) * q[1], T(s) * q[2]};
    Eigen::Quaternion<T> q_last_curr{q[3], q[0], q[1], q[2]};
    Eigen::Quaternion<T> q_identity{T(1), T(0), T(0), T(0)};
    // 考虑运动补偿, kitti点云已经补偿过所以可以忽略下面的对四元数slerp插值以及对平移的线性插值
    q_last_curr = q_identity.slerp(T(s), q_last_curr);
    Eigen::Matrix<T, 3, 1> t_last_curr{T(s) * t[0], T(s) * t[1], T(s) * t[2]};

    Eigen::Matrix<T, 3, 1> lp;
    // Odometry线程时, 下面是将当前帧Lidar坐标系下的cp点变换到上一帧的Lidar坐标系下, 然后在上一帧的Lidar坐标系下计算点到线的残差距离
    // Mapping线程时, 下面是将当前帧Lidar坐标系下的cp点变换到world坐标系下, 然后在world坐标系下计算点到线的残差距离
    lp = q_last_curr * cp + t_last_curr;

    // 点到线的计算如下图所示
    Eigen::Matrix<T, 3, 1> nu = (lp - lpa).cross(lp - lpb);
    Eigen::Matrix<T, 3, 1> de = lpa - lpb;

    // 最终的残差本来是residual[0] = nu.norm() / de.norm(); 为啥也分成3个, 我也不知
    // 道, 从我试验的效果来看, 确实是下面的残差函数形式, 最后输出的pose精度会好一点, 这里需要
    // 注意的是, 所有的residual都不用加fabs, 因为Ceres内部会对其求平方作为最终的残差项
    residual[0] = nu.x() / de.norm();
    residual[1] = nu.y() / de.norm();
    residual[2] = nu.z() / de.norm();

    return true;
}
```

传入点(p_i)和线(p_a, p_b)

转换点云, 并计算残差(与前面推导公式一致), 但不在代码中输入雅可比



位姿优化代码实现

2. 自动求导实现位姿优化(A-LOAM)

2) 面特征

```
struct LidarPlaneFactor {
    LidarPlaneFactor(Eigen::Vector3d curr_point_, Eigen::Vector3d last_point_j_,
                     Eigen::Vector3d last_point_l_, Eigen::Vector3d last_point_m_,
                     double s_)
        : curr_point(curr_point_),
          last_point_j(last_point_j_),
          last_point_l(last_point_l_),
          last_point_m(last_point_m_),
          s(s_) {}

    // 点l、j、m就是搜索到的最近邻的3个点，下面就是计算出这三个点构成的平面ljm的法向量
    ljm_norm = (last_point_j - last_point_l).cross(last_point_j - last_point_m);
    // 归一化法向量
    ljm_norm.normalize();
}

template <typename T>
bool operator()(const T *q, const T *t, T *residual) const {
    Eigen::Matrix<T, 3, 1> cp{T(curr_point.x()), T(curr_point.y()),
                             T(curr_point.z())};
    Eigen::Matrix<T, 3, 1> lpj{T(last_point_j.x()), T(last_point_j.y()),
                             T(last_point_j.z())};
    Eigen::Matrix<T, 3, 1> ljm{T(ljm_norm.x()), T(ljm_norm.y()),
                             T(ljm_norm.z())};

    Eigen::Quaternion<T> q_last_curr{q[3], q[0], q[1], q[2]};
    Eigen::Quaternion<T> q_identity{T(1), T(0), T(0), T(0)};
    q_last_curr = q_identity.slerp(T(s), q_last_curr);
    Eigen::Matrix<T, 3, 1> t_last_curr{T(s) * t[0], T(s) * t[1], T(s) * t[2]};

    Eigen::Matrix<T, 3, 1> lp;
    lp = q_last_curr * cp + t_last_curr;

    // 计算点到平面的残差距离，如下图所示
    residual[0] = (lp - lpj).dot(ljm);

    return true;
}
```

传入点(p_i) 和面(p_j, p_l, p_m)

转换点云，并计算残差
(与前面推导公式一致)，
但在代码中输入雅可比



位姿优化代码实现

3. 解析求导实现位姿优化(F-LOAM)

1) 线特征

```
bool EdgeAnalyticCostFunction::Evaluate(double const *const *parameters,
                                         double *residuals,
                                         double **jacobians) const {
    Eigen::Map<const Eigen::Quaterniond> q_last_curr(parameters[0]);
    Eigen::Map<const Eigen::Vector3d> t_last_curr(parameters[0] + 4);
    Eigen::Vector3d lp;

    lp = q_last_curr * curr_point + t_last_curr; // new point
    Eigen::Vector3d nu = (lp - last_point_a).cross(lp - last_point_b);
    Eigen::Vector3d de = last_point_a - last_point_b;

    residuals[0] = nu.x() / de.norm();
    residuals[1] = nu.y() / de.norm();
    residuals[2] = nu.z() / de.norm();

    if (jacobians != NULL) {
        if (jacobians[0] != NULL) {
            Eigen::Matrix3d skew_lp = skew(lp);
            Eigen::Matrix<double, 3, 6> dp_by_so3;
            dp_by_so3.block<3, 3>(0, 0) = -skew_lp;
            (dp_by_so3.block<3, 3>(0, 3)).setIdentity();
            Eigen::Map<Eigen::Matrix<double, 3, 7, Eigen::RowMajor>> J_se3(
                jacobians[0]);
            J_se3.setZero();
            Eigen::Vector3d re = last_point_b - last_point_a;
            Eigen::Matrix3d skew_re = skew(re);

            J_se3.block<3, 6>(0, 0) = skew_re * dp_by_so3 / de.norm();
        }
    }

    return true;
}
```

转换点云，并计算
残差(与前面推导公
式一致)

手动输入点到线模
型的雅可比



位姿优化代码实现

3. 解析求导实现位姿优化(F-LOAM)

2) 面特征

```
bool SurfNormAnalyticCostFunction::Evaluate(double const *const *parameters,
                                             double *residuals,
                                             double **jacobians) const {
    Eigen::Map<const Eigen::Quaterniond> q_w_curr(parameters[0]);
    Eigen::Map<const Eigen::Vector3d> t_w_curr(parameters[0] + 4);
    Eigen::Vector3d point_w = q_w_curr * curr_point + t_w_curr;

    residuals[0] = plane_unit_norm.dot(point_w) + negative_OA_dot_norm;

    if (jacobians != NULL) {
        if (jacobians[0] != NULL) {
            Eigen::Matrix3d skew_point_w = skew(point_w);

            Eigen::Matrix<double, 3, 6> dp_by_so3;
            dp_by_so3.block<3, 3>(0, 0) = -skew_point_w;
            (dp_by_so3.block<3, 3>(0, 3)).setIdentity();
            Eigen::Map<Eigen::Matrix<double, 1, 7, Eigen::RowMajor>> J_se3(
                jacobians[0]);
            J_se3.setZero();
            J_se3.block<1, 6>(0, 0) = plane_unit_norm.transpose() * dp_by_so3;
        }
    }
    return true;
}
```

转换点云，并计算
残差(与前面推导公
式一致)

手动输入点到面模
型的雅可比



目录



1. 点线面几何基础



2. 点云线面特征提取



3. 基于线面特征的位姿优化



4. 位姿优化代码实现

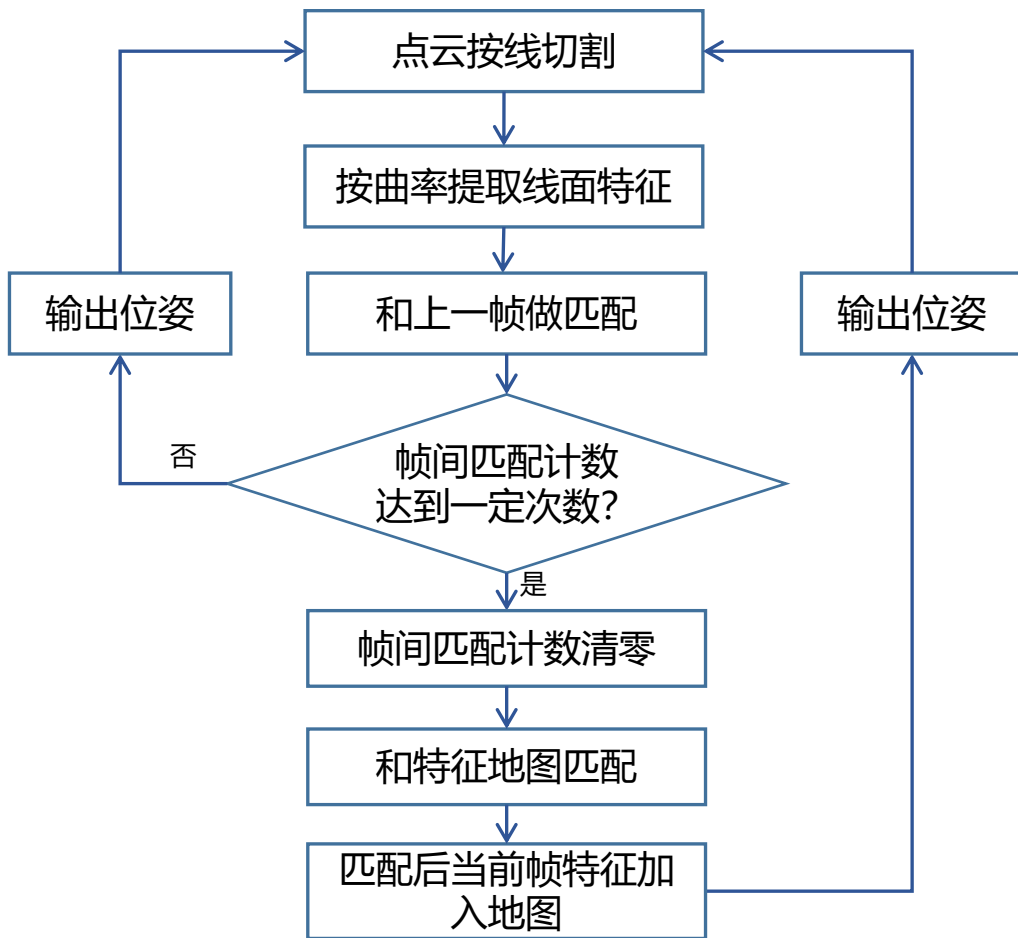


5. 相关开源里程计



相关开源里程计

1. 基于特征的里程计实现流程

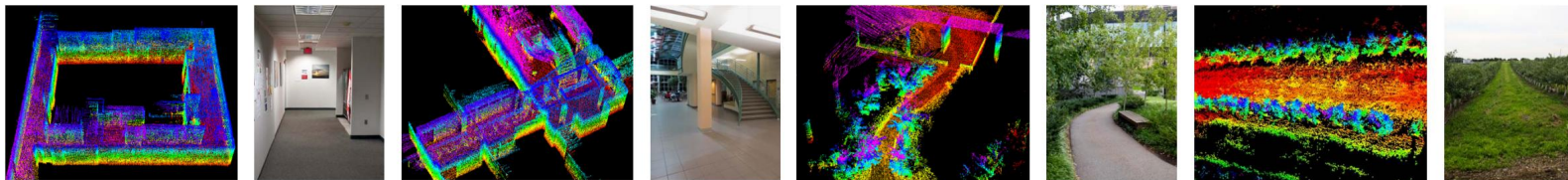




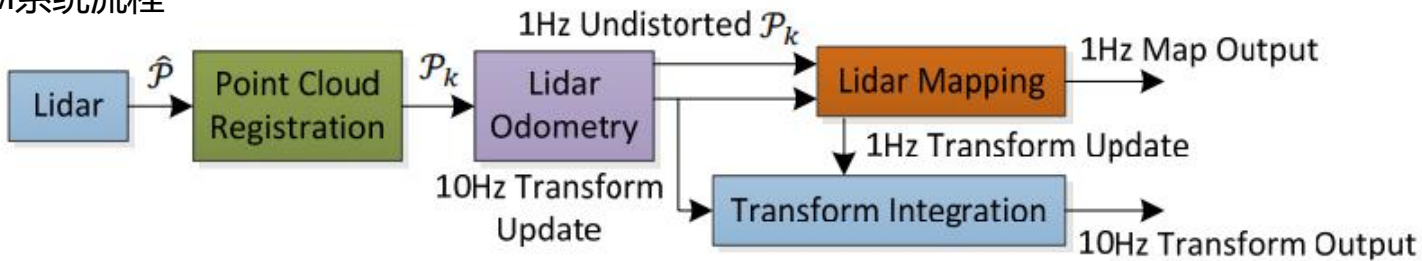
相关开源里程计

1. 基于特征的里程计实现流程

1) LOAM



LOAM系统流程



论文: LOAM: Lidar Odometry and Mapping in Real-time, Ji Zhang and Sanjiv Singh



相关开源里程计

2) A-LOAM

主要特点

- 1) 去掉了和IMU相关的部分
- 2) 使用Eigen (四元数) 做位姿转换, 简化了代码
- 3) 使用ceres做迭代优化, 简化了代码, 但降低了效率

代码: <https://github.com/HKUST-Aerial-Robotics/A-LOAM>

(课程页面也提供下载)

(讲解 A-LOAM 代码)



相关开源里程计

3) F-LOAM

主要特点

1) 整体和ALOAM类似，只是使用残差函数的雅可比使用的是解析式求导

代码: <https://github.com/wh200720041/floam>

(课程页面也提供下载)

(讲解 F-LOAM 代码)



作业

内容：

请使用以下残差模型，推导相应的雅可比，并在 F-LOAM 或 A-LOAM 基于该模型，实现解析式求导。

线特征残差：
$$d_{\mathcal{E}} = \frac{|(\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)|}{|p_a - p_b|}$$

面特征残差：
$$d_{\mathcal{H}} = (\tilde{p}_i - p_j) \bullet \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|}$$

评价标准：

- 1) 及格：推导雅可比，且结果正确；
- 2) 良好：在及格的基础上，编程实现新模型的解析式求导，且结果正常；
- 3) 优秀：在良好的基础上，给出运行结果的精度评测结果(基于evo)。

感谢聆听 !
Thanks for Listening

