深蓝学院
shenlanxueyuan.com

多传感器融合第一章第三题实现讲解

主讲人 周朋伟

第三题

Generalized-ICP 实现思路讲解

给定存在的两组点云集合：$\widehat{\mathrm{A}} = \{\widehat{\mathrm{a}}_i\}$ 和$\widehat{\mathrm{B}} = \{\widehat{\mathrm{b}}_i\}$

每个测量点假设都是服从高斯分布的，记为如下形式：

$$a_i \sim \mathcal{N}\left(\hat{a}_i, C_i^{\dot{A}}\right) \quad b_i \sim \mathcal{N}\left(\hat{b}_i, C_i^{B}\right)$$

$$\hat{b}_i = \mathbf{T}^* \hat{a}_i$$

误差函数

由于我们假设两个集合中的点是服从独立的高斯分布，当$\mathbf{T}^*$为最优解时，误差函数将会服从如下形式的高斯分布：

$$\begin{aligned} d_i^{(\mathbf{T}^*)} &\sim \mathcal{N}\left(\hat{b}_i - (\mathbf{T}^*)\hat{a}_i, C_i^B + (\mathbf{T}^*)C_i^A(\mathbf{T}^*)^T\right) \\ &= \mathcal{N}\left(0, C_i^B + (\mathbf{T}^*)C_i^A(\mathbf{T}^*)^T\right) \end{aligned}$$

之后我们为了求解最优解，只需使得概率乘积达到最大：

$$\mathbf{T} = \underset{\mathbf{T}}{\operatorname{argmax}} \prod_i p\left(d_i^{(\mathbf{T})}\right) = \underset{\mathbf{T}}{\operatorname{argmax}} \sum_i \log\left(p\left(d_i^{(\mathbf{T})}\right)\right)$$

将高斯分布展开之后，等价于最小化下式：

$$\mathbf{T} = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_i d_i^{(\mathbf{T})^T}\left(C_i^B + \mathbf{T}C_i^A\mathbf{T}^T\right)^{-1} d_i^{(\mathbf{T})}$$ 其中：$d_i^{(\mathbf{T})} = (b_i - \mathbf{T}a_i)$

雅克比矩阵

$$f(\boldsymbol{x}) = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_i \left\| \left(C_i^B + \mathbf{T}_{\text{last}}\, C_i^A \mathbf{T}_{\text{last}}{}^T \right)^{-1} (b_i - \mathbf{T} a_i) \right\|^2$$

使用李代数来推导雅克比矩阵的形式: se3_pose = { $\varphi_1, \varphi_1, \varphi_1, x, y, z$ }.

对于旋转部分利用左扰动模型求导

$$\frac{\partial(\boldsymbol{R}\boldsymbol{a}_i + \boldsymbol{t})}{\partial \varphi} = -(\boldsymbol{R}\boldsymbol{a}_i)^\wedge \qquad \frac{\partial(\boldsymbol{R}\boldsymbol{a}_i + \boldsymbol{t})}{\partial \boldsymbol{t}} = \boldsymbol{I}$$

$$\boldsymbol{J}_{[3,6]} = \left(C_i^B + \mathbf{T}_{\text{last}}\, C_i^A \mathbf{T}_{\text{last}}^T \right)^{-1} \{ (\boldsymbol{R}\boldsymbol{a}_i)^\wedge, -\boldsymbol{I} \}$$

$$\boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{J}(\boldsymbol{x}) \Delta \boldsymbol{x} = -\boldsymbol{J}(\boldsymbol{x})^T f(\boldsymbol{x}) \qquad H \Delta x = g$$

计算增量，迭代更新。

协方差矩阵

计算每个点的协方差矩阵后，做SVD分解，按照如下形式进行归一化。

$$C_i^B = \mathbf{U}_{bi} \cdot \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{V}_{bi}^T$$

$$C_i^A = \mathbf{U}_{ai} \cdot \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{V}_{ai}^T$$

其中: ϵ 表示法向量计算出来的不确定度，设定成常数如：1e-2。

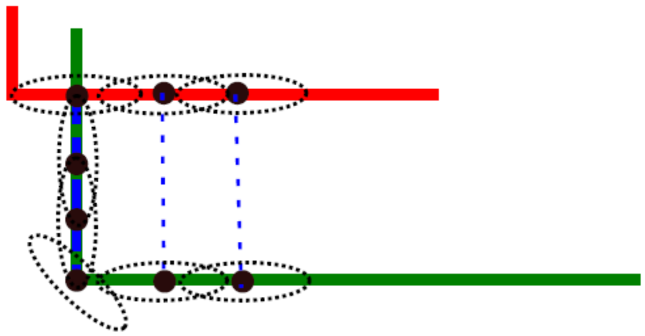

Fig. 1. illustration of plane-to-plane

Generalized-ICP 直观解释，绿色平面中垂直部分的点全部是错误的对应的关系，但是由于这部分点的表面分布不一致，最终累加起来的协方差矩阵将会是各向同性(各个方向的分布近似相等)。其对于误差函数的影响很小。

计算loss和雅克比矩阵

深蓝学院
shenlanxueyuan.com

```
const auto& mean_A = source_->at(i).getVector4fMap();

const auto& cov_A = source_covs[i];

const auto& mean_B = target_->at(target_index).getVector4fMap();

const auto& cov_B = target_covs[target_index];

Eigen::Vector4f mean_A_temp = mean_A;
Eigen::Vector4f transed_mean_A = trans * mean_A;
Eigen::Vector4f d = mean_B - transed_mean_A;
Eigen::Matrix4f RCR = cov_B + trans * cov_A * trans.transpose();
RCR(3, 3) = 1;

Eigen::Matrix4f RCR_inv = RCR.inverse();
Eigen::Vector4f RCRd = RCR_inv * d;

Eigen::Matrix<float, 4, 6> dtdx0 = Eigen::Matrix<float, 4, 6>::Zero();
dtdx0.block<3, 3>(0, 0) = skew(transed_mean_A.head<3>());

dtdx0.block<3, 3>(0, 3) = -Eigen::Matrix3f::Identity();

Eigen::Matrix<float, 4, 6> jlossexp = RCR_inv * dtdx0;
int n = count++;

losses[n] = RCRd.head<3>();
Js[n] = jlossexp.block<3, 6>(0, 0);
```

迭代求解

深蓝学院
shenlanxueyuan.com

```
for(int i = 0; i < max_iterations_; i++) {
    nr_iterations_ = i;
    update_correspondences(x0);
    Eigen::MatrixXf J;
    Eigen::VectorXf loss = loss_ls(x0, &J);

    Eigen::Matrix<float, 6, 1> delta =
    solver.delta(loss.cast<double>(), J.cast<double>()).cast<float>();

    Eigen::Isometry3f x0_ = Eigen::Isometry3f::Identity();
    x0_.linear() = Sophus::SO3f::exp(x0.head<3>()).matrix();
    x0_.translation() = x0.tail<3>();

    Eigen::Isometry3f delta_ = Eigen::Isometry3f::Identity();
    delta_.linear() = Sophus::SO3f::exp(delta.head<3>()).matrix();
    delta_.translation() = delta.tail<3>();

    Eigen::Isometry3f x1_ = delta_.inverse() * x0_;

    x0.head<3>() = Sophus::SO3f(x1_.linear()).log();
    x0.tail<3>() = x1_.translation();

    if(is_converged(delta)) {
        converged_ = true;
        break;
    }
}
```

性能改进

深蓝学院 shenlanxueyuan.com

可尝试不同的降采样策略：

- Voxel sampling
- Random sampling
- Skip sampling

可关闭运动畸变部分的旋转部分，只删除起始和终止附近一定角度的点云

```
float start_orientation = atan2(origin_cloud_ptr->points[0].y, origin_cloud_ptr->points[0].x);
pcl::transformPointCloud(*origin_cloud_ptr, *origin_cloud_ptr, transform_matrix);
```

由于初始的第一个点的角度值计算出来并不稳定，实测大约会在0.3°~9°附近变化，这样会给初始值带来额外的误差。

Reference：

https://github.com/SMRT-AIST/fast_gicp

https://github.com/PointCloudLibrary/pcl/blob/master/registration/include/pcl/registration/impl/gicp.hpp

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening！