多传感器融合第一次作业讲评

主讲人 王志勇

第一题

深蓝学院
shenlanxueyuan.com

- 第一题主要涉及下载数据集和配置环境
- rviz的命令为 rviz -d display_bag.rviz
- 如果环境存在问题可以使用课程提供的docker镜像

第二题

- 在第一题的基础上修改代码。仿照老师写好的NDT匹配写一个PCL的ICP就可以。
- 注意事项 ： 需要使用头文件 <pcl/registration/icp.h>
- 在 lidar_localization/src/models/registration/里 和 lidar_localization/include/lidar_localization/models/registration/ 仿照 老师提供的NDT匹配写一个ICP匹配的.hpp和.cpp
- 修改config.yaml 仿照老师提供的NDT参数写ICP的参数
- 在lidar_localization/src/front_end/front_end.cpp 中 FrontEnd::InitRegistration 添加ICP匹配

第二题

深蓝学院
shenlanxueyuan.com

```cpp
#ifndef LIDAR_LOCALIZATION_MODELS_REGISTRATION_ICP_REGISTRATION_HPP_
#define LIDAR_LOCALIZATION_MODELS_REGISTRATION_ICP_REGISTRATION_HPP_

#include "lidar_localization/models/registration/registration_interface.hpp"
#include <pcl/registration/icp.h>

namespace lidar_localization {
    class ICPRegistraion: public RegistrationInterface {
    public:
        ICPRegistraion(const YAML::Node& node);
        ICPRegistraion(float max_dist, float trans_eps, float eculi_eps, int max_iter);

        bool SetInputTarget(const CloudData::CLOUD_PTR& input_target) override;
        bool ScanMatch(const CloudData::CLOUD_PTR& input_source,
                       const Eigen::Matrix4f& predict_pose,
                       CloudData::CLOUD_PTR& result_cloud_ptr,
                       Eigen::Matrix4f& result_pose) override;

    private:
        bool SetRegistrationParam(float max_dist, float trans_eps, float eculi_eps, int max_iter);
    private:
        pcl::IterativeClosestPoint<CloudData::POINT,CloudData::POINT>::Ptr icp_ptr_;

    };

}

#endif //LIDAR_LOCALIZATION_MODELS_REGISTRATION_ICP_REGISTRATION_HPP_
```

第二题

```cpp
#include "lidar_localization/models/registration/icp_registration.hpp"

#include "glog/logging.h"

namespace lidar_localization {
    ICPRegistraion::ICPRegistraion(const YAML::Node &node)
        :icp_ptr_(new pcl::IterativeClosestPoint<CloudData::POINT,CloudData::POINT>()){

        float max_dist = node["max_dist"].as<float>();
        float trans_eps = node["trans_eps"].as<float>();
        float eculi_eps = node["eculi_eps"].as<float>();
        int max_iter = node["max_iter"].as<int>();

        SetRegistrationParam(max_dist, trans_eps, eculi_eps, max_iter);
    }
    ICPRegistraion::ICPRegistraion(float max_dist, float trans_eps, float eculi_eps, int max_iter)
        :icp_ptr_(new pcl::IterativeClosestPoint<CloudData::POINT,CloudData::POINT>()){

        SetRegistrationParam(max_dist,trans_eps,eculi_eps,max_iter);
    }
    bool ICPRegistraion::SetRegistrationParam(float max_dist, float trans_eps, float eculi_eps, int max_iter) {
        icp_ptr_->setMaxCorrespondenceDistance(max_dist);
        icp_ptr_->setTransformationEpsilon(trans_eps);
        icp_ptr_->setEuclideanFitnessEpsilon(eculi_eps);
        icp_ptr_->setMaximumIterations(max_iter);

        LOG(INFO) << "ICP 的匹配参数为: " << std::endl
                  << "max_dist: " << max_dist << ","
                  << "trans_eps: " << trans_eps << ","
                  << "eculi_eps: " << eculi_eps << ","
                  << "max_iter: " << max_iter << ","
```

第二题

```
        return true;

    }
    bool ICPRegistraion::SetInputTarget(const CloudData::CLOUD_PTR &input_target) {
        icp_ptr_->setInputTarget(input_target);

        return true;
    }
    bool ICPRegistraion::ScanMatch(const CloudData::CLOUD_PTR &input_source,
                                   const Eigen::Matrix4f &predict_pose,
                                   CloudData::CLOUD_PTR &result_cloud_ptr,
                                   Eigen::Matrix4f &result_pose) {
        icp_ptr_->setInputSource(input_source);
        icp_ptr_->align(*result_cloud_ptr,predict_pose);
        result_pose = icp_ptr_->getFinalTransformation();

        return true;

    }
}
```

第二题

深蓝学院
shenlanxueyuan.com

```cpp
bool FrontEnd::InitRegistration(std::shared_ptr<RegistrationInterface>& registration_ptr, const YA
    std::string registration_method = config_node["registration_method"].as<std::string>();
    LOG(INFO) << "点云匹配方式为: " << registration_method;

    if (registration_method == "NDT") {
        registration_ptr = std::make_shared<NDTRegistration>(config_node[registration_method]);
    }
    else if (registration_method == "ICP") {
        registration_ptr = std::make_shared<ICPRegistraion>(config_node[registration_method]);

    }
    else {
        LOG(ERROR) << "没找到与 " << registration_method << " 相对应的点云匹配方式!";
        return false;
    }

    return true;
```

第二题

```
ICP:
    max_dist : 1
    trans_eps : 0.01
    eculi_eps : 0.01
    max_iter : 30
```

第三题

这里介绍来自LZM_HIT同学的优秀作业

3.1 作业思路

考虑到PCL_ICP中默认用的是SVD分解的方法，这里选择使用高斯牛顿的迭代方法求解ICP，自己手动推导一下相关的过程，步骤如下：

- a. 输入目标点云

  目标点云用来构建一个kdtree，方便待匹配的点云寻找最近点。

- b. 输入待匹配的点云以及初始位姿$\mathbf{R}$，$\mathbf{t}$

1.初始化$Hessian<6,6>=Zero, B<6,1>=Zero$，对待匹配点云中的每个点$\boldsymbol{p}_i$,利用初始位姿转换到目标点云坐标系下的到$\boldsymbol{p}_i^{'}=\boldsymbol{R}*\boldsymbol{p_i}+\boldsymbol{t}$

2.根据$kdtree$在目标点云中搜索到离$\boldsymbol{p}_i^{'}$最近的点$\boldsymbol{q}_j^{'}$,计算距离误差$f=\boldsymbol{p}_i^{'}-\boldsymbol{q}_i^{'}$，若$f.norm$小于设置的距离上限，执行下一步，否则，该点对舍弃。

3.对找到的每一个点对，计算`Jacobian<3,6>`，`Hessian_i`，`B_i`

第三题

3.对找到的每一个点对，计算`Jacobian<3,6>`，`Hessian_i`，`B_i`

$$\text{对平移的求导}\ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{t}} = \boldsymbol{I}$$

$$\text{对旋转的求导，右导数}\ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{R}} = -\boldsymbol{R} * \boldsymbol{p}_i^{\wedge}$$

$$Jacobian = [\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{t}}, \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{R}}]$$

$$Hessian_i = Jacobian^T Jacobian$$

$$B_i = -Jacobian^T * \boldsymbol{f}$$

4.计算所有点对计算的`Hessian_i`，`B_i`求和，计算状态增量$\boldsymbol{\delta x}$

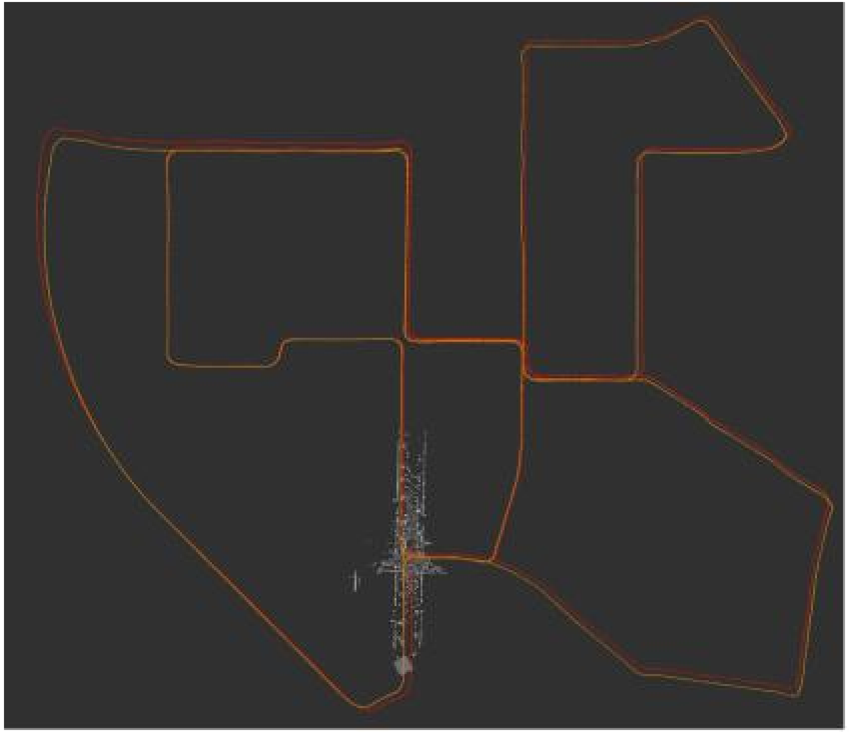
$$Hessian = \Sigma Hessian_i$$

$$B = \Sigma B_i$$

$$\boldsymbol{\delta x} = Hessian.inverse() * B$$

5.利用增量$\boldsymbol{\delta x}$计算新的$\boldsymbol{R}$，$\boldsymbol{t}$，返回第一步计算，直到达到收敛条件或者迭代次数

第三题



3.3 对比结果

分段统计精度	max	mean	median	min	rmse	sse	std
PCL_NDT	**1.852103**	**0.786486**	**0.692123**	0.211582	**0.875525**	**34.494444**	**0.384686**
PCL_ICP	1759.394213	274.894422	76.649318	0.655707	512.77679	11832301.6618	432.866138
ICP_MANUAL	8.473569	3.405850	3.357728	**0.151904**	3.981459	713.340758	2.062087

整体轨迹误差	max	mean	median	min	rmse	sse	std
PCL_NDT	68.038200	22.565926	18.100196	0.000001	30.304401	4161992.585209	20.227103
PCL_ICP	3541.7586	526.18734	105.68439	0.000001	1059.34568	5081381713.0555	919.423817
ICP_MANUAL	**25.627234**	**12.309445**	**11.004779**	0.000001	**14.704755**	**979088.602983**	**8.044090**

从结果可以看出:

对于**分段统计精度**而言，精度从高到低是：`PCL_NDT>ICP_MANUAL>PCL_ICP`

对于**整体轨迹误差**而言，精度从高到低是：`ICP_MANUAL>PCL_NDT>PCL_ICP`

```cpp
void ICPRegistrationManual::calculateTrans(const CloudData::CLOUD_PTR& input_source)
{
 CloudData::CLOUD_PTR transformed_cloud(new CloudData::CLOUD);
 int knn = 1;
 int iterator_num = 0;
 while (iterator_num < max_iterator_)
 {
  pcl::transformPointCloud(*input_source, *transformed_cloud, transformation_);
  Eigen::Matrix<float, 6, 6> Hessian;
  Eigen::Matrix<float, 6, 1> B;
  Hessian.setZero();
  B.setZero();

  for (size_t i = 0; i < transformed_cloud->size(); ++i)
  {
   auto ori_point = input_source->at(i);
   if (!pcl::isFinite(ori_point))
    continue;
   auto transformed_point = transformed_cloud->at(i);
   std::vector<float> distasnces;
   std::vector<int> indexs;
   kdtree_ptr_->nearestKSearch(transformed_point, knn, indexs, distasnces);
   if (distasnces[0] > max_correspond_distance_)
   {
    continue;
   }
   Eigen::Vector3f closet_point = Eigen::Vector3f(target_cloud_->at(indexs[0]).x, target_cloud_->at(indexs[0]).y,
                        target_cloud_->at(indexs[0]).z);
   Eigen::Vector3f err_dis =
     Eigen::Vector3f(transformed_point.x, transformed_point.y, transformed_point.z) - closet_point;

   Eigen::Matrix<float, 3, 6> Jacobian(Eigen::Matrix<float, 3, 6>::Zero());
   Jacobian.leftCols<3>() = Eigen::Matrix3f::Identity();
   Jacobian.rightCols<3>() =
     -rotation_matrix_ * Sophus::SO3f::hat(Eigen::Vector3f(ori_point.x, ori_point.y, ori_point.z));
```

```cpp
            Hessian += Jacobian.transpose() * Jacobian;
            B += -Jacobian.transpose() * err_dis;
        }
        iterator_num++;
        if (Hessian.determinant() == 0)
        {
            continue;
        }
        Eigen::Matrix<float, 6, 1> delta_x = Hessian.inverse() * B;

        translation_ += delta_x.head<3>();
        auto delta_rotation = Sophus::SO3f::exp(delta_x.tail<3>());
        rotation_matrix_ *= delta_rotation.matrix();

        transformation_.block<3, 3>(0, 0) = rotation_matrix_;
        transformation_.block<3, 1>(0, 3) = translation_;
    }
}
} // namespace lidar_localization
```

在线问答

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听!

Thanks for Listening