VIO-hw6-提升作业

VIO

提升题：代码部分

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <Eigen/Eigenvalues>
struct Pose {
    Pose(Eigen::Matrix3d R, Eigen::Vector3d t) : Rwc(R), qwc(R), twc(t) {};
    Eigen::Matrix3d Rwc;
    Eigen::Quaterniond qwc;
    Eigen::Vector3d twc;

    Eigen::Vector2d uv;     // 这帧图像观测到的特征坐标
};
int main() {
    int poseNums = 10;
    double radius = 8;
    double fx = 1.;
    double fy = 1.;
    std::vector<Pose> camera_pose;
    // 随机生成pose的位置
    for (int n = 0; n < poseNums; ++n) {
        double theta = n * 2 * M_PI / (poseNums * 4); // 1/4 圆弧
        // 绕 z轴 旋转
        Eigen::Matrix3d R;
        R = Eigen::AngleAxisd(theta, Eigen::Vector3d::UnitZ());
        Eigen::Vector3d t = Eigen::Vector3d(radius * cos(theta) - radius, radius * sin(theta), 1 * sin(2 * theta));
        camera_pose.push_back(Pose(R, t));
    }

    // 随机数生成 1 个 三维特征点
```

```
    // 只是为了生成landmark，实际是不知道landmark的pose的，只有landmark在uv下的坐标和一个深度值未知的量
    std::default_random_engine generator;
    std::uniform_real_distribution<double> xy_rand(-4, 4.0);
    std::uniform_real_distribution<double> z_rand(8., 10.);
    double tx = xy_rand(generator);
    double ty = xy_rand(generator);
    double tz = z_rand(generator);

    Eigen::Vector3d Pw(tx, ty, tz);
    std::cout << "Ground truth:" << Pw.transpose() << std::endl;
    // 这个特征从第三帧相机开始被观测，i=3
    int start_frame_id = 3;
    int end_frame_id = poseNums;
    //从第三帧开始，计算这一个特征点在每一帧图像里的归一化坐标
    for (int i = start_frame_id; i < end_frame_id; ++i) {
        Eigen::Matrix3d Rcw = camera_pose[i].Rwc.transpose();//原本是wc，transpose变成cw
        Eigen::Vector3d Pc = Rcw * (Pw - camera_pose[i].twc);

        double x = Pc.x();
        double y = Pc.y();
        double z = Pc.z();

        camera_pose[i].uv = Eigen::Vector2d(x / z, y / z);
    }

    /// TODO::homework; 请完成三角化估计深度的代码
    // 遍历所有的观测数据，并三角化
    Eigen::Vector3d P_est;            // 结果保存到这个变量
    P_est.setZero();
    /* your code begin */
    auto loop_times = camera_pose.size() - start_frame_id;
    Eigen::MatrixXd D((loop_times) * 2, 4);
    for (int j = 0; j < loop_times; ++j) {
        Eigen::MatrixXd T_tmp(3, 4);
        T_tmp.block<3, 3>(0, 0) = camera_pose[j + 3].Rwc.transpose();
        T_tmp.block<3, 1>(0, 3) = -camera_pose[j + 3].Rwc.transpose() * camera_pose[j + 3].twc;
        auto P_k1 = T_tmp.block<1, 4>(0, 0);
        auto P_k2 = T_tmp.block<1, 4>(1, 0);
        auto P_k3 = T_tmp.block<1, 4>(2, 0);

        D.block<1, 4>(2 * j, 0) = camera_pose[j + 3].uv[0] * P_k3 - P_k1;
```

```
            D.block<1, 4>(2 * j + 1, 0) = camera_pose[j + 3].uv[1] * P_k3 - P_k2;
        }
        Eigen::Matrix4d D_res = D.transpose() * D;
        Eigen::JacobiSVD<Eigen::Matrix4d> svd(D_res, Eigen::ComputeFullU | Eigen::ComputeFullV);
        auto res_U = svd.matrixU();
        auto res_V = svd.matrixV();
//      std::cout<<"Trans="<<Trans.rows()<<" "<<Trans.cols()<<std::endl;
        std::cout << "U=" << res_U << std::endl;
        auto tmp = res_U.rightCols(1);
        std::cout << "res=" << tmp / tmp(3) << std::endl;
        /* your code end */
//      std::cout <<"D: \n"<< D.transpose()*D <<std::endl;
//      std::cout <<"ground truth: \n"<< Pw.transpose() <<std::endl;
//      std::cout <<"your result: \n"<< P_est.transpose() <<std::endl;
        return 0;
}
```

输出结果：

coordinate	results	groundtruth
x	-2.9477	-2.9477
y	-0.330799	-0.330799
z	8.43792	8.43792