从零开始手写VIO 第六课作业

边城量子 2019.07.24

基础题

1. 证明式(15)中, 取$y = u_4$ 是该问题的最优解。 提示： 设 $y' = u_4 + v$, 其中 v 正交于 u_4 , 证明：

$y'^{\top}\mathbf{D}^{\top}\mathbf{D}y' \geq y^{\top}\mathbf{D}^{\top}\mathbf{D}y$

该方法基于奇异值构造矩阵零空间的理论。

其中式（15）如下所示:

寻找最小二乘解： $\min_{\mathbf{y}} \|\mathbf{D}\mathbf{y}\|_2^2, \qquad s.t. \|\mathbf{y}\| = 1$

对 $\mathbf{D}^{\top}\mathbf{D}$ 进行 SVD:

$\mathbf{D}^{\top}\mathbf{D} = \sum_{i=1}^{4} \sigma_i^2 \mathbf{u}_i \mathbf{u}_j$ ， 其中 σ_i 为奇异值， 且由大到小排列， u_i, u_j 正交。

回答:

- **方法1： 使用不等式进行证明**

  假设矩阵 $\mathbf{D}^{\top}\mathbf{D}$ 的四个特征值为 $\lambda_1, \lambda_2, \lambda_3, \lambda_4$, 对应的特征向量分为 $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4$, 各特征向量相互正交;

  令 $\mathbf{v} = a_1\mathbf{u}_1 + a_2\mathbf{u}_2 + a_3\mathbf{u}_3$, 可知 $\mathbf{u}_4$ 与 $\mathbf{v}$ 也正交;

  设 $\mathbf{y}' = a\mathbf{u}_4 + b\mathbf{v}$, 由于有 $||y|| = 1$, 所以 a, b 满足 $a^2 + b^2 = 1$;

  则:

$$\begin{aligned}\mathbf{y}'^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{y}' &= (\mathbf{u}_4 + \mathbf{v})^{\top}\mathbf{D}^{\top}\mathbf{D}(\mathbf{u}_4 + \mathbf{v}) \\ &= \underbrace{\mathbf{u}_4^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{u}_4}_{\text{不等式右边}} + \underbrace{\mathbf{u}_4^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{v} + \mathbf{v}^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{u}_4}_{u_4 \text{与} v \text{的所有分量均正交，乘积为} 0} + \underbrace{\mathbf{v}^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{v}}_{\text{值} \geq 0} \\ &\geq \mathbf{u}_4^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{u}_4 = \mathbf{y}^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{y} = \lambda_4^2\end{aligned}$$

  备注: 其中 $\mathbf{v}^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{v} = (\mathbf{D}\mathbf{v})^{\top}\mathbf{D}\mathbf{v} = \sum_{i=1}^{3} a_i^2\lambda_i^2\mathbf{u}_i^{\top}\mathbf{u}_i \geq 0$

  针对其他特征向量 $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ 也可以进行相同的操作， 也同样可以得到类似的式子， 最终不等号右边将分别为$\lambda_1^2, \lambda_2^2, \lambda_3^2$。

  但根据已知条件 , λ_4 是四个特征值中最小的。 因此可知上式是是左边最小的， 因此$\mathbf{u}_4$ 是使 $\mathbf{y}^{\top}\mathbf{D}^{\top}\mathbf{D}\mathbf{y}$ 最小的那个 $\mathbf{y}$ 。

  证毕。

- **方法2： 使用矩阵性质进行证明**

$$
\begin{aligned}
\min_{\|\mathbf{y}\|_2=1} \|\mathbf{D}\mathbf{y}\|_2^2 &= \min_{\|\mathbf{y}\|_2=1} \|U\Sigma V^\top \mathbf{y}\|_2^2 \\
&= \min_{\|VV^\top \mathbf{y}\|_2=1} \underbrace{\|U\Sigma(V^\top \mathbf{y})\|_2^2}_{令 V^\top \mathbf{y}=\mathbf{x}} \\
&= \min_{\|\mathbf{x}\|_2=1} \|\Sigma\mathbf{x}\|_2^2 \\
&= \min_{\|\mathbf{x}\|_2=1} \underbrace{(\sigma_1^2 y_1^2 + \cdots + \sigma_n^2 y_n^2)}_{因为\sigma_i依次递减} \geq \sigma_n^2
\end{aligned}
$$

最小的是 $x = e_n$, 即 $y = Vx = vn$

提升题

完成代码部分.

回答

- 修改代码 `triangulate.cpp` 中的 `main()` 函数, 新增代码片段如下:

```
60
61      /// TODO:homework; 请完成三角化估计深度的代码
62      // 遍历所有的观测数据，并三角化
63      Eigen::Vector3d P_est;           // 结果保存到这个变量
64      P_est.setZero();
65
66      /* your code begin */
67      // 定义矩阵D的行数(对应课件公式15)
68      //    注:每一帧做三角化，都会产生2条数据(u,v各产生一条，因此矩阵D的行数为2倍帧数)
69      int matrix_size = 2*( end_frame_id - start_frame_id );
70      // 矩阵D 第四列是 平移部分
71      Eigen::MatrixXd  D ( Eigen::MatrixXd::Zero( matrix_size, 4 ));
72      // 矩阵 DtD = D.transpose() * D, 为方阵
73      Eigen::MatrixXd  DtD ( Eigen::MatrixXd::Zero( matrix_size, matrix_size ));
74
75      // 当前处理的是D矩阵的第几个块(每帧产生的算一块)
76      int index = 0;
77      // 开始构造D矩阵，每便利一帧,都根据公式13，向D矩阵插入2个部分(均为3*4矩阵)
78      for( int i=start_frame_id; i<end_frame_id; ++i) {
79          // 像素坐标(u,v)
80          double u = camera_pose[i].uv.x();
81          double v = camera_pose[i].uv.y();
82          // 得到 Rcw(从word到camer的旋转阵)
83          Eigen::MatrixXd Rcw = camera_pose[i].Rwc.transpose();
84          // X0w = RwcXc + tw
85          // Xt = RX + t
86          // Xc = RcwXw - Rcwtw
87          // Xt = R( R0trans()X0 - R0t0 ) + t
88          // 所给的Pose是在camera下的，转到world系. 上述第三行就是下面公式原因
89          Eigen::Vector3d t = -Rcw * camera_pose[i].twc;
90
91          Eigen::Matrix<double, 3, 4> P;
92          P.block(0,0,3,3).noalias() = Rcw;
93          P.block(0,3,3,1) = t;
94
95          // 得到课件公式12中的P_k,1  P_k,2   P_k,3
96          Eigen::MatrixXd P3 = P.row(2);
97          Eigen::MatrixXd P1 = P.row(0);
98          Eigen::MatrixXd P2 = P.row(1);
99
```

```
99
100         // 套用课件公式13
101         D.row(2*index+0) = u*P3 - P1;
102         D.row(2*index+1) = v*P3 - P2;
103         index++;
104     }
105
106     // 求解Dy=0, 由于方程式超定的, 因此:
107     //     对DTD进行SVD, 然后取最小特征值对应的那个特征向量, 即为y
108     DtD = D.transpose()*D;
109     std::cout << "DTD:\n" << DtD << std::endl;
110
111     Eigen::JacobiSVD<Eigen::MatrixXd> svd_ (DtD,  Eigen::ComputeThinU | Eigen::ComputeThinV );
112     Eigen::MatrixXd U_ = svd_.matrixU();
113     Eigen::MatrixXd V_ = svd_.matrixV();
114     Eigen::MatrixXd S_ = svd_.singularValues();
115
116     std::cout << "U:\n" << U_ << std::endl;
117     std::cout << "V:\n" << V_ << std::endl;
118     std::cout << "S:\n" << S_ << std::endl;
119
120     if ( S_(3) < S_(2) ) {
121         P_est(0) = V_(0,3)/V_(3,3);
122         P_est(1) = V_(1,3)/V_(3,3);
123         P_est(2) = V_(2,3)/V_(3,3);
124     }
125
126     /* your code end */
127
128     std::cout <<"ground truth: \n"<< Pw.transpose() <<std::endl;
129     std::cout <<"your result: \n"<< P_est.transpose() <<std::endl;
130     return 0;
131 }
132
```

- 执行结果如下:

```
hadoop@ubuntu:~/Documents/course6_hw/build$ ./estimate_depth
DTD:
        7         0 -0.486169   24.7361
        0         7   5.90714  -47.5284
-0.486169   5.90714    5.6799  -47.4055
  24.7361  -47.5284  -47.4055   457.196
U:
 0.0530721   0.846878    0.41558  -0.327528
 -0.103079   0.431629  -0.895388 -0.0367562
 -0.102585   0.309021   0.122288   0.937565
  0.987945  0.0316285  -0.103049   0.111113
V:
0.0530721  0.846878   0.41558  0.327528
-0.103079  0.431629 -0.895388 0.0367562
-0.102585  0.309021  0.122288 -0.937565
 0.987945 0.0316285 -0.103049 -0.111113
S:
    468.406
    7.74642
   0.723255
5.30104e-16
ground truth:
  -2.9477 -0.330799   8.43792
your result:
  -2.9477 -0.330799   8.43792
hadoop@ubuntu:~/Documents/course6_hw/build$
```

- 附: 修改后完整的cpp文件代码如下:

```
//
// Created by hyj on 18-11-11.
//
#include <iostream>
#include <vector>
#include <random>
#include <Eigen/Core>
#include <Eigen/Dense>
#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;
    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
    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);
    // 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;
    for (int i = start_frame_id; i < end_frame_id; ++i) {
        Eigen::Matrix3d Rcw = camera_pose[i].Rwc.transpose();
        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 */
    // 定义矩阵D的行数(对应课件公式15)
    //   注:每一帧做三角化，都会产生2条数据(u,v各产生一条，因此矩阵D的行数为2倍帧数)
    int matrix_size = 2*( end_frame_id - start_frame_id );
    // 矩阵D 第四列是 平移部分
    Eigen::MatrixXd  D ( Eigen::MatrixXd::Zero( matrix_size, 4 ));
    // 矩阵 DtD = D.transpose() * D, 为方阵
    Eigen::MatrixXd  DtD ( Eigen::MatrixXd::Zero( matrix_size, matrix_size ));

    // 当前处理的是D矩阵的第几个块(每帧产生的算一块)
    int index = 0;
    // 开始构造D矩阵，每便利一帧,都根据公式13，向D矩阵插入2个部分(均为3*4矩阵)
    for( int i=start_frame_id; i<end_frame_id; ++i) {
        // 像素坐标(u,v)
        double u = camera_pose[i].uv.x();
        double v = camera_pose[i].uv.y();
        // 得到 Rcw(从word到camer的旋转阵)
        Eigen::MatrixXd Rcw = camera_pose[i].Rwc.transpose();
        // X0w = RwcXc + tw
        // Xt = RX + t
        // Xc = RcwXw - Rcwtw
        // Xt = R( R0trans()X0 - R0t0 ) + t
        // 所给的Pose是在camera下的，转到world系. 上述第三行就是下面公式原因
        Eigen::Vector3d t = -Rcw * camera_pose[i].twc;

        Eigen::Matrix<double, 3, 4> P;
        P.block(0,0,3,3).noalias() = Rcw;
        P.block(0,3,3,1) = t;

        // 得到课件公式12中的P_k,1  P_k,2   P_k,3
        Eigen::MatrixXd P3 = P.row(2);
        Eigen::MatrixXd P1 = P.row(0);
        Eigen::MatrixXd P2 = P.row(1);

        // 套用课件公式13
        D.row(2*index+0) = u*P3 - P1;
        D.row(2*index+1) = v*P3 - P2;
        index++;
    }

    // 求解Dy=0，由于方程式超定的，因此:
    //    对DTD进行SVD，然后取最小特征值对应的那个特征向量，即为y
    DtD = D.transpose()*D;
    std::cout << "DTD:\n" << DtD << std::endl;

    Eigen::JacobiSVD<Eigen::MatrixXd> svd_ (DtD,  Eigen::ComputeThinU |
Eigen::ComputeThinV );
```

```
    Eigen::MatrixXd U_ = svd_.matrixU();
    Eigen::MatrixXd V_ = svd_.matrixV();
    Eigen::MatrixXd S_ = svd_.singularValues();

    std::cout << "U:\n" << U_ << std::endl;
    std::cout << "V:\n" << V_ << std::endl;
    std::cout << "S:\n" << S_ << std::endl;

    if ( S_(3) < S_(2) ) {
        P_est(0) = V_(0,3)/V_(3,3);
        P_est(1) = V_(1,3)/V_(3,3);
        P_est(2) = V_(2,3)/V_(3,3);
    }

    /* your code end */

    std::cout <<"ground truth: \n"<< Pw.transpose() <<std::endl;
    std::cout <<"your result: \n"<< P_est.transpose() <<std::endl;
    return 0;
}
```