

## ex.2

关键代码：

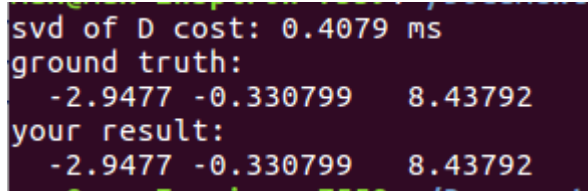
```
Eigen::Vector3d P_est;           // 结果保存到这个变量
P_est.setZero();
/* your code begin */
/// size = rows x 4
int rows = (end_frame_id - start_frame_id)*2;
Eigen::MatrixXd D(Eigen::MatrixXd::Zero(rows, 4));
int k = 0;
for (int i = start_frame_id; i < end_frame_id; ++i)
{
    Eigen::Matrix<double, 3, 4> Pcw = Eigen::Matrix<double, 3, 4>::Zero();
    Pcw.block(0, 0, 3, 3).noalias() += camera_pose[i].Rwc.transpose();
    Pcw.block(0, 3, 3, 1).noalias() += -camera_pose[i].Rwc.transpose() * camera_pose[i].twc;
    //std::cout << Pcw << std::endl;

    D.row(k) = camera_pose[i].uv[0] * Pcw.row(2) - Pcw.row(0);
    D.row(k+1) = camera_pose[i].uv[1] * Pcw.row(2) - Pcw.row(1);
    k++;
}
TicToc t_svd;
Eigen::Vector4d triangulated_point;
triangulated_point = (D.transpose()*D).jacobiSvd(Eigen::ComputeFullU).matrixU().rightCols<1>();
//triangulated_point = D.jacobiSvd(Eigen::ComputeFullV).matrixV().rightCols<1>();
//std::cout << "svd of D cost: " << t_svd.toc() << " ms" << std::endl;
std::cout << "svd of D^TD cost: " << t_svd.toc() << " ms" << std::endl;

P_est(0) = triangulated_point(0) / triangulated_point(3);
P_est(1) = triangulated_point(1) / triangulated_point(3);
P_est(2) = triangulated_point(2) / triangulated_point(3);
/* your code end */
```

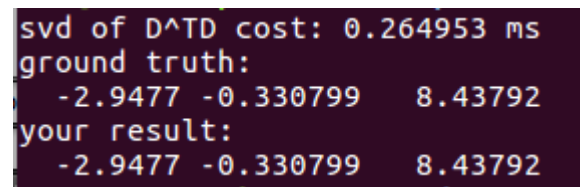
这里直接对 $D$ 进行SVD分解，取 $\mathbf{v}_4$ 也能得到正确结果，理论依据参考ex.1.pdf。这里对 $D^T D$ 进行SVD分解，取 $\mathbf{u}_4$ 。前者因为矩阵规模比后者大，所以时间会耗时比较多，输出结果：

直接对 $D$ 进行SVD分解，取 $\mathbf{v}_4$ ：



```
svd of D cost: 0.4079 ms
ground truth:
-2.9477 -0.330799 8.43792
your result:
-2.9477 -0.330799 8.43792
```

对 $D^T D$ 进行SVD分解，取 $\mathbf{u}_4$ ：



```
svd of D^TD cost: 0.264953 ms
ground truth:
-2.9477 -0.330799 8.43792
your result:
-2.9477 -0.330799 8.43792
```