深蓝学院
shenlanxueyuan.com

基于滤波的融合方法

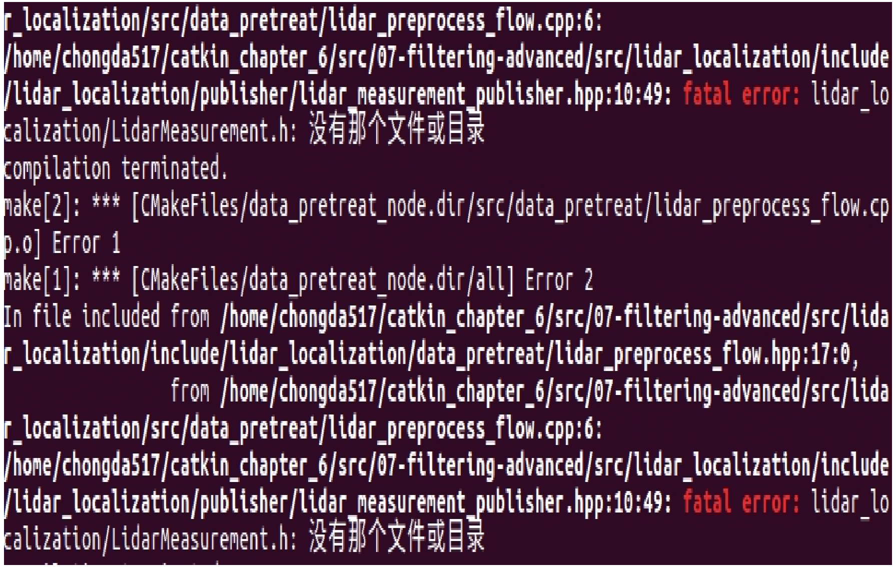
主讲人 雍川

纲要

- 第一部分：框架调试
- 第二部分：实现滤波算法

框架调试

- 说明：调试方法仅适用于第一版框架



```
r_localization/src/data_pretreat/lidar_preprocess_flow.cpp:6:
/home/chongda517/catkin_chapter_6/src/07-filtering-advanced/src/lidar_localization/include
/lidar_localization/publisher/lidar_measurement_publisher.hpp:10:49: fatal error: lidar_lo
calization/LidarMeasurement.h: 没有那个文件或目录
compilation terminated.
make[2]: *** [CMakeFiles/data_pretreat_node.dir/src/data_pretreat/lidar_preprocess_flow.cp
p.o] Error 1
make[1]: *** [CMakeFiles/data_pretreat_node.dir/all] Error 2
In file included from /home/chongda517/catkin_chapter_6/src/07-filtering-advanced/src/lida
r_localization/include/lidar_localization/data_pretreat/lidar_preprocess_flow.hpp:17:0,
                 from /home/chongda517/catkin_chapter_6/src/07-filtering-advanced/src/lida
r_localization/src/data_pretreat/lidar_preprocess_flow.cpp:6:
/home/chongda517/catkin_chapter_6/src/07-filtering-advanced/src/lidar_localization/include
/lidar_localization/publisher/lidar_measurement_publisher.hpp:10:49: fatal error: lidar_lo
calization/LidarMeasurement.h: 没有那个文件或目录
```

图1.1

```
> ${ALL_SRCS})
} ${PROJECT_NAME}_generate_messages_cpp)
${ALL_TARGET_LIBRARIES})

ode.cpp ${ALL_SRCS})
ETS} ${PROJECT_NAME}_generate_messages_cpp)
S} ${ALL_TARGET_LIBRARIES} )

cess_node.cpp ${ALL_SRCS})
ARGETS} ${PROJECT_NAME}_generate_messages_cpp)
RIES} ${ALL_TARGET_LIBRARIES})

${ALL_SRCS})
 ${PROJECT_NAME}_generate_messages_cpp)
{ALL_TARGET_LIBRARIES})

ALL_SRCS})
${PROJECT_NAME}_generate_messages_cpp)
ALL_TARGET_LIBRARIES})

e.cpp ${ALL_SRCS})
TS} ${PROJECT_NAME}_generate_messages_cpp)
} ${ALL_TARGET_LIBRARIES})

SRCS})
PROJECT_NAME}_generate_messages_cpp)
L_TARGET_LIBRARIES})

ng_node.cpp ${ALL_SRCS})
RGETS} ${PROJECT_NAME}_generate_messages_cpp)
IES} ${ALL_TARGET_LIBRARIES})

ss_node.cpp ${ALL_SRCS})
RGETS} ${PROJECT_NAME}_generate_messages_cpp)
IES} ${ALL_TARGET_LIBRARIES})

ins_sim_preprocess_node.cpp ${ALL_SRCS})
ORTED_TARGETS} ${PROJECT_NAME}_generate_messages_cpp)
Y_LIBRARIES} ${ALL_TARGET_LIBRARIES})
```

图1.2

如若出现图1.1中的编译问题，修改目录/06-filtering-yc/src/lidar_localization/下的CMakeLists文件，如图1.2。

框架调试

● 说明：**调试方法仅适用于第一版框架**

修改kitti_filtering.yaml文件下的地图和回环检测所需文件的路径kitti_filtering.yaml的路径如图1.3所示，修改后如图1.4。

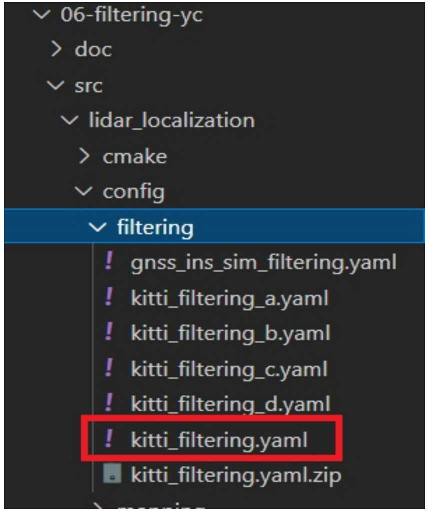


图1.3

```
# loop closure for localization initialization/re-initialization:
loop_closure_method: scan_context # 选择回环检测方法, 目前支持scan_context
scan_context_path: /home/chongda517/catkin_chapter_6/src/07-filtering-advanced/src/lidar_localization/slam_data/scan_context

# 匹配

# 全局地图
map_path: /home/chongda517/catkin_chapter_6/src/07-filtering-advanced/src/lidar_localization/slam_data/map/filtered_map.pcd
global_map_filter: voxel_filter # 选择滑窗地图点云滤波方法, 目前支持: voxel_filter、no_filter

# 局部地图
```

图1.4

框架调试

●说明：调试方法仅适用于第一版框架.

本地编译需要重新生成scan_context需要的文件，如图1.5。在slam_data/文件夹下新建trajectory，如图1.6

打开lidar_localization/config/scan_context文件夹，输入如下命令，生成pb文件

```bash
protoc --cpp_out=./ key_frames.proto
protoc --cpp_out=./ ring_keys.proto
protoc --cpp_out=./ scan_contexts.proto
mv key_frames.pb.cc key_frames.pb.cpp
mv ring_keys.pb.cc ring_keys.pb.cpp
mv scan_contexts.pb.cc scan_contexts.pb.cpp
```

分别修改生成的三个.pb.cpp文件。如下，以ring_keys.pb.cpp为例。

```C++
// Generated by the protocol buffer compiler.  DO NOT EDIT!
// source: ring_keys.proto

#define INTERNAL_SUPPRESS_PROTOBUF_FIELD_DEPRECATION
#include "ring_keys.pb.h" 替换为  #include "lidar_localization/models/scan_context_manager/ring_keys.pb.h"

#include <algorithm>
```

之后，用以上步骤生成的的.pb.h文件替换lidar_localization/include/lidar_localization/models/scan_context_manager
中的同名文件。
将.pb.cpp文件替换lidar_localization/src/models/scan_context_manager中的同名文件。

图1.5

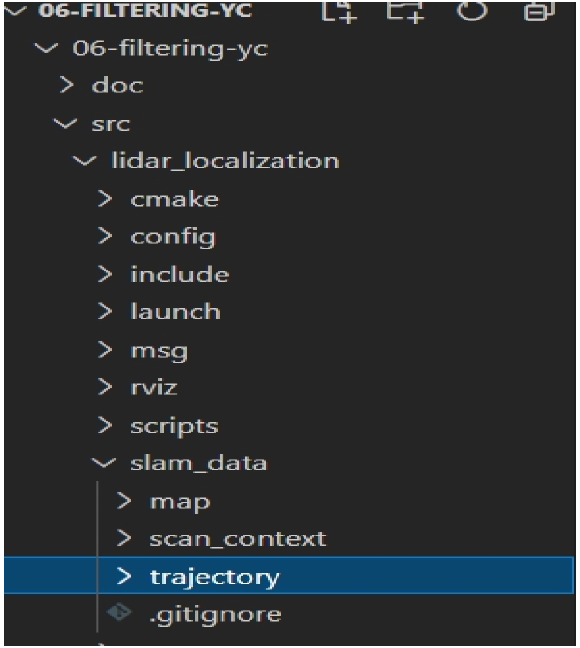
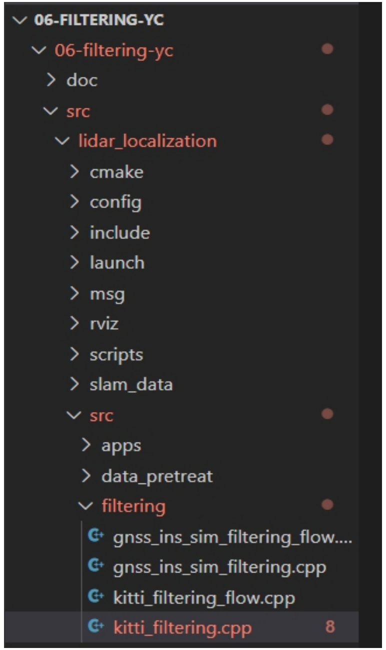


图1.6

框架调试

- 说明：调试方法仅适用于第一版框架.

修改kitti_filtering.cpp。文件目录如左图，修改后如右图。




```
// Kalman correction:
if (
    /*kalman_filter_ptr_->Correct(
        imu_data,
        KalmanFilter::MeasurementType::POSE_VEL, current_measurement_
    )*/
    kalman_filter_ptr_->Correct(
        imu_data,
        KalmanFilter::MeasurementType::POSE, current_measurement_
    )
) {
```

纲要

深蓝学院
shenlanxueyuan.com

- 第一部分：框架调试
- **第二部分：滤波实现**

滤波实现

深蓝学院
shenlanxueyuan.com

●.

基于滤波器的融合

$X = FX + Bw$

$y = Gx + Cn$

1. 状态方程

状态方程由误差方程得来，第6讲已经完成误差方程的推导：

$$\delta\dot{p} = \delta v$$

$$\delta\dot{v} = -R_t[a_t - b_{a_t}]_\times \delta\theta + R_t(n_a - \delta b_a)$$

$$\delta\dot{\theta} = -[\omega_t - b_{\omega_t}]_\times \delta\theta + n_\omega - \delta b_\omega$$

$$\delta\dot{b}_a = n_{b_a} \quad \text{或} \quad \delta\dot{b}_a = 0$$

$$\delta\dot{b}_\omega = n_{b_\omega} \qquad \delta\dot{b}_\omega = 0$$

令 $\delta x = \begin{bmatrix} \delta p \\ \delta v \\ \delta\theta \\ \delta b_a \\ \delta b_\omega \end{bmatrix}$，$W = \begin{bmatrix} n_a \\ n_\omega \\ n_{b_a} \\ n_{b_\omega} \end{bmatrix}$

则误差方程可以写成状态方程的通用形式：

$$\delta\dot{x} = F_t \delta x + B_t w$$

其中

$$F_t = \begin{bmatrix} 0 & I_3 & 0 & 0 & 0 \\ 0 & 0 & -R_t[\bar{a}_t]_\times & -R_t & 0 \\ 0 & 0 & -[\bar{\omega}_t]_\times & 0 & -I_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \bar{a}_t = a_t - b_{a_t}, \quad \bar{\omega}_t = \omega_t - b_{\omega_t}$$

$$B_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ R_t & 0 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \end{bmatrix}$$

注：当选择 $\delta\dot{b}_a = 0$，$\delta\dot{b}_\omega = 0$ 时，矩阵形式不一样，请各位自行推导。

预测状态方程的Ft矩阵需要加速度和旋转角速度信息。而第一期框架的解算函数输出了速度信息，我们可以添加旋转角速度信息，注意修改源文件后同时修改头文件和相关函数。

```cpp
void ErrorStateKalmanFilter::UpdateOdomEstimation(Eigen::Vector3d &linear_acc_mid,Eigen::Vector3d &mid_w) {
    //
    // TODO: this is one possible solution to previous chapter, IMU Navigation, assignment
    //
    static Eigen::Vector3d w_b = Eigen::Vector3d::Zero();

    // get deltas:
    Eigen::Vector3d angular_delta;
    GetAngularDelta(1, 0, angular_delta,mid_w);
```

修改后

```cpp
void ErrorStateKalmanFilter::UpdateOdomEstimation(Eigen::Vector3d &linear_acc_mid) {
    //
    // TODO: this is one possible solution to previous chapter, IMU Navigation, assignment
    //
    static Eigen::Vector3d w_b = Eigen::Vector3d::Zero();

    // get deltas:
    Eigen::Vector3d angular_delta;
    GetAngularDelta(1, 0, angular_delta,mid_w);
```

修改前

滤波实现

基于滤波器的融合

$X = FX + BW$

$Y = GX + Cn$

1. 状态方程

状态方程由误差方程得来，第6讲已经完成误差方程的推导：

$$\delta\dot{p} = \delta v$$

$$\delta\dot{v} = -R_t[a_t - b_{a_t}]_\times \delta\theta + R_t(n_a - \delta b_a)$$

$$\delta\dot{\theta} = -[\omega_t - b_{\omega_t}]_\times \delta\theta + n_\omega - \delta b_\omega$$

$$\delta\dot{b}_a = n_{b_a} \qquad \delta\dot{b}_a = 0$$

或

$$\delta\dot{b}_\omega = n_{b_\omega} \qquad \delta\dot{b}_\omega = 0$$

令 $\delta x = \begin{bmatrix} \delta p \\ \delta v \\ \delta\theta \\ \delta b_a \\ \delta b_\omega \end{bmatrix}$, $W = \begin{bmatrix} n_a \\ n_\omega \\ n_{b_a} \\ n_{b_\omega} \end{bmatrix}$

则误差方程可以写成状态方程的通用形式：

$$\delta\dot{x} = F_t \delta x + B_t w$$

其中

$$F_t = \begin{bmatrix} 0 & I_3 & 0 & 0 & 0 \\ 0 & 0 & -R_t[\bar{a}_t]_\times & -R_t & 0 \\ 0 & 0 & -[\bar{\omega}_t]_\times & 0 & -I_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{aligned} \bar{a}_t &= a_t - b_{a_t} \\ \bar{\omega}_t &= \omega_t - b_{\omega_t} \end{aligned}$$

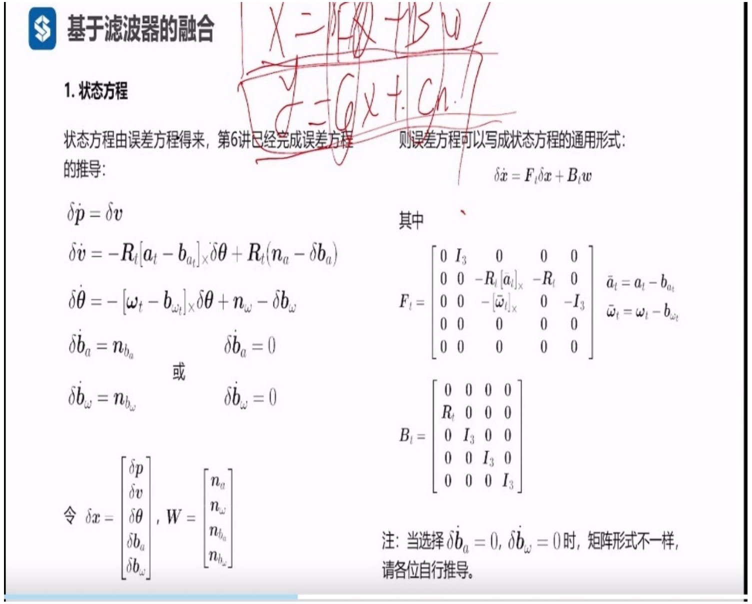
$$B_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ R_t & 0 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \end{bmatrix}$$

注：当选择 $\delta\dot{b}_a = 0$, $\delta\dot{b}_\omega = 0$ 时，矩阵形式不一样，请各位自行推导。

接下来将Ft的初始化。首先在ErrorStateKalmanFilter的构造函数里面初始Ft的单位矩阵。

```
// e. process equation:
//F_.block<3, 3>(  INDEX_ERROR_POS,   INDEX_ERROR_VEL) = Eigen::Matrix3d::Identity();
//F_.block<3, 3>(  INDEX_ERROR_ORI,   INDEX_ERROR_ORI) = Sophus::SO3d::hat(-w_).matrix
F_.block<3, 3>(  0,   INDEX_ERROR_POS) = Eigen::Matrix3d::Identity();
F_.block<3, 3>(  INDEX_ERROR_ORI,   INDEX_ERROR_ACCEL) = Eigen::Matrix3d::Identity();
```

滤波实现



基于滤波器的融合

$X = \Phi x + Bu$

$Y = Gx + Cn$

1. 状态方程

状态方程由误差方程得来，第6讲已经完成误差方程的推导：

$$\delta\dot{p} = \delta v$$

$$\delta\dot{v} = -R_t[a_t - b_{a_t}]_\times \delta\theta + R_t(n_a - \delta b_a)$$

$$\delta\dot{\theta} = -[\omega_t - b_{\omega_t}]_\times \delta\theta + n_\omega - \delta b_\omega$$

$$\delta\dot{b}_a = n_{b_a} \qquad \delta\dot{b}_a = 0$$

或

$$\delta\dot{b}_\omega = n_{b_\omega} \qquad \delta\dot{b}_\omega = 0$$

令 $\delta x = \begin{bmatrix} \delta p \\ \delta v \\ \delta\theta \\ \delta b_a \\ \delta b_\omega \end{bmatrix}$, $W = \begin{bmatrix} n_a \\ n_\omega \\ n_{b_a} \\ n_{b_\omega} \end{bmatrix}$

则误差方程可以写成状态方程的通用形式：

$$\delta\dot{x} = F_t \delta x + B_t w$$

其中

$$F_t = \begin{bmatrix} 0 & I_3 & 0 & 0 & 0 \\ 0 & 0 & -R_t[\bar{a}_t]_\times & -R_t & 0 \\ 0 & 0 & -[\bar{\omega}_t]_\times & 0 & -I_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{aligned} \bar{a}_t &= a_t - b_{a_t} \\ \bar{\omega}_t &= \omega_t - b_{\omega_t} \end{aligned}$$

$$B_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ R_t & 0 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \end{bmatrix}$$

注：当选择 $\delta\dot{b}_a = 0$, $\delta\dot{b}_\omega = 0$时，矩阵形式不一样，请各位自行推导。

然后在ErrorStateKalmanFilter的Update()成员函数里面使用修改后解算函数。并且修改滤波更新函数UpdateErrorEstimation()使其参数里面包含旋转角速度。

```
bool ErrorStateKalmanFilter::Update(const IMUData &imu_data) {
    // update IMU buff:
    if (time_ < imu_data.time) {
        // update IMU odometry:
        Eigen::Vector3d linear_acc_mid;
        Eigen::Vector3d mid_w;
        imu_data_buff_.push_back(imu_data);

        UpdateOdomEstimation(linear_acc_mid,mid_w);
        imu_data_buff_.pop_front();

        // update error estimation:
        double T = imu_data.time - time_;
        UpdateErrorEstimation(T, linear_acc_mid,mid_w);
```

滤波实现

基于滤波器的融合

X = F(X) + B(u)
y = G(x) + C(n)

1. 状态方程

状态方程由误差方程得来，第6讲已经完成误差方程的推导：

$$\delta\dot{p} = \delta v$$

$$\delta\dot{v} = -R_t[a_t - b_{a_t}]_\times \delta\theta + R_t(n_a - \delta b_a)$$

$$\delta\dot{\theta} = -[\omega_t - b_{\omega_t}]_\times \delta\theta + n_\omega - \delta b_\omega$$

$$\delta\dot{b}_a = n_{b_a} \qquad \delta\dot{b}_a = 0$$

或

$$\delta\dot{b}_\omega = n_{b_\omega} \qquad \delta\dot{b}_\omega = 0$$

令 $\delta x = \begin{bmatrix} \delta p \\ \delta v \\ \delta\theta \\ \delta b_a \\ \delta b_\omega \end{bmatrix}$, $W = \begin{bmatrix} n_a \\ n_\omega \\ n_{b_a} \\ n_{b_\omega} \end{bmatrix}$

则误差方程可以写成状态方程的通用形式：

$$\delta\dot{x} = F_t \delta x + B_t w$$

其中

$$F_t = \begin{bmatrix} 0 & I_3 & 0 & 0 & 0 \\ 0 & 0 & -R_t[\bar{a}_t]_\times & -R_t & 0 \\ 0 & 0 & -[\bar{\omega}_t]_\times & 0 & -I_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \bar{a}_t = a_t - b_{a_t}, \quad \bar{\omega}_t = \omega_t - b_{\omega_t}$$

$$B_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ R_t & 0 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \end{bmatrix}$$

注：当选择 $\delta\dot{b}_a = 0$, $\delta\dot{b}_\omega = 0$ 时，矩阵形式不一样，请各位自行推导。

接下来需要在SetProcessEquation函数里面对Ft矩阵的非单位矩阵，即第二行和第三行进行赋值和矩阵B。

```
F_.block<3, 3>(INDEX_ERROR_VEL,INDEX_ERROR_ORI) = -C_nb * Sophus::SO3d::hat(f_n).matrix();
F_.block<3, 3>(INDEX_ERROR_VEL,INDEX_ERROR_GYRO) = -C_nb;

B_.block<3, 3>(INDEX_ERROR_VEL, INDEX_ERROR_POS) = C_nb;
B_.block<3, 3>(INDEX_ERROR_ORI, INDEX_ERROR_VEL) = Eigen::Matrix3d::Identity();
B_.block<3, 3>(INDEX_ERROR_GYRO, INDEX_ERROR_ORI) = Eigen::Matrix3d::Identity();
B_.block<3, 3>(INDEX_ERROR_ACCEL, INDEX_ERROR_GYRO) = Eigen::Matrix3d::Identity();

// b. set process equation for delta ori:
F_.block<3, 3>(INDEX_ERROR_ORI, INDEX_ERROR_ORI) = -Sophus::SO3d::hat(mid_angular_velocity).matrix();
F_.block<3, 3>(INDEX_ERROR_ORI, INDEX_ERROR_ACCEL) = -Eigen::Matrix3d::Identity();
```

滤波实现

最后就只需要修改框架代码里面提示的fix_this部分，按照卡尔曼滤波公式直接补上去就ok。当不考虑Bias时，由于预测量减少，滤波更容易实现，推导公式也更简单，只需要在框架上稍加修改即可。相关噪音参数只需修改kitti_filtering文件。

```
//X_ = F * X_ + B * w ; // fix this
X_ = F * X_;
P_ = F * P_ * F.transpose() + B * Q_ * B.transpose() ; // fix this
```

```
Eigen::Matrix<double, 6, 6> Ct = Eigen::Matrix<double, 6, 6>::Identity();
MatrixRPose R = RPose_; // fix this
K =  P_ * G.transpose() * (G * P_ * G.transpose() + Ct * R * Ct.transpose()).inverse(); // fix this
```

```
P_ = (Eigen::Matrix<double, 15, 15>::Identity() - K * G) * P_; // fix this
X_ = X_ + K * (Y - G * X_); // fix this
```

```
pose_.block<3, 1>(0, 3) = pose_.block<3, 1>(0, 3) + X_.block<3,1>(0,0); // fix this
// b. velocity:
vel_ = vel_ + X_.block<3,1>(3,0);                                        // fix this
// c. orientation:
Eigen::Matrix3d delta_rotation = Sophus::SO3d::hat(X_.block<3, 1>(6, 0)).matrix();// fix this*/
pose_.block<3,3>(0,0) = pose_.block<3,3>(0,0) * (Eigen::Matrix3d::Identity() + delta_rotation);
```

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening