深蓝学院
shenlanxueyuan.com

多传感器融合定位
第七章作业思路

主讲人 陈梓杰

纲要

- 合格要求
- 良好要求
- 优秀要求

合格要求

深蓝学院
shenlanxueyuan.com

思路：

补全error_state_Kalman_filter.cpp的TODO部分

NOTE：

卡尔曼滤波器主要包括预测（update函数）和观测（Correct函数）两个部分：

1. 预测部分接收imu数据，基于惯性结算更新名义状态，更新协方差
2. 观测部分接收同步后的imu数据和定位数据，首先使用同步后的imu数据将状态积分到观测时刻，然后基于观测方程计算误差值，最后利用误差值对名义值进行修正

合格要求

深蓝学院
shenlanxueyuan.com

初始化init

由于代码框架是基于第一期课程，其旋转误差定义在导航系(n系)下；课程改版后，将旋转误差定义在机体系(b系)下，所以**状态方程中要使用b系下的加速度**。

即UpdateProcessEquation()函数传入的linear_acc_mid应该是在b系下的

```
// covert to navigation frame:
linear_acc_init = GetUnbiasedLinearAcc(linear_acc_init, C_nb);
angular_vel_init = GetUnbiasedAngularVel(angular_vel_init, C_nb);

// init process equation, in case of direct correct step:
UpdateProcessEquation(linear_acc_init, angular_vel_init);
```

修改前

```
// using unbiased acc in body frame for process equation
linear_acc_init = linear_acc_init - accl_bias_;
angular_vel_init = GetUnbiasedAngularVel(angular_vel_init, C_nb);

// init process equation, in case of direct correct step:
UpdateProcessEquation(linear_acc_init, angular_vel_init);
```

修改后

合格要求

深蓝学院
shenlanxueyuan.com

预测Update

(1)名义值更新UpdateOdomEstimation

(2)误差状态协方差更新UpdateErrorEstimation

```cpp
bool ErrorStateKalmanFilter::Update(const IMUData &imu_data) {
  //
  // TODO: understand ESKF update workflow
  //
  // update IMU buff:
  if (time_ < imu_data.time) {
    // update IMU odometry:
    Eigen::Vector3d linear_acc_mid = Eigen::Vector3d::Zero();
    Eigen::Vector3d angular_vel_mid = Eigen::Vector3d::Zero();
    imu_data_buff_.push_back(imu_data);
    UpdateOdomEstimation(linear_acc_mid, angular_vel_mid);
    imu_data_buff_.pop_front();

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

    // move forward:
    time_ = imu_data.time;

    return true;
  }

  return false;
}
```

合格要求

深蓝学院 shenlanxueyuan.com

预测Update

(1)名义值更新UpdateOdomEstimation

步骤与上一章惯性导航结算相同

注意：

GetVelocityDelta函数要返回的

linear_acc_mid应该是在b系下

```
// return mid-value acc in body frame for process equation
linear_acc_mid = 0.5*(linear_acc_curr + linear_acc_prev) - accl_bias_;
```

```
void ErrorStateKalmanFilter::UpdateOdomEstimation(
    Eigen::Vector3d &linear_acc_mid, Eigen::Vector3d &angular_vel_mid) {
  //
  // TODO: this is one possible solution to previous chapter, IMU Navigation,
  // assignment
  //
  // get deltas:
  Eigen::Vector3d angluar_delta;
  GetAngularDelta(1, 0, angluar_delta, angular_vel_mid);

  // update orientation:
  Eigen::Matrix3d R_curr, R_prev;
  UpdateOrientation(angluar_delta, R_curr, R_prev);

  // get velocity delta:
  double T = 0.0;
  Eigen::Vector3d velocity_delta;
  GetVelocityDelta(1, 0, R_curr, R_prev, T, velocity_delta, linear_acc_mid);

  // save mid-value unbiased linear acc for error-state update:

  // update position:
  UpdatePosition(T, velocity_delta);
}
```

合格要求

深蓝学院 shenlanxueyuan.com

预测Update

(2)误差状态协方差更新UpdateErrorEstimation

$$\boldsymbol{F}_t=\begin{bmatrix}0 & \boldsymbol{I}_3 & 0 & 0 & 0\\ 0 & 0 & -\boldsymbol{R}_t[\bar{\boldsymbol{a}}_t]_\times & -\boldsymbol{R}_t & 0\\ 0 & 0 & -[\bar{\boldsymbol{\omega}}_t]_\times & 0 & -\boldsymbol{I}_3\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix}\quad \begin{aligned}\bar{\boldsymbol{a}}_t &= \boldsymbol{a}_t-\boldsymbol{b}_{a_t}\\ \bar{\boldsymbol{\omega}}_t &= \boldsymbol{\omega}_t-\boldsymbol{b}_{\omega_t}\end{aligned}$$

其中

$$\delta\boldsymbol{x}_k=\boldsymbol{F}_{k-1}\delta\boldsymbol{x}_{k-1}+\boldsymbol{B}_{k-1}\boldsymbol{w}_k$$

$$\boldsymbol{F}_{k-1}=\boldsymbol{I}_{15}+\boldsymbol{F}_tT$$

$$\boldsymbol{B}_t=\begin{bmatrix}0 & 0 & 0 & 0\\ \boldsymbol{R}_t & 0 & 0 & 0\\ 0 & \boldsymbol{I}_3 & 0 & 0\\ 0 & 0 & \boldsymbol{I}_3 & 0\\ 0 & 0 & 0 & \boldsymbol{I}_3\end{bmatrix}\quad \boldsymbol{B}_{k-1}=\begin{bmatrix}0 & 0 & 0 & 0\\ \boldsymbol{R}_{k-1}T & 0 & 0 & 0\\ 0 & \boldsymbol{I}_3T & 0 & 0\\ 0 & 0 & \boldsymbol{I}_3\sqrt{T} & 0\\ 0 & 0 & 0 & \boldsymbol{I}_3\sqrt{T}\end{bmatrix}$$

其中，T 为 Kalman 的滤波周期。

误差状态预测公式不用实现，k-1时刻误差量已经清零，噪声w的均值为零

$$\boxed{\delta\check{\boldsymbol{x}}_k=\boldsymbol{F}_{k-1}\delta\hat{\boldsymbol{x}}_{k-1}+\boldsymbol{B}_{k-1}\boldsymbol{w}_k}$$

$$\check{\boldsymbol{P}}_k=\boldsymbol{F}_{k-1}\hat{\boldsymbol{P}}_{k-1}\boldsymbol{F}_{k-1}^{\mathrm{T}}+\boldsymbol{B}_{k-1}\boldsymbol{Q}_k\boldsymbol{B}_{k-1}^T$$

```cpp
void ErrorStateKalmanFilter::UpdateErrorEstimation(
    const double &T, const Eigen::Vector3d &linear_acc_mid,
    const Eigen::Vector3d &angular_vel_mid) {
  static MatrixF F_1st;
  static MatrixF F_2nd;
  // TODO: update process equation:
  UpdateProcessEquation(linear_acc_mid, angular_vel_mid);

  // TODO: get discretized process equations:
  MatrixF F = MatrixF::Identity() + F_*T;
  MatrixB B = B_*T;
  if(correct_bias_){
   B.block<6,6>(kIndexErrorAccel, kIndexNoiseBiasAccel) = Eigen::Matrix<double, 6, 6>::Identity() * sqrt(T);
  }

  // TODO: perform Kalman prediction
  P_ = F*P_*F.transpose()+B*Q_*B.transpose();
}
```

```cpp
void ErrorStateKalmanFilter::SetProcessEquation(const Eigen::Matrix3d &C_nb,
                                                const Eigen::Vector3d &f_n,
                                                const Eigen::Vector3d &w_b) {
  // TODO: set process / system equation:
  // a. set process equation for delta vel:
  F_.block<3,3>(kIndexErrorVel, kIndexErrorOri) = -C_nb * Sophus::SO3d::hat(f_n).matrix();
  F_.block<3,3>(kIndexErrorVel, kIndexErrorAccel) = -C_nb;

  // b. set process equation for delta ori:
  F_.block<3,3>(kIndexErrorOri, kIndexErrorOri) = -Sophus::SO3d::hat(w_b).matrix();

  B_.block<3,3>(kIndexErrorVel, kIndexNoiseAccel) = C_nb;
}
```

合格要求

修正Correct

(1)计算误差CorrectErrorEstimation

(2)修正名义值EliminateError

(3)误差值清零ResetState

```cpp
bool ErrorStateKalmanFilter::Correct(const IMUData &imu_data,
                                     const MeasurementType &measurement_type,
                                     const Measurement &measurement) {
  static Measurement measurement_;

  // get time delta:
  double time_delta = measurement.time - time_;

  if (time_delta > -0.05) {
    // perform Kalman prediction:
    // 使用同步的imu数据做一次预测 使得kalman时间与观测时间相同
    if (time_ < measurement.time) {
      Update(imu_data);
    }

    // get observation in navigation frame:
    measurement_ = measurement;
    // measurement_.T_nb = init_pose_ * measurement_.T_nb;
    // measurement_.T_nb = measurement_.T_nb;

    // correct error estimation:
    CorrectErrorEstimation(measurement_type, measurement_);

    // eliminate error:
    EliminateError();

    // reset error state:
    ResetState();

    return true;
  }
```

合格要求

深蓝学院 shenlanxueyuan.com

修正Correct

(1)计算误差CorrectErrorEstimation

参考ppt补全CorrectErrorEstimationPose()

观测量中，$\delta \boldsymbol{p}$ 的计算过程为：

$$\delta \bar{\boldsymbol{p}} = \check{\boldsymbol{p}} - \boldsymbol{p}$$

其中 $\check{\boldsymbol{p}}$ 为 IMU 解算的位置，即预测值。$\boldsymbol{p}$ 为雷达与地图匹配得到的位置，即观测值。

$\delta\boldsymbol{\theta}$ 的计算过程稍微复杂，需要先计算误差矩阵，

$$\delta \bar{\boldsymbol{R}}_t = \boldsymbol{R}_t^T \check{\boldsymbol{R}}_t$$

其中 $\check{\boldsymbol{R}}_t$为 IMU 解算的旋转矩阵，即预测值。$\boldsymbol{R}_t$为雷达与地图匹配得到的旋转矩阵，即观测值。

由于预测值与观测值之间的关系为

$$\check{\boldsymbol{R}}_t \approx \boldsymbol{R}_t(\boldsymbol{I} + [\delta\boldsymbol{\theta}]_\times)$$

因此

$$\delta\boldsymbol{\theta} = (\delta\bar{\boldsymbol{R}}_t - \boldsymbol{I})^\vee$$

$$\boldsymbol{G}_t = \begin{bmatrix} \boldsymbol{I}_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{I}_3 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{K}_k = \check{\boldsymbol{P}}_k \boldsymbol{G}_k^{\mathrm{T}} \left(\boldsymbol{G}_k \check{\boldsymbol{P}}_k \boldsymbol{G}_k^{\mathrm{T}} + \boldsymbol{C}_k \boldsymbol{R}_k \boldsymbol{C}_k^{T} \right)^{-1}$$

$$\boldsymbol{C}_t = \begin{bmatrix} \boldsymbol{I}_3 & 0 \\ 0 & \boldsymbol{I}_3 \end{bmatrix}$$

```cpp
void ErrorStateKalmanFilter::CorrectErrorEstimationPose(
    const Eigen::Matrix4d &T_nb, Eigen::VectorXd &Y, Eigen::MatrixXd &G,
    Eigen::MatrixXd &K) {
  //
  // TODO: set measurement:
  //
  Eigen::Vector3d delta_p = pose_.block<3,1>(0,3) - T_nb.block<3,1>(0,3);
  Eigen::Matrix3d delta_R = T_nb.block<3,3>(0,0).transpose() * pose_.block<3,3>(0,0);
  Eigen::Vector3d delta_theta = Sophus::SO3d::vee(delta_R-Eigen::Matrix3d::Identity());

  // TODO: set measurement equation:
  YPose_.block<3,1>(0,0) = delta_p;
  YPose_.block<3,1>(3,0) = delta_theta;
  Y = YPose_;
  G = GPose_;

  // TODO: set Kalman gain:
  // 这里没必要C*R*C^T 因为C是一个单位阵
  K = P_*G.transpose()*(G*P_*G.transpose() + RPose_).inverse();
}
```

合格要求

深蓝学院
shenlanxueyuan.com

修正Correct

(2)修正名义值EliminateError

(3)误差值清零ResetState

```
void ErrorStateKalmanFilter::ResetState(void) {
    // reset current state:
    X_ = VectorX::Zero();
}
```

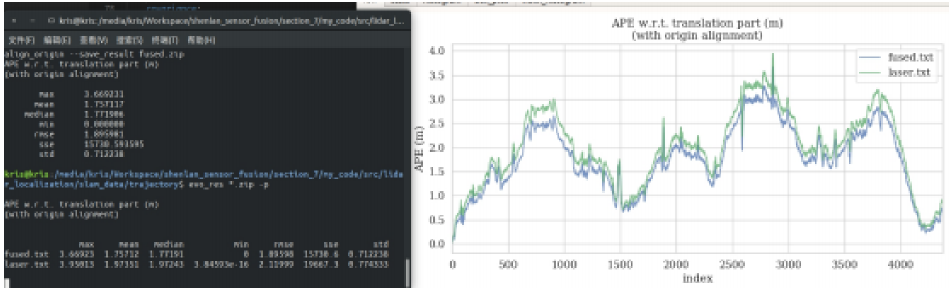
```
void ErrorStateKalmanFilter::EliminateError(void) {
    //
    // TODO: correct state estimation using the state of ESKF
    //
    // a. position:
    // do it!
    pose_.block<3,1>(0,3) = pose_.block<3,1>(0,3) - X_.block<3,1>(kIndexErrorPos,0);
    // b. velocity:
    // do it!
    vel_ = vel_ - X_.block<3,1>(kIndexErrorVel,0);
    // c. orientation:
    // do it!
    Eigen::Matrix3d delta_R = Eigen::Matrix3d::Identity() - Sophus::SO3d::hat(X_.block<3,1>(kIndexErrorOri,0)).matrix();
    Eigen::Quaterniond dq = Eigen::Quaterniond(delta_R);
    dq = dq.normalized();
    // pose_.block<3,3>(0,0) *= dq.toRotationMatrix();
    pose_.block<3,3>(0,0) = pose_.block<3,3>(0,0) * dq.toRotationMatrix();

    gyro_bias_ -= X_.block<3, 1>(kIndexErrorGyro, 0);
    accl_bias_ -= X_.block<3, 1>(kIndexErrorAccel, 0);
```

良好要求

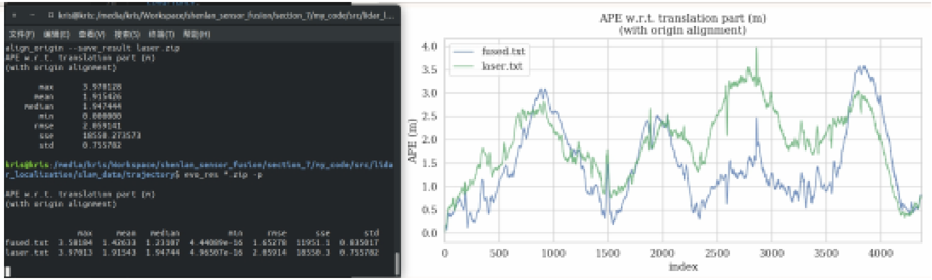
目标：ESKF在不同噪声设置情况下的结果对比

思路：

- 卡尔曼滤波器调参方法近似可以按Q/R的大小来调
- 当Q/R越大，表示Q越大，预测的噪声越大，系统更相信观测
- 当Q/R越小，表示R越大，观测的噪声越大，系统更相信预测



Q减小，R增大，系统相信观测
融合ape曲线与激光ape曲线**相似**



Q增大，R减小，系统相信预测
融合ape曲线与激光ape曲线**不相似**

优秀要求

不考虑随机游走的公式推导结果如下：

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

$$\delta\dot{\boldsymbol{v}} = -\boldsymbol{R}_t[\boldsymbol{a}_t - \boldsymbol{b}_{a_t}]_\times\delta\boldsymbol{\theta} + \boldsymbol{R}_t(\boldsymbol{n}_a - \delta\boldsymbol{b}_a)$$

$$\delta\dot{\boldsymbol{\theta}} = -[\boldsymbol{\omega}_t - \boldsymbol{b}_{\omega_t}]_\times\delta\boldsymbol{\theta} + \boldsymbol{n}_\omega - \delta\boldsymbol{b}_\omega$$

$$\delta\dot{\boldsymbol{b}}_a = 0$$

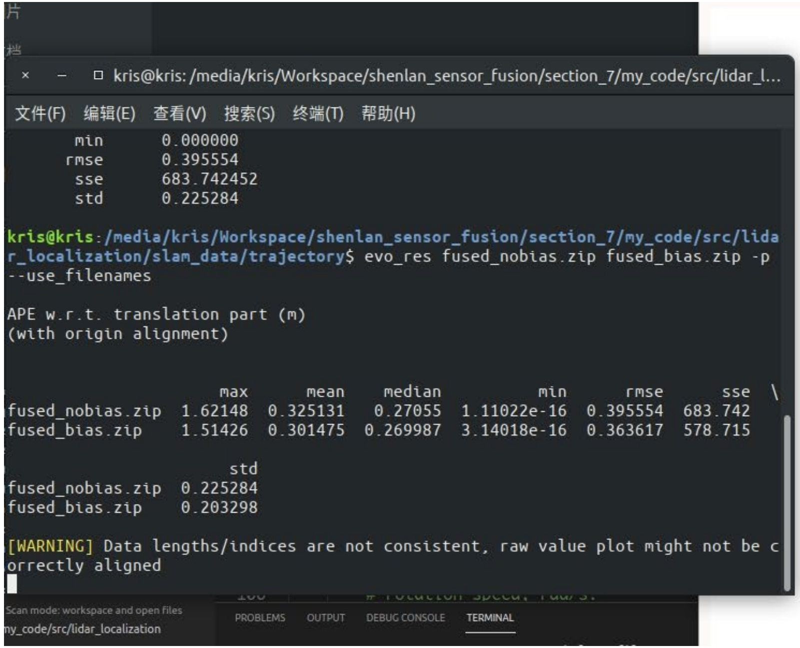
$$\delta\dot{\boldsymbol{b}}_\omega = 0$$

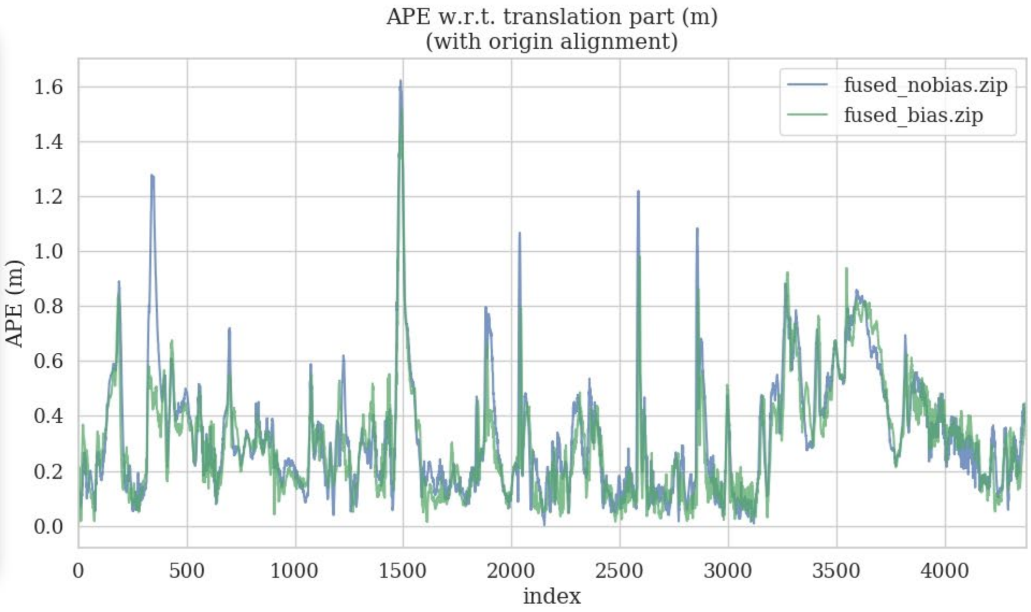
$$B_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ R_t & 0 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B_{k-1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ R_{k-1}T & 0 & 0 & 0 \\ 0 & I_3T & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

注意：不考虑随机游走，bias仍是有更新值的

优秀要求

实验结果：

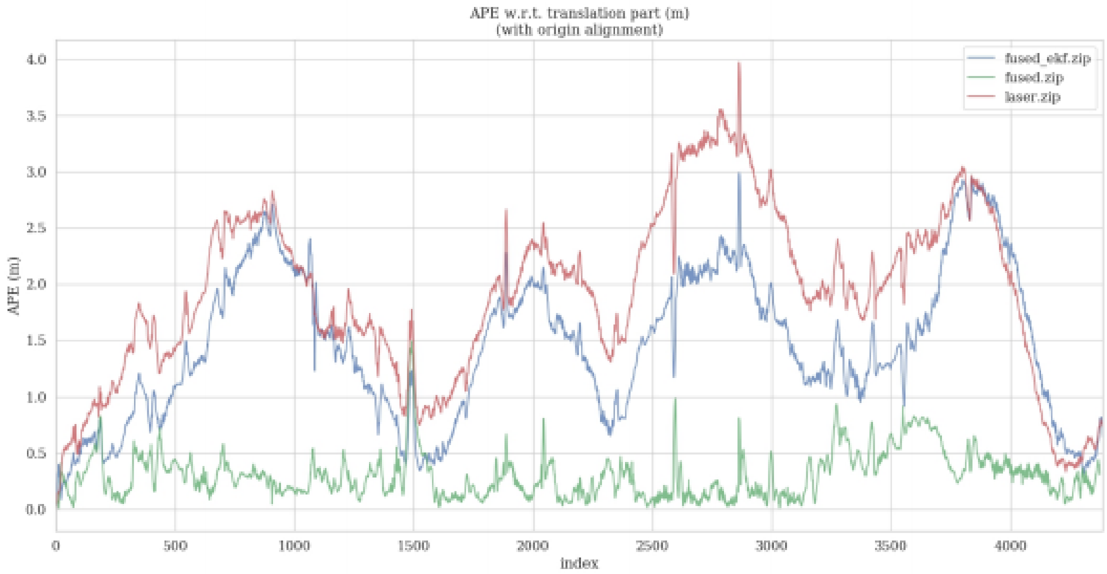

kris@kris: /media/kris/Workspace/shenlan_sensor_fusion/section_7/my_code/src/lidar_l...
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
min 0.000000
rmse 0.395554
sse 683.742452
std 0.225284
kris@kris:/media/kris/Workspace/shenlan_sensor_fusion/section_7/my_code/src/lidar_localization/slam_data/trajectory$ evo_res fused_nobias.zip fused_bias.zip -p --use_filenames
APE w.r.t. translation part (m)
(with origin alignment)
max mean median min rmse sse \
fused_nobias.zip 1.62148 0.325131 0.27055 1.11022e-16 0.395554 683.742
fused_bias.zip 1.51426 0.301475 0.269987 3.14018e-16 0.363617 578.715
std
fused_nobias.zip 0.225284
fused_bias.zip 0.203298
[WARNING] Data lengths/indices are not consistent, raw value plot might not be correctly aligned
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL


APE w.r.t. translation part (m)
(with origin alignment)
fused_nobias.zip
fused_bias.zip
APE (m)
index
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
500
1000
1500
2000
2500
3000
3500
4000

附加题

目标：推导基于名义状态的EKF，并在代码中实现

结论：基于名义状态的EKF对噪声更敏感。同时状态量采用的是四元数，观测更新采用四元数直接相加（四元数不满足加法），所以导致姿态数值不稳定，调参调得不好姿态就会飞



在线问答

深蓝学院
shenlanxueyuan.com

Q&A

感谢各位聆听！

Thanks for Listening