深蓝学院
shenlanxueyuan.com

第七次作业思路分享

主讲人 张松鹏

纲要

- 第一题
- 第三题

第一题：误差状态滤波

- 任务：补全以下文件中的TODO部分

lidar_localization/src/model/kalman_filter/error_state_kalman_filter.cpp

- 代码结构

滤波算法主要包括预测（Update函数）和观测（Correct函数）两个部分：

1) 预测部分接收imu数据，基于惯性解算更新名义值，基于状态方程更新误差值。
2) 观测部分同时接收imu数据和定位数据，首先利用imu数据进行预测保证状态与定位数据时间同步，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，并将误差值清零。

第一题：误差状态滤波

●初始化：Init

这里特别注意，框架是基于第一期课程，其中的旋转误差放在了导航系（n系）下，而第三期将旋转误差放在了机器人坐标系（b系）下，所以公式有所不同，特别是状态方程所用到的加速度应该是在b系下，也就是UpdateProcessEquation函数传入的linear_acc_mid应该是在b系下。所有调用到这个函数的地方都应该进行修改。

Init函数中传入的linear_acc_init修改如下：

```
// covert to navigation frame:
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);
```

第一题：误差状态滤波

●预测：Update

主流程如下：

包括更新名义值UpdateOdomEstimation和误差值UpdateErrorEstimation两部分

```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 angular_vel_mid;
    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;
}
```

第一题：误差状态滤波

- **预测：Update**

1) 更新名义值
UpdateOdomEstimation

这部分是上一章惯性导航解算的内容，如右所示。

```
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 angular_delta;
  GetAngularDelta(1, 0, angular_delta, angular_vel_mid);
  // update orientation:
  Eigen::Matrix3d R_curr, R_prev;
  UpdateOrientation(angular_delta, R_curr, R_prev);
  // get velocity delta:
  double T;
  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);
}
```

这里需要注意，GetVelocityDelta函数中的linear_acc_mid应该返回在b系下的测量值，如下

```
linear_acc_mid = (linear_acc_prev + linear_acc_curr) / 2 - accl_bias_;
```

第一题：误差状态滤波

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

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

●预测：Update

2) 更新误差值UpdateErrorEstimation

首先调用UpdateProcessEquation计算状态方程中的F和B，该函数进一步调用SetProcessEquation函数，对应课件中的公式如右上角所示。

```
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);
```

```
void ErrorStateKalmanFilter::UpdateProcessEquation(
    const Eigen::Vector3d &linear_acc_mid,
    const Eigen::Vector3d &angular_vel_mid) {
  // set linearization point:
  Eigen::Matrix3d C_nb = pose_.block<3, 3>(0, 0);

  // set process equation:
  SetProcessEquation(C_nb, linear_acc_mid, angular_vel_mid);
}
```

```
void ErrorStateKalmanFilter::SetProcessEquation(const Eigen::Matrix3d &C_nb,
                                                const Eigen::Vector3d &f_b,
                                                const Eigen::Vector3d &w_b) {
  // TODO: set process / system equation:
  // a. set process equation for delta vel:
  F_.setZero();
  F_.block<3, 3>(kIndexErrorPos, kIndexErrorVel) = Eigen::Matrix3d::Identity();
  F_.block<3, 3>(kIndexErrorVel, kIndexErrorOri) = -C_nb * Sophus::SO3d::hat(f_b).matrix();
  F_.block<3, 3>(kIndexErrorVel, kIndexErrorAccel) = -C_nb;
  F_.block<3, 3>(kIndexErrorOri, kIndexErrorOri) = -Sophus::SO3d::hat(w_b).matrix();
  F_.block<3, 3>(kIndexErrorOri, kIndexErrorGyro) = -Eigen::Matrix3d::Identity();
  // b. set process equation for delta ori:
  B_.setZero();
  B_.block<3, 3>(kIndexErrorVel, kIndexNoiseAccel) = C_nb;
  B_.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) = Eigen::Matrix3d::Identity();
  B_.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = Eigen::Matrix3d::Identity();
  B_.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = Eigen::Matrix3d::Identity();
}
```

第一题：误差状态滤波

深蓝学院 shenlanxueyuan.com

● 预测：Update

2) 更新误差值UpdateErrorEstimation

然后按照采样时间进行离散化，公式如下

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

其中

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

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

其中，T 为 Kalman 的滤波周期。

```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:
  F_1st = T * F_;
  MatrixF F = MatrixF::Identity() + F_1st;
  MatrixB B = MatrixB::Zero();
  B.block<3, 3>(kIndexErrorVel, kIndexNoiseAccel) = T * B_.block<3, 3>(kIndexErrorVel,
  kIndexNoiseAccel);
  B.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) = T * B_.block<3, 3>(kIndexErrorOri,
  kIndexNoiseGyro);
  B.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = std::sqrt(T) * B_.block<3, 3>
  (kIndexErrorAccel, kIndexNoiseBiasAccel);
  B.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = std::sqrt(T) * B_.block<3, 3>
  (kIndexErrorGyro, kIndexNoiseBiasGyro);
  // TODO: perform Kalman prediction
  // X_ = F * X_ + B * n;
  P_ = F * P_ * F.transpose() + B * Q_ * B.transpose();
}
```

最后计算滤波的前2个公式（第1个公式均是0在传递，只有方差进行了计算）

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

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

第一题：误差状态滤波

深蓝学院
shenlanxueyuan.com

●观测：Correct

主流程如下：

包括预测Update，计算误差值CorrectErrorEstimation，修正名义值EliminateError，清零误差值ResetState共四部分。预测已经介绍过，下面介绍剩余三个部分。

```
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:
    if (time_ < measurement.time) {
      Update(imu_data);
    }

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

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

    // eliminate error:
    EliminateError();

    // reset error state:
    ResetState();

    return true;
  }
```

第一题：误差状态滤波

● 观测：Correct

1) 计算误差值CorrectErrorEstimation

首先调用CorrectErrorEstimationPose计算Y，G，K

```
void ErrorStateKalmanFilter::CorrectErrorEstimation(
    const MeasurementType &measurement_type, const Measurement &measurement) {
  //
  // TODO: understand ESKF correct workflow
  //
  Eigen::VectorXd Y;
  Eigen::MatrixXd G, K;
  switch (measurement_type) {
  case MeasurementType::POSE:
    CorrectErrorEstimationPose(measurement.T_nb, Y, G, K);
    break;
  default:
    break;
  }
```

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。

$\delta \bar{\boldsymbol{\theta}}$ 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

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

因此

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

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

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

计算滤波第3个公式

$$\boldsymbol{K}_k = \check{\boldsymbol{P}}_k \boldsymbol{G}_k^{\mathrm{T}} \left(\boldsymbol{G}_k \check{\boldsymbol{P}}_k \boldsymbol{G}_k^{\mathrm{T}} + \boldsymbol{C}_k \boldsymbol{R}_k \boldsymbol{C}_k^{T} \right)^{-1}$$

```
void ErrorStateKalmanFilter::CorrectErrorEstimationPose(
    const Eigen::Matrix4d &T_nb, Eigen::VectorXd &Y, Eigen::MatrixXd &G,
    Eigen::MatrixXd &K) {
  //
  // TODO: set measurement:
  //
  Eigen::Vector3d dp = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
  Eigen::Matrix3d dR = T_nb.block<3, 3>(0, 0).transpose() * pose_.block<3, 3>(0, 0);
  Eigen::Vector3d dtheta = Sophus::SO3d::vee(dR - Eigen::Matrix3d::Identity());
  YPose_.block<3, 1>(0, 0) = dp;
  YPose_.block<3, 1>(3, 0) = dtheta;
  Y = YPose_;
  // TODO: set measurement equation:
  GPose_.setZero();
  GPose_.block<3, 3>(0, kIndexErrorPos) = Eigen::Matrix3d::Identity();
  GPose_.block<3, 3>(3, kIndexErrorOri) = Eigen::Matrix3d::Identity();
  G = GPose_;
  CPose_.setZero();
  CPose_.block<3, 3>(0, 0) = Eigen::Matrix3d::Identity();
  CPose_.block<3, 3>(3, 3) = Eigen::Matrix3d::Identity();
  // TODO: set Kalman gain:
  K = P_ * G.transpose() * (G * P_ * G.transpose() + CPose_ * RPose_ * CPose_.transpose()).
  inverse();
}
```

第一题：误差状态滤波

● 观测：Correct

1) 计算误差值CorrectErrorEstimation

然后计算滤波的第4，5个公式

$$\hat{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{G}_k\right) \check{\boldsymbol{P}}_k$$

$$\delta \hat{\boldsymbol{x}}_k = \delta \check{\boldsymbol{x}}_k + \boldsymbol{K}_k \left(\boldsymbol{y}_k - \boldsymbol{G}_k \delta \check{\boldsymbol{x}}_k\right)$$

```
void ErrorStateKalmanFilter::CorrectErrorEstimation(
    const MeasurementType &measurement_type, const Measurement &measurement) {
  //
  // TODO: understand ESKF correct workflow
  //
  Eigen::VectorXd Y;
  Eigen::MatrixXd G, K;
  switch (measurement_type) {
  case MeasurementType::POSE:
    CorrectErrorEstimationPose(measurement.T_nb, Y, G, K);
    break;
  default:
    break;
  }

  // TODO: perform Kalman correct:
  P_ = (MatrixP::Identity() - K * G) * P_;
  X_ = X_ + K * (Y - G * X_);
}
```

第一题：误差状态滤波

● 观测：Correct

2) 修正名义值EliminateError

修正平移，速度，旋转和零偏，修正方式是预测值减去误差值，注意框架中的零偏加号改成减号。

3) 清零误差值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) -= X_.block<3, 1>(kIndexErrorPos, 0);
  // b. velocity:
  // do it!
  vel_ -= X_.block<3, 1>(kIndexErrorVel, 0);
  // c. orientation:
  // do it!
  Eigen::Matrix3d dtheta_cross = Sophus::SO3d::hat(X_.block<3, 1>(kIndexErrorOri, 0));
  pose_.block<3, 3>(0, 0) = pose_.block<3, 3>(0, 0) *  (Eigen::Matrix3d::Identity() -
  dtheta_cross);
  Eigen::Quaterniond q_tmp(pose_.block<3, 3>(0, 0));
  q_tmp.normalize();
  pose_.block<3, 3>(0, 0) = q_tmp.toRotationMatrix();
  // d. gyro bias:
    gyro_bias_ -= X_.block<3, 1>(kIndexErrorGyro, 0);
  // if (IsCovStable(kIndexErrorGyro)) {
  // }

  // e. accel bias:
    accl_bias_ -= X_.block<3, 1>(kIndexErrorAccel, 0);
  // if (IsCovStable(kIndexErrorAccel)) {
  // }
}
```

纲要

- 第一题
- 第三题

第三题：不考虑随机游走模型

●公式

不考虑随机游走的公式如下

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

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

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

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

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

比较便捷的方法是直接把B和Q的最后2行2列矩阵块置为0，注意bias仍然是需要更新的。

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

第三题：不考虑随机游走模型

●代码

在构造函数中将噪声项Q相关部分设置为0

```
Q_.block<3, 3>(kIndexNoiseBiasAccel, kIndexNoiseBiasAccel) = Eigen::Matrix3d::Zero();
Q_.block<3, 3>(kIndexNoiseBiasGyro, kIndexNoiseBiasGyro) = Eigen::Matrix3d::Zero();
```

在SetProcessEquation函数和UpdateErrorEstimation函数中将B相关矩阵块置0。

```
// b. set process equation for delta ori:
B_.setZero();
B_.block<3, 3>(kIndexErrorVel, kIndexNoiseAccel) = C_nb;
B_.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) = Eigen::Matrix3d::Identity();
// B_.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = Eigen::Matrix3d::Identity();
// B_.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = Eigen::Matrix3d::Identity();
```

```
MatrixB B = MatrixB::Zero();
B.block<3, 3>(kIndexErrorVel, kIndexNoiseAccel) = T * B_.block<3, 3>(kIndexErrorVel,
kIndexNoiseAccel);
B.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) = T * B_.block<3, 3>(kIndexErrorOri,
kIndexNoiseGyro);
// B.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = std::sqrt(T) * B_.block<3, 3>
(kIndexErrorAccel, kIndexNoiseBiasAccel);
// B.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = std::sqrt(T) * B_.block<3, 3>
(kIndexErrorGyro, kIndexNoiseBiasGyro);
```

在线问答

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening