多传感器融合定位

——第七章作业分享

主讲人 胡存蔚

纲要

- 1、补全代码，实现基于地图的融合定位
- 2、调试参数，与不加滤波时的定位结果做比较
- 3、给出不考虑随机游走模型时的推导过程，并在工程中实现。对比两种方法的性能差异

纲要

- 1、补全代码，实现基于地图的融合定位
- 2、调试参数，与不加滤波时的定位结果做比较
- 3、给出不考虑随机游走模型时的推导过程，并在工程中实现。对比两种方法的性能差异

1、补全代码，实现基于地图的融合定位

1.1 ESKF初始化

- 课程中的姿态误差$\delta\theta$是定义在载体坐标系（b系）下的，状态方程中的加速度也是定义在b系下；

- 修改ESKF初始化函数Init，更新状态方程中的F阵和B阵；

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

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

```
Eigen::Vector3d linear_acc_init(imu_data.linear_acceleration.x,
                                imu_data.linear_acceleration.y,
                                imu_data.linear_acceleration.z);
Eigen::Vector3d angular_vel_init(imu_data.angular_velocity.x,
                                 imu_data.angular_velocity.y,
                                 imu_data.angular_velocity.z);
// covert to navigation frame:
// linear_acc_init = GetUnbiasedLinearAcc(linear_acc_init, C_nb);

// use 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);
```

1、补全代码，实现基于地图的融合定位

深蓝学院
shenlanxueyuan.com

1.2 ESKF预测

1.2.1 名义值更新

补全UpdateOdomEstimation，使用同步后的imu数据将名义状态值积分到观测时刻。

注意： GetAngularDelta输出的加速度是在b系下的

```cpp
/**
 * @brief  update IMU odometry estimation
 * @param  linear_acc_mid, output mid-value unbiased linear acc
 * @return void
 */
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:
  const size_t index_curr = 1;
  const size_t index_prev = 0;
  Eigen::Vector3d angular_delta;
  GetAngularDelta(index_curr, index_prev, angular_delta, angular_vel_mid);
  // update orientation:
  Eigen::Matrix3d R_cur, R_pre;
  UpdateOrientation(angular_delta, R_cur, R_pre);
  // get velocity delta:
  double dt;
  Eigen::Vector3d velocity_delta;
  GetVelocityDelta(index_curr, index_prev, R_cur, R_pre, dt, velocity_delta, linear_acc_mid);
  // save mid-value unbiased linear acc for error-state update:

  // update position:
  UpdatePosition(dt, velocity_delta);

}
```

1、补全代码，实现基于地图的融合定位

深蓝学院
shenlanxueyuan.com

1.2.2 状态方程更新

- 修改状态方程更新函数UpdateProcessEquation，输入的是b系下的加速度，所以重力加速度g_也要转换到b系下。

- 根据课程中的公式，补全**SetProcessEquation**

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

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

```cpp
/**
 * @brief  update process equation
 * @param  imu_data, input IMU measurement
 * @param  T, output time delta
 * @return void
 */
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);
  Eigen::Vector3d f_n = linear_acc_mid + C_nb.transpose() * g_; // trans g_ to body frame

  Eigen::Vector3d w_b = angular_vel_mid;

  // set process equation:
  SetProcessEquation(C_nb, f_n, w_b);
}
```

```cpp
/**
 * @brief  set process equation
 * @param  C_nb, rotation matrix, body frame -> navigation frame
 * @param  f_n, accel measurement in navigation frame
 * @return void
 */
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:
  //第七章ppt， p30
  F_.block<3, 3>(kIndexErrorVel, kIndexErrorOri) = -C_nb * ((Sophus::SO3d::hat(f_n)).matrix());
  F_.block<3, 3>(kIndexErrorVel, kIndexErrorAccel) = -C_nb;

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

}
```

1、补全代码，实现基于地图的融合定位

1.2.3 预测误差状态协方差

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

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

修改函数
UpdateErrorEstimation
.这里主要是根据课程中的公式，更新离散时间下的状态方程中的F阵和B阵，并预测状态误差协方差阵P

状态方程离散化，可以写为

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

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
/**
 * @brief  update error estimation
 * @param  linear_acc_mid, input mid-value unbiased linear acc
 * @return void
 */
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.setZero();
  B.block<3, 3>(kIndexErrorVel, kIndexNoiseAccel) = B_.block<3, 3>(kIndexErrorVel, kIndexNoiseAccel) * T;
  B.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) = B_.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) * T;

  B.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = B_.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) * sqrt(T);
  B.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = B_.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) * sqrt(T);

  // TODO: perform Kalman prediction
  X_ = F * X_; // 这行可以不用，因为x_会被清零
  //Pk = Fk_1*P*Fk_1_T + Bk_1 * Q * Bk_1_T
  P_ = F * P_ * F.transpose() + B * Q_ * B.transpose();

}
```

1、补全代码，实现基于地图的融合定位

深蓝学院
shenlanxueyuan.com

1.3 ESKF更新

当有lidar点云匹配定位数据时，进行ESKF的更新步骤，ESKF更新在Correct函数中进行。需要补全函数CorrectErrorEstimation、CorrectErrorEstimationPose、EliminateError

```
/**
 * @brief  Kalman correction, pose measurement and other measurement in body
 * frame
 * @param  measurement_type, input measurement type
 * @param  measurement, input measurement
 * @return void
 */
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_; //time_: predict 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;
  }
```

1、补全代码，实现基于地图的融合定位

1.3.1 更新G

更新卡尔曼增益G、状态观测量Y：
CorrectErrorEstimationPose

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

```
/**
 * @brief  correct error estimation using pose measurement
 * @param  T_nb, input pose measurement
 * @return void
 */
void ErrorStateKalmanFilter::CorrectErrorEstimationPose(
    const Eigen::Matrix4d &T_nb, Eigen::VectorXd &Y, Eigen::MatrixXd &G,
    Eigen::MatrixXd &K) {
  //
  //set measurement:// ppt p31
  //delta p
  Eigen::VectorXd& Y_measured = Y;
  Y_measured.resize(kDimMeasurementPose, 1);
  Y_measured.block<3, 1>(0,0) = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3); // 预测 - 测量

  //delta theta
  Eigen::Matrix3d delta_Cnb = T_nb.block<3,3>(0, 0).transpose() * pose_.block<3,3>(0, 0);
  Eigen::Matrix3d delta_theta_matrix = delta_Cnb - Eigen::Matrix3d::Identity();
  Y_measured.block<3, 1>(3, 0) = Sophus::SO3d::vee(delta_theta_matrix);

  // set measurement equation:
  G = GPose_;
  YPose_ = Y_measured;
  //const MatrixCPose& C = CPose_;
  // K = P * G^T * (G * P * G^T + C * R * C^T).inverse()
  //K = P_ * G.transpose() * (G * P_ * G.transpose() + C * RPose_ * C.transpose()).inverse();
  K = P_ * G.transpose() * (G * P_ * G.transpose() + RPose_).inverse(); //C是单位阵，可以不写

}
```

1、补全代码，实现基于地图的融合定位

深蓝学院
shenlanxueyuan.com

1.3.2 更新X和P

计算状态误差X,并更新状态误差协方差阵P:CorrectErrorEstimation

$$\hat{P}_k = (I - K_k G_k)\check{P}_k$$

$$\delta\hat{x}_k = \delta\check{x}_k + K_k(y_k - G_k\delta\check{x}_k)$$

```cpp
/**
 * @brief  correct error estimation
 * @param  measurement_type, measurement type
 * @param  measurement, input measurement
 * @return void
 */
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 = (I - K * G) * P
  P_ = (MatrixP::Identity() - K * G) * P_;

  //X = X + K * (y - G * X)
  X_ = X_ + K * (Y - G * X_);

}
```

1、补全代码，实现基于地图的融合定位

1.3.3 名义值更新

采用估计的误差状态量，对名义值进行更新：EliminateError

7）有观测时计算后验位姿

根据后验状态量，更新后验位姿。

$$\hat{\boldsymbol{p}}_k = \check{\boldsymbol{p}}_k - \delta\hat{\boldsymbol{p}}_k$$

$$\hat{\boldsymbol{v}}_k = \check{\boldsymbol{v}}_k - \delta\hat{\boldsymbol{v}}_k$$

$$\hat{\boldsymbol{R}}_k = \check{\boldsymbol{R}}_k(\boldsymbol{I} - [\delta\hat{\boldsymbol{\theta}}_k]_\times)$$

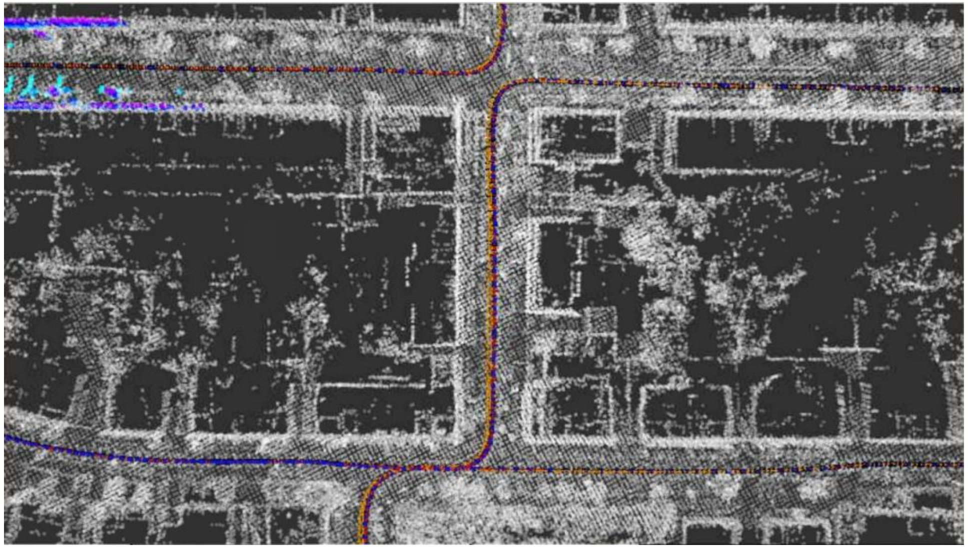
$$\hat{\boldsymbol{b}}_{a_k} = \check{\boldsymbol{b}}_{a_k} - \delta\hat{\boldsymbol{b}}_{a_k}$$

$$\hat{\boldsymbol{b}}_{\omega_k} = \check{\boldsymbol{b}}_{\omega_k} - \delta\hat{\boldsymbol{b}}_{\omega_k}$$

```
/**
 * @brief  eliminate error
 * @param  void
 * @return void
 */
void ErrorStateKalmanFilter::EliminateError(void) {
  //
  // TODO: correct state estimation using the state of ESKF
// a. position:
  pose_.block<3, 1>(0, 3) = pose_.block<3, 1>(0, 3) - X_.block<3, 1>(kIndexErrorPos, 0);

 // b. velocity:
  vel_ = vel_ - X_.block<3, 1>(kIndexErrorVel, 0);

// c. orientation:
  Eigen::Matrix3d delta_R = Eigen::MatrixXd::Identity(3,3) - Sophus::SO3d::hat(X_.block<3, 1>(kIndexErrorOri, 0)).matrix();//ppt p40

  Eigen::Quaterniond dq = Eigen::Quaterniond(delta_R);
  dq = dq.normalized();
  pose_.block<3, 3>(0, 0) = pose_.block<3, 3>(0, 0) * dq.toRotationMatrix();

  // d. gyro bias:
  if (IsCovStable(kIndexErrorGyro))
  {
    gyro_bias_ -= X_.block<3, 1>(kIndexErrorGyro, 0);
  }

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

1、补全代码，实现基于地图的融合定位

1.4 运行效果



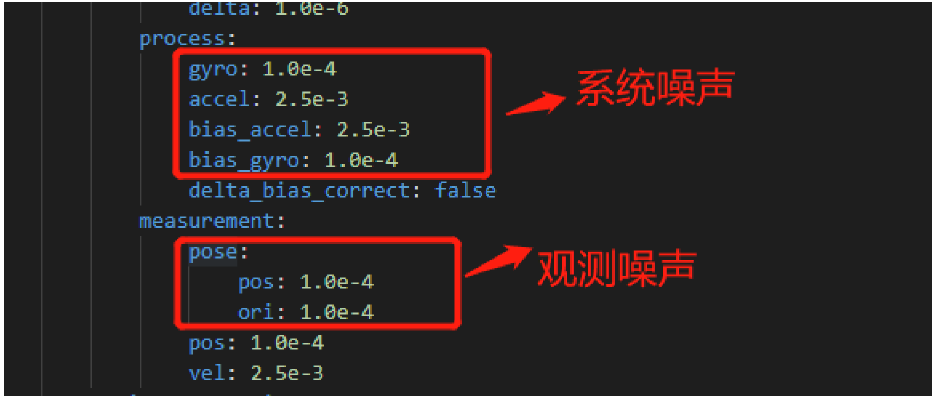
纲要

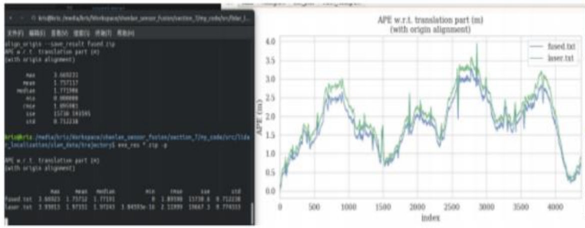
- 1、补全代码，实现基于地图的融合定位
- 2、调试参数，与不加滤波时的定位结果做比较
- 3、给出不考虑随机游走模型时的推导过程，并在工程中实现。对比两种方法的性能差异

2、调试参数，与不加滤波时的定位结果做比较

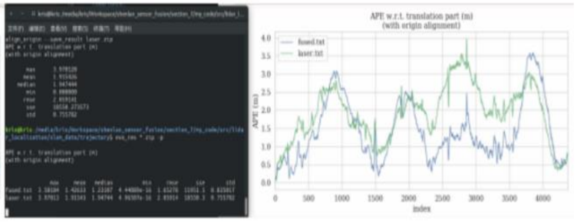
- 通过修改配置文件中过程噪声方差Q和观测噪声方差R之间的相对大小。
- 当Q较大时，预测噪声较大，系统更相信观测
- 当R较大时，观测噪声较大，系统更相信预测。
- 以上结论，符合kalman经典方程：R越大，K越小，说明观测值对状态估计的贡献越小。

$$K_k = \check{P}_k G_k^{\mathrm{T}} \left(G_k \check{P}_k G_k^{\mathrm{T}} + C_k R_k C_k^{\mathrm{T}} \right)^{-1}$$





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**

纲要

- 1、补全代码，实现基于地图的融合定位
- 2、调试参数，与不加滤波时的定位结果做比较
- 3、给出不考虑随机游走模型时的推导过程和实现。对比两种方法的性能差异

3、给出不考虑随机游走模型时的推导过程和实现。对比两种方法的性能差异

3.1 公式推导

➢ 不考虑随机游走的离散时间下的公式推导结果如下图：（F不变）

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

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

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

$$\boxed{\begin{aligned}\delta\dot{\boldsymbol{b}}_a &= 0\\ \delta\dot{\boldsymbol{b}}_\omega &= 0\end{aligned}}$$

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

3、给出不考虑随机游走模型时的推导过程和实现。对比两种方法的性能差异

深蓝学院
shenlanxueyuan.com

3.2 代码实现

➢ 在配置文件中，增加一个配置项delta_bias_correct，用于区分是否考虑随机游走模型。

```yaml
process:
    gyro: 1.0e-4
    accel: 2.5e-3
    bias_accel: 2.5e-3
    bias_gyro: 1.0e-4
    delta_bias_correct: false
```

➢ 加载该配置项

```cpp
    node["covariance"]["process"]["bias_gyro"].as<double>();
COV.PROCESS.DELTA_BIAS_CORRECT =
    node["covariance"]["process"]["delta_bias_correct"].as<bool>();
// d. measurement noise:
```

➢ 根据该配置项，设置状态方程中的B矩阵

```cpp
if(COV.PROCESS.DELTA_BIAS_CORRECT)
{
  B_.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = Eigen::Matrix3d::Identity();
  B_.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = Eigen::Matrix3d::Identity();
}
```

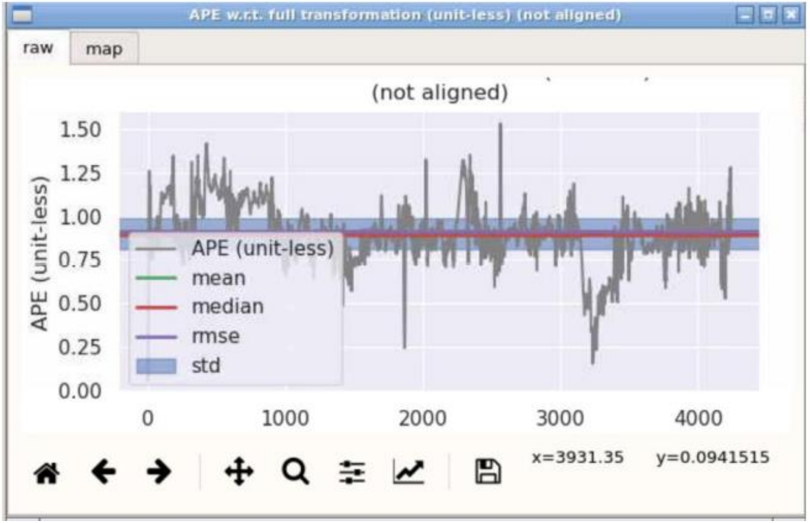
3、给出不考虑随机游走模型时的推导过程和实现。对比两种方法的性能差异

3.3 实验结果

➢ 考虑随机游走模型：

b. fused:

evo_ape kitti ground_truth.txt fused.txt -r full --plot --plot_mode xy




```
ctory# evo_ape kitti ground_truth.txt fused.txt -r full --plot --plot_mode xy
APE w.r.t. full transformation (unit-less)
(not aligned)

       max      1.532248
      mean      0.900310
    median      0.895257
       min      0.054862
      rmse      0.916651
       sse      3565.176209
       std      0.172308
```

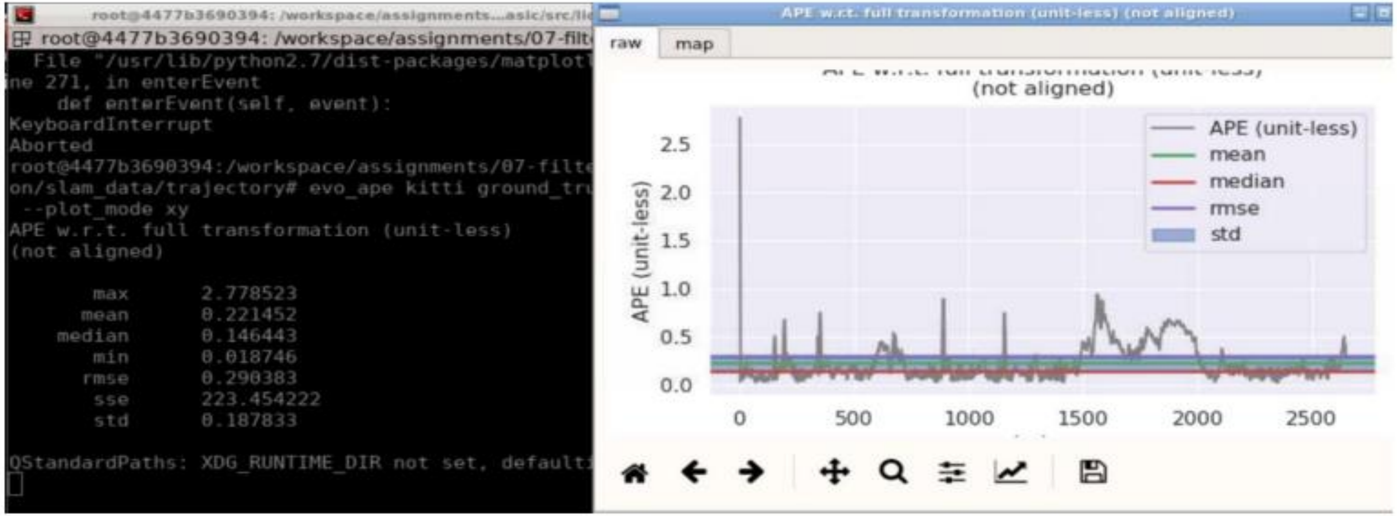
3、给出不考虑随机游走模型时的推导过程和实现。对比两种方法的性能差异

3.3 实验结果

➤ 不考虑随机游走模型：

b. fused:

evo_ape kitti ground_truth.txt fused.txt -r full --plot --plot_mode xy



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