深蓝学院
shenlanxueyuan.com

多传感器融合定位

——第八章作业分享

主讲人 田宇豪

纲要

- 1. 合格要求：融合雷达位姿和编码器速度

实现融合运动模型的滤波算法

1.融合编码器

根据第7讲所内容，状态量为

$$\delta \boldsymbol{x} = \begin{bmatrix} \delta \boldsymbol{p} \\ \delta \boldsymbol{v} \\ \delta \boldsymbol{\theta} \\ \delta \boldsymbol{b}_a \\ \delta \boldsymbol{b}_\omega \end{bmatrix}$$

而融合编码器以后，观测量变为

$$\boldsymbol{y} = \begin{bmatrix} \delta \bar{\boldsymbol{p}} \\ \delta \bar{\boldsymbol{v}}^b \\ \delta \bar{\boldsymbol{\theta}} \end{bmatrix}$$

其中 $\delta \bar{\boldsymbol{v}}^b$ 的观测值可以通过下式获得

$$\delta \bar{\boldsymbol{v}}_b = \tilde{\boldsymbol{v}}^b - \boldsymbol{v}^b = \tilde{\boldsymbol{R}}_{bw} \tilde{\boldsymbol{v}}^w - \begin{bmatrix} \boldsymbol{v}_m \\ 0 \\ 0 \end{bmatrix}$$

```cpp
void ErrorStateKalmanFilter::CorrectErrorEstimationPoseVel(
    const Eigen::Matrix4d &T_nb, const Eigen::Vector3d &v_b, const Eigen::Vector3d &w_b,
    Eigen::VectorXd &Y, Eigen::MatrixXd &G, Eigen::MatrixXd &K
) {
    // TODO: set measurement:
    // delta_p:
    Eigen::Vector3d delta_p = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
    // delta_theta:
    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());
    //delta_v:
    Eigen::Vector2d v_yz(0, 0);
    Eigen::Vector2d delta_v = (pose_.block<3,  3>(0, 0).transpose() * vel_).block<2, 1>(1, 0) - v_yz;
    // set measurement equation:
    YPoseVel_.block<3, 1>(0, 0) = delta_p;
    YPoseVel_.block<2, 1>(3, 0) = delta_v;
    YPoseVel_.block<3, 1>(5, 0) = delta_theta;
    Y = YPoseVel_;
    GPoseVel_.block<3, 3>(0, kIndexErrorPos) = Eigen::Matrix3d::Identity();
    GPoseVel_.block<2, 3>(3, kIndexErrorVel) = (pose_.block<3, 3>(0, 0).transpose()).block<2, 3>(1, 0);
    GPoseVel_.block<2, 3>(3, kIndexErrorOri) = (Sophus::SO3d::hat(pose_.block<3, 3>(0, 0).transpose()*vel_)).block<2, 3>(1, 0);
    GPoseVel_.block<3 ,3>(5, kIndexErrorOri) = Eigen::Matrix3d::Identity();
    G = GPoseVel_;
    CPoseVel_.setIdentity();
    Eigen::MatrixXd C = CPoseVel_;
    // TODO: set Kalman gain:
    K = P_ * G.transpose() * (G*P_*G.transpose() + RPoseVel_).inverse();
}
```

实现融合运动模型的滤波算法

此时的观测方程 $\boldsymbol{y} = \boldsymbol{G}_t \delta \boldsymbol{x} + \boldsymbol{C}_t \boldsymbol{n}$ 中的各变量应重新写为

$$\boldsymbol{G}_t = \begin{bmatrix} \boldsymbol{I}_3 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{R}_{bw} & [\boldsymbol{v}^b]_\times & 0 & 0 \\ 0 & 0 & \boldsymbol{I}_3 & 0 & 0 \end{bmatrix}$$

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

$$\boldsymbol{n} = \begin{bmatrix} n_{\delta \bar{p}_x} & n_{\delta \bar{p}_y} & n_{\delta \bar{p}_z} & n_{\delta \bar{v}_x^b} & n_{\delta \bar{v}_y^b} & n_{\delta \bar{v}_z^b} & n_{\delta \bar{\theta}_x} & n_{\delta \bar{\theta}_y} & n_{\delta \bar{\theta}_z} \end{bmatrix}^T$$

```cpp
void ErrorStateKalmanFilter::CorrectErrorEstimationPoseVel(
    const Eigen::Matrix4d &T_nb, const Eigen::Vector3d &v_b, const Eigen::Vector3d &w_b,
    Eigen::VectorXd &Y, Eigen::MatrixXd &G, Eigen::MatrixXd &K
) {
    // TODO: set measurement:
    // delta_p:
    Eigen::Vector3d delta_p = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
    // delta_theta:
    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());
    //delta_v:
    Eigen::Vector2d v_yz(0, 0);
    Eigen::Vector2d delta_v = (pose_.block<3,  3>(0, 0).transpose() * vel_).block<2, 1>(1, 0) - v_yz;
    // set measurement equation:
    YPoseVel_.block<3, 1>(0, 0) = delta_p;
    YPoseVel_.block<2, 1>(3, 0) = delta_v;
    YPoseVel_.block<3, 1>(5, 0) = delta_theta;
    Y = YPoseVel_;
    GPoseVel_.block<3, 3>(0, kIndexErrorPos) = Eigen::Matrix3d::Identity();
    GPoseVel_.block<2, 3>(3, kIndexErrorVel) = (pose_.block<3, 3>(0, 0).transpose()).block<2, 3>(1, 0);
    GPoseVel_.block<2, 3>(3, kIndexErrorOri) = (Sophus::SO3d::hat(pose_.block<3, 3>(0, 0).transpose()*vel_)).block<2, 3>(1, 0);
    GPoseVel_.block<3 ,3>(5, kIndexErrorOri) = Eigen::Matrix3d::Identity();
    G = GPoseVel_;
    CPoseVel_.setIdentity();
    Eigen::MatrixXd C = CPoseVel_;
    // TODO: set Kalman gain:
    K = P_ * G.transpose() * (G*P_*G.transpose() + RPoseVel_).inverse();
}
```

实现融合运动模型的滤波算法

2.融合编码器基础上添加运动约束

很多时候，硬件平台并没有编码器，不能直接使用上一小节的模型，但是车本身的运动特性(即侧向速度和天向速度为0)仍然可以使用。

它对观测量带来的改变仅仅是少了一个维度(x方向)，而推导方法并没有改变，因此此处直接给出该融合模式下的推导结果。

新的观测量为

$$\boldsymbol{y}=\begin{bmatrix}\delta\bar{\boldsymbol{p}}\\ [\delta\bar{\boldsymbol{v}}^b]_{yz}\\ \delta\bar{\boldsymbol{\theta}}\end{bmatrix}$$

$[\bullet]_{yz}$ 表示只取三维向量或矩阵的后2行

$$\boldsymbol{G}_t=\begin{bmatrix}\boldsymbol{I}_3 & 0 & 0 & 0 & 0\\ 0 & [\boldsymbol{R}_{bw}]_{yz} & [[\boldsymbol{v}^b]_\times]_{yz} & 0 & 0\\ 0 & 0 & \boldsymbol{I}_3 & 0 & 0\end{bmatrix}$$

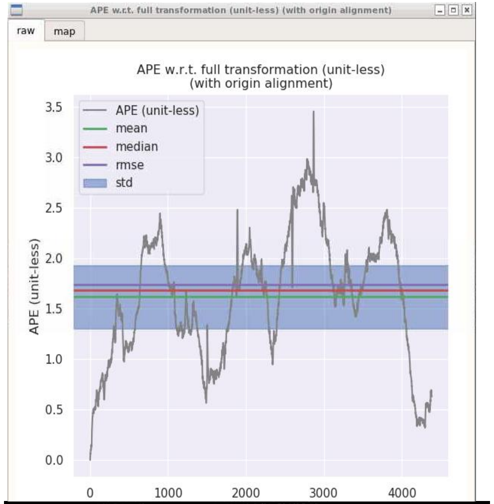
```cpp
void ErrorStateKalmanFilter::CorrectErrorEstimationPoseVel(
    const Eigen::Matrix4d &T_nb, const Eigen::Vector3d &v_b, const Eigen::Vector3d &w_b,
    Eigen::VectorXd &Y, Eigen::MatrixXd &G, Eigen::MatrixXd &K
) {
  // TODO: set measurement:
  // delta_p:
  Eigen::Vector3d delta_p = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
  // delta_theta:
  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());
  //delta_v:
  Eigen::Vector2d v_yz(0, 0);
  Eigen::Vector2d delta_v = (pose_.block<3,  3>(0, 0).transpose() * vel_).block<2, 1>(1, 0) - v_yz;
  // set measurement equation:
  YPoseVel_.block<3, 1>(0, 0) = delta_p;
  YPoseVel_.block<2, 1>(3, 0) = delta_v;
  YPoseVel_.block<3, 1>(5, 0) = delta_theta;
  Y = YPoseVel_;
  GPoseVel_.block<3, 3>(0, kIndexErrorPos) = Eigen::Matrix3d::Identity();
  GPoseVel_.block<2, 3>(3, kIndexErrorVel) = (pose_.block<3, 3>(0, 0).transpose()).block<2, 3>(1, 0);
  GPoseVel_.block<2, 3>(3, kIndexErrorOri) = (Sophus::SO3d::hat(pose_.block<3, 3>(0, 0).transpose()*vel_)).block<2, 3>(1, 0);
  GPoseVel_.block<3 ,3>(5, kIndexErrorOri) = Eigen::Matrix3d::Identity();
  G = GPoseVel_;
  CPoseVel_.setIdentity();
  Eigen::MatrixXd C = CPoseVel_;
  // TODO: set Kalman gain:
  K = P_ * G.transpose() * (G*P_*G.transpose() + RPoseVel_).inverse();
}
```

纲要

- 1. 合格要求：融合雷达位姿和编码器速度
- 2. 良好要求：对融合结果进行evo评估

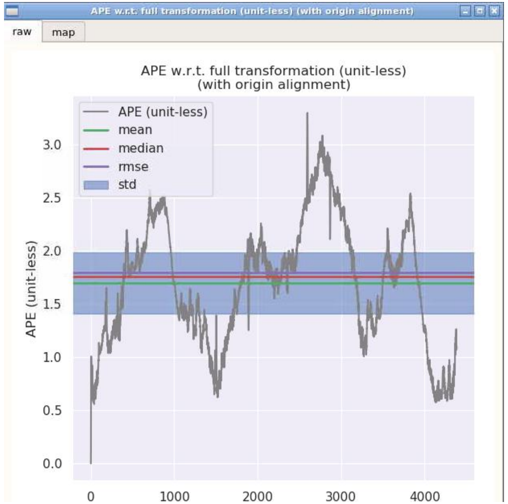
对融合结果进行evo评估

无速度约束



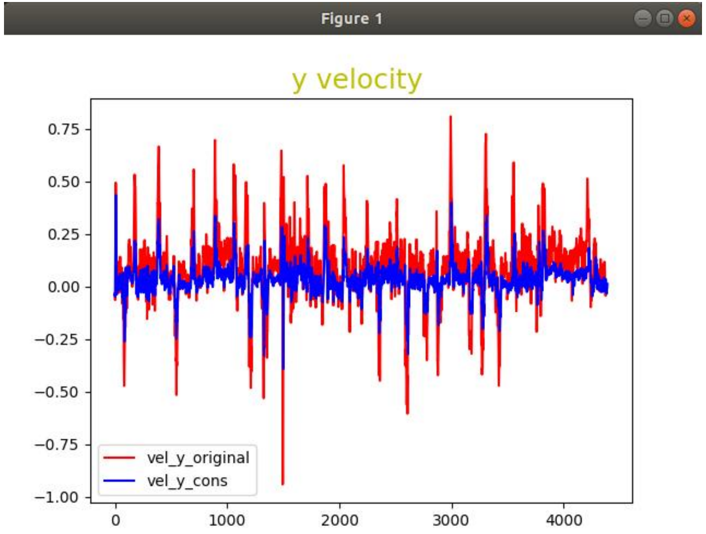
max	3.453187
mean	1.616953
median	1.684239
min	0.000001
rmse	1.735099
sse	13204.354777
std	0.629311

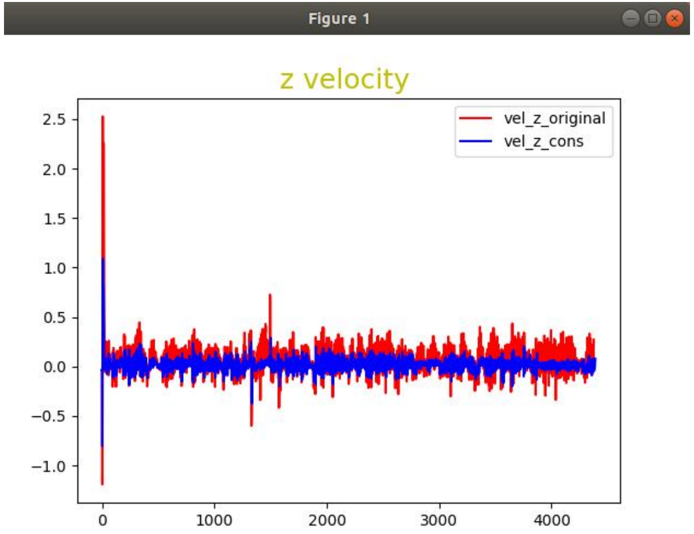
有速度约束



max	3.299906
mean	1.700190
median	1.756252
min	0.000002
rmse	1.795953
sse	14146.805404
std	0.578620

添加速度约束前后速度波动对比





纲要

- 1. 合格要求：融合雷达位姿和编码器速度
- 2. 良好要求：对融合结果进行evo评估
- 3. 优秀要求：融合gps位置和编码器速度

实现GPS与编码器测量值为观测量的融合

1.在(lidar+encoder)的观测上去掉姿态观测部分: $\boldsymbol{y} = \begin{bmatrix} \delta\overline{\boldsymbol{p}} \\ \delta\overline{\boldsymbol{v}}^{\mathrm{b}} \end{bmatrix}$

$$\delta\overline{\boldsymbol{v}}_{\mathrm{b}} = \tilde{\boldsymbol{v}}^{\mathrm{b}} - \boldsymbol{v}^{\mathrm{b}} = \tilde{\boldsymbol{R}}_{\mathrm{bw}}\tilde{\boldsymbol{v}}^{\mathrm{w}} - \begin{bmatrix} \boldsymbol{v}_{\mathrm{m}} \\ 0 \\ 0 \end{bmatrix}$$

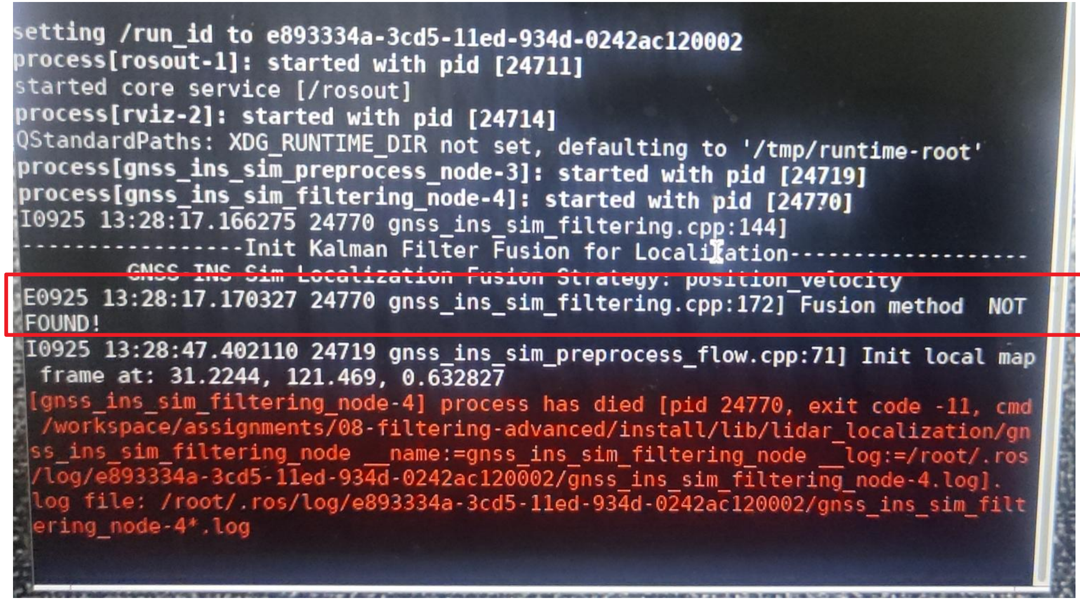
$$\boldsymbol{G}_{\mathrm{t}} = \begin{bmatrix} \boldsymbol{I}_3 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{R}_{\mathrm{bw}} & [\boldsymbol{v}^{\mathrm{b}}]_{\times} & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{C}_{\mathrm{t}} = \begin{bmatrix} \boldsymbol{I}_3 & 0 \\ 0 & \boldsymbol{I}_3 \end{bmatrix}$$

```
void ErrorStateKalmanFilter::CorrectErrorEstimationPosiVel(
    const Eigen::Matrix4d &T_nb, const Eigen::Vector3d &v_b, const Eigen::Vector3d &w_b,
    Eigen::VectorXd &Y, Eigen::MatrixXd &G, Eigen::MatrixXd &K
) {
    // TODO:
    // parse measurement:
    Eigen::Vector3d delta_p = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
    Eigen::Vector3d delta_v = pose_.block<3,  3>(0, 0).transpose() * vel_ - v_b;
    // set measurement equation:
    YPosiVel_.block<3, 1>(0, 0) = delta_p;
    YPosiVel_.block<3, 1>(3, 0) = delta_v;
    Y = YPosiVel_;
    GPosiVel_.block<3, 3>(3, kIndexErrorVel) = pose_.block<3, 3>(0, 0).transpose();
    GPosiVel_.block<3, 3>(3, kIndexErrorOri) = Sophus::SO3d::hat(pose_.block<3, 3>(0, 0).transpose() * vel_) ;
    G = GPosiVel_;
    Eigen::MatrixXd C = CPosiVel_;
    // set Kalman gain:
    K = P_ * G.transpose() * (G*P_*G.transpose() + RPosiVel_).inverse();
}
```

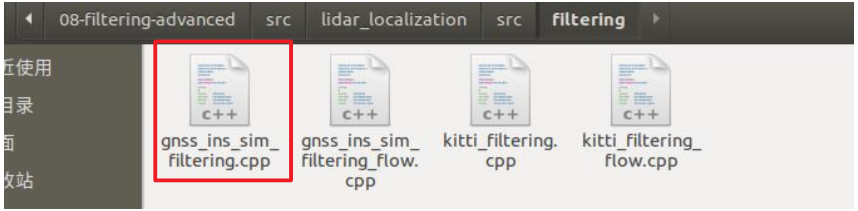
实现GPS与编码器测量值为观测量的融合

2.运行rviz,可能会报错



实现GPS与编码器测量值为观测量的融合

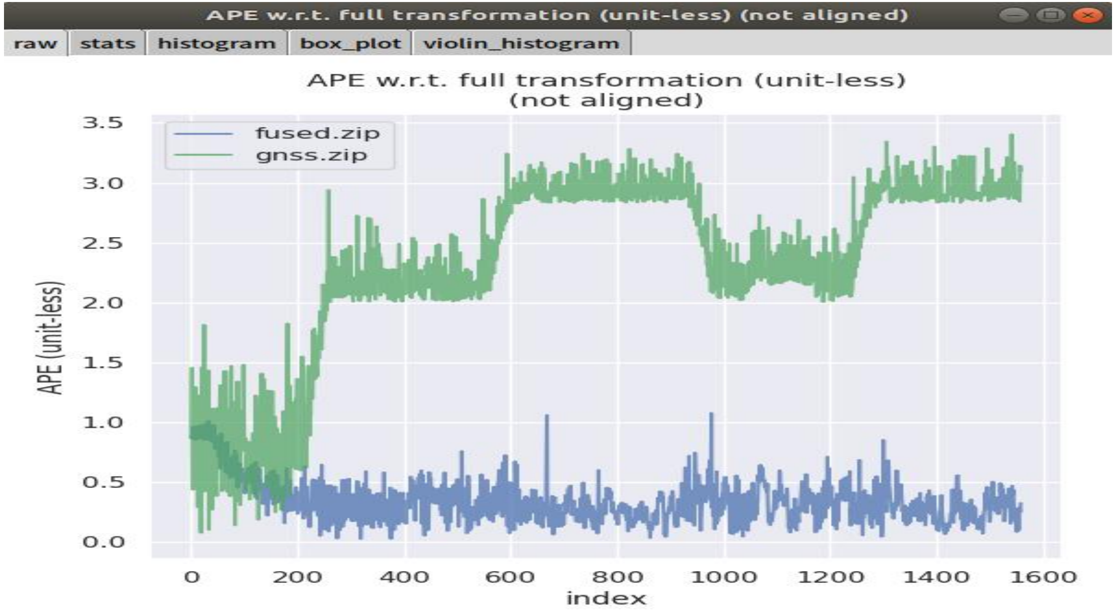
解决方法：




```cpp
bool GNSSINSSimFiltering::InitFusion(const YAML::Node& config_node) {
  // set up fusion strategy:
  CONFIG.FUSION_STRATEGY_ID["position_velocity"] = KalmanFilter::MeasurementType::POSI_VEL;

  std::string fusion_strategy = config_node["fusion_strategy"].as<std::string>();

  if ( CONFIG.FUSION_STRATEGY_ID.end() != CONFIG.FUSION_STRATEGY_ID.find(fusion_strategy) ) {
    CONFIG.FUSION_STRATEGY = CONFIG.FUSION_STRATEGY_ID.at(fusion_strategy);
  } else {
    LOG(ERROR) << "Fusion strategy " << fusion_strategy << " NOT FOUND!";
    return false;
  }
  std::cout << "\tGNSS-INS-Sim Localization Fusion Strategy: " << fusion_strategy << std::endl;

  // set up fusion method:
  CONFIG.FUSION_METHOD = config_node["fusion_method"].as<std::string>();

  if (CONFIG.FUSION_METHOD == "error_state_kalman_filter") {
    kalman_filter_ptr_ = std::make_shared<ErrorStateKalmanFilter>(config_node[CONFIG.FUSION_METHOD]);
  } else {
    LOG(ERROR) << "Fusion method " << CONFIG.FUSION_METHOD << " NOT FOUND!";
    return false;
  }
  std::cout << "\tGNSS-INS-Sim Localization Fusion Method: " << CONFIG.FUSION_METHOD << std::endl;

  return true;
}
```

添加一行代码

实现GPS与编码器测量值为观测量的融合



APE w.r.t. full transformation (unit-less)
(not aligned)

	rmse	mean	median	std	min	max	sse
fused.zip	0.380957	0.33643	0.302169	0.178726	0.0232184	1.07499	226.254
gnss.zip	2.43995	2.328	2.40139	0.730595	0.0744624	3.40497	9281.27

在线问答

Q&A

感谢各位聆听！

Thanks for Listening