各位同学好, 今天为大家带来第8节作业的思路分享。

这一章节主要是对滤波方案进行了扩展，加入了运动模型的约束。该章节的思路如果上一章节写的比较好，这一章节可能会相对比较简单。作业的第一部分为融合雷达位姿与编码器速度. 其核心思想是将激光与地图进行匹配得到的位姿和编码器速度当作测量量, 与imu积分得到的先验位姿进行融合。运动模型约束可以看作是编码器速度的一种特例, 只需要去掉x轴的速度分量就可以转换成运动模型约束，最后我们需要用仿真去验证下我们的结果。

## 作业分析

深蓝学院 shenlanxueyuan.com

在上一讲作业里实现的滤波方案的基础上，实现融合运动模型的滤波方法，并对比加入运动模型约束前后，滤波精度的变化。由于运动模型约束更多的是改善速度的波动，而且是y向和z向的波动，因此要求展示结果时，提供b系y向和z向速度误差的曲线与指标。

注：同样由于kitti数据集质量的问题，效果的改善不一定在所有路段都能体现，可以挑选效果好的路段重点展示。

**评价标准：**

1)及格：实现新模型，且功能正常
2)良好：实现新模型，且部分路段性能有改善
3)优秀：在良好的基础上，增加编码器融合的内容，具体如下：
使用仿真数据(仿真数据需要根据下方链接中的仿真程序自己生成)，实现以gps位置和编码器速度为观测量的融合方法，并分析其精度。

理论上PPT上已经给出，因此不需要额外推导。我们可以专注于代码部分。相比于上一章的作业，主要需要进行改动的就是CorrectErrorEstimation这个函数, 我们需要在其中增加针对编码器速度拓展的新的接口，考虑到KITTI中没有编码器相关数据，因此只对y和z方向进行约束。需要注意，和PPT中的表示方法不同，为了方便处理，建议速度将作为第三类观测量置于角度之后，而不是和PPT之中放在位置和角度之间。

## 实现运动约束模型

深蓝学院 shenlanxueyuan.com

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

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

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

代码方面：

- CorrectErrorEstimation函数： 需要在其中增加针对编码器速度拓展的新的接口
- 运动约束模型：考虑到KITTI中没有编码器相关数据，因此只对y和z方向进行约束。（注意，和PPT中的表示方法不同，为了方便处理，建议速度将作为第三类观测量置于角度之后，而不是和PPT之中放在位置和角度之间。）

这里简单的展示了下代码实现，比较简单，有不会的可以看一下：

首先是代表运动模型约束的部分的实现。

```
  // 预测-观测：位移->角度->速度
  YPoseWithMC_.block<3, 1>(0, 0) = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0,
3);
  YPoseWithMC_.block<3, 1>(3, 0) = deltaTheta;
  Eigen::Vector3d v_b_pred(v_b.norm(), 0.0, 0.0);
  // std::cout << predPose.transpose() * vel_ << std::endl;
  YPoseWithMC_.block<2, 1>(6, 0) = (predPose.transpose() * vel_ -
v_b_pred).block<2, 1>(1, 0);

  // TODO: set measurement equation:
  Y = YPoseWithMC_;
  G = GPoseWithMC_;
  // Rbw = Rbn = predPose.transpose()
  G.block<2, 3>(6, 3) = (predPose.transpose()).block<2, 3>(1, 0);
  G.block<2, 3>(6, 6) = (Sophus::SO3d::hat(v_b).matrix()).block<2, 3>(1, 0);

  // TODO: set Kalman gain:
  K = P_ * G.transpose() * (G * P_ * (G.transpose()) + CPoseWithMC_ * RPoseWithMC_
* CPoseWithMC_.transpose()).inverse();
```

这部分是代表姿态-速度约束的部分的实现。

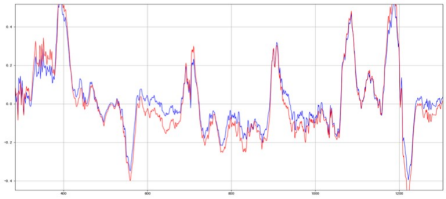
在计算完卡尔曼增益K, 观测矩阵G, 测量值Y之后, 我们就可以根据卡尔曼的更新公式, 完成对误差值的后验估计的计算. 之后就会执行与上一章一样的, 使用误差值对于状态值进行更新并发布. 这个流程展示了对于卡尔曼滤波框架来说, 如何将新增加的观测值融合到系统中, 从而提高系统的精度. 相比于基于优化的融合框架来说, 基于滤波的融合对于新增传感器的处理更为简单, 并不需要对数据做额外的处理, 只要传感器测量值和状态量之间可以建立起直接的观测关系, 就可以进行融合：

实现运动约束模型
深蓝学院 shenlanxueyuan.com

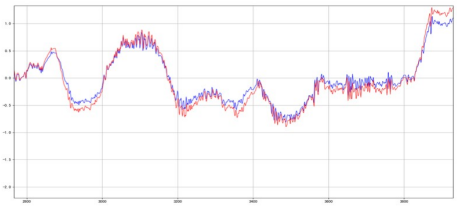
```
  // 预测-观测：
  YPoseVel_.block<3, 1>(0, 0) = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
  YPoseVel_.block<3, 1>(3, 0) = deltaTheta;
  YPoseVel_.block<3, 1>(6, 0) = (predPose.transpose() * vel_ - v_b_pred);

  // TODO: set measurement equation:
  Y = YPoseVel_;
  G = GPoseVel_;
  G.block<3, 3>(6, 3) = predPose.transpose();
  G.block<3, 3>(6, 6) = Sophus::SO3d::hat(v_b_pred).matrix();

  // TODO: set Kalman gain:
  K = P_ * G.transpose() * (G * P_ * (G.transpose()) + CPoseVel_ * RPoseVel_ *
CPoseVel_.transpose()).inverse();
}
```

可视化部分，我自己写了一个可视化程序，用于绘制y轴，z轴相关曲线，左侧展示其中的一部分可视化界面，右侧是定量分析展示：

## 实现运动约束模型



Y轴



Z轴

```
nfos:  4390 poses, 3712.908m path length
PE w.r.t. full transformation (unit-less)
with origin alignment)

     max      3.453187
    mean      1.616198
  median      1.684988
     min      0.000001
    rmse      1.734038
     sse      13200.234901
     std      0.628324

PE w.r.t. full transformation (unit-less)
with origin alignment)

     max      2.399886
    mean      1.136519
  median      1.116511
     min      0.000002
    rmse      1.216617
     sse      6497.891043
     std      0.434145
```

然后是仿真数据融合与评估，位置-速度模型代码展示如下：

## 仿真数据融合与评估

### 0. 位置-速度模型

```
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
) {
  Eigen::Matrix3d predPose = pose_.block<3, 3>(0, 0);
  Eigen::Vector3d v_b_pred(v_b.norm(), 0.0, 0.0);
  // 预测-观测：位移->速度
  YPosiVel_.block<3, 1>(0, 0) = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
  YPosiVel_.block<3, 1>(3, 0) = predPose.transpose() * vel_ - v_b_pred;

  // TODO: set measurement equation:
  Y = YPosiVel_;
  G = GPosiVel_;
  G.block<3, 3>(3, 3) = predPose.transpose();
  G.block<3, 3>(3, 6) = Sophus::SO3d::hat(v_b).matrix();

  // TODO: set Kalman gain:
  K = P_ * G.transpose() * (G * P_ * (G.transpose()) + CPosiVel_ * RPosiVel_ *
CPosiVel_.transpose()).inverse();
}
```

最后这里在运行的时候需要提醒一下大家，左边的两张图是gnss-ins-sim生成测量数据时候的误差等级，课程中默认使用的都是中等精度，也就是说gps位置的协方差应该在2.5e-1左右，而编码器应该在2.5e-3左右，而右边是我们的作业使用的默认配置文件中的误差等级. 可以看出gps的实际误差与我们给的先验值的差距比较巨大，这就是为什么有些同学在程序正确的情况下rviz中显示的姿态抖动非常厉害的原因. 建议大家在左边这两个值的基础上进行调参.

# 仿真数据融合与评估

深蓝学院 shenlanxueyuan.com

## 0. 调整参数

- 对yaml配置进行调整：
  - 更改经纬度
  - 更改经纬度所在地的重力大小
- 调整代码中协方差

```
chen@chen-NH5x-NH7xHP: ~/SLAM_System/evo评测工具/fusion_hw/task8/sim
(py35) chen@chen-NH5x-NH7xHP:~/SLAM_System/evo评测工具/fusion_hw/task8/sim$ sh test.sh
------------------------------------------------------------------------------
name:   fused
infos:  1552 poses, 1881.448m path length
------------------------------------------------------------------------------
name:   gnss
infos:  1552 poses, 2567.403m path length
APE w.r.t. full transformation (unit-less)
(not aligned)

       max      3.807498
      mean      2.330830
    median      2.377389
       min      0.132161
      rmse      2.441675
       sse      9252.675778
       std      0.727327

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

       max      1.106297
      mean      0.339668
    median      0.325489
       min      0.010037
      rmse      0.372026
       sse      214.801686
       std      0.151753

(py35) chen@chen-NH5x-NH7xHP:~/SLAM_System/evo评测工具/fusion_hw/task8/sim$
```

# 融合GPS位置与编码器速度

深蓝学院 shenlanxueyuan.com

```
'gps': {
    'no_error': {
        'stdp': np.array([0.0, 0.0, 0.0]),
        'stdv': np.array([0.0, 0.0, 0.0])
    },
    'high_accuracy': {
        'stdp': np.array([0.10, 0.10, 0.10]),
        'stdv': np.array([0.01, 0.01, 0.01])
    },
    'mid_accuracy': {
        'stdp': np.array([0.50, 0.50, 0.50]),
        'stdv': np.array([0.02, 0.02, 0.02])
    },
    'low_accuracy': {
        'stdp': np.array([1.00, 1.00, 1.00]),
        'stdv': np.array([0.05, 0.05, 0.05])
    }
},
```

```
'odo': {
    'no_error': {
        'scale': 1.00,
        'stdv': 0.0
    },
    'high_accuracy': {
        'scale': 1.00,
        'stdv': 0.01
    },
    'mid_accuracy': {
        'scale': 1.00,
        'stdv': 0.05
    },
    'low_accuracy': {
        'scale': 1.00,
        'stdv': 0.10
    }
}
```

```
measurement:
    pose:
        pos: 1.0e-4
        ori: 1.0e-4
    pos: 1.0e-4
    vel: 2.5e-3
```

最后为大家提供一些和编码器融合的相关文献，希望能给大家一些启发：

# 补充资料

深蓝学院 shenlanxueyuan.com

- Tightly-coupled Monocular Visual-odometric SLAM using Wheels and a MEMS Gyroscope
- Visual-Inertial Odometry Tightly Coupled with Wheel Encoder Adopting Robust Initialization and Online Extrinsic Calibration
- Pose Estimation for Ground Robots: On Manifold Representation, Integration, Repara meterization, and Optimization Mingming
- Visual-Odometric Localization and Mapping for Ground Vehicles Using SE(2)-XYZ Constraints