第四次作业内容

第1题

作业1：任选一种滤波模型和方法，在kitti数据集上，实现基于点云地图的融合定位。

准备数据

由于需要100Hz的imu数据，需要下载2011_10_03_drive_0027_sync.zip、2011_10_03_drive_0027_extract.zip、2011_10_03_calib.zip三个文件，地址如下

```
https://s3.eu-central-1.amazonaws.com/avg-
kitti/raw_data/2011_10_03_drive_0027/2011_10_03_drive_0027_sync.zip
https://s3.eu-central-1.amazonaws.com/avg-
kitti/raw_data/2011_10_03_drive_0027/2011_10_03_drive_0027_extract.zip
https://s3.eu-central-1.amazonaws.com/avg-kitti/raw_data/2011_10_03_calib.zip
```

可以使用反代下载

```
http://us5.funcs.xyz/avg-
kitti/raw_data/2011_10_03_drive_0027/2011_10_03_drive_0027_sync.zip
http://us5.funcs.xyz/avg-
kitti/raw_data/2011_10_03_drive_0027/2011_10_03_drive_0027_extract.zip
http://us5.funcs.xyz/avg-kitti/raw_data/2011_10_03_calib.zip
```

我只下载1、3文件，extract.zip没有下载，感谢群中朋友“无我”提供的oxts_extract.zip文件；

将sync.zip与calib.zip文件解压到同一目录，并使用extract.zip中的oxts替代sync.zip中的oxts目录，目录结构为

```
2011_10_03/
|-2011_10_03_drive_0027_sync/
  |-image_00/
  |-image_01/
  |-image_02/
  |-image_03/
  |-velodyne_points/
  |-oxts/
|-calib_cam_to_cam.txt
|-calib_velo_to_cam.txt
|-calib_imu_to_velo.txt
```

准备kitti2bag

```
sudo pip install -U numpy
sudo pip install kitti2bag
```

生成kitti_2011_10_03_drive_0027_synced.bag，其中imu数据是100Hz的；

```
cd 2011_10_03/..
kitti2bag -t 2011_10_03 -r 0027 raw_synced
```

融合定位

代码见压缩包的project_c4/ws目录，改动为2b10b17b提交；

使用课程中的基于误差状态的滤波方法，在第3章作业的基础上加入imu_laser_fusion_node，使用原来的data_pretreat_node与front_end_node用于生成点云pose；

fusion.launch:

```
<launch>
    <node pkg="rviz"  type="rviz"  name="rviz"  args="-d $(find lidar_localization)/rviz/fusion.rviz"></node>
    <node pkg="lidar_localization"  type="data_pretreat_node" name="data_pretreat_node"  output="screen"></node>
    <node pkg="lidar_localization"  type="front_end_node"  name="front_end_node" output="screen"></node>
    <node pkg="lidar_localization"  type="imu_laser_fusion_node" name="imu_laser_fusion_node"  output="screen"></node>
```

topic

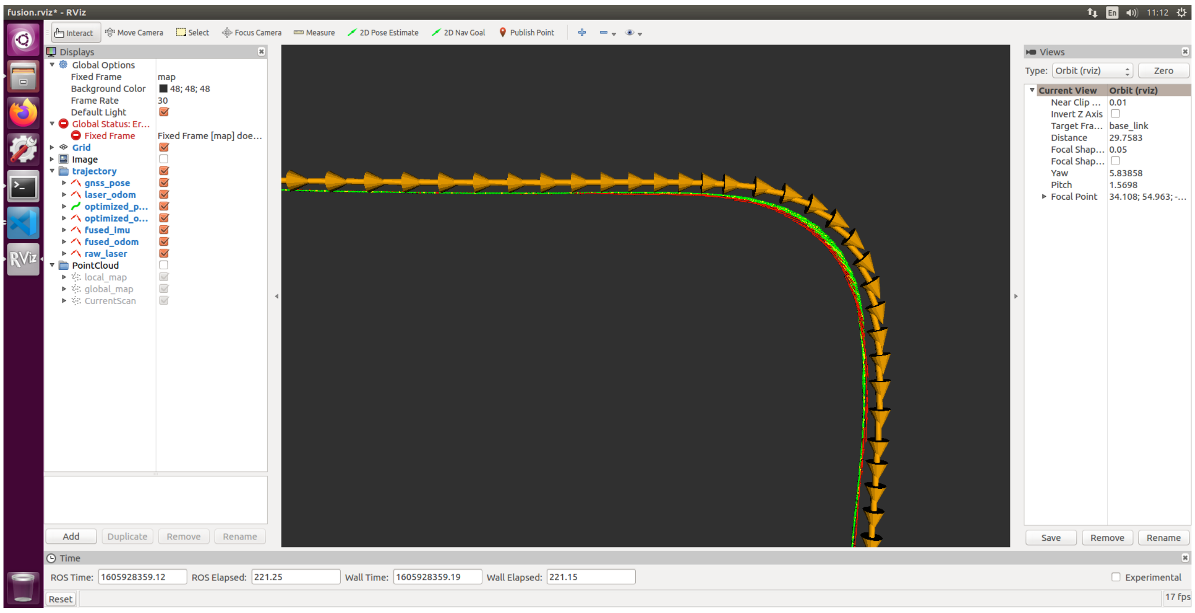
```
/kitti/oxts/imu     //100Hz的imu信息，用于解算与predict
/laser_odom         //由front_end_node产生，用于update
/synced_gnss        //使用其第一帧作为ENU坐标下机器初值
/fused_imu          //由imu_laser_fusion_node产生，解算后的pose
/fused_odom         //由imu_laser_fusion_node产生，observation update后的pose
/raw_laser_odom     //由imu_laser_fusion_node产生，将/laser_odom投影到ENU下的pose
```

主要流程在imu_laser_fusion.cpp，其中ProcessImu()中为解算与predict，ProcessOdom()中为observation update；

```
/*
 * @Description: imu laser kalman filter
 * @Author: abao
 * @Date: 2020-11-14
 */
#include "lidar_localization/mapping/fusion/imu_laser_fusion.hpp"

#include <iostream>
#include <fstream>
#include <boost/filesystem.hpp>
#include "glog/logging.h"

#include "lidar_localization/global_defination/global_defination.h"

namespace lidar_localization {

constexpr double kDegreeToRadian = M_PI / 180.;
constexpr double kRadianToDegree = 180. / M_PI;

ImuLaserFusion::ImuLaserFusion() {
    acc_noise_ = 1e-2;
    gyro_noise_ = 1e-4;
    acc_bias_noise_ = 1e-6;
    gyro_bias_noise_ = 1e-8;
    laser_pose_noise_ = 0.1;
    laser_orient_noise_ = 5*kDegreeToRadian;
}

bool ImuLaserFusion::SetInitPose(const Eigen::Matrix4f& init_pose) {
    initialized_ = true;
    // set global P/v/R
    state_.G_P = Eigen::Vector3d(init_pose(0,3), init_pose(1,3), init_pose(2,3));
    state_.G_v.setZero();
    state_.G_R = init_pose.block<3, 3>(0, 0).cast<double>();
    // set bias
    state_.acc_bias.setZero();
    state_.gyro_bias.setZero();

    // Set covariance.
    state_.cov.setZero();
    state_.cov.block<3, 3>(0, 0) = 100. * Eigen::Matrix3d::Identity(); // position
std: 10 m
    state_.cov.block<3, 3>(3, 3) = 100. * Eigen::Matrix3d::Identity(); // velocity
std: 10 m/s
    // roll pitch std 10 degree.
    state_.cov.block<2, 2>(6, 6) = 10. * kDegreeToRadian * 10. * kDegreeToRadian *
Eigen::Matrix2d::Identity();
    state_.cov(8, 8)             = 100. * kDegreeToRadian * 100. * kDegreeToRadian; //
yaw std: 100 degree.
    // Acc bias.
    state_.cov.block<3, 3>(9, 9) = 0.0004 * Eigen::Matrix3d::Identity();
    // Gyro bias.
    state_.cov.block<3, 3>(12, 12) = 0.0004 * Eigen::Matrix3d::Identity();
    return true;
}

bool ImuLaserFusion::ProcessImu(const IMUData& imu) {
    if (!initialized_)
        return false;

    if (!has_last_imu_) {
        has_last_imu_ = true;
        last_imu_ = imu;
        return false;
    }

    const double delta_t = imu.time - last_imu_.time;
    const double delta_t2 = delta_t * delta_t;
    LOG(INFO) << "imu,diff=" << delta_t;
    if (delta_t <= 0.f) {
        std::cout << "time revert..\n";
        return false;
    }

    State last_state = state_;

    // Acc and gyro.
    const Eigen::Vector3d last_acc = Eigen::Vector3d(last_imu_.linear_acceleration.x,
last_imu_.linear_acceleration.y, last_imu_.linear_acceleration.z);
    const Eigen::Vector3d curr_acc = Eigen::Vector3d(imu.linear_acceleration.x,
imu.linear_acceleration.y, imu.linear_acceleration.z);
```

```
    const Eigen::Vector3d last_gyro = Eigen::Vector3d(last_imu_.angular_velocity.x,
last_imu_.angular_velocity.y, last_imu_.angular_velocity.z);
    const Eigen::Vector3d curr_gyro = Eigen::Vector3d(imu.angular_velocity.x,
imu.angular_velocity.y, imu.angular_velocity.z);
    const Eigen::Vector3d acc_unbias = 0.5 * (last_acc + curr_acc) -
last_state.acc_bias;
    const Eigen::Vector3d gyro_unbias = 0.5 * (last_gyro + curr_gyro) -
last_state.gyro_bias;

    // Normal state
    state_.G_P = last_state.G_P + last_state.G_v * delta_t +
                   0.5 * (last_state.G_R * acc_unbias + gravity_) * delta_t2;
    state_.G_v = last_state.G_v + (last_state.G_R * acc_unbias + gravity_) * delta_t;
    const Eigen::Vector3d delta_angle_axis = gyro_unbias * delta_t;
    if (delta_angle_axis.norm() > 1e-12) {
        state_.G_R = last_state.G_R * Eigen::AngleAxisd(delta_angle_axis.norm(),
delta_angle_axis.normalized()).toRotationMatrix();
    }

    // \dot X = F_tX + B_tW

    // F_{k-1} = I + F_tT
    const double omega = 7.292115e-5;
    const double L = 48.982530923568 * kDegreeToRadian;
    Eigen::Matrix<double, 15, 15> Ft = Eigen::Matrix<double, 15, 15>::Zero();
    Ft.block<3,3>(0,3) = Eigen::Matrix3d::Identity();
    Ft.block<3,3>(3,6) = -state_.G_R * GetSkewMatrix(acc_unbias); // F23
    Ft(6,7) = omega * sin(L);  // F33
    Ft(6,8) = -omega * cos(L);
    Ft(7,6) = -omega * sin(L);
    Ft(8,6) = omega * cos(L);
    Ft.block<3,3>(3,12) = state_.G_R;
    Ft.block<3,3>(6,9) = -state_.G_R;
    Eigen::Matrix<double, 15, 15> Fk_1 = Eigen::Matrix<double, 15, 15>::Identity() +
Ft * delta_t;

    // B_{k-1} = B_tT
    Eigen::Matrix<double, 15, 6> Bt = Eigen::Matrix<double, 15, 6>::Zero();
    Bt.block<3,3>(3,3) = state_.G_R;
    Bt.block<3,3>(6,0) = -state_.G_R;
    Eigen::Matrix<double, 15, 6> Bk_1 = Bt * delta_t;

    Eigen::Matrix<double, 6, 6> Qk = Eigen::Matrix<double, 6, 6>::Zero();
    Qk.block<3, 3>(0, 0) = delta_t2 * gyro_noise_ * Eigen::Matrix3d::Identity();
    Qk.block<3, 3>(3, 3) = delta_t2 * acc_noise_ * Eigen::Matrix3d::Identity();

    // update P
    state_.cov = Fk_1 * last_state.cov * Fk_1.transpose() + Bk_1 * Qk *
Bk_1.transpose();

    last_imu_ = imu;
    return true;
}

bool ImuLaserFusion::ProcessOdom(const PoseData& pose_data) {
    // observation
    // Yk = [dPx, dPy, dPz, dPhx, dPhy, dPhz]
    const Eigen::Matrix3d C_b_n_p = state_.G_R;
    const Eigen::Matrix3d C_b_n = pose_data.pose.cast<double>().block<3,3>(0,0);
    const Eigen::Matrix3d C_n_n = C_b_n_p * C_b_n.transpose();
    const Eigen::Matrix3d phi_x = Eigen::Matrix3d::Identity() - C_n_n;
    // C_n^n \approx I - \phi\times
    double phx = (phi_x(2,1) - phi_x(1,2)) / 2.;
    double phy = (phi_x(0,2) - phi_x(2,0)) / 2.;
    double phz = (phi_x(1,0) - phi_x(0,1)) / 2.;
    //LOG(INFO) << "phi=" << std::endl << phi_x << std::endl << "p=" << px << "," << py
<< "," << pz << std::endl;
    Eigen::Matrix<double, 6, 1> Yk;
    Yk.block<3,1>(0,0) = pose_data.pose.cast<double>().block<3,1>(0,3) - state_.G_P;
    Yk(3,0) = phx;
    Yk(4,0) = phy;
    Yk(5,0) = phz;

    // K_k = P_kG_k^T(G_kP_kG_k^T + C_KR_kC_k^T)^{-1}
    const Eigen::MatrixXd& Pk = state_.cov;
    Eigen::Matrix<double, 6, 15> Gk = Eigen::Matrix<double, 6, 15>::Zero();
    Gk.block<3,3>(0,0) = Gk.block<3,3>(3,6) = Eigen::Matrix3d::Identity();
    Eigen::Matrix<double, 6, 6> Ck = Eigen::Matrix<double, 6, 6>::Identity();
    Eigen::Matrix<double, 6, 6> Rk = Eigen::Matrix<double, 6, 6>::Identity();
    Rk.block<3,3>(0,0) *= laser_pose_noise_;
    Rk.block<3,3>(3,3) *= laser_orient_noise_;
    const Eigen::MatrixXd Kk = Pk * Gk.transpose() * (Gk * Pk * Gk.transpose() + Ck *
Rk * Ck.transpose()).inverse();

    // \hat{P_k} = (I-K_kG_k)\check{P_k}
    state_.cov = (Eigen::Matrix<double, 15, 15>::Identity() - Kk * Gk) * Pk;

    // \hat{x_k} = \check{x_k} + K_k(Y_k-G_k\check{X_k})
    Eigen::Matrix<double, 15, 1> Xk = Eigen::Matrix<double, 15, 1>::Zero();
    const Eigen::VectorXd delta_x = Kk * (Yk - Gk*Xk);

    // \hat{x_k} = \check{x_k} + delta_x
    state_.G_P     += delta_x.block<3, 1>(0, 0);
    state_.G_v     += delta_x.block<3, 1>(3, 0);
    state_.acc_bias  += delta_x.block<3, 1>(9, 0);
    state_.gyro_bias += delta_x.block<3, 1>(12, 0);
    if (delta_x.block<3, 1>(6, 0).norm() > 1e-12) {
        state_.G_R *= Eigen::AngleAxisd(delta_x.block<3, 1>(6, 0).norm(),
delta_x.block<3, 1>(6, 0).normalized()).toRotationMatrix();
    }

    return true;
}

Eigen::Matrix4f ImuLaserFusion::GetPose() {
    Eigen::Matrix4f pose;
    pose.block<3,3>(0,0) = state_.G_R.cast<float>();
    pose(0,3) = state_.G_P(0);
    pose(1,3) = state_.G_P(1);
    pose(2,3) = state_.G_P(2);
    return pose;
}

Eigen::Matrix3d ImuLaserFusion::GetSkewMatrix(const Eigen::Vector3d& v) {
    Eigen::Matrix3d w;
    w <<  0.,   -v(2),  v(1),
          v(2),  0.,   -v(0),
         -v(1),  v(0),  0.;

    return w;
}

}
```

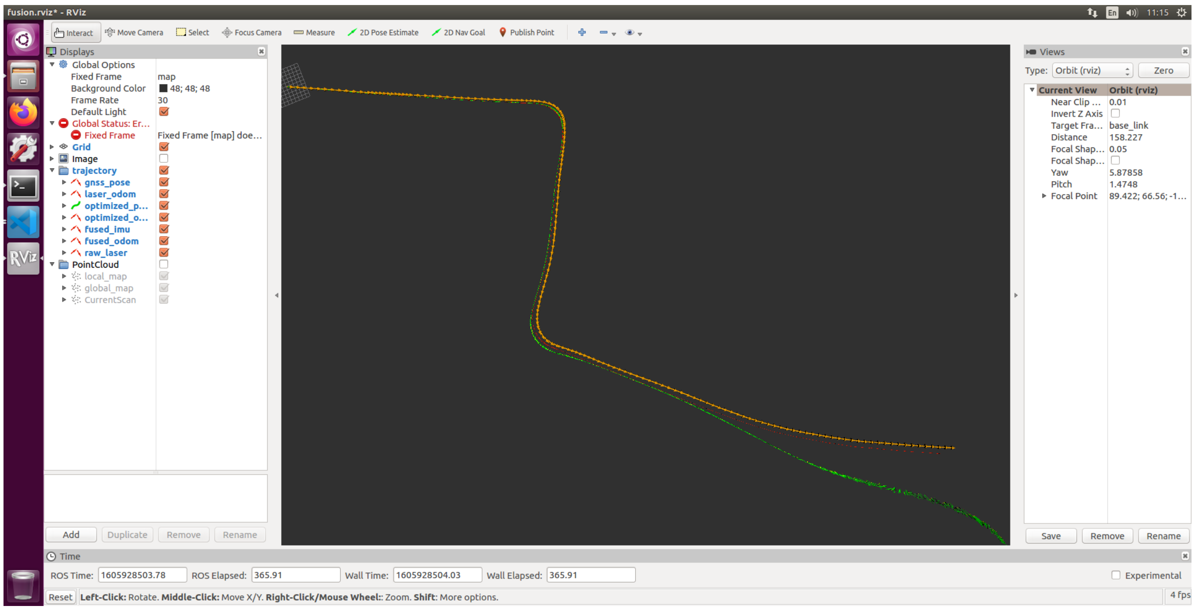
结果如下，其中大箭头是gnss，小红色箭头是/raw_laser_odom，小绿色箭头是融合位姿；



```
Fixed Frame        map
Background Color   48; 48; 48
Frame Rate         30
Fixed Frame        Fixed Frame [map] doe...

Type: Orbit (rviz)   Zero
Current View   Orbit (rviz)
Near Clip ...  0.01
Invert Z Axis
Target Fra...  base_link
Distance       29.7583
Focal Shap...  0.05
Focal Shap...
Yaw            5.83858
Pitch          1.5698
Focal Point    34.108; 54.963; -...

ROS Time: 1605928359.12   ROS Elapsed: 221.25   Wall Time: 1605928359.19   Wall Elapsed: 221.15
17 fps
```

可能是我生成的bag文件有问题，或其中的imu数据有问题，只能跑前面36秒左右数据，后面front_end_node不再生成/laser_odom，没有观测，后面发散了；

不过/laser_odom中断前也有一段位姿离gnss较远，不知道原因；



```
Fixed Frame        map
Background Color   48; 48; 48
Frame Rate         30
Fixed Frame        Fixed Frame [map] doe...

Type: Orbit (rviz)   Zero
Current View   Orbit (rviz)
Near Clip ...  0.01
Invert Z Axis
Target Fra...  base_link
Distance       158.227
Focal Shap...  0.05
Focal Shap...
Yaw            5.87858
Pitch          1.4748
Focal Point    89.422; 66.56; -1...

ROS Time: 1605928503.78   ROS Elapsed: 365.91   Wall Time: 1605928504.03   Wall Elapsed: 365.91
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
4 fps
```

第2题

作业2： 1) 推导组合导航(gps+imu)的滤波模型(相比于基于地图定位，只有观测方程发生了变化)，对静止、 匀速、转向、加减速等不同运动状态下各状态量的可观测性和可观测度进行分析。 2) 使用第三章所述数据仿真软件，产生对应运动状态的数据，进行kalman滤波。 3) 统计kalman滤波中各状态量的收敛速度和收敛精度，并与可观测度分析的结果汇总比较。

gps+imu滤波模型

基于状态误差

状态方程与imu+lidar一致

$$\dot{X} = F_t X + B_t W$$

$$\begin{aligned}
\text{状态量}\ X &= [\delta P^T\ \delta V^T\ \phi^T\ \varepsilon^T\ \nabla^T]^T \\
\delta P^T &= [\delta P_E\ \delta P_N\ \delta P_U]^T \\
\delta V^T &= [\delta V_E\ \delta V_N\ \delta V_U]^T \\
\phi &= [\phi_E\ \phi_N\ \phi_U]^T \\
\varepsilon &= [\varepsilon_x\ \varepsilon_y\ \varepsilon_z]^T \\
\nabla &= [\nabla_x\ \nabla_y\ \nabla_z]^T
\end{aligned}$$

$$\text{器件噪声}\ W = [w_{gx}\ w_{gy}\ w_{gz}\ w_{ax}\ w_{ay}\ w_{az}]^T$$

$$\begin{aligned}
F_t &= \begin{bmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & F_{23} & 0_{3\times3} & C_b^n \\ 0_{3\times3} & 0_{3\times3} & F_{33} & -C_b^n & 0_{3\times3} \\ & & 0_{3\times15} & & \\ & & 0_{3\times15} & & \end{bmatrix} \\
F_{23} &= \begin{bmatrix} 0 & -f_U & f_N \\ f_U & 0 & -f_E \\ -f_N & f_E & 0 \end{bmatrix} \\
F_{33} &= \begin{bmatrix} 0 & \omega sinL & -\omega cosL \\ -\omega sinL & 0 & 0 \\ \omega cosL & 0 & 0 \end{bmatrix}
\end{aligned}$$

$$B_t = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & C_b^n \\ -C_b^n & 0_{3\times3} \\ 0_{6\times3} & 0_{6\times3} \end{bmatrix}$$

观测方程

使用gps中的位置与速度进行观测，将位置与速度转换到ENU下后，有

$$\begin{aligned}
Y &= G_t X + C_t N \\
\text{观测量}\ Y &= [\delta P^T\ \delta V^T]^T \\
\text{状态量}\ X &= [\delta P^T\ \delta V^T\ \phi^T\ \varepsilon^T\ \nabla^T]^T \\
\text{观测噪声}\ N &= [nP_E\ nP_N\ nP_U\ nV_E\ nV_N\ nV_U]^T \\
G_t &= \begin{bmatrix} I_{3\times3} & 0_{3\times3} & 0_{3\times9} \\ 0_{3\times3} & I_{3\times3} & 0_{3\times9} \end{bmatrix} \\
C_t &= \begin{bmatrix} I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_{3\times3} \end{bmatrix}
\end{aligned}$$

可观测性和可观测度

代码见project_c4/vs_projects/observation_analysis目录

静止

系统是定常系统，可预测矩阵Q

$$Q=\begin{bmatrix} G \\ GF \\ \dots \\ GF^{14} \end{bmatrix}$$

为方便计算，右前上与ENU重合，$C_b^n=I$;

匀速

为方便计算，右前上与ENU重合，取其中两个时刻，但两个时刻的F_1与F_2是一样的

$$Q_{SO}=\left[\begin{array}{c} G \\ GF_1 \\ \dots \\ GF_1^{14} \\ \hline G \\ GF_2 \\ \dots \\ GF_2^{14} \end{array}\right]$$

转向

为方便计算，右前上与ENU重合，沿U轴匀速转动90度，取前后两个数据

$$Q_{SO}=\left[\begin{array}{c} G \\ GF_1 \\ \dots \\ GF_1^{14} \\ \hline G \\ GF_2 \\ \dots \\ GF_2^{14} \end{array}\right]$$

加减速

为方便计算，右前上与ENU重合，往前加减速，取开始静止时、匀加速、匀减速三个时刻;

$$Q_{SO}=\left[\begin{array}{c} G \\ GF_1 \\ \dots \\ GF_1^{14} \\ \hline G \\ GF_2 \\ \dots \\ GF_2^{14} \\ \hline G \\ GF_3 \\ \dots \\ GF_3^{14} \end{array}\right]$$

代码流程

代码只是做了Q的计算与SVD分解，再根据假设的Y去分析每一项的奇异值

```
int analyze_acceleration_deceleration() {
    Matrix<double, 15, 15> F1 = make_F(Matrix3d::Identity(), 0, 0, -9.8);
    Matrix<double, 15, 15> F2 = make_F(Matrix3d::Identity(), 0, 10, -9.8);
    Matrix<double, 15, 15> F3 = make_F(Matrix3d::Identity(), 0, -10, -9.8);
    Matrix<double, 6, 15> G = Matrix<double, 6, 15>::Zero();
    G.block<6, 6>(0, 0) = Matrix<double, 6, 6>::Identity();

    Matrix<double, 90 * 3, 15> Q = Matrix<double, 90 * 3, 15>::Zero();
    for (int i = 0; i < 15; ++i) {
        Q.block<6, 15>(i * 6, 0) = G;
        for (int j = 0; j < i; ++j) {
            Q.block<6, 15>(i * 6, 0) *= F1;
        }
        Q.block<6, 15>(90 + i * 6, 0) = G;
        for (int j = 0; j < i; ++j) {
            Q.block<6, 15>(90 + i * 6, 0) *= F2;
        }
        Q.block<6, 15>(180 + i * 6, 0) = G;
        for (int j = 0; j < i; ++j) {
            Q.block<6, 15>(180 + i * 6, 0) *= F3;
        }
    }

    JacobiSVD<MatrixXd> svd(Q, ComputeThinU | ComputeThinV);
    MatrixXd U = svd.matrixU();
    MatrixXd V = svd.matrixV();
    MatrixXd sigma = svd.singularValues();

    Matrix<double, 90*3, 1> Y = Matrix<double, 90*3, 1>::Ones() * 0.1;

    std::vector<double> sorted_sigma(15, 0.);
    for (int i = 0; i < 15; ++i) {
        MatrixXd Xi = U.col(i).transpose() * Y * V.col(i) / svd.singularValues()(i);
        int max_index = get_max_index(Xi, 15);
        sorted_sigma[max_index] = svd.singularValues()(i);
    }
    std::cout << "acceleration/deceleration:" << std::endl;
    for (int i = 0; i < 15; ++i) {
        std::cout << "index[" << i << "]=" << sorted_sigma[i] << std::endl;
    }
    return 0;
}
```

结果分析

状态变量	静止	匀速	转向	加减速
δP_E	1	1.41421	1.41421	1.73205
δP_N	1	1.41421	1.41421	1.73205
δP_U	1	1.41421	1.41421	1.73205
δV_E	1.41421	2	2	2.44949
δV_N	1.41421	2	2	2.44949
δV_U	1.41421	2	2	2.44949
ϕ_E	13.9313	19.7018	19.6512	31.3016
ϕ_N	13.9313	19.7018	19.6512	24.1296
ϕ_U	0	0	0.00050	20
ε_x	13.8593	19.6	19.6	31.2448
ε_y	13.8593	19.6	19.6	24.005
ε_z	0.00050	0.00071	0.00071	20
∇_x	0	0	1.41053	0
∇_y	0	0	1.41053	1.56509
∇_z	1.41421	2	2	2.44949

静止：加速度计水平方向零偏不可观测，U向角度误差不可观测，天向陀螺仪bias观测性差；

匀速：与静止类似，原因可能是没考虑ENU中的姿态随运动的变化；

转向：U向角度误差观测性差，天向陀螺仪bias观测性差；

加减速：右方的加速度计bias观测性差，原因是我只向前加减速；

数据仿真

使用gnss-ins-sim，定义imu模型为'high-accuracy'，打开GPS

demo_abao_c4_gps.py

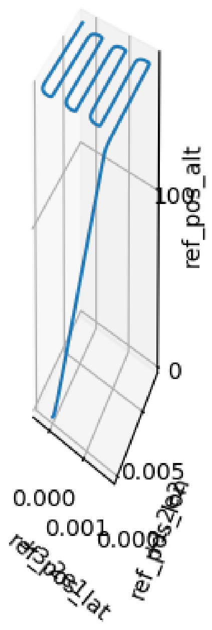
```python
# -*- coding: utf-8 -*-

import os
import math
import numpy as np
from gnss_ins_sim.sim import imu_model
from gnss_ins_sim.sim import ins_sim

# globals
D2R = math.pi/180

motion_def_path = os.path.abspath('.//demo_motion_def_files//')
fs = 100.0          # IMU sample frequency
gps_fs = 10.0
```

```python
def test_motion():
    '''
    An imu-gps demo for Sim.
    '''
    # do not generate GPS and magnetometer data
    imu = imu_model.IMU(accuracy='high-accuracy', axis=6, gps=True)

    #### start simulation
    sim = ins_sim.Sim([fs, gps_fs, 0.0],
                      motion_def_path+"//motion_def_c4_gps.csv",
                      ref_frame=0,
                      imu=imu,
                      mode=None,
                      env=None,
                      algorithm=None)
    sim.run()
    # generate simulation results, summary, and save data to files
    sim.results('')  # save data files
    # plot data
    #sim.plot(['ref_pos'], opt={'ref_pos': '3d'})

if __name__ == '__main__':
    test_motion()
```

运动使用自带的motion_def.csv，将gps打开

demo_motion_def_files/motion_def_c4_gps.csv

```
ini lat (deg),ini lon (deg),ini alt (m),ini vx_body (m/s),ini vy_body (m/s),ini vz_body (m/s),ini yaw (deg),ini pitch (deg),ini roll (deg)
32,120,0,0,0,0,0,0,0
command type,yaw (deg),pitch (deg),roll (deg),vx_body (m/s),vy_body (m/s),vz_body (m/s),command duration (s),GPS visibility
1,0,0,0,0,0,0,10,1
5,0,45,0,10,0,0,250,1
1,0,0,0,0,0,0,10,1
3,90,-45,0,0,0,0,25,1
1,0,0,0,0,0,0,50,1
3,180,0,0,0,0,0,25,1
1,0,0,0,0,0,0,50,1
3,-180,0,0,0,0,0,25,1
1,0,0,0,0,0,0,50,1
3,180,0,0,0,0,0,25,1
1,0,0,0,0,0,0,50,1
3,-180,0,0,0,0,0,25,1
1,0,0,0,0,0,0,50,1
3,180,0,0,0,0,0,25,1
1,0,0,0,0,0,0,50,1
3,-180,0,0,0,0,0,25,1
1,0,0,0,0,0,0,50,1
5,0,0,0,0,0,0,10,1
```



生成仿真数据

```
python demo_abao_c4_gps.py
```

kalman滤波

代码见project_c4/vs_projects/imu_gps_fusion目录

主要流程为imu_gps_eskf.cpp中的processImu与processGps

```
bool ImuGpsEskf::ProcessImu(const imu_t& imu) {
    if (!initialized_)
        return false;

    if (!has_last_imu_) {
        has_last_imu_ = true;
        last_imu_ = imu;
        return false;
    }

    const double delta_t = imu.time - last_imu_.time;
    const double delta_t2 = delta_t * delta_t;

    State last_state = state_;

    // Acc and gyro.
    const Eigen::Vector3d last_acc = Eigen::Vector3d(last_imu_.ax, last_imu_.ay,
last_imu_.az);
    const Eigen::Vector3d curr_acc = Eigen::Vector3d(imu.ax, imu.ay, imu.az);
    const Eigen::Vector3d last_gyro = Eigen::Vector3d(last_imu_.gx, last_imu_.gy,
last_imu_.gz);
    const Eigen::Vector3d curr_gyro = Eigen::Vector3d(imu.gx, imu.gy, imu.gz);
    const Eigen::Vector3d acc_unbias = 0.5 * (last_acc + curr_acc) -
last_state.acc_bias;
```

```
    const Eigen::Vector3d gyro_unbias = 0.5 * (last_gyro + curr_gyro) -
last_state.gyro_bias;

    // Normal state
    state_.G_P = last_state.G_P + last_state.G_v * delta_t +
        0.5 * (last_state.G_R * acc_unbias + gravity_) * delta_t2;
    state_.G_v = last_state.G_v + (last_state.G_R * acc_unbias + gravity_) * delta_t;
    const Eigen::Vector3d delta_angle_axis = gyro_unbias * delta_t;
    if (delta_angle_axis.norm() > 1e-12) {
        state_.G_R = last_state.G_R * Eigen::AngleAxisd(delta_angle_axis.norm(),
delta_angle_axis.normalized()).toRotationMatrix();
    }

    // \dot X = F_tX + B_tW

    // F_{k-1} = I + F_tT
    const double omega = 7.292115e-5;
    const double L = 32 * kDegreeToRadian;
    Eigen::Matrix<double, 15, 15> Ft = Eigen::Matrix<double, 15, 15>::Zero();
    Ft.block<3, 3>(0, 3) = Eigen::Matrix3d::Identity();
    Ft.block<3, 3>(3, 6) = -state_.G_R * GetSkewMatrix(acc_unbias); // F23
    Ft(6, 7) = omega * sin(L);  // F33
    Ft(6, 8) = -omega * cos(L);
    Ft(7, 6) = -omega * sin(L);
    Ft(8, 6) = omega * cos(L);
    Ft.block<3, 3>(3, 12) = state_.G_R;
    Ft.block<3, 3>(6, 9) = -state_.G_R;
    Eigen::Matrix<double, 15, 15> Fk_1 = Eigen::Matrix<double, 15, 15>::Identity() +
Ft * delta_t;

    // B_{k-1} = B_tT
    Eigen::Matrix<double, 15, 6> Bt = Eigen::Matrix<double, 15, 6>::Zero();
    Bt.block<3, 3>(3, 3) = state_.G_R;
    Bt.block<3, 3>(6, 0) = -state_.G_R;
    Eigen::Matrix<double, 15, 6> Bk_1 = Bt * delta_t;

    Eigen::Matrix<double, 6, 6> Qk = Eigen::Matrix<double, 6, 6>::Zero();
    Qk.block<3, 3>(0, 0) = delta_t2 * gyro_noise_ * Eigen::Matrix3d::Identity();
    Qk.block<3, 3>(3, 3) = delta_t2 * acc_noise_ * Eigen::Matrix3d::Identity();

    // update P
    state_.cov = Fk_1 * last_state.cov * Fk_1.transpose() + Bk_1 * Qk *
Bk_1.transpose();

    last_imu_ = imu;
    return true;
}

bool ImuGpsEskf::ProcessGps(const gps_t& gps) {
    // observation
    // Yk = [dPx, dPy, dPz, dVx, dVy, dVz]
    Eigen::Matrix<double, 6, 1> Yk;
    Yk.block<3, 1>(0, 0) = Eigen::Vector3d(gps.px, gps.py, gps.pz) - state_.G_P;
    Yk.block<3, 1>(3, 0) = Eigen::Vector3d(gps.vx, gps.vy, gps.vz) - state_.G_v;

    // K_k = P_kG_k^T(G_kP_kG_k^T + C_KR_kC_k^T)^{-1}
    const Eigen::MatrixXd& Pk = state_.cov;
    Eigen::Matrix<double, 6, 15> Gk = Eigen::Matrix<double, 6, 15>::Zero();
    Gk.block<3, 3>(0, 0) = Gk.block<3, 3>(3, 3) = Eigen::Matrix3d::Identity();
    Eigen::Matrix<double, 6, 6> Ck = Eigen::Matrix<double, 6, 6>::Identity();
    Eigen::Matrix<double, 6, 6> Rk = Eigen::Matrix<double, 6, 6>::Identity();
    Rk.block<3, 3>(0, 0) *= gps_pose_noise_;
    Rk.block<3, 3>(3, 3) *= gps_velocity_noise_;
    const Eigen::MatrixXd Kk = Pk * Gk.transpose() * (Gk * Pk * Gk.transpose() + Ck *
Rk * Ck.transpose()).inverse();

    // \hat{P_k} = (I-K_kG_k)\check{P_k}
    state_.cov = (Eigen::Matrix<double, 15, 15>::Identity() - Kk * Gk) * Pk;

    // \hat{x_k} = \check{x_k} + K_k(Y_k-G_k\check{X_k})
    Eigen::Matrix<double, 15, 1> Xk = Eigen::Matrix<double, 15, 1>::Zero();
    const Eigen::VectorXd delta_x = Kk * (Yk - Gk*Xk);

    // \hat{x_k} = \check{x_k} + delta_x
    state_.G_P += delta_x.block<3, 1>(0, 0);
    state_.G_v += delta_x.block<3, 1>(3, 0);
    state_.acc_bias += delta_x.block<3, 1>(9, 0);
    state_.gyro_bias += delta_x.block<3, 1>(12, 0);
    if (delta_x.block<3, 1>(6, 0).norm() > 1e-12) {
        state_.G_R *= Eigen::AngleAxisd(delta_x.block<3, 1>(6, 0).norm(),
delta_x.block<3, 1>(6, 0).normalized()).toRotationMatrix();
    }

    return true;
}
```

主函数是读取csv数据，按time先后加入imu与gps到eskf

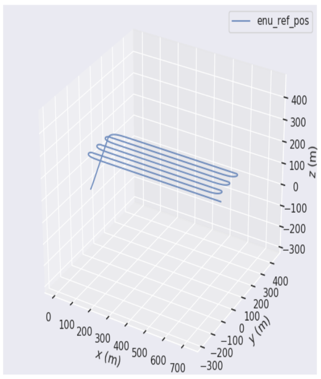
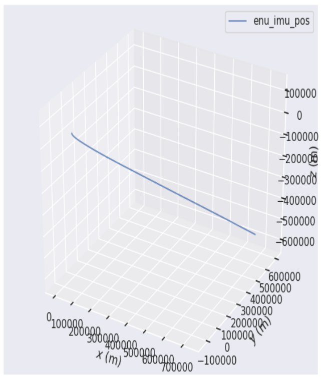
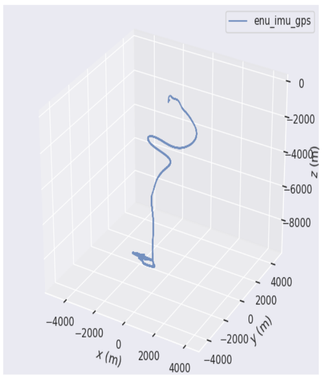
```
int imu_gps_fusion() {

    // load imu
    std::vector<double> times = load_double(path + "time.csv");
    std::vector<accel_t> accels = load_xyz<accel_t>(path + "accel-0.csv");
    std::vector<gyro_t> gyros = load_xyz<gyro_t>(path + "gyro-0.csv");
    assert(times.size() == accels.size() && times.size() == gyros.size());
    std::vector<imu_t> imus;
    imus.reserve(times.size());
    for (size_t i = 0; i < times.size(); ++i) {
        // NED->RFU
        imu_t imu = { times[i], accels[i].y, accels[i].x, -accels[i].z, gyros[i].y,
gyros[i].x, -gyros[i].z };
        imus.push_back(imu);
    }

    // load gps
    std::vector<double> gps_times = load_double(path + "gps_time.csv");
    std::vector<raw_gps_t> raw_gps = load_6d<raw_gps_t>(path + "gps-0.csv");
    assert(gps_times.size() == raw_gps.size());
    Vector3d init_lla(raw_gps[0].d[0], raw_gps[0].d[1], raw_gps[0].d[2]);
    std::vector<gps_t> gpss;
    gpss.reserve(gps_times.size());
    for (size_t i = 0; i < gps_times.size(); ++i) {
        Vector3d lla(raw_gps[i].d[0], raw_gps[i].d[1], raw_gps[i].d[2]);
        Vector3d enu;
        lla_to_enu(init_lla, lla, enu);
        // P: LLA->ENU, V: NED->ENU
        gps_t gps = { gps_times[i], enu(0), enu(1), enu(2), raw_gps[i].d[4],
raw_gps[i].d[3], -raw_gps[i].d[5]};
        gpss.push_back(gps);
    }

    // eskf
    ImuGpsEskf eskf;
    eskf.SetInitPose(Matrix4f::Identity());

    std::ofstream output(path + "enu_imu_gps.txt");
    int imu_index = 0, gps_index = 0;
    while (imu_index < imus.size() || gps_index < gpss.size()) {
        double imu_time = imu_index < imus.size() ? imus[imu_index].time :
std::numeric_limits<double>::max();
        double gps_time = gps_index < gpss.size() ? gpss[gps_index].time :
std::numeric_limits<double>::max();
        if (imu_time <= gps_time) {
            eskf.ProcessImu(imus[imu_index++]);
            save_pose(output, eskf.GetPose());
        } else {
            eskf.ProcessGps(gpss[gps_index++]);
        }
    }
    output.close();
    return 0;
}
```

但实际跑有问题，还不确定原因，不过怀疑是解算的问题，我单纯用ref_accel.csv和ref_gyro.csv解算都有问题，不知道是不是读取数据有问题，或者是坐标系转换问题？待查；

奇怪的是这个解算跟第1题是一样，第1题也没飘这么厉害。

单纯imu解算：

```
int imu_positioning() {
    std::vector<double> times = load_double(path + "time.csv");
    std::vector<accel_t> accels = load_xyz<accel_t>(path + "ref_accel.csv");
    std::vector<gyro_t> gyros = load_xyz<gyro_t>(path + "ref_gyro.csv");
    assert(times.size() == accels.size() && times.size() == gyros.size());
    std::vector<imu_t> imus;
    for (size_t i = 0; i < times.size(); ++i) {
        // Body->RFU
        imu_t imu = { times[i], accels[i].y, accels[i].x, -accels[i].z, gyros[i].y,
gyros[i].x, -gyros[i].z };
        imus.push_back(imu);
    }

    ImuGpsEskf eskf;
    eskf.SetInitPose(Matrix4f::Identity());

    std::ofstream output(path + "enu_imu.txt");
    for (const imu_t& imu : imus) {
        eskf.ProcessImu(imu);
        save_pose(output, eskf.GetPose());
    }
    output.close();
    return 0;
}
```

ref_pose	imu	imu+gps
 	 	

统计收敛速度和收敛精度

TODO