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。

答：

（1）准备数据

采用 kitti_2011_10_03_drive_0027 的数据，下载 sync 版本的所有原始数据以及 extract 版本的 oxts 数据，为了生成 100Hz 的 IMU 数据，将 sync 版本中的除 oxts 以外的其他所有数据拷贝至 extract 文件夹下。为了支持两种数据，修改 kitti2bag 相关代码，主要是以下函数

```
def run_kitti2bag():
    parser = argparse.ArgumentParser(description
    # Accepted argument values
    kitti_types = ["sync", "extract"]
```

具体相关代码的修改见附件。运行以下命令

python kitti2bag.py extract /media/zsp/zhang/downloads/dataset/kitti_2011/ -t 2011_10_03 -r 0027

即可生成用于定位的高频 bag 包。

（2）建图

运行 roslaunch lidar_localization mapping.launch 生成 filtered_map.pcd。同时将 map 原点处的经纬高信息记录以方便定位的初始化以及定位轨迹与 GNSS 真实轨迹的比较。

（3）定位

a) 相关文件

添加或修改以下文件

lidar_localization/CMakeLists.txt

lidar_localization/launch/localization.launch

lidar_localization/config/matching/kalman_filter.yaml

lidar_localization/config/matching/matching.yaml

lidar_localization/include/lidar_localization/matching/localization_flow.hpp

lidar_localization/include/lidar_localization/matching/matching.hpp

lidar_localization/src/matching/localization_flow.cpp

lidar_localization/src/matching/matching.cpp

lidar_localization/src/apps/localization_node.cpp

具体见附件。

b) 定位流程

定位的主流程由 bool LocalizationFlow::Run()函数完成，如下所示

```cpp
bool LocalizationFlow::Run() {
    if (matching_ptr_->HasNewGlobalMap() && global_map_pub_ptr_->HasSubscribers()) {
        CloudData::CLOUD_PTR global_map_ptr(new CloudData::CLOUD());
        matching_ptr_->GetGlobalMap(global_map_ptr);
        global_map_pub_ptr_->Publish(global_map_ptr);
    }

    if (matching_ptr_->HasNewLocalMap() && local_map_pub_ptr_->HasSubscribers())
        local_map_pub_ptr_->Publish(matching_ptr_->GetLocalMap());

    if (!ReadData())
        return false;

    if (!InitCalibration())
        return false;

    if (!InitGNSS())
        return false;

    if (!InitPose())
        return false;

    while(SyncData(true)) {
        Filter();
        TransformData();
        PublishData();
    }

    return true;
}
```

其中 ReadData()函数用于获取传感器数据，InitCalibration()用于获取 IMU 与雷达的外参，InitGNSS()用于初始化地图原点的经纬高信息，InitPose()用于定位初始化，SyncData(bool inited)用于同步数据，Filter()用于滤波定位，TransformData()用于激光雷达去畸变，PublishData()用于发布与保存数据。

c) 基于误差状态的滤波

采用基于误差状态的滤波模型，相关实现在 bool LocalizationFlow::Filter()函数中，如下所示

```cpp
bool LocalizationFlow::Filter()
{
    Predict();
    Correct();
    return true;
}
```

其中 Predict()函数是预测部分，Correct()函数是观测部分，为了方便滤波计算，定义了状态量和误差状态量，如下所示

```cpp
struct State
{
  Eigen::Vector3d p;
  Eigen::Vector3d v;
  Eigen::Quaterniond q;
  Eigen::Vector3d ba;
  Eigen::Vector3d bg;
};
struct ErrorState
{
  Eigen::Matrix<double, 15, 1> x;
  Eigen::Matrix<double, 15, 15> p;
};
```

预测部分完成惯性导航解算和误差状态量的先验值计算，注意惯性解算时需要补偿 IMU 零偏，如下所示

```
Eigen::Vector3d wb;
wb [0] = 0.5 * current_imu_data_[i-1].angular_velocity.x +  0.5 * current_imu_data_[i].angular_velocity.x;
wb [1] =  0.5 * current_imu_data_[i-1].angular_velocity.y +  0.5 * current_imu_data_[i].angular_velocity.y;
wb [2] =  0.5 * current_imu_data_[i-1].angular_velocity.z +  0.5 * current_imu_data_[i].angular_velocity.z;
wb = wb + state_.bg;
wb = wb * dt;
Eigen::Vector3d f1(current_imu_data_[i-1].linear_acceleration.x, current_imu_data_[i-1].linear_acceleration.y,
current_imu_data_[i-1].linear_acceleration.z);
f1 = f1 + state_.ba;
Eigen::Vector3d f2(current_imu_data_[i].linear_acceleration.x, current_imu_data_[i].linear_acceleration.y,
current_imu_data_[i].linear_acceleration.z);
f2 = f2 + state_.ba;
```

而观测则完成雷达匹配对定位的纠正，同时将误差状态量更新到状态量，需要注意 IMU 零偏应该累加，如下所示

```
state_.p = state_.p - error_state_.x.block<3, 1>(0, 0);
state_.v = state_.v - error_state_.x.block<3, 1>(3, 0);
Eigen::Vector3d dphi_dir = error_state_.x.block<3, 1>(6, 0);
double dphi_norm = dphi_dir.norm();
if (dphi_norm != 0)
{
    dphi_dir = dphi_dir / dphi_norm;
    dphi_dir = dphi_dir * std::sin(dphi_norm / 2);
}
Eigen::Quaterniond temp2(std::cos(dphi_norm / 2), dphi_dir[0], dphi_dir[1], dphi_dir[2]);
state_.q = temp2 * state_.q;
state_.bg = state_.bg - error_state_.x.block<3, 1>(9, 0);
state_.ba = state_.ba - error_state_.x.block<3, 1>(12, 0);
```

具体实现见附件。另外，在 Correct()函数中需要调用 Matching::TransformCurrentScan 函数，利用融合之后的定位信息将当前帧点云数据转换到地图坐标系下。

滤波相关的参数包括陀螺仪噪声，加速度计噪声，点云匹配观测噪声（位置和姿态），通过 kalman_filter.yaml 文件进行配置。

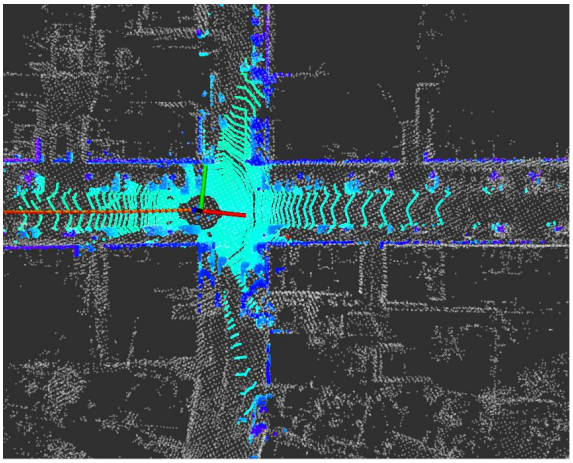
（4）定位运行结果

运行以下命令

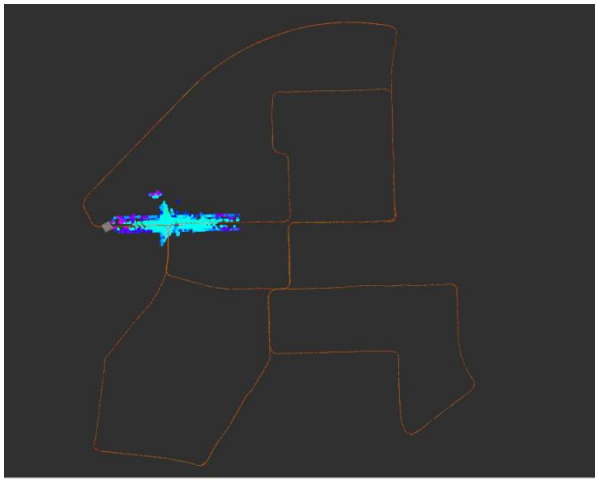
roslaunch lidar_localization localization.launch

rosbag play --clock kitti_2011_10_03_drive_0027_extract.bag

运行中的定位如下所示



最后的 GNSS 轨迹和融合定位轨迹对比如下所示



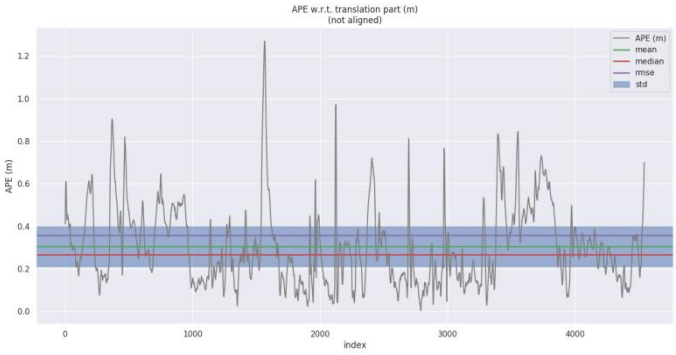
运行如下命令对比两条轨迹的差别

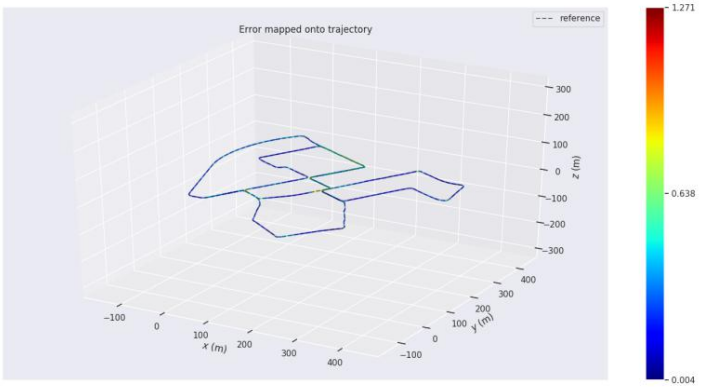
evo_ape kitti ground_truth.txt localization.txt -r full --plot --plot_mode xyz

结果如下，定位平均误差约为 0.3m。

```
APE w.r.t. translation part (m)
(not aligned)

       max      1.270810
      mean      0.305110
    median      0.266801
       min      0.004440
      rmse      0.357888
       sse      582.012715
       std      0.187061
```





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。

答：

（1）基于误差状态的组合导航(gps+imu)滤波模型推导

与上一题唯一不同的地方是观测方程只有位置观测。以下是具体公式推导过程。

状态量：

$$X = (\delta P^T, \delta V^T, \phi^T, \varepsilon^T, \nabla^T)^T$$

状态方程：

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

$$F_t = \begin{pmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & F_{23} & 0_{3\times3} & C_b^n \\ 0_{3\times3} & 0_{3\times3} & F_{33} & -C_b^n & 0_{3\times3} \\ & & 0_{3\times15} & & \\ & & 0_{3\times15} & & \end{pmatrix}$$

$$F_{23} = \begin{pmatrix} 0 & -f_U & f_N \\ f_U & 0 & -f_E \\ -f_N & f_E & 0 \end{pmatrix}$$

$$F_{33} = \begin{pmatrix} 0 & wsinL & -wcosL \\ -wsinL & 0 & 0 \\ wcosL & 0 & 0 \end{pmatrix}$$

$$W = (w_{gx}, w_{gy}, w_{gz}, w_{ax}, w_{ay}, w_{az})^T$$

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

观测方程：

$$Y = G_t X + C_t N$$

$$Y = \delta P$$

$$G_t = (I_{3\times3}, 0_{3\times12})$$

$$C_t = I_{3\times3}$$

$$N = (n_{P_E}, n_{P_N}, n_{P_U})^T$$

离散时间滤波器：

$$\breve{X}_k = F_{k-1}\widehat{X}_{k-1} + B_{k-1}W_k$$

$$\breve{P}_k = F_{k-1}\widehat{P}_{k-1}F_{k-1}^T + B_{k-1}Q_k B_{k-1}^T$$

$$K_k = \breve{P}_k G_k^T (G_k \breve{P}_k G_k^T + C_k R_k C_k^T)^{-1}$$

$$\hat{X}_k = \breve{X}_{k-1} + K_k (Y_k - G_k \breve{X}_k)$$

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

其中

$$F_{k-1} = I + F_t T$$

$$B_{k-1} = B_t T$$

$$X_k = F_{k-1} X_{k-1} + B_{k-1} W_k$$

$$Y_k = G_k X + C_k N$$

（2）代码实现

新建工程 ins，算法部分在源文件 ins.cpp 中实现，包含以下主要部分：

a) 通过以下函数读取仿真数据

```
bool ReadData(const std::vector<std::string> &path)
```

b) 通过以下函数同步 GPS 和 IMU 数据

```
bool SyncData(bool inited)
```

c) 通过以下函数初始化位姿和状态量

```
bool InitPose()
```

d) 通过以下函数进行滤波

```
bool Filter()
{
    Predict();
    if (correct)
    {
        Correct();
    }
    return true;
}
```

e) 通过以下函数保存数据以便后续分析

```
bool SaveData()
{
    SavePose(gt_ofs, current_gt);
    SavePose(pose_ofs, current_pose);
}
```

f) 通过以下函数保存观测度分析所需的 F，G 和 Y

```
bool SaveFG()
```

g) 通过以下部分计算 Qso 以及可观测度系数

```cpp
Eigen::MatrixXd Qso(3*15*FGs.size(), 15);
Eigen::MatrixXd Ys(3*15*FGs.size(), 1);
Eigen::Matrix<double, 15, 15> Fnn = Eigen::Matrix<double, 15, 15>::Identity();
for (int i = 0; i < FGs.size(); ++i)
{
    Eigen::Matrix<double, 15, 15> Fn = Eigen::Matrix<double, 15, 15>::Identity();
    for (int j = 0; j < 15; j++)
    {
        if (j > 0)
        {
            Fn = Fn * FGs[i].F;
        }
        Ys.block<3, 1>(i*3*15+3*j, 0) = FGs[i].Y[j];
        Qso.block<3, 15>(i*3*15+3*j, 0) = FGs[i].G * Fn * Fnn;
    }
    // Fnn = Fnn * Fn;
}
Eigen::JacobiSVD<Eigen::MatrixXd> svd(Qso, Eigen::ComputeFullU | Eigen::ComputeFullV);
std::cout << Qso.rows() << ", " << Qso.cols() << std::endl;
// std::cout << svd.singularValues() << std::endl;
for (int i = 0; i < 15; ++i)
{
    double temp = (svd.matrixU().row(i) * Ys)[0] / svd.singularValues()[i];
    Eigen::MatrixXd Xi = temp * svd.matrixV().col(i);
    // std::cout << Xi.transpose() << std::endl;
    Eigen::MatrixXd::Index maxRow, maxCol;
    Xi = Xi.cwiseAbs();
    double maxvalue = Xi.maxCoeff(&maxRow, &maxCol);
    std::cout << svd.singularValues()(i) / svd.singularValues()(0) << "," << maxRow << std::endl;
    sv_ofs << svd.singularValues()(i) / svd.singularValues()(0) << "," << maxRow << std::endl;
}
```

h)　参数设置

初始化的状态量误差，传感器的误差，可观测度分析的时间段个数（决定了 Qso 矩阵的大小），各分段之间的时间间隔以及滤波时长需要通过 param.yaml 文件进行配置，如下所示

```yaml
init_noise: [1e-4, 1e-4, 1e-6, 1e-6, 1e-6]
gyro_noise: 1e-5
acc_noise: 1e-4
dp_noise: 1e-6
# dv_noise: 1e-6
FGsize: 10
time_interval: 100
end_time: 2000.0
```

具体代码请见附件。

（3）仿真与分析

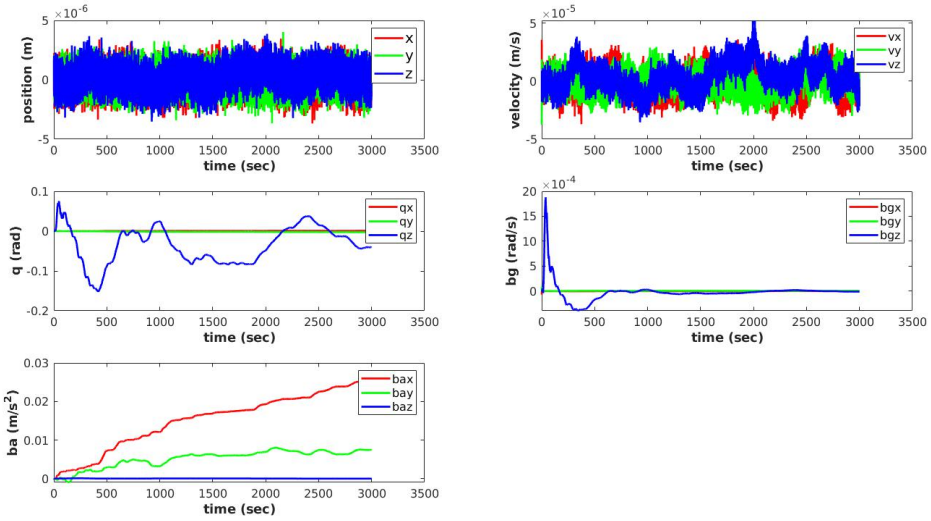
采用 gnss-ins-sim 进行仿真，包括静止，匀速，加减速和转向 4 种情况，采用 imu_data7.py 生成数据，其中 IMU 和 GPS 的参数设置如下所示，

```
# imu_err
imu_err = {'gyro_b': np.array([1e-2, 2e-2, 3e-2]) / D2R * 3600 * 0,
           'gyro_arw': np.array([1e-5, 1e-5, 1e-5]) / D2R * 60 * 0,
           'gyro_b_stability': np.array([1e-5, 1e-5, 1e-5]) / D2R * 3600 * 1e-0,
             'gyro_b_corr': np.array([100.0, 100.0, 100.0]),
           'accel_b': np.array([2.0e-3, 1.0e-3, 5.0e-3]) * 0,
           'accel_vrw': np.array([1e-4, 1e-4, 1e-4]) * 60.0 * 0,
           'accel_b_stability': np.array([1e-4, 1e-4, 1e-4]) * 1.0 * 1e0,
             'accel_b_corr': np.array([200.0, 200.0, 200.0]),
          #    'mag_std': np.array([0.2, 0.2, 0.2]) * 1.0
           }
# generate GPS and magnetometer data
gps_err = {
    'stdp': np.array([1, 1, 1]) * 1e-6,
    'stdv': np.array([0, 0, 0]),
}
```

其中陀螺仪零偏稳定性为 1e-5rad/s（约 2deg/h），加速度计零偏稳定性为 1e-4m/s^2，GPS 位置误差为 1e-6m/s，其它误差量均设置为 0。运行 ins 生成数据后，采用 plot_data.m 来绘制各状态量和真实值的数据，采用 plot_data2.m 来绘制各状态量与真实值相减之后的数据，以分析收敛情况，具体代码见附件。在可观测度分析中，各状态量对应的编号从 0 到 14。

a) 静止状态

仿真命令参见附件文件 imu_def7.csv。各状态量的收敛情况见下图，可观测度系数见下表。从图中可看出，航向角，x 轴加速度零偏和 y 轴加速度零偏均未收敛，而这 3 个状态量对应的可观测度系数均非常小，在 1e-34 到 1e-27 量级，因此 3 个状态量是不可观的。另外，z 轴角速度零偏的收敛速度很慢，直到 10 分钟后才逐渐收敛，它对应的可观测度也很低，在 6e-8 左右，由于其仍然能够收敛，因此认为它是可观的。

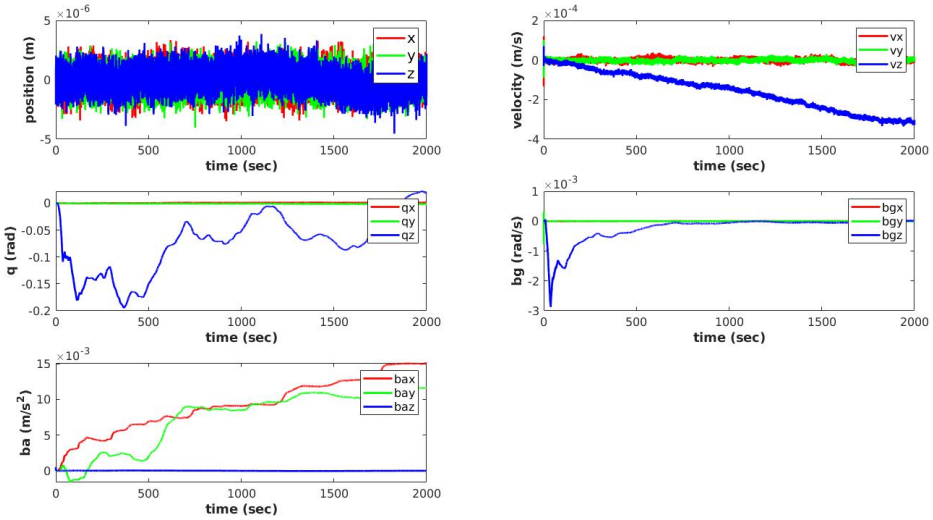


误差值（滤波值减去真实值）

可观测度	1	1	1	0.1	0.1	0.1	0.098457	0.098457	0.01	0.0097985	0.0097985	6.04E-08	1.14E-27	5.02E-29	3.30E-34
状态量	0	1	2	3	4	5	7	6	14	9	10	11	8	13	12

b) 匀速状态

仿真命令参见附件文件 imu_def8.csv ,b 系沿前向(y 轴正向)做匀速运动。各状态量的收敛情况见下图，可观测度系数见下表。从图中可看出，收敛情况与静止状态完全相同，航向角，x 轴加速度零偏和 y 轴加速度零偏均未收敛，这 3 个状态量对应的可观测度系数均很小，在 1e-33 到 1e-28 量级，因此 3 个状态量是不可观的。z 轴角速度零偏收敛速度很慢，对应的可观测度也很低，在 6e-8 左右，但仍然能够收敛，因此是可观的。

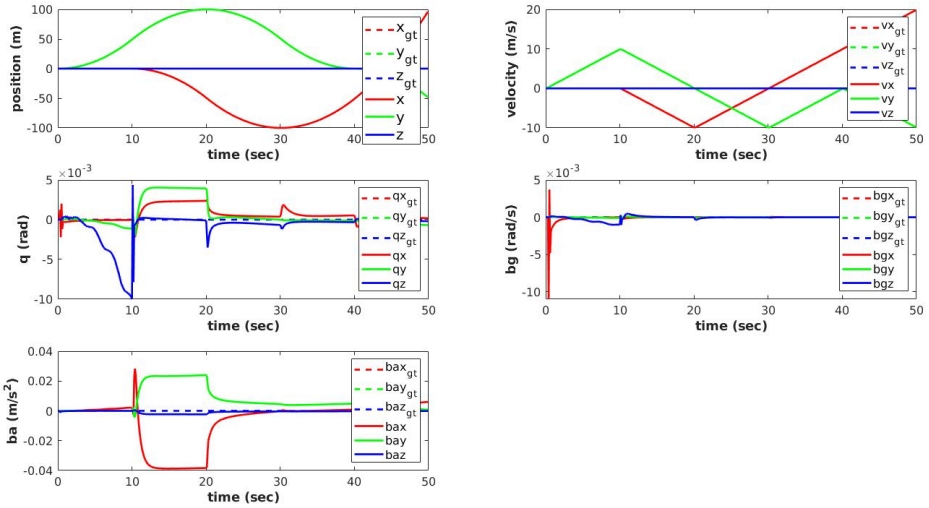


误差值（滤波值减去真实值）

可观测度	1	1	1	0.1	0.1	0.1	0.098457 6	0.098457 6	0.01	0.009794 84	0.009794 84	6.04 E-08	1.00 E-28	4.88 E-30	7.19 E-33
状态量	0	1	2	3	4	5	7	6	14	9	10	11	13	12	8

c) 加减速状态

仿真命令参见附件文件 imu_def9.csv，b 系沿 x 轴和 y 轴两个方向分别做加减速运动，具体运动参考下图中的位置和速度变化曲线。可观测度系数见下表。各状态量对应的可观测度系数均在 7.9e-4 以上，而图中各状态量的误差值均在 0 值附近波动，可认为达到收敛状态。但由于 IMU 数据存在误差，各状态量均存在一定程度的误差。在加减速变换时，角度和零偏的误差值均有一定波动。可观测度系数中最小的 3 个是 z 轴角速度零偏，x 轴加速度零偏，y 轴加速度零偏，均在 1e-3 量级，3 个状态量观测度较低，而收敛速度也较慢，特别是 2 个加速度水平零偏，在 20s 之后误差值才逐渐减小，因此对应的可观测度也最低。航向角的误差值在几次加减速后逐渐下降了一个数量级，对应的可观测度较前面 2 种情况也有大幅提高，约 0.012。

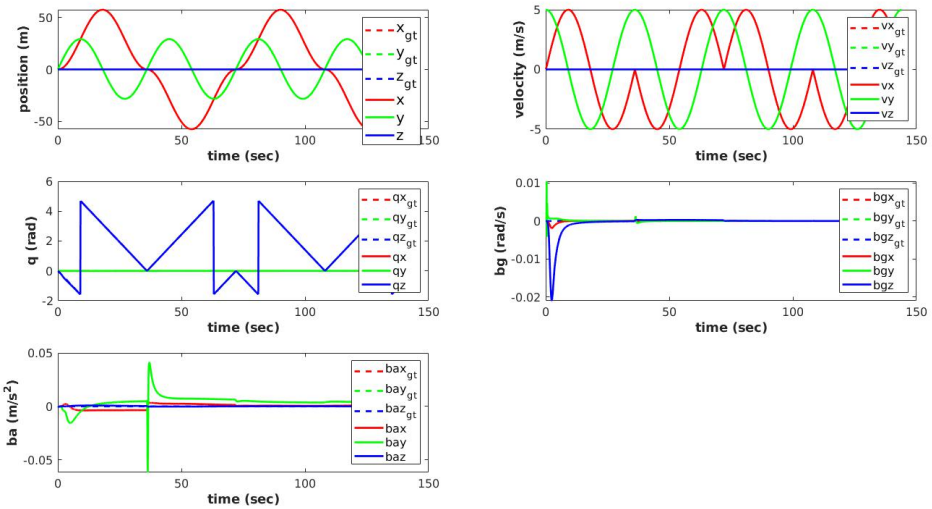


真实值与滤波值

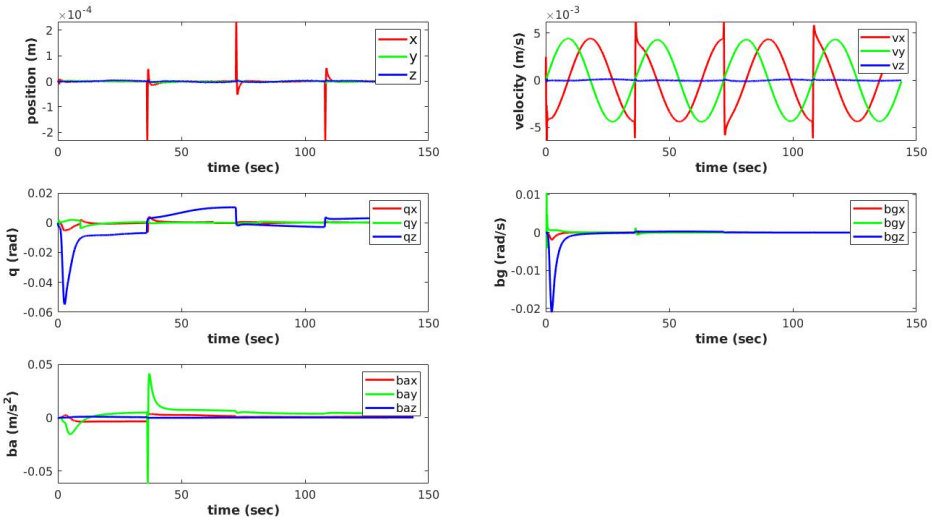
可观测度	1	1	1	0.1	0.1	0.1	0.098968	0.098751	0.012305	0.000999 95	0.000984522	9.83E-03	1.24E-03	9.68E-04	7.93E-04
状态量	0	1	2	3	4	5	6	7	8	14	9	10	11	13	12

d) 转向状态

仿真命令参见附件文件 imu_def10.csv，b 系做绕 8 字运动，具体运动情况见下图的位置和姿态变化。各状态量可观测度系数见下表。各状态量对应的可观测度系数均在 8.7e-4 以上，而图中各状态量的误差值均在 0 值附近波动，可认为达到收敛状态。但由于 IMU 数据存在误差，各状态量均存在一定程度的误差。在做转向变换时，角度和零偏的误差值均有一定波动。可观测度系数小于或者等于 0.01 量级的有 7 个状态量，包括航向角和 6 个零偏参数，这 7 个参数均在开始有所波动，但经过几十秒后收敛。



真实值与滤波值



误差值（滤波值减去真实值）

可观测度	1	1	1	0.1	0.1	0.1	0.098143 4	0.098135 9	0.010 0003	0.01	0.009999 58	9.83 E-03	9.79 E-03	8.69 E-03	8.68 E-04
状态量	0	1	2	3	4	5	6	7	13	14	12	10	9	8	11