补充代码部分

SetProcessEquation

```
// TODO: set process / system equation:
// a. set process equation for delta vel:

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

UpdateOdomEstimation 基本和第六章的作业一样

```
size_t index_curr = 1;
size_t index_prev = 0;
// get deltas:
Eigen::Vector3d angular_delta = Eigen::Vector3d::Zero();
GetAngularDelta(index_curr, index_prev, angular_delta, angular_vel_mid);
// update orientation:
Eigen::Matrix3d R_curr = Eigen::Matrix3d::Zero();
Eigen::Matrix3d R_prev = Eigen::Matrix3d::Zero();
UpdateOrientation(angular_delta, R_curr, R_prev);

// get velocity delta:
double T;
Eigen::Vector3d velocity_delta = Eigen::Vector3d::Zero();
GetVelocityDelta(index_curr, index_prev, R_curr, R_prev, T, velocity_delta, linear_a

// save mid-value unbiased linear acc for error-state update:

// update position:
UpdatePosition(T, velocity_delta);
```

UpdateErrorEstimation 对应ppt中状态方程离散化,注意其中w_k为零均值，故w_k为0

```
    // TODO: update process equation:
    UpdateProcessEquation(linear_acc_mid, angular_vel_mid);
    // TODO: get discretized process equations:
    F_1st = F_ * T;
    F_2nd = MatrixF::Identity() + F_1st;

    MatrixB B = MatrixB::Zero();
    B.block<3, 3>(kIndexErrorVel, kIndexNoiseAccel) = B_.block<3, 3>(kIndexErrorVel, kI
    B.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) = B_.block<3, 3>(kIndexErrorOri, kIn
    B.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = B_.block<3, 3>(kIndexErrorA
    B.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = B_.block<3, 3>(kIndexErrorGyr
    // TODO: perform Kalman prediction
    X_ = F_2nd * X_;
    P_ = F_2nd * P_ * F_2nd.transpose() + B * Q_ * B.transpose();
```

CorrectErrorEstimationPose 对应观察方程计算。这里直接根据ppt中计算后，初始化Y,计算K

```
    Eigen::Vector3d delta_p = pose_.block<3, 1>(0, 3) - T_nb.block<3, 1>(0, 3);
    Eigen::Matrix3d delta_R = T_nb.block<3, 3>(0, 0).transpose() * pose_.block<3, 3>(0,
    Eigen::Vector3d delta_ori = Sophus::SO3d::vee(delta_R - Eigen::Matrix3d::Identity())

    YPose_.block<3, 1>(0, 0) = delta_p;
    YPose_.block<3, 1>(3, 0) = delta_ori;

    Y = YPose_;
    // TODO: set measurement equation:
    G = GPose_;

    // TODO: set Kalman gain:
    K = P_ * G.transpose() * (G * P_ * G.transpose() + CPose_ * RPose_ * CPose_.transpos
```

然后计算P与X

```
    P_ = (MatrixP::Identity() - K * G) * P_;
    X_ = X_ + K * (Y - G * X_);
```

EliminateError 对应ppt中的更新后验位姿

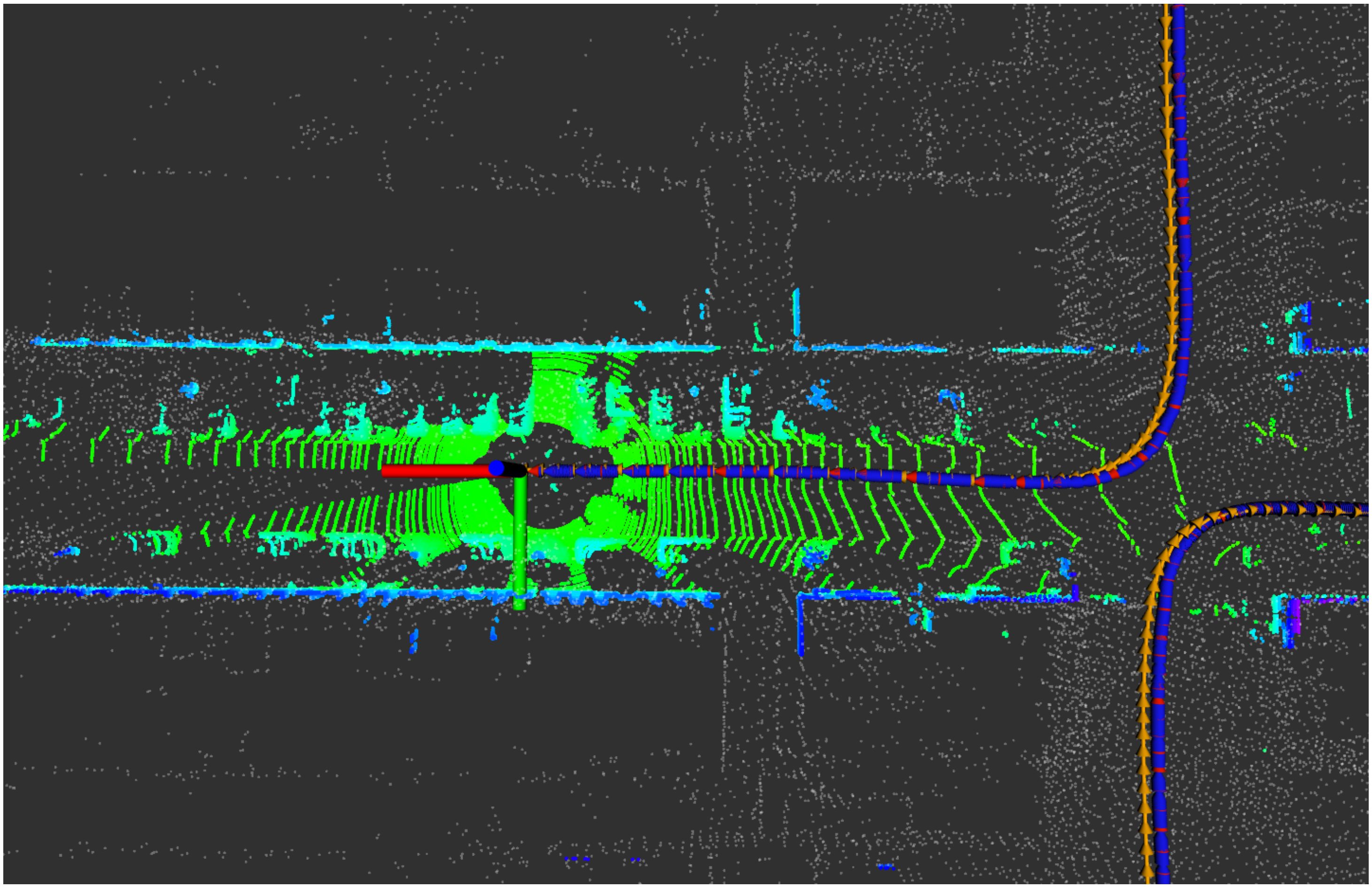
```
    // a. position:
    // do it!
    pose_.block<3, 1>(0, 3) -= X_.block<3, 1>(kIndexErrorPos, 0);
    // b. velocity:
    // do it!
    vel_ -= X_.block<3, 1>(kIndexErrorVel, 0);
    // c. orientation:
    // do it!
    pose_.block<3, 3>(0, 0) = pose_.block<3, 3>(0, 0) * (Eigen::Matrix3d::Identity() - S

    Eigen::Quaterniond q(pose_.block<3, 3>(0, 0));
    q.normalize();
    pose_.block<3, 3>(0, 0) = q.toRotationMatrix();

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

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

效果如下



不考虑随机游走

定义bias_flag,然后在构造函数做如下修改:

```
// c. process noise:
Q_.block<3, 3>(kIndexNoiseAccel, kIndexNoiseAccel) = COV.PROCESS.ACCEL * Eigen::Matr
Q_.block<3, 3>(kIndexNoiseGyro, kIndexNoiseGyro) = COV.PROCESS.GYRO * Eigen::Matrix3
if (bias_flag) {
    Q_.block<3, 3>(kIndexNoiseBiasAccel, kIndexNoiseBiasAccel) = COV.PROCESS.BIAS_AC
    Q_.block<3, 3>(kIndexNoiseBiasGyro, kIndexNoiseBiasGyro) = COV.PROCESS.BIAS_GYR
}
// d. measurement noise:
RPose_.block<3, 3>(0, 0) = COV.MEASUREMENT.POSE.POSI * Eigen::Matrix3d::Identity();
RPose_.block<3, 3>(3, 3) = COV.MEASUREMENT.POSE.ORI * Eigen::Matrix3d::Identity();

// e. process equation:
F_.block<3, 3>(kIndexErrorPos, kIndexErrorVel) = Eigen::Matrix3d::Identity();
F_.block<3, 3>(kIndexErrorOri, kIndexErrorGyro) = -Eigen::Matrix3d::Identity();

B_.block<3, 3>(kIndexErrorOri, kIndexNoiseGyro) = Eigen::Matrix3d::Identity();
if (bias_flag) {
    B_.block<3, 3>(kIndexErrorAccel, kIndexNoiseBiasAccel) = Eigen::Matrix3d::Identi
    B_.block<3, 3>(kIndexErrorGyro, kIndexNoiseBiasGyro) = Eigen::Matrix3d::Identity
}
```

对比odometry数据

```
evo_rpe kitti ground_truth.txt fused.txt -r trans_part --delta 100 --plot --plot_mode xy
```

对比rmse.考虑随机游走：2.687764，不考虑随机游走：2.396791。从结果来看不考虑随机游走误差小些，当然也跟kitti数据有关，任老师说过实际中可以都尝试下，那个优用那个