Imu 模型仿真_中值法

1. Summary

（1）修改程序：减去噪声，发现拟合误差减小；
（2）修改程序：减去噪声，并且改为中值法，发现拟合误差非常小；
（3）原程序中似乎有一个数据数组的下标对齐的问题；
（4）原程序的 pos 和 eular-angle 的关系的分析：运行轨迹完全由 position 确定，而 eular-angle 只是表示 body 系的位姿，不是运动的方向；
eular-angle 只是用来将位姿在 world 系和 body 系之间转换；

2. Code

（1）原始 code：course2_hw_new\vio_data_simulation
（2）修改的程序为：imu.cpp

```
//读取生成的 imu 数据并用 imu 动力学模型对数据进行计算，最后保存 imu 积分以后的轨迹，
//用来验证数据以及模型的有效性。
void IMU::testImu(std::string src, std::string dist)
{
    std::vector<MotionData>imudata;
    LoadPose(src,imudata);

    std::ofstream save_points;
    save_points.open(dist);

    double dt = param_.imu_timestep;
    Eigen::Vector3d Pwb = init_twb_;              // position :    from  imu measurements
    Eigen::Quaterniond Qwb(init_Rwb_);            // quaterniond:  from imu measurements
    Eigen::Vector3d Vw = init_velocity_;          // velocity  :   from imu measurements
    Eigen::Vector3d gw(0,0,-9.81);    // ENU frame
    Eigen::Vector3d temp_a;
    Eigen::Vector3d theta;
```

```
    //for (int i = 1; i < imudata.size(); ++i) { //i:k =>i+1:k+1 //orig;
    for (int i = 0; i < imudata.size()-1; ++i) { //i:k =>i+1:k+1  中值法

        MotionData imupose = imudata[i];
        //MotionData imupose_k1 = imudata[i-1];  //for (int i = 1; i < imudata.size()-1; ++i) {
        MotionData imupose_k1 = imudata[i+1];

        //delta_q = [1 , 1/2 * thetax , 1/2 * theta_y, 1/2 * theta_z]
        Eigen::Quaterniond dq;
        //Eigen::Vector3d dtheta_half =  imupose.imu_gyro * dt /2.0; //orig
        //1/2 * w * dt(delta t);   wbk: imupose.imu_gyro;  bkg: imupose.imu_gyro_bias;
        //Eigen::Vector3d dtheta_half =  (imupose.imu_gyro-imupose.imu_gyro_bias) * dt /2.0; //-bias;  欧拉法;

        Eigen::Vector3d dtheta_half=
               ( 0.5*(imupose.imu_gyro-imupose.imu_gyro_bias+imupose_k1.imu_gyro-imupose.imu_gyro_bias) )*dt/2.0; //-bias;  中值法


        dq.w() = 1;
        dq.x() = dtheta_half.x();
        dq.y() = dtheta_half.y();
        dq.z() = dtheta_half.z();
        dq.normalize();

        Eigen::Quaterniond Qwb_k1 = Qwb * dq;

        /// imu 动力学模型 欧拉积分
        //Eigen::Vector3d acc_w = Qwb * (imupose.imu_acc) + gw;  // aw = Rwb * ( acc_body - acc_bias ) + gw  //orig

        //Eigen::Vector3d acc_w = Qwb * (imupose.imu_acc-imupose.imu_acc_bias) + gw;  // aw = Rwb * ( acc_body - acc_bias ) + gw  //
欧拉法

        Eigen::Vector3d acc_w = 0.5 *( Qwb * (imupose.imu_acc-imupose.imu_acc_bias) + gw +
                                Qwb_k1 * (imupose_k1.imu_acc-imupose.imu_acc_bias) + gw );// 中值法
```

```
        Qwb = Qwb * dq;
        Pwb = Pwb + Vw * dt + 0.5 * dt * dt * acc_w;
        Vw = Vw + acc_w * dt;

        /// 中值积分

        //  按着 imu postion, imu quaternion , cam postion, cam quaternion 的格式存储，由于没有 cam，所以 imu 存了两次
        save_points<<imupose.timestamp<<" "
                   <<Qwb.w()<<" "
                   <<Qwb.x()<<" "
                   <<Qwb.y()<<" "
                   <<Qwb.z()<<" "
                   <<Pwb(0)<<" "
                   <<Pwb(1)<<" "
                   <<Pwb(2)<<" "
                   <<Qwb.w()<<" "
                   <<Qwb.x()<<" "
                   <<Qwb.y()<<" "
                   <<Qwb.z()<<" "
                   <<Pwb(0)<<" "
                   <<Pwb(1)<<" "
                   <<Pwb(2)<<" "
                   <<std::endl;

    }

    std::cout<<"test  end"<<std::endl;

}
```

3.results

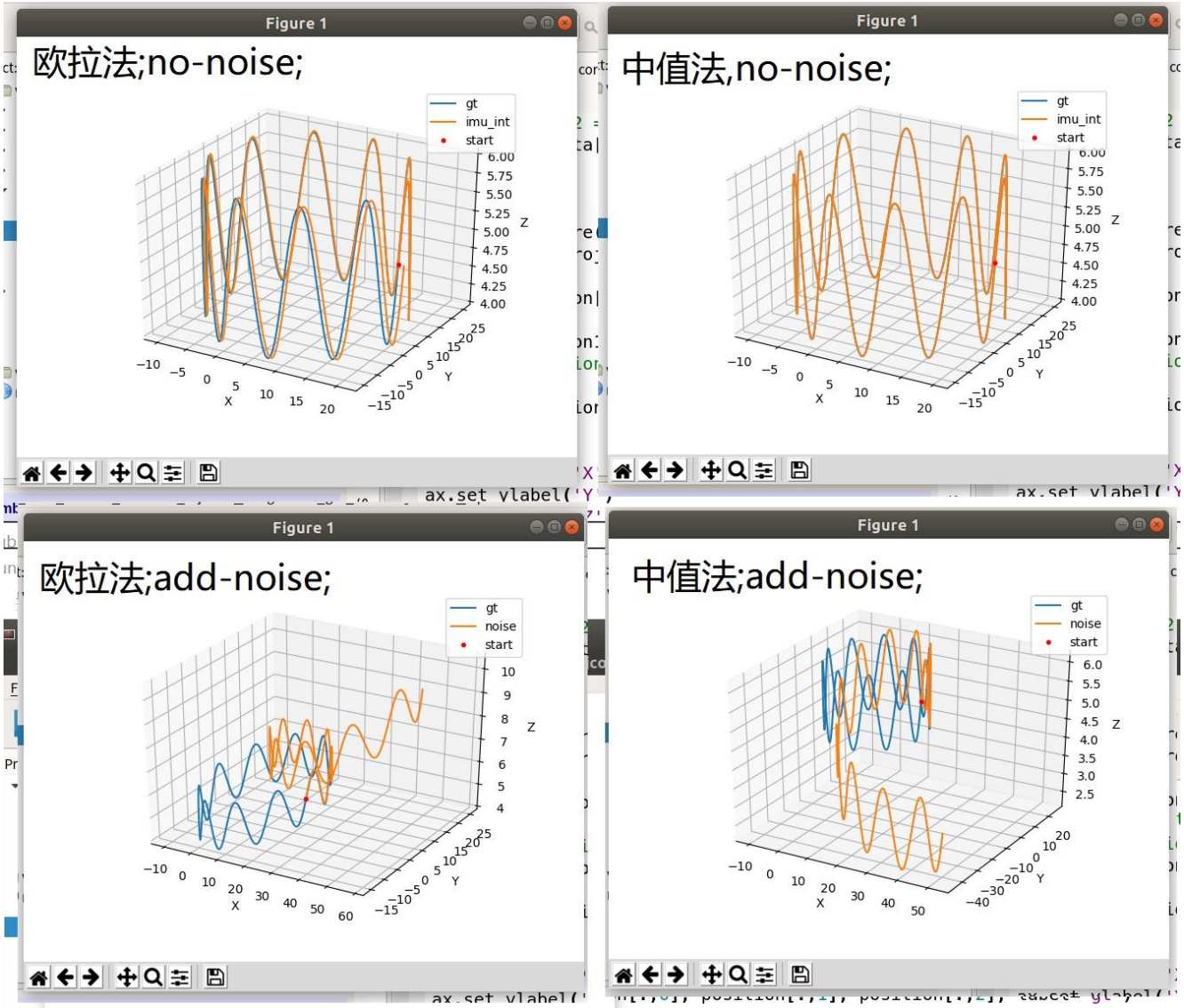
Figure 1
欧拉法;no-noise;
gt
imu_int
start
Figure 1
中值法,no-noise;
gt
imu_int
start
Figure 1
欧拉法;add-noise;
gt
noise
start
Figure 1
中值法;add-noise;
gt
noise
start