第9章作业

一. 及格和良好作业标准

补全预积分的相关公式

预积分的残差如下:

$$\begin{bmatrix} r_p \\ r_q \\ r_v \\ r_{ba} \\ r_{bg} \end{bmatrix} = \begin{bmatrix} q^*_{wb_i}(p_{wb_j} - p_{wb_i} - v^w_i \Delta t + \frac{1}{2} g^w \Delta t^2) - \alpha_{b_i b_j} \\ 2[q^*_{b_i b_j} \bigotimes (q^*_{wb_i} \bigotimes q_{wb_j})]_{xyz} \\ q^*_{wb_i}(v^w_j - v^w_i + g^w \Delta t) - \beta_{b_i b_j} \\ b^a_j - b^a_i \\ b^g_j - b^g_i \end{bmatrix}$$

接下来写出预积分残差对于各个变量的雅可比

姿态残差的雅可比

对i时刻姿态的雅可比

$$\frac{\partial r_q}{\partial \delta \theta_{b_i b_i'}} = -2 \begin{bmatrix} 0 & I \end{bmatrix} [q^*_{wb_j} \bigotimes q_{wb_I}]_L [q_{b_i b_j}]_R \begin{bmatrix} 0 \\ \frac{1}{2} I \end{bmatrix}$$

对j时刻姿态的雅可比

$$\frac{\partial r_q}{\partial \delta \theta_{b_j b_j'}} = -2 \begin{bmatrix} 0 & I \end{bmatrix} [q^*_{wb_j} \bigotimes q_{wb_I}]_L [q_{b_i b_j}]_R \begin{bmatrix} 0 \\ \frac{1}{2} I \end{bmatrix}$$

对i时刻陀螺仪bias偏差的雅可比

$$\frac{\partial r_q}{\partial \delta b^g_i} = -2 \begin{bmatrix} 0 & I \end{bmatrix} [q^*_{wb_j} \bigotimes q_{wb_i} \bigotimes q_{b_i b_j}]_L \begin{bmatrix} 0 \\ \frac{1}{2} J^q_{b^g_i} \end{bmatrix}$$

速度残差的雅可比

对i时刻姿态的雅可比

$$\frac{\partial r_v}{\partial \delta \theta_{b_i b_i'}} = [R_{b_i w}(v^w_j - v^w_i + g^w \Delta t)]_\times$$

对i时刻速度的雅可比

$$\frac{\partial r_v}{\partial \delta v^w_i} = -R_{wb_i}$$

对j时刻速度的雅可比

$$\frac{\partial r_v}{\partial \delta v^w_j} = R_{wb_i}$$

对i时刻加速度计bias的雅可比

$$\frac{\partial r_v}{\partial \delta b^a_i} = -\frac{\partial \beta_{b_i b_j}}{\partial \delta b^a_i} = -J^\beta_{b^a_i}$$

对i时刻陀螺仪bias的雅可比

$$\frac{\partial r_v}{\partial \delta b_i^g} = -\frac{\partial \beta_{b_i b_j}}{\partial \delta b_i^g} = -J_{b_i^g}^{\beta}$$

位置残差的雅可比

对i时刻姿态的雅可比

$$\frac{\partial r_p}{\partial \delta \theta_{b_i b_i'}} = [R_{b_i w}(p_{wb_j} - p_{wb_i} - v_i^w \Delta t + \frac{1}{2} g^w \Delta t^2)]_\times$$

对i时刻速度的雅可比

$$\frac{\partial r_p}{\partial \delta v_i^w} = -R_{wb_i} \Delta t$$

对i时刻位置的雅可比

$$\frac{\partial r_p}{\partial \delta p_i^w} = -R_{wb_i}$$

对j时刻位置的雅可比

$$\frac{\partial r_p}{\partial \delta p_j^w} = R_{wb_i}$$

对i时刻加速度计bias的雅可比

$$\frac{\partial r_p}{\partial \delta b_i^a} = -\frac{\partial \alpha_{b_i b_j}}{\partial \delta b_i^a} = -J_{b_i^a}^{\alpha}$$

对i时刻陀螺仪bias的雅可比

$$\frac{\partial r_p}{\partial \delta b_i^g} = -\frac{\partial \alpha_{b_i b_j}}{\partial \delta b_i^g} = -J_{b_i^g}^{\alpha}$$

加速度计残差的雅可比

对i时刻加速度计bias的雅可比

$$\frac{\partial r_{ba}}{\partial \delta b_i^a} = -I$$

对j时刻加速度计bias的雅可比

$$\frac{\partial r_{ba}}{\partial \delta b_j^a} = I$$

陀螺仪残差的雅可比

对i时刻陀螺仪bias的雅可比

$$\frac{\partial r_{bg}}{\partial \delta b_i^g} = -I$$

对j时刻陀螺仪bias的雅可比

$$\frac{\partial r_{bg}}{\partial \delta b_j^g} = I$$

代码补全

lidar_localization/src/models/pre_integrator/imu_pre_integrator.cpp

FUNCTION: IMUPreIntegrator::UpdateState

```
void IMUPreIntegrator::UpdateState(void) {
    // 更新IMU预积分量
    static double T = 0.0;

    static Eigen::Vector3d w_mid = Eigen::Vector3d::Zero();
    static Eigen::Vector3d a_mid = Eigen::Vector3d::Zero();

    static Sophus::SO3d prev_theta_ij = Sophus::SO3d();
    static Sophus::SO3d curr_theta_ij = Sophus::SO3d();
    static Sophus::SO3d d_theta_ij = Sophus::SO3d();

    static Eigen::Matrix3d dR_inv = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d prev_R = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d curr_R = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d prev_R_a_hat = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d curr_R_a_hat = Eigen::Matrix3d::Zero();

    //
    // parse measurements:
    //
    // get measurement handlers:
    const IMUData &prev_imu_data = imu_data_buff_.at(0);    // 前一帧Imu数据
    const IMUData &curr_imu_data = imu_data_buff_.at(1);    // 当前帧Imu数据

    // get time delta:
    T = curr_imu_data.time - prev_imu_data.time;            // delta_t

    // get measurements:
    const Eigen::Vector3d prev_w(
        prev_imu_data.angular_velocity.x - state.b_g_i_.x(),
        prev_imu_data.angular_velocity.y - state.b_g_i_.y(),
        prev_imu_data.angular_velocity.z - state.b_g_i_.z()
    );
    const Eigen::Vector3d curr_w(
        curr_imu_data.angular_velocity.x - state.b_g_i_.x(),
        curr_imu_data.angular_velocity.y - state.b_g_i_.y(),
        curr_imu_data.angular_velocity.z - state.b_g_i_.z()
    );

    const Eigen::Vector3d prev_a(
        prev_imu_data.linear_acceleration.x - state.b_a_i_.x(),
        prev_imu_data.linear_acceleration.y - state.b_a_i_.y(),
        prev_imu_data.linear_acceleration.z - state.b_a_i_.z()
    );
    const Eigen::Vector3d curr_a(
        curr_imu_data.linear_acceleration.x - state.b_a_i_.x(),
        curr_imu_data.linear_acceleration.y - state.b_a_i_.y(),
        curr_imu_data.linear_acceleration.z - state.b_a_i_.z()
    );

    //
```

```
    // a. update mean:
    //
    // 1. get w_mid:
    w_mid = 0.5 * ( prev_w + curr_w );
    // 2. update relative orientation, so3:
    prev_theta_ij = state.theta_ij_;
    d_theta_ij = Sophus::SO3d::exp(w_mid * T);
    state.theta_ij_ = state.theta_ij_ * d_theta_ij;
    curr_theta_ij = state.theta_ij_;
    // 3. get a_mid:
    a_mid = 0.5 * ( prev_theta_ij * prev_a + curr_theta_ij * curr_a );
    // 4. update relative translation:
    state.alpha_ij_ += (state.beta_ij_ + 0.5 * a_mid * T) * T;
    // 5. update relative velocity:
    state.beta_ij_ += a_mid * T;

    //
    // b. update covariance:
    //
    // 1. intermediate results:
    dR_inv = d_theta_ij.inverse().matrix();
    prev_R = prev_theta_ij.matrix();
    curr_R = curr_theta_ij.matrix();
    prev_R_a_hat = prev_R * Sophus::SO3d::hat(prev_a);
    curr_R_a_hat = curr_R * Sophus::SO3d::hat(curr_a);

    //
    // 2. set up F:
    //
    // F12 & F32:
    F_.block<3, 3>(INDEX_ALPHA, INDEX_THETA) = F_.block<3, 3>(INDEX_BETA, INDEX_THETA) = -0.50 * (prev_R_a_hat + curr_R_a_hat * dR_inv);
    F_.block<3, 3>(INDEX_ALPHA, INDEX_THETA) = 0.50 * T * F_.block<3, 3>(INDEX_ALPHA, INDEX_THETA);
    // F14 & F34:
    F_.block<3, 3>(INDEX_ALPHA,   INDEX_B_A) = F_.block<3, 3>(INDEX_BETA, INDEX_B_A) = -0.50 * (prev_R + curr_R);
    F_.block<3, 3>(INDEX_ALPHA,   INDEX_B_A) = 0.50 * T * F_.block<3, 3>(INDEX_ALPHA,   INDEX_B_A);
    // F15 & F35:
    F_.block<3, 3>(INDEX_ALPHA,   INDEX_B_G) = F_.block<3, 3>(INDEX_BETA, INDEX_B_G) = +0.50 * T * curr_R_a_hat;
    F_.block<3, 3>(INDEX_ALPHA,   INDEX_B_G) = 0.50 * T * F_.block<3, 3>(INDEX_ALPHA,   INDEX_B_G);
    // F22:
    F_.block<3, 3>(INDEX_THETA, INDEX_THETA) = -Sophus::SO3d::hat(w_mid);

    //
    // 3. set up G:
    //
    // G11 & G31:
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_ACC_PREV) = B_.block<3, 3>(INDEX_BETA, INDEX_M_ACC_PREV) = +0.50 * prev_R;
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_ACC_PREV) = 0.50 * T * B_.block<3, 3>(INDEX_ALPHA, INDEX_M_ACC_PREV);
    // G12 & G32:
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_GYR_PREV) = B_.block<3, 3>(INDEX_BETA, INDEX_M_GYR_PREV) = -0.25 * T * curr_R_a_hat;
```

```
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_GYR_PREV) = 0.50 * T * B_.block<3, 3>
(INDEX_ALPHA, INDEX_M_GYR_PREV);
    // G13 & G33:
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_ACC_CURR) = B_.block<3, 3>(INDEX_BETA,
INDEX_M_ACC_CURR) = 0.5 * curr_R;
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_ACC_CURR) = 0.50 * T * B_.block<3, 3>
(INDEX_ALPHA, INDEX_M_ACC_CURR);
    // G14 & G34:
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_GYR_CURR) = B_.block<3, 3>(INDEX_BETA,
INDEX_M_GYR_CURR) = -0.25 * T * curr_R_a_hat;
    B_.block<3, 3>(INDEX_ALPHA, INDEX_M_GYR_CURR) = 0.50 * T * B_.block<3, 3>
(INDEX_ALPHA, INDEX_M_GYR_CURR);

    // 4. update P_:
    MatrixF F = MatrixF::Identity() + T * F_;
    MatrixB B = T * B_;

    P_ = F*P_*F.transpose() + B*Q_*B.transpose();

    //
    // c. update Jacobian:
    //
    J_ = F * J_;
}
```

FILE : lidar_localization/include/lidar_localization/models/graph_optimizer/g2o/edge/edge_prvag_imu_pre_integration.hpp

FUNCTION: computeError

```
virtual void computeError() override {
        g2o::VertexPRVAG* v0 = dynamic_cast<g2o::VertexPRVAG*>(_vertices[0]);
        g2o::VertexPRVAG* v1 = dynamic_cast<g2o::VertexPRVAG*>(_vertices[1]);

        const Eigen::Vector3d &pos_i = v0->estimate().pos;
        const Sophus::SO3d    &ori_i = v0->estimate().ori;
        const Eigen::Vector3d &vel_i = v0->estimate().vel;
        const Eigen::Vector3d &b_a_i = v0->estimate().b_a;
        const Eigen::Vector3d &b_g_i = v0->estimate().b_g;

        const Eigen::Vector3d &pos_j = v1->estimate().pos;
        const Sophus::SO3d    &ori_j = v1->estimate().ori;
        const Eigen::Vector3d &vel_j = v1->estimate().vel;
        const Eigen::Vector3d &b_a_j = v1->estimate().b_a;
        const Eigen::Vector3d &b_g_j = v1->estimate().b_g;


        // TODO: 更新由于imu bias改变所带来的预积分量的改变
        if(v0->isUpdated()){
            Eigen::Vector3d d_b_a_i, d_b_g_i;
            v0->getDeltaBiases(d_b_a_i,d_b_g_i);
            updateMeasurement(d_b_a_i,d_b_g_i);
        }


        // TODO: 计算预积分误差:
        const Eigen::Vector3d &alpha_ij = _measurement.block<3, 1>(INDEX_P, 0);
```

```
        const Eigen::Vector3d &theta_ij = _measurement.block<3, 1>(INDEX_R, 0);
        const Eigen::Vector3d  &beta_ij = _measurement.block<3, 1>(INDEX_V, 0);

        _error.block<3, 1>(INDEX_P, 0) = ori_i.inverse() * (pos_j - pos_i -
(vel_i - 0.50 * g_ * T_) * T_) - alpha_ij;
        _error.block<3, 1>(INDEX_R, 0) =
(Sophus::SO3d::exp(theta_ij).inverse()*ori_i.inverse()*ori_j).log();
        _error.block<3, 1>(INDEX_V, 0) = ori_i.inverse() * (vel_j - vel_i + g_ *
T_) - beta_ij;
        _error.block<3, 1>(INDEX_A, 0) = b_a_j - b_a_i;
        _error.block<3, 1>(INDEX_G, 0) = b_g_j - b_g_i;
    }
```

FILE : lidar_localization/include/lidar_localization/models/graph_optimizer/g2o/vertex/vertex_prvag.hpp

FUNCTION: oplusImpl

```
    virtual void oplusImpl(const double *update) override {
        // 定义g2o节点的更新方式
        _estimate.pos += Eigen::Vector3d(
            update[PRVAG::INDEX_POS + 0], update[PRVAG::INDEX_POS + 1],
update[PRVAG::INDEX_POS + 2]
        );
        _estimate.ori = _estimate.ori * Sophus::SO3d::exp(
            Eigen::Vector3d(
                update[PRVAG::INDEX_ORI + 0], update[PRVAG::INDEX_ORI + 1],
update[PRVAG::INDEX_ORI + 2]
            )
        );
        _estimate.vel += Eigen::Vector3d(
            update[PRVAG::INDEX_VEL + 0], update[PRVAG::INDEX_VEL + 1],
update[PRVAG::INDEX_VEL + 2]
        );

        Eigen::Vector3d d_b_a_i(
            update[PRVAG::INDEX_B_A + 0], update[PRVAG::INDEX_B_A + 1],
update[PRVAG::INDEX_B_A + 2]
        );
        Eigen::Vector3d d_b_g_i(
            update[PRVAG::INDEX_B_G + 0], update[PRVAG::INDEX_B_G + 1],
update[PRVAG::INDEX_B_G + 2]
        );

        _estimate.b_a += d_b_a_i;
        _estimate.b_g += d_b_g_i;

        updateDeltaBiases(d_b_a_i, d_b_g_i);
    }
```

运行

代码运行命令：

```
roslaunch lidar_localization lio_mapping.launch
```

播放数据集命令：

```
rosbag  play kitti_lidar_only_2011_10_03_drive_0027_synced.bag
```

保存地图与Loop Closure Data：

```
rosservice call /optimize_map
rosservice call /save_map
rosservice call /save_scan_context
```

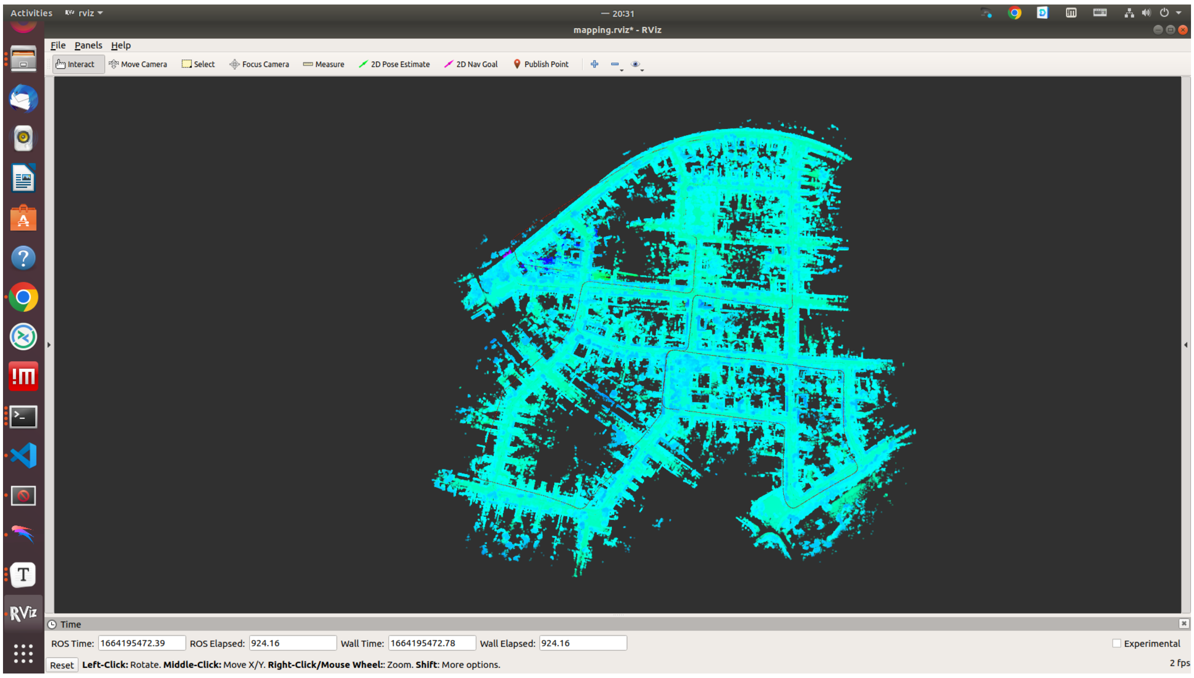
上述三个ROS Service会生成所需的Map、trajectory Data与Scan Context Data. 分别位于:

```
Map: src/lidar_localization/slam_data/map
Scan Context Data: src/lidar_localization/slam_data/scan_context
trajectory Data: src/lidar_localization/slam_data/trajectory
```

evo工具运行命令：

```
evo_rpe kitti ground_truth.txt laser_odom.txt  -r trans_part --delta 100 --plot --plot_mode xyz
evo_rpe kitti ground_truth.txt optimized.txt -r trans_part --delta 100 --plot --plot_mode xyz
evo_ape kitti ground_truth.txt laser_odom.txt  -r full --plot --plot_mode xyz
evo_ape kitti ground_truth.txt optimized.txt -r full --plot --plot_mode xyz
```

RVIZ效果



EVO评估

加入预积分

优化前

rpe

```
      max	15.764265
     mean	7.217924
   median	4.912390
      min	1.362512
     rmse	8.760765
      sse	1458.269054
      std	4.965137
```

ape

```
      max	29.330454
     mean	11.783149
   median	11.009673
      min	0.000001
     rmse	13.870033
      sse	368018.773688
      std	7.316777
```

优化后

rpe

```
      max	4.397120
     mean	1.957728
   median	1.650077
      min	0.831014
     rmse	2.162346
      sse	88.839087
      std	0.918173
```

ape

```
      max	0.519413
     mean	0.067246
   median	0.046663
      min	0.006491
     rmse	0.089805
      sse	15.428082
      std	0.059522
```

未加入预积分

优化前

rpe

```
   max   4.028356
  mean   1.927022
median   1.929623
   min   0.516572
  rmse   2.083237
   sse   82.457656
   std   0.791495
```

ape

```
   max   44.107971
  mean   17.651135
median   17.619473
   min   0.000001
  rmse   21.025774
   sse   845705.078909
   std   11.424561
```

优化后

rpe

```
  max    4.595808
  mean   2.001740
median   1.632527
   min   0.585584
  rmse   2.243285
   sse   95.614240
   std   1.012603
```

ape

```
   max   0.514676
  mean   0.097238
median   0.053598
   min   0.008927
  rmse   0.133644
   sse   34.167559
   std   0.091682
```

二.融合编码器预积分公式推导

测量数据

陀螺仪的角速度

$$\omega_k = \begin{bmatrix} \omega_{xk} \\ \omega_{yk} \\ \omega_{zk} \end{bmatrix}$$

编码器的速度

$$v_k = \begin{bmatrix} v_{xk} \\ 0 \\ 0 \end{bmatrix}$$

预积分量

$$\alpha_{b_ib_j} = \int_{t\in[i,j]} (q_{b_ib_t} v^{bt})\delta t$$
$$q_{b_ib_j} = \int_{t\in[i,j]} q_{b_ib_t} \bigotimes \begin{bmatrix} 0 \\ \frac{1}{2}\omega^{bt} \end{bmatrix}$$

对应的预积分量更新方式为

$$q_{b_ib_k+1} = q_{b_ib_k} \bigotimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega^{b}\delta t \end{bmatrix}$$
$$\alpha_{b_ib_k+1} = \alpha_{b_ib_k} + v^{w}\delta t$$

其中

$$\omega^{b} = \frac{1}{2}[(\omega^{b_k} - b_i^g) + (\omega^{b_{k+1}} - b_i^g)]$$
$$\phi^{w} = \frac{1}{2}(q_{b_ib_k}\phi^{b_k} + q_{b_ib_k}\phi^{b_k})$$

对应的误差传递方程(此处认为编码器没有误差)为

$$\delta\theta_{k+1} = [I - [\frac{\omega^{b_k} + \omega^{b_{k+1}}}{2} - b_{\omega k}]_\times \delta t]\delta\theta_k + \frac{\delta t}{2} n_{\omega_k} + \frac{\delta t}{2} n_{\omega_{k+1}} - \delta b_{\omega_k}\delta t$$
$$\delta\alpha_{k+1} = \delta\alpha_k - (\frac{R_k[v_k]_\times}{2}\delta\theta_k + \frac{R_{k+1}[v_{k+1}]_\times}{2}\delta\theta_{k+1})\delta t$$
$$= \delta\alpha_{k+1} = \delta\alpha_k - (\frac{R_k[v_k]_\times}{2}\delta\theta_k + \frac{R_{k+1}[v_{k+1}]_\times}{2}([I - [\frac{\omega^{b_k} + \omega^{b_{k+1}}}{2} - b_{\omega k}]_\times \delta t]\delta\theta_k + \frac{\delta t}{2} n_{\omega_k} + \frac{\delta t}{2} n_{\omega_{k+1}} - \delta b_{\omega_k}\delta t))\delta t$$
$$= \delta\alpha_k - (\frac{R_k[v_k]_\times + R_{k+1}[v_{k+1}]_\times([I - [\frac{\omega^{b_k}+\omega^{b_{k+1}}}{2} - b_{\omega k}]_\times \delta t])}{2}\delta t)\delta\theta_k + \frac{R_{k+1}[v_{k+1}]_\times \delta t^2}{4}\delta b_{w_k} - \frac{R_{k+1}[v_{k+1}]_\times \delta t^2}{4} n_{\omega_k} - \frac{R_{k+1}[v_{k+1}]_\times \delta t^2}{4} n_{\omega_{k+1}}$$
$$\delta b_{\omega_{k+1}} = \delta b_{\omega_k} + \delta t n_{b_\omega}$$

当bias变化时，预积分的更新方式为

$$\alpha_{b_ib_j} = \alpha_{b_ib_j} + J^{\alpha}_{b_i^g}\delta b_i^g$$
$$q_{b_ib_j} = q_{b_ib_j} \bigotimes \begin{bmatrix} 1 \\ \frac{1}{2}J^{q}_{b_i^g}\delta b_i^g \end{bmatrix}$$

预积分残差为

$$\begin{bmatrix} r_p \\ r_q \\ r_{b_g} \end{bmatrix} = \begin{bmatrix} q^*_{wb_i}(p_{wb_j} - p_{wb_i}) - \alpha_{b_ib_j} \\ 2[q^*_{b_ib_j} \bigotimes (q^*_{wb_i} \bigotimes q_{wb_j})]_{xyz} \\ b_j^g - b_i^g \end{bmatrix}$$

姿态残差的雅可比

对i时刻姿态的雅可比

$$\frac{\partial r_q}{\partial \delta\theta_{b_ib_i'}} = -2\,[0 \quad I\,]\,[q^*_{wb_j} \bigotimes q_{wb_I}]_L [q_{b_ib_j}]_R \begin{bmatrix} 0 \\ \frac{1}{2}I \end{bmatrix}$$

对j时刻姿态的雅可比

$$\frac{\partial r_q}{\partial \delta\theta_{b_jb_j'}} = -2\,[0 \quad I\,]\,[q^*_{wb_j} \bigotimes q_{wb_I}]_L [q_{b_ib_j}]_R \begin{bmatrix} 0 \\ \frac{1}{2}I \end{bmatrix}$$

对i时刻陀螺仪bias偏差的雅可比

$$\frac{\partial r_q}{\partial \delta b_i^g} = -2\begin{bmatrix}0 & I\end{bmatrix}[q_{wb_j}^* \bigotimes q_{wb_i} \bigotimes q_{b_i b_j}]_L \begin{bmatrix} 0 \\ \frac{1}{2} J_{b_i^g}^q \end{bmatrix}$$

位置残差的雅可比

对i时刻姿态的雅可比

$$\frac{\partial r_p}{\partial \delta \theta_{b_i b_i'}} = [R_{b_i w}(p_{wb_j} - p_{wb_i})]_\times$$

对i时刻位置的雅可比

$$\frac{\partial r_p}{\partial \delta p_i^w} = -R_{wb_i}$$

对j时刻位置的雅可比

$$\frac{\partial r_p}{\partial \delta p_j^w} = R_{wb_i}$$

对i时刻陀螺仪bias的雅可比

$$\frac{\partial r_p}{\partial \delta b_i^g} = -\frac{\partial \alpha_{b_i b_j}}{\partial \delta b_i^g} = -J_{b_i^g}^\alpha$$

陀螺仪残差的雅可比

对i时刻陀螺仪bias的雅可比

$$\frac{\partial r_{bg}}{\partial \delta b_i^g} = -I$$

对j时刻陀螺仪bias的雅可比

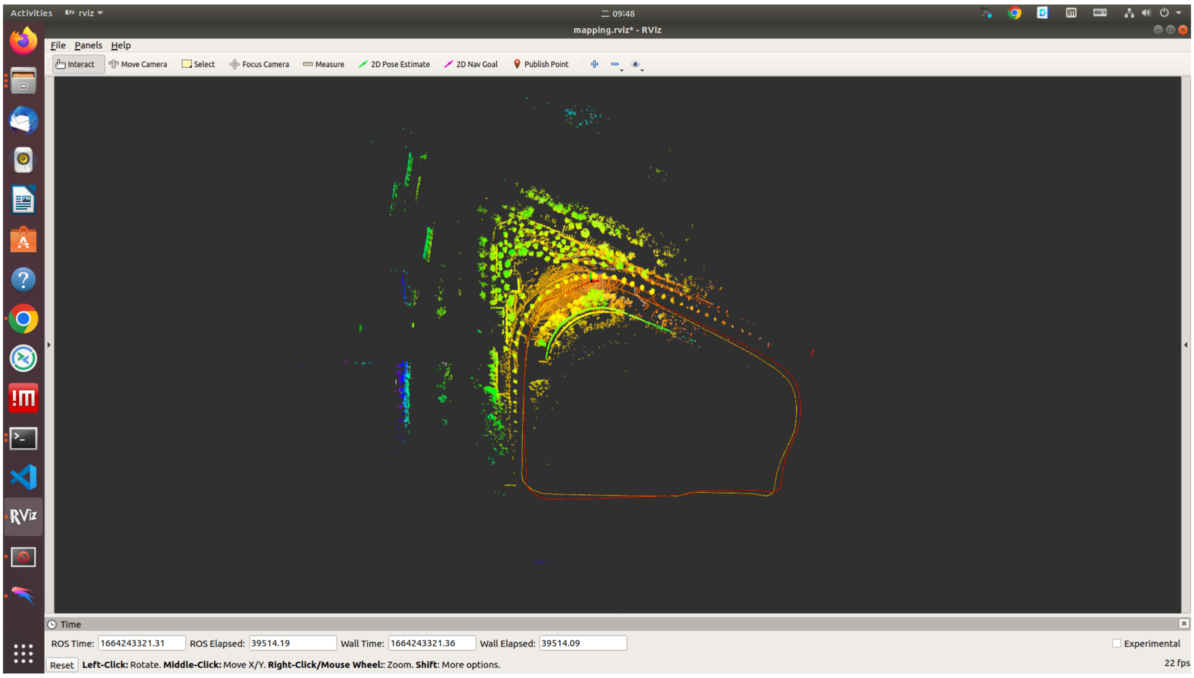
$$\frac{\partial r_{bg}}{\partial \delta b_j^g} = I$$

三.实车部署

实车硬件如下：

1. 松灵Scout2，车速为1.5m/s
2. 速腾16线雷达
3. SBG-ellipse-N 九轴惯导+单天线RTK

rviz效果



evo评估

加入预积分-优化前

rpe

```
       max	10.663504
      mean	7.417549
    median	7.417549
       min	4.171595
      rmse	8.096682
       sse	131.112519
       std	3.245955
```

ape

```
      max	5.122838
      mean	3.364471
    median	3.781946
       min	0.000001
      rmse	3.614327
       sse	3683.866944
       std	1.320489
```

加入预积分-优化后

rpe

```
       max	11.744275
      mean	7.567136
    median	7.567136
       min	3.389998
      rmse	8.643497
       sse	149.420078
       std	4.177139
```

ape

```
   max  0.252908
  mean  0.091635
median  0.086126
   min  0.029102
  rmse  0.096680
   sse  2.635848
   std  0.030823
```

总结和思考

总结

从evo评估可看出，添加了预积分进行优化对轨迹的精度有提升。

思考和疑问

1. 添加了预积分进行优化对于所需计算量有提升吗？如果想要一定的实时性，该如何去确定这个后端优化的频率？