第10章作业

一. 代码补全

实现预积分、地图匹配、边缘化、帧间匹配四种优化因子

激光里程计因子

FILE:lidar_localization/include/lidar_localization/models/sliding_window/factors/factorsfactor_prvag_relative_pose.hpp

```
  virtual bool Evaluate(double const *const *parameters, double *residuals,
double **jacobians) const {
    //
    // parse parameters:
    //
    // a. pose i
    Eigen::Map<const Eigen::Vector3d>     pos_i(&parameters[0][INDEX_P]);
    Eigen::Map<const Eigen::Vector3d> log_ori_i(&parameters[0][INDEX_R]);
    const Sophus::SO3d                    ori_i = Sophus::SO3d::exp(log_ori_i);

    // b. pose j
    Eigen::Map<const Eigen::Vector3d>     pos_j(&parameters[1][INDEX_P]);
    Eigen::Map<const Eigen::Vector3d> log_ori_j(&parameters[1][INDEX_R]);
    const Sophus::SO3d                    ori_j = Sophus::SO3d::exp(log_ori_j);

    //
    // parse measurement:
    //
        const Eigen::Vector3d     &pos_ij = m_.block<3, 1>(INDEX_P, 0);
        const Eigen::Vector3d &log_ori_ij = m_.block<3, 1>(INDEX_R, 0);
    const Sophus::SO3d          ori_ij = Sophus::SO3d::exp(log_ori_ij);

    //
    // TODO: get square root of information matrix:
    //
    Eigen::Matrix<double, 6, 6>  sqrt_info =  Eigen::LLT<Eigen::Matrix<double, 6
,6>>(
      I_
    ).matrixL().transpose() ;

    //
    // TODO: compute residual:
    //
    Eigen::Map<Eigen::Matrix<double, 6 ,1>>  residual(residuals);     //  残差
r_P  r_R

    residual.block(INDEX_P,  0 , 3 , 1)   =  ori_i.inverse() * (pos_j - pos_i) -
pos_ij ;
    residual.block(INDEX_R, 0  , 3 , 1)   =
(ori_i.inverse()*ori_j*ori_ij.inverse()).log( );
    //
    // TODO: compute jacobians: 因为是二元边，所以需要求解两个jacobian
```

```
    //
    if ( jacobians ) {
      // compute shared intermediate results:

      const Eigen::Matrix3d R_i_inv = ori_i.inverse().matrix();
      const Eigen::Matrix3d J_r_inv = JacobianRInv(residual.block(INDEX_R, 0 ,3
, 1));    //  右雅克比
      const Eigen::Vector3d pos_ij = ori_i.inverse() * (pos_j - pos_i) ;

      if ( jacobians[0] ) {  //   残差rL0(rp  rq )  对 T0(p q)  M0(v ba bg) 的雅克
比
        // implement computing:
          Eigen::Map<Eigen::Matrix<double, 6 , 15,  Eigen::RowMajor>>
 jacobian_i (jacobians[0] );  //  col : rp_i[3] rq_i[3]  row : p[3] q[3] v[3]
ba[3] bg[3]
        jacobian_i.setZero();

        jacobian_i.block<3, 3>(INDEX_P, INDEX_P) =  -R_i_inv;
        jacobian_i.block<3, 3>(INDEX_R,INDEX_R)  =  -J_r_inv*
(ori_ij*ori_j.inverse()*ori_i).matrix();
        jacobian_i.block<3, 3>(INDEX_P,INDEX_R) =
 Sophus::SO3d::hat(pos_ij).matrix();

        jacobian_i = sqrt_info * jacobian_i ;        //   注意 sqrt_i 为对角的协方差
矩阵对角线为观测的方差，可理解为传感器的测量误差，用于调整权重用
      }

      if ( jacobians[1] ) {    //  残差rL0(rp  rq )  对 T0(p q)  M0(v ba bg) 的雅克
比
        // implement computing:
        Eigen::Map<Eigen::Matrix<double, 6 ,15, Eigen::RowMajor>> jacobian_j
(jacobians[1]);
        jacobian_j.setZero();

        jacobian_j.block<3, 3>(INDEX_P, INDEX_P) = R_i_inv;
        jacobian_j.block<3, 3>(INDEX_R,INDEX_R)  = J_r_inv*ori_ij.matrix();

        jacobian_j = sqrt_info * jacobian_j ;
      }
    }

    //
    // TODO: correct residual by square root of information matrix:
    //
    residual = sqrt_info * residual;

    return true;
  }
```

地图匹配因子

FILE:lidar_localization/include/lidar_localization/models/sliding_window/factors/factor_prvag_map_matching_pose.hpp

```
  virtual bool Evaluate(double const *const *parameters, double *residuals,
double **jacobians) const {
    //
```

```
    // parse parameters:
    //
    // pose
    Eigen::Map<const Eigen::Vector3d>     pos(&parameters[0][INDEX_P]);
    Eigen::Map<const Eigen::Vector3d> log_ori(&parameters[0][INDEX_R]);
    const Sophus::SO3d                    ori = Sophus::SO3d::exp(log_ori);

    //
    // parse measurement:
    //
        const Eigen::Vector3d     &pos_prior = m_.block<3, 1>(INDEX_P, 0);
        const Eigen::Vector3d &log_ori_prior = m_.block<3, 1>(INDEX_R, 0);
    const Sophus::SO3d         ori_prior = Sophus::SO3d::exp(log_ori_prior);

    //
    // TODO: get square root of information matrix:
    //
    // Cholesky 分解 : http://eigen.tuxfamily.org/dox/classEigen_1_1LLT.html
    Eigen::Matrix<double, 6, 6> sqrt_info = Eigen::LLT<Eigen::Matrix<double, 6,
6>>(
      I_
    ).matrixL().transpose();

    //
    // TODO: compute residual:
    //
    Eigen::Map<Eigen::Matrix<double, 6 ,1>>  residual(residuals);

              residual.block(INDEX_P, 0 , 3 , 1 )    =  pos - pos_prior ;
              residual.block(INDEX_R,0 , 3 , 1 )     =
(ori*ori_prior.inverse()).log();
    //
    // TODO: compute jacobians:
    //
    if ( jacobians ) {
      if ( jacobians[0] ) {
        // implement jacobian computing:
        Eigen::Map<Eigen::Matrix<double, 6, 15,  Eigen::RowMajor>>
jacobian_prior(jacobians[0] );
        jacobian_prior.setZero();

        jacobian_prior.block<3, 3>(INDEX_P,  INDEX_P) =
Eigen::Matrix3d::Identity();
        jacobian_prior.block<3, 3>(INDEX_R, INDEX_R)  = JacobianRInv(
                                     residual.block(INDEX_R, 0, 3, 1)) *
ori_prior.matrix();

        jacobian_prior = sqrt_info * jacobian_prior ;
      }
    }

    //
    // TODO: correct residual by square root of information matrix:
    //
        residual = sqrt_info * residual;

    return true;
  }
```

IMU预积分因子

FILE:lidar_localization/include/lidar_localization/models/sliding_window/factors/factor_prvag_imu_pre_integration.hpp

```
  virtual bool Evaluate(double const *const *parameters, double *residuals,
double **jacobians) const {
    //
    // parse parameters:
    //
    // a. pose i
    Eigen::Map<const Eigen::Vector3d>     pos_i(&parameters[0][INDEX_P]);
    Eigen::Map<const Eigen::Vector3d> log_ori_i(&parameters[0][INDEX_R]);
    const Sophus::SO3d                    ori_i = Sophus::SO3d::exp(log_ori_i);
        Eigen::Map<const Eigen::Vector3d>     vel_i(&parameters[0][INDEX_V]);
        Eigen::Map<const Eigen::Vector3d>     b_a_i(&parameters[0][INDEX_A]);
        Eigen::Map<const Eigen::Vector3d>     b_g_i(&parameters[0][INDEX_G]);

    // b. pose j
    Eigen::Map<const Eigen::Vector3d>     pos_j(&parameters[1][INDEX_P]);
    Eigen::Map<const Eigen::Vector3d> log_ori_j(&parameters[1][INDEX_R]);
    const Sophus::SO3d                    ori_j = Sophus::SO3d::exp(log_ori_j);
        Eigen::Map<const Eigen::Vector3d>     vel_j(&parameters[1][INDEX_V]);
        Eigen::Map<const Eigen::Vector3d>     b_a_j(&parameters[1][INDEX_A]);
        Eigen::Map<const Eigen::Vector3d>     b_g_j(&parameters[1][INDEX_G]);

    //
    // parse measurement:
    //
        const Eigen::Vector3d &alpha_ij = m_.block<3, 1>(INDEX_P, 0);
        const Eigen::Vector3d &theta_ij = m_.block<3, 1>(INDEX_R, 0);
        const Eigen::Vector3d  &beta_ij = m_.block<3, 1>(INDEX_V, 0);

    //
    // TODO: get square root of information matrix:
    //
    // Cholesky 分解 : http://eigen.tuxfamily.org/dox/classEigen_1_1LLT.html
    Eigen::LLT<Eigen::Matrix<double,15,15>> LowerI(I_);
    // sqrt_info 为上三角阵
    Eigen::Matrix<double,15,15> sqrt_info = LowerI.matrixL().transpose();

    //
    // TODO: compute residual:
    //
     Eigen::Map<Eigen::Matrix<double, 15, 1>>  residual(residuals);

     residual.block<3, 1>(INDEX_P, 0) = ori_i.inverse().matrix() * (pos_j -
pos_i - (vel_i - 0.5 * g_ * T_) * T_) - alpha_ij ;
     residual.block<3, 1>(INDEX_R,0) =
(Sophus::SO3d::exp(theta_ij).inverse()*ori_i.inverse()*ori_j).log();
     residual.block<3, 1>(INDEX_V,0)  = ori_i.inverse()* (vel_j - vel_i + g_ *
T_) - beta_ij ;
     residual.block<3, 1>(INDEX_A,0)  = b_a_j - b_a_i ;
     residual.block<3, 1>(INDEX_G,0)  = b_g_j - b_g_i;

    //
```

```
    // TODO: compute jacobians:  imu预积分的残差 对状态量的雅克比，第九章已推导
    //
    if ( jacobians ) {
      // compute shared intermediate results:
      const Eigen::Matrix3d R_i_inv = ori_i.inverse().matrix();
      const Eigen::Matrix3d J_r_inv = JacobianRInv(residual.block(INDEX_R, 0 ,3
, 1));    //  右雅克比

      if ( jacobians[0] ) {
        Eigen::Map<Eigen::Matrix<double, 15 , 15,  Eigen::RowMajor>>  jacobian_i
(jacobians[0] );
        jacobian_i.setZero();

         // a. residual, position:
        jacobian_i.block<3, 3>(INDEX_P, INDEX_P) = -R_i_inv;
        jacobian_i.block<3, 3>(INDEX_P, INDEX_R) = Sophus::SO3d::hat(
          ori_i.inverse() * (pos_j - pos_i - (vel_i - 0.50 * g_ * T_) * T_)
        );
        jacobian_i.block<3, 3>(INDEX_P, INDEX_V) = -T_*R_i_inv;
        jacobian_i.block<3, 3>(INDEX_P, INDEX_A) = -J_.block<3,3>(INDEX_P,
INDEX_A);
        jacobian_i.block<3, 3>(INDEX_P, INDEX_G) = -J_.block<3,3>(INDEX_P,
INDEX_G);

        // b. residual, orientation:
        jacobian_i.block<3, 3>(INDEX_R, INDEX_R) = -J_r_inv*
(ori_j.inverse()*ori_i).matrix();
        jacobian_i.block<3, 3>(INDEX_R, INDEX_G) = -J_r_inv*(
          Sophus::SO3d::exp(residual.block<3, 1>(INDEX_R, 0))
        ).matrix().inverse()*J_.block<3,3>(INDEX_R, INDEX_G);

        // c. residual, velocity:
        jacobian_i.block<3, 3>(INDEX_V, INDEX_R) = Sophus::SO3d::hat(
          ori_i.inverse() * (vel_j - vel_i + g_ * T_)
        );
        jacobian_i.block<3, 3>(INDEX_V, INDEX_V) = -R_i_inv;
        jacobian_i.block<3, 3>(INDEX_V, INDEX_A) = -J_.block<3,3>(INDEX_V,
INDEX_A);
        jacobian_i.block<3, 3>(INDEX_V, INDEX_G) = -J_.block<3,3>(INDEX_V,
INDEX_G);

        // d. residual, bias accel:
        jacobian_i.block<3, 3>(INDEX_A, INDEX_A) = -Eigen::Matrix3d::Identity();

        // d. residual, bias accel:
        jacobian_i.block<3, 3>(INDEX_G, INDEX_G) = -Eigen::Matrix3d::Identity();

        jacobian_i = sqrt_info * jacobian_i;
      }

      if ( jacobians[1] ) {
        Eigen::Map<Eigen::Matrix<double, 15, 15, Eigen::RowMajor>>
jacobian_j(jacobians[1]);
        jacobian_j.setZero();

        // a. residual, position:
        jacobian_j.block<3, 3>(INDEX_P, INDEX_P) = R_i_inv;
```

```
        // b. residual, orientation:
        jacobian_j.block<3, 3>(INDEX_R, INDEX_R) = J_r_inv;

        // c. residual, velocity:
        jacobian_j.block<3, 3>(INDEX_V, INDEX_V) = R_i_inv;

        // d. residual, bias accel:
        jacobian_j.block<3, 3>(INDEX_A, INDEX_A) = Eigen::Matrix3d::Identity();

        // d. residual, bias accel:
        jacobian_j.block<3, 3>(INDEX_G, INDEX_G) = Eigen::Matrix3d::Identity();

        jacobian_j = sqrt_info * jacobian_j;
      }
    }

    //
    // TODO: correct residual by square root of information matrix:
    //
        residual = sqrt_info * residual;

    return true;
  }
```

边缘化先验因子

FILE:lidar_localization/include/lidar_localization/models/sliding_window/factors/factor_prvag_marginalization.hpp

```
  void SetResMapMatchingPose(
    const ceres::CostFunction *residual,
    const std::vector<double *> &parameter_blocks
  ) {
    // init:
    ResidualBlockInfo res_map_matching_pose(residual, parameter_blocks);
    Eigen::VectorXd residuals;
    std::vector<Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic,
Eigen::RowMajor>> jacobians;

    // compute:
    Evaluate(res_map_matching_pose, residuals, jacobians);
    const Eigen::MatrixXd &J_m = jacobians.at(0);

    //
    // TODO: Update H:
    //
    // a. H_mm:
    H_.block<15,  15>(INDEX_M, INDEX_M) += J_m.transpose() * J_m ;

    //
    // TODO: Update b:
    //
    // a. b_m:
    b_.block<15, 1>(INDEX_M , 0) +=  J_m.transpose() * residuals ;  //  因子图叠加

  }
```

```
  void SetResRelativePose(
    const ceres::CostFunction *residual,
    const std::vector<double *> &parameter_blocks
  ) {
    // init:
    ResidualBlockInfo res_relative_pose(residual, parameter_blocks);
    Eigen::VectorXd residuals;
    std::vector<Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic,
Eigen::RowMajor>> jacobians;

    // compute:
    Evaluate(res_relative_pose, residuals, jacobians);
    const Eigen::MatrixXd &J_m = jacobians.at(0);
    const Eigen::MatrixXd &J_r = jacobians.at(1);

    //
    // TODO: Update H:
    //
    // a. H_mm:
    H_.block<15, 15>(INDEX_M, INDEX_M)  += J_m.transpose() * J_m;
    // b. H_mr:
    H_.block<15, 15>(INDEX_M, INDEX_R)   += J_m.transpose()* J_r;
    // c. H_rm:
    H_.block<15,  15>(INDEX_R, INDEX_M)  += J_r.transpose() * J_m;
    // d. H_rr:
    H_.block<15,  15>(INDEX_R, INDEX_R)   += J_r.transpose() * J_r;


    //
    // TODO: Update b:
    //
    // a. b_m:
    b_.block<15,  1>(INDEX_M, 0) += J_m.transpose() * residuals ;
    // a. b_r:
    b_.block<15,   1>(INDEX_R,  0) += J_r.transpose() * residuals ;
  }

  void SetResIMUPreIntegration(
    const ceres::CostFunction *residual,
    const std::vector<double *> &parameter_blocks
  ) {
    // init:
    ResidualBlockInfo res_imu_pre_integration(residual, parameter_blocks);
    Eigen::VectorXd residuals;
    std::vector<Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic,
Eigen::RowMajor>> jacobians;

    // compute:
    Evaluate(res_imu_pre_integration, residuals, jacobians);
    const Eigen::MatrixXd &J_m = jacobians.at(0);
    const Eigen::MatrixXd &J_r = jacobians.at(1);

    //
    // TODO: Update H:
    //
    // a. H_mm:
    H_.block<15, 15>(INDEX_M, INDEX_M)  += J_m.transpose() * J_m;
    // b. H_mr:
```

```
    H_.block<15, 15>(INDEX_M, INDEX_R)   += J_m.transpose()* J_r;
    // c. H_rm:
    H_.block<15,  15>(INDEX_R, INDEX_M)  += J_r.transpose() * J_m;
    // d. H_rr:
    H_.block<15,  15>(INDEX_R, INDEX_R)   += J_r.transpose() * J_r;


    //
    // Update b:
    //
    // a. b_m:
    b_.block<15,  1>(INDEX_M, 0) += J_m.transpose() * residuals ;
    // a. b_r:
    b_.block<15,   1>(INDEX_R,  0) += J_r.transpose() * residuals ;
  }

  void Marginalize(
    const double *raw_param_r_0
  ) {
    // TODO: implement marginalization logic
        //  save x_m_0
    Eigen::Map<const  Eigen::Matrix<double, 15 , 1>>  x_0(raw_param_r_0);
    x_0_ = x_0 ;
    //   marginalize
    const Eigen::MatrixXd  &H_mm = H_.block<15,  15>(INDEX_M, INDEX_M);
    const Eigen::MatrixXd  &H_mr    = H_.block<15,   15>(INDEX_M,INDEX_R);
    const Eigen::MatrixXd  &H_rm    = H_.block<15,   15>(INDEX_R,INDEX_M);
    const Eigen::MatrixXd  &H_rr      = H_.block<15,    15>(INDEX_R,INDEX_R);

    const Eigen::VectorXd  &b_m = b_.block<15,  1>(INDEX_M, 0);
    const Eigen::VectorXd  &b_r    = b_.block<15,   1>(INDEX_R, 0);

    Eigen::MatrixXd H_mm_inv = H_mm.inverse();
    Eigen::MatrixXd H_marginalized = H_rr - H_rm * H_mm_inv * H_mr ;
    Eigen::MatrixXd b_marginalized = b_r - H_rm * H_mm_inv * b_m ;
    // 线性化残差 和 雅克比
    Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> saes(H_marginalized);
    Eigen::VectorXd S = Eigen::VectorXd(
      (saes.eigenvalues().array() > 1.0e-5).select(saes.eigenvalues().array(),
0)
    );
    Eigen::VectorXd S_inv = Eigen::VectorXd(
      (saes.eigenvalues().array() > 1.0e-
5).select(saes.eigenvalues().array().inverse(), 0)
    );

    Eigen::VectorXd S_sqrt = S.cwiseSqrt();
    Eigen::VectorXd S_inv_sqrt = S_inv.cwiseSqrt();

    // finally:
    J_ = S_sqrt.asDiagonal() * saes.eigenvectors().transpose();         //  b0
    e_ = S_inv_sqrt.asDiagonal() * saes.eigenvectors().transpose() *
b_marginalized;      //   eo
  }

  virtual bool Evaluate(double const *const *parameters, double *residuals,
double **jacobians) const {
    //
```

```
    // parse parameters:
    //
    Eigen::Map<const Eigen::Matrix<double, 15, 1>> x(parameters[0]);
    Eigen::VectorXd dx = x - x_0_;

    //
    // TODO: compute residual:
    //
    Eigen::Map<Eigen::Matrix<double, 15, 1>>  residual(residuals);
    residual = e_ + J_ * dx ;       //  e_prior

    //
    // TODO: compute jacobian:
    //
    if ( jacobians ) {
      if ( jacobians[0] ) {
        // implement computing:
        Eigen::Map<Eigen::Matrix<double, 15, 15, Eigen::RowMajor>>
 jacobian_marginalization(jacobians[0]);
        jacobian_marginalization.setZero();

        jacobian_marginalization = J_ ;
      }
    }

    return true;
  }
```

将四种约束因子，加入滑窗，进行优化

将因子添加到优化框架中

FILE : lidar_localization/src/matching/back_end/sliding_window.cpp

```
bool SlidingWindow::Update(void) {
    static KeyFrame last_key_frame_ = current_key_frame_;

    //
    // add node for new key frame pose:
    //
    // fix the pose of the first key frame for lidar only mapping:
    if ( sliding_window_ptr_->GetNumParamBlocks() == 0 ) {
        // TODO: add init key frame
        sliding_window_ptr_->AddPRVAGParam(current_key_frame_, true);

    } else {
        // TODO: add current key frame
        sliding_window_ptr_->AddPRVAGParam(current_key_frame_, false);

    }

    // get num. of vertices:
    const int N = sliding_window_ptr_->GetNumParamBlocks();
    // get param block ID, current:
    const int param_index_j = N - 1;

    //
```

```
    // add unary constraints:
    //
    //
    // a. map matching / GNSS position:
    //
    if ( N > 0 && measurement_config_.source.map_matching ) {
        // get prior position measurement:
        Eigen::Matrix4d prior_pose =
current_map_matching_pose_.pose.cast<double>();

        // TODO: add constraint, GNSS position:
            sliding_window_ptr_->AddPRVAGMapMatchingPoseFactor(
            param_index_j,
            prior_pose, measurement_config_.noise.map_matching
        );
    }

    //
    // add binary constraints:
    //
    if ( N > 1 ) {
        // get param block ID, previous:
        const int param_index_i = N - 2;

        //
        // a. lidar frontend:
        //
        // get relative pose measurement:
        Eigen::Matrix4d relative_pose = (last_key_frame_.pose.inverse() *
current_key_frame_.pose).cast<double>();
        // TODO: add constraint, lidar frontend / loop closure detection:
        sliding_window_ptr_->AddPRVAGRelativePoseFactor(
            param_index_i, param_index_j,
            relative_pose, measurement_config_.noise.lidar_odometry
        );
        //
        // b. IMU pre-integration:
        //
        if ( measurement_config_.source.imu_pre_integration ) {
            // TODO: add constraint, IMU pre-integraion:
            sliding_window_ptr_->AddPRVAGIMUPreIntegrationFactor(
                param_index_i, param_index_j,
                imu_pre_integration_
            );
        }
    }

    // move forward:
    last_key_frame_ = current_key_frame_;

    return true;
}
```

ceres 中添加残差块

FILE : lidar_localization/src/models/sliding_window/ceres_sliding_window.cpp

```
bool CeresSlidingWindow::Optimize() {
    static int optimization_count = 0;

    // get key frames count:
    const int N = GetNumParamBlocks();

    if (
        (kWindowSize + 1 <= N)
    ) {
        // TODO: create new sliding window optimization problem:
        ceres::Problem problem;

        // TODO: a. add parameter blocks:
        for ( int i = 1; i <= kWindowSize + 1; ++i) {
            auto &target_key_frame = optimized_key_frames_.at(N - i);

            ceres::LocalParameterization *local_parameterization = new
sliding_window::ParamPRVAG();

            // TODO: add parameter block:  添加待优化的参数快

            problem.AddParameterBlock(target_key_frame.prvag, 15,
local_parameterization);

            if( target_key_frame.fixed ) {
                    problem.SetParameterBlockConstant(target_key_frame.prvag);
            }
        }

        // TODO: add residual blocks:

        // b.1. marginalization constraint:
        if (
            !residual_blocks_.map_matching_pose.empty() &&
            !residual_blocks_.relative_pose.empty() &&
            !residual_blocks_.imu_pre_integration.empty()
        ) {
            auto &key_frame_m = optimized_key_frames_.at(N - kWindowSize - 1);
            auto &key_frame_r = optimized_key_frames_.at(N - kWindowSize - 0);

            const ceres::CostFunction *factor_map_matching_pose =
GetResMapMatchingPose(
                residual_blocks_.map_matching_pose.front()
            );
            const ceres::CostFunction *factor_relative_pose =
GetResRelativePose(
                residual_blocks_.relative_pose.front()
            );
            const ceres::CostFunction *factor_imu_pre_integration =
GetResIMUPreIntegration(
                residual_blocks_.imu_pre_integration.front()
            );
```

```
            sliding_window::FactorPRVAGMarginalization *factor_marginalization =
new sliding_window::FactorPRVAGMarginalization();

            factor_marginalization->SetResMapMatchingPose(
                factor_map_matching_pose,
                std::vector<double *>{key_frame_m.prvag}
            );
            factor_marginalization->SetResRelativePose(
                factor_relative_pose,
                std::vector<double *>{key_frame_m.prvag, key_frame_r.prvag}
            );
            factor_marginalization->SetResIMUPreIntegration(
                factor_imu_pre_integration,
                std::vector<double *>{key_frame_m.prvag, key_frame_r.prvag}
            );
            factor_marginalization->Marginalize(key_frame_r.prvag);

            // add marginalization factor into sliding window
            problem.AddResidualBlock(
                factor_marginalization,
                NULL,
                key_frame_r.prvag  //  一元边
            );

            residual_blocks_.map_matching_pose.pop_front();
            residual_blocks_.relative_pose.pop_front();
            residual_blocks_.imu_pre_integration.pop_front();
        }

        // TODO: b.2. map matching pose constraint:
        if ( !residual_blocks_.map_matching_pose.empty() ) {
            for ( const auto &residual_map_matching_pose:
residual_blocks_.map_matching_pose ) {
                auto &key_frame =
optimized_key_frames_.at(residual_map_matching_pose.param_index);

                sliding_window::FactorPRVAGMapMatchingPose
*factor_map_matching_pose = GetResMapMatchingPose(
                    residual_map_matching_pose
                );

                // TODO: add map matching factor into sliding window
                problem.AddResidualBlock(
                            factor_map_matching_pose,
                            NULL,     // loss_function
                            key_frame.prvag     //  一元边
                );
            }
        }

        // TODO: b.3. relative pose constraint:
        if ( !residual_blocks_.relative_pose.empty() ) {
            for ( const auto &residual_relative_pose:
residual_blocks_.relative_pose ) {
                auto &key_frame_i =
optimized_key_frames_.at(residual_relative_pose.param_index_i);
                auto &key_frame_j =
optimized_key_frames_.at(residual_relative_pose.param_index_j);
```

```
                sliding_window::FactorPRVAGRelativePose *factor_relative_pose =
GetResRelativePose(
                    residual_relative_pose
                );

                // TODO: add relative pose factor into sliding window
                problem.AddResidualBlock(
                        factor_relative_pose,
                        NULL,    // loss_function
                        key_frame_i.prvag, key_frame_j.prvag        //  二元边
                );
            }
        }

        // TODO: b.4. IMU pre-integration constraint
        if ( !residual_blocks_.imu_pre_integration.empty() ) {
            for ( const auto &residual_imu_pre_integration:
residual_blocks_.imu_pre_integration ) {
                auto &key_frame_i =
optimized_key_frames_.at(residual_imu_pre_integration.param_index_i);
                auto &key_frame_j =
optimized_key_frames_.at(residual_imu_pre_integration.param_index_j);

                sliding_window::FactorPRVAGIMUPreIntegration
*factor_imu_pre_integration = GetResIMUPreIntegration(
                    residual_imu_pre_integration
                );

                // TODO: add IMU factor into sliding window
                problem.AddResidualBlock(
                        factor_imu_pre_integration,
                        NULL,    // loss_function
                        key_frame_i.prvag,  key_frame_j.prvag           // 二元边
                );
            }
        }

        // solve:
        ceres::Solver::Summary summary;

        auto start = std::chrono::steady_clock::now();
        ceres::Solve(config_.options, &problem, &summary);
        auto end = std::chrono::steady_clock::now();
        std::chrono::duration<double> time_used = end-start;

        // prompt:
        LOG(INFO) << "------ Finish Iteration " << ++optimization_count << " of
Sliding Window Optimization -------" << std::endl
                  << "Time Used: " << time_used.count() << " seconds." <<
std::endl
                  << "Cost Reduced: " << summary.initial_cost -
summary.final_cost << std::endl
                  << summary.BriefReport() << std::endl
                  << std::endl;

        return true;
    }
```

```
    return false;
}
```

二. 运行及性能评估

代码运行命令：

```
roslaunch lidar_localization lio_localization.launch
```

播放数据集命令：

```
rosbag  play kitti_lidar_only_2011_10_03_drive_0027_synced.bag
```

保存数据：

```
rosservice call /save_odometry
```

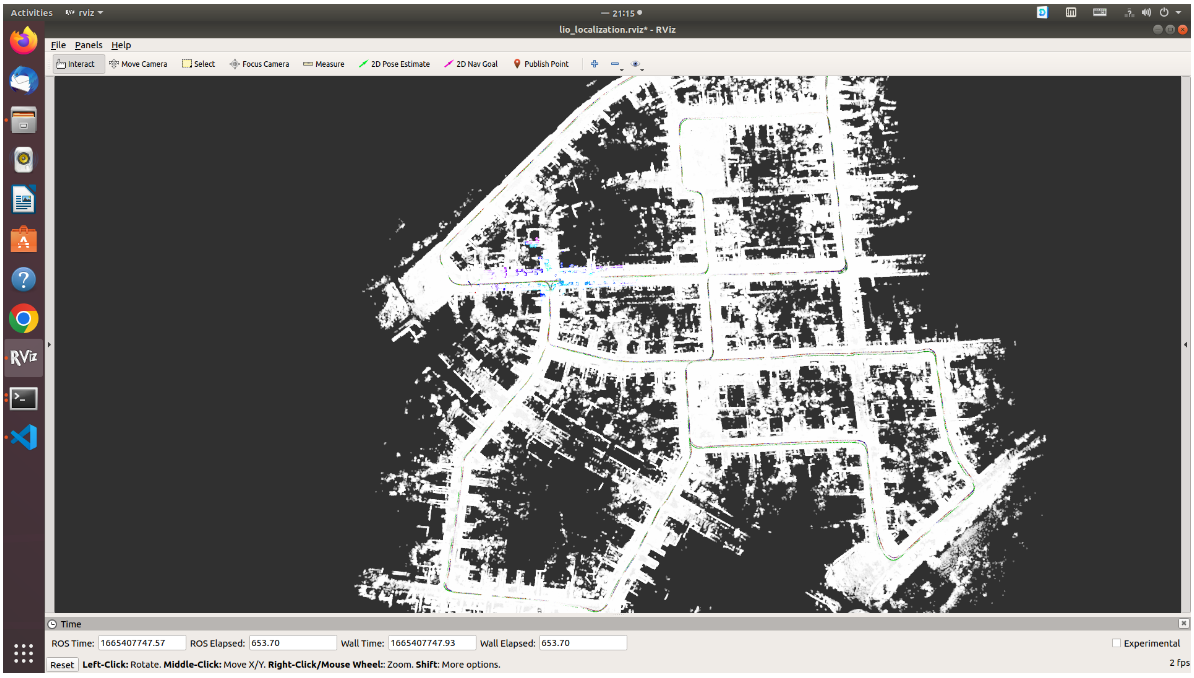
上述ROS Service会生成所需的rajectory Data位于:

```
trajectory Data: src/lidar_localization/slam_data/trajectory
```

evo工具运行命令：

```
evo_ape kitti ground_truth.txt optimized.txt -r full --plot --plot_mode xyz
```

RVIZ效果



EKF与因子图优化方法EVO指标比较

因子图优化：

```
   max	3.414162
  mean	0.982082
median	0.940413
   min	0.000001
  rmse	1.119414
   sse	5672.725224
   std	0.537217
```

EKF：

```
   max	1.063830
  mean	0.248405
median	0.193894
   min	0.016452
  rmse	0.299318
   sse	391.693699
   std	0.166992
```

EKF的指标要比图优化的指标要好，但是这并不一定说明图优化的效果不好，原因是KITTI数据集自身也存在一定的问题。

不同滑窗长度比较

滑窗长度10

```
   max	3.302303
  mean	1.273325
median	1.270163
   min	0.000001
  rmse	1.409168
   sse	8989.507795
   std	0.603652
```

滑窗长度20

```
   max	3.414162
  mean	0.982082
median	0.940413
   min	0.000001
  rmse	1.119414
   sse	5672.725224
   std	0.537217
```

滑窗长度30

```
   max    3.661999
  mean   1.512391
median   1.546260
   min   0.000001
  rmse   1.608281
   sse   11714.571538
   std   0.547030
```

1.滑动窗口的长度也是一个关键的因素，过高或者过低的窗口长度会造成精度的降低，选取适当滑窗大小是有必要的，能够直接影响最后的性能。

2.EKF的滤波方法就相当于滑动窗口为1的情况，

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

EKF与因子图优化方法EVO指标比较

因子图优化：

```
       max   4.554999
      mean   2.480754
    median   2.732886
       min   0.000001
      rmse   2.873453
       sse   19353.784484
       std   1.450032
```

EKF：

```
   max   1.116956
  mean   0.621030
median   0.606449
   min   0.118306
  rmse   0.674251
   sse   276.405189
   std   0.262556
```

不同滑窗长度比较

滑窗长度10

```
   max   4.599473
  mean   2.488457
median   2.740690
   min   0.000001
  rmse   2.878093
   sse   19424.624099
   std   1.446031
```

滑窗长度20

```
   max   4.554999
  mean   2.480754
median   2.732886
   min   0.000001
  rmse   2.873453
   sse   19353.784484
   std   1.450032
```

滑窗长度30

```
   max   4.546941
  mean   2.472068
median   2.723494
   min   0.000001
  rmse   2.864954
   sse   19255.878474
   std   1.448048
```

总结和思考

1. 为什么在自采的数据集中，滑窗的指标也是比EKF的指标要差呢？