vins_estimator

- rosNodeTest.cpp
  - 读取配置文件的参数
    readParameters(config_file);
    - readParameters()将配置文件的参数读取到parameters.h中
  - estimator.setParameter()
    - 设置estimator的外参、td、g
    - 设置FeatureManager的旋转外参
    - 设置ProjectionFactor的协方差（此处是协方差的开方）
    - 设置FeatureTracker的内参
    - 如果是多线程，processMeasurements()会一直工作
  - 让发布者注册话题
    registerPub(ros::NodeHandle &n)
    [http://wiki.ros.org/sensor_msgs sensor...](http://wiki.ros.org/sensor_msgs)
  - 订阅者订阅话题
    - imu_callback
      - 从消息中获取时间戳、线加速度和角加速度
      - 将imu信息输入到estimator中estimator.inputIMU(t, acc, gyr)
        - 将值加到accBuf和gyrBuf中
        - 根据上一帧的pvq和imu的输入来更新此时的pvq
          fastPredictIMU(t, linearAcceleration, angularVelocity)
          - $$\begin{aligned} p_{b_{i+1}}^w &= p_{b_i}^w + v_{b_i}^w \delta t + \frac{1}{2}\bar{a}_i \delta t^2 \\ v_{b_{i+1}}^w &= v_{b_i}^w + \bar{\hat{a}}_i \delta t \\ q_{b_{i+1}}^w &= q_{b_i}^w \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\bar{\omega}_i \delta t \end{bmatrix} \\ \bar{\hat{a}}_i &= \frac{1}{2}[q_i(\hat{a}_i - b_{a_i}) - g^w + q_{i+1}(\hat{a}_{i+1} - b_{a_i}) - g^w] \\ \bar{\widehat{\omega}}_i &= \frac{1}{2}(\widehat{\omega}_i + \widehat{\omega}_{i+1}) - b_{\omega_i} \end{aligned}$$
        - 发布最新的pvq信息
          pubLatestOdometry(latest_P, latest_Q, latest_V, t)
    - feature_callback
      - 获得点云的id、cameraid、3d坐标、像素坐标、像素速度、时间戳信息
      - 将特征点信息输入到estimator中
        estimator.inputFeature(t, featureFrame)
        - 将值加入到featureBuf中
        - processMeasurements()
    - img0_callback
      - 获得左目的图片消息，存到rosNodeTest的img0_buf中
    - img1_callback
      - 获得右目的图片消息，存到rosNodeTest的img1_buf中
    - restart_callback
      重启estimator，重新设置参数
      - estimator.clearState()
        把buf清空，参数设置成初始值

            - estimator.setParameter()
        - imu_switch_callback
            - 更改estimator中是否使用imu选项
              estimator.changeSensorType()
                - 如果现在使用了imu，要重启estimator
        - cam_switch_callback
            - 更改estimator中是否使用imu选项
              estimator.changeSensorType()
    - 将图像送给estimator
      std::thread sync_thread{sync_process}
      不断执行
        - 双目
            - 从img0_buf和img1_buf中判断两帧的时间差不超过0.003s
            - 通过getImageFromMsg()获得左右目的cv::Mat图像
            - estimator.inputImage(time, image0, image1)
        - 单目
            - 从img0_buf中取第一帧，getImageFromMsg()获得图像
            - estimator.inputImage(time, image)
                - featureFrame = featureTracker.trackImage(t, _img)
                    - cv::calcOpticalFlowPyrLK()
                    - 反向追踪
                    - reduceVector()把没追踪到的点除去
                    - 使特征点分布均匀
                      setMask()
                        - 设置mask图
                        - 将当前追踪的点按追踪次数降序排
                        - 清空cur_pts、ids、track_cnt
                        - 通过画圈的方式使特征点均匀，重新填充cur_pts、ids、track_cnt
                    - cv::goodFeaturesToTrack()提取shi-tomas角点
                    - 把新角点增加进cur_pts等
                    - undistortedPts(cur_pts, m_camera[0])将像素坐标恢复成归一化坐标放到cur_un_pts中
                    - ptsVelocity(ids, cur_un_pts, cur_un_pts_map, prev_un_pts_map)求像素速度（实际上是在归一化平面求）把结果放入pts_velocity
                        - 把id和当前帧像素坐标对应
                        - 通过前后两帧像素差除以时间求速度
                    - 左右目光流追踪
                    - 反向左右目光流追踪
                    - 和单目相似，更新右目信息
                    - drawTrack()
                    - prev=cur
                      hasPrediction = false
                    - 将归一化坐标、真正的像素坐标、归一化平面xy的速度封装成featureFrame，如果有右目，再来一次，返回featureFrame
                - 获得左目图像，pubTrackImage(imgTrack, t)发布消息用于可视化
                - featureBuf.push(make_pair(t, featureFrame))
                - processMeasurements()

- feature为当前帧的特征点，加上时间戳
  curTime是当前时间
- 循环等待当前时刻的imu数据到来
- 把前一帧和当前帧之间的buf数据取出放Vector中
  getIMUInterval(prevTime, curTime, accVector, gyrVector)
- initFirstIMUPose(accVector)
  vins中的坐标系变换及g2r函数_乌龟抓...
  - 取这段时间的平均加速度作为重力
  - 让加速度和重力对其，修正Rs[0]
- 根据上一图像帧的位姿和之间的imu数据进行粗略的预积分得到现在图像帧的pvq
  processIMU(accVector[i].first, dt, accVector[i].second, gyrVector[i].second)
  - 把dt，加速度、角速度加到当前帧的pre_integrations中，再加入到当前帧的对应的buf中
  - $$\begin{aligned} p_{b_{i+1}}^{w} &= p_{b_i}^{w} + v_{b_i}^{w}\delta t + \frac{1}{2}\bar{a}_i\delta t^2 \\ v_{b_{i+1}}^{w} &= v_{b_i}^{w} + \overline{\hat{a}}_i\delta t \\ q_{b_{i+1}}^{w} &= q_{b_i}^{w} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\bar{\omega}_i\delta t \end{bmatrix} \\ \overline{\hat{a}}_i &= \frac{1}{2}[q_i(\hat{a}_i - b_{a_i}) - g^w + q_{i+1}(\hat{a}_{i+1} - b_{a_i}) - g^w] \\ \overline{\widehat{\omega}}_i &= \frac{1}{2}(\widehat{\omega}_i + \widehat{\omega}_{i+1}) - b_{\omega_i} \end{aligned}$$
- processImage(feature.second, feature.first)
  - addFeatureCheckParallax(frame_count, image, td)判断边缘化最老帧还是次新帧
    - 更新feature数组
    - compensatedParallax2(it_per_id, frame_count)算视差
      - 在归一化平面计算点的距离
    - 通过视差判断是否为关键帧
  - 将当前帧封装成ImageFrame，装进all_image_frame
  - 获得特征点在两帧下的归一化坐标
    getCorresponding(frame_count - 1, frame_count)
  - CalibrationExRotation(corres, pre_integrations[frame_count]->delta_q, calib_ric)
    - 根据对极约束求位姿，8点法
      solveRelativeR(corres)
      - 通过两帧的像素坐标(归一化坐标XY)求本质矩阵
      - 将E（-E）分解成R和t
        decomposeE()
        - $$\begin{gathered} E = U\operatorname{diag}(\sigma, \quad \sigma, \quad 0)V^T \\ t_1 = U(:,2) \quad R_1 = UR_Z\left(\frac{\pi}{2}\right)V^T \\ t_2 = -U(:,2) \quad R_2 = UR_Z^T\left(\frac{\pi}{2}\right)V^T \\ R_Z\left(\frac{\pi}{2}\right) = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, R_Z^T\left(\frac{\pi}{2}\right) = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{gathered}$$

- 检验四组解
  testTriangulation(ll, rr, R1, t1)获得真正的R
  - 通过cv::triangulatePoints()检验深度为正的点的比例
- 求qbc，详细看《手写VIO》第七讲第10页
  四元数_陋室逢雨的博客-CSDN博客

$$\mathbf{q}_{b_k b_{k+1}} \otimes \mathbf{q}_{bc} = \mathbf{q}_{bc} \otimes \mathbf{q}_{c_k c_{k+1}}$$

$$\left(\left[\mathbf{q}_{b_k b_{k+1}} \right]_L - \left[\mathbf{q}_{c_k c_{k+1}} \right]_R \right) \mathbf{q}_{bc} = \mathbf{Q}_{k+1}^k \cdot \mathbf{q}_{bc} = \mathbf{0}$$

$$\mathcal{L}(q_a) q_b = \begin{bmatrix} s_a & -z_a & y_a & x_a \\ z_a & s_a & -x_a & y_a \\ -y_a & x_a & s_a & z_a \\ -x_a & -y_a & -z_a & s_a \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \\ s_b \end{bmatrix}$$

$$\begin{bmatrix} w_1^0 \cdot \mathbf{Q}_1^0 \\ w_2^1 \cdot \mathbf{Q}_2^1 \\ \vdots \\ w_N^{N-1} \cdot \mathbf{Q}_N^{N-1} \end{bmatrix} \mathbf{q}_{bc} = \mathbf{Q}_N \cdot \mathbf{q}_{bc} = \mathbf{0}$$

$$w_{k+1}^k = \begin{cases} 1, & r_{k+1}^k < \text{threshold} \\ \frac{\text{threshold}}{r_{k+1}^k}, & \text{otherwise} \end{cases}$$

$$\operatorname{tr}(\mathbf{R}) = 1 + 2\cos\theta$$

$$r_{k+1}^k = \operatorname{acos}\left(\left(\operatorname{tr}\left(\hat{\mathbf{R}}_{bc}^{-1} \mathbf{R}_{b_k b_{k+1}}^{-1} \hat{\mathbf{R}}_{bc} \mathbf{R}_{c_k c_{k+1}} \right) - 1 \right) / 2 \right)$$

- 如果成功把结果给calib_ric_result
- 初始化
  - 单目+imu
    - initialStructure()
      - imu激励是否足够
        - 计算所有帧的加速度的标准差
      - 创建Q、T、sfm_f、sfm_tracked_points
      - 把feature信息填充到sfm_f中
      - 确定参考帧l，求最新帧到它的位姿
        relativePose(relative_R, relative_T, l)
        - 遍历滑窗，获得第i帧和最新帧的共视点
        - 计算共视点的平均视差并判断
        - 对极约束，五点法求位姿
          solveRelativeRT(corres, relative_R, relative_T)
          - RANSAC求解本质矩阵E
          - 对极约束恢复位姿
            cv::recoverPose(E, ll, rr, cameraMatrix, rot, trans, mask)
      - SFM
        sfm.construct(frame_count + 1, Q, T, l, relative_R, relative_T,sfm_f, sfm_tracked_points)
        - 创建c_XXX数组表示第l帧相机在别的帧相机系下的表示
        - 三角化l和最新帧
          triangulateTwoFrames()
          - 遍历特征点，如果是这两个帧的共视点triangulatePoint()，求出在l相机系的空间坐标，更新sfm_f

    - pnp求l+1帧的位姿
      solveFrameByPnP()
      - 获得特征点的3d坐标和2d坐标
      - 将cv::Eigen转化为cv::Mat
      - cv::solvePnP()求旋转向量和平移向量
    - 三角化l+1帧和最新帧
      triangulateTwoFrames()
    - 三角化l帧和l+1帧
    - pnp求l-1帧
    - 三角化l-1帧和l帧
    - triangulatePoint()三角化剩余点，更新sfm_f
    - 全局ba
      - 用ceres求解，添加参数块，将先验设为恒定
      - ReprojectionError3D定义残差
        - 重投影-光流
      - 添加残差块，ceres求解
      - 更新滑窗每一帧的q和T，填充sfm_tracked_points
  - pnp求所有帧
    - 如果当前帧在滑窗内，更新ImageFrame的R和T
    - 如果当前帧不在滑窗内，找这个帧被三角化的特征点，获得3d、2d坐标
    - cv::solvePnP()求解该帧的位姿
    - 更新该帧ImageFrame的R和T
  - 视觉imu对齐
    visualInitialAlign()
    - VisualIMUAlignment(all_image_frame, Bgs, g, x)
      - solveGyroscopeBias(all_image_frame, Bgs)
        - $$\mathbf{q}_{b_k b_{k+1}} \approx \hat{\mathbf{q}}_{b_k b_{k+1}} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{J}^{\mathbf{q}}_{b^g}\delta\mathbf{b}^g \end{bmatrix}$$
          $$\mathbf{q}_{b_k b_{k+1}} = \mathbf{q}^{-1}_{c_0 b_k} \otimes \mathbf{q}_{c_0 b_{k+1}}$$
          $$\mathbf{J}^{\mathbf{q}}_{b^g}\delta\mathbf{b}^g = 2\hat{\mathbf{q}}^{-1}_{b_k b_{k+1}} \otimes \mathbf{q}_{b_k b_{k+1}}$$
        - ldlt求解
          更新滑窗内的Bgs数组
        - 所有帧重新预积分
          repropagate(Vector3d::Zero(), Bgs[0])
      - LinearAlignment(all_image_frame, g, x)
        - $$\mathcal{X}_I = \left[\mathbf{v}_0^{b_0}, \mathbf{v}_1^{b_1}, \cdots \mathbf{v}_n^{b_n}, \mathbf{g}^{c_0}, s\right]^\top$$
          $$\mathbf{H}^{b_k}_{b_{k+1}} \mathcal{X}_I^k = \hat{\mathbf{z}}^{b_k}_{b_{k+1}}$$
          $$\mathbf{H}^{b_k}_{b_{k+1}} = \begin{bmatrix} -\mathbf{I}\Delta t_k & \mathbf{0} & \frac{1}{2}\mathbf{R}_{b_k c_0}\Delta t_k^2 & \mathbf{R}_{b_k c_0}(\bar{\mathbf{p}} \\ -\mathbf{I} & \mathbf{R}_{b_k c_0}\mathbf{R}_{c_0 b_{k+1}} & \mathbf{R}_{b_k c_0}\Delta t_k & \end{bmatrix}$$
          $$\hat{\mathbf{z}}^{b_k}_{b_{k+1}} = \begin{bmatrix} \hat{\boldsymbol{\alpha}}_{b_k b_{k+1}} - \mathbf{p}_{bc} + \mathbf{R}_{b_k c_0}\mathbf{R}_{c_0 b_{k+1}}\mathbf{p}_{bc} \\ \hat{\boldsymbol{\beta}}_{b_k b_{k+1}} \end{bmatrix}$$

- RefineGravity(all_image_frame, g, x)
  - $$\hat{\mathbf{g}}^{c_0} = \|g\| \cdot \hat{\bar{\mathbf{g}}}^{c_0} + w_1 \vec{b}_1 + w_2 \vec{b}_2$$
    $$\vec{b}_1 = \begin{cases} \left(\hat{\bar{\mathbf{g}}}^{c_0} \times [1,0,0]\right), & \hat{\bar{\mathbf{g}}}^{c_0} \neq [1,0,0]^\top \\ \left(\hat{\bar{\mathbf{g}}}^{c_0} \times [0,0,1]\right), & \text{otherwise} \end{cases}$$
    $$\vec{b}_2 = \hat{\bar{\mathbf{g}}}^{c_0} \times \vec{b}_1$$
    $$\mathcal{X}_I^k = \begin{bmatrix} \mathbf{v}_k^{b_k} \\ \mathbf{v}_{k+1}^{b_{k+1}} \\ \mathbf{g}^{c_0} \\ s \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{v}_k^{b_k} \\ \mathbf{v}_{k+1}^{b_{k+1}} \\ \mathbf{w}^{c_0} \\ s \end{bmatrix}$$
    $$\mathbf{H}_{b_{k+1}}^{b_k} = \begin{bmatrix} -\mathbf{I}\Delta t_k & \mathbf{0} & \frac{1}{2}\mathbf{R}_{b_k c_0}\Delta t_k^2 & \mathbf{R}_{b_k c_0} \\ -\mathbf{I} & \mathbf{R}_{b_k c_0}\mathbf{R}_{c_0 b_{k+1}} & \mathbf{R}_{b_k c_0}\Delta t_k & \end{bmatrix}$$
    $$\hat{\mathbf{z}}_{b_{k+1}}^{b_k} = \begin{bmatrix} \boldsymbol{\alpha}_{b_k b_{k+1}} - \mathbf{p}_{bc} + \mathbf{R}_{b_k c_0}\mathbf{R}_{c_0 b_{k+1}}\mathbf{p}_{bc} - \frac{1}{2}\mathbf{R}_{b_k c_0}\Delta t_k^2 \| \\ \boldsymbol{\beta}_{b_k b_{k+1}} - \mathbf{R}_{b_k c_0}\Delta t_k \|g\| \cdot \hat{\bar{\mathbf{g}}}^{c_0} \end{bmatrix}$$
- 用尺度、优化后的速度更新滑窗内的Ps、Rs、Vs，并把坐标系从第l帧相机系转到世界系
  - 对滑窗内的帧重新预积分
    repropagate(Vector3d::Zero(), Bgs[i])
  - 求将第l帧g变换到世界系的g的旋转，把参考系从第l帧变到第0帧body系（世界系），现在的Rs、Ps、Vs是第i帧imu在第0帧body系
  - clearDepth()
    清楚feature数组FeaturePerId的深度
  - 重新求深度（相对于世界系）
    triangulate(frame_count, Ps, Rs, tic, ric)
    - 获得首次观测到这个特征点的帧
      获得左目右目的像素坐标
      triangulatePoint()三角化求世界系的空间坐标
    - feature数组的FeaturePerId的深度是在首次观测到该特征点的帧的相机系的z
    - 三角化的两帧时首次观测到该特征点的帧和它的下一帧
    - 详细看手写VIO第6讲25页
      $$\begin{bmatrix} u_1\mathbf{P}_{1,3}^\top - \mathbf{P}_{1,1}^\top \\ v_1\mathbf{P}_{1,3}^\top - \mathbf{P}_{1,2}^\top \\ \vdots \\ u_n\mathbf{P}_{n,3}^\top - \mathbf{P}_{n,1}^\top \\ v_n\mathbf{P}_{n,3}^\top - \mathbf{P}_{n,2}^\top \end{bmatrix} y = \mathbf{0}$$
    - SVD求解
- optimization()
- updateLatestStates()
- slideWindow()
- slideWindow()

- 双目+imu

- initFramePoseByPnP(frame_count, Ps, Rs, tic, ric)
  - 遍历滑窗特征点，获得世界系3d坐标和最新帧的像素坐标
  - solvePoseByPnP()求位姿
    - 主要是格式转换
      旋转向量和旋转矩阵转换
      cv::solvePnP()
  - 更新最新帧的Ps、Rs
- triangulate(frame_count, Ps, Rs, tic, ric)
- 更新all_image_frame的RT（all_image_frame都在滑窗内）
- solveGyroscopeBias(all_image_frame, Bgs)
- repropagate(Vector3d::Zero(), Bgs[i])重新预积分
- optimization()
- updateLatestStates()
- slideWindow()

- 仅双目
  - initFramePoseByPnP(frame_count, Ps, Rs, tic, ric)
  - triangulate(frame_count, Ps, Rs, tic, ric)
  - optimization()
  - updateLatestStates()
  - slideWindow()
- 把当前帧信息pvqb给下一帧

- initFramePoseByPnP(frame_count, Ps, Rs, tic, ric)
- 估计滑窗特征点的深度
  三角化滑窗中的特征点，把结果给feature数组featureperid的estimated_depth，这个深度是对于首次观测到它的帧的相机系
  triangulate(frame_count, Ps, Rs, tic, ric)，把
- optimization()
  - vector2double()
    - 填充para_Pose、para_SpeedBias、para_Ex_Pose、para_Feature、para_Td
      其中para_Feature是逆深度，取自于featurePerId.estimated_depth的倒数，且只有4帧及以上追踪到这个特征点才考虑
  - 添加参数块
  - 添加残差块（imu、视觉、先验）
  - ceres::Solve()
  - double2vector()
    - 把double数组填进Ps、Rs等
    - setDepth(dep)
      - featurePerId的estimated_depth，通过深度正负设置featurePerId的solve_flag为1或2，为之后的removeFailures()使用
  - 边缘化
    - 边缘化最老帧
      - vector2double()
      - 把最老帧有关的因子 （先验因子、imu因子、视觉因子和drop_set增加进marginalization_info中

- 求雅克比和残差
  marginalization_info->preMarginalize()
- 构建H和b，反解出J和b
  marginalization_info->marginalize()
- 填充addr_shift
- 新值赋给旧值

- 边缘化次新帧
  - vector2double()
  - 边缘化先验因子中和次新帧有关的变量，通过drop_set的形式添加进marginalization_info中
  - marginalization_info->preMarginalize()
  - marginalization_info->marginalize()
  - 填充addr_shift
  - 新值赋给旧值

- 外点检测
  outliersRejection(removeIndex)
  - 获得这个特征点首次观测的帧和共视帧
  - 求重投影误差
    reprojectionError
  - 如果是双目，要增加和共视帧右目的重投影误差
  - 平均误差大于阈值，认为是外点
- 去除外点
  removeOutlier(removeIndex)
  - 根据removeIndex把feature中对应的特征点去除
- removeOutliers(removeIndex)
  - 在featureTracker中把prev_pts、ids、track_cnt对应的点去除
- 预测下一帧特征点出现的位置
  predictPtsInNextFrame()
  - 获得的当前帧和上一帧的位姿
    getPoseInWorldFrame()
  - 假设下一帧的位姿和当前帧到前一帧的位姿相同，进而求下一帧位姿
  - 通过位姿重投影，形成预测点
  - setPrediction(predictPts)
    - 填充featuretracker的predict_pts，存的是真正的像素坐标，用于下一帧光流追踪
- slideWindow()
  - 最老帧
    - 信息前移
      Ps、Rs、Headers、pre_integrations、dt_buf、linear_acceleration_buf、angular_velocity_buf、Vs、Bas、Bgs
    - 第10帧信息赋值给第11帧，重新创建第11帧的预积分，清空第11帧的buf
    - 从all_image_frame中删除最老帧之前的帧
    - slideWindowOld()
      - removeBackShiftDepth(R0, P0, R1, P1)
        - start_frame--
        - 更新featurePerId和featurePerFrame
        - 重新计算featurePerId的estimated_depth

- removeBack()
  - start_frame--
  - 更新featurePerId和featurePerFrame
- 次新帧
  - 用最新帧信息覆盖次新帧
  - 对于imu，最新帧的buf直接拼接到次新帧上（此时的次新帧是最新帧，信息已被覆盖）
  - 对第11帧重新创建预积分，清空第11帧的buf
  - slideWindowNew()
    - removeFront(frame_count)
      - 更新featurePerId和featurePerFrame
- 把滑窗中深度为负的点移除
  removeFailures()
- 将滑窗中的Ps填充进key_poses数组中，用于可视化
- 更新last_RP、last_R0P0
- updateLatestStates()
  - 更新latest_xxx变量，用于fastpredictimu()
  - 用最新的数据不断fastpredictimu()
- printStatistics()输出数据
- 向rziv发布

- IntegrationBase
  - repropagate()
    - 把预积分相关的变量设为初始值
    - 遍历所有imu帧，预积分
      propagate(dt_buf[i], acc_buf[i], gyr_buf[i])
      - midPointIntegration(_dt, acc_0, gyr_0, _acc_1, _gyr_1, delta_p, delta_q, delta_v,
        linearized_ba, linearized_bg,
        result_delta_p, result_delta_q, result_delta_v,
        result_linearized_ba, result_linearized_bg, 1)
        - $$\begin{gathered}\bar{\hat{a}}_i = \frac{1}{2}\left[q_i(\hat{a}_i - b_{a_i}) + q_{i+1}(\hat{a}_{i+1} - b_{a_i})\right] \\ \bar{\hat{w}}_i = \frac{1}{2}(\hat{w}_i + \hat{w}_{i+1}) - b_{w_i} \\ \hat{\alpha}_{i+1}^{b_k} = \hat{\alpha}_i^{b_k} + \hat{\beta}_i^{b_k}\delta t + \frac{1}{2}\bar{\hat{a}}_i\delta t^2 \\ \hat{\beta}_{i+1}^{b_k} = \hat{\beta}_i^{b_k} + \bar{\hat{a}}_i\delta t \\ \hat{\gamma}_{i+1}^{b_k} = \hat{\gamma}_i^{b_k} \otimes \hat{\gamma}_{i+1}^{i} = \hat{\gamma}_i^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\bar{\hat{w}}_i\delta t \end{bmatrix}\end{gathered}$$
        - $$\begin{bmatrix} \delta\boldsymbol{\alpha}_{b_{k+1}b'_{k+1}} \\ \delta\boldsymbol{\gamma}_{b_{k+1}b'_{k+1}} \\ \delta\boldsymbol{\beta}_{b_{k+1}b'_{k+1}} \\ \delta\mathbf{b}_{k+1}^{a} \\ \delta\mathbf{b}_{k+1}^{g} \end{bmatrix} = \mathbf{F}\begin{bmatrix} \delta\boldsymbol{\alpha}_{b_k b'_k} \\ \delta\boldsymbol{\gamma}_{b_k b'_k} \\ \delta\boldsymbol{\beta}_{b_k b'_k} \\ \delta\mathbf{b}_{k}^{a} \\ \delta\mathbf{b}_{k}^{g} \end{bmatrix} + \mathbf{G}\begin{bmatrix} \mathbf{n}_k^a \\ \mathbf{n}_k^g \\ \mathbf{n}_{k+1}^a \\ \mathbf{n}_{k+1}^g \\ \mathbf{n}_{\mathbf{b}_k^a} \\ \mathbf{n}_{\mathbf{b}_k^g} \end{bmatrix}$$

-
$$\mathbf{F}=\begin{bmatrix}\mathbf{I} & \mathbf{f}_{12} & \mathbf{I}\delta t & -\frac{1}{4}\left(\mathbf{q}_{b_ib_k}+\mathbf{q}_{b_ib_{k+1}}\right)\delta t^2 & \mathbf{f}_{15}\\ \mathbf{0} & \mathbf{I}-[\boldsymbol{\omega}]_\times & \mathbf{0} & \mathbf{0} & -\mathbf{I}\delta t\\ \mathbf{0} & \mathbf{f}_{32} & \mathbf{I} & -\frac{1}{2}\left(\mathbf{q}_{b_ib_k}+\mathbf{q}_{b_ib_{k+1}}\right)\delta t & \mathbf{f}_{35}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}\end{bmatrix}$$
$$\mathbf{G}=\begin{bmatrix}\frac{1}{4}\mathbf{q}_{b_ib_k}\delta t^2 & \mathbf{g}_{12} & \frac{1}{4}\mathbf{q}_{b_ib_{k+1}}\delta t^2 & \mathbf{g}_{14} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \frac{1}{2}\mathbf{I}\delta t & \mathbf{0} & \frac{1}{2}\mathbf{I}\delta t & \mathbf{0} & \mathbf{0}\\ \frac{1}{2}\mathbf{q}_{b_ib_k}\delta t & \mathbf{g}_{32} & \frac{1}{2}\mathbf{q}_{b_ib_{k+1}}\delta t & \mathbf{g}_{34} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}\delta t & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}\delta t\end{bmatrix}$$
-
$$\begin{aligned}J_{k+1}&=FJ_k,\quad J_0=I\\ P_{k+1}&=FP_kF^T+GQG^T,P_0=0\\ Q^{18\times18}&=\left(\sigma_a^2,\sigma_w^2,\sigma_a^2,\sigma_w^2,\sigma_{b_a}^2,\sigma_{b_w}^2\right)\end{aligned}$$
- 新值赋给旧值

- evaluate()
  -
$$\begin{aligned}\alpha_{b_{k+1}}^{b_k}&\approx\hat{\alpha}_{b_{k+1}}^{b_k}+J_{b_a}^{\alpha}\delta b_a+J_{b_w}^{\alpha}\delta b_w\\ \beta_{b_{k+1}}^{b_k}&\approx\hat{\beta}_{b_{k+1}}^{b_k}+J_{b_a}^{\beta}\delta b_a+J_{b_w}^{\beta}\delta b_w\\ \gamma_{b_{k+1}}^{b_k}&\approx\hat{\gamma}_{b_{k+1}}^{b_k}\otimes\begin{bmatrix}1\\ \frac{1}{2}J_{b_w}^{\gamma}\delta b_w\end{bmatrix}\end{aligned}$$
  -
$$\begin{bmatrix}\delta\alpha_{b_{k+1}}^{b_k}\\ \delta\theta_{b_{k+1}}^{b_k}\\ \delta\beta_{b_{k+1}}^{b_k}\\ \delta b_a\\ \delta b_g\end{bmatrix}=\begin{bmatrix}R_w^{b_k}\left(p_{b_{k+1}}^w-p_{b_k}^w-v_{b_k}^w\Delta t_k+\frac{1}{2}g^w\Delta t_k^2\right)-\alpha_{b_{k+1}}^{b_k}\\ 2\left[\gamma_{b_{k+1}}^{b_k}-1\otimes q_{b_k}^w\otimes q_{b_{k+1}}^w\right]_{xyz}\\ R_w^{b_k}\left(v_{b_{k+1}}^w-v_{b_k}^w+g^w\Delta t_k\right)-\beta_{b_{k+1}}^{b_k}\\ b_{a_{b_{k+1}}}-b_{a_{b_k}}\\ b_{\omega_{b_{k+1}}}-b_{\omega_{b_k}}\end{bmatrix}$$
  -
$$\left[p_{b_k}^w,q_{b_k}^w\right],\left[v_{b_k}^w,b_{a_k},b_{\omega_k}\right],\left[p_{b_{k+1}}^w,q_{b_{k+1}}^w\right],\left[v_{b_{k+1}}^w,b_{a_{k+1}},b_{\omega_{k+1}}\right]$$

- IMUFactor
  - evaluate()
    - 计算残差
      pre_integration->evaluate(Pi, Qi, Vi, Bai, Bgi,
      Pj, Qj, Vj, Baj, Bgj)

- $$J[0]^{15\times 7}=\left[\frac{\partial r_B}{\partial p_{b_k}^w},\frac{\partial r_B}{\partial q_{b_k}^w}\right]=\begin{bmatrix}-R_w^{b_k} & \left[R_w^{b_k}\left(p_{b_{k+1}}^w-p_{b_k}^w-v_{b_k}^w\Delta t_k+\frac{1}{2}g^w\Delta t_k^2\right)\right]^{\wedge}\\ 0 & -\mathcal{L}\left[{q_{b_{k+1}}^w}^{-1}\otimes q_{b_k}^w\right]\mathcal{R}\left[\gamma_{b_{k+1}}^{b_k}\right]\\ 0 & \left[R_w^{b_k}\left(v_{b_{k+1}}^w-v_{b_k}^w+g^w\Delta t_k\right)\right]^{\wedge}\\ 0 & 0\\ 0 & 0\end{bmatrix}$$
- $$J[1]^{15\times 9}=\left[\frac{\partial r_B}{\partial v_{b_k}^w},\frac{\partial r_B}{\partial b_{a_k}},\frac{\partial r_B}{\partial b_{w_k}}\right]=\begin{bmatrix}-R_w^{b_k}\Delta t & -J_{b_a}^{\alpha} & -J_{b_\omega}^{\alpha}\\ 0 & 0 & -\mathcal{L}\left[{q_{b_{k+1}}^w}^{-1}\otimes q_{b_k}^w\otimes\gamma_{b_{k+1}}^{b_k}\right]J_{b_\omega}^{\gamma}\\ -R_w^{b_k} & -J_{b_a}^{\beta} & -J_{b_\omega}^{\beta}\\ 0 & -I & 0\\ 0 & 0 & -I\end{bmatrix}$$
- $$J[2]^{15\times 7}=\left[\frac{\partial r_B}{\partial p_{b_{k+1}}^w},\frac{\partial r_B}{\partial q_{b_{k+1}}^w}\right]$$
- $$J[3]^{15\times 9}=\left[\frac{\partial r_B}{\partial v_{b_{k+1}}^w},\frac{\partial r_B}{\partial b_{a_{k+1}}},\frac{\partial r_B}{\partial b_{w_{k+1}}}\right]=\begin{bmatrix}0&0&0\\0&0&0\\R_w^{b_k}&0&0\\0&I&0\\0&0&I\end{bmatrix}$$

ProjectionFactor

- $$\left[p_{b_i}^w,q_{b_i}^w\right],\left[p_{b_{j'}}^w,q_{b_j}^w\right],\left[p_c^b,q_c^b\right],\lambda_l$$
- $$\mathbf{r}_c=\begin{bmatrix}\frac{x_{c_j}}{z_{c_j}}-u_{c_j}\\ \frac{y_{c_j}}{z_{c_j}}-v_{c_j}\end{bmatrix}$$
- $$\frac{\partial\mathbf{r}_c}{\partial\mathbf{f}_{c_j}}=\begin{bmatrix}\frac{1}{z_{c_j}} & 0 & -\frac{x_{c_j}}{z_{c_j}^2}\\ 0 & \frac{1}{z_{c_j}} & -\frac{y_{c_j}}{z_{c_j}^2}\end{bmatrix}$$
- $$J[0]^{3\times 7}=\left[\frac{\partial f_{c_j}}{\partial p_{b_i}^w},\frac{\partial f_{c_j}}{\partial q_{b_i}^w}\right]=\left[R_b^cR_w^{b_j}\quad -R_b^cR_w^{b_j}R_{b_i}^w\left(R_c^b\frac{1}{\lambda_l}\bar{P}_l^{c_i}+p_c^b\right)^{\wedge}\right]$$
$$J[2]^{3\times 7}=\left[\frac{\partial f_{c_j}}{\partial p_c^b},\frac{\partial f_{c_j}}{\partial q_c^b}\right]$$
$$=\begin{bmatrix}R_b^c\left(R_w^{b_j}R_{b_i}^w-I_{3\times 3}\right)\\ -R_b^cR_w^{b_j}R_{b_i}^wR_c^b\left(\frac{\bar{P}_l^{c_i}}{\lambda_l}\right)^{\wedge}+\left(R_b^cR_w^{b_j}R_{b_i}^wR_c^b\frac{\bar{P}_l^{c_i}}{\lambda_l}\right)^{\wedge}+\left\{R_b^c\left[R_w^{b_j}\left(R_{b_i}^wp_c^b+p_{b_i}^w-p_{b_j}^w\right)-p_c^b\right]\right\}^{\wedge}\end{bmatrix}^T$$
$$J[3]^{3\times 1}=\frac{\partial f_{c_j}}{\partial\lambda_l}=-R_b^cR_w^{b_j}R_{b_i}^wR_c^b\frac{\bar{P}_l^{c_i}}{\lambda_l^2}$$

- $$\Sigma_{vis}^{-1} = \left(\frac{1.5}{f} I_{2\times2}\right)^{-1} = \frac{f}{1.5} I_{2\times2}$$

- MarginalizationFactor
  - ResidualBlockInfo
    - Evaluate()
      - 调用对应残差的Evalua()函数求残差和雅克比
        cost_function->Evaluate(parameter_blocks.data(), residuals.data(), raw_jacobians)
      - 如果有核函数，残差和雅克比需要缩放
        Modeling Non-linear Least Squares &...
  - MarginalizationInfo
    - addResidualBlockInfo()
      - 填充parameter_block_size和parameter_block_idx，先记为0
    - getParameterBlocks()
      - 将边缘化保留的变量存入到keep_block_xxx中
    - preMarginalize()
      - 调用对应的Evaluate()求残差和雅克比
      - 填充parameter_block_data
    - marginalize()
      - 把边缘化掉的变量排在前面，在parameter_block_idx中体现
      - 构造H和b，舒尔补求新的H和b
        - $$A_{ij} = (\frac{\partial e}{\partial x_i})^T(\frac{\partial e}{\partial x_j})$$
          $$b_i = (\frac{\partial e}{\partial x_i})^T e$$
        - $$\begin{bmatrix} A_{mm} & A_{mr} \\ A_{rm} & A_{rr} \end{bmatrix} \begin{bmatrix} x_m \\ x_r \end{bmatrix} = \begin{bmatrix} b_m \\ b_r \end{bmatrix}$$
        - $$A_{rr} = A_{rr} - A_{rm}A_{mm}^{-1}A_{mr}$$
          $$b_r = b_r - A_{rm}A_{mm}^{-1}b_{mm}$$
      - 将H和b分解出J和e
        - $$A = VSV^T$$
          $$J = \sqrt{S}V^T$$
          $$e = \sqrt{S}^{-1}V^Tb$$
  - MarginalizationFactor
    - Evaluate()
      - 计算残差
        $$r = r_0 + J_x^r dx$$
      - 雅克比就是marginalization_info的雅克比