State Estimation with IMU/INS and camera information

Zhang Handuo

February 16, 2020

Abstract

This document is a report on Visual Inertial SLAM using an efficient method of IMU pre-integration. The pre-integration method combines high frequency IMU data as a single observation before fusion with camera images, resulting in a much reduced state and observation graph structure. An accurate map and robot path is hence obtainable in real-time. We present the pre-integration theory, including formulation of motion state, observation, uncertainty, observation model and the Jacobians.

Contents

Chapter 1

Introduction

In this work, we firstly introduce a perception system with a set of stereo camera with IMU. The integration with INS and GPS will be introduced in the next sections because their formulation is much simpler than that with IMU.

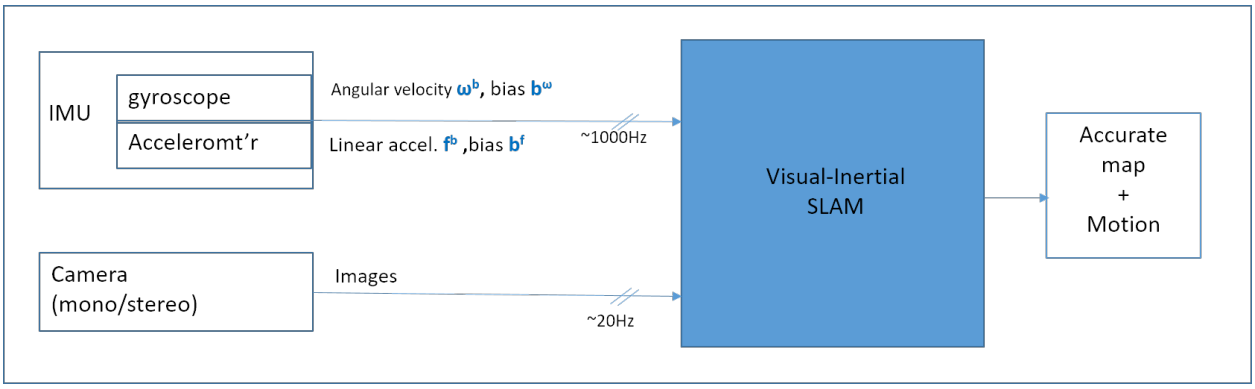


Figure 1.1: Visual Inertial System

States The state vector to be estimated contains the 3-D body position $\boldsymbol{p}^{\mathrm{w}}$, velocity $\boldsymbol{v}^{\mathrm{w}}$ and rotations in form of quaternion $\boldsymbol{q}^{\mathrm{w}}$ (converted from raw Euler angles $\boldsymbol{A}^{\mathrm{w}} = [\alpha, \beta, \gamma]$); as well as the M feature locations ($\boldsymbol{f}_i^{\mathrm{w}}$) representing visual landmarks where i = 1,...N.

IMU measurement model For the notation of coordinate transformation and vectors, superscript refers to the reference frame and here $\{\cdot\}^{\mathrm{w}}$ is the reference(**world**) frame. R_b^{w} and Ω_b^{w} are the rotation and angular rate matrices from body frame to world frame, respectively.

$$\begin{cases} \hat{\boldsymbol{a}}^b = \boldsymbol{a}^b + \boldsymbol{b}_a + \boldsymbol{R}_w^b \boldsymbol{g}^w + \boldsymbol{n}_a \\ \hat{\boldsymbol{\omega}}^b = \boldsymbol{\omega}^b + \boldsymbol{b}_\omega + \boldsymbol{n}_\omega \end{cases} \tag{1.1}$$

where $\hat{\boldsymbol{a}}^b$ and $\hat{\boldsymbol{\omega}}^b$ represent the measured acceleration and angular velocity information of IMU; $\boldsymbol{a}^b$ and $\boldsymbol{\omega}^b$ represent true acceleration and angular velocity of the body itself; $\boldsymbol{b}_a$ and $\boldsymbol{b}_\omega$ represent

the bias of the accelerator and gyroscope, modeled as the random walk process:

$$\begin{cases} \boldsymbol{n}_{\boldsymbol{b}_a} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\sigma}_{b_a}^2\right), \boldsymbol{n}_{\boldsymbol{b}_\omega} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\sigma}_{b_\omega}^2\right) \\ \dot{\boldsymbol{b}_a} = \boldsymbol{n}_{\boldsymbol{b}_a}, dot\boldsymbol{b}_\omega = \boldsymbol{n}_{\boldsymbol{b}_\omega} \end{cases} \tag{1.2}$$

where $\boldsymbol{R}_w^b$ is the coordinate rotation matrix from world coordinate to ego-body coordinate; $\boldsymbol{g}^w$ is the gravity acceleration vector under world coordinate; $\boldsymbol{n}_a \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\sigma}_a^2\right)$ and $\boldsymbol{n}_\omega \sim \left(\mathbf{0}, \boldsymbol{\sigma}_\omega^2\right)$ are the Gaussian white noise of the IMU sensor.

Naive visual inertial model The motion model based on IMU reading can be stated as

$$\begin{aligned} \text{time difference:} \quad & \triangle t = t_{t+1} - t_t \\ \text{acceleration:} \quad & \boldsymbol{a}_t^{\mathrm{w}} = R_{bt}^{\mathrm{w}}(\boldsymbol{a}_t^{\mathrm{b}} - \boldsymbol{b}_a) \\ \text{velocity:} \quad & \boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \boldsymbol{a}_t^{\mathrm{w}} \triangle t + \boldsymbol{g}^{\mathrm{w}} \triangle t \\ \text{position:} \quad & \boldsymbol{p}_{t+1} = \boldsymbol{p}_t + \boldsymbol{v}_t \triangle t + \boldsymbol{a}_t^{\mathrm{w}} \triangle t^2 \\ \text{rotation:} \quad & \boldsymbol{q}_{t+1} = \boldsymbol{q}_t \otimes \Omega_{bt}^{\mathrm{w}}(\hat{\boldsymbol{\omega}}_t^b - \boldsymbol{b}_\omega) \triangle t \end{aligned} \tag{1.3}$$

Chapter 2

Measurement Preprocessing

2.1 The original IMU problem

For a system composed of an IMU and stereo camera navigating with N camera poses, K IMU samples per image and M features, the naive VI problem includes robot poses at all IMU sample times.

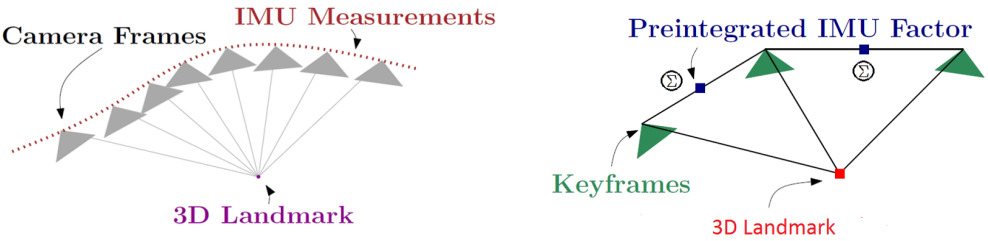
State Vector the state vector $\mathbf{X}$ is defined as:

$$\begin{aligned}\mathbf{x} =(&\overbrace{\mathbf{q}_{10}, \mathbf{p}_{10}, \mathbf{q}_{20}, \mathbf{p}_{20}, ..., \mathbf{p}_{K-1,0}, \mathbf{p}_{K-1,0}, \mathbf{q}_{0,1}, \mathbf{p}_{0,1}, ..., \mathbf{q}_{K-1,1}, \mathbf{p}_{K-1,1}, ...\mathbf{q}_{K-1,N-1}, \mathbf{p}_{K-1,N-1}, \mathbf{q}_{0N}, \mathbf{p}_{0N}}^{K\times(N-1) \text{ poses}},\\ &\overbrace{\mathbf{F}_1, \mathbf{F}_2, ..., \mathbf{F}_M}^{M \text{ features}},\\ &\overbrace{\mathbf{v}_{00}, \mathbf{v}_{10}..., \mathbf{v}_{K-1,0}, ..., \mathbf{v}_{0,N-1}, ..., \mathbf{v}_{K-1,N-1}, \mathbf{v}_{0N}}^{(K\times(N-1)+1) \text{ velocities}},\\ &\mathbf{g}^{\mathrm{n}}, \mathbf{A}_{u2c}, \mathbf{T}_{u2c}, \mathbf{b}_f, \mathbf{b}_w)^{\mathrm{T}}\end{aligned}$$

Observation Vector The Observation vector $\mathbf{Z}$ is defined as:

$$\begin{aligned}\mathbf{z}_{raw} &= (\mathbf{z}_{camera}, \mathbf{z}_{IMUraw}, \mathbf{z}_{Tv})'\\ &=(\overbrace{\mathbf{uv}_{11}, \mathbf{uv}_{21}, ..., \mathbf{uv}_{M1}, ..., \mathbf{uv}_{1N}, \mathbf{uv}_{2N}, ..., \mathbf{uv}_{MN}}^{M\times N \text{ pixels}},\\ &\quad\overbrace{\omega\mathbf{f}_{01}, \omega\mathbf{f}_{11}, ..., \omega\mathbf{f}_{(K-1)1}, ..., \omega\mathbf{f}_{0(N-1)}, \omega\mathbf{f}_{1(N-1)}, ..., \omega\mathbf{f}_{(K-1)(N-1)}}^{K\times(N-1) \text{ IMU readings}},\\ &\quad\overbrace{\mathbf{0}, \mathbf{0}, ..., \mathbf{0}}^{K\times(N-1) \text{ zero contraints}} \quad)^{\mathrm{T}}\end{aligned}$$

Clearly, such a large state space quickly becomes difficult to manage in practice. Further, each step of re-linearization, the integration from acceleration to velocity then to position has to be recomputed, which brings lots of additional time.



(a) (b)

Figure 2.1: (a) Camera+IMU (b) Inertial-delta: preintegrated information [2]

2.1.1 The Preintegration algorithm

Todd Lupton first proposed the Preintegration method in 2012 [3]: integrate a large number of high rate IMU observations into a single observation, making it faster and easier to deal with in a SLAM filter. IMU data is integrated in a body fixed frame that moves with the vehicle, transformation to reference frame only happens at the end of integration, hence the algorithm is referred to as **Pre-Integration**.

The reference frame in Preintegration visual inertial system is defined as the body frame at initial robot pose, instead of the traditional globally referenced frame, preventing the need to reintegrate the state dynamics at each optimization step.

2.2 Formulation of Preintegration

For time consecutive frames b_k and b_{k+1}, there exists several measurements in the time interval $[t_k, t_{k+1}]$. Given the bias estimation, we integrate them in the **local frame** b_K as:

$$\begin{cases} \boldsymbol{p}_{b_{k+1}}^{b_k} = \iint_{t\in[t_k,t_{k+1}]} \boldsymbol{R}_t^{b_k} \left(\hat{\boldsymbol{a}}_t - \boldsymbol{b}_{a_t}\right) dt^2 \\ \boldsymbol{v}_{b_{k+1}}^{b_k} = \int_{t\in[t_k,t_{k+1}]} \boldsymbol{R}_t^{b_k} \left(\hat{\boldsymbol{a}}_t - \boldsymbol{b}_{a_t}\right) dt \\ \boldsymbol{q}_{b_{k+1}}^{b_k} = \int_{t\in[t_k,t_{k+1}]} \frac{1}{2}\boldsymbol{\Omega}\left(\hat{\boldsymbol{\omega}}_t - \boldsymbol{b}_{\omega_t}\right) \boldsymbol{q}_t^{b_k} dt \end{cases} \tag{2.1}$$

where $\boldsymbol{p}$, $\boldsymbol{v}$ and $\boldsymbol{q}$ are respectively the incremental preintegration items for position, velocity and rotation (quaternion form). Here

$$\boldsymbol{\Omega}\left(\omega\right) = \begin{bmatrix} -\lfloor\omega\rfloor_\times & \omega \\ -\omega^{\mathrm{T}} & 0 \end{bmatrix}, \lfloor\omega\rfloor_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ -\omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{2.2}$$

The details can be found in Appendix (TODO).

Bias Correctioin of Incremental preintegration

If the estimation of bias changes minorly, we adjust $\boldsymbol{p}_{b_{k+1}}^{b_k}$, $\boldsymbol{v}_{b_{k+1}}^{b_k}$, and $\boldsymbol{q}_{b_{k+1}}^{b_k}$ by their first-order approximations with respect to the bias as

$$\begin{cases} \boldsymbol{p}_{b_{k+1}}^{b_k} \approx \hat{\boldsymbol{p}}_{b_{k+1}}^{b_k} + \boldsymbol{J}_{\boldsymbol{b}_{ak}}^{\boldsymbol{p}} \delta \boldsymbol{b}_a + \boldsymbol{J}_{\boldsymbol{b}_{\omega}}^{\boldsymbol{p}} \delta \boldsymbol{b}_{\omega_k} \\ \boldsymbol{v}_{b_{k+1}}^{b_k} \approx \hat{\boldsymbol{v}}_{b_{k+1}}^{b_k} + \boldsymbol{J}_{\boldsymbol{b}_{ak}}^{\boldsymbol{v}} \delta \boldsymbol{b}_a + \boldsymbol{J}_{\boldsymbol{b}_{\omega}}^{\boldsymbol{v}} \delta \boldsymbol{b}_{\omega_k} \\ \boldsymbol{q}_{b_{k+1}}^{b_k} \approx \hat{\boldsymbol{q}}_{b_{k+1}}^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} \boldsymbol{J}_{\boldsymbol{b}_{\omega_k}}^{\boldsymbol{q}} \delta \boldsymbol{b}_{\omega_k} \end{bmatrix}. \end{cases} \tag{2.3}$$

Otherwise if the estimation of bias changes significantly, which far away from the linearization point, we have to do re-propagation under the new bias estimation. So we do not have to propagate IMU measurements repeatedly which save a lot of computational resources.

Chapter 3

System Initialization

As we are using stereo cameras, we don't need to estimate an unknown metric scale factor. The initialization can be simplified into two parts: (1) Vision only SFM (Structure from Motion) (2) Alignment of vision and IMU.

3.1 Vision-only SFM

- Solve the relative pose using essential matrix.
- Triangulate the key points to get the depth and 3D position.
- Solve the pose using PnP method.
- Coordinate transformation.

3.2 Alignment of Vision and IMU

Check the code in "VisualIMUAlignment"

- Gyroscope bias calibration.
- Vlocity, Gravity vector initialization.
- Gravity refinement.
- Completing initialization.

Set the rotation matrix between two camera frames $\boldsymbol{R}_{c_k}^{c_{k+1}}$, the orientation matrix of IMU is $\boldsymbol{R}_{b_k}^{b_{k+1}}$ obtained by IMU pre-integration), and the extrinsic calibration between camera and IMU is $\boldsymbol{R}_b^c$, then for any two adjacent frames, we have:

$$\boldsymbol{R}_{b_k}^{b_{k+1}} \boldsymbol{R}_b^c = \boldsymbol{R}_b^c \boldsymbol{R}_{c_k}^{c_{k+1}} \tag{3.1}$$

Similarly, the orientation matrix of INS can be directly read as $\boldsymbol{R}_w^{b_{k+1}}$, the rotation from $k+1$ frame w.r.t. ENU coordinate, denoted as w. So we can obtain the rotation matrix to the absolute origin.

$$\boldsymbol{R}_w^{c_{k+1}} = \boldsymbol{R}_w^{b_{k+1}} \boldsymbol{R}_b^c \tag{3.2}$$

Chapter 4

Backend Optimization

We utilize the "tightly coupled" optimization framework for the final pose output. The idea is to construct a joint optimization problem by putting the residuals of vision observations and IMU/GPS/INS observations to obtain a global optimal result.

To constraint the number of optimization variables, we use sliding window method with the state variables:

$$\begin{aligned} \mathcal{X} &= [\mathbf{x}_0, \mathbf{x}_0, \cdots, \mathbf{x}_n, \lambda_0, \lambda_1, \cdots, \lambda_m] \\ \mathbf{x}_k &= \left[\mathbf{x}_{b_k}^w, \mathbf{v}_{b_k}^w, \mathbf{q}_{b_k}^w\right], k \in [0, n] \end{aligned} \tag{4.1}$$

where $\mathbf{x}_k$ represents all the camera states inside the sliding window containing $n+1$ keyframes (including positions, rotations, and velocities); λ_k is inverse depth of $m+1$ 3D points **??**.

Additionally for IMU sensor, the states $\mathbf{x}_k$ also contain biases of gyroscope and accelerator $\mathbf{b}_g$ and $\mathbf{b}_a$.

Here we list the parametrization of map points defined in the code:

```
class FeaturePerId {
//Index of this map point.
const int feature_id;
// The frame index of first observation of this map point.
int start_frame;
// Vector of features in each frame(observation).
// Its size is the number of observations.
vector<FeaturePerFrame> feature_per_frame;
// Size of @ref feature_per_frame, observation times.
int used_num;
// Estimated depth in meters.
double estimated_depth;
// Flag of depth estimation successful or not.
// 0 haven't solve yet; 1 solve succ; 2 solve fail (estimated depth <0);
int solve_flag;
};
```

So the cost function for minimizing the residuals from all sensors can be formulated as a bundle adjustment problem:

$$\min_{\mathcal{X}} \left\{ \underbrace{\left\|\mathbf{r}_p - \mathbf{H}_p\mathcal{X}\right\|^2}_{\text{Marginalization Priors}} + \underbrace{\sum_{k\in\mathcal{B}} \left\|\mathbf{r}_{\mathcal{B}}\left(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X}\right)\right\|_{\mathbf{P}_{b_k+1}^2}^2}_{\text{IMU/INS Factors}} + \underbrace{\sum_{(l,j)\in\mathcal{C}} \rho\left(\left\|\mathbf{r}_{\mathcal{C}}\left(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X}\right)\right\|_{\mathbf{P}_l^{c_j}}^2\right)}_{\text{Vision Factors}} \right\} \tag{4.2}$$

Note that:

- All the residuals represent Mahalanobis distance by dividing the scalar measurements with standard deviation (this process is called *whitening* to eliminate the metrics [1]).
- We use Huber loss function as the robust kernel function $\rho(\cdot)$ to tackle visual outliers.

4.1 Vision Factors

4.1.1 Residuals

Vision factors are **reprojection error of feature points**. The visual residuals together with their Jacobian matrix calculation can be referred to in `ProjectionFactor::Evaluate` function.

For the key point in the ith frame, we can get the projection location under the camera coordinate of the jth frame.

$$\begin{bmatrix} x_{c_j} \\ y_{c_j} \\ z_{c_j} \\ 1 \end{bmatrix} = \mathbf{T}_{bc}^{-1}\mathbf{T}_{wb_j}^{-1}\mathbf{T}_{wb_i}\mathbf{T}_{bc} \begin{bmatrix} \frac{1}{\lambda}u_{c_i} \\ \frac{1}{\lambda}v_{c_i} \\ \frac{1}{\lambda} \\ 1 \end{bmatrix} \tag{4.3}$$

where u_{c_i} and v_{c_i} are the image pixel location of ith frame with inverse depth λ; $\mathbf{T}_{bc}$ is the transformation matrix (4×4) from camera to IMU, $\mathbf{T}_{wb_i}$ is the transformation matrix from from ith IMU frame to ENU coordinate.

So its 3D position can be represented as:

$$\begin{aligned} \mathbf{P}_{c_j} = \begin{bmatrix} x_{c_j} \\ y_{c_j} \\ z_{c_j} \end{bmatrix} &= \mathbf{R}_{bc}^{\top}\left(\mathbf{R}_{wb_j}^{\top}\left(\mathbf{R}_{wb_i}\left(\mathbf{R}_{bc}\mathbf{P}_{c_i} + \mathbf{p}_{bc}\right) + \mathbf{p}_{wb_i} - \mathbf{p}_{wb_j}\right) - \mathbf{p}_{bc}\right) \\ &= \mathbf{R}_{bc}^{\top}\mathbf{R}_{wb_j}^{\top}\mathbf{R}_{wb_i}\mathbf{R}_{bc}\mathbf{P}_{c_i} + \mathbf{R}_{bc}^{\top}\left(\mathbf{R}_{wb_j}^{\top}\left(\left(\mathbf{R}_{wb_i}\mathbf{p}_{bc} + \mathbf{p}_{wb_i}\right) - \mathbf{p}_{wb_j}\right) - \mathbf{p}_{bc}\right) \end{aligned} \tag{4.4}$$

where

$$\mathbf{P}_{c_i} = \frac{\bar{\mathbf{P}}_{c_i}}{\lambda} = \frac{1}{\lambda}\begin{bmatrix} u_{c_i} \\ v_{c_i} \\ 1 \end{bmatrix} \tag{4.5}$$

For the convenience of calculating Jacobian, we disassemble equation4.4 and define the following variables:

$$
\begin{aligned}
\mathbf{f}_{b_i} &= \mathbf{R}_{bc}\mathbf{P}_{c_i} + \mathbf{p}_{bc} \\
\mathbf{f}_{w} &= \mathbf{R}_{wb_i}\mathbf{f}_{b_i} + \mathbf{p}_{wb_i} \\
\mathbf{f}_{b_j} &= \mathbf{R}_{wb_j}^{\top}\left(\mathbf{f}_{w} - \mathbf{p}_{wb_j}\right) \\
\mathbf{P}_{c_j} &= \mathbf{R}_{bc}^{\top}\left(\mathbf{f}_{b_j} - \mathbf{p}_{bc}\right)
\end{aligned}
\tag{4.6}
$$

```
// Convert the 3D point under frame i to the frame j coordinate.
Eigen::Vector3d pts_camera_i = pts_i / inv_dep_i;
Eigen::Vector3d pts_imu_i    = qic * pts_camera_i + tic;
Eigen::Vector3d pts_w        = Qi * pts_imu_i + Pi;
Eigen::Vector3d pts_imu_j    = Qj.inverse() * (pts_w - Pj);
Eigen::Vector3d pts_camera_j = qic.inverse() * (pts_imu_j - tic);
```

So the vision residuals are:

$$
\mathbf{r}_c = \Sigma \cdot \begin{bmatrix} \frac{x_{c_j}}{z_{c_j}} - u_{c_j} \\ \frac{y_{c_j}}{z_{c_j}} - v_{c_j} \end{bmatrix} = \Sigma \cdot \left(\frac{P_{c_j}}{Z_j} - \hat{\bar{P}}_{c_j} \right)_2 \in \mathbb{R}^{2\times 1}
\tag{4.7}
$$

```
Eigen::Map<Eigen::Vector2d> residual(residuals);
residual =  tangent_base * (pts_camera_j.normalized() - pts_j.normalized());
// Get the error on the normalized plane.
double dep_j = pts_camera_j.z();
residual = (pts_camera_j / dep_j).head<2>() - pts_j.head<2>();
// transform Euclidean distance to Mahalanobis distance
residual = sqrt_info * residual;
```

4.1.2 Covariance Matrix

We set a fixed covariance matrix by setting the image standard deviation as 1.5 pixels.

$$
\Sigma = \begin{bmatrix} \frac{f}{1.5} & 0 \\ 0 & \frac{f}{1.5} \end{bmatrix}
\tag{4.8}
$$

```
ProjectionFactor::sqrt_info = FOCAL_LENGTH / 1.5 * Matrix2d::Identity();
```

4.1.3 Jacobian Matrix

Jacobian matrix is defined as the derivatives of visual residuals to the states and reverse depth values of two time stamps.

$$
\mathbf{J} = \frac{\partial \mathbf{r}_c}{\partial \mathbf{X}_c} = \begin{bmatrix} \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{wb_i} \\ \delta \boldsymbol{\theta}_{wb_i} \end{bmatrix}} & \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{wb_j} \\ \delta \boldsymbol{\theta}_{wb_j} \end{bmatrix}} & \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{bc} \\ \delta \boldsymbol{\theta}_{bc} \end{bmatrix}} & \frac{\partial \mathbf{P}_{c_j}}{\partial \delta \lambda} \end{bmatrix}
\tag{4.9}
$$

According to the chain rule, we have

$$\begin{aligned}\mathbf{J} &= \frac{\partial \mathbf{r}_c}{\partial \mathbf{P}_{c_j}} \frac{\partial \mathbf{P}_{c_j}}{\partial \mathbf{X}_c} = \mathbf{J}_1 \mathbf{J}_2 \\ &= \frac{\partial \mathbf{r}_c}{\partial \mathbf{P}_{c_j}} \begin{bmatrix} \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{wb_i} \\ \delta \boldsymbol{\theta}_{wb_i} \end{bmatrix}} & \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{wb_j} \\ \delta \boldsymbol{\theta}_{wb_j} \end{bmatrix}} & \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{bc} \\ \delta \boldsymbol{\theta}_{bc} \end{bmatrix}} & \frac{\partial \mathbf{P}_{c_j}}{\partial \delta \lambda} \end{bmatrix}\end{aligned} \tag{4.10}$$

For the first part $\mathbf{J}_1$, according to equation(4.7):

$$\mathbf{J}_1 = \frac{\partial \mathbf{r}_c}{\partial \mathbf{P}_{c_j}} = \Sigma \cdot \begin{bmatrix} \frac{1}{z_{c_j}} & 0 & -\frac{x_{c_j}}{z_{c_j}^2} \\ 0 & \frac{1}{z_{c_j}} & -\frac{y_{c_j}^2}{z_{c_j}^2} \end{bmatrix} \in \mathbb{R}^{2\times 3} \tag{4.11}$$

```
Eigen::Matrix<double, 2, 3> reduce(2, 3);
reduce << 1. / dep_j, 0, -pts_camera_j(0) / (dep_j * dep_j),
0, 1. / dep_j, -pts_camera_j(1) / (dep_j * dep_j);
reduce = sqrt_info * reduce;
```

Then for the second part $\mathbf{J}_2$, according to equation(4.4)(4.5)(4.6), we get the partial derivatives of different state variables:

(a) Partial derivatives of state variables $[\delta p_{b_i}^w, \delta \theta_{b_i}^w]$ at i th timestamp

$$\begin{aligned}\frac{\partial \mathbf{P}_{c_j}}{\partial \delta \mathbf{p}_{wb_i}} &= \mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \\ \frac{\partial \mathbf{P}_{c_j}}{\partial \delta \boldsymbol{\theta}_{wb_i}} &= \frac{\partial \mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} \left(\mathbf{I} + [\delta \boldsymbol{\theta}_{wb_i}]_\times\right) \mathbf{f}_{b_i}}{\partial \delta \boldsymbol{\theta}_{wb_i}} \quad \text{(Abandon irrelavent parts)} \\ &= -\mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} [\mathbf{f}_{b_i}]_\times\end{aligned} \tag{4.12}$$

$$\mathbf{J}_2[0] = \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{wb_i} \\ \delta \boldsymbol{\theta}_{wb_i} \end{bmatrix}} = \begin{bmatrix} \mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top & -\mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} [\mathbf{f}_{b_i}]_\times \end{bmatrix} \in \mathbb{R}^{3\times 6} \tag{4.13}$$

(b) Partial derivatives of state variables $[\delta p_{b_j}^w, \delta \theta_{b_j}^w]$ at j timestamp

$$\begin{aligned}\frac{\partial \mathbf{P}_{c_j}}{\partial \delta \mathbf{p}_{wb_j}} &= -\mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \\ \frac{\partial \mathbf{P}_{c_j}}{\partial \delta \boldsymbol{\theta}_{wb_j}} &= \frac{\partial \mathbf{R}_{bc}^\top \left(\mathbf{I} - \left[\delta \boldsymbol{\theta}_{wb_j}\right]_\times\right) \mathbf{R}_{wb_j}^\top \left(\mathbf{f}_w - \mathbf{p}_{wb_j}\right)}{\partial \delta \boldsymbol{\theta}_{wb_j}} \\ &= \frac{\partial \mathbf{R}_{bc}^\top \left(\mathbf{I} - \left[\delta \boldsymbol{\theta}_{wb_j}\right]_\times\right) \mathbf{f}_{b_j}}{\partial \delta \boldsymbol{\theta}_{wb_j}} \\ &= \mathbf{R}_{bc}^\top \left[\mathbf{f}_{b_j}\right]_\times\end{aligned} \tag{4.14}$$

$$\mathbf{J}_2[1] = \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{wb_j} \\ \delta \boldsymbol{\theta}_{wb_j} \end{bmatrix}} = \begin{bmatrix} -\mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top & \mathbf{R}_{bc}^\top \left[\mathbf{f}_{b_j}\right]_\times \end{bmatrix} \in \mathbb{R}^{3\times 6} \tag{4.15}$$

(c) Partial derivatives of extrinsic parameters between IMU and cameras $[\delta p_c^b, \delta\theta_c^b]$ (Optional)

$$\begin{aligned}
\frac{\partial \mathbf{P}_{c_j}}{\partial \delta \mathbf{p}_{bc}} &= \mathbf{R}_{bc}^\top \left(\mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} - \mathbf{I}_{3\times 3} \right) \\
\frac{\partial \mathbf{P}_{c_j}}{\partial \delta \boldsymbol{\theta}_{bc}} &= \frac{\partial \mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} \mathbf{R}_{bc} \mathbf{P}_{c_i}}{\partial \delta \boldsymbol{\theta}_{bc}} + \frac{\partial \mathbf{R}_{bc}^\top \left(\mathbf{R}_{wb_j}^\top \left((\mathbf{R}_{wb_i} \mathbf{p}_{bc} + \mathbf{p}_{wb_i}) - \mathbf{p}_{wb_j} \right) - \mathbf{p}_{bc} \right)}{\partial \delta \boldsymbol{\theta}_{bc}} \\
&= \frac{\partial \left(\mathbf{I} - [\delta \boldsymbol{\theta}_{bc}]_\times \right) \mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} \mathbf{R}_{bc} \left(\mathbf{I} + [\delta \boldsymbol{\theta}_{bc}]_\times \right) \mathbf{P}_{c_i}}{\partial \delta \boldsymbol{\theta}_{bc}} \\
&\quad + \frac{\partial \left(\mathbf{I} - [\delta \boldsymbol{\theta}_{bc}]_\times \right) \mathbf{R}_{bc}^\top \left(\mathbf{R}_{wb_j}^\top \left((\mathbf{R}_{wb_i} \mathbf{p}_{bc} + \mathbf{p}_{wb_i}) - \mathbf{p}_{wb_j} \right) - \mathbf{p}_{bc} \right)}{\partial \delta \boldsymbol{\theta}_{bc}} \\
&= \left(-\mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} \mathbf{R}_{bc} \left[\mathbf{P}_{c_i} \right]_\times + \left[\mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} \mathbf{R}_{bc} \mathbf{P}_{c_i} \right]_\times \right) \\
&\quad + \left[\mathbf{R}_{bc}^\top \left(\mathbf{R}_{wb_j}^\top \left((\mathbf{R}_{wb_i} \mathbf{p}_{bc} + \mathbf{p}_{wb_i}) - \mathbf{p}_{wb_j} \right) - \mathbf{p}_{bc} \right) \right]_\times
\end{aligned} \tag{4.16}$$

$$\mathbf{J}_2[2] = \frac{\partial \mathbf{P}_{c_j}}{\partial \begin{bmatrix} \delta \mathbf{p}_{bc} \\ \delta \boldsymbol{\theta}_{bc} \end{bmatrix}} = \begin{bmatrix} \frac{\partial \mathbf{P}_{c_j}}{\partial \delta \mathbf{p}_{bc}} & \frac{\partial \mathbf{P}_{c_j}}{\partial \delta \boldsymbol{\theta}_{bc}} \end{bmatrix} \in \mathbb{R}^{3\times 6} \tag{4.17}$$

(d) Partial derivatives of feature point inverse depth $\delta\lambda$

$$\begin{aligned}
\frac{\partial \mathbf{P}_{c_j}}{\partial \delta \lambda} &= \frac{\partial \mathbf{P}_{c_j}}{\partial \mathbf{P}_{c_i}} \frac{\partial \mathbf{P}_{c_i}}{\partial \delta \lambda} \\
&= \mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} \mathbf{R}_{bc} \left(-\frac{1}{\lambda^2} \begin{bmatrix} u_{c_i} \\ v_{c_i} \\ 1 \end{bmatrix} \right) \\
&= -\frac{1}{\lambda} \mathbf{R}_{bc}^\top \mathbf{R}_{wb_j}^\top \mathbf{R}_{wb_i} \mathbf{R}_{bc} \mathbf{P}_{c_i}
\end{aligned} \tag{4.18}$$

$$\mathbf{J}_2[3] = \frac{\partial \mathbf{P}_{c_j}}{\partial \delta \lambda} \in \mathbb{R}^{3\times 1} \tag{4.19}$$

All the above codes can be seen in `factor/projectionXFrameXCamFactor.cpp`.

4.2 IMU Factor

This section is quite complicated.

4.2.1 Residuals

According to the explanation of IMU pre-integration in section 2.2, we acquire the pre-integration residuals:

$$\mathbf{r}_{\mathcal{B}}\left(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X}\right) = \begin{bmatrix} \delta\boldsymbol{\alpha}_{b_{k+1}}^{b_k} \\ \delta\boldsymbol{\theta}_{b_{k+1}}^{b_k} \\ \delta\boldsymbol{\beta}_{b_{k+1}}^{b_k} \\ \delta\mathbf{b}_{b_k} \\ \delta\mathbf{b}_{g_k} \end{bmatrix}_{15\times1} = \begin{bmatrix} \mathbf{R}_w^{b_k}\left(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w + \frac{1}{2}\mathbf{g}^w\Delta t_k^2 - \mathbf{v}_{b_k}^w\Delta t_k\right) - \boldsymbol{\alpha}_{b_{k+1}}^{b_k} \\ 2\left[\mathbf{q}_{b_k}^{w^{-1}} \otimes \mathbf{q}_{b_{k+1}}^w \otimes \left(\gamma_{b_{k+1}}^{b_k}\right)^{-1}\right]_{xyz} \\ \mathbf{R}_w^{b_k}\left(\mathbf{v}_{b_{k+1}}^w + \mathbf{g}^w\Delta t_k - \mathbf{v}_{b_k}^w\right) - \boldsymbol{\beta}_{b_{k+1}}^{b_k} \\ \mathbf{b}_{ab_{k+1}} - \mathbf{b}_{ab_k} \\ \mathbf{b}_{wb_{k+1}} - \mathbf{b}_{wb_k} \end{bmatrix} \tag{4.20}$$

where $[\alpha_{b_{k+1}}^{b_k}, \gamma_{b_{k+1}}^{b_k}, \beta_{b_{k+1}}^{b_k}]$ is the bias correction value.
Rewrite the above equation (4.20) into quaternion form:

$$\mathbf{r}_{\mathcal{B}} = \begin{bmatrix} \mathbf{r}_p \\ \mathbf{r}_q \\ \mathbf{r}_v \\ \mathbf{r}_{ba} \\ \mathbf{r}_{bg} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_{b_iw}\left(\mathbf{p}_{wb_j} - \mathbf{p}_{wb_i} - \mathbf{v}_i^w\Delta t + \frac{1}{2}\mathbf{g}^w\Delta t^2\right) - \boldsymbol{\alpha}_{b_ib_j} \\ 2\left[\mathbf{q}_{b_jb_i} \otimes \left(\mathbf{q}_{b_iw} \otimes \mathbf{q}_{wb_j}\right)\right]_{xyz} \\ \mathbf{q}_{b_iw}\left(\mathbf{v}_j^w - \mathbf{v}_i^w + \mathbf{g}^w\Delta t\right) - \boldsymbol{\beta}_{b_ib_j} \\ \mathbf{b}_j^a - \mathbf{b}_i^a \\ \mathbf{b}_j^g - \mathbf{b}_i^g \end{bmatrix}_{15\times1} \tag{4.21}$$

```
Eigen::Quaterniond dq_rected = delta_q * Utility::deltaQ(dq_dbg * dbg);
Eigen::Vector3d    dv_rected = delta_v + dv_dba * dba + dv_dbg * dbg;
Eigen::Vector3d    dp_rected = delta_p + dp_dba * dba + dp_dbg * dbg;

residuals.block<3, 1>(O_P, 0)  =
Qi.inverse() * (0.5*G*sum_dt*sum_dt + Pj - Pi - Vi*sum_dt) - dp_rected;
residuals.block<3, 1>(O_R, 0)  =
2 * (dq_rected.inverse() * (Qi.inverse() * Qj)).vec();
residuals.block<3, 1>(O_V, 0)  =
Qi.inverse() * (G * sum_dt + Vj - Vi) - dv_rected;
residuals.block<3, 1>(O_BA, 0) = Baj - Bai;
residuals.block<3, 1>(O_BG, 0) = Bgj - Bgi;
```

4.2.2 Covariance Matrix

The covariance matrix $\boldsymbol{\Sigma}$ is the $\mathbf{P}$ calculated from the previous IMU pre-integration.

```
Eigen::Map<Eigen::Matrix<double, 15, 1>> residual(residuals);
residual=pre_integration->evaluate(Pi, Qi, Vi, Bai, Bgi, Pj, Qj, Vj, Baj, Bgj);

Eigen::Matrix<double, 15, 15> cov_inv = pre_integration->covariance.inverse();
Eigen::Matrix<double, 15, 15> sqrt_info =
Eigen::LLT<Eigen::Matrix<double, 15, 15>>(cov_inv).matrixL().transpose();

residual = sqrt_info * residual;
```

Once we get the covariance (`sqrt_info`), we use that to calculate the Mahalanobis distance by multiplying with `residual`

$$\|\mathbf{r}\|^2_{\mathbf{\Sigma}} = \mathbf{r}^T \mathbf{\Sigma}^{-1} \mathbf{r} \tag{4.22}$$

4.2.3 Jacobian Matrix

We still need to calculate the partial derivatives of error state variables according to equation (4.21).

1. Partial derivatives to $[\delta p^w_{b_i}, \delta\theta^w_{b_i}]$ at ith frame.

$$\mathbf{J}[0] = \begin{bmatrix} -\mathbf{R}_{b_i w} & \left[\mathbf{R}_{b_i w}\left(\mathbf{p}_{wb_j} - \mathbf{p}_{wb_i} - \mathbf{v}^w_i \Delta t + \frac{1}{2}\mathbf{g}^w \Delta t^2\right)\right]_\times \\ \mathbf{0} & -2\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix}\left[\mathbf{q}^*_{wb_j} \otimes \mathbf{q}_{wb_i}\right]_L \left[\mathbf{q}_{b_i b_j}\right]_R \begin{bmatrix}\mathbf{0} \\ \frac{1}{2}\mathbf{I}\end{bmatrix} \\ \mathbf{0} & \left[\mathbf{R}_{b_i w}\left(\mathbf{v}^w_j - \mathbf{v}^w_i + \mathbf{g}^w \Delta t\right)\right]_\times \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{15\times 7} \tag{4.23}$$

2. Partial derivatives to $[\delta v^w_i, \delta b^a_i, \delta b^g_i]$ at ith frame.

$$\mathbf{J}[1] = \begin{bmatrix} -\mathbf{R}_{b_i w}\Delta t & -\mathbf{J}^\alpha_{b^a_i} & -\mathbf{J}^\alpha_{b^g_i} \\ \mathbf{0} & \mathbf{0} & -2\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix}\left[\mathbf{q}^*_{wb_j} \otimes \mathbf{q}_{wb_i} \otimes \mathbf{q}_{b_i b_j}\right]_L \begin{bmatrix}\mathbf{0} \\ \frac{1}{2}\mathbf{J}^q_{b^g_i}\end{bmatrix} \\ -\mathbf{R}_{b_i w} & -\mathbf{J}^\beta_{b^a_i} & -\mathbf{J}^\beta_{b^g_i} \\ \mathbf{0} & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{I} \end{bmatrix} \in \mathbb{R}^{15\times 9} \tag{4.24}$$

3. Partial derivatives to $[\delta p^w_{b_j}, \delta\theta^w_{b_j}]$ at ith frame.

$$\mathbf{J}[2] = \begin{bmatrix} \mathbf{R}_{b_i w} & \mathbf{0} \\ \mathbf{0} & 2\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix}\left[\mathbf{q}^*_{b_i b_j} \otimes \mathbf{q}^*_{wb_i} \otimes \mathbf{q}_{wb_j}\right]_L \begin{bmatrix}\mathbf{0} \\ \frac{1}{2}\mathbf{I}\end{bmatrix} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{15\times 7} \tag{4.25}$$

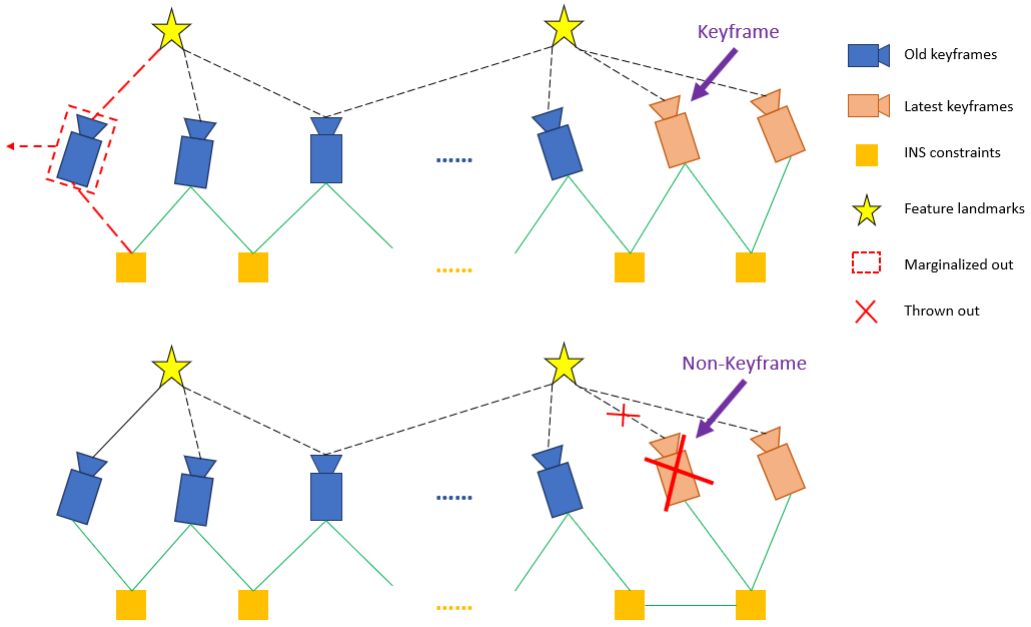
4. Partial derivatives to $[\delta v_j^w, \delta b_j^a, \delta b_j^g]$ at ith frame.

$$\mathbf{J}[3] = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{R}_{b_i w} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \in \mathbb{R}^{15\times 9} \tag{4.26}$$

4.3 INS Factor

4.4 GPS Factor

4.5 Marginalization



Sliding window constraints the number of key-frames to keep a bound of finite complexity by restraining the accumulating of pose and feature numbers with time going on.

4.5.1 Marginalization Concept

When we are using sliding window, we have to abandon old poses, thus culling out some constraints, which is bound to cause the decrease of accuracy. So instead of directly "throwing out" the constraints, it is better converting the related constraints into a prior constraint in the optimization called "marginalization".

The process of marginalization is the process of **decomposing joint probability distribution into marginal probability distribution and conditional probability distribution**,

also can be taken as the reduction of parameters to be optimized using Shur Complement algorithm.

It is known that there exists sparsity in the Hessian matrix in the visual SLAM case according to the optimization theory. For example, a residual only describes the error that at camera pose $\mathbf{x}_i$ we observed landmark $\mathbf{p}_j$, so the Jacobian matrix of this residual contains all 0 values except the $\mathbf{x}_i$ and $\mathbf{p}_j$ entries. Hessian matrix is the quadratic form of Jacobian: $\mathbf{H} = \mathbf{J}^T\mathbf{T}$.

So for the Hessian matrix we can decompose it into four blocks by converting the equation $\mathbf{p}_j$ into:

$$\begin{bmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^{\mathbf{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}. \tag{4.27}$$

where $\boldsymbol{B}$ is a diagonal matrix with dimension of each diagonal block the same as camera parameter and the number of diagonal blocks the same as camera variable numbers. As each landmark we only record the inverse depth value, so the dimension of each block is 1×1.

The use of marginalization is very popular among key-frame based SLAM methods. It is used in *g2o* for calculation speedup by eliminating landmark variables first and then try to solve camera pose first to in turn solve the landmark variables. While in our system, we need to really marginalize out the oldest or last new frames to maintain their relationships with the current sliding window without the need to calculate them.

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{E}\boldsymbol{C}^{-\mathbf{1}} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^{\mathbf{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{E}\boldsymbol{C}^{-\mathbf{1}} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}. \tag{4.28}$$

Then we get:

$$\begin{bmatrix} \boldsymbol{B} - \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}} & \mathbf{0} \\ \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} - \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{w} \\ \boldsymbol{w} \end{bmatrix}. \tag{4.29}$$

where $\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}}$ is called the **Schur factor** of $\boldsymbol{C}$ in $\boldsymbol{E}$. After elimination, the first line becomes irrelevant with $\Delta \boldsymbol{x}_p$. So we extract it and get the incremental equation of the "pose" part.

$$\left[\boldsymbol{B} - \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}}\right] \Delta \boldsymbol{x}_c = \boldsymbol{v} - \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{w}. \tag{4.30}$$

The dimension of this linear equation is the same as $\boldsymbol{B}$ matrix. So we first solve the equation and replace the solved $\Delta \boldsymbol{x}_c$ into the original equation and get $\Delta \boldsymbol{x}_p$. This process is called Schur Elimination.

4.5.2 Toy Example of Marginalization

The process of marginalization and the follow-up phenomenon called "fill-in" (the Hessian matrix becomes dense) can be illustrated in this section.

Suppose we have 4 camera poses $\boldsymbol{x}_{pi}$ and 6 landmarks $\boldsymbol{x}_{mk}$, the links between camera and landmarks represent a visual observation and the links between camera poses are INS/IMU constraints. Then we can get:

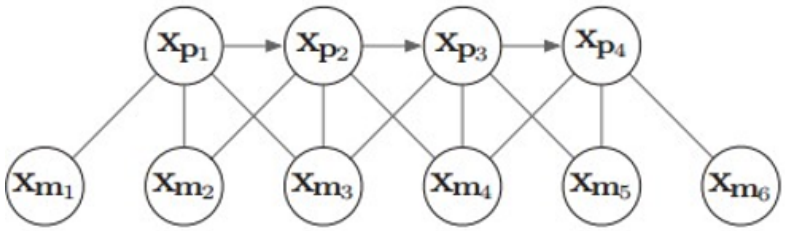


Figure 4.1: Graph relationships of 4 camera poses observing 6 landmarks.

Then if we try to marginalize out $\boldsymbol{x}_{p1}$, and then marginalize out $\boldsymbol{x}_{m1}$, the change of Hessian matrix $\boldsymbol{H}$ is illustrated as:

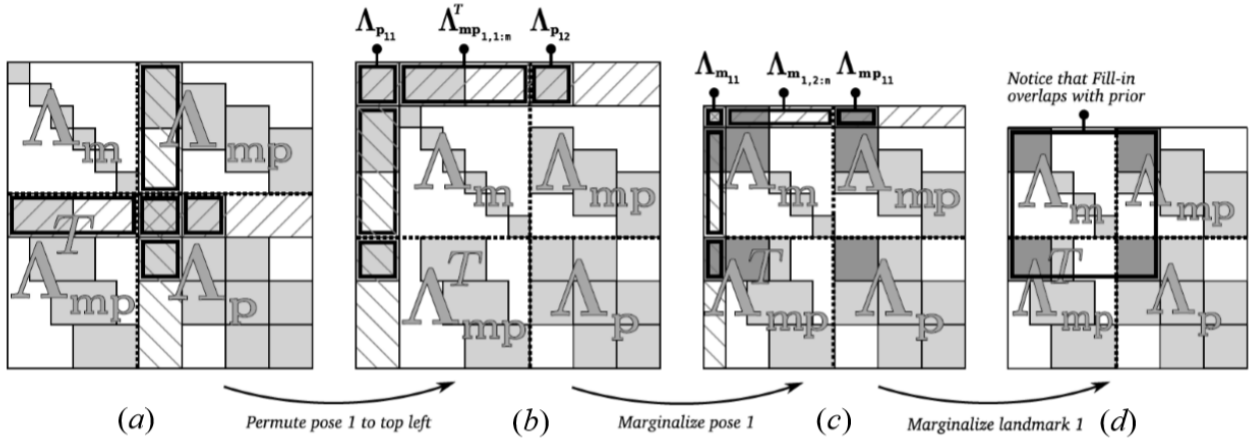
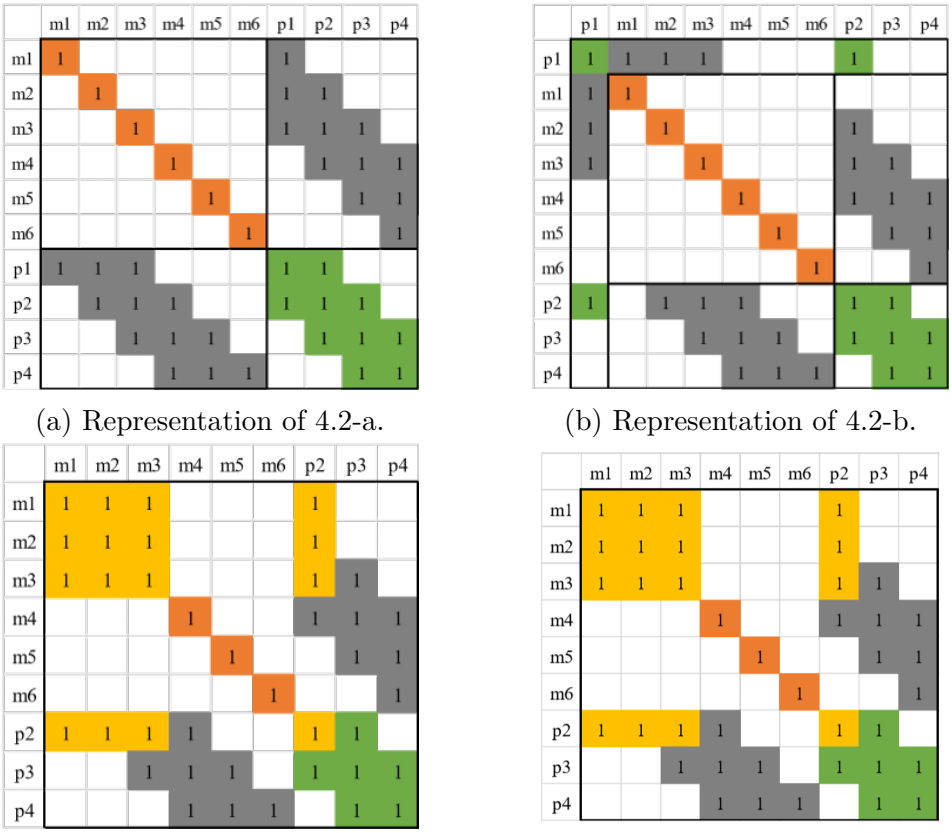


Figure 4.2: The change of $\boldsymbol{H}$ matrix when marginalizing out pose and landmark.

To explain this in details, let 4.2-(a) represent the original $\boldsymbol{H}$ matrix with the top-left part landmark related information and bottom-right part pose information. 4.2-(b) moves the part related to $\boldsymbol{x}_{p1}$ to the top left corner of the $\boldsymbol{H}$ matrix.

4.2-(c) is the matrix after marginalizing out the $\boldsymbol{x}_{p1}$ values, which represents the $\left(\boldsymbol{B}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}}\right)$ in equation (4.30). As you can see, compared to (b), 4.2-(c) becomes denser (the color becomes darker) by filling-in three places.



(a) Representation of 4.2-a.

(b) Representation of 4.2-b.

(c) Representation of 4.2-a.

(d) Representation of 4.2-b.

Figure 4.3: Detailed illustration of Hessian matrix during marginalization.

At this time, the new graph becomes:

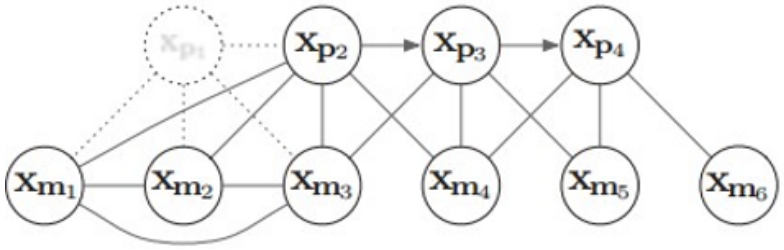


Figure 4.4: Graph relationships after marginalizing out $\boldsymbol{x}_{p1}$.

As you can see there exist new constraints between $\boldsymbol{x}_{m1}$, $\boldsymbol{x}_{m2}$ and $\boldsymbol{x}_{m3}$, and also between $\boldsymbol{x}_{p2}$ and $\boldsymbol{x}_{m1}$.

If you further marginalize out $\boldsymbol{x}_{m1}$, actually $\boldsymbol{H}$ does not become denser as seen in 4.2-(d). That's because $\boldsymbol{x}_{m1}$ does not relate to other poses except $\boldsymbol{x}_{p1}$. So marginalizing out landmarks not observed by other frames will not make the $\boldsymbol{H}$ matrix denser.

4.5.3 Implementation

We first define the marginalization factor in *Ceres* according to the dimension of different variables to be optimized.

```
struct ResidualBlockInfo{
ceres::CostFunction *_cost_function,
ceres::LossFunction *_loss_function,
// Variable values to be optimized.
std::vector<double *> _parameter_blocks,
// Variable values to be marginalized out.
std::vector<int> _drop_set}
```

```
class MarginalizationInfo {
// (1) All the observable factors (in form of residual block information).
std::vector<ResidualBlockInfo *> factors;
// (2) pos: size of all variables (local size).
// (3) n: number of remaining variables to be optimized (local size).
// (4) m: number of variables to be marginalized (local size).
int m, n;

// "parameter_block_sizes":
// parameters in cost_function is an array of pointers to arrays
// csame number of elements as parameter_block_sizes.
// Parameter blocks are in the same order as parameter_block_sizes. i.e.,
//   parameters_[i] = double[parameter_block_sizes_[i]]

// (5) The memory address of variables to be optimized.
// <long, int> where first element is address of variable.
// second element is the number of outputs (residuals)
std::unordered_map<long, int> parameter_block_size;
// (6) Corresponding data (double*) of "parameter_block_size".
std::unordered_map<long, double *> parameter_block_data;
int sum_block_size;
// Ids in "parameter_block_size", the memory address
// of variables to be marginalized.
std::unordered_map<long, int> parameter_block_idx; //local size

std::vector<int> keep_block_size; //global size
std::vector<int> keep_block_idx;  //local size
std::vector<double *> keep_block_data;
Eigen::MatrixXd linearized_jacobians;
Eigen::VectorXd linearized_residuals;
// Add residual block information (all variables to be optimized).
void addResidualBlockInfo(ResidualBlockInfo *residual_block_info);

// Calculate Jacobian of each residual, and update "parameter_block_data"
void preMarginalize();

// pos: dimension of all variables.
void marginalize();
}
```

- `parmaeter_block_data`
  When the oldest keyframe is to be marginalized, the term `parameter_block_size` is the local size of all variables and `parameter_block_data` is the corresponding stored data (double* type). Set the oldest frame (namely $0th$ frame) the camera observed $k = 68$

landmarks, then the size of both `parameter_block_size` and `parameter_block_data` vector is:

$$2(V_0, V_1) + 11(P_{0:10}) + 68(\lambda_{1:k}) = 81$$

Considering the variable dimension, the total local size of all variables is:

$$pos = 2 \times 9 + 11 \times 6 + 68 = 152$$

- `parameter_block_idx`

  `parameter_block_idx` is the variables that need to be marginalized. So the size of the array of `parameter_block_idx` is:

$$1(P_0) + 1(V_0) + 68(\lambda_{1:k}) = 70$$

  If we use m to represent the total local size of the variables to be marginalized, (namely $m = 6 + 9 + 68 = 83$), then we use n to indicate the total local size of the variable to be kept: $n = pos - m = 152 - 83 = 69$.

Inspired by VINS [4], we introduce a two-way marginalization method to handle the static motion case.

1. `MARGIN_OLD`: If the last new frame is a keyframe, we abandon the oldest frame in the sliding window and implement marginalization to tackle the related constraints of this old frame. It is worth noting that, if this keyframe is the first frame to observe some 3D map point, we need to convert the depth of this map point to the next keyframe.

   (a) Transmit the residual item (size is n) of last prior factor to the current prior facotr and remove the states to be abandoned.

   (b) If there is IMU pre-integration items (when there is IMU sensor input), we put the IMU constraint between the $0th$ and $1th$ frame into the `marginalization_info` class.

   (c) For the landmark points that were observed for the first time at the $0th$ frame, we put them into the `marginalization_info` class.

   (d) `marginalization_info->preMarginalize()`: Calculate the `parameter_blocks` and the corresponding jacobians and residuals before formulating the optimization problem.

   (e) `marginalization_info->marginalize()`: Calculate the equation (4.30).

```
// Line 301 in sslam_estimator/src/factor/
// marginalization_factor.cpp.
Eigen::VectorXd bmm = b.segment(0, m);
Eigen::MatrixXd Amr = A.block(0, m, m, n);
Eigen::MatrixXd Arm = A.block(m, 0, n, m);
Eigen::MatrixXd Arr = A.block(m, m, n, n);
Eigen::VectorXd brr = b.segment(m, n);
A = Arr - Arm * Amm_inv * Amr;
b = brr - Arm * Amm_inv * bmm;
```

2. `MARGIN_NEW`: If the last new frame is not a keyframe, then abandon it without the need of marginalization. Since our strategy of judging whether a frame is a keyframe is that it has low parallax with the previous frame, we can directly replace the current frame with the last new frame. The similar constraints will not bring additional effective information so we can directly delete that frame without losing accuracy.

So if the UGV stays in the same location for a long time, the VO system will always implement `MARGIN_NEW` and the sliding window will only keep the poses that are relative "old" but with a big parallax. In this way we can reduce the accumulating drift cost by camera and INS/IMU small errors.

Bibliography

[1] Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.

[2] Christian Forster and et al. IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation. *Robotics: Science and Systems*, 11(6), 2015. doi: 10.15607/rss.2015.xi.006.

[3] Todd Lupton and Salah Sukkarieh. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Transactions on Robotics*, 28(1):61–76, 2012. ISSN 15523098. doi: 10.1109/TRO.2011.2170332.

[4] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.