Notes on
Various Errors and Jacobian Derivations for SLAM

Gyubeom Edward Im*

May 1, 2024

Contents

*blog: alida.tistory.com, email: criterion.im@gmail.com

1 Introduction

In this post, we discuss the definitions of various errors used in SLAM and the Jacobians utilized for their optimization. The errors covered in this post are as follows...

- Reprojection error
$$\mathbf{e} = \mathbf{p} - \hat{\mathbf{p}} \in \mathbb{R}^2 \tag{1}$$
- Photometric error
$$\mathbf{e} = \mathbf{I}_1(\mathbf{p}_1) - \mathbf{I}_2(\mathbf{p}_2) \in \mathbb{R}^1 \tag{2}$$
- Relative pose error (PGO)
$$\mathbf{e}_{ij} = \mathrm{Log}(\mathbf{z}_{ij}^{-1}\hat{\mathbf{z}}_{ij}) \in \mathbb{R}^6 \tag{3}$$
- Line reprojection error
$$\mathbf{e}_l = \begin{bmatrix} \frac{\mathbf{x}_s^\intercal l_c}{\sqrt{l_1^2+l_2^2}}, & \frac{\mathbf{x}_e^\intercal l_c}{\sqrt{l_1^2+l_2^2}} \end{bmatrix} \in \mathbb{R}^2 \tag{4}$$
- IMU measurement error :
$$\mathbf{e}_{\mathcal{B}} = \begin{bmatrix} \delta\alpha_{b_{k+1}}^{b_k} \\ \delta\boldsymbol{\theta}_{b_{k+1}}^{b_k} \\ \delta\beta_{b_{k+1}}^{b_k} \\ \delta\mathbf{b}_a \\ \delta\mathbf{b}_g \end{bmatrix} = \begin{bmatrix} \mathbf{R}_w^{b_k}(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w - \mathbf{v}_{b_k}^w \Delta t_k + \frac{1}{2}\mathbf{g}^w \Delta t_k^2) - \hat{\alpha}_{b_{k+1}}^{b_k} \\ 2\Big[\Big(\hat{\gamma}_{b_{k+1}}^{b_k}\Big)^{-1} \otimes (\mathbf{q}_{b_k}^w)^{-1} \otimes \mathbf{q}_{b_{k+1}}^w\Big]_{xyz} \\ \mathbf{R}_w^{b_k}(\mathbf{v}_{b_{k+1}}^w - \mathbf{v}_{b_k}^w + \mathbf{g}^w \Delta t_k) - \hat{\beta}_{b_{k+1}}^{b_k} \\ \mathbf{b}_{a_{k+1}} - \mathbf{b}_{a_k} \\ \mathbf{b}_{g_{k+1}} - \mathbf{b}_{g_k} \end{bmatrix} \tag{5}$$

Depending on whether the camera pose is expressed as a rotation matrix $\mathbf{R} \in SO(3)$ or a transformation matrix $\mathbf{T} \in SE(3)$, different Jacobians are derived. Jacobians for reprojection errors are derived for SO(3), and Jacobians for photometric errors are derived for SE(3). The representation of a point in 3D space as $\mathbf{X} = [X, Y, Z, W]^\intercal$ or using inverse depth ρ also affects the Jacobian derivation. The derivation processes for both cases are explained.

The Jacobians discussed in this post are as follows.

- Camera pose (SO(3)-based)
$$\frac{\partial \mathbf{e}}{\partial \mathbf{R}} \rightarrow \frac{\partial \mathbf{e}}{\partial \Delta\mathbf{w}} \tag{6}$$
- Camera pose (SE(3)-based)
$$\frac{\partial \mathbf{e}}{\partial \mathbf{T}} \rightarrow \frac{\partial \mathbf{e}}{\partial \Delta\xi} \tag{7}$$
- Map point
$$\frac{\partial \mathbf{e}}{\partial \mathbf{X}} \tag{8}$$
- Relative pose (SE(3)-based)
$$\frac{\partial \mathbf{e}_{ij}}{\partial \Delta\xi_i}, \frac{\partial \mathbf{e}_{ij}}{\partial \Delta\xi_j} \tag{9}$$
- 3D plücker line
$$\frac{\partial \mathbf{e}_l}{\partial l}, \frac{\partial l}{\partial \mathcal{L}_c}, \frac{\partial \mathcal{L}_c}{\partial \mathcal{L}_w}, \frac{\partial \mathcal{L}_w}{\partial \delta_{\boldsymbol{\theta}}} \tag{10}$$
- Quaternion representation
$$\frac{\partial \mathbf{X}'}{\partial \mathbf{q}} \tag{11}$$

- Camera intrinsics

$$\frac{\partial \mathbf{e}}{\partial f_x}, \frac{\partial \mathbf{e}}{\partial f_y}, \frac{\partial \mathbf{e}}{\partial c_x}, \frac{\partial \mathbf{e}}{\partial c_y} \tag{12}$$

- Inverse depth

$$\frac{\partial \mathbf{e}}{\partial \rho} \tag{13}$$

- IMU error-state system kinematics :

$$\mathbf{J}_{b_{k+1}}^{b_k} \tag{14}$$

- IMU measurement :

$$\frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial [\mathbf{p}_{b_k}^w, \mathbf{q}_{b_k}^w]}, \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial [\mathbf{v}_{b_k}^w, \mathbf{b}_{a_k}, \mathbf{b}_{g_k}]}, \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial [\mathbf{p}_{b_{k+1}}^w, \mathbf{q}_{b_{k+1}}^w]}, \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial [\mathbf{v}_{b_{k+1}}^w, \mathbf{b}_{a_{k+1}}, \mathbf{b}_{g_{k+1}}]} \tag{15}$$

2 Optimization formulation

2.1 Error derivation

In SLAM, the error is defined as the difference between the observed value (measurement) $\mathbf{z}$ and the predicted value (estimate) $\hat{\mathbf{z}}$ based on sensor data.

$$\boxed{\mathbf{e}(\mathbf{x}) = \mathbf{z} - \hat{\mathbf{z}}(\mathbf{x})} \tag{16}$$

- $\mathbf{x}$: model state variables

As such, the difference between the observed and predicted values is defined as the error, and the optimal state variables $\mathbf{x}$ that minimize this error become the optimization problem in SLAM. In general, since the state variables in SLAM include non-linear terms related to rotation, the non-linear least squares method is mainly used.

2.2 Error function derivation

Typically, when a large amount of sensor data comes in, dozens to hundreds of errors are calculated in vector form. At this time, it is assumed that the error follows a normal distribution, and the work of converting it into an error function is performed.

$$\mathbf{e}(\mathbf{x}) = \mathbf{z} - \hat{\mathbf{z}} \sim \mathcal{N}(0, \mathbf{\Sigma}) \tag{17}$$

Tip

The multivariate normal distribution of the probability variable $\mathbf{x}$ for modeling the error function is as follows.

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\intercal \mathbf{\Omega}(\mathbf{x}-\mu)\right) \sim \mathcal{N}(\mu, \mathbf{\Sigma}) \tag{18}$$

- $\mathbf{\Omega} = \mathbf{\Sigma}^{-1}$: information matrix (inverse of covariance matrix)

The error can be modeled as a multivariate normal distribution with mean 0 and variance $\mathbf{\Sigma}$. Applying the log-likelihood to this equation, $\ln p(\mathbf{e})$ is as follows.

$$\begin{aligned} \ln p(\mathbf{e}) &\propto -\frac{1}{2}(\mathbf{z}-\hat{\mathbf{z}})^T \mathbf{\Omega}(\mathbf{z}-\hat{\mathbf{z}}) \\ &\propto -\frac{1}{2}\mathbf{e}^\intercal \mathbf{\Omega} \mathbf{e} \end{aligned} \tag{19}$$

Finding $\mathbf{x}^*$ where log-likelihood $\ln p(\mathbf{e})$ is maximized results in the highest probability of the multivariate normal distribution. This is called Maximum Likelihood Estimation (MLE). Since $\ln p(\mathbf{e})$ has a negative (-) sign in front, finding the minimum of the negative log-likelihood $\ln p(\mathbf{e})$ is as follows.

$$\mathbf{x}^* = \arg\max p(\mathbf{e}) = \arg\min \mathbf{e}^T \mathbf{\Omega} \mathbf{e} \tag{20}$$

If all errors are added instead of a single error, it is expressed as follows, and this is called the error function E. In actual optimization problems, not the single error $\mathbf{e}_i$ but the error function E that minimizes $\mathbf{x}^*$ is found.

$$\boxed{\begin{aligned}\mathbf{E}(\mathbf{x}) &= \sum_i \mathbf{e}_i^T \mathbf{\Omega}_i \mathbf{e}_i \\ \mathbf{x}^* &= \arg\min \mathbf{E}(\mathbf{x})\end{aligned}} \tag{21}$$

2.3 Non-linear least squares

The final optimization equation to be solved is as follows.

$$\mathbf{x}^* = \arg\min \mathbf{E}(\mathbf{x}) = \arg\min \sum_i \mathbf{e}_i^T \mathbf{\Omega}_i \mathbf{e}_i \tag{22}$$

In the above formula, the optimal parameter $\mathbf{x}^*$ that minimizes the error must be found. **However, the above formula typically includes non-linear terms related to rotation in SLAM, so no closed-form solution exists. Therefore, non-linear optimization methods (Gauss-Newton (GN), Levenberg-Marquardt (LM)) must be used to solve the problem.** Among the actual implemented SLAM codes, the information matrix $\mathbf{\Omega}_i$ is often set to $\mathbf{I}_3$ to find the optimal value for $\mathbf{e}_i^\intercal \mathbf{e}_i$.

For example, let's assume that the problem is solved using the GN method. The order of solving the problem is as follows.

- Define the error function
- Approximate linearization using Taylor expansion
- Set the first derivative to zero.
- Calculate the value and substitute it into the error function
- Repeat until convergence.

If the error function $\mathbf{e}$ is detailed, it appears as $\mathbf{e}(\mathbf{x})$, meaning that the value of the error function changes according to the robot's pose vector $\mathbf{x}$. The GN method updates the increment $\Delta\mathbf{x}$ iteratively in a direction that reduces the error for $\mathbf{e}(\mathbf{x})$.

$$\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})^\intercal \Omega \mathbf{e}(\mathbf{x}+\Delta\mathbf{x}) \tag{23}$$

When $\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})$ is used near $\mathbf{x}$ with a first-order Taylor expansion, the above equation is approximated as follows.

$$\begin{aligned}\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})|_{\mathbf{x}} &\approx \mathbf{e}(\mathbf{x}) + \mathbf{J}(\mathbf{x}+\Delta\mathbf{x}-\mathbf{x}) \\ &= \mathbf{e}(\mathbf{x}) + \mathbf{J}\Delta\mathbf{x}\end{aligned} \tag{24}$$

At this time, $\mathbf{J} = \frac{\partial \mathbf{e}(\mathbf{x}+\Delta\mathbf{x})}{\partial \mathbf{x}}$. When this is applied to the entire error function, it is as follows.

$$\mathbf{e}(\mathbf{x}+\Delta\mathbf{x})^\intercal \Omega \mathbf{e}(\mathbf{x}+\Delta\mathbf{x}) \approx (\mathbf{e}+\mathbf{J}\Delta\mathbf{x})^\intercal \Omega (\mathbf{e}+\mathbf{J}\Delta\mathbf{x}) \tag{25}$$

After expanding the above equation and substituting, it is as follows.

$$\begin{aligned}&= \underbrace{\mathbf{e}^\intercal \Omega \mathbf{e}}_{\mathbf{c}} + 2\underbrace{\mathbf{e}^\intercal \Omega \mathbf{J}}_{\mathbf{b}}\Delta\mathbf{x} + \Delta\mathbf{x}^\intercal \underbrace{\mathbf{J}^\intercal \Omega \mathbf{J}}_{\mathbf{H}}\Delta\mathbf{x} \\ &= \mathbf{c} + 2\mathbf{b}\Delta\mathbf{x} + \Delta\mathbf{x}^\intercal \mathbf{H}\Delta\mathbf{x}\end{aligned} \tag{26}$$

The overall error applied is as follows.

$$\mathbf{E}(\mathbf{x}+\Delta\mathbf{x}) = \sum_i \mathbf{e}_i^\intercal \Omega_i \mathbf{e}_i = \mathbf{c} + 2\mathbf{b}\Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{H}\Delta\mathbf{x} \tag{27}$$

$\mathbf{E}(\mathbf{x}+\Delta\mathbf{x})$ is in a quadratic form about $\Delta\mathbf{x}$ and since $\mathbf{H} = \mathbf{J}^\intercal \Omega \mathbf{J}$ is a positive definite matrix, the first derivative of $\mathbf{E}(\mathbf{x}+\Delta\mathbf{x})$ set to zero determines the minimum of $\Delta\mathbf{x}$.

$$\frac{\partial \mathbf{E}(\mathbf{x}+\Delta\mathbf{x})}{\partial \Delta\mathbf{x}} \approx 2\mathbf{b} + 2\mathbf{H}\Delta\mathbf{x} = 0 \tag{28}$$

This leads to the following formula being derived.

$$\mathbf{H}\Delta\mathbf{x} = -\mathbf{b} \tag{29}$$

Thus obtained $\Delta\mathbf{x} = -\mathbf{H}^{-1}\mathbf{b}$ is updated to $\mathbf{x}$.

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x} \tag{30}$$

The algorithm that iteratively performs the process so far is called the Gauss-Newton method. The LM method, compared to the GN method, has the same overall process, however, in the formula for calculating the increment, a damping factor λ term is added.

$$\boxed{\begin{array}{l}\text{(GN) } \mathbf{H}\Delta\mathbf{x} = -\mathbf{b} \\ \text{(LM) } (\mathbf{H} + \lambda\mathbf{I})\Delta\mathbf{x} = -\mathbf{b}\end{array}} \tag{31}$$

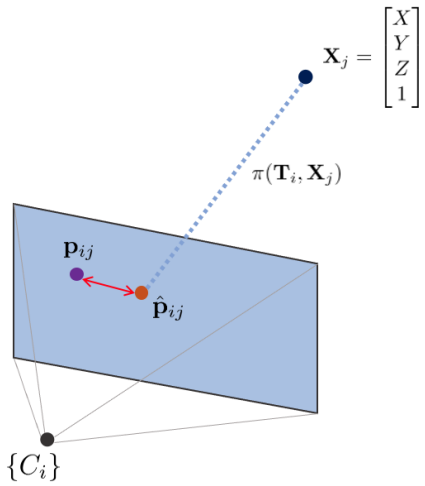
3 Reprojection Error

Reprojection error is primarily used in feature-based Visual SLAM. It is commonly used when performing feature-based method visual odometry (VO) or bundle adjustment (BA). For more details on BA, refer to the post [SLAM] Bundle Adjustment Concept Review.

NOMENCLATURE of reprojection error

- $\tilde{\mathbf{p}} = \pi_h(\cdot) : \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} X'/Z' \\ Y'/Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ 1 \end{bmatrix}$
  - Point $\mathbf{X}'$ in 3D space non-homogeneously transformed to be projected onto the image plane
- $\hat{\mathbf{p}} = \pi_k(\cdot) = \tilde{\mathbf{K}}\tilde{\mathbf{p}} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix} \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ 1 \end{bmatrix} = \begin{bmatrix} f\tilde{u} + c_x \\ f\tilde{v} + c_y \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}$
  - Point projected onto the image plane after lens distortion correction. If distortion correction has already been performed at the input stage, $\pi_k(\cdot) = \tilde{\mathbf{K}}(\cdot)$.
- $\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$: Camera's intrinsic parameters
- $\tilde{\mathbf{K}} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix}$: Omitting the last row of the intrinsic parameters for projection from $\mathbb{P}^2 \rightarrow \mathbb{R}^2$.
- $\mathcal{X} = [\mathcal{T}_1, \cdots, \mathcal{T}_m, \mathbf{X}_1, \cdots, \mathbf{X}_n]^\intercal$: Model's state variables
- m: Number of camera poses
- n: Number of 3D points
- $\mathcal{T}_i = [\mathbf{R}_i, \mathbf{t}_i]$
- $\mathbf{e}_{ij} = \mathbf{e}_{ij}(\mathcal{X})$: Notation simplified by omitting $\mathcal{X}$
- $\mathbf{p}_{ij}$: Observed pixel coordinates of a feature point
- $\hat{\mathbf{p}}_{ij}$: Estimated pixel coordinates of a feature point
- $\mathbf{T}_i\mathbf{X}_j$: Transformation, 3D point $\mathbf{X}_j$ transformed to camera coordinate system $\{i\}$, $\left(\mathbf{T}_i\mathbf{X}_j = \begin{bmatrix} \mathbf{R}_i\mathbf{X}_j + \mathbf{t}_i \\ 1 \end{bmatrix} \in \mathbb{R}^{4\times 1}\right)$
  - $\mathbf{X}' = \mathbf{T}\mathbf{X} = [X', Y', Z', 1]^\intercal = [\tilde{\mathbf{X}}', 1]^\intercal$
- $\oplus$: Operator that updates rotation matrix $\mathbf{R}$ and 3D vector $\mathbf{t}, \mathbf{X}$ simultaneously.

- $\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathcal{X}} = \frac{\partial \mathbf{e}}{\partial [\mathcal{T}, \mathbf{X}]}$
- $\mathbf{w} = \begin{bmatrix} w_x & w_y & w_z \end{bmatrix}^\intercal$: Angular velocity
- $[\mathbf{w}]_\times = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}$: Skew-symmetric matrix of angular velocity $\mathbf{w}$



When there is a pinhole camera pose $\{C_i\}$ and a world point $\mathbf{X}_j$, $\mathbf{X}_j$ is projected onto the image plane through the following transformation.

$$\text{projection model:} \quad \hat{\mathbf{p}}_{ij} = \pi(\mathbf{T}_i, \mathbf{X}_j) \tag{32}$$

3차원 점	$\mathbf{T}_i\mathbf{X}_j$ Transformation	$\tilde{\mathbf{p}} = \pi_h(\cdot)$ Non-homogeneous	$\hat{\mathbf{p}} = \pi_k(\cdot)$ Undistort	Project to Image Plane	
$\mathbf{X}_j$ $[X_j, Y_j, Z_j, 1]^\intercal$	$\mathbf{X}'_j = \begin{bmatrix} \mathbf{R}_i\mathbf{X}_j + \mathbf{t}_i \\ 1 \end{bmatrix} = [X'_j, Y'_j, Z'_j, 1]^\intercal$	$\tilde{\mathbf{p}}_{ij} = [X'_j/Z'_j,\ Y'_j/Z'_j,\ 1]^\intercal = [\tilde{u}_{ij}, \tilde{v}_{ij}, 1]^\intercal$	$\begin{cases} \bar{u}_{ij} = \tilde{u}_{ij}(1 + k_1 r^2 + k_2 r^4) \\ \bar{v}_{ij} = \tilde{v}_{ij}(1 + k_1 r^2 + k_2 r^4) \end{cases}$	$\begin{cases} u_{ij} = f_x \bar{u}_{ij} + c_x \\ v_{ij} = f_y \bar{v}_{ij} + c_y \end{cases}$	$\hat{\mathbf{p}}_{ij}$ $[u_{ij}, v_{ij}]^\intercal$

The model utilizing the camera's intrinsic/extrinsic parameters is called the projection model. Reprojection error is defined as follows:

$$\boxed{\begin{aligned} \mathbf{e}_{ij} &= \mathbf{p}_{ij} - \hat{\mathbf{p}}_{ij} \\ &= \mathbf{p}_{ij} - \pi(\mathbf{T}_i, \mathbf{X}_j) \\ &= \mathbf{p}_{ij} - \pi_k(\pi_h(\mathbf{T}_i\mathbf{X}_j)) \end{aligned}} \tag{33}$$

The error function for all camera poses and 3D points is defined as follows.

$$\mathbf{E}(\mathcal{X}) = \sum_i \sum_j \|\mathbf{e}_{ij}\|^2 \tag{34}$$

$$\begin{aligned} \mathcal{X}^* &= \arg\min_{\mathcal{X}^*} \mathbf{E}(\mathcal{X}) \\ &= \arg\min_{\mathcal{X}^*} \sum_i \sum_j \|\mathbf{e}_{ij}\|^2 \\ &= \arg\min_{\mathcal{X}^*} \sum_i \sum_j \mathbf{e}_{ij}^\intercal \mathbf{e}_{ij} \\ &= \arg\min_{\mathcal{X}^*} \sum_i \sum_j (\mathbf{p}_{ij} - \hat{\mathbf{p}}_{ij})^\intercal (\mathbf{p}_{ij} - \hat{\mathbf{p}}_{ij}) \end{aligned} \tag{35}$$

The error $\|\mathbf{e}(\mathcal{X}^*)\|^2$ satisfying $\mathbf{E}(\mathcal{X}^*)$ can be calculated iteratively through non-linear least squares. By repeatedly updating a small increment $\Delta\mathcal{X}$ to $\mathcal{X}$, the optimal state is found.

$$\arg\min_{\mathcal{X}^*} \mathbf{E}(\mathcal{X} + \Delta\mathcal{X}) = \arg\min_{\mathcal{X}^*} \sum_i \sum_j \|\mathbf{e}(\mathcal{X} + \Delta\mathcal{X})\|^2 \tag{36}$$

Strictly speaking, since the state increment $\Delta\mathcal{X}$ includes an SO(3) rotation matrix, it is correct to add it using the $\oplus$ operator to the existing state $\mathcal{X}$, but the $+$ operator is used for convenience of expression.

$$\mathbf{e}(\mathcal{X} \oplus \Delta\mathcal{X}) \quad \rightarrow \quad \mathbf{e}(\mathcal{X} + \Delta\mathcal{X}) \tag{37}$$

This equation can be expressed through the first-order Taylor approximation as follows.

$$\begin{aligned} \mathbf{e}(\mathcal{X} + \Delta\mathcal{X}) &\approx \mathbf{e}(\mathcal{X}) + \mathbf{J}\Delta\mathcal{X} \\ &= \mathbf{e}(\mathcal{X}) + \mathbf{J}_c\Delta\mathcal{T} + \mathbf{J}_p\Delta\mathbf{X} \\ &= \mathbf{e}(\mathcal{X}) + \frac{\partial \mathbf{e}}{\partial \mathcal{T}}\Delta\mathcal{T} + \frac{\partial \mathbf{e}}{\partial \mathbf{X}}\Delta\mathbf{X} \end{aligned} \tag{38}$$

$$\arg\min_{\mathcal{X}^*} \mathbf{E}(\mathcal{X} + \Delta\mathcal{X}) \approx \arg\min_{\mathcal{X}^*} \sum_i \sum_j \|\mathbf{e}(\mathcal{X}) + \mathbf{J}\Delta\mathcal{X}\|^2 \tag{39}$$

The optimal increment $\Delta\mathcal{X}^*$ is found by differentiating the above expression. The derivation process is omitted in this section. For detailed information on the derivation process, refer to the previous section.

$$\begin{aligned} \mathbf{J}^\intercal\mathbf{J}\Delta\mathcal{X}^* &= -\mathbf{J}^\intercal\mathbf{e} \\ \mathbf{H}\Delta\mathcal{X}^* &= -\mathbf{b} \end{aligned} \tag{40}$$

This equation is in the form of a linear system $\mathbf{Ax} = \mathbf{b}$, thus $\Delta\mathcal{X}^*$ can be found using various linear algebra techniques like schur complement and cholesky decomposition. In this case, $\mathbf{t}, \mathbf{X}$ among the existing states $\mathcal{X}$ exist in a linear vector space, so there is no difference depending on whether they are added from the right or from the left, but **since the rotation matrix R belongs to the non-linear SO(3) group, it depends on whether it is multiplied from the right or from the left whether to update the pose seen in the local coordinate system (right) or the pose seen in the global coordinate system (left). Reprojection error updates the transformation matrix of the global coordinate system, so it generally uses the left multiplication method.**

$$\mathcal{X} \leftarrow \mathcal{X} \oplus \Delta\mathcal{X}^* \tag{41}$$

$\mathcal{X}$ consists of $[\mathcal{T}, \mathbf{X}]$, so it can be described as follows.

$$\begin{aligned} \mathcal{T} &\leftarrow \mathcal{T} \oplus \Delta\mathcal{T}^* \\ \mathbf{X} &\leftarrow \mathbf{X} \oplus \Delta\mathbf{X}^* \end{aligned} \tag{42}$$

The definition of the left multiplication $\oplus$ operation is as follows.

$$\begin{aligned} \mathbf{R} \oplus \Delta\mathbf{R}^* &= \Delta\mathbf{R}^*\mathbf{R} \\ &= \exp([\Delta\mathbf{w}^*]_\times)\mathbf{R} \quad \cdots \text{ globally updated (left mult)} \\ \mathbf{t} \oplus \Delta\mathbf{t}^* &= \mathbf{t} + \Delta\mathbf{t}^* \\ \mathbf{X} \oplus \Delta\mathbf{X}^* &= \mathbf{X} + \Delta\mathbf{X}^* \end{aligned} \tag{43}$$

3.1 Jacobian of the Reprojection Error

3.1.1 Jacobian of Camera Pose

The Jacobian of the pose $\mathbf{J}_c$ can be decomposed as follows.

$$\begin{aligned} \mathbf{J}_c = \frac{\partial \mathbf{e}}{\partial \mathcal{T}} &= \frac{\partial}{\partial \mathcal{T}}(\mathbf{p} - \hat{\mathbf{p}}) \\ &= \frac{\partial}{\partial \mathcal{T}}\Big(\mathbf{p} - \pi_k(\pi_h(\mathbf{T}_i\mathbf{X}_j))\Big) \\ &= \frac{\partial}{\partial \mathcal{T}}\Big(\ - \pi_k(\pi_h(\mathbf{T}_i\mathbf{X}_j))\Big) \end{aligned} \tag{44}$$

Using the chain rule, the above formula is organized as follows. For convenience, $\mathbf{T}_i\mathbf{X}_j \rightarrow \mathbf{X}'$ is denoted.

$$\begin{aligned}\mathbf{J}_c &= \frac{\partial \hat{\mathbf{p}}}{\partial \tilde{\mathbf{p}}} \frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{X}'} \frac{\partial \mathbf{X}'}{\partial [\mathbf{w}, \mathbf{t}]} \\ &= \mathbb{R}^{2\times 3} \cdot \mathbb{R}^{3\times 4} \cdot \mathbb{R}^{4\times 6} = \mathbb{R}^{2\times 6}\end{aligned} \tag{45}$$

The reason for calculating the Jacobian $\frac{\partial \mathbf{X}'}{\partial \mathbf{w}}$ for the angular velocity w instead of the Jacobian $\frac{\partial \mathbf{X}'}{\partial \mathbf{R}}$ for the rotation matrix R is explained in the next section. Also, depending on whether the error is defined as $\mathbf{p} - \hat{\mathbf{p}}$ or $\hat{\mathbf{p}} - \mathbf{p}$, the sign of $\mathbf{J}_c$ also changes, so this should be carefully applied when implementing the actual code. The sign is considered as + and marked in this material.

If it is assumed that undistortion has already been performed during the image input process, $\frac{\partial \hat{\mathbf{p}}}{\partial \tilde{\mathbf{p}}}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \hat{\mathbf{p}}}{\partial \tilde{\mathbf{p}}} &= \frac{\partial}{\partial \tilde{\mathbf{p}}} \tilde{\mathbf{K}} \tilde{\mathbf{p}} \\ &= \tilde{\mathbf{K}} \\ &= \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix} \in \mathbb{R}^{2\times 3}\end{aligned}} \tag{46}$$

Next, $\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{X}'}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{X}'} &= \frac{\partial [\tilde{u}, \tilde{v}, 1]}{\partial [X', Y', Z', 1]} \\ &= \begin{bmatrix} \frac{1}{Z'} & 0 & \frac{-X'}{Z'^2} & 0 \\ 0 & \frac{1}{Z'} & \frac{-Y'}{Z'^2} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 4}\end{aligned}} \tag{47}$$

Next, $\frac{\partial \mathbf{X}'}{\partial \mathbf{t}}$ needs to be calculated. This can be relatively simply obtained as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{X}'}{\partial \mathbf{t}} &= \frac{\partial}{\partial [t_x, t_y, t_z]} \begin{bmatrix} \mathbf{R}\mathbf{X} + \mathbf{t} \\ 1 \end{bmatrix} \\ &= \frac{\partial}{\partial [t_x, t_y, t_z]} \begin{bmatrix} \mathbf{t} \\ 1 \end{bmatrix} \\ &= \frac{\partial}{\partial [t_x, t_y, t_z]} \left(\begin{bmatrix} t_x \\ t_y \\ t_z \\ 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 3}\end{aligned}} \tag{48}$$

3.1.2 Lie Theory-Based SO(3) Optimization

Finally, $\frac{\partial \mathbf{X}'}{\partial \mathbf{w}}$ needs to be calculated. In this case, the rotation-related parameter was represented as angular velocity $\mathbf{w}$ instead of the rotation matrix $\mathbf{R}$. The rotation matrix $\mathbf{R}$ has 9 parameters, whereas the actual rotation is limited to 3 degrees of freedom, therefore it is over-parameterized. The disadvantages of an over-parameterized representation are as follows:

- Due to the calculation of redundant parameters, the amount of computation required for optimization increases.
- Additional degrees of freedom can cause problems with numerical instability.
- Each time parameters are updated, it must be checked whether they always satisfy the constraints.

Lie theory allows optimization to be performed free from constraints. Therefore, instead of the lie group SO(3) $\mathbf{R}$, the lie algebra so(3) $[\mathbf{w}]_\times$ is used to freely update parameters from constraints. Here, $\mathbf{w} \in \mathbb{R}^3$ denotes the angular velocity vector.

$$\mathbf{J}_c = \frac{\partial \mathbf{e}}{\partial [\mathbf{R}, \mathbf{t}]} \rightarrow \frac{\partial \mathbf{e}}{\partial [\mathbf{w}, \mathbf{t}]} \tag{49}$$

However, since $\mathbf{w}$ is not directly visible from $\mathbf{X}'$, $\mathbf{X}'$ must be represented in lie algebra. At this time, since the Jacobian for the $\mathbf{w}$ term related to rotation needs to be calculated, let's assume that the 3D point $\mathbf{X}_t$ is the point $\mathbf{X}$ translated by $\mathbf{t}$ and then $\mathbf{X}'$ is the point $\mathbf{X}_t$ rotated by $\mathbf{R}$.

$$\begin{aligned}\mathbf{X}_t &= \mathbf{X} + \mathbf{t} \\ \mathbf{X}' &= \mathbf{R}\mathbf{X}_t \\ &= \exp([\mathbf{w}]\times)\mathbf{X}_t\end{aligned} \tag{50}$$

Tip

$\exp([\mathbf{w}]_\times) \in SO(3)$ denotes the operation of converting angular velocity $\mathbf{w}$ into a 3D rotation matrix $\mathbf{R}$ using exponential mapping. For detailed information on exponential mapping, see this link.

$$\exp([\mathbf{w}]_\times) = \mathbf{R} \tag{51}$$

In this case, depending on how the small lie algebra increment $\Delta\mathbf{w}$ is updated to the existing $\exp([\mathbf{w}]_\times)$, there are two ways to update. First, there is [1] the basic lie algebra update method. Next, there is [2] the update method using the perturbation model.

$$\begin{aligned}&\exp([\mathbf{w}]_\times) \leftarrow \exp([\mathbf{w} + \Delta\mathbf{w}]_\times) \quad \cdots [1] \\ &\exp([\mathbf{w}]_\times) \leftarrow \exp([\Delta\mathbf{w}]_\times)\exp([\mathbf{w}]_\times) \quad \cdots [2]\end{aligned} \tag{52}$$

Tip

There is the following relationship between the above two methods. For detailed information, see this link chapter 4.3.3.

$$\begin{aligned}\exp([\Delta\mathbf{w}]_\times)\exp([\mathbf{w}]_\times) &= \exp([\mathbf{w} + \mathbf{J}_l^{-1}\Delta\mathbf{w}]_\times) \\ \exp([\mathbf{w} + \Delta\mathbf{w}]_\times) &= \exp([\mathbf{J}_l\Delta\mathbf{w}]_\times)\exp([\mathbf{w}]_\times)\end{aligned} \tag{53}$$

[1] **Lie Algebra-Based Update:** First, using method [1] to directly calculate the Jacobian $\frac{\partial \mathbf{R}\mathbf{X}_t}{\partial \mathbf{w}}$ results in the following complex formula.

$$\begin{aligned}\frac{\partial \mathbf{R}\mathbf{X}_t}{\partial \mathbf{w}} &= \lim_{\Delta\mathbf{w}\to 0} \frac{\exp([\mathbf{w} + \Delta\mathbf{w}]_\times)\mathbf{X}_t - \exp([\mathbf{w}]_\times)\mathbf{X}_t}{\Delta\mathbf{w}} \\ &\approx \lim_{\Delta\mathbf{w}\to 0} \frac{\exp([\mathbf{J}_l\Delta\mathbf{w}]_\times)(\exp([\mathbf{w}]_\times)\mathbf{X}_t - \exp([\mathbf{w}]_\times)\mathbf{X}_t}{\Delta\mathbf{w}} \\ &\approx \lim_{\Delta\mathbf{w}\to 0} \frac{(\mathbf{I} + [\mathbf{J}_l\Delta\mathbf{w}]_\times)(\exp([\mathbf{w}]_\times)\mathbf{X}_t - \exp([\mathbf{w}]_\times)\mathbf{X}_t}{\Delta\mathbf{w}} \\ &= \lim_{\Delta\mathbf{w}\to 0} \frac{[\mathbf{J}_l\Delta\mathbf{w}]_\times\mathbf{R}\mathbf{X}_t}{\Delta\mathbf{w}} \qquad (\because \exp([\mathbf{w}]_\times)\mathbf{X}_t = \mathbf{R}\mathbf{X}_t) \\ &= \lim_{\Delta\mathbf{w}\to 0} \frac{-[\mathbf{R}\mathbf{X}_t]_\times\mathbf{J}_l\Delta\mathbf{w}}{\Delta\mathbf{w}} \\ &= -[\mathbf{R}\mathbf{X}_t]_\times\mathbf{J}_l \\ &= -[\mathbf{X}']_\times\mathbf{J}_l\end{aligned} \tag{54}$$

[2] **Perturbation Model-Based Update:** To calculate a simpler Jacobian without using $\mathbf{J}_l$, the perturbation model of lie algebra so(3) is generally used. Calculating the Jacobian $\frac{\partial \mathbf{R}\mathbf{X}_t}{\partial \Delta\mathbf{w}}$ using the perturbation model

Tip

In the above formula, the second row uses the BCH approximation to derive the left Jacobian (left jacobian) $\mathbf{J}_l$, and the third row applies the first-order Taylor approximation for small rotation $\exp([\mathbf{J}_l\Delta\mathbf{w}]_\times)$. For more information on $\mathbf{J}_l$, see Visual SLAM Introduction Chapter 4.
To understand the third row's approximation, given an arbitrary rotation vector $\mathbf{w} = [w_x, w_y, w_z]^\intercal$, the rotation matrix can be expanded in exponential mapping form as follows.

$$\mathbf{R} = \exp([\mathbf{w}]_\times) = \mathbf{I} + [\mathbf{w}]_\times + \frac{1}{2}[\mathbf{w}]_\times^2 + \frac{1}{3!}[\mathbf{w}]_\times^3 + \frac{1}{4!}[\mathbf{w}]_\times^4 + \cdots \tag{55}$$

For a small rotation matrix $\Delta\mathbf{R}$, higher-order terms beyond the second can be ignored, and it can be approximated as follows.

$$\Delta\mathbf{R} \approx \mathbf{I} + [\Delta\mathbf{w}]_\times \tag{56}$$

results in the following.

$$\begin{aligned}
\frac{\partial \mathbf{R}\mathbf{X}_t}{\partial \Delta\mathbf{w}} &= \lim_{\Delta\mathbf{w}\to 0} \frac{\exp([\Delta\mathbf{w}]_\times)\exp([\mathbf{w}]_\times)\mathbf{X}_t - \exp([\mathbf{w}]_\times)\mathbf{X}_t}{\Delta\mathbf{w}} \\
&\approx \lim_{\Delta\mathbf{w}\to 0} \frac{(\mathbf{I} + [\Delta\mathbf{w}]_\times)\exp([\mathbf{w}]_\times)\mathbf{X}_t - \exp([\mathbf{w}]_\times)\mathbf{X}_t}{\Delta\mathbf{w}} \\
&= \lim_{\Delta\mathbf{w}\to 0} \frac{[\Delta\mathbf{w}]_\times \mathbf{R}\mathbf{X}_t}{\Delta\mathbf{w}} \qquad (\because \exp([\mathbf{w}]_\times)\mathbf{X}_t = \mathbf{R}\mathbf{X}_t) \\
&= \lim_{\Delta\mathbf{w}\to 0} \frac{-[\mathbf{R}\mathbf{X}_t]_\times \Delta\mathbf{w}}{\Delta\mathbf{w}} \\
&= -[\mathbf{R}\mathbf{X}_t]_\times \\
&= -[\mathbf{X}']_\times
\end{aligned} \tag{57}$$

The second row in the above formula uses the approximation $\exp([\Delta\mathbf{w}]_\times) \approx \mathbf{I} + [\Delta\mathbf{w}]_\times$ for a small rotation matrix. **Therefore, using method [2], there is an advantage that the Jacobian can be simply calculated using the skew-symmetric matrix of the 3D point $\mathbf{X}'$. In the case of reprojection error optimization, since the error of feature points in sequentially incoming images is optimized, the camera pose changes are not large, and thus $\Delta\mathbf{w}$ is also not large, so the above Jacobian is commonly used.** Using method [2], the existing rotation matrix $\mathbf{R}$ is updated with a small increment $\Delta\mathbf{w}$ as in (43).

$$\mathbf{R} \leftarrow \Delta\mathbf{R}^*\mathbf{R} \quad \text{where, } \Delta\mathbf{R}^* = \exp([\Delta\mathbf{w}^*]_\times) \tag{58}$$

Therefore, the existing Jacobian changes from $\frac{\partial \mathbf{X}'}{\partial[\mathbf{w},\mathbf{t}]}$ to $\frac{\partial \mathbf{X}'}{\partial[\Delta\mathbf{w},\mathbf{t}]}$ and this is as follows.

$$\boxed{\frac{\partial}{\partial[\Delta\mathbf{w},\mathbf{t}]}\begin{bmatrix}\mathbf{R}\mathbf{X}+\mathbf{t}\\1\end{bmatrix} = \begin{bmatrix}0 & Z' & -Y' & 1 & 0 & 0\\ -Z' & 0 & X' & 0 & 1 & 0\\ Y' & -X' & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \in \mathbb{R}^{4\times 6}} \tag{59}$$

The final Jacobian of the pose $\mathbf{J}_c$ is as follows.

$$\boxed{\begin{aligned}
\mathbf{J}_c &= \frac{\partial \hat{\mathbf{p}}}{\partial \tilde{\mathbf{p}}}\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{X}'}\frac{\partial \mathbf{X}'}{\partial[\Delta\mathbf{w},\mathbf{t}]} \\
&= \begin{bmatrix}f & 0 & c_x\\ 0 & f & c_y\end{bmatrix}\begin{bmatrix}\frac{1}{Z'} & 0 & \frac{-X'}{Z'^2} & 0\\ 0 & \frac{1}{Z'} & \frac{-Y'}{Z'^2} & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}0 & Z' & -Y' & 1 & 0 & 0\\ -Z' & 0 & X' & 0 & 1 & 0\\ Y' & -X' & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \\
&= \begin{bmatrix}-\frac{fX'Y'}{Z'^2} & \frac{f(1+X'^2)}{Z'^2} & -\frac{fY'}{Z'} & \frac{f}{Z'} & 0 & -\frac{fX'}{Z'^2}\\ -\frac{f(1+y^2)}{Z'^2} & \frac{fX'Y'}{Z'^2} & \frac{fX'}{Z'} & 0 & \frac{f}{Z'} & -\frac{fY'}{Z'^2}\end{bmatrix} \in \mathbb{R}^{2\times 6}
\end{aligned}} \tag{60}$$

3.2 Jacobian of Map Point

The Jacobian $\mathbf{J}_p$ of the 3D point $\mathbf{X}$ can be calculated as follows.

$$\begin{aligned}\mathbf{J}_p = \frac{\partial \mathbf{e}}{\partial \mathbf{X}} &= \frac{\partial}{\partial \mathbf{X}}(\mathbf{p} - \hat{\mathbf{p}}) \\ &= \frac{\partial}{\partial \mathbf{X}}\Big(\mathbf{p} - \pi_k(\pi_h(\mathbf{T}_i\mathbf{X}_j))\Big) \\ &= \frac{\partial}{\partial \mathbf{X}}\Big(- \pi_k(\pi_h(\mathbf{T}_i\mathbf{X}_j))\Big)\end{aligned} \tag{61}$$

Using the chain rule, the above formula is organized as follows.

$$\begin{aligned}\mathbf{J}_p &= \frac{\partial \hat{\mathbf{p}}}{\partial \tilde{\mathbf{p}}}\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{X}'}\frac{\partial \mathbf{X}'}{\partial \mathbf{X}} \\ &= \mathbb{R}^{2\times 3}\cdot\mathbb{R}^{3\times 4}\cdot\mathbb{R}^{4\times 4} = \mathbb{R}^{2\times 4}\end{aligned} \tag{62}$$

Among these, $\frac{\partial \hat{\mathbf{p}}}{\partial \tilde{\mathbf{p}}}\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{X}'}$ is the same as the Jacobian calculated earlier. Therefore, only $\frac{\partial \mathbf{X}'}{\partial \mathbf{X}}$ needs to be calculated.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{X}'}{\partial \mathbf{X}} &= \frac{\partial}{\partial \mathbf{X}}\begin{bmatrix}\mathbf{R}\mathbf{X}+\mathbf{t}\\ 1\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{R}\\ 0\end{bmatrix}\end{aligned}} \tag{63}$$

Therefore, $\mathbf{J}_p$ is as follows.

$$\boxed{\begin{aligned}\mathbf{J}_p &= \begin{bmatrix}f & 0 & c_x\\ 0 & f & c_y\end{bmatrix}\begin{bmatrix}\frac{1}{Z'} & 0 & \frac{-X'}{Z'^2} & 0\\ 0 & \frac{1}{Z'} & \frac{-Y'}{Z'^2} & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{R}\\ 0\end{bmatrix} \\ &= \begin{bmatrix}\frac{f}{Z'} & 0 & -\frac{fX'}{Z'^2} & 0\\ 0 & \frac{f}{Z'} & -\frac{fY'}{Z'^2} & 0\end{bmatrix}\begin{bmatrix}\mathbf{R}\\ 0\end{bmatrix} \in \mathbb{R}^{2\times 4}\end{aligned}} \tag{64}$$

Typically, the last column of $\mathbf{J}_p$ is always 0, so it is often omitted and represented in non-homogeneous form.

$$\boxed{\mathbf{J}_p = \begin{bmatrix}\frac{f}{Z'} & 0 & -\frac{fX'}{Z'^2}\\ 0 & \frac{f}{Z'} & -\frac{fY'}{Z'^2}\end{bmatrix}\mathbf{R} \in \mathbb{R}^{2\times 3}} \tag{65}$$

3.3 Code Implementations

- g2o code: edge_project_xyz.cpp#L80
- g2o code: edge_project$_x yz.cpp\#L82$

4 Photometric Error

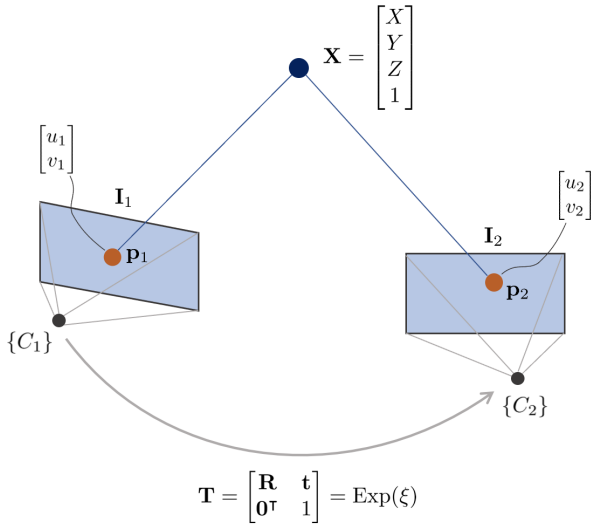
Photometric error is primarily used in direct Visual SLAM. It is commonly utilized in direct method-based visual odometry (VO) or bundle adjustment (BA). For more detailed information on the direct method, refer to the post at [SLAM] Optical Flow and Direct Method Concept and Code Review.

NOMENCLATURE of Photometric Error

- $\tilde{\mathbf{p}}_2 = \pi_h(\cdot) : \begin{bmatrix}X'\\ Y'\\ Z'\\ 1\end{bmatrix} \rightarrow \begin{bmatrix}X'/Z'\\ Y'/Z'\\ 1\end{bmatrix} = \begin{bmatrix}\tilde{u}_2\\ \tilde{v}_2\\ 1\end{bmatrix}$
  - The point $\mathbf{X}'$ in 3D space transformed to a non-homogeneous point on the image plane.
- $\mathbf{p}_2 = \pi_k(\cdot) = \tilde{\mathbf{K}}\tilde{\mathbf{p}}_2 = \begin{bmatrix}f & 0 & c_x\\ 0 & f & c_y\end{bmatrix}\begin{bmatrix}\tilde{u}_2\\ \tilde{v}_2\\ 1\end{bmatrix} = \begin{bmatrix}f\tilde{u}+c_x\\ f\tilde{v}+c_y\end{bmatrix} = \begin{bmatrix}u_2\\ v_2\end{bmatrix}$

– The point projected onto the image plane after correcting for lens distortion. If distortion correction has already been performed at the input stage, $\pi_k(\cdot) = \tilde{\mathbf{K}}(\cdot)$.

- $\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$: Camera's intrinsic parameters.
- $\tilde{\mathbf{K}} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix}$: Omitting the last row of the intrinsic parameters for projection from $\mathbb{P}^2 \to \mathbb{R}^2$.
- $\mathcal{P}$: Set of all feature points in the image.
- $\mathbf{e}(\mathbf{T}) \to \mathbf{e}$: Generally abbreviated for simplicity.
- $\mathbf{p}_1^i, \mathbf{p}_2^i$: Pixel coordinates of the ith feature point in the first and second images.
- $\oplus$: Operator for combining two SE(3) groups (composition).
- $\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{T}} = \frac{\partial \mathbf{e}}{\partial [\mathbf{R}, \mathbf{t}]}$
- $\mathbf{X}' = [X, Y, Z, 1]^\intercal = [\tilde{\mathbf{X}}', 1]^\intercal = \mathbf{T}\mathbf{X}$
- $\mathbf{TX}$: Transformation, transforming the 3D point $\mathbf{X}$ into camera coordinates, $\left(\mathbf{TX} = \begin{bmatrix} \mathbf{R}\mathbf{X} + \mathbf{t} \\ 1 \end{bmatrix} \in \mathbb{R}^{4\times 1} \right)$
- $\mathbf{X}' = [X', Y', Z', 1]^\intercal = [\tilde{\mathbf{X}}', 1]^\intercal$
- $\xi = [\mathbf{w}, \mathbf{v}]^\intercal = [w_x, w_y, w_z, v_x, v_y, v_z]^\intercal$: Vector consisting of 3D angular velocity and velocity, called a twist.
- $[\xi]_\times = \begin{bmatrix} [\mathbf{w}]_\times & \mathbf{v} \\ \mathbf{0}^\intercal & 0 \end{bmatrix} \in \mathrm{se}(3)$: Lie algebra of the twist applied with the hat operator (4x4 matrix)
- $\mathcal{J}_l$: Jacobian for left multiplication. It is not used in actual calculations and hence not detailed here.



In the above figure, the world coordinates of the 3D point $\mathbf{X}$ are $[X, Y, Z, 1]^\intercal \in \mathbb{P}^3$, and the corresponding pixel coordinates on the two camera image planes are $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{P}^2$. Assuming the internal parameters $\mathbf{K}$ of the two cameras $\{C_1\}, \{C_2\}$ are the same. When camera $\{C_1\}$ is considered the origin ($\mathbf{R} = \mathbf{I}, \mathbf{t} = \mathbf{0}$), the pixel coordinates $\mathbf{p}_1, \mathbf{p}_2$ are projected through the 3D point $\mathbf{X}$ as follows:

$$\mathbf{p} = \pi(\mathbf{T}, \mathbf{X}) \tag{66}$$

3차원 점	$\mathbf{TX}$ Transformation	$\tilde{\mathbf{p}} = \pi_h(\cdot)$ Non-homogeneous	$\mathbf{p} = \pi_k(\cdot)$ Undistort	Project to Image Plane	2차원 이미지 픽셀
$\mathbf{X}$ $[X, Y, Z, 1]^\intercal$	$\mathbf{X}' = \begin{bmatrix} \mathbf{R}\mathbf{X} + \mathbf{t} \\ 1 \end{bmatrix} = [X', Y', Z', 1]^\intercal$	$\tilde{\mathbf{p}} = [X'/Z',\ Y'/Z',\ 1]^\intercal = [\tilde{u}, \tilde{v}, 1]^\intercal$	$\begin{cases} \bar{u} = \tilde{u}(1 + k_1 r^2 + k_2 r^4) \\ \bar{v} = \tilde{v}(1 + k_1 r^2 + k_2 r^4) \end{cases}$	$\begin{cases} u = f_x \bar{u} + c_x \\ v = f_y \bar{v} + c_y \end{cases}$	$\hat{\mathbf{p}}$ $[u, v]^\intercal$

$$\begin{aligned}\mathbf{p}_1 &= \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} = \pi(\mathbf{I}, \mathbf{X}) = \pi_k(\pi_h(\mathbf{X})) \\ \mathbf{p}_2 &= \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} = \pi(\mathbf{T}, \mathbf{X}) = \pi_k(\pi_h(\mathbf{T}\mathbf{X}))\end{aligned} \tag{67}$$

One characteristic of the direct method, unlike feature-based methods, is the absence of a way to determine which $\mathbf{p}_2$ matches $\mathbf{p}_1$. Therefore, the position of $\mathbf{p}_2$ is found based on the current pose estimate. Thus, the camera's pose is optimized to make $\mathbf{p}_2$ and $\mathbf{p}_1$ similar, and this problem is solved by minimizing the photometric error. The photometric error is as follows:

$$\boxed{\begin{aligned}\mathbf{e}(\mathbf{T}) &= \mathbf{I}_1(\mathbf{p}_1) - \mathbf{I}_2(\mathbf{p}_2) \\ &= \mathbf{I}_1\Big(\pi_k(\pi_h(\mathbf{X}))\Big) - \mathbf{I}_2\Big(\pi_k(\pi_h(\mathbf{T}\mathbf{X}))\Big)\end{aligned}} \tag{68}$$

Photometric error is based on the assumption of grayscale invariance and holds scalar values. The following error function $\mathbf{E}(\mathbf{T})$ can be defined to solve non-linear least squares:

$$\mathbf{E}(\mathbf{T}) = \sum_{i \in \mathcal{P}} \|\mathbf{e}_i\|^2 \tag{69}$$

$$\begin{aligned}\mathbf{T}^* &= \arg\min_{\mathbf{T}^*} \mathbf{E}(\mathbf{T}) \\ &= \arg\min_{\mathbf{T}^*} \sum_{i \in \mathcal{P}} \|\mathbf{e}_i\|^2 \\ &= \arg\min_{\mathbf{T}^*} \sum_{i \in \mathcal{P}} \mathbf{e}_i^\intercal \mathbf{e}_i \\ &= \arg\min_{\mathbf{T}^*} \sum_{iin\mathcal{P}} \Big(\mathbf{I}_1(\mathbf{p}_1^i) - \mathbf{I}_2(\mathbf{p}_2^i)\Big)^\intercal \Big(\mathbf{I}_1(\mathbf{p}_1^i) - \mathbf{I}_2(\mathbf{p}_2^i)\Big)\end{aligned} \tag{70}$$

$\mathbf{E}(\mathbf{T}^*)$ that satisfies $\|\mathbf{e}(\mathbf{T}^*)\|^2$ can be calculated iteratively through non-linear least squares. Small increments $\Delta\mathbf{T}$ are iteratively updated to $\mathbf{T}$ to find the optimal state.

$$\arg\min_{\mathbf{T}^*} \mathbf{E}(\mathbf{T} + \Delta\mathbf{T}) = \arg\min_{\mathbf{T}^*} \sum_{i \in \mathcal{P}} \|\mathbf{e}_i(\mathbf{T} + \Delta\mathbf{T})\|^2 \tag{71}$$

Technically, since the state increment $\Delta\mathbf{T}$ is a SE(3) transformation matrix, it should be added to the existing state $\mathbf{T}$ using the $\oplus$ operator, but the $+$ operator is used here for convenience of expression.

$$\mathbf{T} \oplus \Delta\mathbf{T} \quad \rightarrow \quad \mathbf{T} + \Delta\mathbf{T} \tag{72}$$

It is expressed through the first-order Taylor approximation as follows.

$$\begin{aligned}\mathbf{e}(\mathbf{T} + \Delta\mathbf{T}) &\approx \mathbf{e}_i(\mathbf{T}) + \mathbf{J}\Delta\mathbf{T} \\ &= \mathbf{e}_i(\mathbf{T}) + \frac{\partial \mathbf{e}}{\partial \mathbf{T}}\Delta\mathbf{T}\end{aligned} \tag{73}$$

$$\arg\min_{\mathbf{T}^*} \mathbf{E}(\mathbf{T} + \Delta\mathbf{T}) = \arg\min_{\mathbf{T}^*} \sum_{i \in \mathcal{P}} \|\mathbf{e}_i(\mathbf{T}) + \mathbf{J}\Delta\mathbf{T}\|^2 \tag{74}$$

When differentiating to find the optimal increment $\Delta\mathbf{T}^*$, the following results. The detailed derivation process is omitted in this section. If you want to know more about the derivation process, refer to the previous section here.

$$\begin{aligned}\mathbf{J}^\intercal\mathbf{J}\Delta\mathbf{T}^* &= -\mathbf{J}^\intercal\mathbf{e} \\ \mathbf{H}\Delta\mathbf{T}^* &= -\mathbf{b}\end{aligned} \tag{75}$$

Since the above formula forms a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, various linear algebra techniques such as schur complement, cholesky decomposition can be used to find $\Delta\mathbf{T}^*$. The optimal increment found in this way is then added to the current state. **Depending on whether it is multiplied to the right or left of the existing state T, it changes whether to update the pose in the local coordinate system (right) or the pose in the global coordinate system (left). Since photometric error updates the transformation matrix of the global coordinate system, the left multiplication method is generally used.**

$$\mathbf{T} \leftarrow \mathbf{T} \oplus \Delta\mathbf{T}^* \tag{76}$$

The definition of the left multiplication $\oplus$ operation is as follows.

$$\begin{aligned}\mathbf{T} \oplus \Delta\mathbf{T}^* &= \Delta\mathbf{T}^*\mathbf{T} \\ &= \exp([\Delta\xi^*]_\times)\mathbf{T} \quad \cdots \text{ globally updated (left mult)}\end{aligned} \tag{77}$$

4.1 Jacobian of the Photometric Error

To perform (75), the Jacobian $\mathbf{J}$ of the photometric error must be determined. It can be represented as follows.

$$\begin{aligned}\mathbf{J} &= \frac{\partial \mathbf{e}}{\partial \mathbf{T}} \\ &= \frac{\partial \mathbf{e}}{\partial [\mathbf{R}, \mathbf{t}]}\end{aligned} \tag{78}$$

Expanding this in detail results in the following.

$$\begin{aligned}\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{T}} &= \frac{\partial}{\partial \mathbf{T}} \Big(\mathbf{I}_1(\mathbf{p}_1) - \mathbf{I}_2(\mathbf{p}_2) \Big) \\ &= \frac{\partial}{\partial \mathbf{T}} \Big(\mathbf{I}_1 \Big(\pi_k(\pi_h(\mathbf{X})) \Big) - \mathbf{I}_2 \Big(\pi_k(\pi_h(\mathbf{TX})) \Big) \Big) \\ &= \frac{\partial}{\partial \mathbf{T}} \Big(- \mathbf{I}_2 \Big(\pi_k(\pi_h(\mathbf{TX})) \Big) \Big) \\ &= \frac{\partial}{\partial \mathbf{T}} \Big(- \mathbf{I}_2 \Big(\pi_k(\pi_h(\mathbf{X}')) \Big) \Big)\end{aligned} \tag{79}$$

Applying the chain rule re-expresses the above equation as follows.

$$\begin{aligned}\frac{\partial \mathbf{e}}{\partial \xi} &= \frac{\partial \mathbf{I}}{\partial \mathbf{p}_2} \frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2} \frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'} \frac{\partial \mathbf{X}'}{\partial \xi} \\ &= \mathbb{R}^{1\times 2} \cdot \mathbb{R}^{2\times 3} \cdot \mathbb{R}^{3\times 4} \cdot \mathbb{R}^{4\times 6} = \mathbb{R}^{1\times 6}\end{aligned} \tag{80}$$

The reason for computing the Jacobian $\frac{\partial \mathbf{X}'}{\partial \xi}$ instead of $\frac{\partial \mathbf{X}'}{\partial \mathbf{T}}$ will be explained in the next section.
First, $\frac{\partial \mathbf{I}}{\partial \mathbf{p}_2}$ refers to the gradient of the image.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{I}}{\partial \mathbf{p}_2} &= \begin{bmatrix} \frac{\partial \mathbf{I}}{\partial u} & \frac{\partial \mathbf{I}}{\partial v} \end{bmatrix} \\ &= \begin{bmatrix} \nabla \mathbf{I}_u & \nabla \mathbf{I}_v \end{bmatrix}\end{aligned}} \tag{81}$$

If it is assumed that undistortion was already performed during image input, $\frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2} &= \frac{\partial}{\partial \tilde{\mathbf{p}}_2} \tilde{\mathbf{K}} \tilde{\mathbf{p}}_2 \\ &= \tilde{\mathbf{K}} \\ &= \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix} \in \mathbb{R}^{2\times 3}\end{aligned}} \tag{82}$$

Next, $\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'}$ is as follows.

$$\boxed{\begin{aligned}\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'} &= \frac{\partial [\tilde{u}_2, \tilde{v}_2, 1]}{\partial [X', Y', Z', 1]} \\ &= \begin{bmatrix} \frac{1}{Z'} & 0 & \frac{-X'}{Z'^2} & 0 \\ 0 & \frac{1}{Z'} & \frac{-Y'}{Z'^2} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 4}\end{aligned}} \tag{83}$$

4.1.1 Lie Theory-based SE(3) Optimization

Finally, $\frac{\partial \mathbf{X}'}{\partial \mathbf{T}} = \frac{\partial \mathbf{X}'}{\partial [\mathbf{R}, \mathbf{t}]}$ must be computed. At this time, the term related to position $\mathbf{t}$ is a 3D vector, and the size of this vector is the minimum degree of freedom, 3 degrees of freedom, for representing 3D position, so there is no separate constraint when performing optimization updates. **On the other hand, the rotation matrix R has 9 parameters, which is more than the minimum degrees of freedom, 3 degrees of freedom, for representing 3D rotation, so various constraints exist. This is called being over-parameterized. The disadvantages of over-parameterized representation are as follows.**

- It is necessary to calculate redundant parameters, which increases the computation during optimization.

- Additional degrees of freedom can cause numerical instability.
- It is necessary to check whether the constraints are satisfied each time the parameters are updated.

Therefore, the optimization method based on lie theory, which is free from constraints, is generally used. **The lie group SE(3) based optimization method refers to the method of updating SE(3) by finding the optimal twist $\Delta\xi^*$ after changing the term related to rotation from $\mathbf{R} \rightarrow \mathbf{w}$ and the term related to position from $\mathbf{t} \rightarrow \mathbf{v}$, and then updating SE(3) through exponential mapping of lie algebra se(3)** $[\Delta\xi]_\times$.

$$\Delta\mathbf{T}^* \rightarrow \Delta\xi^* \tag{84}$$

The Jacobian ξ is as follows.

$$\begin{aligned} \mathbf{J} = \frac{\partial \mathbf{e}}{\partial [\mathbf{R}, \mathbf{t}]} \quad &\rightarrow \frac{\partial \mathbf{e}}{\partial [\mathbf{w}, \mathbf{v}]} \\ &\rightarrow \frac{\partial \mathbf{e}}{\partial \xi} \end{aligned} \tag{85}$$

The existing equation is changed as follows through this.

$$\begin{aligned} \mathbf{e}(\mathbf{T}) &\rightarrow \mathbf{e}(\xi) \\ \mathbf{E}(\mathbf{T}) &\rightarrow \mathbf{E}(\xi) \\ \mathbf{e}(\mathbf{T}) + \mathbf{J}'\Delta\mathbf{T} &\rightarrow \mathbf{e}(\xi) + \mathbf{J}\Delta\xi \\ \mathbf{H}\Delta\mathbf{T}^* = -\mathbf{b} &\rightarrow \mathbf{H}\Delta\xi^* = -\mathbf{b} \\ \mathbf{T} \leftarrow \Delta\mathbf{T}^*\mathbf{T} &\rightarrow \mathbf{T} \leftarrow \exp([\Delta\xi^*]_\times)\mathbf{T} \end{aligned} \tag{86}$$

- $\mathbf{J}' = \frac{\partial \mathbf{e}}{\partial \mathbf{T}}$
- $\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \xi}$

> Tip
>
> $\exp([\xi]_\times) \in \mathrm{SE}(3)$ refers to the operation of transforming the twist ξ through exponential mapping into a 3D pose. For more details on exponential mapping, refer to the related link.
>
> $$\exp([\Delta\xi]_\times) = \Delta\mathbf{T} \tag{87}$$

Until now, the Jacobians were easy to calculate, whereas $\frac{\partial \mathbf{X}'}{\partial \xi}$ requires changing $\mathbf{X}'$ into a term related to lie algebra as it is not immediately apparent from $\mathbf{X}'$ parameters ξ.

$$\mathbf{X}' \rightarrow \mathbf{T}\mathbf{X} \rightarrow \exp([\xi]_\times)\mathbf{X} \tag{88}$$

At this time, depending on the update method of the small lie algebra increment $\Delta\xi$ to the existing $\exp([\xi]_\times)$, it is divided into two methods. First, there is [1] the basic update method using lie algebra. Next, there is [2] the update method using the perturbation model.

$$\begin{aligned} &\exp([\xi]_\times) \leftarrow \exp([\xi + \Delta\xi]_\times) \quad \cdots [1] \\ &\exp([\xi]_\times) \leftarrow \exp([\Delta\xi]_\times)\exp([\xi]_\times) \quad \cdots [2] \end{aligned} \tag{89}$$

Among the two methods, method [1] is a method of adding a fine increment $\Delta\xi$ to the existing ξ and performing exponential mapping to obtain the Jacobian, while method [2] is a method of updating the existing state by multiplying the perturbation model $\exp([\Delta\xi]_\times)$ to the left of the existing ξ.

> Tip
>
> The following transformation exists between the two methods, known as the BCH approximation. For more details, refer to Introduction to Visual SLAM Chapter 4.
>
> $$\begin{aligned} \exp([\Delta\xi]_\times)\exp([\xi]_\times) &= \exp([\xi + \mathcal{J}_l^{-1}\Delta\xi]_\times) \\ \exp([\xi + \Delta\xi]_\times) &= \exp([\mathcal{J}_l\Delta\xi]_\times)\exp([\xi]_\times) \end{aligned} \tag{90}$$

Since a very complex equation is derived when using method [1], this method is not commonly used and method [2] of the perturbation model is mainly used. Therefore, $\frac{\partial \mathbf{X}'}{\partial \xi}$ is transformed as follows.

$$\frac{\partial \mathbf{X}'}{\partial \xi} \rightarrow \frac{\partial \mathbf{X}'}{\partial \Delta\xi} \tag{91}$$

The Jacobian for $\frac{\partial \mathbf{X}'}{\partial \Delta\xi}$ can be calculated as follows.

$$\begin{aligned}
\frac{\partial \mathbf{X}'}{\partial \Delta\xi} &= \lim_{\Delta\xi \to 0} \frac{\exp([\Delta\xi]_\times)\mathbf{X}' - \mathbf{X}'}{\Delta\xi} \\
&\approx \lim_{\Delta\xi \to 0} \frac{(\mathbf{I} + [\Delta\xi]_\times)\mathbf{X}' - \mathbf{X}'}{\Delta\xi} \\
&= \lim_{\Delta\xi \to 0} \frac{[\Delta\xi]_\times \mathbf{X}'}{\Delta\xi} \\
&= \lim_{\Delta\xi \to 0} \frac{\begin{bmatrix} [\Delta\mathbf{w}]_\times & \Delta\mathbf{v} \\ \mathbf{0}^\intercal & 0 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{X}}' \\ 1 \end{bmatrix}}{\Delta\xi} \\
&= \lim_{\Delta\xi \to 0} \frac{\begin{bmatrix} [\Delta\mathbf{w}]_\times \tilde{\mathbf{X}}' + \Delta\mathbf{v} \\ \mathbf{0}^\intercal \end{bmatrix}}{[\Delta\mathbf{w}, \Delta\mathbf{v}]^\intercal} = \begin{bmatrix} -[\tilde{\mathbf{X}}']_\times & \mathbf{I} \\ \mathbf{0}^\intercal & \mathbf{0}^\intercal \end{bmatrix} \in \mathbb{R}^{4\times 6}
\end{aligned} \tag{92}$$

Therefore, using method [2] of the perturbation model has the advantage of simplifying the Jacobian calculation using the skew-symmetric matrix of the 3D point $\mathbf{X}'$. Since photometric error optimization generally involves optimizing the error in brightness changes in sequentially incoming images, the camera pose changes are not large, and thus $\Delta\xi$ is also not large, so the above Jacobian is commonly used. Method [2] of the perturbation model is used, so the small increment $\Delta\xi^*$ is updated as (77).

$$\mathbf{T} \leftarrow \Delta\mathbf{T}^*\mathbf{T} = \exp([\Delta\xi^*]_\times)\mathbf{T} \tag{93}$$

Tip

The second row of the above equation is a form where the first-order Taylor approximation is applied to a small twist increment $\exp([\Delta\xi]_\times)$. To understand the approximation in the second row, when an arbitrary twist $\xi = [\mathbf{w}, \mathbf{v}]^\intercal$ is given, the transformation matrix $\mathbf{T}$ can be expanded into an exponential mapping form as follows.

$$\begin{aligned}
\mathbf{T} = \exp([\xi]_\times) &= \mathbf{I} + \begin{bmatrix} [\mathbf{w}]_\times & \mathbf{v} \\ \mathbf{0}^\intercal & 0 \end{bmatrix} + \frac{1}{2!}\begin{bmatrix} [\mathbf{w}]_\times^2 & [\mathbf{w}]_\times \mathbf{v} \\ \mathbf{0}^\intercal & 0 \end{bmatrix} + \frac{1}{3!}\begin{bmatrix} [\mathbf{w}]_\times^3 & [\mathbf{w}]_\times^2 \mathbf{v} \\ \mathbf{0}^\intercal & 0 \end{bmatrix} + \cdots \\
&= \mathbf{I} + [\xi]_\times + \frac{1}{2!}[\xi]_\times^2 + \frac{1}{3!}[\xi]_\times^3 + \cdots
\end{aligned} \tag{94}$$

For a small magnitude of twist increment $\Delta\xi$, higher-order terms can be ignored to approximately express it as follows.

$$\exp([\Delta\xi]_\times) \approx \mathbf{I} + [\Delta\xi]_\times \tag{95}$$

The final Jacobian $\mathbf{J}$ for the pose is as follows.

$$\begin{aligned}
\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \Delta\xi} &= \frac{\partial \mathbf{I}}{\partial \mathbf{p}_2} \frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2} \frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'} \frac{\partial \mathbf{X}'}{\partial \Delta\xi} \\
&= \begin{bmatrix} \nabla\mathbf{I}_u & \nabla\mathbf{I}_v \end{bmatrix} \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix} \begin{bmatrix} \frac{1}{Z'} & 0 & \frac{-X'}{Z'^2} & 0 \\ 0 & \frac{1}{Z'} & \frac{-Y'}{Z'^2} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -[\tilde{\mathbf{X}}']_\times & \mathbf{I} \\ \mathbf{0}^\intercal & \mathbf{0}^\intercal \end{bmatrix} \\
&= \begin{bmatrix} \nabla\mathbf{I}_u & \nabla\mathbf{I}_v \end{bmatrix} \begin{bmatrix} -\frac{fX'Y'}{Z'^2} & \frac{f(1+X'^2)}{Z'^2} & -\frac{fY'}{Z'} & \frac{f}{Z'} & 0 & -\frac{fX'}{Z'^2} \\ -\frac{f(1+Y'^2)}{Z'^2} & \frac{fX'Y'}{Z'^2} & \frac{fX'}{Z'} & 0 & \frac{f}{Z'} & -\frac{fY'}{Z'^2} \end{bmatrix} \in \mathbb{R}^{1\times 6}
\end{aligned} \tag{96}$$

Since the last row of $\frac{\partial \mathbf{X}'}{\partial \Delta\xi}$ is always 0, it is often omitted and calculated.

4.2 Code Implementations

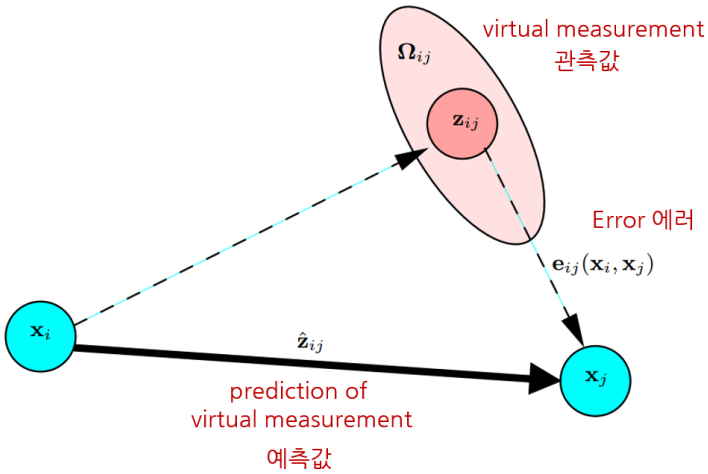
- Introduction to Visual SLAM Chapter 8 code: direct_sparse.cpp#L111
- DSO code: CoarseInitializer.cpp#L430
- DSO code2: CoarseTracker.cpp#L320

5 Relative pose error

Relative pose error is commonly used in pose graph optimization (PGO). For more information about PGO, refer to the post [SLAM] Conceptual explanation and example code analysis of Pose Graph Optimization.

NOMENCLATURE of relative pose error

- (Node) $\mathbf{x}_i = \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \\ \mathbf{0}^\intercal & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}$
- (Edge) $\mathbf{z}_{ij} = \begin{bmatrix} \mathbf{R}_{ij} & \mathbf{t}_{ij} \\ \mathbf{0}^\intercal & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}$
- $\hat{\mathbf{z}}_{ij} = \mathbf{x}_i^{-1}\mathbf{x}_j$: Predicted value
- $\mathbf{z}_{ij}$: Observed value (virtual measurement)
- $\mathbf{x} = [\mathbf{x}_1, \cdots, \mathbf{x}_n]$: All pose nodes in the pose graph
- $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) \leftrightarrow \mathbf{e}_{ij}$: Notation is simplified for convenience.
- $\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{x}}$
- $\oplus$: Operator that combines two SE(3) groups (composition)
- Log$(\cdot)$: Operator that transforms SE(3) into a twist $\xi \in \mathbb{R}^6$. For detailed information about Logarithm mapping, refer to this post.



When two nodes $\mathbf{x}_i, \mathbf{x}_j$ are given on the pose graph, the difference between the newly calculated relative pose (observed value) $\mathbf{z}_{ij}$ and the known relative pose (predicted value) $\hat{\mathbf{z}}_{ij}$ is defined as the relative pose error. (Refer to the figure from Freiburg Univ. Robot Mapping Course).

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij}^{-1}\hat{\mathbf{z}}_{ij} = \mathbf{z}_{ij}^{-1}\mathbf{x}_i^{-1}\mathbf{x}_j \tag{97}$$

The process of optimizing the relative pose error is called pose graph optimization (PGO), and it is also known as the back-end algorithm of graph-based SLAM. The nodes $\mathbf{x}_i, \mathbf{x}_{i+1}, \cdots$, sequentially calculated by the front-end visual odometry (VO) or lidar odometry (LO), do not undergo PGO because the observed and predicted values are the same. However, when loop closing occurs and a non-sequential edge connects two nodes $\mathbf{x}_i, \mathbf{x}_j$, a difference between the observed and predicted values arises, leading to the execution of PGO.

In other words, PGO is typically performed when special situations like loop closing occur. When the robot revisits the same location while moving, a loop detection algorithm operates to determine the loop. At this time, if a loop is detected, the existing node $\mathbf{x}_i$ and the node $\mathbf{x}_j$ created by revisiting are connected by a loop edge, and an observed value is produced by various matching algorithms (GICP, NDT, etc...). **Such observed values, not actually observed but created by matching algorithms, are called virtual measurements.**

The relative pose error for all nodes on the pose graph can be defined as follows.

$$\mathbf{E}(\mathbf{x}) = \sum_i \sum_j \|\mathbf{e}_{ij}\|^2 \tag{98}$$

$$\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}^*} \mathbf{E}(\mathbf{x}) \\ &= \arg\min_{\mathbf{x}^*} \sum_i \sum_j \|\mathbf{e}_{ij}\|^2 \\ &= \arg\min_{\mathbf{x}^*} \sum_i \sum_j \mathbf{e}_{ij}^\intercal \mathbf{e}_{ij} \end{aligned} \tag{99}$$

$\mathbf{E}(\mathbf{x}^*)$ can be calculated iteratively through non-linear least squares by updating a small increment $\Delta\mathbf{x}$ to $\mathbf{x}$ repeatedly to find the optimal state.

$$\arg\min_{\mathbf{x}^*} \mathbf{E}(\mathbf{x} + \Delta\mathbf{x}) = \arg\min_{\mathbf{x}^*} \sum_i \sum_j \|\mathbf{e}_{ij}(\mathbf{x}_i + \Delta\mathbf{x}_i, \mathbf{x}_j + \Delta\mathbf{x}_j)\|^2 \tag{100}$$

Technically speaking, since the state increment $\Delta\mathbf{x}$ is an SE(3) transformation matrix, it should be added to the existing state $\mathbf{x}$ through the $\oplus$ operator, but for convenience of expression, the $+$ operator is used.

$$\mathbf{e}_{ij}(\mathbf{x}_i \oplus \Delta\mathbf{x}_i, \mathbf{x}_j \oplus \Delta\mathbf{x}_j) \quad \rightarrow \quad \mathbf{e}_{ij}(\mathbf{x}_i + \Delta\mathbf{x}_i, \mathbf{x}_j + \Delta\mathbf{x}_j) \tag{101}$$

This equation can be expressed through a first-order Taylor approximation as follows.

$$\begin{aligned} \mathbf{e}_{ij}(\mathbf{x}_i + \Delta\mathbf{x}_i, \mathbf{x}_j + \Delta\mathbf{x}_j) &\approx \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) + \mathbf{J}_{ij} \begin{bmatrix} \Delta\mathbf{x}_i \\ \Delta\mathbf{x}_j \end{bmatrix} \\ &= \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) + \mathbf{J}_i\Delta\mathbf{x}_i + \mathbf{J}_j\Delta\mathbf{x}_j \\ &= \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) + \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{x}_i}\Delta\mathbf{x}_i + + \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{x}_j}\Delta\mathbf{x}_j \end{aligned} \tag{102}$$

$$\arg\min_{\mathbf{x}^*} \mathbf{E}(\mathbf{x} + \Delta\mathbf{x}) \approx \arg\min_{\mathbf{x}^*} \sum_i \sum_j \left\| \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) + \mathbf{J}_{ij} \begin{bmatrix} \Delta\mathbf{x}_i \\ \Delta\mathbf{x}_j \end{bmatrix} \right\|^2 \tag{103}$$

Differentiating this to find the optimal increment $\Delta\mathbf{x}^*$ for all nodes results in the following. The derivation process is omitted in this section. If you want to know the detailed derivation process, refer to the previous section.

$$\begin{aligned} \mathbf{J}^\intercal\mathbf{J}\Delta\mathbf{x}^* &= -\mathbf{J}^\intercal\mathbf{e} \\ \mathbf{H}\Delta\mathbf{x}^* &= -\mathbf{b} \end{aligned} \tag{104}$$

This equation forms a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, and the optimal increment $\Delta\mathbf{x}^*$ can be found using various linear algebra techniques such as the schur complement and Cholesky decomposition. The obtained optimal increment is then added to the current state. Depending on whether it is multiplied on the right or the left of the existing state $\mathbf{x}$, it updates the pose viewed from the local coordinate system (right) or the global coordinate system (left). Since the relative pose error is related to the relative pose of the two nodes, right multiplication, which updates in the local coordinate system, is applied.

$$\mathbf{x} \leftarrow \mathbf{x} \oplus \Delta\mathbf{x}^* \tag{105}$$

The definition of the right multiplication $\oplus$ operation is as follows.

$$\begin{aligned} \mathbf{x} \oplus \Delta\mathbf{x}^* &= \mathbf{x}\Delta\mathbf{x}^* \\ &= \mathbf{x}\exp([\Delta\xi^*]_\times) \quad \cdots \text{ locally updated (right mult)} \end{aligned} \tag{106}$$

5.1 Jacobian of relative pose error

To perform (104), it is necessary to compute the Jacobian $\mathbf{J}$ of the relative pose error. For the given non-sequential nodes $\mathbf{x}_i, \mathbf{x}_j$, their Jacobian $\mathbf{J}_{ij}$ can be expressed as follows.

$$\begin{aligned}\mathbf{J}_{ij} &= \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{x}_{ij}} \\ &= \frac{\partial \mathbf{e}_{ij}}{\partial [\mathbf{x}_i, \mathbf{x}_j]} \\ &= [\mathbf{J}_i, \mathbf{J}_j]\end{aligned} \tag{107}$$

If we elaborate on this, it looks like the following.

$$\begin{aligned}\mathbf{J}_{ij} = \frac{\partial \mathbf{e}_{ij}}{\partial [\mathbf{x}_i, \mathbf{x}_j]} &= \frac{\partial}{\partial [\mathbf{x}_i, \mathbf{x}_j]} \left(\mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij} \right) \\ &= \frac{\partial}{\partial [\mathbf{R}_i, \mathbf{t}_i, \mathbf{R}_j, \mathbf{t}_j]} \left(\mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij} \right)\end{aligned} \tag{108}$$

5.1.1 Lie theory-based SE(3) optimization

When calculating the above Jacobian, since the term $\mathbf{t}$ related to the position is a 3-dimensional vector and the size of this vector is the minimum degrees of freedom to represent 3-dimensional position, which is 3 degrees of freedom, there is no separate constraint when performing optimization updates. **However, the rotation matrix R has 9 parameters, which is more than the minimum degrees of freedom to represent 3-dimensional rotation, which is 3 degrees of freedom, thus various constraints exist. This is referred to as being over-parameterized. The disadvantages of an over-parameterized representation are as follows.**

- Because redundant parameters must be calculated, the computational load increases during optimization.
- Additional degrees of freedom can cause numerical instability issues.
- Parameters must always be checked to satisfy constraints whenever they are updated.

Therefore, a minimal parameter representation free from constraints, a Lie theory-based optimization method, is generally used. **The Lie group SE(3) based optimization method refers to the method of updating SE(3) by calculating the optimal twist $\Delta\xi^*$ by changing the term related to rotation from $\mathbf{R} \rightarrow \mathbf{w}$ and the term related to position from $\mathbf{t} \rightarrow \mathbf{v}$, and then using exponential mapping of the lie algebra se(3) $[\Delta\xi]_\times$.**

$$\left[\Delta \mathbf{x}_i^*, \Delta \mathbf{x}_j^*\right] \rightarrow \left[\Delta \xi_i^*, \Delta \xi_j^*\right] \tag{109}$$

The Jacobian for ξ is as follows.

$$\mathbf{J}_{ij} = \frac{\partial \mathbf{e}_{ij}}{\partial [\mathbf{x}_i, \mathbf{x}_j]} \quad \rightarrow \frac{\partial \mathbf{e}_{ij}}{\partial [\xi_i, \xi_j]} \tag{110}$$

This changes the existing formula as follows.

$$\begin{aligned}
\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) &\rightarrow \mathbf{e}_{ij}(\xi_i, \xi_j) \\
\mathbf{E}(\mathbf{x}) &\rightarrow \mathbf{E}(\xi) \\
\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) + \mathbf{J}_i' \Delta \mathbf{x}_i + \mathbf{J}_j' \Delta \mathbf{x}_j &\rightarrow \mathbf{e}_{ij}(\xi_i, \xi_j) + \mathbf{J}_i \Delta \xi_i + \mathbf{J}_j \Delta \xi_j \\
\mathbf{H} \Delta \mathbf{x}^* = -\mathbf{b} &\rightarrow \mathbf{H} \Delta \xi^* = -\mathbf{b} \\
\mathbf{x} \leftarrow \Delta \mathbf{x}^* \mathbf{x} &\rightarrow \mathbf{x} \leftarrow \exp([\Delta \xi^*]_\times) \mathbf{x}
\end{aligned} \tag{111}$$

- $\mathbf{J}_{ij}' = \frac{\partial \mathbf{e}}{\partial [\mathbf{x}_i, \mathbf{x}_j]}$
- $\mathbf{J}_{ij} = \frac{\partial \mathbf{e}}{\partial [\xi_i, \xi_j]}$

$\frac{\partial}{\partial \xi}(\mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij})$ does not directly appear in the parameters ξ from $\mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij}$, so it needs to be changed into a term related to lie algebra.

$$\mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij} \rightarrow \mathrm{Log}(\mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij}) \tag{113}$$

At this time, $\mathrm{Log}(\cdot)$ means logarithm mapping that changes SE(3) into twist $\xi \in \mathbb{R}^6$. For detailed information about Logarithm mapping, refer to this post. Therefore, the SE(3) version of the relative pose error $\mathbf{e}_{ij}$ is changed as follows.

$$\boxed{\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij} \quad \rightarrow \quad \mathbf{e}_{ij}(\xi_i, \xi_j) = \mathrm{Log}(\mathbf{z}_{ij}^{-1} \hat{\mathbf{z}}_{ij})} \tag{114}$$

Tip

$\exp([\xi]_\times) \in \mathrm{SE}(3)$ refers to the operation of transforming the twist ξ through exponential mapping into a 3-dimensional pose. For detailed information about exponential mapping, refer to this link.

$$\exp([\Delta\xi]_\times) = \Delta\mathbf{x} \tag{112}$$

This is elaborated as follows.

$$\begin{aligned}\mathbf{e}_{ij}(\xi_i, \xi_j) &= \mathrm{Log}(\mathbf{z}_{ij}^{-1}\hat{\mathbf{z}}_{ij}) \\ &= \mathrm{Log}(\mathbf{z}_{ij}^{-1}\mathbf{x}_i^{-1}\mathbf{x}_j) \\ &= \mathrm{Log}(\exp([-\xi_{ij}]_\times)\exp([-\xi_i]_\times)\exp([\xi_j]_\times))\end{aligned} \tag{115}$$

From this equation, we can see that the parameters ξ_i, ξ_j in $\mathbf{z}_{ij}$ are connected through exponential mapping. If we apply the left perturbation model to the second line of the formula and express the increment, it looks like this.

$$\mathbf{e}_{ij}(\xi_i + \Delta\xi_i, \xi_j + \Delta\xi_j) = \mathrm{Log}(\hat{\mathbf{z}}_{ij}^{-1}\mathbf{x}_i^{-1}\exp(-[\Delta\xi_i]_\times)\exp([\Delta\xi_j]_\times)\mathbf{x}_j) \tag{116}$$

Tip

To arrange the term in the form $\mathbf{e} + \mathbf{J}\Delta\xi$ by moving the incremental term to the left or right, the following property of the adjoint matrix of SE(3) must be used. For more information about the adjoint matrix, refer to this post.

$$\exp([\mathrm{Ad}_\mathbf{T} \cdot \xi]_\times) = \mathbf{T} \cdot \exp([\xi]_\times) \cdot \mathbf{T}^{-1} \tag{117}$$

Transforming the above formula for $\mathbf{T} \to \mathbf{T}^{-1}$, we get the following.

$$\exp([\mathrm{Ad}_{\mathbf{T}^{-1}} \cdot \xi]_\times) = \mathbf{T}^{-1} \cdot \exp([\xi]_\times) \cdot \mathbf{T} \tag{118}$$

And simplifying gives the following formula.

$$\exp([\xi]_\times) \cdot \mathbf{T} = \mathbf{T}\exp([\mathrm{Ad}_{\mathbf{T}^{-1}} \cdot \xi]_\times) \tag{119}$$

Using (119), it is possible to move the $\exp(\cdot)\exp(\cdot)$ term in the middle of (116) to the right or left. This post describes the process of moving it to the right. This is expanded for each $\Delta\xi_i, \Delta\xi_j$ as follows.

$$\begin{aligned}\mathbf{e}_{ij}(\xi_i + \Delta\xi_i, \xi_j) &= \mathrm{Log}(\hat{\mathbf{z}}_{ij}^{-1}\mathbf{x}_i^{-1}\exp(-[\Delta\xi_i]_\times)\mathbf{x}_j) \\ &= \mathrm{Log}(\mathbf{z}_{ij}^{-1}\mathbf{x}_i^{-1}\mathbf{x}_j\exp([-\mathrm{Ad}_{\mathbf{x}_j^{-1}}\Delta\xi_i]_\times)) \quad \cdots [1] \\ \mathbf{e}_{ij}(\xi_i, \xi_j + \Delta\xi_j) &= \mathrm{Log}(\hat{\mathbf{z}}_{ij}^{-1}\mathbf{x}_i^{-1}\exp([\Delta\xi_j]_\times)\mathbf{x}_j) \\ &= \mathrm{Log}(\mathbf{z}_{ij}^{-1}\mathbf{x}_i^{-1}\mathbf{x}_j\exp([\mathrm{Ad}_{\mathbf{x}_j^{-1}}\Delta\xi_j]_\times)) \quad \cdots [2]\end{aligned} \tag{120}$$

To express this simply using substitution, [1], [2] are as follows.

$$\begin{aligned}&\mathrm{Log}(\exp([\mathbf{a}]_\times)\exp([\mathbf{b}]_\times)) \quad \cdots [1] \\ &\mathrm{Log}(\exp([\mathbf{a}]_\times)\exp([\mathbf{c}]_\times)) \quad \cdots [2]\end{aligned} \tag{121}$$

- $\exp([\mathbf{a}]_\times) = \mathbf{z}_{ij}^{-1}\mathbf{x}_i^{-1}\mathbf{x}_j$: Transformation matrix expressed as an exponential term. According to (114), $\mathbf{a} = \mathbf{e}_{ij}(\xi_i, \xi_j)$.
- $\mathbf{b} = -\mathrm{Ad}_{\mathbf{x}_j^{-1}}\Delta\xi_i$
- $\mathbf{c} = \mathrm{Ad}_{\mathbf{x}_j^{-1}}\Delta\xi_j$

This formula can be organized using the right BCH approximation.
Using the BCH approximation, (121) is organized as follows.

$$\begin{aligned}\mathrm{Log}(\exp([\mathbf{a}]_\times)\exp([\mathbf{b}]_\times)) &= \mathrm{Log}(\exp([\mathbf{a} + \mathcal{J}_r^{-1}\mathbf{b}]_\times)) \\ &= \mathbf{a} + \mathcal{J}_r^{-1}\mathbf{b} \quad \cdots [1] \\ \mathrm{Log}(\exp([\mathbf{a}]_\times)\exp([\mathbf{c}]_\times)) &= \mathrm{Log}(\exp([\mathbf{a} + \mathcal{J}_r^{-1}\mathbf{c}]_\times)) \\ &= \mathbf{a} + \mathcal{J}_r^{-1}\mathbf{c} \quad \cdots [2]\end{aligned} \tag{123}$$

Tip

The right BCH approximation is as follows.

$$\begin{aligned}\exp([\xi]_\times)\exp([\Delta\xi]_\times) &= \exp([\xi + \mathcal{J}_r^{-1}\Delta\xi]_\times) \\ \exp([\xi + Delta\xi]_\times) &= \exp([\xi]_\times)\exp([\mathcal{J}_r\Delta\xi]_\times)\end{aligned} \tag{122}$$

For detailed information, refer to Introduction to Visual SLAM Chapter 4.

Finally, undoing the substitution and combining the $\Delta\xi_i, \Delta\xi_j$ formulas gives the SE(3) version of the formula in (102).

$$\boxed{\begin{aligned}\mathbf{e}_{ij}(\xi_i + \Delta\xi_i, \xi_j + \Delta\xi_j) &= \mathbf{a} + \mathcal{J}_r^{-1}\mathbf{b} + \mathcal{J}_r^{-1}\mathbf{c} \\ &= \mathbf{e}_{ij}(\xi_i, \xi_j) - \mathcal{J}_r^{-1}\mathrm{Ad}_{\mathbf{x}_j^{-1}}\Delta\xi_i + \mathcal{J}_r^{-1}\mathrm{Ad}_{\mathbf{x}_j^{-1}}\Delta\xi_j \\ &= \mathbf{e}_{ij}(\xi_i, \xi_j) + \frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_i}\Delta\xi_i + \frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_j}\Delta\xi_j\end{aligned}} \tag{124}$$

Therefore, the final relative pose error Jacobian for SE(3) is as follows.

$$\boxed{\begin{aligned}\frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_i} &= -\mathcal{J}_r^{-1}\mathrm{Ad}_{\mathbf{x}_j^{-1}} \in \mathbb{R}^{6\times 6} \\ \frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_j} &= \mathcal{J}_r^{-1}\mathrm{Ad}_{\mathbf{x}_j^{-1}} \in \mathbb{R}^{6\times 6}\end{aligned}} \tag{125}$$

At this time, $\mathcal{J}_r^{-1}$ is generally approximated as follows or used by setting it as $\mathbf{I}_6$.

$$\mathcal{J}_r^{-1} \approx \mathbf{I}_6 + \frac{1}{2}\begin{bmatrix}[\mathbf{w}]_\times & [\mathbf{v}]_\times \\ \mathbf{0} & [\mathbf{w}]_\times\end{bmatrix} \in \mathbb{R}^{6\times 6} \tag{126}$$

If $\mathcal{J}_r^{-1} = \mathbf{I}_6$ is assumed and optimization is performed, there is a reduction in computational load, but the optimization performance is slightly superior when using the approximated Jacobian as above. For detailed information, refer to Introduction to Visual SLAM Chapter 11.

5.2 Code implementations

- g2o code: edge_se3_expmap.cpp#L55
  - In the above g2o code, the error is defined as $\mathbf{e}_{ij} = \mathbf{x}_j^{-1}\mathbf{z}_{ij}\mathbf{x}_i$, so the Jacobian is slightly different from the explanation above.
  - $\frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_i} = \mathcal{J}_l^{-1}\mathrm{Ad}_{\mathbf{x}_j^{-1}\mathbf{z}_{ij}}$
  - $\frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_j} = -\mathcal{J}_r^{-1}\mathrm{Ad}_{\mathbf{x}_i^{-1}\mathbf{z}_{ij}^{-1}}$
  - This follows the same form as combining after arranging the $\Delta\xi_i$ to the left and $\Delta\xi_j$ to the right in (120).
  - It also appears that $\mathcal{J}_l^{-1} \approx \mathbf{I}_6, \mathcal{J}_r^{-1} \approx \mathbf{I}_6$ is approximated. Thus, the actual implemented code is as follows.
    - $\frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_i} \approx \mathrm{Ad}_{\mathbf{x}_j^{-1}\mathbf{z}_{ij}}$
    - $\frac{\partial \mathbf{e}_{ij}}{\partial\Delta\xi_j} \approx -\mathrm{Ad}_{\mathbf{x}_i^{-1}\mathbf{z}_{ij}^{-1}}$

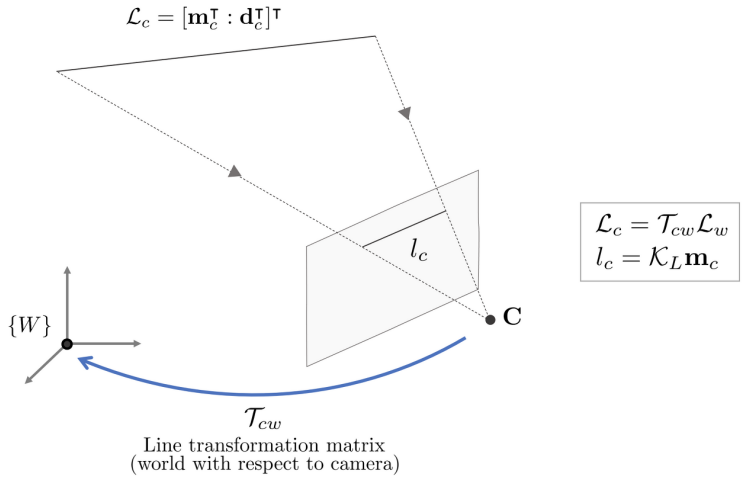
6 Line Reprojection Error

Line reprojection error is used to optimize a 3D line expressed in Plücker coordinates. For more details on Plücker coordinates, refer to the post Plücker Coordinate Concept Summary.

NOMENCLATURE of line reprojection error

- $\mathcal{T}_{cw} \in \mathbb{R}^{6\times 6}$: Transformation matrix for the Plücker line

- $\mathcal{K}_L$: Internal parameter matrix for the line (line intrinsic matrix)
- $\mathbf{U} \in SO(3)$: Rotation matrix for the 3D line
- $\mathbf{W} \in SO(2)$: Matrix containing distance information of the 3D line from the origin
- $\boldsymbol{\theta} \in \mathbb{R}^3$: Parameters corresponding to the SO(3) rotation matrix
- $\theta \in \mathbb{R}$: Parameter corresponding to the SO(2) rotation matrix
- $\mathbf{u}_i$: ith column vector
- $\mathcal{X} = [\delta_{\boldsymbol{\theta}}, \delta_\xi]$: State variable
- $\delta_{\boldsymbol{\theta}} = [\boldsymbol{\theta}^\intercal, \theta] \in \mathbb{R}^4$: State variable in orthonormal representation
- $\delta_\xi = [\delta\xi] \in se(3)$: Update method through Lie theory, refer to this link
- $\oplus$: Operator to update the state variables $\delta_{\boldsymbol{\theta}}, \delta_\xi$ at once.
- $\mathbf{J} = \frac{\partial \mathbf{e}_l}{\partial \mathcal{X}} = \frac{\partial \mathbf{e}_l}{\partial [\delta_{\boldsymbol{\theta}}, \delta_\xi]}$



A line in 3D space can be expressed as a 6-dimensional column vector using Plücker Coordinates.

$$\mathcal{L} = [\mathbf{m}^\intercal : \mathbf{d}^\intercal]^\intercal = [m_x : m_y : m_z : d_x : d_y : d_z]^\intercal \tag{127}$$

The order in papers using Plücker Coordinates is mostly $[\mathbf{m} : \mathbf{d}]$, hence this section uses this order to represent the line. This line representation has scale ambiguity (up to scale), so it has 5 degrees of freedom; $\mathbf{m}, \mathbf{d}$ do not need to be unit vectors, and the line can be uniquely represented by the ratio of the two vector values.

6.1 Line Transformation and Projection

If we refer to a line in the world coordinate system as $\mathcal{L}_w$, then its transformation to the camera coordinate system can be expressed as follows:

$$\mathcal{L}_c = \begin{bmatrix} \mathbf{m}_c \\ \mathbf{d}_c \end{bmatrix} = \mathcal{T}_{cw}\mathcal{L}_w = \begin{bmatrix} \mathbf{R}_{cw} & \mathbf{t}^\wedge\mathbf{R}_{cw} \\ 0 & \mathbf{R}_{cw} \end{bmatrix} \begin{bmatrix} \mathbf{m}_w \\ \mathbf{d}_w \end{bmatrix} \tag{128}$$

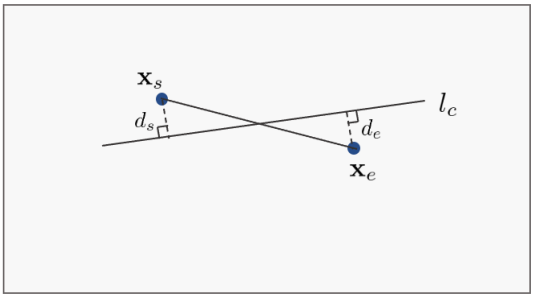
The projection of this line onto the image plane is as follows:

$$l_c = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} = \mathcal{K}_L\mathbf{m}_c = \begin{bmatrix} f_y & & \\ & f_x & \\ -f_y c_x & -f_x c_y & f_x f_y \end{bmatrix} \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} \tag{129}$$

$\mathcal{K}_L$ means $\mathcal{P} = [\det(\mathbf{N})\mathbf{N}^{-\intercal}|\mathbf{n}^\wedge\mathbf{N}]$ where $\mathbf{P} = K[\mathbf{I}|\mathbf{0}]$. Thus, $\mathcal{P} = [\det(\mathbf{K})\mathbf{K}^{-\intercal}|\mathbf{0}]$, so the $\mathbf{d}$ term of $\mathcal{L}$ is eliminated. Therefore, when $\mathbf{K} = \begin{bmatrix} f_x & & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix}$, the following equation is derived:

$$\mathcal{K}_L = \det(\mathbf{K})\mathbf{K}^{-\intercal} = \begin{bmatrix} f_y & & \\ & f_x & \\ -f_y c_x & -f_x c_y & f_x f_y \end{bmatrix} \in \mathbb{R}^{3\times 3} \tag{130}$$

6.2 Line Reprojection Error



$l_c = \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix}^\intercal$

$\mathbf{x}_s, \mathbf{x}_e$ from LSD

$$\mathbf{e}_l = \left[\frac{\mathbf{x}_s^\intercal l_c}{\sqrt{l_1^2+l_2^2}}, \quad \frac{\mathbf{x}_e^\intercal l_c}{\sqrt{l_1^2+l_2^2}} \right]$$

The reprojection error $\mathbf{e}_l$ of the line can be expressed as follows:

$$\mathbf{e}_l = \begin{bmatrix} d_s, & d_e \end{bmatrix} = \left[\frac{\mathbf{x}_s^\intercal l_c}{\sqrt{l_1^2+l_2^2}}, \quad \frac{\mathbf{x}_e^\intercal l_c}{\sqrt{l_1^2+l_2^2}} \right] \in \mathbb{R}^2 \tag{131}$$

This can be expressed using the distance from a point to a line formula. Here, $\{\mathbf{x}_s, \mathbf{x}_e\}$ represent the starting and ending points of the line extracted using a line feature extractor (e.g., LSD). **In other words, l_c is the predicted value obtained through modeling, and the line connecting $\mathbf{x}_s, \mathbf{x}_e$ becomes the observed value measured through sensor data.**

6.3 Orthonormal Representation

Using the previously calculated $\mathbf{e}_l$ for BA optimization poses problems when using the Plücker Coordinate representation directly because Plücker Coordinates must always satisfy the Klein quadric constraint $\mathbf{m}^\intercal \mathbf{d} = 0$, which implies 5 degrees of freedom, making it over-parameterized compared to the minimum 4 parameters needed to represent a line. The disadvantages of an over-parameterized representation are as follows:

- Redundant parameters must be calculated, increasing computational load during optimization.
- Additional degrees of freedom can lead to numerical instability.
- The parameters must always be checked to satisfy the constraint after each update.

Therefore, when optimizing a line, it is common to change to a 4-degree of freedom using the orthonormal representation method. **That is, while lines are represented using Plücker Coordinates, optimization is performed using the orthonormal representation, and then the optimized values are converted back to Plücker Coordinates.**

Orthonormal representation is as follows. A line in 3D space can always be represented as follows:

$$(\mathbf{U}, \mathbf{W}) \in SO(3) \times SO(2) \tag{132}$$

Any given Plücker line $\mathcal{L} = [\mathbf{m}^\intercal : \mathbf{d}^\intercal]^\intercal$ always has a corresponding $(\mathbf{U}, \mathbf{W})$, and this representation method is called the orthonormal representation. When a line $\mathcal{L}_w = [\mathbf{m}_w^\intercal : \mathbf{d}_w^\intercal]^\intercal$ in the world is given, $\mathcal{L}_w$ can be obtained through QR decomposition as follows:

$$\begin{bmatrix} \mathbf{m}_w \mid \mathbf{d}_w \end{bmatrix} = \mathbf{U} \begin{bmatrix} w_1 & 0 \\ 0 & w_2 \\ 0 & 0 \end{bmatrix}, \quad \text{with set: } \mathbf{W} = \begin{bmatrix} w_1 & -w_2 \\ w_2 & w_1 \end{bmatrix} \tag{133}$$

In this case, the upper triangle matrix $\mathbf{R}$'s $(1,2)$ element is always 0 due to the Plücker constraint (Klein quadric). $\mathbf{U}, \mathbf{W}$ represent 3D and 2D rotation matrices, respectively, so $\mathbf{U} = \mathbf{R}(\boldsymbol{\theta}), \mathbf{W} = \mathbf{R}(\theta)$ can be represented as follows:

$$\begin{aligned} \mathbf{R}(\boldsymbol{\theta}) = \mathbf{U} &= \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{m}_w}{\|\mathbf{m}_w\|} & \frac{\mathbf{d}_w}{\|\mathbf{d}_w\|} & \frac{\mathbf{m}_w \times \mathbf{d}_w}{\|\mathbf{m}_w \times \mathbf{d}_w\|} \end{bmatrix} \\ \mathbf{R}(\theta) = \mathbf{W} &= \begin{bmatrix} w_1 & -w_2 \\ w_2 & w_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ &= \frac{1}{\sqrt{\|\mathbf{m}_w\|^2 + \|\mathbf{d}_w\|^2}} \begin{bmatrix} \|\mathbf{m}_w\| & \|\mathbf{d}_w\| \\ -\|\mathbf{d}_w\| & \|\mathbf{m}_w\| \end{bmatrix} \end{aligned} \tag{134}$$

When actually performing optimization, $\mathbf{U} \leftarrow \mathbf{U}\mathbf{R}(\boldsymbol{\theta}), \mathbf{W} \leftarrow \mathbf{W}\mathbf{R}(\theta)$ are updated as follows. **Therefore, the orthonormal representation can represent a 3D line through $\delta_{\boldsymbol{\theta}} = [\boldsymbol{\theta}^\intercal, \theta] \in \mathbb{R}^4$.** The updated $[\boldsymbol{\theta}^\intercal, \theta]$ is converted back to $\mathcal{L}_w$ as follows:

$$\mathcal{L}_w = \begin{bmatrix} w_1 \mathbf{u}_1^\intercal & w_2 \mathbf{u}_2^\intercal \end{bmatrix} \tag{135}$$

6.4 Error Function Formulation

To optimize the line reprojection error $\mathbf{e}_l$, nonlinear least squares methods such as Gauss-Newton (GN), Levenberg-Marquardt (LM), etc., are used to iteratively update the optimal variables. The error function using reprojection error is expressed as follows:

$$\mathbf{E}_l(\mathcal{X}) = \sum_i \sum_j \|\mathbf{e}_{l,ij}\|^2 \tag{136}$$

$$\begin{aligned} \mathcal{X}^* &= \arg\min_{\mathcal{X}^*} \mathbf{E}_l(\mathcal{X}) \\ &= \arg\min_{\mathcal{X}^*} \sum_i \sum_j \|\mathbf{e}_{l,ij}\|^2 \\ &= \arg\min_{\mathcal{X}^*} \sum_i \sum_j \mathbf{e}_{l,ij}^\intercal \mathbf{e}_{l,ij} \end{aligned} \tag{137}$$

The $\mathbf{E}_l(\mathcal{X}^*)$ that satisfies $\|\mathbf{e}_l(\mathcal{X}^*)\|^2$ can be computed iteratively through non-linear least squares. A small increment $\Delta\mathcal{X}$ is iteratively updated to $\mathcal{X}$ to find the optimal state.

$$\arg\min_{\mathcal{X}^*} \mathbf{E}_l(\mathcal{X} + \Delta\mathcal{X}) = \arg\min_{\mathcal{X}^*} \sum_i \sum_j \|\mathbf{e}_l(\mathcal{X} + \Delta\mathcal{X})\|^2 \tag{138}$$

Strictly speaking, the state increment $\Delta\mathcal{X}$ includes an SE(3) transformation matrix, so it is correct to add it to the existing state $\mathcal{X}$ through the $\oplus$ operator, but the $+$ operator is used for simplicity of expression.

$$\mathbf{e}_l(\mathcal{X} \oplus \Delta\mathcal{X}) \quad \rightarrow \quad \mathbf{e}_l(\mathcal{X} + \Delta\mathcal{X}) \tag{139}$$

The above equation can be expressed through a Taylor first-order approximation as follows:

$$\begin{aligned} \mathbf{e}_l(\mathcal{X} + \Delta\mathcal{X}) &\approx \mathbf{e}_l(\mathcal{X}) + \mathbf{J}\Delta\mathcal{X} \\ &= \mathbf{e}_l(\mathcal{X}) + \mathbf{J}_{\boldsymbol{\theta}}\Delta\delta_{\boldsymbol{\theta}} + \mathbf{J}_\xi\Delta\delta_\xi \\ &= \mathbf{e}_l(\mathcal{X}) + \frac{\partial \mathbf{e}_l}{\partial \delta_{\boldsymbol{\theta}}}\Delta\delta_{\boldsymbol{\theta}} + \frac{\partial \mathbf{e}_l}{\partial \delta_\xi}\Delta\delta_\xi \end{aligned} \tag{140}$$

$$\arg\min_{\mathcal{X}^*} \mathbf{E}_l(\mathcal{X} + \Delta\mathcal{X}) \approx \arg\min_{\mathcal{X}^*} \sum_i \sum_j \|\mathbf{e}_l(\mathcal{X}) + \mathbf{J}\Delta\mathcal{X}\|^2 \tag{141}$$

The optimal increment $\Delta\mathcal{X}^*$ is obtained by differentiating the above. The detailed derivation process is omitted in this section. For detailed information on the derivation process, refer to the previous section.

$$\begin{aligned} \mathbf{J}^\intercal\mathbf{J}\Delta\mathcal{X}^* &= -\mathbf{J}^\intercal\mathbf{e} \\ \mathbf{H}\Delta\mathcal{X}^* &= -\mathbf{b} \end{aligned} \tag{142}$$

6.4.1 The Analytical Jacobian of 3D Line

As explained in the previous section, to perform nonlinear optimization, $\mathbf{J}$ must be calculated. $\mathbf{J}$ is composed as follows:

$$\mathbf{J} = [\mathbf{J}_{\boldsymbol{\theta}}, \mathbf{J}_\xi] \tag{143}$$

$[\mathbf{J}_{\boldsymbol{\theta}}, \mathbf{J}_\xi]$ can be expanded as follows:

$$\begin{aligned} \mathbf{J}_{\boldsymbol{\theta}} &= \frac{\partial \mathbf{e}_l}{\partial \delta_{\boldsymbol{\theta}}} = \frac{\partial \mathbf{e}_l}{\partial l}\frac{\partial l}{\partial \mathcal{L}_c}\frac{\partial \mathcal{L}_c}{\partial \mathcal{L}_w}\frac{\partial \mathcal{L}_w}{\partial \delta_{\boldsymbol{\theta}}} \\ \mathbf{J}_\xi &= \frac{\partial \mathbf{e}_l}{\partial \delta_\xi} = \frac{\partial \mathbf{e}_l}{\partial l}\frac{\partial l}{\partial \mathcal{L}_c}\frac{\partial \mathcal{L}_c}{\partial \delta_\xi} \end{aligned} \tag{144}$$

$\frac{\partial \mathbf{e}_l}{\partial l}$ can be obtained as follows. Note that l is a vector and l_i is a scalar.

$$\boxed{\frac{\partial \mathbf{e}_l}{\partial l} = \frac{1}{\sqrt{l_1^2 + l_2^2}} \begin{bmatrix} x_s - \frac{l_1 \mathbf{x}_s l}{\sqrt{l_1^2+l_2^2}} & y_s - \frac{l_2 \mathbf{x}_s l}{\sqrt{l_1^2+l_2^2}} & 1 \\ x_e - \frac{l_1 \mathbf{x}_e l}{\sqrt{l_1^2+l_2^2}} & y_e - \frac{l_2 \mathbf{x}_e l}{\sqrt{l_1^2+l_2^2}} & 1 \end{bmatrix} \in \mathbb{R}^{2\times 3}} \tag{145}$$

$\frac{\partial l}{\partial \mathcal{L}_c}$ can be obtained as follows:

$$\boxed{\frac{\partial l}{\partial \mathcal{L}_c} = \frac{\partial \mathcal{K}_L \mathbf{m}_c}{\partial \mathcal{L}_c} = \begin{bmatrix} \mathcal{K}_L & \mathbf{0}_{3\times 3} \end{bmatrix} = \begin{bmatrix} f_y & & & 0 & 0 & 0 \\ & f_x & & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 6}} \tag{146}$$

$\frac{\partial \mathcal{L}_c}{\partial \mathcal{L}_w}$ can be obtained as follows:

$$\boxed{\frac{\partial \mathcal{L}_c}{\partial \mathcal{L}_w} = \mathbf{J}_{SE(3)}(\mathcal{L}_c) = \begin{bmatrix} \mathbf{I}_{3\times 3} & -[\mathbf{t}]_\times \end{bmatrix} = \begin{bmatrix} 1 & 0 & -y_c \\ 0 & 1 & x_c \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}} \tag{147}$$

$\frac{\partial \mathcal{L}_w}{\partial \delta_{\boldsymbol{\theta}}}$ can be obtained as follows:

$$\boxed{\frac{\partial \mathcal{L}_w}{\partial \delta_{\boldsymbol{\theta}}} = \frac{\partial \mathcal{L}_w}{\partial \mathbf{R}} \frac{\partial \mathbf{R}}{\partial \delta_{\boldsymbol{\theta}}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{0} & -\mathbf{W} \\ \mathbf{W} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{2\times 4}} \tag{148}$$

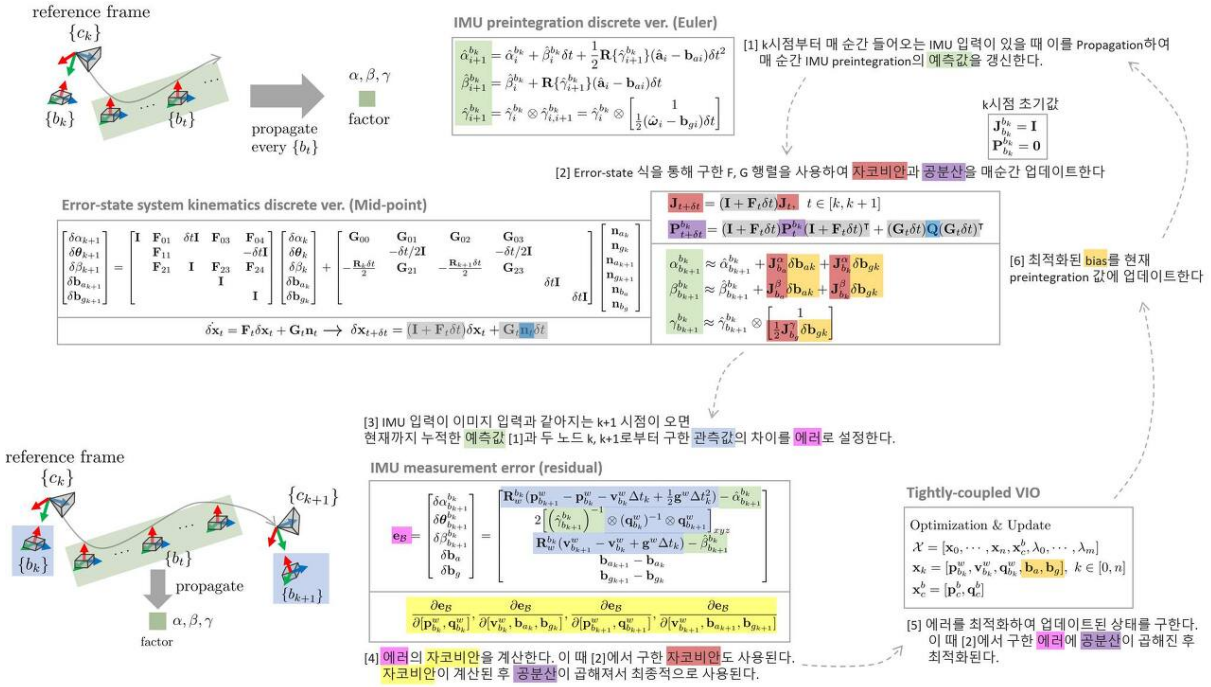
$\frac{\partial \mathcal{L}_c}{\partial \delta_\xi}$ can be obtained as follows:

$$\boxed{\frac{\partial \mathcal{L}_c}{\partial \delta_\xi} = \mathbf{J}_{SO(3)}(\mathcal{L}_c) = \frac{\partial \mathcal{L}_c}{\partial \mathbf{R}} \frac{\partial \mathbf{R}}{\partial \delta_\xi} = \begin{bmatrix} -\mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \begin{bmatrix} -\sin\theta & -\cos\theta & 0 \\ \cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\sin\theta \\ 0 & 0 & \cos\theta \end{bmatrix} \in \mathbb{R}^{6\times 1}} \tag{149}$$

6.5 Code implementations

- Structure PLP SLAM 코드: g2o/se3/pose_opt_edge_line3d_orthonormal.h#L62
- Structure PLP SLAM 코드2: g2o/se3/pose_opt_edge_line3d_orthonormal.h#L81

7 IMU measurement error

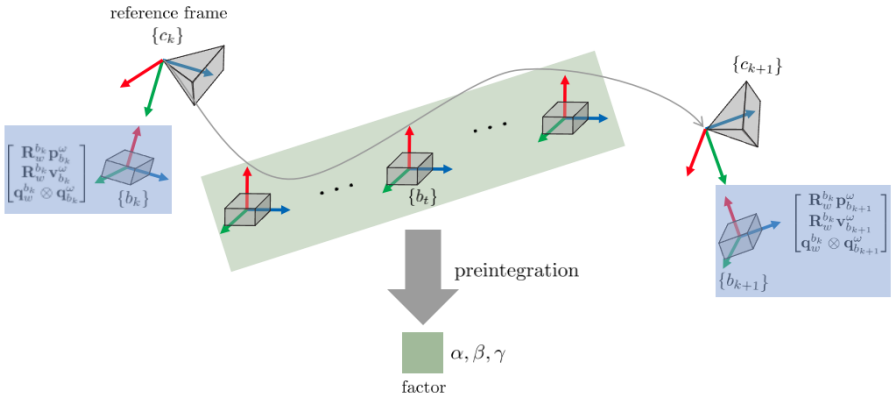


To calculate the error in IMU measurement, it is first necessary to understand IMU preintegration techniques and error-state modeling. The figure above illustrates the overall IMU measurement error-based optimization

process. Steps [1]-[6] should be followed in order. For more details, refer to [SLAM] Formula Derivation and Analysis of VINS-mono content summary.

NOMENCLATURE of IMU measurement error

- $\alpha_{b_{k+1}}^{b_k} \in \mathbb{R}^{3\times1}$: observed accumulated position during $t \in [b_k, b_{k+1}]$
- $\hat{\alpha}_{b_{k+1}}^{b_k} \in \mathbb{R}^{3\times1}$: predicted accumulated position during $t \in [b_k, b_{k+1}]$
- $\beta_{b_{k+1}}^{b_k} \in \mathbb{R}^{3\times1}$: observed accumulated velocity during $t \in [b_k, b_{k+1}]$
- $\hat{\beta}_{b_{k+1}}^{b_k} \in \mathbb{R}^{3\times1}$: predicted accumulated velocity during $t \in [b_k, b_{k+1}]$
- $\gamma_{b_{k+1}}^{b_k} \in \mathbb{R}^{3\times1}$: observed accumulated orientation during $t \in [b_k, b_{k+1}]$
- $\hat{\gamma}_{b_{k+1}}^{b_k} \in \mathbb{R}^{3\times1}$: predicted accumulated orientation during $t \in [b_k, b_{k+1}]$
- $\mathcal{X} = [\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_n, \mathbf{x}_c^b, \lambda_0, \lambda_1, \cdots, \lambda_m]$: all state variables
- $\mathbf{x}_k = [\mathbf{p}_{b_k}^w, \mathbf{v}_{b_k}^w, \mathbf{q}_{b_k}^w, \mathbf{b}_a, \mathbf{b}_g]$: IMU model state variables at specific k
- $\mathbf{x}_c^b = [\mathbf{p}_c^b, \mathbf{q}_c^b]$: extrinsic parameters of the camera and IMU
- $\mathcal{X}_k$: state variables for the specific two points $[b_k, b_{k+1}]$. This is thus $\mathcal{X}_k = (\mathbf{x}_k, \mathbf{x}_{k+1})$.
- λ: inverse depth of feature points
- $\otimes$: quaternion multiplication operator. (e.g., $\mathbf{q} = \mathbf{q}_1 \otimes \mathbf{q}_2$)
- $\mathcal{B}$: set of all IMU b_k values
- $\ominus$: operator for subtracting vectors and quaternions at once
- $\mathbf{P}_{\mathcal{B}}$: covariance of all IMU b_k values
- $\mathbf{\Omega}_{\mathcal{B}}$: inverse matrix of covariance $\mathbf{P}_{\mathcal{B}}$. Represents the information matrix.
- $\mathbf{e}_{\mathcal{B},k} = \mathbf{e}_{\mathcal{B}}(\mathcal{X}_k)$



$$\mathbf{e}_{\mathcal{B}} = \begin{bmatrix} \delta\alpha_{b_{k+1}}^{b_k} \\ \delta\boldsymbol{\theta}_{b_{k+1}}^{b_k} \\ \delta\beta_{b_{k+1}}^{b_k} \\ \delta\mathbf{b}_a \\ \delta\mathbf{b}_g \end{bmatrix} = \begin{bmatrix} \mathbf{R}_w^{b_k}\left(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w - \mathbf{v}_{b_k}^w \Delta t_k + \frac{1}{2}\mathbf{g}^w \Delta t_k^2\right) - \hat{\alpha}_{b_{k+1}}^{b_k} \\ 2\left[\left(\hat{\gamma}_{b_{k+1}}^{b_k}\right)^{-1} \otimes \left(\mathbf{q}_{b_k}^w\right)^{-1} \otimes \mathbf{q}_{b_{k+1}}^w\right]_{xyz} \\ \mathbf{R}_w^{b_k}\left(\mathbf{v}_{b_{k+1}}^w - \mathbf{v}_{b_k}^w + \mathbf{g}^w \Delta t_k\right) - \hat{\beta}_{b_{k+1}}^{b_k} \\ \mathbf{b}_{a_{k+1}} - \mathbf{b}_{a_k} \\ \mathbf{b}_{g_{k+1}} - \mathbf{b}_{g_k} \end{bmatrix}$$

IMU measurement error is defined as the difference between observed and predicted values, similar to the errors described in the previous section. **In detail, the IMU measurement error $\mathbf{e}_{\mathcal{B}}$ refers to the difference between the observed values ($\mathbf{z}_{b_{k+1}}^{b_k}$) and predicted values ($\hat{\mathbf{z}}_{b_{k+1}}^{b_k}$) of the accumulated IMU**

data and bias $[\alpha, \beta, \gamma, \mathbf{b}_a, \mathbf{b}_g]$ over the time $t \in [b_k, b_{k+1}]$.

$$\boxed{\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k) = \mathbf{z}^{b_k}_{b_{k+1}} \ominus \hat{\mathbf{z}}^{b_k}_{b_{k+1}} = \begin{bmatrix} \alpha^{b_k}_{b_{k+1}} - \hat{\alpha}^{b_k}_{b_{k+1}} \\ \beta^{b_k}_{b_{k+1}} - \hat{\beta}^{b_k}_{b_{k+1}} \\ \gamma^{b_k}_{b_{k+1}} \otimes \hat{\gamma}^{b_k}_{b_{k+1}} \\ \mathbf{b}_{a_k} - \hat{\mathbf{b}}_a \\ \mathbf{b}_g - \hat{\mathbf{b}}_g \end{bmatrix}} \tag{150}$$

Let's look in detail at the observed and predicted values. **First, the observed values can be obtained using the positions p, velocities v, and orientations q at two points b_k, b_{k+1}.** The formula for IMU kinematics over the interval $[b_k, b_{k+1}]$ is as follows.

$$\begin{aligned} \mathbf{R}^{b_k}_w \mathbf{p}^w_{b_{k+1}} &= \mathbf{R}^{b_k}_w (\mathbf{p}^w_{b_k} + \mathbf{v}^w_{b_k} \Delta t - \frac{1}{2}\mathbf{g}^w \Delta t_k^2) + \alpha^{b_k}_{b_{k+1}} \\ \mathbf{R}^{b_k}_w \mathbf{v}^w_{b_{k+1}} &= \mathbf{R}^{b_k}_w (\mathbf{v}^w_{b_k} - \mathbf{g}^w \Delta t_k) + \beta^{b_k}_{b_{k+1}} \\ \mathbf{q}^{b_k}_w \otimes \mathbf{q}^w_{b_{k+1}} &= \gamma^{b_k}_{b_{k+1}} \end{aligned} \tag{151}$$

Therefore, the observed values can be calculated as follows.

$$\boxed{\mathbf{z}^{b_k}_{b_{k+1}} = \begin{bmatrix} \alpha^{b_k}_{b_{k+1}} \\ \beta^{b_k}_{b_{k+1}} \\ \gamma^{b_k}_{b_{k+1}} \\ \mathbf{b}_{a_{k+1}} - \mathbf{b}_{a_k} \\ \mathbf{b}_{g_{k+1}} - \mathbf{b}_{g_k} \end{bmatrix} = \begin{bmatrix} \mathbf{R}^{b_k}_w (\mathbf{p}^w_{b_{k+1}} - \mathbf{p}^w_{b_k} - \mathbf{v}^w_{b_k} \Delta t_k + \frac{1}{2}\mathbf{g}^w \Delta t_k^2) \\ \mathbf{R}^{b_k}_w (\mathbf{v}^w_{b_{k+1}} - \mathbf{v}^w_{b_k} + \mathbf{g}^w \Delta t_k) \\ (\mathbf{q}^w_{b_k})^{-1} \otimes \mathbf{q}^w_{b_{k+1}} \\ \mathbf{b}_{a_{k+1}} - \mathbf{b}_{a_k} \\ \mathbf{b}_{g_{k+1}} - \mathbf{b}_{g_k} \end{bmatrix}} \tag{152}$$

Next, the predicted values can be obtained through the accumulated preintegration values during the time $t \in [b_k, b_{k+1}]$. To calculate the predicted values using the preintegration formula, see the following.

$$\begin{aligned} \hat{\alpha}^{b_k}_{b_{k+1}} &= \iint_{t \in [k, k+1]} \mathbf{R}^{b_k}_t (\hat{\mathbf{a}}_t - \mathbf{b}_{at} - \mathbf{n}_a) dt^2 \\ \hat{\beta}^{b_k}_{b_{k+1}} &= \int_{t \in [k, k+1]} \mathbf{R}^{b_k}_t (\hat{\mathbf{a}}_t - \mathbf{b}_{at} - \mathbf{n}_a) dt \\ \hat{\gamma}^{b_k}_{b_{k+1}} &= \int_{t \in [k, k+1]} \frac{1}{2} \Omega_R (\hat{\boldsymbol{\omega}}_t - \mathbf{b}_{gt} - \mathbf{n}_g) \gamma^{b_k}_t dt \end{aligned} \tag{153}$$

The above formula is applicable for continuous signals, but real IMU signals come as discrete signals, so the differential equation should be expressed as a difference equation. In this process, various numerical integration algorithms are used, such as zero-order hold (Euler), first-order hold (mid-point), and higher order (RK4). **Among these, the mid-point method used in VINS-mono is expressed as follows.**

$$\begin{aligned} \hat{\alpha}^{b_k}_{t+1} &= \hat{\alpha}^{b_k}_t + \frac{1}{2}(\hat{\beta}^{b_k}_t + \hat{\beta}^{b_k}_{t+1}) \delta t \\ &= \hat{\alpha}^{b_k}_t + \hat{\beta}^{b_k}_t \delta t + \frac{1}{4}[\mathbf{R}\{\hat{\gamma}^{b_k}_t\}(\hat{\mathbf{a}}_t - \mathbf{b}_{at}) + \mathbf{R}\{\hat{\gamma}^{b_k}_{t+1}\}(\hat{\mathbf{a}}_{t+1} - \mathbf{b}_{at})] \delta t^2 \\ \hat{\beta}^{b_k}_{t+1} &= \hat{\beta}^{b_k}_t + \frac{1}{2}[\mathbf{R}\{\hat{\gamma}^{b_k}_t\}(\hat{\mathbf{a}}_t - \mathbf{b}_{at}) + \mathbf{R}\{\hat{\gamma}^{b_k}_{t+1}\}(\hat{\mathbf{a}}_{t+1} - \mathbf{b}_{at})] \delta t \\ \hat{\gamma}^{b_k}_{t+1} &= \hat{\gamma}^{b_k}_t \otimes \hat{\gamma}^{b_k}_{t,t+1} = \hat{\gamma}^{b_k}_t \otimes \begin{bmatrix} 1 \\ 1/4(\hat{\boldsymbol{\omega}}_t + \hat{\boldsymbol{\omega}}_{t+1} - 2\mathbf{b}_{gt}) \delta t \end{bmatrix} \end{aligned} \tag{154}$$

Thus, the predicted values can be obtained as the accumulated values of (154) over the time $t \in [b_k, b_{k+1}]$. Since bias values cannot be predicted, they are set to zero.

$$\boxed{\hat{\mathbf{z}}^{b_k}_{b_{k+1}} = \begin{bmatrix} \hat{\alpha}^{b_k}_{b_{k+1}} \\ \hat{\beta}^{b_k}_{b_{k+1}} \\ \hat{\gamma}^{b_k}_{b_{k+1}} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}} \tag{155}$$

Based on the values obtained so far, IMU measurement error can be represented as follows.

$$\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k) = \mathbf{z}_{b_{k+1}}^{b_k} \ominus \hat{\mathbf{z}}_{b_{k+1}}^{b_k} = \begin{bmatrix} \mathbf{R}_w^{b_k}(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w - \mathbf{v}_{b_k}^w \Delta t_k + \frac{1}{2}\mathbf{g}^w \Delta t_k^2) - \hat{\alpha}_{b_{k+1}}^{b_k} \\ \mathbf{R}_w^{b_k}(\mathbf{v}_{b_{k+1}}^w - \mathbf{v}_{b_k}^w + \mathbf{g}^w \Delta t_k) - \hat{\beta}_{b_{k+1}}^{b_k} \\ \left(\hat{\gamma}_{b_{k+1}}^{b_k}\right)^{-1} \otimes (\mathbf{q}_{b_k}^w)^{-1} \otimes \mathbf{q}_{b_{k+1}}^w \\ \mathbf{b}_{a_{k+1}} - \mathbf{b}_{a_k} \\ \mathbf{b}_{g_{k+1}} - \mathbf{b}_{g_k} \end{bmatrix} \tag{156}$$

7.1 Error function formulation

The error function for all preintegrations and biases is defined as follows.

$$\mathbf{E}_{\mathcal{B}}(\mathcal{X}) = \sum_{k \in \mathcal{B}} \|\mathbf{e}_{\mathcal{B},k}\|_{\mathbf{P}_{\mathcal{B}}}^2 \tag{157}$$

$$\begin{aligned} \mathcal{X}^* &= \arg\min_{\mathcal{X}^*} \mathbf{E}_{\mathcal{B}}(\mathcal{X}) \\ &= \arg\min_{\mathcal{X}^*} \sum_{k \in \mathcal{B}} \|\mathbf{e}_{\mathcal{B},k}\|_{\mathbf{P}_{\mathcal{B}}}^2 \\ &= \arg\min_{\mathcal{X}^*} \sum_{k \in \mathcal{B}} \mathbf{e}_{\mathcal{B},k}^{\intercal} \mathbf{\Omega}_{\mathcal{B}} \mathbf{e}_{\mathcal{B},k} \\ &= \arg\min_{\mathcal{X}^*} \sum_{k \in \mathcal{B}} (\mathbf{z}_{b_{k+1}}^{b_k} \ominus \hat{\mathbf{z}}_{b_{k+1}}^{b_k})^{\intercal} \mathbf{\Omega}_{\mathcal{B}} (\mathbf{z}_{b_{k+1}}^{b_k} \ominus \hat{\mathbf{z}}_{b_{k+1}}^{b_k}) \end{aligned} \tag{158}$$

The formula $\mathbf{e}_{\mathcal{B},k} = \mathbf{e}_{\mathcal{B}}(\mathcal{X}_k)$ is implied here.

> **Tip**
>
> In actual VINS-mono implementation, not only the IMU measurement error but also the visual residual $\mathbf{r}_{\mathcal{C}}$, marginalization prior residual $\mathbf{r}_p$ are simultaneously optimized to perform tightly-coupled VIO. In VINS-mono, the IMU measurement error is expressed as the residual $\mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X})$.
>
> $$\min_{\mathcal{X}} \left\{ \|\mathbf{r}_p - \mathbf{J}_p \mathcal{X}\|_{\mathbf{P}_M} + \sum_{k \in \mathcal{B}} \left\| \mathbf{r}_{\mathcal{B}}\left(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X}\right) \right\|_{\mathbf{P}_B} + \sum_{(l,j) \in \mathcal{C}} \left\| \mathbf{r}_{\mathcal{C}}\left(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X}\right) \right\|_{\mathbf{P}_l^{c_j}} \right\} \tag{159}$$
>
> This section explains only the IMU measurement error $\mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X})$.

$\mathbf{E}_{\mathcal{B}}(\mathcal{X}^*)$ that satisfies $\|\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k^*)\|_{\mathbf{P}_{\mathcal{B}}}^2$ can be iteratively computed through non-linear least squares. Small increments $\Delta\mathcal{X}$ are iteratively updated to $\mathcal{X}$ to find the optimal state.

$$\arg\min_{\mathcal{X}^*} \mathbf{E}_{\mathcal{B}}(\mathcal{X} + \Delta\mathcal{X}) = \arg\min_{\mathcal{X}^*} \sum_{k \in \mathcal{B}} \|\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k + \Delta\mathcal{X}_k)\|^2 \tag{160}$$

Strictly speaking, since the state increment $\Delta\mathcal{X}$ includes quaternions, it should be added to the existing state $\mathcal{X}$ using the $\oplus$ operator, but the $+$ operator is used for simplicity of expression.

$$\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k \oplus \Delta\mathcal{X}_k) \quad \rightarrow \quad \mathbf{e}_{\mathcal{B}}(\mathcal{X}_k + \Delta\mathcal{X}_k) \tag{161}$$

The above equation can be expressed through a first-order Taylor approximation as follows.

$$\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k + \Delta\mathcal{X}_k) \approx \mathbf{e}_{\mathcal{B}}(\mathcal{X}) + \mathbf{J}\Delta\mathcal{X}_k$$

$$\begin{aligned}
&= \mathbf{e}_{\mathcal{B}}(\mathcal{X}_k) + \begin{bmatrix} \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{p}^w_{b_k}, \mathbf{q}^w_{b_k}]} & \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}^w_{b_k}, \mathbf{b}_{ak}, \mathbf{b}_{gk}]} & \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{p}^w_{b_{k+1}}, \mathbf{q}^w_{b_{k+1}}]} & \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}^w_{b_{k+1}}, \mathbf{b}_{ak+1}, \mathbf{b}_{ak+1}]} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{p}^w_k \\ \Delta\mathbf{q}^w_k \\ \Delta\mathbf{v}^w_k \\ \Delta\mathbf{b}_{ak} \\ \Delta\mathbf{b}_{gk} \\ \Delta\mathbf{p}^w_{k+1} \\ \Delta\mathbf{q}^w_{k+1} \\ \Delta\mathbf{v}^w_{k+1} \\ \Delta\mathbf{b}_{ak+1} \\ \Delta\mathbf{b}_{gk+1} \end{bmatrix} \\
&= \mathbf{e}_{\mathcal{B}}(\mathcal{X}_k) + \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{p}^w_{b_k}, \mathbf{q}^w_{b_k}]}(\Delta\mathbf{p}^w_k, \Delta\mathbf{q}^w_k) + \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}^w_{b_k}, \mathbf{b}_{ak}, \mathbf{b}_{gk}]}(\Delta\mathbf{v}^w_k, \Delta\mathbf{b}_{ak}, \Delta\mathbf{b}_{gk}) \\
&+ \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{p}^w_{b_{k+1}}, \mathbf{q}^w_{b_{k+1}}]}(\Delta\mathbf{p}^w_{k+1}, \Delta\mathbf{q}^w_{k+1}) + \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}^w_{b_{k+1}}, \mathbf{b}_{ak+1}, \mathbf{b}_{ak+1}]}(\Delta\mathbf{v}^w_{k+1}, \Delta\mathbf{b}_{ak+1}, \Delta\mathbf{b}_{gk+1})
\end{aligned} \tag{162}$$

Both $[\mathbf{p}^w_{b_k}, \mathbf{v}^w_{b_k}, \mathbf{q}^w_{b_k}, \mathbf{b}_{a_k}, \mathbf{b}_{g_k}]$ at the point b_k and $[\mathbf{p}^w_{b_{k+1}}, \mathbf{v}^w_{b_{k+1}}, \mathbf{q}^w_{b_{k+1}}, \mathbf{b}_{a_{k+1}}, \mathbf{b}_{g_{k+1}}]$ at the point b_{k+1} are involved in the error value, so the Jacobian for all 10 variables must be calculated. In VINS-mono, state variables are grouped into 4 groups as follows.

$$\begin{aligned}
&[\mathbf{p}^w_{b_k}, \mathbf{q}^w_{b_k}] && \cdots \text{ for } \mathbf{J}[0] \\
&[\mathbf{v}^w_{b_k}, \mathbf{b}_{ak}, \mathbf{b}_{gk}] && \cdots \text{ for } \mathbf{J}[1] \\
&[\mathbf{p}^w_{b_{k+1}}, \mathbf{q}^w_{b_{k+1}}] && \cdots \text{ for } \mathbf{J}[2] \\
&[\mathbf{v}^w_{b_{k+1}}, \mathbf{b}_{ak+1}, \mathbf{b}_{ak+1}] && \cdots \text{ for } \mathbf{J}[3]
\end{aligned} \tag{163}$$

Tightly-coupled VIO optimizes the state variables $\mathcal{X}$ which include the inverse depth λ and external parameters (extrinsic parameters) $\mathbf{x}^b_c$, time difference td, but it is important to note that in the IMU measurement error, only pose, velocity, and bias values for two points $[b_k, b_{k+1}]$ are updated.

The error function can be approximated as follows.

$$\arg\min_{\mathcal{X}^*} \mathbf{E}_{\mathcal{B}}(\mathcal{X} + \Delta\mathcal{X}) \approx \arg\min_{\mathcal{X}^*} \sum_{k\in\mathcal{B}} \|\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k) + \mathbf{J}\Delta\mathcal{X}_k\|^2_{\mathbf{P}_{\mathcal{B}}} \tag{164}$$

Differentiating this to find the optimal increment $\Delta\mathcal{X}^*$ results in the following. The detailed derivation process is omitted in this section. For a detailed derivation, refer to the previous section.

$$\begin{aligned}
\mathbf{J}^\intercal\mathbf{J}\Delta\mathcal{X}^* &= -\mathbf{J}^\intercal\mathbf{e} \\
\mathbf{H}\Delta\mathcal{X}^* &= -\mathbf{b}
\end{aligned} \tag{165}$$

This equation is in the form of a linear system $\mathbf{Ax} = \mathbf{b}$, so $\Delta\mathcal{X}^*$ can be found using various linear algebra techniques such as schur complement, cholesky decomposition. The optimal increment found in this way is added to the current state. **In this case, whether the existing state x is multiplied on the right or left determines whether the pose viewed from the local coordinate system is updated (right) or the pose viewed from the global coordinate system is updated (left). Since IMU measurement error is related to two nodes b_k, b_{k+1}, right multiplication applicable to local coordinate system updates is applied.**

$$\mathcal{X} \leftarrow \mathcal{X} \oplus \Delta\mathcal{X}^* \tag{166}$$

$\mathcal{X}$ being updated by IMU measurement error $\mathcal{X}_k$ consists of $[\mathbf{p}^w_{b_k}, \mathbf{v}^w_{b_k}, \mathbf{q}^w_{b_k}, \mathbf{b}_{a_k}, \mathbf{b}_{g_k}, \mathbf{p}^w_{b_{k+1}}, \mathbf{v}^w_{b_{k+1}}, \mathbf{q}^w_{b_{k+1}}, \mathbf{b}_{a_{k+1}}, \mathbf{b}_{g_{k+1}}]$

so it can be expressed as follows.

$$
\begin{aligned}
\mathbf{p}_{b_k}^{w} &\leftarrow \mathbf{p}_{b_k}^{w} \oplus \Delta\mathbf{p}_{b_k}^{w*} \\
\mathbf{q}_{b_k}^{w} &\leftarrow \mathbf{q}_{b_k}^{w} \oplus \Delta\mathbf{q}_{b_k}^{w*} \\
\mathbf{v}_{b_k}^{w} &\leftarrow \mathbf{v}_{b_k}^{w} \oplus \Delta\mathbf{v}_{b_k}^{w*} \\
\mathbf{b}_{ak} &\leftarrow \mathbf{b}_{ak} \oplus \Delta\mathbf{b}_{ak}^{*} \\
\mathbf{b}_{gk} &\leftarrow \mathbf{b}_{gk} \oplus \Delta\mathbf{b}_{gk}^{*} \\
\mathbf{p}_{b_{k+1}}^{w} &\leftarrow \mathbf{p}_{b_{k+1}}^{w} \oplus \Delta\mathbf{p}_{b_{k+1}}^{w*} \\
\mathbf{q}_{b_{k+1}}^{w} &\leftarrow \mathbf{q}_{b_{k+1}}^{w} \oplus \Delta\mathbf{q}_{b_{k+1}}^{w*} \\
\mathbf{v}_{b_{k+1}}^{w} &\leftarrow \mathbf{v}_{b_{k+1}}^{w} \oplus \Delta\mathbf{v}_{b_{k+1}}^{w*} \\
\mathbf{b}_{ak+1} &\leftarrow \mathbf{b}_{ak+1} \oplus \Delta\mathbf{b}_{ak+1}^{*} \\
\mathbf{b}_{gk+1} &\leftarrow \mathbf{b}_{gk+1} \oplus \Delta\mathbf{b}_{gk+1}^{*}
\end{aligned}
\tag{167}
$$

Right multiplication $\oplus$ operation definition is as follows.

$$
\begin{aligned}
\mathbf{p}_{b_k}^{w} &\leftarrow \mathbf{p}_{b_k}^{w} + \Delta\mathbf{p}_{b_k}^{w*} \\
\mathbf{q}_{b_k}^{w} &\leftarrow \mathbf{q}_{b_k}^{w} \otimes \Delta\mathbf{q}_{b_k}^{w*} \quad \cdots \text{ locally updated (right mult)} \\
\mathbf{v}_{b_k}^{w} &\leftarrow \mathbf{v}_{b_k}^{w} + \Delta\mathbf{v}_{b_k}^{w*} \\
\mathbf{b}_{ak} &\leftarrow \mathbf{b}_{ak} + \Delta\mathbf{b}_{ak}^{*} \\
\mathbf{b}_{gk} &\leftarrow \mathbf{b}_{gk} + \Delta\mathbf{b}_{gk}^{*} \\
\mathbf{p}_{b_{k+1}}^{w} &\leftarrow \mathbf{p}_{b_{k+1}}^{w} + \Delta\mathbf{p}_{b_{k+1}}^{w*} \\
\mathbf{q}_{b_{k+1}}^{w} &\leftarrow \mathbf{q}_{b_{k+1}}^{w} \otimes \Delta\mathbf{q}_{b_{k+1}}^{w*} \quad \cdots \text{ locally updated (right mult)} \\
\mathbf{v}_{b_{k+1}}^{w} &\leftarrow \mathbf{v}_{b_{k+1}}^{w} + \Delta\mathbf{v}_{b_{k+1}}^{w*} \\
\mathbf{b}_{ak+1} &\leftarrow \mathbf{b}_{ak+1} + \Delta\mathbf{b}_{ak+1}^{*} \\
\mathbf{b}_{gk+1} &\leftarrow \mathbf{b}_{gk+1} + \Delta\mathbf{b}_{gk+1}^{*}
\end{aligned}
\tag{168}
$$

7.2 Jacobian of IMU measurement error

To perform (165), the Jacobian $\mathbf{J}$ for the IMU measurement error must be calculated. It can be represented as follows.

$$
\begin{aligned}
\mathbf{J} &= \begin{bmatrix} \mathbf{J}[0] & \mathbf{J}[1] & \mathbf{J}[2] & \mathbf{J}[3] \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{p}_{b_k}^{w}, \mathbf{q}_{b_k}^{w}]} & \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}_{b_k}^{w}, \mathbf{b}_{ak}, \mathbf{b}_{gk}]} & \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{p}_{b_{k+1}}^{w}, \mathbf{q}_{b_{k+1}}^{w}]} & \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}_{b_{k+1}}^{w}, \mathbf{b}_{ak+1}, \mathbf{b}_{ak+1}]} \end{bmatrix} \\
&= \frac{\partial}{\partial[\mathbf{p}_{b_k}^{w}, \mathbf{q}_{b_k}^{w}], [\mathbf{v}_{b_k}^{w}, \mathbf{b}_{ak}, \mathbf{b}_{gk}], [\mathbf{p}_{b_{k+1}}^{w}, \mathbf{q}_{b_{k+1}}^{w}], [\mathbf{v}_{b_{k+1}}^{w}, \mathbf{b}_{ak+1}, \mathbf{b}_{ak+1}]} \begin{bmatrix} \mathbf{R}_{w}^{b_k}(\mathbf{p}_{b_{k+1}}^{w} - \mathbf{p}_{b_k}^{w} - \mathbf{v}_{b_k}^{w}\Delta t_k + \frac{1}{2}\mathbf{g}^{w}\Delta t_k^2) - \hat{\alpha}_{b_{k+1}}^{b_k} \\ \mathbf{R}_{w}^{b_k}(\mathbf{v}_{b_{k+1}}^{w} - \mathbf{v}_{b_k}^{w} + \mathbf{g}^{w}\Delta t_k) - \hat{\beta}_{b_{k+1}}^{b_k} \\ \left(\hat{\gamma}_{b_{k+1}}^{b_k}\right); -1 \otimes (\mathbf{q}_{b_k}^{w}); -1 \otimes \mathbf{q}_{b_{k+1}}^{w} \\ \mathbf{b}_{a_{k+1}} - \mathbf{b}_{a_k} \\ \mathbf{b}_{g_{k+1}} - \mathbf{b}_{g_k} \end{bmatrix} \\
&= \begin{bmatrix} \mathbb{R}^{15\times 7} & \mathbb{R}^{15\times 9} & \mathbb{R}^{15\times 7} & \mathbb{R}^{15\times 9} \end{bmatrix} = \mathbb{R}^{15\times 32}
\end{aligned}
\tag{169}
$$

7.2.1 Lie theory-based SO(3) optimization

When calculating the above Jacobian, terms related to position p, velocity v, and biases $\mathbf{b}_a, \mathbf{b}_g$ are each 3-dimensional vectors, so they do not have any constraints when performing optimization updates. However, the quaternion q has 4 parameters and represents 3 degrees of freedom, which is more than the minimal degrees of freedom required to represent 3-dimensional rotation, thus having various constraints. This is known as being over-parameterized. The disadvantages of over-parameterized representation include:

- Increased computation due to redundant parameters during optimization.
- Potential numerical instability issues due to additional degrees of freedom.
- The need to ensure that constraints are met every time parameters are updated.

Using lie theory, optimization can be performed free from constraints. Therefore, instead of using quaternion $\mathbf{q}$, lie algebra so(3) $[\boldsymbol{\theta}]_\times$ is used, freeing parameters from constraints. Here, $\boldsymbol{\theta} \in \mathbb{R}^3$ represents the angular velocity vector. Detailed content on SO(3)-based optimization is the same as in the reprojection error section and is omitted here.

When using angular velocity vector $\boldsymbol{\theta}$, the original Jacobian of quaternion $\mathbf{q}$ is changed as follows.

$$\begin{aligned} \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial \mathbf{q}_{b_k}^w} &\rightarrow \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial \begin{bmatrix} 1 & \frac{1}{2}\boldsymbol{\theta}_{b_k}^w \end{bmatrix}} \\ \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial \mathbf{q}_{b_{k+1}}^w} &\rightarrow \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial \begin{bmatrix} 1 & \frac{1}{2}\boldsymbol{\theta}_{b_{k+1}}^w \end{bmatrix}} \end{aligned} \tag{170}$$

Tip

Given an arbitrary angle-axis vector $\boldsymbol{\theta} = \theta\mathbf{u}$, its corresponding exponential map can be expressed using an extended version of Euler's formula.

$$\mathbf{q} \triangleq \operatorname{Exp}(\boldsymbol{\theta}) = \operatorname{Exp}(\theta\mathbf{u}) = e^{\theta\mathbf{u}/2} = \cos\frac{\theta}{2} + \mathbf{u}\sin\frac{\theta}{2} = \begin{bmatrix} \cos(\theta/2) \\ \mathbf{u}\sin(\theta/2) \end{bmatrix} \tag{171}$$

For sufficiently small $\boldsymbol{\theta}$ values, $\cos\frac{\theta}{2} \approx 1$ and $\sin\frac{\theta}{2} \approx \frac{\theta}{2}$ hold true, thus the following formula for sufficiently small quaternion values is valid.

$$\mathbf{q} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\boldsymbol{\theta} \end{bmatrix} \tag{172}$$

More details can be found in the Quaternion kinematics for the error-state Kalman filter content summary post, section 4.4.

Typically, the errors used in optimization are small, so it is assumed that the error $\left(\hat{\gamma}_{b_{k+1}}^{b_k}\right)^{-1} \otimes (\mathbf{q}_{b_k}^w)^{-1} \otimes \mathbf{q}_{b_{k+1}}^w$ is also small. Therefore, only the imaginary part $[x, y, z] = \frac{1}{2}\boldsymbol{\theta}$ of the actual quaternion $\mathbf{q} = [w, x, y, z]$ is used in optimization. Through this, the γ part is transformed as follows.

$$\begin{aligned} \gamma &\rightarrow 2[\gamma]_{xyz} = 2[x, y, z] = \boldsymbol{\theta} \\ \left(\hat{\gamma}_{b_{k+1}}^{b_k}\right)^{-1} \otimes (\mathbf{q}_{b_k}^w)^{-1} \otimes \mathbf{q}_{b_{k+1}}^w &\rightarrow 2\left[\left(\hat{\gamma}_{b_{k+1}}^{b_k}\right)^{-1} \otimes (\mathbf{q}_{b_k}^w)^{-1} \otimes \mathbf{q}_{b_{k+1}}^w\right]_{xyz} \end{aligned} \tag{173}$$

The final SO(3) version IMU measurement error $\mathbf{e}_{\mathcal{B}}$ is as follows.

$$\mathbf{e}_{\mathcal{B}}(\mathcal{X}_k) = \begin{bmatrix} \mathbf{R}_w^{b_k}(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w - \mathbf{v}_{b_k}^w \Delta t_k + \frac{1}{2}\mathbf{g}^w \Delta t_k^2) - \hat{\alpha}_{b_{k+1}}^{b_k} \\ 2\left[\left(\hat{\gamma}_{b_{k+1}}^{b_k}\right)^{-1} \otimes (\mathbf{q}_{b_k}^w)^{-1} \otimes \mathbf{q}_{b_{k+1}}^w\right]_{xyz} \\ \mathbf{R}_w^{b_k}(\mathbf{v}_{b_{k+1}}^w - \mathbf{v}_{b_k}^w + \mathbf{g}^w \Delta t_k) - \hat{\beta}_{b_{k+1}}^{b_k} \\ \mathbf{b}_{a_{k+1}} - \mathbf{b}_{a_k} \\ \mathbf{b}_{g_{k+1}} - \mathbf{b}_{g_k} \end{bmatrix} \tag{174}$$

For easier calculation of Jacobians for $[\mathbf{p}, \mathbf{q}], [\mathbf{v}, \mathbf{b}_a, \mathbf{b}_g]$, the order of the second line β and the third line γ in the original state variables was switched.

The final SO(3) version IMU measurement error Jacobian can be calculated as follows. The detailed derivation process can be referred to in the Formula Derivation and Analysis of VINS-Mono paper's Appendix section.

$$\mathbf{J}[0]_{15\times 6} = \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial [\mathbf{p}_{b_k}^w, \mathbf{q}_{b_k}^w]} = \begin{bmatrix} -\mathbf{R}_w^{b_k} & [\mathbf{R}_w^{b_k}(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w - \mathbf{v}_{b_k}^w \Delta t_k + \frac{1}{2}\mathbf{g}^w \Delta t_k^2)]_\times \\ 0 & [\gamma_{b_{k+1}}^{b_k}]_R [(\mathbf{q}_{b_{k+1}}^w)^{-1} \otimes \mathbf{q}_w^{b_k}]_{L,3\times 3} \\ 0 & [\mathbf{R}_w^{b_k}(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w + \mathbf{g}^w \Delta t_k)]_\times \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{175}$$

$$\mathbf{J}[1]_{15\times 9} = \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}^w_{b_k}, \mathbf{b}_{a_k}, \mathbf{b}_{g_k}]} = \begin{bmatrix} -\mathbf{R}^{b_k}_w \Delta t_k & -\mathbf{J}^{\alpha}_{b_a} & -\mathbf{J}^{\alpha}_{b_g} \\ 0 & 0 & -[(\hat{\gamma}^{b_k}_{b_{k+1}})^{-1} \otimes (\mathbf{q}^w_{b_k})^{-1} \otimes \mathbf{q}^w_{b_{k+1}}]_{R,3\times 3}\mathbf{J}^{\gamma}_{b_g} \\ -\mathbf{R}^{b_k}_w & -\mathbf{J}^{\beta}_{b_a} & -\mathbf{J}^{\beta}_{b_g} \\ 0 & -\mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{bmatrix} \tag{176}$$

$$\mathbf{J}[2]_{15\times 6} = \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{p}^w_{b_{k+1}}, \mathbf{q}^w_{b_{k+1}}]} = \begin{bmatrix} \mathbf{R}^{b_k}_w & 0 \\ 0 & [(\hat{\gamma}^{b_k}_{b_{k+1}})^{-1} \otimes (\mathbf{q}^w_{b_k})^{-1} \otimes \mathbf{q}^w_{b_{k+1}}]_L \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{177}$$

$$\mathbf{J}[3]_{15\times 9} = \frac{\partial \mathbf{e}_{\mathcal{B}}}{\partial[\mathbf{v}^w_{b_{k+1}}, \mathbf{b}_{a_{k+1}}, \mathbf{b}_{g_{k+1}}]} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mathbf{R}^{b_k}_w & 0 & 0 \\ 0 & \mathbf{I} & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix} \tag{178}$$

NOTICE: The original $\mathbf{J}[0], \mathbf{J}[2] \in \mathbb{R}^{15\times 7}$, but since quaternion is updated based on SO(3) using only $[xyz]$ part, w part is always 0. By omitting the w part, $\mathbf{J}[0], \mathbf{J}[2] \in \mathbb{R}^{15\times 6}$.

NOTICE: Looking at the above formula, it can be seen that another Jacobian $\mathbf{J}^{\alpha}_{b_a}, \mathbf{J}^{\alpha}_{b_g}, \mathbf{J}^{\beta}_{b_a}, \mathbf{J}^{\beta}_{b_g} \mathbf{J}^{\gamma}_{b_g}$ is used within the Jacobian. This refers to the partial Jacobians derived from the error-state equations of the IMU $\mathbf{J}^{b_k}_{b_{k+1}}$.

Tip

The error-state equation for the discrete IMU signal is as follows. (Using Mid-point approximation)

$$\begin{bmatrix} \delta\alpha_{k+1} \\ \delta\boldsymbol{\theta}_{k+1} \\ \delta\beta_{k+1} \\ \delta\mathbf{b}_{a_{k+1}} \\ \delta\mathbf{b}_{g_{k+1}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{F}_{01} & \delta t\mathbf{I} & \mathbf{F}_{03} & \mathbf{F}_{04} \\ & \mathbf{F}_{11} & & & -\delta t\mathbf{I} \\ & \mathbf{F}_{21} & \mathbf{I} & \mathbf{F}_{23} & \mathbf{F}_{24} \\ & & & \mathbf{I} & \\ & & & & \mathbf{I} \end{bmatrix} \begin{bmatrix} \delta\alpha_k \\ \delta\boldsymbol{\theta}_k \\ \delta\beta_k \\ \delta\mathbf{b}_{a_k} \\ \delta\mathbf{b}_{g_k} \end{bmatrix} + \begin{bmatrix} \mathbf{G}_{00} & \mathbf{G}_{01} & \mathbf{G}_{02} & \mathbf{G}_{03} & & \\ & -\delta t/2\mathbf{I} & & -\delta t/2\mathbf{I} & & \\ -\frac{\mathbf{R}_k\delta t}{2} & \mathbf{G}_{21} & -\frac{\mathbf{R}_{k+1}\delta t}{2} & \mathbf{G}_{23} & & \\ & & & & \delta t\mathbf{I} & \\ & & & & & \delta t\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{n}_{a_k} \\ \mathbf{n}_{g_k} \\ \mathbf{n}_{a_{k+1}} \\ \mathbf{n}_{g_{k+1}} \\ \mathbf{n}_{b_a} \\ \mathbf{n}_{b_g} \end{bmatrix} \tag{179}$$

Here, the Jacobian for state variables $\mathbf{J}^{b_k}_t$ is updated as follows.

$$\mathbf{J}^{b_k}_{t+\delta t} = (\mathbf{I} + \mathbf{F}_t\delta t)\mathbf{J}^{b_k}_t, \quad t \in [k, k+1] \tag{180}$$

For more details, refer to the [SLAM] Formula Derivation and Analysis of VINS-mono content summary post, sections 2.3, 2.4.

7.3 Code implementations

- VINS-mono code: integration_base.h#L180
  - SO(3) version IMU measurement error $\mathbf{e}_{\mathcal{B}}$ is implemented here.
- VINS-mono code: imu_factor.h#L86
  - $\mathbf{J}[0], \mathbf{J}[1], \mathbf{J}[2], \mathbf{J}[3]$ are implemented here.
  - Jacobian and error function are multiplied by the square root inverse of covariance $\sqrt{(\mathbf{P}^{b_k}_{b_{k+1}})^{-1}} = \sqrt{\boldsymbol{\Omega}_{\mathcal{B}}}$ in the form of information matrix.
    - $\mathbf{e}_{\mathcal{B},k} \rightarrow \sqrt{\boldsymbol{\Omega}_{\mathcal{B}}}^{\intercal}\mathbf{e}_{\mathcal{B},k}$: in actual code implementation, the right error term is optimized.
    - This is because the error function $\mathbf{E}_{\mathcal{B}}(\mathcal{X}) = \mathbf{e}^{\intercal}_{\mathcal{B},k}\boldsymbol{\Omega}_{\mathcal{B}}\mathbf{e}_{\mathcal{B},k}$ is set in the code as the square root $\sqrt{\boldsymbol{\Omega}_{\mathcal{B}}}^{\intercal}\mathbf{e}_{\mathcal{B},k}$.

- VINS-mono code: integration_base.h#L90
  - The state transition matrices $\mathbf{F}, \mathbf{G}$ for error state equations approximated by Mid-point method are implemented here.
  - Jacobian update formula for IMU state variables $\mathbf{J}_{t+\delta t}^{b_k} = (\mathbf{I} + \mathbf{F}_t \delta t)\mathbf{J}_t^{b_k}$ is implemented here.
  - Covariance update formula for IMU state variables $\mathbf{P}_{t+\delta t}^{b_k} = (\mathbf{I}+\mathbf{F}_t \delta t)\mathbf{P}_t^{b_k}(\mathbf{I}+\mathbf{F}_t \delta t)^\intercal + (\mathbf{G}_t \delta t)\mathbf{Q}(\mathbf{G}_t \delta t)^\intercal$ is implemented here.

8 Other Jacobians

8.1 Jacobian of unit quaternion

NOMENCLATURE of Jacobian of unit quaternion

- $\mathbf{X} = [X, Y, Z, 1]^\intercal = [\tilde{\mathbf{X}}, 1]^\intercal \in \mathbb{P}^3$
- $\tilde{\mathbf{X}} = [X, Y, Z]^\intercal \in \mathbb{P}^2$
- $\mathbf{q} = [w, x, y, z]^\intercal = [w, \mathbf{v}]^\intercal$
  - Quaternion represented using Hamilton notation. For detailed information, refer to this post.

As explained in the previous section on reprojection error, the Jacobian is as follows:

$$\mathbf{J}_c = \frac{\partial \hat{\mathbf{p}}}{\partial \tilde{\mathbf{p}}} \frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{X}'} \frac{\partial \mathbf{X}'}{\partial [\mathbf{R}, \mathbf{t}]} \tag{181}$$

Among these, $\frac{\partial \mathbf{X}'}{\partial \mathbf{R}}$ is a Jacobian that can be used when rotation is represented by rotation matrix $\mathbf{R}$. In this section, the Jacobian $\frac{\partial \mathbf{X}'}{\partial \mathbf{q}}$ that can be used when the rotation is represented by the unit quaternion $\mathbf{q}$ is described.

When a point $\mathbf{X}$ in three-dimensional space is given, the point $\mathbf{X}'$ rotated by an arbitrary unit quaternion $\mathbf{q}$ can be represented as follows:

$$\tilde{\mathbf{X}'} = \mathbf{q} \otimes \tilde{\mathbf{X}} \otimes \mathbf{q}^* \tag{182}$$

Expanding this further:

$$\begin{aligned}
\tilde{\mathbf{X}'} &= \mathbf{q} \otimes \tilde{\mathbf{X}} \otimes \mathbf{q}^* \\
&= (w + \mathbf{v}) \otimes \tilde{\mathbf{X}} \otimes (w - \mathbf{v}) \\
&= w^2 \tilde{\mathbf{X}} + w(\mathbf{v} \otimes \tilde{\mathbf{X}} - \tilde{\mathbf{X}} \otimes \mathbf{v}) - \mathbf{v} \otimes \tilde{\mathbf{X}} \otimes \mathbf{v} \\
&= w^2 \tilde{\mathbf{X}} + 2w(\mathbf{v} \times \tilde{\mathbf{X}}) - [(-\mathbf{v}^\intercal \tilde{\mathbf{X}} + \mathbf{v} \times \tilde{\mathbf{X}}) \otimes \mathbf{v}] \\
&= w^2 \tilde{\mathbf{X}} + 2w(\mathbf{v} \times \tilde{\mathbf{X}}) - [(-\mathbf{v}^\intercal \tilde{\mathbf{X}})\mathbf{v} + (\mathbf{v} \times \tilde{\mathbf{X}}) \otimes \mathbf{v}] \\
&= w^2 \tilde{\mathbf{X}} + 2w(\mathbf{v} \times \tilde{\mathbf{X}}) - [(-\mathbf{v}^\intercal \tilde{\mathbf{X}})\mathbf{v} - (\mathbf{v} \times \tilde{\mathbf{X}}) \times \mathbf{v}] \\
&= w^2 \tilde{\mathbf{X}} + 2w(\mathbf{v} \times \tilde{\mathbf{X}}) + 2(\mathbf{v}^\intercal \tilde{\mathbf{X}})\mathbf{v} - (\mathbf{v}^\intercal \mathbf{v})\tilde{\mathbf{X}}
\end{aligned} \tag{183}$$

Using this, the Jacobian with respect to the quaternion $\frac{\partial \tilde{\mathbf{X}'}}{\partial \mathbf{q}}$ can be determined. It is divided into the scalar part $\frac{\partial \tilde{\mathbf{X}'}}{\partial w}$ and the vector part $\frac{\partial \tilde{\mathbf{X}'}}{\partial \mathbf{v}}$ as follows:

$$\begin{aligned}
\frac{\partial \tilde{\mathbf{X}'}}{\partial w} &= 2(w\tilde{\mathbf{X}} + \mathbf{v} \times \tilde{\mathbf{X}}) \\
\frac{\partial \tilde{\mathbf{X}'}}{\partial \mathbf{v}} &= -2w[\tilde{\mathbf{X}}]_\times + 2(\mathbf{v}^\intercal \tilde{\mathbf{X}}\mathbf{I} + \mathbf{v}\tilde{\mathbf{X}}^\intercal) - 2\tilde{\mathbf{X}}\mathbf{v}^\intercal \\
&= 2(\mathbf{v}^\intercal \tilde{\mathbf{X}}\mathbf{I} + \mathbf{v}\tilde{\mathbf{X}}^\intercal - \tilde{\mathbf{X}}\mathbf{v}^\intercal - w[\tilde{\mathbf{X}}]_\times)
\end{aligned} \tag{184}$$

In this case, the $\tilde{\mathbf{X}}$ entering in the middle of quaternion multiplication is actually transformed into the form of a pure quaternion $[0, X, Y, Z]^\intercal$ with a scalar value of 0. **Therefore, in the above formula, the**

Jacobian with respect to the scalar $\frac{\partial \tilde{\mathbf{X}}'}{\partial w}$ is not calculated separately because it is not used in actual optimization, and only the Jacobian with respect to the vector $\frac{\partial \tilde{\mathbf{X}}'}{\partial \mathbf{v}}$ is calculated.

$$\begin{gathered}\tilde{\mathbf{X}}' = \mathbf{q} \otimes \tilde{\mathbf{X}} \otimes \mathbf{q}^* \quad \rightarrow \quad \begin{bmatrix} 0 \\ \tilde{\mathbf{X}}' \end{bmatrix} = \mathbf{q} \otimes \begin{bmatrix} 0 \\ \tilde{\mathbf{X}} \end{bmatrix} \otimes \mathbf{q}^* \quad \cdots \text{ strict notation} \\ \text{Then, } \frac{\partial \tilde{\mathbf{X}}'}{\partial w} \text{ is going to be useless}\end{gathered} \tag{185}$$

Additionally, assuming that the quaternion $\mathbf{q}$ is sufficiently small, it can be approximated as the identity ($\mathbf{q} \approx \mathbf{q}_1 = [1, 0, 0, 0]^\intercal$), similar to the method previously used to approximate a sufficiently small rotation matrix $\mathbf{R} \approx \mathbf{I} + [\mathbf{w}]_\times$.

$$\begin{aligned}\left.\frac{\partial \tilde{\mathbf{X}}'}{\partial \mathbf{v}}\right|_{\mathbf{q} \approx \mathbf{q}_1} &= 2(\mathbf{v}^\intercal \tilde{\mathbf{X}} \mathbf{I} + \mathbf{v}\tilde{\mathbf{X}}^\intercal - \tilde{\mathbf{X}}\mathbf{v}^\intercal - w[\tilde{\mathbf{X}}]_\times) \\ &= -2[\tilde{\mathbf{X}}]_\times\end{aligned} \tag{186}$$

Therefore, the final Jacobian with respect to the quaternion $\frac{\partial \tilde{\mathbf{X}}'}{\partial \mathbf{q}}$ is as follows.

$$\boxed{\frac{\partial \tilde{\mathbf{X}}'}{\partial \mathbf{q}} = -2[\tilde{\mathbf{X}}]_\times = -2 \begin{bmatrix} 0 & -Z & Y \\ Z & 0 & -X \\ -Y & X & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}} \tag{187}$$

8.1.1 Code Implementations

- ProSLAM code: trajectory_analyzer.cpp#L253
  - Based on the blog post by jinyongjeong.

8.2 Jacobian of camera intrinsics

NOMENCLATURE of jacobian of camera intrinsics

- $\pi^{-1}(\cdot) = Z\mathbf{K}^{-1}(\cdot)$: Function that back-projects a point in the image plane to three-dimensional space
- $\pi(\cdot) = \pi_k(\pi_h(\cdot)) = \mathbf{K}(\frac{1}{Z}\cdot)$: Function that projects a point from three-dimensional space onto the image plane
- $\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$: Camera intrinsic parameters
- $\mathbf{K}^{-1} = \begin{bmatrix} f_x^{-1} & 0 & -f_x^{-1}c_x \\ 0 & f_y^{-1} & -f_y^{-1}c_y \\ 0 & 0 & 1 \end{bmatrix}$
- $\tilde{\mathbf{K}} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{bmatrix}$: Omits the last row of the intrinsic parameters for projection from $\mathbb{P}^2 \rightarrow \mathbb{R}^2$.
- $\mathbf{X} = [\tilde{\mathbf{X}}, 1]^\intercal$

Performing camera calibration for SLAM allows obtaining intrinsic parameters (intrinsic matrix) $\mathbf{c} = [f_x, f_y, c_x, c_y]$ and lens distortion parameters $\mathbf{d} = [k_1, k_2, p_1, p_2]$. However, since the calibration values do not exactly match the actual sensor parameters, they can be fine-tuned through optimization. This section describes the process of deriving the Jacobian $\mathbf{J_c}$ for $\mathbf{c}$. It is assumed that the focal lengths $f_x \neq f_y$.

For example, consider deriving the Jacobian $\mathbf{J_c}$ for the photometric error (68). It can be expressed as follows:

$$\begin{aligned}\mathbf{J_c} &= \frac{\partial \mathbf{e}}{\partial \mathbf{c}} \\ &= \frac{\partial \mathbf{I}}{\partial \mathbf{p}_2} \frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2} \frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'} \frac{\partial \mathbf{X}'}{\partial \mathbf{c}} \\ &= \mathbb{R}^{1\times 2} \cdot \mathbb{R}^{2\times 3} \cdot \mathbb{R}^{3\times 4} \cdot \mathbb{R}^{4\times 4} = \mathbb{R}^{1\times 4}\end{aligned} \tag{188}$$

The first term $\frac{\partial \mathbf{I}}{\partial \mathbf{p}_2}$ is the Jacobian required for calculating the photometric error, and the remaining three Jacobians are always required irrespective of the reprojection or photometric error terms. **Thus, deriving $\frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2}\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'}\frac{\partial \mathbf{X}'}{\partial \mathbf{c}}$ can be universally applied to the error terms used in SLAM.**

The relationship between points $\mathbf{p}_1, \mathbf{p}_2$ on the image planes of cameras $\{C_1\}, \{C_2\}$ can be expressed as follows:

$$\begin{aligned} \mathbf{p}_1 &= \begin{bmatrix} u_1 & v_1 \end{bmatrix}^\intercal \\ \mathbf{p}_2 &= \begin{bmatrix} u_2 & v_2 \end{bmatrix}^\intercal \end{aligned} \tag{189}$$

$$\begin{aligned} \mathbf{p}_2 &= \pi(\mathbf{X}') \\ &= \pi(\mathbf{R}\mathbf{X} + \mathbf{t}) \\ &= \pi(\mathbf{R}\pi^{-1}(\mathbf{p}_1) + \mathbf{t}) \quad \cdots \text{ apply back-projection} \\ &= \pi(\mathbf{R}(Z\mathbf{K}^{-1}\mathbf{p}_1) + \mathbf{t}) \\ &= \pi_k(\pi_h(\mathbf{R}(Z\mathbf{K}^{-1}\mathbf{p}_1) + \mathbf{t})) \\ &= \pi_k(\frac{Z}{Z'}\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_1 + \frac{1}{Z'}\mathbf{t}) \quad \cdots \text{ apply } \pi_h(\cdot) \\ &= \frac{Z}{Z'}\tilde{\mathbf{K}}\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_1 + \frac{1}{Z'}\tilde{\mathbf{K}}\mathbf{t} \quad \cdots \text{ apply } \pi_k(\cdot) \end{aligned} \tag{190}$$

Back projection of $\mathbf{p}_1$ followed by transformation matrix application leads to $\mathbf{p}_2$ due to a series of projections. As can be seen above, $\frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2}\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'}\frac{\partial \mathbf{X}'}{\partial \mathbf{c}}$ includes parameters from $\mathbf{p}_2$ to $\mathbf{c}$. Therefore, these three Jacobians must be combined to compute $\frac{\partial \mathbf{p}_2}{\partial \mathbf{c}}$ at once:

$$\begin{aligned} \frac{\partial \mathbf{p}_2}{\partial \mathbf{c}} &= \frac{\partial}{\partial \mathbf{c}}\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} \\ &= \frac{\partial}{\partial [f_x, f_y, c_x, c_y]}\begin{bmatrix} f_x\tilde{u}_2 + c_x \\ f_y\tilde{v}_2 + c_y \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial u_2}{\partial f_x} & \frac{\partial u_2}{\partial f_y} & \frac{\partial u_2}{\partial c_x} & \frac{\partial u_2}{\partial c_y} \\ \frac{\partial v_2}{\partial f_x} & \frac{\partial v_2}{\partial f_y} & \frac{\partial v_2}{\partial c_x} & \frac{\partial v_2}{\partial c_y} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{u}_2 + f_x\frac{\partial \tilde{u}_2}{\partial f_x} & f_x\frac{\partial \tilde{u}_2}{\partial f_y} & f_x\frac{\partial \tilde{u}_2}{\partial c_x} + 1 & f_x\frac{\partial \tilde{u}_2}{\partial c_y} \\ f_y\frac{\partial \tilde{v}_2}{\partial f_x} & \tilde{v}_2 + f_y\frac{\partial \tilde{v}_2}{\partial f_y} & f_y\frac{\partial \tilde{v}_2}{\partial c_x} & f_y\frac{\partial \tilde{v}_2}{\partial c_y} + 1 \end{bmatrix} \in \mathbb{R}^{2\times 4} \end{aligned} \tag{191}$$

The elements of the above equation should be calculated next:

$$\begin{pmatrix} \frac{\partial \tilde{u}_2}{\partial f_x} & \frac{\partial \tilde{u}_2}{\partial f_y} & \frac{\partial \tilde{u}_2}{\partial c_x} & \frac{\partial \tilde{u}_2}{\partial c_y} \\ \frac{\partial \tilde{v}_2}{\partial f_x} & \frac{\partial \tilde{v}_2}{\partial f_y} & \frac{\partial \tilde{v}_2}{\partial c_x} & \frac{\partial \tilde{v}_2}{\partial c_y} \end{pmatrix} \tag{192}$$

To derive this, first compute $\tilde{\mathbf{p}}_2 = [\tilde{u}_2, \tilde{v}_2, 1]^\intercal$ as follows:

$$\begin{aligned} \tilde{\mathbf{p}}_2 &= [\tilde{u}_2, \tilde{v}_2, 1]^\intercal \\ &= \frac{1}{Z'}\tilde{\mathbf{X}}' \\ &= \frac{1}{Z'}(\mathbf{R}\tilde{\mathbf{X}} + \mathbf{t}) \\ &= \frac{Z}{Z'}\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_1 + \frac{1}{Z'}\mathbf{t} \\ &= \frac{Z}{Z'}\begin{bmatrix} & \mathbf{R} & \end{bmatrix}\begin{bmatrix} f_x^{-1} & & -f_x^{-1}c_x \\ & f_y^{-1} & -f_y^{-1}c_y \\ & & 1 \end{bmatrix}\begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} + \frac{1}{Z'}\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \\ &= \frac{Z}{Z'}\begin{bmatrix} & \mathbf{R} & \end{bmatrix}\begin{bmatrix} f_x^{-1}(u_1 - c_x) \\ f_y^{-1}(v_1 - c_y) \\ 1 \end{bmatrix} + \frac{1}{Z'}\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \\ &= \frac{Z}{Z'}\begin{bmatrix} r_{11}f_x^{-1}(u_1 - c_x) + r_{12}f_y^{-1}(v_1 - c_y) + r_{13} \\ r_{21}f_x^{-1}(u_1 - c_x) + r_{22}f_y^{-1}(v_1 - c_y) + r_{23} \\ r_{31}f_x^{-1}(u_1 - c_x) + r_{32}f_y^{-1}(v_1 - c_y) + r_{33} \end{bmatrix} + \frac{1}{Z'}\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \end{aligned} \tag{193}$$

This equation can be organized as follows:

$$\begin{bmatrix}\tilde{u}_2\\ \tilde{v}_2\\ 1\end{bmatrix} = \begin{bmatrix}\frac{r_{11}f_x^{-1}(u_1-c_x)+r_{12}f_y^{-1}(v_1-c_y)+r_{13}+\frac{1}{Z}t_x}{r_{31}f_x^{-1}(u_1-c_x)+r_{32}f_y^{-1}(v_1-c_y)+r_{33}+\frac{1}{Z}t_z}\\ \frac{r_{21}f_x^{-1}(u_1-c_x)+r_{22}f_y^{-1}(v_1-c_y)+r_{23}+\frac{1}{Z}t_y}{r_{31}f_x^{-1}(u_1-c_x)+r_{32}f_y^{-1}(v_1-c_y)+r_{33}+\frac{1}{Z}t_z}\\ 1\end{bmatrix} \tag{194}$$

Based on this, (192) can be derived as follows:

$$\begin{aligned}
\frac{\partial\tilde{u}_2}{\partial f_x} &= \frac{Z}{Z'}(r_{31}\tilde{u}_2 - r_{11})f_x^{-2}(u_1-c_x)\\
\frac{\partial\tilde{u}_2}{\partial f_y} &= \frac{Z}{Z'}(r_{32}\tilde{u}_2 - r_{12})f_y^{-2}(v_1-c_y)\\
\frac{\partial\tilde{u}_2}{\partial c_x} &= \frac{Z}{Z'}(r_{31}\tilde{u}_2 - r_{11})f_x^{-1}\\
\frac{\partial\tilde{u}_2}{\partial c_y} &= \frac{Z}{Z'}(r_{32}\tilde{u}_2 - r_{12})f_y^{-1}\\
\frac{\partial\tilde{v}_2}{\partial f_x} &= \frac{Z}{Z'}(r_{31}\tilde{v}_2 - r_{21})f_x^{-2}(u_1-c_x)\\
\frac{\partial\tilde{v}_2}{\partial f_y} &= \frac{Z}{Z'}(r_{32}\tilde{v}_2 - r_{22})f_y^{-2}(v_1-c_y)\\
\frac{\partial\tilde{v}_2}{\partial c_x} &= \frac{Z}{Z'}(r_{31}\tilde{v}_2 - r_{21})f_x^{-1}\\
\frac{\partial\tilde{u}_2}{\partial c_y} &= \frac{Z}{Z'}(r_{32}\tilde{v}_2 - r_{22})f_y^{-1}
\end{aligned} \tag{195}$$

Finally, (191) appears as follows:

$$\begin{aligned}
\frac{\partial\mathbf{p}_2}{\partial\mathbf{c}} &= \begin{bmatrix}\frac{\partial u_2}{\partial f_x} & \frac{\partial u_2}{\partial f_y} & \frac{\partial u_2}{\partial c_x} & \frac{\partial u_2}{\partial c_y}\\ \frac{\partial v_2}{\partial f_x} & \frac{\partial v_2}{\partial f_y} & \frac{\partial v_2}{\partial c_x} & \frac{\partial v_2}{\partial c_y}\end{bmatrix}\\
&= \begin{bmatrix}\tilde{u}_2 + f_x\frac{\partial\tilde{u}_2}{\partial f_x} & f_x\frac{\partial\tilde{u}_2}{\partial f_y} & f_x\frac{\partial\tilde{u}_2}{\partial c_x}+1 & f_x\frac{\partial\tilde{u}_2}{\partial c_y}\\ f_y\frac{\partial\tilde{v}_2}{\partial f_x} & \tilde{v}_2 + f_y\frac{\partial\tilde{v}_2}{\partial f_y} & f_y\frac{\partial\tilde{v}_2}{\partial c_x} & f_y\frac{\partial\tilde{v}_2}{\partial c_y}+1\end{bmatrix}\\
&= \begin{bmatrix}\tilde{u}_2 + \frac{Z}{Z'}f_x^{-1}(r_{31}\tilde{u}_2 - r_{11})(u_1-c_x) & \frac{Z}{Z'}f_xf_y^{-2}(r_{32}\tilde{u}_2 - r_{12})(v_1-c_y) & \frac{Z}{Z'}(r_{31}\tilde{u}_2 - r_{11})+1 & \frac{Z}{Z'}f_xf_y^{-1}(r_{32}\tilde{u}_2 - r_{12}\\ \frac{Z}{Z'}f_x^{-2}f_y(r_{31}\tilde{v}_2 - r_{21})(u_1-c_x) & \tilde{v}_2 + \frac{Z}{Z'}f_y^{-1}(r_{32}\tilde{v}_2 - r_{22})(v_1-c_y) & \frac{Z}{Z'}f_x^{-1}f_y(r_{31}\tilde{v}_2 - r_{21}) & \frac{Z}{Z'}(r_{32}\tilde{u}_2 - r_{12})+1\end{bmatrix}
\end{aligned} \tag{196}$$

8.2.1 Code Implementations

- DSO code: Residuals.cpp#L123
  - For detailed explanation of the code, refer to [SLAM] Direct Sparse Odometry (DSO) Paper and Code Review (2).

8.3 Jacobian of inverse depth

NOMENCLATURE of Jacobian of inverse depth

- $\mathbf{X} = [X, Y, Z, 1]^\intercal = [\tilde{\mathbf{X}}, 1]^\intercal \in \mathbb{P}^3$
- $\tilde{\mathbf{X}} = [X, Y, Z]^\intercal \in \mathbb{P}^2$
- $\rho = \frac{1}{Z}, \rho^{-1} = Z$

8.3.1 Inverse depth parameterization

In SLAM, inverse depth parameterization refers to representing a 3D point $\mathbf{X}$ not with three parameters $[X, Y, Z, 1]$ but with a single parameter (the reciprocal of Z, ρ). This allows a 3D point $\mathbf{X}$ to be fully represented using only the inverse depth ρ, given the pixel location $\mathbf{p} = [u, v]$ on the image plane. This offers computational advantages as only one parameter needs to be estimated during optimization.

8.3.2 Jacobian of inverse depth

Let's assume calculating the Jacobian $\mathbf{J}_\rho$ for photometric error. It can be expressed as follows:

$$\begin{aligned}\mathbf{J}_\rho &= \frac{\partial \mathbf{e}}{\partial \rho} \\ &= \frac{\partial \mathbf{I}}{\partial \mathbf{p}_2}\frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2}\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'}\frac{\partial \mathbf{X}'}{\partial \rho} \\ &= \mathbb{R}^{1\times 2}\cdot\mathbb{R}^{2\times 3}\cdot\mathbb{R}^{3\times 4}\cdot\mathbb{R}^{4\times 1} = \mathbb{R}^{1\times 1}\end{aligned} \tag{197}$$

Here, the term $\frac{\partial \mathbf{I}}{\partial \mathbf{p}_2}$ is the Jacobian needed for computing the photometric error, and the remaining three Jacobians are always required regardless of the reprojection or photometric error terms. **Therefore, computing $\frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2}\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'}\frac{\partial \mathbf{X}'}{\partial \rho}$ can be universally applied to the error terms used in SLAM.**

First, let's express $\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'}$ in terms of inverse depth, equivalent to substituting $\rho' = \frac{1}{Z'}$:

$$\boxed{\begin{aligned}\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'} &= \frac{\partial [\tilde{u}_2, \tilde{v}_2, 1]}{\partial \mathbf{X}'} \\ &= \begin{bmatrix}\rho' & 0 & -\rho'^2 X' & 0 \\ 0 & \rho' & -\rho'^2 Y' & 0 \\ 0 & 0 & 0 & 0\end{bmatrix} \in \mathbb{R}^{3\times 4}\end{aligned}} \tag{198}$$

Next, calculate $\frac{\partial \mathbf{X}'}{\partial \rho}$. The expression for $\mathbf{X}'$ can be decomposed as follows:

$$\begin{aligned}\mathbf{X}' = \begin{bmatrix}\tilde{\mathbf{X}}' \\ 1\end{bmatrix} &= \begin{bmatrix}\mathbf{R}\tilde{\mathbf{X}} + \mathbf{t} \\ 1\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{R}\big(Z\mathbf{K}^{-1}\tilde{\mathbf{X}}\big) + \mathbf{t} \\ 1\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{R}\big(\frac{\mathbf{K}^{-1}\tilde{\mathbf{X}}}{\rho}\big) + \mathbf{t} \\ 1\end{bmatrix}\end{aligned} \tag{199}$$

Using the above, derive $\frac{\partial \mathbf{X}'}{\partial \rho}$ as follows:

$$\boxed{\begin{aligned}\frac{\partial \mathbf{X}'}{\partial \rho} &= \begin{bmatrix}-\mathbf{R}\big(\frac{\mathbf{K}^{-1}\tilde{\mathbf{X}}}{\rho^2}\big) \\ 0\end{bmatrix} \\ &= \begin{bmatrix}-\frac{\tilde{\mathbf{X}}' - \mathbf{t}}{\rho} \\ 0\end{bmatrix} \\ &= -\rho^{-1}\begin{bmatrix}X' - t_x \\ Y' - t_y \\ Z' - t_z \\ 0\end{bmatrix} \in \mathbb{R}^{4\times 1}\end{aligned}} \tag{200}$$

Using these two Jacobians, finally compute $\frac{\partial \mathbf{p}_2}{\partial \rho}$ as follows:

$$\boxed{\begin{aligned}\frac{\partial \mathbf{p}_2}{\partial \rho} &= \frac{\partial \mathbf{p}_2}{\partial \tilde{\mathbf{p}}_2}\frac{\partial \tilde{\mathbf{p}}_2}{\partial \mathbf{X}'}\frac{\partial \mathbf{X}'}{\partial \rho} \\ &= \begin{bmatrix}f_x & 0 & c_x \\ 0 & f_y & c_y\end{bmatrix}\begin{bmatrix}\rho' & 0 & -\rho'^2 X' & 0 \\ 0 & \rho' & -\rho'^2 Y' & 0 \\ 0 & 0 & 0 & 0\end{bmatrix}\cdot -\rho^{-1}\begin{bmatrix}X' - t_x \\ Y' - t_y \\ Z' - t_z \\ 0\end{bmatrix} \\ &= -\rho^{-1}\rho'\begin{bmatrix}f_x(\tilde{u}_2 t_z - t_x) \\ f_y(\tilde{v}_2 t_z - t_y)\end{bmatrix} \in \mathbb{R}^{2\times 1}\end{aligned}} \tag{201}$$

- $\tilde{u}_2 = \frac{X'}{Z'} = \rho' X'$
- $\tilde{v}_2 = \frac{Y'}{Z'} = \rho' Y'$

8.3.3 Code Implementations

- DSO code: CoarseInitializer.cpp#L424
  - For a detailed explanation of the code, refer to [SLAM] Direct Sparse Odometry (DSO) Paper and Code Review (2).

9 References

[1] [Blog] [SLAM] Bundle Adjustment Concept Review: Reprojection error

[2] [Blog] [SLAM] Optical Flow and Direct Method Concept and Code Review: Photometric error

[3] [Blog] [SLAM] Pose Graph Optimization Concept Explanation and Example Code Analysis: Relative pose error

[4] [Blog] Plücker Coordinate Concept Summary: Line projection error

[5] [Blog] [SLAM] Formula Derivation and Analysis of the VINS-mono Content Summary: IMU measurement error

10 Revision log

- 1st: 2023-01-21
- 2nd: 2023-01-22
- 3rd: 2023-01-25
- 4th: 2023-01-28
- 5th: 2023-09-26
- 6th: 2023-11-14
- 7th: 2024-02-06
- 8th: 2024-04-02
- 9th: 2024-05-01