Tightly-coupled Visual-Inertial Sensor Fusion based on IMU Pre-Integration

Zhenfei Yang and Shaojie Shen

December 27, 2016

Some Appendix

1 Quaternion

1.1 Definition

$$\mathbf{q} \triangleq \begin{bmatrix} q_w \\ \vec{\mathbf{q}} \end{bmatrix}$$

$$\bar{\mathbf{x}} \triangleq \begin{bmatrix} \mathbf{0} \\ \vec{\mathbf{x}} \end{bmatrix}$$

$$\mathbf{q}^* \triangleq \begin{bmatrix} q_w \\ -\vec{\mathbf{q}} \end{bmatrix}$$

$$[\mathbf{q}]_{xyz} \triangleq \vec{\mathbf{q}}$$

1.2 Composition through *linear* Matrix Multiplication

$$\mathbf{q} \otimes \mathbf{p} = \mathbf{L}(\mathbf{q})\mathbf{p} = \begin{bmatrix} q_w & -\vec{\mathbf{q}}^\top \\ \vec{\mathbf{q}} & q_w\mathbf{I} + \hat{\vec{\mathbf{q}}} \end{bmatrix} \mathbf{p}$$

$$\mathbf{L}(\mathbf{q}) = \mathbf{L}^\intercal(\mathbf{q}^*)$$

$$\mathbf{q} \otimes \mathbf{p} = \mathbf{R}(\mathbf{p})\mathbf{q} = \begin{bmatrix} p_w & -\vec{\mathbf{p}}^\intercal \\ \vec{\mathbf{p}} & p_w\mathbf{I} + \hat{\vec{\mathbf{p}}} \end{bmatrix} \mathbf{q}$$

$$\mathbf{R}(\mathbf{p}) = \mathbf{R}^\intercal(\mathbf{p}^*)$$

1.3 Commutative Law

Because

$$\mathbf{L}(\mathbf{q})\mathbf{R}(\mathbf{p})\mathbf{r} = \mathbf{q} \otimes \mathbf{r} \otimes \mathbf{p} = \mathbf{R}(\mathbf{p})\mathbf{L}(\mathbf{q})\mathbf{r},$$

we have

$$\mathbf{L}(\mathbf{q})\mathbf{R}(\mathbf{p}) = \mathbf{R}(\mathbf{p})\mathbf{L}(\mathbf{q}).$$

1.4 Rotate a Vector

$$\begin{aligned}
\mathbf{q} \times \mathbf{x} &\triangleq \mathcal{R}(\mathbf{q})\mathbf{x} \\
&= \mathbf{q} \otimes \bar{\mathbf{x}} \otimes \mathbf{q}^* \\
&= \mathbf{x} + 2\overrightarrow{\mathbf{q}} \times (\overrightarrow{\mathbf{q}} \times \mathbf{x} + q_w \mathbf{x}) \\
&= \mathbf{x} + 2\left[\overrightarrow{\mathbf{q}}\right]_\times^2 \mathbf{x} + 2q_w \left[\overrightarrow{\mathbf{q}}\right]_\times \mathbf{x} \\
&= \mathbf{x} + 2(\overrightarrow{\mathbf{q}}\overrightarrow{\mathbf{q}}^\intercal - \mathbf{I})\mathbf{x} + 2q_w \left[\overrightarrow{\mathbf{q}}\right]_\times \mathbf{x}
\end{aligned}$$

1.5 Conversion to Rotation Matrix

$$\begin{aligned}
\mathbf{q} \otimes \bar{\mathbf{x}} \otimes \mathbf{q}^* &= \mathbf{L}(\mathbf{q})\mathbf{R}(\mathbf{q}^*)\bar{\mathbf{x}} \\
\begin{bmatrix} \mathbf{0} \\ \mathcal{R}\overrightarrow{\mathbf{x}} \end{bmatrix} &= \mathbf{L}(\mathbf{q})\mathbf{R}(\mathbf{q}^*) \begin{bmatrix} \mathbf{0} \\ \overrightarrow{\mathbf{x}} \end{bmatrix} \\
\mathcal{R}(\mathbf{q}) &= \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{L}(\mathbf{q})\mathbf{R}(\mathbf{q}^*) \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}
\end{aligned}$$

1.6 Covariance Propagation of Process Noise

1.6.1 Discrete-time System

$$\begin{aligned}
x_k &= x_{k-1} + w_{k-1} \\
w_k &\simeq (0, Q) \\
x_0 &= 0
\end{aligned}$$

The discrete-time system can be solved as follows:

$$x_k = w_0 + w_1 + \cdots + w_{k-1}$$

, and the covariance of the state is therefore given as:

$$\begin{aligned}
E[x_k x_k^\intercal] &= E[(w_0 + w_1 + \cdots + w_{k-1})(w_0 + w_1 + \cdots + w_{k-1})^\intercal] \\
&= E[x_0 x_0^\intercal] + E[x_1 x_1^\intercal] + \cdots + E[x_{k-1} x_{k-1}^\intercal] \\
&= kQ
\end{aligned}$$

1.6.2 Continuous-time System

$$\begin{aligned}
\dot{x}(t) &= w(t) \\
w(t) &\simeq (0, Q) \\
x(0) &= 0
\end{aligned}$$

We **propose** the following definition for continuous-time white noise:

$$E[w(t)w(\tau)^\intercal] = \frac{Q}{T}\delta(t - \tau)$$

. Then we are able to compute the covariance of $x(t)$:

$$\begin{aligned} E[x(t)x^{\intercal}(t)] &= E[\int_0^t w(\alpha)d\alpha \int_0^t w^{\intercal}(\beta)d\beta] \\ &= \int_0^t \int_0^t E[w(\alpha)w^{\intercal}(\beta)]d\alpha d\beta \\ &= \int_0^t \int_0^t \frac{Q}{T}\delta(t-\tau)d\alpha d\beta \\ &= \int_0^t \frac{Q}{T}d\beta \\ &= \frac{Qt}{T} \end{aligned}$$

Recalling that $t = kT$, we can write the above equation as:

$$E[x(t)x^{\intercal}(t)] = kQ$$

.

The above equation implies continuous-time system with white noise

$$w(t) \simeq (0, Q_c), Q_c = \frac{Q}{T}$$

is **equivalent** to discrete-time system with noise

$$w_k \simeq (0, Q)$$

.

Finally, we are able to predict the covariance of $x(t)$ in the same way to that we used in discrete-time system:

$$E[x(t)x^{\intercal}(t)] = kQ = kQ_cT = tQ_c$$

, which is regardless of sample time T.

2 Formulation

2.1 IMU Model

$$\begin{aligned} \mathbf{acc}_t &= \mathbf{acc} + \mathbf{na} + \mathbf{ba} \\ \mathbf{gyr}_t &= \mathbf{gyr} + \mathbf{ng} + \mathbf{bg} \end{aligned}$$

A Gaussian white noise $\mathbf{n}$ with standard derivation σ is defined as:

$$\begin{aligned} \mathbf{E}[\mathbf{n}] &= \mathbf{0}_{3\times1} \\ \mathbf{E}[\mathbf{n}(t+\tau)\mathbf{n}^{\intercal}(t)] &= \delta(\tau)\sigma^2\mathbf{I}_{3\times3} \end{aligned}$$

We assume $\mathbf{na}, \mathbf{ng}, \dot{\mathbf{ba}}, \dot{\mathbf{bg}}$ are Gaussian white noise with standard derivation $\sigma_{na}, \sigma_{ng}, \sigma_{wa}, \sigma_{wg}$.

In discrete implementation, we get:

At time t, sampled $\mathbf{na}_t, \mathbf{ng}_t$ follow Gaussian distribution: $\mathbf{na}_t \sim \mathcal{N}(\mathbf{0}, \sigma_{na}^2\mathbf{I}), \mathbf{ng}_t \sim \mathcal{N}(\mathbf{0}, \sigma_{ng}^2\mathbf{I})$.

During time interval $[t, t+dt]$, Integrated $\Delta\mathbf{ba}_t = \int_t^{t+dt} \dot{\mathbf{ba}}\mathrm{d}t, \Delta\mathbf{bg}_t = \int_t^{t+dt} \dot{\mathbf{bg}}\mathrm{d}t$ follow Gaussian distribution: $\Delta\mathbf{ba}_t \sim \mathcal{N}(\mathbf{0}, (\sigma_{na}\sqrt{dt})^2\mathbf{I}), \Delta\mathbf{bg}_t \sim \mathcal{N}(\mathbf{0}, (\sigma_{ng}\sqrt{dt})^2\mathbf{I})$.

The latter one follows theory in stochastic integration.

2.2 PreIntegration

In this part, the goal is:

1. propagate the *mean*.
2. propagate the *covariance*.
3. propagate the *Jacobian*.

First, we define:

$$\mathbf{x}_n = \begin{bmatrix} \alpha_n \\ \beta_n \\ \mathbf{q}_n \\ \mathbf{ba}_n \\ \mathbf{bg}_n \end{bmatrix}_{16\times 1}, \mathbf{x}_n \sim \mathcal{N}(\mu_n, \Sigma_n)$$

$$\mathbf{x}_{n+1} = f(\mathbf{x}_n, dt, \mathbf{acc}_n, \mathbf{gyr}_n, \mathbf{acc}_{n+1}, \mathbf{gyr}_{n+1})$$

$$\mathbf{J}_n = \frac{\partial \mathbf{x}_n}{\partial \mathbf{x}_0}$$

where $f()$ could be Euler, Midpoint or RK4 integration.
Begin with

$$\mu_0 = \begin{bmatrix} \mathbf{0}_{3\times 1} \\ \mathbf{0}_{3\times 1} \\ \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} \\ \mathbf{0}_{3\times 1} \\ \mathbf{0}_{3\times 1} \end{bmatrix}_{16\times 1}, \Sigma_0 = \mathbf{0}_{16\times 16}, \mathbf{J}_0 = \mathbf{I}_{16\times 16}$$

the distribution is propagated recursively:

$$\mu_{n+1} = f(\mu_n, dt, \mathbf{acc}_n, \mathbf{gyr}_n, \mathbf{acc}_{n+1}, \mathbf{gyr}_{n+1})$$

$$\mathbf{A}_{16\times 16} = \frac{\partial f}{\partial \mathbf{x}}, \mathbf{B}_{16\times 12} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{acc}_n} & \frac{\partial f}{\partial \mathbf{gyr}_n} & \frac{\partial f}{\partial \mathbf{acc}_{n+1}} & \frac{\partial f}{\partial \mathbf{gyr}_{n+1}} \end{bmatrix}, \mathbf{C}_{16\times 6} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

$$\Sigma_{n+1} = \mathbf{A}\Sigma_n\mathbf{A}^\intercal + \mathbf{B} \begin{bmatrix} \sigma_{na}^2\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \sigma_{nw}^2\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sigma_{na}^2\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \sigma_{ng}^2\mathbf{I} \end{bmatrix}_{12\times 12} \mathbf{B}^\intercal + \mathbf{C} \begin{bmatrix} dt\sigma_{wa}^2\mathbf{0} & \mathbf{0} \\ \mathbf{0} & dt\sigma_{wg}^2\mathbf{0} \end{bmatrix}_{6\times 6} \mathbf{C}^\intercal$$

$$\mathbf{J}_{n+1} = \mathbf{A}\mathbf{J}_n$$

Finally, we need to shrink both mean and covariance to its minimal form. After chaining all of the propagations between two keyframes, define

$$\mathbf{x}_{N-1} = g(\mathbf{x}_0, dt, \mathbf{acc}_{0\ldots N-1}, \mathbf{gyr}_{0\ldots N-1})$$

$$\mathbf{L}_{15\times 16} = \frac{\partial \delta\mathbf{x}_{N-1}}{\partial \mathbf{x}_{N-1}} = \frac{\partial(\mathbf{x}_{N-1} \ominus \mu_{N-1})}{\partial \mathbf{x}_{N-1}}$$

$$\mathbf{R}_{16\times 15} = \frac{\partial \mathbf{x}_{N-1}}{\partial \delta\mathbf{x}_0} = \frac{\partial \mathbf{x}_{N-1}}{\partial \mathbf{x}_0}\frac{\partial \mathbf{x}_0}{\partial \delta\mathbf{x}_0} = \mathbf{J}_{N-1}\frac{\partial(\mu_0 \oplus \delta\mathbf{x}_0)}{\partial \delta\mathbf{x}_0},$$

then we have:

$$\mathbf{J}_{minimal} = \mathbf{L}\mathbf{J}_{N-1}\mathbf{R}$$
$$\Sigma_{minimal} = \mathbf{L}\Sigma_{N-1}\mathbf{L}^{\intercal}$$

2.3 MLE Formulation

$$\mathbf{residual}_{ij} = \begin{bmatrix} \mathbf{Ep}_{ij} \\ \mathbf{Ev}_{ij} \\ \mathbf{Eq}_{ij} \\ \mathbf{Eba}_{ij} \\ \mathbf{Ebg}_{ij} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_i^* \times (\frac{1}{2}\mathbf{g}dt^2 + \mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i dt) - \alpha_{ij} \\ \mathbf{q}_i^* \times (\mathbf{g}dt + \mathbf{v}_j - \mathbf{v}_i) - \beta_{ij} \\ 2\left[\Delta\mathbf{q}_{ij}^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j\right]_{xyz} \\ \mathbf{ba}_j - \mathbf{ba}_i \\ \mathbf{bg}_j - \mathbf{bg}_i \end{bmatrix}$$

The propagated mean and covariance can be used to define the cost function based on MLE.
The propagated Jacobian is required by Gauss-Newton optimization.
So-called PreIntegration method is doable after observing the block elements of the Jacobian.

3 Analytic Jacobian

3.1 With respect to Bias

3.2 With respect to Rotation

3.2.1 $\frac{\partial \mathbf{r}_{ij}}{\partial \theta_j}$

$$\begin{aligned}
\frac{\partial \mathbf{r}_{ij}}{\partial \theta_j} &= \frac{\partial 2\left[\Delta\mathbf{q}_{ij}^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\theta_j \end{bmatrix}\right]_{xyz}}{\partial \theta_j} \\
&= 2\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix} \frac{\partial(\Delta\mathbf{q}_{ij}^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\theta_j \end{bmatrix})}{\partial \theta_j} \\
&= 2\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix} \frac{\partial(\mathbf{L}(\Delta\mathbf{q}_{ij}^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j) \begin{bmatrix} 1 \\ \frac{1}{2}\theta_j \end{bmatrix})}{\partial \theta_j} \\
&= 2\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix} \frac{\partial(\mathbf{L}(\Delta\mathbf{q}_{ij}^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j) \begin{bmatrix} 1 \\ \frac{1}{2}\theta_j \end{bmatrix})}{\partial \begin{bmatrix} 1 \\ \frac{1}{2}\theta_j \end{bmatrix}} \frac{\partial \begin{bmatrix} 1 \\ \frac{1}{2}\theta_j \end{bmatrix}}{\partial \theta_j} \\
&= 2\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix} (\mathbf{L}(\Delta\mathbf{q}_{ij}^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j)) \begin{bmatrix} 0 \\ \frac{1}{2}\mathbf{I} \end{bmatrix} \\
&\approx \mathbf{I}
\end{aligned}$$

Given

$$\Delta\mathbf{q}_{ij} \approx \mathbf{q}_i^* \otimes \mathbf{q}^j,$$

we have

$$\frac{\partial \mathbf{r}_{ij}}{\partial \theta_j} \approx \mathbf{I}$$

3.3 $\frac{\partial \mathbf{r}_{ij}}{\partial \theta_i}$

$$\begin{aligned}
\frac{\partial \mathbf{r}_{ij}}{\partial \theta_i} &= \frac{\partial 2 \left[\Delta\mathbf{q}_{ij}^* \otimes [\mathbf{q}_i \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}]^* \otimes \mathbf{q}_j \right]_{xyz}}{\partial \theta_i} \\
&= \frac{\partial -2 \left[[\Delta\mathbf{q}_{ij}^* \otimes [\mathbf{q}_i \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}]^* \otimes \mathbf{q}_j]^* \right]_{xyz}}{\partial \theta_i} \\
&= \frac{\partial -2 \left[[\mathbf{q}_j^* \otimes [\mathbf{q}_i \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}] \otimes \Delta\mathbf{q}_{ij} \right]_{xyz}}{\partial \theta_i} \\
&= -2 \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \frac{\partial [\mathbf{q}_j^* \otimes [\mathbf{q}_i \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}] \otimes \Delta\mathbf{q}_{ij}]}{\partial \theta_i} \\
&= -2 \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \frac{\partial [\mathbf{q}_j^* \otimes \mathbf{q}_i \otimes [\begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix} \otimes \Delta\mathbf{q}_{ij}]]}{\partial \theta_i} \\
&= -2 \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \frac{\partial [\mathbf{q}_j^* \otimes \mathbf{q}_i \otimes [\mathbf{R}(\Delta\mathbf{q}_{ij}) \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}]]}{\partial \theta_i} \\
&= -2 \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \frac{\partial [\mathbf{L}(\mathbf{q}_j^* \otimes \mathbf{q}_i)\mathbf{R}(\Delta\mathbf{q}_{ij}) \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}]}{\partial \theta_i} \\
&= -2 \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \frac{\partial [\mathbf{L}(\mathbf{q}_j^* \otimes \mathbf{q}_i)\mathbf{R}(\Delta\mathbf{q}_{ij}) \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}]}{\partial \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}} \frac{\partial \begin{bmatrix} 1 \\ \frac{1}{2}\theta_i \end{bmatrix}}{\partial \theta_i} \\
&= -2 \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} [\mathbf{L}(\mathbf{q}_j^* \otimes \mathbf{q}_i)\mathbf{R}(\Delta\mathbf{q}_{ij})] \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{I} \end{bmatrix}
\end{aligned}$$

Given

$$\Delta\mathbf{q}_{ij} \approx \mathbf{q}_i^* \otimes \mathbf{q}^j,$$

we have

$$\frac{\partial \mathbf{r}_{ij}}{\partial \theta_i} \approx -\mathcal{R}(\Delta\mathbf{q}_{ij}^*)$$

3.4 $\frac{\partial \mathbf{r}_{ij}}{\partial \theta_{ij}}$

$$
\begin{aligned}
\frac{\partial \mathbf{r}_{ij}}{\partial \theta_i} &= \frac{\partial 2\left[[\Delta \mathbf{q}_{ij} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{J}\Delta\theta_{ij} \end{bmatrix}]^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j\right]_{xyz}}{\partial \Delta\theta_{ij}} \\
&= \frac{\partial -2\left[[[\Delta \mathbf{q}_{ij} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{J}\Delta\theta_{ij} \end{bmatrix}]^* \otimes \mathbf{q}_i^* \otimes \mathbf{q}_j]^*\right]_{xyz}}{\partial \Delta\theta_{ij}} \\
&= \frac{\partial -2\left[\mathbf{q}_j^* \otimes \mathbf{q}_i \otimes [\Delta \mathbf{q}_{ij} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{J}\Delta\theta_{ij} \end{bmatrix}]\right]_{xyz}}{\partial \Delta\theta_{ij}} \\
&= -2\begin{bmatrix} 0 & \mathbf{I} \end{bmatrix} \frac{\partial [\mathbf{q}_j^* \otimes \mathbf{q}_i \otimes [\Delta \mathbf{q}_{ij} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{J}\Delta\theta_{ij} \end{bmatrix}]]}{\partial \Delta\theta_{ij}} \\
&= -2\begin{bmatrix} 0 & \mathbf{I} \end{bmatrix} \mathbf{L}(\mathbf{q}_j^* \otimes \mathbf{q}_i \otimes \Delta \mathbf{q}_{ij}) \begin{bmatrix} 0 \\ \frac{1}{2}\mathbf{J} \end{bmatrix}
\end{aligned}
$$

Given

$$\Delta \mathbf{q}_{ij} \approx \mathbf{q}_i^* \otimes \mathbf{q}^j,$$

we have

$$\frac{\partial \mathbf{r}_{ij}}{\partial \theta_{ij}} \approx \mathbf{J}$$

3.5 With respect to Velocity

3.6 With respect to Position

4 How to compare two approaches?

In the following two sections, we analysis the complexity of the optimization-based approach, and compare it with the filter-based approach [1].

4.1 Equivalent Computation

The analysis is under the equivalent-computation assumption: the two approaches share the exactly same computation procedure, which is one-step linearizion and minimization, but using different/dual covariance representation. One-step linearizion and minimization is widely acceptable for filter methods but not for optimization methods. People may feel confused about the one-step gauss-newton iteration in the optimization method. However, the mathematical equivalence between these two methods explains it is reasonable.

4.1.1 Equivalent Problem Size

We define the length of the sliding window is N, the number of feature involved in this optimization/update is M, and the average feature tracking length is L. For comparison, we use xyz parameterization for feature position in both approaches. We ignore the complexity of jacobian computation and state propagation as they are unavoidable for both methods.

4.1.2 Equivalent Marginalization Strategy

Each feature is used by the filter/optimiztion only once after its tracking is lost or longer than window size. After that the feature is marginalized.

4.1.3 Is complexity everything?

The algorithm complexity is not the only factor that has affects on the running speed of the program. We should also care about how cache friendly it is, how to benefit from multiple threads. But it is still a good reference.

5 Complexity Analysis of the Optimization

5.1 Information matrix construction

The jacobian for each projection is $J = \begin{bmatrix} \frac{\partial \begin{bmatrix} u \\ v \end{bmatrix}}{\partial \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix}} & \frac{\partial \begin{bmatrix} u \\ v \end{bmatrix}}{\partial \mathbf{f}} \end{bmatrix}_{2\times 9}$. Computing $J^\intercal J$ costs $9 \times 2 \times 9$. The complexity of adding all information submatrix together is $O(9 \times 2 \times 9ML)$.

5.2 Solving linear equation

The efficiency in this step relies on the sparsity of the information matrix:

$$\begin{bmatrix} B & E \\ E^\top & C \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix}, \tag{1}$$

where Δy is the error state of poses and Δz is the error state of feature.

After observing

$$\Delta z = C^{-1}(w - E^\intercal \Delta y),$$

we eliminate Δz first to form:

$$\left[B - EC^{-1}E^\top\right] \Delta y = v - EC^{-1}w. \tag{2}$$

The best news here is the inverse of the diagonal matrix C can be obtained in $O(3M)$.

Computing $B - EC^{-1}E^\intercal$ needs $O(6N \times 3M + 6N \times 3M \times 6N)$. It is also worth to mention that the computation can be further accelerated using sparse matrix library since E is also sparse. Solving the normal equation using cholesky decomposition is proved to be **far** faster than matrix multiplication in modern CPU[1] (x86 or ARM) although its complexity is the same: $O((6N)^3)$.

It is worth to note that during inversing of C we can easily handle the numerical issue caused by small parallax feature projections (set an epsilon). This is a good approximation that is similar to the damped Gauss-Newton step.

5.3 Marginalization

Compared with the filter method, marginalization in the optimization method is more complex. We follow two steps to do the marginalization. Note that the difference between the two-step marginalization and the one-step one is essentially because of different elimination order, and thus they are equivalent.

5.3.1 Marginalize all features

(2) is exactly the system after feature marginalization.

[1]http://eigen.tuxfamily.org/dox/group__DenseDecompositionBenchmark.html

5.3.2 Marginalize the oldest pose

The reduced system (2) is block-wise:

$$\begin{bmatrix} A_{mm} & A_{mr} \\ A_{rm} & A_{rr} \end{bmatrix} x = \begin{bmatrix} b_{mm} \\ b_{rr} \end{bmatrix}, \tag{3}$$

. Schur Complement gives:

$$\begin{aligned} A &= A_{rr} - A_{rm} * A_{mm}^{-1} * A_{mr} \\ b &= b_{rr} - A_{rm} * A_{mm}^{-1} * b_{mm}, \end{aligned} \tag{4}$$

whose complexity is $O(6N \times 15 \times 6N + 15 \times 15 \times 15)$

6 Complexity Analysis of the Filter

6.1 Structureless Visual Constraints

For each feature f, its L projections connect L poses:

$$r = H_x x + H_f f \tag{5}$$

where H_x is $2L \times 6L$ and H_f is $2L \times 3$. We use given rotations to triangulate H_f to be upper triangular matrix R:

$$r_3 r_2 r_1 H_f = R \tag{6}$$

. The null space of H_f can be obtained from the right $2L - 3$ columns of $(r_3 r_2 r_1)^\top$. Then projecting/rotating H_x and H_f into the null space can be done without explicitly evaluating $r_3 r_2 r_1$ because

$$Null(H_f) = (r_3 r_2 r_1)^\top \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{7}$$

and

$$Null(H_f)^\top H_x = \begin{bmatrix} 0 & I \end{bmatrix} (r_3 r_2 r_1) H_x \tag{8}$$

where r_i is 2D rotation in $2L$ dimension space.

The projection cost $O(2L \times 3 \times 6L)$ and the overall complexity is $O(2L \times 3 \times 6L \times M)$

6.2 Reduce Structureless Visual Constraints

Stacking all the M structureless visual constraints gives us:

$$r = Hx$$

where H is $(2L-3)M \times 6N$. Again we use given rotations to triangulate H to reduce the size of H. The triangulation costs $O((2L-3)M \times 6N \times 6N)$. But also the H is sparse as the average tracking length is L, so the real complexity is $O((2L-3)M \times 6L \times 6N)$

6.3 Kalman update

We only talk about the full matrix (whose size if $6N \times 6N$) in this part. There are 6 matrix multiplication and 1 matrix inversion. The overall complexity is $O(6N \times 6N \times 6N \times 7)$.

References

[1] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, Roma, Italy, Apr. 2007, pp. 3565–3572.