IMU Measurement Model with Sensor Offset and Misalignment

We consider an IMU mounted at a known offset $\mathbf{r}$ from the body reference frame origin, with a known fixed rotation $\mathbf{R}_{BS}$ from body frame (B) to sensor frame (S). The IMU provides measurements of linear acceleration and angular velocity. The 19-dimensional state vector is defined as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \mathbf{v} \\ \boldsymbol{\omega} \\ \mathbf{a} \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} p_x, p_y, p_z \\ q_0, q_1, q_2, q_3 \\ v_x, v_y, v_z \\ \omega_x, \omega_y, \omega_z \\ a_x, a_y, a_z \\ \alpha_x, \alpha_y, \alpha_z \end{bmatrix},$$

where $\mathbf{q}$ is the quaternion representing orientation from inertial to body frame.

The IMU provides two measurements:

1. Accelerometer measurement (3×1):

$$\mathbf{a}_{IMU} = \mathbf{R}_{BS} \left[\mathbf{a} + \mathbf{R}(\mathbf{q})^T \mathbf{g} + \boldsymbol{\alpha} \times \mathbf{r} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}) \right]$$

2. Gyroscope measurement (3×1):

$$\boldsymbol{\omega}_{IMU} = \mathbf{R}_{BS}\, \boldsymbol{\omega}$$

Here: - $\mathbf{R}(\mathbf{q})^T$ rotates vectors from inertial to body frame. - $\mathbf{R}_{BS}$ is a constant rotation matrix from body to sensor frame (misalignment). - $\mathbf{g}$ is gravitational acceleration in inertial coordinates. - $\mathbf{r}$ is lever arm vector (body coordinates).

Jacobian Derivation

The Jacobian matrix $\mathbf{H}$ is defined as:

$$\mathbf{H} = \frac{\partial}{\partial \mathbf{x}} \begin{bmatrix} \mathbf{a}_{IMU} \\ \boldsymbol{\omega}_{IMU} \end{bmatrix}_{6 \times 1},$$

thus it has dimensions 6×19.

We compute each block separately:

Accelerometer Measurement Jacobians (3×19)

- With respect to linear position (3×3):

$$\frac{\partial \mathbf{a}_{IMU}}{\partial \mathbf{p}} = \mathbf{0}_{3 \times 3}.$$

- With respect to quaternion orientation (3×4), gravity term only:

$$J_q = \frac{\partial (\mathbf{R}_{BS}\, \mathbf{R}(q)^T g)}{\partial q} = \mathbf{R}_{BS} \frac{\partial (\mathbf{R}(q)^T g)}{\partial q},$$

where:

$$\frac{\partial (\mathbf{R}(q)^T g)}{\partial q} = 2 \left[(q_0 g + \tilde{q} \times g), \quad (\tilde{q}\, g^T + (q_0 I + [\tilde{q}]_\times)[g]_\times) \right]_{3 \times 4},$$

with quaternion $q = [q_0, \tilde{q}^T]^T$, and $[\tilde{q}]_\times$ skew-symmetric operator.

- With respect to linear velocity (3×3):

$$\frac{\partial \mathbf{a}_{IMU}}{\partial \mathbf{v}} = 0_{3 \times 3}.$$

- With respect to angular velocity (3×3): Define: $h(\boldsymbol{\omega}) = \boldsymbol{\omega} \times (\boldsymbol{\omega} \times r)$, then:

$$\frac{\partial h}{\partial \boldsymbol{\omega}} = [(\boldsymbol{\omega} \times r)]_\times - [r]_\times [\boldsymbol{\omega}]_\times.$$

Thus:

$$\frac{\partial \mathbf{a}_{IMU}}{\partial \boldsymbol{\omega}} = \mathbf{R}_{BS} \left([(\boldsymbol{\omega} \times r)]_\times - [r]_\times [\boldsymbol{\omega}]_\times \right).$$

- With respect to linear acceleration (3×3): Direct identity mapping:

$$\frac{\partial \mathbf{a}_{IMU}}{\partial \mathbf{a}} = \mathbf{R_{BS}}.$$

- With respect to angular acceleration (3×3): Tangential acceleration depends linearly:

$$\frac{\partial (\boldsymbol{\alpha} \times r)}{\partial \boldsymbol{\alpha}} = \mathbf{R}_{BS} [r]_\times.$$

Thus accelerometer Jacobian block row is:

$$H_{accel} = [\underbrace{0_{(3\times3)}}_{\mathbf{p}}, \quad J_q^{grav}(q)_{(3\times4)}, \quad 0_{(3\times3)}, \quad \mathbf{R}_{BS} \left([(\boldsymbol{\omega} \times r)]_\times - [r]_\times [\boldsymbol{\omega}]_\times \right), \quad R_{BS}, \quad R_{BS}[r]_\times]$$

Gyroscope Measurement Jacobians (3×19)

Only angular velocity states affect gyroscope directly:

$$H_{gyro} = [\, 0_{(3\times3)}, \quad 0_{(3\times4)}, \quad 0_{(3\times3)}, \quad R_{BS}, \quad 0_{(3\times3)}, \quad 0_{(3\times3)} \,].$$

Thus the full Jacobian matrix (6×19) is clearly given by stacking both blocks:

$$H = \begin{bmatrix} 0 & J_q^{grav} & 0 & R_{BS} \left([(\boldsymbol{\omega} \times r)]_\times - [r]_\times [\boldsymbol{\omega}]_\times \right) & R_{BS} & R_{BS}[r]_\times \\ 0 & 0 & 0 & R_{BS} & 0 & 0 \end{bmatrix}.$$

This final form correctly incorporates both sensor offset and misalignment rotation.

1 IMU Quaternion Jacobian Derivation & Validation

1.1 Problem Definition

For an IMU with orientation represented by unit quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$, we derive the Jacobian for the gravity compensation term:

$$\mathbf{f}(\mathbf{q}) = \mathbf{R_{BS}} \mathbf{R}(\mathbf{q})^{\mathbf{T}} \mathbf{g_W} \tag{1}$$

where $\mathbf{g_W} = [0, 0, g]^T$ is gravity in world coordinates, and $\mathbf{R_{BS}}$ is the fixed body-to-sensor rotation.

1.2 Analytical Derivation

The rotation matrix $\mathbf{R}(\mathbf{q})^{\mathbf{T}}$ for a unit quaternion is:

$$\mathbf{R}(\mathbf{q})^{\mathbf{T}} = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y + q_w q_z) & 2(q_x q_z - q_w q_y) \\ 2(q_x q_y - q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z + q_w q_x) \\ 2(q_x q_z + q_w q_y) & 2(q_y q_z - q_w q_x) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix} \tag{2}$$

The body-frame gravity vector and its Jacobian:

$$\mathbf{R}(\mathbf{q})^{\mathbf{T}} \mathbf{g_W} = g \begin{bmatrix} 2(q_x q_z - q_w q_y) \\ 2(q_y q_z + q_w q_x) \\ 1 - 2(q_x^2 + q_y^2) \end{bmatrix}, \quad J_q^{\text{grav}} = 2g \begin{bmatrix} -q_y & q_z & -q_w & q_x \\ q_x & q_w & q_z & q_y \\ 0 & -2q_x & -2q_y & 0 \end{bmatrix} \tag{3}$$

2 Numerical Validation Methodology

2.1 Core Principles

- Maintain unit quaternion constraint $\|\mathbf{q}\| = 1$
- Use consistent frame conventions (ROS REP 103 recommended)
- Validate identity case before complex orientations

2.2 Perturbation Technique

2.2.1 Exponential Map Formulation

For valid quaternion perturbations:

$$\delta\mathbf{q} = \exp\left(\frac{1}{2}\delta\boldsymbol{\theta}\right) \approx \begin{bmatrix} 1 \\ \delta\theta_x/2 \\ \delta\theta_y/2 \\ \delta\theta_z/2 \end{bmatrix}, \quad \|\delta\boldsymbol{\theta}\| \ll 1 \tag{4}$$

Implementation code:

```
const double eps = 1e-6; // Optimal for double precision
Eigen::Vector3d delta_theta(eps, 0, 0); // X-axis perturbation

Eigen::Quaterniond dq(1, delta_theta.x()/2,
delta_theta.y()/2,
delta_theta.z()/2);
dq.normalize();
Eigen::Quaterniond q_perturbed = q * dq;
```

2.3 Validation Protocol

1. **Baseline Check**: Identity quaternion $\mathbf{q} = [1, 0, 0, 0]^T$
2. **Canonical Rotations**: 90° about each principal axis
3. **Arbitrary Orientation**: Random unit quaternion
4. **Error Analysis**:

$$\varepsilon_{\text{rel}} = \frac{\|J_{\text{ana}} - J_{\text{num}}\|}{\max(\|J_{\text{ana}}\|, \epsilon)} \tag{5}$$

   - Accept: $\varepsilon_{\text{rel}} < 1\%$
   - Investigate: $1\% \leq \varepsilon_{\text{rel}} \leq 5\%$
   - Reject: $\varepsilon_{\text{rel}} > 5\%$

3 Case Study: 90° Y-Rotation

3.1 Test Configuration

- Quaternion: $[0.7071, 0, 0.7071, 0]^T$ (90° Y-rotation)
- Gravity: $g = 9.80665\,\text{m/s}^2$
- Perturbation: $\delta\theta = 1\mu\text{rad}$

3.2 Results & Analysis

Term	Analytical	Numerical	Error
$\partial g_x/\partial q_w$	-13.87	-13.87	0.00%
$\partial g_z/\partial q_y$	-27.74	-27.74	0.00%
$\partial g_y/\partial q_z$	13.87	13.87	0.00%

4 Best Practices & Troubleshooting

4.1 Implementation Checklist

- ☐ Use exponential map perturbations
- ☐ Normalize after quaternion operations
- ☐ Verify $\mathbf{R_{BS}}$ separately
- ☐ Test positive/negative perturbations
- ☐ Check magnitude and sign

4.2 Common Issues & Solutions

Symptom	Resolution
Sign mismatches	Verify quaternion multiplication order (Hamilton vs JPL)
Null Z-derivatives	Confirm gravity vector alignment in world frame
2x error scaling	Check 1/2 factor in exponential map
Discontinuities at identity	Use $\mathbf{q} = [0.999, 0, 0, 0.001]$ near identity
Frame inconsistencies	Validate $\mathbf{R_{BS}}$ matrix independently
Sensitivity to perturbation size	Test $\epsilon \in [10^{-7}, 10^{-4}]$

A Reference Implementation

Complete Jacobian computation:

```
Eigen::Matrix<double, 3, 4> ComputeQuatJacobian(
const Eigen::Quaterniond& q,
double g,
const Eigen::Matrix3d& R_BS)
{
        Eigen::Matrix<double, 3, 4> J;
        const double two_g = 2 * g;

        J << two_g*q.y(),  two_g*q.z(),  two_g*q.w(),  two_g*q.x(),
        -two_g*q.x(), -two_g*q.w(),  two_g*q.z(),  two_g*q.y(),
        0.0, -4*g*q.x(),   -4*g*q.y(),             0.0;

        return R_BS * J;
}
```