Error State Kalman Filter (ESKF) for IMU-GPS Sensor Fusion with Sensor Offset and Delay Estimation

Ehsan KhademOlama

April 3, 2025

Abstract

This report details the formulation and implementation of an Error State Kalman Filter (ESKF) designed to fuse measurements from an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS) receiver. The filter estimates the system's orientation, velocity, position, and the unknown physical offset between the IMU and the GPS antenna. A key feature of this implementation is its ability to handle and estimate the time delay inherent in GPS measurements using a retrospective update and repropagation strategy coupled with an adaptive delay estimation mechanism. The state representation utilizes quaternions for orientation to avoid singularity issues. The mathematical framework and algorithm steps are presented, referencing a corresponding Python implementation.

Contents

1 Introduction

Sensor fusion combines data from multiple sensors to achieve better estimates than could be obtained from each sensor individually. Inertial Navigation Systems (INS) based on IMUs (accelerometers and gyroscopes) provide high-frequency motion data but suffer from drift over time due to noise integration. GPS provides absolute position information but at a lower rate, with higher noise, potential signal loss, and inherent measurement delays.

The Error State Kalman Filter (ESKF) is a popular approach for INS/GPS fusion. It maintains a high-fidelity *nominal state* propagated using the non-linear system dynamics and IMU measurements. A separate Kalman filter operates on a linearly-approximated *error state*, representing small deviations from the nominal state. When GPS measurements arrive, the error state is estimated and used to correct the nominal state. The error state is then reset to zero. This approach avoids issues with linearizing highly non-linear dynamics directly and handles the quaternion unit norm constraint elegantly.

This report focuses on an ESKF that explicitly includes:

- Estimation of the 3D sensor offset vector ($\mathbf{r}$) between the IMU and the GPS antenna.
- Handling of significant, potentially variable, GPS measurement delay (τ).
- An adaptive mechanism to estimate the GPS delay online.

2 System Modeling

2.1 State Definition

We define two state vectors: the nominal state $\mathbf{x}$ and the error state $\delta\mathbf{x}$.

2.1.1 Nominal State

The nominal state vector $\mathbf{x}$ represents the main estimate of the system's status. It includes orientation, velocity, position, and the sensor offset.

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \mathbf{v} \\ \mathbf{p} \\ \mathbf{r} \end{bmatrix} \in \mathbb{R}^{13} \tag{1}$$

where:

- $\mathbf{q} \in \mathbb{R}^4$ is the unit quaternion representing the rotation from the body frame (B) to the inertial frame (I). $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ with $\|\mathbf{q}\| = 1$.
- $\mathbf{v} \in \mathbb{R}^3$ is the velocity of the body frame origin expressed in the inertial frame.
- $\mathbf{p} \in \mathbb{R}^3$ is the position of the body frame origin expressed in the inertial frame.
- $\mathbf{r} \in \mathbb{R}^3$ is the constant lever arm vector from the IMU location (body frame origin) to the GPS antenna phase center, expressed in the body frame.

2.1.2 Error State

The error state vector $\delta\mathbf{x}$ represents small deviations between the true state and the nominal state estimate. For orientation, a minimal 3-parameter representation (rotation vector $\delta\boldsymbol{\theta}$) is used for the error.

$$\delta\mathbf{x} = \begin{bmatrix} \delta\boldsymbol{\theta} \\ \delta\mathbf{v} \\ \delta\mathbf{p} \\ \delta\mathbf{r} \end{bmatrix} \in \mathbb{R}^{12} \tag{2}$$

where:

- $\delta\boldsymbol{\theta} \in \mathbb{R}^3$ is the small angle rotation vector error such that the true quaternion $\mathbf{q}$ relates to the nominal quaternion $\hat{\mathbf{q}}$ by
$$\mathbf{q} \approx \hat{\mathbf{q}} \otimes \delta\mathbf{q}(\delta\boldsymbol{\theta}),$$
where
$$\delta\mathbf{q}(\delta\boldsymbol{\theta}) \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix}.$$
- $\delta\mathbf{v} \in \mathbb{R}^3$ is the velocity error: $\mathbf{v} = \hat{\mathbf{v}} + \delta\mathbf{v}$.
- $\delta\mathbf{p} \in \mathbb{R}^3$ is the position error: $\mathbf{p} = \hat{\mathbf{p}} + \delta\mathbf{p}$.
- $\delta\mathbf{r} \in \mathbb{R}^3$ is the sensor offset error: $\mathbf{r} = \hat{\mathbf{r}} + \delta\mathbf{r}$.

The dimension of the error state ($n_{err} = 12$) is smaller than the nominal state ($n = 13$) due to the minimal representation of orientation error and the unit norm constraint on quaternions.

2.2 Continuous-Time Dynamics

2.2.1 Nominal State Kinematics

The nominal state is propagated using the IMU measurements: measured angular velocity $\boldsymbol{\omega}_m$ and measured specific force (acceleration) $\mathbf{a}_m$. These measurements are corrupted by noise and bias (bias is ignored in this simplified model but often included).

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_m \end{bmatrix} \tag{3}$$
$$\dot{\hat{\mathbf{v}}} = \mathbf{R}(\hat{\mathbf{q}})\Big(\mathbf{a}_m - [\dot{\boldsymbol{\omega}}_m]_\times \hat{\mathbf{r}} - [\boldsymbol{\omega}_m]_\times [\boldsymbol{\omega}_m]_\times \hat{\mathbf{r}}\Big) + \mathbf{g} \tag{4}$$
$$\dot{\hat{\mathbf{p}}} = \hat{\mathbf{v}} \tag{5}$$
$$\dot{\hat{\mathbf{r}}} = \mathbf{0} \tag{6}$$

where:

- $\otimes$ denotes quaternion multiplication.
- $\mathbf{R}(\hat{\mathbf{q}})$ is the rotation matrix corresponding to $\hat{\mathbf{q}}$, transforming vectors from body to inertial frame.
- $\mathbf{g}$ is the gravity vector in the inertial frame (e.g. $\begin{bmatrix} 0 \\ 0 \\ -9.81 \end{bmatrix}$).
- $[\mathbf{v}]_\times$ is the skew-symmetric matrix of vector $\mathbf{v}$.
- $\boldsymbol{\omega}_m$ and $\mathbf{a}_m$ are the gyroscope and accelerometer measurements, respectively.
- $\dot{\boldsymbol{\omega}}_m$ is the angular acceleration, often approximated as zero or derived from $\boldsymbol{\omega}_m$. The term $\mathbf{R}(\hat{\mathbf{q}})\big(\ldots\hat{\mathbf{r}}\big)$ corrects the measured acceleration for the lever arm effect (centripetal and tangential accelerations at the antenna location due to body rotation). Equation (4) is slightly different from the code's `f_state`, which calculates the offset acceleration in the inertial frame and subtracts it after transforming $\mathbf{a}_m$. The underlying physics is the same. The specific force $\mathbf{a}_m$ measured by the accelerometer at the origin is
$$\mathbf{a}_m = \mathbf{R}(\mathbf{q})^\mathsf{T}(\ddot{\mathbf{p}} - \mathbf{g}) - \Big([\dot{\boldsymbol{\omega}}]_\times \mathbf{r} + [\boldsymbol{\omega}]_\times [\boldsymbol{\omega}]_\times \mathbf{r}\Big).$$

Rearranging gives

$$\mathbf{R}(\mathbf{q})\mathbf{a}_m = (\ddot{\mathbf{p}} - \mathbf{g}) - \mathbf{R}(\mathbf{q})\Big([\dot{\boldsymbol{\omega}}]_\times \mathbf{r} + [\boldsymbol{\omega}]_\times [\boldsymbol{\omega}]_\times \mathbf{r}\Big).$$

Thus

$$\ddot{\mathbf{p}} = \mathbf{R}(\mathbf{q})\mathbf{a}_m + \mathbf{R}(\mathbf{q})\Big([\dot{\boldsymbol{\omega}}]_\times \mathbf{r} + [\boldsymbol{\omega}]_\times [\boldsymbol{\omega}]_\times \mathbf{r}\Big) + \mathbf{g}.$$

This matches the structure in Equation (4).

- The sensor offset $\hat{\mathbf{r}}$ is assumed constant during the prediction step ($\dot{\hat{\mathbf{r}}} = \mathbf{0}$).

2.2.2 Error State Dynamics

Linearizing the true state dynamics around the nominal state trajectory yields the continuous-time error state dynamics:

$$\dot{\delta\mathbf{x}}(t) = \mathbf{F}(t)\delta\mathbf{x}(t) + \mathbf{G}(t)\mathbf{w}(t) \tag{7}$$

where $\mathbf{w}(t)$ represents the process noise (primarily from IMU noise), and $\mathbf{F}(t)$ and $\mathbf{G}(t)$ are the state transition and noise input matrices, respectively. For our state definition, $\mathbf{F}$ takes the approximate form:

$$\mathbf{F} = \begin{bmatrix} -[\boldsymbol{\omega}_m]_\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -\mathbf{R}(\hat{\mathbf{q}})\,[(\mathbf{a}_m - \dots)]_\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{R}(\hat{\mathbf{q}})\Big([\dot{\boldsymbol{\omega}}_m]_\times + [\boldsymbol{\omega}_m]_\times^2\Big) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}. \tag{8}$$

The term $-\mathbf{R}(\hat{\mathbf{q}})\,[(\mathbf{a}_m - \dots)]_\times$ in $F_{2,1}$ represents how orientation error $\delta\boldsymbol{\theta}$ affects velocity through incorrect transformation of the measured acceleration (including lever arm terms). The term $F_{2,4}$ represents how sensor offset error $\delta\mathbf{r}$ affects velocity via the lever arm terms.

The Python code uses a simplified version of $\mathbf{F}$ where $\dot{\boldsymbol{\omega}}_m$ is assumed zero, and the dependency of velocity error on offset error ($F_{2,4}$) is neglected in the prediction Jacobian:

$$\mathbf{F}_{code} \approx \begin{bmatrix} -[\boldsymbol{\omega}_m]_\times & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\mathbf{R}(\hat{\mathbf{q}})\,[\mathbf{a}_m]_\times & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{9}$$

The process noise matrix $\mathbf{G}$ maps the IMU noise onto the error state derivatives. Assuming gyroscope noise $\mathbf{w}_g$ affects $\delta\boldsymbol{\theta}$ and accelerometer noise $\mathbf{w}_a$ affects $\delta\mathbf{v}$:

$$\mathbf{G} = \begin{bmatrix} -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -\mathbf{R}(\hat{\mathbf{q}}) \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \mathbf{w}_g \\ \mathbf{w}_a \end{bmatrix} \tag{10}$$

where $\mathrm{Cov}(\mathbf{w}(t)) = \mathbf{Q}_c = \mathrm{diag}(\sigma_g^2\mathbf{I}_3, \sigma_a^2\mathbf{I}_3)$.

2.3 Discretization

For a discrete-time implementation with time step Δt, the nominal state is propagated using Euler or higher-order integration of Eqs. (3)–(6). The error state covariance $\mathbf{P}$ is propagated using the discretized dynamics. The discrete-time state transition matrix $\boldsymbol{\Phi}_k$ and process noise covariance $\mathbf{Q}_d$ are approximated as:

$$\boldsymbol{\Phi}_k \approx \mathbf{I} + \mathbf{F}(\hat{\mathbf{x}}_k)\Delta t, \tag{11}$$

$$\mathbf{Q}_{d,k} \approx \mathbf{G}(\hat{\mathbf{x}}_k)\mathbf{Q}_c\mathbf{G}(\hat{\mathbf{x}}_k)^\mathsf{T}\Delta t \quad \text{(simplified version used in code)}. \tag{12}$$

The code directly defines $\mathbf{Q}_d$ based on noise standard deviations:

$$\mathbf{Q}_d = \mathrm{diag}(\sigma_g^2 \Delta t \mathbf{I}_3, \sigma_a^2 \Delta t \mathbf{I}_3, \mathbf{0}_{3\times3}, \mathbf{0}_{3\times3}). \tag{13}$$

This assumes noise directly impacts the error state variables $\delta\boldsymbol{\theta}$ and $\delta\mathbf{v}$ integrated over Δt, neglecting the rotation matrix $\mathbf{R}$ in the noise propagation for simplicity.

2.4 Measurement Model (GPS)

GPS provides measurements $\mathbf{z}_k$ of the antenna's absolute position $\mathbf{p}_{ant}$ at discrete times t_k, but with a delay τ.

$$\mathbf{p}_{ant}(t) = \mathbf{p}(t) + \mathbf{R}(\mathbf{q}(t))\mathbf{r}. \tag{14}$$

The measurement $\mathbf{z}_k$ received at time t_k corresponds to the state at time $t_k - \tau$:

$$\mathbf{z}_k = \mathbf{p}_{ant}(t_k - \tau) + \mathbf{n}_k = \mathbf{p}(t_k - \tau) + \mathbf{R}(\mathbf{q}(t_k - \tau))\mathbf{r} + \mathbf{n}_k, \tag{15}$$

where $\mathbf{n}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{meas})$ is the measurement noise, with $\mathbf{R}_{meas} = \sigma_{gps}^2 \mathbf{I}_3$.

Linearizing the measurement around the nominal state estimate at the delayed time $t_{k-d} = t_k - \hat{\tau}$ (where $d = \hat{\tau}/\Delta t$ is the estimated delay in steps):

$$\mathbf{z}_k \approx \left(\hat{\mathbf{p}}_{k-d} + \mathbf{R}(\hat{\mathbf{q}}_{k-d})\hat{\mathbf{r}}_{k-d}\right) + \mathbf{H}_{k-d}\delta\mathbf{x}_{k-d} + \mathbf{n}_k. \tag{16}$$

The measurement residual (innovation) $\mathbf{y}_k$ is:

$$\mathbf{y}_k = \mathbf{z}_k - \left(\hat{\mathbf{p}}_{k-d} + \mathbf{R}(\hat{\mathbf{q}}_{k-d})\hat{\mathbf{r}}_{k-d}\right). \tag{17}$$

The measurement Jacobian $\mathbf{H}_{k-d}$ relates the error state $\delta\mathbf{x}_{k-d}$ to the measurement residual:

$$\mathbf{H}_{k-d} = \left.\frac{\partial \mathbf{p}_{ant}}{\partial \delta\mathbf{x}}\right|_{t_{k-d}} = \begin{bmatrix} -\mathbf{R}(\hat{\mathbf{q}}_{k-d})\,[\hat{\mathbf{r}}_{k-d}]_\times & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{R}(\hat{\mathbf{q}}_{k-d}) \end{bmatrix}. \tag{18}$$

The code uses a simplified Jacobian, assuming the effect of orientation error ($H_{1,1}$) and offset error ($H_{1,4}$) on the *predicted* measurement is small compared to the position error itself, especially if the offset estimate is reasonable. This is common in ESKF for simplicity:

$$\mathbf{H}_{code} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}. \tag{19}$$

This simplification means the GPS update directly corrects only the position error $\delta\mathbf{p}$, relying on the filter's correlation (encoded in $\mathbf{P}$) to implicitly correct other states like orientation and offset.

3 ESKF Algorithm with Delay Compensation

The algorithm proceeds in discrete time steps k.

3.1 Initialization

- Initialize nominal state $\hat{\mathbf{x}}_0$ (e.g., known starting position/velocity, identity quaternion, zero initial offset estimate).
- Initialize error state covariance $\mathbf{P}_0^+$ (e.g., diagonal matrix reflecting initial uncertainty).
- Initialize delay estimate $\hat{\tau}_0$.
- Store initial state and covariance in history buffers: `x_hat_hist`$[0] = \hat{\mathbf{x}}_0$, `P_hist`$[0] = \mathbf{P}_0^+$.

3.2 Prediction Step (IMU Propagation)

For each time step $k = 1, 2, \ldots$:

1. **Nominal State Propagation:** Propagate the previous nominal state $\hat{\mathbf{x}}_{k-1}^{+}$ to $\hat{\mathbf{x}}_{k}^{-}$ using the IMU measurements $\boldsymbol{\omega}_{m,k-1}, \mathbf{a}_{m,k-1}$ and the discrete-time integration of Eqs. (3)–(6). Normalize the resulting quaternion $\hat{\mathbf{q}}_{k}^{-}$.
2. **Error State Covariance Propagation:**
   - Compute the discrete error state transition matrix $\boldsymbol{\Phi}_{k-1} \approx \mathbf{I} + \mathbf{F}_{code}(\hat{\mathbf{x}}_{k-1}^{+})\Delta t$, using the simplified Jacobian from Eq. (9).
   - Compute the discrete process noise covariance $\mathbf{Q}_{d,k-1}$.
   - Propagate the covariance: $\mathbf{P}_{k}^{-} = \boldsymbol{\Phi}_{k-1}\mathbf{P}_{k-1}^{+}\boldsymbol{\Phi}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{d,k-1}$.
3. **Store History:** Save the predicted state and covariance: `x_hat_hist`$[k] = \hat{\mathbf{x}}_{k}^{-}$ and `P_hist`$[k] = \mathbf{P}_{k}^{-}$.
4. **Reset Error State Mean:** The predicted error state mean is always zero: $\hat{\delta\mathbf{x}}_{k}^{-} = \mathbf{0}$.

Set $\hat{\mathbf{x}}_{k}^{+} = \hat{\mathbf{x}}_{k}^{-}$ and $\mathbf{P}_{k}^{+} = \mathbf{P}_{k}^{-}$ temporarily. These will be overwritten if a GPS update occurs.

3.3 Measurement Update Step (GPS Correction)

If a GPS measurement $\mathbf{z}_k$ is available at step k:

1. **Determine Delayed Index:** Calculate the index corresponding to the estimated time of measurement emission:
$$d = \mathrm{round}(\hat{\tau}_{k-1}/\Delta t),$$
clipped to ensure $0 \le d < k$. The relevant past index is $j = k - d$.
2. **Retrieve Past State and Covariance:** Get the stored nominal state $\hat{\mathbf{x}}_{j}^{-} =$ `x_hat_hist`$[j]$ and covariance $\mathbf{P}_{j}^{-} =$ `P_hist`$[j]$ from the history buffers.
3. **Predict Measurement:** Calculate the expected GPS measurement based on the past state:
$$\hat{\mathbf{z}}_j = \hat{\mathbf{p}}_{j}^{-} + \mathbf{R}(\hat{\mathbf{q}}_{j}^{-})\hat{\mathbf{r}}_{j}^{-}.$$
(Note: The code simplifies this to $\hat{\mathbf{z}}_j = \hat{\mathbf{p}}_{j}^{-}$ corresponding to the simplified $\mathbf{H}_{code}$.)
4. **Compute Innovation:** $\mathbf{y}_k = \mathbf{z}_k - \hat{\mathbf{z}}_j$.
5. **Compute Innovation Covariance:** $\mathbf{S}_k = \mathbf{H}_{code}\mathbf{P}_{j}^{-}\mathbf{H}_{code}^{\mathsf{T}} + \mathbf{R}_{meas}$.
6. **Compute Kalman Gain:** $\mathbf{K}_k = \mathbf{P}_{j}^{-}\mathbf{H}_{code}^{\mathsf{T}}\mathbf{S}_{k}^{-1}$.
7. **Estimate Error State (at time j):** $\hat{\delta\mathbf{x}}_{j}^{+} = \mathbf{K}_k\mathbf{y}_k$.
8. **Update Covariance (at time j):** $\mathbf{P}_{j}^{corr} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_{code})\mathbf{P}_{j}^{-}$.
9. **Correct Nominal State (at time j):** Inject the estimated error $\hat{\delta\mathbf{x}}_{j}^{+}$ into the past nominal state $\hat{\mathbf{x}}_{j}^{-}$ to obtain the corrected past state $\hat{\mathbf{x}}_{j}^{corr}$:
$$\begin{aligned}
\hat{\mathbf{q}}_{j}^{corr} &= \hat{\mathbf{q}}_{j}^{-} \otimes \delta\mathbf{q}(\hat{\delta\boldsymbol{\theta}}_{j}^{+}) \quad \text{(Normalize after)},\\
\hat{\mathbf{v}}_{j}^{corr} &= \hat{\mathbf{v}}_{j}^{-} + \hat{\delta\mathbf{v}}_{j}^{+},\\
\hat{\mathbf{p}}_{j}^{corr} &= \hat{\mathbf{p}}_{j}^{-} + \hat{\delta\mathbf{p}}_{j}^{+},\\
\hat{\mathbf{r}}_{j}^{corr} &= \hat{\mathbf{r}}_{j}^{-} + \hat{\delta\mathbf{r}}_{j}^{+}.
\end{aligned}$$

10. **Repropagate:** Propagate the corrected state $\hat{\mathbf{x}}_j^{corr}$ and its covariance $\mathbf{P}_j^{corr}$ forward from step $j+1$ to the current step k using the stored IMU measurements and the prediction step logic. This yields the final updated state $\hat{\mathbf{x}}_k^+$ and covariance $\mathbf{P}_k^+$.

11. **Update History:** Overwrite the history buffers from index j to k with the repropagated values. Set `x_hat_hist`$[k] = \hat{\mathbf{x}}_k^+$ and `P_hist`$[k] = \mathbf{P}_k^+$.

12. **Update Delay Estimate:** Use an adaptive rule. The code uses:

$$\hat{\tau}_k = \max\left(0, \hat{\tau}_{k-1} + \gamma_\tau \frac{(\hat{\mathbf{v}}_k^+)^\mathsf{T}\mathbf{y}_k}{\left\|\hat{\mathbf{v}}_k^+\right\|^2 + \epsilon}\right), \tag{20}$$

   where γ_τ is a tuning gain, $\hat{\mathbf{v}}_k^+$ is the final updated velocity at step k, and ϵ is a small constant for numerical stability. This rule heuristically adjusts the delay based on the projection of the position innovation onto the current velocity direction. If the delay estimate is too small, the innovation might point opposite to the velocity, and vice versa.

If no GPS measurement is available, set $\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^-$, $\mathbf{P}_k^+ = \mathbf{P}_k^-$, and $\hat{\tau}_k = \hat{\tau}_{k-1}$. Store $\hat{\tau}_k$ in `tau_est_hist`$[k]$.

4 Implementation Details (Python Code Reference)

The provided Python code implements the ESKF algorithm described above. Key functions and their mathematical counterparts:

- `quaternion_mult, quat_derivative, normalize_quat, rotation_matrix`: Standard quaternion operations used for orientation representation and propagation (Eq. (3)).

- `f_state(x, u, dt, g)`: Implements the discrete-time nominal state propagation (step 1 in Prediction). It takes state $\mathbf{x}$, input $\mathbf{u} = \{\boldsymbol{\omega}_m, \mathbf{a}_m, \dot{\boldsymbol{\omega}}_m\}$ (where $\dot{\boldsymbol{\omega}}_m$ is often zero), timestep Δt, and gravity $\mathbf{g}$. It computes $\hat{\mathbf{x}}_k^-$ from $\hat{\mathbf{x}}_{k-1}^+$. The lever arm acceleration `a_offset` is computed and subtracted as described.

- `run_ESKF_with_delay_estimation(gains)`: Contains the main filter loop.
  - **Initialization:** Sets up $\hat{\mathbf{x}}_0$, $\mathbf{P}_0$, $\hat{\tau}_0$, and history arrays.
  - **Prediction Loop (for k in range(1, steps)):**
    - `x_hat_pred = self.f_state(...)`: Nominal state prediction.
    - `F = np.zeros(...)` and setting elements of `F`: Constructs the simplified error state Jacobian $\mathbf{F}_{code}$ (Eq. (9)).
    - `Phi = np.eye(n_err) + F * dt`: Calculates discrete transition matrix $\mathbf{\Phi}_k$.
    - `Q = np.zeros(...)` and setting elements of `Q`: Defines discrete process noise $\mathbf{Q}_d$.
    - `P = Phi @ P @ Phi.T + Q_d`: Error covariance prediction.
    - `x_hat = x_hat_pred.copy()`: Updates nominal state (temporarily).
    - `x_hat_hist`[k] = `x_hat` and `P_hist`[k] = `P`: Stores history.
  - **Measurement Update (if not np.isnan(self.z_gps[k, 0])):**
    - `delay_steps_est = int(...)` and `delayed_index = k - delay_steps_est`: Calculates past index j (step 1).
    - `x_delayed = x_hat_hist[delayed_index].copy()` and `P_delayed = P_hist[delayed_index]` Retrieves past state/covariance (step 2).

        * `p_pred = x_delayed[7:10]` and `y = z - p_pred`: Computes innovation $\mathbf{y}_k$ using the simplified measurement prediction (steps 3 and 4).
        * `H = np.zeros(...)` with `H[:, 6:9] = np.eye(3)`: Defines simplified measurement Jacobian $\mathbf{H}_{code}$ (Eq. (19)).
        * `R_meas = np.eye(3) * ...`: Defines measurement noise covariance $\mathbf{R}_{meas}$.
        * `S = H @ P_delayed @ H.T + R_meas` and `K = P_delayed @ H.T @ np.linalg.inv(S)`: Calculates innovation covariance $\mathbf{S}_k$ and Kalman gain $\mathbf{K}_k$ (steps 5 and 6).
        * `delta_x = K @ y`: Estimates error state $\hat{\delta\mathbf{x}}_j^+$ (step 7).
        * `P_corr = (np.eye(n_err) - K @ H) @ P_delayed`: Updates covariance $\mathbf{P}_j^{corr}$ (step 8).
        * `delta_theta = delta_x[0:3] ... x_corr = np.concatenate(...)`: Corrects nominal state $\hat{\mathbf{x}}_j^{corr}$ (step 9).
        * **Repropagation Loop (for j in range(delayed_index+1, k+1)):**
            · `x_update = self.f_state(...)`: Propagates corrected nominal state forward.
            · Computes `F_temp`, `Phi_temp`, `Q_temp` at each step.
            · `P_update = Phi_temp @ P_update @ Phi_temp.T + Q_temp`: Propagates corrected covariance forward.
        * `x_hat = x_update.copy()` and `P = P_update.copy()`: Sets current state/covariance to repropagated values (end of step 10).
        * Updates history buffers: `x_hat_hist`[k] = `x_hat` and `P_hist`[k] = `P` (step 11).
        * `vel_update = x_update[4:7] ... tau_est = max(tau_est, 0)`: Updates delay estimate $\hat{\tau}_k$ using Eq. (20) (step 12).
    – `tau_est_hist`[k] = `tau_est`: Stores current delay estimate.

- `IMUSimulator`: Class managing the simulation setup (true trajectory generation, sensor noise simulation, GPS delay).

- `objective`, **Optuna integration:** Used for tuning the observer gains (γ_τ, τ_0) by minimizing the Root Mean Squared Error (RMSE) of the position estimate.

5 Conclusion

The Error State Kalman Filter provides a robust framework for fusing IMU and GPS data. By operating on an error state, it effectively handles the non-linearities of the system dynamics and the quaternion constraints. The inclusion of sensor offset estimation improves accuracy when the IMU and GPS antenna are not co-located. The retrospective update and repropagation strategy allows the filter to correctly incorporate delayed GPS measurements. Furthermore, the adaptive delay estimation mechanism provides resilience against unknown or varying GPS latencies. The mathematical formulation presented here, implemented in the referenced Python code, demonstrates a practical approach to solving this common navigation problem. Tuning of parameters like process/measurement noise covariances and the delay adaptation gain is crucial for optimal performance.