Extended Kalman Filter for Trans-Lunar Coast Tracking

by Aaryan Sonawane

April 18, 2025

1 Introduction

During the trans-lunar coast, a spacecraft is tracked by the Goldstone DSN ground station via range, range-rate, and bearing measurements. I implement an Extended Kalman Filter (EKF) to estimate the spacecraft's inertial position and velocity in the ECI frame.

2 Data Description

I use two data files:

- `MANE6964_HW5_traj.csv`: True spacecraft state every 60 s (time, r_x, r_y, r_z, v_x, v_y, v_z).
- `MANE6964_HW5_meas.csv`: DSN measurements every 300 s (time, ρ, $\dot{\rho}$, bearing ENU components).

3 Mathematical Implementation

In this section, I derive a discrete-time Extended Kalman Filter (EKF) to fuse two-body dynamics with Goldstone DSN measurements (range ρ, range-rate $\dot{\rho}$, and ENU-frame bearing). The continuous-time state evolves under

$$\dot{\mathbf{x}} = f(\mathbf{x}) + G\,w, \quad w \sim \mathcal{N}(\mathbf{0}, Q),$$

with

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \end{bmatrix}, \qquad f(\mathbf{x}) = \begin{bmatrix} \mathbf{v} \\ -\mu \dfrac{\mathbf{r}}{\|\mathbf{r}\|^3} \end{bmatrix}, \qquad G = \begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix}.$$

We linearise about the current estimate $\mathbf{x}$ to obtain the state-transition Jacobian $F(\mathbf{x})$, and propagate the mean and covariance over each $\Delta t = 60\,\mathrm{s}$ via

$$\hat{\mathbf{x}}_{k|k-1} = \hat{\mathbf{x}}_{k-1|k-1} + f\big(\hat{\mathbf{x}}_{k-1|k-1}\big)\,\Delta t, \quad P_{k|k-1} = P_{k-1|k-1} + \Big(F P_{k-1|k-1} + P_{k-1|k-1} F^{\mathsf{T}} + G Q_c G^{\mathsf{T}}\Big)\,\Delta t.$$

At each measurement epoch (t_k), we form the residual

$$\mathbf{y}_k = \mathbf{z}_k - h\big(\hat{\mathbf{x}}_{k|k-1}\big),$$

where $h(\mathbf{x})$ maps the state into $(\rho, \dot{\rho}, \mathbf{u}_{\mathrm{ENU}})$. The Kalman gain is

$$K_k = P_{k|k-1} H_k^{\mathsf{T}} \left(H_k \, P_{k|k-1} \, H_k^{\mathsf{T}} + R_k \right)^{-1},$$

and the update equations are

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k \, \mathbf{y}_k, \quad P_{k|k} = \big(I - K_k H_k \big) \, P_{k|k-1}.$$

This framework yields a continuously refined inertial state estimate with formally propagated uncertainty. Detailed derivations of F, the measurement Jacobians H_ρ, $H_{\dot{\rho}}$, and H_{bear}, and the noise covariances Q_c and R_k are given in this report.

3.1 Frames and Notation

- **ECI** (Earth-Centred Inertial): inertial X, Y, Z axes, origin at Earth CM, used for $\mathbf{r}, \mathbf{v}$.
- **ECEF** (Earth-Centred Earth-Fixed): rotates with Earth at ω_E.
- **ENU** (East-North-Up): local tangent frame at the Goldstone antenna.

$\mathbf{r}_S, \mathbf{v}_S$ = satellite state, $\mathbf{r}_G, \mathbf{v}_G$ = station state.

3.2 State Vector and Priors

We track six states:

$$\mathbf{x} \; = \; \big[x, \, y, \, z, \, v_x, \, v_y, \, v_z \big]^{\mathsf{T}} \quad [\mathrm{km}, \, \mathrm{km\,s^{-1}}].$$

Initially start at

$$\hat{\mathbf{x}}_0 = \begin{bmatrix} 500 \\ 6500 \\ 3500 \\ -10 \\ 2 \\ 3 \end{bmatrix}, \qquad P_0 = \mathrm{diag}\big(100^2, \, 100^2, \, 100^2, \, 0.1^2, \, 0.1^2, \, 0.1^2\big).$$

```
x = np.array([500, 6500, 3500, -10, 2, 3], dtype=float)
P = np.diag([100**2]*3 + [0.1**2]*3)
```

3.3 Continuous-Time Dynamics

Physics. Two body equations:

$$\dot{\mathbf{r}} = \mathbf{v}, \qquad \dot{\mathbf{v}} = -\mu \frac{\mathbf{r}}{r^3} + w$$

where noise is given by $w \sim \mathcal{N}(\mathbf{0}, \sigma_w^2 I_3)$

Compact form.

$$\dot{\mathbf{x}} \;=\; f(\mathbf{x}) \;+\; G\,w, \qquad G = \begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix}, \quad Q_c = \sigma_w^2 I_3.$$

Jacobian F matrix. We linearise the dynamic model $\dot{\mathbf{v}}(\mathbf{r}) = \mathbf{a}(\mathbf{r}) = -\mu\,\dfrac{\mathbf{r}}{r^3},\ r = \|\mathbf{r}\|$ about the current estimate. The Jacobian block we need is $\dfrac{\partial \mathbf{a}}{\partial \mathbf{r}} = \left[\dfrac{\partial a_i}{\partial r_j}\right]_{i,j=1..3}$.

Step 1 – rewrite accelaration component.

$$\mathbf{a}(\mathbf{r}) = -\mu\, r^{-3}\,\mathbf{r}.$$

Step 2 – product rule for each component.
Let $g(\mathbf{r}) = r^{-3}$ and $h(\mathbf{r}) = \mathbf{r}$. For any component $a_i = g\,h_i$:

$$\frac{\partial a_i}{\partial r_j} = \underbrace{\frac{\partial g}{\partial r_j}}_{\text{(i)}} h_i + g\,\underbrace{\frac{\partial h_i}{\partial r_j}}_{\text{(ii)}}.$$

Step 3 – evaluate the two pieces.
(ii)Derivative of $h_i = r_i$*:* $\dfrac{\partial h_i}{\partial r_j} = \delta_{ij}$
(i)Derivative of $g = r^{-3}$*:*
Because $r = (\mathbf{r}^{\mathsf T}\mathbf{r})^{1/2}$,

$$\frac{\partial r}{\partial r_j} = \frac{r_j}{r}, \qquad \frac{\partial g}{\partial r_j} = \frac{\partial}{\partial r_j}\left(r^{-3}\right) = -3\,r^{-4}\,\frac{\partial r}{\partial r_j} = -3\,\frac{r_j}{r^5}.$$

Step 4 – assemble $\partial a_i/\partial r_j$.

$$\frac{\partial a_i}{\partial r_j} = \left(-3\,\frac{r_j}{r^5}\right) r_i + r^{-3}\,\delta_{ij} = r^{-3}\left(\delta_{ij} - 3\,\frac{r_i r_j}{r^2}\right).$$

Step 5 – convert to matrix form
Collecting the components gives the 3×3 matrix

$$\boxed{\frac{\partial \mathbf{a}}{\partial \mathbf{r}} = -\mu\left(\frac{I_3}{r^3} - 3\,\frac{\mathbf{r}\,\mathbf{r}^{\mathsf T}}{r^5}\right)}.$$

Step 6 – Partial Derivative added to the 6×6 state Jacobian.

$$F(\mathbf{x}) = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ -\mu\left(\dfrac{I_3}{r^3} - 3\dfrac{\mathbf{r}\mathbf{r}^{\mathsf T}}{r^5}\right) & 0_{3\times3} \end{bmatrix}.$$

```
def dynamics(x):
    r, v = x[:3], x[3:]
    a = -MU * r / np.linalg.norm(r)**3
    return np.hstack((v, a))

def jacobian_F(x):
    r = x[:3]; r_norm = np.linalg.norm(r)
    dadr = -MU * (np.eye(3)/r_norm**3
                  - 3*np.outer(r, r)/r_norm**5)
    F = np.zeros((6,6))
    F[:3, 3:] = np.eye(3)
    F[3:, :3] = dadr
    return F
```

3.4 Time Discretisation (60-s Step)

For small Δt the exact matrix exponential $\Phi = \exp(F\Delta t)$ can be replaced by the first-order Euler $\Phi \approx I + F\Delta t$.

$$\hat{\mathbf{x}}_{k|k-1} = \hat{\mathbf{x}}_{k-1|k-1} + f\big(\hat{\mathbf{x}}_{k-1|k-1}\big)\,\Delta t, \tag{1}$$

$$P_{k|k-1} = P_{k-1|k-1} + \big(FP_{k-1|k-1} + P_{k-1|k-1}F^{\mathsf{T}} + GQ_cG^{\mathsf{T}}\big)\Delta t. \tag{2}$$

Because we update every minute (and $|F|\Delta t \ll 1$) the truncation error is negligible compared to process noise.

3.5 Goldstone Antenna in ECI

Goldstone's latitude φ_G, longitude λ_G, altitude h_G. An ECEF position is

$$\mathbf{r}_G^{\text{ECEF}} = (R_E + h_G)\begin{bmatrix}\cos\varphi_G\cos\lambda_G\\ \cos\varphi_G\sin\lambda_G\\ \sin\varphi_G\end{bmatrix}.$$

Rotate about $+Z$ at Earth rate ω_E:

$$C_{\text{ECI}\leftarrow\text{ECEF}}(t) = \begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}, \qquad \theta = \omega_E t.$$

Velocity is $\dot{\mathbf{r}}_G = \boldsymbol{\omega}_E \times \mathbf{r}_G$.

```
def station_eci(t):
    theta = OMEGA_E * t
    c,s = np.cos(theta), np.sin(theta)
    x_e, y_e, z_e = r_station_ecef
    r = np.array([ c*x_e - s*y_e,
                   s*x_e + c*y_e,
                   z_e ])
    v = OMEGA_E * np.array([-r[1], r[0], 0.0])
    return r, v
```

3.6 Measurement Models H matrix

(a) Range

$$\rho = \|\mathbf{r}_S - \mathbf{r}_G\|, \quad h_\rho(\mathbf{x}) = \rho, \quad H_\rho = \begin{bmatrix} \dfrac{(\mathbf{r}_S - \mathbf{r}_G)^\mathsf{T}}{\rho} & 0_{1\times 3} \end{bmatrix}.$$

(b) Range-Rate

$$\dot{\rho} = \frac{(\mathbf{r}_S - \mathbf{r}_G)^\mathsf{T}}{\rho}(\mathbf{v}_S - \mathbf{v}_G).$$

Let $\mathbf{u} = (\mathbf{r}_S - \mathbf{r}_G)/\rho$ (unit line-of-sight). Define the "sliding" matrix $S = (I_3 - \mathbf{u}\mathbf{u}^\mathsf{T})/\rho$. Then

$$H_{\dot{\rho}} = \left[(\mathbf{v}_S - \mathbf{v}_G)^\mathsf{T} S \;\mid\; \mathbf{u}^\mathsf{T}\right].$$

(c) Bearing (ENU Unit Vector) We need LOS expressed in the local ENU frame:

$$\mathbf{u}_{\text{ENU}} = C_{\text{ENU}\leftarrow\text{ECI}}(t)\,(\mathbf{r}_S - \mathbf{r}_G)/\rho, \qquad h_{\text{bear}}(\mathbf{x}) = \mathbf{u}_{\text{ENU}}.$$

Using the orthogonality projector $M = (I_3 - \mathbf{u}\mathbf{u}^\mathsf{T})/\rho$ and chain rule,

$$H_{\text{bear}} = \left[\, M\, C_{\text{ENU}\leftarrow\text{ECI}}(t) \;\mid\; 0_{3\times 3}\right].$$

3.7 Measurement Noise R Matrix

Given values: $\sigma_\rho = 10$ km, $\sigma_{\dot{\rho}} = 0.1\ \text{km}\,\text{s}^{-1}$, $\sigma_{\text{bear}} = 1$ arc-min $= 2.91 \times 10^{-4}$ rad.

Therefore $R_\rho = \sigma_\rho^2$, $R_{\dot{\rho}} = \sigma_{\dot{\rho}}^2$, $R_{\text{bear}} = \sigma_{\text{bear}}^2 I_3$.

3.8 EKF Update at Time t_k

$$\mathbf{y}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}), \tag{3}$$
$$S_k = H_k P_{k|k-1} H_k^\mathsf{T} + R_k, \tag{4}$$
$$K_k = P_{k|k-1} H_k^\mathsf{T} S_k^{-1}, \tag{5}$$
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k \mathbf{y}_k, \tag{6}$$
$$P_{k|k} = (I_6 - K_k H_k) P_{k|k-1}. \tag{7}$$

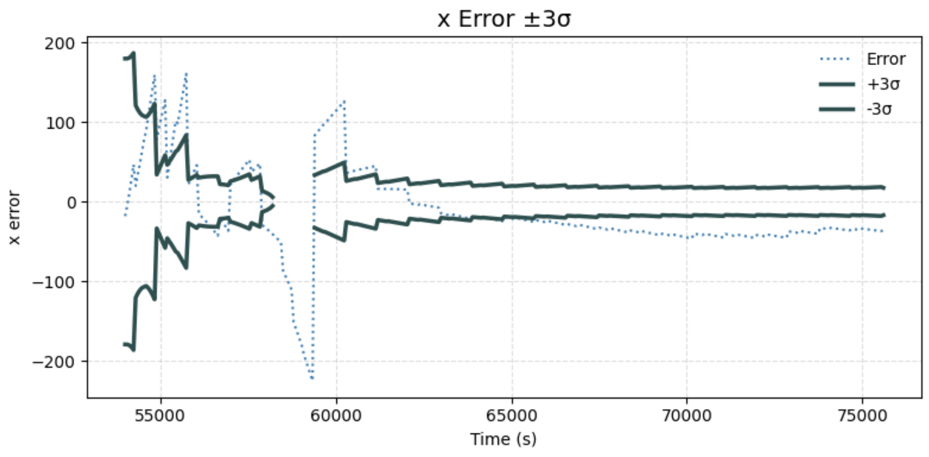

Figure 1: ECI position error in the x-axis with $\pm 3\sigma$ bounds.

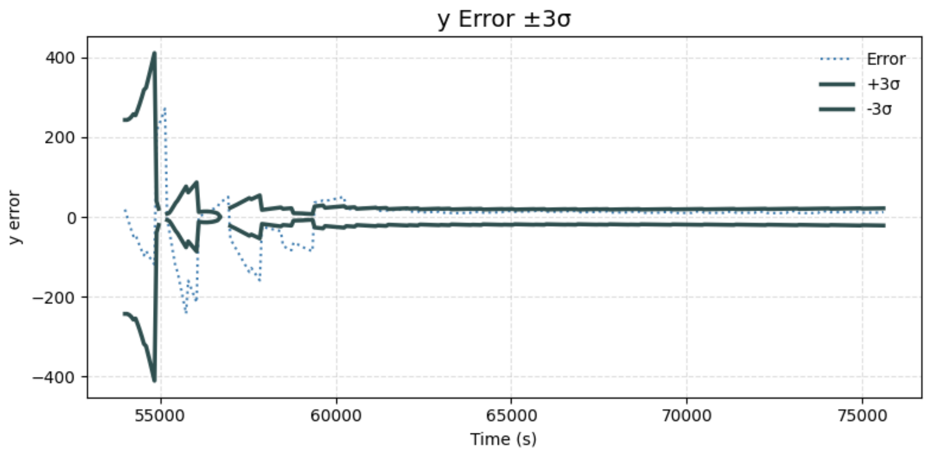

Figure 2: ECI position error in the y-axis with $\pm 3\sigma$ bounds.

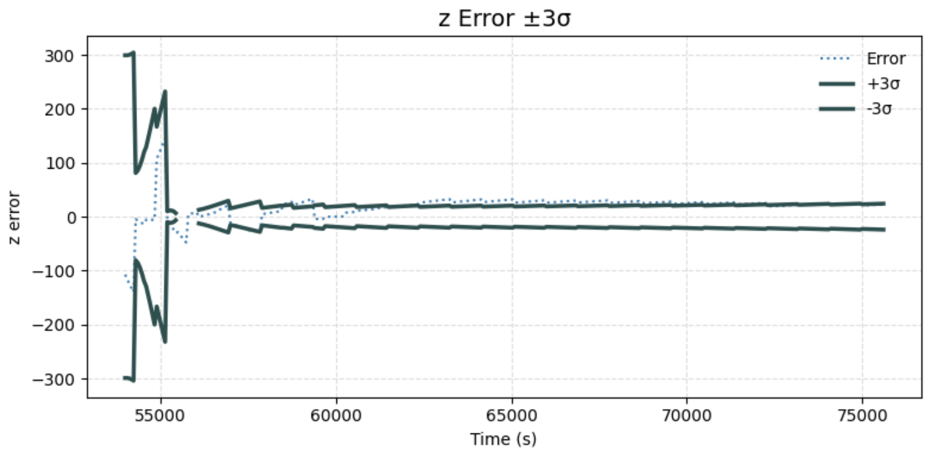

Figure 3: ECI position error in the z-axis with $\pm 3\sigma$ bounds.

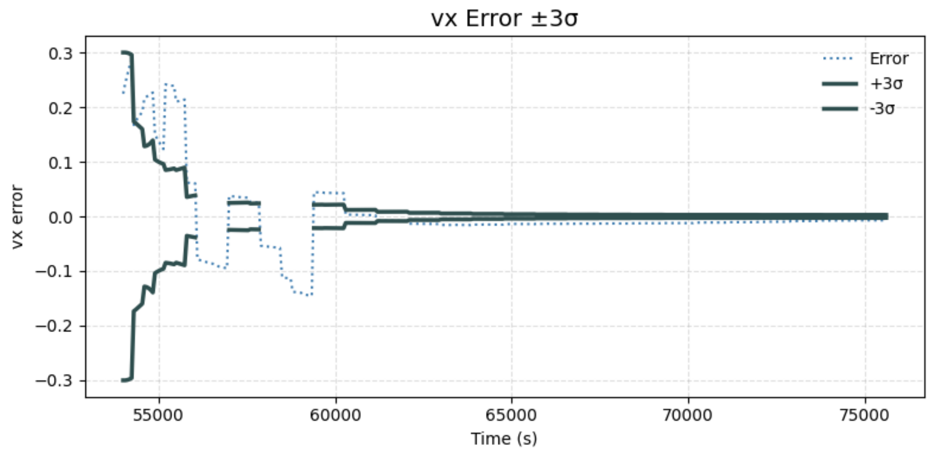

Figure 4: ECI velocity error in the v_x component with $\pm 3\sigma$ bounds.

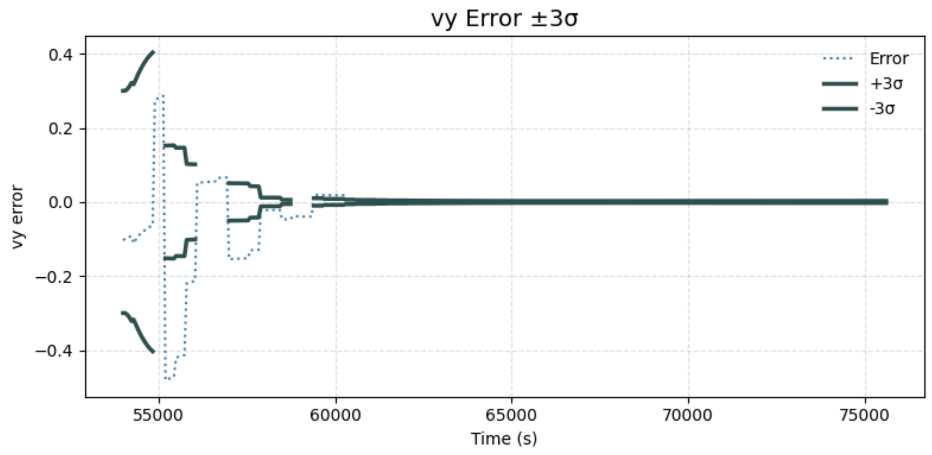

Figure 5: ECI velocity error in the v_y component with $\pm 3\sigma$ bounds.

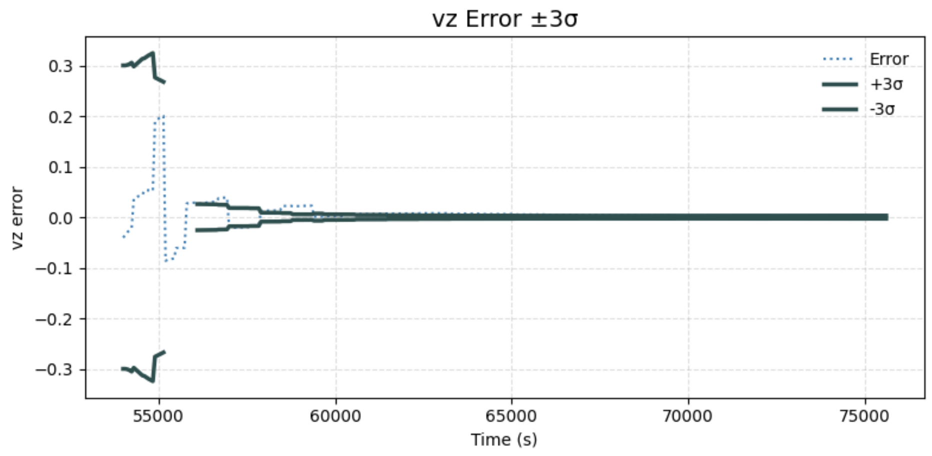

Figure 6: ECI velocity error in the v_z component with $\pm 3\sigma$ bounds.

4 Results

Almost all EKF errors fall within ±3, validating our Q and R settings. The error in x (20km) reflects limited observability from a single station. Enhancements could include multi-station data and higher-fidelity propagation.

A Python Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import files

uploaded = files.upload()

df_meas = pd.read_csv('MANE6964_HW5_meas.csv')
df_traj = pd.read_csv('MANE6964_HW5_traj.csv')

# Constants
mu = 398600.0               # Earth's gravitational parameter, km
    ^3/s^2
omega_earth = 7.2921159e-5  # Earth's rotation rate, rad/s
R_earth = 6371.0            # Earth's radius, km
lat = np.deg2rad(35 + 25/60 + 36/3600)
lon = np.deg2rad(-(116 + 53/60 + 24/3600))
alt = 0.900                 # Station altitude, km
r_stn_ecef = (R_earth + alt) * np.array([
    np.cos(lat)*np.cos(lon),
    np.cos(lat)*np.sin(lon),
    np.sin(lat)
])

# ECEF->ENU rotation
E_enu = np.array([
    [-np.sin(lon),              np.cos(lon),               0],
    [-np.sin(lat)*np.cos(lon), -np.sin(lat)*np.sin(lon),  np.cos(
        lat)],
    [ np.cos(lat)*np.cos(lon),  np.cos(lat)*np.sin(lon),  np.sin(
        lat)]
])

# Process noise (acceleration)
sigma_w = 1e-5
Q = sigma_w**2 * np.eye(3)
```

```
# Measurement noise
sigma_r = 10.0                    # km
sigma_rr = 0.1                    # km/s
sigma_bearing = np.deg2rad(1/60)  # rad
R_range = np.array([[sigma_r**2]])
R_rr = np.array([[sigma_rr**2]])
R_bearing = sigma_bearing**2 * np.eye(3)

# Initial state & covariance
x = np.array([500., 6500., 3500., -10., 2., 3.])
P = np.diag([100.**2, 100.**2, 100.**2, 0.1**2, 0.1**2, 0.1**2])

# Helper functions
def station_eci(t):
    theta = omega_earth * t
    c, s = np.cos(theta), np.sin(theta)
    x_e, y_e, z_e = r_stn_ecef
    # ECI position
    r = np.array([c*x_e - s*y_e, s*x_e + c*y_e, z_e])
    # ECI velocity (omega cross r)
    v = omega_earth * np.array([-r[1], r[0], 0.0])
    return r, v

def dynamics(x):
    r, v = x[:3], x[3:]
    r_norm = np.linalg.norm(r)
    a = -mu * r / r_norm**3
    return np.hstack((v, a))

def jacobian_F(x):
    r = x[:3]
    r_norm = np.linalg.norm(r)
    I3 = np.eye(3)
    dadr = -mu * (I3 / r_norm**3 - 3 * np.outer(r, r) / r_norm**5)
    F = np.zeros((6,6))
    F[0:3,3:6] = I3
    F[3:6,0:3] = dadr
    return F

def predict(x, P, dt):
    F = jacobian_F(x)
    G = np.vstack((np.zeros((3,3)), np.eye(3)))
    # State propagation (Euler)
    x = x + dynamics(x) * dt
    # Covariance propagation
    P = P + (F @ P + P @ F.T + G @ Q @ G.T) * dt
    return x, P

def h_range(x, t):
    r_sat = x[:3]
    r_stn, _ = station_eci(t)
    return np.linalg.norm(r_sat - r_stn)

def H_range(x, t):
    r_sat = x[:3]
    r_stn, _ = station_eci(t)
    diff = r_sat - r_stn
    rho = np.linalg.norm(diff)
    H = np.zeros((1,6))
    H[0, :3] = diff / rho
    return H

def h_range_rate(x, t):
    r_sat = x[:3]
    v_sat = x[3:]
    r_stn, v_stn = station_eci(t)
    diff_r = r_sat - r_stn
    diff_v = v_sat - v_stn
    u = diff_r / np.linalg.norm(diff_r)
    return u.dot(diff_v)

def H_range_rate(x, t):
    r_sat = x[:3]
    v_sat = x[3:]
    r_stn, v_stn = station_eci(t)
    diff_r = r_sat - r_stn
    diff_v = v_sat - v_stn
    rho = np.linalg.norm(diff_r)
    u = diff_r / rho
    #       / r
    d_u = (np.eye(3)/rho - np.outer(u,u)/rho)
    Hr = diff_v @ d_u
    H = np.zeros((1,6))
    H[0, :3] = Hr
    H[0, 3:6] = u
    return H

def h_bearing(x, t):
    r_sat = x[:3]
    r_stn, _ = station_eci(t)
    diff_r = r_sat - r_stn
    # ECI->ECEF
    theta = omega_earth * t
    c, s = np.cos(theta), np.sin(theta)
    C = np.array([[ c, s,0],[-s, c,0],[0,0,1]])
    r_ecef = C @ diff_r
    # ENU
    r_enu = E_enu @ r_ecef
    return r_enu / np.linalg.norm(r_enu)

def H_bearing(x, t):
    r_sat = x[:3]
    r_stn, _ = station_eci(t)
    diff_r = r_sat - r_stn
    # Rotation
    theta = omega_earth * t
    c, s = np.cos(theta), np.sin(theta)
    C = np.array([[ c, s,0],[-s, c,0],[0,0,1]])
    r_ecef = C @ diff_r
    r_enu = E_enu @ r_ecef
    norm_ = np.linalg.norm(r_enu)
    u = r_enu / norm_
    M = (np.eye(3) - np.outer(u,u)) / norm_
    H3 = M @ E_enu @ C
    H = np.hstack((H3, np.zeros((3,3))))
    return H

# Map measurement times for fast lookup
meas_dict = {row['time (sec)']: row for _, row in df_meas.iterrows
   ()}

# Time vector (every 60s)
times = df_traj['time (sec)'].values
n = len(times)

# Storage
x_est = np.zeros((n,6))
P_est = np.zeros((n,6))

# EKF loop
t_prev = times[0]
for i, t in enumerate(times):
    dt = t - t_prev if i > 0 else 0
    if i > 0:
        x, P = predict(x, P, dt)
    # Measurement update if available
    if t in meas_dict:
        m = meas_dict[t]
        # Range update
        rho = m[' range (km)']
        if rho != 0:
            z = np.array([[rho]])
            H = H_range(x, t)
            y = z - np.array([[h_range(x,t)]])
            S = H @ P @ H.T + R_range
            K = P @ H.T @ np.linalg.inv(S)
            x = x + (K @ y).flatten()
            P = (np.eye(6) - K @ H) @ P
        # Range-rate update
        rr = m[' range rate (km/s)']
        if rr != 0:
            z = np.array([[rr]])
            H = H_range_rate(x, t)
            y = z - np.array([[h_range_rate(x,t)]])
            S = H @ P @ H.T + R_rr
            K = P @ H.T @ np.linalg.inv(S)
            x = x + (K @ y).flatten()
            P = (np.eye(6) - K @ H) @ P
        # Bearing update
        b = np.array([m[' bear. x ENU'], m['  bear. y ENU'], m[' 
           bear. z ENU']])
        if np.linalg.norm(b) > 0:
            z = b.reshape(3,1)
            H = H_bearing(x, t)
            y = z - h_bearing(x,t).reshape(3,1)
            S = H @ P @ H.T + R_bearing
            K = P @ H.T @ np.linalg.inv(S)
            x = x + (K @ y).flatten()
            P = (np.eye(6) - K @ H) @ P
    # Store
    x_est[i] = x
    P_est[i] = np.sqrt(np.diag(P))
    t_prev = t

# True states
true = df_traj[[' rx ECI (km)',' ry ECI (km)',' rz ECI (km)',
                ' vx ECI (km/s)',' vy ECI (km/s)',' vz ECI (km/s)'
                  ]].values
```

```
# Plotting errors + 3

labels = ['x','y','z','vx','vy','vz']
for idx in range(6):
    plt.figure(figsize=(8,4))
    err    = true[:,idx] - x_est[:,idx]
    sigma3 = 3 * P_est[:,idx]
    plt.plot(times, err, color='steelblue', linestyle=':',
       linewidth=1.5, label='Error')
    plt.plot(times,  sigma3, color='darkslategray', linestyle='-',
        linewidth=2.5, label='+3  ')
    plt.plot(times, -sigma3, color='darkslategray', linestyle='-',
        linewidth=2.5, label='-3  ')
    plt.xlabel('Time (s)')
    plt.ylabel(f'{labels[idx]} error')
    plt.title(f'{labels[idx]} Error   3   ', fontsize=14)
    plt.legend(frameon=False)
    plt.grid(True, linestyle='--', alpha=0.4)
    plt.tight_layout()
    plt.show()
```