Extended Kalman Filter

Core Concept: A recursive state estimation algorithm designed to handle real-world nonlinearities by constructing a localised linearization of the system at each time step. While the standard Kalman Filter (KF) is restriced to linear systems, the EKF uses Taylor Series to perform Jacobian Linearization, creating localised linear approximations of nonlinear dynamics.

1. The Big Picture - Why EKF?:

The standard KF assumes linearity, but many robots (a great example being drones!) can only be described via nonlinear dynamics. Ex:

- **Nonlinear Motion:** a drone rotates and tilts, meaning we require trigonometry (sin, cos) to describe the motion. This cannot be represented by a constant state transition matrix F.
- **Nonlinear Sensing:** converting raw 3D vectors (like magnetic field that is measured by a magnetometer) into a state (like Yaw) also requires nonlinear operations, meaning we cannot have a constant observation matrix H.

2. The "Variables" - From Matrices to Functions:

Most of the variables from KF are reused, but we replace linear F, H matrices with nonlinear functions (f, h) and their Jacobians:

↳ $f(x_{t-1}, u_t)$: The nonlinear state transition function which takes the previous state (x_{t-1}) and current control input (u_t) and maps it to the next state prior ($\hat{x}_t$), based on the system dynamics.

↳ $h(\hat{x}_t)$: The nonlinear observation function. Takes the current state (x_t) and maps it to what the sensor is expected to see.

↳ F_t: The Jacobian of f. A matrix using partial derivatives of f that represents a local linear approximation of the motion. ($\frac{\partial f}{\partial x}$)

↳ H_t: The Jacobian of h. A matrix using partial derivatives of h that represents a local linear approximation of the sensor model. $\left(\frac{\partial h}{\partial x}\right)$

3. The Recursive EKF Algorithm:

- **Step A: Prediction (The Prior):**

i) **State Prediction:** $\hat{x}_t = f(x_{t-1}, u_t)$ ①

- This step is fairly self-explanatory. We our propagting forwards our state via a prediction using the previous state (x_{t-1}) and the control input (u_t).
- Comparing to the KF, f replaces the role of F and B in the equation $\hat{x}_t = Fx_{t-1} + Bu_t$, to enable nonlinearity.

ii) **Uncertainty Propagation:** $P_t = F_t P_{t-1} F_t^T + Q$ ②

- Note this equation is almost identical to its counterpart from KF. The only difference is that F has been swapped for F_t.
- Note F_t is denoted as such because it is recalculated every time step, to act as a local linearization at that instance.
- We simply calculate $F_t = \frac{\partial f}{\partial x}\Big|_{x_{t-1}}$ and propagate P. To understand in more detail why this works, see proof #1 in appendix.

- **Step B: Correction (The Posterior):**

i) **Kalman Gain:** $K_t = P_t H_t^T (H_t P_t H_t^T + R)^{-1}$ ③

- This is also almost identical to the counterpart from KF. The only difference is that H has been swapped for H_t.
- Similar to F_t, H_t is also recomputed on each correction time step.
- We take $H_t = \frac{\partial H}{\partial x}\Big|_{\hat{x}_t}$ and the compute K_t for same purpose as KF. See proof #2 in appendix for why this works.

ii) **State Upatate:** $x_t = \hat{x}_t + K_t(z_t - h(\hat{x}_t))$ ④ ($\hat{x}_t$ prior, x_t posterior)

- Very similar to counterpart from KF, but $H\hat{x}_t$ term becomes $h(\hat{x}_t)$.
- This enables nonlinearity in the sensor model!
- Note: the innovation is now given by $(z_t - h(\hat{x}_t))$.

iii) Uncertainty Update: $P_t = (I - K_t H_t)\hat{P}_t$ ($\hat{P}_t$ prior, P_t posterior)

- Also almost identical to KF counterpart, with H swapped for H_t.
- This is the same H_t from this time step's Kalman Gain computation.
- Again, proof #2 explains why the Jacobian is a valid approximation.

4. Concrete Example - 3D Drone AHRS EKF:

Scenario: We want to construct an EKF to track a drone's orientation in 3D space via its roll, pitch and yaw angles. This EKF acts as an AHRS (Attitude & Heading Reference System) for the drone. We use a gyroscope as the control input and an accelerometer and magnetometer for correction. This mirrors a fairly simple yet realistic sensor setup for a drone: a 6-DOF IMU (3D gyro + 3D accel) and a 3-DOF magnetometer.

1. Define Vectors ($\vec{x}, \vec{u}, \vec{z}$):

i) We define the state as $\vec{x} = [\phi, \theta, \psi]^T$, where ϕ is roll, θ is pitch and ψ is yaw. All are in radians.

ii) We define the control input (gyro) as $\vec{u} = [p, q, r]^T$ where p is the roll rate, q is pitch rate and r is yaw rate, all in rad/s.

iii) We have two measurement vectors $\vec{z}$, one for accel and one for mag.

↳ $\vec{z}_{acc} = [a_x, a_y, a_z]^T$, all in m/s^2.

↳ $\vec{z}_{mag} = [m_x, m_y, m_z]^T$, normalised unit vector.

2. Derive Physical Models (f, h_{acc}, h_{mag}):

i) We define f using Euler kinematic equations as follows. Letting $x_t = [\hat{\phi}, \hat{\theta}, \hat{\psi}]^T$, $x_{t-1} = [\phi, \theta, \psi]^T$ and $u_t = [p, q, r]^T$:

$$f(x_{t-1}, u_t) = \begin{bmatrix} \hat{\phi} \\ \hat{\theta} \\ \hat{\psi} \end{bmatrix} = x_{t-1} + \begin{bmatrix} p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\ q\cos\phi - r\sin\phi \\ q(\sin\phi/\cos\theta) + r(\cos\phi/\cos\theta) \end{bmatrix} \Delta t$$

*If curious about where these equations come from, search 'Euler Angle Rate Equations'

To compute the Jacobian, call the vector in the state transition function $\vec{g}$ such that $f(x_{t-1}, u_t) = x_{t-1} + \vec{g}\Delta t$. Note the g is really a vector-valued function $g(x_{t-1}, u_t)$ since it depends on those inputs.

$$\text{Then: } \frac{\partial F}{\partial x} = \frac{\partial}{\partial x}(x_{t-1} + g(x,u)\Delta t) = \frac{\partial x_{t-1}}{\partial x} + \frac{\partial g}{\partial x}\Delta t$$

$$\text{Noting } F_t = \frac{\partial f}{\partial x}\Big|_{x_{t-1}} \Rightarrow F_t = I_3 + \Delta t \frac{\partial g}{\partial x}\Big|_{x_{t-1}}$$

$$\text{where: } \frac{\partial g}{\partial x} = \begin{bmatrix} q\cos\phi\tan\theta - r\sin\phi\tan\theta, & q\sin\phi\sec^2\theta + r\cos\phi\sec^2\theta, & 0 \\ -q\sin\phi - r\cos\phi, & 0, & 0 \\ (q\cos\phi - r\sin\phi)\sec\theta, & (q\sin\phi + r\cos\phi)\sec\theta\tan\theta, & 0 \end{bmatrix}$$

So, we now have a deterministic formula for F_t for each step!

ii) We derive h_{acc} as follows, via the inertial gravity vector pointing straight down with $\|g\| = 9.81$. Define $\hat{x}_t = [\phi, \theta, \psi]^T$. Then:

$$h_{acc}(\hat{x}_t) = \begin{bmatrix} -g\sin\theta \\ g\cos\theta\sin\phi \\ g\cos\theta\cos\phi \end{bmatrix} \quad \left(= \text{rotation\_matrix}(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \right)$$

This is simply the gravity vector rotated to the body frame (math left as an exercise to the reader. Then, for the Jacobian $H_{acc,t}$:

$$H_{acc,t} = \frac{\partial h_{acc}}{\partial x} = \begin{bmatrix} 0 & -g\cos\theta & 0 \\ g\cos\theta\cos\phi & -g\sin\theta\sin\phi & 0 \\ -g\cos\theta\sin\phi & -g\sin\theta\sin\phi & 0 \end{bmatrix}$$

iii) Finally, we derive h_{mag}. Details for this one get messy, especially for the Jacobian which is not nice to solve analytically. At a high level, we normalise the reading such that we take $b_{earth} = [1, 0, 0]^T$.

$$h_{mag}(\hat{x}_t) = \text{rotation\_matrix}(\phi, \theta, \psi) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

More details, including the Jacobian, are in `update_mag` in `aekf.py`.

3. Derive Noise Models (Q, R_{acc}, R_{mag}):

Here we simply take sensor variance from datasheet and construct very simple noise models as follows:

i) $Q = I_3 \cdot \text{gyro\_variance}$
ii) $R_{acc} = I_3 \cdot \text{accel\_variance}$
iii) $R_{mag} = I_3 \cdot \text{mag\_variance}$

Where $I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

4. Running the EKF: reference aekf.py

- On init, we set $x_0 = [0,0,0]^T$ and $P_0 = I_3$ like we did in KF.
- Suppose we get IMU (accel + gyro) data @100Hz, and mag data @20Hz.
- predict is called @100Hz on every gyro update, performs the update $\hat{x}_t = f(x_{t-1}, u)$, computes F_t and then does $P_t = F_t P_{t-1} F_t^T + Q$.
- update_accel is called @100Hz (once per gyro predict call, right after). Computes H_t^{acc}, solves for $K_t = P_t H_t^T (H_t P_t H_t^T + R)^{-1}$ and does $x_t = \hat{x}_t + K_t(z_{acc} - h^{acc}(\hat{x}_t))$.
- update_mag is called @20Hz (once per 5 gyro predict calls, right after). Computes H_t^{mag}, solves for K_t and does $x_t = \hat{x}_t + K_t(z_{mag} - h_{mag}(\hat{x}_t))$.

5. Looking Ahead - Solving Gimbal Lock with MEKF:

- Euler Limitation: Using (ϕ, θ, ψ) creates mathematical singularities. Terms in F_t like $\tan\theta$ blow up as pitch approaches $\pm 90°$ (Gimbal Lock), causing filter to crash.
- Quaternion Solution: We replace Euler angles (ϕ, θ, ψ) with quaternions $(q = [q_w, q_x, q_y, q_z])$. These are numerically stable at any orientation and avoid sigularities.
- Multiplicative EKF (MEKF): Standard "Additive" EKFs fail in this case since adding 3D noise to a 4D quaternion breaks its unit-length constraint. MEKF splits state into (1) Nominal State Quaternion (4D) and (2) 3D Error State. Kalman math is done on the error state and then multiplied into the 4D nominal state, hence the name!

Appendix

Proof #1: $P_t = F_t P_{t-1} F_t^T + Q$, where $F_t = \frac{\partial f}{\partial x}\Big|_{x_{t-1}}$

Let the nonlinear system be: $x_t = f(\hat{x}_{t-1}, u_t) + q_t$, $q_t \sim N(0, Q)$.
Then, the state estimate error is $e_t = x_t - \hat{x}_t$, where $\hat{x}_t$ is the prediction from $\hat{x}_t = f(\hat{x}_{t-1}, u_t)$. Linearize $f(x, u)$ around $\hat{x}_{t-1}$ ...

Taylor Series (1st Order):

$$x_t \approx f(x_{t-1}, u) + \frac{\partial f}{\partial x}\Big|_{x_{t-1}} (x_{t-1} - \hat{x}_{t-1}) + q_t$$

$$x_t \approx \hat{x}_t + F_t(x_{t-1} - \hat{x}_{t-1}) + q_t \qquad \left[F_t \triangleq \frac{\partial f}{\partial x}\Big|_{x_{t-1}}\right]$$

$$x_t - \hat{x}_t \approx F_t(x_{t-1} - \hat{x}_{t-1}) + q_t$$

$$\underline{\underline{e_t \approx F_t e_{t-1} + q_t}}$$

So, the error dynamics are now in an approx. linear form! In fact:

$$P_t = \text{cov}(e_t) \approx \text{cov}(F_t e_{t-1} + q_t)$$

$$\approx \text{cov}(F_t e_{t-1}) + q_t$$

$$\approx F_t \,\text{cov}(e_{t-1}) F_t^T + Q$$

$$P_t \approx F_t P_{t-1} F_t^T + Q, \quad \text{QED}$$

Proof #2: $H_t = \frac{\partial h}{\partial x}\Big|_{x_t}$ is a good local linear approx. for H matrix.

Similar to previous proof, can define $z = h(\hat{x}) + r$, where $r \sim N(0, R)$.
We then take our first order Taylor Series linearization around $\hat{x}$:

$$h(x) \approx h(\hat{x}) + \frac{\partial h}{\partial x}\Big|_{\hat{x}} (x - \hat{x}) + r$$

$$\Rightarrow \quad z \approx h(\hat{x}) + H_t(x - \hat{x}) + r$$

This means innovation residual is $y = z - h(\hat{x}) \approx H_t(x - \hat{x}) + r$. Note $(x - \hat{x})$ is just error, so $y \approx H_t e + r$, so we again have error linearised.
Rest of proof for Kalman Gain / new P_t follows same as KF case, QED.