LAB 6: EKF FOR ATTITUDE

OVERVIEW

In this lab, you will:

- Use the **Navigation Toolbox** in MATLAB/Simulink (ensure it's installed).
- Work with an **STM32F411RE Nucleo** board and the **X-Nucleo-IKS02A1** sensor expansion board (IMU).
- Acquire real-time 3-axis accelerometer, gyroscope, and magnetometer data.
- **Align** the magnetometer axes with the accelerometer/gyro.
- **Measure** and **remove** biases in the accelerometer and gyro, and compute sensor noise.
- **Implement** an **Extended Kalman Filter (EKF)** at 100 Hz to estimate roll ϕ\phi, pitch θ\theta, and yaw ψ\psi.
- **Visualise** the fused orientation in real time (Simulink 3D animation).

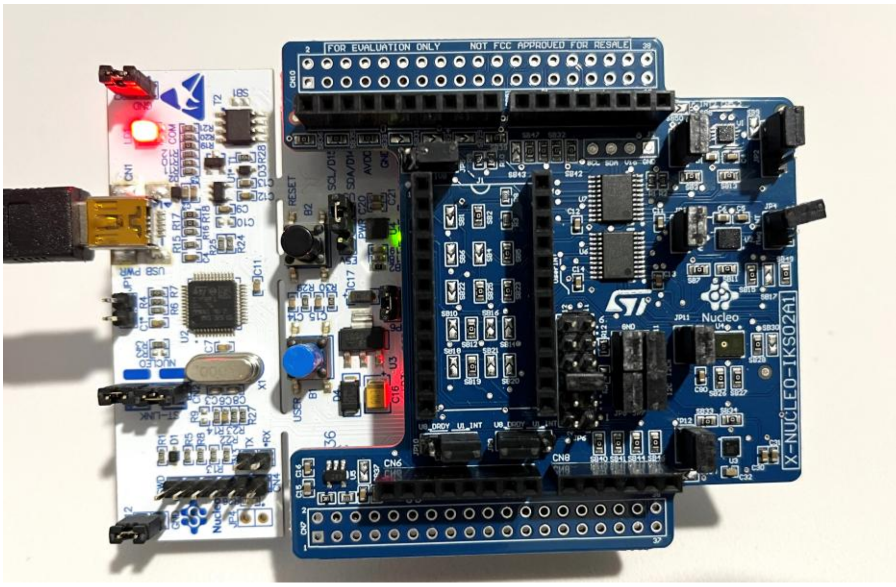


Figure 1: *The STM32F411RE Nucleo and X-Nucleo-IKS02A1 boards.*

TASK 1: SETUP AND MAGNETOMETER ALIGNMENT

Objective

1. Ensure all software is installed (MATLAB, Simulink, Navigation Toolbox).
2. Verify you can read raw data in Simulink from all three sensors (accel, gyro, mag).
3. Rotate the **magnetometer** outputs by 180° about the **Y-axis** to align axes with the accelerometer/gyro.

Instructions

1. **Open Simulink** in MATLAB.
2. **Unzip** the **lab6** materials and open the provided model: **lab6.slx**.
3. In **lab6.slx**, you will see:
    - **Accelerometer** block (ISM330DHCX)
    - **Gyroscope** block (ISM330DHCX)
    - **Magnetometer** block (IIS2MDC)
    - **STM32** configuration block
    - A pink **EKF** MATLAB Function block for the filter code
    - A **2D angle scope** block to visualize angles
    - Scope blocks for real-time signals

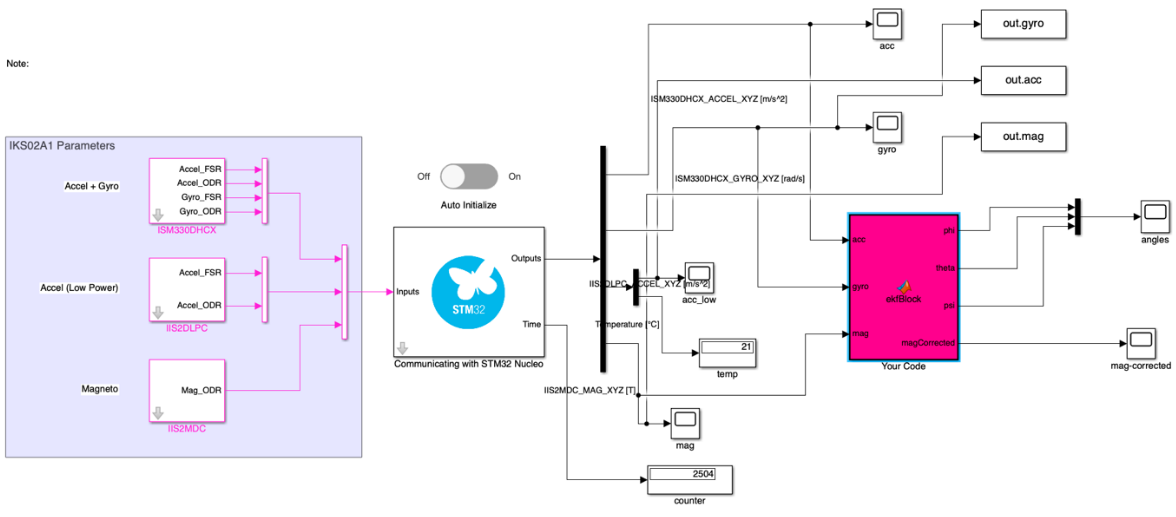


Figure 2: *The Simulink model.*

4. **Connect** your Nucleo (with IKS02A1) to your PC via USB.
5. **Click “Run”** in Simulink and observe the raw signals in the scope blocks.
6. The magnetometer chip (IIS2MDC) is physically mounted in such a way that its X and Z axes are reversed compared to the accelerometer/gyro ISM330DHCX (rotated 180° around the Y axis). If not corrected, the sensor fusion algorithm would interpret magnetometer readings incorrectly. Thus, we need to implement the following change to ensure consistent axes:

$$m'_x = -m_x,\ m'_y = m_y,\ m'_z = -m_z.$$

   Add this code into the pink EKF block:

```
magCorrected = [-mag(1); mag(2); -mag(3)];
```

7. **Observe** updated magnetometer scope signals to confirm alignment.

Demonstration Checkpoint:
Show the instructor/TA that you can see consistent real-time data for all axes, and that the magnetometer is aligned.

TASK 2: BIAS AND NOISE MEASUREMENT (ACCELEROMETER & GYRO)

Objective

- **Measure** constant offsets (bias) in the gyro and accelerometer when the board is still.
- **Calculate** each sensor’s noise standard deviation (σ\sigma) to use later in the EKF.

Instructions

1. **Stationary Logging**
   - Keep the board **flat and motionless** on the desk.
   - The **raw** accelerometer (accelData) and **raw** gyro (gyroData) in Simulink are logged via the **To Workspace** blocks that already exist (no need to be added).

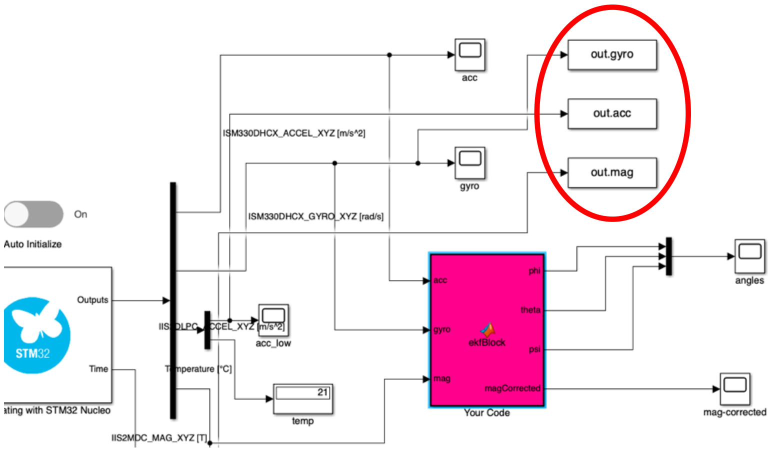


Figure 3: *'To Workspace' blocks on the Simulink model.*

- Run for ~2 minutes to capture data. Data will be logged to variables in the workspace.

2. **Compute Bias** in MATLAB:

```
tempAcc = squeeze(out.acc)' ;  %convert multidimensional array
tempAcc_std = std(tempAcc);
tempgyro = squeeze(out.gyro)' ;
tempgyro_std = std(tempgyro);
gyroBias  = mean(tempgyro_std, 1);   % [gx_bias, gy_bias, gz_bias]
accelBias = mean(tempAcc_std, 1);  % [ax_bias, ay_bias, az_bias]

fprintf('Gyro bias  = [%f, %f, %f]\n', gyroBias);
fprintf('Accel bias = [%f, %f, %f]\n', accelBias);
```

- Ideally, gyro at rest → [0,0,0][0,0,0].
- Accelerometer at rest → typically [0,0,±g][0,0,±g], depending on orientation.

3. **Apply Bias Correction**

```
gyroCorrected = out.gyro - gyroBias, accelCorrected = out.acc - accelBias
```

4. **Measure Noise (Std Dev)**

- Use the **bias-corrected** signals:

```
stdGyro  = std(gyroCorrected);   % [σ_gx, σ_gy, σ_gz]
stdAccel = std(accelCorrected);  % [σ_ax, σ_ay, σ_az]
```

- These values form the **R** matrix entries in the EKF's measurement noise.

Demonstration Checkpoint:

- Show your bias calculations to the instructor/TA.
- Verify bias-corrected outputs are near zero (gyro) and near ±g\pm g (accel) at rest.

TASK 3: SIMPLE MAGNETOMETER HEADING

Objective

Use the (aligned) magnetometer to compute a stand-alone heading (yaw) for demonstration.

Instructions

1. From the *aligned* magnetometer (m_x', m_y', m_z'), compute. $\psi_{\mathrm{mag}} = \mathrm{atan2}(\mathrm{m_y}', \mathrm{m_x}')$ by adding this into the code block:

   ```
   psi = atan2(magCorrected(2), magCorrected(1));
   ```

2. Check how `psi` changes when you rotate the board around the desk (flat).

(You can skip this task if time is short.)

TASK 4: EXTENDED KALMAN FILTER IMPLEMENTATION

Objective

Combine **gyro** in the prediction step with **accelerometer + magnetometer** in the measurement step to estimate Euler angles ϕ, θ, ψ at 100 Hz, reducing drift and noise.

State Definition

the 3–2–1 (roll, pitch, yaw) Euler angles.

$$X = \begin{bmatrix} \phi \\ \theta \\ \Psi \end{bmatrix}$$

Gyro-Based Prediction (continuous time)

$$\dot{\phi} = p + q\, sin(\phi)\, tan(\theta) + r\, cos(\phi)\, tan(\theta)$$

$$\dot{\theta} = q \cos\phi - r \sin\phi$$

$$\dot{\psi} = \frac{q \sin\phi + r \cos\phi}{\cos\phi}$$

Discretize at 100 Hz using:

$$X_{k+1} = X_k + \dot{X}\Delta t$$

Measurement Model

$$Z = \begin{bmatrix} a_x \\ a_y \\ a_z \\ m_x \\ m_y \\ m_z \end{bmatrix}, \quad h(\phi, \theta, \Psi) = \begin{bmatrix} R_{ib}(\phi, \theta, \Psi)\boldsymbol{g}_{inertial} \\ R_{ib}(\phi, \theta, \Psi)\boldsymbol{m}_{inertial} \end{bmatrix}$$

Where $\boldsymbol{g}_{inertial} = [0,0,-g]^T$ and $\boldsymbol{m}_{inertial}$ is the local Earth magnetic field vector in the inertial frame

Jacobian

- F = ∂f/∂x for the prediction step.
- H = ∂h/∂x for the measurement step (6×3).

Covariances

- **Q**: process noise (accounts for gyro bias/drift).
- **R**: measurement noise (diagonal from σ_acc and σ_mag).

Solution Code

Below is **sample code** for the pink **EKF** MATLAB Function block. It uses the bias-corrected accelerometer/gyro as inputs, plus the aligned magnetometer. You can adapt the code, or simply use it to verify your solution:

```
function [phi, theta, psi, magCorrected] = ekfBlock( ...
    acc, ...      % Accelerometer (body frame)
    gyro, ...   % Gyro (body rates)
    mag )         % Magnetometer (body frame)
%#codegen
%
% Demo solution for Task 4:
% 3-state EKF that fuses (a_x,a_y,a_z,m_x,m_y,m_z) with gyro inputs
% to estimate Euler angles (phi,theta,psi).
% Replaces these with your biases calculated previously
accX = acc(1)-712.7836e-003;
accY = acc(2)-(-669.2423e-003);
accZ = acc(3) - 241.0936e-003;
```

```
gyroX = gyro(1)-30.5751e-006;
gyroY = gyro(2)- (-5.1492e-003);
gyroZ = gyro(3) - (-603.1789e-006) ;

magX = -1E-4*mag(1);   %convert to Tesla
magY = 1E-4*mag(2);
magZ = -1E-4*mag(3);
magCorrected = [magX; magY; magZ];

% -------------------------------------------------------------
% 1) Persistent state and covariance
% -------------------------------------------------------------
persistent x P
if isempty(x)
    % State vector: x = [phi; theta; psi]
    x = zeros(3,1);
    P = eye(3)*0.1;  % initial guess for covariance
    P(3,3) = (180/pi)^2;  % initial guess for heading is bad
end

% -------------------------------------------------------------
% 2) Known constants and sensor-noise settings
%    (Use your measured std devs from Task 2!)
% -------------------------------------------------------------
dt = 0.01;      % 100 Hz
g  = 9.81;      % gravitational acceleration [m/s^2]

% Example: local magnetic field in London (approx).
% In an "NED" coordinate system, let's say B ≈ [18.5;  -2.9; 48.7] microTesla
% for an inclination ~66° down from horizontal. You can scale to Tesla or Gauss.
% We'll store it in [X; Y; Z] inertial, with Z downward.
% Adjust these numbers for your reference frame/orientation.
M_inertial = [ 19437.4e-9;  181.2E-9;  45090.4E-9 ];
% (You may need to rotate or tweak this to match your "inertial" frame choice.)

% Example process noise (Q): Tweak if needed.
% You might base these on gyro bias/drift or small angle random walk.
Q = diag([1e-3, 1e-3, 1e-3]);

% Example measurement noise (R): 6x6, for (a_x,a_y,a_z,m_x,m_y,m_z).
% Replace the diag entries with (sigma_acc^2, sigma_mag^2, etc.)
% from Task 2.
R = diag([ ...
    (70E-3)^2, (70E-3)^2, 0.02^2, ...   % accelerometer noise
    0.005^2, 0.005^2, 0.005^2 ... % magnetometer noise
    ]);

% -------------------------------------------------------------
% 3) EKF Prediction Step
%    Using continuous-time Euler-angle kinematics + Euler discretization.
% -------------------------------------------------------------

% Current state
phi_k   = x(1);
theta_k = x(2);
psi_k   = x(3);
```

```
% Body rates from gyro (p,q,r)
p = gyroX;
q = gyroY;
r = gyroZ;

% --- f(x,u): Euler-angle rates ---
phi_dot = p + q*sin(phi_k)*tan(theta_k) + r*cos(phi_k)*tan(theta_k);
theta_dot = q*cos(phi_k) - r*sin(phi_k);
psi_dot = (q*sin(phi_k) + r*cos(phi_k)) / cos(theta_k);

% Discrete prediction: x_pred = x + f*dt
x_pred = x + dt * [phi_dot; theta_dot; psi_dot];

% --- Compute the Jacobian F of f wrt x (3x3) ---
% Partial derivatives of [phi_dot; theta_dot; psi_dot] wrt (phi,theta,psi)
% ignoring direct psi dependency in the standard 3-2-1 equations:
F_ct = zeros(3);
%
% d(phi_dot)/d(phi)
F_ct(1,1) = q*cos(phi_k)*tan(theta_k) - r*sin(phi_k)*tan(theta_k);
% d(phi_dot)/d(theta)
sec2_th = 1/cos(theta_k)^2;  % derivative of tan() is sec^2()
F_ct(1,2) = (q*sin(phi_k) + r*cos(phi_k)) * sec2_th;
%
% d(theta_dot)/d(phi)
F_ct(2,1) = -q*sin(phi_k) - r*cos(phi_k);
%
% d(psi_dot)/d(phi)
one_over_costh = 1 / cos(theta_k);
F_ct(3,1) = one_over_costh * (q*cos(phi_k) - r*sin(phi_k));
%
% d(psi_dot)/d(theta)
% derivative of (1/cos(theta)) = sec(theta)*tan(theta)
F_ct(3,2) = (q*sin(phi_k) + r*cos(phi_k)) * (sin(theta_k)/(cos(theta_k)^2));
% or more explicitly: one_over_costh^2 * sin(theta_k)

% Discrete approximation: F = I + dt*F_ct
F = eye(3) + dt * F_ct;

% Predicted covariance
P_pred = F * P * F' + Q;

% --------------------------------------------------------------
% 4) EKF Measurement Step
%    z = [accX; accY; accZ;  magX; magY; magZ]
%    h(x) = R(phi,theta,psi)*g_inertial  +  R(phi,theta,psi)*M_inertial
% --------------------------------------------------------------

% Construct measurement vector from sensor
z = [accX; accY; accZ;  magX; magY; magZ];

% We'll define the inertial gravity vector as [0; 0; -g].
g_inertial = [0; 0; g];

% ---- Predict sensor readings from x_pred ----
```

```
% Build the total rotation from Euler angles
phi_p   = x_pred(1);
theta_p = x_pred(2);
psi_p   = x_pred(3);

% rotation matrix from inertial frame to body (3-2-1)
Rib = rotX(phi_p)*rotY(theta_p)*rotZ(psi_p);

% Predicted accelerometer output (gravity only)
acc_pred = Rib * g_inertial;

% Predicted magnetometer output
mag_pred = Rib * M_inertial;

% h(x_pred)
z_pred = [acc_pred; mag_pred];

% ---- Compute H = d(h)/d(x) (6x3) ----
% We need partial derivatives of (Rib*g_inertial) wrt phi,theta,psi
% plus partial derivatives of (Rib*M_inertial) wrt phi,theta,psi
H = zeros(6,3);

% We can build them by computing d/d(phi)[Rib*g_inertial], etc.
% For brevity, let's do a small numeric approximation or call a local
% function "dRdEuler(...)".  Here is a simple numeric approach for demo:

eps = 1e-6;
for i = 1:3
    x_shift = x_pred;
    x_shift(i) = x_shift(i) + eps;
    R_shift = rotX(x_shift(1))*rotY(x_shift(2))*rotZ(x_shift(3));
    z_shift = [R_shift*g_inertial; R_shift*M_inertial];
    % finite-diff derivative
    H(:,i) = (z_shift - z_pred)/eps;
end

% Residual
y = z - z_pred;

% S = H P_pred H' + R
S = H * P_pred * H' + R;

% Kalman gain
K = P_pred * H' / S;

% Updated state & covariance
x_upd = x_pred + K*y;
P_upd = (eye(3) - K*H) * P_pred;

% --------------------------------------------------------------
% 5) Save and output
% --------------------------------------------------------------
x = x_upd;
P = P_upd;

phi   = x(1);
```

```matlab
theta = x(2);
psi   = x(3);

end

% ------------------------------------------------------------
% Local rotation helper functions
% ------------------------------------------------------------
function Rx = rotX(a)
    ca = cos(a); sa = sin(a);
    Rx = [1  0   0;
          0  ca -sa;
          0  sa  ca];
end

function Ry = rotY(b)
    cb = cos(b); sb = sin(b);
    Ry = [ cb  0  sb;
           0   1  0;
          -sb  0  cb];
end

function Rz = rotZ(c)
    cc = cos(c); sc = sin(c);
    Rz = [ cc -sc  0;
           sc  cc  0;
           0   0   1];
end
```

Instructions:

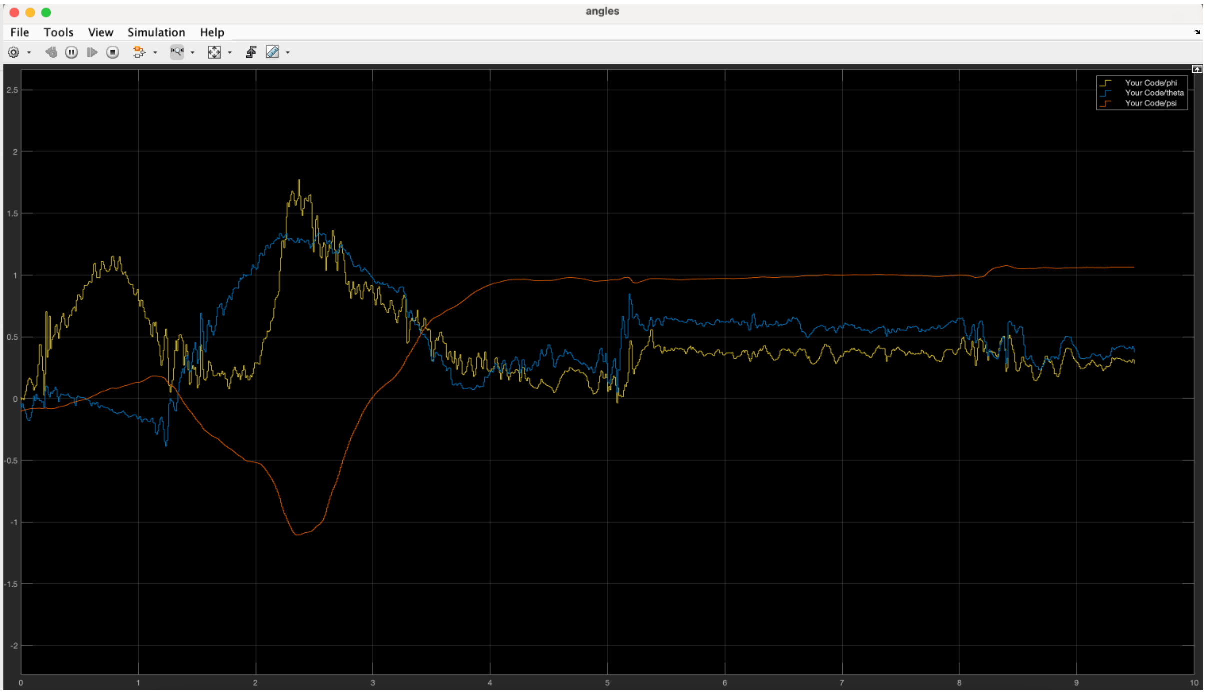
1. **Double-click** the **EKF** block in lab6.slx.
2. **Paste** the above code (or adapt it).
3. **Run** the model. Observe angles on the scope output.

Demonstration Checkpoint:
Show the instructor/TA how the fused angles track smoothly when the board is tilted or rotated, with minimal drift (thanks to bias removal).

Conclusion

- You have successfully implemented an EKF to fuse **accelerometer**, **gyro**, and **magnetometer** data.
- By removing biases (Task 2) and using the correct noise models in **Q** and **R**, you greatly reduce drift.
- Try logging the EKF angles to the workspace and analyzing them vs. raw sensor data to see the filter's effectiveness!

End of Lab 6



[**Figure 3:** *Placeholder for final screenshot (EKF angles vs. raw). Possibly show the 3D orientation plot.*]