Table of Contents


```
function [xhat, meas] = filterTemplate(calAcc, calGyr, calMag)

% FILTERTEMPLATE  Filter template
%
% This is a template function for how to collect and filter data
% sent from a smartphone live.  Calibration data for the
% accelerometer, gyroscope and magnetometer assumed available as
% structs with fields m (mean) and R (variance).
%
% The function returns xhat as an array of structs comprising t
% (timestamp), x (state), and P (state covariance) for each
% timestamp, and meas an array of structs comprising t (timestamp),
% acc (accelerometer measurements), gyr (gyroscope measurements),
% mag (magnetometer measurements), and orint (orientation quaternions
% from the phone).  Measurements not availabe are marked with NaNs.
%
% As you implement your own orientation estimate, it will be
% visualized in a simple illustration.  If the orientation estimate
% is checked in the Sensor Fusion app, it will be displayed in a
% separate view.
%
% Note that it is not necessary to provide inputs (calAcc, calGyr,
 calMag).
```

Setup necessary infrastructure

```
import('com.liu.sensordata.*');  % Used to receive data.

% Filters used
magFilt = true;
accFilt = true;
```

Filter settings

```
t0 = [];  % Initial time (initialize on first data received)
nx = 4;   % Assuming that you use q as state variable.
```

```
% Add your filter settings here.
T = 1/25;
% Gyroscope covariances
sigma_w = 1; % was 0.01
Rw = diag([sigma_w^2 sigma_w^2 sigma_w^2]);
sigma_v = 0.0014;
Rv = diag([sigma_v^2 sigma_v^2 sigma_v^2]);
gyr_old = [0; 0; 0];

% Accelerometer parameters and covariances
g_abs = 9.82;   % 9.8908 from estimate using stationary data.
g_lim = [0.8 1.2]*g_abs;    % +- 20% marginal
sigma_a = 0.3; % From variance estimate using stationary data.
Ra = diag([sigma_a^2 sigma_a^2 sigma_a^2]);

% Magnetometer parameters and covariances
m_abs = 39.1103;   % 39.1103 from estimate using stationary data.
m_lim = [0.6 1.4]*m_abs;    % +- 20% marginal
sigma_m = 0.9; % From variance estimate using stationary data.s
Rm = diag([sigma_m^2 sigma_m^2 sigma_m^2]);


% Current filter state.
x = [1; 0; 0; 0];
P = eye(nx, nx);

% Saved filter states.
xhat = struct('t', zeros(1, 0),...
              'x', zeros(nx, 0),...
              'P', zeros(nx, nx, 0));

meas = struct('t', zeros(1, 0),...
              'acc', zeros(3, 0),...
              'gyr', zeros(3, 0),...
              'mag', zeros(3, 0),...
              'orient', zeros(4, 0));
try
```

Create data link

```
  server = StreamSensorDataReader(3400);
  % Makes sure to resources are returned.
  sentinel = onCleanup(@() server.stop());

  server.start();  % Start data reception.

  % Used for visualization.
  figure(1);
  subplot(1, 2, 1);
  ownView = OrientationView('Own filter', gca);  % Used for
visualization.
  googleView = [];
  counter = 0;  % Used to throttle the displayed frame rate.
```

```
********************
* SensorDataReader started on 10.0.161.113:3400
********************
ERROR:  Failed to open port or no connection attempted.
Unsuccessful connecting to client!
Make sure to start streaming from the phone *after*running this
 function!
```

Filter loop

```
    while server.status()  % Repeat while data is available
      % Get the next measurement set, assume all measurements
      % within the next 5 ms are concurrent (suitable for sampling
      % in 100Hz).
      data = server.getNext(5);

      if isnan(data(1))  % No new data received
        continue;        % Skips the rest of the look
      end
      t = data(1)/1000;  % Extract current time

      if isempty(t0)  % Initialize t0
        t0 = t;
      end

      gyr = data(1, 5:7)';
      % Gyroscope
      if ~any(isnan(gyr))
          gyr_old = gyr;

          [x, P] = tu_qw(x, P, gyr, T, Rw);
          % Include normalization of quaternion before using... where?
          [x, P] = mu_normalizeQ(x, P);
      else
%             [x, P] = tu_qw_pred(x, P, gyr_old, T, Rw);
%             [x, P] = mu_normalizeQ(x, P);
      end



      acc = data(1, 2:4)';
      if accFilt
          if ~any(isnan(acc))  % Acc measurements are available.
          % Approximate g, skip update if "outlier"
          a_abs = sqrt(sum(acc.^2));
              if (a_abs > g_lim(1)) && (a_abs < g_lim(2))
                  g0 = [0; 0; a_abs];
                  [x, P] = mu_g(x, P, acc, Ra, g0);
                  [x, P] = mu_normalizeQ(x, P);
              end
          end
      end
```

```
        mag = data(1, 8:10)';
        if magFilt
            if ~any(isnan(mag))  % Mag measurements are available.
                % Approximate m, skip update if "outlier"
                m_abs = sqrt(sum(mag.^2));
                if (m_abs > m_lim(1)) && (m_abs < m_lim(2))
                    m0 = [0; 12; -36.09]; m0 = m0/norm(m0);
                    [x, P] = mu_m(x, P, mag, Rm, m0);
                    [x, P] = mu_normalizeQ(x, P);
                end
            end
        end

        orientation = data(1, 18:21)';  % Google's orientation estimate.


        % Visualize result
        if rem(counter, 10) == 0
          setOrientation(ownView, x(1:4));
          title(ownView, 'OWN', 'FontSize', 16);
          if ~any(isnan(orientation))
            if isempty(googleView)
              subplot(1, 2, 2);
              % Used for visualization.
              googleView = OrientationView('Google filter', gca);
            end
            setOrientation(googleView, orientation);
            title(googleView, 'GOOGLE', 'FontSize', 16);
          end
        end
        counter = counter + 1;

        % Save estimates
        xhat.x(:, end+1) = x;
        xhat.P(:, :, end+1) = P;
        xhat.t(end+1) = t - t0;

        meas.t(end+1) = t - t0;
        meas.acc(:, end+1) = acc;
        meas.gyr(:, end+1) = gyr;
        meas.mag(:, end+1) = mag;
        meas.orient(:, end+1) = orientation;

      end

    catch e
      fprintf(['Unsuccessful connecting to client!\n' ...
        'Make sure to start streaming from the phone *after*'...
               'running this function!']);
    end

  end
```

```
ans =

  struct with fields:

    t: [1×0 double]
    x: [4×0 double]
    P: [4×4×0 double]
```

"tu_qw"

```
function [x, P] = tu_qw(x, P, omega, T, Rw)
% EKF time update step

% Gyroscope measurement noise covariance (estimated from stationary
 data)
sigma_v = 0.0014;
Rv = diag([sigma_v^2 sigma_v^2 sigma_v^2]);

% Process matrices
F = T/2*Somega(omega) + eye(4);
F_tilde  = F;
G = T/2*Sq(x);
G_tilde = G;

% Prediction step
xp = F*x;
Pp = F_tilde*P*F_tilde' + G_tilde*Rw*G_tilde';



    % Update step
    % Equation: yk = H*xk + B*xkmin1_kmin1 + vk
    H = 2/T*pinv(Sq(x));
    B = -H;
    Bx = B*x;
    yhat = H*xp + Bx;

    S = H*Pp*H' + B*P*B' + Rv;
    K = Pp*H'*(S^-1);

    x = xp + K*(omega - yhat);
    P = Pp - K*S*K';

end
```

"tu_qw_pred"

```
function [x, P] = tu_qw_pred(x, P, omega, T, Rw)
```

```
% EKF time update step

% Gyroscope measurement noise covariance (estimated from stationary
 data)
sigma_v = 0.14;
Rv = diag([sigma_v^2 sigma_v^2 sigma_v^2]);

% Process matrices
F = T/2*Somega(omega) + eye(4);
F_tilde  = F;
G = T/2*Sq(x);
G_tilde = G;

% Estimate omega
xp = F*x;
H = 2/T*pinv(Sq(x));
B = -H;
Bx = B*x;
omegaHat = H*xp + Bx;

% recalculate Process matrices
F = T/2*Somega(omegaHat) + eye(4);
F_tilde  = F;
G = T/2*Sq(x);
G_tilde = G;


% Prediction step
x = F*x;
P = F_tilde*P*F_tilde' + G_tilde*Rw*G_tilde';


end
```

"mu_g"

```
function [x, P] = mu_g(x, P, y, Ra, g0)
% EKF accelerometer measurement update

% Measurement matrices
hx = Qq(x)'*g0;
[dQ0, dQ1, dQ2, dQ3] = dQqdq(x);
Jhx = [dQ0'*g0 dQ1'*g0 dQ2'*g0 dQ3'*g0];

% Measurement update
S = Jhx*P*Jhx' + Ra;
K = P*Jhx'*(S^-1);

x = x + K*(y - hx);
P = P - K*S*K';
```

```
end
```

"mu_m"

```
function [x, P] = mu_m(x, P, y, Rm, m0)
% EKF acclerometer measurement update

% normalize
ynorm = y/norm(y);
m0 = m0/norm(m0);
Rm = Rm/norm(y);

% Measurement matrices
hx = Qq(x)'*m0;
[dQ0, dQ1, dQ2, dQ3] = dQqdq(x);
Jhx = [dQ0'*m0 dQ1'*m0 dQ2'*m0 dQ3'*m0];

% Measurement update
S = Jhx*P*Jhx' + Rm;
K = P*Jhx'*(S^-1);



x = x + K*(y - hx);
P = P - K*S*K';

end
```

"Project_Implementation"

```
% SSY345 Project Implementation

% SECTION 4.1
% Task 2) See file Project_test.m. Measurement data loaded below.

load('data_FrassePhone_stationary.mat');

%% SECTION 4.2
% Task 3) Design the EKF time update step
% Implemented in file "tu_qw.m".

% Time between samples
T = 1/fs;

% Sizes
K = length(t);
n = 4;

% Gyroscope covariances
```

```
sigma_w = 0.01;
Rw = diag([sigma_w^2 sigma_w^2 sigma_w^2]);
sigma_v = 0.0014;
Rv = diag([sigma_v^2 sigma_v^2 sigma_v^2]);

% Accelerometer parameters and covariances
g_abs = 9.82;   % 9.8908 from estimate using stationary data.
g_lim = [0.9 1.1]*g_abs;    % +- 10% marginal
sigma_a = 0.03; % From variance estimate using stationary data.
Ra = diag([sigma_a^2 sigma_a^2 sigma_a^2]);

% Magnetometer parameters and covariances
m_abs = 39.1103;   % 39.1103 from estimate using stationary data.
m_lim = [0.9 1.1]*m_abs;    % +- 10% marginal
sigma_m = 0.5; % From variance estimate using stationary data.s
Rm = diag([sigma_m^2 sigma_m^2 sigma_m^2]);

% Initial estimates and covariances
x_0 = [0; 0; 1; 0];
P_0 = diag([1 1 1 1]);

% Visualisation stuff
figure(1)
ownView = OrientationView('Own filter', gca);

xw = zeros(4,K);
Pw = zeros(4,4,K);
xw(:,1) = x_0;
Pw(:,:,1) = P_0;

xa = zeros(4,K);
Pa = zeros(4,4,K);
xm = zeros(4,K);
Pm = zeros(4,4,K);

x = zeros(4,K);
P = zeros(4,4,K);
NaN_gyr = 1;
NaN_acc = 1;
for k = 2:K

    % Gyroscope measurement computations
    % Prediction:
    [xw(:,k), Pw(:,:,k)] = tu_qw(xw(:,k-1), Pw(:,:,k-1), y_gyr(:,k),
 T, Rw);

    % Include normalization of quaternion before using... where?
    [xw(:,k), Pw(:,:,k)] = mu_normalizeQ(xw(:,k), Pw(:,:,k));

    % Accelerometer computations
    % Measurement update:
    if ~isnan(y_acc(:,k))
        % Approximate g, skip update if "outlier"
        a_abs = sqrt(sum(y_acc(:,k).^2));
```

```
        if (a_abs > g_lim(1)) && (a_abs < g_lim(2))
            g0 = [0; 0; a_abs];
            [xa(:,k), Pa(:,:,k)] = mu_g(xw(:,k), Pw(:,:,k),
y_acc(:,k), Ra, g0);
            [xa(:,k), Pa(:,:,k)] = mu_normalizeQ(xa(:,k), Pa(:,:,k));
        else
            xa(:,k) = xw(:,k);
            Pa(:,:,k) = Pw(:,:,k);
        end
    else
        xa(:,k) = xw(:,k);
        Pa(:,:,k) = Pw(:,:,k);
    end

    % Magnetometer computations
    % Measurement update:
    if ~isnan(y_mag(:,k))
        % Approximate m, skip update if "outlier"
        m_abs = sqrt(sum(y_mag(:,k+1).^2));
        if (m_abs > m_lim(1)) && (m_abs < m_lim(2))
            m0 = [0; 19.18; -34.09];
            [xm(:,k), Pm(:,:,k)] = mu_m(xa(:,k), Pa(:,:,k),
y_mag(:,k), Rm, m0);
            [xm(:,k), Pm(:,:,k)] = mu_normalizeQ(xm(:,k), Pm(:,:,k));
        elseif ~isnan(y_acc(:,k))
            xm(:,k) = xa(:,k);
            Pm(:,:,k) = Pa(:,:,k);
        else
            xm(:,k) = xw(:,k);
            Pm(:,:,k) = Pw(:,:,k);
        end
    elseif ~isnan(y_acc(:,k))
        xm(:,k) = xa(:,k);
        Pm(:,:,k) = Pa(:,:,k);
    else
        xm(:,k) = xw(:,k);
        Pm(:,:,k) = Pw(:,:,k);
    end

    % Combine accelerometer and magnetometer information





    % Update:
    % Temporary for only gyroscope (Task 5)
%     x(:,k+1) = xwp(:,k+1);
%     P(:,:,k+1) = Pwp(:,:,k+1);

%     x(2:4,k+1) = xa(:,k+1);
%     P(2:4,2:4,k+1) = Pa(:,:,k+1);
%     [x(:,k+1),P(:,:,k+1)] = mu_normalizeQ(xwp(:,k+1), Pwp(:,:,k+1));
```

```
    setOrientation(ownView, xm(:,k));
    title(ownView, 'OWN', 'FontSize', 16);
    pause(0.01)
end
```

Published with MATLAB® R2017a