Table of Contents

Extended Kalman filter implementation

```
function [xhat, meas] = Myfilter(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.
  load flat_data

  % 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.
  Rw = cov_gyr;

  Ra = cov_acc;

  g0 = mean_acc;

  accOut = 0;
  magOut = 0;

  m0 = [0; sqrt(mean_mag(1)^2 + mean_mag(2)^2); mean_mag(3)];
```

```
Rm = cov_mag;

alpha = 0.01;

L = norm(mean_mag);

gyr_bool = false;
acc_bool = false;
mag_bool = false;

% 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.

  % 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;
```

```
      xhat.t(end+1) = 0;
    end

    acc = data(1, 2:4)';
    if ~any(isnan(acc))  % Acc measurements are available.
      acc_bool = true;
    end
    gyr = data(1, 5:7)';
    if ~any(isnan(gyr))  % Gyro measurements are available.
      gyr_bool = true;
    end

    mag = data(1, 8:10)';
    if ~any(isnan(mag))  % Mag measurements are available.
      mag_bool = true;
    end

    if gyr_bool == true
        dt = t - t0 - xhat.t(end);

        [x, P] = tu_qw(x, P, gyr, dt, Rw);       % Time update of
EKF
        [x, P] = mu_normalizeQ(x, P);
        gyr_bool = false;
    else
        dt = t - t0 - xhat.t(end);

        [x, P] = tu_q(x, P, dt, Rw);
        [x, P] = mu_normalizeQ(x, P);
    end

    if acc_bool == true
        if abs(9.81 - norm(acc)) <= 0.2
            [x, P] = mu_acc(x, P, acc, Ra, g0);
            [x, P] = mu_normalizeQ(x, P);
            accOut = 0;
        else
            accOut = 1;
        end
        ownView.setAccDist(accOut)
        acc_bool = false;
    end

    if mag_bool == true
        L = (1-alpha)*L + alpha*norm(mag);
        if abs(L - norm(mag)) <= 1.5
          [x, P] = mu_m(x, P, mag, m0,Rm);
          [x, P] = mu_normalizeQ(x, P);
          magOut = 0;
        else
            magOut = 1;
        end
        ownView.setMagDist(magOut);
        mag_bool = false;
```

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

% Time update
function [x, P] = tu_qw(x, P, omega, T, Rw)
% Task 4: Time update function of the EKF
% x         Last time step mean
% P         Last time step covariance matrix
% omega     Measured angular rate
% T         Time since last measurement
% Rw        Process noise covariance matrix


S_w = Somega(omega);
S_q = Sq(x);

F = (eye(4) + (T/2)*(S_w));
G = (T/2)*(S_q);
```

```
x  = F*x;

P = F*P*F.' + G*Rw*G.';

end
```

Random walk

```
function [x, P] = tu_q(x, P, T, Rw)
% Task 4: Time update function of the EKF
% x         Last time step mean
% P         Last time step covariance matrix
% omega     Measured angular rate
% T         Time since last measurement
% Rw        Process noise covariance matrix


S_q = Sq(x);

F = eye(4) ;
G = (T/2)*(S_q);

x  = F*x;

P = F*P*F.' + G*Rw*G.';

end
```

Accelerometer measurement update

```
function [x, P] = mu_acc(x, P, yacc, Ra, g0)
% EKF update using the accelerometer measurements

% yacc      measured acceleration vector
% Ra        Measurement noise covariannce matrix
% g0        nominal gravity vector

Q = Qq(x);
[dQ0, dQ1, dQ2, dQ3] = dQqdq(x);

hx = Q'*g0;
Hx = [dQ0'*g0, dQ1'*g0, dQ2'*g0, dQ3'*g0];

S = Hx*P*Hx.' + Ra;
S = (S + S')/2;

K = P*Hx.'*inv(S);

x = x + K*(yacc - hx);

P = P - K*S*K.';
```

```
end
```

Magnetometer measurement update

```
function [x, P] = mu_m(x, P, mag, m0, Rm)
% EKF update using the magnetometer measurements

% mag       Measured magnetic field vector
% Rm        Measurement noise covariannce matrix
% m0        nominal magnetic field

Q = Qq(x);
[dQ0, dQ1, dQ2, dQ3] = dQqdq(x);

hx = Q'*m0;
Hx = [dQ0'*m0, dQ1'*m0, dQ2'*m0, dQ3'*m0];

S = Hx*P*Hx.' + Rm;
S = (S + S')/2;

K = P*Hx.'*inv(S);

x = x + K*(mag - hx);

P = P - K*S*K.';


end
```

Published with MATLAB® R2020a