# *Orientation estimation using smartphone sensors*

Sensor fusion and nonlinear filtering
SSY345

Gunnar Bolmvall
Måns Östman

October 18, 2017

# Introduction

In this report the implemented solutions, results and discussions are presented for project 1 in the sensor fusion course. The task is to develop a filter to estimate the orientation of a smartphone using its built in sensors.

# 1 Gyroscope measurements as input

Using the gyroscope measurements as input allows us to derive a less complex model in the sense that we get fewer parameters in our $A$ matrix. We do however work under the assumption that the measurements are noise free, which is fine as long as the gyroscope is accurate enough. If we were to have a less accurate sensor one might want to include it in the states instead as we can model the noise for the gyroscope states separately.

# 2 Data collection and filter design

## 2.1 Initial data examination

For this task a Samsung Galaxy S7 was used to collect static measurements when the phone was laying down on a flat surface. From the measurements, histograms and time plots were generated in figure (1). The mean and covariance are stated in table (1,2). In the histograms one can see that the accelerometer and gyroscope sensor produce gaussian distributed measurements. This is also verified when obtaining the results from the time plots. For the magnetometer, the histogram does not look as gaussian as the for the two other sensors due to a larger tail to the right.

The reason for this tail is likely due to disturbances in the magnetic field rather noise from the sensor itself, which can be obtained in the time data. From this plot one can see that the magnetic field is more dynamic compared to the other sensors.

When comparing the covariance table 1 one see that the gyroscope has the smallest noise, the magnetometer the largest and the accelerometer in between. This order will therefore matter for the EKF filter of which sensor to trust the most. The biases/means of the data sequence are stated in table (2). For the accelerometer the biases will be used for calibration, which is equal to setting it equal to $g_0$. The same will be set for magnetometers bias, using the equation (9) in pm. For the gyro, bias is very small and can therefore be neglected since we are only interested in the relative changes.

Table 1: Calculated covariance of calibration data

| | Accelerometer | Gyroscope | Magnetometer |
|---|---|---|---|
| $\sigma_x^2$ | $0.099 \cdot 10^{-3}$ | $0.228 \cdot 10^{-5}$ | 0.153 |
| $\sigma_y^2$ | $0.116 \cdot 10^{-3}$ | $0.123 \cdot 10^{-5}$ | 0.066 |
| $\sigma_z^2$ | $0.056 \cdot 10^{-3}$ | $0.173 \cdot 10^{-5}$ | 0.309 |

Table 2: Calculated mean values of calibration data

| | Accelerometer | Gyroscope | Magnetometer |
|---|---|---|---|
| $\mu_x$ | $-0.04$ | $0.38 \cdot 10^{-3}$ | -16.03 |
| $\mu_y$ | 0.19 | $-0.79 \cdot 10^{-3}$ | 7.06 |
| $\mu_z$ | 9.82 | $0.57 \cdot 10^{-3}$ | -48.08 |

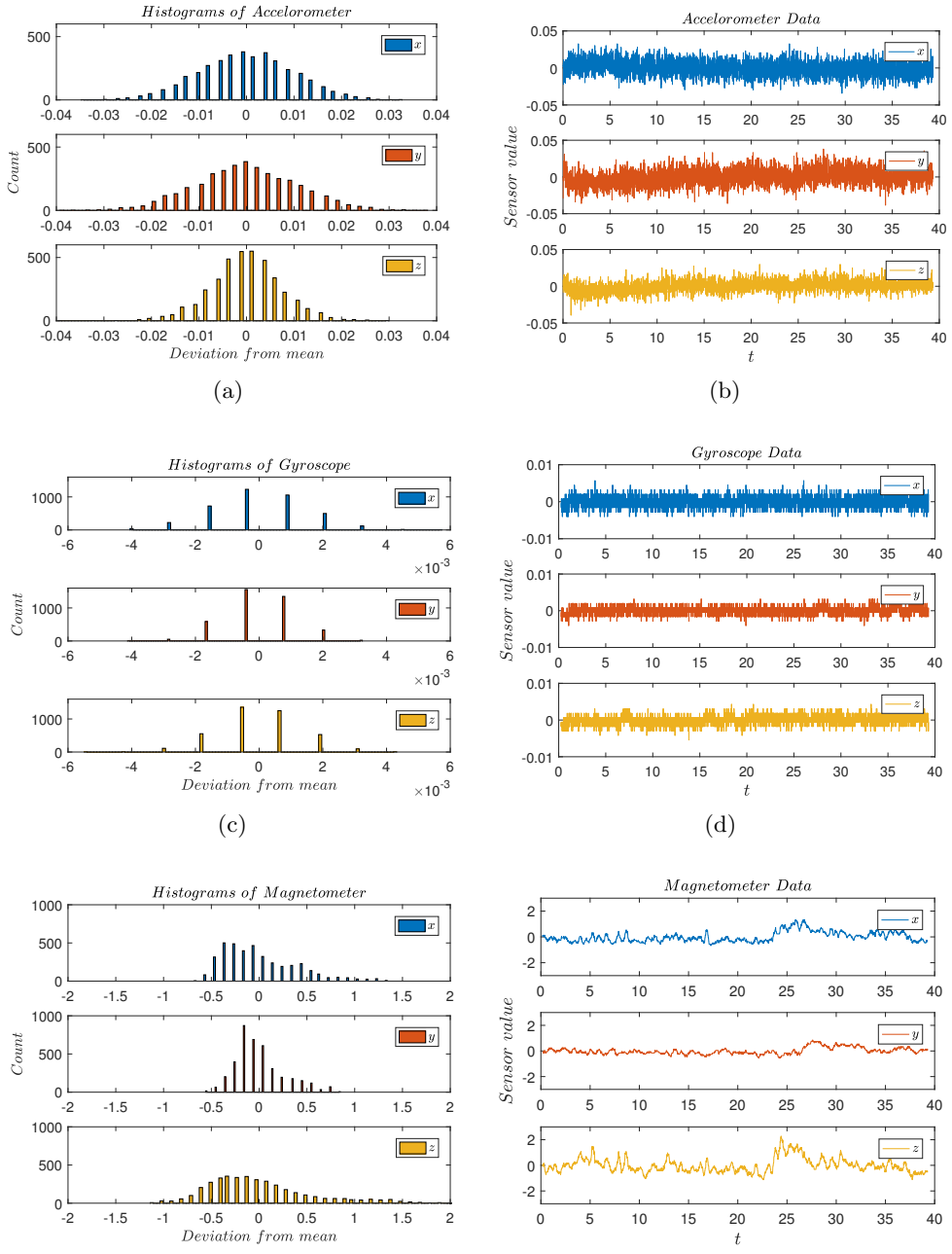


Figure 1: Histograms (with 100 bins) and time plots of accelerometer, gyroscope and magnetometer measurements. The mean values has been subtracted from all plots.

## 2.2 Designing the EKF time update step

### Deriving the prediction model

$$\dot{q}(t) = \frac{1}{2} S(\omega_{k-1} + v_{k-1}) q(t)$$

We know that $A_{k-1} = \exp(T\tilde{A})$ and make the assumption that ($\exp \mathbf{A} \approx \mathbf{I} + \mathbf{A}$). Since $\tilde{A} = \frac{1}{2} S(\omega_{k-1} + v_{k-1})$ we can rewrite the formula as

$$q_k = \mathbf{I} q_{k-1} + \frac{T}{2} S(\omega_{k-1}) q_{k-1} + \frac{T}{2} \overline{S}(q_{k-1}) v_{k-1}.$$

By using the approximation $G(q_{k-1})v_{k-1} \approx G(\hat{q}_{k-1})v_{k-1}$ we get expressions for $F$ and $G$.

$$q_k = \left( \mathbf{I} + \frac{T}{2} S(\omega_{k-1}) \right) q_{k-1} + \frac{T}{2} \overline{S}(\hat{q}_{k-1}) v_{k-1} = F(\omega_{k-1}) q_{k-1} + G(\hat{q}_{k-1}) v_{k-1}.$$

The last approximation we use is reasonable since we linearize the filter around our estimate in the previous time step.

### Performance of the filter

From these equations the following code was implemented. If no measurements are available, random walk is assumed with an arbitrary small process covariance.

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

    % Calculate F and G
    F = eye(size(x,1))+(T/2)*Somega(omega);
    G = (T/2)*Sq(x);

    % Update x and P
    x = F*x;
    P = F*P*F' + G*Rw*G';
end


%%%%%%%%%% Part of main filter loop %%%%%%%%%%

gyr = data(1, 5:7)';
if ~any(isnan(gyr))  % Gyro measurements are available.
    [x, P] = tu_qw(x, P, gyr, t-t0-meas.t(end), Rw); % Update state estimate
    [x, P] = mu_normalizeQ(x,P); % Normalize state vector
else
    % If no measurements are available assume random walk model
    P = P + 0.001*eye(4);
end

%%%%%%%%%% Part of main filter loop %%%%%%%%%%
```

The filter performs rather well when starting with the phone oriented according to the prior quaternion (lying flat and pointing north). The filter is however unable to handle any other starting positions, this is reasonable since the angular rates tells us nothing about the absolute orientation of the phone, only the relative.

## 2.3 An EKF update using accelerometer measurements

### Adding accelerometer update to the EKF

$$
\begin{aligned}
h(q_k) &= Q(q_k)^T g^0 \\
S_k &= h'(\hat{q}_{k|k-1}) P_{k|k-1} h'(\hat{q}_{k|k-1})^T + R_a \\
K_k &= P_{k|k-1} h'(\hat{q}_{k|k-1})^T S_k^{-1} \\
P_{k|k} &= P_{k|k-1} - K_k S_k K_k^T \\
\hat{q}_{k|k} &= \hat{q}_{k|k-1} + K_k (y_k^a - h(\hat{q}_{k|k-1}))
\end{aligned}
$$

From these equations the following code was implemented.

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

    % Calculate measurement estimate
    hx = Qq(x)'*g0;

    % Get derivatives of Q
    [Q0, Q1, Q2, Q3] = dQqdq(x);

    % Calculate Jacobian matrix
    Hx = [Q0'*g0 Q1'*g0 Q2'*g0 Q3'*g0];

    % Calculate Innovation covariance and Kalman gain
    Sk = Hx*P*Hx'+Ra;
    Kk = P*Hx'/Sk;

    % Update x and P
    x = x + Kk*(yacc-hx);
    P = P-Kk*Sk*Kk';
end
```

When testing the performance including the accelerometer update, the filter performs better than before. Now we are able to track how the phone is rotated around $x$ and $y$ relative the world coordinate system. Rotation around $z$ works quite well too but drifts over time. When performing fast translation movements, the filter will perceive the applied force as a rotational movement, since it affects the accelerometers vectors in $x, y, z$. This is due to the assumption that $f_k^a$ is equal zero.

### Adding outlier rejection of accelerometer to the EKF

To avoid the behaviour of interpreter translations as rotations, an outlier rejection function was implemented to disregard measurements if the norm of the accelerometer differs too much from its nominal value. The function was implemented as follows.

```
%%%%%%%%%% Part of main filter loop %%%%%%%%%%

acc = data(1, 2:4)';
if ~any(isnan(acc))  % Acc measurements are available.
    % If the 2-norm for acc measurement is to large/small, skip
    % update step
    if  norm(acc)<9.81*1.25 && norm(acc)>9.81*0.75
        [x, P] = mu_g(x, P, acc, Ra, g0); % Update state estimate
        [x, P] = mu_normalizeQ(x,P); % Normalize state vector
        accOut = 0;
    else
        accOut = 1;
```

```
    end
end

%%%%%%%%%% Part of main filter loop %%%%%%%%%%
```

Both an upper and lower boundary was introduced to handle accelerations and constant acceleration downwards (free fall). When performing tests with this outlier rejection implemented, the phone does not considering translations as temporary rotations which results in a better performance than before.

## 2.4 An EKF update using magnetometer measurements

### Adding magnetometer update to the EKF

$$
\begin{aligned}
h(q) &= Q(q)^T m^0 \\
S_k &= h'(\hat{q}_{k|k-1}) P_{k|k-1} h'(\hat{q}_{k|k-1})^T + R_m \\
K_k &= P_{k|k-1} h'(\hat{q}_{k|k-1})^T S_k^{-1} \\
P_{k|k} &= P_{k|k-1} - K_k S_k K_k^T \\
\hat{q}_{k|k} &= \hat{q}_{k|k-1} + K_k (y_k^m - h(\hat{q}_{k|k-1}))
\end{aligned}
$$

From these equations the following code was implemented.

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

    % Calculate measurement estimate
    hx = Qq(x)'*m0;

    % Get derivatives of Q
    [Q0, Q1, Q2, Q3] = dQqdq(x);

    % Calculate Jacobian matrix
    Hx = [Q0'*m0 Q1'*m0 Q2'*m0 Q3'*m0];

    % Calculate Innovation covariance and Kalman gain
    Sk = Hx*P*Hx'+Rm;
    Kk = P*Hx'/Sk;

    % Update x and P
    x = x + Kk*(mag-hx);
    P = P-Kk*Sk*Kk';
end
```

When testing the performance including the magnetometer update, the filter manage to keep track of the orientation in $z$. The filter is tho sensitive to disturbances and if any is introduced close to the phone, the filter will perform bad and loose the orientation of the phone. This is due to the measurement will become larger disordered in $x, y, z$ than our expected value which is based on the calibrated $m_0$. Therefore we will overcompensate for our orientation and become disoriented.

### Adding outlier rejection of magnetometer to the EKF

To minimize the impact of the disturbance an outlier rejection function was implemented to reject measurements if considered as an outlier.

```
%%%%%%%%%% Part of main filter loop %%%%%%%%%%
```

```
mag = data(1, 8:10)';
if ~any(isnan(mag))  % Mag measurements are available.
    % AR-filter to account for that the magnitude of m0 might drift
    Lk = (1-alpha)*Lk + alpha*norm(mag);

    % If magnitude of measurement is too small or large, skip update step
    if 35<Lk && Lk<55 % Thresholds for magnetic field
        [x, P] = mu_m(x, P, mag, m0, Rm); % Update state estimate
        [x, P] = mu_normalizeQ(x,P); % Normalize state vector
        magOut = 0;
    else
        magOut = 1;
    end
end

%%%%%%%%%% Part of main filter loop %%%%%%%%%%
```

Here $L_k$ is lowpass filtered norm of the measurement to allow the magnetometer to slowly drift over time. To detect an outlier, empirically selected thresholds has been used to set a upper and lower boundary to reject measurement if the magnetic field is larger or smaller than normal. Here we assume that the magnetic field does not change too much and ends up outside of the selected margin. The angle estimation will tho keep the bias of the selected threshold if exceeded, and will therefore perform worse under influence of an magnetic disturbance. When running the filter, it keeps track of the orientation well as before but when an external disturbance is applied, the measurement is now being rejected instead and the filter performs relatively good. When the disturbance source is removed, it takes some time for the filter to adapt to the new magnetic field (due to the lowpass filtering) and then it runs smooth again, as expected.

## 2.5 Final evaluation

When using all three sensors the performance of the filter is comparable to the one implemented in the phone, this can be seen in figure 2. The difference is small but one can see that the filters are tuned a bit differently. The developed filter is a bit quicker which implies that trusting the measurements, while Google trusts the model more, resulting in smoother filtered angles.

The performance of the filter naturally decreases when one of the sensors are removed. How the filter performs without accelerometer and magnetometer has been discussed earlier. Without the gyroscope we modeled the state prediction as a random walk process and it works surprisingly well, the filter is more noisy but manages to position the phone correct most of the time.

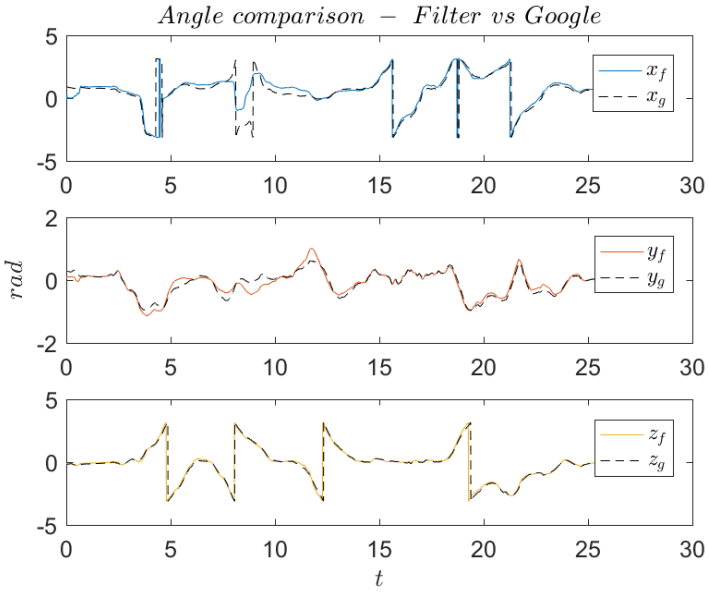


Figure 2: Comparison of euler angles between the filter using all three sensors and the one implemented in the phone by Google.

# 3 Complete filter code

```
function [xhat, meas] = filterTemplate2(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.

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

%%%%%%% Filter settings %%%%%%%%
% S7 Calibration matrices
```

```
Rw = diag([2.285e-06 1.231e-06 1.729e-06]);
Ra = diag([9.946e-05 1.163e-04 5.608e-05]);
Rm = diag([0.152 0.0658 0.309]);
g0 = [-0.040 0.189 9.823]';
m0 = [0 17.517 -48.080]';

% Other parameters
magOut = 1;
accOut = 1;
alpha = 0.01;
Lk = norm(m0);

% 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;
        end

        gyr = data(1, 5:7)';
        if ~any(isnan(gyr))  % Gyro measurements are available.
            [x, P] = tu_qw(x, P, gyr, t-t0-meas.t(end), Rw); % Update state estimate
            [x, P] = mu_normalizeQ(x,P); % Normalize state vector
        else
            % If no measurements are available assume random walk model
            P = P + 0.001*eye(4);
        end
```

```
        acc = data(1, 2:4)';
        if ~any(isnan(acc))  % Acc measurements are available.
            % If the 2-norm for acc measurement is to large/small, skip
            % update step
            if  norm(acc)<9.81*1.25 && norm(acc)>9.81*0.75
                [x, P] = mu_g(x, P, acc, Ra, g0); % Update state estimate
                [x, P] = mu_normalizeQ(x,P); % Normalize state vector
                accOut = 0;
            else
                accOut = 1;
            end
        end

        mag = data(1, 8:10)';
        if ~any(isnan(mag))  % Mag measurements are available.
            % AR-filter to account for that the magnitude of m0 might drift
            Lk = (1-alpha)*Lk + alpha*norm(mag);

            % If magnitude of measurement is too large, skip update step
            if 35<Lk && Lk<55 % Thresholds for magnetic field
                [x, P] = mu_m(x, P, mag, m0, Rm); % Update state estimate
                [x, P] = mu_normalizeQ(x,P); % Normalize state vector
                magOut = 0;
            else
                magOut = 1;
            end
        end

        orientation = data(1, 18:21)';  % Google's orientation estimate.
        % Visualize result
        if rem(counter, 10) == 0
            setOrientation(ownView, x(1:4));
            ownView.setAccDist(accOut);
            ownView.setMagDist(magOut);
            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
```