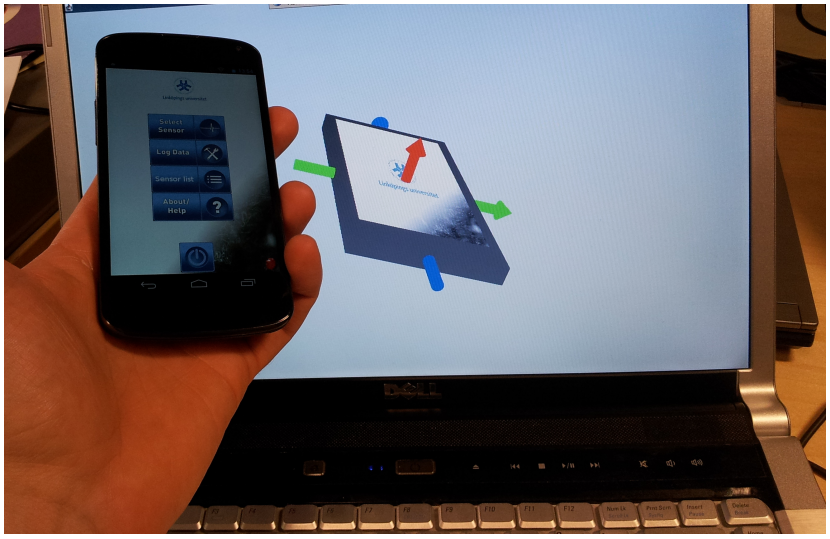
Statistical Sensor Fusion — Lab 2

Orientation Estimation using Smartphone Sensors

This version: May 2014



REGLERTEKNIK

AUTOMATIC CONTROL

LINKÖPING

Name:	
P-number:	
Date:	
Passed:	

1 Introduction

Navigation is one of the first and one of the most important applications of sensor fusion. At the same time, it is quite challenging both from a complexity and a numerical point of view. The core in any navigation system is the orientation filter that integrates inertial information from gyroscopes and accelerometers with magnetometer measurements and other supporting sensors that relate to the orientation of the platform with respect to the world.

This laboration aims at providing theoretical insights into the design and implementation of such filters, and also to give practical experience in issues such as tuning, and disturbance rejection and sensitivity.

2 The Sensor Fusion Android app

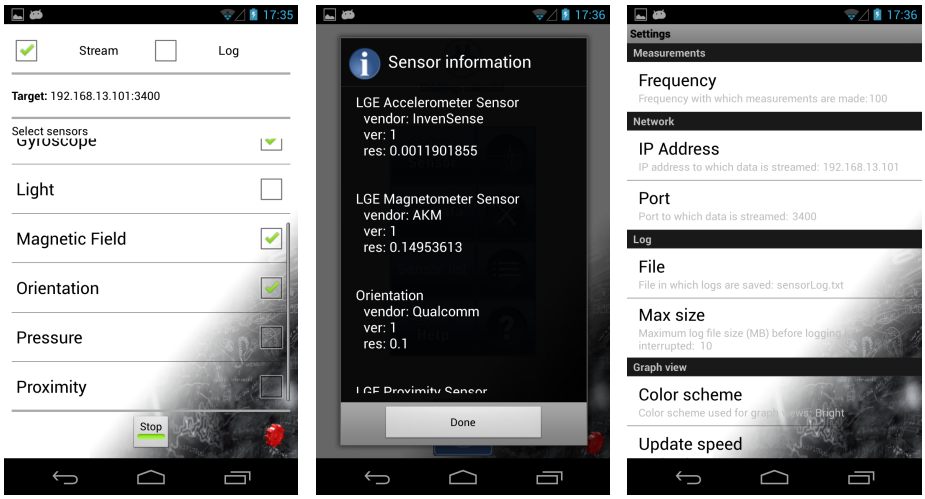
During the lab session you will work with data that you collect online with an Android smartphone (technically any Android device with suitable sensors that can be connected to Internet) using an app that streams the measurements from the phone to a PC, where it can be accessed in for example Matlab. The application is available for free from Google Play (`http://goo.gl/0qNyU`) under the name *Sensor Fusion*.

First time launching the Sensor Fusion app you end up in its home screen (Figure 1(a)), from which all the app's basic functionality can be reached.

Select Sensor Clicking "Select Sensor" yields a new view similar to the one in Figure 1(b). Exactly what exact sensors show up depends on what

(a) Home screen. (b) Sensor selection screen. (c) Graph of gyroscope data.

Figure 1: Screen shots from the Sensor Fusion app.



(a) Stream view. (b) Sensor information. (c) Settings screen.

Figure 2: Screen shots from the Sensor Fusion app.

sensors are available in your phone, this varies between different brands and models. When selecting a sensor, sensor readings from it are visualized. Figure 1(c) shows the result from clicking the "Gyroscope" button. The gyroscope measurements are visualized as three curves in a graph, together with the norm of the measurements.

Log Data Clicking "Log Data" brings up a view (Figure 2(a)) from which it is possible to stream data over internet or save it to a file on the device to be read out and processed at a later time. What sensor data is collected is determined by checking the respective check box in the sensor list. The check boxes "Stream" and "Log" determine if the collected measurements should be streamed over internet and/or saved to a log file, respectively. The details about where to stream the data and the log file are displayed in the field below and can be changed by clicking the red gear at the bottom right of the screen. Data collection is started and stopped using the button at the bottom of the page. (Note that before starting to stream data a receiver must be available, or the connection will be automatically shut down.)

Sensor list Clicking "Sensor list" brings up a dialog (Figure 2(b)) enumerating all sensory measurements that are available on request by an Android app. Note that some of the listed sensors are physical sensors, *e.g.*, accelerometers and magnetometers; whereas others are virtual sensors (the measurements are obtained from fusing other sensors), *e.g.*, orientation and gravity. (The orientation sensor will be used to benchmark the orientation filter you construct in this lab.)

About/Help This displays some information about the app.

Clicking the red gear in any view where it is present brings up the preferences view (Figure 2(c)). Changes made in this view are stored in the device. The most important fields are the Measurement>Frequency which determine the measurement rate (default: 100 Hz); the Network settings which are needed to determine where data is streamed; and the Log settings which decide where the log file is created.

3 Summary of Relevant Theory

This section provides relevant background theory needed to perform this lab. The presentation here could be considered complimentary to the description in Chapter 13 of the textbook, and can in many cases replace it for the purposes of this lab.

3.1 Representing Rotations using Quaternions

In this exercise two inertial frames are considered:

- The *world frame*, $\mathcal{W}$, which is fixed to the earth, is aligned with its x-axis pointing east, its y-axis pointing north, and its z-axis pointing up. This way an orthogonal right-hand system is formed, sometimes denoted an ENU-system (east-north-up-system).
- The *sensor frame*, $\mathcal{S}$, is centered in the smartphone and moves with it. Ideally it would be centered in the accelerometer to avoid measuring accelerations due to pure rotations, but as the exact location of it is unknown, the frame is considered positioned in the center of the phone. Laying with the screen face up the x-axis points right, the y-axis points forward, and the z-axis points up. This is the same coordinate system used in both the Android and the iOS APIs for writing apps.

See Figure 3 for an illustration of how the world frame, $\mathcal{W}$, and the sensor frame, $\mathcal{S}$, relate to each other.

The world and sensor systems are related via a linear transformation, which can be described mathematically as

$$p^{\mathcal{W}} = R^{\mathcal{W}/\mathcal{S}} p^{\mathcal{S}} + t^{\mathcal{W}/\mathcal{S}}, \tag{1}$$

where $p^{\mathcal{S}}$ is a point expressed in the sensor frame and $p^{\mathcal{W}}$ is the same point expressed in the world frame. The relative rotation between the two frames is given by $R^{\mathcal{W}/\mathcal{S}}$, the rotation aligns the world frame with the sensor frame. The translation between centers of the coordinate frames is given by $t^{\mathcal{W}/\mathcal{S}}$; however, as only the rotation is of interest in this lab, $t^{\mathcal{W}/\mathcal{S}}$ will not be considered further.

The focus of this lab is to estimate the rotation $R^{\mathcal{W}/\mathcal{S}}$ based on the sensor measurements available from a smartphone. The rotation describes how to

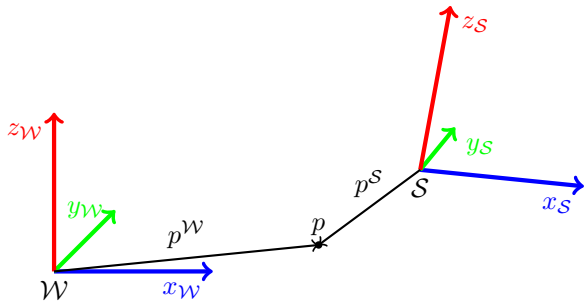


Figure 3: Illustration of the two inertial frames used here; the world fixed frame, $\mathcal{W}$, and the sensor (phone) fixed frame, $\mathcal{S}$.

rotate points in the sensor frame, $\mathcal{S}$, to describe them in the world frame, $\mathcal{W}$. It can also be interpretation as the rotation that is applied to the world frame to align it with the sensor system.

A rotation can be described in many ways. One way is to use a rotation matrix. This is a description most of us are familiar with and know well how to manipulate. However, it uses 9 values to represent 3 degrees of freedom. Hence, it is not a minimal representation, which makes estimating the rotation matrix difficult. Another alternative is to use Euler angles, *i.e.*, a representation in terms of three consecutive rotations around predefined axes. Even though represented by three values, Euler angles are not unique — several combinations of rotations result in the same final rotation — the representation has discontinuities in the parameters, and suffers from the gimbal lock effect. These are all factors that make the Euler angles unsuitable for estimation purposes. A third alternative is to use a unit length quaternion representation. This representation suffers less from the problems of the two other representations and is therefore a popular choice for orientation estimation.

The quaternion representation of a rotation can be interpreted as an axis angle representation of the rotation according to

$$q = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\frac{1}{2}\alpha) \\ \sin(\frac{1}{2}\alpha) \begin{pmatrix} \hat{v}_x \\ \hat{v}_y \\ \hat{v}_z \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \cos(\frac{1}{2}\alpha) \\ \sin(\frac{1}{2}\alpha)\hat{v} \end{pmatrix},$$

where α is a positive rotation around the axis of rotation $\hat{v}$, assumed to be of unit length, *i.e.*, $\|\hat{v}\| = 1$. Note, this interpretation clearly shows that the quaternion representation is in fact not unique, q and $-q$ represent the same rotation. (To show this is a straightforward application of trigonometric identities on the axis angle representation.) However, this is much less of a problem than in the case with Euler angles. In most cases, $q_0 \geq 0$ is assumed and enforced to get an unambiguous representation.

The following mathematical relations are useful when working with orientation estimation using the quaternion representation:

- The formula to convert from a unit length quaternion to a rotation matrix,

$$R = Q(q) = \begin{pmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 2q_0^2 - 1 + 2q_3^2 \end{pmatrix}. \tag{2a}$$

  This is implemented in `Qq(q)` in Appendix A.2.3. Note how all occurrences of q_i are in pairs, hence, changing the sign on all quaternion components does not change the resulting rotation matrix $Q(q) = Q(-q)$, as noted previously.

  The reverse operation is given by

$$q = Q^{-1}(R) = \begin{pmatrix} t/2 \\ (R_{23} - R_{32})/(2t) \\ (R_{31} - R_{13})/(2t) \\ (R_{12} - R_{21})/(2t) \end{pmatrix}, \tag{2b}$$

  where $t = \sqrt{R_{11} + R_{22} + R_{33} + 1}$. A certain degree of caution must be applied to ensure numerical stability performing this conversion.

- The angle, Δ, between two unit length quaternions q^0 and $\hat{q}$ can be calculated as

$$\Delta = 2 \arccos\left(\left| \sum_{i=0}^{3} q_i^0 \hat{q}_i \right| \right). \tag{2c}$$

  This way Δ describes the difference between $\hat{q}$ and q^0 as the angle $\hat{q}$ must be turned to obtain q^0.

- The time derivative of a quaternion, in terms of angular velocities, ω, (where the quaternion represents $R^{\mathcal{W}/\mathcal{S}}$ and ω is given in the sensor frame) can be shown to be

$$\dot{q} = \frac{1}{2} S(\omega) q = \frac{1}{2} \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix} q \tag{2d}$$

$$= \frac{1}{2} \bar{S}(q) \omega = \frac{1}{2} \begin{pmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}. \tag{2e}$$

  The functions $S(\omega)$ and $\bar{S}(q)$ are implemented in `Somega(omega)` and `Sq(q)`, in Appendices A.2.5 and A.2.4, respectively.

- Differentiating $Q(q)$ with respect to q is a bit more tricky as the result is a three-dimensional tensor. Differentiating with respect to one component

at the time yields:

$$\frac{dQ(q)}{dq_0} = 2\begin{pmatrix} 2q_0 & -q_3 & q_2 \\ q_3 & 2q_0 & -q_1 \\ -q_2 & q_1 & 2q_0 \end{pmatrix}, \tag{2f}$$

$$\frac{dQ(q)}{dq_1} = 2\begin{pmatrix} 2q_1 & q_2 & q_3 \\ q_2 & 0 & -q_0 \\ q_3 & q_0 & 0 \end{pmatrix}, \tag{2g}$$

$$\frac{dQ(q)}{dq_2} = 2\begin{pmatrix} 0 & q_1 & q_0 \\ q_1 & 2q_2 & q_3 \\ -q_0 & q_3 & 0 \end{pmatrix}, \tag{2h}$$

$$\frac{dQ(q)}{dq_3} = 2\begin{pmatrix} 0 & -q_0 & q_1 \\ q_0 & 0 & q_2 \\ q_1 & q_2 & 2q_3 \end{pmatrix}. \tag{2i}$$

The quaternion derivative is implemented in `dQqdq(q)`, in Appendix A.2.6.

- The discrete time update of a quaternion based on given angular velocities, ω_k, and process noise, w_k in (3), can be approximated with

$$\begin{aligned} q_{k+1} &= e^{\frac{1}{2}S(\omega_k+w_k)T}q_k \\ &= \left(\cos\left(\tfrac{\|\omega_k+w_k\|T}{2}\right)I + \tfrac{T}{2}\cdot\frac{\sin\left(\frac{\|\omega_k+w_k\|T}{2}\right)}{\frac{\|\omega_k+w_k\|T}{2}}S(\omega_k+w_k)\right)q_k \qquad (2j) \\ &\approx \left(I + \tfrac{1}{2}S(\omega_k)T\right)q_k + \tfrac{T}{2}\bar{S}(q_k)w_k. \qquad (2k) \end{aligned}$$

3.2 Sensor Fusion

The basic sensor fusion model is

$$x_{k+1} = f(x_k, u_k, w_k), \tag{3a}$$

$$y_k = h(x_k, u_k, e_k), \tag{3b}$$

where x_k is the state at time k, u_k known (or measured) input to the system, w_k is process noise, y_k is the measurements (or system output), and e_k is the measurement noise. Here, the goal is to estimate the orientation, represented by the quaternion q_k. The sensors provide 3D measurements of:

- Angular rates ω_k, denoted y_k^ω when used as measurements, at sampling rate f_s^ω.
- Accelerations a_k, denoted y_k^a when used as measurements, at sampling rate f_s^a.
- Magnetic field m_k, denoted y_k^m when used as measurement, at sampling rate f_s^m.

When designing a filter to estimate the phone's orientation there are several things to take into consideration: how to divide the measurements in input and output, which sensor biases, if any, to estimate, and how to handle asynchronous measurements.

There are many ways to divide the measured quantities in inputs u_k and outputs y_k in (3). For clarity, everything that is measured is referred to as measurements, and can be used as inputs or outputs depending on the chosen filter structure. The main structures are as follows:

1. The simplest model is to let

$$x_k = q_k, \qquad u_k = \omega_k, \qquad y_k = \begin{pmatrix} y_k^{a,T} & y_k^{m,T} \end{pmatrix}^T, \tag{4a}$$

   and not to attempt any bias estimation.

2. The most complex model is to let

$$x_k = \begin{pmatrix} q_k^T & \omega_k^T & b_k^{\omega,T} & b_k^{a,T} \end{pmatrix}^T, \quad y_k = \begin{pmatrix} y_k^{\omega,T} & y_k^{a,T} & y_k^{m,T} \end{pmatrix}^T, \tag{4b}$$

   and estimate both the accelerometer bias, b^a, and the gyroscope bias, b^ω.

3. An intermediate model is

$$x_k = \begin{pmatrix} q_k^T & b_k^{\omega,T} & b_k^{a,T} \end{pmatrix}^T, \quad u_k = \omega_k, \quad y_k = \begin{pmatrix} y_k^{a,T} & y_k^{m,T} \end{pmatrix}^T, \tag{4c}$$

   where the gyroscope measurement is considered as input, but the biases are estimated.

4. Since the accelerometer measures both the specific force F_k (that accelerates the platform) and the gravity g^0, another alternative is to include position and velocity in the state, and to let acceleration either be a state or an output. That is, to also estimate $t^{\mathcal{W}/\mathcal{S}}$. This is a much harder problem which needs some sort of absolute measurement to be observable, or the estimate will quickly start to drift.

A more philosophical question is what the difference of an input and output is from a filtering perspective. There are many explanations to this degree of freedom. One specific aspect here is the following:

- If the angular rates are integrated as inputs in the time update, the motion affects the mean of the state estimate directly.

- If the angular rates are used in the measurement update, this new information is weighted together with the prior in a Bayesian fashion. The mean of the orientation state does not take the full step according to the measurement. There will also be a delay in information, because it takes one time update until the derivative affects the angular state.

The conclusion is that we prefer to consider $u_k = \omega_k$ as an input. Both a_k and m_k are subject to disturbances, and outlier rejection is most naturally implemented in the measurement update. This is another reason why we advocate the first approach above.

A remaining question is whether an attempt should be done to estimate the biases in the sensors, b^ω and b^a above.

The multi-rate, or asynchronous, measurement aspect is an issue in practice if all sensors have different rates. Consider case 1 above. If the sample rate f_s^ω is an integer multiple of f_s^a and f_s^m, then the measurement update can be performed at regular cycles after the time update. In the general case, the piece-wise constant assumption has to be used, and the time update is matched to the next output observation.

4 Preparations

Given the discussion above, use a state-space model with the rotation $R^{\mathcal{W}/\mathcal{S}}$ represented as a quaternion as state vector.

4.1 Dynamic Equation

The dynamic model is then given by (2j). Before the lab session, write a Matlab function `[x, P] = tu_qw(x, P, omega, T, Rw)` that implements the time update function, where `omega` is the measured angular rate, `T` the time since the last measurement, and `Rw` the process noise covariance matrix. For your convenience, the functions $\bar{S}(q)$ and $S(\omega)$ are available in App. A.2.4 and A.2.5, respectively.

Furthermore, write a similar function to handle the time update in the case that there is no angular rate measurement available.

4.2 Acceleration Measurement Equation

The measurements from the accelerometer can be modeled using the relation

$$y_k^a = Q^T(q_k)(g^0 + F_k) + e_k^a, \tag{5}$$

where no specific force is assumed present (*i.e.*, $F_k = 0$), g^0 is the nominal gravity vector, and e_k^a is the measurement noise.

Before the lab session, prepare a Matlab function `[x, P] = mu_g(x, P, yacc, Ra, g0)` that implements the accelerometer measurement update, where `yacc` is a shorthand for y_k^a and `Ra` is the measurement noise covariance matrix. You can let g^0 be a constant in your implementation. The functions to compute $Q(q)$ and $dQ(q)/dq$ are available in App. A.2.3 and A.2.6, respectively.

4.3 Magnetic Field Measurement Equation

Before the lab session prepare a Matlab function `[x, P] = mu_m(x, P, ymag, Rm, m0)` that implements a magnetometer measurement update, where `ymag` is a shorthand for y_k^m.

Hint: The magnetometer measures the earth magnetic field (possibly including disturbances) in sensor coordinates,

$$y_k^m = Q^T(q_k)m^0 + e_k^m, \tag{6}$$

where m^0 is the earth magnetic field in world coordinates. By definition there should be no magnetic field component in the west-east direction. Hence, the nominal magnetic field can be derived from magnetic field measurements as the phone lies horizontally without knowledge of the direction of north using the relation

$$m^0 = \begin{pmatrix} 0 & \sqrt{m_x^2 + m_y^2} & m_z \end{pmatrix}^T, \tag{7}$$

where m is the measured magnetic field.

4.4 Quaternion Normalization

In order for the quaternion representation of the orientation to work, it is essential that the quaternion is of unit length. The extended Kalman filter does not automatically maintain this quaternion property. Therefore, before the lab session, figure out a way to ensure the quaternion keeps unit length.

5 Exercises

The following exercises should be completed during the lab session. The passages prefixed by "**Check-point:**" should be presented to a lab assistant to pass the lab.

Consider the structure (4a) and make use of the code skeleton in Appendix A.1.

1. **Connect the phone with your lab computer:**
   - Download all the supporting Matlab files and the `sensorfusion.jar` file from: `http://goo.gl/DGBSOC`. Extract the files in the zip-archive to your lab folder.
   - Run the `startup` function to initialize the environment correctly. Note, this must be done each time Matlab is restarted.
   - Connect the smartphone to eduroam using the instructions in Appendix B.
   - Determine the IP of your lab computer. One way to do this is to run `showIP` in Matlab.

- Enter the lab computer's IP into the app's configuration view (see Figure 2(c).) At the same time, make sure the port is set to 3400 and that the measurement frequency is 100 Hz.
- Always start your Matlab program before initiating the streaming from the smartphone, it acts as a server to which the client on the phone tries to connect to.

2. **Get to know your data:** Spend a few minutes to play around with the app and the sensors in the "Select Sensor" view to get an initial feeling for the data you are going to work with.

   Use `filterTemplate` to collect a few seconds of data and compute mean and variance for the accelerometer, gyroscope, and magnetometer. Also, analyze the measurements to see if the noise is Gaussian and if it has any trends. What do the results tell you? Your results should be used to tune the filter.

   Hint: Unavailable measurements are marked with a NaN (not a number) value. Make sure to take this into consideration when computing means etc. The function `isnan(x)` can be used to find NaN values; the average acceleration can be computed as:
   `mean(meas.acc(:, ~any(isnan(meas.acc), 1)), 2)`

Check-point: Be prepared to show the lab assistant:

- A histogram of the measurements for each sensor and axis.
- A plot of the signals over time. If there are trends figure out a way to deal with these.
- The determined average acceleration vector, angular velocity vector, and magnetic field, and their respective covariance matrices.

3. **Add the EKF time update step:** Start to implement an EKF to estimate the orientation by adding the *time update*. (Make a copy of `filterTemplate.m` and add the time update step you wrote in Section 4.1 as part of your lab preparations.) Use the previously estimated mean and variance as input to the filter and use it to compensate for bias and noise levels.

   You should now have a very reactive estimate with no absolute orientation. What happens if you start the filter with the phone on the side instead of laying face up on the desk? Why? Shake the phone and explain what happens.

Check-point: Show the filter running to the lab assistant, and be prepared to present your findings from the experiments indicated above.

4. **Add the EKF accelerometer measurement update step:** Add the *accelerometer update* to the EKF (use `mu_g(x, P, yacc, Ra)` you wrote in Section 4.2 preparing for the lab). Now, the orientation should be correct up to a rotation in the horizontal plane. Test the sensitivity to

specific forces (body accelerations) with different experiments. As an example, slide the device quickly back and forth on the horizontal surface of a table.

5. **Add accelerometer outlier rejection:** Add a simple *outlier rejection* algorithm for a_k, based on the assumption that $\|a_k\| \approx g = 9.81$ without specific force. Use the orientation view's `setAccDist` function to indicate when the accelerometer measurements are considered disturbed/outliers.

   Repeat the same experiments as in the previous two steps.

Check-point: Show the filter running to the assistant, and be ready to explain what you observed when shaking and sliding your phone on the desk with and without outlier rejection.

6. **Add the EKF magnetometer measurement update step:** Implement a *magnetometer update*, based on your preparations in Section 4.3. You should now have a orientation filter that behaves much like the one implemented in the phone. What happens if you introduce a magnetic disturbance? (The power supply in the monitor can provide a suitable disturbance.)

7. **Add magnetometer outlier rejection:** Try to come up with a way to perform *outlier rejection* for magnetic disturbances, *e.g.*, by estimating the strength of the magnetic field and reject measurements when the field strength differs too much from what you expect. Implement the outlier rejection, and evaluate if it helps. Use the orientation view's `setMagDist` function to indicate when the magnetic field measurements are considered disturbed/outliers. What assumptions do you rely on? When are they reasonable? What happens now when you introduce a magnetic disturbance?

   Use the `q2euler` function (Appendix A.2.2) and plot the Euler angles of both your orientation filter and the built in filter in the phone.

Check-point: Show the filter running to the assistant, and be ready to explain what you observed when using the filter with and without a magnetic disturbance.

8. **Test your filter without gyroscope measurements:** Turn off the gyroscope measurements and evaluate the filter; slide the phone on the desk, shake it, and introduce magnetic disturbances.

Check-point: Show the filter running without gyroscope measurements to the assistant

9. **If you are interested and have time:**

   - Compare the results difference between using (2k) and the exact form (2j).
   - Evaluate the remaining combinations of measurements in the filter: only acceleromter, only magnetometer, gyroscope and magnetomter.

10. **Wipe your account details from the phone:** Once you have passed the lab, wipe your login details for eduroam from the phone. Got to the network settings screen again, press the eduroam entry until you get the option to "Forget network" and do so.

A Provided Code

The following functionality is provided in the zip-archive http://goo.gl/DGBSOC. Make sure to run the included `startup` function to initialize the package before doing anything else. Without this step it is not possible to establish a connection with the phone. When using `filterTemplate` always start the Matlab program before initializing the streaming from the phone as the Matlab program acts as a server for the data from the phone

A.1 Code Skeleton

```
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.

  %% Setup necessary infrastructure
  import('com.liu.sensordata.*');  % Used to receive data.

  %% Filter settings
  t0 = [];  % Initial time (initialize on first data received)
  nx = 4;
  % Add your filter settings here.

  % 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;
        end
        t = data(1)/1000;  % Extract current time

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

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

        mag = data(1, 8:10)';
        if ~any(isnan(mag))  % Mag measurements are available.
          % Do something
        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))
 95           if isempty(googleView)
                subplot(1, 2, 2);
                % Used for visualization.
                googleView = OrientationView('Google filter', gca);
              end
100           setOrientation(googleView, orientation);
              title(googleView, 'GOOGLE', 'FontSize', 16);
            end
          end
          counter = counter + 1;
105
          % Save estimates
          xhat.x(:, end+1) = x;
          xhat.P(:, :, end+1) = P;
          xhat.t(end+1) = t - t0;
110
          meas.t(end+1) = t - t0;
          meas.acc(:, end+1) = acc;
          meas.gyr(:, end+1) = gyr;
          meas.mag(:, end+1) = mag;
115       meas.orient(:, end+1) = orientation;
        end
      catch e
        fprintf(['Unsuccessful connecting to client!\n' ...
          'Make sure to start streaming from the phone *after*'...
120             'running this function!']);
      end
    end
```

A.2 Utility Functions

A.2.1 Normalize quaternion

```
1   function [x, P] = mu_normalizeQ(x, P)
    % MU_NORMALIZEQ  Normalize the quaternion

      x(1:4)  = x(1:4) / norm(x(1:4));
5     x = x*sign(x(1));
    end
```

A.2.2 Convert quaternions to Euler angles

```
1   function euler = q2euler(q)
    % Q2EULER  Convert quaternions to Euler angles

      euler = zeros(3, size(q, 2));
5
      xzpwy = q(2, :).*q(4, :) + q(1, :).*q(3, :);

      IN = xzpwy+sqrt(eps)>0.5;  % Handle the north pole
      euler(1, IN) = 2*atan2(q(2, IN), q(1, IN));
10    IS = xzpwy-sqrt(eps)<0.5;  % Handle the south pole
```

```
    euler(1, IS) = -2*atan2(q(2, IS), q(1, IS));

    I = ~(IN | IS);  % Handle the default case
    euler(1, I) = atan2(-2*(q(1, I).*q(3, I) - q(1, I).*q(4, I)),...
                        1-2*(q(3, I).^2 + q(4, I).^2));
    euler(3, I) = atan2(2*(q(3, I).*q(4, I) - q(1, I).*q(2, I)),...
                        1-2*(q(2, I).^2 + q(3, I).^2));

    euler(2, :) = asin(2*xzpwy);

    euler(1, :) = rem(euler(1, :), 2*pi);
end
```

A.2.3 $Q(q)$

```
function Q=Qq(q)
% The matrix Q(q) defined in (13.16)
    q0=q(1);    q1=q(2);    q2=q(3);    q3=q(4);
    Q = [2*(q0^2+q1^2) - 1   2*(q1*q2-q0*q3)     2*(q1*q3+q0*q2);
         2*(q1*q2+q0*q3)     2*(q0^2+q2^2) - 1   2*(q2*q3-q0*q1);
         2*(q1*q3-q0*q2)     2*(q2*q3+q0*q1)     2*(q0^2+q3^2) - 1];
end
```

A.2.4 $\bar{S}(q)$

```
function S=Sq(q)
% The matrix S(q) defined in (13.11c)
    q0=q(1);    q1=q(2);    q2=q(3);    q3=q(4);
    S=[-q1 -q2 -q3;
        q0 -q3  q2;
        q3  q0 -q1;
       -q2  q1  q0];
end
```

A.2.5 $S(\omega)$

```
function S=Somega(w)
% The matrix S(omega) defined in (13.11b)
    wx=w(1);    wy=w(2);    wz=w(3);
    S=[ 0  -wx  -wy  -wz;
        wx    0   wz  -wy;
        wy  -wz    0   wx;
        wz   wy  -wx    0];
end
```

A.2.6 $dQ(q)/dq$

```
function [Q0, Q1, Q2, Q3] = dQqdq(q)
% The derivative of Q(q) wrt qi, i={0,1,2,3}, as implicitely defined
% in (13.20d)
    q0=q(1);    q1=q(2);    q2=q(3);    q3=q(4);
    Q0 = 2* [2*q0   -q3    q2;
               q3  2*q0   -q1;
              -q2    q1  2*q0];
    Q1 = 2* [2*q1    q2    q3;
               q2     0   -q0;
               q3    q0     0];
    Q2 = 2* [   0    q1    q0;
               q1  2*q2    q3;
              -q0    q3     0];
    Q3 = 2* [   0   -q0    q1;
               q0     0    q2;
               q1    q2  2*q3];
end
```

A.2.7 OrientationView

The class orientation view provides a way to easily visualize orientations. The class is used in the filter template.

self = OrientationView(figname) Create a OrientationView figure with a given figure name.

setOrientation(self, q, P) Set the orientation to display, if P is given it is used to visualize the uncertainty in the orientation.

title(self, str) Set the title of the view to str.

setStandStill(self, flag) Set the stand still indicator on or off.

setAccDist(self, flag) Set the acceleration disturbance indicator on or off.

setMagDist(self, flag) Set the magnetometer disturbance indicator on or off.

A.2.8 showIP()

showIP provides easy access to the IP the computer is using.

A.3 sensordata.jar

The Java package sensordata.jar provides the functionality needed to communicate with the Sensor Fusion app and to integrate the measurements in real time into for instance Matlab. The documentation of the public interface can be found here: http://goo.gl/I7KpfL.

B Configure the Phone for eduroam

To be able to stream measurement data from the Android device to the computers in the ISY computer labs, the phone needs to be connected to a LiU network, *i.e.*, *eduroam* or *Netlogon*. We recommend using eduroam. To connect via wireless network to eduroam follow these instructions:

- Obtain a eduroam password; you can either reuse a previously generate one or generate a new one:
  http://goo.gl/b3zb7
- Configure the phone to connect to eduroam. The different steps are illustrated in Figure 4.
  1. Drag down the status bar at the top.
  2. Click the configuration to bring out the settings menu.
  3. Select the wifi settings.
  4. Find eduroam in the list, and configure it with the following values:
     - EAP method: PEAP
     - Phase 2 authentification: MSCHAPV2
     - Identity: <your-liu-id>@liu.se
     - Anonymous identity: <your-liu-id>@liu.se
     - Password: <the eduroam password generated above>

If your are trying to connect your own phone, things might look slightly different depending on the make of the device and the version of Android it is running. In this cases, the following link can be useful: http://goo.gl/eDm1B.

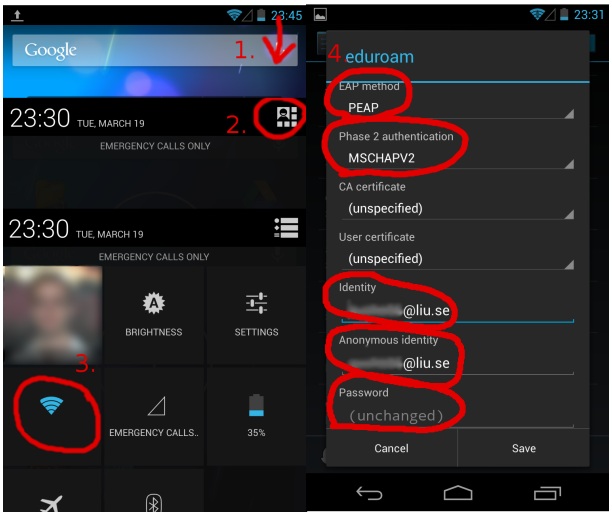


Figure 4: Illustration of the basic steps needed to configure the device for the eduroam network.