Sensor Fusion Project 2020
Group 43

Sara Roth, Chalmers ID: sararot
Jonas Lindberg, Chalmers ID: jlindbe

18 Maj 2020

Task 1

In this problem the state vector used is a unit length quaternion representation. We do not use Euler angles is because Euler angles are not unique, so several combinations of rotations result in the same final rotation. So Euler angler suffers from the gimballock effect. That is losing a degree of freedom in a three-dimensional system, e.g. when the phone stands on it's side yaw rotation is also rotation around the x-/y-axis. This makes Euler angles unsuitable for estimation purposes. The rotation matrix is neither a good choice since it is not a minimal representation. That makes estimating the rotation matrix difficult. The quaternions suffer less from the problems of the rotation matrix and the Euler angles and is therefore a popular choice for orientation estimation.
In this project the gyroscope measurements is used as inputs, i.e., $u_k = \omega_k$. One reason why this is a good choice is that gyroscope measurements provided by smartphones are normally rather accurate. If the gyroscope is not accurate, then this would be a bad choice. The accelerometer measures both the specific force f_k^a, that accelerates the platform, and the gravity g_0. If the position and velocity were included in the state, and acceleration would either be a state or an output. That would be to also estimate $t^{W/S}$. This is a much harder problem which needs some sort of absolute measurement to be observable, or the estimate will quickly start to drift. If the measurements is not accurate, then it would be a good idea to include the angular velocity in the inputs.

Task 2

Measurement signals

Figure 1 is generated when the phone is lying still on a table with the screen facing upwards.

There seems to be no drift in the measurements since the gyro data remains on the same level through the whole time lap. It's also clear that the covariance on the measurements is quite small, since the amplitude in the gyro measurements is small.

The accelerometer shows $9.81 m/s^2$ in z which is the normal force from the table on the phone due to the gravity which is pointing down. Since the gravity is pointing down and the normal force up, we get a positive measurement.

As was stated in the assignment, the magnetometer was first calibrated by rotating the phone.

Overall, the sensors seem stable over the time period sensor data was collected in sense that there is not apparent drift and no bias that needs to be taken care of. That indicates that the measurements can be trusted since the assumed measurement noise covariance is also small.

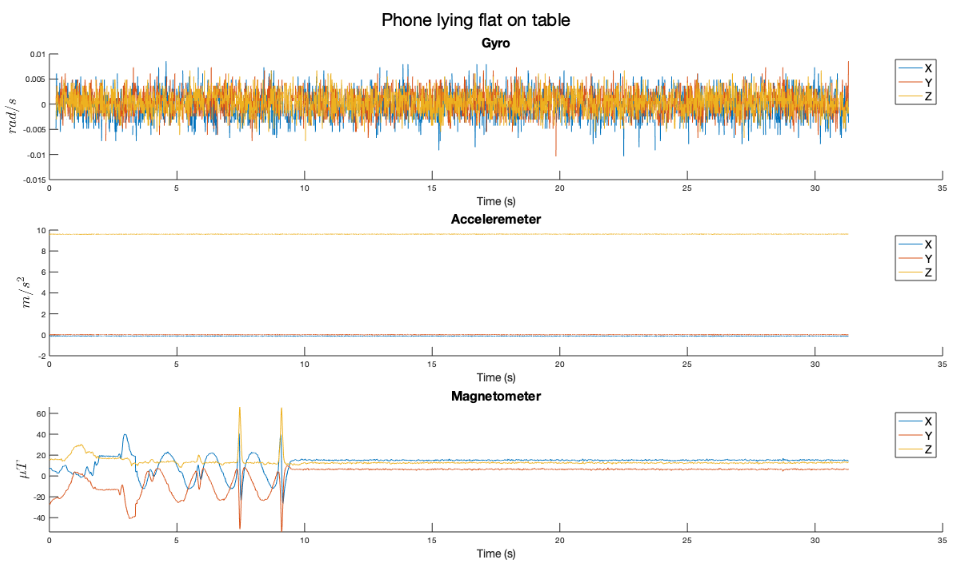

Figure 1: Measurement signals when the phone is lying still on the table with screen pointing upwards

Histograms of the measurement signals

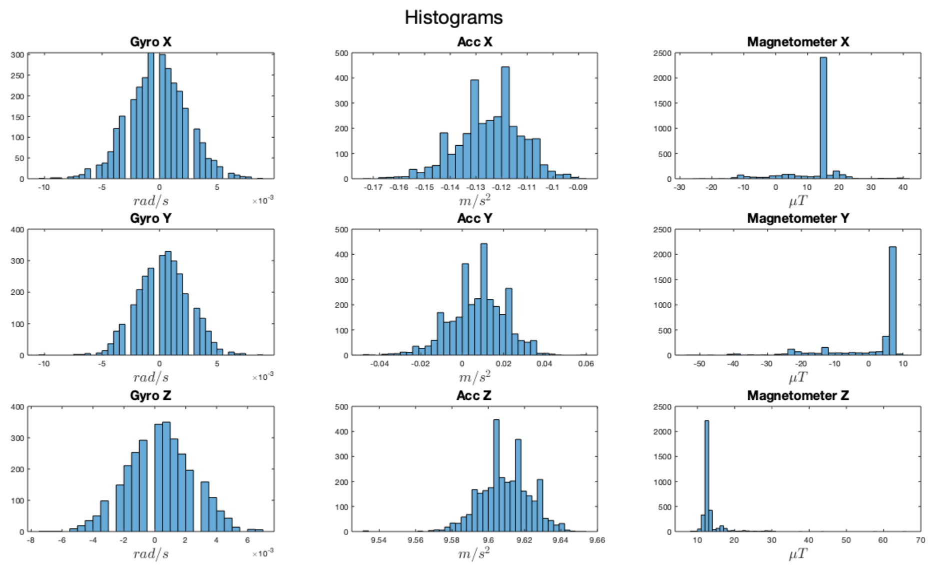

Figure 2: Histograms of the measurements for some sensors and axes

Mean and covariances for the acceleration vector, angular velocity vector, and magnetic field

Below is the covariance. The variance for each sensor would be the diagonal values of the covariance. The covariance matrices are used for tuning the filter and was hence computed.

$$\mu_{\text{Gyr}} = \begin{bmatrix} -0.1678 \times 10^{-3} \\ 0.3465 \times 10^{-3} \\ 0.2961 \times 10^{-3} \end{bmatrix} \qquad R_{\text{Gyr}} = \begin{bmatrix} 0.1007 & 0.0012 & 0.0007 \\ 0.0012 & 0.0530 & 0.0026 \\ 0.0007 & 0.0026 & 0.0467 \end{bmatrix} \times 10^{-4}$$

$$\mu_{\text{Acc}} = \begin{bmatrix} -0.1247 \\ 0.0077 \\ 9.6096 \end{bmatrix} \quad R_{\text{Acc}} = \begin{bmatrix} 0.1545 & 0.0753 & -0.0842 \\ 0.0753 & 0.1792 & -0.0840 \\ -0.0842 & -0.0840 & 0.2263 \end{bmatrix} \times 10^{-3}$$

$$\mu_{\text{Mag}} = \begin{bmatrix} 14.4197 \\ 4.6618 \\ 12.9508 \end{bmatrix} \quad R_{\text{Mag}} = \begin{bmatrix} 20.3254 & 8.4193 & -1.4920 \\ 8.4193 & 42.0064 & -6.4142 \\ -1.4920 & -6.4142 & 6.6191 \end{bmatrix}$$

Design the EKF time update step

Task 3

To derive a discrete time prediction model we make use of the continuous time model

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

Which is on the form $\dot{q} = Aq$. We assume that the process noise enters additively on $\omega(t)$, but in order to simplify the expressions we assume that $\omega(t) = \omega_{k-1}$ and $v(t) = v_{k-1}$ are piece-wise constant between the sampling times t_{k-1} and t_k. $v_k \sim \mathcal{N}(0, R_v)$. Since we know that this is piece wise constant, the matrix S will also be piece wise constant.

We solve the differential equation in (1) by applying the analytical solution for linear systems:

$$x(t+T) = \exp(AT)x(t) + (IT + \frac{AT^2}{2} + \frac{A^2T^3}{3!}...)b$$

This is found in lecture 5 slide 20. Since we don't have a term b the analytical solution reduces to:

$$x(t+T) = \exp(AT)x(t)$$

Applying the analytical solution to our model yields:

$$q(t+T) = \exp\left(\frac{1}{2}S(\omega_{k-1} + v_{k-1})T\right)q(t)$$

Applying $\exp(A) = I + A$:

$$q(t+T) = \left(I + \frac{1}{2}S(\omega_{k-1} + v_{k-1})T\right)q(t)$$

$$q_k = \left(I + \frac{1}{2}S(\omega_{k-1} + v_{k-1})T\right)q_{k-1} \tag{2}$$

We note that:

$$S(\omega_1 + \omega_2) = S(\omega_1) + S(\omega_2) \tag{3}$$

$$\bar{S}(q_1 + q_2) = S(q_1) + S(q_2) \tag{4}$$

and that we can denote

$$q_k = q(t_k) \quad \text{and} \quad q_{k-1} = q(t_{k-1})$$

We can then disctretize model (2) using (3)(4)

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

$$q_k = q_{k-1} + \frac{1}{2}S(\omega_{k-1})Tq_{k-1} + \frac{1}{2}S(v_{k-1})Tq_{k-1}$$

From the assignment we get:

$$S(v)q = \bar{S}(q)v$$

This gives:

$$q_k = q_{k-1} + \frac{1}{2}S(\omega_{k-1})q_{k-1}T + \frac{1}{2}\bar{S}(q_{k-1})v_{k-1}T$$

We use the approximation:

$$G(q_{k-1})v_{k-1} = G(\hat{q}_{k-1})v_{k-1}$$

$$q_k = \underbrace{\left(I + \frac{1}{2}S(\omega_{k-1})T\right)}_{F} q_{k-1} + \underbrace{\frac{1}{2}\bar{S}(\hat{q}_{k-1})T}_{G} v_{k-1} \tag{5}$$

The last approximation comes from the linearization used in EKF filter.

Task 4

Based on the model that was derived in the previous task, a Matlab function

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

was created. The function implements the time update function

- ω = measured angular rate
- T = time since the last measurement
- R_w = the process noise covariance matrix

If the gyroscope measurements are available we use this function to calculate the mean and covariance with the motion model

$$\text{States, x = quaternion, q: } x = F \cdot x$$

$$\text{Covariance: } P_k = F \cdot P_{k-1} \cdot F' + G \cdot R_w \cdot G$$

Where F and G is taken from equation (5)

If the gyroscope measurements are not available we use the noise covariance e_k^q and approximate the states and covariance asa random walk:

$$x = x$$

$$P_k = P_{k-1} + \underbrace{I \cdot 0.001}_{\text{Noise covaraince}}$$

The functions can be seen below:

```
function [x, P] = tu_qw(x, P, omega, T, Rw)
% tu_qw implements the time update function.
% Inputs:
% x - States quaternion
% P -
% omega - Measured angular rate
% T - Time since the last measurement
% Rw - Process noise covariance
%
% Outputs
% x - States quaternion
% P - Covariance
    F = eye(size(x,1)) + 1/2 * Somega(omega) * T;
    G = 1/2 * Sq(x) * T;

    %Predicted mean
    x = F*x;
    % We update the covariance and the process noise covariance
    P = F*P*F' + G*Rw*G';
end
```

```
function [x, P] = tu_qw_omegaNAN(x, P)
% tu_qw implements the time update function.
% Inputs:
% x - States quaternion
% P -
% omega - Measured angular rate
% T - Time since the last measurement
% Rw - Process noise covariance
%
% Outputs
% x - States quaternion
% P - Covarance

% If no measured angle rate, use random walk
x = x;
P = P + eye(size(x,1)) * 0.001;
end
```

Task 5

The filter performs well but only works if the starting position is flat on the table. If the phone starts streaming data when not fully aligned with the table the filter can't tell the position of the phone correctly. The local coordinate system isn't updated since we have no way of telling how the phone is oriented. This happens because we are only measuring the angular velocity and hence can't tell the orientation but rather the change in orientation.

Experimenting shows that there is a drift in the filtering. If the phone is put on a table after being rotated in different angles the filter estimate does not show the phone is laying fully flat. This is due to integration drift.

If the phone is shaken the filter can't handle the big changes in angular velocity and there will be an error.

Task 6

The EKF update step is taken by approximating

$$y_k = H(x_k) + r_k \approx h(\hat{x}_{k|k-1}) + h'(\hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1}) + r_{k-1}$$

The linear Kalman update equations can than be used. First calulate

$$\text{Innovation covariance: } S_k = h'(\hat{x}_{k|k-1})P_{k|k-1}h'(\hat{x}_{k|k-1})^T + R_k$$

$$\text{Gain: } K_k = P_{k|k-1}h'(\hat{x}_{k|k-1})^T S_k^{-1}$$

Then we can preform the update step and calculate the mean and the covariance

$$\text{Mean: } \hat{x}_{k|k-1} + K_k(y_k - h(\hat{x}_{k|k-1}))$$

$$\text{Covariance: } P_{k|k} = P_{k|k-1} - K_k S_k K^T$$

In our case we have:

$$h(x) = Q^T(q_k)(g^0 + f_k^a)$$

Assuming no acceleration in the world frame yields:

$$h(x) = Q^T(q_k) \times g^0$$

$$h'(x) = \frac{\partial Q^T}{\partial q_k}(q_k) \times g^0$$

Since we have four states the Jacobian will be:

$$h'(x) = \left[\frac{\partial Q^T}{\partial q_0} \times g^0 \quad \frac{\partial Q^T}{\partial q_1} \times g^0 \quad \frac{\partial Q^T}{\partial q_2} \times g^0 \quad \frac{\partial Q^T}{\partial q_3} \times g^0 \right]$$

With this, we can simply input $h(x)$ and $h'(x)$ into the EKF update step using R_a as the measurement noise covariance and y_a^k as the measurement. In our case, g^0 is the mean of the acceleration reading when the phone is lying flat on a table, from Task 2. See code below:

```matlab
function [x, P] = mu_g(x, P, yacc, Ra, g0)

    y = yacc;
    R = Ra;

    % We need hx and Hx (jacobian)
    hx = Qq(x)' * g0;

    [Q0, Q1, Q2, Q3] = dQqdq(x);
    Hx = [Q0'* g0, Q1'* g0, Q2'* g0, Q3'* g0];

    % EKF Update step from HA3

    S = Hx * P * Hx' + R;
    K = P * Hx' / S;

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

Task 7

Implementing the accelerometer filtering means that the estimate is now correct in terms of orientation. This is since we are now measuring the phones orientation relative to the earth gravitational force. The phone was slided across a table to try the filter. Ideally, this would not affect the estimate since we are assuming no acceleration in the world frame. However, experiments show that rapid movements are interpreted as rotational movement. This is since we are assuming that no other forces than earths gravity affects the sensor. Applying other forces will be interpreted as changes in the gravitional force and hence result in an estimated rotation.

Task 8

An outlier rejection algorithm was implemented for a_k. The algorithm is based on that $||a_k|| \approx g = 9.81$. when f_k^a is zero. The idea is to simply skip the updates when we think the measurement is an outlier, i.e. when an external force is acting on the sensor. The interval used in this case is

$$0.95 \times ||g^0|| < ||a_k|| < 1.05 \;\times ||g^0||$$

So measurements inside this interval is considered and the measurements outside is simply skipped. This is done to avoid the additive forces to affect the estimate. Ideally we would like to only consider measurements $||a_k|| = ||g^0||$ but since we don't have a perfect sensor this is not possible.

Task 9

As shown in task 6 the EKF update step is taken by approximating

$$y_k = H(x_k) + r_k \approx h(\hat{x}_{k|k-1}) + h'(\hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1}) + r_{k-1}$$

The linear Kalman update equations can than be used. First calulate

$$\text{Innovation covariance: } S_k = h'(\hat{x}_{k|k-1})P_{k|k-1}h'(\hat{x}_{k|k-1})^T + R_k$$

$$\text{Gain: } K_k = P_{k|k-1}h'(\hat{x}_{k|k-1})^T S_k^{-1}$$

Then we can preform the update step and calculate the mean and the covariance

$$\text{Mean: } \hat{x}_{k|k-1} + K_k(y_k - h(\hat{x}_{k|k-1}))$$

$$\text{Covariance: } P_{k|k} = P_{k|k-1} - K_k S_k K^T$$

In our case we have:

$$h(x) = Q^T(q_k)(m^0 + f_k^m)$$

Assuming that there is no other magnetic fields than the earths in the world frame yields:

$$h(x) = Q^T(q_k) \times m^0$$

$$h'(x) = \frac{\partial Q^T}{\partial q_k}(q_k) \times m^0$$

Since we have four states the Jacobian will be:

$$h'(x) = \begin{bmatrix} \frac{\partial Q^T}{\partial q_0} \times m^0 & \frac{\partial Q^T}{\partial q_1} \times m^0 & \frac{\partial Q^T}{\partial q_2} \times m^0 & \frac{\partial Q^T}{\partial q_3} \times m^0 \end{bmatrix}$$

With this, we can simply input $h(x)$ and $h'(x)$ into the EKF update step using R_m as the measurement noise covariance and y_m^k as the measurement. In our case, m^0 the mean of the magnetic field reading when the phone is lying flat on a table, from Task 2, pointing north.

$$m^0 = \begin{bmatrix} 0 & \sqrt{m_x^2 + m_y^2} & m_z \end{bmatrix}^T$$

See code below:

```
function [x, P] = mu_m(x, P, mag, m0, Rm)

    y = mag;
    R = Rm;

    % We need hx and Hx (jacobian)
    hx = Qq(x)' * m0;

    [Q0, Q1, Q2, Q3] = dQqdq(x);
    Hx = [Q0'* m0, Q1'* m0, Q2'* m0, Q3'* m0];

    % EKF Update step from HA3

    S = Hx * P * Hx' + R;
    K = P * Hx' / S;

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

Task 10

Implementing the magnetometer filter to the estimate makes us able to better track the starting position in rotation around the z-axis relative to the position the phone was in when the mean values were collected.

As was the case in Task 7, applying additional forces will cause the estimate to go bad. In this case we apply an additional magnetic field when we have stated that no other magnetic fields than the earths is present. This will indeed cause disturbances as the filter will interpret the change in magnetic field as rotation of the device.

Task 11

Similar to the accelerometer update, we have based our update on the assumption that the magnetic disturbances are zero, i.e. that $f_k^m = 0$. To be able to handle disturbances we implement a outlier rejection for magnetic disturbances by estimating the strength of the magnetic field and reject measurements when the field strength differs too much from whats expected. This assumption is reasonable as long as the phone is in one constant magnetic field. The assumptions fall when moving into a different magnetic field or when applying a magnetic disturbance.
The filter used to estimate the magnitude is a AR(1)-filter of type

$$L^k = (1-\alpha)L^{k-1} + \alpha||m_k||$$

Where α is a small positive number and m_k is the magnetometer reading at time instance k. The bounds for the AR(1)-filter was found through some trial and error.

The AR(1) filter is an autoregressive filter. With this filter, we will accept small changes in magnetic field strength. When the phone enters an area with additional magnetic fields, the change in strength will be too big for the filter to pass through. After a while, if the magnetic fields are stable the filter will pass. This can be viewed as a low-pass filter of the magnetic field strength. With this outlier rejection we can move fairly well through different magnetic disturbances.

When all the parts of the filter is implemented tests are performed to evaluate the performance. The filter is compared with Googles own estimate and the result can be see below in Figure 3.

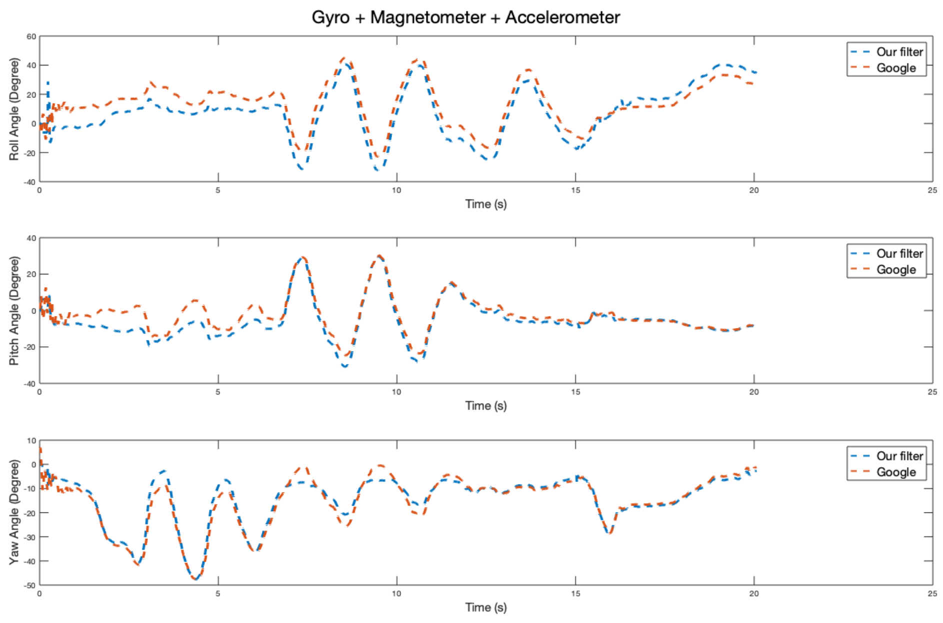

Figure 3: Comparison between developed filter and Googles estimate

Viewing Googles estimate as a true state it can be noted that the filter seems to perform very good and is almost equal to the Google estimate.

Task 12

Gyro and Accelerometer

In the case where the magnetometer is ignored the filter looses one of its orientation references. This means the relative movements estimates might be correct but offset. In Figure 4 the comparison can be seen when the phone starts streaming data in a correct position (that is flat on surface pointing north).

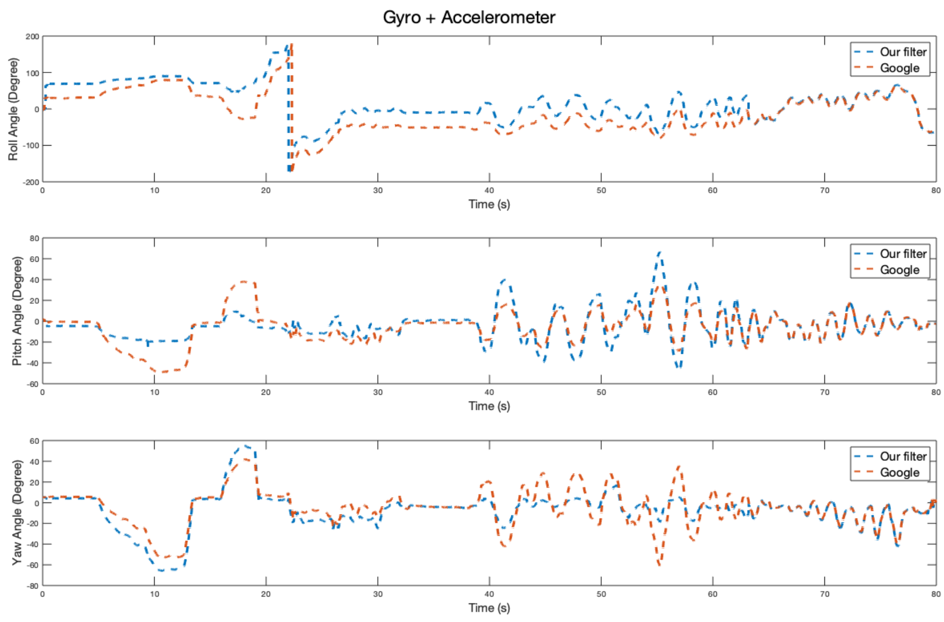

Figure 4: Comparison between developed filter and Googles estimate using gyroscope and accelerometer

In Figure 5 the phone starts disoriented from the "correct" position and the offset behaviour is clearly visible. The estimated rotations are correct but has an offset.

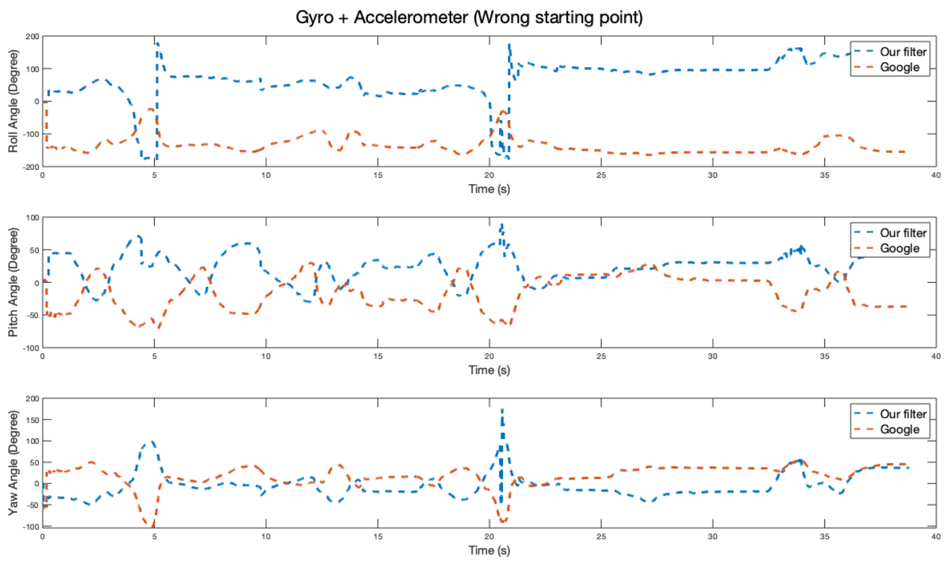

Figure 5: Comparison between developed filter and Googles estimate using gyroscope and accelerometer with wrong starting point

Gyro and Magnetometer

When ignoring the accelerometer data we loose the gravitational reference and the orientation will depend on the magnetometer, see figure 6. However, since the magnetometer has larger error and its reading drifts over time the orientation estimate will not be as good. We have a large error covariance on the magnetometer and this, together with the drift in magnetic field strength, means that the filter will take long to stabilize.

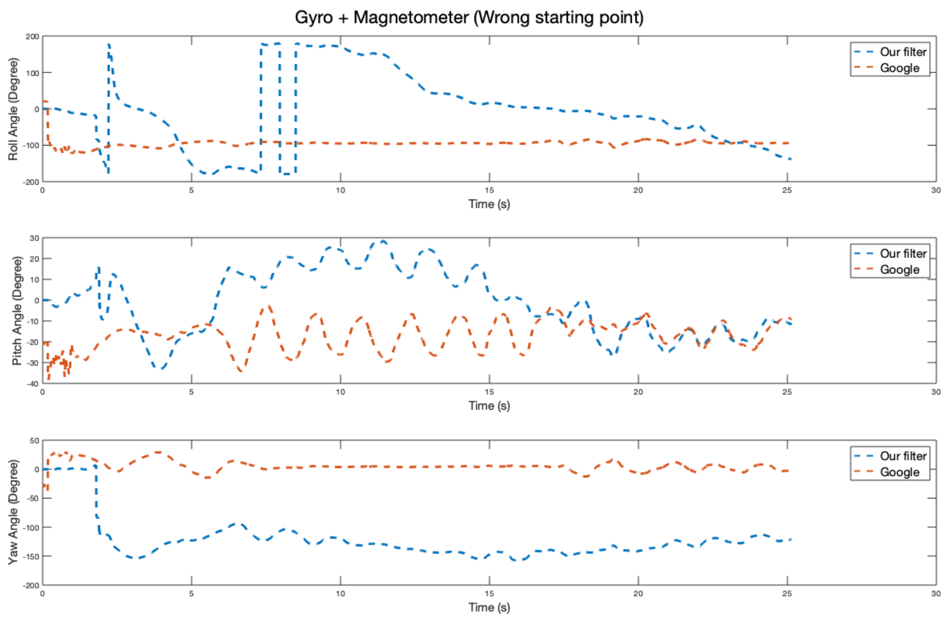

Figure 6: Comparison between developed filter and Googles estimate using gyroscope and magnetometer

Magnetometer and Accelerometer

By ignoring the gyro data we model the angular velocity as random walk, see figure 7. Since we have a low covariance in the random walk step this approximation will work when the phone is moved around slowly.

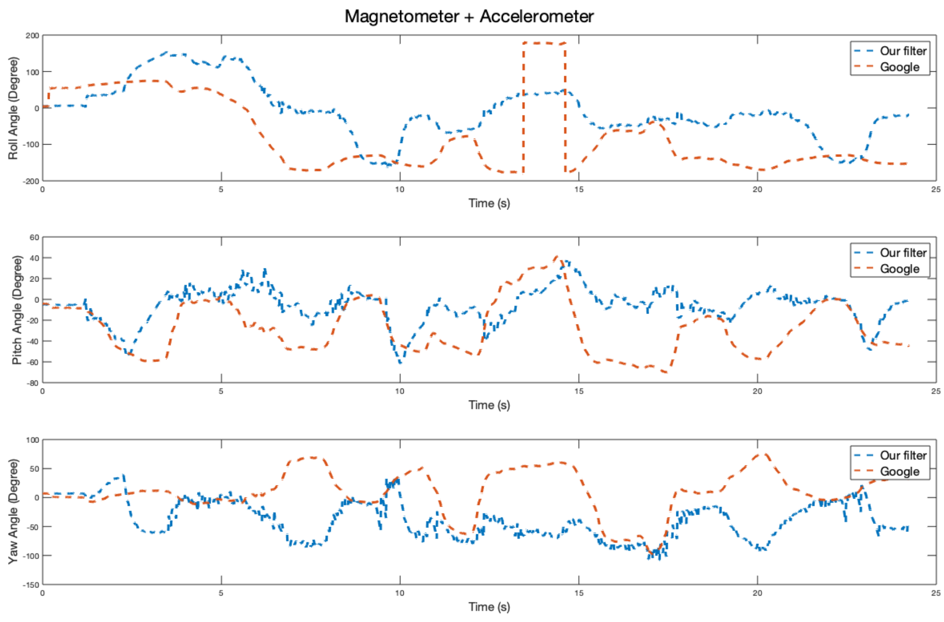

Figure 7: Comparison between developed filter and Googles estimate using accelerometer and magnetometer

Conclusions

One important aspect on this tests is that the magnetometer is sensitive for disturbances. For example, the tests made with the magnetometer has to be done far away from the computer since our computers has some magnets attached to their screen. If the tests is performed too close to one of the computers or another magnetic field, the tests will turn out really bad since the magnetometer get disturbed. Another conclusion is that our magnetometer in this phone is probably pretty bad since the we are not able to get the magnetometer as accurate as the gyro and the acceleometer. This can be also be verified from the greater covariance of the magnetometer (see Task 2). We guess that different phones have different quality of the sensors which will affect the results.

The estimation of the angles was of course best if all three sensors was used. The second best is when the magnetometer is missing and the phone is starting at the same point as we estimated our sensor characteristics, i.e flat on the table, screen up and pointing at north. This is due to the gyroscope and accelerometer together perform well in estimating the orientation in reference to this point.

Appendix

```matlab
function [xhat, meas] = ourFilter(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.
  % Add your filter settings here.

  % Sensor covariance matrices, tuned
  Rw = [0.1007,0.0012,0.0007;
    0.0012,0.0530,0.0026;
    0.0007,0.0026,0.0467]*1e-4;

  Ra = [0.1545,0.0753,-0.0842;
    0.0753,0.1792,-0.0840;
   -0.0842,-0.0840,0.2263]*1e-1;

  Rm = [20.3254 ,   8.4193 ,  -1.4920;
    8.4193 ,  42.0064,   -6.4142;
   -1.4920 ,  -6.4142 ,   6.6191]*1e1;

  % Define gravitational vector. Captured when phone was lying flat on
  % table
  g0 = [-0.1247; 0.0077; 9.6096];

  % Define earths magnetic field according to assignment
  mx = 14.4197;
  my = 4.6618;
  mz = 12.9508;

  m0 = [0, sqrt(mx^2 + my^2), mz]';

  % Define parameters for AR(1)-filter
  L = norm(m0);
  alpha = 0.01;

  % 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

      acc = data(1, 2:4)';
      if ~any(isnan(acc)) % Acc measurements are available.
          % Set bounds for outlier rejection
          if norm(acc) < norm(g0) * 1.05 && norm(acc) > norm(g0) * 0.95
              [x, P] = mu_g(x, P, acc, Ra, g0);
              [x, P] = mu_normalizeQ(x, P);
              ownView.setAccDist(0); % Acc reading are not outliers
          else % Discard outliers
              ownView.setAccDist(1); % Acc reading are outliers
          end
      end

      gyr = data(1, 5:7)';
      if ~any(isnan(gyr))  % Gyro measurements are available.
        [x, P] = tu_qw(x, P, gyr, 0.01, Rw);
        [x, P] = mu_normalizeQ(x, P);
      else % If gyro values are NaN, assume random walk
        [x, P] = tu_qw_omegaNAN(x, P);
        [x, P] = mu_normalizeQ(x, P);
      end


      mag = data(1, 8:10)';
      if ~any(isnan(mag))  % Mag measurements are available.
          % Calculate AR(1) filter value
          L = (1 - alpha) * L + alpha * norm(mag);

          % Set bound for AR(1)-filter
          if abs(L - norm(mag)) < L * 0.1
              [x, P] = mu_m(x, P, mag, m0, Rm);
              [x, P] = mu_normalizeQ(x, P);
              ownView.setMagDist(0); % Acc reading are not outliers
          else % Reading are outliers, discard!
              ownView.setMagDist(1); % Mag reading are not outliers
          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
```