Processing of gyroscope and accelerometer readings

Dirk van Dooren, Ioannis Karagiannis, Johan Bjurgert

September 2, 2014

Abstract

This project is about determining the pose of a robot by using a gyroscope and an accelerator. We make use of a Raspberry Pi, a combined accelerometer and gyroscope and fuse the data using both a Kalman filter and the so called Complementary filter. We implement the Kalman filter both in c and matlab and show how performance is improved in c. We also compare the output of the Kalman filter to the output of the Complementary filter and conclude that the Kalman filter is to prefer.

1 Sensors

When the robot is moving at different speeds, which is the case when balancing, both a gyroscope and an accelerometer is needed to obtain a reliable estimate of the tilt angle.

1.1 Gyroscopes

A gyroscope measures rate of rotation r without the need of an external reference. Gyroscopes are commonly used to measure angles ϕ, which is done by adding up the gyroscope readings over time dt. $\phi_k = \phi_{k-1} + r_k * dt$. The gyroscope has a *bias* (from white noise like errors), which over time accumulates and is referred to as *drift*.

1.2 Accelerometers

An accelerometer behaves much like a mass suspended with a spring system, see figure 1. When using such a device to estimate static tilt, it is clear that the estimate would be off if the accelerometer itself would be accelerated. We use a 3 degree of freedom accelerometer but are mainly interested in accelerations in the x, y directions.

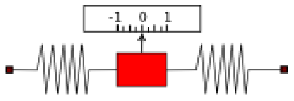


Figure 1: Principle of measuring accelerations.

2 Filtering the sensor readings

The sensor readings need to be fused to get a reliable estimate of the tilt angle. We make use of the Kalman filter and compare it to a simpler filter called the complementary filter. We also give a derivation of the complementary filter.

2.1 The Complementary filter

The gyroscope, ω, has highpass characteristics and the accelerometer, a, has lowpass characteristics. In the frequency domain, this takes the form

$$\theta(s) = \frac{1}{1+Ts}a(s) + \frac{Ts}{1+Ts}\frac{1}{s}\omega(s) = \frac{a(s)+T\omega(s)}{1+Ts}, \tag{1}$$

where T is the cutoff frequency of the filters. There are many ways of mapping a continous filter to a discrete one and in this case we make use of the backward difference mapping

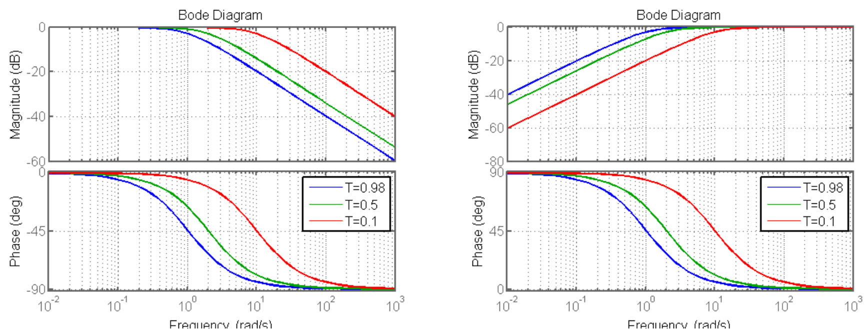
$$s := \frac{1}{dt}(1-\frac{1}{z}). \tag{2}$$

It can be showed that the mapping preserves stability.

Plugging in 2 into 1 yields

$$\begin{aligned}
\theta(z)(1+\frac{T}{dt}-\frac{T}{dt}\frac{1}{z}) &= a(z)+T\omega(z) & \underbrace{\Rightarrow}_{\text{z-inv}} \quad & (3)\\
\theta_k(1+\frac{T}{dt}) - \theta_{k-1}\frac{T}{dt} &= a_k + T\omega_k & \Leftrightarrow \quad & (4)\\
\theta_k &= \frac{a_k}{1+\frac{T}{dt}} + \frac{\theta_{k-1}\frac{T}{dt}}{1+\frac{T}{dt}} + \frac{T\omega_k}{1+\frac{T}{dt}} & \Leftrightarrow \quad & (5)\\
\theta_k &= \alpha(\theta_{k-1}+dt\omega_k) + (1-\alpha)a_k, & \alpha = \frac{\frac{T}{dt}}{1+\frac{T}{dt}} \quad & (6)
\end{aligned}$$

The end result 6 is known as the complementary filter and by varying T, different behaviour is achieved.



(a) (b)

Figure 2: Impact on the lowpass filter and highpass filter by varying T

2.2 The Kalman filter

A more theoretical approach is to use the Kalman filter to reduce the noise v_k from sensor readings, a_k and ω_k, and then correct the angle with its bias ω_b.

$$x_k = \begin{pmatrix} \theta^x \\ \theta^y \\ \omega_b^x \\ \omega_b^y \end{pmatrix}_k \quad F = \begin{pmatrix} 1 & 0 & -dt & 0 \\ 0 & 1 & 0 & -dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} dt & 0 \\ 0 & dt \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} u_k = \begin{pmatrix} \omega^x \\ \omega^y \end{pmatrix}_k$$

$$P_1 = \begin{pmatrix} 0.001 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0.003 & 0 \\ 0 & 0 & 0 & 0.003 \end{pmatrix} dt \quad P_2 = \begin{pmatrix} 0.03 & 0 \\ 0 & 0.03 \end{pmatrix} y_k = \begin{pmatrix} a^x \\ a^y \end{pmatrix}_k$$

Kalman Filter Algorithm

Prediction

$$\begin{aligned} \hat{\mathbf{x}}_{n+1|n} &= \mathbf{F}\hat{\mathbf{x}}_{n|n} + \mathbf{B}\mathbf{u}_n \\ \mathbf{P}_{n+1} &= \mathbf{F}\mathbf{Q}_n\mathbf{F}^{\mathrm{T}} + \mathbf{G}\mathbf{P1}_n\mathbf{G}^{\mathrm{T}} \end{aligned}$$

Update

$$\begin{aligned} \mathbf{L}_{n+1} &= \mathbf{P}_{n+1}\mathbf{H}^{\mathrm{T}}[\mathbf{H}\mathbf{P}_{n+1}\mathbf{H}^{\mathrm{T}} + \mathbf{P2}]^{-1} \\ \hat{\mathbf{x}}_{n+1|n+1} &= \hat{\mathbf{x}}_{n+1|n} + \mathbf{L}_{n+1}(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_{n+1|n}) \\ \mathbf{Q}_{n+1} &= \mathbf{P}_{n+1} - \mathbf{P}_{n+1}\mathbf{H}^{\mathrm{T}}[\mathbf{H}\mathbf{P}_{n+1}\mathbf{H}^{\mathrm{T}} + \mathbf{P2}]^{-1}\mathbf{H}\mathbf{P}_{n+1} \end{aligned}$$

A closer look shows how the prediction equals the previous estimate plus the unbiased rate multiplied with dt.

$$\begin{aligned} \begin{pmatrix} \theta^x \\ \theta^y \\ \omega_b^x \\ \omega_b^y \end{pmatrix}_{k|k-1} &= \begin{pmatrix} 1 & 0 & -dt & 0 \\ 0 & 1 & 0 & -dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \theta^x \\ \theta^y \\ \omega_b^x \\ \omega_b^y \end{pmatrix}_{k-1} + \begin{pmatrix} dt & 0 \\ 0 & dt \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \end{pmatrix}_{k-1} \\ &= \begin{pmatrix} \theta^x + dt(\omega^x - \omega_b^x) \\ \theta^y + dt(\omega^y - \omega_b^y) \\ \omega_b^x \\ \omega_b^y \end{pmatrix}_{k-1} \end{aligned}$$

3 Implementation

All code is written in C and is executed on a Raspberry Pi connected with a Pololu MinIMU-9, which is an inertial measurement unit (IMU) that packs an L3G4200D 3-axis gyro and an LSM303DLM 3-axis accelerometer and 3-axis magnetometer onto a 0.9 inch times 0.6 inch board. An I^2C interface accesses nine independent rotation, acceleration, and magnetic measurements that can be used to calculate the sensor's absolute orientation. We make use of the GNU Scientific Library to handle matrix operations and the low level code for handling sensor input was retrieved from `https://github.com/mwilliams03/Raspberry-Gyro-Acc/blob/master/main.c`.

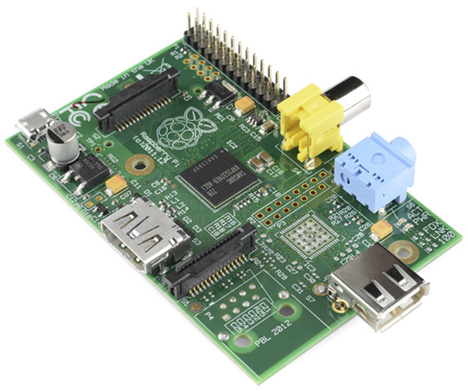
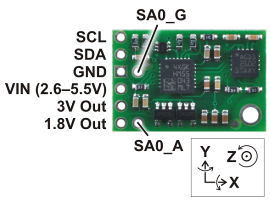
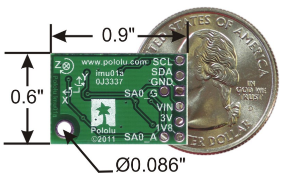


Figure 3: Raspberry Pi computer.



(a)



(b)

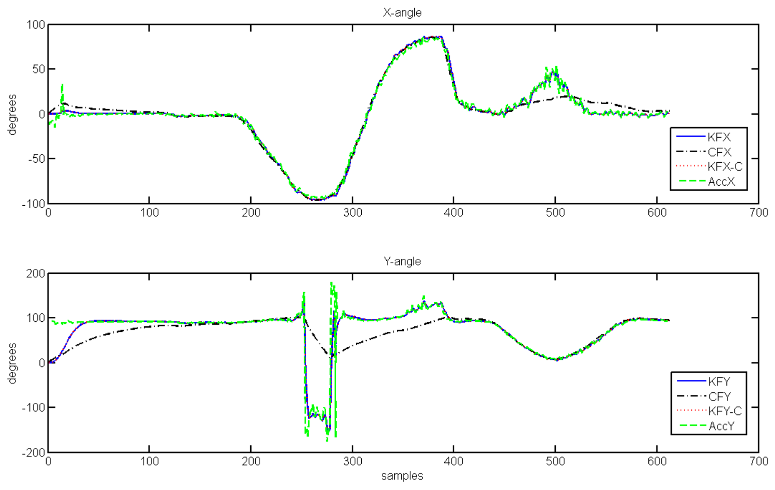
Figure 4: The IMU

4 Results

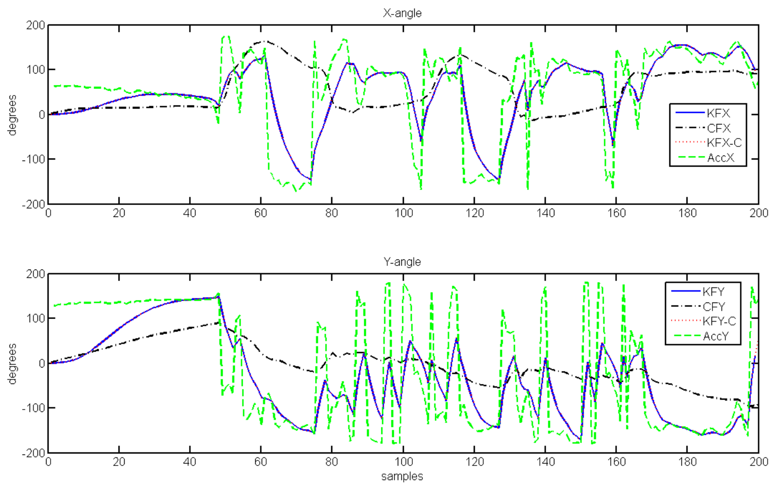
In all cases we assumed that we have higher uncertainty in our measurements than in our model. That is reasonable since we do not trust the biased measurements. We observe that the Kalman filter implemented in C fits the one implemented in Matlab. The c-code takes roughly 12μs whereas the matlab-code takes 70μs per iteration. The problem with the complementary filter, is that it needs tuning of T and if it is good for the accelerometer, then it is not good for the gyroscope. This can be seen from 2b and 2a and by comparing 5a to 5b. The Kalman filter does not have this problem and it is well known that it is the optimal filter choice, at least for Gaussian noises and linear models.

5 Future work

Buy wheels and motor and design a controller to make the robot self-balance.



(a)



(b)

Figure 5: In 5a, the sensor is held still and rotated and in 5b the sensor is shaken and rotated.