Hochschule Esslingen
University of Applied Sciences

Team Project

Quadrocopter

in the degree course ASM-SB
of the Faculty Graduate School
ASM2

Oliver Breuning
Martin Brodbeck
Jürgen Schmidt
Phillip Woditsch

Periode: Summersemester 2015
Professor: Prof. Dr. Jörg Friedrich

Contents

List of Figures

1 Introduction Sensor Fusion

For enabling a autonomous flight, the Raspberry Pi has to know the orientation. Figure 1.1 shows the axes and the naming of the rotation around the axes. These rotations are later used for the calculation for the roll, pitch and yaw angles.

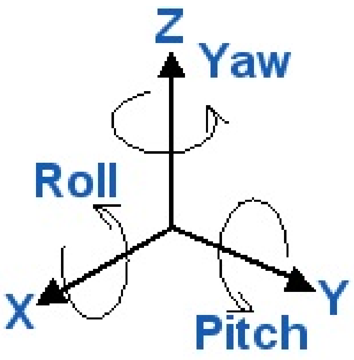

Figure 1.1: Roll, Pitch, Yaw [BorEng]

To get reliable stable orientation of the Raspberry Pi and so from the Quadrocopter several sensors can be used. Either a acceleration sensor, a magnetic sensor or a gyroscope can be used. But each of them have some drawbacks.
The acceleration sensor is very fast and delivers reliable pitch and roll angles, but the yaw angle itself can't be calculated. Another problem is, that due to vibrations a smooth angle is not possible to calculate.

The magnetometer can be used to calculate the heading, this means the yaw angle but not the roll and pitch angle. Another problem when the sensor has a roll and pitch angle, the heading can not be easily calculated. In this case a tilt compensation has to be done. In the figure 1.2 the logging of the pitch and roll angle calculated from the acceleration sensor and the yaw angle calculated with the magnetometer can be seen. When those sensors are not combined errors occur during the calculation. In the time from 250 seconds to 450 seconds a yaw angle is applied which leads just to a yaw angle change. In the time area from 500 seconds to 600 seconds a roll angle is applied which also leads to a yaw change which is an error. In the time area from 700 seconds to 800 seconds a pitch angle is applied which also leads to a yaw change which is an error.

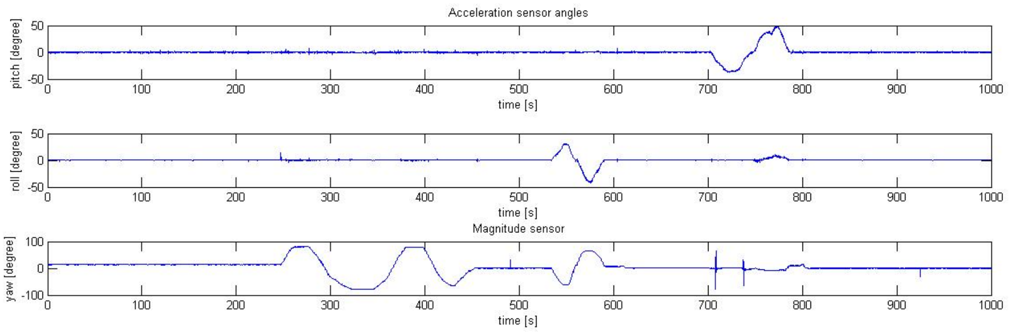

Figure 1.2: Magnitude / Acceleration angles

The last sensor is the gyroscope. The angle can be easily calculated by integrating the rates of the gyroscope. The sensor is not as fast as the acceleration sensor. So vibrations make no problems for the calculation. But due to the problem of the offset of the gyroscope, the angles will be drifting because of the integration of the rotation rates. This can be seen in figure 1.3

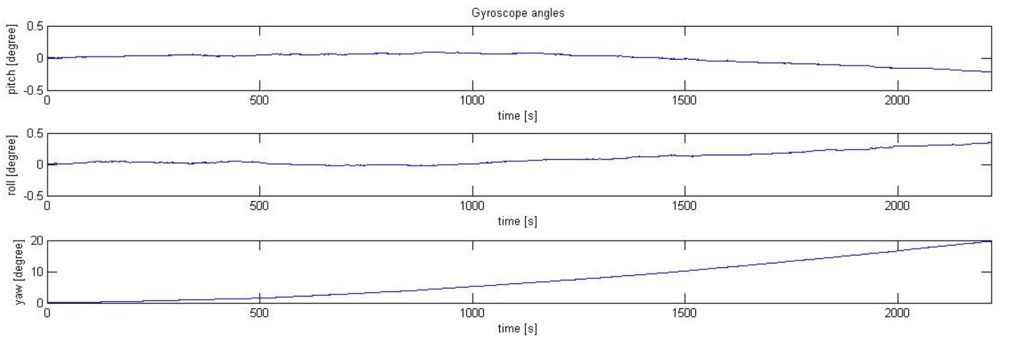

Figure 1.3: Gyroscope angles

To use all positive features of the sensors and reduce the negative drawbacks a sensor fusion is the needed solution. There are many possibilities for fusion algorithms. In this project a Complementary-Filter and Kalman-Filter is implemented.

2 Complementary-Filter

The first approach for a sensor fusion is the complementary filter. This filter uses a highpass filter for the gyroscopes and a lowpass filter for the acceleration sensor. The highpass filter is used after the integration of the rotation rates. This can be seen in figure 2.1.

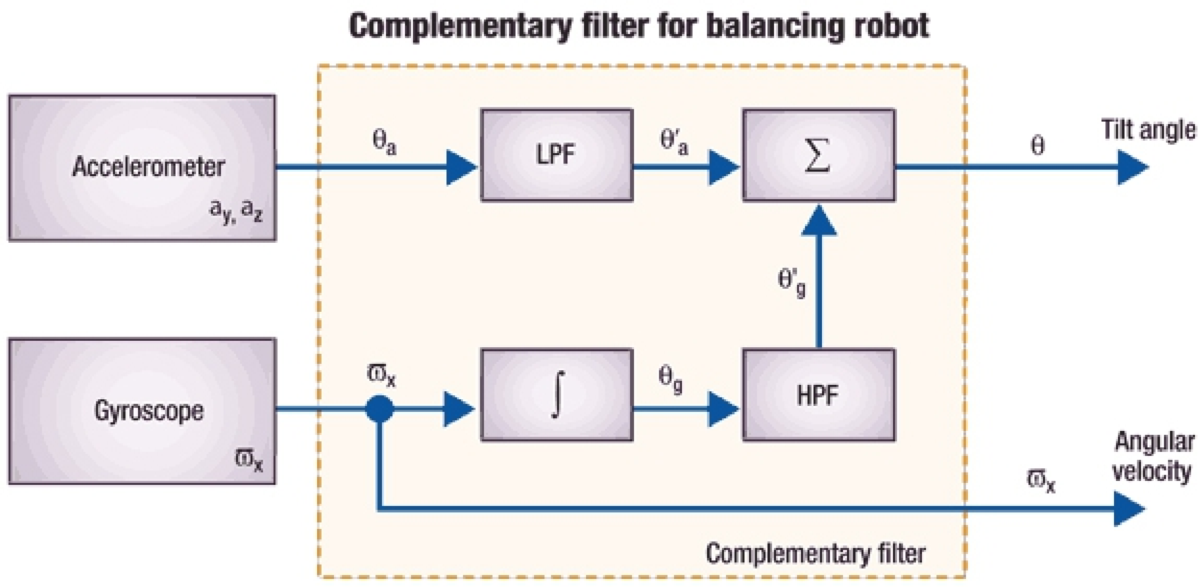

Figure 2.1: Complementary-Filter[STM]

The implementation effort is extremely lower than with the Kalman Filter. Because here not matrices and matrix operations are needed. Also are there less calculations and so fits this algorithm better to microprocessors. The only thing what needs to be checked is the lowpass/highpass-filter coefficient. Next the needed calculations are mentioned to show the difference of the complementary and Kalman filter.

- Calculation of the accelerometer and magnetometer angles
- Calculation of the gyroscope angles
- Complementary-Filter usage with defined filter time of the lowpass and highpass-filter

3 Kalman-Filter

The second approach for a sensor fusion is the Kalman filter. This filter is based on the state space modelling where between the dynamic of the system and the process of the measurement is differentiated. Although the measuring is faulty and the systemstate is noisy, this filter guesses with the help of a set of equations the correct true state of the system. Instead of using the absolut value it uses the meanvalue and variance of the normal distribution. The meanvalue is the perfect measurement and the variance tells the uncertainty of the measurement. Figure 3.1 shows a mesurement of an acceleration sensor.

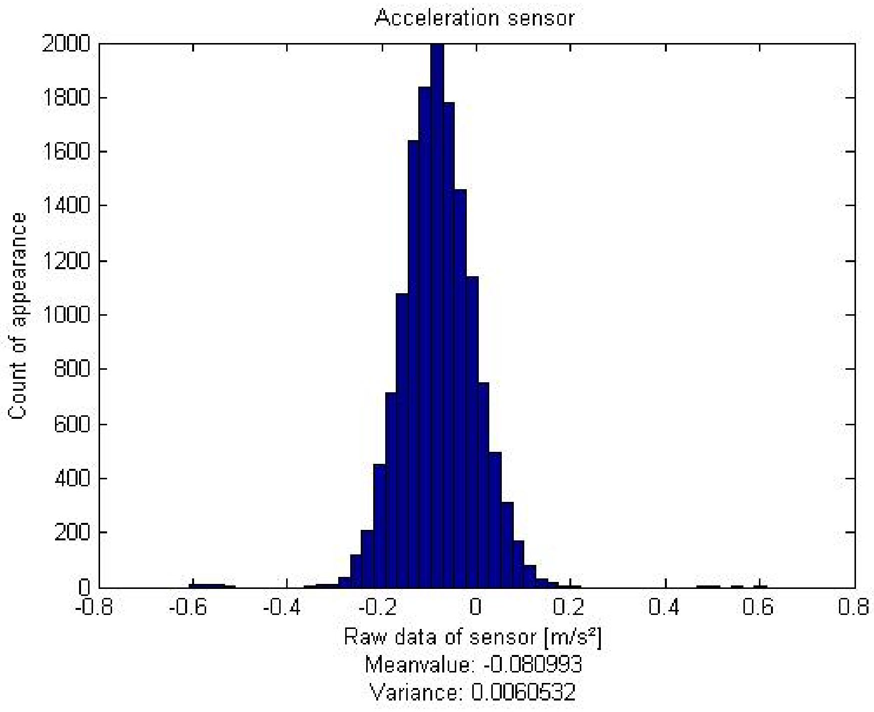

Figure 3.1: Variance and Meanvalue of an acceleration sensor

The following state space description shows the calculation of the new states which in the case of this project represent the angles pitch, roll and yaw. Equation 3.1 shows the defined state vector.

$$\vec{x} = \begin{pmatrix} pitch \\ roll \\ yaw \end{pmatrix} \tag{3.1}$$

The calculation of a new state and the output can be seen in the formulas 3.2 and 3.3.

$$\vec{x_k} = \underline{A}\vec{x}_{k-1} + \underline{B}\vec{u}_{k-1} + W_{k-1} \tag{3.2}$$

$$\vec{y_k} = \underline{H}\vec{x}_k + V_k \tag{3.3}$$

Legend of the formula shown above:

- $\vec{x}_k$ state vector of actual step
- $\vec{x}_{k-1}$ state vector of previous step
- $\underline{A}$ system matrix
- $\underline{B}$ input matrix
- $\vec{u}_{k-1}$ input vector of previous step
- $\underline{W}$ process noise
- $\vec{y}_k$ output vector
- $\underline{H}_k$ output matrix
- $\underline{V}$ measurement noise

Because of the measurement and the process noise the new state is not good. The Kalman filter takes the process and measurement noise into account to improve the state estimation. To do so the Kalman filter is splitted into two parts, the prediction process (time update) and correction process (measurement update). In the prediction process the filter estimates the states in the next step and calculates the new covariance. Those values are calculated for the states which in the following step are expected to be reached. In the next step the correction update, the filter checks if the precalculated state is reached. According to the difference the correction for the following prediction is done.

Prediction step:

Predict the next state:

$$\vec{x}_k = \underline{A}\vec{x}_{k-1} + \underline{B}\vec{u}_{k-1} \tag{3.4}$$

Predict the covariance for the next step:

$$\underline{P}_k = \underline{A}\underline{P}_{k-1}\underline{A}^T + \underline{Q} \tag{3.5}$$

<u>Correction step:</u>

Computation of the Kalman gain:

$$\underline{K}_k = \underline{P}_k\underline{H}^T(\underline{H}\underline{P}_k\underline{H}^T + \underline{R})^{-1} \tag{3.6}$$

Updating state prediction with new measurement:

$$\vec{x}_k = \vec{x}_k + \underline{K}_k(\vec{z}_k - \underline{H}\vec{x}_k) \tag{3.7}$$

Updating the error covariance:

$$\underline{P}_k = (\underline{I} - \underline{K}_k\underline{H})\underline{P}_k \tag{3.8}$$

These steps have to be done more than just ones, so when the correction is finished, the update has to be called again and so on.
As can be seen the implementation effort is higher than with the Complementary Filter. Because matrices and matrix operations are needed, more calculation steps are needed and the functionality of the Kalman filter is not as intuitive like with the complementary filter. Also the uncertainty of the process itself and the measurement error needs to be known. Nevertheless the results from the Kalman filter are better than those of the complementary filter. This can be seen in the results showing chapter of this project.

Bibliography

[BorEng] Alex, The quadcopter : control the orientation, 2012, http://theboredengineers.com/2012/05/the-quadcopter-basics/

[STM] Jay Esfandyari, Roberto De Nuccio, Gang Xu; STMicroelectronics, http://uk.mouser.com/applications/sensor_solutions_mems/