Otto von Guericke University Magdeburg
Chair of Intelligent Systems
Institute of Knowledge and Language Engineering

Digital Engineering Project

# Analysis of Extended Movement Models for Autonomous Quadrocopters

Bodnár, Dávid
david.bodnar@st.ovgu.de

Magdeburg, 2015

Supervisor:
Steup, Christoph
steup@ivs.cs.uni-magdeburg.de

## Abstract

The reliable estimation of the motion state of a moving object is very important if the motion state needs to be controlled. This problem gets more advanced once the system is not fully observable and the sensor values are noisy or incorrect.

To be able to handle this problem a Kalman filter might be the optimal solution. In this documentation the implementation of such a filter is discussed for a quadrocopter application. In the following the basic description of the system, the implementation and the evaluation of the Kalman filter will be specified.

**Keywords:** Quadcopter, Kalman-Filter, Motion tracking, Paparazzi

# Contents

# 1 Introduction

Nowadays quadcopters are slowly getting part of our everyday life. First they were just used as a toy or to shoot photographs or capture videos that people were not able to do before. The new prototypes are developed to make a wider range of applications possible, for example package delivery services, border protection or construction of rope bridges.

In all of these cases precise motion state control is required to make it possible for the quadcopters to follow specified paths accurately. Unfortunately in most of the cases the raw sensor values are not accurate enough to base the whole control on them. Not only the noisy or faulty measurement values but also the different measurement frequencies might cause problems.

To overcome these limitations a Kalman filter might be the optimal solution. The Kalman filter is a linear quadratic estimator which uses prediction and correction steps to give an accurate result for the process variables.

In this documentation I will present the basic idea of the Kalman filter and the implementation of one for motion state estimation of a quadcopter. In the following chapters the equations of the Kalman filter and the physical model of the quadcopter will be discussed. After that I will go through the steps of the implementation and list the limitations of the designed system.

# 2 Kalman filter

As already mentioned, the Kalman filter is a linear quadratic estimator. The most well-known application of the filter is the GPS navigation system used in cars. In the following, I will use this example to describe the behavior of a Kalman filter.

But before discussing the mathematical background of the system, some requirements of the Kalman filter need to be observed. The filter was invented for tracking discrete linear dynamic systems which are disturbed by white Gaussian noise. If these system requirements are fulfilled, than it can be mathematically shown that the Kalman filter is the optimal linear estimator of the system. This means that the system tries to minimize the mean square error of the parameters.

If our system is not linear, it needs to be linearized or an extended Kalman filter should be applied. If the noise is not Gaussian, it can be also shown mathematically that the Kalman filter is the best linear estimator which can be taken.

The filter can be split into two processes, the prediction and the correction, which do not necessarily occur with the same frequency. To be exact, the prediction happens in most of the cases at least with the same frequency as the measurement, but in most real life applications it happens more often. In the following these two phases will be discussed.

## 2.1 Prediction

In the prediction phase the filter gives an estimation about how the system variables might have changed during the elapsed time. This process can be described with the following two equations:

$$\overline{\mathbf{x}}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t \tag{2.1}$$

$$\overline{\mathbf{P}}_t = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T + \mathbf{Q} \tag{2.2}$$

Here and in the following the overline refers to a predicted value. The first equation describes the change of the system variables. The second one characterizes the goodness of the estimation by defining the covariance for the estimation.

The vector $\mathbf{x}$ is the system state vector and is used to refer to the system variables. These are the values which describe the state of the system for a given time. If the values of $\mathbf{x}$ are known, the behavior of the system over time can be described. It has the dimension of $[n \times 1]$.

The vector $\mathbf{u}$ is called the input vector and is used to refer to the excitation of the system. The values of $\mathbf{u}$ describe the system input which is controlled by the user directly or indirectly through another process. It has the dimension of $[m \times 1]$.

The matrix $\mathbf{A}$ is the system state transition model matrix. It has the dimension of $[n \times n]$. This matrix describes the system behavior if it gets no excitation from its environment. Taking this into account the first part of the equation 2.1 is just the effect of the system variables on each other as time goes by.

The second part of the equation describes the system excitation. The matrix $\mathbf{B}$ (control input model matrix) has the dimension of $[n \times m]$. The matrix shows to which system state variables the user or the environment has direct access by the control. The matrix **A** describes the indirect access to the system variables and the matrix **B** shows the direct access to the system variables.

If one constructs a state space controller and writes up the state equation of the system without sensor observation, the equation is the same as the equation 2.1. This is the most common way of getting started constructing a Kalman filter.

In the second equation (2.2) the uncertainty of the current prediction is calculated. The matrix **Q** is the so called system process noise matrix. By choosing these values correctly, the simplified model can be extended and the effects of neglected environment (wind, calibration error etc.) can be taken into account. The matrix **Q** has the same size as the matrix **A**.

The matrix **P** is the system state covariance matrix which has the size of $[n \times n]$. This defines the part of the n dimensional space around the predicted system state where the system might be. This uncertainty keeps growing, because the filter algorithm keeps adding the static process noise (**Q**) to the uncertainty, as long as no correction takes place.

As it can be seen from the equations the prediction is a recursive process. This means that both the matrix **P** and the vector **x** need to be initialized at the beginning to be able to start the Kalman filter.

From the prediction equations one can see that matrices **A**, **B** and **Q** are constants. They characterize the simplified linear physical model of the system so they need to be constructed during the implementation.

To show the importance of the prediction phase let us take a look at the GPS example. Suppose that you are driving on the motorway and you have a navigation system on board which navigates you during your trip. You drive into a tunnel where you lose the signal of the satellites. In this case your navigation will suppose that you are driving with a constant speed through the tunnel and it might be able to assist you if you have to switch the traffic lane or take another exit. As this example shows, the system cannot measure where you are, but it gives a prediction where you might be and uses it as a system output. Of course if there are additional sensors on board, it might provide you a more sophisticated prediction for your movement in the tunnel.

## 2.2 Correction

If you would slow down your car in the tunnel, it would have no effect on the prediction at all. The on board navigation would still suggest that you have the same speed as you entered the tunnel. As you get out of the tunnel your navigation will have satellite signal again. This means that the system is able to measure the current state of your car. The measured position will definitely differ from the result of the prediction. Using the correction step the Kalman filter corrects the system state according to the following equations:

$$\mathbf{y}_t = \mathbf{z}_t - \mathbf{H}\overline{\mathbf{x}}_t \tag{2.3}$$

$$\mathbf{S}_t = \mathbf{H}\overline{\mathbf{P}}_t\mathbf{H}^T + \mathbf{R} \tag{2.4}$$

$$\mathbf{K}_t = \overline{\mathbf{P}}_t\mathbf{H}^T\mathbf{S}_t^{-1} \tag{2.5}$$

$$\mathbf{x}_t = \overline{\mathbf{x}}_t + \mathbf{K}_t\mathbf{y}_t \tag{2.6}$$

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\overline{\mathbf{P}}_t \tag{2.7}$$

The sensor measurement values are represented in the vector **z**, which is the so called observation vector. This vector has the same size as the system state vector **x**. In most of the cases the vector **z** will have the state variables in the same form as the vector **x**. In these cases the observation model matrix (**H**) is a simple identity matrix (**I**). If this is not the case then the $[n \times n]$ observation model matrix **H** is used to transform the predicted system state into the observation space.

The vector **y** is the measurement residual which contains the error between the predicted states and the measurement values. It has the same dimensions as the vectors **z** and **x**.

Beside the error for the system variables the error for the uncertainties has to be computed. So the residual covariance is calculated. This is the matrix **S** which has the same dimensions as **P**. The residual is put together from two separate parts. First of all the predicted estimate covariance is transformed into the measurement space. After that the observation covariance matrix (**R**) is added which includes the uncertainty of the sensor measurements.

After all of these steps everything is given to be able to compute the so called Kalman gain which is represented by the matrix **K**. This matrix has the same dimensions as **P**. The actual meaning of the Kalman gain is how much the sensors can be trusted. The equations 2.6 and 2.7 are the consequences of this meaning. If the sensors are more reliable the measured values should get a higher weight in the result, if not the weight of the predicted values should be increased. Using the weights the filter results for the system state and the uncertainty of these state variables can be obtained.

From the equations one can easily see that only the matrices **H** and **R** are constants. They characterize the measurement model of the system.

It is very important to know that the equation 2.7 can only be used if the calculated Kalman gain is the optimal Kalman gain. In other cases the Kalman gain is not computed with the equation 2.5 or the designed Kalman filter has problems with numerical stability.

In these cases the so called Joseph form needs to be used.

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\overline{\mathbf{P}}_t(\mathbf{I} - \mathbf{K}_t\mathbf{H})^T + \mathbf{K}_t\mathbf{R}\mathbf{K}_t^T \tag{2.8}$$

The Joseph form produces more stable results for the posterior error but computationally it costs more than the equation 2.7. So in most of the cases people use the equation 2.7 when implementing a Kalman filter.

# 3 Theoretical model

In this chapter I will discuss two models of the systems. First of all, a nonlinear system model will be shown. After that, I will modify this model to make a Kalman filter constructible with the resulted equations.

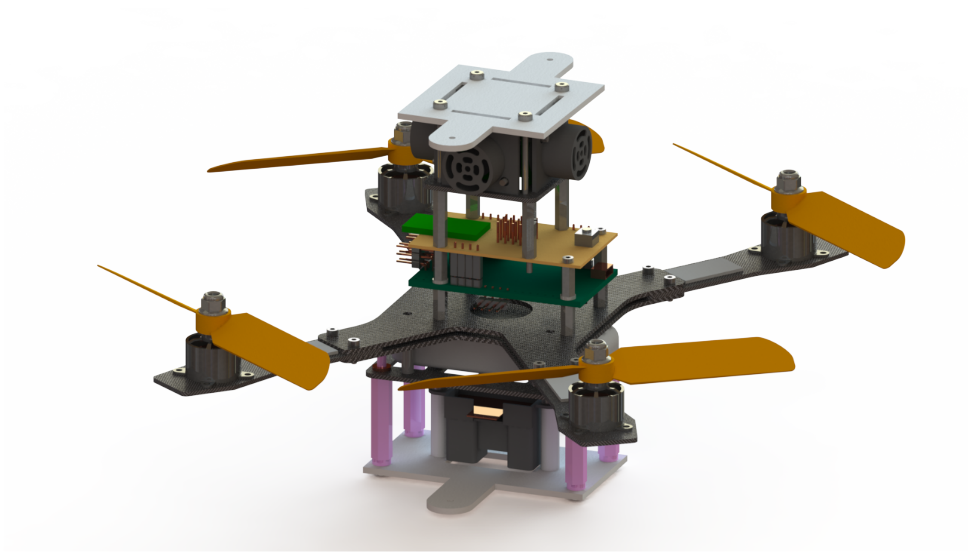

**Figure 3.1:** Rendered CAD model of the quadrotor

## 3.1 Nonlinear model

To be able to model the behavior of the quadcopter the model will be split into three separate parts. First I will discuss the rotor model. After that the force and the momentum balance equations will be analyzed.

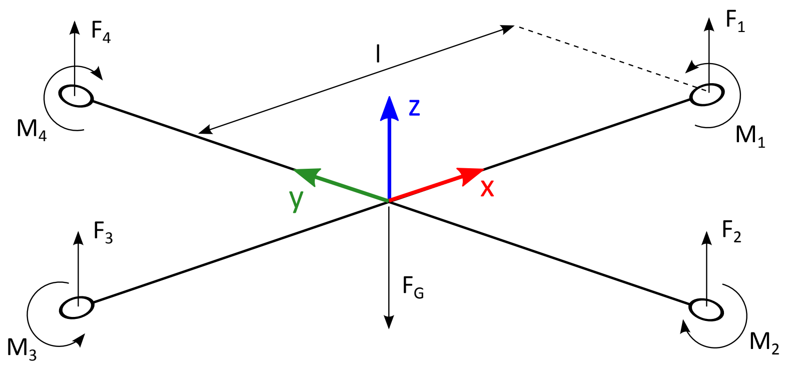

**Figure 3.2:** Model of the quadcopter

### 3.1.1 Rotor model

First let us take a look at one motor. For the quadcopter BLDC (Brushless Direct Current) motors were used. In the motor model the electronic behavior of the motor will not be analyzed, only its mechanical part is relevant. For the transformation of the electrical energy into mechanical energy a LUT (look-up table) needs to be constructed on measurement basis.

The motors are the actuators in the system which can be controlled to change the state of the quadcopter. In this way the input will be the angular speed of the motor and the resulting force and momentum need to be specified. They can be calculated using the following equations:

$$F_M = c \cdot r_{rotor} \cdot \omega_{rotor}^2 \tag{3.1}$$

$$M_M = (\theta_{motor} + \theta_{rotor}) \cdot \dot{\omega}_{rotor} \tag{3.2}$$

The constant $c$ is a rotor specific constant which comes in most of the cases from the manufacturer. The $r_{rotor}$ is the angular velocity of the rotor. With $\theta$ I marked the moment of inertia in the equations.

As the motor forces and moments always point in the $z$ direction the equations were formed in 2D and all 3D relevant parts of the equations were neglected.

### 3.1.2 Force balance

By using the force balance equation the connection between the motor forces and the acceleration of the quadcopter can be constructed.

$$\sum \mathbf{F} = m \cdot \mathbf{a} \tag{3.3}$$

To be able to handle the rotation of the system rotational matrices were used in the equations. Based on the fact that rotational matrices are orthogonal matrices if I use a matrix to transform from local to global coordinate system, I can use the transpose of it to get the backward transformation. To identify the angles the Euler angles were used. The angle state of the copter is defined in the following way:

$$\eta = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{3.4}$$

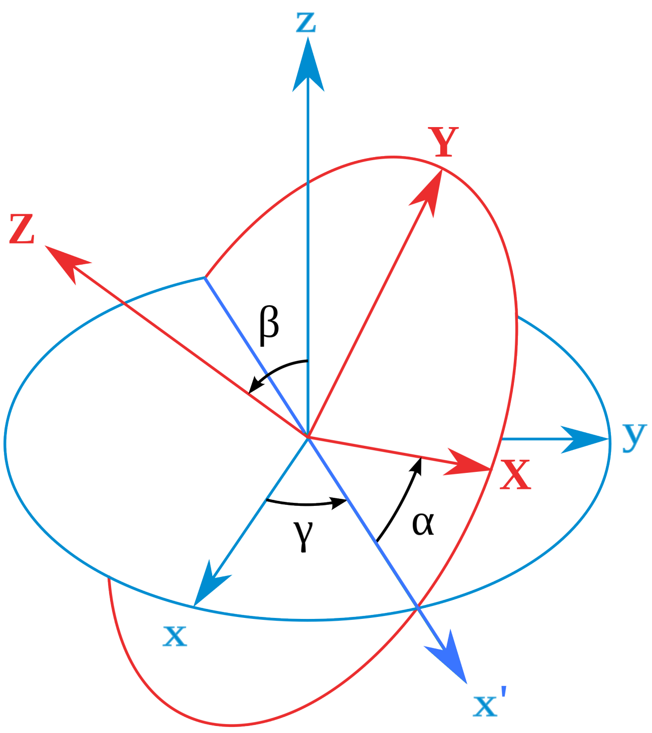

**Figure 3.3:** Euler angle convention

The rotation convention is also showed on the figure 3.3. To calculate the backward transformation from local to global coordinate system the following rotational matrix $\mathbf{R}$ was used:

$$\mathbf{R} = (\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x)^T \tag{3.5}$$

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{3.6}$$

$$\mathbf{R}_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{3.7}$$

$$\mathbf{R}_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

$$\mathbf{R} = \begin{bmatrix} \cos(\gamma)\cos(\beta) & \sin(\gamma)\cos(\beta) & -\sin(\beta) \\ \cos(\gamma)\sin(\beta)\sin(\alpha) - \sin(\gamma)\cos(\alpha) & \sin(\gamma)\sin(\beta)\sin(\alpha) + \cos(\gamma)\cos(\alpha) & \cos(\beta)\sin(\alpha) \\ \cos(\gamma)\sin(\beta)\cos(\alpha) + \sin(\gamma)\sin(\alpha) & \sin(\gamma)\sin(\beta)\cos(\alpha) - \cos(\gamma)\sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \tag{3.9}$$

The motor forces always point in the positive $z$ direction in the local coordinate system. This needs to be transformed into the global coordinate system by using the matrix $\mathbf{R}$. The gravitation always indicates a force in the negative $z$ direction in the global coordinate system.

By taking these into account the force balance equation for the system can be written in this way:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \frac{1}{m}\left(\mathbf{R}\begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ F_g \end{bmatrix}\right) \tag{3.10}$$

### 3.1.3 Momentum balance

The rotation of the motors not just indicate a force but a torque, too. These torques and the lifting motor force differences define the angular state of the quadcopter. This can be characterized with the following equations:

$$\begin{bmatrix} \ddot{\alpha} \\ \ddot{\beta} \\ \ddot{\gamma} \end{bmatrix} = \theta^{-1}\left(\sum \mathbf{M}\right) \tag{3.11}$$

$$\sum \mathbf{M} = \mathbf{R}\left(\begin{bmatrix} 0 \\ 0 \\ M_1 + M_2 + M_3 + M_4 \end{bmatrix} + \begin{bmatrix} (F_4 - F_2)\cdot l \\ (F_3 - F_1)\cdot l \\ 0 \end{bmatrix}\right) \tag{3.12}$$

Until this point I characterized the motors and the resulting accelerations and angular accelerations. This means that everything is available to construct the Kalman filter. In the following this model will be modified to better suit the criteria of the filter.

## 3.2 Modified model

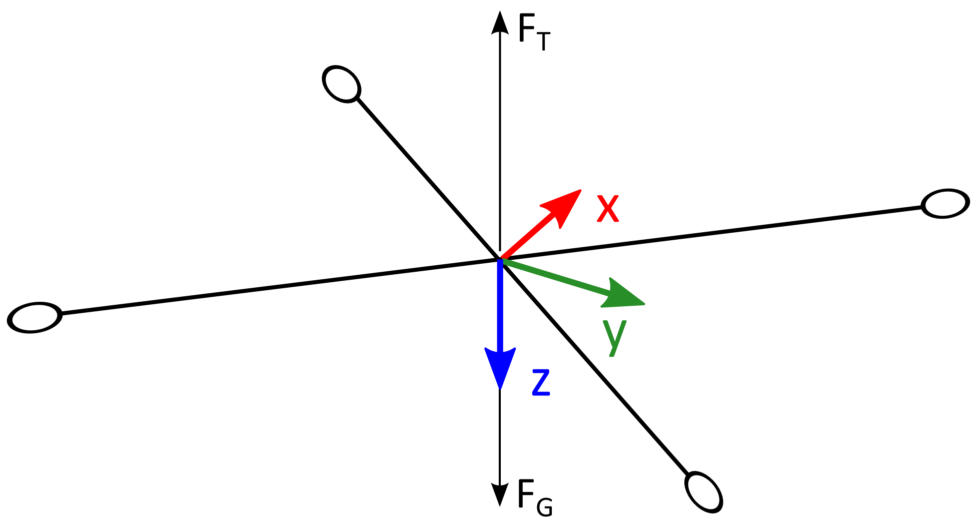

**Figure 3.4:** Simplified model for linearisation

The first modification that I apply to the model is changing the coordinate system. The equations until now were easier describable in the coordinate system shown in figure 3.2. Some of the equations above will not be implemented because the Paparazzi system provides other possibilities to describe the system.

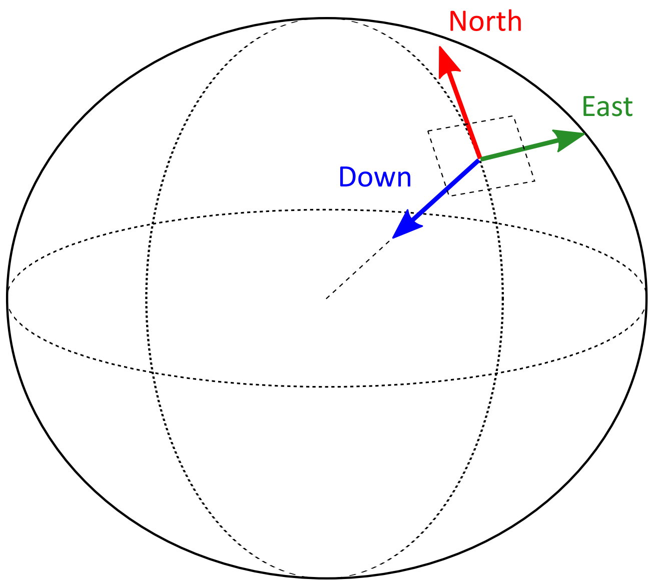

**Figure 3.5:** NED (North-East-Down) coordinate system

The quadcopter operates in NED (North-East-Down) coordinate system. This changes significantly some parts of the equations because some trigonometric parts will occur and the gravity vector will be directed in the opposite direction.

The Kalman filter requires a linear system model. This means that trigonometric terms are not allowed in the matrix equations. To overcome this limitation I decided to integrate all non-linear terms in the system excitation and the measurement process. This means that the resulting motor force is taken as control input. The control input and the sensor values for the measurement need to be rotated in the required direction before being processed.

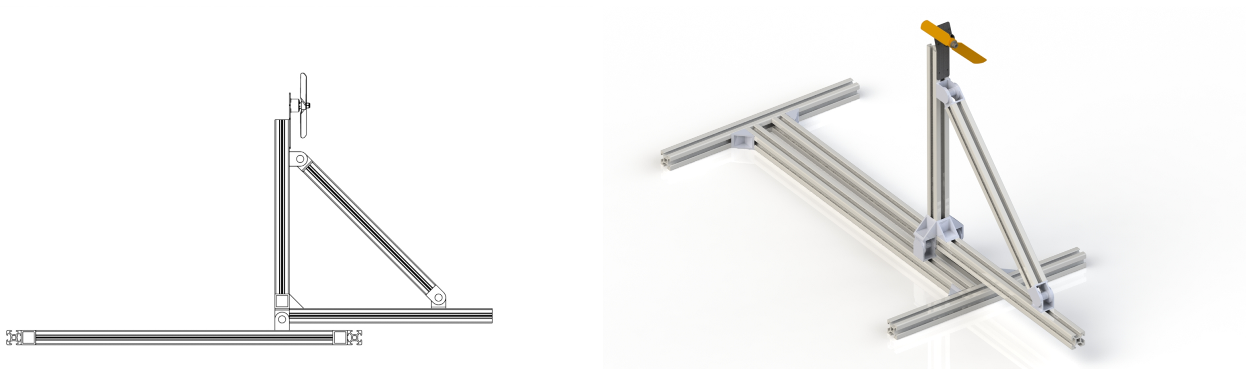

**Figure 3.6:** Measurement construction

After this simplification the motion state of the copter in all directions can be characterized using the following equations:

$$\mathbf{a} = \frac{1}{m}\mathbf{F}_t \tag{3.13}$$

$$\mathbf{v} = \mathbf{a} \cdot t \tag{3.14}$$

$$\mathbf{s} = \mathbf{v} \cdot t + \frac{\mathbf{a}}{2}t^2 \tag{3.15}$$

This way I got a number of linear system equations so the requirements of the Kalman filter are fulfilled in the model. The remaining part of the system model is a LUT which will construct the resulting motor force for the system. This will be discussed in the next section.

## 3.3 LUT for motor

The quadcopter control is based on the control of the throttle of the motors. To be able to calculate the control input forces of the Kalman filter a LUT was needed which converts the throttle [%] values into thrust [g].

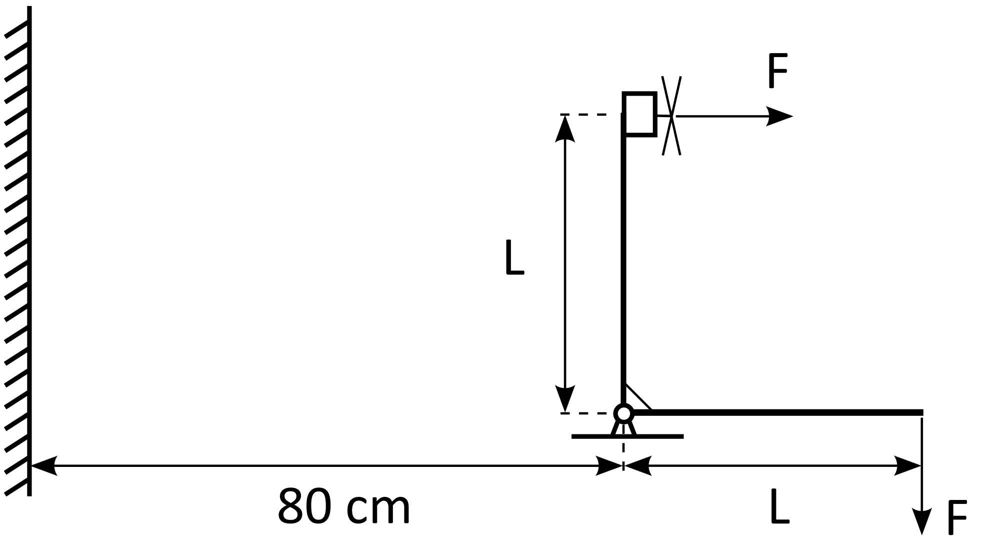

**Figure 3.7:** Simplified measurement model

To create the LUT the characteristic needed to be measured. To do this the test station on figure 3.6 was built. The simplified model of the measurement setup can be seen on figure 3.7. The rotation of the rotor indicates a force $F$. This force is transmitted to the bottom of the measurement set where it was measured by using a scale. If the system is considered to be frictionless, the scale shows the thrust in grams.

An Arduino program was written to control the throttle of the motor. Using a Raspberry Pi I tried to measure the dynamic characteristic of the motor too. Unfortunately, the signal was hard to measure due to the high angular velocity and the measurement error was significant so the motor model was based on the pure static behavior.

Using the Arduino program we were able to control the throttle and measure the indicated thrust to defined throttle levels. This can be seen on figure 3.8. The circles indicate the measurements. The second order regression model was calculated using Microsoft Excel. The resulting LUT function was constructed by Christoph Pahlke which is:

$$T = 0.01514k^2 + 0.65268k \tag{3.16}$$

Where $k$ is the throttle in [%] and $T$ is the thrust in $[g]$.

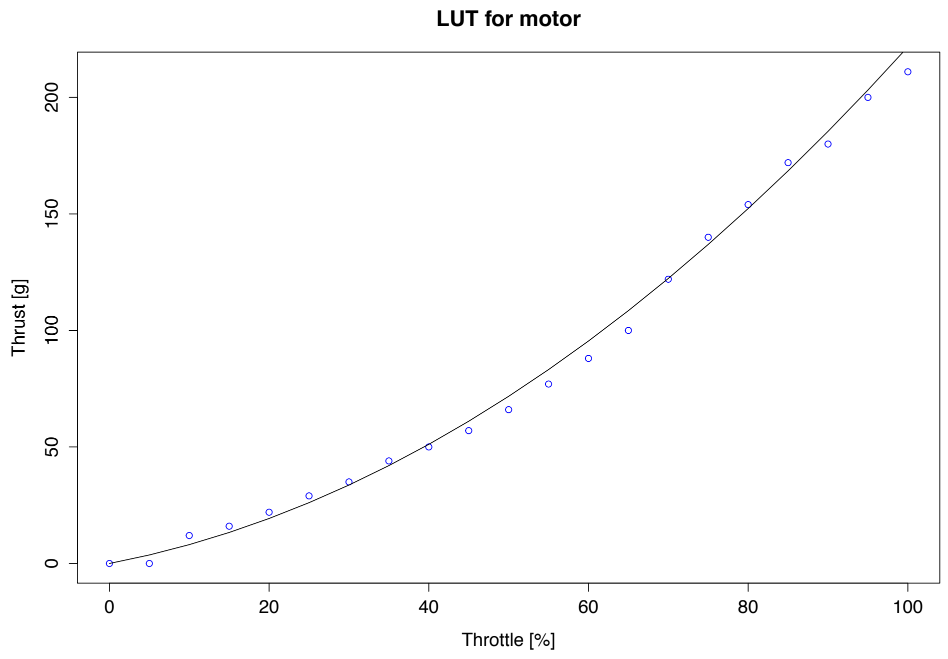

**Figure 3.8:** LUT for one motor

Of course, the resulting thrust of the four motors is not the same as four times the result of the LUT. But unfortunately we have no direct access to the control of the motors and modeling the motor mixture would be too complex. Because of this during the implementation I will consider the resulting thrust to be four times the result of the LUT.

During the measurement of the thrust it was also important how far the measurement system is from the next wall. To avoid big differences between the measurement and the real environment the distance between them was set to 80 [cm] because the altitude controller was designed to hold the quadrotor at this height.

# 4 Implementation

## 4.1 Libfixkalman

To solve the implementation task efficiently I decided to use the libfixkalman library. Libfixkalman is a Kalman filter library which is designed for microcontrollers without floating point unit. The libfixkalman depends on the libraries libfixmath and libfixmatrix.

The libfixkalman stores the values as 16.16 bit fixed point values in integers. Using these arithmetic the microcontroller is able to solve the operations significantly faster. The library gives the possibility to implement controlled and uncontrolled Kalman filter.

To be able to use this library efficiently to implement the Kalman filter some modifications needed to be done. First of all there was a bug in the controlled Kalman filter implementation which needed to be corrected.

Another problem was that the implemented controlled Kalman filter was a Time-varying Kalman filter. In the Time-varying Kalman filter the system process noise matrix is not constant but is defined in the following way:

$$\mathbf{Q} = \mathbf{B}\sigma\mathbf{B}^T \tag{4.1}$$

Where $\sigma$ is a $[m \times m]$ matrix which contains the covariance of the process noise. It is called the control input covariance matrix. The elements of the matrix define the covariance of the control input.

This means, that the matrix $\mathbf{Q}$ is not directly controllable which can cause some problems in the application. First of all this causes an additional calculation during the run of the Kalman filter. As the frequency for calling the function is fix, it would just waste resources. Another problem is that the double integration for the position would cause a $t^4$ element in the resulting $\mathbf{Q}$ matrix. As the call frequency is $f = 15$ [Hz] this would result in a $\frac{1}{15^4}$ term which is very small and would definitely cause numerical instability.

With this we get to the last problem of the current libfixkalman implementation. This is the numerical instability. As I already mentioned in the chapter about the Kalman filter, if the computation is not numerically stable, the equation 2.7 cannot be used. It needs to be replaced by the Joseph form (equation 2.8).

These modifications were applied to the libfixkalman library. The Joseph form and the Time-varying Kalman can be activated or deactivated from the settings.h header file. Please note that these last two modifications are not yet present in the official libfixkalman library.

## 4.2 Kalman filter implementation

First I implemented one Kalman filter to handle the problem. The problem with this implementation was that it caused quite big static matrices in the memory with a large number of zeros which just slow down the run of the whole process. To overcome this problem I decided to split the big Kalman filter into three smaller one for each direction. As the directions are independent from each other this will not cause any difference in the results but it will speed up the computation significantly. In the following matrices the indices x, y, z are refering to the Kalman filter in the defined direction. Please note that to reduce the unnecessary memory consumption the FIXMATRIX_MAX_SIZE constant in the fixmatrix.h was set to 3.

The Kalman filter in the z direction is always compensated with the gravity in the computation of the excitation and the sensor values.

$$\mathbf{A}_x = \begin{bmatrix} 1 & t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{A}_y = \begin{bmatrix} 1 & t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{A}_z = \begin{bmatrix} 1 & t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.2}$$

The system state transition model matrix is the implementation of Newton's first law of motion. It is very important to note that the rows containing the acceleration values are filled with zeros. This means that the system will not be able to track the acceleration of the system as it will delete the uncertainty of this value in each prediction step because of the equation 2.2. This means that the relation of the matrices $\mathbf{Q}_i$ and $\mathbf{R}_i$ will define the exact result of the Kalman filter for the accelerations.

$$\mathbf{B}_x = \begin{bmatrix} \frac{t^2}{2m} \\ \frac{t}{m} \\ \frac{1}{m} \end{bmatrix} \qquad \mathbf{B}_y = \begin{bmatrix} \frac{t^2}{2m} \\ \frac{t}{m} \\ \frac{1}{m} \end{bmatrix} \qquad \mathbf{B}_z = \begin{bmatrix} \frac{t^2}{2m} \\ \frac{t}{m} \\ \frac{1}{m} \end{bmatrix} \tag{4.3}$$

To get the excitation of the system motion the throttle is converted using the LUT to thrust and decomposed into the forces in the different directions. The impact of these forces are transferred into the Kalman filter using the control input model matrix. This is the implementation of Newton's second law of motion.

The mass used in this computation is not the real mass of the copter anymore. As the throttle is completely controlled by the altitude controller a virtual mass needed to be computed to handle the difference between the real world and the system model. The real mass of the quadrotor is 0.304 [kg] and the computed virtual mass was set to 0.380 [kg]. This virtualization of the mass can compensate the zero level difference between reality and the system model.

$$\mathbf{P}_{x_{init}} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_{y_{init}} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_{z_{init}} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

As already mentioned the initialization of the uncertainty is also required. As I can be certain in the position and the velocity values after starting the module I set the corresponding covariance values to a lower value. The motors produce significant vibration even in idle running. This means that the initial value can be quite different from the actual acceleration. To overcome this effect a higher value was set.

$$\mathbf{H}_\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{H}_\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{H}_\mathbf{z} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

To transfer the state vector into the measurement space identity matrices are used as the measurement values are computed independently from each other and the observation vector is ordered the same way as the state vector.

$$\mathbf{Q_x} = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{Q_y} = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{Q_z} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{4.6}$$

$$\mathbf{R}_x = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.65 & 0 \\ 0 & 0 & 3 \end{bmatrix} \quad \mathbf{R}_y = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.65 & 0 \\ 0 & 0 & 3 \end{bmatrix} \quad \mathbf{R}_z = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 3.5 \end{bmatrix} \tag{4.7}$$

The matrices **Q** and **R** define the default uncertainty of the prediction and the measurement which is added in each step to the current uncertainty of the system which is tracked in **P**. This means that the relation between these two matrices affects whether the prediction or the measurement is considered to be more reliable.

The matrices for the directions x and y are defined in the same way because the observation of these states and the basic model in the two directions are the same. As already mentioned the uncertainty values of the acceleration are lost in each step. In this way the prediction value damps the sensor measurement with a factor of about $\frac{1}{3}$. As the noise generated by the rotor rotation can be considered to be Gaussian this damping will produce acceptable results.

In direction z the values differ. The altitude controller produces higher peaks in this direction as this is the controlled direction. Secondly, the sensor values are more accurate for this Kalman filter except from the acceleration.

The basic idea to get these constants was to do static measurements to observe the noise occurring on the output of the different sensors. Unfortunately, as the telemetry is the limiting factor for this measurements no distribution tests could be made.

As an alternative method, standard deviation computation and the produced peaks were taken into consideration. The acceleration values were very well reproducible but had quite big standard deviation even in static case. By the flow sensor the repeatability was sometimes quite problematic as the sensor is very sensitive to light intensity and some settings produce significant change in the behavior. The biggest problem was the virtual position sensor in x and y directions as the integrator timestamp might not be accurate. Based on these tests the only reliable measurement values were produced by the ultrasonic sensor which measured the altitude of the quadrotor.

The constants in matrices **R** were computed based on the standard deviation and slightly adjusted taking the peaks into consideration. After that process the matrices **Q** were set in an experimental way. At the end both matrices were adjusted until the best results were produced.

# 5 Evaluation

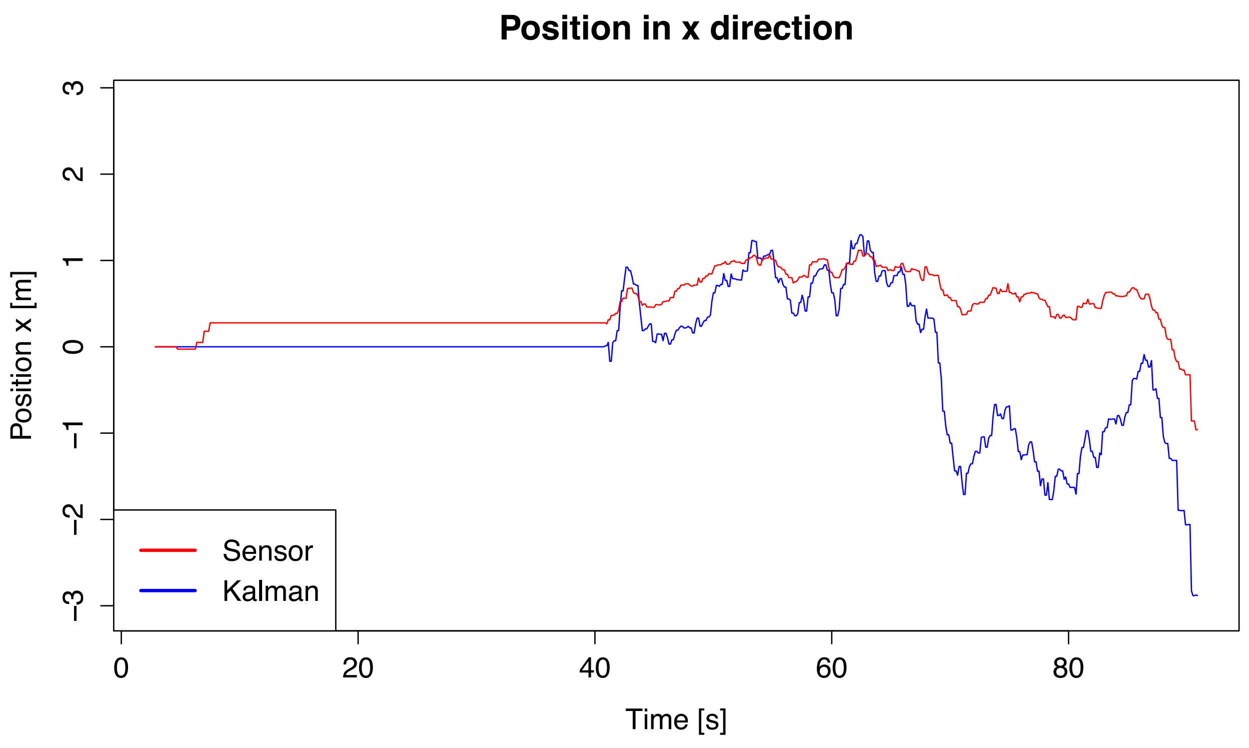

**Figure 5.1:** Sensor and Kalman filter data of position in x direction

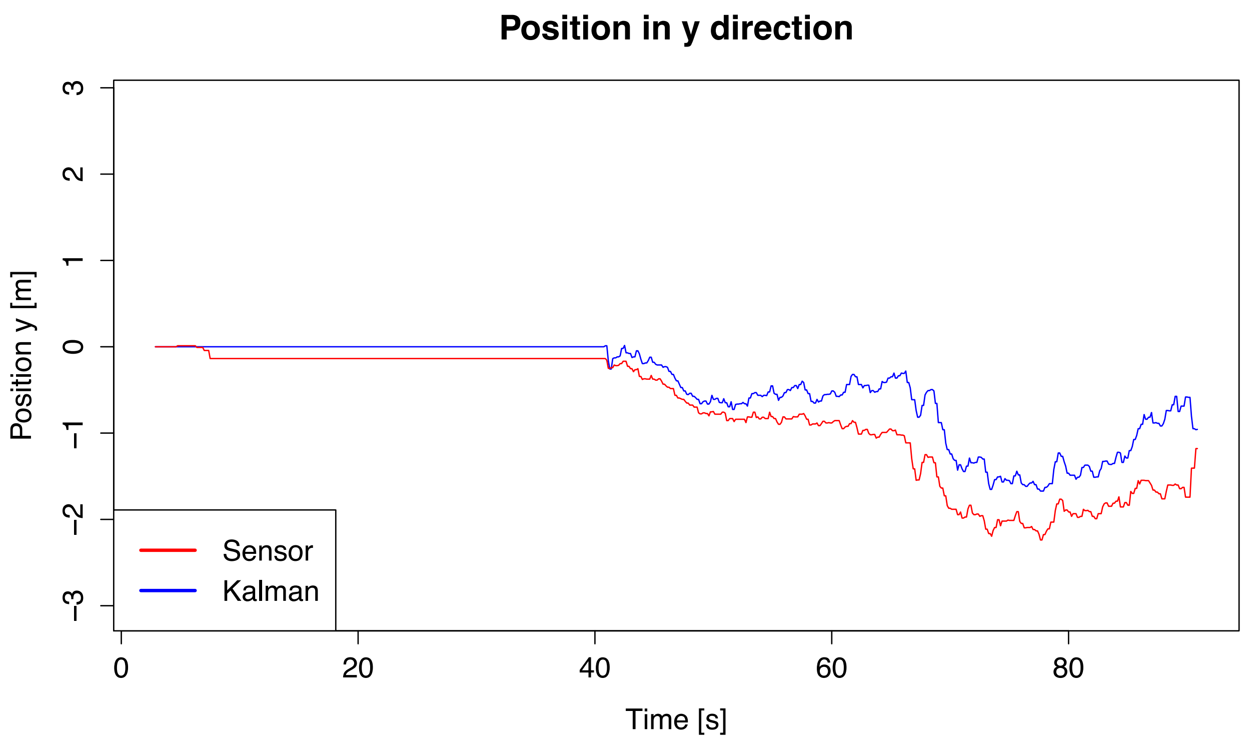

**Figure 5.2:** Sensor and Kalman filter data of position in y direction

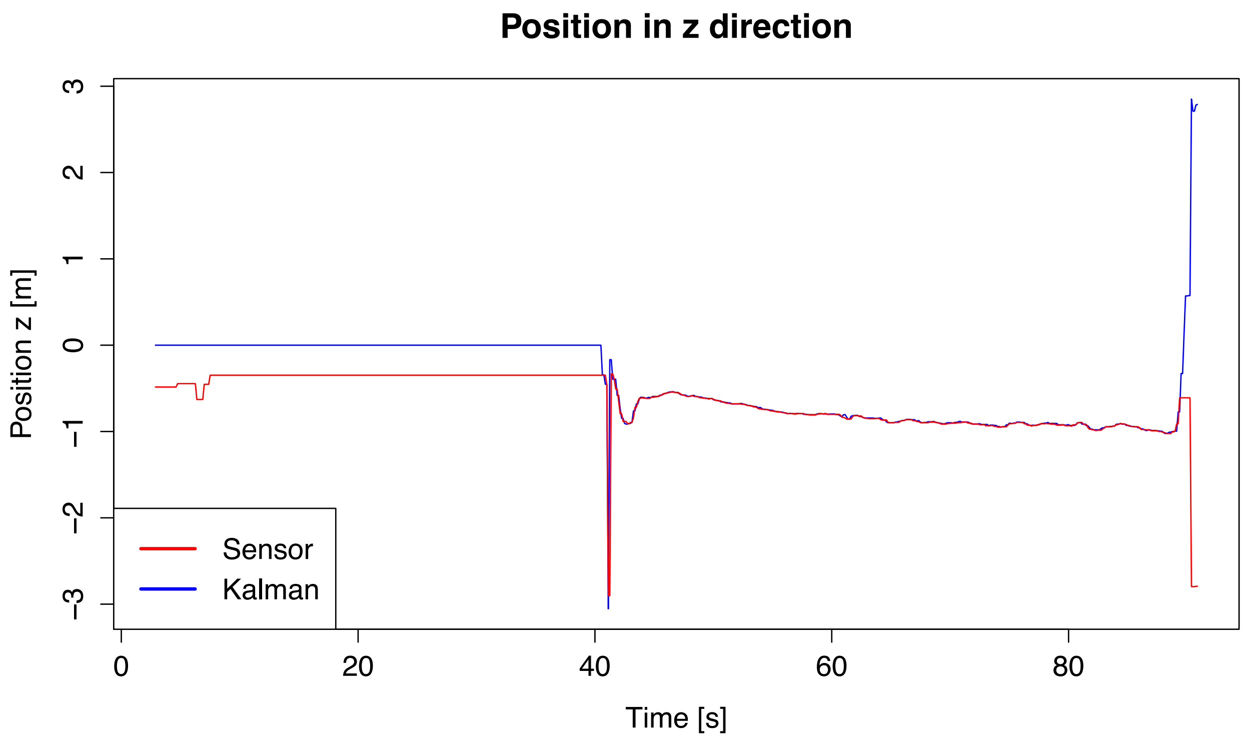

**Figure 5.3:** Sensor and Kalman filter data of position in z direction

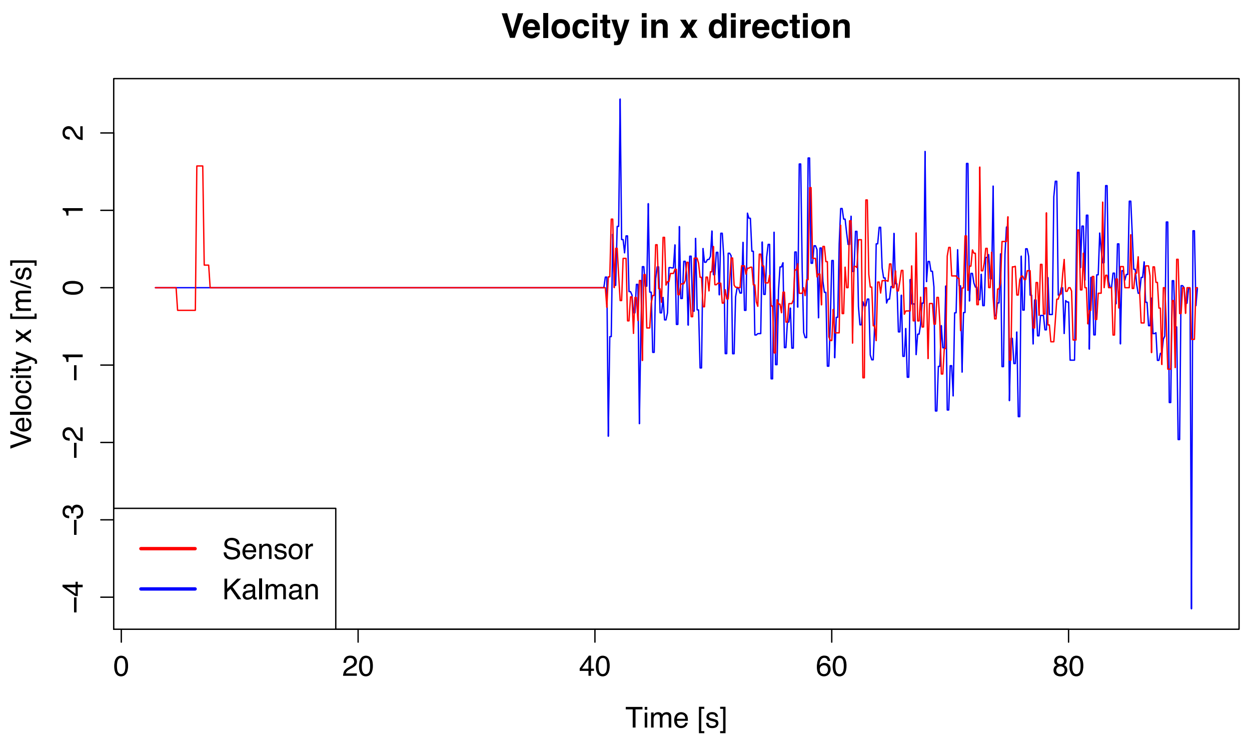

**Figure 5.4:** Sensor and Kalman filter data of velocity in x direction

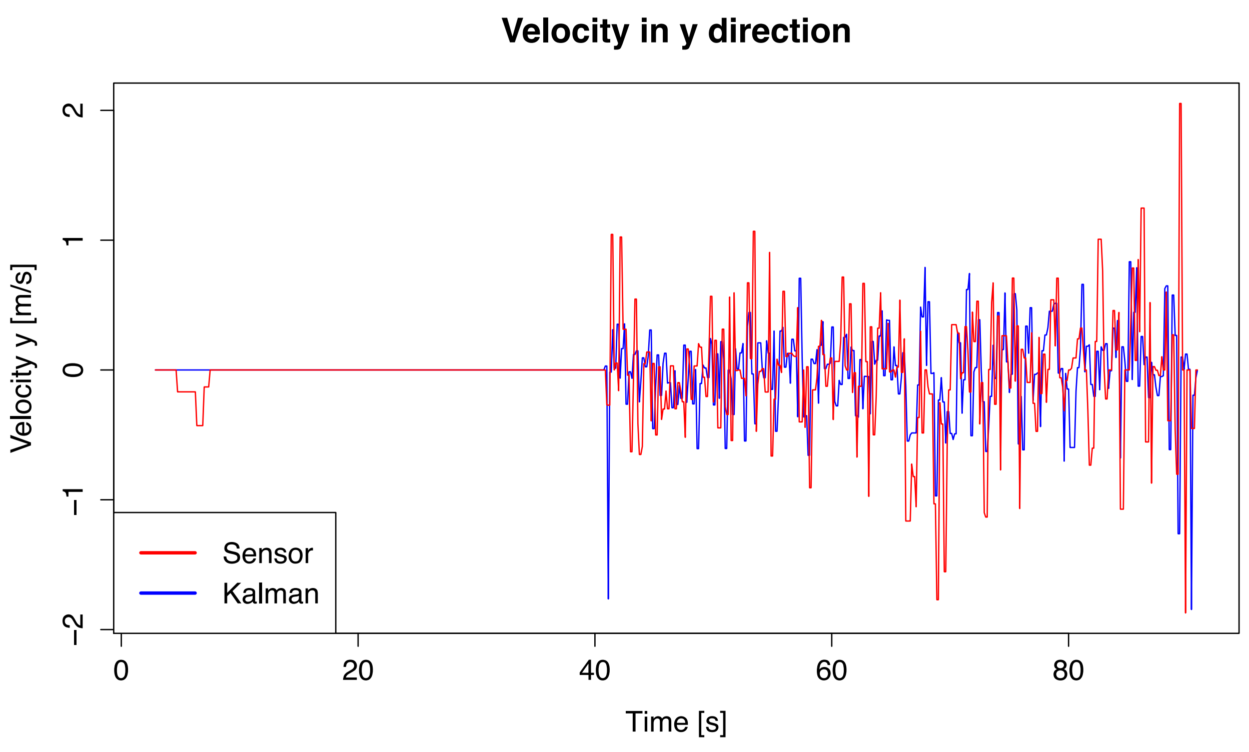

**Figure 5.5:** Sensor and Kalman filter data of velocity in y direction

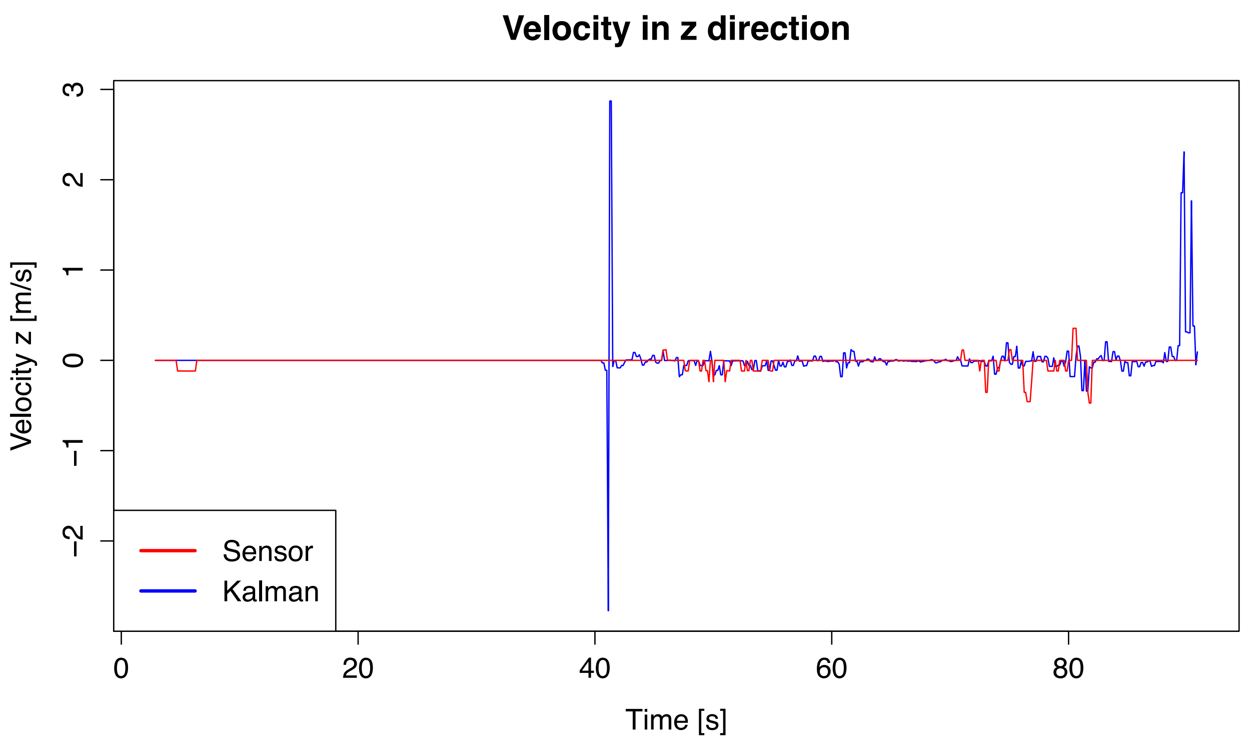

**Figure 5.6:** Sensor and Kalman filter data of velocity in z direction

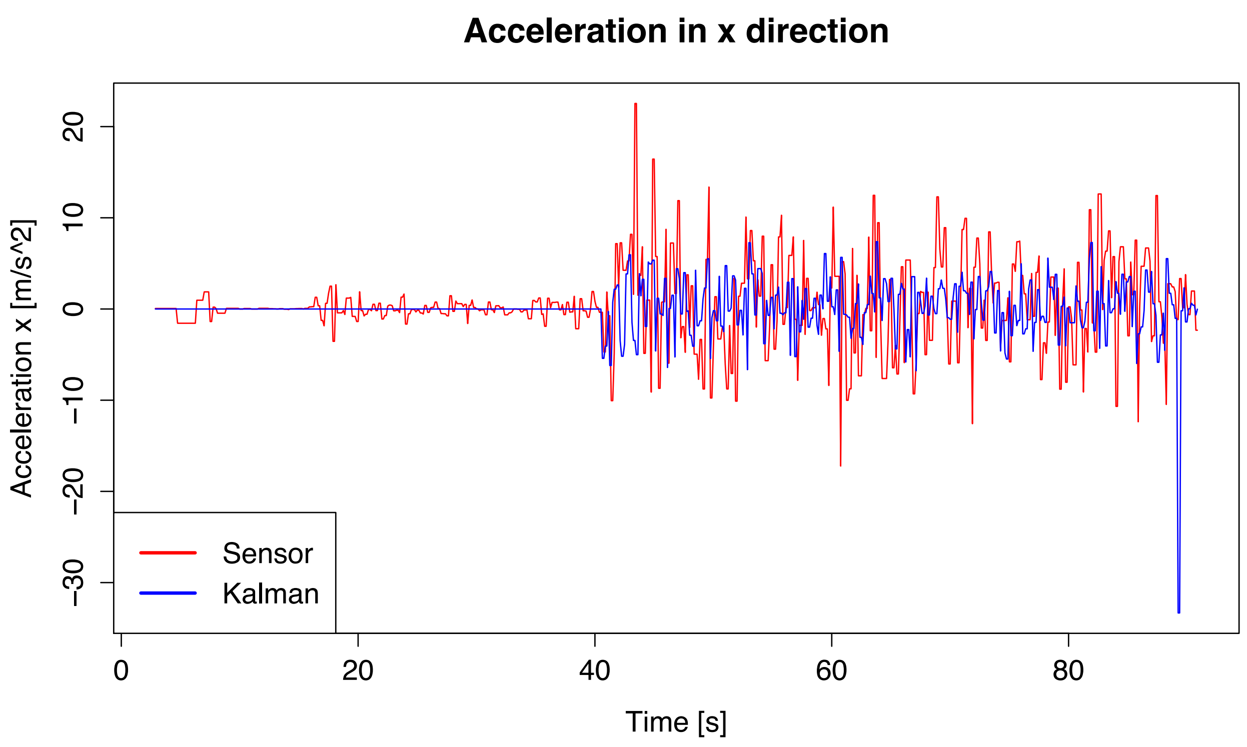

**Figure 5.7:** Sensor and Kalman filter data of acceleration in x direction

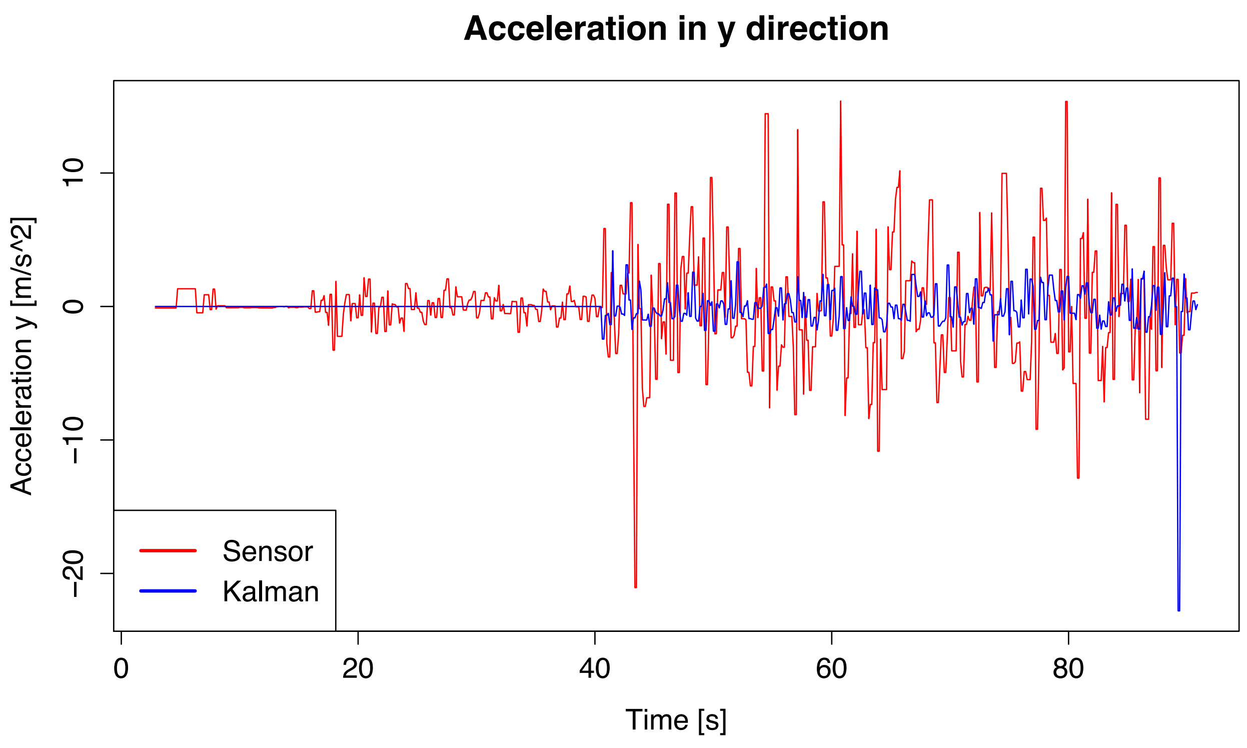

**Figure 5.8:** Sensor and Kalman filter data of acceleration in y direction

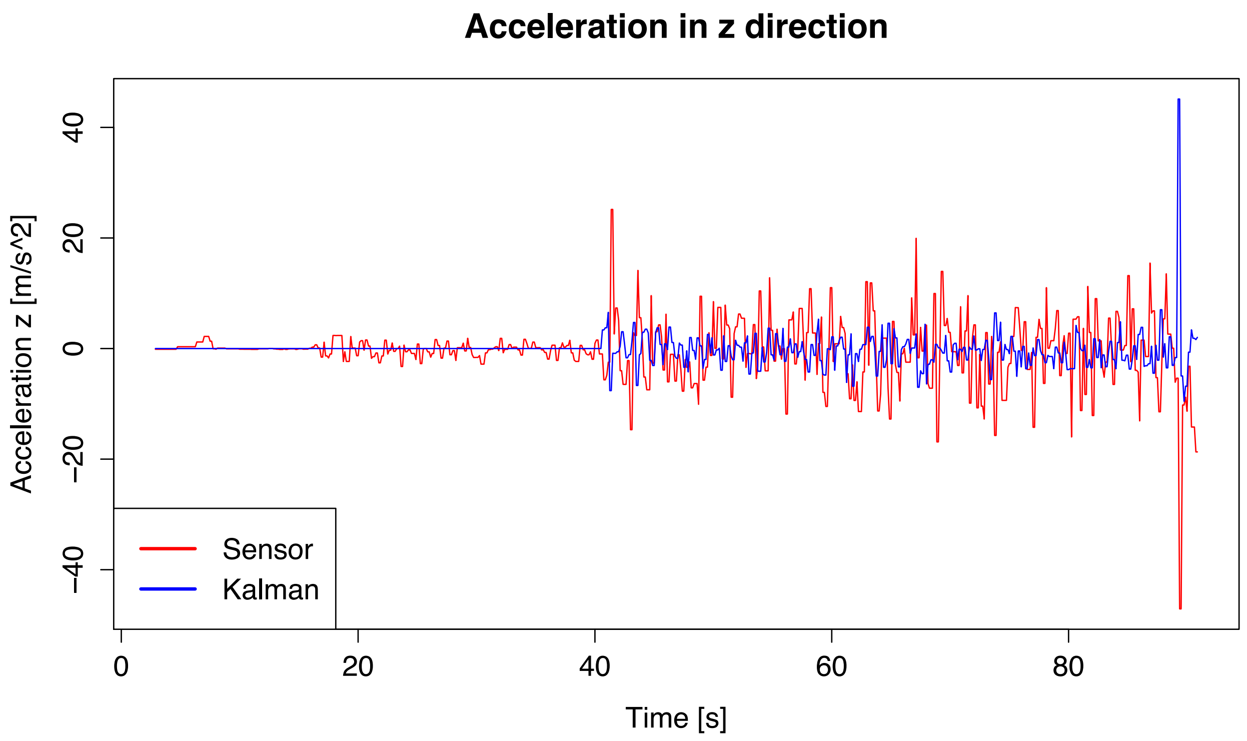

**Figure 5.9:** Sensor and Kalman filter data of acceleration in z direction

The figures from figure 5.1 to 5.9 show the evaluation results. First of all I must note that these are very good results compared to other tests because sometimes the module produces much worse results. This might be the result of some low-level modules not being initialized properly or crashing during execution.

On the graphs in the first 40 [s] time the Kalman module produces no results because in that time the module is waiting for the altitude controller to start. The sensor results show differences because they are already active. At the end at about 90 [s] the quadrotor fell down and flipped.

The sensor values are always in local NED coordinate system while the Kalman module is recomputing everything to the fixed global NED coordinate system. This produces differences on the diagrams. The yaw angle of the quadrotor cannot be tracked accurately that is why it was set to constant zero in both cases.

The acceleration results show the already described $\sim \frac{1}{3}$ damping ratio in most cases. At the end the altitude controller tries to compensate the decreasing maximal throttle by landing. This way it produces an outlier in every direction as the copter is flipping.

The velocity and the position in z direction are reasonable. By takeoff the controller tries to reach the required height as fast as possible. So it produces a strong controller output. The same way it tries to compensate the decreasing maximal throttle at the end of the flight. The position jumps to 3 [m] as the quadrotor senses the ceiling. The position tracking in the direction z is very accurate. In the velocity some outliers might occur every time as there a virtual sensor value is given to the Kalman filter which is a discrete derivation of the position.

In the x and y directions neither the Kalman filter nor the sensor model results are accurate. Two basic problems can be noticed. First the sensor measurements are not very realistic. The arena itself is at least 4 x 4 [m]. As I was flying back and forth in the arena the sensor model produced only 1 - 1.5 [m] position change. This can be caused for example by the following phenomena: If the module is not scheduled periodically by the core of the paparazzi, the motion is too fast for the sensor to be tracked or the pattern is not recognizable for the flow sensor.

The second problem is that the directions x and y differ somehow. As I did multiple measurements they always showed that in the y direction the Kalman is following the sensor values closer than in the x direction. This was independent from the direction of the flight. This might be caused by a scaling error or an improper configuration and/or calibration of the flow sensor. Unfortunately trying to reconfigure the flow sensor did not brought any change so I suppose that some low level modules are also involved in this conflict.

The Kalman filter takes the control input also into account. In this way it recognizes that the sensor values cannot be real and indicates higher position change. Unfortunately, as the linear system model implemented in the matrices **A** and **B** are not so close to the reality the corrected positions are also not good enough for the quadrotor to be tracked accurately.

When I let the quadrotor fly back and forth in the x or in the y direction one can notice a constant drift in the results in one direction. This is caused be some under sampling effect. As the position is predicted through a double integrator and corrected through a single integrator it will never be stable. The 15 [Hz] frequency which is the call frequency of the Kalman module is unfortunately not enough to track fast motions and this causes a drift, similarly to the case when I was flying the quadrotor back and forth.

# 6 Conclusion

At the end of the documentation I would like to summarize the results of the project.

Unfortunately the Kalman filter module cannot be used for motion state tracking in this state. This is the result of inaccurate sensor results and the unpredictable behavior of the quadrotor.

The constant drift in the flight of the quadrotor and the compensation signal influences the results of the Kalman filter module so a proper initialization of the 0 angles for pitch, roll and yaw is also required.

The linear system model cannot handle the tracking problem as the noise occurring in the system is not Gaussian. To overcome this issue an extended Kalman filter might be the solution which will require more computation power. In this case a better MCU might be required.

As all states of the copter are only observable through the current telemetry, an observation process with higher throughput might also be needed.

Unfortunately the current state of the quadcopter and the implementation cannot fulfill the goal of the project which was an accurate motion state tracking of the quadrotor but the results might be helpful in the further development of the FINken project and a more robust motion state tracking project in the future.

# 7 Acknowledgement

I would like to take this opportunity to express my gratitude and regards to my supervisor, Christoph Steup for his guidance, help and monitoring throughout the project. Especially for repairing the quadrotor every time it produced some strange error.

I would also like to say thank you to Sebastian Mai for his help during the project, Christoph Pahlke for the LUT construction and Martin Knoll for his Kalman filter example.

# Bibliography

[1] INC., W. F.: *Kalman filter.* https://en.wikipedia.org/wiki/Kalman_filter. Version: 2015

[2] MAYER, M. : *libfixkalman: Function reference.* https://github.com/sunsided/libfixkalman/blob/master/FUNCTIONS.rst. Version: 2015

[3] SINGHAL, T. ; HARIT, A. ; VISHWAKARMA, D. N.: Kalman Filter Implementation on an Accelerometer sensor data for three state estimation of a dynamic system.

[4] STILLER, P. D. I. C.: *Grundlagen der Mess- und Regelungstechnik.* Engler-Bunte-Ring 21, 76131 Karlsruhe, Germany, 2006