Hochschule Esslingen
University of Applied Sciences

Hochschule Esslingen
Fakultät Informationstechnik
Flandernstraße 101
73732 Esslingen

Development of a state space controller for a quadrocopter

Research paper SS 2011

Tutor: B. Eng. Dionysios Satikidis

Alexander Stoltz matric no.: 735596

Benjamin Jaißle matric no.: 735552

29. Juni 2011

Foreword

Imagine somebody gives you a remote control with four slide controls. In front of you, on the floor, you can see a helicopter without tail rotor, but four head rotors. Each slide control controls the engine speed of one rotor. Would you be able to steer this flying object? Certainly not. But with some help of a controller you will be able to steer this flying object, called quadrocopter. So, you need a controller - and this is, what this paper is about.

This research paper is the result of a student project, realized in SS2011 at the University of Applied Sciences Esslingen. The goal of this project was to design and implement a state space controller for a quadrocopter in MATLAB Simulink and we are glad to report success. We want to thank our tutor, B. Eng Dionysios Satikidis, for the assistance during the project, as well as Prof. Dr. Hermann Kull for the occasionally hints and his manuscript. Last but not least, we want to thank Prof. Dr. Jörg Friedrich, our examinor.

Content

1 Introduction 10

1.1 General . . . 10

1.2 V-Model . . . 11

2 Physical Model of the Quadrocopter 13

2.1 Earthframe and bodyframe . . . 13

2.2 Motion of the quadrocopter . . . 15

2.2.1 Thrust . . . 15

2.2.2 Pitch . . . 16

2.2.3 Roll . . . 16

2.2.4 Yaw . . . 17

2.3 Process variables . . . 18

2.4 Sensors . . . 19

3 Model of the Quadrocopter in MATLAB Simulink 20

3.1 Overview . . . 20

3.2 Remote control block . . . 22

3.3 Controller block . . . 23

3.4 Dynamics block . . . 24

3.4.1 Green section . . . 25

3.4.2 Red section . . . 26

3.4.3 Blue section . . . 27

3.4.4 Yellow section . . . 28

3.5 Sensors block 29

4 Theory of the state space controller 30

4.1 State space 30

4.2 State space controller 32

4.3 Pole placement 34

4.4 State observer 38

4.5 MIMO-Systems 39

5 Design and implementation in MATLAB/Simulink 41

5.1 Approach 41

5.2 State space representation 42

5.2.1 Controllability 42

5.2.2 Model reduction 43

5.2.3 Observability 44

5.3 Simulink 45

5.4 Pole placement 46

5.5 State variable feedback 48

5.6 Control deviation 50

6 Final tests 53

6.1 Simulation test 53

6.2 Hardware In the Loop test 58

6.3 Real quadrocopter test 58

7 Future prospects **59**

Literatur **60**

8 Appendix **61**

List of figures

1 customized V-Model for this project . . . 11
2 Earthframe (green) and bodyframe (red) . . . 13
3 Association between earthframe and bodyframe . . . 14
4 Thrust . . . 15
5 Pitch . . . 16
6 Roll . . . 16
7 Yaw . . . 17
8 closed loop abstract . . . 18
9 Overview Closed Loop . . . 20
10 Remote control block overview . . . 22
11 Stimuli . . . 22
12 Controller block . . . 23
13 Sections of the Dynamics block . . . 24
14 Green section . . . 25
15 Motor . . . 25
16 Red section . . . 26
17 Blue section . . . 27
18 Yellow Section . . . 28
19 Sensors block . . . 29
20 Simple example of a SISO process . . . 30
21 Principle of controllability . . . 32

22 Principle of observability 32

23 Simple process example including the state space controller 33

24 Step response of the process without the state space controller 34

25 Matrix view of the process 35

26 Matrix view of the process including the state space controller 35

27 Step response of the process including the state space controller with a pre-intensification of one 36

28 Step response of the process including the state space controller and a calculated pre-intensification 37

29 Simple process including a state space controller and an state observer 38

30 Principle model of a third order MIMO process including a state space controller 39

31 The five development steps 41

32 Model of one motor including the time delay 43

33 Model of one motor without the time delay 44

34 Overview over the whole state space controller 45

35 Transformation of the set values to pseudo forces 46

36 Pole/zero-map of the reduced quadrocopter process 47

37 Feedback part of the state space controller 50

38 Pre-intensification factors for the different control paths 52

39 awdwadwd 53

40 awdwadwd 54

41 awdwadwd 54

42 awdwadwd 55

43 awdwadwd 55

44 awdwadwd 56

45 awdwadwd 56

46 awdwadwd 57

47 awdwadwd 57

Abkürzungsverzeichnis

asd	asdsada

1 Introduction

1.1 General

As mentioned in the foreword, the goal of this student project is to design and implement a state space controller for a quadrocopter. But what is a state space controller? And what at all is a quadrocopter?

The first question is too complex to give a short overview. So it is answered in detail in chapter 4. But it is possible to give a short overview over the quadrocopter, although it is described in detail in chapter 2.

A quadrocopter is a four-engined aircraft, similar to a helicopter with four rotors. The four rotors are placed at the four 'edges' of the aircraft and allow the quadrocopter to start and land vertically.

Bearbeitungshinweis: Foto vom Quadrocopter

To achieve horizontal movement, the quadrocopter gets pitched, and by crossover speed up and slowdown of the propellers, it is possible to achieve vertical rotation. The quadrocopter, this project is based on, gets steered by a four-way remote control. The pilot is able to control the horizontal angles, the vertical rate of rotation and the average speed of the propellers. So the quadrocopter definetely needs an interface, to interpret the commands of the remote control and calculate the engine speed of every motor/propeller. In other words - an embedded controller is needed. State of the art is a cascaded PI controller, which works fine. Nevertheless it is reasonable to design and implement this mysterious state space controller, to see if it works as well as the PI controller or even better. Besides, a state space controller has some advantages over a classical PID controller, what is mentioned in detail in chapter 4.

The development of the controller is an engineering process, that needs structured proceeding, so the next step of this paper is a view on the project management and the structure of this document.

1.2 V-Model

The project was realized on the basis of the V-Model. The V-Model is a common graphical model to plan an engineering process, that get's customized for each project. In the following, the customized V-Model for this project will be explained.

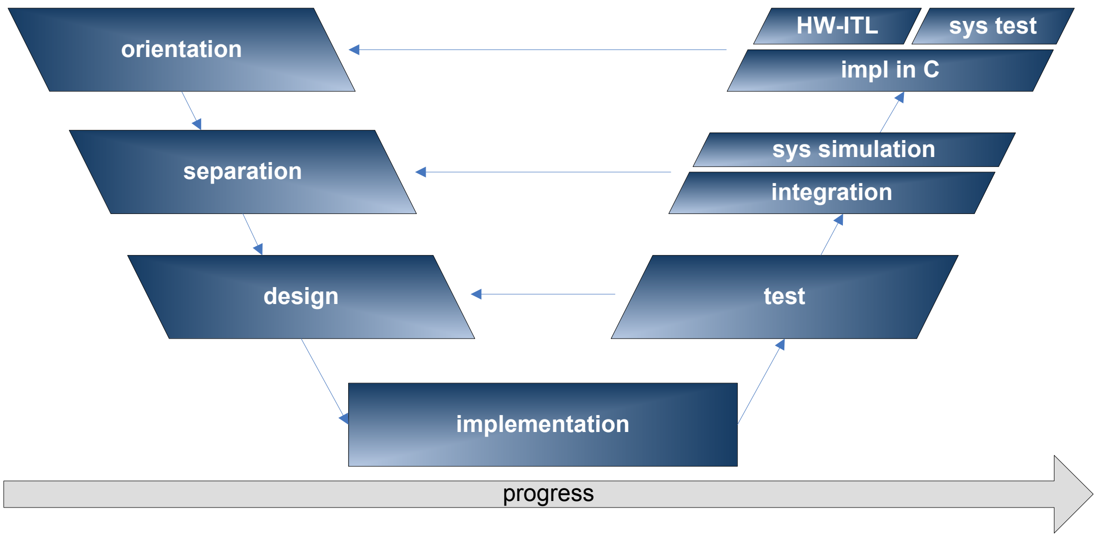

Figure 1: customized V-Model for this project

Due to, there is a solid basis, on which this project can be build on, the first thing to do is to learn the ropes. The main topic in this step is to understand the quadrocopter's physical model, just as its, already existing, model in MATLAB Simulink - and certainly the functionality of a state space controller. The next step is to separate the relevant part of the Simulink model, so it is possible to work at one 'module', which alleviates to keep the overview. On the basis of this separate part, the next step can be done, which is the beginning of the real project - the design of the state space controller. The lowermost crossbar is the main topic in this engineering process, which means the implementation of the state space controller in MATLAB.
The lowermost part of the right leg of the 'V' includes the test, in this case the simulation, of the implemented state space controller. If there are failures - for example unexpected behavior of a controlled variable, the way follows the lowermost arrow, which guides from the right to the left, back to the design of the state space controller. This circle gets run through, until there are no failures at all. Then the process climbs to the next step, in this case the integration of the state space controller into the original, non-separated Simulink model of the quadrocopter. Then testing, which means still simulation, starts again. If there are failures, the way follows the middle horizontal arrow back to the separation part, because apparently

the separated process and the original process don't match. If there are no failures in the system-simulation-part, the next step upward to 'implementaion in C' can be done. In this part the graphical model of Simulink gets converted in C-code which is executable by the microcontroller installed in the quadrocopter. This code has to be tested **H**ardware-**I**n-the-**L**oop (HIL) on the real microcontroller. This means, that the code is executed on the real microcontroller, but the output, in this case the actuating variables, gets looped back into the simulation in MATLAB Simulink, which safes ressources, because critical failures can be detected without risking a crash of the quadrocopter. Last but not least, if everything looks great in the HIL simulation, the state space controller gets tested and fine-tuned 'on the fly' with the quadrocopter.

This document leads through this whole engineering process chapter by chapter, starting with understanding the quadrocopter's physical model (chapter 2) and its model in MATLAB Simulink (chapter 3). The next step is to understand the background and layout of a state space controller in chapter 4. Then, with this knowledge, the next step is to design and implement the controller in MATLAB/Simulink in chapter 5 and run simulations in MATLAB Simulink to validate the functionality of the controller in chapter 6. After that, the implementation of the controller in C is the topic of chapterWX, leading to the HIL tests, just as the tests with the real copter in chapter 6. At the end of this document you can find some future prospects in chapter 7.

With the next chapter this documentation changes over to the engineering process, starting with the physics of the quadrocopter, the first step in the V-Model.

Bearbeitungshinweis: Kapitelnummern anpassen

2 Physical Model of the Quadrocopter

2.1 Earthframe and bodyframe

This - and the following - chapter discuss the behaviour of a quadrocopter. To achieve a better understanding, graphics with a simple model of the quadrocopter are used. This simple model consists of a middle cross, and the four rotors with arrows, which show the direction of rotation. The rotor at the front is marked in yellow. Placing a coordinate system on the cross of the quadrocopter, the x-axis points to the front, the y-axis to the left and the z-axis points upwards. This coordinate system is called the bodyframe and shown in red in the picture below.

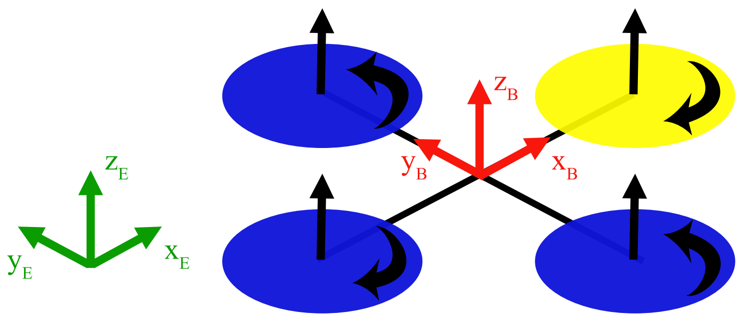

Figure 2: Earthframe (green) and bodyframe (red)

The green coordinate System is called the earthframe. This coordinate system is fix and independent of the movement of the quadrocopter. It describes the position of the copter, whereat the copter is interpreted as a point. It also describes the associated velocities and accelerations in direction of x, y and z of the earthframe. So the earthframe does not contain information about the attitude of the quadrocopter. This information is held by the bodyframe, that includes the angles of the coordinate system in relation to the earthframe, as well as the angular rates and -accelerations. It also holds information about the movement in direction of the axis of the bodyframe coordinate system, wherat the movement in direction of x and y of the bodyframe is a side effect of gravity, while movement in direction of z can be controlled directly.

So, 'sitting' on the bodyframe of the quadrocopter, it is not possible to know the position in space, but it is possible to know the deviation from horizontal in degrees as well as the vertical speed of spinning. Sitting on the eartframe, it is possible to see where the quadrocopter is, and how fast it is moving, but it is impossible to know if it is perhaps flying upside down.

Though these two frames hold different information, there is a mathematical association in between. This is explained in a simple two-dimensional example below.

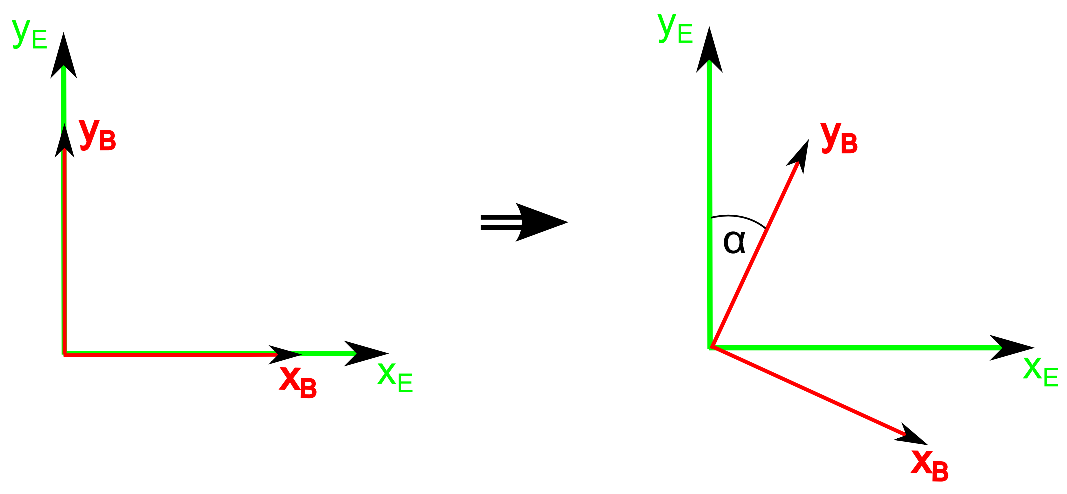

Figure 3: Association between earthframe and bodyframe

In the left picture, the two-dimensional bodyframe is congruent with the earthframe. If there is a constant velocity v_{yB} in direction of y_B:

$$v_{yE} = v_{yB}$$

Furthermore

$$v_{xE} = 0$$

In the second picture, the bodyframe is rotated a constant angle alpha; v_{yB} is still constant. Now it is:

$$v_{yE} = cos(\alpha) * v_{yB}$$

and

$$v_{xE} = sin(\alpha) * v_{yB}$$

So it is possible to transform coordinates of the bodyframe into coordinates of the earthframe and vice versa.

2.2 Motion of the quadrocopter

Theoretically a quadrocopter has six degrees of freedom:

1. Moving in direction of x (forward, backward)
2. Moving in direction of y (left, right)
3. Moving in direction of z (up, down)
4. Rotation around the x-axis, called 'Roll'
5. Rotation around the y-axis, called 'Pitch'
6. Rotation around the z-axis, called 'Yaw'

Effectively there are only four degrees of freedom, because movement in direction of x and y is only a side-effect by gravity, like it is described in chapter 2.1. These four degrees of freedom are exemplified in the following.

2.2.1 Thrust

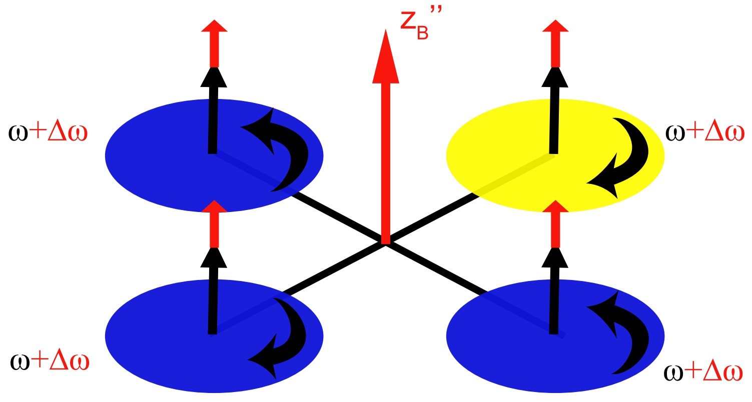

Figure 4: Thrust

If all rotors are spinning with constant angular rate ω and a new offset $\Delta\omega$ is added, the ascending force of each rotor increases in same way, which results in a force vector, pointing in direction of z of the bodyframe. So the quadrocopter accelerates in direction of z.

2.2.2 Pitch

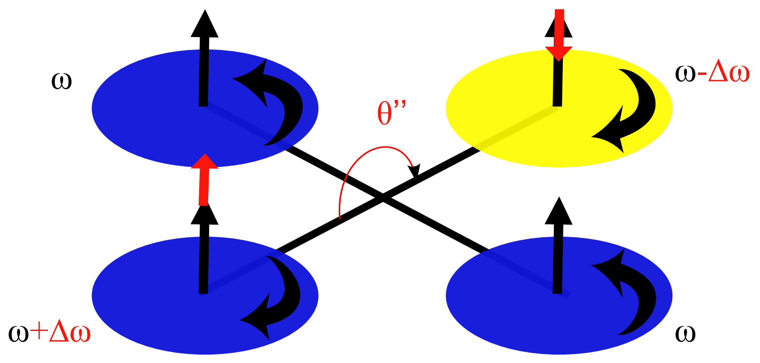

Figure 5: Pitch

To achieve forward-pitching, the rotor in front is slowed down, what means a negative offset $\Delta\omega$. In addition a positive offset is added to the rear motor. The result is a change of the angular acceleration and therefore a change of the angle θ (theta).

2.2.3 Roll

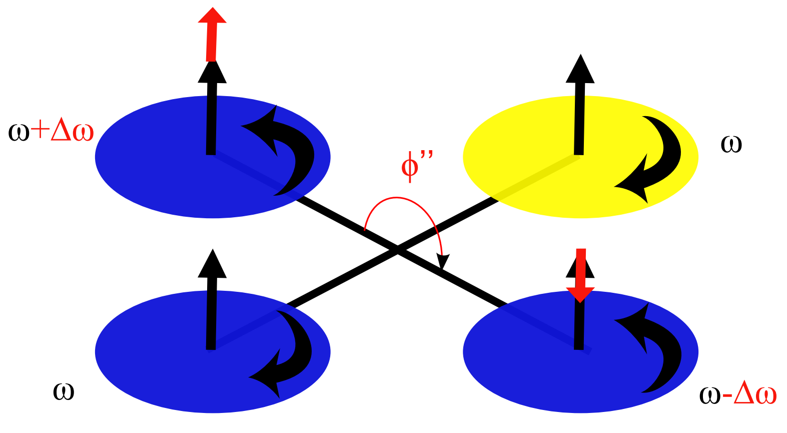

Figure 6: Roll

The principle to get a roll movement is the same one, like it is used for the pitch movement. To achieve a roll movement to the right, a negative offset $\Delta\omega$ is added to the rotor on the right side of the quadrocopter. In addition a positive offset is added to the left rotor. Again the result is a change of the angular acceleration and therefore a change of the angle Φ(phi)

2.2.4 Yaw

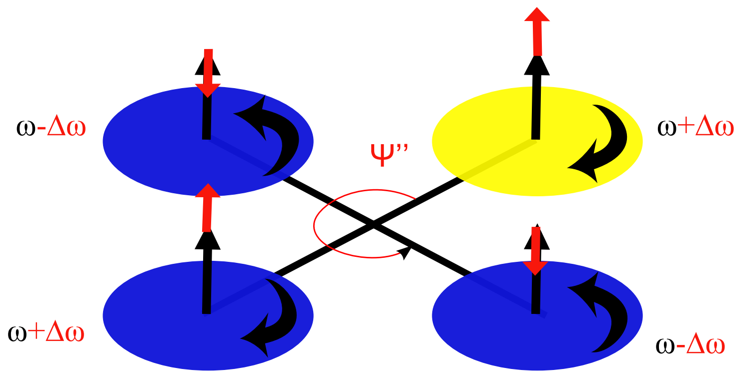

Figure 7: Yaw

To achieve rotation around the z-axis, called 'yaw' is more tricky than the roll and pitch movements. It is visible in the grapics, that the rotors at the front and the back of the quadrocopter spin clockwise, while the two rotors on the side of the quadrocopter spin counterclockwise. This is necessary because of the inertia of the propellers. If all propellers would spin clockwise, the quadrocopter would steadily rotate counterclockwise. This effect is used, to rotate the quadrocopter. To achieve a rotation counterclockwise, a positive offset is added to the front rotor and the rear rotor. To intensify this effect, the left and the right rotor, which hold up, are slowed down by adding a negative offset to their motor speed. The result is a vertical, counterclockwise rotation Ψ(psi).

2.3 Process variables

This chapter gives a short overview over the controlled process variables, the actuating variables and the set points. The grafic below shows, where these values are placed in the closed loop.
The first rectangle represents the controller, that is designed and implemented in this project. The second rectangle represents the quadrocopter; the process that has to be controlled.

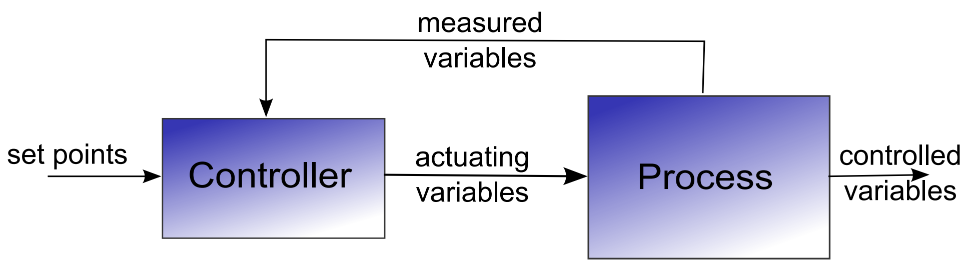

Figure 8: closed loop abstract

The set points are equal to the controlled variables. The pilot is able to control the angle of *phi* (roll), the angle of *theta* (pitch) and the *rate of psi* (yaw). In chapter 2.2, one degree of freedom is 'thrust', which in some way also is a set point, but it is not a controlled variable. Actually it is an offset, that is added to the actuating variables, that have been calculated by the controller before. These actuating variables are some sort of normalized forces, every propeller has to provide. They are called 'pseudo forces' in this document.
Last but not least, there are the measured variables, that are needed by the controller. These variables are content of the next chapter.

2.4 Sensors

Without of sensors it is not possible to control any process. The quadrocopter carrys two sensors:

1. A gyro-sensor, that provides angular rates in direction of *phi*(roll), *theta*(pitch) and *psi*(yaw).
2. A linear-acceleration-sensor, that provides accelerations in direction of x, y and z of the bodyframe.

The measured variables of the first sensor are used directly by the controller. The variables of the second sensor - the linear-acceleration-sensor - are used to calculate the real angles *phi* and *theta*. The angle of *psi* is not relevant for the controller, so the z-value of the sensor is not relevant as well.

3 Model of the Quadrocopter in MATLAB Simulink

The topic of this chapter is the graphical model of the quadrocopter in MATLAB Simulink. Actually it describes the whole control loop, including the controler and the sensors, called closed loop. While the first subchapter shows an overview over the complete loop, each part of the loop is discussed seperately in the following subchapters.

3.1 Overview

This chapter leads once through the control loop, that is pictured below.

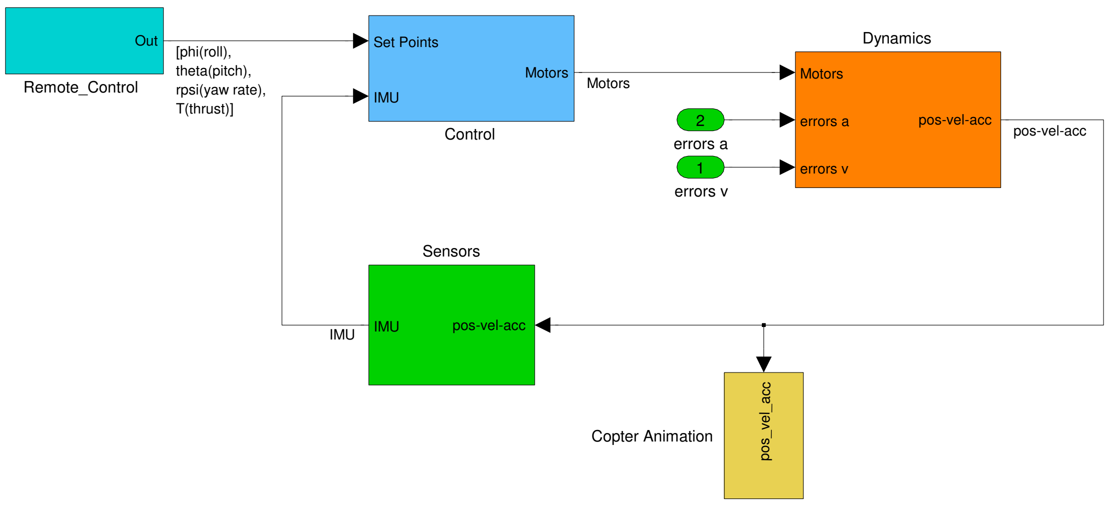

Figure 9: Overview Closed Loop

The cyan-colored block top left, represents the remote control. This block 'generates' the set points. These set points are combined in one vector, consisting of:

1. the angle of *phi* (roll)
2. the angle of *theta* (pitch)
3. the angular rate of *psi* (yaw rate)
4. the average motorspeed (thrust)

The next block, colored blue, is the controller block. The state space controller, developed in this project, gets implemented in this block. In there, the set points, given by the first block, get 'converted' into actuating variables. Again these variables are combined in one vector, consisting of the four peudo forces.
This vector is connected to the input of the next (orange) block, that represents the quadrocopter. Symbolic blocks represent the physical characteristics of the copter. So this block calculates, how the speed of each motor affects the movement of the copter. The result is a big vector called 'pos-vel-acc', which stands for 'positions-velocities-accelerations'. So this vector consists of all states of the quadrocopter, meaning the actual angle of *phi*, *theta*, *psi*; its angular *rate of phi*, *theta*, *psi*; its actual position in x, y, z of the earthframe, and so on. The complete vector is pictured in chapter 3.4. This vector is the input of the next block of the closed loop - the sensors block, pictured in green - which returns the measured variables, combined in the vector 'IMU'. This vector consists of:

1. acceleration in direction of x of the bodyframe
2. acceleration in direction of y of the bodyframe
3. the angular *rate of phi* (roll rate)
4. the angular *rate of theta* (pitch rate)
5. the angular *rate of psi* (yaw rate)

Also connected to the vector 'pos-vel-acc' is the yellow block at the bottom of the model. This block includes the animation of the quadrocopter. Though, this is not a real part of the control loop, it is an important assistance for testing the controller, because it allows to fly the quadrocopter virtually.

The next subchapters will walk once through this whole process, described above, starting with the remote control block.

3.2 Remote control block

Figure 9 shows an overview over the whole process. This chapter will show the 'innards' of the first block, pictured in cyan. Double-clicking the mentioned block openes an other MATLAB Simulink file (*.mdl-files) showed below.

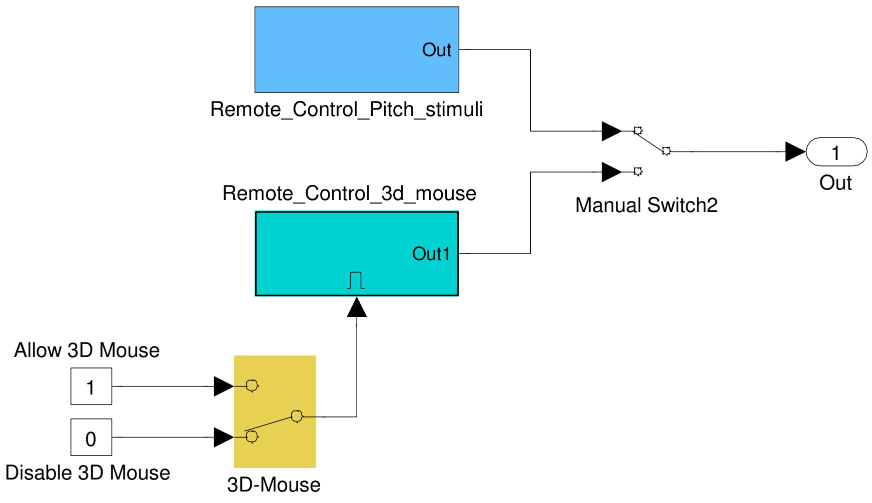

Figure 10: Remote control block overview

The cyan block in this figure, allows to connect a 3D-mouse to the computer to steer the quadrocopter in the animation. This research paper doesn't elaborate on this block. The block in light-blue allows to generate manual stimuli. The content of this block is showed below.

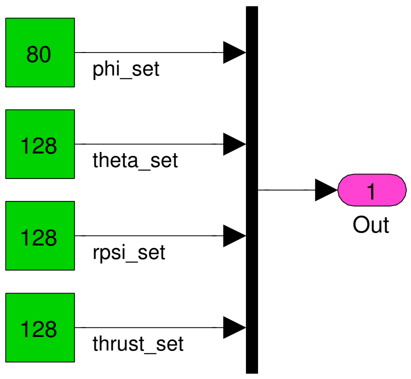

Figure 11: Stimuli

By double clicking the green blocks, it is possible to change the values of these blocks. The values range from 0 to 255, where 128 means angle zero or rather angular rate zero. Value 255 of thrust means full-speed to all motors. So the stimuli in the left figure say 'no pitching, no yaw rate and a negative angle towards the horizontal (roll left) at half thrust'.

3.3 Controller block

Double clicking the light blue 'Control' block in figure 9 opens the figure below.

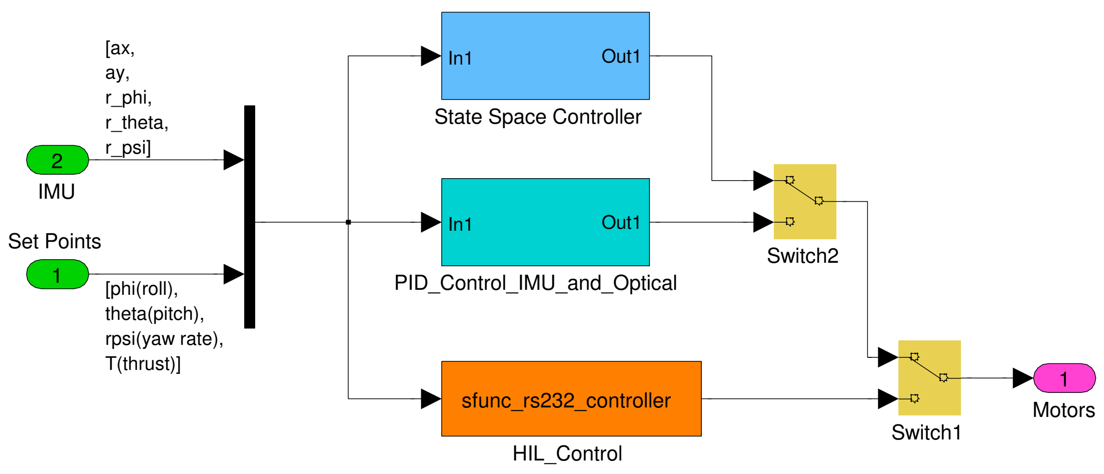

Figure 12: Controller block

The controller block has two inputs, 'Set Points' and 'IMU'. The black bar combines them to one vector with the elements:

1. *ax* (acceleration in direction of x of bodyframe; IMU)
2. *ay* (acceleration in direction of y of bodyframe; IMU)
3. *rphi* (rate of phi; IMU)
4. *rtheta* (rate of theta; IMU)
5. *rpsi* (rate of psi; IMU)
6. *phi* (Set point of phi)
7. *theta* (Set point of theta)
8. *rpsi* (Set point or rate of psi)
9. *thrust* (Set point of thrust)

This vector is the input of three blocks. The topmost, light blue block is the new state space controller, implemented in this project. The second block, pictured in cyan, is the, already existing, PID-controller. The lowermost, orange block is the block, that is used to test the quadrocopter hardware in the loop (HIL). The two switches on the right side define, which controller is enabled or rather whose actuating variables are connected to the prozess. The output of each controller block are the 'pseudo forces', every propeller has to provide.

3.4 Dynamics block

The MATLAB Simulink block, that includes the physical model of the quadrocopter, pictured in orange in figure 9, is the topic of this chapter. Due to, this model doesn't fit on one page, it is split up into multiple sections. The picture below shows the whole process, divided into four areas. The following chapters discuss these areas as appropriate. It is not necessary to discuss each block in detail, to be able to develop a controller for this process.

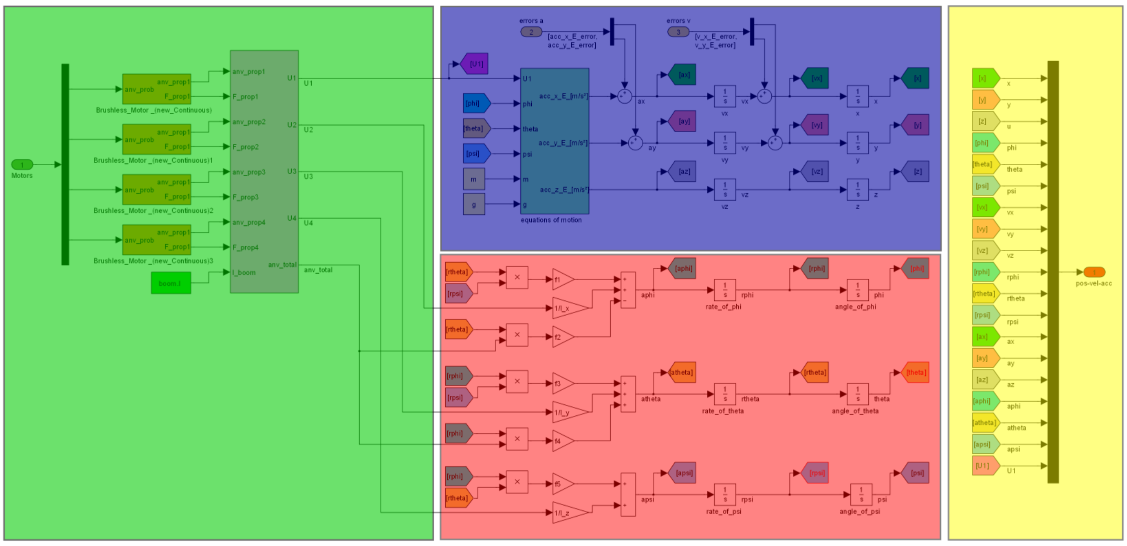

Figure 13: Sections of the Dynamics block

3.4.1 Green section

The green section has one input vector. It holds the pseudo forces, every propeller has to provide, given by the controller.

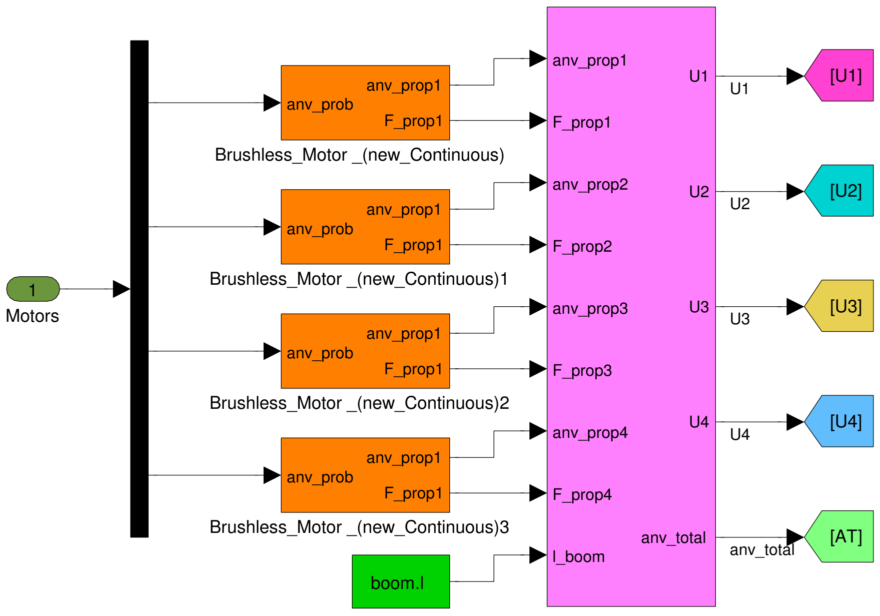

Figure 14: Green section

Each pseudo force is the input of one motor (orange blocks). Double clicking on one of the motors opens the graphic below.

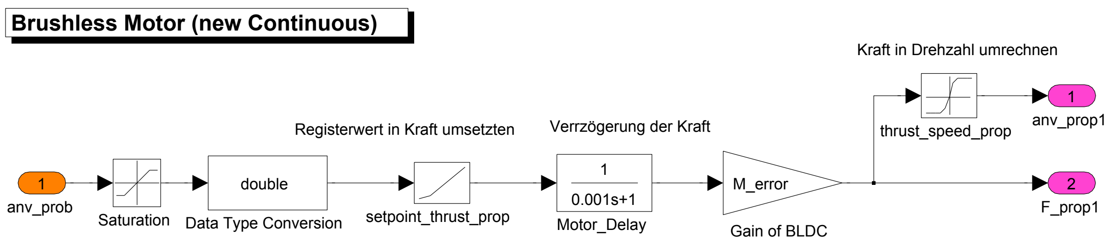

Figure 15: Motor

The pseudo force, coming from the controller, gets converted into a real force by the characteristic diagram 'setpoint_thrust_prop'. Due to the inertia of the propeller this force is delayed by a PT1-element. The motor has two outputs, *anv_propX* and *F_propX*, where *F_prop* is the force, the propeller produces vertically to itself. *anv_prop* is the rotary speed of the propeller, that gets calculated by the characteristic diagram 'thrust_speed_prop'.

The big purple block in figure 14 uses these forces to calculate the angular accelerations in direction of phi (U2), theta(U3) and psi(U4). For this calculation the lever principle (force * lever) is used. The small green block *boom.l* provides the information about the levers. The purple block also has two other output values *U1* and *AT*. *U1* is the sum of all propeller forces *F_propX*, *AT* is the sum of all rotary speeds *anv_propX*.

3.4.2 Red section

This section represents the bodyframe of the quadrocopter and consits of three very similar lines.

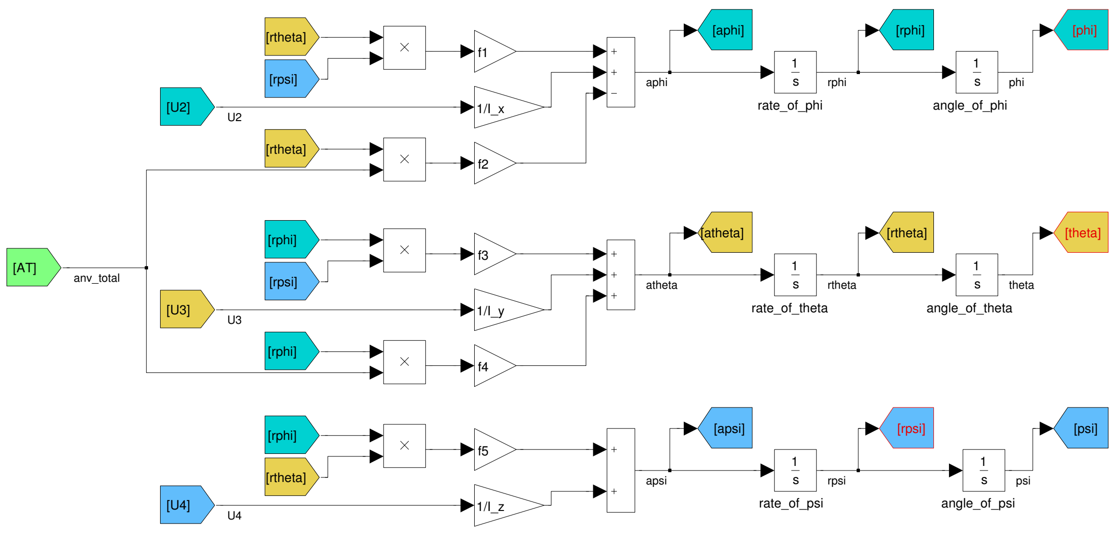

Figure 16: Red section

The topmost line uses the rate of theta *rtheta*, the rate of psi *rpsi*, the angular acceleration in direction of phi *U2* and the sum of all rotary speeds *AT* to calculate the resulting angular acceleration in direction of phi *aphi* of the physical model. This shows, that movement in direction of *phi* is influenced by movement in direction of *theta* and *psi*. This is momentous for the development of the controller.
The angular acceleration gets integrated once, resulting in the angular rate of phi *rphi*, and twice, resulting in the angle *phi*. The other two lines for *theta* and *psi* are very similar, so this is not discussed here in detail. The controlled variables *phi*, *theta* and *rpsi* are pictured in red letters.

3.4.3 Blue section

While the red section represents the bodyframe, this section represents the earthframe.

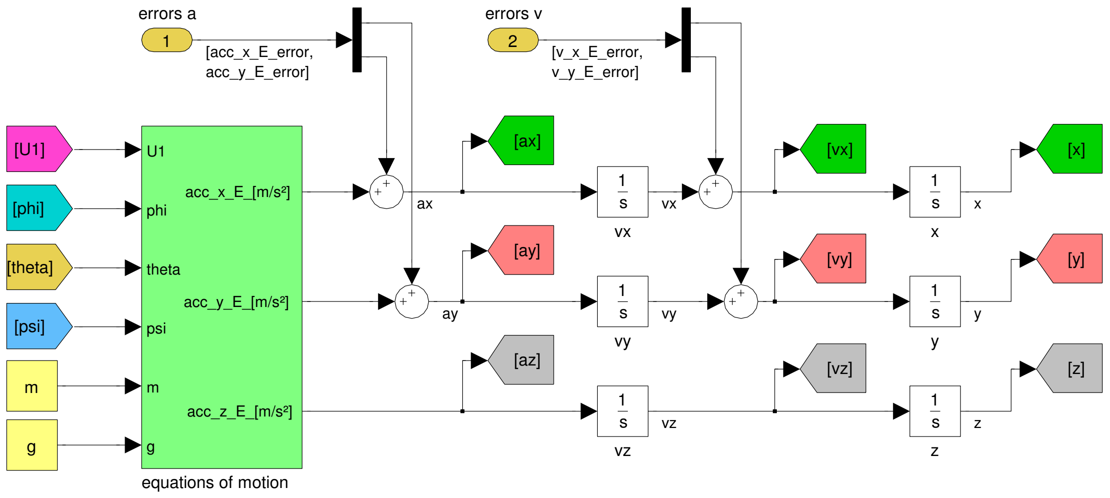

Figure 17: Blue section

At the ’input’ of this section is one big block, ’equations of motion’. It uses the ’output’ of the red section, *phi*, *theta* and *psi*, plus the sum of all propeller forces *U1* and two constants m (mass of the quadrocopter) and g (gravitational acceleration) to calculate the accelerations in direction of x, y and z of the earthframe. These values get integrated to gain the linar speeds and the position of the quadrocopter. By using the inputs ’error a’ and ’error v’, it is possible to simulate wind.

3.4.4 Yellow section

The yellow section shows the output vector of the dynamics block, called *pos_vel_acc* (positions-velocities-accelerations).

- position x of earthframe
- position y of earthframe
- position z of earthframe
- angle *phi* of bodyframe
- angle *theta* of bodyframe
- angle *psi* of bodyframe
- linear speed in direction of x
- linear speed in direction of y
- linear speed in direction of z
- angular speed in direction of *phi*
- angular speed in direction of *theta*
- angular speed in direction of *psi*
- linear acceleration in dir. of x
- linear acceleration in dir. of y
- linear acceleration in dir. of z
- angular acceleration in dir. of *phi*
- angular acceleration in dir. of *theta*
- angular acceleration in dir. of *psi*
- sum of all propeller forces

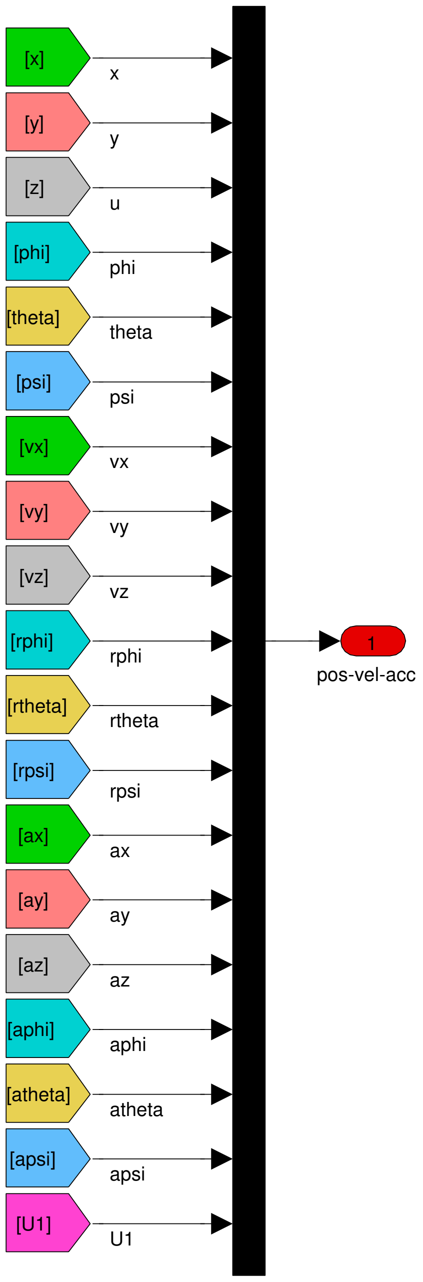

Figure 18: Yellow Section

3.5 Sensors block

The first important hint in this chapter is, that the input is on the right side of the model, because the sensors are in the backward line of the closed loop in figure 9 (green block).

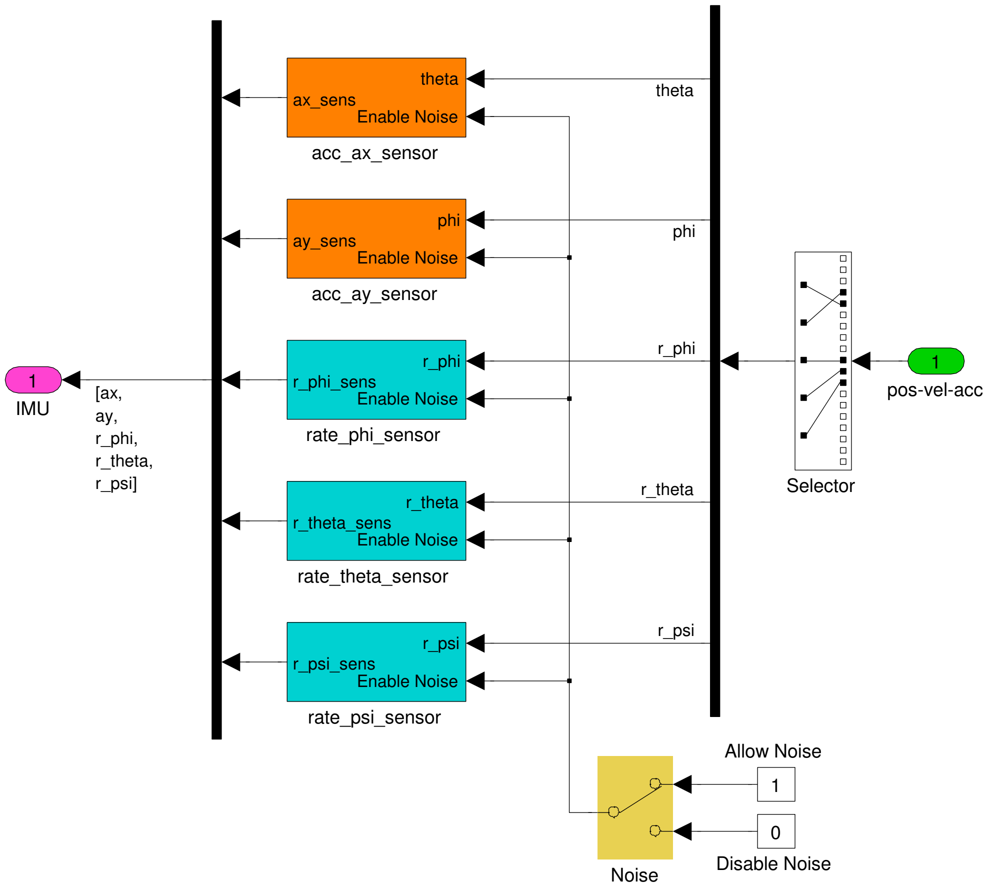

Figure 19: Sensors block

The first block after the input is a line-selector, that picks the elements out of the *pos_vel_acc*-vector, that get measured in the real copter by the sensors. These values are the inputs of each sensor. The upper, orange blocks represent the linear acceleration sensors, that measure the accelerations in direction of x and y of the bodyframe. These accelerations are not contained in the *pos_vel_acc*-vector. So, they have to be calculated in the (model of the) sensor, using the angle and a trigonometric function. The angular velocities are included in the output vector of the dynamics block, so they get directly 'measured' by the sensors. All measured values are combined in the output vector *IMU* of the sensors block.

4 Theory of the state space controller

4.1 State space

In this section, the basic knowledge for understanding the so-called state space representation, is described. The state space representation is a mathematical system of a physical model. It consists of inputs and outputs and the state variables that are related to the differential equations of that model. One huge advantage of this idea is, that the Laplace transformation is not needed. Therefore the state space representation is also called 'time-domain approach'. Another opportunity is the usage of initial conditions and that a system can have nonlinear constraints.

So the next step is to understand the state variables. As already mentioned, every state space system consists of inputs, outputs and state variables. These state variables describe the entire state of the system at any time. The simplest way to decide where to set the state variables is, to put them behind every single integrator block. So there is the state variable itself behind this block and the corresponding derivative in front. Figure 20 shows a simple example for a third order system. This system is called a SISO system, that means 'Single Input Single Output'. In this case u is the single input and y the single output of the system. Besides, there are MIMO systems having 'Multiple Input Multiple Output'. Chapter 4.5 handles them.

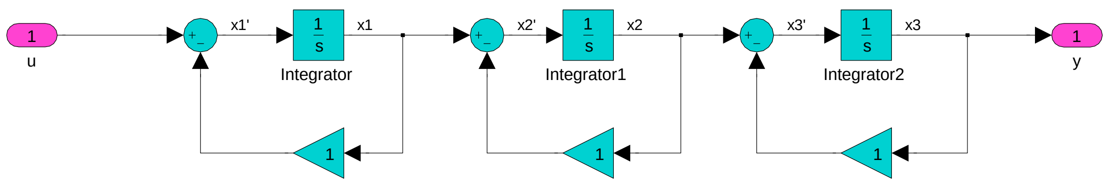

Figure 20: Simple example of a SISO process

The differential equations for this system look like that:

$$x1' = -x1 + u$$
$$x2' = x1 - x2$$
$$x3' = x2 - x3$$
$$y = x3$$

These differential equations can be transformed into four matrices:

$$\begin{array}{c|cccc|} & x1 & x2 & x3 & u \\ x1' & -1 & 0 & 0 & 1 \\ x2' & 1 & -1 & 0 & 0 \\ x3' & 0 & 1 & -1 & 0 \\ y & 0 & 0 & 1 & 0 \end{array} \tag{1}$$

The most important is the A matrix, called the 'state matrix'. This matrix declares the dependency between all state variables in the whole system. With this matrix it is possible to get information about the response characteristics and the stability of the system. The B matrix is called the 'input matrix', the C matrix is called the 'output matrix' and at least the D matrix is called the 'feedforward matrix'.

In most cases - as well as in this project, there is no direct connection between an input and an output. Therefore the 'D matrix' is a zero matrix. The input matrix B describes the dependency of the state variables to the input vectors. This fact is needed later on, to decide whether the system is controllable. The output matrix C describes the same as the input matrix, but for the outputs. The C matrix is used to declare which states are observable.

Now two more terms are mentioned - controllability and observability.

The controllability first. To achieve a required dynamic behavior of the process, it is necessary, that all of its poles can be moved independently by the input.

$$Q_s = \begin{bmatrix} b & A*b & A^2*b & \cdots & A^{n-1}*b \end{bmatrix} \tag{2}$$

According to Kalman, the controllability matrix Q_s (equation 2) have to be full rank of the whole system. This is the most important check in the beginning of any project about state space controlling. For the simple example process the controllability is given, because the rank of the Q_s matrix is three and that system is third order. Figure 21 shows the controllability graphical.

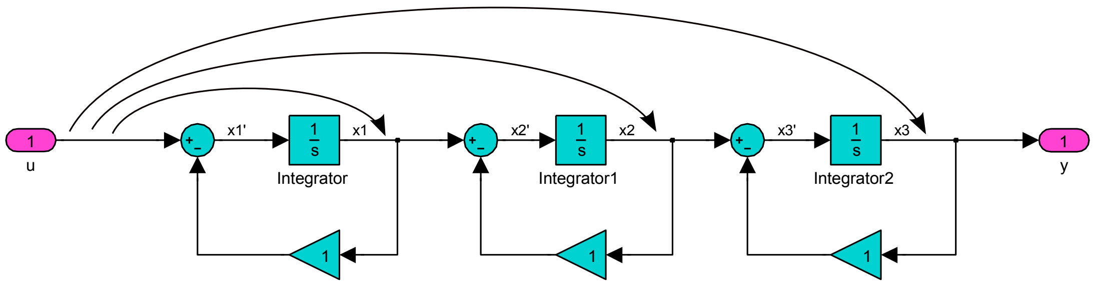

Figure 21: Principle of controllability

The second term mentioned is observability. It is very similar to the controllability, but it declares whether each state variable has influence on the output of the system.

$$Q_b = \begin{bmatrix} c^T & c^T * A & c^T * A^2 & \cdots & c^T * A^{n-1} \end{bmatrix}^T \tag{3}$$

The condition according to Kalman is very akin to the condition for controllability - the observability matrix Q_b (equation 3) must be full rank of the system. The rank of the Q_b matrix in this example is three - so each state is observable. Figure 22 shows the observability graphical.

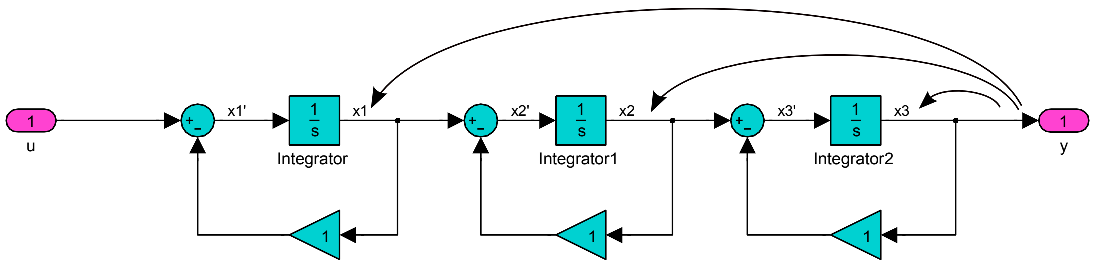

Figure 22: Principle of observability

4.2 State space controller

In chapter 4.1 the basic idea of the state space representation is explained. This chapter deals with the part, controlling the process of Figure 20.

But how to do that?

Thus, one opportunity is, to use several PI(D) controllers in an cascade control architecture. But - in this case - a state space controller is the candidate. The fundamental idea of this controller is, to feed back all required internal state variables. In the feedback of each state variable, there have to be a specific multiplicity. This multiplicity has to be determined by the pole placement in chapter 4.3. In order to be able to develop a state space controller for a process - as the dynamics of the quadrocopter are - it is necessary, that the process is controllable.

How does a state space controller for the process of Figure 20 look like?

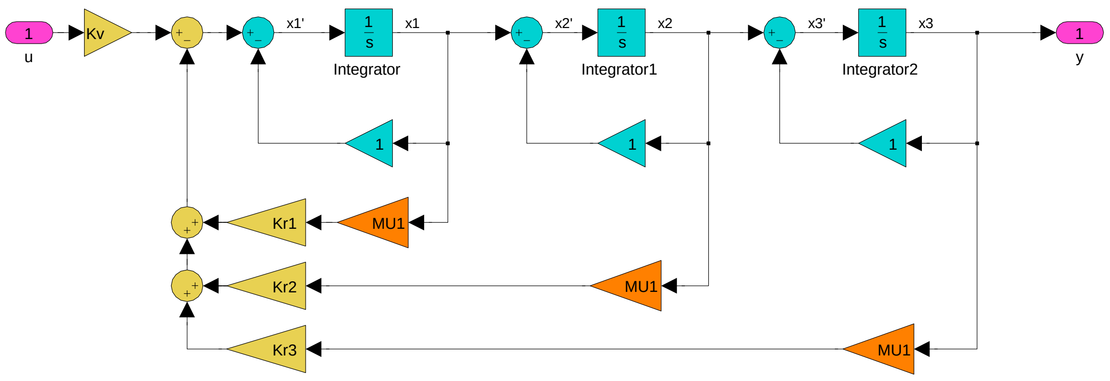

Figure 23: Simple process example including the state space controller

There it is - the controller for the simple process of Figure 20. The orange marked MUx are the measuring converters, that are not important in this simple example. That is, why they are not described more precisely in this chapter. The controller is the whole part, marked in yellowish brown. On the one hand there are the feedback factors KRx and on the other hand the pre-intensification factor Kv. All in all, this are four independent parameters, which have to be calculated while designing the state space controller. For a MIMO system, chapter 4.5 deals with, there are much more independent parameters that have to be defined while designing. As mentioned, the pre-intensification factor Kv is one part of our controller. Its function is, to eliminate the steady control deviation. How this factor and the other part of the state space controller, the feedback factors KRx are calculated, is described in the chapter pole placement 4.3.

4.3 Pole placement

The pole placement is the most important step in developing a state space controller. With an state space controller it is possible - if the process is controllable - to move the poles that exist in the open loop process like 20, to a specific position in the closed loop 23. There are several possibilities to arrange the poles in the s-plane to gain a short rise time, a short control settling time and no ringing. The more the poles are moved to the negative real axes, the faster the closed loop process will be. But for that, very high actuating variables are needed, that might not be available. For the quadrocopter the motors are the limiting factor.

As mentioned in chapter 4.1, the example process is controllable, so the poles can be placed. The poles of the process are at [-1 -1 -1] in the s-plane and the corresponding step response is:

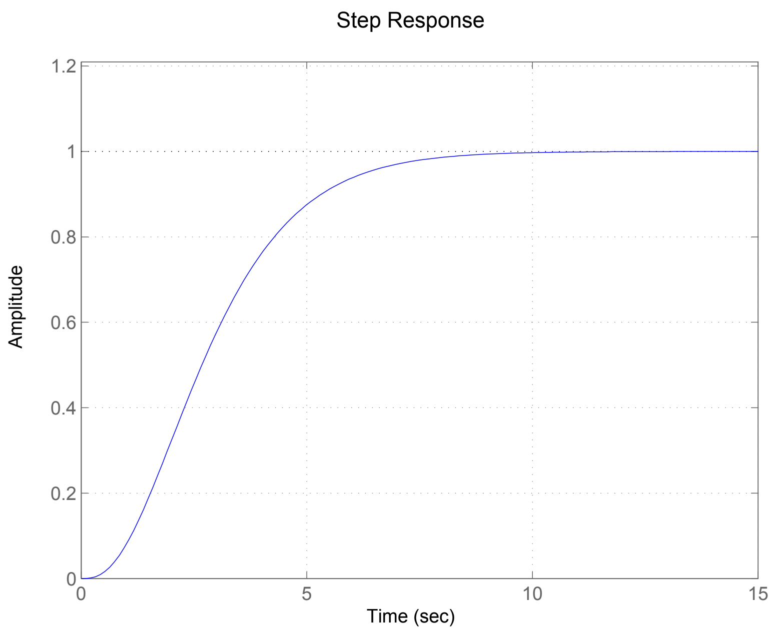

Figure 24: Step response of the process without the state space controller

This is not a good result, although the poles are on the left side of the s-plane. Making the step response faster in the closed loop, the poles are moved to [-4 -3 -2]. These new positions, the poles are moved to, are chosen without any ulterior motive.

So how to calculate the state variable feedback factors KRx? The process can be described in a very nice vector notation, including the state space matrices. The D matrix is zero in this process.

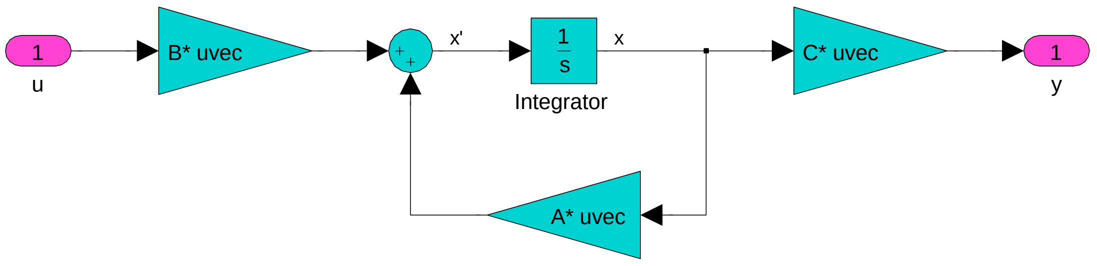

Figure 25: Matrix view of the process

$$\underline{x}' = \underline{A} * \underline{x} + B * u \tag{4}$$

$$y = \underline{C} * \underline{x} \tag{5}$$

The duty of the pole placement is, to transform this open loop process to a closed loop control with the desired poles.

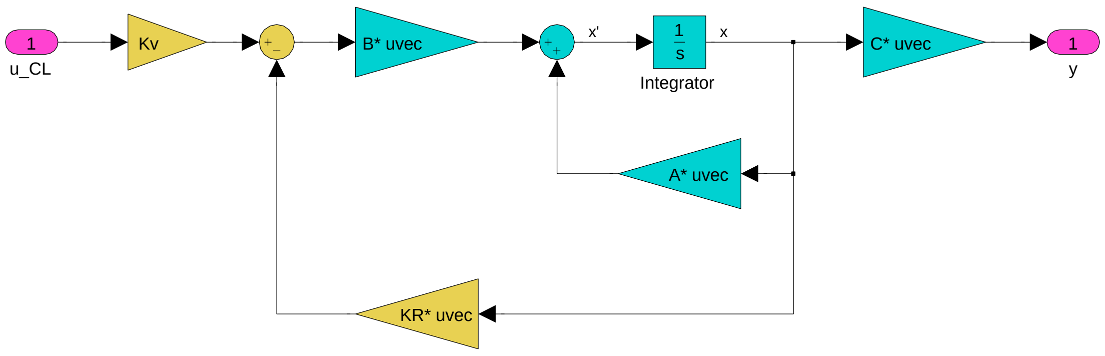

Figure 26: Matrix view of the process including the state space controller

Therefore the input u (in front of the 'B gain block') can be described:

$$u \;=\; Kv * u\_CL - \underline{KR} * \underline{x} \tag{6}$$

Finally the closed loop control equations are:

$$\underline{x}' = (\underline{A} - B * KRx) * \underline{x} + B * Kv * u \tag{7}$$
$$y = \underline{C} * \underline{x} \tag{8}$$

And the new system matrix for the closed loop is:

$$\underline{A_{CL}} \;=\; \underline{A} - B * \underline{KR} \tag{9}$$

With this 'formula' it is easy to calculate the state variable feedback factors KRx if the state matrix of the open loop and the state matrix of the desired closed loop is known. So, for this example - recognizing that the poles are moved to [-4 -3 -2] - the KR factors are [6 11 6]. The step response of the closed loop seems to be much nicer:

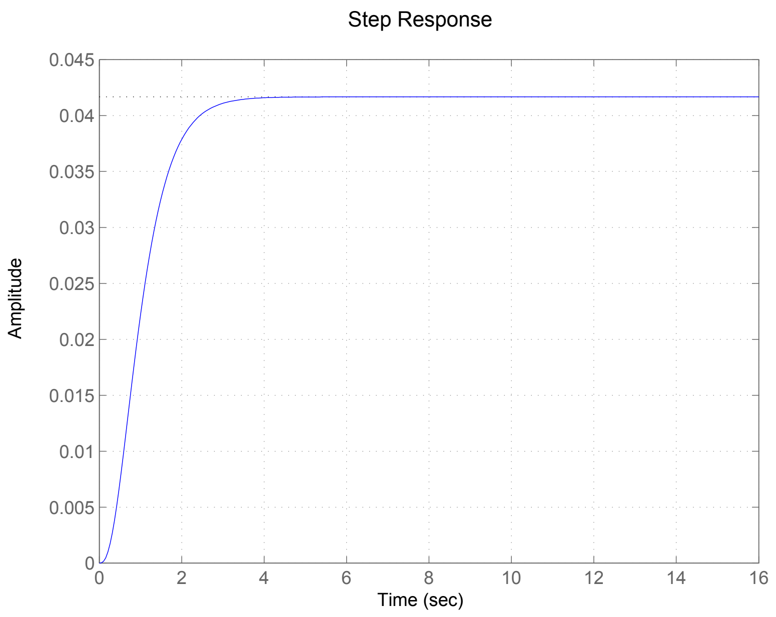

Figure 27: Step response of the process including the state space controller with a pre-intensification of one

The step response is much faster - but - it only steps to $y_\infty = 0.0417$. The second part of the state space controller have to be calculated. To calculate the pre-intensification factor Kv for a very easy SISO system, the steady state value of the system's input must be divided by the system's output (equation 10).

$$Kv = \frac{u}{y_\infty} \tag{10}$$

For the system of the quadrocopter that is a MIMO system the calculation of the pre-intensification factor Kv is much more complex (equation 11).

$$Kv = -\left[C * (A - B * KR)^T * B\right]^{-1} \tag{11}$$

The example used here is a simple system and the pre-intensification factor Kv can be calculated easily:

$$Kv = \frac{1}{0.0417} \approx 24 \tag{12}$$

With this pre-intensification factor the step response really steps up to one:

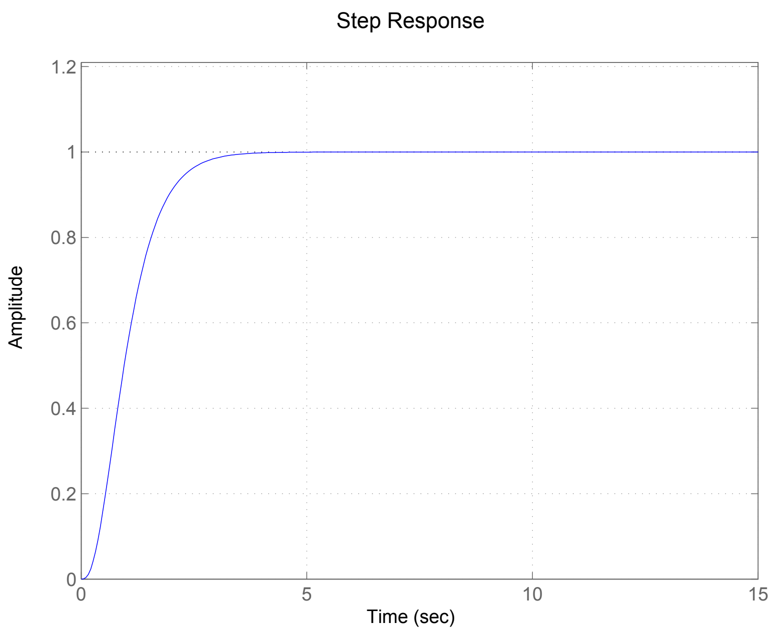

Figure 28: Step response of the process including the state space controller and a calculated pre-intensification

4.4 State observer

In the simple example for an state space controller in figure 23, it was implied, that all state space variables can be measured. And this measured variables can be feeded back in order to let the state space controller work. But - what if either a state variable can not be measured or the sensor costs to much to be economic?

David Gilbert Luenberger, working as professor at the Stanford University, invented a state observer called Luenberger observer. With this technique it is possible to use a state space controller, although not all state variable can be measured.

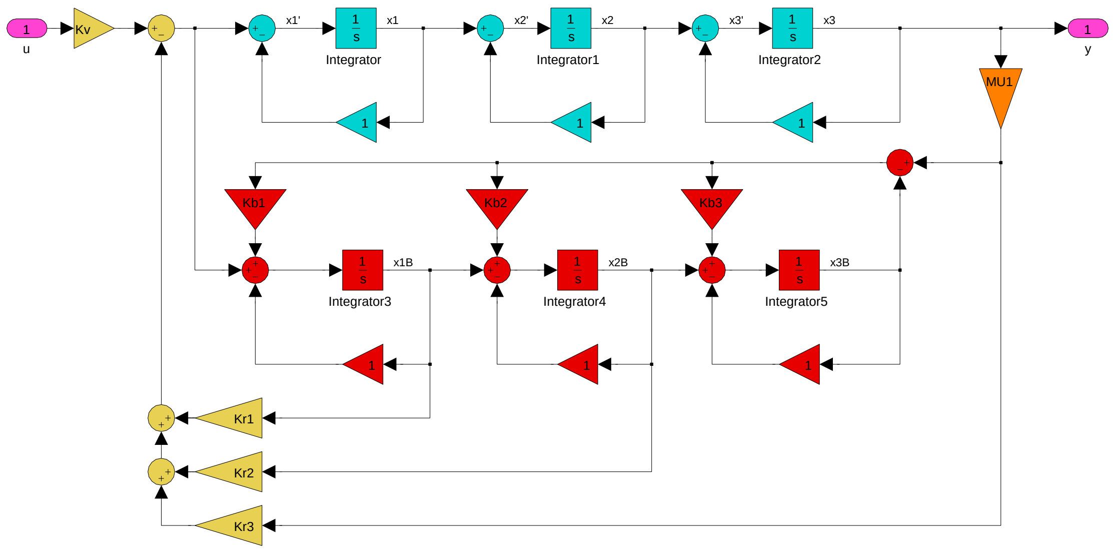

Figure 29: Simple process including a state space controller and an state observer

In figure 29 the state space controller is extended with an Luenberger observer. The observer is marked in red color. So, what exactly is shown there? To control the process with the standard state space controller, three measurement converters MUx are needed. With the Luenberger observer, only one of them is required. In the example, there is a sensor for the third state variable. This state variable is feeded back, just as common from the state space controller. But the other two state variables can not be measured, so the observer mirrors the whole open loop process and implements this process in the controller. So the state variables *x1B* and *x2B* are feeded back and the state space controller itself will not notice that. These values are only estimated values but damned good ones. To correct and stabilize the values of the observer, the measured value *x3* is injected in the process and moves the simulated process in the right direction. This correcting works with the observer feedback factors *Kbx*.

Based on the fact, that the state space controller for the quadrocopter will not require an observer, this chapter will not be enlarged.

4.5 MIMO-Systems

This part of the research paper reveals the secret of MIMO-Systems. MIMO is the abbreviation for 'Multiple Input Multiple Output'. All chapters before this one, deal with the simple 'SISO' example. But the quadrocopter process is very complex and like chapter 2.2 explains, there are three important control variables, that have to be controlled. And again - in this chapter - a little example for a MIMO system is explained, before the complex quadrocopter system is presented in the next chapter.

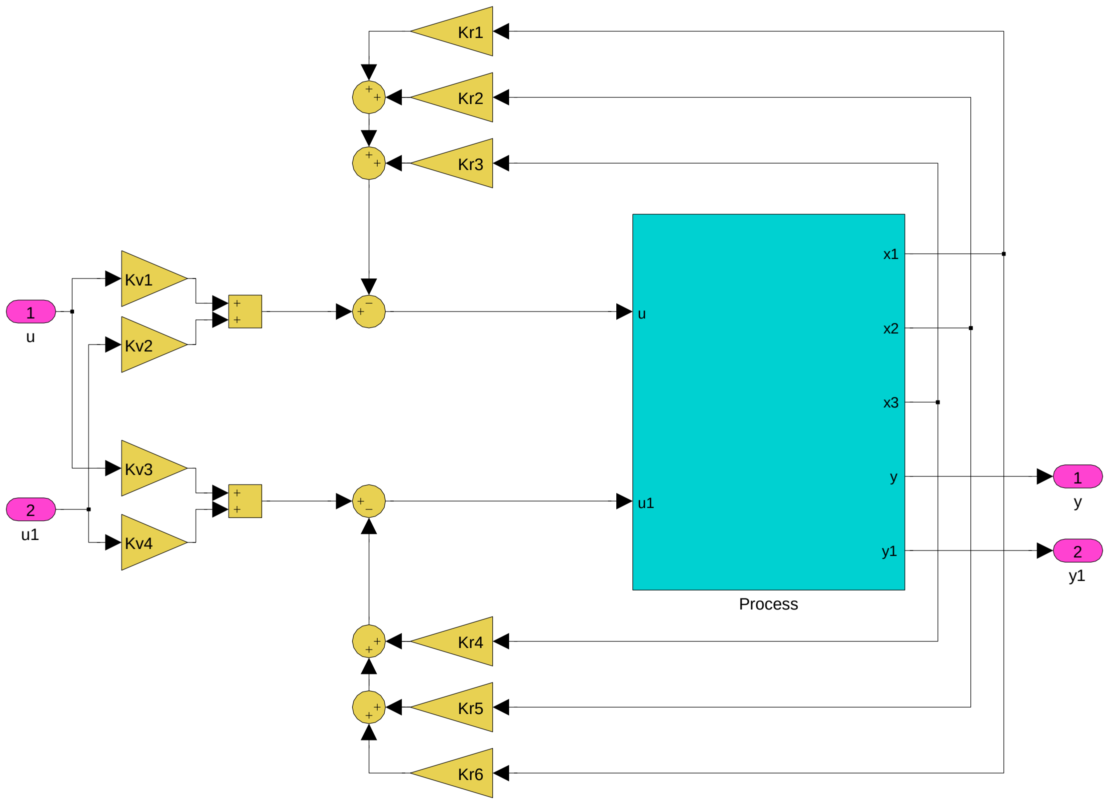

Figure 30: Principle model of a third order MIMO process including a state space controller

Now in figure 30 there is a simple MIMO-System. What exactly happens in the 'process-block' is not important, but there are two inputs *u* and *u1*. Also, there are five outputs - three times the measured state variables *x1*, *x2* and *x3* and the two outputs *y* and *y1*. The state space controller, marked in yellowish brown, is a bit more complex than for a SISO-System. So - how to calculate the state variable feedback factors and the pre-intensification factors now? Interesting answer, but it is the same for MIMO-System as for SISO-Systems. The only thing is - it is more complex, because of more matrix equations. But those matrix equations are not the problem - therefore MATLAB is the tool.

5 Design and implementation in MATLAB/Simulink

5.1 Approach

At this point, all basics about the quadrocopter itself, the existing MATLAB Simulink model and about state space controlling should be clear. This chapter deals with the main part of what was invented in this research paper.

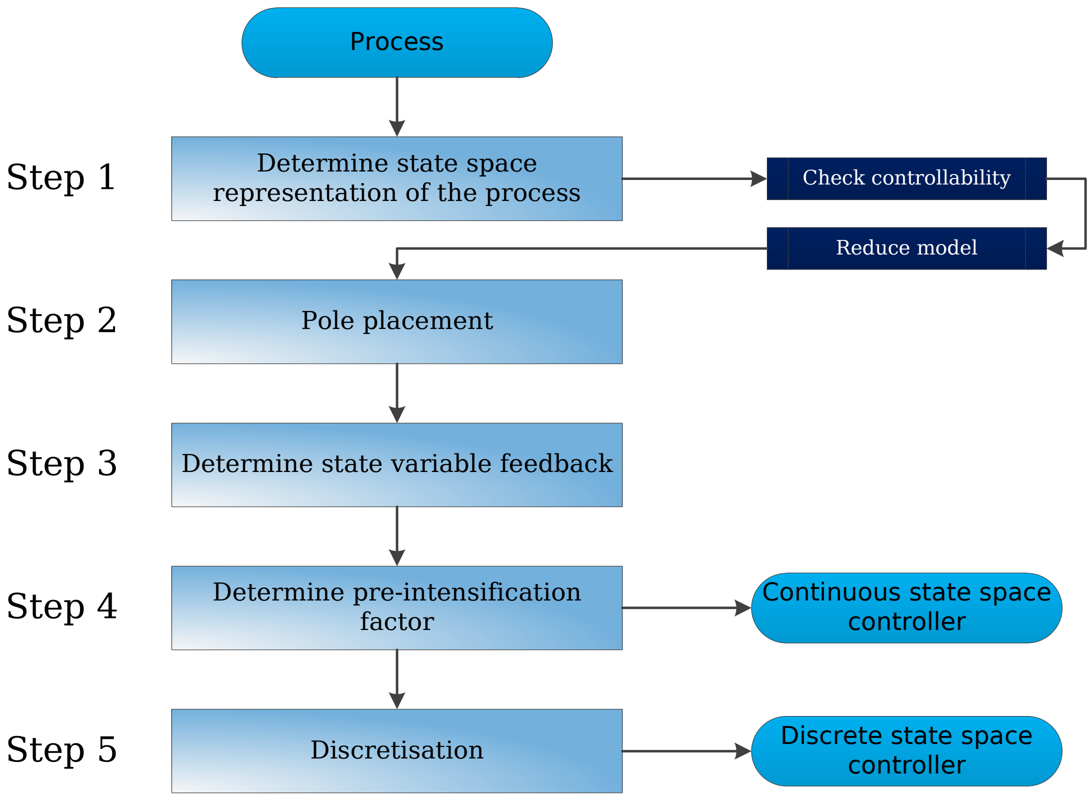
Process
Step 1
Determine state space representation of the process
Check controllability
Reduce model
Step 2
Pole placement
Step 3
Determine state variable feedback
Step 4
Determine pre-intensification factor
Continuous state space controller
Step 5
Discretisation
Discrete state space controller

Figure 31: The five development steps

To develop the state space controller the 'Five Steps', as they are named in the script of Prof.Kull ([2]), are used (figure 31). These steps are represented by the following sections. First step is to determine the state space representation of the process and check the controllability. Maybe it is necessary to reduce the model of the process. Thereafter, in the second step, the poles have to placed in a wise order. In step three the state variable feedback factors for this specific pole constellation must be calculated. Further on in step four, the corresponding pre-intensification factors must be estimated. After these four steps, the continuous state space controller for the MATLAB/Simulink simulation and the basis for the implementation in C is completed. But to implement the controller into the quadrocopter - means in C - a discretization is needed. The next chapter deals with that circumstance.

There are several reasons why MATLAB and MATLAB/Simulink are used as development and simulation tool. One point is, that the cornerstone was implemented in MATLAB (chapter 3). Another point is the uniqueness of MATLAB and MATLAB/Simulink to combine all of the mentioned development steps in one tool. Even the HIL-Tests can be performed in this tool. In the following sections the purpose, of why using MATLAB, will become clear.

5.2 State space representation

As mentioned in the approach 5.1, first step in developing the state space controller is, to analyze the process of the quadrocopter. This process is introduced and explained in chapter 3.4. The chapter 4.1 explains the basic knowledge about state space representation, needed in this chapter. Figure 13 shows the process of the quadrocopter.

First of all it is necessary to find out, which part of the process is really needed to control the quadrocopter. So the earthframe part (blue) of the quadrocopter can be eliminated. Figures 14 and 16 are showing, how the remain looks like. That is only the bodyframe part of the process. And - this part - is that one, the state space controller have to deal with.

5.2.1 Controllability

Now the part of the process, that has to be controlled, is filtered out. Next step is, to check whether the process is controllable or not. To do so, the rank of the controllability matrix has to be the same as the number of state variables in the process. The easiest way to prove the controllability is, to use the ctrb() command in MATLAB. It returns the rank of the controllability matrix, which have to be compared with the number of state variables. With linmod(), it is possible to get the state space matrices of a Simulink model.

Like figures 14 and 16 are showing, there are nine state variables that have to be controlled. These nine state variables, are the four motor delays (3.4.1), the rate and angle of phi, the rate and angle of theta and the rate of psi.

```
System = linmod('dynamics_reduced');
StateSpace = ss(System.a, System.b, System.c, System.d);
rank(ctrb(StateSpace))
% ans = 4
```

How the state space matrices determined with the code above exactly look like, can be found in the Appendix (APPENDIXREF). As the code snippet reveals, only four of the nine state variables can be controlled. Now there are two options to choose from. One is, to cancel the development process of the state space controller. But that is not the engineers way. The other option is, to continue reducing the model to gain full controllability. The next section deals with that.

5.2.2 Model reduction

The already reduced process, is not controllable (chapter 5.2.1). So there is need to reduce the model again. But all remaining parts of the model are needed to construct the state space controller. In this stage, the reduced process have to be examined very carefully. Looking at the delay in the four motors of the process, reveals, that there is only one millisecond delay in each motor (figure 32).

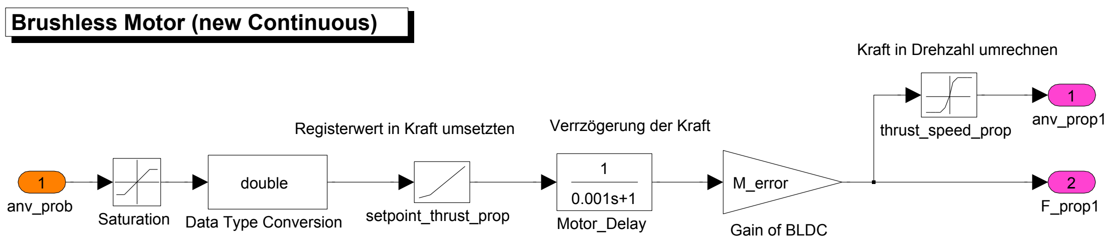

Figure 32: Model of one motor including the time delay

In comprehension to the delays in each of the processes remaining integrators, the one millisecond can be disregarded (figure 33). Now the same code already used in chapter 5.2.1 have to run again, to check the controllability.

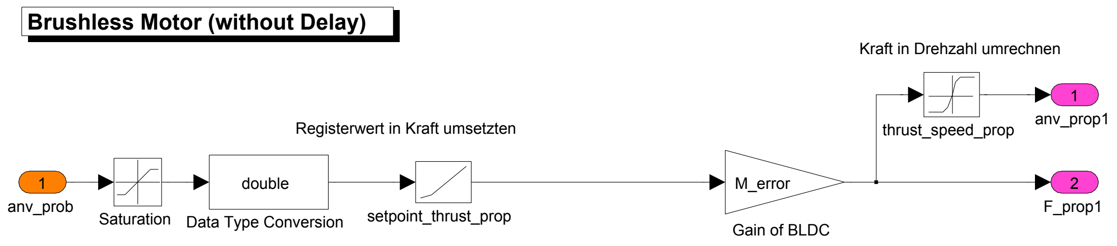

Figure 33: Model of one motor without the time delay

```
System = linmod('dynamics_reduced_without_motordelay');
StateSpace = ss(System.a, System.b, System.c, System.d);
rank(ctrb(StateSpace))
% ans = 5
```

How the state space matrices determined with the code above exactly look like, can be found in the Appenix (8). Now the rank of the controllability matrix is five. That is the right number of controllable state variables, because the four motor delays are removed and so only five state variables have to remain. At this point of the development it is not sure, if the state space controller, developed for this model without motor delay, will work later on with the process with the motor delay.

5.2.3 Observability

Just a few words about the observability of the process. Five states are controllable - the rate of phi, theta, psi and the angle of phi and theta. Chapter 2.4 shows the sensors, used in the quadrocopter. For each of the five state variables there is a sensor. That is the reason, why no observer is needed in this project. But - maybe interesting for an further project - the observability of the system is given. So it might be possible to reduce the number of sensors and develop an observer instead. The following code snippet shows that.

```
System = linmod('dynamics_reduced_without_motordelay');
StateSpace = ss(System.a, System.b, System.c, System.d);
rank(obsv(StateSpace))
% ans = 5
```

5.3 Simulink

Finally, before calculating the state variable feedback factors in the next chapter, it is advisable to create a dummy state space controller in Simulink. Therefore it is necessary to know what the actuating variables are. Figure 11 shows these input variables named *phi_set*, *theta_set*, *rpsi_set* and *thrust_set*. The thrust is controlled directly by the remote control and does not have to be controlled in the quadrocopter. The other three variables are the set variables that must be controlled internal.
In addition the input of the process - thus the output of the controller - have to be defined. As figure 12 shows, the output of the controller block are the desired pseudo forces for the four motors of the quadrocopter.

The three set variables calculated by the controller and the thrust are mixed together and four pseudo forces are the result. This is very easy to understand - for example *phi_set* tells, that the quadrocopter shall move to the left, so the left motor is slowed down and the right one is accelerated.

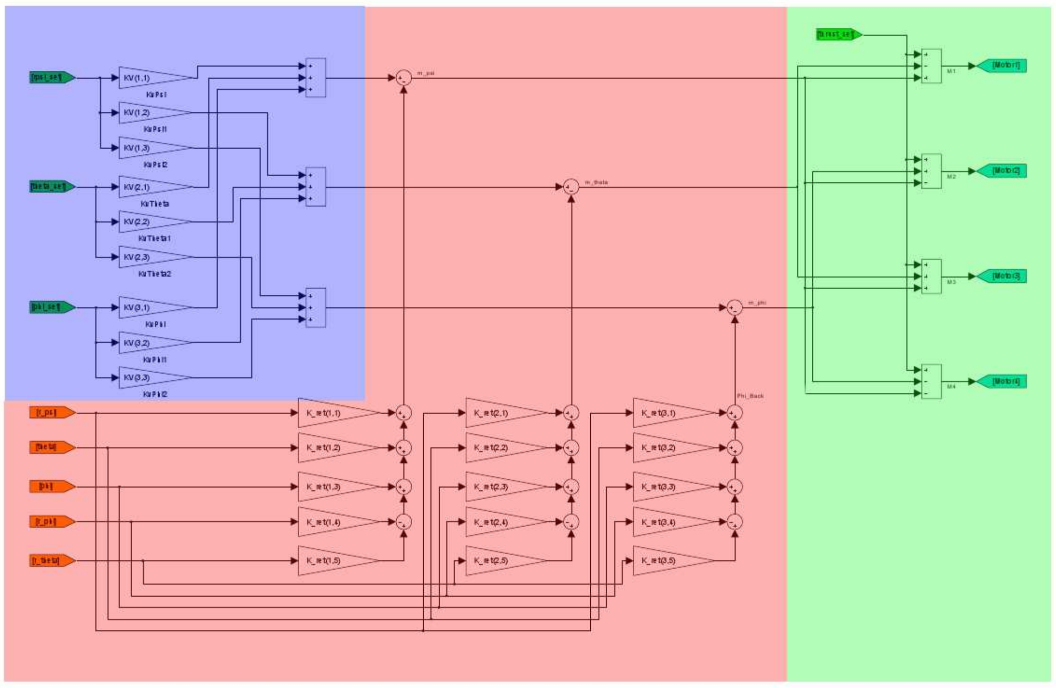

Figure 34: Overview over the whole state space controller

Figure 34 shows the complete state space controller. In the following sub chapters the red and blue part are explained.

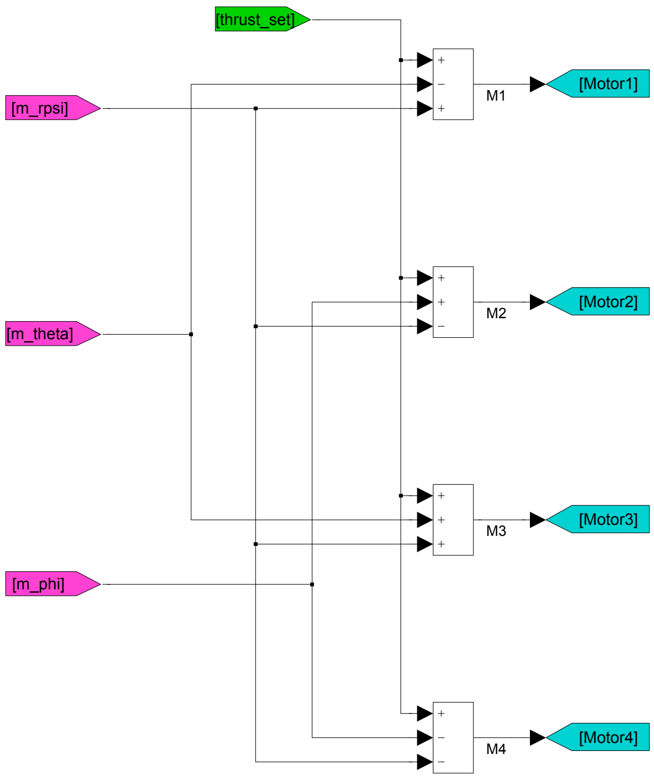

Figure 35: Transformation of the set values to pseudo forces

Figure 35 shows this green part in detail. This part is something like a transformation from set values, for an moving direction of the quadrocopter, to pseudo forces. Giving more *thrust*, all four motors are accelerated and the result is, that the quadrocopter raises. Changing *m_rpsi* means, accelerating two motors and decelerating the other two. In this case, the quadrocopter is yawing. For *m_theta* and *m_phi* it is both nearly the same. One motor is accelerated and the opposing motor is decelerated. The quadrocopter flies into the direction of the decelerated motor.
For sure, there are all combinations of these four input values possible.

5.4 Pole placement

There are two requirements, that have to be checked before starting with the pole placement. First one is, that the physical process that shall be controlled is available as a mathematical model - e.g in Simulink. The other one is, that this process is controllable. Chapter 5.2 deals with that fact.

First of all it is necessary to check where the poles of the process are. This can be done with the following MATLAB code.

```
System = linmod('dynamics_reduced_without_motordelay');
StateSpace = ss(System.a, System.b, System.c, System.d);
eig(StateSpace)
pzmap(StateSpace)
```

The output of the eig() function is:

```
        0
   0.2378
  -0.2378
        0
        0
```

And the pole/zero-map looks like that:

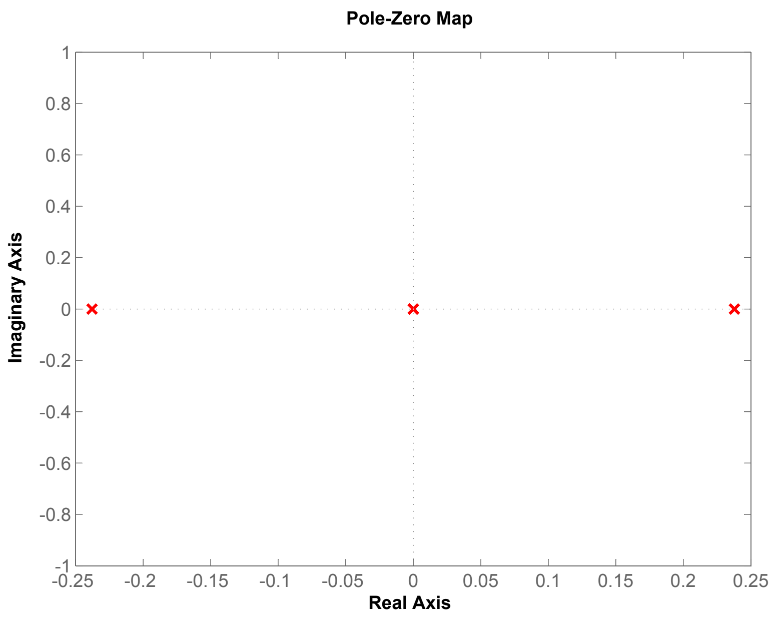

Figure 36: Pole/zero-map of the reduced quadrocopter process

So there are five poles - three at zero, one at 0.2378 and one at -0.2378. This pole constellation - and consequently the process - is not stable, because one pole is at the right half-plane. So it is coercible necessary to use a controller for that process. The function of the pole placement is, to move at least the positive pole to the left half-plane. The three poles at zero are also not really nice, because they are not stable like the pole on the left half-plane. Poles on the imaginary axis can oscillate without damping. So the first requirement is, that all poles have to be moved to the left half-plane. Next step is thinking about to which values the poles should be moved. For an aircraft overshooting is not a good feature. So the second requirement is, that all poles have to stay at the real axis of the s-plane. And the third requirement is, that the poles, corresponding to the rates have to be faster than those, belonging to the angles.

To challenge all these requirements the poles are moved to [-18 -5 -7 -10 -9]. Maybe these poles are too fast and they have to be moved to lower negative values. But it is not possible to know about that at this step of the development. Chapter (REF ZUM TESTKAPITEL) deals with that circumstance.

5.5 State variable feedback

It is no problem for a state space controller to move the poles wherever the engineer wants them to go. How to do this by hand is explained in chapter 4.3. For this much more complicate process MATLAB is used. In chapter 5.4 the decision to move the poles to [-18 -5 -7 -10 -9] is made. As mentioned, MATLAB is used to find the state variable feedback factors needed by the state space controller to move the poles to the specific position. The following code calculates them.

```
System = linmod('dynamics_reduced_without_motordelay');
StateSpace = ss(System.a, System.b, System.c, System.d);
K_ret = place(System.a, System.b, [-18 -5 -7 -10 -9]);
```

The function place() calculates the state variable feedback factors. The result is:

$$\begin{array}{c|ccccc|} & rpsi & theta & phi & rphi & rtheta \\ \hline rpsi\_set & 7.27 & 0 & -0.57 & -0.07 & 0.03 \\ theta\_set & 0.76 & 79.36 & 0.03 & 0 & 20.28 \\ phi\_set & 0 & 0 & 65.05 & 15.80 & 0 \end{array} \qquad (13)$$

This matrix describes with which gain, which state variable have to be feeded back. For example the state variable theta have to be feeded back with a gain of 20.28 to the theta_set path.

There are two ways to prove, that the poles are moved to the estimated places. The first way is, to calculate the closed loop state matrix and the output matrix with the calculated feedback factors. The following code is doing that.

```
A_CL = System.a - System.b * K_ret;
C_CL = System.c - System.d * K_ret;
Sys_ClosedLoop = ss(A_CL, System.b, C_CL, System.d);
eig(Sys_ClosedLoop)
```

The output of the eig() function are the poles that are passed to the place() function earlier. So it is clear, that the poles now are, where they should be.

The second way is, to create the closed loop Simulink block diagram and get the state space representation with linmod(). At this point it is also possible to check weather it was a good or a bad decision to ignore the motor delays. The Simulink block diagram used in the following includes the whole state space controller and the process. Furthermore the motors with delay are used.

```
S_CL = linmod('ControllerAndProcess');
SS_ClosedLoop = ss(S_CL.a, S_CL.b, S_CL.c, S_CL.d);
eig(SS_ClosedLoop)
```

And like expected, the result is:

```
-9,66
-7,13
-4,99
-9,07
-18,47
-1016
-991
-977
-1000
```

The first five poles are the poles expected - not exactly, but a good result. The four remaining poles are those for the motor delays. So it is possible to see, that they do not really disturb the others, because they are very far away on the left side of the s-plane.

So how does this feeback look like in a Simulink block diagram?

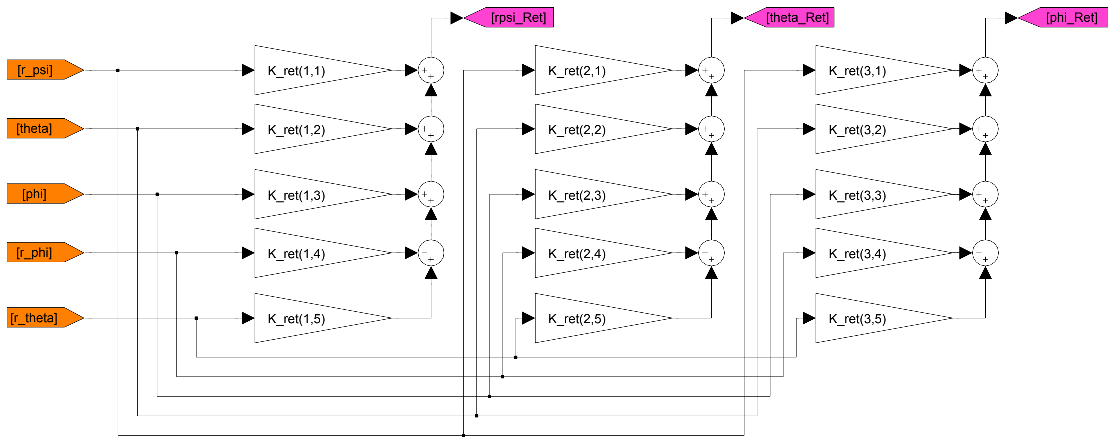

Figure 37: Feedback part of the state space controller

Figure 37 shows the red part of the overview figure 34. The input values marked orange, are the measured state variables of the process. The output marked in magenta, are the added state variables with their specific feedback factors. The calculated matrix (13) can be written in the gain blocks in this diagram. Now there is maybe some unclarity, because some of those feedback factors are zero and even so there is a gain block for them. The reason is, that if the pole constellation is changed, these calculated values may change from zero to an important value. Another blur is the minus sign in the feedback factors for *rphi*. The answer is very simple - the sensor, measuring this value, is build in the wrong way. So all values have to be flipped.

5.6 Control deviation

After calculating the state variable feedback, the dynamic behavior of the quadrocopter is fine. But, the quadrocopter will not fly. There is something missing - the pre-intensivication factors. In this MIMO system, each input variable $rpsi\_set$, $theta\_set$ and phi_set influences each of the three control paths. But not in the same intensity.

First, the set variables are standardized, so that the range of the input values from -128 to 128 now reaches from -0.75 to 0.75. This is necessary, because the angels *theta* and *phi* are calculated in rad and the stationary throughput is calculated being one. That means, a maximum angle of approximately 45°is reached. Maybe it sounds very little, but believe - it is not. The yaw rate *rpsi* is calculated in *rad/s* so this value of 0.75 (45°/s) sounds nice for that purpose, too.

Calculating the pre-intensification factors is not very complex. The following little code snippet shows that.

```
KV = -(System.c * A_CL^-1 * System.b)^-1;
KV = KV';
```

The result is the following matrix.

$$\begin{array}{c c} & \begin{array}{ccc} rpsi\_Path & theta\_Path & phi\_Path \end{array} \\ \begin{array}{c} rpsi\_set \\ theta\_set \\ phi\_set \end{array} & \begin{bmatrix} 7.27 & -0,32 & 0 \\ 0 & 79.36 & 0 \\ -0,57 & 0,03 & 65.05 \end{bmatrix} \end{array} \tag{14}$$

Looking closer at the matrix reveals that these factors are nearly the same as the state variable feedback factors.

As the final step, the implementation in Simulink. This part is the red one in the overview figure 34.

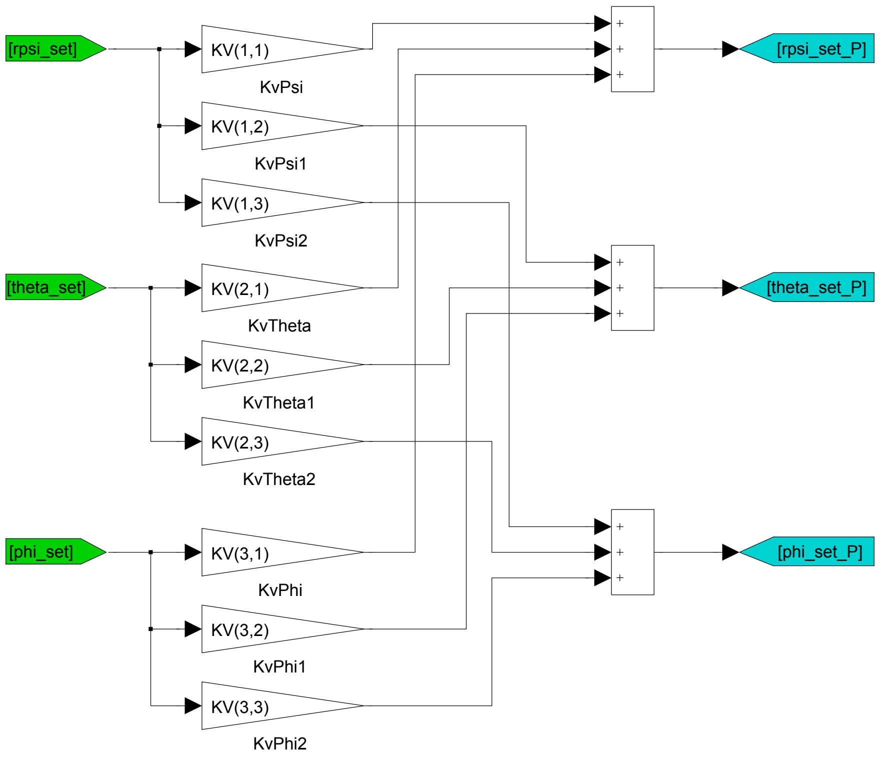

Figure 38: Pre-intensification factors for the different control paths

Figure 38 shows a detailed look at that part. The input values, marked green, are the standardized set values from the remote control. The output, marked cyan, are the calculated set values for each path, but including the influence of the other set values.

At this step of development the continuous state space controller is finished as figure 31 shows. The simulation results can be found in chapter (6).

6 Final tests

The subchapters in this chapter are dealing with the different steps of testing the state space controller. First, chapter REF deals with the Matlab/Simulink imulation test. Second, the chapter REF deals with the Hardware In the Loop test and finally chapter REF deals with the test in the real quadrocopter.

6.1 Simulation test

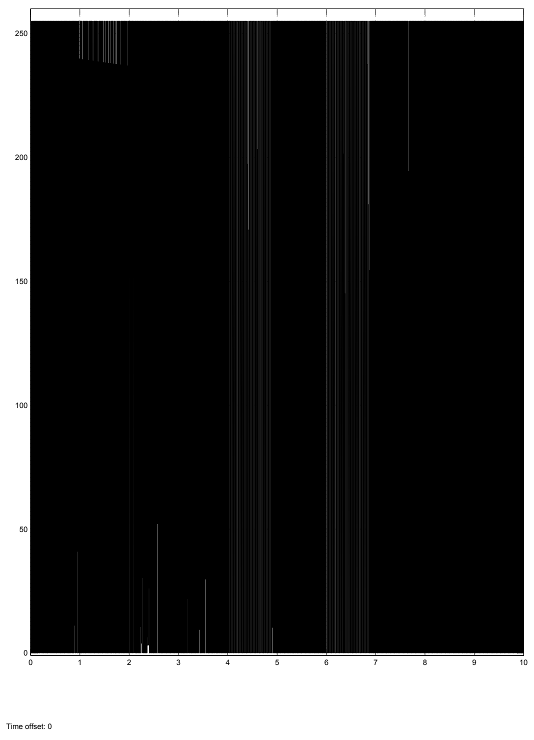

Figure 39: awdwadwd

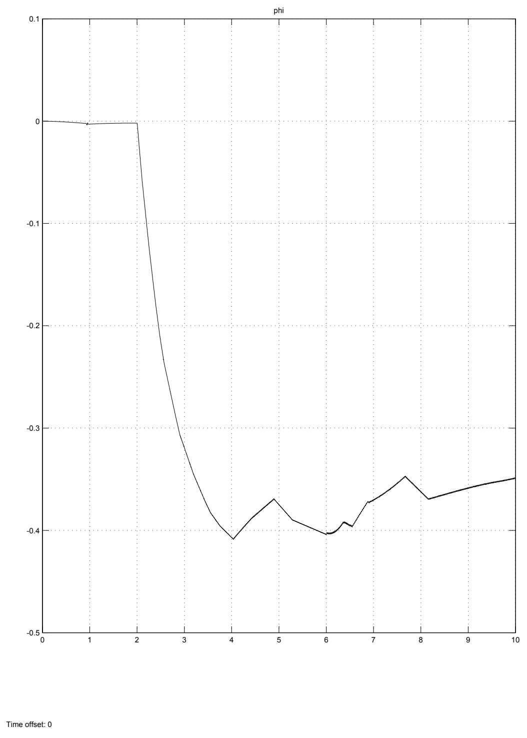

Figure 40: awdwadwd

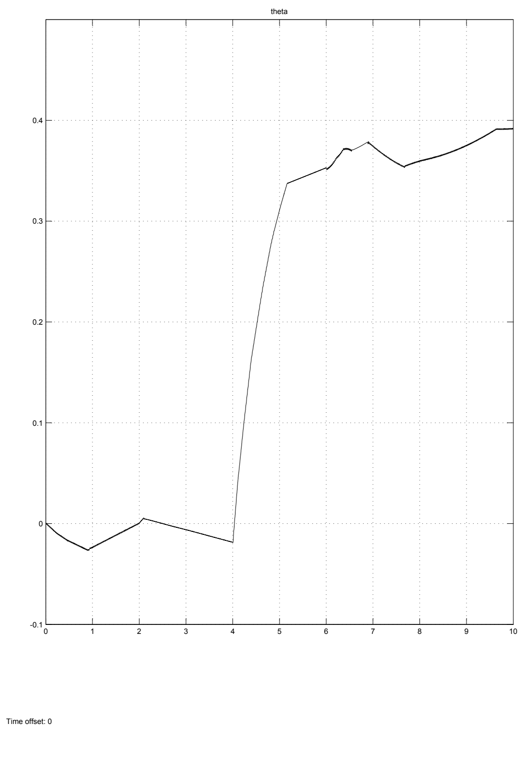

Figure 41: awdwadwd

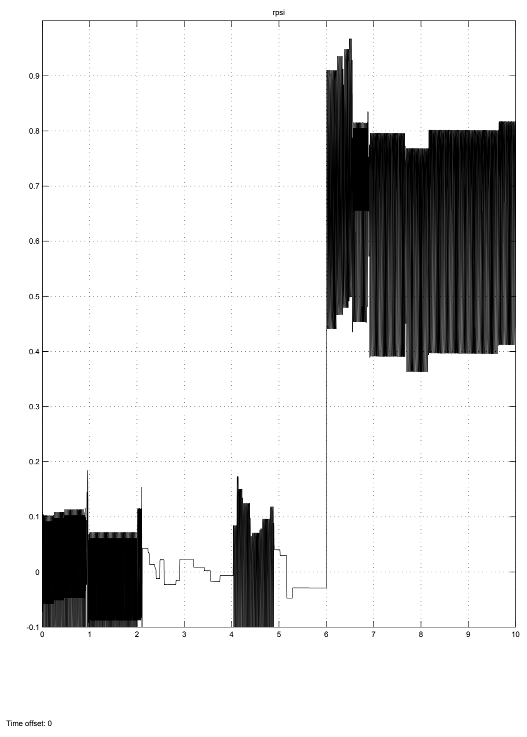

Figure 42: awdwadwd

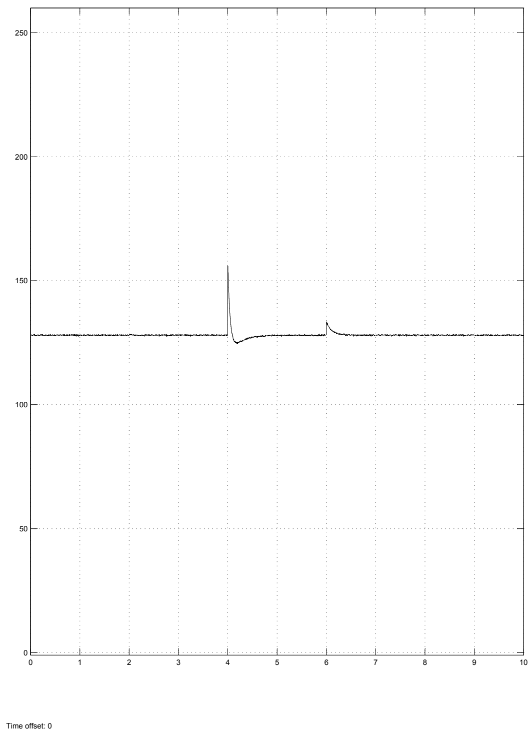

Figure 43: awdwadwd

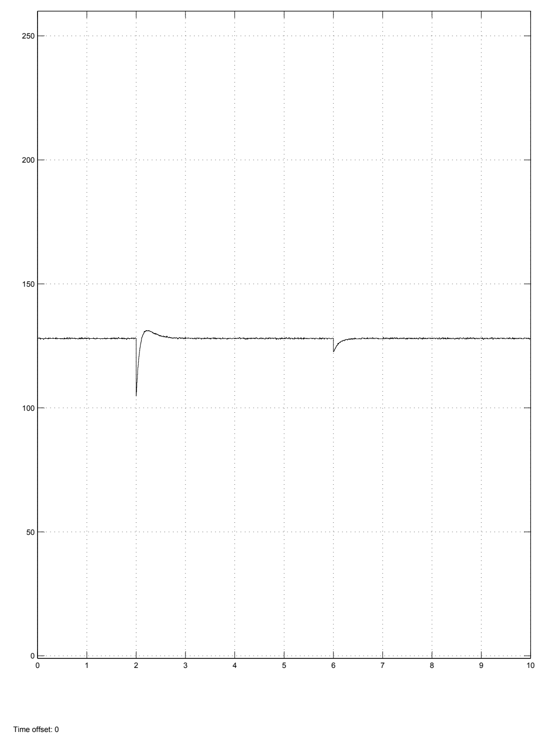

Figure 44: awdwadwd

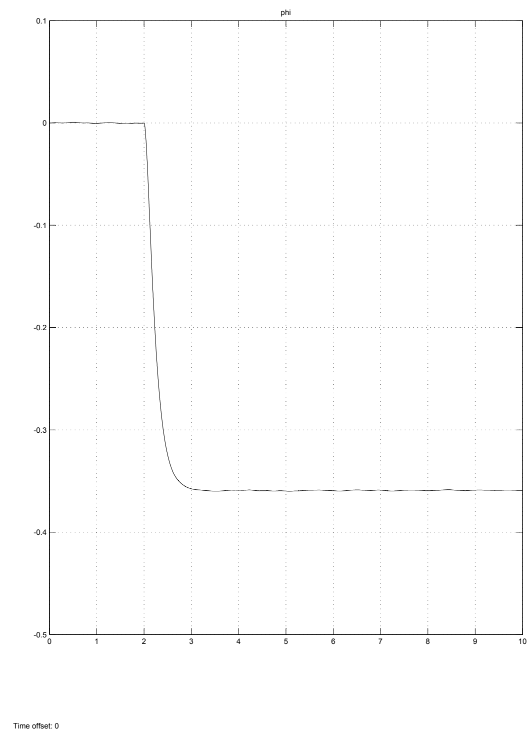

Figure 45: awdwadwd

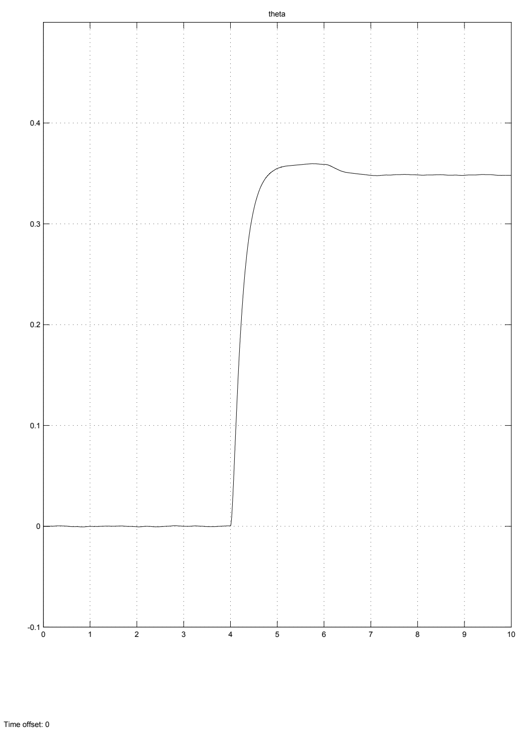

Figure 46: awdwadwd

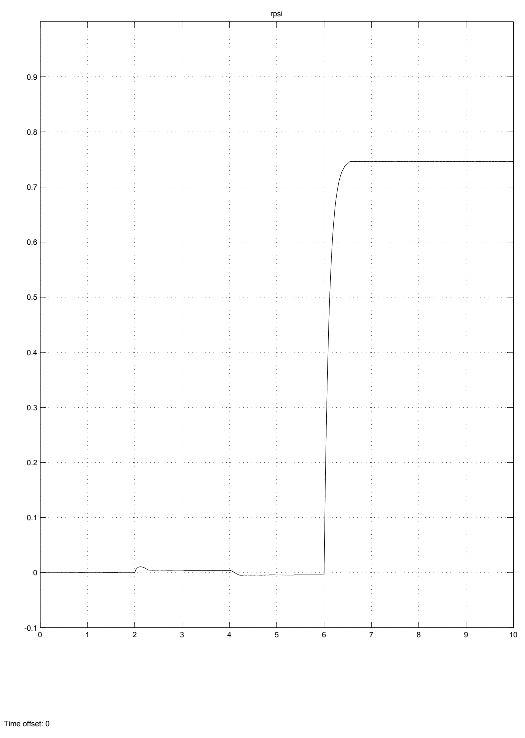

Figure 47: awdwadwd

6.2 Hardware In the Loop test

6.3 Real quadrocopter test

7 Future prospects

Literatur

[1] WOHER, *Titel für Quelle*, ART der QUELLE, Aufruf: tt.mm.jjjj, `https://blubb.de`

[2] WOHER, *Titel für Quelle*, ART der QUELLE, Aufruf: tt.mm.jjjj, `https://blubb.de`

8 Appendix

Hallo Ich bin der Appendix - Asterix's großer Bruder.

Bearbeitungshinweis: Zustandmatrizen IMPL ohne Reduction!

$$A = \begin{bmatrix} 0 & 0 & 0 & 10441 & -10441 & 10441 & -10441 & 0 & 0,0462 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1000 & 0 \\ 0 & 0 & 0 & 0 & 18159 & 0 & -18159 & 0 & 0 \\ 1,224 & 0 & 0 & -19138 & 0 & 19138 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0,00625 & 0 & 0 & 0 \\ 0 & 0,00625 & 0 & 0 \\ 0 & 0 & 0,00625 & 0 \\ 0 & 0 & 0 & 0,00625 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Bearbeitungshinweis: Zustandmatrizen IMPL MIT Reduction!

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0,0462 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1,224 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0,065 & -0,065 & 0,065 & -0,065 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0,113 & 0 & -0,113 \\ -0,120 & 0 & 0,120 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$