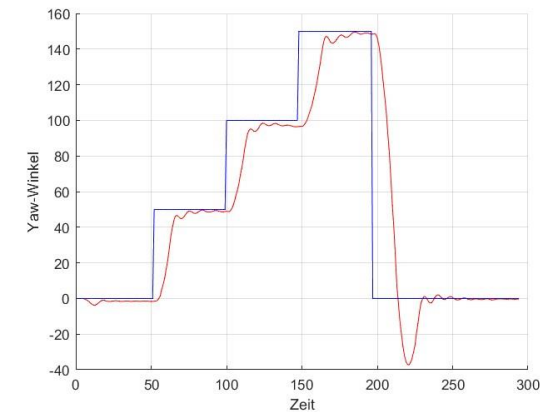
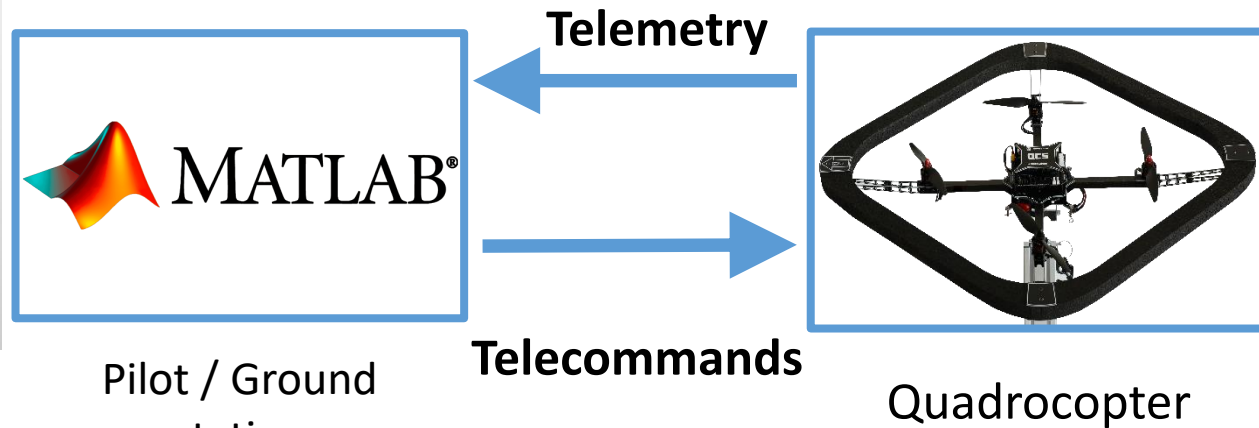


Matlab

PID-Controller

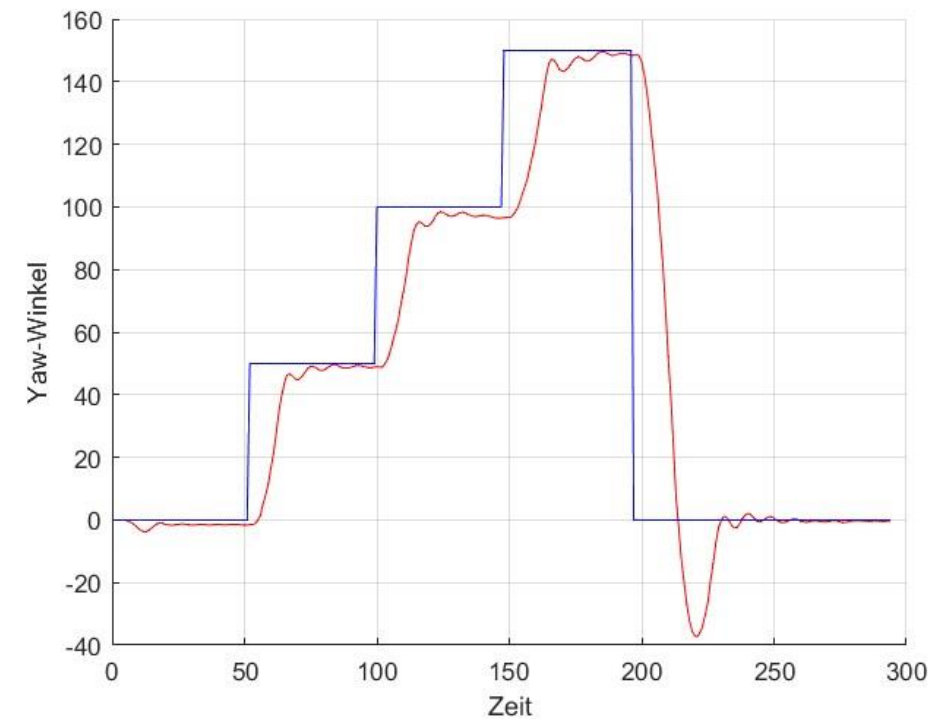
Lesson 12





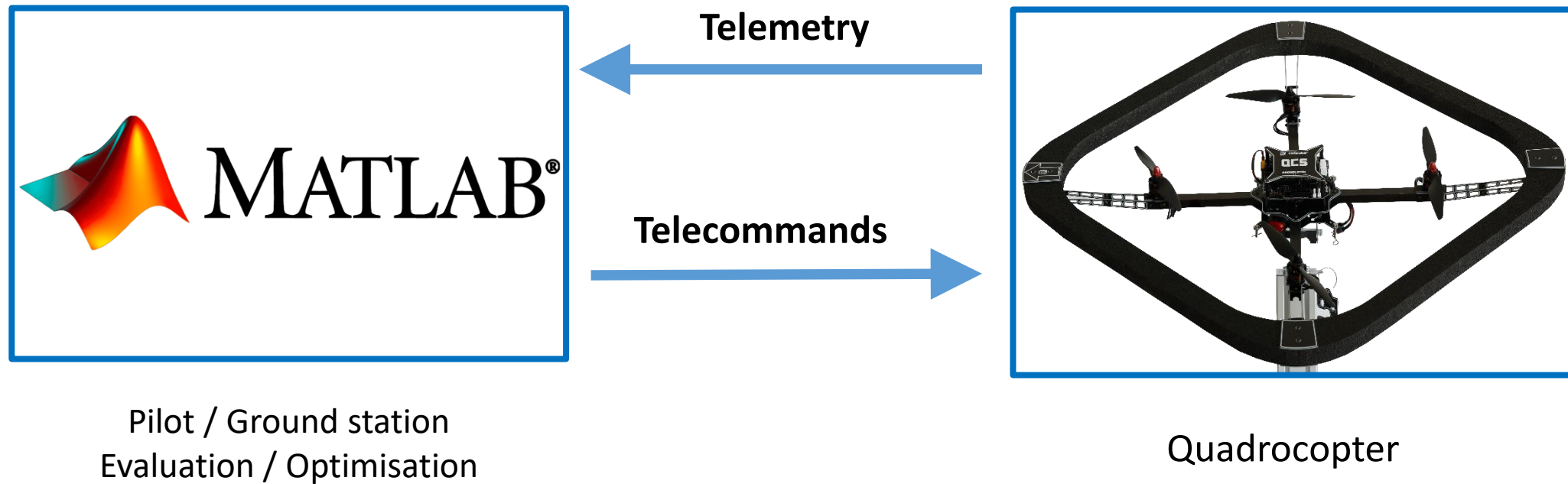
- Matlab: Telemetry & Telecommands
- Yaw control
- Theory of control
- Superposition
- Exercises

Time scope: approx. 2-3 hours



Matlab

Telemetry & Telecommands



Yaw control



Definition Yaw control:

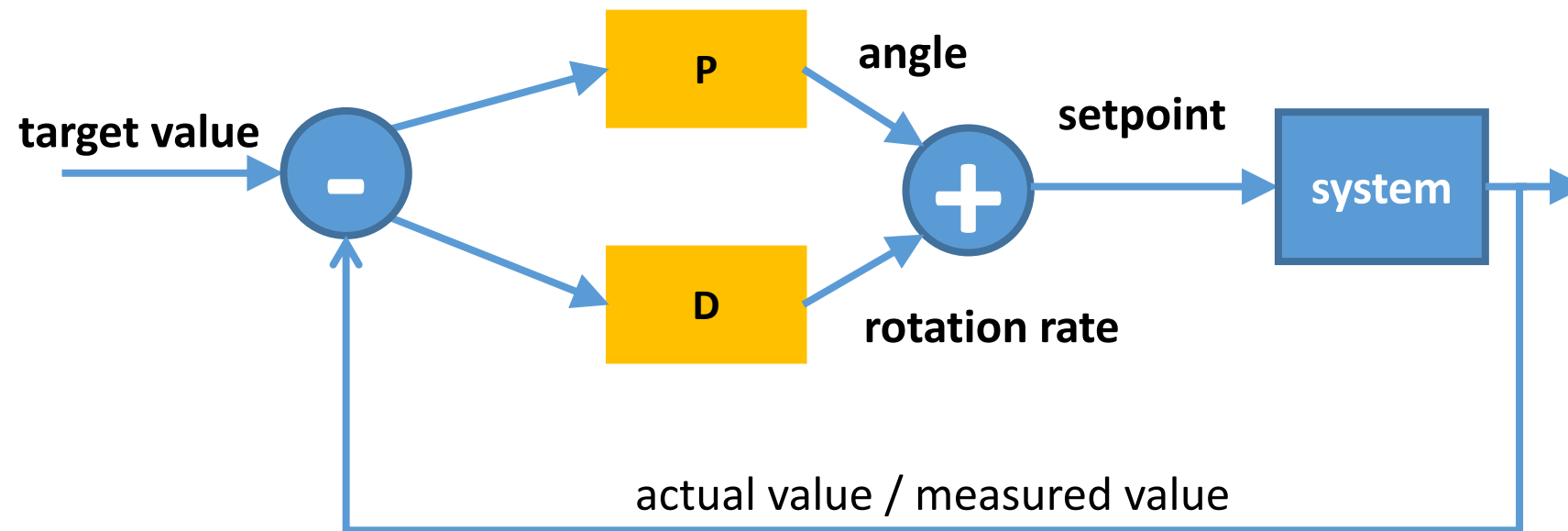
The torque of each motor causes the quadrocopter to yaw around its own axis (z-axis). Therefore, two motors always rotate to the left and two to the right. So that this does not collide with the position control, the opposite motors always rotate in the same direction .

Using a yaw controller, yaw maneuvers can then be executed or disturbances can be corrected. Compare to position control, this control is usually much simpler because the system does not fight against gravity and therefore behaves more favourably.

Theory of control



The system can be controlled using a PD controller and a gyro sensor. The yaw angle serves as the control variable.



Problem: 2 motor pairs / setpoint, 1 measured value (angle)

Lösung: Simplify by controlling the difference between the setpoints of the two motor pairs

Superposition



Superposition:

With superposition, the controllers are simply superimposed additively. The superposition assumes that the controllers are independent of each other.

The aim of this exercise is to control a quadrocopter. All the required controllers are developed individually. The degree of freedom RPY must be controlled so that the quadrocopter remains in the air. This is realised by 2 (3) controllers:

- ***Attitude-Controller (x2 = roll + pitch)***
- ***Yaw-Controller***

At the end, the two (three) controllers are simply superpositioned. For the two-dimensional case (1 yaw controller, 1 attitude controller), this results in the following for the motor setpoints:

$M1 = (\text{unsigned char})(\text{START_GAS} + \text{roll} + \text{yaw});$

$M2 = (\text{unsigned char})(\text{START_GAS} - \text{yaw});$

$M3 = (\text{unsigned char})(\text{START_GAS} - \text{roll} + \text{yaw});$

$M4 = (\text{unsigned char})(\text{START_GAS} - \text{yaw});$

Note:

The superposition is a simplification. In reality, the controllers influence each other. Therefore, more needs to be done during realisation!



Necessary hardware:

- EMQ3000
- QCS / QCSF
- Mini-USB cable for USART connection and flashing

Necessary software:

- Matlab version → R2019b, no additional toolboxes
- AVR Studio 32 installed (with toolchain and flip driver)
- Project code and library: **EMQ_QCSF_Matlab**



Help and background:

The exercises are to be solved with the help of the project „**EMQ_QCSF_Matlab**“. In addition, create your own file path for your Matlab files and add this to your Matlab workspace.

Exercise 1:

Based on the exercise „**Matlab Telemetry and Telecommands**“, the function **set_Yaw_angle.m** is to be implemented in this exercise, with which the yaw angle can be set by Matlab. Call this function in your programme (from exercise 4 in the last unit) in a loop with angles of **0-150 degrees** and save the plot including the target angle. Make sure that the controller has enough time to regulate the corresponding angle and only then is the next angle regulated.

Exercise 2:

The next step is to implement the functions **set_P_Yaw**, **set_I_Yaw** and **set_D_Yaw** which can be used to set the P-, I- and D parameters for the yaw angle. Vary the PID parameter and compare the control behaviour of the system. Try to set PID parameter so that the system reacts as quickly as possible but does not start to oscillate.

As **my_Attitude_Control** also controls roll and pitch, the PID parameters for roll and pitch must also be set to zero to ensure stable control of the system.



Exercise 3:

In this exercise, the aim is to realise your own PID controller in Matlab which transfers motor setpoints to AVR.

- a. To prepare the motor control via Matlab the AVR project must be configured.
Open the **basics_qcs.h** file and make the following changes:

```
#define USE_MY_ACTRL_FUNCTION    comment out  
#define USE_MY_MOTOR_STEERING  comment in  
#define ENGINE_ACTRL_ON         comment out
```
- b. Implement the function **set_Motor_parameter.m**, which sends the corresponding setpoint for each motor via USART.
- c. Now develop a PID controller in Matlab using the previous knowledge from lesson 8: „***Yaw control***“.
- d. Compare the responses with the plots saved in exercise 1.
Can you recognise any differences? What could be the reason for this?



Assistance for exercise 3:

- When implementing the PID controller, you can use the C function **my_Attitude_Control** in **AttitudeControl.c** in the Control folder as a guide.
- Use global variables for the Matlab function of the PID controller and call this function in the main programme in a loop.
- Global variables must be called first in the function in which they are described and must also be declared as global in every function and script in which they are used.
- Make sure that the controller works with the current values of the IMU that are output via the telemetry.
- If you want to use a function similar to the static C function for the I parameter, this can be realised using the persistent function. More detailed information on use can be found in the Matlab help.
- If the controller works but not regulate smoothy, the pause times can be reduced. However, it should be ensured that the pauses are not so short that errors occur in the communication.



Notes on controller dimensioning:

The following parameter/guidelines should help you: Please note that these are only approximate values as a guide, without sample time or scaling:

Sample-Time:	10ms		
Bias-Samples	1000 bis 2000		
P	3	 6	
D	0.5	 2	(slows down P)
I (optional)	0.001	 0.01	(vs. Steady-state control deviation)

Procedure for controller parameterisation:

Start with **D = 0** and **I = 0** and increase P under the system „*regulates*“. If it becomes unstable, P is already too high. If the system never reaches the target value then P is too low. Normally, the system should oscillate i.e. beat back and forth. Next, increase D to dampen overshoot. D should counteract the movement (change).

Idea:

The P component indicates how quickly/severely an error is reacted to. With the D component can then be used to slow down the system (counter-control) if, for example, an error occurs but the system is already in motion.