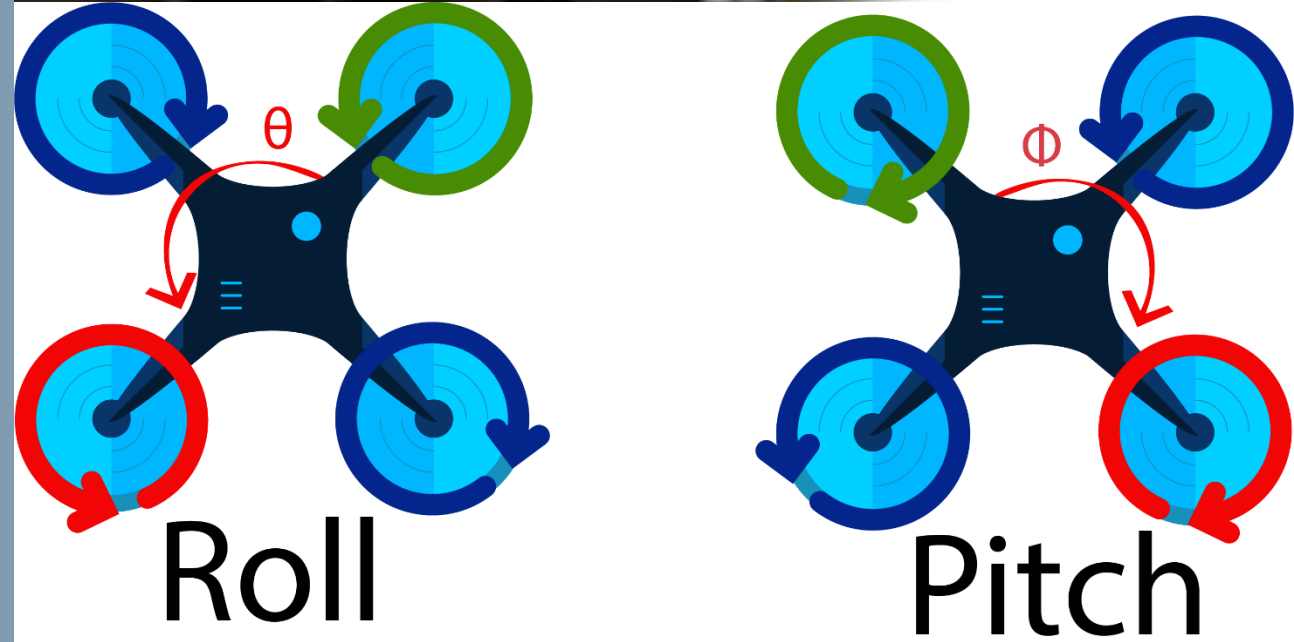


# Attitude control

Actuator technology and attitude control

Lesson 7



# Contents



- Attitude control
- Brushless motors
- Motor controller
- EMQ Framework
- Safety briefing
- Control theory
- Exercises & tips

**Time scope: approx. 2-4 hours**



# Attitude control



## Definition of attitude control:

Attitude control is the core component of the quadrocopter. It is used to keep it in the air. In contrast to other aircrafts (airplane, helicopter), the quadrocopter is an unstable system and control (roll, pitch) is therefore essential:

- 2 position controllers take over the control in the horizontal plane
- Corresponds to the problem of the seesaw control: 2 motor control





## **Brushless motors (BL-Motors):**

So-called brushless DC motors are used. In contrast to brushed motors, electronic commutation is used here. Despite the name, the functional principle corresponds to a three-phase motor with permanent magnets and so-called three phases: (+/-/0)

### **Advantages of BL-motors:**

- High power density (power / weight)
- High speeds
- Hardly any wear (only bearings, no brushes)
- Wide distribution / selection

### **Disadvantage of BL-motors:**

- A motor controller is required to generate the synchronous rotating field for the motors (motors have 3 connections)
- Costs
- Low speeds difficult or very expensive



# Motor controller



## Motor controller:

The motor controller (Brushless Controller (**BlCtrlr**), Electronic Speed Controller (**ESC**)) controls the motors and in our case is addressed via **I<sup>2</sup>C**. It therefore performs the commutation, i.e. switching the phases at the right time. One phase is always alternately connected to (+), one phase to (-) and one phase to an ADC. The voltage applied to the BL-motor (classically regulated via PWM) determines the speed, whereas the BlCtrlr always switches at zero crossing by measuring the free phase, thus ensuring continuous operation.

## Advantages of the BlCtrlr used:

- Wiring (**I<sup>2</sup>C**)
- Setpoint adjustment time < 0,5ms (fast)

**Setpoint: 1 Byte U8**

**Value range: 0 to 255**





## Control of the motors:

The framework includes two functions that significantly simplify the control of the motors:

- **Motors\_Init**

Initializes the motor driver and the communication protocol. This function must be activated once before use.

- **Motors\_run**

Starts the motors according to the setpoints transferred in the parameter of the specified struct motor\_data (do not change the struct!). The setpoints are one byte (character) per motor. The setpoints must be sent continuously, otherwise the motors will switch off.

- **steering.engine\_on**

The motors only start if steering.engine\_on is true (e.g. > 0). The control data structure steerData must not be modified.

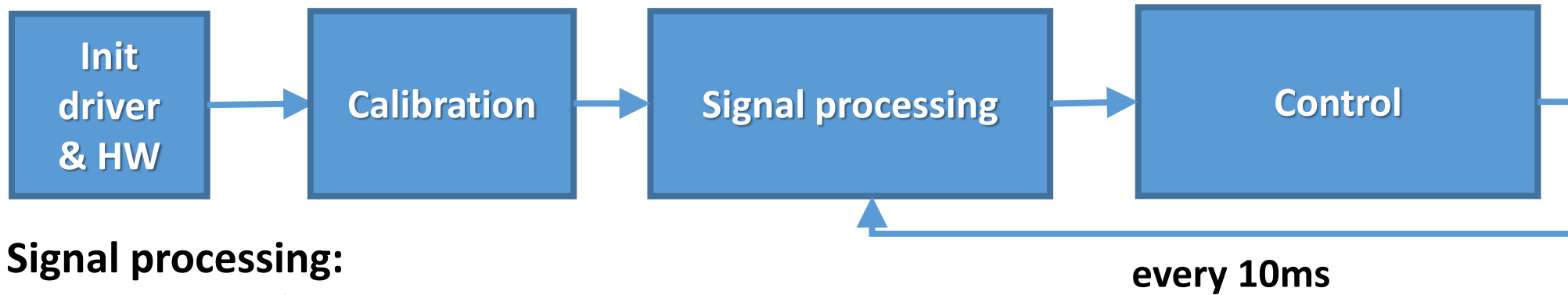
```
// Initialize motor  
communication packages  
void Motors_Init();
```

```
// Run 4 Motors with set points  
of motor_data  
inline void  
Motors_run(motor_data *m);
```

```
// Motor Values  
typedef struct {  
unsigned char M1; // Motor 1 (left)  
unsigned char M2; // Motor 2 (rear)  
unsigned char M3; // Motor 3 (right)  
unsigned char M4; // Motor 4 (front)  
} motor_data;
```



## Overview:



## Signal processing:

- Reading out the sensors
- Conditioning the values (processing)
- Data fusion (e.g. Kalman Filter)

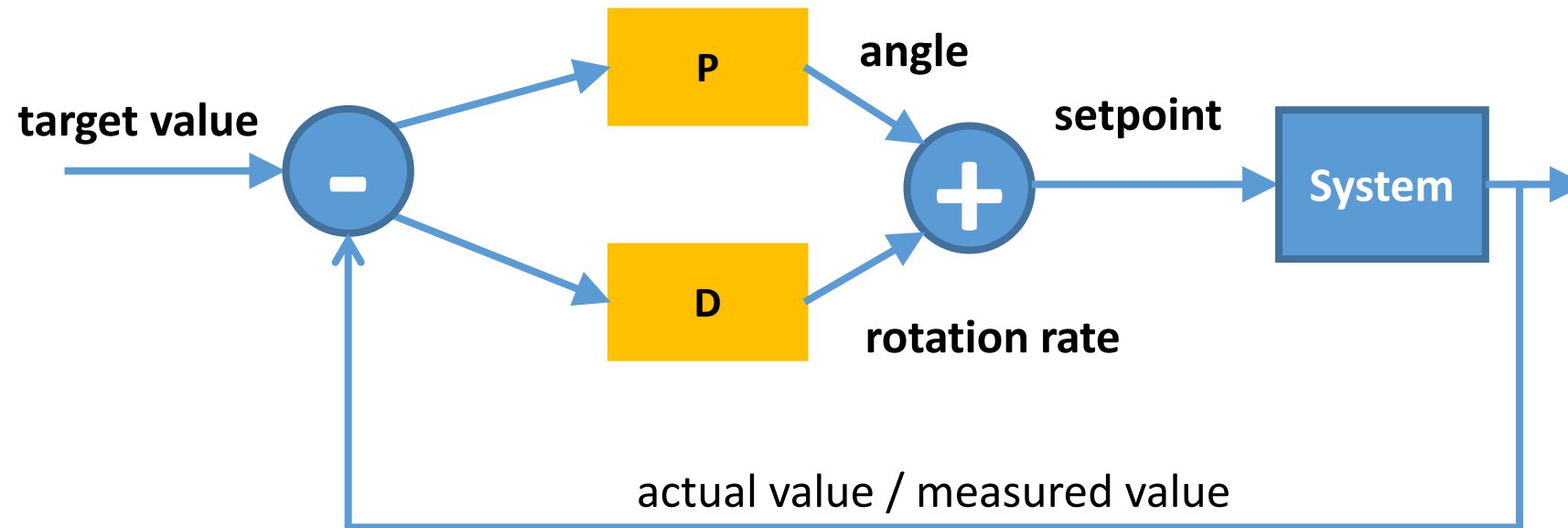
## Control:

- Implementation of the controller (**PID/PD**)
- Controlling the motors with new setpoints
- Important: The **cycle time** must always be adhered to exactly

# Control theory



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



**Problem:** 2 motors / setpoints, 1 measured value (angle)

**Solution:** Simplification by controlling the difference between the setpoints of the two motors



## Necessary hardware:

- EMQ3000
- Mini-USB cable for power and flashing
- Mini-USB cable for communication: USART to PC
- QCS in 1DOF attitude control mode

## Necessary software:

- AVR Studio 32 installed (with toolchain and flip driver)
- EMQ\_Framework (AVR-Code)
- Documents:
  - EMQ\_Framework.pdf
  - QCS/F-Quickstart.pdf
- Qt Creator 2.0.1 and Qt 4.7.0 (32bit)
- EMQt\_Framework\quatplay\quatplay.pro (Code)

# Safety briefing



## Safety briefing:

- Before starting the motors, always check that everything is tight:  
This applies to all screws, motors, booms, Y-parts, knurled screws, etc.
- **ONLY** connect the power supply unit and start the motors with the authorisation of the supervisor.
- Keep your distance. Warn neighbours. Exercise caution. Have the on/off button ready (finger over it)!



## Emergency stop options:

- On/Off button
- Switch off or disconnect the power source
- Disconnect **I2C** cable (pull out)
- Switch off **MCU** (disconnect USB cable)



## **Exercise 1a:**

Write a program that addresses the motor controllers and controls the motors. The motors are to be started and stopped by USART command.

The program should turn up each individual motor in turn from setpoint 0 to setpoint 50. This means that motor 0 starts first with the setpoint 1 and then increments to 50 in approx. 5 s.

When motor 0 has reached the value 50, motor 0 stops at the value 0 and motor 1 follows, etc.

When the command to switch off the motors is given, all motors must be switched off immediately (within 10 ms). When restarting, the program should continue or restart.

## **Exercise 1b:**

It should be possible to optionally display the four currently valid setpoints of the motors for debugging purposes. Either via console or (better but more complex) in the telemetry display (as a separate graph).

***Note: Before you try out the program (start the motors), contact the supervisor and make sure that nobody can come to any harm (note the safety instructions).***



## Exercise 2:

Implement a PD controller for attitude control that keeps the Qopter horizontal. Set the controller parameters empirically by trial and error. The task is solved when a controller can be identified that reproducibly keeps the system horizontal for a short time and also compensates for small disturbances. A perfect solution is not to be expected with this method (see next page)! Initially use **ONLY** the gyrometer.



## Notes on controller dimensioning:

The following parameters/specifications should help you: (only approximate values as a guide)

|                      |               |      |                                      |
|----------------------|---------------|------|--------------------------------------|
| <b>Sample-Time:</b>  | 10ms          |      |                                      |
| <b>Bias-Samples:</b> | 1000 bis 2000 |      |                                      |
| <b>P</b>             | 2             | 4    |                                      |
| <b>D</b>             | 0.4           | 0.8  | (slows down P)                       |
| <b>I (optional)</b>  | 0.01          | 0.02 | (vs. steady-state control deviation) |

## **Procedure for controller parameterization:**

Start with **D = 0** and **I = 0** and increase **P** until the system "**regulates**". If it becomes unstable, P is already too high. If the system never reaches the setpoint, **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. 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.



## Notes on controller dimensioning:

Finding parameters for a stable system is a complex problem. If the sensor values are already poor, e.g. jumping due to vibrations, then even an otherwise good controller will fail. It is therefore advisable to use the gyrometer only to adjust the controller for the time being, as vibrations have a direct effect on the accelerometer and it is much more difficult to parameterize both the **KF** (Kalman Filter) and the controller correctly at the same time. In other words, initially use only the measured angle values for the controller, which are obtained by integrating the gyro values. Drift can then be corrected later with the **KF**. Likewise, the cycle time must not vary (greatly), i.e. it should be strictly adhered to.

The reason for an oscillating system is, with a low **P component**, usually an excessively high **D component**. If the **D component** regulates in the wrong direction or is too high/small, the system will not be stable. First of all, the sign of the **D component** must be correct, i.e. it must counteract (dampen) a movement (change in angle). Then the **D component** must be selected so that it dampens the **P component**, but not so strongly that it itself stimulates rocking. Too high an **I component** also leads to instability due to oscillation. The **I component** is used for steady-state control deviation and should be added last with caution. Stability can be achieved without the **I component**. The **I component** should therefore always be left at 0 initially.



## **Exercise 3:**

Now use the data fusion of the Kalman Filter from the gyroscope and accelerometer. Parameterize the Kalman Filter so that vibrations do not distort the orientation. The system should no longer drift. In this context, a weak I-component should also be added to the controller so that the system overcomes the stationary control deviation.

## **Note:**

Task 3 is optional. The exercise on the Kalman Filter should be completed beforehand.