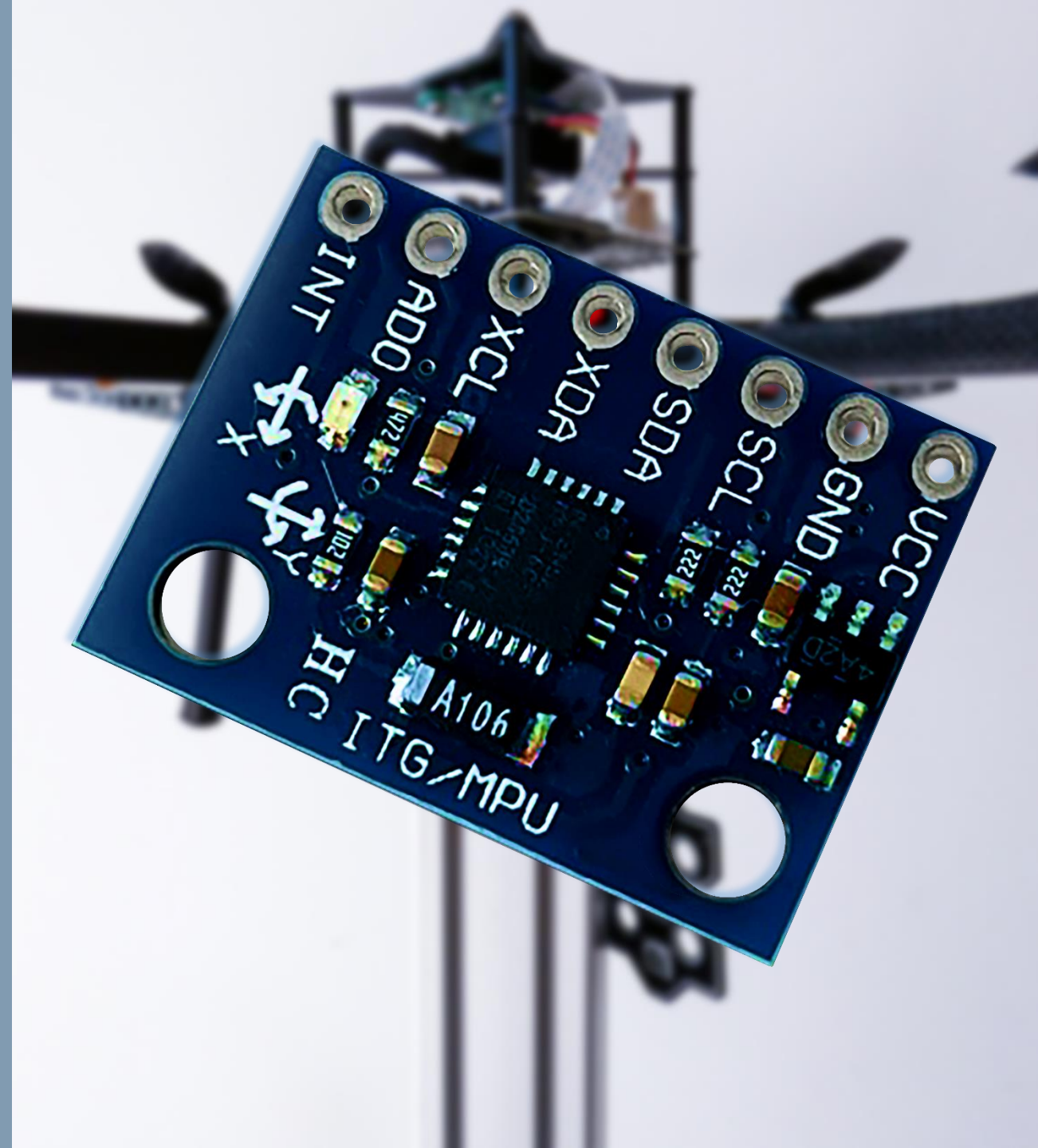


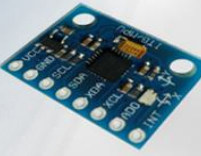
Sensors (IMU)

Inertial Measurement Unit
MPU6050

Lesson 2

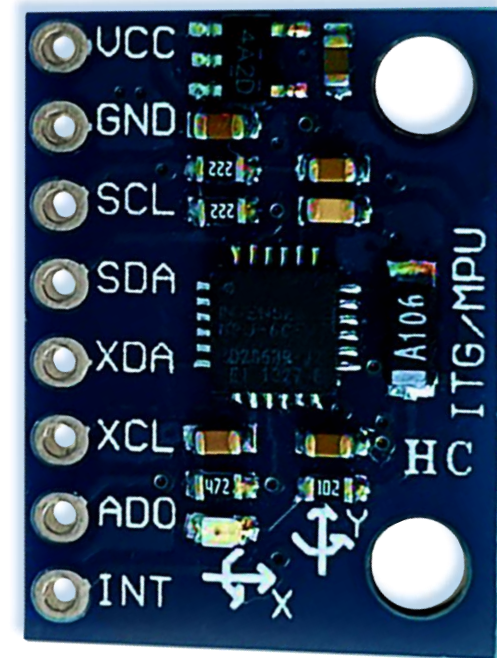


Contents

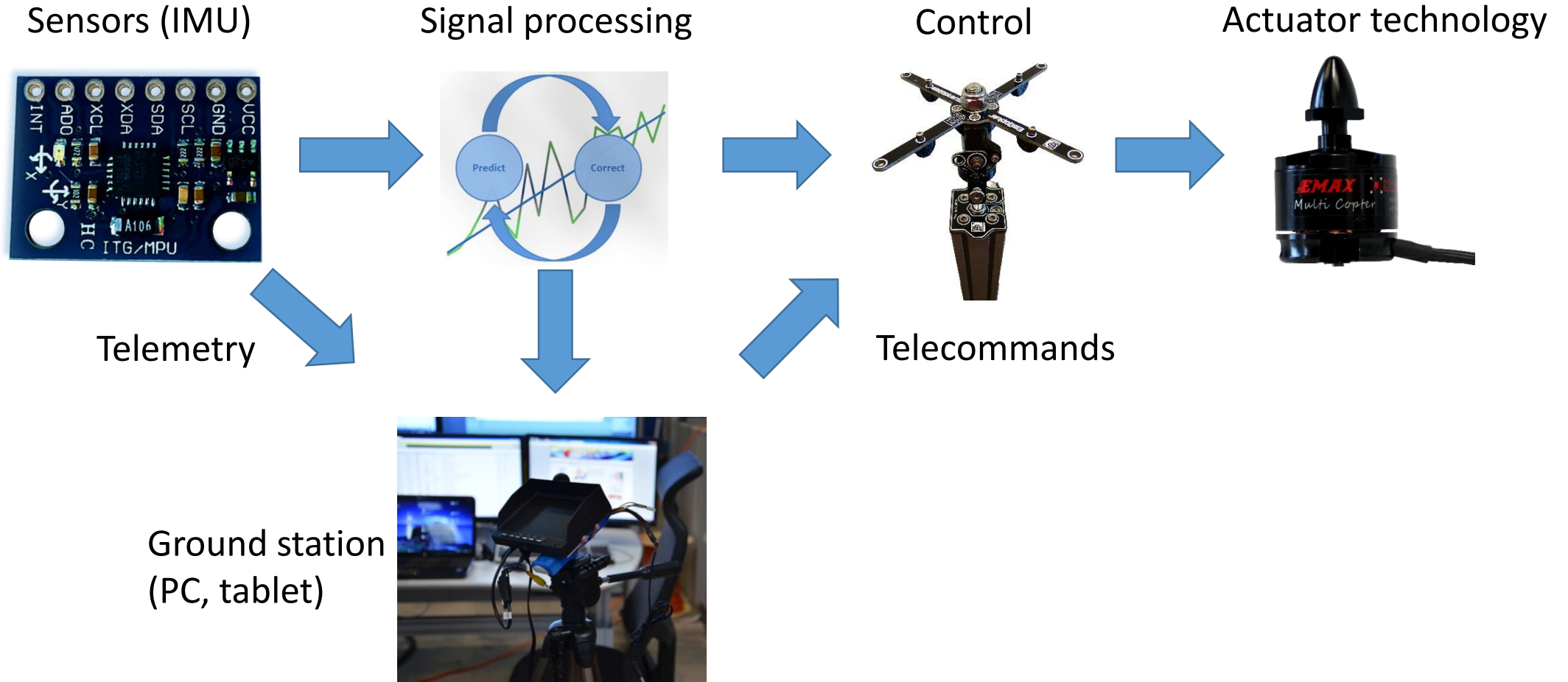


- IMU (Inertial Measurement Unit)
- TWI (Two Wire Interface)
- EMQ Framework
- Exercises

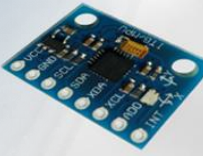
Time scope: approx. 1-3 hours



Topic overview



IMU (Inertial Measurement Unit)



Definition:

An **IMU** is a combination of several inertial sensors (inertial sensors). A triaxial structure consisting of acceleration (accelerometer) and rotation rate sensors (gyroscope) is common. The **IMU** is the sensor unit of an inertial navigation system that records changes in position and orientation. The orientation of a system can be determined based on this data. This serves as input for the position control.

Position determination (1D, simplified):

$$P = \int \int a$$

Orientation determination (1D, simplified):

$$\varphi = \int \omega$$

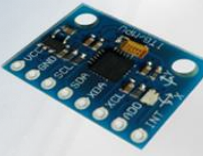
P: Position

a: Acceleration vector

φ: Orientation

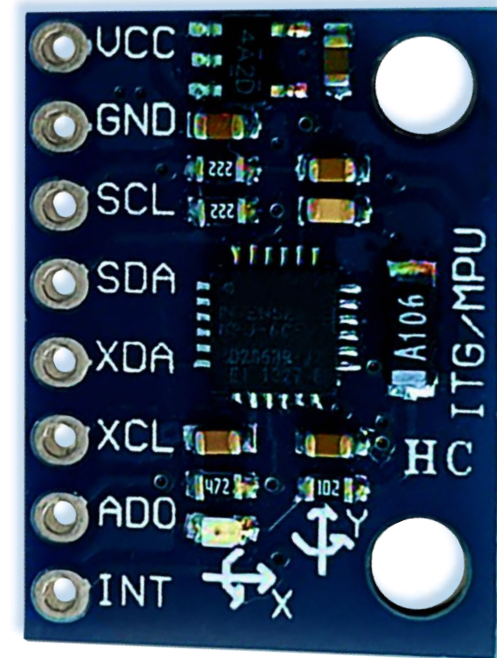
ω: Rotation rate vector

IMU (Inertial Measurement Unit)

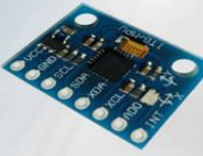


IMU ITG/MPU

- MPU6050
- Gyroscope + accelerometer
- Controlled via TWI
- Data sheets:
 - IMU_MPU_6000+6050_RegisterMap.pdf
 - IMU_MPU_6000+6050_Specs.pdf
- Before the sensor can be used, it must be configured and calibrated

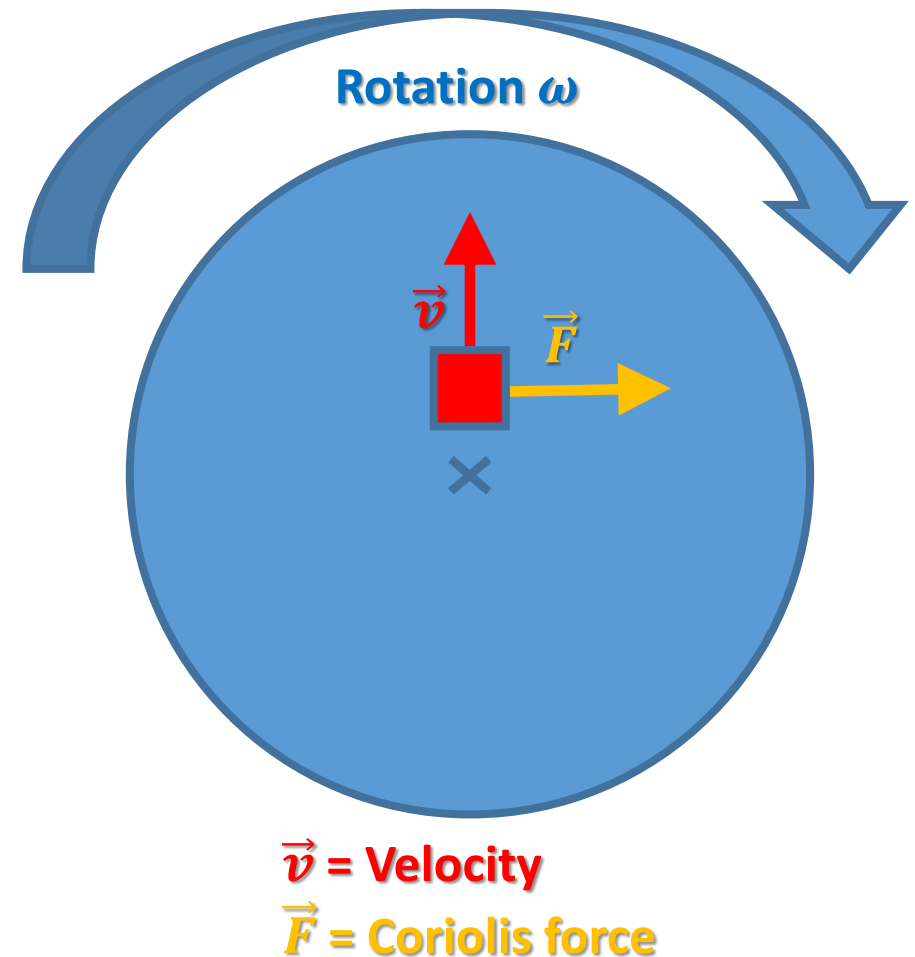


IMU (Inertial Measurement Unit)

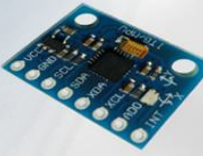


Functional principle of the gyroscope (gyro)

- Measuring principle Coriolis force (analogue to Lorenz force):
 - Moving particles ($\boldsymbol{v}$) with mass ($\boldsymbol{m}$) (cause)
 - Simultaneous rotation ($\boldsymbol{\omega}$) (mediation)
-> Effect: Force ($\boldsymbol{F}$) perpendicular to cause and mediation
 - Force is perpendicular to the axis of rotation and movement
 - Occurs in rotating reference systems ($\boldsymbol{\omega}$) with simultaneous movement ($\boldsymbol{v}$)
 - $\boldsymbol{F} = -2 \cdot \boldsymbol{m} \cdot \boldsymbol{\omega} \cdot \boldsymbol{v}$
- **MEMS** = micro electro mechanical systems
- Triaxial (x,y,z = 3 DOF)
- Rotation rate sensor: Measures rotational speeds
- Measurement: Capacitive

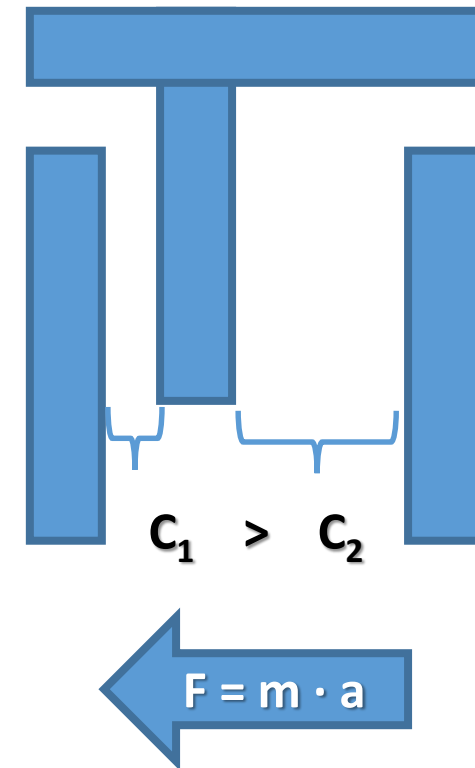
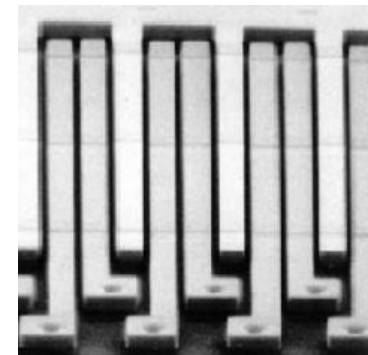


IMU (Inertial Measurement Unit)

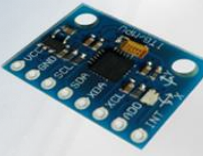


Accelerometer (Acc) operating principle

- Measuring principle Spring-mass principle
 - Two rows of combs
 - The distances change due to inertia
 - Capacitive measurement of the change in distance
- **MEMS** = micro electro mechanical systems
- Triaxial (x,y,z = 3 DOF)
- Acceleration sensor: Measures accelerations (translational)



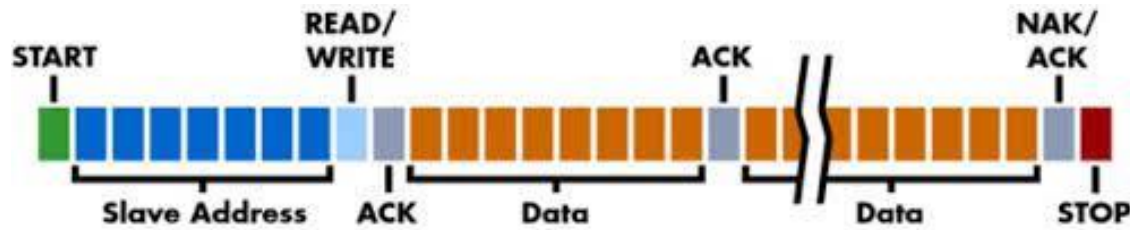
TWI (Two Wire Interface)



TWI

- Serial master-slave-bus
- Also called **I²C** or **I2C** (Inter-Integrated Circuit)
- 2 bus lines: **SCL** (serial clock), **SDA** (serial data line)
- **Advantages:** low cost, less wiring
- **Disadvantages:** susceptible to interference, short cable length
- Clock modes: standard up to **100 kHz**, fast mode up to **400 kHz**
- Transmission rate: max. **3.4 Mbit/s** (relatively slow)

TWI (Two Wire Interface)



TWI Data exchange

- Start signal
- 1 Byte: Address (7bit) + **R/W** (1bit = 1 for write)
- Acknowledge (**ACK**) from slave
- Byte-by-byte data packets each acknowledged with ACK
- Stop signal
- The specified address is the **I²C/TWI** slave address of the addressed participant

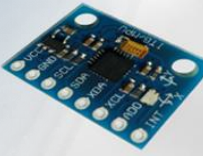
Implementation:

- Use **TWI** Framework

Data packages:

- At least 2 bytes for the **AVR** Framework
- **1st byte command** or adresse, for **R/W**
- **2nd byte** + others are **data (R/W)**
- In burst mode, several bytes are transmitted starting from a start address

TWI (Two Wire Interface)



TWI Framework

- Uses `twi_package_t` structure
- `chip` = **TWI** slave address
- `addr` = **Command**
- `addr_length` = Length of the command in byte
Usually = 1
- **buffer** = Pointer to memory area of the **data**
 - Send / Write (**W**): This data is sent
 - Receive / Read (**R**): The received data is written here
- `length` = Number of bytes that are sent / received

```
typedef struct
{
    char chip; // TWI Slave Address

    unsigned int addr; // Command Byte to be sent

    int addr_length; // Amount of Command Bytes

    void *buffer; // Buffer for Data (Send/Receive)

    unsigned int length; // Amount of Data Bytes
} twi_package_t;
```

TWI (Two Wire Interface)



TWI Framework example

```
char data_received_mpu_acc[6];
twi_package_t packet_mpu6000_acc;

packet_mpu6000_acc.chip = 0x69;
packet_mpu6000_acc.addr_length = 1;
packet_mpu6000_acc.length = 6;
packet_mpu6000_acc.addr = 0x3B;
packet_mpu6000_acc.buffer = data_received_mpu_acc;

read_from_twi(&packet_mpu6000_acc, 0);
```

Then the **6** data bytes from the sensor memory starting with address **0x3B** are in data_received_mpu_acc

A message is issued in the event of an error. This error message can be assigned using the second parameter, here **0**.

Read / receive data:

```
read_from_twi(const twi_package_t *twi, int error_id);
```

Write / Send data:

```
write_to_twi(const twi_package_t *twi);
```

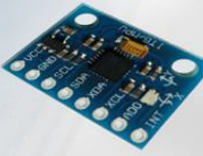
Sensor configuration (MPU6050)



Configure IMU MPU6050

- How the sensor can be configured is described in the data sheet
- **5** bytes are written for configuration (recommended see table)
- Therefore write **5x** each TWI address, command and data according to the table
- Note: **buffer** is a pointer to the data below
- TWI adress: **chip = 0x69** Alternatively: **chip = 0x68**

Configuration-Command (addr)	Configuration-Data (buffer)	Function
0x6B	0x03	Power Mode (z-reference)
0x19	0x09	Samplerate 10ms
0x1A	0x04	1kHz DLPF with 19ms delay
0x1B	0x10	+/- 1000°/s
0x1C	0x08	+/- 4g

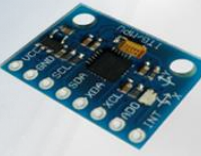


EMQ Framework

- Supplied as a library with open source part
- Specially developed for the quadrocopter lab
- Contains drivers with functions for
 - Hardware: CPU, IRQ, TC, Remote, ADC, USART, TWI, Delay, Motors, LED
 - Software:
 - Basics for an easy start
 - function templates to fill in
- For details see software documentation: EMQ_Framework.pdf
- The EMQ Framework does not use an operating system
- The entire code runs in an infinite loop (While)

EMQ Data types

- The file EMQ_Interface_Data.h contains several available EMQ data types
- The data types with "***dummy***" should later be filled with meaningful content
- Do **NOT** modify any other data types due to library dependency.
- The system time is in the variable tc_ticks [ms].



Necessary hardware:

- EMQ3000
- Mini-USB cable for power supply and flashing
- USB cable for communication / or communication via RS232 interface with adapter to PC
- Sensor **IMU**(MPU6050) + extra cable

Necessary software:

- AVR Studio 32 installed (with toolchain and flip driver)
- EMQ Framework (code)
- Documents:
 - EMQ_Framework.pdf
 - IMU_MPU_6000+6050_RegisterMap.pdf
 - IMU_MPU_6000+6050_Specs.pdf

Exercises



Exercise 1a:

Familiarise yourself with the EMQ Framework. Read the documentation and look at all source code files before you start.

Exercise 1b:

Configure the two sensors. Complete the `my_imu_init()` function. Use the values from slide 14 and the framework from slides 12 and 13.

Exercise 1c:

Read and condition the sensors. To do this, you must look up the register address of the values in the data sheet. Conditioning means that the raw data from the sensors is converted into correctly scaled values. Each sensor supplies 6 bytes, i.e. one high and one low byte per axis. The scaling factors can be found in the data sheets.

Exercises



Exercise 2:

Calibrate the sensors. Calculate an average value over an adjustable number of measurements. Use this to correct the bias. The gyro is to be calibrated to **0/0/0** and the Acc is to be calibrated to the gravity vector.

Exercise 3a:

Perform the one-dimensional integration for the measurements of the gyro by integrating the three axes independently. A 90° rotation should also result in 90°.

Note:

The sensor should be placed on the table and connected using an extra cable.

Exercise 3b:

The three angles should be shown in the display. To do this, modify the `renew_display()` function.