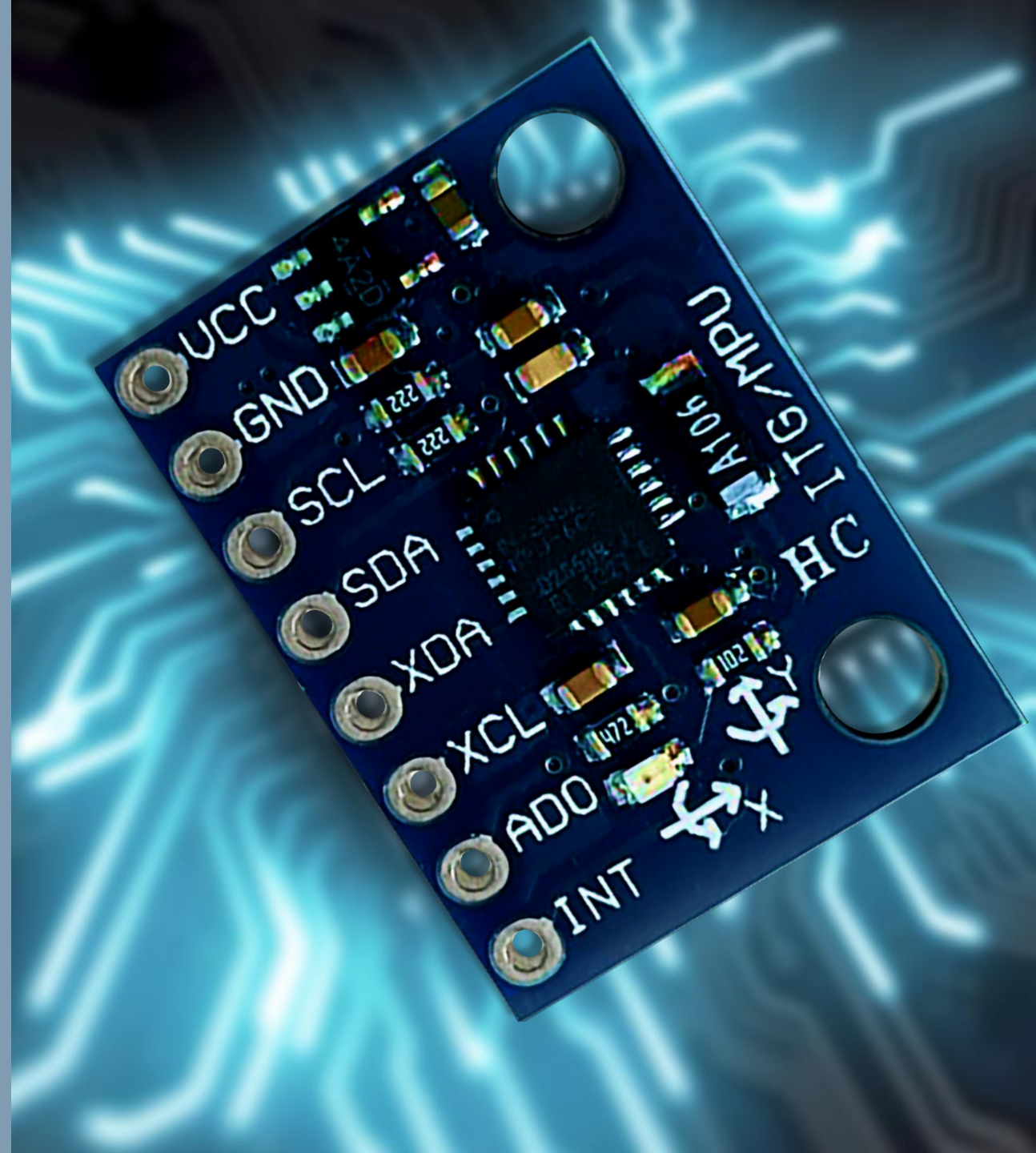


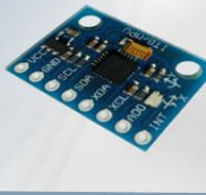
Signal processing

Quaternion

Lesson 4

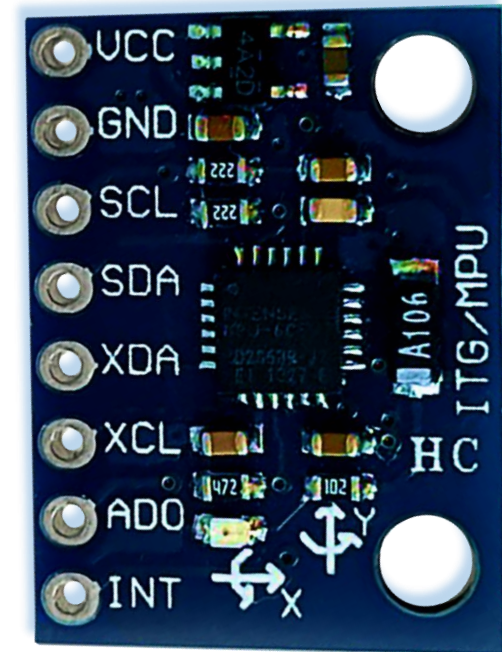


Contents



- Signal processing
- Quaternions
- Advantages and disadvantages of quaternions
- Implementation
- Exercises 1-4

Time scope: approx. 1-3 hours



Signal processing



- General (abbreviated): The Signal processing includes all processing steps that aim to extract information from a signal.
- **Here:** Reading out, preparing, filtering, further processing and fusing informations from the sensor, i.e.:
 - Reading out → Obtaining raw data from the sensors
 - Preparing → Scaling, conditioning (calibrating)
 - Filtering → Eliminate further error / noises
 - **Further processing** → Further processing of data → e.g. representation as quaternion
 - Fusing → Merging data from several sources → e.g. Kalman Filter



What are quaternions?

- Quaternions q are a **4D-number system** of their own.
- Comparable to imaginary numbers (three „*imaginary*“ parts: i, j, k)
- Each quaternion consists of a scalar part x_0 and a vector part $\vec{x} = [x_1, x_2, x_3]$
- A normalized quaternion has the absolute value $\|q\| = 1$
- The quaternion multiplication is not commutative
- The orientation of a body in three-dimensional space can be uniquely described by a quaternion

$$q = [x_0 \quad x_1 \quad x_2 \quad x_3]^T$$

$$q = x_0 + \vec{x}$$

$$q = x_0 + x_1 \cdot i + x_2 \cdot j + x_3 \cdot k,$$

$$\forall x_0, x_1, x_2, x_3 \in \mathbb{R},$$

$$\text{with } i^2 = j^2 = k^2 = ijk = -1$$

$$x_0^2 + x_1^2 + x_2^2 + x_3^2 = 1 \text{ (normalization)}$$

Quaternion multiplication: let w, q, p be quaternions

$$w = q \odot p = [x_0 \quad x_1 \quad x_2 \quad x_3]^T \odot [y_0 \quad y_1 \quad y_2 \quad y_3]^T = \begin{pmatrix} x_0 & -x_1 & -x_2 & -x_3 \\ x_1 & x_0 & -x_3 & x_2 \\ x_2 & x_3 & x_0 & -x_1 \\ x_3 & -x_2 & x_1 & x_0 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} 1$$



What are quaternions?

- A quaternion corresponds to a rotational transformation
- The quaternion multiplication corresponds to a rotation
- Rotation of a 3D vector $\mathbf{v}$ to $\mathbf{w}$:

$$\mathbf{w} = \mathbf{q} \odot [\mathbf{0}, \mathbf{v}]^T \odot \mathbf{q}^*$$

- $\mathbf{q}^*$ is the conjugate complex of the quaternion $\mathbf{q}$
- Conversion to rotation matrix:

$$\mathbf{C} = \begin{pmatrix} (q_1^2 + q_2^2 - q_3^2 - q_4^2) & 2(q_2 q_3 - q_1 q_4) & 2(q_2 q_4 + q_1 q_3) \\ 2(q_2 q_3 + q_1 q_4) & (q_1^2 - q_2^2 + q_3^2 - q_4^2) & 2(q_3 q_4 - q_1 q_2) \\ 2(q_2 q_4 - q_1 q_3) & 2(q_3 q_4 + q_1 q_2) & (q_1^2 - q_2^2 - q_3^2 + q_4^2) \end{pmatrix}$$

- Conversion of RPY (ϕ, θ, ψ) to quaternions:

$$\mathbf{q} = \begin{bmatrix} \cos(\phi/2) \cos(\theta/2) \cos(\psi/2) + \sin(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \sin(\phi/2) \cos(\theta/2) \cos(\psi/2) - \cos(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \sin(\theta/2) \cos(\psi/2) + \sin(\phi/2) \cos(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \cos(\theta/2) \sin(\psi/2) - \sin(\phi/2) \sin(\theta/2) \cos(\psi/2) \end{bmatrix}$$

$$\mathbf{q} = [x_0 \quad x_1 \quad x_2 \quad x_3]^T$$

$$\mathbf{q}^* = [x_0 \quad -x_1 \quad -x_2 \quad -x_3]^T$$

- Conversion quaternion to RPY (ϕ, θ, ψ) :

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(x_0 x_1 + x_2 x_3), 1 - 2(x_1^2 + x_2^2)) \\ \text{asin}(2(x_0 x_2 - x_1 x_3)) \\ \text{atan2}(2(x_0 x_3 + x_1 x_2), 1 - 2(x_2^2 + x_3^2)) \end{bmatrix}$$

Quaternions



What are quaternions?

- Quaternions as a rotation (transformation) of two vectors
- $\vec{v}$ and $\vec{s}$ are two vectors with the angle $\cos \phi = \vec{v} \cdot \vec{s}$
- Then the quaternion q , as the axis of rotation that rotates $\vec{v}$ in $\vec{s}$, is calculated as follows:

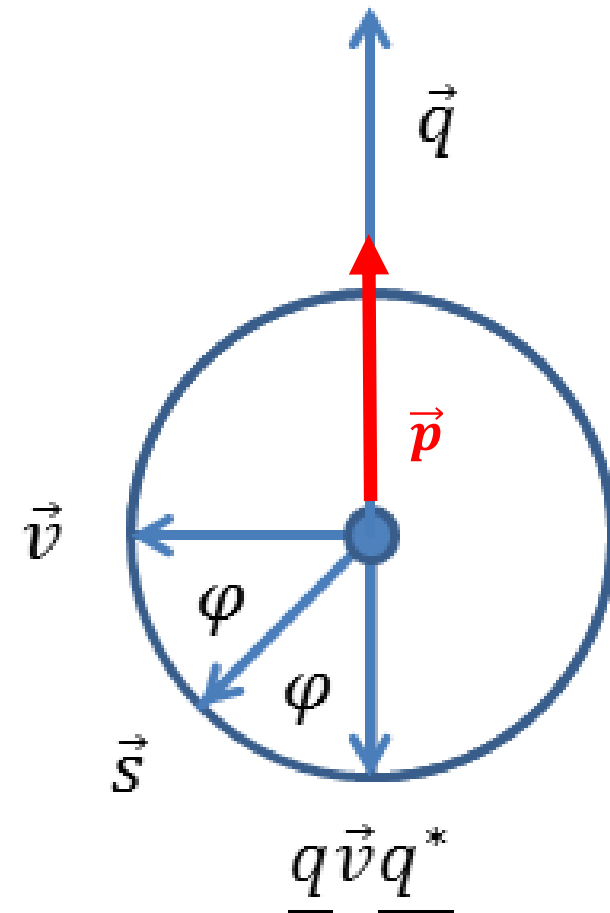
$$\vec{u}_q = \frac{\vec{v} \times \vec{s}}{|\vec{v} \times \vec{s}|}$$

$$q = x_0 + \vec{q} = (\cos \phi, \vec{u}_q \cdot \sin \phi)$$

To obtain the quaternion p of $\vec{v}$ to $\vec{s}$, rotate the angle by half

$$\varphi = \frac{\phi}{2}: p = x_0 + \vec{p} = (\cos \frac{\phi}{2}, \vec{u}_q \cdot \sin \frac{\phi}{2})$$

- **The vector part** corresponds to the **axis of rotation**
- **The scalar part** corresponds to the **angle of rotation**





Advantages and disadvantages of quaternions

Orientation representation	Redundant elements	Clarity	Singular points	Performance
Rotation matrix	6	low	no	good
Euler angle	0	good	yes	bad
Rotation vector & -angle	1	low	yes	bad
Quaternions	1	very low	no	very good

Quaternions compared with other orientation representations

- **Redundant elements:** A rotation in three-dimensional space is completely described by 3 elements.
- **Clarity:** How well can the represented orientation be understood without any help tools.
- **Singular points:** Do discontinuity points (division by **0**) occur in the calculation, also in relation to ambiguities and gimbal locks.
- **Performance:** Higher performance means less computationally intensive during the implementation.

Implementation



- Quaternions are the method of choice when implementing an orientation representation (see table on slide 7)
- Euler angles only for user representation (no continuous representation)
- The following **cooking recipe** is used for implementation (determination of orientation with an IMU):
 - 1.) Gyroscope provides rotation rates $\underline{\omega}$
 - 2.) Integration of the rotation rates provides angles, **but** rotations must be „*rotated up*“ infinitesimally in each time step, because rotations are not commutative, therefore: $\Delta \underline{\phi} = \underline{\omega} \cdot T$
 - 3.) Transformation to quaternion, i.e. $\Delta \underline{\phi}$ interpreted as RPY/DCM¹ → e.g. transform-RPY-to-quaternion, i.e. $\Delta \underline{\phi} \rightarrow d\underline{q}$ (interpretation as RPY/DCM is correct as it is valid for small angle).
 - 4.) Normalize
 - 5.) Determine the orientation quaternion $\underline{Q}(n + 1) = \underline{Q}(n) \odot d\underline{q}$
 - 6.) Normalize

¹: *Direction Cosine Matrix*



Necessary hardware:

- EMQ3000
- Mini-USB cable for power and flashing
- Sensor MPU6050 or IMU 3000 + extra cable

Necessary software:

- AVR Studio 32 installed (with Toolchain and flip driver)
- EMQ Framework (code)
- Documents:
 - EMQ_Framework.pdf

Exercises



Exercise 1:

Familiarize yourself with the EMQ Framework. Read the documentation and look at all source code files, before you start. Especially the quaternions module template is needed in this exercise: Quaternion.c & Quaternion.h

Exercise 2:

Now fill out the module and continuously determine the change in orientation in three-dimensional room with the help of quaternions by implementing the cooking recipe. Let the resulting quaternion be continuously output in Euler angle on the screen.

Exercise 3:

Compare the result with the simplified integration without quaternions. Show both on the output screen. What do you find? In particular, you should rotate over several axis when testing.

Exercise 4:

Now perform a roll, yaw, and pitch movement of **90°** each, after which the sensor is back in its initial position. The result of the quaternion should now be **approx. 0°** (optimum), whereby errors up to **max. of 10°** due to inaccuracies (calibration, drift, quantization) are still acceptable. A larger error is definitely a programming error.