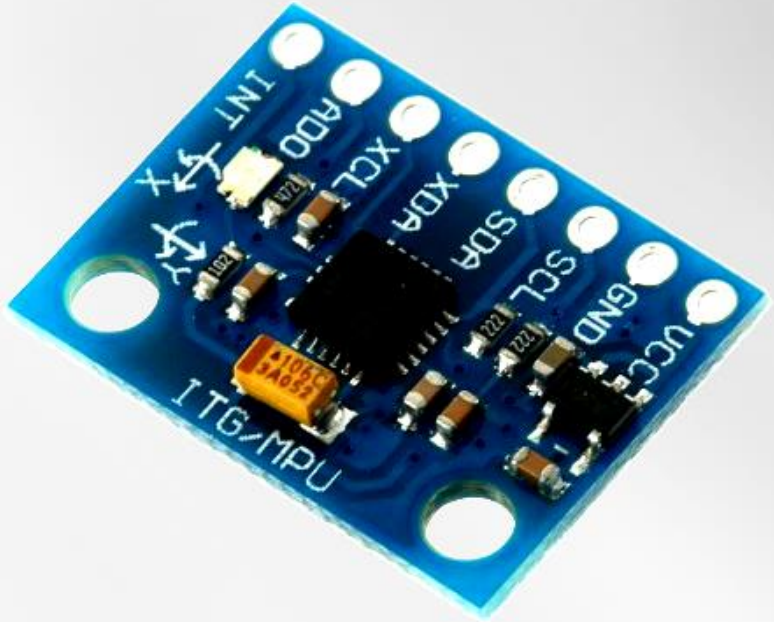
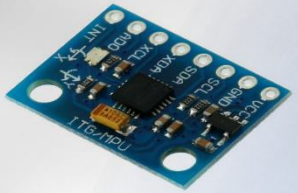
Signalverarbeitung

Quaternion

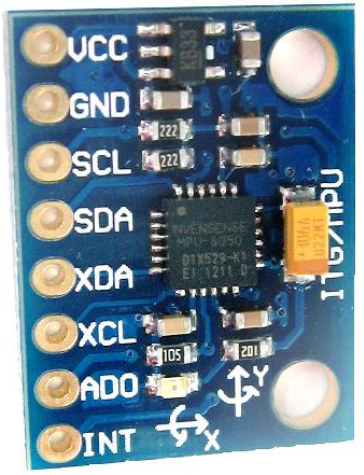


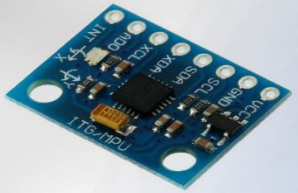


Inhalt

Umfang: ca. 1-3 Zeitstunden

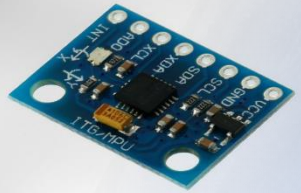
- Signalverarbeitung
- Quaternionen
- Vor- und Nachteile von Quaternionen
- Implementierung
- EMQ Framework
- Aufgaben 1-4





Signalverarbeitung

- Allgemein (gekürzt): Die Signalverarbeitung beinhaltet alle Bearbeitungsschritte, die das Ziel haben, Informationen aus einem Signal zu extrahieren.
- Hier: Informationen der Sensoren auslesen, aufbereiten, filtern, weiterverarbeiten, fusionieren, d.h.:
  - Auslesen -> Rohdaten der Sensoren erhalten
  - Aufbereiten -> Skalieren, Konditionieren (Kalibrieren)
  - Filtern -> Weitere Fehler / Rauschen beseitigen
  - **Weiterverarbeiten-**> z.B. Daten weiter verrechnen -> z.B. Repräsentation als Quaternion
  - Fusionieren -> Daten mehrerer Quellen zusammenführen -> z.B. Kalman Filter



Quaternionen

Was sind Quaternionen?

- Quaternionen q sind ein eigenes 4D Zahlensystem.
- Vergleichbar mit imaginären Zahlen (drei „imaginäre“ Anteile: i, j, k)
- Jedes Quaternion besteht aus einem Skalarteil x_0 und Vektorteil $\underline{x} = [x_1, x_2, x_3]$
- Ein normalisiertes Quaternion hat den Betrag 1.
- Die Quaternionenmultiplikation ist nicht kommutativ
- Die Orientierung eines Körpers im dreidimensionalen Raum lässt sich durch ein Quaternion eindeutig beschreiben

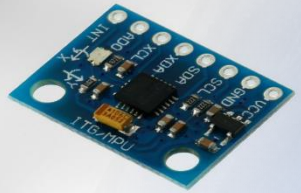
$$q = [x_0 \quad x_1 \quad x_2 \quad x_3]^T$$

$$q = x_0 + x_1 \cdot i + x_2 \cdot j + x_3 \cdot k,$$
$$\forall\, x_i \in \mathbb{R}, i \in [0, 1, 2, 3],$$
mit $i^2 = j^2 = k^2 = -1$

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

Quaternionenmultiplikation :
w, q, p seien Quaternionen

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



Quaternionen

Was sind Quaternionen?

- Ein Quaternion entspricht einer rotatorischen Transformation
- Die Quaternionenmultiplikation entspricht einer Drehung
- Rotation eines 3D-Vektors **v** zu **w**:

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

- q^* ist das konjugiert, komplexe des Quaternion q
- Umrechnung in Rotationsmatrix:

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

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

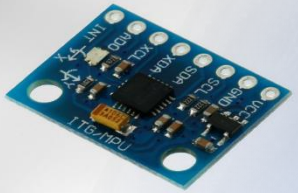
$$q^* = [x_0 \quad -x_1 \quad -x_2 \quad -x_3]^T$$

- Umrechnung von RPY (Φ,Θ,Ψ) in Quaternionen:

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

- Umrechnung von Quaternion in RPY (Φ,Θ,Ψ) :

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(2(x_0x_1 + x_2x_3), 1 - 2(x_1^2 + x_2^2)) \\ \text{asin}(2(x_0x_2 - x_1x_3)) \\ atan2(2(x_0x_3 + x_1x_2), 1 - 2(x_2^2 + x_3^2)) \end{bmatrix}$$



Quaternionen

Was sind Quaternionen?

- Quaternionen als Drehung (Transformation) zweier Vektoren
- v und s sind zwei Vektoren mit dem Winkel cos Φ = v · s
- Dann berechnet sich das Quaternion q, dass v in s rotiert, wie folgt:

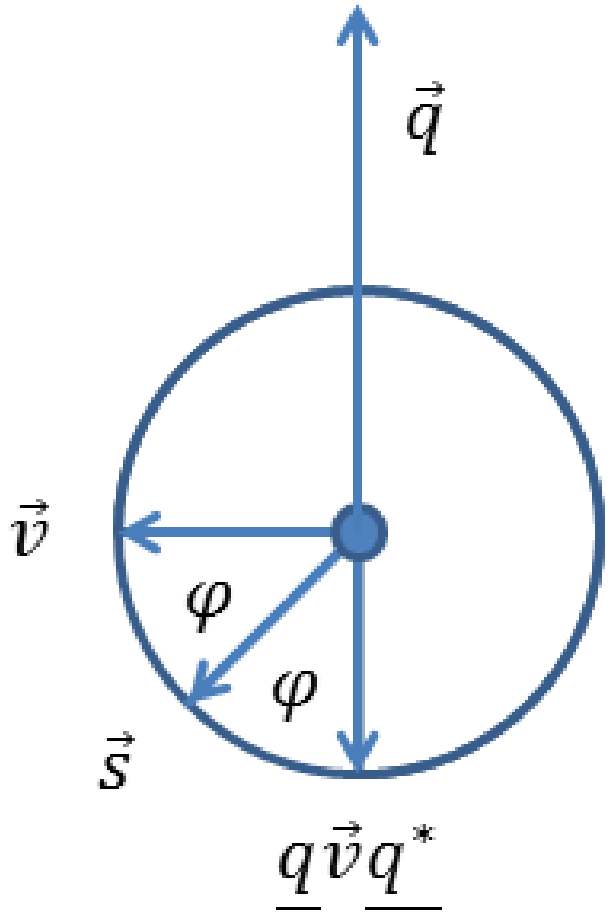
$$\underline{U}_q = \frac{v \times s}{|v \times s|}$$

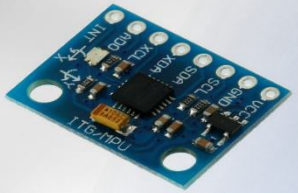
$$q = x_0 + \underline{\mathbf{x}} = (\cos \Phi, \underline{U}_q \cdot \sin \Phi)$$

Um das Quaternion p von v zu s zu erhalten drehe um den halben Winkel:

$$p = x_0 + \underline{\mathbf{x}} = (\cos \frac{\Phi}{2}, \underline{U}_q \cdot \sin \frac{\Phi}{2})$$

- Der Vektorteil entspricht der Drehachse
- Der Skalarteil entspricht dem Drehwinkel



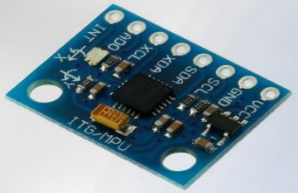


Vor- und Nachteile von Quaternionen

Orientierungsdarstellung	Redundante Elemente	Anschaulichkeit	Singuläre Stellen	Performance
Rotationsmatrix	6	gering	nein	gut
Euler-Winkel	0	gut	ja	schlecht
Drehvektor & Drehwinkel	1	gering	ja	-
Quaternionen	1	sehr gering	nein	sehr gut

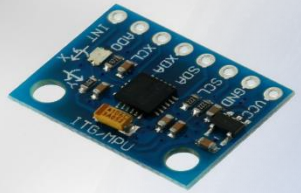
Quaternionen verglichen mit anderen Orientierungsdarstellungen

- Redundante Elemente: Eine Drehung im dreidimensionalen Raum ist durch 3 Elemente vollkommen beschrieben.
- Anschaulichkeit: Wie gut kann man die dargestellte Orientierung ohne Hilfsmittel begreifen.
- Singuläre Stellen: Treten bei der Berechnung Unstetigkeitsstellen (Division durch 0) auf,
  auch in Zusammenhang mit Mehrdeutigkeiten und Gimbal Locks.
- Performance: Höhere Performance bedeutet weniger rechenintensiv bei der Umsetzung.



Implementierung

- Bei der Implementierung einer Orientierungsdarstellung sind Quaternionen Mittel der Wahl (SdT), vgl. Folie 8
- Euler-Winkel werden nur noch zur Benutzer-Darstellung verwendet (keine kontinuierliche Darstellung)
- Zur Umsetzung dient folgendes **Kochrezept** (Bestimmung der Orientierung mit einer IMU):
    1.) Gyroskop liefert Drehraten $\underline{\omega}$
    2.) Integration der Drehraten liefert Winkel, **aber** Rotationen müssen infinitesimal in jedem Zeitschritt „aufrotiert“ werden, denn Rotationen sind nicht kommutativ, deshalb: $\underline{\Delta\phi} = \underline{\omega} \cdot T$
    3.) Transformation in Quaternion, d.h. $\underline{\Delta\phi}$ interpretiert als RPY/DCM -> z.B. Transform-RPY-zu-Quaternion, d.h. $\underline{\Delta\phi}$ -> $\underline{dq}$ (Interpretation als RPY/DCM ist richtig, da für kleine Winkel gültig.)
    4.) Normieren
    4.) Orientierungsquaternion $\underline{Q}(n+1) = \underline{Q}(n) \odot \underline{dq}$ bestimmen
    5.) Normieren



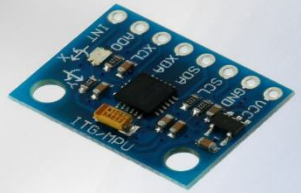
Implementierung

Programmierhilfen / Short-Cuts / Features:

- Strg halten + Linke Maustaste oder F3 und dabei Typ / Variable / Funktion selektieren -> Sprung zur Deklaration
- Strg + Leertaste (Autovervollständigung)
- Strg + B (Build all)
- Strg + Alt + H oder Rechtsklick -> Open Call Hierarchie:
  Vorher eine Funktion auswählen, dann wird angezeigt, wo diese überall aufgerufen wird.
- Strg + Alt + Pfeiltaste hoch/runter: Zeile kopieren und oben/unten einfügen

Wenn etwas nicht geht, hilft es manchmal, das Projekt neu zu öffnen (Import), zu kompilieren oder Index zu aktualisieren (Rebuild).

Wer mehr weiß, gerne per E-Mail melden!



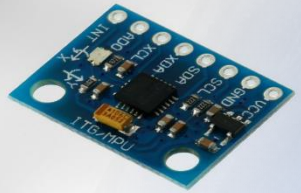
EMQ Framework

EMQ Framework

- Mitgeliefert als Library mit Open-Source-Teil
- Extra für das Quadrokoter-Labor entwickelt
- Enthält Treiber mit Funktionen für
  - Hardware: CPU, IRQ, TC, Remote, ADC, USART, TWI, Delay, Motoren, LED
  - Software:
    - Grundlagen zum Einfachen starten
    - Funktionsvorlagen zum Ausfüllen
- Details siehe Softwaredokumentation: EMQ_Framework.pdf
- Das EMQ Framework verwendet kein Betriebssystem
- Der gesamte Code läuft in einer Endlosschleife (While)

EMQ Datentypen

- In der Datei EMQ_Interface_Data.h stehen einige, verfügbare EMQ Datentypen
- Die Datentypen mit dummy sollen später mit sinnvollen Inhalt gefüllt werden
- Alle anderen Datentypen **NICHT** modifizieren.
- Die Systemzeit steht in der Variable tc_ticks [ms].



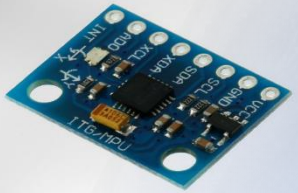
Aufgaben

Notwendige Hardware:

- EMQ3000
- Mini USB Kabel für Strom und zum Flashen
- Sensor MPU6050 oder IMU 3000 + Extra-Kabel

Notwendige Software:

- AVR Studio 32 installiert (mit Tool Chain und FLIP Treiber)
- EMQ Framework (Code)
- Dokumente:
  - EMQ_Framework.pdf



Aufgaben

Aufgabe 1:
Mache dich mit dem EMQ Framework vertraut. Lies die Dokumentation und gucke dir alle Quellcode-Dateien an, bevor du anfängst. Insbesondere die Quaternionen-Modul-Vorlage wird in dieser Übung benötigt: Quaternion.c & Quaternion.h

Aufgabe 2:
Fülle nun das Modul aus und bestimme fortlaufend die Orientierungsänderung im dreidimensionalen Raum mit Hilfe von Quaternionen, indem du das Kochrezept umsetzt. Lass dir das resultierende Quaternion fortlaufend in Eulerwinkeln auf dem Display ausgeben.

Aufgabe 3:
Vergleiche das Ergebnis mit der vereinfachten Integration ohne Quaternionen. Lass beides auf dem Display ausgeben. Was stellst du fest? Insbesondere sollst du beim Testen über mehrere Achsen drehen.

Aufgabe 4:
Führe nun eine je 90° Roll-, Gier- und Pitch-Bewegung durch, nach der sich der Sensor wieder in der Ausgangslage befindet. Das Ergebnis des Quaternion sollte nun ca. 0° (optimal) sein, wobei Fehler bis max. 10° aufgrund von Ungenauigkeiten (Kalibrierung, Drift, Quantisierung) noch akzeptabel sind. Ein größerer Fehler ist definitiv ein Programmierfehler.