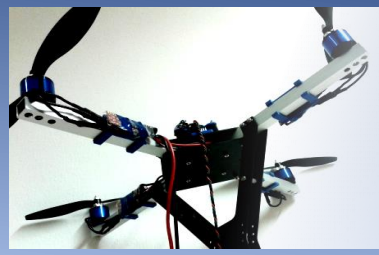
Lageregelung

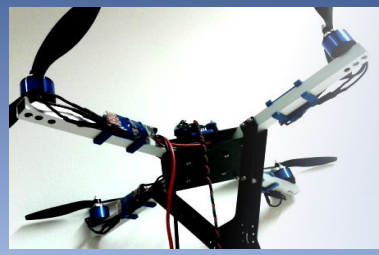
Aktuatorik und Lageregelung



Inhalt

Umfang: ca. 2-4 Zeitstunden

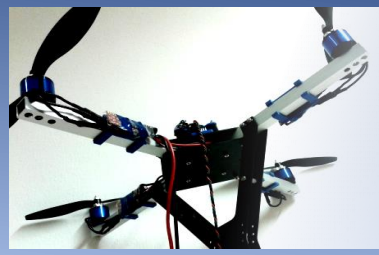
- Lageregelung
- Bürstenlose Motoren
- Motorregler
- EMQ Framework
- Sicherheitseinweisung
- Theorie zur Regelung
- Aufgaben & Hinweise



Lageregelung

Definition Lageregelung:
Die Lageregelung (engl. Attitude Control) ist das Kernstück des Quadrokopters. Durch sie wird er in der Luft gehalten. Im Gegensatz zu anderen Fluggeräten (Flugzeug, Hubschrauber) ist der Quadrokopter ein instabiles System und eine Regelung (Roll, Pitch) somit unverzichtbar:

-> 2 Lageregler übernehmen die Lageregelung in der z- Ebene
-> Entspricht dem Problem der Wippenregelung: 2 Motorenregelung

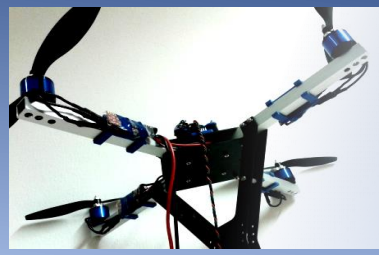


Bürstenlose Motoren

Bürstenlose Motoren (engl. Brushless Motors, Bl-Motor):
Es werden sogenannte bürstenlose Gleichstrommotoren verwendet. Im Gegensatz zu Bürstenmotoren wird hier also eine elektronische Kommutierung verwendet. Trotz des Namens entspricht das Funktionsprinzip einem Drehstrom-Motor mit Permanentmagneten und sogenannten drei Phasen: (+/-/0)

Vorteile der Bl-Motoren:
- Hohe Leistungsdichte (Leistung / Gewicht)
- Hohe Drehzahlen
- Kaum Verschleiß (nur Lager, keine Bürsten)
- Große Verbreitung / Auswahl

Nachteil:
- Es wird ein Motorregler benötigt, womit das synchrone Drehfeld für die Motoren erzeugt werden kann (Motoren haben 3 Anschlüsse).
- Kosten
- Niedrige Drehzahlen schwierig bzw. sehr teuer



Motorregler

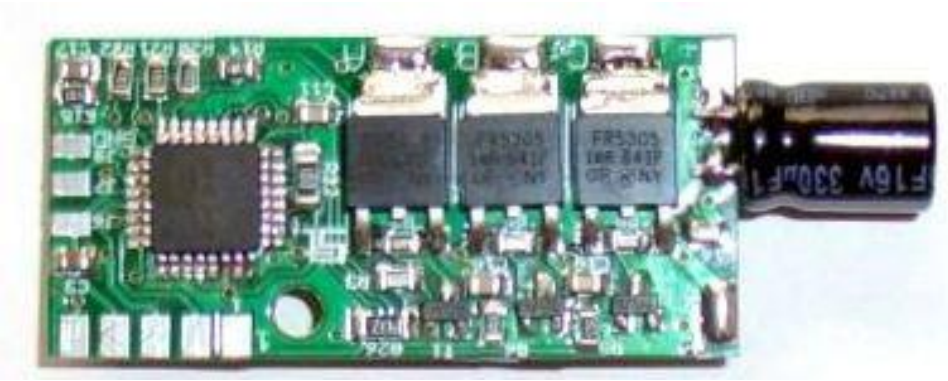
Motorregler:

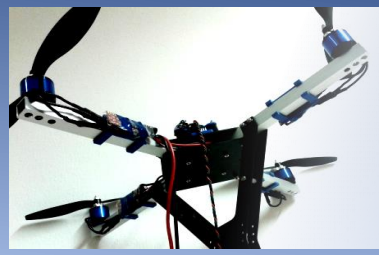
Der Motorregler (engl. Brushless Controller (BlCtrler), Electronic Speed Controller (ESC)) steuert die Motoren an und wird per I²C angesprochen. Er leistet also die Kommutierung, d.h. das Umschalten der Phasen zum richtigen Zeitpunkt. Dabei ist stets wechselnd eine Phase mit (+), eine Phase mit (-) und eine Phase mit einem ADC verbunden. Die angelegte Spannung am Bl-Motor (klassisch via PWM reguliert) gibt die Geschwindigkeit vor, wobei der BlCtrler durch Messung der freien Phase stets beim Nulldurchgang umschaltet und so einen kontinuierlichen Lauf gewährleistet.

Vorteile der verwendeten BlCtrler:

- Verkabelung (I²C)
- Sollwertstellzeit < 0,5ms (schnell)

Stellwert: 1 Byte U8
Wertebereich 0 bis 255





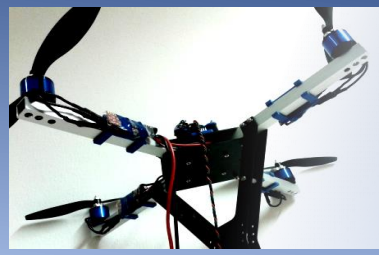
EMQ Framework

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].



EMQ Framework

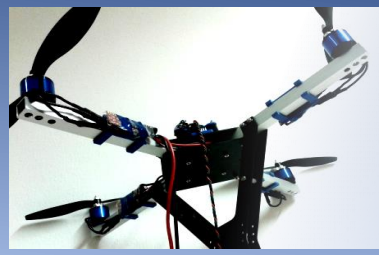
Ansteuerung der Motoren:

Zum Framework gehören zwei Funktionen, die das Ansteuern der Motoren deutlich vereinfachen:

- Motors_Init
  Initialisiert den Motor-Treiber sowie das Kommunikationsprotokoll. Diese Funktion muss vor der Verwendung einmalig aufgerufen werden.
- Motors_run
  Startet die Motoren entsprechend der im Parameter übergebenen Stellwerte der spezifizierten Struct motor_data (Struct nicht ändern!). Die Stellwerte sind je ein Byte (Character) pro Motor. Die Stellwerte müssen kontinuierlich gesendet werden, sonst schalten die Motoren aus.
- steering.engine_on
  Die Motoren starten nur, sofern steering.engine_on wahr ist (z.B. > 0). Die Steuerdaten Struktur steerData darf nicht modifiziert werden.

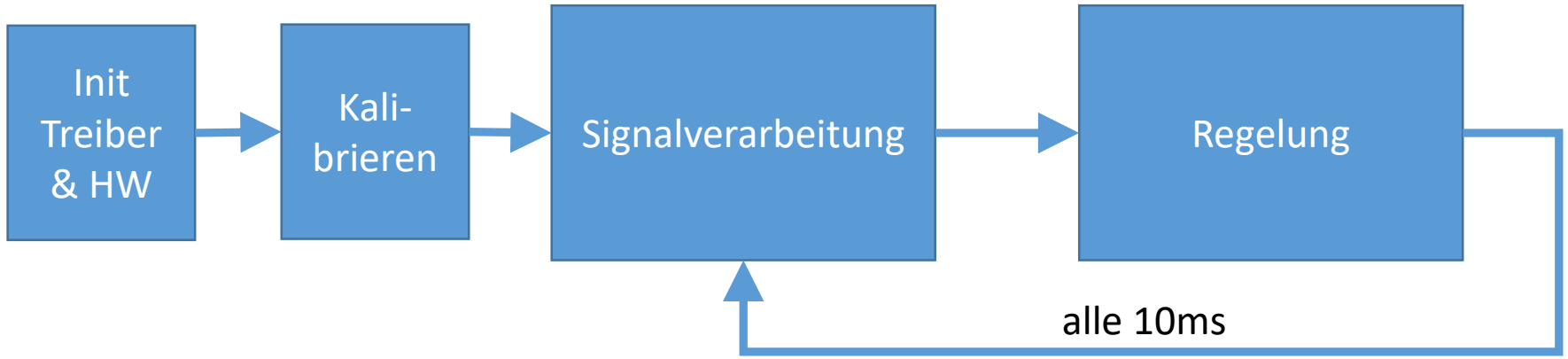
```
// Initialize motor
communication packages
void Motors_Init();

// 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;
```



EMQ Framework

Überblick:

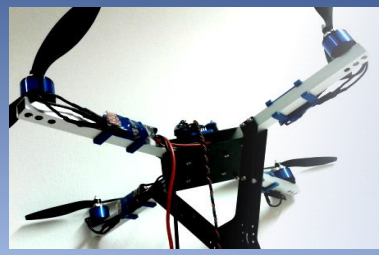


Signalverarbeitung:

- Auslesen der Sensoren
- Konditionieren der Werte (Verarbeitung)
- Datenfusion (z.B. Kalman Filter)

Regelung

- Implementierung des Reglers (PID/PD)
- Ansteuerung der Motoren mit neuen Stellwerten
- Wichtig: Die **Samplezeit** muss stets genau eingehalten werden



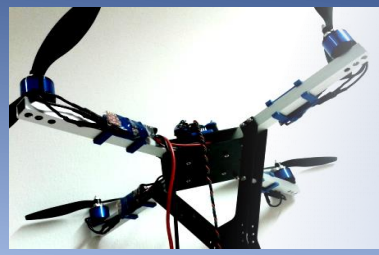
Sicherheitsbelehrung

Sicherheitsbelehrung:

- Vor dem Start der Motoren stets prüfen, dass alles festsitzt:
  Dies betrifft alle Schrauben, Motoren, Ausleger, Y-Teile, Rändelschrauben, etc.
- NUR mit Genehmigung des Betreuers die Motoren anschließen und starten.
- Abstand halten. Nachbarn warnen. Vorsicht walten lassen. Ausknopf parat (Finger drüber)!

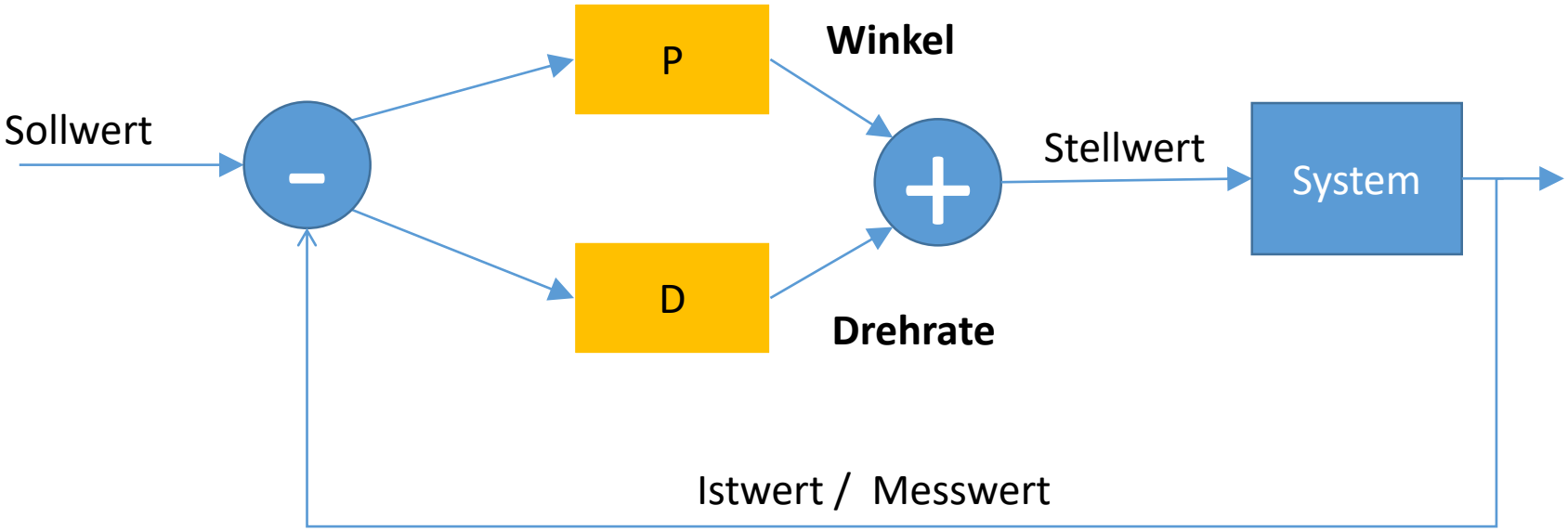
Not-Aus:

- Spannungsquelle ausschalten bzw. trennen
- I2C Kabel trennen (herausziehen)
- MCU ausschalten

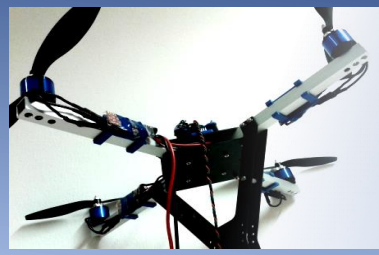


Theorie zur Regelung

Mit einem PD-Regler und einem Gyrosensor lässt sich das System regeln. Als Regelgröße dient der Neigungswinkel.



Problem: 2 Motoren / Stellwerte , 1 Messwert (Winkel)
Lösung: Vereinfach durch Regelung der Differenz der Stellwerte beider Motoren



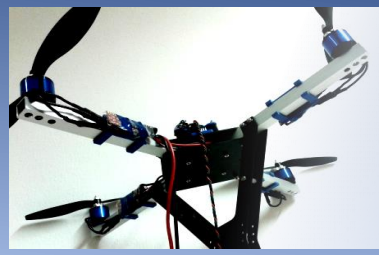
Aufgaben

Notwendige Hardware:

- EMQ3000
- Mini USB Kabel für Strom und zum Flashen
- Mini USB Kabel für Kommunikation: RS232 auf PC (RS232 oder USB)
- QCS im 1DOF Attitude Control Modus

Notwendige Software:

- AVR Studio 32 installiert (mit Tool Chain und FLIP Treiber)
- EMQ Framework (Code)
- Dokumente:
  - EMQ_Framework.pdf
- Qt SDK Version 4.8.1 oder 4.7.4.
- Quatplay Qt Framework (Code)



Aufgaben

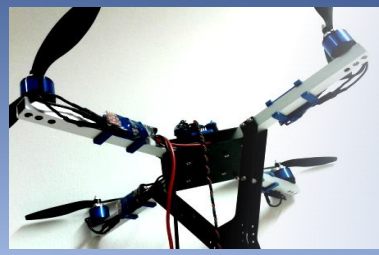
Aufgabe 1a:
Schreibe ein Programm, dass die Motorregler anspricht und die Motoren steuert. Die Motoren sollen per USART Kommando gestartet und gestoppt werden.

Das Programm soll nacheinander jeden einzelnen Motor von Stellwert 0 auf Stellwert 50 hochdrehen lassen. D.h. als erstes startet Motor 0 mit dem Stellwert 1 und inkrementiert dann in ca. 5s auf 50. Wenn Motor 0 den Wert 50 erreicht hat, stoppt Motor 0 beim Wert 0 und es folgt Motor 1, usw.

Wenn das Kommando zum Ausstellen der Motoren erfolgt, dann haben alle Motoren sofort aus zu sein (innerhalb von 10ms). Bei erneutem Starten soll das Programm fortfahren oder neustarten.

Aufgabe 1b:
Die vier aktuell gültigen Sollwerte der Motoren sollen zur Debug-Hilfe optional angezeigt werden können. Entweder per Konsole oder (besser aber aufwändiger) im Telemetriedisplay (Extra Graph).

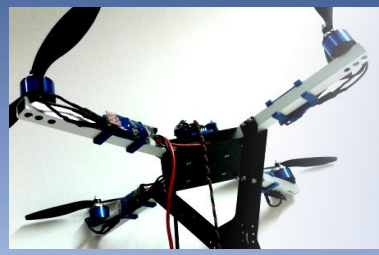
Hinweis: Bevor du das Programm ausprobierst (die Motoren startest), melde dich beim Betreuer und stell sicher, dass niemand zu Schaden kommen kann (Beachte Sicherheitseinweisung).



Aufgaben

Aufgabe 2:

Implementiere einen PD-Regler zur Lageregelung, der die Wippe in der Waagerechten hält. Stelle die Reglerparameter empirisch durch Ausprobieren ein. Die Aufgabe ist gelöst, wenn eine Regelung zu erkennen ist, die reproduzierbar das System für kurze Zeit in der Waagerechten hält und auch kleine Störungen kompensiert. Eine perfekte Lösung ist mit diesem Verfahren nicht zu erwarten (siehe nächste Seite)! Verwende zunächst **nur** den Gyrometer.



Aufgaben

Hinweise zur Regler-Dimensionierung:

Folgende Parameter/Vorgaben soll dir helfen: Nur Richtwerte als Größenordnungshilfe, ohne Sample-Time oder Skalierung:

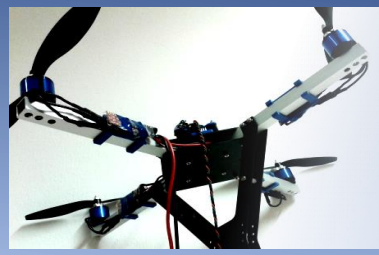
Sample-Time:	10ms		
Bias-Samples	1000 bis 2000		
P	2	| 4	
D	0.4	| 0.8	(bremst P ab)
I (optional)	0.01	| 0.02	(vs. stationäre Regelabweichung)

Vorgehen bei der Reglerparametrisierung:

Fange mit D = 0 und I = 0 an und erhöhe P solange, bis das System „regelt“. Wenn es instabil wird, ist P bereits zu hoch. Wenn das System den Sollwert nie erreicht, ist P zu niedrig. Im Normalfall sollte das System oszillieren, d.h. hin und her schlagen. Als Nächstes erhöhe D um ein Überschwingen abzudämpfen. D sollte der Bewegung (Änderung) entgegenwirken.

Idee:

Der P-Anteil gibt an, wie schnell/heftig auf einen Fehler reagiert wird. Mit dem D-Anteil kann dann das System gebremst werden (Gegenregelung), wenn z.B. ein Fehler auftritt aber das System bereits in Bewegung ist.



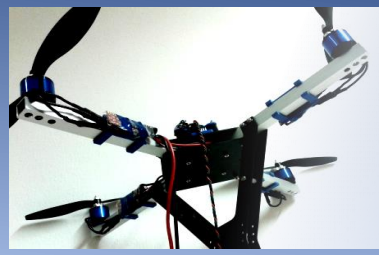
Aufgaben

Hinweise zur Regler-Dimensionierung:

Ein stabiles System zu finden ist ein komplexes Problem. Wenn die Sensorwerte bereits schlecht sind, z.B. aufgrund von Vibrationen springen, dann versagt auch ein ansonsten guter Regler. Deshalb empfiehlt es sich mit dem Gyrometer erstmal nur den Regler einzustellen, da Vibrationen direkt auf das Accelerometer durchschlagen und es sehr viel schwerer ist, gleichzeitig sowohl den KF (Kalman Filter) als auch den Regler richtig zu parametrisieren. D.h. verwendet für den Regler erst mal nur die Messwerte des Winkels, die durch Integration der Gyrowerte gewonnen werden. Später kann dann mit dem KF ein Driften behoben werden. Ebenso darf die Sampletime nicht (stark) variieren, d.h. diese sollte unbedingt eingehalten werden.

Wenn das System instabil ist (oszilliert), liegt dies, wenn der P-Anteil nicht allzu riesig gewählt wurde, oft an einem falschen D-Anteil. Wenn der D-Anteil in die falsche Richtung regelt oder zu hoch/klein ist, wird das System nicht stabil. Zunächst muss das Vorzeichen des D-Anteils stimmen, d.h. er muss einer Bewegung (Winkeländerung) entgegenwirken (dämpfen). Dann ist D so zu wählen, dass es den P-Anteil dämpft, aber nicht so stark, dass es selbst wieder zum Aufschaukeln anreget.

Ein zu hoher I-Anteil führt auch zu Instabilität durch Aufschwingen. Der I-Anteil dient der stationären Regelabweichung und sollte mit Bedacht (am Ende) dazugegeben werden. Stabilität kann ohne I-Anteil erreicht werden. Somit sollte der I-Anteil zunächst immer auf 0 gelassen werden.



Aufgaben

Aufgabe 3:
Verwende nun die Datenfusion des Kalman Filters aus Gyroskope und Accelerometer. Parametrisiere den Kalman Filter so, dass Vibrationen nicht die Orientierung verfälschen. Das System sollte nun nicht mehr Driften. In diesem Zusammenhang ist auch ein schwacher I-Anteil dem Regler hinzuzufügen, damit das System die stationäre Regelabweichung überwindet.

Hinweis:
Aufgabe 3 ist optional. Es sollte vorab die Übung zum Kalman Filter bearbeitet werden.