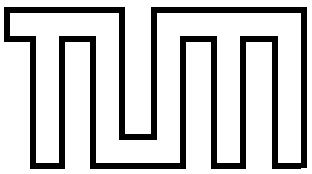


Technische Universität München

Fakultät für Informatik
Forschungs- und Lehreinheit Informatik VI

Übung zur Vorlesung Echtzeitsysteme

Aufgabe 3

Philipp Heise
heise@in.tum.de

Steffen Wittmeier
steffen.wittmeier@in.tum.de

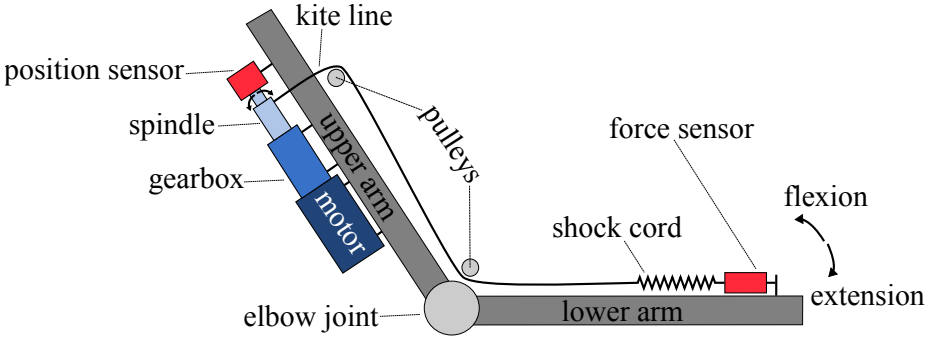
Christoph Staub
staub@in.tum.de

Michael Jäntsch
michael.jaentsch@in.tum.de

Wintersemester 2012/13

ECCEROBOT

Beim Projekt "Embodied Cognition in a Compliantly Engineered Robot" (ECCEROBOT) ging es um die Entwicklung von humanoiden Robotern, bei denen nicht nur die äußere Form des Menschen nachgebildet wurden, sondern auch der Bewegungsapparat. Dies bedeutet, dass ein Skelett entwickelt wurde, dass durch künstliche Muskeln bewegt wird, wobei ein Einzelmuskel aus jeweils einem DC-Elektromotor besteht, der eine Sehne (*kite line*) aufwickelt. Jeder dieser Muskeln ist des Weiteren mit einem elastischen Element (*shock cord*) versehen, um die Elastizität des menschlichen Muskels nachzubilden.



Neben dem Konstruktion und Herstellung war ein weiteres Projektziel die Regelung des Systems. Die Ganzkörperregelung ist jedoch aufgrund der Kopplungen zwischen den Muskeln und Gelenken ein extrem komplexes Thema. Daher wird im weiteren lediglich die Regelung eines einzelnen Muskels durch eine lokale Regelschleife betrachtet. Hier können unterschiedliche Regelungsmodi – Motorposition, Strom, und Kraft – implementiert werden. Für diesen Zweck wurde eine Platine entwickelt, die in der Lage ist jeweils zwei Muskeln anzusteuern. Das Herzstück bildet ein STMicroelectronics Microcontroller auf Basis eines ARM Prozessors. Das Datenblatt zu diesem Controller vom Typ STM32F103 findet man unter:
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/CD00171190.pdf

Neben digitalen Ein-/Ausgängen und analogen Eingängen, sind auch diverse Timer mit PWM Funktionalität verfügbar.

Das Ziel dieser Aufgabe ist auf der genannten Platine einen Regler für die Motorposition zu entwickeln der per CAN-Bus Referenzwerte annimmt und den Motor entsprechend ausregelt. Hierfür wird ein Framework für die Microcontroller Firmware bereitgestellt, in dem ein Großteil der nötigen Hardwareabstraktionen und auch die Kommunikation mit dem PC bereits implementiert ist. Im Folgenden sollen lediglich:

- das Encoder-Interface,
- die Erzeugung des PWM Signals und
- der Motorpositions-Regler

implementiert werden.

Die Mikrocontrollerfunktionalitäten, wie GPIOs, ADCs, Timer, etc., können direkt über die entsprechenden Register angesprochen werden. Allerdings gibt es für den STM32F103 eine Firmware Library, die diese Funktionen bereits abstrahiert und die hier verwendet werden soll. Eine Dokumentation ist an das Aufgabenblatt angehängt.

Build Environment

Um die Programmsourcen zu kompilieren, wird eine cross-toolchain für den ARM Prozessor benötigt. Unter Linux kann hier die toolchain von Codesourcery verwendet werden. Unter Ubuntu können alle nötigen Pakete mit dem mitgelieferten Installationsskript installiert werden. Im Detail handelt es sich um CMake und die genannte toolchain. Bei CMake – Cross Platform Make – handelt es sich um ein Werkzeug mit dem ganze Projekte auf unterschiedlichen Platformen kompiliert werden können. Unter Linux werden hierzu Makefiles erzeugt, wobei die Übersetzung *out of source* erfolgt, d.h. die beim kompilieren erzeugten Dateien werden außerhalb der Quellcode Struktur abgelegt und können so jederzeit gelöscht und neu generiert werden. Hierfür muss ein zusätzlicher Ordner (z.B. in der Baumstruktur neben dem Source Ordner) angelegt werden und anschließend in diesem Ordner der folgende Befehl ausgeführt werden:

```
# cmake ../src
```

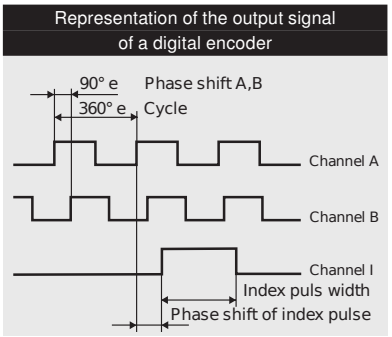
CMake ist außerdem in der Lage *Eclipse* Projekte zu erzeugen:

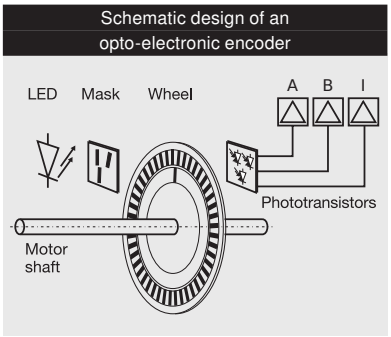
```
# cmake -G"Eclipse CDT4 - Unix Makefiles" ../src
```

Aufgabe 3: Muskelregelung

Aufgabe: Encoder Interface

Implementieren Sie das vorgegebene Encoder-Interface in `src/hal/Encoder.cpp`. Hierfür wird der Pegel der zwei Encoder Kanäle (Channel A, Channel B) überprüft und dementsprechend entweder einen Zähler inkrementiert oder dekrementiert. Die Position des Encoders soll dabei jederzeit durch `getAngPosition()` abgerufen werden können.





Hinweise:

Bei der Implementierung eines Encoder-Interfaces ist es möglich entweder hardwareinterruptgesteuert oder timergesteuert zu arbeiten. Im ersten Fall wird jede steigende Flanke auf Kanal A einen Interrupt auslösen und auf Basis des Pegels von Kanal B entschieden in welche Richtung gezählt werden muss. Alternativ kann durch einen Timer regelmäßig der Pegel beider Kanäle überprüft werden und entsprechend gezählt werden. In dieser Aufgabe sollte Zweiteres Implementiert werden, da diese Möglichkeit deutlich weniger anfällig gegenüber prellenden Signalen und dadurch hervorgerufene doppelt gezählte Flanken ist. Eine gut Abfragefrequenz ist hier 260 kHz.

Gehen Sie wie folgt vor:

1. Richten Sie im Konstruktor der Klasse Encoder einen Timer ein, der mit 260 kHz einen Interrupt generiert.
   - `NVIC_INIT`
   - `TIM_TimeBaseInit`
   - `TIM_CMD`
2. Implementieren Sie die ISR, so dass ein Counter entsprechend der Drehrichtung inkrementiert oder dekrementiert wird, wobei Kanal A auf GPIOA_9 und Kanal B auf GPIOA_10 liegt. Hierbei ist zu beachten, dass der Timer den Interrupt mehrfach auslösen kann, ohne dass sich die Pegel ändern. Der Status eines GPIO pins kann mit Hilfe der Funktion `GPIO_ReadInputDataBit` abgefragt werden. Es muss also eine

Variable eingeführt werden, die den Zustand speichert und nur bei einer Änderung des Zustandes die Encoderticks gezählt werden.

3. Implementieren Sie die Funktion `getAngPosition()`, so dass der Wert des Counters zurückgegeben wird.

Aufgabe: PWM

Der STM32F hat die Möglichkeit einen Timer direkt zu verwenden um ein PWM Signal zu erzeugen. Die Hardware ist so in der Lage das Signal zu erzeugen ohne den Prozessor zu belasten. Implementieren Sie das Motor Interface in `src/hal/Motor.cpp` so dass der Timer im Hintergrund läuft und über `setVoltage()` eine Spannung (als Anteil an der Maximalspannung) gesetzt werden kann. Die Drehrichtung kann über einen weiteren Pin gesetzt werden. Dieser schaltet die H-Brücke vom Vorwärts- in den Rückwärtsbetrieb.

Hinweise:

Der Motor wird duch ein PWM Signal auf dem Pin GPIOB_10 angesteuert, wobei das Signal durch Timer2, Output Compare Register 3 erzeugt wird. Die Richtung wird durch den Pin GPIOB_8 vorgegeben.

Gehen Sie wie folgt vor:

1. Richten Sie im Konstruktor der Klasse `Motor` den Timer mit einer Reload Rate von 50 kHz ein und setzen Sie den Output Compare Kanal 3 (OC3) in den PWM Modus.
   - `TIM_TimeBaseInit`
   - `TIM_OC3PreloadConfig`
2. Implementieren Sie die Funktionen `setVoltage()` und `getVoltage()`, wobei eine negative Spannung über den Pegel des Drehrichtungspins (GPIOB_8) signalisiert wird. Der Duty Cycle wird als Wert im Output Compare Register 3 dargestellt.

Aufgabe: Regler

Die Klasse `Muscle` abstrahiert die Kommunikation der zentralen Steuereinheit mit dem Microcontroller über den CAN Bus und ermöglicht die Ansteuerung des Motors über die verschiedenen Regelungsmodi. Implementieren Sie einen proportionalen Motorpositionsregler (P-Regler) , der zunächst die momentane Motorposition φ ausliest, mit der Referenzmotorposition φ_{ref} vergleicht und anschließend eine Spannung berechnet und in die Variable `voltage` schreibt.

$$\text{voltage} = P \cdot (\varphi_{\text{ref}} - \varphi) \tag{1}$$

Beachten Sie hierbei, dass die Referenzmotorposition in Grad am Getriebeausgangsschaft angegeben wird, der Encoder jedoch lediglich die motorseitigen Ticks zählt. Die verwendete Motor/Getriebe/Encoder-Kombination gibt hierbei die Umrechnung vor, wobei der Encoder mit 256 Ticks pro Umdrehung auf der schnelldrehenden Motorwelle sitzt. Die Drehzahl des Motors wird durch das Getriebe mit einem Übersetzungsverhältnis von 51 : 1 umgesetzt. Die notwendige Encoderconversion c_{enc} ist wie folgt definiert.

$$\varphi[\text{deg}] = c_{\text{enc}} \cdot \text{counter}[\text{ticks}] \tag{2}$$

Hinweise:

1. Implementieren Sie in `src/Muscle.cpp` in der Funktion `stateOnExecute()` den Motorregler. Verwenden sie die Methode `Muscle::getMotorPosition()` für die momentane Motorposition und die Membervariable `Muscle::pMotorPosition` für den proportionalen Regelanteil (P).
2. Sowohl `Muscle::pMotorPosition`, als auch `Muscle::encoderConversion` werden in der Initialisierungsphase per CAN Kommunikation von der zentralen Steuereinheit übertragen. Hierfür wird ein *XML* geparst. Passen Sie den Wert für die `<encoderConversion>` in `src/xml/test.xml` so an, dass er zu der hier verwendeten Motor/Getriebe/Encoder-Kombination passt.