

Echtzeitsysteme

Physikalisches System ↔ Kontrollierendes Rechensystem

Peter Ulbrich

Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

<https://www4.cs.fau.de>

28. Oktober 2016



Fragestellungen



Echtzeitbetrieb bedeutet **Rechtzeitigkeit** (vgl. Folie II/10 ff)

- Die funktionale Korrektheit ist nicht ausreichend
- Zeitliche (temporale) Korrektheit!



Terminvorgaben sind **anwendungsabhängig**

- Komplexität des physikalischen Systems (vgl. Folie II/18 ff)
- Bestimmt durch die Kopplung an die (reale) Umwelt



Geschwindigkeit ist keine Garantie

- Komplexität des Echtzeitrechensystems
- Entscheidend ist das tatsächliche Laufzeitverhalten



Woher kommen die zeitlichen Vorgaben und Eigenschaften?

- Wo sind die Berührungspunkte mit dem **physikalischen System**?
- Welche Rolle spielt das **Echtzeitrechensystem**?



Exkurs: Funktionale und nicht-funktionale Eigenschaften

■ Funktionale Eigenschaften

- Werden **direkt implementiert**

Eine Funktion

```
uint16_t regelschritt(uint8_t sensorwert)
```

■ Nicht-funktionale Eigenschaften

- Beispielsweise **Energie**, **Speicherverbrauch**, **Laufzeitverhalten**
- Lassen sich **nicht direkt implementieren**
- Sind **querschneidend** ~ erst im konkreten Kontext bestimmt



Zeit aus Sicht des Softwareengineering **nicht-funktional**

- Führt häufig zu Verwirrung im Kontext von Echtzeitsystemen
- Die **rechtzeitige** Auslösung des Airbags ist funktional?!



Es kommt auf die Betrachtungsebene an!



Gliederung

1 Physikalisches System und Echtzeitanwendung

- Kontrolliertes Objekt
- Zusammenspiel

2 Echtzeitrechensystem

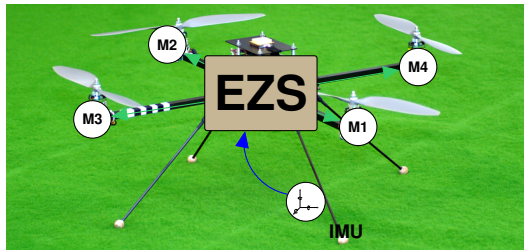
- Grundlagen: Programmunterbrechungen
- Ausnahmebehandlung
- Zustandssicherung
- Ableitung des Zeitbedarfs

3 Zusammenfassung



Aufbau des Demonstrators

Eine elementare Kontrollschleife: Die Fluglageregelung



⚠ Quadrokopter sind **inhärent instabil** $\leadsto$ ständige, aktive Kontrolle

■ **Aufgabe** des Echtzeitsystems: **Fluglageregelung** (Stabilisierung)

- Bewegung im Raum bestimmen (engl. *inertial measurement unit*)
- Vorgabe der Motor- und damit der Rotordrehzahl

👉 Physikalisches **Objekt**, Echtzeit-**Anwendung** und -**Rechnersystem**



Kontrolle und Regelung des physikalischen Objekts

Übertragungsfunktion und Antwortfunktion

■ **Lage im Raum** wird durch **Änderung der Rotordrehzahl** des Quadropters beeinflusst, bis **Gleichgewicht** zwischen Ist- und Sollzustand

⚠ **Wie lange dauert es** bis zum Gleichgewicht?

- Gewicht, Leistungsfähigkeit der Motoren, Bauart der Rotorblätter, ...
- **Objektdynamik** und -**physik**

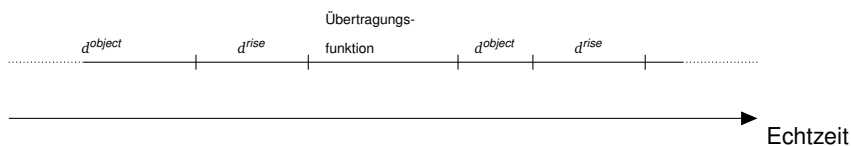
👉 Dies ist die Welt der **Steuerungs-** und **Regelungsanwendungen**

- Regelungstechnische Abstraktion des Quadropters:
Dynamisches System welches **Eingangs-** in **Ausgangssignale** überführt
- Ziel ist die mathematische Beschreibung des Systemverhaltens mittels einer **Übertragungsfunktion** (engl. *transfer function*)
- Reaktion kann errechnet und gezielt beeinflusst werden



Zeitparameter des physikalischen Objekts

Die Physik kommt ins Spiel ... [3, Kapitel 1.3]



👉 **Zeitverzögerung** (d , **delay**) des Quadropters:

d^{object} Zeitdauer bis zum Beginn der Lageänderung

- Hervorgerufen durch die (initiale) Trägheit des Objektes
- **Prozessverzögerung** (engl. *process/object delay*)

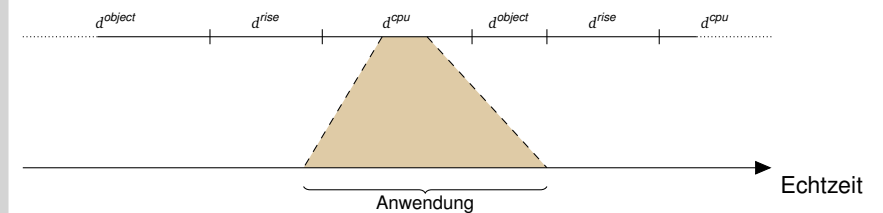
d^{rise} Zeitdauer bis zum (erneuten) Gleichgewicht

- Allgemein: Erreichen der Zielgröße (typ. 66 % bzw. 90 %)
- Bestimmt durch die Fähigkeit der Aktorik $\rightarrow$ Einschwingverhalten
- **Anregel-/ Anlaufzeit** (engl. *rise time*) einer Sprungantwort



Zeitparameter des Rechnersystems

Berechnung der Übertragungsfunktion: Alles braucht seine Zeit



👉 **Zeitverzögerung** des Rechnersystems

→ **Auswertung**: Abweichung (Soll/Ist) und Übertragungsfunktion (Regler)

⚠ Das Rechnersystem benötigt Zeit für die Berechnung

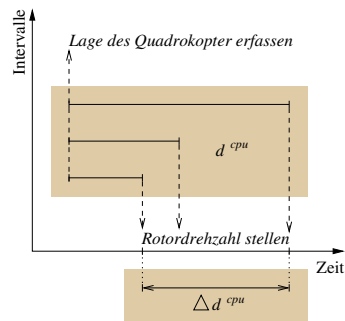
d^{cpu} Zeitdauer bis zur Ausgabe des neuen Stellwertes

- **Erfassung** der Umgebung durch Sensoren
- **Berechnung** des Regelalgorithmus
- **Kontrollieren** des Objekts durch Aktorik



Schwankungen (engl. *jitter*) in den Zeitparametern

Der komplexe Einfluss des Echtzeitrechnungssystems



d^{cpu} Auch bei konstantem Rechenaufwand zur Stellwertbestimmung variabel

- Verdrängende Einplanung
- Überlappende Ein-/Ausgabe
- Programmunterbrechungen
- Busüberlastung, DMA

Δd^{cpu} Fügt Unschärfe zum Zeitpunkt der Lagebestimmung hinzu

- Bewirkt zusätzlichen Fehler
- Beeinträchtigt die Dienstgüte

⚠ **Unbekannte variable Verzögerungen** (engl. *jitter*) schwer kompensierbar

- Bekannte konstante Verzögerungen schon $\leadsto d^{dead}$
- Randbedingung: $\Delta d^{cpu} \ll d^{cpu}$



Physikalisches Objekt $\leftrightarrow$ Echtzeitrechnungssystem



Zeitverzögerung des Regelkreises: **Totzeit** (engl. *dead time*)

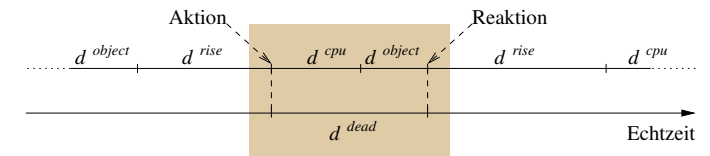
- Entsteht aus dem Zusammenspiel zwischen **Objekt** und **Rechnungssystem**

d^{dead}

Zeitintervall zwischen Berechnungsbeginn und Wahrnehmung einer Reaktion nach erfolgter Steuerung

- setzt sich zusammen aus d^{cpu} und d^{object} :

- 1 Implementierung des kontrollierenden Rechnersystems
- 2 Dynamik des kontrollierten Objektes



Auswirkung Güte und **Stabilität** der Regelung

- Insbesondere bei hoher Varianz von $\Delta d^{dead} \rightarrow$ Jitter
- $\rightarrow$ Relative Ungewissheit über die erzielte Wirkung



Gliederung

1 Physikalisches System und Echtzeitanwendung

- Kontrolliertes Objekt
- Zusammenspiel

2 Echtzeitrechnungssystem

- Grundlagen: Programmunterbrechungen
- Ausnahmebehandlung
- Zustandssicherung
- Ableitung des Zeitbedarfs

3 Zusammenfassung



Zeitliches Verhalten von Echtzeitanwendungen

Allgemeine Kosten des Echtzeitrechnungssystems

■ Welche **Elemente** müssen betrachtet werden?

- Beschränkung auf die Echtzeitanwendung (Regelung)?
- Vernachlässigung des Echtzeitbetriebssystems?
- Wie stark hängt dies vom verwendeten Prozessor ab?

■ Auf **welcher Ebene** muss die Betrachtung durchgeführt werden?

- Genügt es eine hohe Abstraktionsebene heranzuziehen?
- Wo entscheidet sich das zeitliche Ablaufverhalten?



Verwaltungsgemeinkosten der Laufzeitumgebung



Exemplarische Illustration anhand von **Programmunterbrechungen**



Unterbrechungsarten

- ☞ Zwei Arten von Programmunterbrechungen:
- synchron** die „Falle“ (engl. *trap*)
 - asynchron** die „Unterbrechung“ (engl. *interrupt*)

■ Unterschiede ergeben sich hinsichtlich:

- Quelle
- Synchronität
- Vorhersagbarkeit
- Reproduzierbarkeit

⚠ Behandlung ist **zwingend** und grundsätzlich **prozessorabhängig**

Wiederholung/Vertiefung empfohlen...

Unterbrechungen siehe auch Vorlesung „Betriebssysteme“ [4, Kapitel 2-3]



Synchrone Programmunterbrechung (engl. *trap*)

- ☞ Ursachen einer **synchronen Programmunterbrechung**:
- Unbekannter Befehl, falsche Adressierungsart oder Rechenoperation
 - Systemaufruf, Adressraumverletzung, unbekanntes Gerät

Trap → **synchron, vorhersagbar, reproduzierbar**

- Abhängig vom Arbeitszustand des laufenden Programms:
 - Unverändertes Programm, mit den selben Eingabedaten versorgt
 - Auf ein und dem selben Prozessor zur Ausführung gebracht
- Unterbrechungsstelle im Programm ist vorhersehbar



Programmunterbrechung/-verzögerung ist **deterministisch**



Asynchrone Programmunterbrechung (engl. *interrupt*)

- ☞ Ursachen einer **asynchronen Programmunterbrechung**:
- Signalisierung „externer“ Ereignisse
 - Beendigung einer DMA- bzw. E/A-Operation

Interrupt → **asynchron, unvorhersagbar, nicht reproduzierbar**

- Unabhängig vom Arbeitszustand des laufenden Programms:
 - Hervorgerufen durch einen „externen Prozess“ (z.B. ein Gerät)
 - Signalisierung eines Ereignis
- Unterbrechungsstelle im Programm ist nicht vorhersehbar



Programmunterbrechung/-verzögerung ist **nicht deterministisch**



Ausnahmesituationen (engl. *exception*) – Beispiele

- **Ereignisse**, oftmals unerwünscht aber nicht immer eintretend:
 - Signale von der Peripherie (z.B. E/A, Zeitgeber oder „Wachhund“)
 - Wechsel der Schutzdomäne (z.B. Systemaufruf)
 - Programmierfehler (z.B. ungültige Adresse)
 - unerfüllbare Speicheranforderung (z.B. bei Rekursion)
 - Einlagerung auf Anforderung (z.B. beim Seitenfehler)
 - Warnsignale von der Hardware (z.B. Energiemangel)
- **Ereignisbehandlung**, die problemspezifisch zu gewährleisten ist:
 - Ausnahme während der „normalen“ Programmausführung



Ausnahmebehandlung (engl. *exception handling*)

Abrupter Zustandswechsel

⚠ Programmunterbrechungen implizieren **nicht-lokale Sprünge**:

- vom $\left\{ \begin{array}{l} \text{unterbrochenen} \\ \text{behandelnden} \end{array} \right\}$ zum $\left\{ \begin{array}{l} \text{behandelnden} \\ \text{unterbrochenen} \end{array} \right\}$ Programm

👁 Sprünge (und Rückkehr davon) ziehen **Kontextwechsel** nach sich:

- Maßnahmen zur Zustandssicherung/-wiederherstellung erforderlich
- Mechanismen dazu liefern das behandelnde Programm selbst
 - bzw. eine tiefer liegende Systemebene (Betriebssystem, CPU)

⚠ **Prozessorstatus** unterbrochener Programme muss invariant sein



Zustandssicherung

Prozessorstatus invariant halten

Hardware (CPU) sichert einen Zustand minimaler Größe¹

- Statusregister (SR)
- Befehlszeiger (engl. *program counter*, PC)

Software (Betriebssystem/Kompilierer) sichert den Rest

- alle $\left\{ \begin{array}{l} \text{dann noch ungesicherten} \\ \text{flüchtigen} \\ \text{im weiteren Verlauf verwendeten} \end{array} \right\}$ CPU-Register



Abhängig von der CPU werden wenige bis sehr viele Daten(bytes) bewegt
~ **Zeitbedarf!**

¹Möglicherweise aber auch den kompletten Registersatz.



Prozessorstatus sichern und wiederherstellen

Unabhängigkeit von der Sprachebene der Behandlungsprozedur

- Sichern aller ungesicherten Register auf Befehlssatz-Ebene:

Zeile	x86	m68k
1:	train:	train:
2:	pushal	moveml d0-d7/a0-a6,a7@-
3:	call handler	jsr handler
4:	popal	moveml a7@+,d0-d7/a0-a6
5:	iret	rte

- **train (trap/interrupt):**

- Arbeitsregisterinhalte im RAM sichern (2) und wiederherstellen (4)
- Unterbrechungsbehandlung durchführen (3)
- Ausführung des unterbrochenen Programms wieder aufnehmen (5)



Prozessorstatus sichern und wiederherstellen (Forts.)

Abhängigkeit von den Eigenschaften des Kompilierers

- Kontextsicherung durch Instrumentierung des Compilers:

```
gcc
void __attribute__((interrupt)) train () {
    handler();
}
```

- **__attribute__((interrupt))**

- Generierung der speziellen Maschinenbefehle durch den **Kompilierer**
 - Sicherung/Wiederherstellung der Arbeitsregisterinhalte
 - Wiederaufnahme der Programmausführung
- Nicht jeder „Prozessor“ (für C/C++) implementiert dieses Attribut



Verwaltungsgemeinkosten des schlimmsten Falls

(engl. *worst-case administrative overhead, WCAO*)

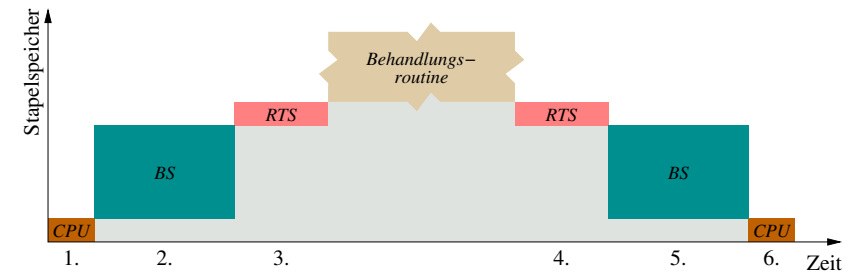
- **Latenz** bis zum Start der Unterbrechungsbehandlung:
 - 1 Annahme der Unterbrechung durch die Hardware
 - 2 Sicherung der Inhalte der (flüchtigen) CPU-Register
 - 3 Aufbau des Aktivierungsblocks der Behandlungsprozedur
- **Latenz** bis zur Fortführung des unterbrochenen Programms:
 - 4 Abbau des Aktivierungsblocks der Behandlungsprozedur
 - 5 Wiederherstellung der Inhalte der (flüchtigen) CPU-Register
 - 6 Beendigung der Unterbrechung

⚠ **Zeitpunkte** und **Häufigkeit** der Gemeinkosten sind i. A. unbestimmbar



Verwaltungsgemeinkosten des schlimmsten Falls (Forts.)

Speicherplatz vs. Laufzeit



Werte mit fester oberer Schranke sind gefordert:

- Prozessor respektive Rechensystem (z.B. ADCs)
- Echtzeitbetriebssystem (engl. *real-time operating system*, RTOS)
- Echtzeitanwendung (Behandlungsroutine)



Zeitbedarf im Echtzeitrechensystem

Welche Komponenten benötigen wie viel Zeit?

- ⚠ Häufig ist eine eigenständige Beurteilung des Zeitbedarfs nicht möglich, Herstellerangaben ermöglichen die Abschätzung des **schlimmsten Falls**.

- Beispiel Quadrokopter:

d_{imu}^{imu} Gyroskop ITG-3200 – Abtastrate: 4 Hz – 8 kHz [1]
 d_{adc}^{adc} Infineon TriCore ADC: 280 ns – 2,5 μ s @ 10 Bit [2]
 d_{irq}^{irq} Infineon TriCore Arbitrierung: 5 - 11 Takte @ 150 MHz [2]
 d_{OS}^{OS} CiAO OS Fadenwechsel: ≤ 219 Takte @ TriCore (50 MHz) [5]

- ⚠ Alleine die **Anwendung** kann (fast) komplett kontrolliert werden.²



²Lässt man zugelieferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.

Abbildung der Fluglageregelung

Ein paar Daumenregeln

- Die Lage des Quadrokopter wird zyklisch abgetastet, um Abweichungen der aktuellen Lage vom Gleichgewicht zu erkennen:

$d^{control}$ Zeitabstand (konstant) zwischen zwei Regelschritten
 ■ Faustregel: $d^{sample} < (d^{rise}/10)$
 → Quasi-kontinuierliches Verhalten des diskreten Systems

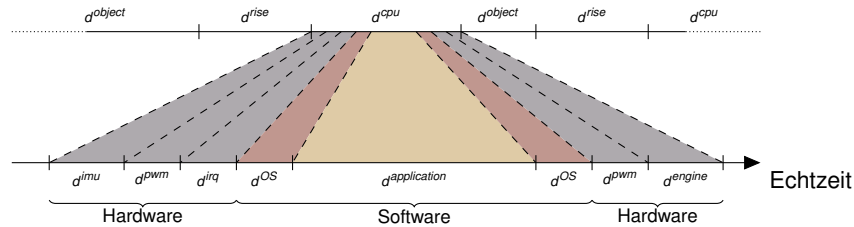
f^{sample} Abtastfrequenz, entspricht $1/d^{sample}$
 ■ Analoge auf digitale Werte abbilden $\leadsto$ A/D-Wandlung
 → Nyquist-Shannon-Abtasttheorem



The Big Picture – Rechenzeitbedarf

Aus welchen Komponenten setzt sich d^{cpu} zusammen?

Ein Echtzeitsystem setzt sich aus verschiedenen Hardware, Sensoren, Peripherie-Elementen, Echtzeitbetriebssystem und Softwarekomponenten zusammen.



⚠ Alle Komponenten müssen bedacht werden!

Sensoren/Aktoren Abtaste ($\sim d^{imu}$), Motorleistung ($\sim d^{engine}$)
Mikrocontroller Signalverarbeitung ($\sim d^{pwm}$), IRQ ($\sim d^{irq}$)
Betriebssystem Unterbrechungslatenz, Kontextwechsel ($\sim d^{OS}$)
Anwendung Steuerung, Regelung ($\sim d^{application}$)



Gliederung

- 1 Physikalisches System und Echtzeitanwendung
 - Kontrolliertes Objekt
 - Zusammenspiel
- 2 Echtzeitrechnungssystem
 - Grundlagen: Programmunterbrechungen
 - Ausnahmebehandlung
 - Zustandssicherung
 - Ableitung des Zeitbedarfs
- 3 Zusammenfassung



Resümee

Zusammenspiel Kontrolliertes Objekt ↔ Kontrollierendes Rechensystem

- Die Objektdynamik definiert den zeitlichen Rahmen durch Termine
- Die Echtzeitanwendung muss diese Termine einhalten
- Ihr Anteil am kontrollierenden Rechensystem ist eher gering

Programmunterbrechung in synchroner oder asynchroner Ausprägung

- Beeinflussen den Ablauf der Echtzeitanwendung
- Zustandssicherung, Verwaltungsgemeinkosten des schlimmsten Falls



Literaturverzeichnis

- [1] Inc., I. :
ITG-3200 Product Specification Revision 1.4.
<http://invensense.com/mems/gyro/documents/PS-ITG-3200A.pdf>, 2010. – Data Sheet
- [2] Infineon Technologies AG (Hrsg.):
TC1796 User's Manual (V2.0).
St.-Martin-Str. 53, 81669 München, Germany: Infineon Technologies AG, Jul. 2007
- [3] Kopetz, H. :
Real-Time Systems: Design Principles for Distributed Embedded Applications.
Kluwer Academic Publishers, 1997. – ISBN 0-7923-9894-7
- [4] Lohmann, D. :
Vorlesung: Betriebssysteme, Friedrich-Alexander-Universität Erlangen-Nürnberg.
https://www4.cs.fau.de/Lehre/WS15/V_BS, 2015
- [5] Lohmann, D. ; Hofer, W. ; Schröder-Preikschat, W. ; Streicher, J. ; Spinczyk, O. :
CiAO: An Aspect-Oriented Operating-System Family for Resource-Constrained Embedded Systems.
In: Proceedings of the 2009 USENIX Annual Technical Conference.
Berkeley, CA, USA : USENIX Association, Jun. 2009. – ISBN 978-1-931971-68-3, S. 215-228



EZS – Cheat Sheet

Typographische Konvention

Der erste Index gibt die Aufgabe an (z.B. D_i), der Zweite (optional) bezieht sich auf den Arbeitsauftrag (z.B. $d_{i,j}$). Exponenten zeigen verschiedene Varianten einer Eigenschaft an (z.B. T^{HI} , T^{MED} , T^{LO}). Funktionen beschreiben zeitlich variierende Eigenschaften (z.B. $P(t)$).

Eigenschaften

t (Real-)Zeit
 d Zeitverzögerung (engl. delay)

