

# Echtzeitsysteme

Physikalisches System  $\leftrightarrow$  Kontrollierendes Rechensystem

**Peter Ulbrich**


Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

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

23. Oktober 2017



-  **Echtzeitbetrieb** bedeutet **Rechtzeitigkeit** (vgl. Folie II/10 ff)
- Die funktionale Korrektheit ist nicht ausreichend
- Zeitliche (temporale) Korrektheit!



 **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



 **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

 **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**?



- Funktionale Eigenschaften
  - Werden **direkt implementiert**

### Eine Funktion

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



- 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**  $\rightsquigarrow$  erst im konkreten Kontext bestimmt



- 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**  $\rightsquigarrow$  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?!



- 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**  $\rightsquigarrow$  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!



## 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 Flugregelung

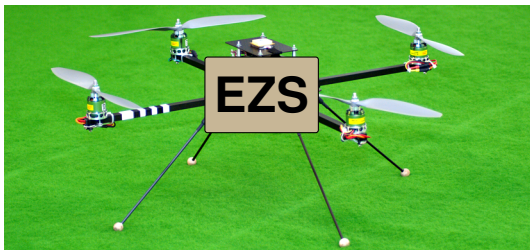


Quadrocopter sind **inhärent instabil**  $\leadsto$  ständige, aktive Kontrolle



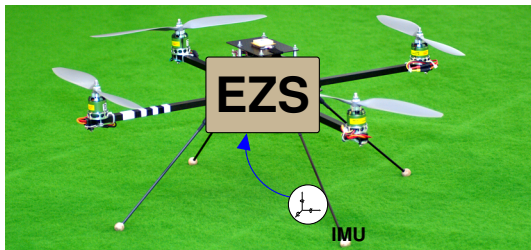
# Aufbau des Demonstrators

Eine elementare Kontrollschleife: Die Fluglageregelung

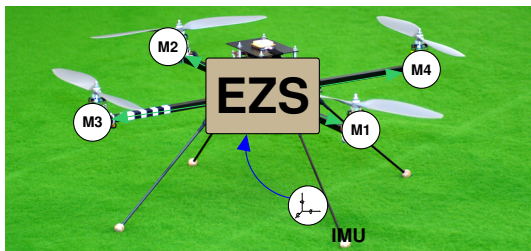


- ⚠ Quadrokopter sind **inhärent instabil**  $\leadsto$  ständige, aktive Kontrolle
- **Aufgabe** des Echtzeitsystems: **Fluglageregelung** (Stabilisierung)





- ⚠ Quadrokopter sind **inhärent instabil**  $\leadsto$  ständige, aktive Kontrolle
- **Aufgabe** des Echtzeitsystems: **Fluglageregelung** (Stabilisierung)
  - Bewegung im Raum bestimmen (engl. *inertial measurement unit*)



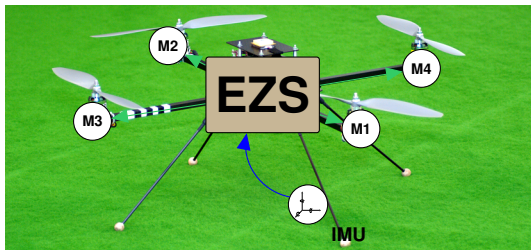
Quadrokopter sind **inhärent instabil**  $\leadsto$  ständige, aktive Kontrolle

- **Aufgabe** des Echtzeitsystems: **Fluglageregelung** (Stabilisierung)
  - Bewegung im Raum bestimmen (engl. *inertial measurement unit*)
  - Vorgabe der Motor- und damit der Rotordrehzahl



# 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 -**Rechensystem**



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



- 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



- 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



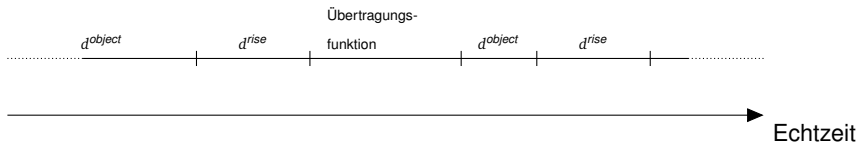
Dies ist die Welt der Steuerungs- und Regelungsanwendungen

- 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*)

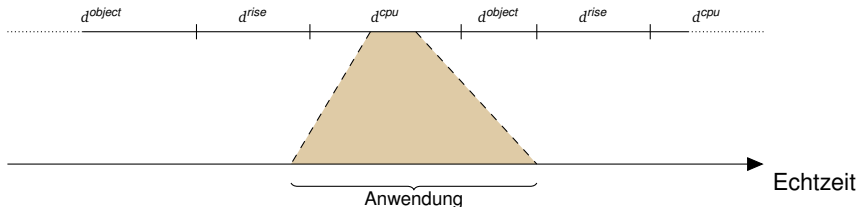
$d^{rise}$  Zeitdauer bis zum (erneuten) Gleichgewicht

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



# Zeitparameter des Rechengystems

Berechnung der Übertragungsfunktion: Alles braucht seine Zeit



## Zeitverzögerung des Rechengystems

- Auswertung: Abweichung (Soll/Ist) und Übertragungsfunktion (Regler)
- ⚠ Das Rechengystem benötigt Zeit für die Berechnung

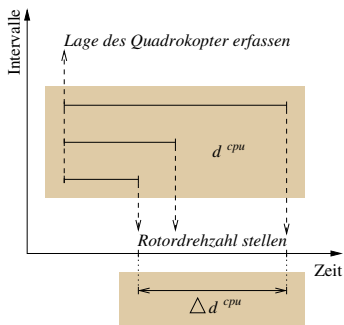
$d^{cpu}$  Zeitdauer bis zur Ausgabe des neuen Stellwertes

- Erfassung der Umgebung durch Sensoren
- Berechnung des Regelungsalgorithmus
- Kontrollieren des Objekts durch Aktorik



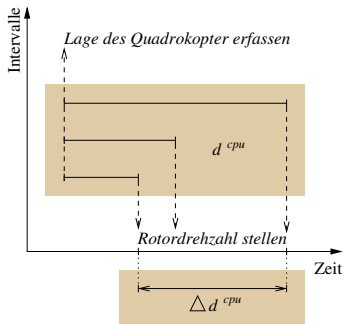
# Schwankungen (engl. *jitter*) in den Zeitparametern

Der komplexe Einfluss des Echtzeitrechensystems



# Schwankungen (engl. *jitter*) in den Zeitparametern

Der komplexe Einfluss des Echtzeitsystems



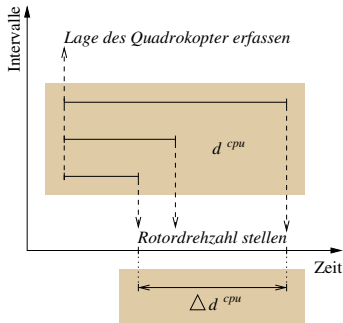
$d^{cpu}$  Auch bei konstantem Rechenaufwand zur Stellwertbestimmung variabel

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



# Schwankungen (engl. *jitter*) in den Zeitparametern

Der komplexe Einfluss des Echtzeitsystems



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

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

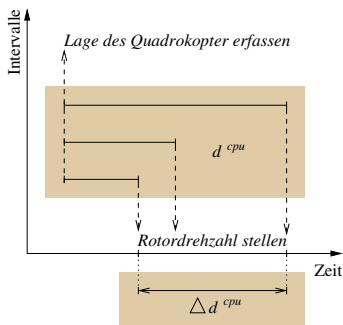
$\Delta d^{cpu}$  Fügt Unschärfe zum Zeitpunkt der Lagebestimmung hinzu

- Bewirkt zusätzlichen Fehler
- Beeinträchtigt die Dienstgüte



# Schwankungen (engl. *jitter*) in den Zeitparametern

Der komplexe Einfluss des Echtzeitsystems



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

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

$\Delta d^{cpu}$  Fügt Unschärfe zum Zeitpunkt der Lagebestimmung hinzu

- Bewirkt zusätzlichen Fehler
- Beeinträchtigt die Dienstgüte



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

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





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

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

$d^{dead}$  Zeitintervall zwischen Berechnungsbeginn und Wahrnehmung einer Reaktion nach erfolgter Steuerung

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

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



# Physikalisches Objekt $\leftrightarrow$ Echtzeitrechnungssystem

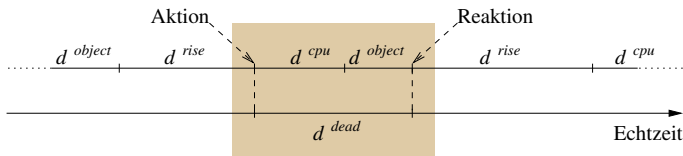
☞ Zeitverzögerung des Regelkreises: **Totzeit** (engl. *dead time*)

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

$d^{dead}$  Zeitintervall zwischen Berechnungsbeginn und Wahrnehmung einer Reaktion nach erfolgter Steuerung

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

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



# Physikalisches Objekt $\leftrightarrow$ Echtzeitrechnungssystem

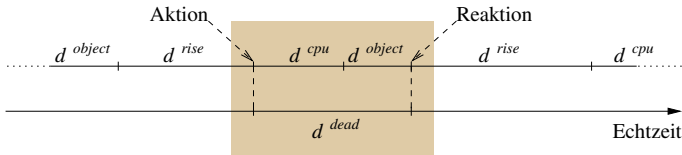
☞ Zeitverzögerung des Regelkreises: **Totzeit** (engl. *dead time*)

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

$d^{dead}$  Zeitintervall zwischen Berechnungsbeginn und Wahrnehmung einer Reaktion nach erfolgter Steuerung

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

- 1 Implementierung des kontrollierenden Rechnensystems
- 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



## 1 Physikalisches System und Echtzeitanwendung

- Kontrolliertes Objekt
- Zusammenspiel

## 2 Echtzeitrechensystem

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

## 3 Zusammenfassung



- Welche Elemente müssen betrachtet werden?
  - Beschränkung auf die Echtzeitanwendung (Regelung)?
  - Vernachlässigung des Echtzeitbetriebssystem?
  - 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?



- Welche Elemente müssen betrachtet werden?
  - Beschränkung auf die Echtzeitanwendung (Regelung)?
  - Vernachlässigung des Echtzeitbetriebssystem?
  - 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



- Welche Elemente müssen betrachtet werden?
  - Beschränkung auf die Echtzeitanwendung (Regelung)?
  - Vernachlässigung des Echtzeitbetriebssystem?
  - 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





Zwei Arten von Programmunterbrechungen:

**synchron** die „Falle“ (engl. *trap*)

**asynchron** die „Unterbrechung“ (engl. *interrupt*)





Zwei Arten von Programmunterbrechungen:

**synchron** die „Falle“ (engl. *trap*)

**asynchron** die „Unterbrechung“ (engl. *interrupt*)

■ Unterschiede ergeben sich hinsichtlich:

- Quelle
- Synchronität
- Vorhersagbarkeit
- Reproduzierbarkeit





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**



# 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]




 Ursachen einer **synchrone Programmunterbrechung**:

- Unbekannter Befehl, falsche Adressierungsart oder Rechenoperation
- Systemaufruf, Adressraumverletzung, unbekanntes Gerät

*Trap*  $\mapsto$  **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
- $\rightarrow$  Unterbrechungsstelle im Programm ist vorhersehbar

 Programmunterbrechung/-verzögerung ist **deterministisch**



 Ursachen einer **asynchronen Programmunterbrechung**:

- Signalisierung „externer“ Ereignisse
- Beendigung einer DMA- bzw. E/A-Operation

*Interrupt*  $\mapsto$  **asynchron, unvorhersagbar, nicht reproduzierbar**

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



Programmunterbrechung/-verzögerung ist **nicht deterministisch**



- **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)



- **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





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





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)





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



Hardware (CPU) sichert einen Zustand minimaler Größe<sup>1</sup>

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

---

<sup>1</sup>Möglicherweise aber auch den kompletten Registersatz.



**Hardware** (CPU) sichert einen Zustand minimaler Größe<sup>1</sup>

- 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

---

<sup>1</sup>Möglicherweise aber auch den kompletten Registersatz.



**Hardware** (CPU) sichert einen Zustand minimaler Größe<sup>1</sup>

- 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!**

---

<sup>1</sup>Möglicherweise aber auch den kompletten Registersatz.



- Sichern aller ungesicherten Register auf Befehlssatz-Ebene:

Zeile

1:  
2:  
3:  
4:  
5:

x86

```
train:  
  pushal  
  call handler  
  popal  
  iret
```



- Sichern aller ungesicherten Register auf Befehlssatz-Ebene:

Zeile

```
1: train:
2:   pushal
3:   call handler
4:   popal
5:   iret
```

x86

m68k

```
train:
  moveml d0-d7/a0-a6,a7@-
  jsr handler
  moveml a7@+,d0-d7/a0-a6
  rte
```

- `train` (trap/interrupt):
  - Arbeitsregisterinhalte im RAM sichern (2) und wiederherstellen (4)
  - Unterbrechungsbehandlung durchführen (3)
  - Ausführung des unterbrochenen Programms wieder aufnehmen (5)



- 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



- 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

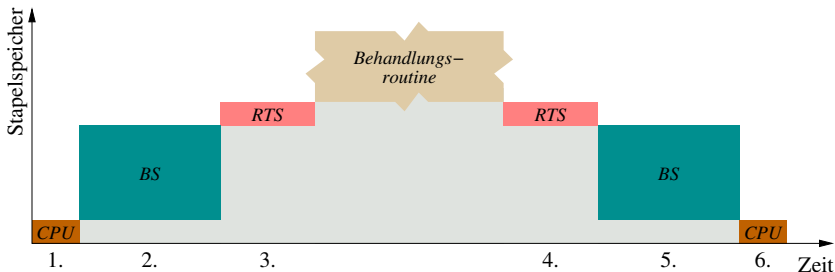


- **Latenz** bis zum Start der Unterbrechungsbehandlung:
  - 1 Annahme der Unterbrechung durch die Hardware
  - 2 Sicherung der Inhalte der (flüchtigen) CPU-Register
  - 3 Aufbau des Aktivierungsblocks der Behandlungsprozedur
  
- **Latenz** bis zur Fortführung des unterbrochenen Programms:
  - 4 Abbau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



- **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





Werte mit fester oberer Schranke sind gefordert:

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





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



<sup>2</sup>Lässt man zugeliferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.



Häufig ist eine eigenständige Beurteilung des Zeitbedarfs nicht möglich,



Herstellerangaben ermöglichen die Abschätzung des **schlimmsten Falls**.



## ■ Beispiel Quadrokopter:

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

---


<sup>2</sup>Lässt man zugeliferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.



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

## ■ Beispiel Quadrokopter:

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

-  Alleine die **Anwendung** kann (fast) komplett kontrolliert werden.<sup>2</sup>

---

<sup>2</sup>Lässt man zugeliferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.

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



- Die Lage des Quadropter 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



- Die Lage des Quadropter 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  $\rightsquigarrow$  A/D-Wandlung
- Nyquist-Shannon-Abtasttheorem

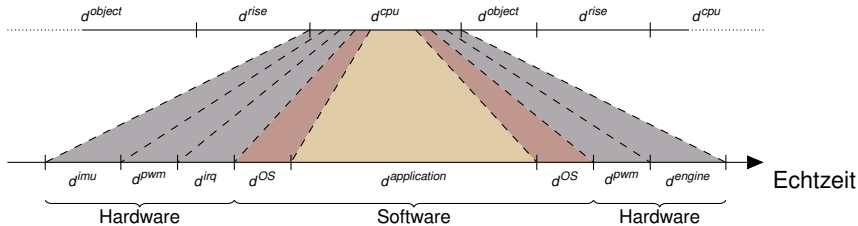


# The Big Picture – Rechenzeitbedarf

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** Abtastrate ( $\sim d^{imu}$ ), Motorleistung ( $\sim d^{engine}$ )

**Mikrocontroller** Signalverarbeitung ( $\sim d^{pwm}$ ), IRQ ( $\sim d^{irq}$ )

**Betriebssystem** Unterbrechungslatenz, Kontextwechsel ( $\sim d^{OS}$ )

**Anwendung** Steuerung, Regelung ( $\sim d^{application}$ )



## 1 Physikalisches System und Echtzeitanwendung

- Kontrolliertes Objekt
- Zusammenspiel

## 2 Echtzeitrechensystem

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

## 3 Zusammenfassung



## Zusammenspiel Kontrolliertes Objekt $\leftrightarrow$ Kontrollierendes Rechensystem

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



## Zusammenspiel Kontrolliertes Objekt $\leftrightarrow$ 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



- [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](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



## 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)

