Echtzeitsysteme

Physikalisches System ⟷ Kontrollierendes Rechensystem

Sommersemester 2024

Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Fragestellungen

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

- Die funktionale Korrektheit ist nicht ausreichend

→ Zeitliche (temporale) Korrektheit!

Fragestellungen

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

- Die funktionale Korrektheit ist nicht ausreichend

→ Zeitliche (temporale) Korrektheit!

⚠ Geschwindigkeit ist **keine Garantie**

- Komplexität des Echtzeitrechensystems

→ Entscheidend ist das tatsächliche Laufzeitverhalten

Fragestellungen

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

- Die funktionale Korrektheit ist nicht ausreichend

→ Zeitliche (temporale) Korrektheit!

⚠ Geschwindigkeit ist **keine Garantie**

- Komplexität des Echtzeitrechensystems

→ Entscheidend ist das tatsächliche Laufzeitverhalten

⚠ Terminvorgaben sind **anwendungsabhängig**

- Komplexität des physikalischen Systems (vgl. Folie II/19 ff)

→ Bestimmt durch die Kopplung an die (reale) Umwelt

Fragestellungen

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

- Die funktionale Korrektheit ist nicht ausreichend

→ Zeitliche (temporale) Korrektheit!

⚠ Geschwindigkeit ist **keine Garantie**

- Komplexität des Echtzeitrechensystems

→ Entscheidend ist das tatsächliche Laufzeitverhalten

⚠ Terminvorgaben sind **anwendungsabhängig**

- Komplexität des physikalischen Systems (vgl. Folie II/19 ff)

→ Bestimmt durch die Kopplung an die (reale) Umwelt

- 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 ){  ...  }
```

Exkurs: Funktionale und nicht-funktionale Eigenschaften

- *Funktionale* Eigenschaften
  - Werden direkt implementiert

Eine Funktion

```
uint16_t   regelschritt ( uint8_t   sensorwert ){  ...  }
```

- *Nicht-funktionale* Eigenschaften
  - Beispiel: Energie, Speicherverbrauch, Laufzeitverhalten
  - Lassen sich nicht direkt implementieren
  - Sind querschneidend $\rightsquigarrow$ erst im konkreten Kontext bestimmt

Exkurs: Funktionale und nicht-funktionale Eigenschaften

- *Funktionale* Eigenschaften
  - Werden direkt implementiert

Eine Funktion

```
uint16_t   regelschritt ( uint8_t   sensorwert ){  ...  }
```

- *Nicht-funktionale* Eigenschaften
  - Beispiel: 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?!

Exkurs: Funktionale und nicht-funktionale Eigenschaften

- *Funktionale* Eigenschaften
  - Werden direkt implementiert

Eine Funktion

```
uint16_t   regelschritt ( uint8_t   sensorwert ){  ...  }
```

- *Nicht-funktionale* Eigenschaften
  - Beispiel: Energie, Speicherverbrauch, Laufzeitverhalten
  - Lassen sich nicht direkt implementieren
  - Sind querschneidend $\rightsquigarrow$ erst im konkreten Kontext bestimmt

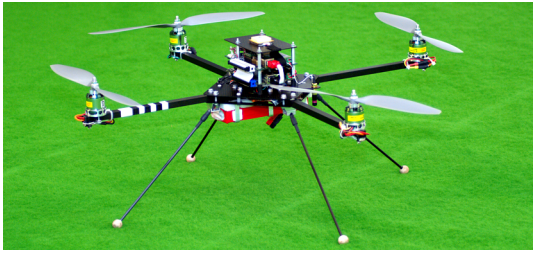
⚠ Zeit aus Sicht des Softwareengineering nicht-funktional
  - Führt häufig zu Verwirrung im Kontext von Echtzeitsystemen

  → Die rechtzeitige Auslösung des Airbags ist funktional?!

- Es kommt auf die **Betrachtungsebene** an!

Übersicht

Aufbau des Demonstrators



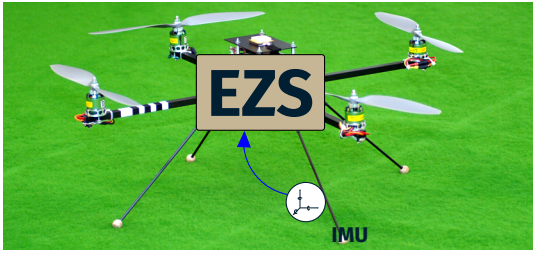
⚠ Quadrokopter sind inhärent instabil $\rightsquigarrow$ ständige, aktive Kontrolle

Aufbau des Demonstrators

⚠ Quadrokopter sind inhärent instabil $\rightsquigarrow$ ständige, aktive Kontrolle

- Aufgabe des Echtzeitsystems: Fluglageregelung (Stabilisierung)

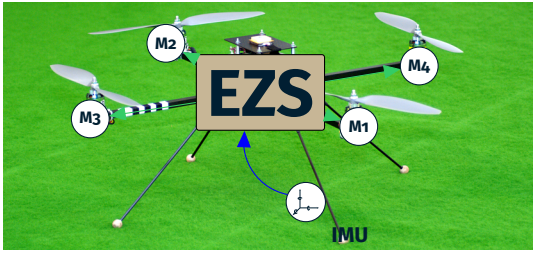
Aufbau des Demonstrators



⚠ Quadrokopter sind inhärent instabil $\rightsquigarrow$ ständige, aktive Kontrolle

- Aufgabe des Echtzeitsystems: Fluglageregelung (Stabilisierung)
  - Bewegung im Raum bestimmen (engl. ***i****nertial* ***m****easurement* ***u****nit*)

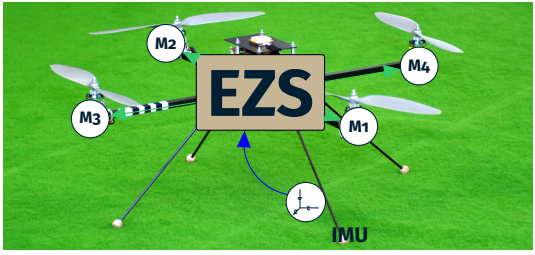
Aufbau des Demonstrators



⚠ Quadrokopter sind inhärent instabil $\rightsquigarrow$ ständige, aktive Kontrolle

- Aufgabe des Echtzeitsystems: Fluglageregelung (Stabilisierung)
  - Bewegung im Raum bestimmen (engl. ***i**nertial **m**easurement **u**nit*)
  - Vorgabe der Motor- und damit der Rotordrehzahl

Aufbau des Demonstrators



⚠ Quadrokopter sind inhärent instabil ⇝ ständige, aktive Kontrolle

- Aufgabe des Echtzeitsystems: Fluglageregelung (Stabilisierung)
  - Bewegung im Raum bestimmen (engl. ***i**nertial **m**easurement **u**nit*)
  - Vorgabe der Motor- und damit der Rotordrehzahl
- Physikalisches Objekt, Echtzeit-Anwendung und -Rechensystem

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

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

⚠ Wie lange dauert es bis zum Gleichgewicht?

- Gewicht, Leistungsfähigkeit der Motoren, Bauart der Rotorblätter, …

→ **Objektdynamik**

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

⚠ Wie lange dauert es bis zum Gleichgewicht?

- Gewicht, Leistungsfähigkeit der Motoren, Bauart der Rotorblätter, …

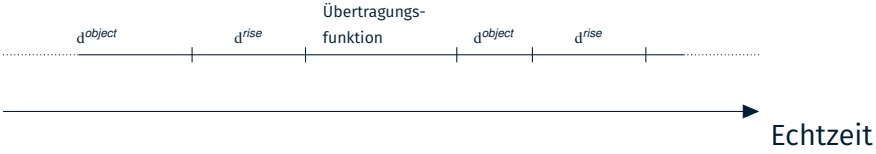
→ **Objektdynamik**

- Dies ist die Welt der **Steuerungs**- und **Regelungsanwendungen**
  - Regelungstechnische Abstraktion des Quadrokopters: 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 physikalisches Objekt

Die Physik kommt ins Spiel … [3]



- Zeitverzögerung (d, delay) des Quadrokopters:

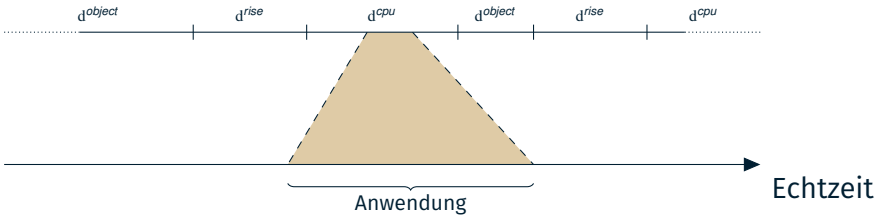
  d^{object} Zeitdauer bis zum Beginn der Lageänderung
  - Hervorgerufen durch die (initiale) *Trägheit des Objektes*

  → *Prozessverzögerung* (engl. *process/object delay*)

  d^{rise} Zeitdauer bis zum (erneuten) Gleichgewicht
  - Allgemein: Erreichen der Zielgröße (typisch: 66 % bzw. 90 %)
  - Bestimmt durch die Fähigkeit der Aktorik ↦ Einschwingverhalten

  → Anregel-/ *Anlaufzeit* (engl. *rise time*) einer Sprungantwort

Zeitparameter des Rechensystems

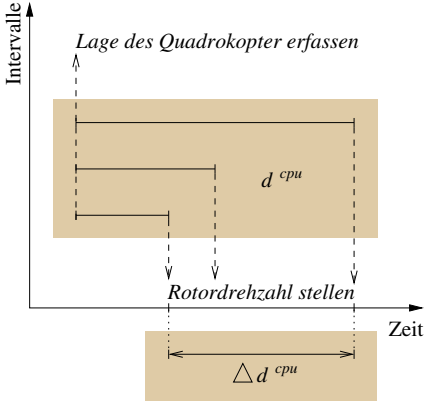


- Zeitverzögerung des Rechensystems
  - → Auswertung: Abweichung (Soll/Ist) und Übertragungsfunktion (Regler)
  - ⚠ Das Rechensystem benötigt Zeit für die Berechnung

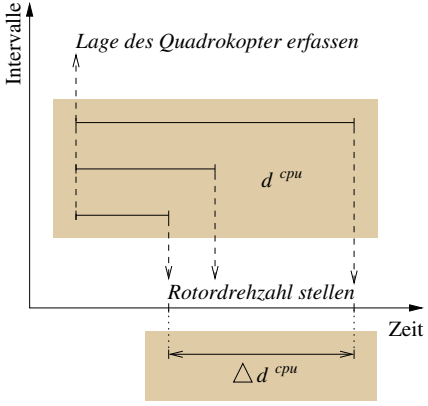
d^{cpu} Zeitdauer bis zur Ausgabe des neuen Stellwertes

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

Schwankungen (engl. *jitter*) in Zeitparametern



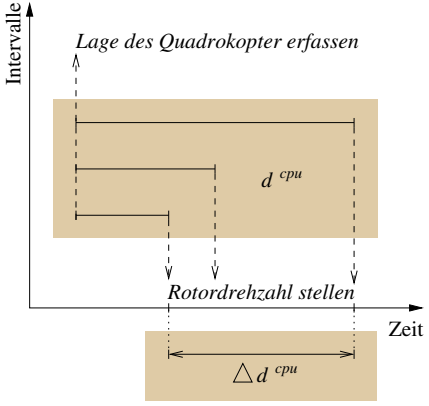
Schwankungen (engl. *jitter*) in Zeitparametern



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

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

Schwankungen (engl. *jitter*) in Zeitparametern



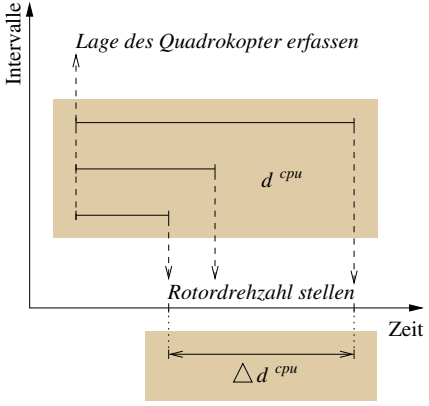
d^{cpu} Auch bei konstantem Rechenaufwand zur Stellwertbestimmung variabel

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

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

- Bewirkt zusätzlichen Fehler
- Beeinträchtigt die Dienstgüte

Schwankungen (engl. *jitter*) in Zeitparametern



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

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

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

- Bewirkt zusätzlichen Fehler
- Beeinträchtigt die Dienstgüte

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

- Bekannte konstante Verzögerungen schon $\rightsquigarrow \mathrm{d}^{dead}$
- Randbedingung: $\triangle \mathrm{d}^{cpu} \ll \mathrm{d}^{cpu}$

Physikalisches Objekt ⟷ Echtzeitrechensystem

- 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

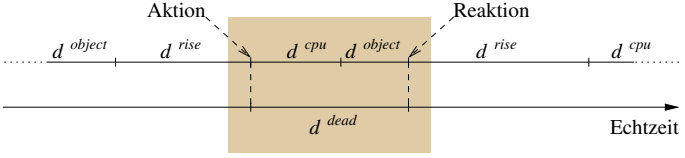
Physikalisches Objekt ⟷ Echtzeitrechensystem

- 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}:

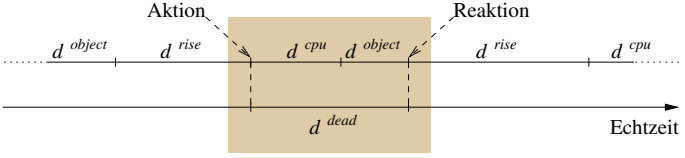
1. Implementierung des kontrollierenden Rechensystems
2. Dynamik des kontrollierten Objektes



Physikalisches Objekt ⟷ Echtzeitrechensystem

- 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



⚠ Auswirkung Güte und Stabilität der Regelung
  - Insbesondere bei hoher Varianz von Δd^{dead} ↦ Jitter

  → Beeinflusst Aussagekraft über die erzielte Wirkung

Übersicht

Zeitliches Verhalten von EZ-Anwendungen

- **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?

Zeitliches Verhalten von EZ-Anwendungen

- **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 (engl. *overheads*) der Laufzeitumgebung

Zeitliches Verhalten von EZ-Anwendungen

- **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 (engl. *overheads*) der Laufzeitumgebung

- Exemplarische Illustration anhand von *Programmunterbrechungen*

Unterbrechungsarten

- Zwei Arten von Programmunterbrechungen:

  synchron die „Falle“ (engl. *trap*)

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

Unterbrechungsarten

- Zwei Arten von Programmunterbrechungen:

  synchron die „Falle“ (engl. *trap*)

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

- Unterschiede ergeben sich hinsichtlich:
  - Quelle
  - Synchronität
  - Vorhersagbarkeit
  - Reproduzierbarbeit

Unterbrechungsarten

- Zwei Arten von Programmunterbrechungen:

  synchron die „Falle“ (engl. *trap*)

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

- Unterschiede ergeben sich hinsichtlich:
  - Quelle
  - Synchronität
  - Vorhersagbarkeit
  - Reproduzierbarbeit

⚠ 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
  - Reproduzierbarbeit

⚠ 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)

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*)

⚠ Programmunterbrechungen implizieren nicht-lokale Sprünge:

- vom { unterbrochenen / behandelnden } zum { behandelnden / unterbrochenen } Programm

Ausnahmebehandlung (engl. *exception handling*)

⚠ Programmunterbrechungen implizieren nicht-lokale Sprünge:

- vom { unterbrochenen / behandelnden } zum { behandelnden / unterbrochenen } 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)

Ausnahmebehandlung (engl. *exception handling*)

⚠ Programmunterbrechungen implizieren nicht-lokale Sprünge:

- vom { unterbrochenen / behandelnden } zum { behandelnden / unterbrochenen } 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[1]

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

[1] Möglicherweise aber auch den kompletten Registersatz.

Hardware (CPU) sichert einen Zustand minimaler Größe[1]

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

Software (Betriebssystem/Compiler) sichert den Rest

- alle { dann noch ungesicherten / flüchtigen / im weiteren Verlauf verwendeten } CPU-Register

[1] Möglicherweise aber auch den kompletten Registersatz.

Zustandssicherung

Hardware (CPU) sichert einen Zustand minimaler Größe[1]

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

Software (Betriebssystem/Compiler) sichert den Rest

- alle { dann noch ungesicherten / flüchtigen / im weiteren Verlauf verwendeten } CPU-Register

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

[1] Möglicherweise aber auch den kompletten Registersatz.

Prozessorstatus sichern und wiederherstellen

- Sichern aller ungesicherten Register auf Befehlssatz-Ebene:

Zeile	x86
1:	`train:`
2:	`  pushal`
3:	`  call handler`
4:	`  popal`
5:	`  iret`

Prozessorstatus sichern und wiederherstellen

- Sichern aller ungesicherten Register auf Befehlssatz-Ebene:

Zeile

```
1:
2:
3:
4:
5:
```

x86

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

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)

Prozessorstatus sichern und wiederherstellen (Forts.)

- Kontextsicherung durch Instrumentierung des Compilers:

gcc

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

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

Aktivierungsblock (engl. *activation record*)

Sicherung/Wiederherstellung nicht-flüchtiger Register (engl. *non-volatile register*)

Türme von Hanoi

```
void hanoi (int n, char from, char to, char via) {
    if (n > 0) {
        hanoi(n - 1, from, via, to);
        printf("schleppe Scheibe %u von %c nach %c\n", n, from, to);
        hanoi(n - 1, via,  to,  from);
    }
}
```

Aktivierungsblock (engl. *activation record*)

Sicherung/Wiederherstellung nicht-flüchtiger Register (engl. *non-volatile register*)

Türme von Hanoi

```
void hanoi (int n, char from, char to, char via) {
    if (n > 0) {
        hanoi(n - 1, from, via, to);
        printf("schleppe Scheibe %u von %c nach %c\n", n, from, to);
        hanoi(n - 1, via,  to,  from);
    }
}
```

Aufwand je nach CPU, Prozedur, Compiler: `gcc -O6 -S hanoi.c`

`hanoi()`-Eintritt

```
pushl %ebp
movl  %esp,%ebp
pushl %edi
pushl %esi
pushl %ebx
subl  $12,%esp
```

`hanoi()`-Austritt

```
leal  -12(%ebp),%esp
popl  %ebx
popl  %esi
popl  %edi
popl  %ebp
ret
```

Für eine Prozedur aufrufende Ebene inhaltsinvariante Register der CPU, deren Inhalte jedoch innerhalb einer aufgerufenen Prozedur verändert werden:

gcc/x86 ∼ `ebp`, `edi`, `esi`, `ebx`

(Aktivierungsblock auch *Stackframe*)

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

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

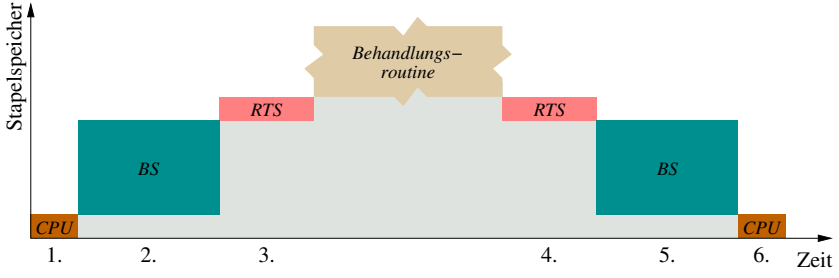
Verwaltungsgemeinkosten des schlimmsten Falls

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 & Häufigkeit der Gemeinkosten müssen für Rechtzeitigkeit nach oben abgeschätzt und beschränkt werden



⚠ Werte mit oberer Schranke sind gefordert auf allen Ebenen:

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

⚠ Häufig ist **isolierte Beurteilung** des Zeitbedarfs **nicht möglich**

- Herstellerangaben ermöglichen Abschätzung des schlimmsten Falls

[2] Lässt man zugelieferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.

Zeitbedarf im EZS

Welche Komponenten benötigen wie viel Zeit?

⚠ Häufig ist **isolierte Beurteilung** des Zeitbedarfs **nicht möglich**

- Herstellerangaben ermöglichen Abschätzung des schlimmsten Falls

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

[2] Lässt man zugelieferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.

Zeitbedarf im EZS

Welche Komponenten benötigen wie viel Zeit?

⚠ Häufig ist **isolierte Beurteilung** des Zeitbedarfs **nicht möglich**

- Herstellerangaben ermöglichen Abschätzung des schlimmsten Falls

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

⚠ Alleine die Anwendung kann (fast) komplett kontrolliert werden[2]

[2] Lässt man zugelieferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.

Abbildung der Fluglageregelung

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

Abbildung der Fluglageregelung

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

  $d^{control}$ Zeitabstand (konstant) zwischen zwei Regelschritten

  - Faustregel: $\mathrm{d}^{sample} < (\mathrm{d}^{rise}/10)$

  → Quasi-kontinuierliches Verhalten des diskreten Systems

Abbildung der Fluglageregelung

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

$d^{control}$ Zeitabstand (konstant) zwischen zwei Regelschritten

- Faustregel: $\mathrm{d}^{sample} < (\mathrm{d}^{rise}/10)$

→ Quasi-kontinuierliches Verhalten des diskreten Systems

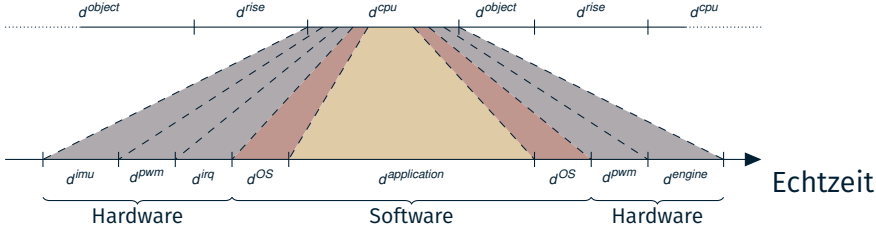
f^{sample} Abtastfrequenz, entspricht $1/\mathrm{d}^{sample}$

- Analoge auf digitale Werte abbilden $\rightsquigarrow$ A/D-Wandlung

→ Nyquist-Shannon-Abtasttheorem

The Big Picture – Rechenzeitbedarf

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



⚠ Alle Komponenten müssen bedacht werden!

Sensoren/Aktoren Abtastrate ($\rightsquigarrow \mathrm{d}^{imu}$), Motorträgheit ($\rightsquigarrow \mathrm{d}^{engine}$)

Mikrocontroller Signalverarbeitung ($\rightsquigarrow \mathrm{d}^{pwm}$), IRQ ($\rightsquigarrow \mathrm{d}^{irq}$)

Betriebssystem Unterbrechungslatenz, Kontextwechsel ($\rightsquigarrow \mathrm{d}^{OS}$)

Anwendung Steuerung, Regelung ($\rightsquigarrow \mathrm{d}^{application}$)

Übersicht

Resümee

Zusammenspiel kontrolliertes Objekt ⟷ kontrollierendes Rechensystem

- Die **Objektdynamik** definiert den zeitlichen Rahmen durch Termine
- Die Echtzeitanwendung muss diese Termine einhalten

Resümee

Zusammenspiel kontrolliertes Objekt ⟷ kontrollierendes Rechensystem

- Die **Objektdynamik** definiert den zeitlichen Rahmen durch Termine
- Die Echtzeitanwendung muss diese Termine einhalten

Programmunterbrechung in synchroner oder asynchroner Ausprägung

- Beeinflussen den Ablauf der Echtzeitanwendung
- Zustandssicherung für Transparenz im Ablauf
- Verwaltungsgemeinkosten des schlimmsten Falls notwendig

Literaturverzeichnis (1)

[1] Infineon Technologies AG.
***TC1796 User's Manual (V2.0)*, July 2007.**

[2] InvenSense Inc.
ITG-3200 Product Specification Revision 1.4.
`http://invensense.com/mems/gyro/documents/PS-ITG-3200A.pdf`, 2010.

[3] Hermann Kopetz.
***Real-Time Systems: Design Principles for Distributed Embedded Applications*.**
Kluwer Academic Publishers, first edition, 1997.

Literaturverzeichnis (2)

[4] Daniel Lohmann.
Vorlesung: Betriebssysteme, Friedrich-Alexander-Universität Erlangen-Nürnberg.
https://www4.cs.fau.de/Lehre/WS15/V_BS, 2015.

[5] Daniel Lohmann, Wanja Hofer, Wolfgang Schröder-Preikschat, Jochen Streicher, and Olaf Spinczyk.
CiAO: An aspect-oriented operating-system family for resource-constrained embedded systems.
In *Proceedings of the 2009 USENIX Annual Technical Conference*, pages 215–228, Berkeley, CA, USA, June 2009. USENIX Association.

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)