Anwendungsanalyse von Echtzeitsystemen

Vorgehen in der Praxis

Florian Franzmann Martin Hoffmann Tobias Klaus
Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

17. November 2014

1. Anwendungsanalyse
   - Prequel: Anforderungsanalyse
   - Anwendungsanalyse
2. Hinweise zu Aufgabe 4
   - Software Tracing

Problemstellung

Wie komme ich vom Echtzeitproblem zum Echtzeitsystem?

- Zunächst *Anforderungsanalyse*
- Danach *Anwendungsanalyse*:
  - Einordnung
  - Zielsetzung
  - Problematik
  - Lösungsansätze
  - Anforderungen und Fakten

Analyse der Problemstellung

- methodisch gestütztes Aufstellen von Anforderungen
- *Anforderung* (engl. requirements) $\leadsto$ Aussage über
  - eine zu erbringende Leistung eines Produkts oder eines Systems
  - eine Eigenschaft, die erfüllt sein muss, damit ein bestimmter Vorgang gelingen kann
  - ein Leistungsmerkmal (nicht nur) von Software
- Zusammenfassung im *Lasten-/Pflichtenheft* als Bestandteil eines zu erstellenden Anforderungsdokuments, das
  - die durch das System zu lösende Aufgabe beschreibt
  - die im Projekt zu erreichenden Ziele definiert
  - den Benutzerkreis des zu entwickelnden Systems festlegt
  - . . . in Zusammenarbeit mit dem Kunden

Spezifikationstechniken

Allgemeine Klassifikation:

formal (engl. formal) rigorose mathematische Grundlage
$\rightsquigarrow$ *formale Notation*

informell (engl. informal) wenn *Transkription* in formale Notation mit zugeordneten Regeln schwierig
$\rightsquigarrow$ z. B. Ablaufdiagramm (engl. flowchart)

halbförmlich (engl. semiformal) Mischung aus formal und informell, z. B. UML:

- Zustandsdiagramm (engl. statechart) formal
- andere Konzepte eher pseudomathematisch

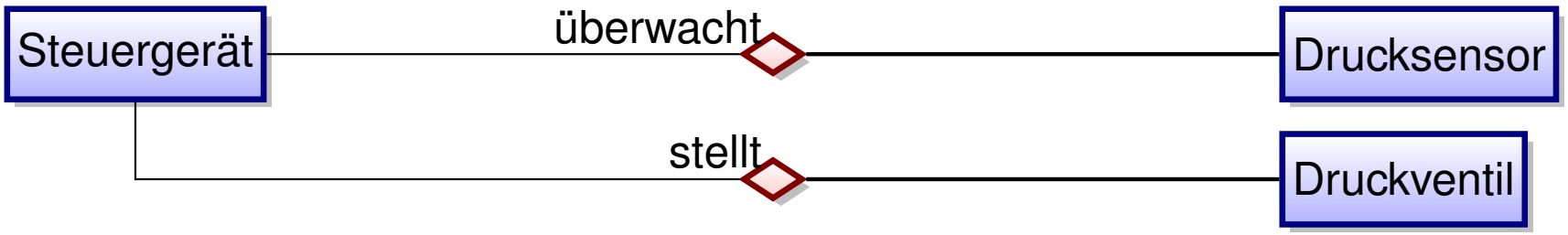
Echtzeitsysteme mit strikt einzuhaltenden Anforderungen

. . . *erfordern eine formale Begründung* der Leistungscharakteristiken und Anforderungen!

Ausflug: Datenmodellierung

- Grundlage ist *Entity-Relationship*-Ansatz
- Modelliert werden Ausschnitte der Realität durch . . .

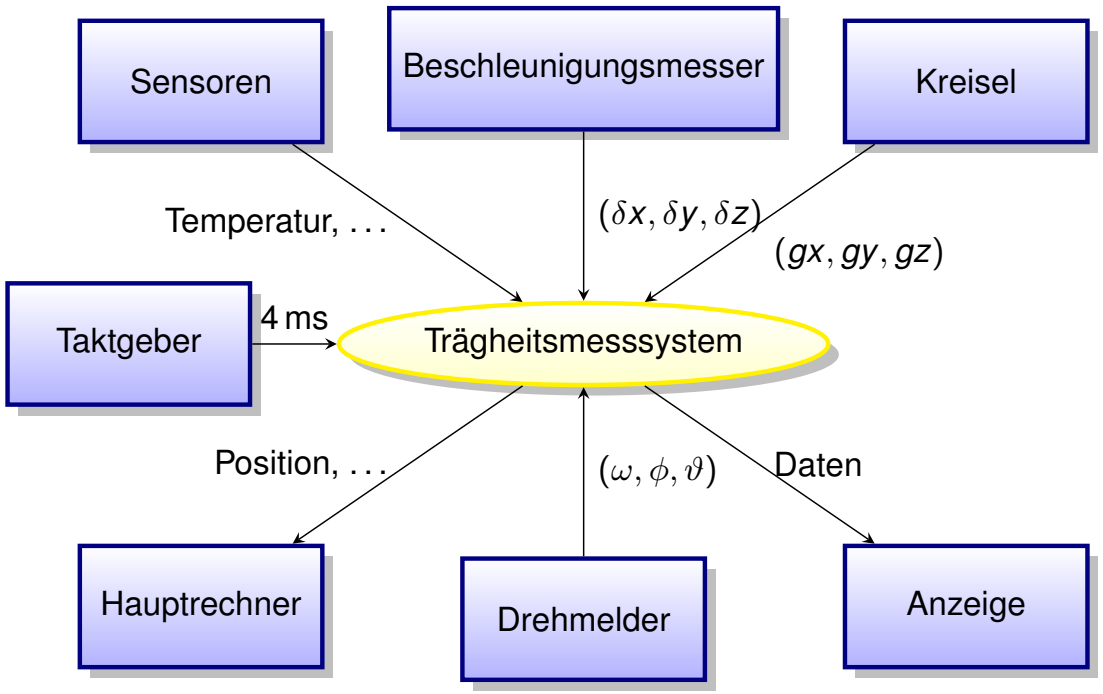
  Gegenstandstypen engl. entity types
  Beziehungstypen engl. relation types
  Attribute engl. attributes



- ☺ Einfach und klar, ideal für Datenbanken
- ☹ Weder Funktionalität noch Verhalten von Systemen
- ☹ Keine Dekomposition
- ☹ Keine Datenkapselung

Ausflug: Strukturierte Analyse

- Grundlage: *Datenflussdiagramme*
- Modellierung von Systemfunktionalität
- Beschreibung des Systemkontexts
  - Interaktion
  - Ein-/Ausgabe



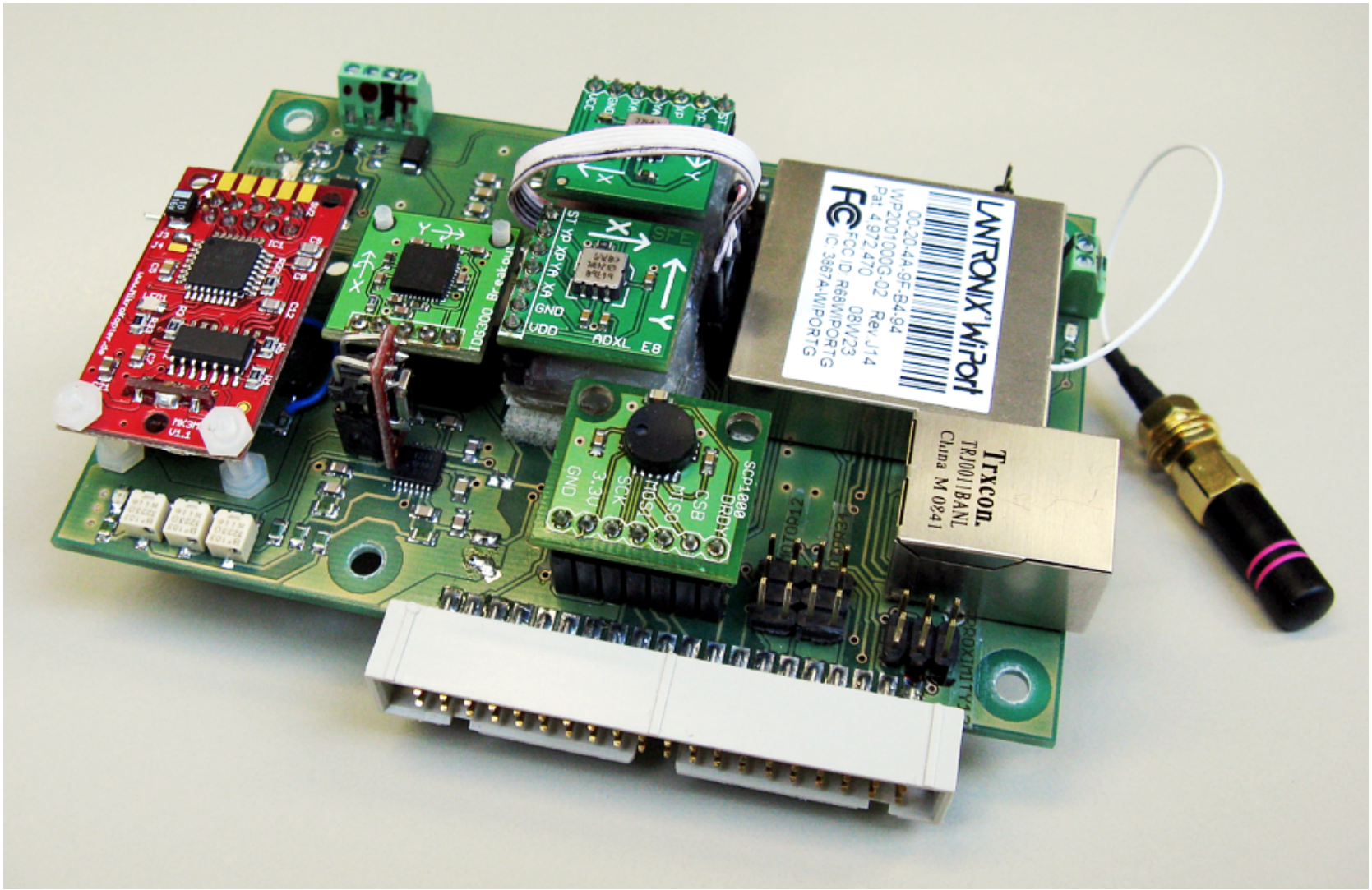
☺ Anschaulich, Dekomposition

☹ keine Lokalität, begrenzte Kapselungsfähigkeit

☹ keine nichtfunktionalen Eigenschaften

☹ *Strukturbruch:* Spezifikation $\leftrightarrow$ Implementierung

Fakten: I4Copter

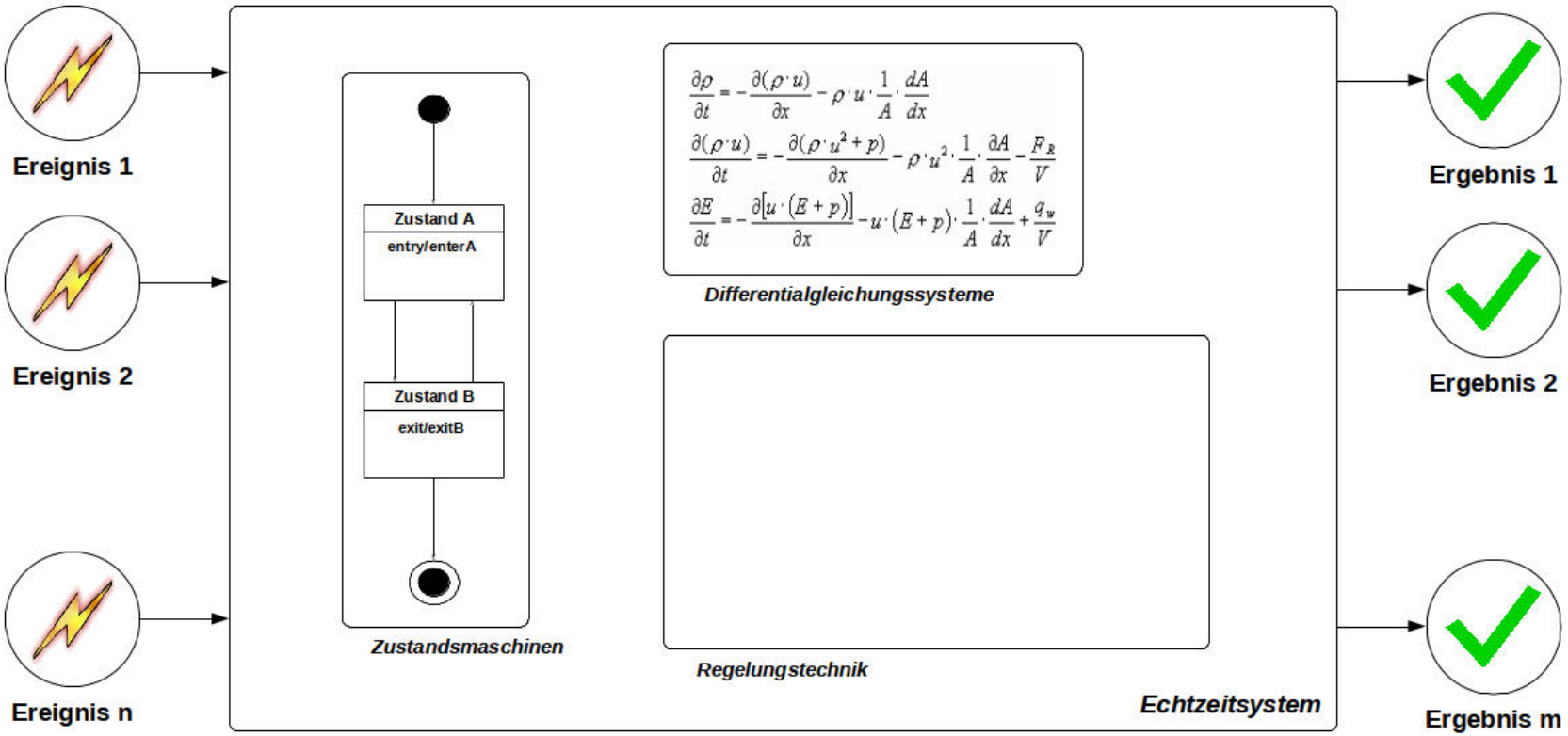


Einordnung – Anwendungsanalyse

Beziehung zwischen *Ereignis n* und *Ergebnis m*

Zeitlich: Wie viel Zeit darf verstreichen? $\rightsquigarrow$ Termine

Physikalisch: Wie ist das Ergebnis zu bestimmen?



Fragestellungen der Echtzeitanwendungsanalyse

- *Physikalisches Objekt*:
  - Welche Größen sind relevant?
  - Wie hängen die Größen zusammen?
- *Echtzeitsystem*:
  - Welche Ereignisse gilt es zu behandeln?
  - Welche Zeitschranken gilt es einzuhalten?
  - Welche Beziehung Zeitschranke $\leftrightarrow$ physikalisches Objekt?
- Wie sieht das *physikalische Modell* aus?
  - Welche Größen muss man abbilden?
  - Wie bildet man diese Größen ab?

Herausforderungen

Problematik

- einfach erscheinende Objekte physikalisch äußerst komplex
  - $\leadsto$ Vereinfachende Annahmen unabdingbar
- Beispiel: „Hau den Lukas“ vs. Quadrocopter

Lösungsansätze

- Reduktion auf den Zustand
- Regelungstechnik

Reduktion auf den Zustand . . .

. . . genauer: den beobachtbaren Zustand

- Idee: Man kann den Zustand . . .
  - . . . *immer beobachten*
  - . . . *gezielt* und *exakt manipulieren*

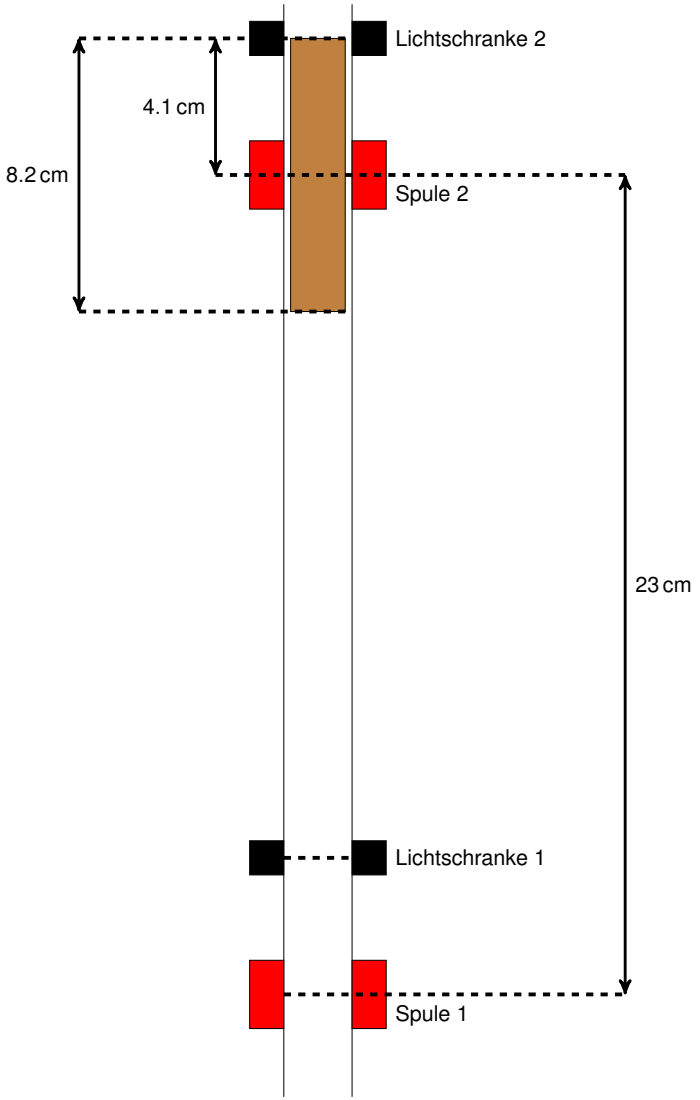
⇝ Konsequenz für unser Modell:

- es kann auf den *beobachtbaren Zustand* reduziert werden
- häufig nur noch *diskretisierte Wertebereiche*
- reine Kausalitätsbeziehungen

☺ drastische *Vereinfachung* des Modells

⇝ ohne relevante Eigenschaften zu verlieren

Beispiel: „Hau den Lukas“



- *Beobachtung:*
  - der Eisenkernposition
  - der Bewegungsrichtung

  durch Lichtschranken

- *Manipulation:* Eisenkern kann
  - festgehalten
  - fallengelassen
  - angehoben

  werden

☺ Zustand ist
  - vollständig beobachtbar
  - gezielt manipulierbar

Nachteile

- Modell sagt nichts aus über . . .
  - den Eisenkern
  - die verwendeten Spulen
  - die Umgebungstemperatur

⇝ Gezielte Manipulation *nicht garantierbar* . . .
. . . wenn diese Größen verändert werden

⇝ System muss sich *gutmütig* verhalten
  - ggü. Parametern, die man nicht kontrollieren kann
  - diese sind dann vernachlässigbar

☹ Längst nicht alle Systeme erfüllen diese Eigenschaft

Regelungstechnik

Problem:
- interner Zustand *nicht beobachtbar*
- *interne Parameter beeinflussen das System* in relevantem Umfang

Idee: Nachbildung/Berechnung des internen Zustands

$\rightsquigarrow$ physikalisches Modell
- inklusive internem Verhalten

- Konsequenz für unser Modell
  - *Mathematisch/physikalische Beschreibung* des Systems
  - Bestimmung der Systemparameter
    $\rightsquigarrow$ Trägheit, Widerstand, ...
  - Berücksichtigung *vergangener Zustände*

☺ Beschreibung des Systems durch das Modell
☹ detaillierte Analyse notwendig

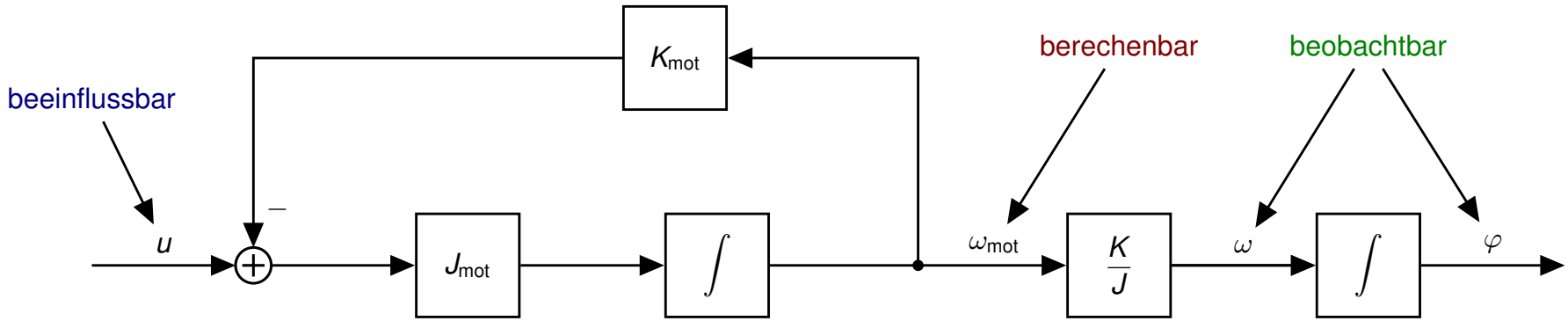
Regelungstechnik

Systemtheorie

Eingangsvektor Abweichung vom Sollwert des Systems

Ausgangsvektor Ist-Wert des Systems

Systemmatrix Rückkopplung interner Zustände



Systemmodell

Beschreibt Zusammenhänge zwischen beeinflussbaren, beobachtbaren, berechenbaren Systemparametern

Beispiel: Quadcopter

- *Beobachtung*: Winkelgeschwindigkeit und Lage um x-/y-Achse
- *Manipulation*:
  - erzeugte Schubkraft kann variiert werden
  - geregelt wird *Spannung* der Motoren
- *Reaktion*:
  - abhängig von Momenten des Objekts (Masse, Trägheit)
  - und Motor/Propeller (Trägheit, Reibung, Wirkungsgrad)

☺ Zustand nicht beobachtbar, aber berechenbar

Modellbildung

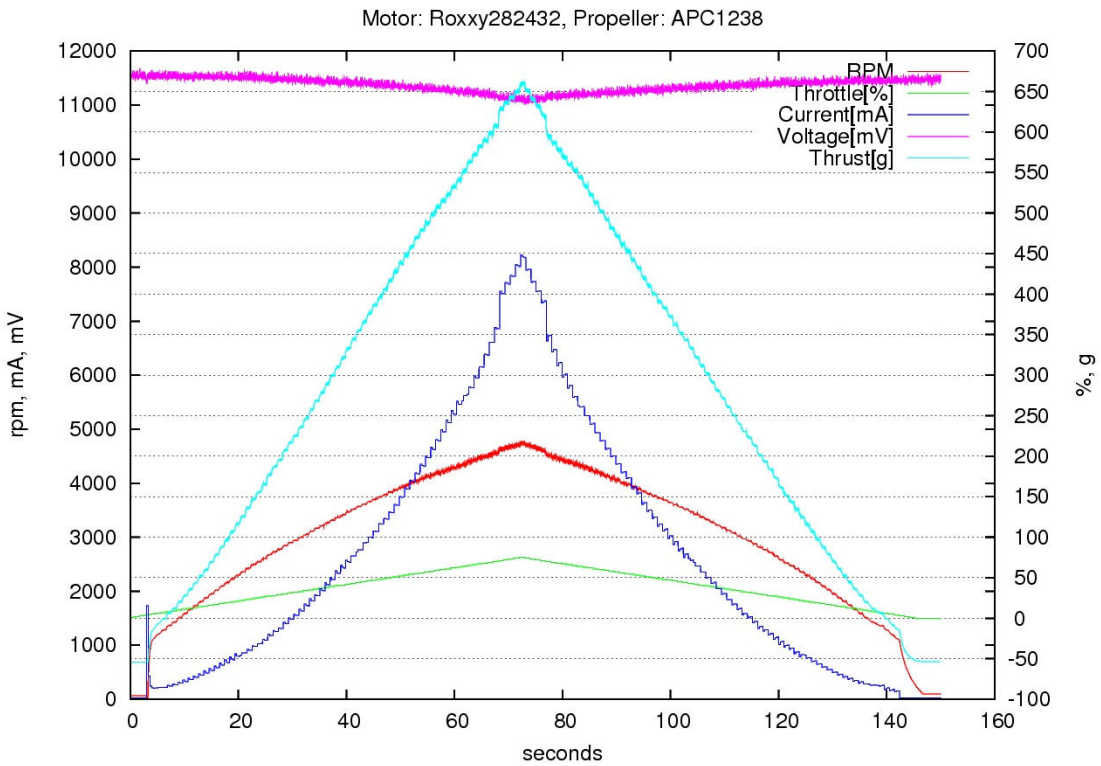
- Berechnung beliebiger Zustände
- basierend auf *Messungen*
  - ⇝ Kennlinien erstellen ⇝ Messung zusätzlicher Parameter
    - Herleitung von Konstanten/Funktionen

- z. B. Motor:
  - Schubkraft
  - Drehzahl
  - Stromaufnahme
  - Spannung
  - diskrete Stellgröße

⇝ Motorkonstante

⇝ *Funktion*

Stellgröße $\mapsto$ Schub



Fazit

- scheinbar einfache Vorgänge physikalisch sehr komplex
- physikalische Vorgänge mathematisch oft *nicht analytisch lösbar*
  - ⇝ Vereinfachung des Modells
    - numerische Lösungsansätze
- Massive Vereinfachung *notwendig*
  - Was muss man wirklich wissen?
  - Was kann man wissen/messen?
  - Welche Einschränkungen sind damit verbunden?

Entwurf des Echtzeitsystems setzt Vertrautheit mit physikalischem Objekt voraus!

Ergebnis der Analyse

- Welche *Aktivitäten* laufen in dem System ab?
  - Können diese Aktivitäten feiner strukturiert werden?
  - Laufen Elemente einer Aktivität zeitgleich ab?

  $\rightsquigarrow$ *Aufgaben*
- Wann werden diese Aktivitäten ausgeführt?
  - Welche zeitlichen Eigenschaften haben diese Zeitpunkte?

  $\rightsquigarrow$ *Ereignisse*
- Was hängt von den berechneten Ergebnissen ab?
  - Wie viel Zeit darf dabei verstreichen?

  $\rightsquigarrow$ *Termine*

Entwicklung im Emulator

- Ab Aufgabe 4 verwenden wir einen i386-Emulator
  - ⇝ FailBochs
- Was ändert sich dadurch für euch?
  - Zeitgeber deutlich geringerer Auflösung
  - `make flash` funktioniert nicht mehr
    ⇝ stattdessen `make sim`, `make run` und `make ddd`
  - `gdb` bzw. `ddd` anstatt `trace32` ☹

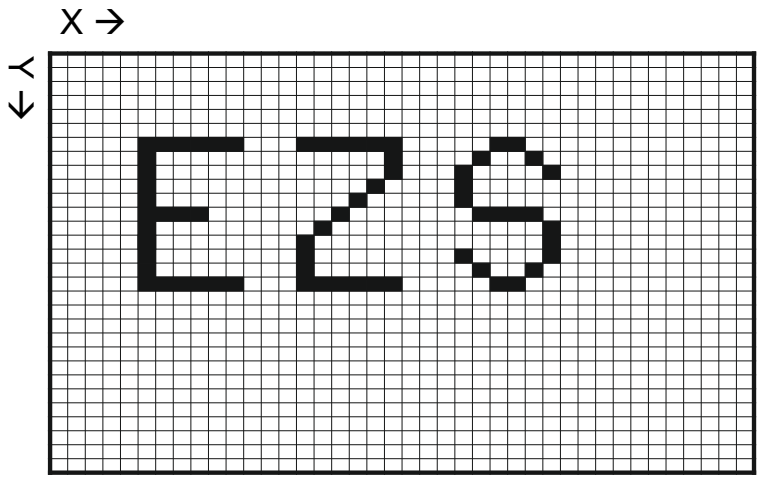
Bearbeitung der Übungsaufgaben im Informatik-CIP-Pool

- Kein Dienstags-Übungstermin mehr
- Sobald alle Teilnehmer Aufgabe 3 abgegeben haben

⇝ *wir schreiben eine E-Mail sobald es soweit ist*

Grafische Ausgabe – Das Display

Framebuffer Grundlagen

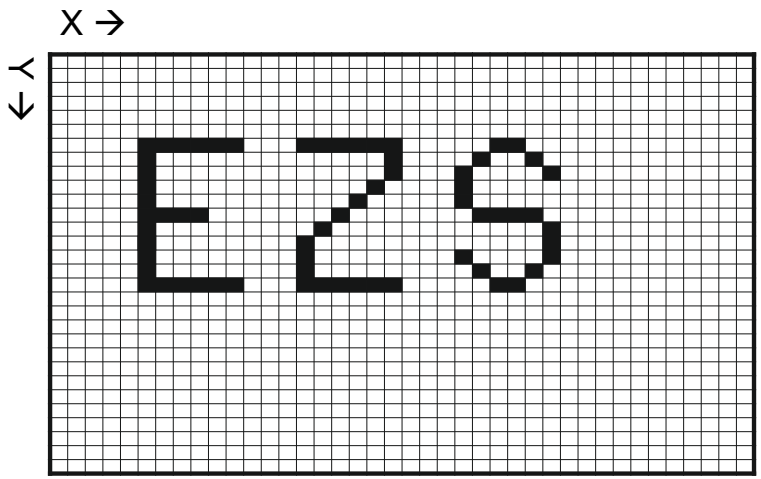


- **Framebuffer** $\rightsquigarrow$ Grafische Ausgabe
- eCos bietet einheitliche, *hardwareunabhängig Schnittstelle*[1]
- Auflösung in unserem Fall: 320x240x16 bit (qemu & Hardware)
  - Zwecks Portabilität immer Variablen nutzen:
  - Breite: `CYG_FB_WIDTH(FRAMEBUF)`
  - Höhe: `CYG_FB_HEIGHT(FRAMEBUF)`

[1] `ecos.sourceware.org/docs-latest/ref/io-framebuf.html`

Grafische Ausgabe – Das Display

Framebuffer Grundlagen



- Nützliche Funktionen:
  - `void ezs_fb_init(void)`
  - `void ezs_fb_clear(cyg_fb_colour color)`
  - `void ezs_fb_fill_block(cyg_ucount16 x, cyg_ucount16 y, cyg_ucount16 width,cyg_ucount16 height,cyg_fb_colour color)`
  - `void ezs_fb_print_string_cur(char* c, cyg_ucount16 x, cyg_ucount16 y, cyg_fb_colour color)`

Leistungsdichtespektrum in libEZS

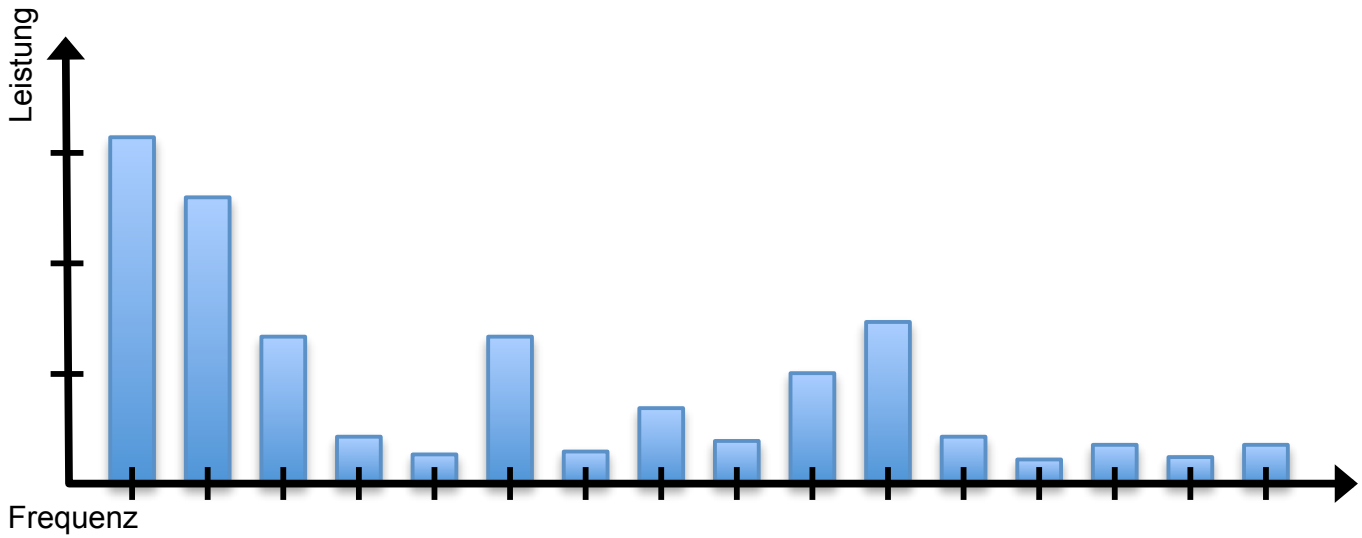
```
void ezs_power_density_spectrum(float in[], float out[], int N)
```

- `in[]` *Eingabe*, Abschnitt des Zeitbereichssignals
- `out[]` *Ausgabe*, Leistungsdichtespektrum (LDS)
- `N` *Anzahl der Abtastwerte*, wobei N Zweierpotenz

- Zeitbereichssignal der Länge $N \mapsto$ LDS[2] der Länge $\frac{N}{2}$
- Höchste Frequenz im Spektrum aus Abtasttheorem $\Rightarrow \frac{f_{\text{Abtast}}}{2}$

[2] http://en.wikipedia.org/wiki/Spectral_density

LDS – Beispiel

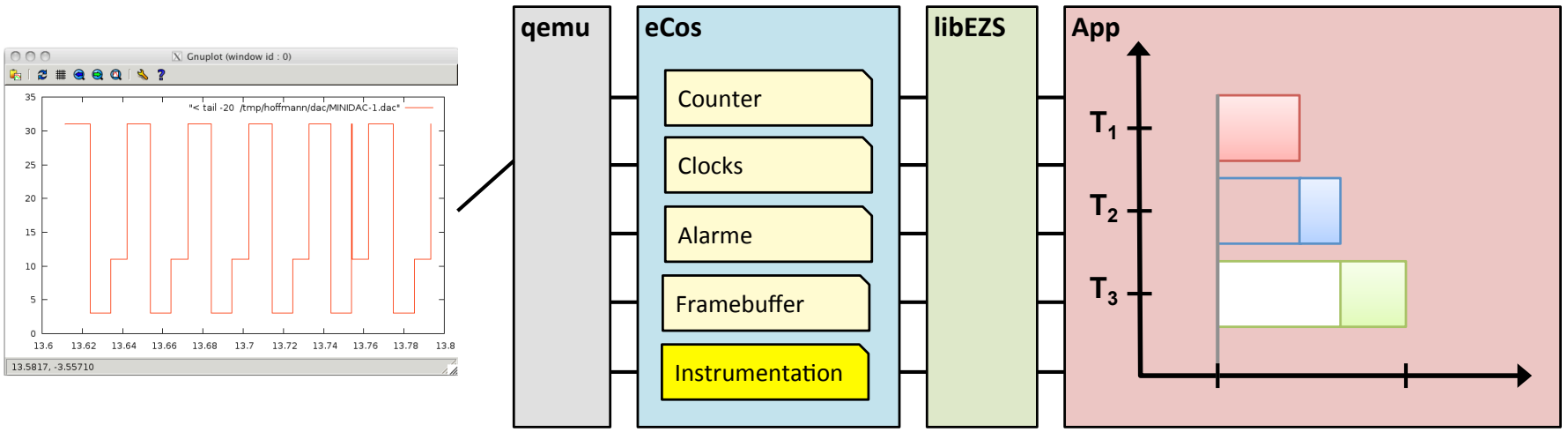


- Darstellung mit 16 Werten:
  $\Rightarrow$ LDS der Länge 32 notwendig
- 1 Hz Abstand $\rightsquigarrow$ Spektrum bis 16 Hz erfassen
  $\Rightarrow$ Abtastfrequenz 32 Hz

$\Rightarrow$ Balken repräsentieren Leistung

Software Tracing

Darstellung mittels Ablaufgraph



- **Visualisierung der Threads** $\rightsquigarrow$ Softwarebasiertes Tracing
- eCos Instrumentations[3]
- Ausgabe der Priorität $\rightsquigarrow$ **Ablaufgraph**

[3] ecos.sourceware.org/docs-latest/user-guide/kernel-instrumentation.html