

DIY – Individual Prototyping and Systems Engineering

Grundlagen und Terminologie

Peter Ulbrich

Lehrstuhl für Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

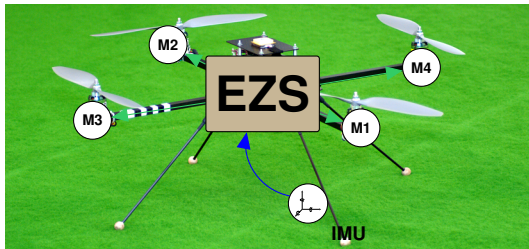
<https://www4.cs.fau.de>

17. Mai 2017



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



Wie lange dauert es bis zum Gleichgewicht?

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



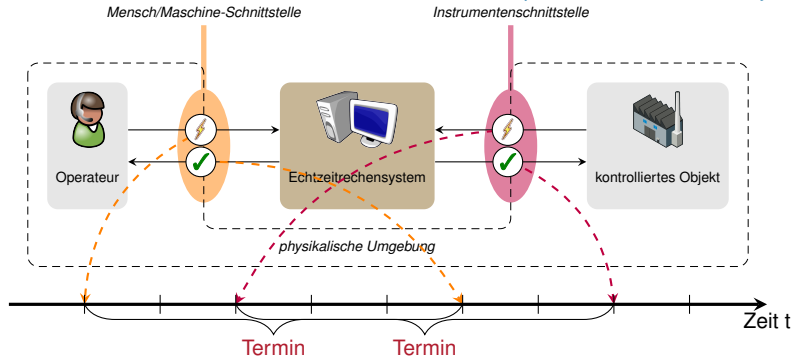
Dies ist die Welt der Steuerungs- und Regelungsanwendungen

- 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



Kopplung mit der (realen) Umwelt

Komponenten eines Echtzeitsystems



- Echtzeitsystem interagiert mit der **physikalischen Umwelt**
- Berechnet als Reaktion auf **Ereignisse** ⚡ (engl. *event*, Stimuli) der Umgebung **Ergebnisse** ✓ (engl. *result*)
- Zeitpunkt, zu dem ein Ergebnis vorliegen muss, wird als **Termin** oder **Frist** (engl. *deadline*) bezeichnet





Echtzeitbetrieb bedeutet **Rechtzeitigkeit**

- Funktionale Korrektheit reicht für korrektes Systemverhalten nicht aus
- **Rechtzeitige** Bereitstellung der Ergebnisse ist **entscheidend**



Den Rahmen stecken der **Eintrittspunkt** des Ereignisses und der entsprechende **Termin** ab



Termine hängen dabei von der Anwendung ab

wenige Mikrosekunden z.B. Drehzahl- und Stromregelung bei der Ansteuerung von Elektromotoren

einige Millisekunden z.B. Multimedia-Anwendungen (Übertragung von Ton- und Video)

Sekunden, Minuten, Stunden z.B. Prozessanlagen (Erhitzen von Wasser)





Geschwindigkeit ist keine Garantie für die rechtzeitige Bereitstellung von Ergebnissen

- **Asynchrone Programmunterbrechungen** (engl. *interrupts*) können **unvorhersagbare Laufzeitvarianzen** verursachen
- Schnelle Programmausführung ist bestenfalls hinreichend für die rechtzeitige Bearbeitung einer Aufgabe



Zeit ist keine intrinsische Eigenschaft des Rechensystems

- Die Zeitskala des Rechensystems muss nicht mit der durch die Umgebung vorgegebenen (Realzeit) übereinstimmen $\leadsto$ Zeitgeber?
- Temporale Eigenschaften des kontrollierten (physikalischen) Objekts müssen im Rechner system geeignet abgebildet werden





Konsequenzen überschrittener Termine

Verbindlichkeit von Terminvorgaben

- **Weich** (engl. *soft*) auch „schwach“
 - **Ergebnis verliert** mit zunehmender Terminüberschreitung **an Wert** (z.B. Bildrate bei Multimediasystemen)
→ Terminverletzung ist tolerierbar
- **Fest** (engl. *firm*) auch „stark“
 - **Ergebnis wird** durch eine Terminüberschreitung **wertlos** und wird verworfen (z.B. Abgabetermin einer Übungsaufgabe)
→ Terminverletzung ist tolerierbar, führt zum Arbeitsabbruch
- **Hart** (engl. *hard*) auch „strikt“
 - **Terminüberschreitung** kann zum **Systemversagen** führen und eine „Katastrophe“ hervorrufen (z.B. Airbag)
→ Terminverletzung ist keinesfalls tolerierbar





Arten von Echtzeitsystemen

Fest $\longleftrightarrow$ Hart

- **Fest/Hart** $\mapsto$ Terminverletzung ist nicht ausgeschlossen¹
 - Terminverletzung wird vom Betriebssystem erkannt
 - $\rightarrow$ Weiteres Vorgehen hängt von der Art des Termins ab

Fest $\leadsto$ plangemäß weiterarbeiten

- Betriebssystem bricht den Arbeitsauftrag ab
- Nächster Arbeitsauftrag wird (planmäßig) gestartet
- $\rightarrow$ Transparent für die Anwendung

hart $\leadsto$ sicheren Zustand finden

- Betriebssystem löst eine **Ausnahmesituation** aus
- Ausnahme ist **intransparent für die Anwendung**
- $\rightarrow$ **Anwendung** behandelt diese Ausnahme

¹ Auch wenn Ablaufplan und Betriebssystem auf dem Blatt Papier Determinismus zeigen, kann das im Feld eingesetzte technische System von unbekannten/unvermeidbaren Störeinflüssen betroffen sein!



■ Hard real-time computer system

(dt. Hartes Echtzeitrechnungssystem)

- Rechensystem mit mind. einem hartem Termin
- Garantiert unter allen (spezifizierten) Last- und Fehlerbedingungen
- Laufzeitverhalten ist ausnahmslos **deterministisch**
- Typisch für **sicherheitskritische Echtzeitrechnungssysteme**
 - engl. *safety-critical real-time computer system*
 - Beispiel: Fluglageregelung, Airbag, ...

■ Soft real-time computer system

(dt. Weiches Echtzeitrechnungssystem)

- Rechensystem welches keinen harten Termin erreichen muss
- Termine können gelegentlich verpasst werden



1 Problemstellung

- Kontrolliertes Objekt
- Realzeitbetrieb
- Termine

2 Allgemeine Grundlagen

- Begriffsdefinition: Echtzeitrechensystem
- Terminologie
- Programmunterbrechungen
- Verdrängbarkeit

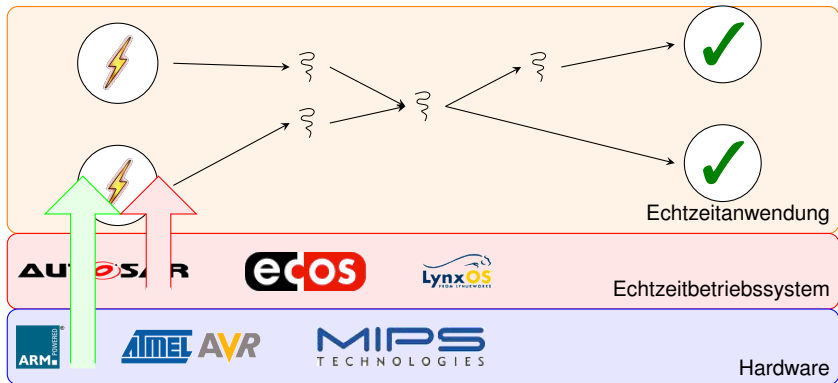
3 Echtzeit-Aufgaben

- Grundsätzliche Umsetzungsalternativen
- Periodische Aufgaben
- Nicht-periodische Aufgaben

4 Umsetzung und Ablaufplanung

- Zeitgesteuerte Ausführung
- Ereignisgesteuerte Ausführung
- Rangfolge





- 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** $\leadsto$ 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!



- 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





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 **synchronen 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
- 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**





Arbeitsaufträge könn(t)en verschränkt ausgeführt werden, wenn:

- Diese verdrängbar sind (typischerweise durch den Planer)
- Die Zeitbedingungen (engl. *time constraints*) es erlauben



Präemptivität (engl. *preemptivity*) ist eine Eigenschaft des jeweiligen Arbeitsauftrags:

- **Verdrängbar** (engl. *preemptable*) ist ein Arbeitsauftrag, wenn seine Ausführung suspendiert werden darf
 - An beliebigen Stellen (engl. *fully preemptive*)
 - An ausgewiesenen Stellen (engl. *preemption points*)
- **Unverdrängbar** (engl. *non-preemptable*), sonst
 - Der Arbeitsauftrag läuft durch (engl. *run-to-completion*)



Mischbetrieb $\leadsto$ Präemptivität als **Auftragattribut** implementiert



1 Problemstellung

- Kontrolliertes Objekt
- Realzeitbetrieb
- Termine

2 Allgemeine Grundlagen

- Begriffsdefinition: Echtzeitrechnungssystem
- Terminologie
- Programmunterbrechungen
- Verdrängbarkeit

3 Echtzeit-Aufgaben

- Grundsätzliche Umsetzungsalternativen
- Periodische Aufgaben
- Nicht-periodische Aufgaben

4 Umsetzung und Ablaufplanung

- Zeitgesteuerte Ausführung
- Ereignisgesteuerte Ausführung
- Rangfolge



Zeitgesteuert (engl. *time-triggered*, auch *time driven*) ✓

- Einlastung nur zu festen Zeitpunkten
→ Vorgegeben durch das Echtzeitrechnungssystem
- Offline (entkoppelte) Einplanung

Reihum gewichtet (engl. *weighted round robin*)

- Echtzeitverkehr in Hochgeschwindigkeitsnetzen
 - im Koppelnetz (engl. *switched network*)
- Untypisch für die Einplanung von **Arbeitsaufträgen**

Ereignisgesteuert (engl. *event-triggered*, auch *event driven*) ✓

- Einlastung zu Ereigniszeitpunkten
→ Vorgegeben durch das kontrollierte Objekt
- Online (gekoppelte) Einplanung





Zeitgesteuertes (engl. *time-triggered*) Echtzeitsystem



Einlastungszeitpunkte von Arbeitsaufträgen *à priori* bestimmt

- Alle Parameter aller Arbeitsaufträge sind *off-line* bekannt
- WCET, Betriebsmittelbedarf (z.B. Speicher, Fäden, Energie), ...



Verwaltungsgemeinkosten zur Laufzeit sind minimal

■ Einlastung erfolgt in **variablen** oder **festen Intervallen**

- Variables Intervall durch **Zeitgeber** (engl. *timer*) mit der Länge des jeweils einzulastenden Arbeitsauftrags programmiert $\mapsto$ WCET (siehe Kapitel III-3)
 - Jeder Zeitablauf bewirkt eine asynchrone Programmunterbrechung
 - Als Folge findet die Einlastung des nächsten Arbeitsauftrags statt
- Festes Intervall mittels regelmäßiger Unterbrechungen (Zeitgeber)
 - Festes Zeitraster liegt über die Ausführung der Arbeitsaufträge
 - Dient z.B. dem Abfragen (engl. *polling*) von Sensoren/Geräten





Ereignisgesteuertes (engl. *event-triggered*) Echtzeitsystem



Einplanung und Einlastungszeitpunkte **vorab nicht bekannt**

- Asynchrone Programmunterbrechungen: Hardwareereignisse
 - Zeitsignal, Bereitstellung von Sensordaten, Beendigung von E/A
- Synchronisationspunkte: ein-/mehrseitige Synchronisation
 - Schlossvariable, Semaphor, Monitor



Ereignisse haben **Prioritäten** → **Dringlichkeiten**

- Zuteilung von Betriebsmitteln erfolgt **prioritätsorientiert**
 - Arbeitsaufträge höherer Priorität haben Vorrang
- Prioritäten werden *offline* vergeben und ggf. *online* fortgeschrieben
 - Arbeitsaufträge haben eine statische oder dynamische Priorität
- Betriebsmittel (insb. CPU) bleiben niemals absichtlich ungenutzt
 - Im Gegensatz zur Ereignissteuerung, die Betriebsmittel brach liegen lässt





Periodische Aufgaben

Aufgaben die in **regelmäßigen Zeitintervallen**² kontinuierlich eine vorgegebene Systemfunktion erbringen.

Eine periodische Aufgabe (T_i) ist eine Abfolge von Arbeitsaufträgen ($J_{i,j}$) mit vorgegebenen zeitlichen Eigenschaften.



$$T_i = (p_i, e_i, D_i, \phi_i)$$

p_i Periode (engl. *period*)

e_i Maximale Ausführungszeit (WCET)

D_i Relativer Termin (engl. *deadline*)

ϕ_i Phase (engl. *phase*)

$$J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$$

²Nach [3, S. 40 ff] ist eine periodische Aufgabe nicht wirklich periodisch, da die Abstände zwischen den **Auslösezeiten** (engl. *interrelease time*) eines Arbeitsauftrags einer periodischen Aufgabe nicht der Periode selbst entsprechen müssen. Anderswo werden solche Aufgaben verschiedentlich als sporadische Aufgaben bezeichnet.



Rückgekoppelte Regelschleife (engl. *feedback control loop*)

initialisiere Stellwert;

initialisiere Zeitgeber und Unterbrecher;

bei Zeitgeberunterbrechung **erledige** /* *abtasten, regeln, steuern* */

A/D-Wandlung der Echtzeitinstanz, Echtzeitabbild ziehen;

Echtzeitdatenbasis aktualisieren, neuen Stellwert berechnen;

D/A-Wandlung des Stellwerts, Echtzeitinstanz verändern;

basta



Die Berechnung von Stellwerten für Aktoren ist eine typische Aufgabe von Echtzeitsystemen

- Das kontrollierte Objekt erfährt eine **direkte digitale Regelung**
 - Regelungsanwendungen zeigen dabei eine hohe **Regelmäßigkeit**
- Meist endlose Sequenz von Regelzyklen



Lassen sich Echtzeitsysteme ausschließlich aus periodischen Aufgaben aufbauen?



■ Periodische Regelungsaufgaben im *I4Copter*:

alle 3 ms Sensorabtastung, Sensordatenfusion

alle 9 ms Fluglageregelung

alle 21 ms Höhenregelung



Die **zeitliche Auflösung** der Regelung richtet sich nach der **Objektdynamik**





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

■ Beispiel Quadrocopter:

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



Alleine die **Anwendung** kann (fast) komplett kontrolliert werden.³

³Lässt man zugelieferte Bibliotheksfunktionen oder zugekaufte Codegeneratoren außer Acht.



- 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



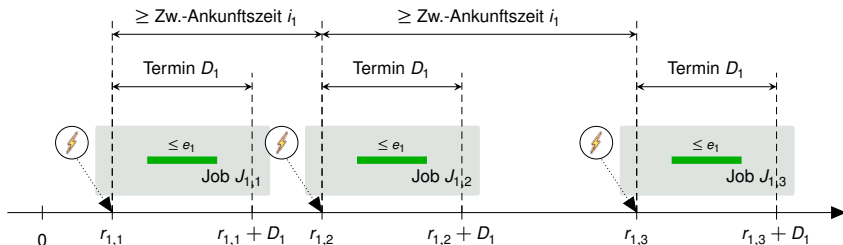


Nicht-periodische Aufgaben (engl. *non-periodic tasks*)

Nicht-periodische Aufgaben

Erbringen in **unregelmäßigen Zeitintervallen** eine vorgegebene Systemfunktion. Jede nicht-periodische Aufgabe (T_i^S) ist eine Abfolge von Arbeitsaufträgen ($J_{i,j}$) mit vorgegebenen zeitlichen Eigenschaften.

- **Weiche/feste** Termine $\mapsto$ **Aperiodische Aufgabe** (engl. *aperiodic task*)
- **Harte** Termine $\mapsto$ **Sporadische Aufgaben** (engl. *sporadic tasks*)
- Besitzen **Minimale Zwischenankunftszeit** (engl. *minimum interarrival-time*) i_j mit $[r_{i,j}; r_{i,j+1}]$ zwischen den Auslösezeiten von T_i^S



Quelle nicht-periodischer Aufgaben

Nicht-periodische Aufgaben behandeln Ereignisse, die sich aus **Zustandsänderungen** des zu kontrollierenden Systems ableiten.

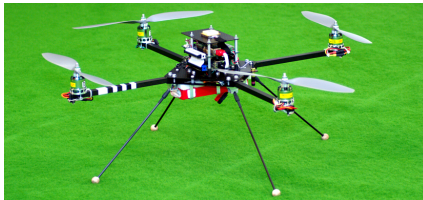
■ Beispiele für Zustandsänderungen:

■ Mensch-Maschine-Interaktion

⚠ Menschliches Verhalten ist kaum quantifizierbar

■ Kommunikation

■ Fehlerbehandlung



Beispiel I4Copter:

■ Steuerkommandos

- Empfang über die Fernbedienung
- Schlimmster Fall: Alle 100 ms

■ Telemetriedaten-Übertragung

- Füllen eines internen Puffers
- Schlimmster Fall: Alle 9 ms

Behandlung nicht-periodischer Aufgaben

Grundlegende Behandlungsmethoden für nicht-periodische Ereignisse lassen sich mit minimaler **Unterstützung des Laufzeitsystems** umsetzen. Sie sind sowohl für zeit- als auch für ereignisgesteuerte Systeme geeignet und teilweise vollständig auf Anwendungsebene umsetzbar.

- **Unterbrecherbetrieb** $\leadsto$ **Nicht-periodische Aufgaben haben Vorfahrt**
 - Ereignisbehandlung direkt in der Unterbrechungsbehandlung
 - Mittels **Ausnahmebehandlungen**
- **Hintergrundbetrieb** $\leadsto$ **Periodische Aufgaben haben Vorfahrt**
 - Phasen der Untätigkeit für nicht-periodische Aufgaben nutzen
 - Mittels **Verdrängung**
- **Periodischer Zusteller** $\leadsto$ **Alles ist eine periodische Aufgabe**
 - Abfragen nicht-periodische Ereignisse durch periodische Aufgaben
 - **Einphasen** nicht-periodischer Aufträge mit bekannten Mitteln



- 1 Problemstellung
 - Kontrolliertes Objekt
 - Realzeitbetrieb
 - Termine
- 2 Allgemeine Grundlagen
 - Begriffsdefinition: Echtzeitrechensystem
 - Terminologie
 - Programmunterbrechungen
 - Verdrängbarkeit
- 3 Echtzeit-Aufgaben
 - Grundsätzliche Umsetzungsalternativen
 - Periodische Aufgaben
 - Nicht-periodische Aufgaben
- 4 Umsetzung und Ablaufplanung
 - Zeitgesteuerte Ausführung
 - Ereignisgesteuerte Ausführung
 - Rangfolge





Die *Busy Loop*

Die wirklich einfachste Variante für die Implementierung zyklischer Systeme?



Periodische Aufgaben wiederholt in einer Schleife ausführen

```
int main(void) {  
    unsigned long cnt = 0;  
    while(1) {  
        warte_durchlauf();  
        kontrolle_start();  
        aufgabe1();  
        kontrolle_stop();  
        if(cnt % 2 == 0) {  
            aufgabe2_1();  
        }  
        10ms_nach_aufgabe1();  
        if(cnt % 2 == 0) {  
            aufgabe2_2();  
        }  
        ++cnt;  
    }  
    return 0;  
}
```

- Längere Perioden lassen sich durch einen **Rundenzähler** ableiten
 - die Schleife definiert einen **Rahmen**
 - Ausrichtendes Raster für **alle Aktivitäten**
- Explizite Überwachung der **Rahmendauer**
 - Ausführungszeit ist i.d.R. **nicht konstant**
- Schwierige Spezifikation **zeitlichen Versatzes**
 - Abhängigkeit von der **tats. Ausführungszeit**
- Konflikte durch **lange andauernde Aufträge**
 - Evtl. ist eine **manuelle Aufteilung** nötig
- **Überwachung** der Ausführungszeit
 - Schwieriger **Abbruch** des betroffenen Auftrags





Vorberechneter (statischer) Ablaufplan $\mapsto$ **Ablauftabelle**

- Jeder Tabelleneintrag entspricht einer Einplanungsentscheidung zu einem (vorab) bestimmten Zeitpunkt auf der Echtzeitachse
 - Bei Einlastung wird ein **Zeitgeber** (engl. *timer*) programmiert und der Arbeitsauftrag wird gestartet
 - Kurzzeitwecker auf nächsten Entscheidungszeitpunkt stellen
 - Einzustellender Wert ist im aktuellen Tabelleneintrag zu finden
- Ein **Zeitgebersignal** schaltet zum nächsten Tabelleneintrag weiter



Am Tabellenende wird wieder zum -anfang gesprungen

- **Zyklischer Ablaufplan** (engl. *cyclic schedule*) periodischer Aufgaben
- Die **Hyperperiode** (siehe Folie ??) gibt die Tabellengröße vor





Einplanung von Arbeitsaufträgen erfolgt zu **Ereigniszeitpunkten**

- Ihr Auftreten ist nicht (exakt) vorhersehbar
- Ereignisauslöser sind kontrollierte Objekte/andere Arbeitsaufträge
- Die Ereignisverarbeitung unterliegt einer gewissen **Dringlichkeit**



Ereignisse haben Prioritäten die dem Ereignisauslöser und/oder der Ereignisverarbeitung zugeordnet sind

Feste Zuordnung → Ereignisverarbeitung/-auslöser

- Arbeitsaufträge erhalten **absolute Priorität**

Variable Zuordnung → Ereignisverarbeitung

- Arbeitsaufträge erhalten **relative Priorität**

Auch **prioritätsorientierte Einplanung** (engl. *priority-driven scheduling*)





Verfahren zur **prioritätsorientierten Einplanung** periodischer Arbeitsaufträge werden folglich in zwei Gruppen eingeteilt:

Feste Priorität (engl. *fixed priority* oder *static priority*)

- Priorität der Aufträge einer Aufgabe sind **unveränderlich**
- Die Aufgabenpriorität steht unabhängig von der Auslösung bzw. Beendigung von Arbeitsaufträgen fest
- Prioritäten werden **statisch zum Entwurfszeitpunkt** vergeben

Dynamische Priorität (engl. *dynamic priority*)

- Priorität der Aufträge einer Aufgabe sind **veränderlich**
- Aufgabenpriorität variiert relativ zu anderen Aufgaben, wenn Arbeitsaufträge ausgelöst bzw. beendet werden
- Prioritäten werden **dynamisch zur Laufzeit** vergeben





Einplanung gemäß Ausführungsrate

Rate $1/p_i$ einer Aufgabe T_i ist die Inverse ihrer Periode p_i

- Bezogen auf die Auslöserate von Arbeitsaufträgen in T_i

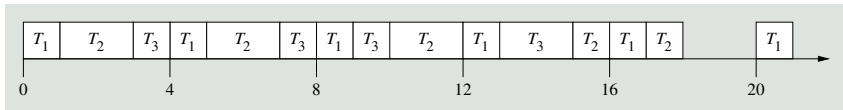
→ Je kleiner die Periode, desto höher die **Priorität** P_i von T_i

- **Beispiel:** $T_1 = (4, 1)$, $T_2 = (5, 2)$, $T_3 = (20, 5)$

- Perioden $p = \{4, 5, 20\}$, Ausführungszeiten $e = \{1, 2, 5\}$

⚠ Termin und Phase optional bei $D_i = p_i$ und $\phi_i = 0$

- **Ablaufplan:**



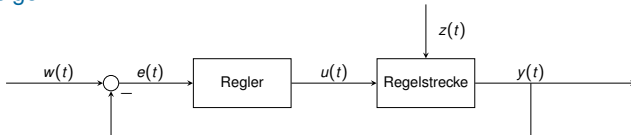
Arbeitsaufträge werden in ihren Aufgabenperioden ausgeführt

→ RM lässt Prozessor bei ausführerbereiten Aufträgen nicht untätig





Ausführung von Arbeitsaufträgen unterliegt häufig einer **bestimmten Reihenfolge**
→ **Rangfolge**



- Beispiel: **Regelungsanwendung**
 - Signalverarbeitungsauftrag muss vor der Regelung gelaufen sein
 - Beispiel: **Kommunikationssystem**
 - Sendeauftrag muss vor Empfangsauftrag gelaufen sein
 - Empfangsauftrag muss vor Bestätigungsauftrag gelaufen sein
 - Beispiel: **Anfragesystem**
 - Eingabeauftrag muss vor Suchauftrag gelaufen sein
 - Suchauftrag muss vor Ausgabeauftrag gelaufen sein
- Rangfolge ist oft in **Datenabhängigkeiten** begründet





Arbeitsaufträge benötigen ggf. **konsumierbare Betriebsmittel**

- Anzahl ist (log.) unbegrenzt: Nachrichten, Signale, Interrupts

Produzent kann beliebig viele davon erzeugen

Konsument zerstört sie wieder bei Inanspruchnahme

→ Zwischen ihnen besteht eine **gerichtete Abhängigkeit**



Produzent und Konsument sind voneinander **abhängige Entitäten**

- Abhängigkeit: Konsument → Produzent

- Betriebsmittel muss vor Inanspruchnahme zunächst bereitgestellt werden

- Abhängigkeit: Produzent → Konsument (seltener)

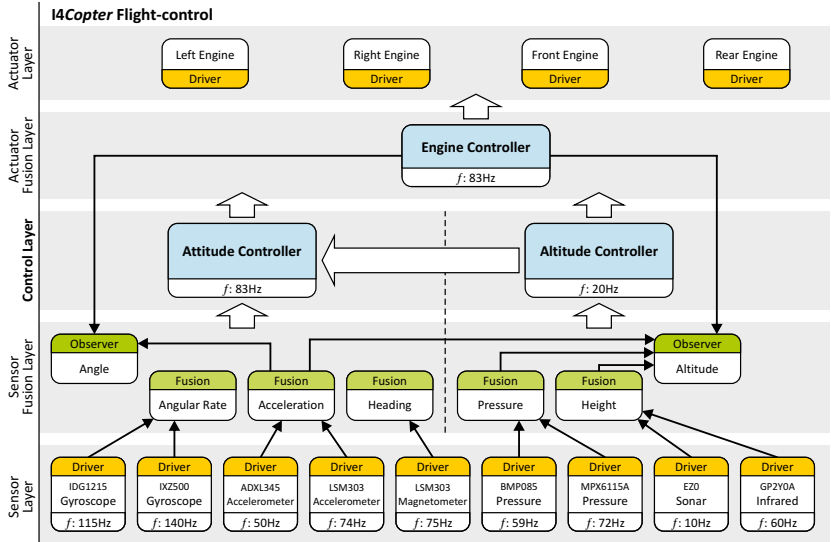
- Abbildung **konsumierbare** ↦ **wiederverwendbare Betriebsmittel**

- Beispiel: **begrenzter Puffer** (engl. *bounded buffer*)

- Produzent fordert ein wiederverwendbares Betriebsmittel an, welches vom Konsumenten später wieder freizugeben ist



Datenabhängigkeiten im I4Copter





Übergang zwischen zeitlichen Domänen

Produzenten und Konsumenten werden mit unterschiedlichen Raten aktiviert



Koordinierung verschiedener zeitlicher Domänen (vgl. Folie 38)

- Unterschiedliche Raten in den Bereichen des Echtzeitsystems

→ Gerichtete Abhängigkeiten erfordern **Angleichung**



Datenaustausch zwischen Produzent und Konsument

- Erfolgt in Abstimmung → Konsument erwartet Daten
- Aufwand abhängig von der Diskrepanz der Raten



Typisches Vorgehen in Echtzeitanwendungen

- **Gemeinsamer Puffer** als Zwischenspeicher → Produzent schneller
 - Problem: Puffergröße und WCET (Abarbeitung des Rückstands)
- **Prädikation** durch Beobachter → Konsument schneller⁴
 - Generierung von Zwischenwerten kompensiert langsamen Produzenten
- **Letzter Wert genügt** (engl. *last is best*) → beidseitig
 - Verzicht auf explizite Abstimmung (**simpel**)
 - **Alter unterliegt gewissen Schwankungen**

⁴Sonderfall in der digitalen Signalverarbeitung: Zukünftige Messwerte lassen sich mittels Modellen des physikalischen Systems in gewissem Umfang vorhersagen.



- [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] Liu, J. W. S.:
Real-Time Systems.
Englewood Cliffs, NJ, USA : Prentice Hall PTR, 2000. –
ISBN 0-13-099651-3
- [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

