Lecture 1

[RTCS Ch. 1 & 2]

- Real-Time System Definitions
- Real-Time System Characteristics
- Real-Time System Paradigms

Real-Time Systems

"any information processing system which has to respond to externally generated input stimuli within a finite and specified period."

"real-time systems are those in which the correctness of the system depends not only on the logical results of the computation but also on the time at which the results are produced"

Definitions

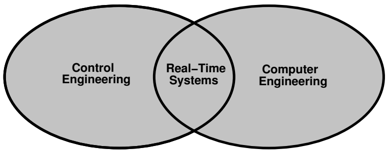
A *hard real-time system* is a system where it is absolutely imperative that the responses occur within the required deadline.

- safety-critical applications
- e.g., aerospace, automotive, ...

A *soft real-time system* is a system where deadlines are important but where the system still functions if the deadlines are occasionally missed.

- e.g. multimedia, user interfaces, ...

Real-Time and Control



A Natural Connection

- All control systems are real-time systems
- Many hard real-time systems are control systems

Real-Time and Control

- Control engineers need real-time systems to implement their systems.
- Computer Engineers need control theory to build 'controllable' systems
- Interesting research problems in the interface

Hard Real-Time Systems

The focus of this course.

Many (most??) hard real-time systems are real-time control systems.

Most real-time control systems are **not** hard real-time systems.

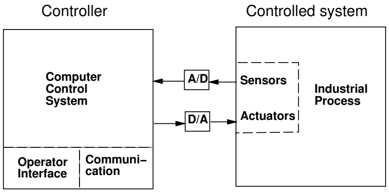
Many hard real-time systems are safety-critical.

Common misconception:

- real-time = high-speed computations
- not true
- execute at a speed that makes it possible to fulfill the timing requirements

Real-Time Control Systems

Many real-time systems are real-time control systems.



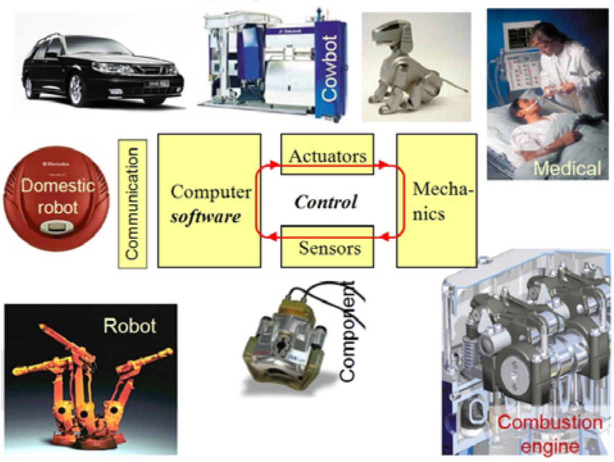
- control algorithms
- process presentation
- operator communication
- data communication

Real-Time Control Systems

Two types of real-time control systems:

1. Embedded Systems
   - dedicated control systems
   - the computer is an embedded part of some piece of equipment
   - microprocessors, real-time kernels, RTOS
   - aerospace, industrial robots, vehicular systems, ...
2. Industrial Control Systems
   - distributed control systems (DCS), programmable logic controllers (PLC), Soft-PLCs
   - hierarchically organized, distributed control systems
   - process industry, manufacturing industry, ...

Products relying on embedded control

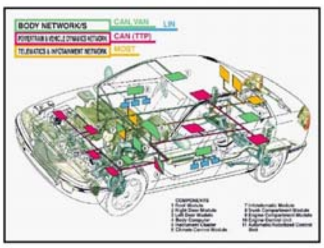


Some more ....

Example: Modern Cars

- **Embedded control systems in modern car** (brakes, transmission, engine, safety, climate, emissions, ...)

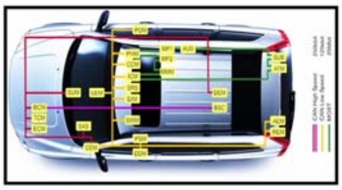
40-100 ECUs in a new car
~ 2-5 miljon lines of code



Example: Modern Cars

- Networked Systems

Volvo XC 90:
- *3 CAN-buses*
- *+ other buses*



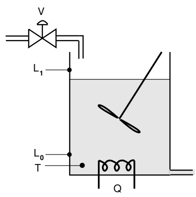
Embedded Control Characteristics

- Limited computing and communication resources
  - Often mass-market products, e.g., cars
  - CPU time, communication bandwidth, energy, memory, ...
- Autonomous operation
  - No human “operator” in the loop
  - Several use-cases and complex functionality
    - Often large amounts of software
  - Need for formal guarantees

Embedded Control Characteristics

- Limited resources ⇒ Efficiency
  - Code-size efficiency
  - Run-time efficiency
  - Energy efficiency
  - Weight and size efficiency
  - Cost efficiency
- Autonomous operation ⇒ Dependability
  - Reliability
  - Availability
  - Safety
  - Security
  - Maintainability

A Typical Control Problem: The Buffer Tank



Raw material buffer + heating

Goals:

- Level control: open V when level below L_0, keep the valve open until level above L_1
- Temperature control: PI-controller

Typical Characteristics

- Parallel activities.
- Timing requirements – more or less hard.
- Discrete and analog signals.
- Continuous (time-driven) control and Discrete (event-driven), sequential Control

All control systems have these characteristics.

Continuous Time-Driven Control

Controller on continuous (analog form)

- e.g. PI-controller

$$u(t) = K((y_{ref}(t) - y(t)) + \frac{1}{T_i}\int^t (y_{ref}(\tau) - y(\tau))d\tau)$$

Can be implemented in several ways, e.g., using analog electronics

Here, we will assume that it is implemented using a computer.

How, should this be done?

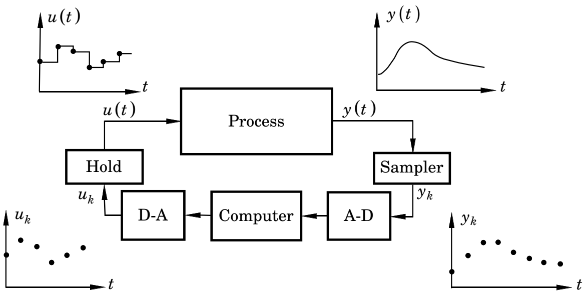
Sampling - Control - Actuation

Frequently:

- Sampling of measured signal $y(t)$
- Calculation of control signal (software algorithm)
- Actuation of calculated control signal $u(k)$

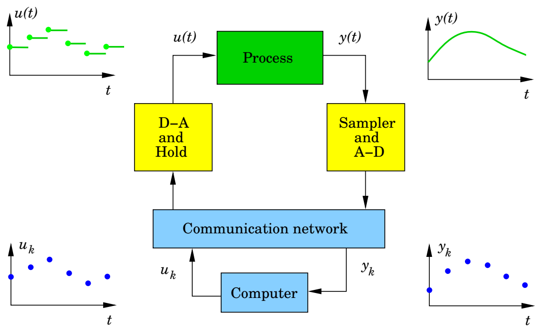
In most cases periodically, i.e. driven by a clock (time)

Sampled-data control systems



- Mix of continuous-time and discrete-time signals
- Discretization in time and in space

Networked control systems



- Extra delay, possibly lost packets

Design Approaches

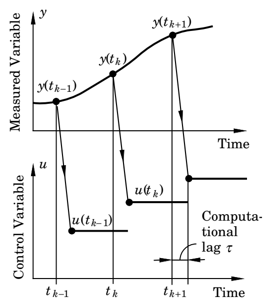
Sampled control-design:

- Discrete-time design
- Use a model of the plant that only describes the behaviour at the sampling instants – sampling the system

Approximation of a continuous-time design

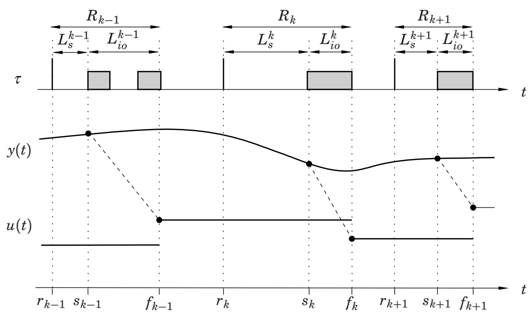
- Design the controller assuming a continuous-time implementation
- Approximate this controller by a discrete-time controller

Ideal Controller Timing



- Output $y(t)$ sampled periodically at time instants $t_k = kh$
- Control $u(t)$ generated after short and constant time delay τ

Real Controller Timing



- Control task τ released periodically at time instances $r_k = kh$
- Output $y(t)$ sampled after time-varying **sampling latency** L_s
- Control $u(t)$ generated after time-varying **input-output latency** L_{io}

Non-Deterministic Timing

Caused by sharing of computing resources

- multiple tasks sharing the CPU
- preemptions, blocking, priority inversion, varying computation times, ...

Caused by sharing of network bandwidth

- control loops closed over communication networks
- network interface delay, queuing delay, transmission delay, propagation delay, resending delay, ACK delay, ...
- lost packets

How can we minimize the non-determinism?

How does the non-determinism effect control performance?

Discrete Event-Driven Control

Event-driven:

- wait for a condition to become true or an event to occur
- perform some actions
- wait for some new conditions
- ....

The event can be a clock-tick

Often modeled using state machine/automata-based formalisms

In many cases implemented using periodic sampling

Events

Real-Time systems must respond to events.

- Periodic events
- Non-periodic events
  - aperiodic events
    - unbounded arrival frequency
  - sporadic events
    - bounded arrival frequency

Events can be external or internal.

Each event requires a certain amount of processing and has a certain deadline.

Parallelism

The real world is parallel

Events may occur at the same time.

The work that has to be done to service an event is called the task associated with the event.

It is often natural to handle the different tasks independently during design.

Temperature Loop

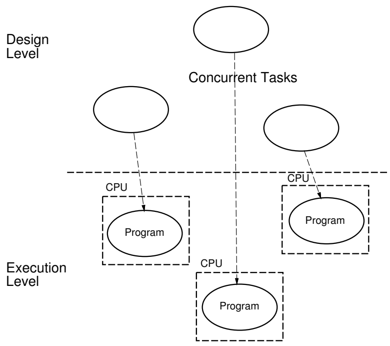
```
while (true) {
  Measure temperature;
  Calculate temperature error;
  Calculate the heater signal with PI−control;
  Output the heater signal;
  Wait for h seconds;
}
```

Level Loop

```
while (true) {
  Wait until level below L0;
  Open inlet valve;
  Wait unitil level above L1;
  Close inlet valve;
}
```

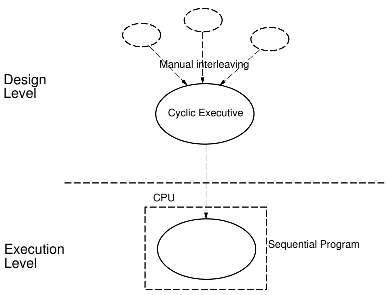

Paradigms

Parallel (multi-core) programming:



Paradigms

Sequential programming:



Interleaved temperature and level loops

```
while (true) {
 while (level above L0) {
  Measure temperature;
  Calculate temperature error;
  Calculate the heater signal with PI−control;
  Output the heater signal;
  Wait for h seconds;
 }
 Open inlet valve;
 while (level below L1) {
  Measure temperature;
  Calculate temperature error;
  Calculate the heater signal with PI−control;
  Output the heater signal;
  Wait for h seconds;
 }
 Close inlet valve;
}
```

Complex and non user-friendly code

Can, however, often be automated.

Static Sequential Approaches

Advantages:

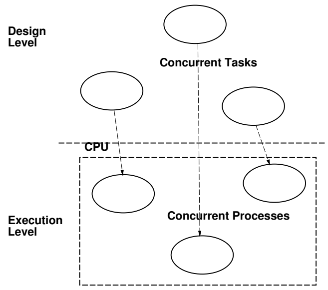
- determinism
- a lot of different constraints can be ensured
- simple real-time computing platforms may be used

Disadvantages:

- inflexible
- generation of the sequential process can be a difficult optimization problem

Paradigms

Concurrent programming:



The CPU is shared between the process (switches)

Real-Time Operating Systems:

- switches between processes
  - Real-Time Kernel
- timing primitives
- process communication

Non real-time OS:

- polling (inefficient)

Polled Temperature Loop

```
        while (true) {
          Measure temperature;
          Calculate temperature error;
          Calculate the heater signal with PI−control;
          Output the heater signal;
          counter = 0;
Polling | while (counter < hcount) {
        |   INC(counter);
        | }
        }
```


Real-time OS:

- timing primitives and interrupts
- CPU free to service other tasks

Temperature Loop with Sleep

```
while (true) {
  Measure temperature;
  Calculate temperature error;
  Calculate the heater signal with PI−control;
  Output the heater signal;
  Sleep(h);
}
```


Real-Time System Characteristics

- timing requirements
- must be deterministic and predictable
- worst-case response times of interest rather than average-case
- large and complex
- distributed
- tight interaction with hardware
- safety critical
- execution is time dependent
- testing is difficult
- operating over long time periods

Real-Time Systems Course

In this course, as in most of industry, we will follow the concurrent programming paradigm.

Two different environments will be used during the lectures:

- Java
  - concurrency through Java threads
  - language used in projects
- STORK
  - real-time kernel implemented in Modula-2
  - close in nature to commercial real-time kernels and real-time operating systems (OS)
  - makes it possible to teach how a real-time kernel is implemented

Java in Real-Time – NO

- Java was not developed for real-time applications.
- The just-in-time compilation in Java and the dynamic method dispatching makes Java non-deterministic and slow.
- The automatic garbage collection makes Java execution non-deterministic.
- Java lacks many important real-time primitives.

Java in Real-Time – YES

- a nice concurrent programming language
- a nice object-oriented language
- a nice teaching language
- strong trends towards Real-Time Java
- many of the shortcomings of Java can be handled, e.g., the garbage collection problem
- Microsoft's .NET and C# (a Java clone) will strongly increase the industrial use in the near future