Introduction to Real-Time Systems

Real-Time Systems, Lecture 1

Martina Maggio and Karl-Erik Årzén
21 January 2020
Lund University, Department of Automatic Control

Content

[Real-Time Control System: Chapter 1, 2]

1. Real-Time Systems: Definitions
2. Real-Time Systems: Characteristics
3. Real-Time Systems: Paradigms

Real-Time Systems: Definitions

Real-Time Systems

"Any information processing system which has to responde to externally generated input stimuli within a finite and specified period"

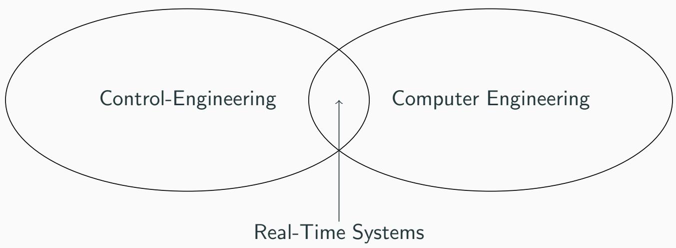
"Real-Time systems are those in which the correctness of the system depends not only on the logical results of the computation but also on the time at which results are produced"

Definitions

A *hard real-time* system is a system where it is absolutely imperative that the responses occur within the required deadline (for example because in *safety-critical applications* in aerospace, automotive and so on).

A *soft real-time* system is a system where deadlines are important, but where the system still functions if the deadlines are occasionally missed (for example in multimedia systems, user interfaces and so on).

Real-Time and Control



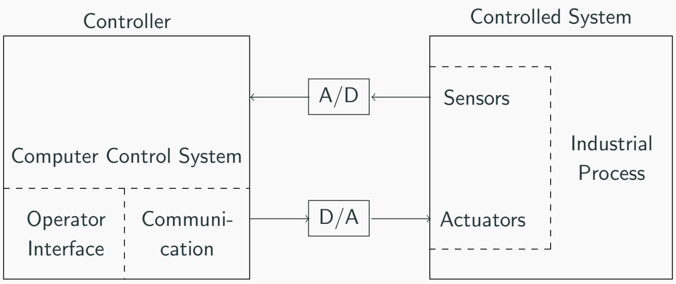
- All control systems are real-time systems.
- Many hard real-time systems are control systems.

- 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 safely-critical.
- Common misconception: Real time equals high-speed computations. This is not true. Real-time systems execute at a speed that makes it possible to fulfill the timing requirements.

Real-Time Control Systems

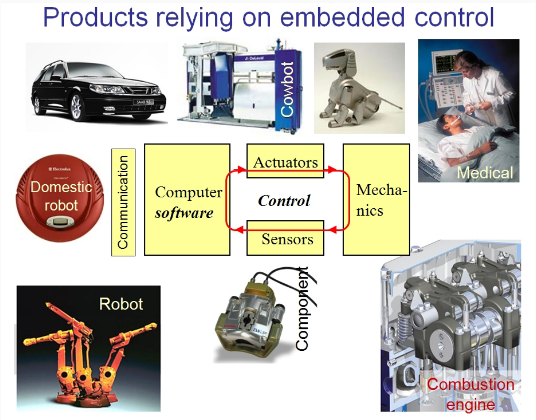


Real-Time Control Systems

Two types of real-time control systems:

- Embedded systems:
  - dedicated control system
  - the computer is an embedded part of some equipment
  - microprocessors, real-time kernels, RTOS
  - aerospace, industrial robots, vehicular systems
- Industrial control systems:
  - distributed control systems (DCS)
  - programmable logic controllers (PLC)
  - hierarchically organized
  - process industry, manufacturng industry

Example



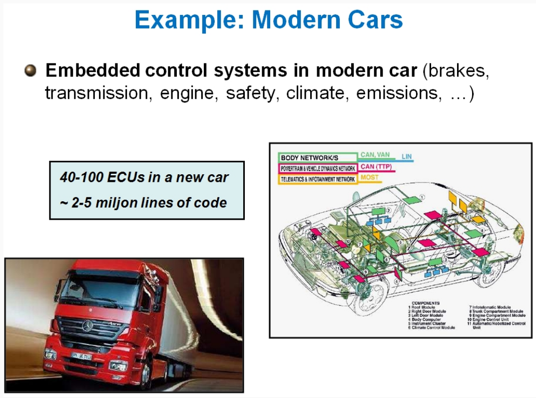
Some more

Example

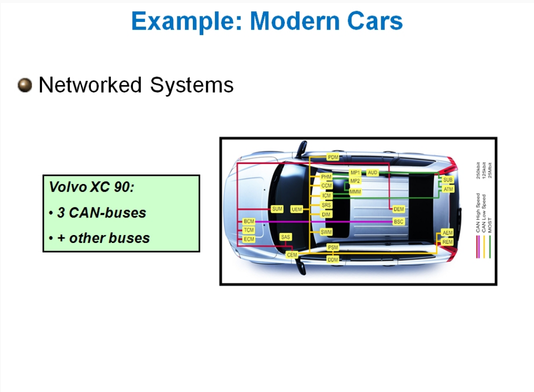
In a modern car

- **Embedded control systems**: brakes, transmission, engine, safety, climate, emissions: 40-100 ECUs in a new car, 2-5 milion lines of code;
- **Networked systems**: VOLVO XC 90 has 3 CAN-buses and other buses.

Networked Control

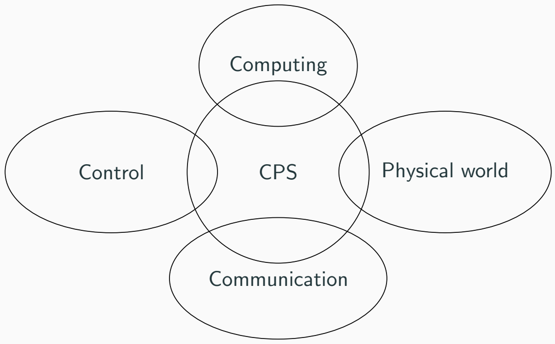


Networked Control



Cyber-Physical Systems

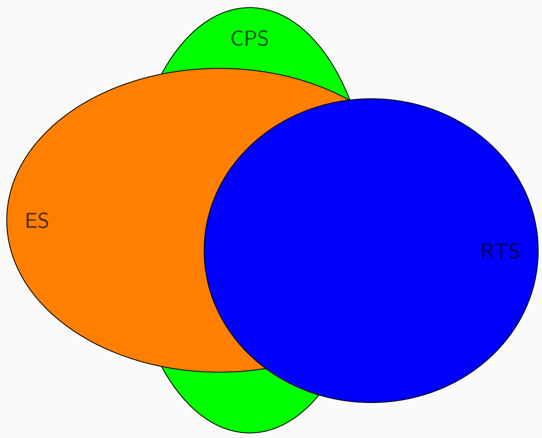
- Name coined in the US around 2008.
- Denotes systems with a very tight connection between computing, communication, control and the physical world.



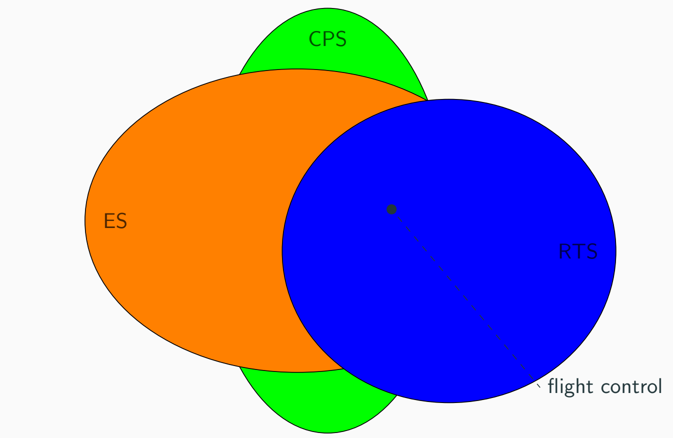
CPS Examples

- Smart/Green/Low-energy buildings: require interaction between architects, mechanical engineers and control engineers; require interaction between a number of subsystems (cooling, lightning, security);
- Green cars;
- Smart Power Grids;
- Server Farms/Data Centers: require interaction between load balancing and energy consumption;
- Battery-driven computing and communication devices (like smart phones, laptops and sensor networks);
- Cross-layer design and optimization in networks: in embedded systems resource-aware design.

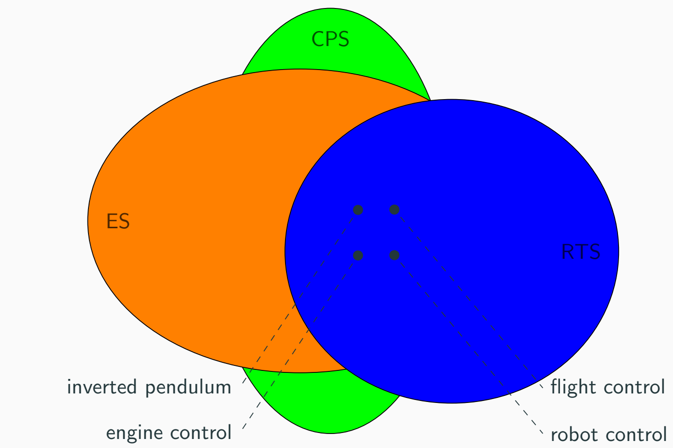
Systems: Embedded, Cyber-Physical, Real-Time



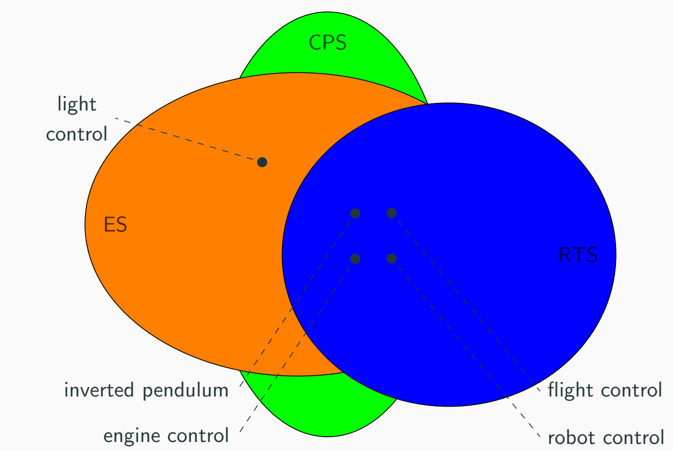
Systems: Embedded, Cyber-Physical, Real-Time



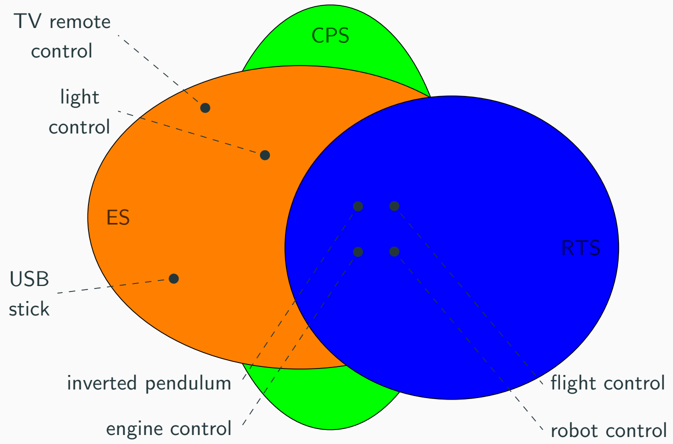
Systems: Embedded, Cyber-Physical, Real-Time



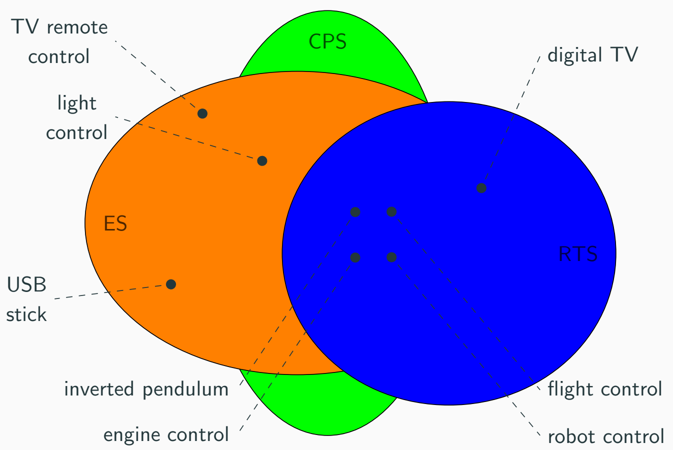
Systems: Embedded, Cyber-Physical, Real-Time



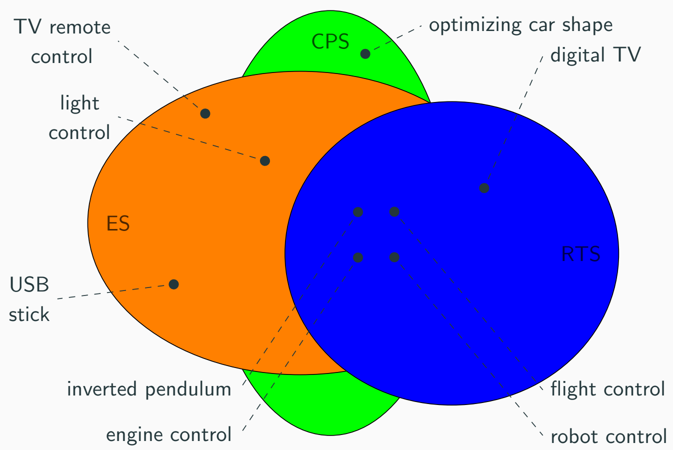
Systems: Embedded, Cyber-Physical, Real-Time



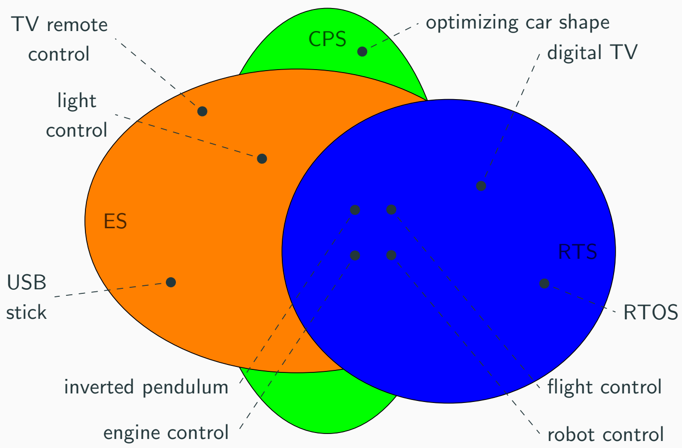
Systems: Embedded, Cyber-Physical, Real-Time



Systems: Embedded, Cyber-Physical, Real-Time



Systems: Embedded, Cyber-Physical, Real-Time



Real-Time Systems: Characteristics

Embedded Control Characteristics

- Limited computing and communication resources:
  - often mass-market products, like cars
  - CPU time, communication bandwidth, energy, memory
- Autonomous operation:
  - No human operator in the loop
  - Several use-cases and complex functionality, often large amount of software
  - need for formal guarantees

Embedded Control Characteristics

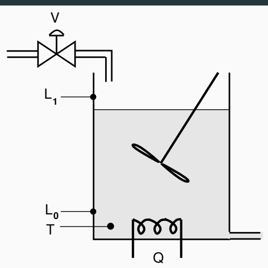
Limited Resources $\Longrightarrow$ Efficiency

- Code-size efficiency
- Run-time efficiency
- Energy efficiency
- Weight and size efficiency
- Cost efficiency

Autonomous operations $\Longrightarrow$ Dependability

- Reliability
- Availability
- Safety
- Security
- Maintainability

Example: The Buffer Tanks



Raw Material and 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.

Continuous Time-Driven Control

Controller on continuous (analog form)

- 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?

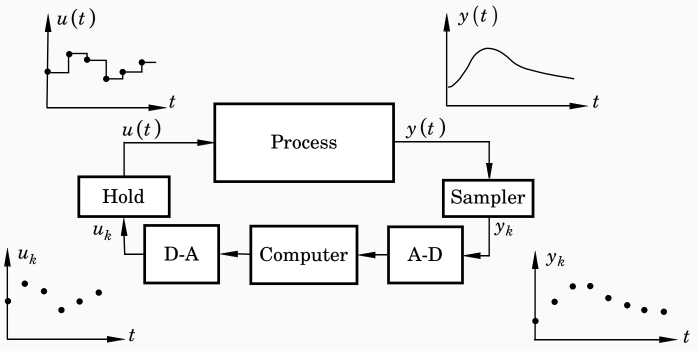
Sampling - Control - Actuation

Frequently:

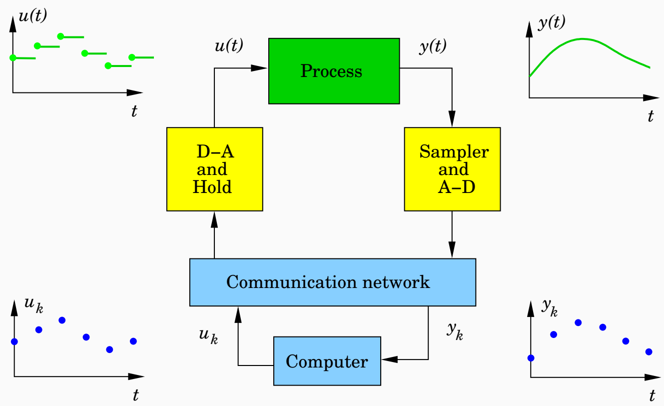
- Sampling of measured signal $y(t)$,
- Calculation of control signal (software algorithm),
- Actuation of calculated control signal $u(k)$.

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

Sampled-Data Control Systems



Networked Control Systems



Design Approaches

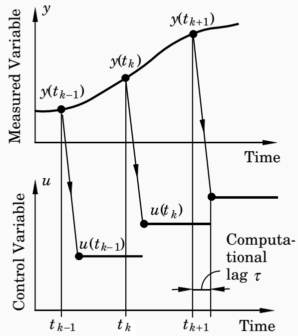
Sampled control-design:

- 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:

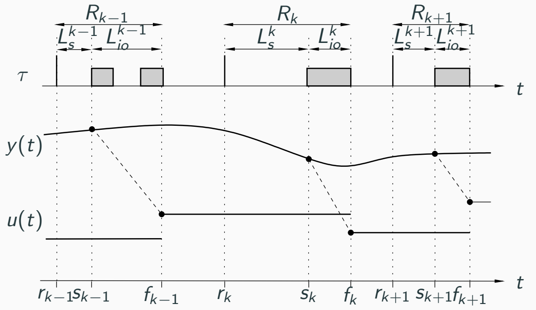
- Design the controller assuming a continuous-time implementation,
- Approximate this controller by a discrete-time controller.

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, and so on.

Caused by sharing of network bandwidth:

- control loops closed over communication networks,
- network interface delay, queuing delay, transmission delay, propagation delay, re-sending 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 formalism.

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.

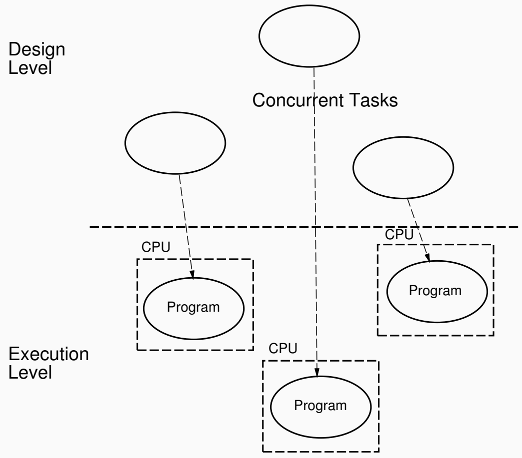
Events may occur at the same time.

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

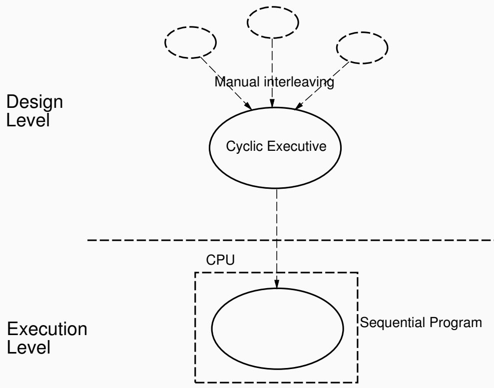
It is often natural to handle the different tasks independently during design.

Real-Time Systems: Paradigms

Parallel Multi-Core Programming



Sequential Programming



Interleaved Code

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 often be automated.
Sequential program = static schedule (cyclic executive).

Static Sequential Approaches

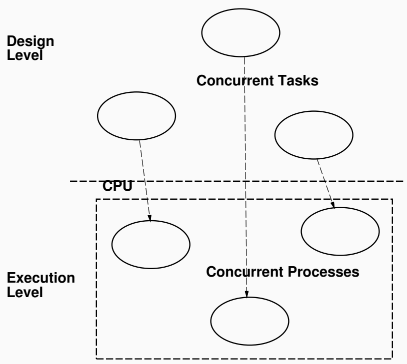
Advantages:

- 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.

Concurrent Programming



The CPU is shared between the processes (switches).

Real-Time Operating Systems

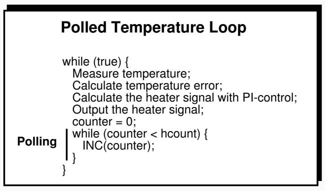
- Switches between processes (real-time kernel),
- Timing primitives and interrupts,
- Process communication,
- CPU free to service other tasks.

Temperature Loop with Sleep

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

Non-Real-Time Operating Systems

- Polling (inefficient).



Real-Time Systems 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) + Google's Android has strongly increased the industrial use of Java.