Real-Time Systems
The Basics

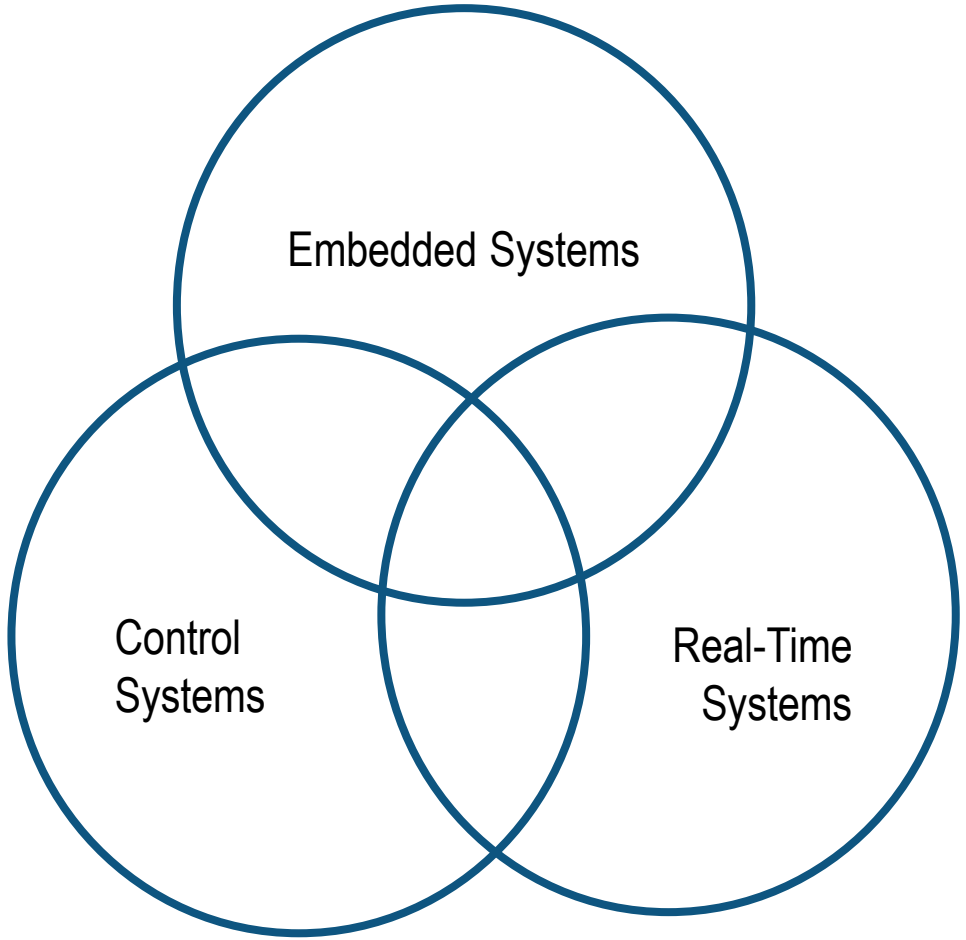
Dr. ASHRAF E. SUYYAGH

THE UNIVERSITY OF JORDAN

DEPARTMENT OF COMPUTER ENGINEERING

SPRING 2020

Introduction

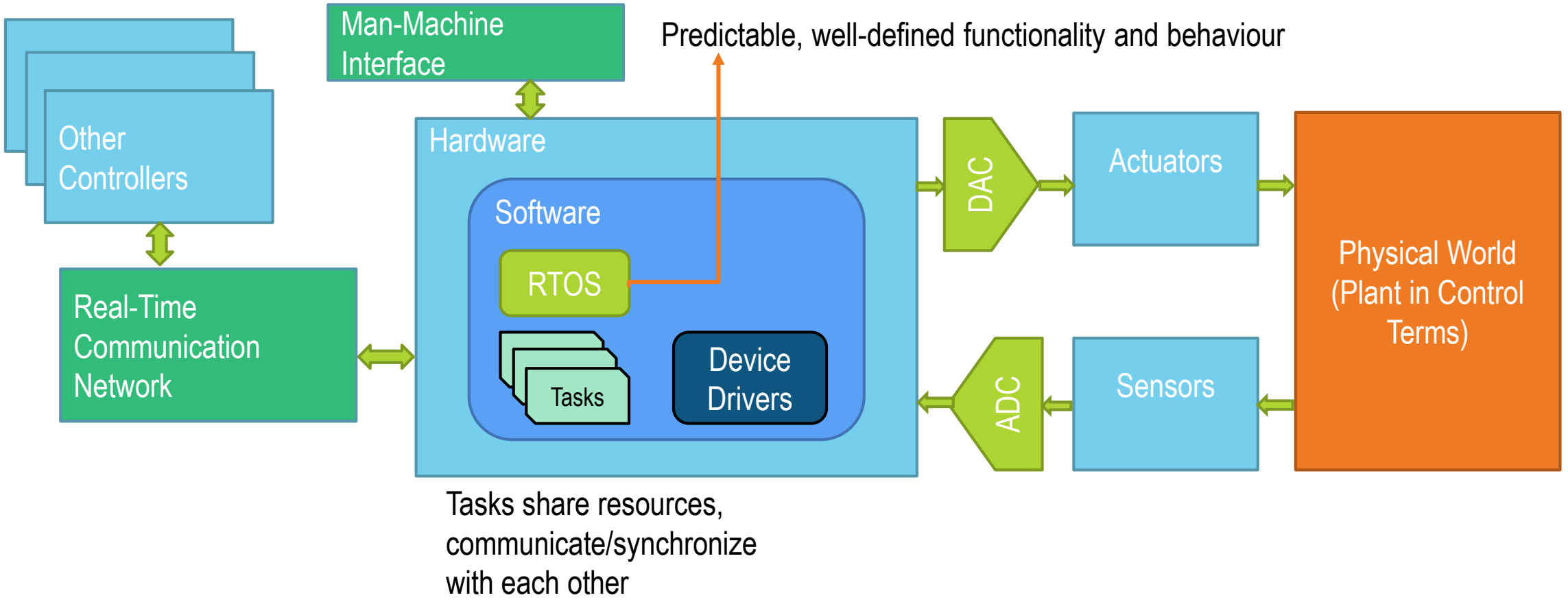


Not necessarily real-time, nor advanced control systems (dynamic or adaptive or robust)

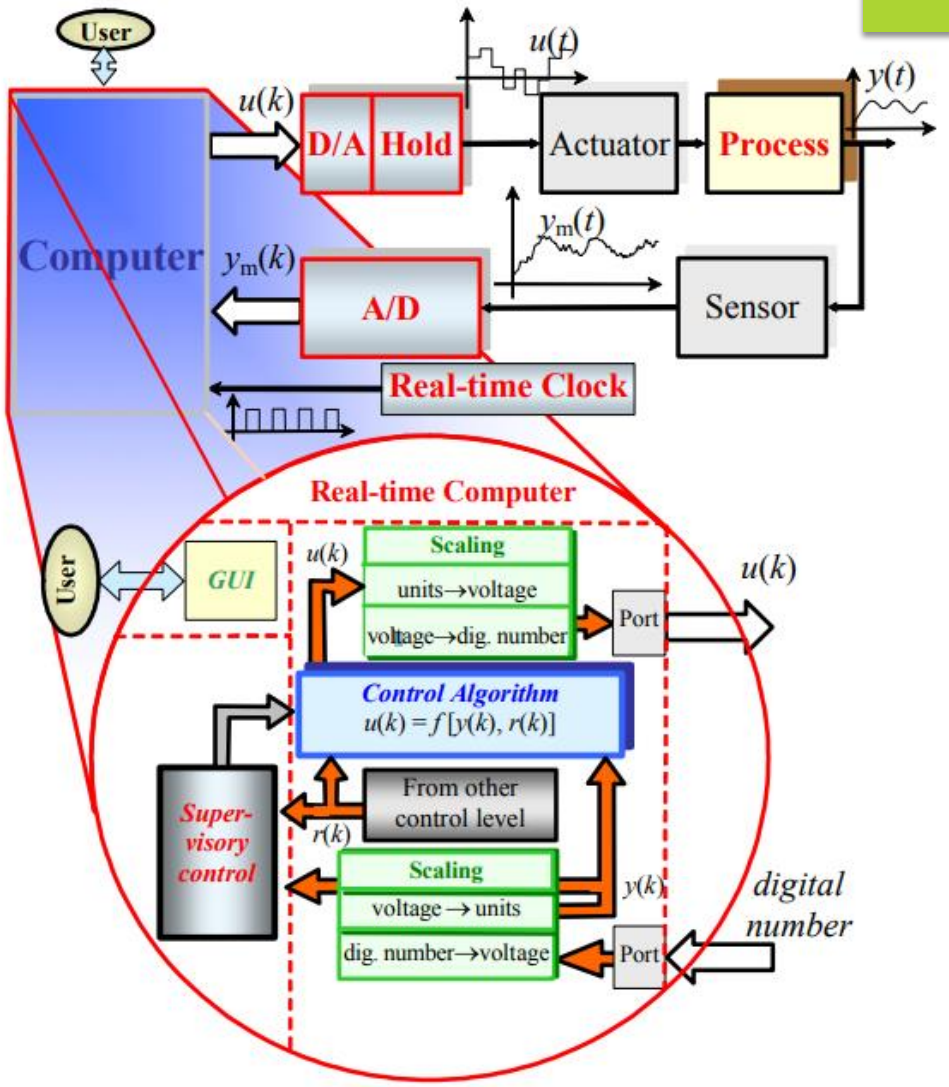
Classical Control Theory, Modern Digital Control (In the past, not necessarily digital). Not necessarily embedded.

Not necessarily control or embedded, but applications with timing constraints (e.g. video frame processing). In some instances, subject to QoS.

Overview of a Real-Time Control System (1)



Overview of a Real-Time Control System (2)



Graph Reference:
A. Gambier, "Real-time control systems: a tutorial," *2004 5th Asian Control Conference (IEEE Cat. No.04EX904)*, Melbourne, Victoria, Australia, 2004, pp. 1024-1031 Vol.2

The Basics

General Purpose Systems:

1. Operations are not subject to **performance constraints**.
2. There may be **desirable response characteristics**, but there are **no hard deadlines** and no **detrimental consequences** other than perhaps **poor quality of service** if the response times are unusually long.
3. OS is responsible for managing the hardware resources of a computer and hosting applications that run on the computer.

Real-time systems:

1. The time at which a response is delivered is as important as the correctness of that response, and
2. The consequences of a late response are just as hazardous as the consequences of an incorrect response. **Delays may prove dangerous or even catastrophic (i.e. loss of life)**.
3. RTOS performs these tasks, but is also specially designed to run applications with very precise timing and a high degree of reliability

- We are dealing with the notion of the **timeliness** of the system → In some systems, tasks (processes/threads) should not be started before certain time (e.g. dependencies)
- **Important** → Deadline does not necessarily imply "imminency". How?
- *Real-time* systems are meant to monitor, interact with, control, or respond to the physical environment. How are they different from embedded systems? Cyber-physical systems?

Real-Time Systems Classification by Deadline Type (1)

Timing Requirements (e.g. deadlines) are categorized into**:**

1. Absolute: response must occur at defined deadlines.
2. Relative: response must occur within a specified period of time following an event.

Real-Time Systems Classification by Deadline Type:

1. **Hard** Real-Time Systems
2. **Firm** Real-Time Systems
3. **Soft** Real-Time Systems

Real-Time Systems Classification by Deadline Type (2)

1. Hard Real-Time (HRT) Systems

→ Under all circumstances, ALL 'hard (critical)' tasks MUST meet ALL their deadlines.

→ If not, system failure causes catastrophe or death.

→ Imperative that responses occur within the required deadline.

→ Response after deadline has no value!

- Guaranteed services required → functional correctness and timing correctness.
- Hard Real-Time Systems must be PREDICTABLE and DETERMINISTIC
- Analysis of estimated worst-case time → Scheduling algorithm and system must pass schedulability test
- In practice, the time bounds for HRT ranges from microseconds to milliseconds.
- Hard real-time task does not need to be completed within the shortest time possible (fast computing) → Only within the bound

Real-Time Systems Classification by Deadline Type (3)

2. Firm Real-Time Systems

→ Tasks missing their deadline **will not** result in a system failure, but not catastrophe or death

→ Infrequent misses lead to ***performance degradation*** **(loss of QoS)**

→ Response following a deadline has no value

3. Soft Real-Time Systems

→ Deadlines desired to be enforced, but they are not strict. (Best-effort service → deals with average response times)

→ Frequent deadline misses do not cause errors, but the result of the task **might** no longer be as useful.

→ Response following a deadline is not wasted, but degrades as more time passes

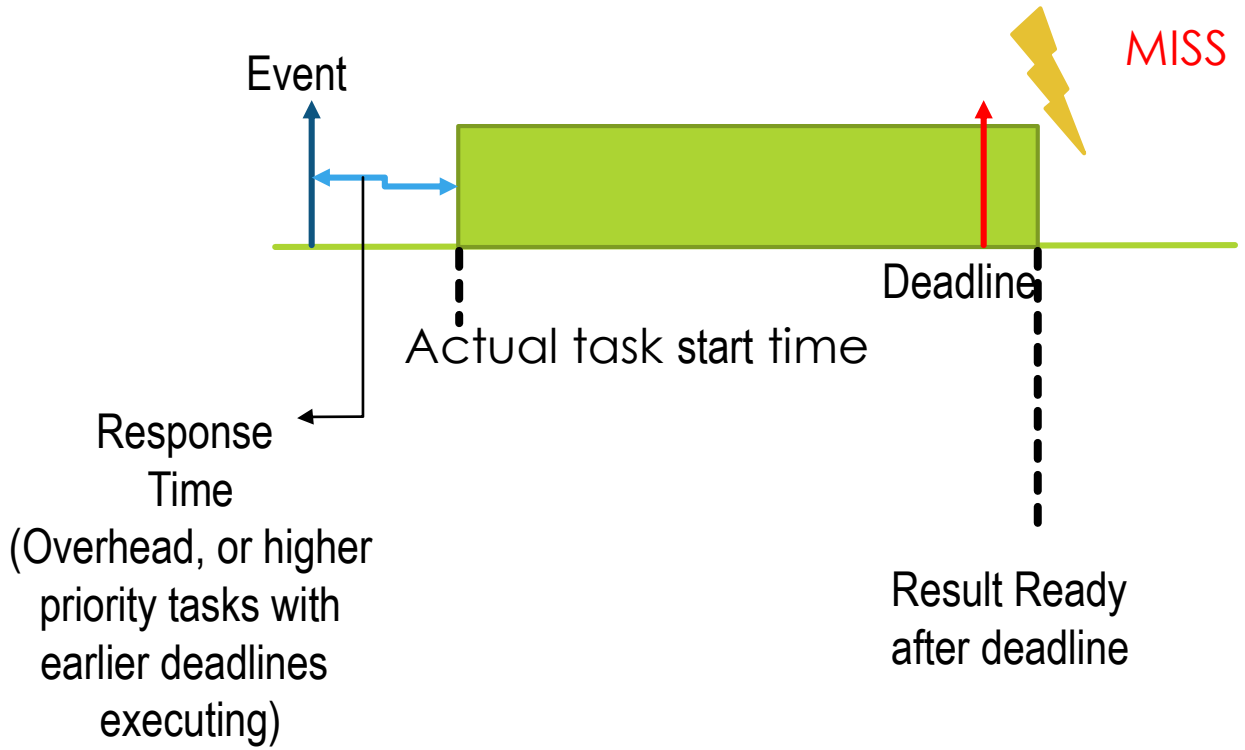
→ Usually specified by some probability? What is the probability that task A misses its deadlines 10% of the time?

→ Probabilistic analysis → complexity at design time!

→ Time bounds between fraction of a second to few seconds

- Complex real-time systems could consist of subsystems of any of the three types.

Real-Time Systems Classification by Deadline Type (4)



- Did Catastrophe/death happen? → Hard RT
- No catastrophe/death, but the result is too late and has no value → Firm RT
- No catastrophe/death, but even if result is too late, it has value and we can use it → Soft RT

Examples

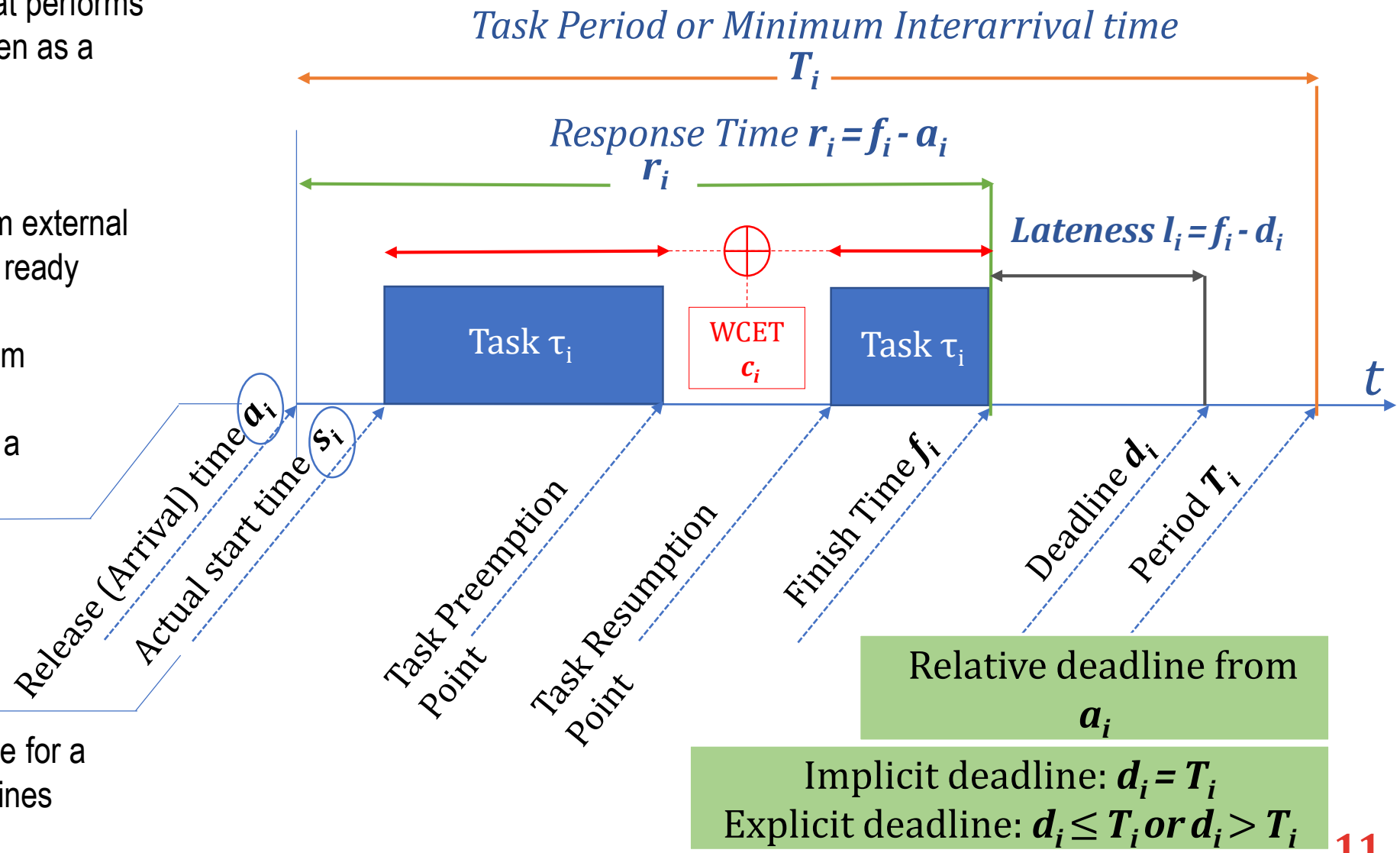
- Transportation: self-driving cars, auto-pilots, guidance systems, spacecraft, ABS, automotive (e.g., ECS); (Mostly Hard)
- Military: Weapon systems (e.g. missiles and anti-missiles), target auto-tracking and locking; (Mostly Hard)
- Industry: control of production lines and manufacturing, robotics; (Mostly Hard)
- Medical: Patient-monitoring, defibrillation (Mostly Hard)

- Telecommunication: TV, digital video, networked multimedia systems and streaming (Mostly Firm)
- Household and building management: HVAC, Building security, control appliances (Mostly Soft)

- Interesting to note that 99% of all processors are for the embedded systems market.
- Many of which are for RT applications!

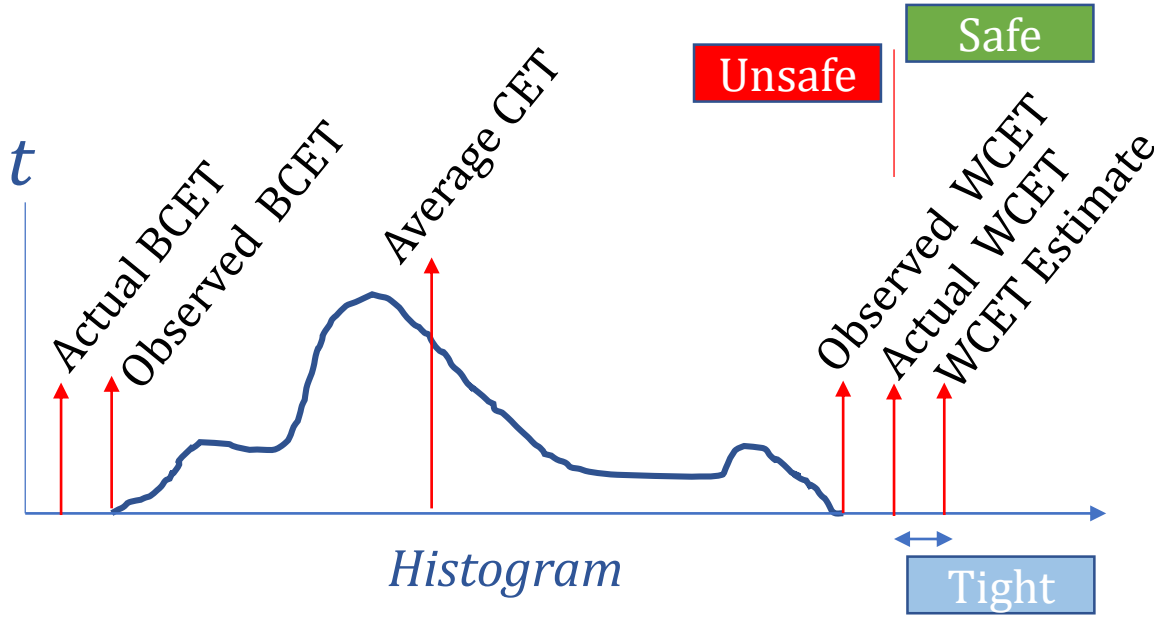
Know *thyTask* (RT Task Terminology)

- A task is a piece of code that performs a specific function (*i.e.* written as a thread)
- A task is activated by an internal/external event
- Event could be a signal from external sensor that denotes data is ready (periodic/ aperiodic)
- Could be triggered by system timer/RTOS timer (periodic)
- Activated tasks are put into a **READY** pool
- Scheduler selects from available ready tasks which one to execute next
- Goal: for all tasks $f_i \leq d_i$
- Lateness should be negative for a system that meets its deadlines



Task Worst Case Execution Time (WCET) (1)

- If you run a task with different inputs, you might get different execution times.
- Execution time highly depends on:
  1. The task input
  2. Initial system state
- This is because different inputs possibly change the execution path of if/else statements, or the number of times the loop repeats. Also, in RT systems with caches (firm/soft RT systems), the cache state will produce different hit/miss cases, thus more time to fetch data from the main memory.

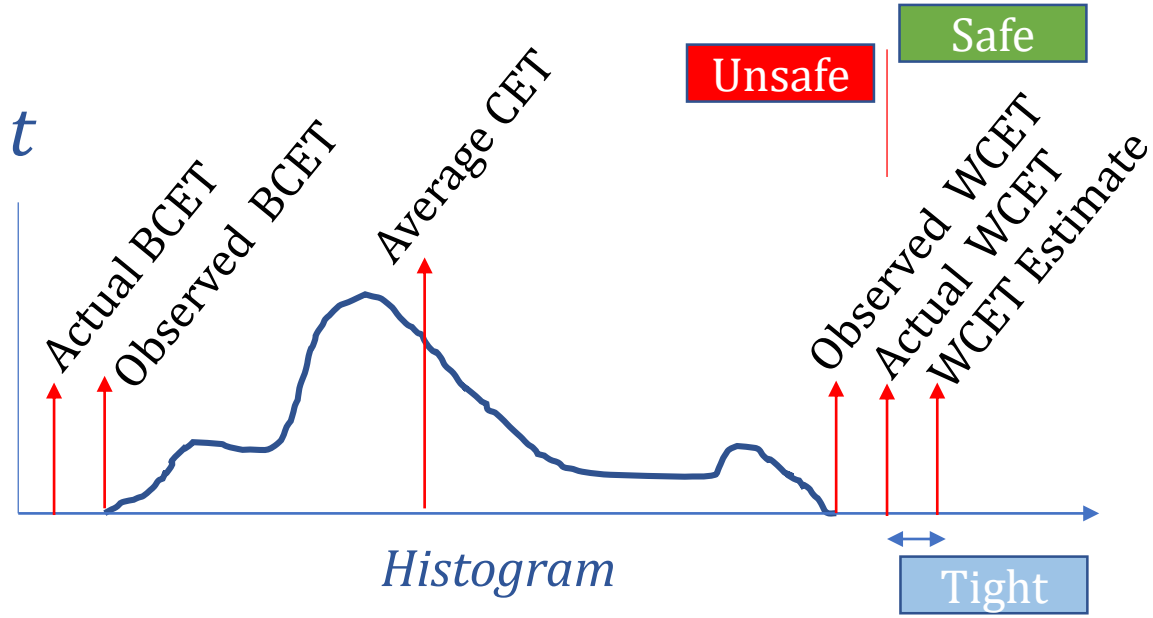



```
function task_one (x, y){
For (i = x; i<= y; i++){
      do something;
}
```

```
function task_one (x, y){
If (x < 10){
     average the first x values
     of array y
Else if (x < 20)
     average the first 2x values
     of array y }}
```

Task Worst Case Execution Time (WCET) (2)

- Therefore, RT systems **DO NOT** deal with average execution times, instead, they work with Task **W**orst **C**ase **E**xecution **T**ime (WCET).
- This is important for system safety, by working with WCET, we can design a working system that does not exceed the task deadlines which is the essence of real-time systems.
- By taking numerous measurements, we can draw a histogram of task execution times. Remember, these are observed values (measurements). We might or might not capture the ACTUAL best or worst case (unless we test all possibilities which is almost impossible).
- Because we cannot easily get **EXACT** WCET, we resort to estimates.



However, these estimates must be **safe** and **tight**.

- Safe such that they are never less than the actual WCET, because if our estimate is wrong, and in real-life the task executes for a time more than the estimate and misses the deadline → Problem!
- Tight such that our estimate should not far exceed the actual WCET.
- In summary be as close as possible, but never less!

Estimating the task WCET

Measurement-Based Analysis (MBA)

- This technique creates numerous test cases based on lots of possible inputs and different initial system states. Each possibility is called a test vector. The procedure goes by running the task with a test vector one at a time, and by using breakpoints or other hardware tool, the tester measures the execution time. However, this approach is:
  - ❖ Costly, imagine the test cases and resulting test vectors! Gets more tedious as the number of inputs increases. You can end up with millions of cases. Could take you weeks, months, or years!
  - ❖ No guarantees for safety. Even if you try numerous test cases, you might miss the one which captures the WCET.
  - ❖ In industry, they arbitrarily add a safety margin to the maximum observed WCET from the measurements (e.g. 15 to 20%).
  - ❖ No one knows where this percentages comes from, they just claim it worked ***so far***!

Probabilistic-Based Analysis (PBA)

Fairly new technique. Based on MBA and SBA. Uses far less test vectors and uses probability to estimate the WCET. Mostly based on the Final-Value Theorem

Static-Based Analysis (SBA)

- This technique is the industry's favourite, because it gives safer and tighter estimates than MBA. They analyze the code at the assembly code level. They also require lots of input test vectors, and also a timing model for the processor/hardware that the code is going to be executed on. However, this approach is:
  - ❖ Still costly, it needs to analyze all possible paths the task can go through. The more branching/loops, the more sophisticated it gets.
  - ❖ Requires the developer to guide the tool through "annotation", that is to identify branches, loops, restrictions, and constraints. So the tester must know the code well.
  - ❖ The companies that develop the processors / hardware might not share their timing models freely, or at all. Also, the model might not be 100% accurate. Mistakes happen!

Functional Requirements (1)

1. Data Collection

A. Acquire data from sensors

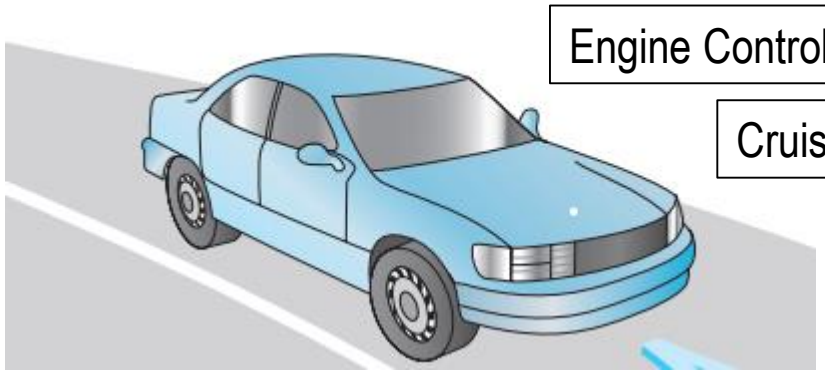
Data is a snapshot of the state of the plant (e.g., Car (speed, position of switches, engine temperature, road slope))

Available data is called State Variable → Significant and Insignificant state variables

→ Significant state variable → Within ***sphere of control*** of a subsystem

→ Also called RT Entity (from RT perspective)

→ Temporally Accurate for a limited time



Engine Control System

Cruise Control System

Air Bag Control System

HVAC Control System

RT Entity Updates are time-triggered or event triggered

Depends on system dynamics

Relatively short accuracy interval

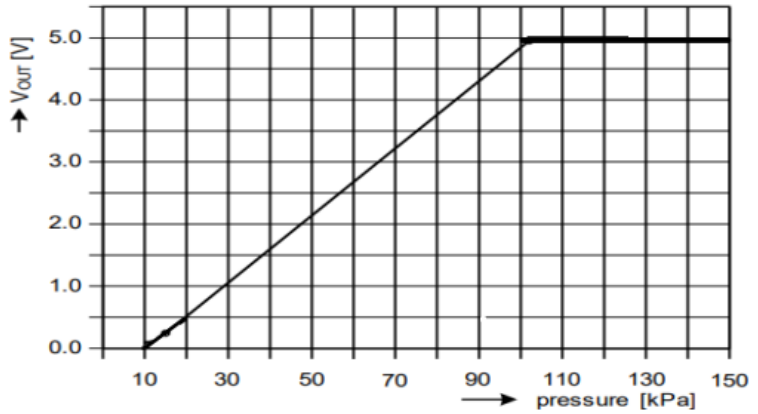
Relatively long accuracy interval

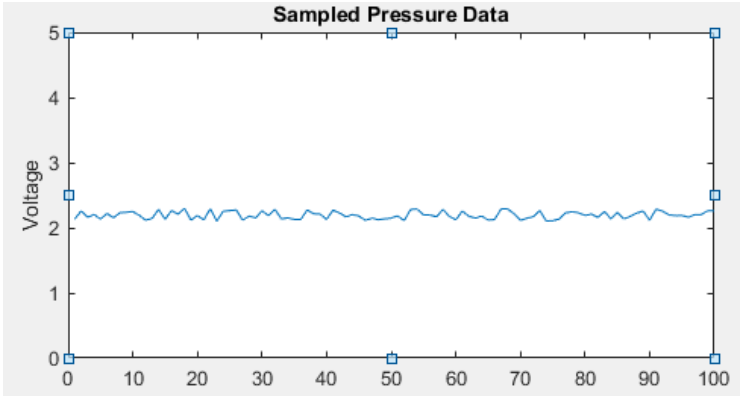
Functional Requirements (2)

B. Signal Conditioning

Signal conditioning is used to refer to all the processing steps that are necessary to obtain meaningful *measured data* of an RT entity from the raw sensor data.

- Sensors produce raw data! (e.g., voltages).
- Scaling to required values (e.g., Voltage to Pressure or temperature, etc.).
- Inherent Measurement errors (e.g. A/D Quantization).
- Sensors also need calibration at many times.
- Noise effects → Must be filtered out (Anti-Aliasing filters, Digital Filters, etc.)





Functional Requirements (3)

C. Alarm Monitoring

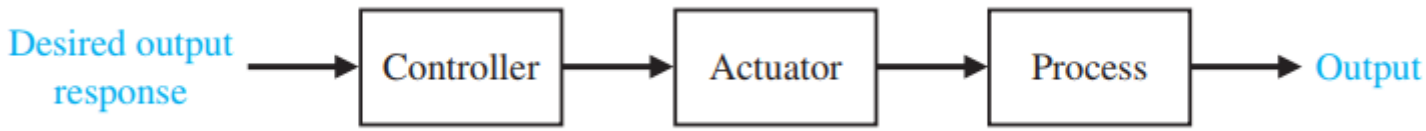
- Continuous monitoring of the RT entities to detect abnormal process behaviors.
- E.g., Pipe Rupture in a chemical plant → many RT entities (diverse pressures, temperatures, liquid levels) will deviate from their normal operating ranges, and to **cross some preset alarm limits** → Alarm Shower
- Must identify primary event! → logging with exact timing
- In industrial plants, real-time control systems are interfaced with sophisticated knowledge-based systems for alarm analysis!

Functional Requirements (4)

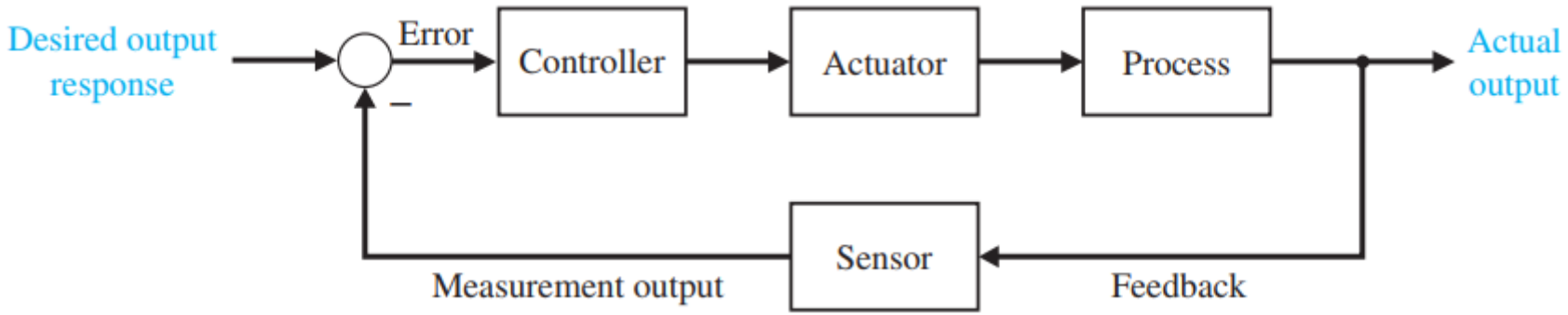
2. Direct Digital Control

- Consists of an (infinite) sequence of control periods (Sampling RT entity → Algorithm Execution →New Setpoint to actuator)
- A proper control algorithm compensates for the random disturbances that perturb the controlled object.

- **Open-loop control**



- **Closed-loop feedback control**



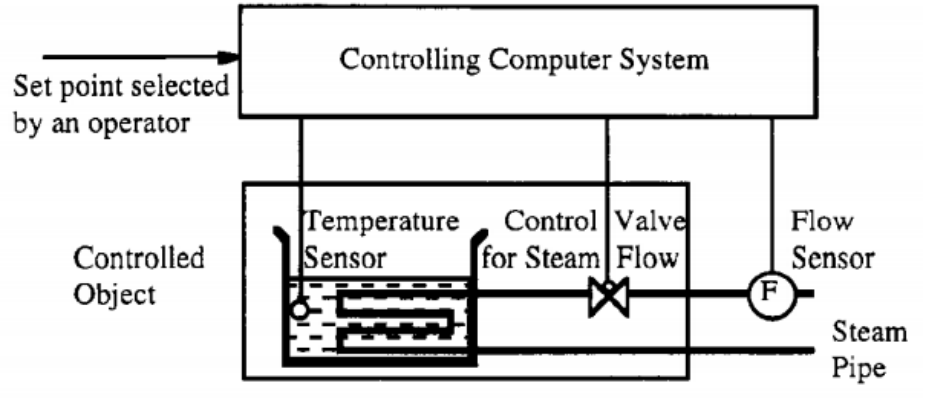
Functional Requirements (5)

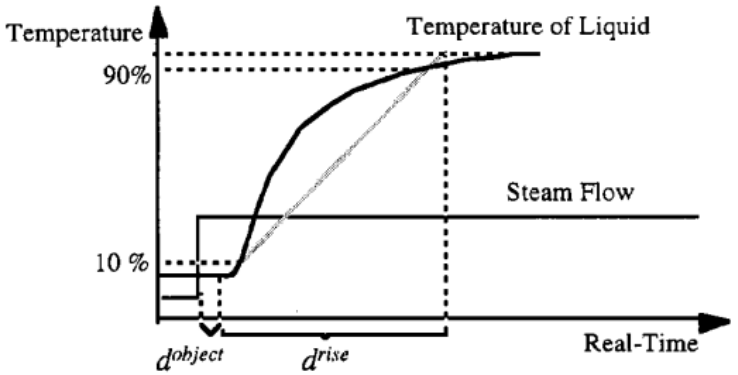
3. **Man-Machine Interaction**

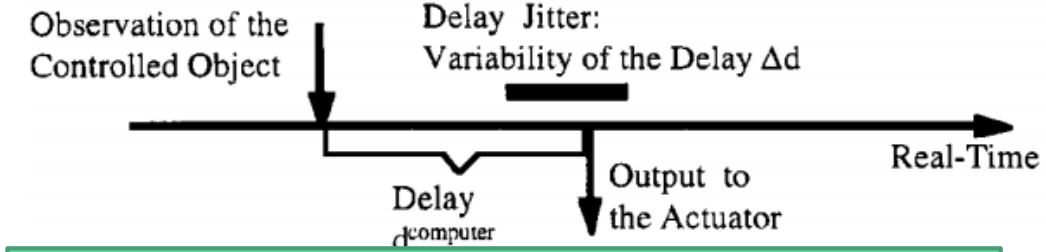
- Must inform the operator of the current state of the controlled object, and must assist the operator in controlling the machine or plant object **(Process-Control)**
- Many catastrophic computer-related accidents in safety-critical real-time systems have been traced to mistakes made at the man-machine interface.
- Many sophisticated industrial control systems consider logging data as a functional requirement.

  Example: Laws force pharmaceutical industries to store all relevant batch parameters! **Why Important?**

Temporal Requirements (1)







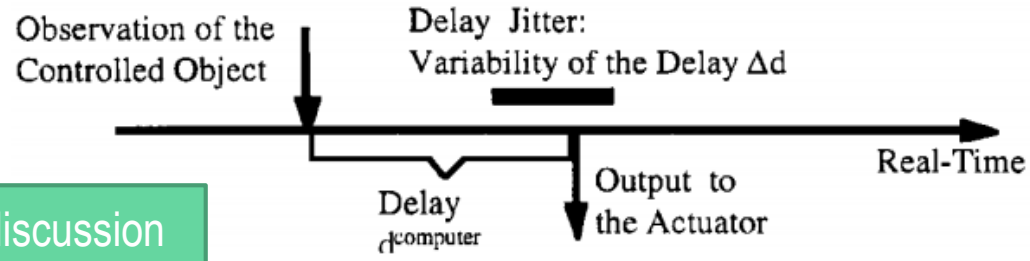
Symbol	Parameter	Sphere of Control	Relationships
d^{object}	controlled object delay	controlled object	physical process
d^{rise}	rise time of step response	controlled object	physical process
d^{sample}	sampling period	computer	$d^{sample} << d^{rise}$
$d^{computer}$	computer delay	computer	$d^{computer} < d^{sample}$
$\Delta d^{computer}$	jitter of the computer delay	computer	$\Delta d^{computer} << d^{computer}$
$d^{deadtime}$	dead time	computer and controlled object	$d^{computer} + d^{object}$

Refer to the book chapter for the detailed discussion

Temporal Requirements (2)

Minimal Latency Jitter

- $\Delta d << d_{Computer}$
- When delay Δd is too small, almost constant!
- Control algorithm can compensate and deal with known constant delay.
- Delay Jitter (Highly variable Δd) brings uncertainty → Negatively affects quality of control!
- Uncertainty about the actual observation time, which means additional error!



Refer to the book chapter for the detailed discussion

Temporal Requirements (3)

Minimum Error Detection Latency

- Hard real-time applications are, by definition, safety-critical.
- Error within the control system (e.g., the loss or corruption of a message or the failure of a node) is detected within a short time with a very high probability.
- *Error-detection latency* must be in the same order of magnitude as the sampling period of the fastest critical control loop → Allows for fast corrective action before the error causes severe system failure
- Jitterless systems will always have a shorter error-detection latency than systems that allow for jitter.

Refer to the book chapter for the detailed discussion

Non-Functional Requirements (1)

1. Dependability Requirements:

- **Reliability:** probability that a system will provide the specified service until time t, given that the system was operational at $t = t_0$
  - * $R(t) = e^{-\lambda(t-t_0)}$, where λ is constant failure rate in failures/hour
  - * Ultra-high reliability when $\lambda < 10^{-9}$
  - * Mean Time To Failure (MTTF) = $1/\lambda$
- $MTTF = Number\ of\ built\ components \times operational\ hours \times days$
  So if a car company produces 3 million cars in one year, and on average each cars drives two hour a day each day of the year, and only one car fails during this whole year, this means that:

MTTF = 1,500,000 × 2 × 365 = 1,095,000,000 hours, which means that the reliability λ is close to the order of 10^{-9}

Non-Functional Requirements (2)

1. Dependability Requirements (continued):

Safety requires certification

- **Safety**: defined as 1. responses to protect the system from harm (e.g., error detection)
  2. reliability against critical failure modes (e.g., plane crash, self-driving car accident)

  For example, *safeRTOS* is a certified safe version of *freeRTOS*
  components must successfully pass certain tests like the FCC, CE, EMC
  must comply with certified industry safety standards (e.g. aviation or automotive safety standards)

- **Fault-tolerance:** protection from design and operational faults? How?
  Hardware redundancy → E.g., Two lock-step processors in tandem.
  Roll back/recovery and checkpoints (similar to computer games ☺), however; in HRT:
  1. Difficult to guarantee a deadline when error occurs → roll-back and recovery can take unpredictable time.
  2. The error could have caused irrevocable action (remember we are connected to other hardware which affects the plant/controlled objects)
  3. Temporal accuracy of the checkpoint data is invalidated by passage of time

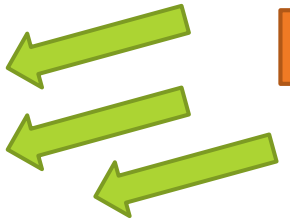
- **Security**: protect system from intentional harm or access

Non-Functional Requirements (3)

2. Performance: timing of responses or throughput necessary?

3. Robustness: protection from external interference and perturbations

Must remain at 30m



Wind

Collision with Objects

4. Scalability: Perform reasonably in an environment with added load

Fail-safe vs. Fail-Operational Real-Time Control Systems

- Some hard-real time systems can have safe states **(fail-safe)** → When system fails, go to safe state
  Examples: Railroad signaling systems
  In generic terms, an electrical fuse provides fail-safe mechanism by preventing permeant damage!
  * Requires high error-detection coverage → the probability that an error is detected, provided it has occurred, must be close to one
  * Possible Implementation → Watchdog timers

- In certain applications, you cannot identify a safe state!
  Example: Flight control system of airplane or space craft!
  * Must provide minimum level of service to avoid catastrophe even if failures occur
  * These systems are called ***fail-operational***

Required Reading

- Chapter 1: *Introduction to Real-Time Systems* from the book:

  “A practical introduction to real-time systems for undergraduate engineering”, Harder and Zarnet, 2018

- Chapter 1: *The Real-Time Environment* from the book:

  “Real-Time Systems, Design Principles for Distributed Embedded Applications”, Hermann Kopetz, 2002