EE5903 RTS
Chapter 1
Introduction to RTS

Bharadwaj Veeravalli

elebv@nus.edu.sg

Acknowledgements: Parts of this material from Hard Real-Time Computing Systems, G.C, Buttazzo, Springer, 3rd Edition; Introduction chapter 4 slides from A/Prof Mandar;

RTS Introduction Outline

- General introduction
- RTS roles in different environments
- Characteristics of a RTS
- Reliability/Availability in RTS
- RTS Design considerations / challenges
- Example – HRTS - Intelligent Transportation Systems
- Distributed RTS
- Embedded RTS
- Other Issues – Synchronization, Deadlocks, Fault-tolerance, etc
- *Buzz words in this domain* - ☺

RTS General Introduction

- What is **Real-time?**

  “*Timeliness is as important as correctness of the outputs*”

RTS *need not* be a “fast” system – All that it needs is to obey the timeliness in the sense of meeting specific constraints imposed by the requirements of an application and also must exhibit predictable performance.

RTS General Intro (cont'd)

- Examples:

  - Air-traffic/Flight Control Systems

  - Airline reservation System

  - Autonomous Vehicles

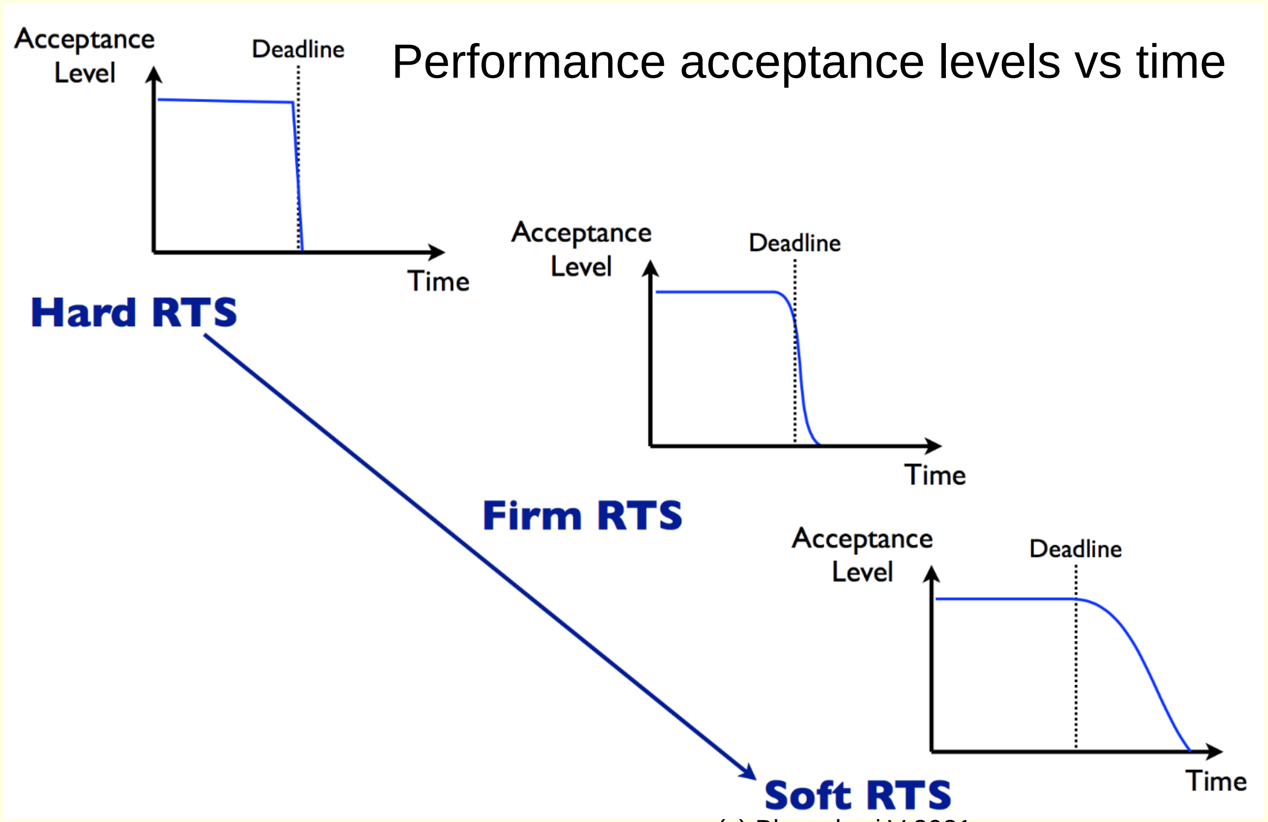
  - Automated Guided Vehicles (AGVs) in a manufacturing system

  - Networked Media Service Systems – Crucial for a Service Provider;

…

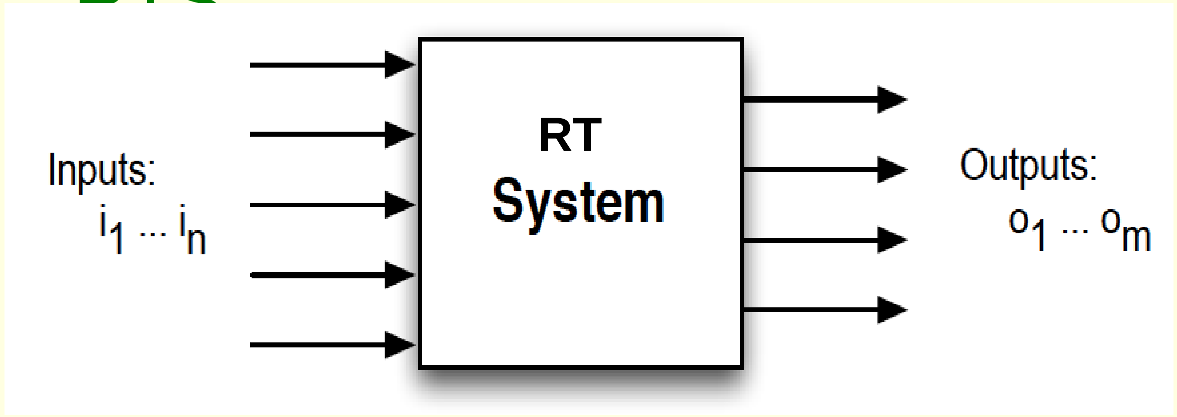
RTS General Intro (cont'd)

Types of RTS – Hard, Firm, Soft (*all tied to our "timeliness" definition*)



RTS General Intro (cont'd)

Formal Definitions of a RTS:



A system has a set of one or more inputs and a set of one or more outputs

The time between the presentation of a set of inputs to a system, and the appearance of the associated outputs is called the **response time** of the system

RTS General Intro (cont'd)

Definition 1:

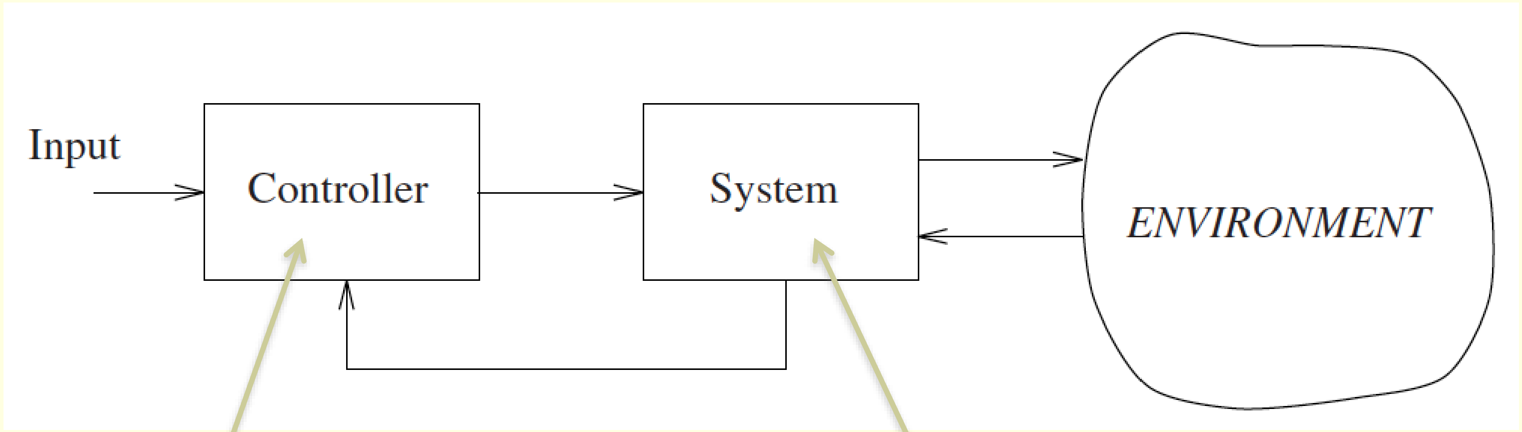
A real-time system is a system that must satisfy explicit bounded response-time constraints.

Definition 2:

If the logical correctness of a system is based on both the correctness of the outputs and their timeliness, then the system is a RTS.

RTS General Intro (cont'd)

A generic, complex, control system comprises:

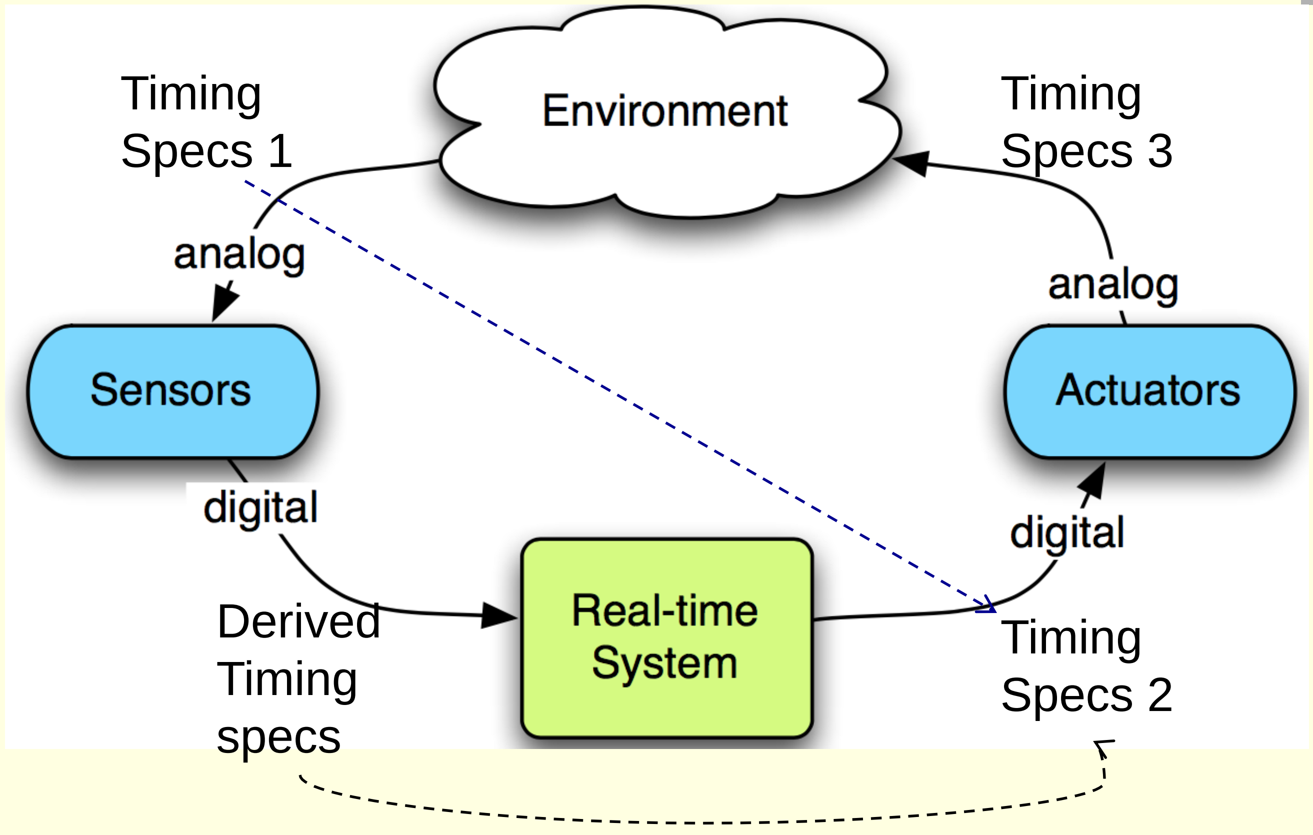


A computing system, capable
of providing right inputs at the right time
instants to meet the desired objective

System to be controlled
(Car, robot, machine, etc)

Remarks: Associated with this there are two peripheral sub-systems – actuation system and sensory system;

RTS General Intro (cont'd)



TS1: Sampling

DTS: Involves processing like A2D and hence takes some time;

TS2: Related to TS1 from application's Perspective;

TS3: Actual impact to the environment; Referred to as a Perceivable Response;

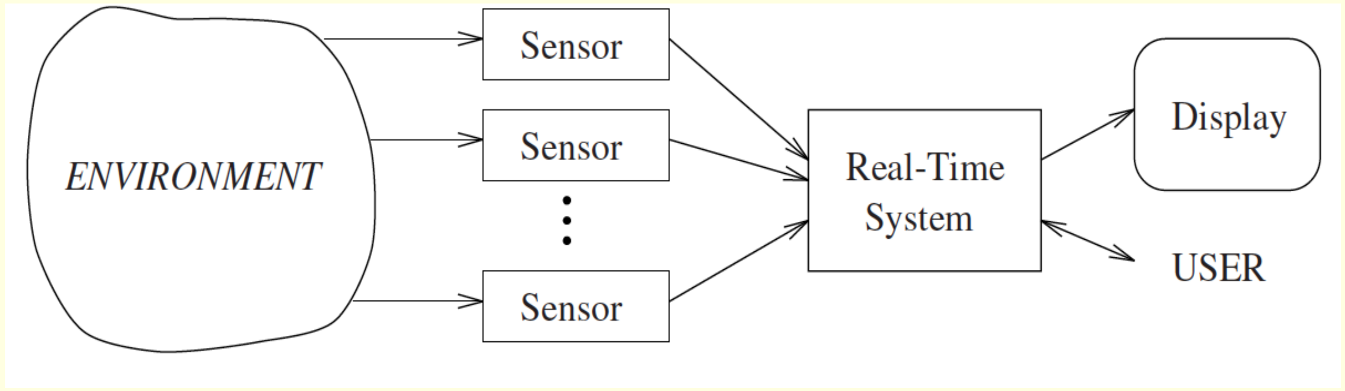
Example of a typical Control System that has an (Embedded) RTS

RTS roles in different environments

We have three types of environments in which RTS, depending on the interactions between the controller and the environment, can fit in:

- Monitoring systems
- Open-loop control systems
- Feedback control systems

RTS roles in different environments - Monitoring Systems



Example of a typical **RT Monitoring** system

MSs do not modify the environment but only use sensors to perceive its state, process sensory data, and display the results to the user.

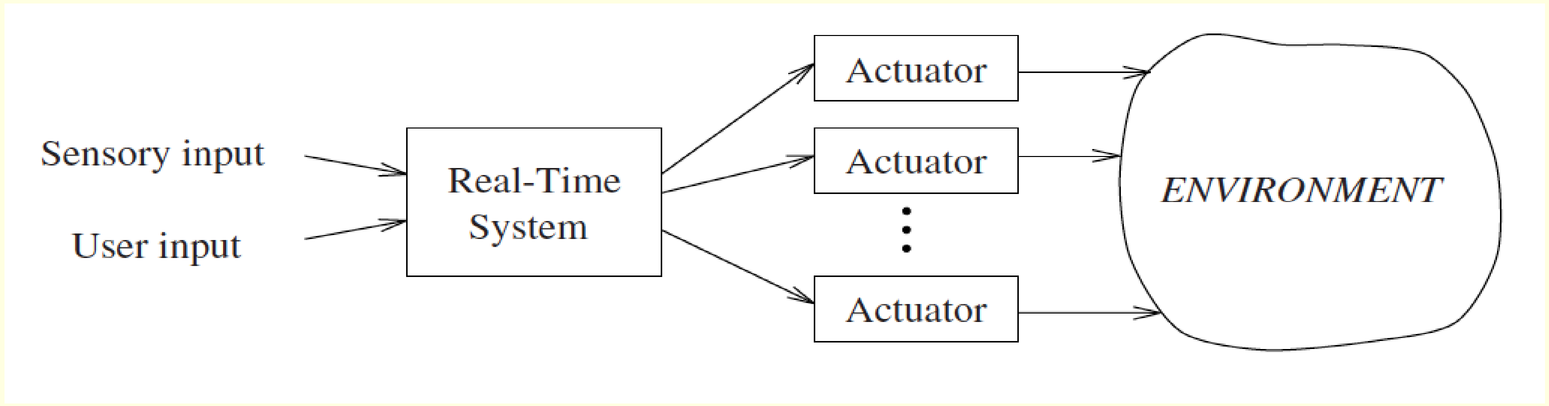
Examples – Air-traffic control system, environmental pollution monitoring, Surveillance systems, etc

These applications demand periodic data acquisitions from multiple sensors and each sensor may have a different yet constant sampling rate.

RTS roles in different environments - Monitoring Systems

- If multiple sensors are used to detect critical conditions, the sampling rate of each sensor has to be constant in order to perform a correct reconstruction of the external signals. In these cases, using a hard real-time kernel is a necessary condition for guaranteeing a predictable behavior of the system.
- If sensory acquisition is carried out by a set of concurrent periodic tasks (characterized by proper periods and deadlines), the task set can be analyzed off-line to verify the feasibility of the schedule within the imposed timing constraints.

RTS roles in different environments – Open-Loop based Systems



Example of a typical **Open-Loop based RT** control system

Example – Robotic platform equipped with a vision based technology used to take snap and relay the coordinates of an object to a robot whose job is to pick that object in a manufacturing environment;

RTS roles in different environments

- Example system: Consider a robot workstation equipped with a vision subsystem, whose task is to take a picture of an object, identify its location, and send the coordinates to the robot for triggering a pick-and-place operation. In this task, once the object location is identified and the arm trajectory is computed based on visual data, the robot motion does not need to be modified online;

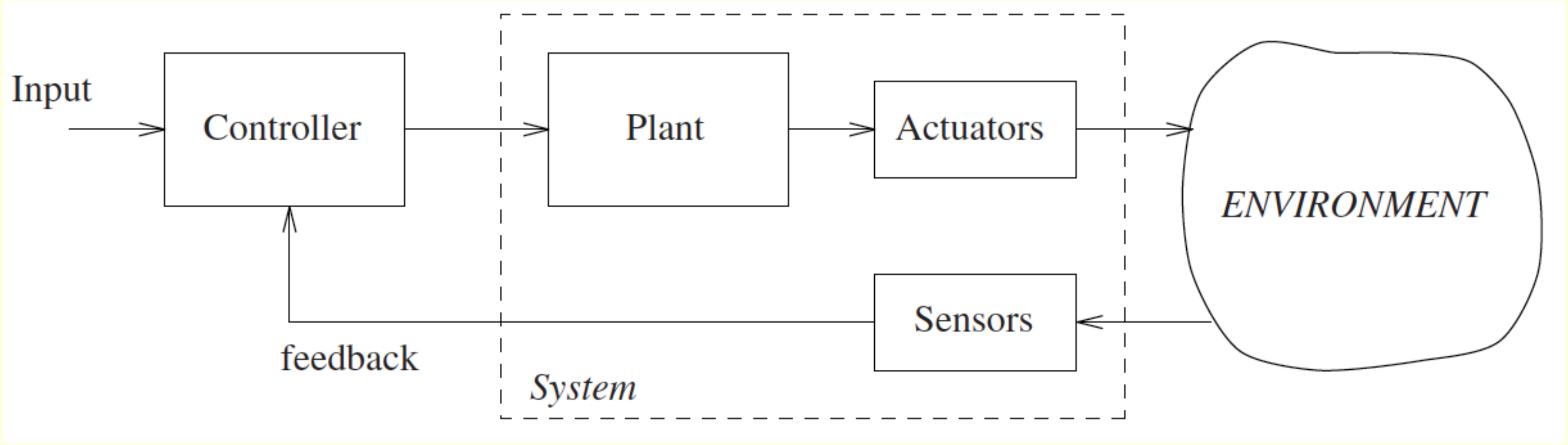
- Therefore, no real-time processing is required. Note that real-time computing is not needed even though the pick-and-place operation has to be completed within a deadline. In fact, the correct fulfillment of the robot operation does not depend on the kernel but on other factors, such as the action planner, the processing speed of visual data, and the robot speed. For this control problem, fast computing and smart programming may suffice to meet the goal.

RTS roles in different environments - Open-Loop Systems

Characteristics

- Open-loop control systems are systems that interact with the environment.
- However, the actions performed by the actuators do not strictly depend on the current state of the environment.
- Sensors are used to plan actions, but there is no feedback between sensors and actuators. This means that, once an action is planned, it can be executed independently of new sensory data.
- No RT need here in most OLSs. Non-RT systems use this type of control predominantly

RTS roles in different environments - Feedback RT Systems



Example of a typical **Feedback RT** system

Example – Fly-by-wire aircraft control dynamics fits in this case;

Feedback response from sensors to MC as well as from actuators exist;

RTS roles in different environments

Feedback Systems Characteristics

- Feedback control systems (or closed-loop control systems) are systems that have frequent interactions with the environment in both directions; that is, the actions produced by the actuators strictly depend on the current sensory information.
- In these systems, sensing and control are tied together, and one or more feedback paths exist from the sensory subsystem to the controller.
- Sensors are often mounted on actuators and are used to probe the environment and continuously correct the actions based on actual data

Three major characteristics of a RTS

- Interaction with the real-world

  System must respond to any non-deterministic asynchronous stimuli/demands

- Concurrency

  Multiple CPUs / Scheduling Strategies / Handling access to any Shared resource / inter-process Comms. & signaling / Handling interrupts

- Long-term Reliability / Fault-tolerance

  Time continuous support (24/7) / fault-tolerance / predictable failures / Graceful degradation

Reliability / Availability in RTS

- Failure can occur at any part of a RTS – from sensors to actuators, including computing components

  -- Failures due to - Stress/pressure / heat / current and voltage surges / software bugs / unexpected critical inputs / …

- Failure can occur at any part of a RTS, either independently or in a dependent way – from sensors to actuators, including computing components

Reliability … (Cont'd)

- *Time-to-Failure* – An important parameter of interest to RTS;

  -- The ***Time-to-Failure*** **T**, of a component (or system) is *random variable* taking values on the range [0, infinity); Also called "life-time" and "failure-time".

- Parameter T may not represent time always – Example: distance traveled by an AGV, # of actuations of a switch, etc.;
- A particular value of T is denoted by 't'

Reliability … (Cont’d)

- Some meaningful questions:

  -What is the probability that a component fails at t, i.e., P(T=t)?

  - P(a < T <b) = ?

  - What is the expected value of T?

  - What is the spread of T?

  - How is T distributed?

In a multiple/multi-core CPU system – failure of one core affects the performance in a cooperating computing environment!

Reliability … (Cont’d)

Example: Assume that a multi-core CPU based HRT system involved in a cooperative computing task. Suppose T is the time-to-failure of a core expressed in hours. Determine the probability that the HRT system will fail in the second hour assuming T has the following pdf:

$$f(t) = 0.2e^{-0.2t} \quad ; t \geq 0$$

We assume a single core failure would terminate the cooperative computing Task to miss its deadline.

$$\text{P(Failure in second hour)} = P(1 < T < 2)$$

$$= 0.2\int_{1}^{2} \exp(-0.2t)dt$$

$$= e^{-0.2} - e^{-0.4} = 0.148$$

Reliability … (Cont'd)

MTTF – Mean time to failure is the expectation of T, given by,

$$\begin{aligned}\text{MTTF} &= \mu_T \\ &= E[T] \\ &= \int_0^\infty t\, f(t)\, dt\end{aligned}$$

Continuing our example, we have MTTF as,

$$\begin{aligned}\text{MTTF} &= E[T] \\ &= \int_0^\infty 0.2te^{-0.2t}\, dt \\ &= 5\ \text{hrs}\end{aligned}$$

The variance of T, denoted $\sigma_T^{\,2}$, is the expectation of $(T - \mu_T)^2$

Reliability … (Cont'd)

Variance:

$$\sigma_T{}^2 = E[(T - \mu_T)^2]$$
$$= E(T^2 - 2\mu_T T + \mu_T{}^2$$
$$= E(T^2) - 2\mu_T E(T) + \mu_T{}^2$$
$$= E(T^2) - \mu_T{}^2$$

Continuing our example, we have,

$$\sigma_T{}^2 = E[T^2] - \mu_T{}^2$$
$$= \int_0^\infty t^2 f(t)dt - \mu_T{}^2$$
$$= \int_0^\infty 0.2t^2 e^{-0.2t} dt - 25$$
$$= 25\,\text{hr}^2$$
$$\sigma_T = 5\,\text{hrs}$$

Reliability … (Cont'd)

- **RELIABILITY R(t)** is defined as the probability that a system/component will function over some time period t. Let T be a continuous RV denoting the time to failure of the system. Then:

$$\boxed{\begin{aligned} R(t) &= P(T \geq t) \\ &= \int_t^\infty f(t)\,dt \end{aligned}}$$

- The **FALLIBILITY/failure F(t)** is the probability that a failure occurs before time t, i.e., in the time interval [0, t].

$$F(t) = P(0 \leq T \leq t)$$

Hence:

$$= \int_0^t f(t)\,dt$$

$$= 1 - R(t)$$

Reliability … (Cont'd)

- Clearly, F(t) is the complement of R(t) and it is also the *cumulative distribution function* (CDF), of T.
- Some important points to note:
- R(t) and F(t) are probabilities and hence they are bounded by 0 and 1.
- R(t) monotonically decreases from 1 to 0. $R(0) = 1 \quad R(\infty) = 0$
- F(t) monotonically increases from 0 to 1. $F(0) = 0 \quad F(\infty) = 1$
- $f(t) = \frac{dF(t)}{dt} = -\frac{dR(t)}{dt}$
- $P(a < T < b) = F(b) - F(a) = R(a) - R(b)$
- Alternatively, MTTF can be obtained directly from R(t): $MTTF = \int_0^\infty R(t)\,dt$

Reliability … (Cont'd)

Continuing our example, we have:

$$R(t) = e^{-0.2t} \qquad ; t \geq 0$$ (*Verify!*)

$$F(t) = 1 - e^{-0.2t} \quad ; t \geq 0$$

We will study other properties, characteristics in Chapter 4 in detail; This includes, *how a fault/failure in one component of a RTS triggers and influences failures in others leading to system failure, deriving failure rate when components fail independently, etc.*

RTS Design considerations / challenges

1. User/Application requirement gathering
2. System design – concurrency, throughput, error recovery must be considered and any high level design decisions must take into account of RT constraints, if any.
3. Implementation & testing phase must consider all RT needs – end to end;
4. Choice of operating system (OS) & implementation platform - process scheduling, interrupt handling, exception processing, etc.
5. Platforms - System on a chip (SoC), application specific integrated circuit (ASIC), field programmable gate array (FPGA) based implementations

Interrupt programming in Embedded Systems

- Interrupts sequence – Common behavior
  - An interrupt is a signal to the processor/controller raised by h/w or s/w indicating that an event needs an immediate attention;
  - Current instruction completed; Current status saved in a stack - flags/regs/certain mem. contents, etc;
  - Control jumps to where the corresponding ISR resides; ISR is executed and control returns with appropriate loading of the earlier status from the stack to the controller;

Note: ISRs have a fixed location for certain type of interrupts in a given microcontroller

Interrupt programming in ES

- Hardware interrupts – Could be from another h/w device at a specific pin of a microcontroller; Related issue – false alarm possible – triggered by other system faults!

- Software interrupts – Triggered by an exceptional condition or via any special instruction (div by zero, etc); S/w bugs in programming can trigger an interrupt! Or, a bad programming can miss giving certain "interrupts" at the required time epochs!

Alternative scheme to handle events - Polling

- Polling – To monitor the status of a device or an entity in a time-continuous manner; Polls to check for any service needed and hence consumes CPU resources;
- Interrupts – Only when a request arrives service is initiated

RTS can use both the methods above depending on the complexity of the system; When arrival of requests is known to be sporadic an underlying RTS can wait for an interrupt to occur and respond; For a small-scale Soft RTS with less number of service options, polling may be a better solution!

Interrupt programming in ES - Handling Interrupts – Edge-triggering (ET) or Level-triggering (LT) ?

- In RTS context:

ET module generates an interrupt only when it detects an asserting edge of the interrupt source. The edge gets detected when the interrupt source level actually changes (L->H or H->L).

This is useful when we attempt to detect an event by periodic sampling; Usually the response would be faster for systems employing ET;

ET interrupts attempts to keep the firmware's code complexity low to the best possible extent, reduce the number of conditions for firmware, and provide more flexibility when interrupts are handled.

Interrupt programming in ES - Handling Interrupts – Edge-triggering (ET) or Level-triggering (LT) ?

- In RTS context:

LT module generates an interrupt whenever the level of the interrupt source is asserted.

Multiple interrupt triggers possible. That is, if the interrupt source is still asserted when the firmware interrupt handler handles the interrupt, the interrupt module will regenerate the interrupt, causing the interrupt handler to be invoked again.

Level-triggered interrupts are complex for firmware.

Interrupt programming in ES

- Priorities – Interrupts are prioritized; User defined interrupts possible and these reside at a fixed place in the memory;
- Priority 0 (high) to 255 (low) number of priorities possible (depending on the MC);
- Handling multi-level interrupts – A current task triggered by a priority #x can be interrupted by another interrupt with a priority #y, if y < x. In RTOS, *Priority Inversion Protocols (PIPs)* are designed to handle such events. (*Chapter 6!*)

Importance of Interrupt programming – A real-life example

- Gulf War (1991) - Sequence of events are as follows:

(i) Radar detects a scud missile directed to Saudi Arabia;

(ii) On-board CPU predicts its trajectory, verifies, classifies as a "false alarm";

(iii) Scud hits a city (Dhahran) - Severe damage/loss of lives;

So what went wrong?

Importance of Interrupt handling – A real-life example

- Interrupt handling routine with disabled interrupts was the key problem;

- Because of the disabled interrupts, the RT clock on the on-board CPU was “missing” some clock interrupts, which caused a delay accumulation of nearly 50+ msecs per minute!!

- For a total time span of 100 hours, on that day, a total delay of nearly 300+ msecs was observed to be accumulated which caused a prediction error in the verification phase of ~ 650+ meters!!

Example of a Hard-Real Time System - Intelligent Transportation Systems(ITS)

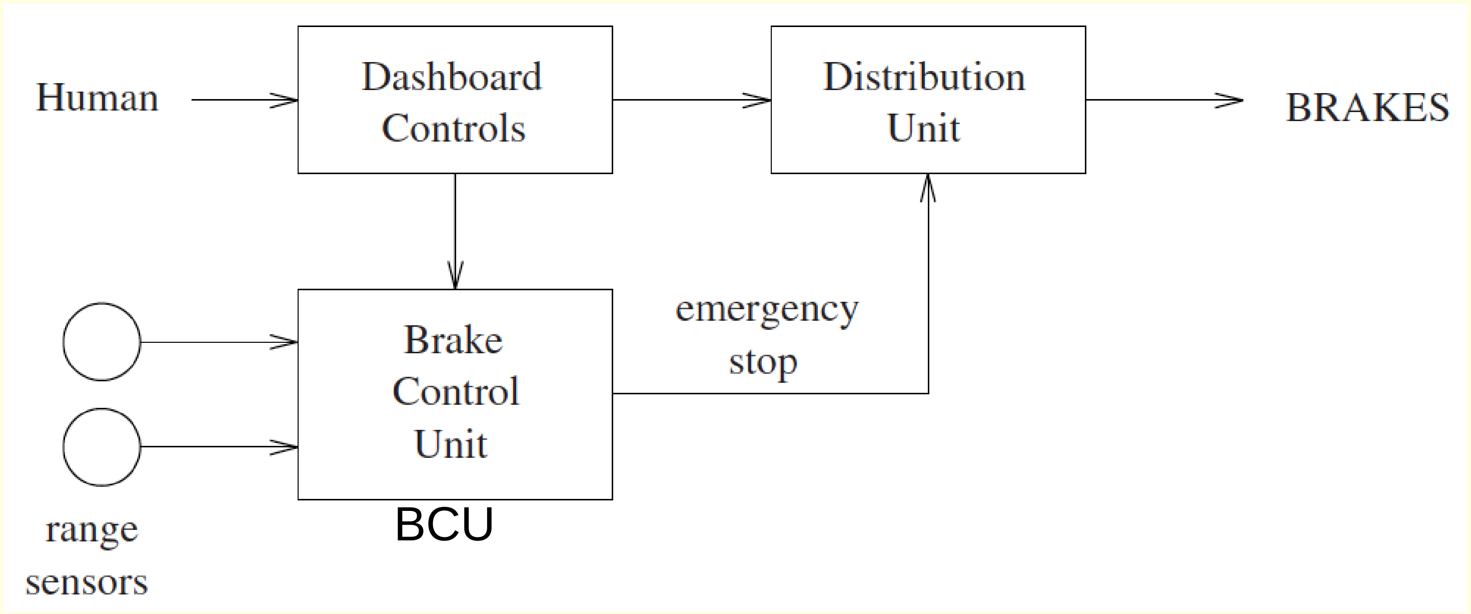
Autonomous Vehicle – An imperative Smart City component!

- Obstacle avoidance – Typical example for the design of a RT braking system design.

Some intriguing questions:

- What are the known parameters?
- What is the ***response time*** of the system?
- What is that we are computing here?

HRTS – Example ITS …(Cont'd)



- Vehicle travels in a straight line with a constant velocity;
- BCU continuously samples/monitors the activities by reading the state variables from the dashboard of the vehicle and decides to apply brakes, if needed.

HRTS – Example ITS …(Cont'd)

Acquiring data from a pair of range sensors, computing the distance of the obstacle (if any), reading the state variables of the vehicle from instruments on the dashboard, and deciding whether an emergency stop has to be superimposed.

Thus, given the criticality of the braking action, this task has to be periodically executed on BCU. Let T be its period.

What must be guaranteed?

In particular, the system must ensure that the maximum latency from the time at which an obstacle appears and the time at which the vehicle reaches a complete stop is less than the time to impact.

Equivalently, the distance D of the obstacle from the vehicle must always be greater than the minimum distance L needed for a complete stop.

HRTS – Example ITS …(Cont'd)

Parameters to consider:

Periodic execution by BCU – time period T units (we need to determine a safe value for this parameter)

Distance between the Vehicle and Obstacle – D units;
(We assume that the obstacles are always detected at a distance of D)

Minimum stopping distance for a complete stop – L units;

Velocity (instantaneous) of the vehicle – **v** m/sec;

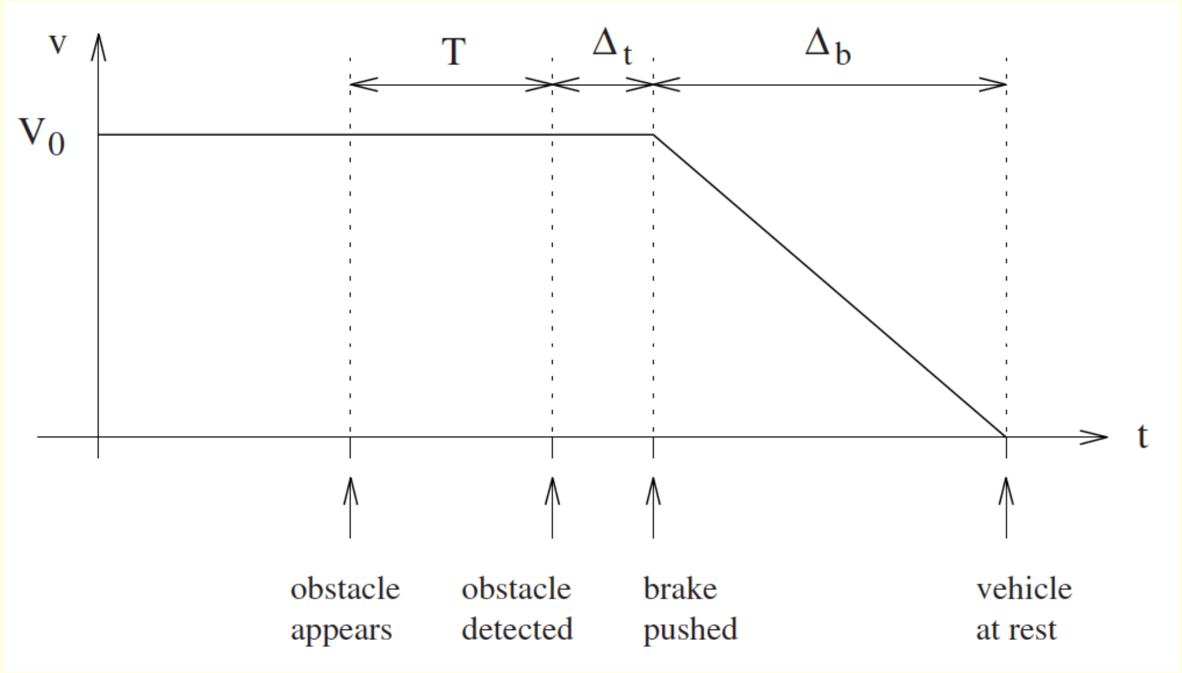
Coefficient of Friction – μ;

HRTS – Example ITS …(Cont'd)

Parameters to consider:

Interval between the time at which the stop command is activated by the BCU and the time at which the command starts to be actuated by the brakes – Δ_t ;

Time interval needed for a complete stop, i.e., the braking duration – Δ_b ;



HRTS – Example ITS …(Cont'd)

The braking duration is given by,

$$\Delta_b = v / (\mu.g), \quad \text{g: accl. due to gravity}$$

This results in a braking space, x_b, given by,

$$x_b = (v.\ \Delta_b)/2 = v^2 / 2.(\mu.g)$$

So the total distance needed for complete stopping:

$$L = v.(T+\Delta_t) + x_b = v.(T+\Delta_t) + v^2 / 2.(\mu.g)$$

Using the constraint D > L, we obtain:

$$T < (D/v) - (v / 2.(\mu.g)) - \Delta_t$$

Let D = 100m
$\mu = 0.5$, $\Delta_t = 250$ ms
v = 30 m/sec
we note that the sampling period of BCU T < 22 msecs

HRTS – Example ITS …(Cont’d)

Using this analysis we can quickly estimate how long we can look away from the road while we are driving at a certain speed and visibility conditions !!

For example, if D = 50 m (visibility under foggy conditions), μ = 0.5, Δ_t = 300 ms (our typical reaction time), and v = 60 km/h (about 16.67 m/s or 37 mi/h), we can look away from the road for no more than one second! (*Verify!*)

Distributed RTS

- Achieving control in a network-based environment – *a grand challenge*!
- Key issue

  - How do I deal with “old information”?

  - How useful is the “old information”?

Based on the status of a specific node, how do I decide about what actions I need to perform in another node?

Example – Task migration under lightly and heavily loaded conditions;

Delay estimation

- Network node input/output parameters:

  Transmission, Propagation, queuing, processing

  Arrival distribution (Poisson/Exp);

  Execution distribution (Poisson/Exp);

Steps to carry out load balancing:

- Delay & number of packets in a node to be estimated for a buddy set;
- initiate job migration;
- Designed algorithm must guarantee on "no cycles"! (*Why?*)

Distributed RTS

- Consider another example of Video/Movie-On-Demand by a Service Provider (SP);

What are the real-time issues here?

What are the real-time constraints?

What happens if there are network delays and how to mitigate them?

Pay attention in the class for the derivation and discussions!

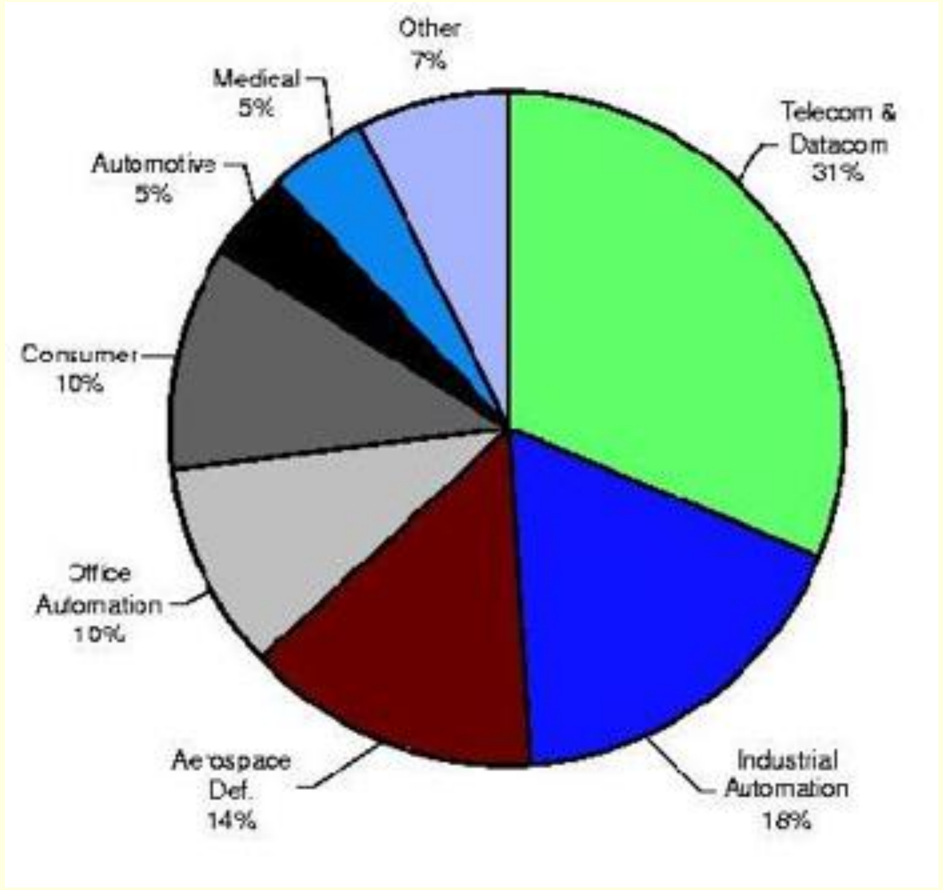
Embedded RTS

- An **embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints

Could be part of a device or a stand-alone system;

Embedded systems often play a crucial role in controlling several devices meeting RT demands

Embedded RTS



80-90% of modern day CPUs deployed are embedded systems;

Embedded RTS

- Almost all real-life applications have **ERTS**
- *Why this trend?* (mid-90s till date)

- Heterogeneity in complex systems

- Demand for miniaturization – Physical size reduction on any real-life product;

- To reduce: Processor manufacturing cost (transistor packing density has significantly increased), Memory cost, and Power reduction;

- Increased s/w design flexibility;

ERTS Applications

- Modern Cars – More than 100 embedded CPUs; Easily accounting 25-30% of total cost;
- Mobile Phones & its avatars!
- Laser Printing / 3D printing
- Coffee-maker!
- Home theater/audio systems
- TV set-top boxes
- Routers on a computer network

Other Issues – Synchronization, Deadlocks, Fault-tolerance, etc

Timing Synchronization:

-- Scheduling

-- Clocks (Ex: Distributed networks)

-- Events

Deadlocks:

-- Competing processes/events to access shared resources

-- Contention – Hot-spots generation which must be avoided

Other issues… (cont'd)

Fault-Tolerance:

-- Reliability measures

-- Robustness

-- Sustaining failures (k out of N failures)

RIAD storage technology is an example;

Multiple Server retrieval technology for MoD / Content Distribution networks

Buzz words/phrases in this domain ☺

Resource sharing

Process Synchronization

Dynamic scheduling

Hard RTs / Soft RTs

"RT needs…"

preemptive

"…time-overlap…"

Mutex

"Jitter-free…"

Semaphore

…

Deadlock

Embedded RTSs

Non-preemptive

Predictable behavior

Critical section

System Utilization

Distributed RTSs

THANK YOU!