ST. XAVIER'S COLLEGE

MAITIGHAR, KATHMANDU

Real Time System

Theory Assignment #1

Submitted by:

Aashish Raj Shrestha

013BSCCSIT002

Submitted to:

Mr. Ganesh Yogi Lecturer, Department of Computer Science St. Xavier's College	

Date of Submission: 24th August, 2016

Table of Contents

1. Digital Controller

Digital control methods have seen practical development over the last two decades. Up-to-date digital controllers have now replaced most conventional analog types. This is due to the fact that controlling a system or a plant using a computer offers great advantages over conventional techniques. These include greater controller flexibility, simpler data processing, superior sensitivity, fewer drift effects, less internal noise, and greater reliability; the units are also cheaper and smaller in size. The major disadvantage in their use is the degree of error introduced during quantization [1].

From the theoretical and design points of view, digital and conventional methods essentially involve the same degree of difficulty. From the applications point of view, digital controllers are usually easier to apply. For this reason they have been successfully applied to almost all categories of control systems, ranging from position control systems to industrial processes, reactors, robots, weapon systems, space applications, and others [1].

1.1. Types of Digital Controller

1.1.1. Sampled Data System

Before the 1950s most control systems were analog, whereas today, most of the systems include a digital computer as their crucial part. Indeed, computer-controlled systems are now a prevalent configuration used in practice [8].

A sampled-data system involves both continuous-time and discrete-time signals in its operation. In fact, most plants and processes found in engineering practice are continuous-time. The altitude and speed of an aircraft, fluid level in a vessel, temperature and pressure in a distillation column and voltage between two nodes in an electronic circuit are a few examples of continuous-time signals measured from various continuous-time plants. ***By controlling a continuous-time plant using a digital controller that operates in a discrete-time environment, we form a sampled-data system.*** [8]

A sampled-data control system therefore consists of a continuous-time plant/process controlled by a digital controller, either as a digital computer or as a simple microchip providing the control action. Consequently, a sampled-data control system is often referred to as computer-controlled system. A general configuration of a sampled-data control system is given schematically in Figure 1. [8]

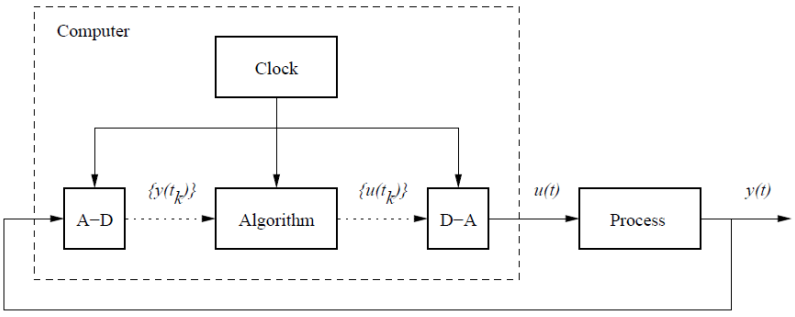


Figure 1: General sampled-data control system configuration.

In Figure 1, the output from the process y(t) is a continuous-time signal. The output is converted into digital form by the analog-to-digital (A/D) converter. The conversion is done at the sampling times, t_k. The control algorithm interprets the converted signal, $\{y(t_k)\}$, to calculate the required control sequence, $\{u(t_k)\}$. This sequence is converted to an analog signal by a digital-to-analog (D/A) converter. All these events are synchronized by a real-time clock in the computer. The digital computer operates sequentially in time, and each operation takes some time. The D/A converter must, however, produce a continuous-time signal. The simplest way to do this is to keep the control signal constant between each conversion time. In this case the system runs open-loop in the time interval between the sampling instants because the control signal is constant, irrespective of the value of the plan output y(t). In practice, the A/D and D/A converters can be parts of the computer or may be built as separate units.

Most plants and processes are nonlinear in nature. While it is possible to use a linear approximation around a prescribed operating point for analysis and controller design, there are many situations when nonlinearities cannot be neglected. Phenomena such as saturation, hysteresis, dead-zone and dry friction are a few examples of common nonlinearities that often arise in practice. In these cases, a nonlinear model is needed to obtain a more accurate representation of the dynamics of the system. A sampled-data control system which includes a nonlinear plant, controlled by either a linear or a nonlinear controller, is classified as a nonlinear sampled-data control system.

There is a wide area of applications for sampled-data control systems, where non-linear phenomena cannot be avoided. These applications range from the man oeuvre control of an aircraft, such as a VTOL aircraft systems, ship or submarine vehicle control, position control for robotic systems in a precision manufacturing process, autonomous vehicle systems, biochemical reactors, power plants and many others. Therefore, control of nonlinear sampled-data systems is an important area of control engineering with a range of potential applications.

In many cases, it is possible to use either analog or digital controllers. However, there are certain situations where the complexity and required flexibility of a system can only be achieved using digital technology.

It typically samples (i.e., reads) and digitizes the analog sensor readings periodically and carries out its control-law computation every period. The sequence of digital outputs thus produced is then converted back to an analog form needed to activate the actuators [2].

Sampling:

If you are driving a car and want to know your speed, you do not have to watch the speedometer continuously. It is sufficient that you monitor the continuously indicated velocity by an occasional glance, and that is "Sampling" [3]. Other areas of sampling are: Stock market quotations are set every working day, and the temperature of a sick person is checked several times per day.

> ***Sampling*** *is a process used in statistical analysis in which a predetermined number of observations are taken from a larger population. The methodology used to sample from a larger population depends on the type of analysis being performed, but may include* ***simple random sampling*** *or* ***systematic sampling*** [4].

Such sampling also occurs in various technical control systems because of the applied measuring procedures [3]. Sampling occurs when the actuator signal in a control system can be changed only at certain time instants. In alternating current rectifiers the ignition angle can be determined only once per period [3].

The term sampled-data systems covers normal analog (continuous-time) systems with the distinctive characteristic that the input r(t) [From Fig. 1.1] and the output y(t) [From figure 1.1] are piecewise constant signals: in other words, they are constant over each interval between two consecutive sampling points. This means that sampled-data systems may be described and subsequently studied similarly to discrete-time systems. This fact is of particular importance because it unifies the study of hybrid systems that consist of continuous-time and discrete-time subsystems using a common mathematical tool, namely the difference equation. For this reason, and for reasons of simplicity, sampled-data systems are usually addressed in the literature as discrete-time systems. It is noted that sampled-data systems are also called discretized systems [1].

A system in which the data appear at one or more points as a sequence of numbers or as pulses is known as a ***sampled-data system*** [6].

Control systems that combine an analog part with some digital components are traditionally referred to as sampled-data systems. Alternative names such as hybrid systems or cyber-physical systems (CPS) have also been used more recently [8].

Sampled-data system operates in continuous time, but some continuous time signals are sampled at certain time instants (usually periodically), yielding discrete-time signals. Sampled-data systems are thus hybrid systems, involving both continuous-time and discrete-time signals [5].

Sampling Instant:

Discrete/Digital and Continuous/Analogue System:

Systems by which signals are recorded, communicated, or displayed may represent the data in **discrete form (**e.g. as integers) or in **continuous form** (as "real" numbers). An important classification results from the choice of discrete or continuous representation of the amplitude, and of discrete or continuous representation of the time at which the amplitude occurred. Analog computers employ physical quantities that are approximations to continuous representations. Discrete representations of both time and amplitude are required by digital computers [7].

A continuous or analogue system has variables which are continuous functions of time i.e. their value is known at all instance of time. Whereas discrete systems have variables which are known only at sampling instants [6].

Sampling instant *can be defined as the time at which a sample was taken from the system.*

An Example:

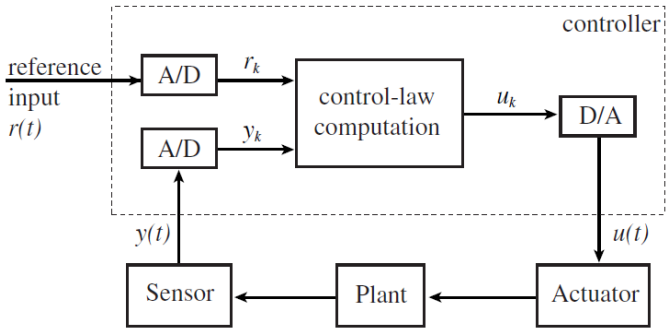


Fig. 1.1: A Digital Controller

As an example we consider an analogue single-input/single-output PID (Proportional, Integral, and Derivative) controller. This simple kind of controller is commonly used in practice. The analogue sensor reading y(t) gives the measured state of the plant at a time t.

Let e(t) = r (t) − y(t) denote the difference between the desired state r (t) and the measured state y(t) at time t . The output u(t) of the controller consists of three terms: a term that is proportional to e(t), a term that is proportional to the integral of e(t) and a term that is proportional to the derivative of e(t).

In the sampled data version, the inputs to the control-law computation are the sampled values y_k and r_k, for k = 0, 1, 2, . . . , which analog-to-digital converters produce by sampling and digitizing y(t) and r (t) periodically every T units of time. $e_k = r_k - y_k$ is the k^{th} sample value of e(t). There are many ways to discretize the derivative and integral of e(t). For example, we can approximate the derivative of e(t) for $(k - 1)T \leq t \leq kT$ by $(e_k - e_{k-1})/T$ and use the trapezoidal rule of numerical integration to transform a continuous integral into a discrete form. The result is the following incremental expression of the k^{th} output u_k:

$$u_k = u_{k-2} + \alpha e_k + \beta e_{k-1} + \gamma e_{k-2} \_\_\_\_\_ (1.1)$$

α, β, and γ are proportional constants; they are chosen at design time.1 During the k^{th} sampling period, the real-time system computes the output of the controller according to this expression. You can see that this computation takes no more than 10–20 machine instructions. Different discretization methods may lead to different expressions of u_k, but they all are simple to compute.

From Eq. (1.1), we can see that during any sampling period (say the k^{th}), the control output u_k depends on the current and past measured values y_i for $i \leq k$. The future measured values y_i's for $i > k$ in turn depend on u_k. Such a system is called a (feedback) control loop or simply a loop.

We can implement it as an infinite timed loop:

Set timer to interrupt periodically with period T;
At each timer interrupt, do
Do analog-to-digital conversion to get y;
Compute control output u;
Output u and do digital-to-analog conversion;
End Do;

Here, we assume that the system provides a timer. Once set by the program, the timer generates an interrupt every T units of time until its setting is cancelled.

Selection of Sampling Period:

The length T of time between any two consecutive instants at which y(t) and r (t) are sampled is called the sampling period. T is a key design choice. The behavior of the resultant digital controller critically depends on this parameter. Ideally we want the sampled data version to behave like the analog version. This can be done by making the sampling period small. However, a small sampling period means more frequent control-law computation and higher processor-time demand. **We want a sampling period T that achieves a good compromise.**

In making this selection, we need to consider two factors. The first is the perceived responsiveness of the overall system (i.e., the plant and the controller). Oftentimes, the system is operated by a person (e.g., a driver or a pilot).

The operator may issue a command at any time, say at t. The consequent change in the reference input is read and reacted to by the digital controller at the next sampling instant. This instant can be as late as t + T. Thus, sampling introduces a delay in the system response. The operator will feel the system sluggish when the delay exceeds a tenth of a second. Therefore, the sampling period of any manual input should be under this limit.

The second factor is the dynamic behavior of the plant. We want to keep the oscillation in its response small and the system under control. To illustrate, we consider the disk drive controller described in [AsWi]. The plant in this example is the arm of a disk. The controller is designed to move the arm to the selected track each time when the reference input changes.

At each change, the reference input r (t) is a step function from the initial position to the final position. In Fig 1.2, these positions are represented by 0 and 1, respectively, and the time origin is the instant when the step in r (t) occurs. The dashed lines in Fig 1.2(a)

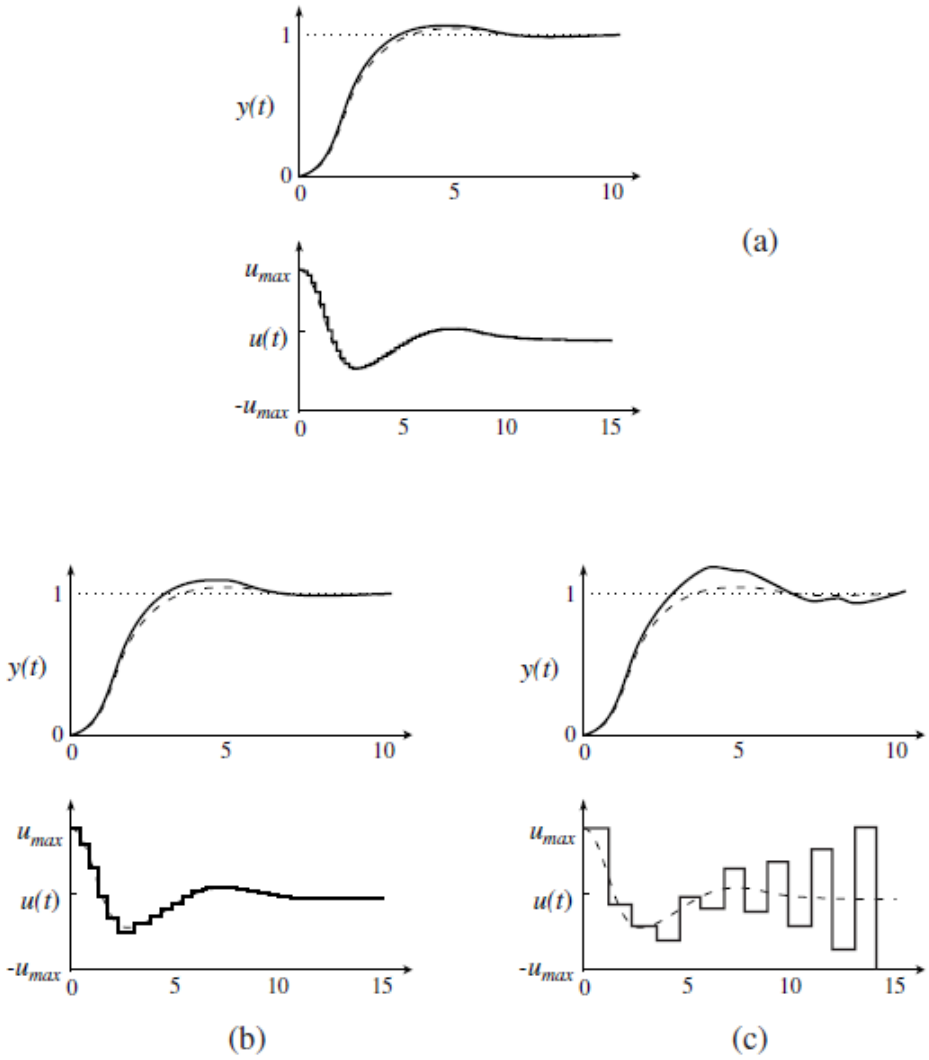


Fig. 1.2: Effects of Sampling Periods

give the output u(t) of the analog controller and the observed position y(t) of the arm as a function of time. The solid lines in the lower and upper graphs give, respectively, the analog control signal constructed from the digital outputs of the controller and the resultant observed position y(t) of the arm. At the sampling rate shown here, the analog and digital versions are essentially the same. The solid lines in Fig. 1.2(b) give the

behavior of the digital version when the sampling period is increased by 2.5 times. The oscillatory motion of the arm is more pronounced but remains small enough to be acceptable. However, when the sampling period is increased by five times, as shown in Fig. 1.2(c), the arm requires larger and larger control to stay in the desired position; when this occurs, the system is said to have become unstable.

In general, the faster a plant can and must respond to changes in the reference input,

the faster the input to its actuator varies, and the shorter the sampling period should be. We can measure the responsiveness of the overall system by its rise time R. This term refers to the amount of time that the plant takes to reach some small neighborhood around the final state in response to a step change in the reference input. In the example in Fig. 1.2, a small neighborhood of the final state means the values of y(t) that are within 5 percent of the final value. Hence, the rise time of that system is approximately equal to 2.5.

A good rule of thumb is the ratio R/T of rise time to sampling period is from 10 to 20 [AsWi, FrPW].2 In other words, there are 10 to 20 sampling periods within the rise time. A sampling period of R/10 should give an acceptably smooth response. However, a shorter sampling period (and hence a faster sampling rate) is likely to reduce the oscillation in the system response even further. For example, the sampling period used to obtain Fig. 1.2(b) is around R/10, while the sampling period used to obtain Fig. 1.2(a) is around R/20.

The above rule is also commonly stated in terms of the bandwidth, ω, of the system. The bandwidth of the overall system is approximately equal to 1/2R Hz. So the sampling rate (i.e., the inverse of sampling period) recommended above is 20 to 40 times the system bandwidth ω. The theoretical lower limit of sampling rate is dictated by Nyquist sampling theorem [Shan]. The theorem says that any time-continuous signal of bandwidth ω can be reproduced faithfully from its sampled values if and only if the sampling rate is 2ω or higher. We see that the recommended sampling rate for simple controllers is significantly higher than this lower bound. The high sampling rate makes it possible to keep the control input small and the control-law computation and digital-to-analog conversion of the controller simple.

1.1.2. Complex Digital Controller

The simplicity of a PID or similar digital controller follows from three assumptions. First, sensor data give accurate estimates of the state-variable values being monitored and controlled. This assumption is not valid when noise and disturbances inside or outside the plant prevent accurate observations of its state. Second, the sensor data give the state of the plant. In general, sensors monitor some observable attributes of the plant. The values of the state variables must be computed from the measured values (i.e., digitized sensor readings). Third, all the parameters representing the dynamics of the plant are known. This assumption is not valid for

some plants. (An example is a flexible robot arm. Even the parameters of typical manipulators used in automated factories are not known accurately.)

When any of the simplifying assumptions is not valid, the simple feedback loop no longer suffices. Since these assumptions are often not valid, you often see digital controllers implemented as follows.

Set timer to interrupt periodically with period *T;*
At each clock interrupt, do
Sample and digitize sensor readings to get measured values;
Compute control output from measured and state-variable values;
Convert control output to analog form;
Estimate and update plant parameters;
Compute and update state variables;
End do;

The last two steps in the loop can increase the processor time demand of the controller significantly. We now give two examples where the state update step is needed.

1.1.2.1. Deadbeat Controller

A discrete-time control scheme that has no continuous-time equivalence is deadbeat control. In response to a step change in the reference input, a deadbeat controller brings the plant to the desired state by exerting on the plant a fixed number (say n) of control commands. A command is generated every T seconds. (T is still called a sampling period.) Hence, the plant reaches its desired state in nT second.

In principle, the control-law computation of a deadbeat controller is also simple. The output produced by the controller during the k^{th} sampling period is given by

$$u_k = \alpha \sum_{i=0}^{k} (r_i - y_i) + \sum_{i=0}^{k} \beta_i x_i$$

[This expression can also be written in an incremental form similar to Eq. (1.1).] Again, the constants α and β_i's are chosen at design time. x_i is the value of the state variable in the i^{th} sampling period. During each sampling period, the controller must compute an estimate of x_k from measured values y_i for $i \leq k$. In other words, the state update step in the above do loop is needed.

1.1.2.2. Kalman Filter

Kalman filtering is a commonly used means to improve the accuracy of measurements and to estimate model parameters in the presence of noise and uncertainty. To illustrate, we consider a simple monitor system that takes a measured value y_k every sampling period k in order to estimate the value x_k of a state variable. Suppose that starting from time 0, the value of this state variable is equal to a constant x. Because of noise, the measured value y_k is equal to $x + \varepsilon_k$, where ε_k is a random variable whose average value is 0 and standard deviation is σ_k. The Kalman filter starts with the initial estimate $\tilde{x}1 = y1$ and computes a new estimate each sampling period. Specifically, for $k > 1$, the filter computes the estimate $\widetilde{x_k}$ as follows:

$$\tilde{x}_k = \tilde{x}_{k-1} + K_k(y_k - \tilde{x}_{k-1}) \qquad (1.2a)$$

In this expression,

$$K_k = \frac{P_k}{\sigma_k^2 + P_k} \qquad (1.2b)$$

is called the Kalman gain and P_k is the variance of the estimation error $\widetilde{x_k} - x$; the latter is given by

$$P_k = E[(\tilde{x}_k - x)^2] = (1 - K_{k-1})P_{k-1} \qquad (1.2c)$$

This value of the Kalman gain gives the best compromise between the rate at which P_k decreases with k and the steady-state variance, that is, P_k for large k. In a multivariate system, the state variable x_k is an n-dimensional vector, where n is the number of variables whose values define the state of the plant. The measured value y_k is an n'-dimensional vector, if during each sampling period, the readings of n' sensors are taken.

We let A denote the measurement matrix; it is an n × n matrix that relates the n' measured variables to the n state variables. In other words,

$$\mathbf{y}_k = \mathbf{A}\mathbf{x}_k + \mathbf{e}_k$$

The vector $\mathbf{e}_k$ gives the additive noise in each of the n' measured values. Eq. (1.2a) becomes an n-dimensional vector equation

$$\tilde{\mathbf{x}}_k = \tilde{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{A}\tilde{\mathbf{x}}_{k-1})$$

Similarly, Kalman gain K_k and variance P_k are given by the matrix version of Eqs. (1.2b) and (1.2c). So, the computation in each sampling period involves a few matrix multiplications and additions and one matrix inversion.

2. Other

2.1.Real Time System

A real-time system is a type of hardware or software that operates with a time constraint [10].

A real-time system is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure [8].

A failed system is a system that cannot satisfy one or more of the requirements stipulated in the formal system specification [8].

Because of this definition of failure, precise specification of the system operating criteria, including timing constraints, is important.

Various other definitions exist for real-time, depending on which source is consulted. Nonetheless, the common theme among all definitions is that the system must satisfy deadline constraints in order to be correct. For example, an alternative definition might be:

A real-time system is one whose logical correctness is based on both the correctness of the outputs and their timeliness [8].

Then from above ***a real-time system is one in which the correctness of the computations not only depends on their logical correctness, but also on the time at which the result is produced*** i.e. a late answer is a wrong answer [9].

For example, many embedded systems are referred to as real-time systems. Cruise control, telecommunications, flight control and electronic engines are some of the popular real-time system applications where as computer simulation, user interface and Internet video are categorized as non-real time applications.

Below we discuss a couple of known real-time systems [9].

Electronic Engine: Here comes a real-time system example. Consider a computer-controlled machine on the production line at a bottling plant. The machine's function is simply to cap each bottle as it passes within the machine's field of motion on a continuously moving conveyor belt. If the machine operates too quickly, the bottle won't be there yet. If the machine operates too slowly, the bottle will be too far along for the machine to reach it. Stopping the conveyor belt is a costly operation as the entire production will come to halt.

Thus the range of motion of the machine coupled with the speed of the conveyor belt establishes a window of opportunity for the machine to put the cap on the bottle. This window of opportunity imposes timing constraints on the operation of the machine. Software applications with these kinds of timing constraints are termed as real-time applications. Here, the timing constraints are in the form of a period and deadline.

2.2.Real Time Operating System

A Real Time Operating System (RTOS) generally contains a real-time kernel and other higher-level services such as file management, protocol stacks, a Graphical User Interface (GUI), and other components. Most additional services revolve around I/O devices.

A real-time kernel is software that manages the time and resources of a microprocessor, microcontroller or Digital Signal Processor (DSP), and provides indispensable services to your applications [13].

In general, an operating system (OS) is responsible for managing the hardware resources of a computer and hosting applications that run on the computer. An RTOS performs these tasks, but is also specially designed to run applications with very precise timing and a high degree of reliability. This can be especially important in measurement and automation systems where downtime is costly or a program delay could cause a safety hazard [12].

A real-time operating system (RTOS) is an operating system that guarantees a certain capability within a specified time constraint [11].

To be considered "real-time", an operating system must have a known maximum time for each of the critical operations that it performs (or at least be able to guarantee that maximum most of the time). Some of these operations include OS calls and interrupt handling.

Operating systems that can absolutely guarantee a maximum time for these operations are commonly referred to as "hard real-time", while operating systems that can only guarantee a maximum most of the time are referred to as "soft real-time".

In general, real-time operating systems are said to require [11]:

- Multitasking
- Process threads that can be prioritized
- A sufficient number of interrupt levels

Real-time operating systems are often required in small embedded operating systems that are packaged as part of micro devices. Some kernels can be considered to meet the requirements of a real-time operating system. However, since other components, such as device drivers, are also usually needed for a particular solution, a real-time operating system is usually larger than just the kernel [11].

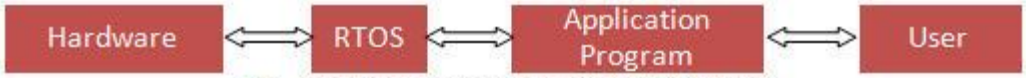


Fig. Real time embedded system with RTOS

As we now know, real-time operating system or embedded operating system is a computer operating system designed to handle events as they occur. Real-time operating systems are

commonly found and used in robotics, cameras, complex multimedia animation systems, communications, and have various military and government uses [14].

Embedded operating systems are stored in a ROM chip instead of a hard drive and contain only the absolutely necessary files for the device it is running on. Because it does not load executable files, it is only capable of running one application at a time.

Many operating systems, such as Windows and Linux, have embedded versions. Other examples include Chimera, Lynx, MTOS, QNX, RTMX, RTX, and VxWorks [14].

In practice, these strict categories have limited usefulness - each RTOS solution demonstrates unique performance characteristics and the user should carefully investigate these characteristics [12].

To fully grasp these concepts, it is helpful to consider an example. Imagine that you are designing an airbag system for a new model of car. In this case, a small error in timing (causing the airbag to deploy too early or too late) could be catastrophic and cause injury. Therefore, a hard real-time system is needed; you need assurance as the system designer that no single operation will exceed certain timing constraints. On the other hand, if you were to design a mobile phone that received streaming video, it may be ok to lose a small amount of data occasionally even though on average it is important to keep up with the video stream. For this application, a soft real-time operating system may suffice.

The main point is that, if programmed correctly, an RTOS can guarantee that a program will run with very consistent timing. Real-time operating systems do this by providing programmers with a high degree of control over how tasks are prioritized, and typically also allow checking to make sure that important deadlines are met.

In contrast to real-time operating systems, the most popular operating systems for personal computer use (such as Windows) are called general-purpose operating systems. While more in-depth technical information on how real-time operating systems differ from general-purpose operating systems is given in a section below, it is important to remember that there are advantages and disadvantages to both types of OS. Operating systems like Windows are designed to maintain user responsiveness with many programs and services running (ensuring "fairness"), while real-time operating systems are designed to run critical applications reliably and with precise timing (paying attention to the programmer's priorities) [12].

2.3.Non Real Time Operating System

Also known as General purpose Operating System. An operating system is a computer program that supports a computer's basic functions, and provides services to other programs (or applications) that run on the computer. The applications provide the functionality that the user of the computer wants or needs. The services provided by the operating system make writing the applications faster, simpler, and more maintainable. If you are reading this web page, then you are using a web browser

(the application program that provides the functionality you are interested in), which will itself be running in an environment provided by an operating system [16].

- In most cases the (soft) real-time aspect may be constructed (e.g. acceptable response time to user input).
- Computer system is backed up by hardware (e.g. end position switches)
- Quite often simply oversized computers.

2.4.Comparisons between RTOS and Non-RTOS [15]

· **Determinism** - The key difference between general-computing operating systems and real-time operating systems is the "deterministic " timing behavior in the real-time operating systems. "Deterministic" timing means that OS consume only known and expected amounts of time. RTOS have their worst case latency defined. Latency is not of a concern for General Purpose OS.

· **Task Scheduling** - General purpose operating systems are optimized to run a variety of applications and processes simultaneously, thereby ensuring that all tasks receive at least some processing time. As a consequence, low-priority tasks may have their priority boosted above other higher priority tasks, which the designer may not want. However, RTOS uses priority-based preemptive scheduling, which allows high-priority threads to meet their deadlines consistently. All system calls are deterministic, implying time bounded operation for all operations and ISRs. This is important for embedded systems where delay could cause a safety hazard. The scheduling in RTOS is time based. In case of General purpose OS, like Windows/Linux, scheduling is process based.

· **Preemptive kernel** - In RTOS, all kernel operations are pre-emptible

· **Priority Inversion** - RTOS have mechanisms to prevent priority inversion

· **Usage** - RTOS are typically used for embedded applications, while General Purpose OS are used for Desktop PCs or other generally purpose PCs.

Other Differences [12]:

Operating systems such as Microsoft Windows and Mac OS can provide an excellent platform for developing and running your non-critical measurement and control applications. However, these operating systems are designed for different use cases than real-time operating systems, and are not the ideal platform for running applications that require precise timing or extended up-time. This section will identify some of the major under-the-hood differences between both types of operating systems, and explain what you can expect when programming a real-time application.

Setting Priorities

When programming an application, most operating systems (of any type) allow the programmer to specify a priority for the overall application and even for different tasks within the application

(threads). These priorities serve as a signal to the OS, dictating which operations the designer feels are most important. The goal is that if two or more tasks are ready to run at the same time, the OS will run the task with the higher priority.

In practice, general-purpose operating systems do not always follow these programmed priorities strictly. Because general-purpose operating systems are optimized to run a variety of applications and processes simultaneously, they typically work to make sure that all tasks receive at least some processing time. As a result, low-priority tasks may in some cases have their priority boosted above other higher priority tasks. This ensures some amount of run-time for each task, but means that the designer's wishes are not always followed.

In contrast, real-time operating systems follow the programmer's priorities much more strictly. On most real-time operating systems, if a high priority task is using 100% of the processor, no other lower priority tasks will run until the high priority task finishes. Therefore, real-time system designers must program their applications carefully with priorities in mind. In a typical real-time application, a designer will place time-critical code (e.g. event response or control code) in one section with a very high priority. Other less-important code such as logging to disk or network communication may be combined in a section with a lower priority.

Interrupt Latency

Interrupt latency is measured as the amount of time between when a device generates an interrupt and when that device is serviced. While general-purpose operating systems may take a variable amount of time to respond to a given interrupt, real-time operating systems must guarantee that all interrupts will be serviced within a certain maximum amount of time. In other words, the interrupt latency of real-time operating systems must be bounded.

Performance

One common misconception is that real-time operating systems have better performance than other general-purpose operating systems. While real-time operating systems may provide better performance in some cases due to less multitasking between applications and services, this is not a rule. Actual application performance will depend on CPU speed, memory architecture, program characteristics, and more.

Though real-time operating systems may or may not increase the speed of execution, they can provide much more precise and predictable timing characteristics than general-purpose operating systems.

2.5.RTS Types

RTOS specifies a known maximum time for each of the operations that it performs. Based upon the degree of tolerance in meeting deadlines, RTOS are classified into following categories [15]:

· **Hard real-time:** Degree of tolerance for missed deadlines is negligible. A missed deadline can result in catastrophic failure of the system

· **Firm real-time:** Missing a deadly ne might result in an unacceptable quality reduction but may not lead to failure of the complete system

· **Soft real-time:** Deadlines may be missed occasionally, but system doesn't fail and also, system quality is acceptable

For a life saving device, automatic parachute opening device for skydivers, delay can be fatal. Parachute opening device deploys the parachute at a specific altitude based on various conditions. If it fails to respond in specified time, parachute may not get deployed at all leading to casualty. Similar situation exists during inflation of air bags, used in cars, at the time of accident. If airbags don't get inflated at appropriate time, it may be fatal for a driver. So such systems must be hard real time systems, whereas for TV live broadcast, delay can be acceptable. In such cases, soft real time systems can be used.

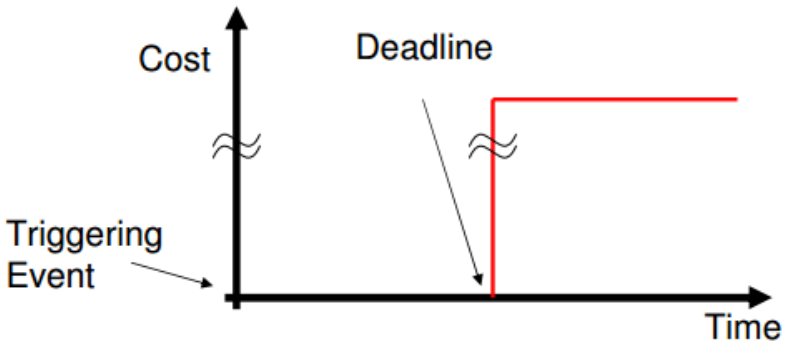
2.5.1. Hard Real Time System

• An overrun in response time leads to potential loss of life and/or big financial damage

• Many of these systems are considered to be safety critical.

• Sometimes they are "only" mission critical, with the mission being very expensive.

• In general there is a cost function associated with the system.



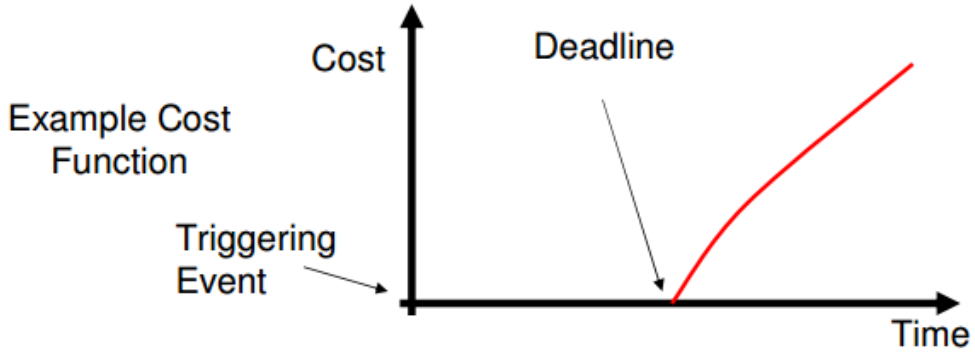
2.5.2. Soft Real Time System

• Deadline overruns are tolerable, but not desired.

• There are no catastrophic consequences of missing one or more deadlines.

• There is a cost associated to overrunning, but this cost may be abstract.

• Often connected to Quality-of-Service (QoS).

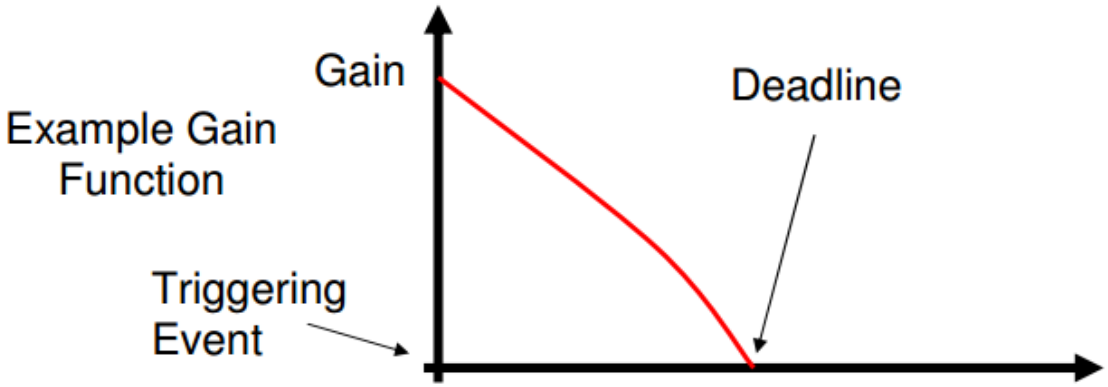


2.5.3. Firm Real Time System

• The computation is obsolete if the job is not finished on time.

• Cost may be interpreted as loss of revenue.

• Typical example are forecast systems.



2.5.4. Weak Hard Real Time System

• Systems where m out of k deadlines have to be met.

• In most cases feedback control systems, in which the control becomes unstable with too many missed control cycles.

• Best suited if system has to deal with other failures as well (e.g. Electro Magnetic Interference EMI).

• Likely probabilistic guarantees sufficient.

References

[1] Paraskevopoulos P.N., "Digital Control System," in *CONTROL SYSTEMS, ROBOTICS, AND AUTOMATION,* Vol. II, Ed. [ONLINE]. Available: http://www.eolss.net/sample-chapters/c18/e6-43-04.pdf 1 [Accessed: 22- August- 2016].

[2] Jane W. S. Liu, "Sampled Data System," in *Real Time System,* 2000, pp. 2

[3] J. Ackermann, "Sampling, Sampled-Data Controllers," in *Sampled-Data Controlled System: Analysis and Synthesis, Robust System Design,* New York*, Ed. 1985, pp. 1*

[4] "Sampling", investopedia.com, 2016. [Online]. Available: http://www.investopedia.com/terms/s/sampling.asp [Accessed: 22-August-2016]

[5] Tongwen Chen, Bruce Francis, "Sample Data System" in *Optimal Sample-Data Control Systems*, London, 1st Ed. 1995, pp. 1-8

[6] Gene F. Franklin, John R. Ragazzini, "Introduction," in *Sampled Data Control System*, New York, Ed. 1958, pp. 1-3

[7] "discrete and continuous systems", encyclopedia.com, 2016. [Online]. Available: http://www.encyclopedia.com/doc/1O11-discreteandcontinussystms.html [Accessed: 22-August-2016].

[8] Phillip A. Laplante, "Real-Time Definitions," in *REAL-TIME SYSTEMS DESIGN AND ANALYSIS, NJ 07030,* 3rd Ed., 2004, pp. 4-7

[9] "Real Time Systems - An Overview", peterindia.net, 2016, [Online]. Available: http://www.peterindia.net/Real_timeSystems.html [Accessed: 24-August-2016].

[10] "real-time system", whatis.techtarget.com, 2016, [Online]. Available: http://whatis.techtarget.com/definition/real-time-system [Accessed: 24-August-2016].

[12] "What is a Real-Time Operating System (RTOS)?", ni.com, 2016, [Online]. Available: http://www.ni.com/white-paper/3938/en [Accessed: 24-August-2016].

[13] "What is a Real-Time Operating System?", micrium.com, 2016, [Online]. Available: https://www.micrium.com/rtos/what-is-an-rtos [Accessed: 24-August-2016].

[14] "Real-time operating system", computerhope.com, 2016, [Online]. Available: http://www.computerhope.com/jargon/r/realsyst.htm [Accessed: 24-August-2016].

[15] "RTOS - Real Time Operating System", engineersgarage.com, 2016, [Online]. Available: http://www.engineersgarage.com/articles/rtos-real-time-operating-system [Accessed: 24-August-2016].

[16] “What is a General Purpose Operating System?”, freertos.org, 2016, [Online]. Available: http://www.freertos.org/about-RTOS.html [Accessed: 24-August-2016].

[17] “Real-Time Systems”, cse.unsw.edu.au, 2016, [Online]. Downloadable: http://www.cse.unsw.edu.au/~cs9242/08/lectures/09-realtimex2.pdf [Accessed: 24-August-2016].

[18] “full solution manual real time system by jane w s liu solution manual”, scribd.com , 2016, []. Available: https://www.scribd.com/document/183328015/full-solution-manual-real-time-system-by-jane-w-s-liu-solution-manual-pdf [Accessed: 24-August-2016].