

## Foundation of Cyber Physical Systems

Prof. Soumyajit Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Module No # 05

Lecture No # 25

Delay-Aware Design; Platform Effect on Stability/Performance (Contd.,)


Hello welcome back to this lecture series on foundations of cyber physical systems. So let us continue on our attempt of discretizing continuous plant models. So we will just start from where we left off here.

(Refer Slide Time: 00:42)

**Discretization**

- The value of  $u(t)$  changes only at every sampling period, i.e.  
 $u(t) = u(t_k)$ , for  $t \in (t_k, t_{k+1})$
- Therefore,  
$$\int_{t_k}^t e^{-At} Bu(t) dt = Bu(t_k) \left[ \frac{e^{-At}}{-A} \right]_{t_k}^t$$
- Hence,  
$$e^{-At} x(t) - e^{-At_k} x(t_k) = Bu(t_k) \left[ \frac{e^{-At}}{-A} \right]_{t_k}^t = \frac{Bu(t_k)}{-A} (e^{-At} - e^{-At_k})$$
  
$$\Rightarrow x(t) = e^{A(t-t_k)} x(t_k) + \frac{Bu(t_k)}{-A} (e^{-A(t-t_k)} - 1)$$

*Handwritten notes:*  
Multiply both sides with  $e^{At}$   
 $e^{A(t-t_k)} - 1$



So one small issue in this derivation that we have been doing. So you see these were the limits of this integral right and so this negative sign is already accounted for here because the  $t_k$  term comes first. So this would not be here and of course we have multiplied everything with  $e$  to the power  $At$  on both the sides right. So that would mean fine you had this limits of  $e$  to the power I mean the initial term was  $-At$  here so you have a minus here and once you multiply both sides.

So we take the power it this becomes 1 and here of course you have it then the resulting expression here is e to the power A (t - t<sub>k</sub>) right -1 because of this multiplying of both sides e to the power A t. So this is 1 and the other effect is well you are writing -A t<sub>k</sub> first. So this subtraction here this minus sign is off and due to this multiplying on both sides here we have t - t<sub>k</sub> so this sign is also not there and with this if we just continue to the next slide.

(Refer Slide Time: 02:47)

**Discretization**

- Let,  $t_k = kh$  and  $t = (k+1)h$ . Therefore

$$x((k+1)h) = e^{Ah}x(kh) + \frac{Bu(kh)}{A}(e^{Ah} - 1) = \phi x(kh) + \Gamma u(kh)$$

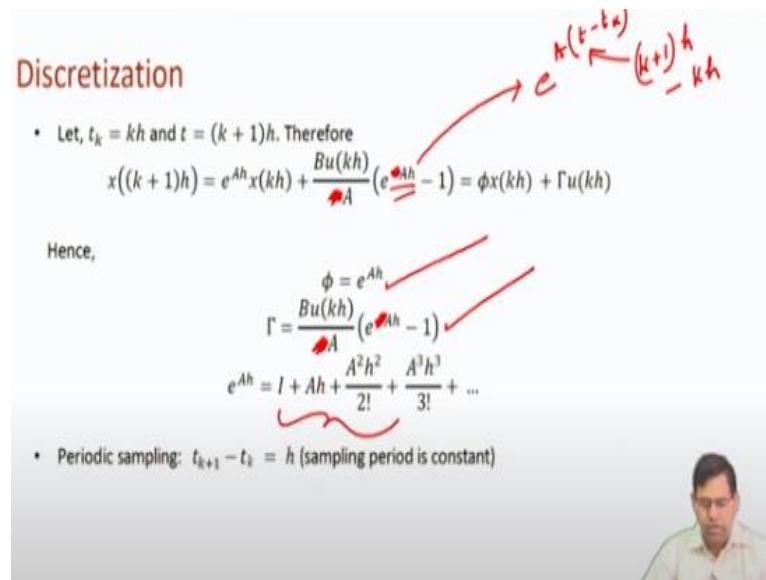
Hence,

$$\phi = e^{Ah}$$

$$\Gamma = \frac{Bu(kh)}{A}(e^{Ah} - 1)$$

$$e^{Ah} = I + Ah + \frac{A^2h^2}{2!} + \frac{A^3h^3}{3!} + \dots$$

- Periodic sampling:  $t_{k+1} - t_k = h$  (sampling period is constant)



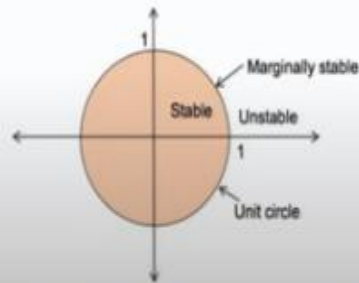
So you will have this. Now let us understand how this e to the power h comes will just explain once again. So if you recall from the last slide you had e to the power A (t - t<sub>k</sub>) here now of course your t becomes like we discussed already k + 1 times h and t<sub>k</sub> is k h so you have h remaining that is e to the power h right. So again we will follow up the correction here. So this is your Phi which is e to the power h and this is your gamma B u(kh) by A and u k h by A e to the power h-1.

So these are kind of standard derivations from the continuous to the discrete time right you can find in most of the popular books for example one reference is the book on a digital controller implementations by Astrom. You can find this done in a similar method in that book only. So like we said that for a given system if we want to follow carry forward this discretization you can even approximate this e series with the first few terms and just evaluate what is  $\Phi$  and what is  $\Gamma$  and carry on from there. So fine with this if we just continue.

(Refer Slide Time: 04:18)

## Discrete-time System Stability

- Stable system  
⇒ Absolute values of all poles *lesser than unity*
- Marginally stable system  
⇒ Absolute values of one or multiple poles *are unity*
- Unstable system  
⇒ Absolute values of one or more poles are *greater than unity*



The next thing we have already talked about is the notion of stability in this grid time. So in this case your systems you have to interpret the poles and the zeros in the Z domain. And here you have to just check whether the poles are located inside, on or outside the unit circle and accordingly you will infer whether the system is stable, marginally stable or unstable.

(Refer Slide Time: 04:42)

## Control Design: Discrete Domain

- Given system
- Objective:
  1. Place system poles
  2. Achieve  $y \rightarrow r$  as  $t \rightarrow \infty$
  3. Design  $K$  and  $F$
- Control Law:  $u[k] = Kx[k] + Fr$

$$\begin{aligned} x[k+1] &= \phi x[k] + \Gamma u[k] \\ y[k] &= Cx[k] \end{aligned}$$

$$\begin{aligned} x(t) &\sim A, B \\ \phi &= e^{Ah} \\ \Gamma &= \int_0^h e^{A\tau} B d\tau \end{aligned}$$

So now let us come to the issue of where we have successfully understood that given the system in the continuous domain with parameters  $A$  and  $B$  how to transform it to  $\Phi$  and  $\Gamma$  right and  $B$  and this gamma equal to if you recall from this slide. So it was  $B u$  by  $A e$  to the power  $h-1$ ,  $u$  at  $k h$  of course and based on that we can derive  $\Phi$  and  $\Gamma$  and then you have this discrete time model. Here

and for this discrete time model let us be the case that we want to design the controller that means we want to place the system poles in a suitable location on the Z plane.

So that we achieve this similar thing  $y$  is tending towards  $r$  as  $t$  tends to infinity and you want to design the  $K$  and  $F$  values. So that you have a controller like this. Again I will just repeat the sign depends on how you model the system if you consider  $u = -Kx + Fr$  that is fine. That is how you it will be in the flow diagram and you can just proceed accordingly so this does not really matter.

(Refer Slide Time: 06:02)

### What is Controllability?

An LTI system is considered to be **controllable** if, for any initial state  $x[0]$  a control input  $u(t)$  can be designed such that within a finite time  $t$  the state of the system reaches any final state  $x[t]$

$$\begin{aligned} x[k+1] &= \phi x[k] + \Gamma u[k] \Rightarrow x[k+1] = \phi^2 x[k-1] + \phi \Gamma u[k-1] + \Gamma u[k] = \dots \\ &= \phi^{k+1} x[0] + \phi^k \Gamma u[0] + \dots + \phi \Gamma u[k-1] + \Gamma u[k] \end{aligned}$$

$$\Rightarrow x[k+1] - \phi^{k+1} x[0] = \begin{bmatrix} \Gamma & \phi \Gamma & \phi^2 \Gamma & \dots & \phi^k \Gamma \end{bmatrix} \begin{bmatrix} u(k) \\ u(k-1) \\ \vdots \\ u(0) \end{bmatrix}$$

- Above system of equation has a solution if  $\begin{bmatrix} \Gamma & \phi \Gamma & \phi^2 \Gamma & \dots & \phi^k \Gamma \end{bmatrix}$  has full row rank
- $\begin{bmatrix} \Gamma & \phi \Gamma & \phi^2 \Gamma & \dots & \phi^k \Gamma \end{bmatrix}$  is called controllability matrix
- The intuition is to check whether the input is rich enough to determine the state : **Observe**



So the first thing we talked about controllability in our previous attempt of basic controller design in the continuous domain right so just getting back to that here also you will need to do the controllability check. But let us also understand what do we really mean by controllability. So an LTI system will be considered controllable if for any initial state. Let us say it is  $x(0)$  right for the initial state and there is a control input. So you have the initial state as  $x(0)$  the question is whether you can design this sequence of controls that is a control input  $u$ ,  $u(t)$  so that you can ensure that your system is eventually steering towards a final state  $x(t)$  that satisfies your requirement of the target trajectory.

So in a nutshell we can say that it is controllable if for any initial state  $x(0)$  control input  $u(t)$  can be designed such that inside this finite timeline  $t$  the state of the system reaches the target final state  $x(t)$  which is let size near about to some reference  $r$ . I mean it is really controllable

in that way. So I mean so the question is why do you really want to do this test? You will like to do this test in order to justify whether I should really attempt the controller design problem or not right. So the way we arrive at this controllability matrix is something like this so you just unroll  $x(k+1)$ . So for  $x(k)$  you have this difference equation right.

So be it in the continuous domain or be it in the discrete domain I mean you can just try this in both cases. So if you do it in continue discrete domain so that would mean just keep on applying this equation for different values of  $k$  right. So you have this relating  $x(k+1)$  and  $x(k)$  right now you can just unroll  $x_k$  that means you can represent it in terms of  $x(k-1)$  then  $x(k-2)$  like that. So for example  $x_k$  if you unroll you have this expression right because this is nothing but so that would give you  $\Phi^2 x(k-1) + 2\Gamma u(k-1) + \Gamma u(k)$ . This  $u(k)$  the  $\Gamma u(k)$  which is the last term here right and in this way if you keep on unrolling and if you go all the way up to up to  $x_{naught}$  and  $u_{naught}$  right.

Then you get this sequence right so you have  $\Phi^{k+1} x(0) + \Phi^k \Gamma u(0)$  in that way you can finally have this thing as  $\Gamma u(k)$  right. So you have  $\Gamma u(k)$  then  $\Phi \Gamma u(k-1)$  and so on so forth right. So what you can do is well let us bring this  $x$  term on the right hand side so you have  $x(k+1) - \Phi$  to the power  $k+1 x(0)$  equal to now the rest of the expression that you have you see you can just write it as a dot product of this vector. This Vector which is comprising only the  $\Phi$  and  $\Gamma$  terms right because this is gone to this side so what remains is well if you look from this and this side.

So you have  $\Gamma u(k)$  then you have  $\Phi \Gamma u(k-1)$  similarly the previous term here would be  $\Phi^2 \Gamma u(k-2)$  in this way at the last you will have  $\Phi$  to the power  $k \Gamma u(0)$  right. So this is the representation and if you see what this really means that well if this matrix will have a full rank that would mean that will you can really design this system of you use right from  $u(0)$  to  $u(k)$  and this really would make a set of possible solutions for  $x(k+1)$  right. So that is why we will say that this system has a solution if this matrix starting from  $\Phi$  to so  $\Gamma$  to  $\Phi^k \Gamma$  this has a full rank and that is why this is called the controllability matrix.

So the idea is that well if you can check controllability and you see that the matrix has full rank then this  $u_k$  actually exists and otherwise this system of equation becomes trivial and it is futile

to go about designing the control inputs in that case. So fine this is about controllability which we talked about earlier and just to kind of appraise you there is also a dual characteristics which is known as observability where the intuition is to check whether the input is rich enough to determine the state.

(Refer Slide Time: 11:31)

## What is Observability?

An LTI system is considered to be **observable** if, for any final output  $y[t]$  over a finite time interval  $t \in [0, kh]$  is enough to reconstruct its unique initial state  $x[0]$  (if required, for a given control inputs  $u[0] \dots u[t]$ )

$$y[k] = Cx[k] = C\phi^k x[0] + C\phi^{k-1}\Gamma u[0] + \dots + C\phi\Gamma u[k-2] + C\Gamma u[k-1]$$

$$y[k-1] = C\phi^{k-1}x[0] + C\phi^{k-2}\Gamma u[0] + \dots + C\phi\Gamma u[k-3] + C\Gamma u[k-2]$$

$$\dots y[0] = C\phi^0 x[0]$$

$$\Rightarrow \begin{bmatrix} C \\ C\phi \\ \vdots \\ C\phi^k \end{bmatrix} x[0] = \begin{bmatrix} y[0] \\ y[1] - C\Gamma u[0] \\ \vdots \\ y[k] - C\phi^{k-1}\Gamma u[0] - \dots - C\Gamma u[k-1] \end{bmatrix}$$

- Above system of equation has unique solution if observability matrix

$$\begin{bmatrix} C \\ C\phi \\ \vdots \\ C\phi^k \end{bmatrix}$$

has full column rank

- The intuition is to check whether the output is rich enough to determine the state

So just like controllability the dual properties that will consider the system to be observable if for any final input over a time interval. So you have this final input  $y$   $t$  over this time interval you say that it is enough to reconstruct its unique initial state. So suppose you know  $y(t)$  the question is given this whether you can I mean if giving the final input it is a good enough information from which you can really reconstruct its state now. If you can look at a similar sequence of equations for  $y(k)$  so you can express  $y(k)$  in terms of  $x(k)$  and similarly you can write  $x(k)$ . You can just write this expression of  $x(k)$  unrolled up to  $x(0)$  and  $u$  and this  $u(0)$  up to  $u(k-1)$  like we discussed earlier and similarly you can write that for  $y(k-1)$  and up to  $y(0)$  right.

From this system of equations you can write this representation right so you have this times  $x(0)$  equal to this matrix comprising the sequence of  $y$  zeros right and this will also have a unique solution. If this matrix that you have uncovered here right and that has a full rank. So this tells you that will from the observability point is that well if I know the final output the question is whether it is possible that there exists an initial state from which the system eventually steers towards the final output. So you can observe that kind of states to be really true and with that it eventually

comes to the final output. But anyway for our control system design problem let us go back to the context, we are talking about controllability check.

**(Refer Slide Time: 13:21)**

## Control Design: Discrete Domain

Control Law:  $u[k] = Kx[k] + Fr$

1. Check controllability of  $(\phi, \Gamma) \rightarrow$  must be controllable.  $\gamma$  must be invertible where,

$$\gamma = [\Gamma \quad \phi\Gamma \quad \phi^2\Gamma \dots \phi^{n-1}\Gamma]$$

2. Choose the desired closed-loop poles at

$$[\alpha_1 \quad \alpha_2 \quad \alpha_3 \dots \alpha_n]$$

3. Apply Ackermann's formula:

$$K = -[0 \ 0 \ 0 \dots 1]\gamma^{-1}H(\phi)$$

$$H(\phi) = (\phi - \alpha_1 I)(\phi - \alpha_2 I) \dots (\phi - \alpha_n I)$$

4. Feedforward gain:  $F = \frac{1}{C(I - \phi - \Gamma K)^{-1}\Gamma}$

So what we need to do is well for our problem we need to do the controllability check and then we will do just the similar thing like earlier. We want to choose the closed loop poles at some locations and we will apply the Ackermann's formula like we discussed earlier. But now note that what we are doing is we are doing it all for the discretized discrete domain plant model right. And then you apply the Ackermann's formula and you apply the feed forward gain formula and well you have the controller's design. So it all remains the same and all you are doing is you are doing it on the discretized plant model so you just get a controller for the discrete plant model.

So fine with this we will end our treatment here and we now move on to the second part where we will consider situations like well how when you are executing the controller there may be some imprecisions or in a I mean uncertainties in the system in terms of timing and how those things can disturb your design and whether your system design actually accounts for that. So what we essentially mean is well you have the controller for a discretized plan so your controller is now a software which is fine but now the software is going to run as a task and in a real time processor. So there may be delays in the scheduling there may be delays due to other tasks.




So due to such delays the controller may not be executing and providing control outputs exactly at points when you want them to be there. So due to such delays whether your system's design is not satisfying the desired control performance criteria is something you want to know about. And so how do I how do I analyze that and how I can design a controller which will take care of such delays and it will work and meet its desired target objective inspite of such delays.

So that is precisely what we will be looking for in this part of the coverage. So some of these materials have been taken from Professor Samarjit Chakraborty's course of course we have adapted it for our purpose. So let us first talk about this idea that well why such delays should eventually come in a control loop.

**(Refer Slide Time: 15:37)**

## Embedded Controller Implementation

- A sensor task ( $T_m$ ) reads sensor data and process them to extract state information. Typically, A/D conversion and signal/image processing are performed in this task. 
- A controller task ( $T_c$ ) implements the control law and computes the control input. The execution time of this task depends on the complexity of the control algorithm.
- An actuator task ( $T_a$ ) writes the control input onto the actuator to be applied to the plant. Typically, D/A conversion and post-processing is done in this task.



So let us understand that how embedded controllers are really implemented some of this we have already seen from our previous coverage on real time scheduling of multiple tasks together. So when you are implementing a control task you are also going to have some associated tasks. So let us say you have a sensor task that is going to read some sensor data and process them to extract some state information. So there can also be some A2D conversion so there will also be some image processing tasks maybe some estimation tasks etc etc., right some filters which are implemented as software all these tasks will be implemented and there will be running in the real time.



Well there also will be some controller tasks which implement the actual control law and which will compute the control input. And note that control tasks not all controllers may be simple pole placement based controllers. They can be let us say a data driven controller they can be an optimal controller. So it is not that all the controllers are very simple and they do not take much time to execute some of the controllers may be quite intensive in terms of their computation that is why they may give delays to some other tasks other control tasks.

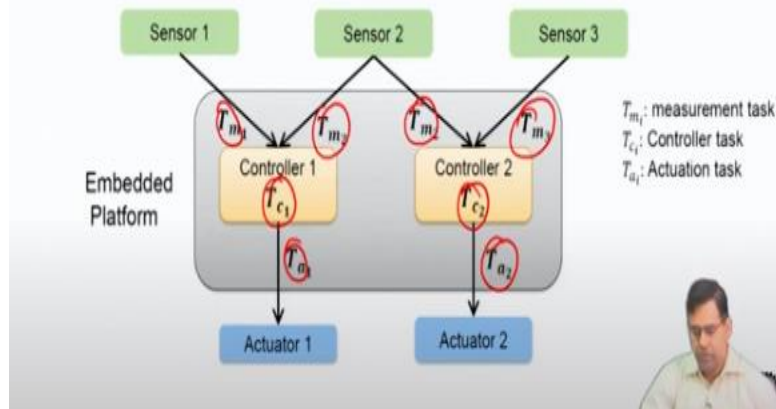
And also the control input has to be sent to the actuator where and for this conversion from the control input to a real signal that the actuator circuit would understand you may have specialized software right. So for that you will have separate tasks. So let us so for example when we talk about sensing tasks so well A2D conversion is part of the hardware job but that signal processing, filtering, estimation these are typically the tasks which are performed on the raw sensor data and again this is the hardware task and for the actuator side.

Like I said that you have computed a control input but when it is going to reach the actual actuator it may be some mechanical device right and that may need some conversion here from this control values to some suitable converted values which this mechanical devices input we are going to understand right. So you will also have a separate software logic which will do all those stuff and that is what we can abstract out here in our analysis as a actuator task. So now consider this example. So note one thing that we are not talking about in a real example here right.

So we are just taking an abstraction, we are saying that well a sensor task will do this this functionality and actuator task may have this kind of functionality but of course in real life these things will vary based on the application right. So all we are doing is we are abstracting things out and we are trying to see that how their timings are going to impact the mathematical model of the controller here.

**(Refer Slide Time: 18:28)**

## Embedded Controller Implementation



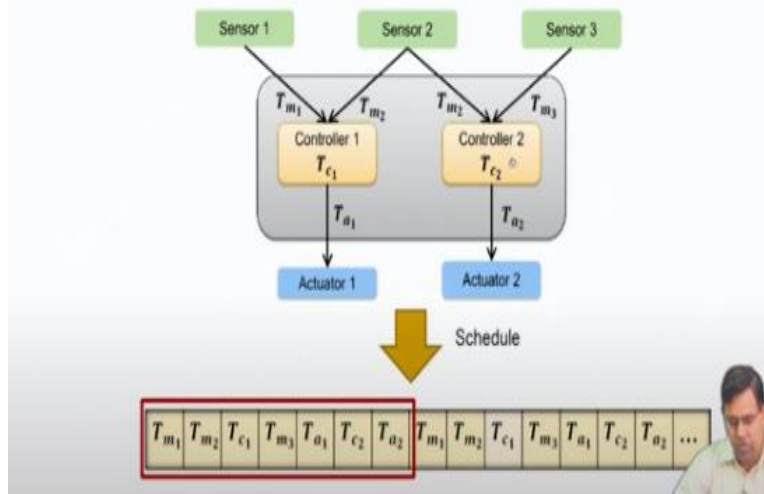
So let us take a scenario. We have let us say a task graph we have multiple sensors right and they are providing inputs to multiple controllers here. So let us say sensor 1 and sensor 2 they are corresponding to some sensing tasks right and those sensing tasks once those are done they lead to some control tasks. So sensor 1 and sensing to their processing will must precede control task  $T_{c_1}$  and once the control task is done there will be a corresponding actuation task and then the output flows to actuator.

So if you remember our discussions on scheduling we talked about actuations and communication from the actuation signal there would be related communication task etc., also. So coming here let us say this sensor is a, is actually sampling some common information for multiple controllers so from this you have sensing tasks also specific to controller 2. And then controller 2 will execute and then the actuation task corresponding to controller 2 will also execute and all these tasks must be executing on the let us say a shared processor which is an embedded platform here.

So you have all these tasks that need to execute and all these dependence relations need to be satisfied. So for doing all that you needs scheduling so there must be a scheduler on this embedded platform which will schedule all these tasks in a perfect in a particular sequence without breaking their dependencies right.

**(Refer Slide Time: 20:01)**

## Embedded Controller Implementation

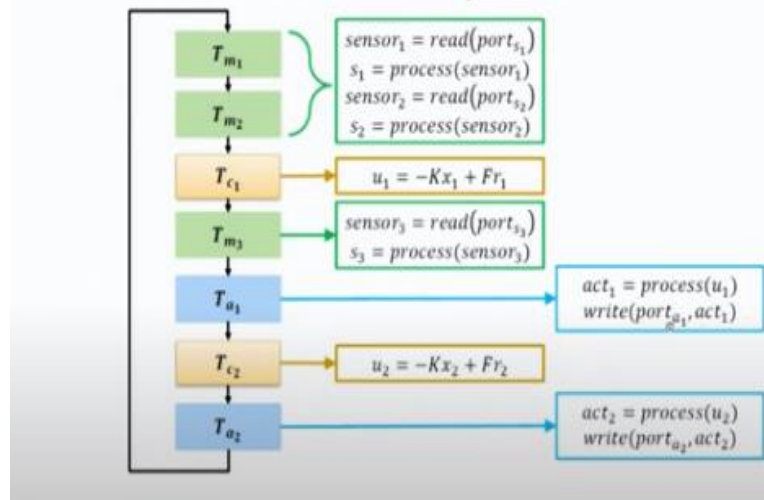


So let us say we consider this sample schedule. So let us say well  $T_{m_1}$  and  $T_{m_2}$  must be say  $T_{c_1}$  and  $T_{m_1}$  and  $T_{m_2}$  can execute any order. So let us say this is the order we have chosen followed by  $T_{c_1}$  and then let us say  $T_{m_2}$  and  $T_{m_3}$  is executing here let us say now and then well after  $T_{c_1}$  has executed only then  $T_{a_1}$  can execute. So let us say we put  $T_{a_1}$  here and then  $T_{c_2}$  here and  $T_{a_2}$  here right.

And of course there is no particular ordering required between  $T_{c_1}$  and  $T_{c_2}$ . But let us say this is our sequence that  $T_{c_1}$  comes here  $T_{c_2}$  comes here  $T_{m_3}$  is here and like that. Let us say that well this is the schedule that keeps on repeating. So this is the schedule that keeps on repeating here this up to this and then again as you can see this is just repeating.

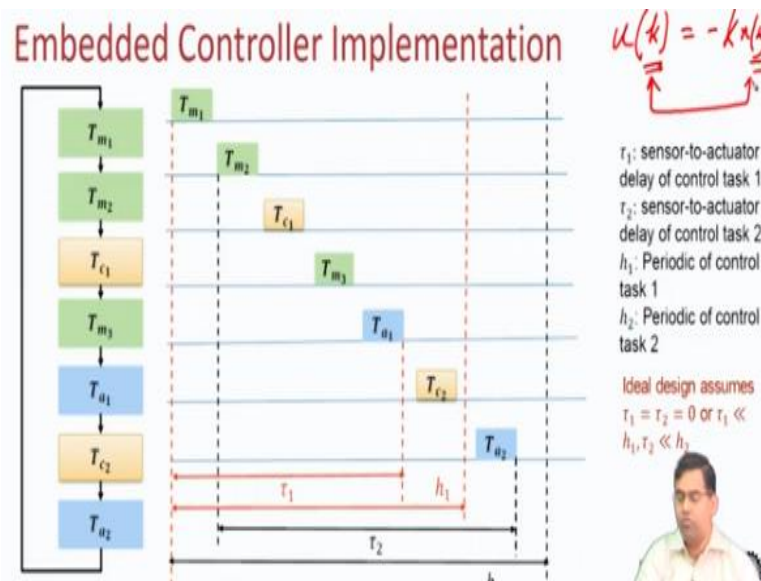
**(Refer Slide Time: 20:56)**

## Embedded Controller Implementation



So on the embedded processor what you have is you have this task square sequence so let us say this is a pure static scheduling. That means you have this task sequence and they are just executing in this kind of a particular order right. So let us say the measurement or sensing tasks you have their corresponding code is sensor 1 reading something from a port then doing some processing then sensor 2 reading from a port and doing some processing this control task will execute the corresponding control algorithm. Then there is the sensing transport 3 the third sensor then some actuation work that is sending some values to the some specific port and before that doing some processing something like that and then you have this control task and the other actuation task etc.

(Refer Slide Time: 21:46)



So let us say this is the last sequence that is going to execute and let us say you have one executing after another and this is your timing diagram of these tasks. Now if you look at these different timing values we have something marked as  $\tau_1$ . So  $\tau_1$  is what we call as the sensor to actuation delay of  $\tau_1$ . So once some value is sensed the corresponding actuation value is computed and it is ready for the actuator after this interval. So that is the sensing the sensor to actuator delay similarly  $\tau_2$  is the sensor to actuator delay of the control task 2 and here you let us say you have the period of some control task.

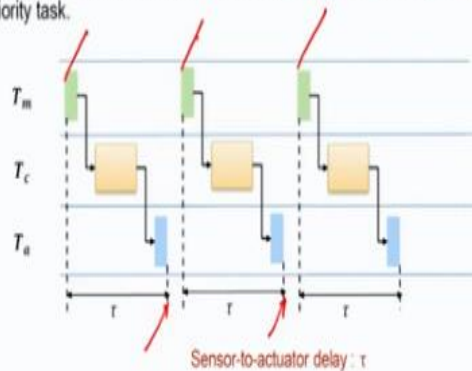
So you have this as the period for  $\tau_2$  and similarly you have this for the period of task 1. So here you have the period of task 1 that means after this the inside another amount of  $h_1$  task 1 must repeat like that. So when you are doing your mathematical modeling and design of controllers we make one important assumption. So what we assume is that well these delays are very small or negligible and we almost set them to 0. That means we assume that well once the measurement  $x$  is available immediately I mean  $u = -k x$  that evaluation is instantaneous because that is why we are writing the equation as  $u_k$ . If you recall we are writing it like this right.

So here also the  $k$ th instant and they are also the  $k$ th instant and so we are assuming that this thing is happening instantaneously. But of course that is not the case as we can see that well in real life you have these delays due to interferences from other tasks. And actual execution time mapping time and all this stuff which will happen in a task in a processor.

**(Refer Slide Time: 23:41)**

## Task Triggering

- In general,  $T_m$  and  $T_a$  consume negligible computational time and are time-triggered.
- $T_c$  needs finite computation time and is event-triggered and preemptive.
- When multiple tasks are running on a processor,  $T_c$  can be preempted by a higher priority task.

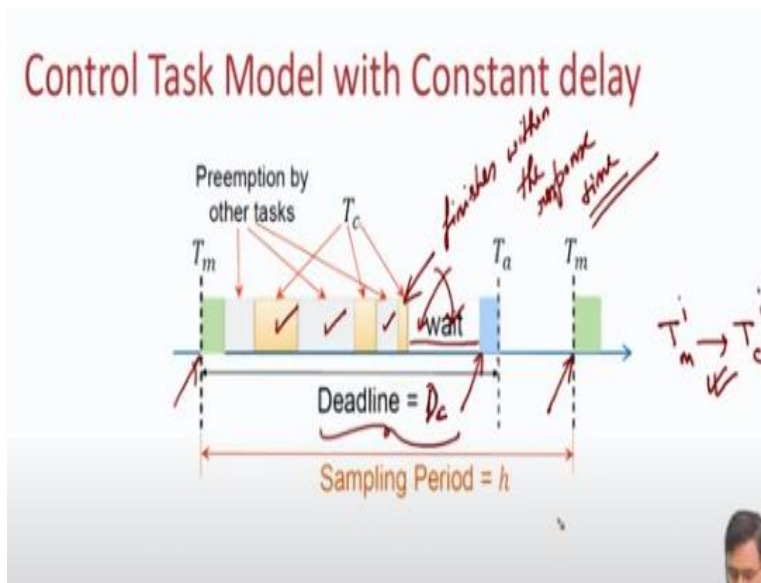


So in general this measurement or sensing tasks and the actuation tasks they consume negligible computation time and they are kind of time triggered. That means the sensor is always going to sense values and the corresponding processing tasks of the measurement will always happen in a fixed periodic manner. So they are like time trigger they are triggered by this period similar thing for actuation the sensor and the actuation tasks are all kind of time triggered.

So this is going to work here and then again exactly after this period  $h$  this will again be triggered something like that. So like this actuation task is here and then again after some this fixed period will repeat and like that. Now this  $T_c$  this control task needs finite computation and typically the way it will be implemented is you will have a preemptive scheduler which can actually preempt this task in between its execution provided there is something of higher priority to execute. So that is why this is preemptive in many practical cases and also we say this event-triggered because this may have dependencies on processing right previous stage processing.

So only when those things are happen that may end other high priority tasks do not exist are not executing then only this will execute. So this is kind of decided based on the relative events that are there around it right so it should happen sometime between  $T_m$  and  $T_a$  but that exact time is it is not deterministic it will be depending on the events that are happening around you the other tasks and stuff. So that is kind of an issue and we need to understand that well does it affect the control design.

(Refer Slide Time: 25:30)



So let us say your task the sensing task is here in blue, sorry in green and your actuation task is here and after this fixed sampling period the sensing task is again time triggered and in between you wanted to execute this control task. So you started it some I mean initially it was blocked by some other task then it has started here and then again some other task in and trigger some preempted it again it started here and it could not start initially because it can only start after the measurement of the or the sensor task executing. Let us say again it has to wait here because some resource is busy and let us eventually it executes and finishes here.

So and it finishes here and let us say immediately here this actuation task could not start here and so there was some wait because somebody else was occupying the CPU. Then eventually this control input was processed by this actuation task here and this is where the actuation output is available. This wait is also actually not due to preemption but the reason is please let me repeat this. So like I said that typically the way we are designing is this sensing on the measurement tasks and the actuation tasks are going to be time triggered, because they are computationally lightweight and no need to make them as a function of some scheduling decision.

So they are going to be always happening with this regular interval  $h$  not only for the sensor but also for the actuation. So your design should be such that this would always be kind of triggered from here actually up to this the 2 start points they will be separated by this  $h$ . So that is why if



you come to our next picture you have a specific point right where this will work out and then this will start and also exactly a point where  $T_a$  is going to execute. So and this point is decided based on the maximum response time you may be budgeting for the control tasks. So let us say the control task I mean by your design it will finish by that maximum response time right and any and after it got executed it before it got executed let us say this is the  $i$ th instance.

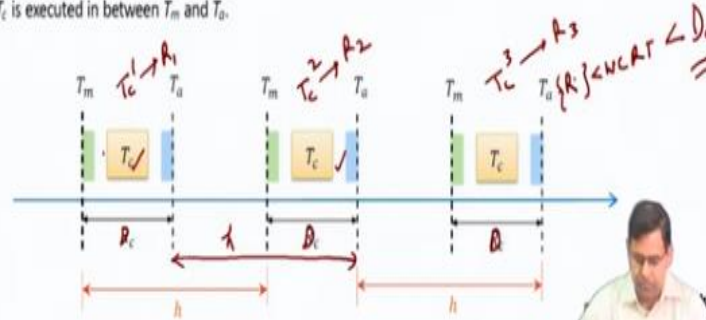
The sensor task or the corresponding measurement task must execute and it will satisfy this precedence relation and then maybe some other task of higher priority was executing. Then it executed then again it is preempted then again executed then again preempted and it got finished here. But this guy it can start because its event triggers it will start precisely here only and that is why we have this wait. In between other tasks we execute and they may be preempted again other custom execute stuff like that but the wait is specifically for this control loop. So the data that has been computed by this control loops control task is still not being processed by for actuation and that is going to happen here.

So the way we are deciding this triggering point is that we are setting a deadline for this control task and we are saying that well you must finish by this deadline and that is exactly the time when I am going to trigger the actuation task. So if you see it is like that everything is happening in a cyclic fashion, so  $T_m$  is time triggered with period  $h$ ,  $T_i$  is time triggered with period  $h$  but the relative offset between  $T_m$  and  $T_a$  this is given by as the deadline of the control task and it must finish somewhere in between. So like I said that everything is periodic at least  $T_m$  and  $T_a$ , and  $T_c$  but among them this guy is time triggered this guy is time triggered they may have this mutual offset and this offset time is the deadline inside which the controller must be able to execute.

**(Refer Slide Time: 30:15)**

## Control Task Model with Constant delay

- $T_m$  is triggered periodically with a period equal to the sampling period  $h$ . Schedule for  $T_m$  is assumed as  $\{0, 0, h\}$ , i.e., periodic with zero offset, negligible execution time and period  $h$ .
- $T_a$  is also triggered periodically with the same period  $h$ . Schedule for  $T_a$  is assumed as  $\{D_c, 0, h\}$ , i.e., periodic with constant offset  $D_c$ , negligible execution time and period  $h$ .
- $T_c$  is executed in between  $T_m$  and  $T_a$ .



So I hope this is clear. So we are just showing that picture here in the time continuum. So the way we are modeling is what we call as the constant delay model. So like I said the  $T_c$  can finish inside various times right so return it and in various instances of  $T_c$  can have various finishing time. So what we do is we can compute the worst case response time and based on that we can fix our deadline for that. So that it is greater than the worst case response time and that can be my choice of offset for this task  $T_a$ . So this offset will be equal to the deadline for the control task and at different instances of the control task it can finish at various points but all of those points should be inside the deadline right.

So to sum it up  $T_m$  is periodically triggered and period is equal to sampling period. Let us say the offset is 0 the  $T_n$  the sensing is precisely happening right at the starting of the sampling period. Execution time is assumed negligible and the period is  $h$  the timing model for  $T_i$  is this is also periodically triggered but like I said the offset is equal to the control tasks deadline because we want the control tasks response to happen inside this deadline. So this response time the what in the worst case should be less than this  $T_c$  and sometimes it may be finishing here sometimes it will be finishing here but always inside this  $T_c$ .

So if you can see that this is the period here and this is also precisely equal to  $h$  and this difference is the deadline  $T_c$  and  $R_c$  I mean this is basically  $D_c$  here and it can have several possible response times. Let us say for  $T_{c1}$  the response time was  $R_1$ , for  $T_{c2}$  the response time was  $R_2$ ,  $T_{c3}$  there is a

third instance response time was found to be  $R_3$ . So the worst case response time must be greater than all of these  $R_i$ 's and it should be less than this value of deadline of the control task that we are assuming in this design and that is equal to the offset of this task  $T_a$ . So with this we can have to proceed so what we are doing is we are assuming the constant delay because we are getting a guarantee that  $T_c$  must finish before the delay.

So that when we model the system we will assume that well the control task may finish earlier but it would not be that output on we used the moment it is finished. Because you see then the design would not deterministic. Sometimes the actuation will be earlier though sometimes the actuation will be later on and that may be very difficult to model. So in a simplistic manner we assume this constant delay and we will implement it in this way that means even if it finishes early the actuation happens exactly after the deadline. Now why we will do it is because then the modeling of the system becomes simplistic and then your parameters that you can derive for the system will be quite deterministic.

(Refer Slide Time: 33:32)

## System Stability and Control Performance

- Deadline  $D_c$  for a control task  $T_c$  are often *firm* rather than hard.

- This introduces the concept of weakly-hard deadlines which relaxes a few hard Real-time constraints.

- It's okay to miss a few deadlines, but not too many in a row.

- And it depends on what happens if the deadline is missed.

- Task is allowed to complete late.

- Task is aborted at the deadline.

$(m, k)$

Do we allow delayed control?  
OR  
Do we skip??



So let us say how this modeling can be done? So the way typically we consider the scheduling of the system is that well the deadline is mostly considered as firm. That means the control task must finish inside this. Now one important question is well what happens if the control task somehow overruns this deadline. Am I going to allow it to execute it complete it and then apply. So that means do I allow for what we call as delayed control execution or do we skip that execution? So

based on this I mean how we implement the control we have different kinds of models of embedded control tasks.

So in case we skip the execution then that means well this firm deadline thing is anyway not happening and you are not updating the control for certain control task instances. Now when this gives rise to a concept called weakly-hard deadline. Now this is a popular way of implementing real-time controllers nowadays. So what people do is people will show that well for the control loop as long as out of every  $k$  consecutive actuations you can actually achieve the execution the actuation in  $m$  of the instances you can theoretically prove that well the this this control loop is fine.

So in theoretical way I can show that well as long as the controller actually updates for,  $m$  out of  $k$  number of instances the value of  $m$  and  $k$  is specific to the model and controller we are talking about. So if this can be shown and it can be shown in practice for many systems then we have this idea of weakly-hard scheduling that means well my deadline is firm but it is to violate the deadline in a bounded manner. That means as long as out of every let us say 10 consecutive deadlines I meet 5 of them, or 8 of them I say it is fine and that guarantee may come from the controller itself. So that is called weakly-hard. But anyway this is for your future reference.

(Refer Slide Time: 36:02)

## System Stability and Control Performance

- Deadline  $D_c$  for a control task  $T_c$  are often *firm* rather than hard.
- This introduces the concept of *weakly-hard deadlines*, which relaxes a few hard Real-time constraints.
- Its okay to miss a few deadlines, but not too many in a row.
- And it depends on what happens if the deadline is missed.

- Task is allowed to complete late.
- Task is aborted at the deadline.

u<sub>k</sub> is computed beyond deadline

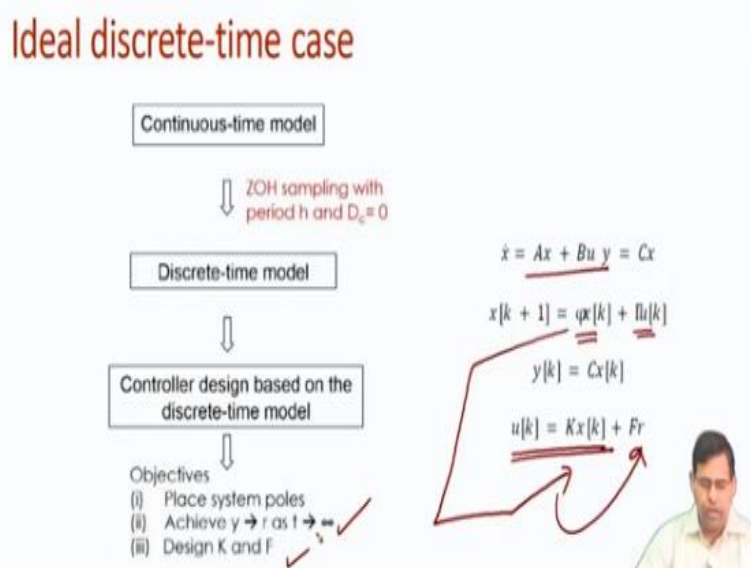
u<sub>k</sub> = 1



Let us come back to our simplistic case where we are talking about firm deadlines. So we will think that well if a deadline miss happens the controller is skipping I mean the controller I mean it is an issue for the controller. So we are assuming that the design is such that the deadline is not missed but what is happening is that controller always finishes its task inside the deadline and then it waits and once we have this this time  $D_c$  gone that the actuation task will be triggered exactly at that point and the control update will happen.

So just to repeat these are interesting task models allowing the control task to execute that means you are allowing more time to compute  $u_k$  or you are not actually computing  $u_k$  at all. So here what is happening is  $u(k)$  is not updated so it just remains as  $u(k) = u(k-1)$ . So these are the 2 situations and accordingly the control loops behavior will keep on changing.

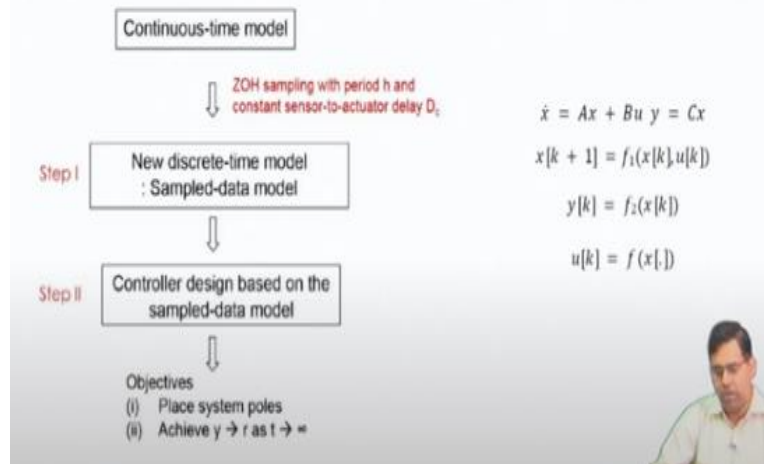
**(Refer Slide Time: 37:25)**



So the ideal discrete time case is this that you start with the continuous time model you have 0 order hold sampling your deadline is 0 that means immediately you get the control computed. So you have this situation which we do not have here and then you design the controller based on the discrete time model. So from this you get to design this and from phi gamma and stuff you apply and all this on this you apply the pole placement or other techniques and you derive  $K$  and  $F$ . And you have seen that where you are able to meet your objectives of catching up to the reference trajectory or not stuff like that. So that is your ideal discrete time case.

**(Refer Slide Time: 38:16)**

## Controller Design Steps for Case A: $D_c < h$



But in this case your situation is different. You have to think that well how should I design my controller that well I am able to tolerate this delay and nothing gets violated. So fine we will continue from here in the next lecture so I think our time is up here. Thank you for your attention.