**Foundation of Cyber Physical Systems**

**Prof. Soumyajit Dey**

**Department of Computer Science and Engineering**

**Indian Institute of Technology, Kharagpur**

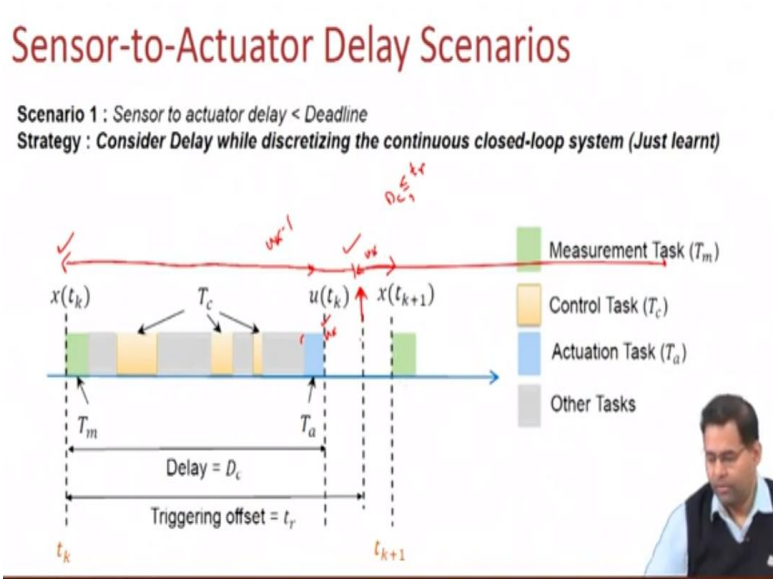
**Module No # 06**

**Lecture No # 26**

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

Hello and welcome back to this lecture series on foundations of cyber physical systems. So in the last lecture we talked about. Well due to the delay in the control update how the augmented system analyzes had to be created and the controller has to be designed corresponding to that augmented system right. So we will just go a bit deeper into what is the effect of such delay on stability and performance of control systems.

**(Refer Slide Time: 00:58)**




So if you remember the discussion we had was that well the sensor to actuator delay was not trivial but it was some significant value but although also it was always getting computed below I mean within the deadline. So let us say this is where your $t_k$ that means the sampling happens at the k th point and here the sampling will happen at the k+1th point. So with the kth measurements the control task computation is going to happen.

So this is the measurement happening here and then the control task let us say it executes here and then again somehow it is getting preempted here and then it is getting computed here but it finishes here. So and the point is we are given a fixed delay, this was the constant delay model where the delay is basically governed by some value which may be less than or equal to the triggering offset that means this is where the update of the control action will happen. And then with the updated control action that means $u(t_k)$, u will be, the system will evolve for this part.

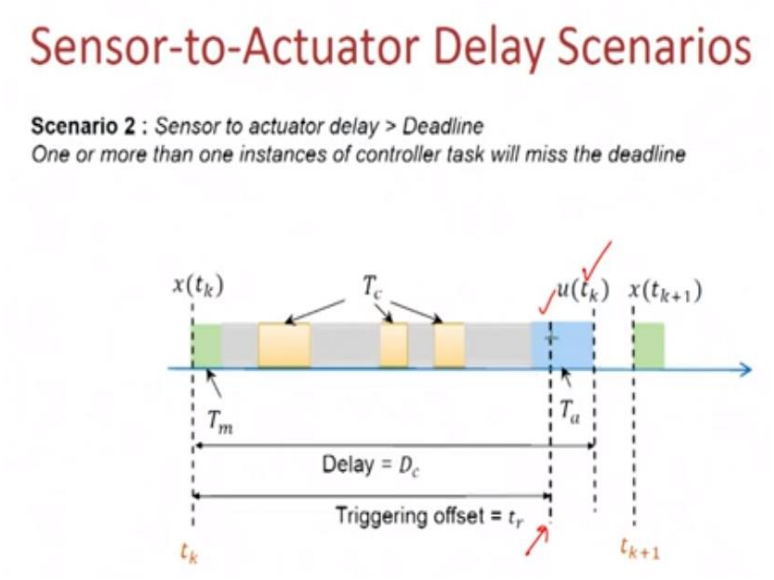
And before this the systems evolution up to this is happening with the previous control action. And just for your idea these are the standard task notations we have been using T m is the measurement task this is the control task and then we have the actuation task which will be executing here. So that the actuation update is really computed before the triggering offset. So we are assuming that the triggering is happening here, that means, here you have u k available just repeating a correction here. So this is where you have u(k-1) right and then we assume the constant delay model we model the delay to be constant and this should be less than or equal to the triggering time.

So in a nice case we can actually have this right here also. So we are assuming that well the task will be triggered eventually here that means the actuator will get start using the updated control input right here. And that is why we set the delay somewhere before that so that the actuation task executes before that and the value is available. And then when the actuator is going to actually use the value that is the point which is here right. So that up to this, so you have u(k-1) being used. Here u(k) has been computed but this using this less than equal to we can say that well but it is still not used and it will be used here.

When the triggering actually happens and the actuator actually gets that value. So I mean please do not mind the scale of this picture it may be quite possible that this entire dimension starting from here up to here this interval is small and really this interval is much bigger. So this is just for illustration here. So the point is the control task will execute then the actuation task will execute and this entire thing should happen within this delay. So the actuation task will finish right here that means once the actuation task finishes the u k update is already there. But the triggering that

means the actuator using the actual value may start with a mild offset here, so from here the u(k) value is going to be used for the rest of the period. all right

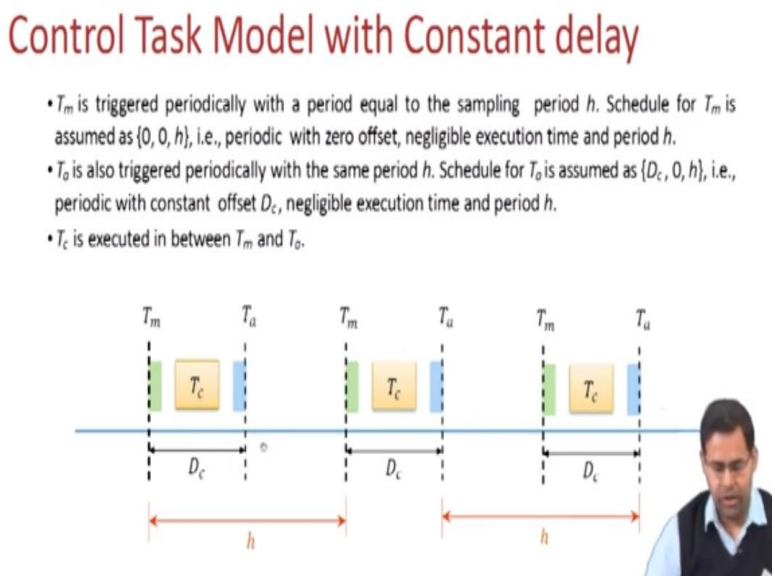
**(Refer Slide Time: 04:50)**




Now the question that comes is well what about the other scenario that is your sensor to actuation delay that we finally suffer that is greater than this deadline. So that means the actuator is supposed to sample the updated control right at this point right where the triggering will happen right. So the actuator is supposed to sample the updated control. But that is not really available. Because let us say due to delay in the control task or the actuation task the final update on the control happens here which is beyond the triggering of the actual of the actuator right.

So that means in this case the control the entire control command updation task is missing is deadline. So I mean it can happen in a various possible ways that it has been preempted too many times and that was delayed and then the activation and the actuation task also need to happen within the deadline so all these things could not be completed and before that the actuator actually try to sample the update if there is any on the control task with this offset. So just recalling from our previous picture.

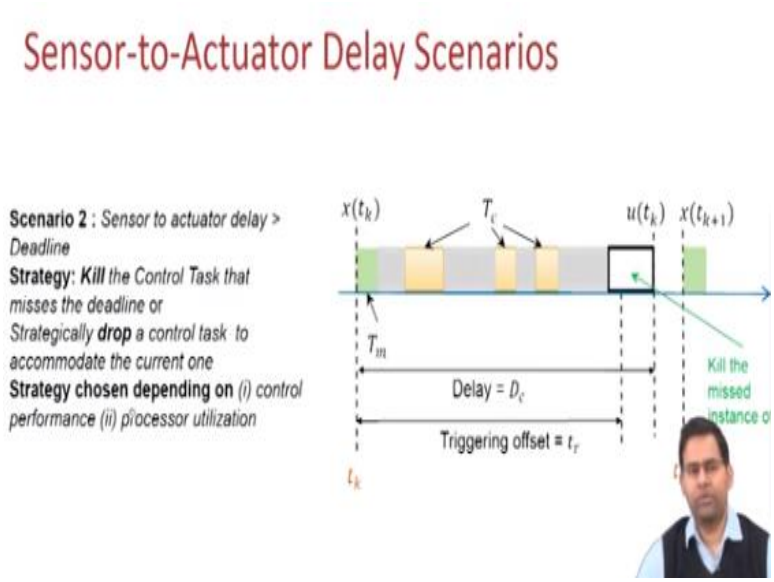
**(Refer Slide Time: 06:04)**




So this is what we had, like here is the measurement then there is a task actuation right happening here. There again measurement actuation happening here it is supposed to be like this right. And this delay is kept something below the triggering interval but somehow if this delay goes beyond that interval then what situation will happen that is what we are talking about here. So let us understand.

So this is where the actuator is trying to sample if there is any update on the control but the overall sense to actuation delay in this case due to tough preemption or extra computation or data communication delay etc. between the actuator and the controller or the controller and the bus although there were some delays due to which this computation got delayed and it was completed beyond this triggering offset. That simply means that well the actuator is unable to get this update right. So what happens in this scenario the actuator needs to continue with the older control update only right.

**(Refer Slide Time: 07:08)**

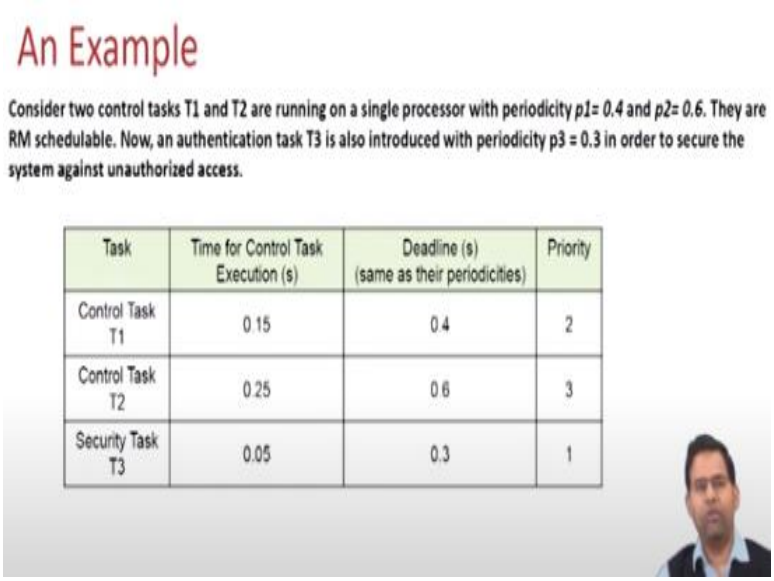



So you need to have some strategy like what to do here. So one strategy is that you kill the control task that is going to miss the deadline. So that means the control task execution is here right and it is still not finishing and the actuation actuator is expecting the update to be here. But the update is still not computed. So right here when the system observes that there is no update it can kill the control tasks execution right at that point and it can just continue with the older value.

There is also another strategy which is called delayed control that will you can actually delay the actuation itself. But that will depend on the physics on that means how the system is really implemented whether the actuator is actually I mean allowed to delay the updating the control from that task right. So in pure, in very simple systems what happens is that well the actuator simply does not wait because it will be working with a fixed period h and it will be working with this offset $t_r$.

So every time at a fixed period it is going to sample and figure out whether there is any control update or not. And in that case if there is no update it will simply continue with the older control update. The current control task execution will simply be dropped. So that is a kill job strategy. So the strategy I mean to be chosen it will depend on the performance of the system I mean well how good the control loop is performing with respect to its current sampling and plant model stability requirement etc., and also what is the amount of processor utilization that is there.

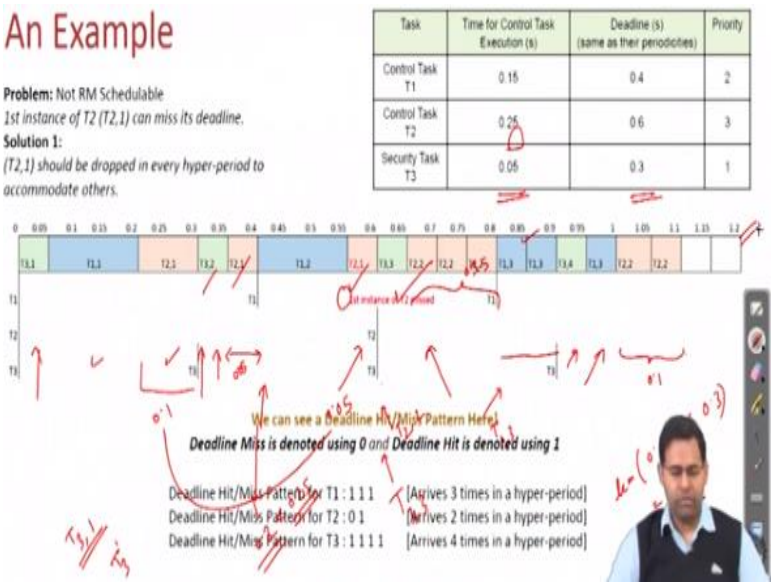
**(Refer Slide Time: 08:56)**




So as an example let us consider a situation. We will see that well how this kill job strategy can kind of affect the control task execution. So suppose you have 2 control tasks T 1 and T 2 running on a single processor with these periodicities as given. So T 1 has 0.4, T 2 has something 0.6. So T 1 will have a higher priority and let us say there is also an authentication task which will always be running with a much smaller periodicity and it will be running with 0.3.

And it is trying to secure the system I mean is basically monitoring the values etc., right. Since these are the very smaller priority it will run much frequently. Now let us consider that we are looking for an RM scheduling. So that would mean that this priorities will be in the order of frequency of the tasks. That means the smaller the periodicity the higher the priority though this will be the priority order. Security task will have the highest priority then you have control task 1 priority then you have control task 2 priority.

**(Refer Slide Time: 09:58)**




So observe that this also I just to understand why it would not be let us look at the schedule here. So this is the table we had here so we are just reproducing here and based on an RM strategy we are trying to build the schedule. So you see that 3 has the highest priority so 3's first instance will go in here. When I write T 3,1 it means task 3's first instance, it is something like a notation we used earlier which is this but here we are writing with the comma that is fine.. So after this finishes we will have T 1 because that is the second priority and then we will have T 2 which has a third priority. Then T 3 is has an instance which will arrive right here because it has a period of 0.3 deadline is same as period in this case.

So that is why it will preempt T 2,1 and then T 3 is second instance will execute here right and it will take 0.05 unit of time. So the time will progress from 0.3 to 0.35 and then T 2,1 spending instance which was preempted it will start and T 2,1 in total is we are going to take 0.6 execution time. And as you can see that well complete here some as I mean it will still execute here. But then well it would not be already complete right because it needs 0.6 and here you have a 0.1 done, here you have a 0.05 done, so not really completing right. So in this time T 1 second instance will arrive so T 1 second instance will now fully occupy the CPU for 0.25 seconds. So it will end somewhere at so T 1 second instance starts here and yeah it is going to take 0.15 second and so it starts at 0.4 and it will end at 0.55.

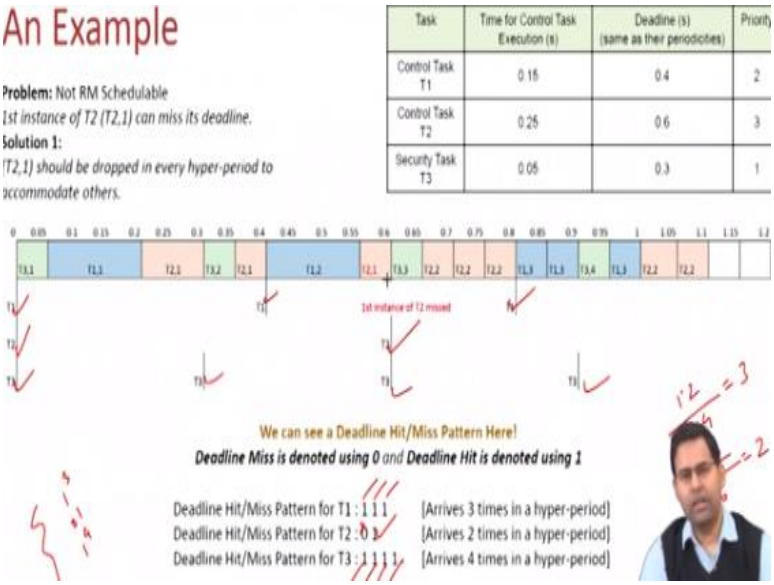
So then again T 2's second instance can start right and so now T 2's second instance will take this much amount of time and T 2's second instance will finish right. But if you see T for T 2 it will have its when is it supposed to come again. Here its period and deadline are both 0.6 right. So here T 2 second instance is coming and T 3 third instance is coming right. But could T 2's first instance finish. Let us calculate the execution time. Here it was 0.1. So in total we have 0.2 of the job done but in total it needed 0.6. So it does not finish. So that means before it finishes it will again be preempted and by now the deadline is also over right.

So then T 2's first instance is going to miss the deadline right here right. So fine then T 2's second instance has come and this is third instance has come. So let us say third instance will have a higher priority and we will execute it here and then we will continue with T 2's second instance. So first instance is already missed his deadline no point in continuing with which right so T 2 second instance continues here and then T 1's first instance is going to come. So if you see here T 2's first instance I could execute only 0.2 seconds but we in total required this. So that was not satisfied so it missed its deadline. So T 3's third instances arrive it will execute and then T 2 second instance will execute because T 1 second instance is here to arrive.

So T 2 second instance here it will execute for 0.15 seconds and at the end of it T 1's third instance arrived right. So T 1's third instances arrived, so T 2 will again be preempted it needs another 0.1 second right. So it will be preempted T 1's third instance will execute here and then again T 3 will come T c will occupy the CPU for some small amount of time. And then again T 1 third instance will occupy the CPU and it will end here right. So once it ends T 2 second instance will again restart and whatever 0.2 second was left it will continue it will complete right here.

So if you see in this period so we have actually drawn this thing up to the hyper period of these 3 tasks. So that is the hyper period of the task. So we have drawn it up to 1.2 and because we understand that if we can schedule these tasks up to the hyper period. Then the tasks will be infinitely schedulable for any number of periodic actuations the task will be schedulable right. So if we see that inside this hyper period whatever has been the instances of T 1 and T 3 the tasks with the higher priorities they could all be scheduled by T 2's first instance could not be scheduled.

**(Refer Slide Time: 16:35)**




Now what does this really mean? That means that well you understand that this pattern will continue like who will be schedule level who will be not schedulable this will continue in all the consecutive hyper periods right. So this means for the task T 1 I had how many instances so inside the hyper period. So I had 3 instances. So T 1 arrived here, here and here. All were schedulable so I can write a deadline hit miss pattern for T 1 like this a deadline heat that means the deadline being satisfied is denoted by a 1. And the deadline being missed that is the deadline overrun I mean the execution overrun the deadline is denoted by 0.So then the hit miss pattern for T 1 would be this right. Now similarly for T 2 how many instances are there?

1.2 is the hyper period 0.6 is the execution time so that is 2 right. So T 2 we had one instance here and another instance here. Now as we saw from our analysis T 2 missed its first deadline and then its met its second deadline. So for T 2 the hit miss pattern is 0 1 right and then for T 3 how many instances were there? For T 3 I had this as well as this and if we do the hyper period analysis total hyper is 1.2. The period of T 3 is this. So I will indeed have 4 instances and all were being executed right. So the pattern is 1 to the power 4. So I can write the patterns are 1 cube that is 3 ones the pattern is 0 1 and the pattern is 1 to the power 4 for T 1, T 2, and T 3. What is the importance of these patterns?

Let us understand that when this task will execute by RM on the processor under the strategy that whenever a task misses its deadline it is immediately killed right because that is what we did we

did not allow T 2,1 to continue beyond its deadline although it had some remaining execution. So under that we can actually derive this kind of patterns and we can say that well over the hyper period these are the ways in which the task will execute and they will hit and miss following this pattern forever. So for T 1 if I continue in RM, T 1 will always meet the sensor to actuation delay I mean or the period that has been assigned to it corresponding and not the generation delay but the period that has been assigned to it.

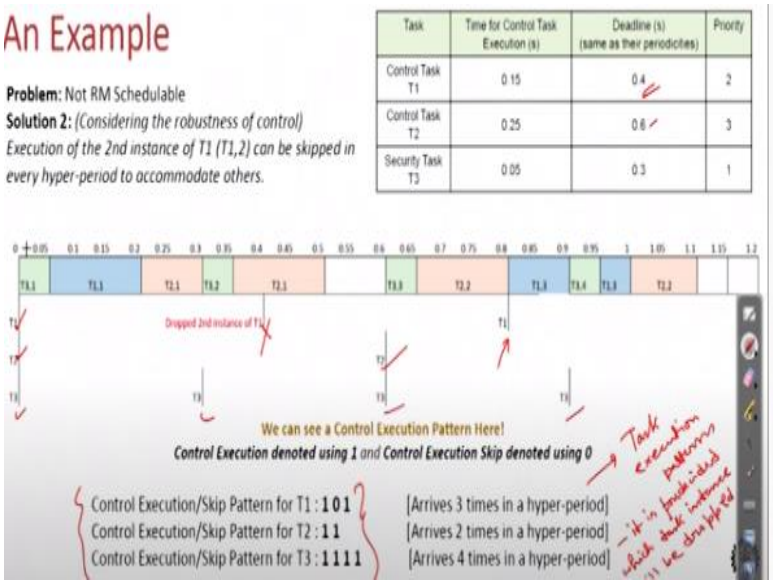
And also for T 2 I can say that well this task does not really work because it will always miss its first instance and it will actually satisfy its second instance. So that that is how we are showing it. So that that is how we are showing it. Now note one thing when we are doing this analysis we are actually ignoring certain things right because we are only considering that the control tasks execution can be preempted by other control tasks and security tasks. After that there may be an actuation task where may be delays we are ignoring those things. So we are simply saying that well if the control task executes within this time then it will satisfy whatever is the delay it is supposed to be designed with right.

And if the control task does not meet its deadline here on the CPU then fine. Then that means we after this deadline immediately there must have been a task triggering. So it has missed that task triggering also. So this is a simplistic method we can actually create problems with much more detailed models and that we will see while doing our tutorials and stuff. So here for the time being lets go on with this simplistic model. We are just assuming that if the control task is dropped that means it misses the deadline.

That means for the corresponding control action there has been a deadline over run and this control task instance has been killed and the control update did not happen. So u(k) works with u(k -1). So that is how simplistically we are continuing here. But I will just repeat what we really mean here I mean what would have actually happened in the architecture would have been this that well. This task missed his deadline and due to that we did not allow it to continue. The reason is that well even if it continued it will finish at a later time and then there would be an activation task which is going to take this input and then write the value in the actuator and then whatever time that will take before that the actuator's triggering will happen.

So that is what we are assuming here that this deadline is sacrosanct. If this deadline is missed then immediately as a consequent event the actuator triggering will not get this control tasks update. And hence we must kill the control task whenever it misses the deadline on the processor so that is how we are modeling it here.

**(Refer Slide Time: 21:44)**




So if we continue like this now the question is well what to do about it right? So let us try to see that what can be done. For example we see that since the tasks are not RM schedulable one possible solution would be that well we drop this T 2's execution. So we drop this T 2. T2's first instance execution or do something about it. So if we really see in our analysis like we saw that well this T 1,2 has missed deadline. So that means well there was no point in giving it the CPU at all right because eventually it got killed in this way right. And now see that what can be other situations? The other situations can be let us say like this.

Suppose I do not want this to happen that this task 2 should not miss the deadline. But then we have to do something about it right if I do not want task 2 to miss the deadlines then I should be the only thing maybe that I can do is I should drop some other instance of some other task actuation. Now the question is why should I do that? The idea is that well it can be done it depends on what are the control tasks and which control loops they are controlling. So typically when somebody is designing a controller and implementing it people usually over budget.

That means let us say you have a plant which you are sampling pretty fast, and due to that fast sampling you have got the corresponding control tasks which are also executing very frequently. So that means whatever would have been the minimum required sampling period so that the corresponding controller stabilizes that loop you may have chosen much higher period with respect to that and that is why your task is executing more frequently, that is like a robustness of the loop right. Now the thing is that since the low that by design the controller has some robustness it may not matter if that some instances of the controller are missed right.

So it depending on that suppose I have a set of control tasks which I am co-scheduling, I can go and check that which are the tasks which are more robust. That means if I can eliminate 1 or 2 instances of those tasks here and there it does not really matter for the corresponding controller. Now this is also a control theoretic problem so for which the real time platform designer has to take the feedback from the control engineer and understand that well you see for this task it is not that much the plant may be not that much sensitive to the controller actions because it is not a very fast control so for that control loop maybe if you drop some instances it may be right.

So depending on that such decisions can be taken here we are just showing it from a scheduling perspective. So we are just telling that suppose the previous solution that we had that well I just go by RM scheduling since the tasks are not actually RM schedulable. So what happens is over the period this task T 2 is missing its first deadline for the first instance. So that means T 2 will execute with let us say 50% of control actions really happening and 50% getting dropped. But let us say T 2 is controlling a plant for which this much amount of drops is not allowable right. The control engineer tells you to buy some analysis that is not allowable then what to do let us look for some other tasks.

Let us say for T 1 for T 1, I get the feedback set well for T 1 if you drop 33% or some kind of something like that many control actuations it is fine right. So then let us say that well can I intentionally drop certain instances of T 1 what does it mean by dropping the instance of T 1? That means I do not go by simple periodic scheduling instead of dispatching all instances of the task I dispatch as per this kind of patterns. So what I do is for T 1 I figure out that well may be giving

30% or 40% drops that mean not giving the control updates is fine. Because the plant can withstand that and still give me the required performance. So I will nicely try to choose that well these are the instances of T 1 that I can drop and those empty slots of the processor I can give to T 2 so that T 2 can execute with all its instances.

So let us suppose we decide something like this and now let us see that how the execution happens. So suppose I decide that well I will drop the second instance of T 1, T 1,2 in every hyper period. Now it is not that I am running T 1 and it is going to miss this deadline right. So that is not going to happen because it is an RM strategy what I will do is when I will dispatch all these tasks T 1, T 2 and T 3 I will not dispatch all its instances. I will judiciously choose and dispatch following this kind of pre-designed hit miss patterns that pre or let us call them a task execution patterns. So again I will repeat. I do not do periodic scheduling that means I do not really dispatch all the tasks all instances that means all the jobs corresponding to each task with the desired periodicity by or by but I really dispatch them following these patterns.

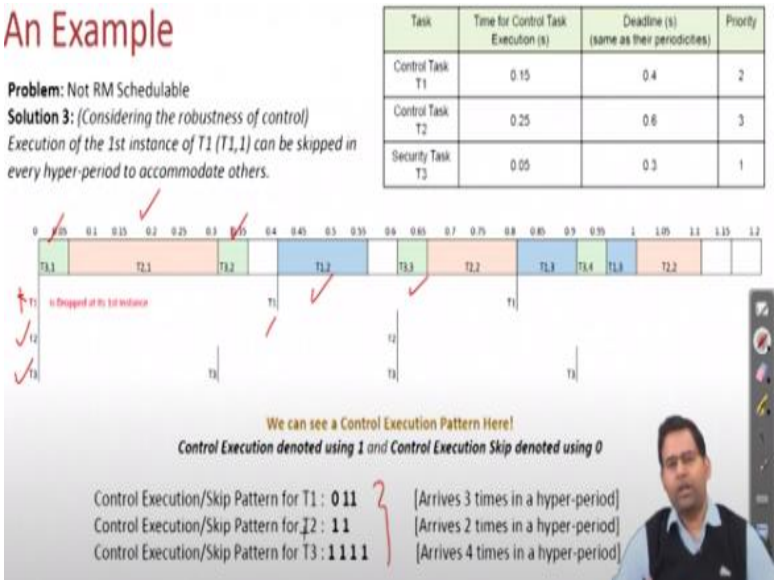
So that means I will dispatch T 2's first instance at 0 offset I will dispatch T 2 second instance at 0.6 of 0.6 seconds right inside this hyper period. And then again at 1.2 when the first hyper period ends and the next one starts I will again dispatch T 2 of T 2 instance etc. And for T 1, I will dispatch the first instance as you we are assuming here there are no offsets but there can be offsets fine as per the scheduling. Then I do not dispatch the second instance at this 0.4. But then again I dispatch the third instance at point a so that is how I do it here. So let us see that if I do that well I am giving T 1's first instance here I do not give T 1 second instance here I give T 1 third instance here. I dispatch T 2's first instance here and a dispatch T 2 second instance here and I dispatch T 3 is all 4 instances at this positions.

Now let us schedule them. So here T 3 will execute highest priority. Then T1's first instance executes, then T 2's first instance executes as per the priority because the RM does not know right that what we are dispatching or not. Whatever you are dispatching based on that RM is deciding based on the priorities right. Then T 2's second instance and then T 2's first instance will continue here and here you see there is a nice blank spot because now I have not dispatched T 1. So T 1 is not causing any contention here. T 1 is not causing any contention to T 2,1. So T 2 1 does not need

to drop here and T 2,1 can continue up to this. So here T 2,1 and executed for 0.1 second and here T 2,1 could execute for 0.15 second and T 2,1 can completes its execution of 0.25.

So it finishes here right again here T 2's second instance will come, T 2's third instance will come which will execute then second instance starts. Now T 1's instance will come it will execute this is fourth instance will preempt it T 1 third instance will resume and finish and then T 2 second instance will resume and finish. So all good only what we did is to make this task schedulable we take two judicious decisions that well I will intentionally drop T 1's second instance so that things work.

**(Refer Slide Time: 31:26)**



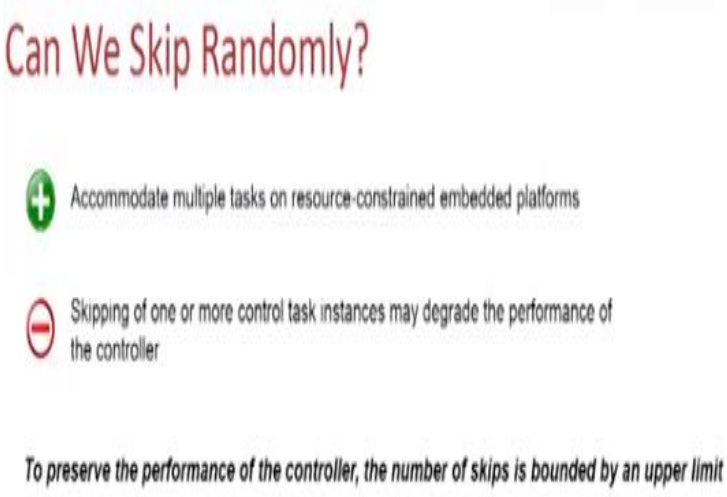

Now it need not be the only solution. There could have been another solution also I mean you can actually figure out which are the solutions by doing schedulability tests. So suppose instead of dropping the second you could have also dropped T 1's first instance so that the pattern is this right. So then what will happen is you drop T 1's first instance that will also have an effect that will then you do not have arrival of T 1 here you have arrivals of T 2 and T 3. T 3 will execute then immediately T 2 will get the CPU when it will execute for its desired time then T 3's second instance will come and then T 1 second instance which you have not dropped will come and it will occupy the CPU.

So you see that here again all these tasks are schedulable with T 1 just first instance drop so these are important design decisions. Now what is the advantage of this kind of a periodic scheduling? The advantage is that you do not need to reduce the control loop periodicity because if you reduce the control looks periodicity then your practically dropping more instances of actuation update and sampling you could have done. So you are doing a fine grain scheduling of the controllers and you are judiciously choosing which instances of the task to drop. And at the same time you are not kind of reducing the period that we do typically in periodic scheduling of control loops.

So this is one possible technique that people have used and you have found that well with respect to schedulability this is fine so you can choose what should be a hit miss pattern and because in case stuffs are not schedulable you can choose which of the tasks to actually drop. So you know that this will not finish so either you drop that instance or if you see that well this task instance for this control loop must execute. So in order to execute it you have to avoid interference from higher priority tasks and if the control loops corresponding to those higher priority tasks are robust and some of their executions can be skipped then you intentionally skip 1 or 2 of those executions.

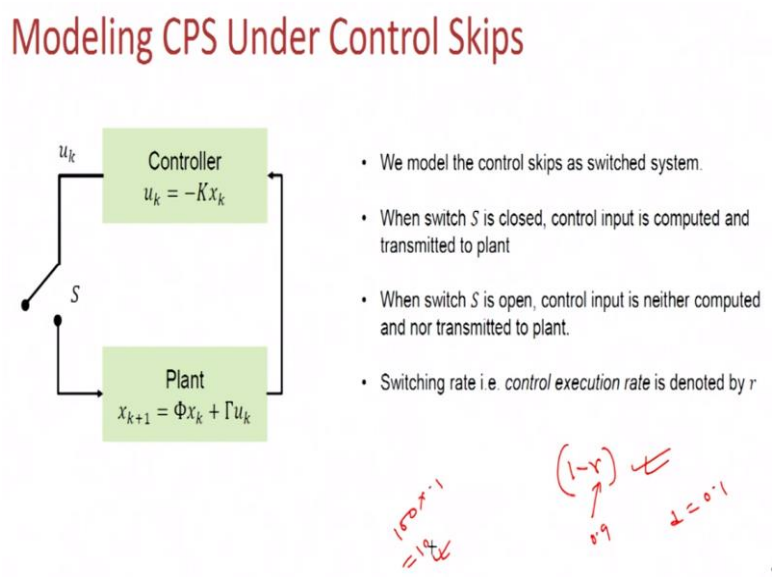
As we see in these examples and you decide that well instead of doing full periodic control, I will do a periodic pattern based control I will give the paper trans. And say that well inside your hyper period you need not execute all the tasks that arrive. You actually execute some of them following this kind of patterns so that is the learning from this discussion here.

**(Refer Slide Time: 34:05)**




Now here we have a very important question. Can we just choose and skip anything? Because then you see the control engineer will be very angry with you that well you cannot really do that I mean there is a mathematical guarantee to be maintained etc. So you cannot ad hocly decide that where I will skip these or these there has to be a mathematical reward of guarantee for that. So the number of skips is typically bounded by upper limit.

**(Refer Slide Time: 34:31)**

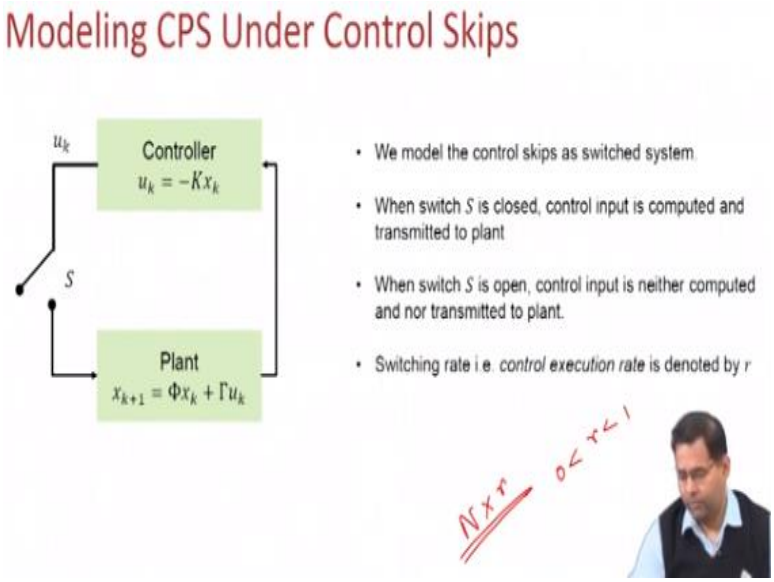



So let us say you model this as a switched control system. So whenever you I mean the switch is closed the control input is computed and transmitted to the plant, when the switch is open that means the control input is neither computed nor not transmitted to the plant that means the plant

will simply continue with its older control input that was there. Let us say this switching rate is r. So r is a rate that means 1- r is what we call as the drop rate.

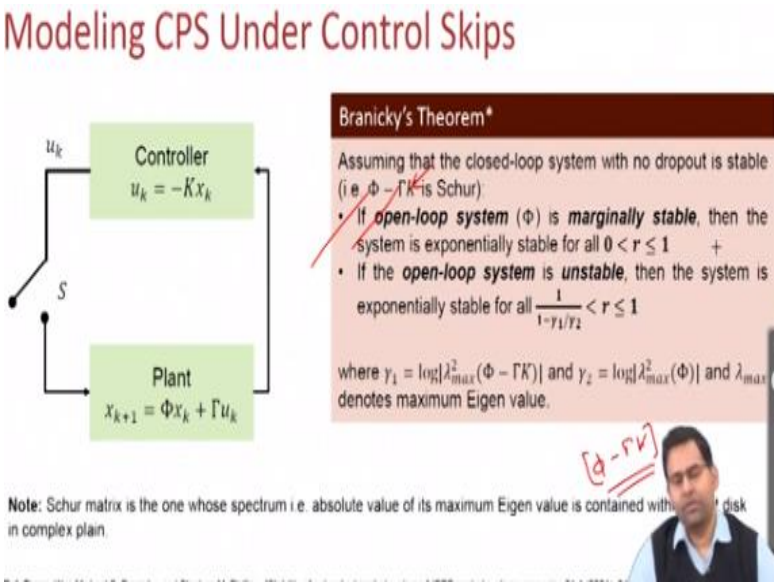
That means that means let us say there are 100 consecutive iterations, r is the execution rate. Let that be 0.9. So then drop date is 0.1. So that means you are saying that where out of every 100 consecutive control actions on the plant. Let us say 10 of them will be dropped. Now the question is there exists rigorous theory which says that well what is the upper bound of this drop rates or what is the lower bound of this execution rate that I mean minimum number of times you must execute so that, by the way note that these a rate. This is not a value.

**(Refer Slide Time: 35:51)**




That I mean if you have the rate as r, then that means that out of n consecutive cycles of execution this n times r number of cycles the control must really be updated. So r is basically a value between 0 and 1 here.

**(Refer Slide Time: 36:09)**




So there are several theorems in on this in control theory literature. Se will just show one of the simpler and popular ones. So this is based on Branicky's paper on around this time, the reference is here. So what it says is that, suppose you have the closed loop and the closed loop is stable without any kind of this kind of drops that are happening and for the closed loop you have the plant's model as Φ and Γ and the controller is K. So the closed loop is given by Φ - Γ K this is under digital discrete domain. In the discrete domain we can say that well this matrix is Schur. that means it is Schur stable. That means all the Eigen values of this closed loop of this system matrix in approach to be within the unit circle.

So when do I say what does the theorem state? The theorem said that if the open loop system that means the system without the control is marginally stable then the system is exponentially stable for all switching, all execution rates. What does it mean? Suppose even without the control the system is marginally stable that means what that means you need not really compute and execute you do not really care. Because well whatever is your control execution rate you are always able to meet the deadline of the control task or you are not able to meet that frequently does not matter the system will be exponentially stable. That's what the theorem says.

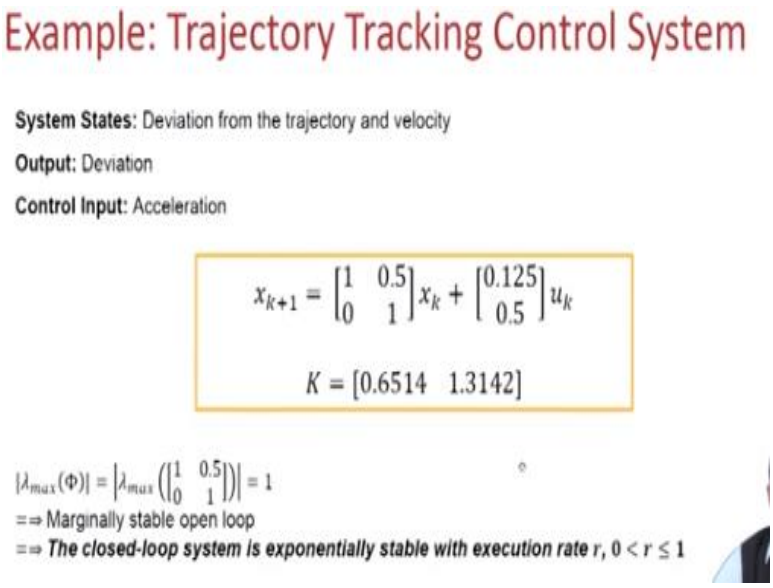
And then that, now comes the other tricky part. That suppose in open loop the system is unstable because that is the most common case right that is why you will design the controller or you will design the controller to have higher performance for a system which is may be open loop stable

but not performing well right. So in the second case when your open loop is unstable then and but the additional condition is well in open loop it is unstable but after closing the loop your Φ - Γ K is actually stable. That means the closed loop is stable then under what drop rate the system will still remain exponentially stable is given by this bound. That means r of course it is less than equal to 1.

But r must be greater than this factor, where this factor has 2 parameters $\gamma_1$ and $\gamma_2$ and $\gamma_1$ is parameterized with the Eigen value of the closed loop and similarly $\gamma_2$ is also parameterized with the maximum Eigen value of the closed loop. So what this theorem does is you I mean we are not going into the details and how this theorem can be proved those things are they are nice in these papers.

So what you can do is you can use the closed loop parameter to figure out using this theorem that what should be the minimum rate at which the switch must close. That means out of n number of executions how many of those executions must meet the deadline. That means I also know that how many of those executions can be actually skipped in case I have to accommodate other tasks while scheduling them so that that is the learning we wanted to impart from this theorem.

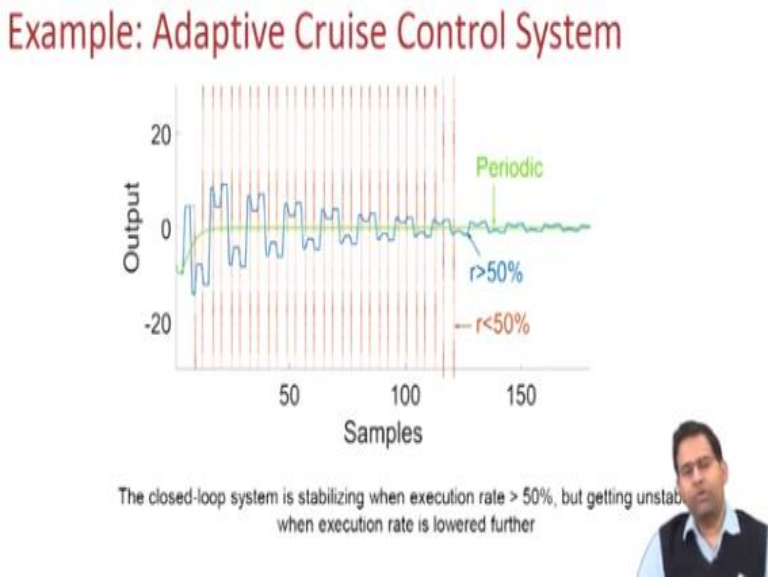
**(Refer Slide Time: 39:33)**




So just one example. So suppose you have our old favorite trajectory control system states the deviation of from trajectory and velocity the output is the deviation and the control input is the

acceleration. So for this your open loop $\lambda_{max}$ is 1. So that means the system is marginally stable in the open loop so the closed loop system is stable with any execution rate even if the loop does not close.

**(Refer Slide Time: 40:00)**




So you now see the plot you will see that well the output is I mean you have 3 plots here with 50% skip on the control input here. And you have 75% skip on the control input and you also have no skip that means is the proper suitable periodic situation user you see that in all cases you are kind of stabilizing right. Because of system by nature is open loop stable because the maximum Eigen value just considering Φ is 1 that means all the other eigen values are within the unit circle. So it is a marginally stable system. So whatever how many drops you give here does not matter the system's output is stable. But now let us take a system with a set of different parameters. So when you take the system with a set of different parameters it is an unstable open loop for this system matrix you see that the $\lambda_{max}$ is greater than 1.

So here you can see that well you can actually design the r based on this maximum Eigen values of the open loop and the closed loop and in this case this thing will evaluate to 0.5. So that means the minimum rate that you need to maintain is you need to actually complete half of those control executions. So you see this really comes out nicely so the green curve is for perfect periodic control, the blue curve is actually saying that well if your execution rate is at least 50% that means

at least half of the control executions you are really completing without missing the task deadlines. So you see that the system response is not as nice as the periodic case right but still so that means there will be jerks in the vehicles movement.

But eventually the vehicle will kind of hover around the target speed right there will be disturbances there will be jerks but it will hover around not smoothly but it will right. So it will not be unstable but see the moment the red goes below 50% that means you are not closing the loop fast enough you are missing too many control updates is completely unstable. So that is the knowledge we wanted to impart as a using a real example here. So I hope with this you have got enough idea about how performance and scheduling.

So this is a very important point we spend a lot of time here the reason is this is really what we should mean by cyber physical systems. You have knowledge from embedded and real time systems and scheduling and timing analysis. You have knowledge from control theory and you will like to apply them together to design the final implementation. So that is really how CPS systems need to be studied and designed. Thank you for your attention we will end this module here.