Intelligent Systems and Control

Prof. Laxmidhar Behera

Department of Electrical Engineering

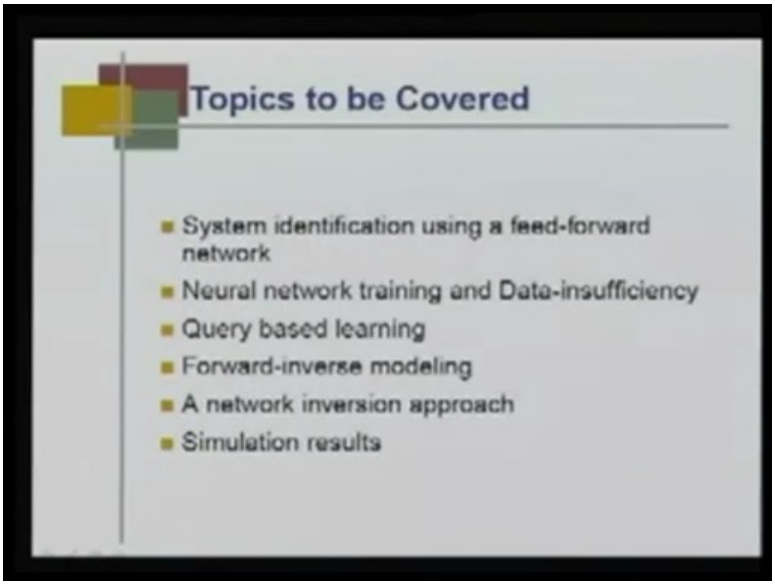
Indian Institute of Technology, Kanpur

Module – 3 Lecture – 4

Indirect Adaptive Control of a Robot Manipulator

This is a lecture on indirect adaptive control of a robot manipulator. This is the fourth lecture in this module on neural control. In the previous two classes, we covered the background that is necessary to understand this lecture. First we talked about network inversion. Given a network that has learnt a dynamic model, can I use that network to compute what is input given for a target output? Next we talked about the problem of having a model of a robot manipulator. The reason is, robot manipulator is an open-loop unstable system; data generation can be only done using a PD controller.

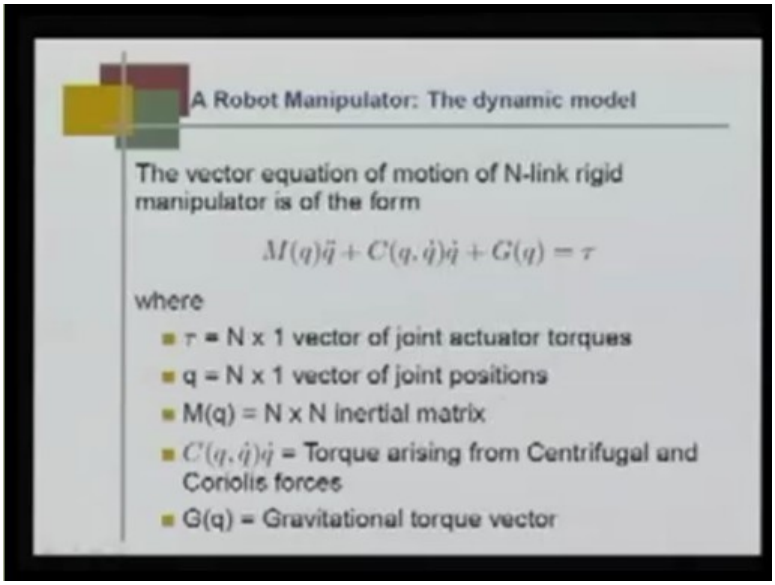
(Refer Slide Time: 01:35)



Given those two backgrounds we will have the topics that will be covering: System identification using a feed-forward network, we have already discussed this; we will be overviewing neural network training and data-insufficiency, we have discussed earlier and query based learning, these are all from previous classes. Today, we will be talking

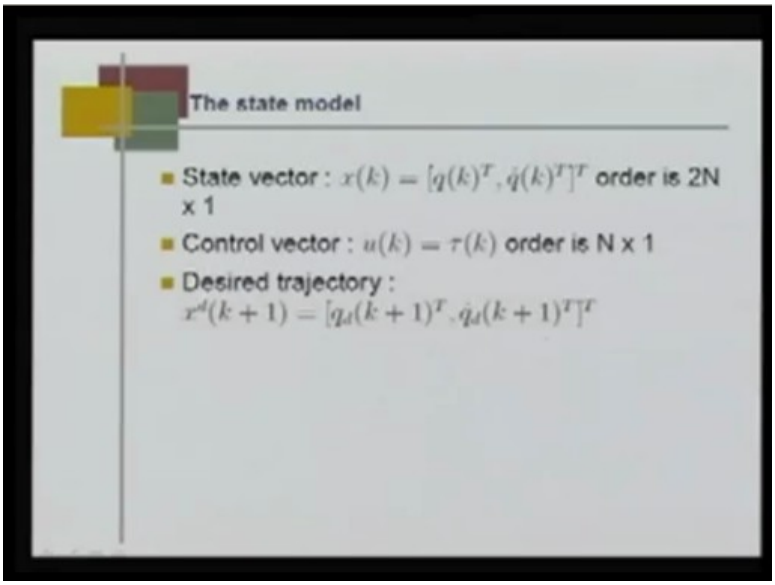
about two different indirect adaptive control schemes. One is based on forward-inverse modeling and second is network inversion approach and finally, we will show these methods how they can be validated through simulation.

(Refer Slide Time: 02: 25)



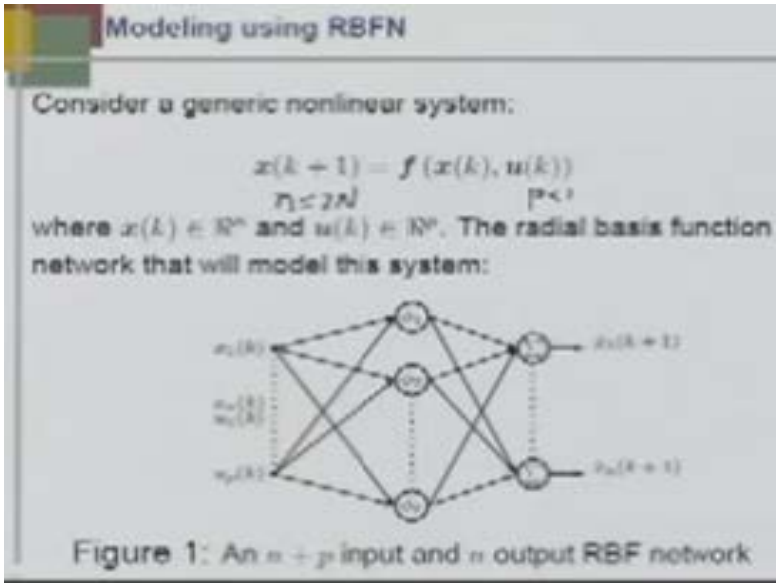
Robot manipulator, we have been talking about this model since last two classes. This is vector equation of motion of N-link rigid manipulator; this is the state space model.

(Refer Slide Time: 02: 41)



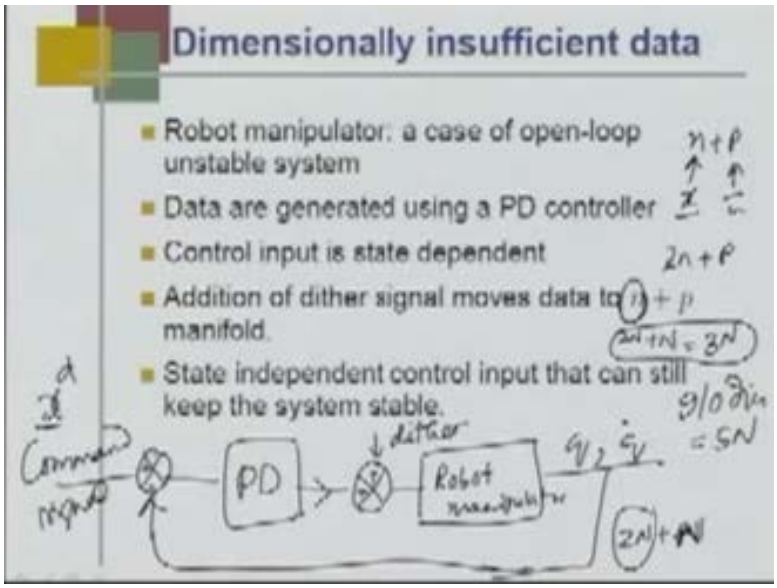
We have 2N states, with N angular position and angular velocity vector then, the control vector is that joint torque actuated which is N dimensional and it is desired that robot manipulators should follow this desired trajectory.

(Refer Slide Time: 03: 01)



We have already talked about given generic system of this particular form. Where this is our output state, you can easily see for robot manipulator n is 2N and u is…the dimension is P; this is n state system, small n, for robot manipulator it will be 2 into capital N. We collect the data online from a robot manipulator by taking the links; the manipulator link around desired trajectory. We get various data from the robot manipulator. We use those data to model the robot manipulator in this particular form which is x (k plus 1) is f(x(k), u(k)). You can easily see that, this radial basis function network that has n plus p, input x_1 to x_n, u_1 to u_p and the outputs are x_1 to x_n at sampling instant k plus 1, input are at sampling instant at k. These are the values taken from actual plant, these are the control action actuated and you get the output of the radial basis function network. If it is trained properly, then, it can mimic the robot dynamic behavior. Last class also we talked about dimensionally insufficient data which means, the robot manipulator is an open-loop unstable system.

(Refer Slide Time: 05: 11)



We said that if this is my plant, this is my robot manipulator then, I use a PD controller here and I put some dither signal and here is your command signal and this is your sensors. The output of robot manipulator if we actuate with joints, we observe what are the link angular positions vector and angular velocity vector. They are fed back at the command signal and command signal is normally x - desired, the desired trajectory that link should follow, you have this PD controller. PD output we add this dither signal just to make sure that, the data that are being collected that only spread over n dimension rather n plus, actually 2n plus p, 2n plus p is also here n. Actually, given the state space model, we talked about n plus p, n is for x, p is for u. Input-output dimension is 2n plus p because, your number of input is n plus p and that is when you have a network. If I have a network that has modeled this robot manipulator then, you have n plus P input where, n is capital 2N, n is capital 2N and P is also N. So that is 3N number of input, number of output is 2N, 2 capital N. So, the total input- output dimension is 5N; the objective is to collect data in 5N dimension. To be able to do that is very difficult job because, we do not have a method by which we can independently give input signal to the plant or to the robot manipulator.

The input actuation cannot be made state independent because we are using a PD controller. If I do not give dither signal, I am simply collecting data in n dimension. This

n is actually 2N and if I add dither signal the maximum dimensionality into which the data will span is 2N plus N is 3N. This is the problem of dimensionally insufficient data. What is a neural network? Neural network means, we are simply fitting the input data from input to output; it is simply a data mapping. How well data are generated? This point we discussed in the last class and that is why we proposed a query based learning algorithm.

(Refer Slide Time: 09:38)

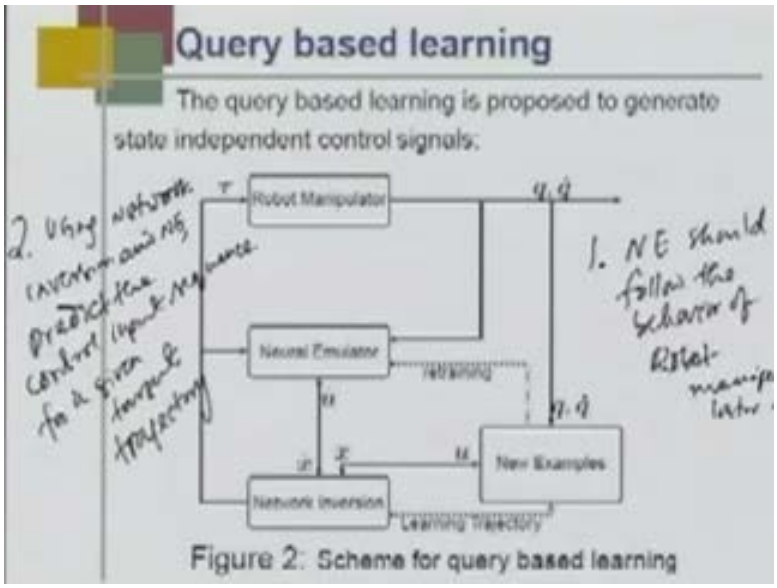


Figure 2: Scheme for query based learning

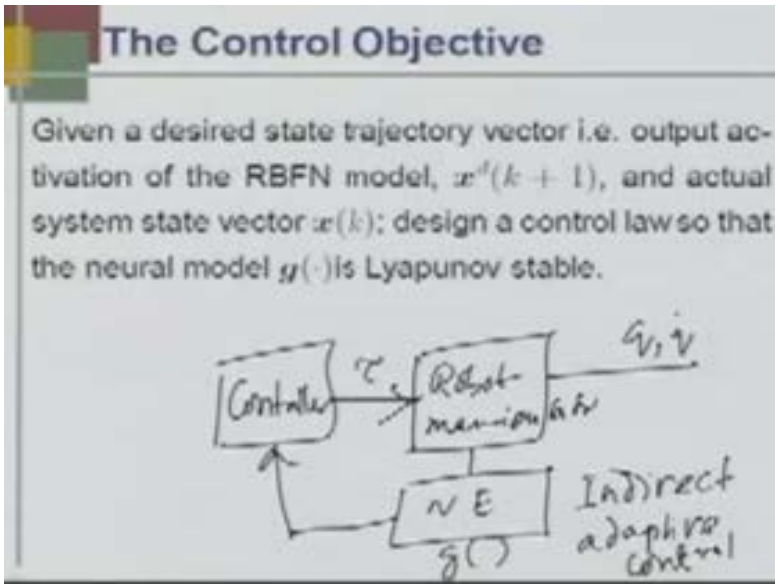
In query based learning algorithm what we proposed that after the way we generated a data here, we generated data like this and then we provided those data to the radial basis function network to emulate an actual robot manipulator. So that first iteration robot emulator or neural emulator is placed in parallel to the robot manipulator.

What this query based learning algorithm has to do is that; this neural emulator will give new learning trajectories. Along each learning trajectory, through neural emulator, the query through network inversion will be asked. What would be the input for the desired target point defined by learning trajectory? Through network inversion we compute what should be the tau, we actuate that to the actual physical plant, and if we see the response of the physical plant in the next sampling instant is far away from the actual value of that is set as according to the learning trajectory. Then we accept that as a new example

because, in that zone the neural emulator does not have proper information. The basic motivation here is that, if I have a neural emulator that has modeled robot manipulator. Kindly, hear very attentively this particular statement, what I am trying to say is that, if this neural emulator or this model of a robot manipulator, if it has actually trained properly to capture the dynamic behavior of the robot manipulator, it is not sufficient, to simply ask this question given a desired sequence of desired input. If the neural emulator is able to follow the behavior of robot manipulator then, this is a proper emulator; now we are going a step beyond. Here, what we are saying the first step is that, neural emulator should follow the behavior of robot manipulator. What it meant is, given a sequence of control actuation to both the robot manipulators as well as neural emulator, the response of neural emulator and robot manipulator should match, this is standard practice. But, we are introducing a second constraint to say that, the neural emulator actually is emulating robot manipulator properly.

Second point is that, we ask this question to neural emulator. Given a target point, what should be the control input? We give that control input to the actual robot manipulator and we see if their manipulator goes to the target point. If that is the case then we would say that, the neural emulator is properly mimicking the robot manipulators. In this case, using network inversion and neural emulator predict the control input for a given target trajectory. Control input sequence I would say. This is second point that we are imposing to say that a robot manipulator model in using neural network, how that can be robust? Not only should it be able to predict the target point, it should also predict the target input because, to this robot manipulator given tau it produces q q dot. So, one way we can say that, if this is a neural emulator give the same q q dot to the neural emulator and tell him it should predict what tau is. The other way is that gives the tau to the neural emulator and predicts q q dot and we have to do both; if neural emulators satisfy both conditions then, we say this is an exact replica of the robot manipulator. Of course, we cannot say exact replica but, in some sense it has much more robust behavior than taking only this condition alone. We have already discussed these things.

(Refer Slide Time: 15:37)



The control objective is given a desired state trajectory vector that is output activation of the radial basis function network model which is x d (k plus 1) and actual system state vector x (k): design a control law so that neural model g is Lyapunov stable. What we are trying to say is that, this is our robot manipulator. This is our neural emulator; neural emulator that mimics the robot manipulator behavior. Now, I utilize this neural manipulator and design a controller here. This controller will actuate a control signal tau to the robot manipulator and robot follows certain trajectory. In the state space up angular position and angular velocity, this is called in literature which we say indirect adaptive control, this is indirect adaptive control. What you are seeing here is that, what is meaning of Lyapunov stable is that, if I represent this NE by a function g because, the normal stability study is that, I can put instead of robot manipulator this NE. There neural emulator and neural emulator in the feedback loop, they should be Lyapunov stable that is the control objective.

(Refer Slide Time: 17:54)

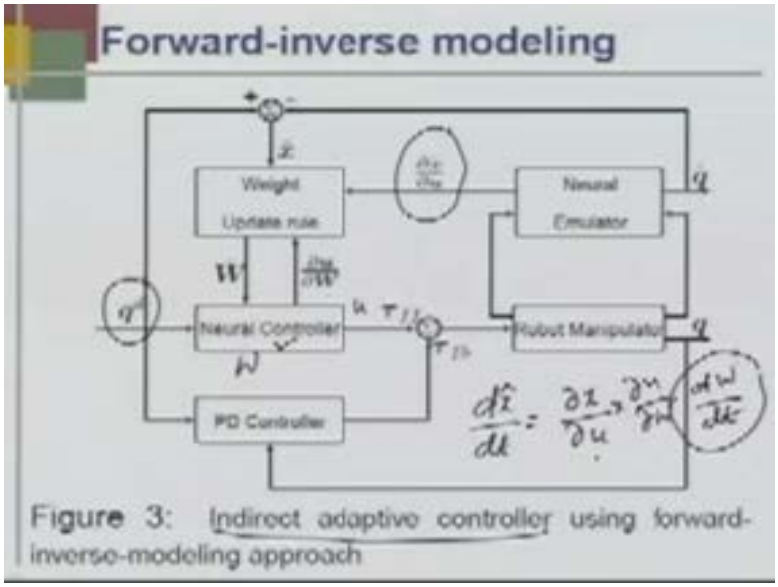


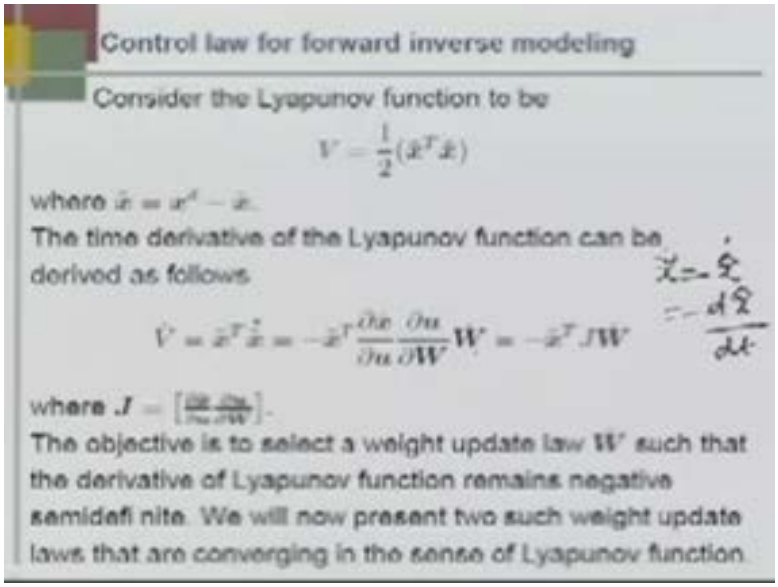
Figure 3: Indirect adaptive controller using forward-inverse-modeling approach

Here, the first type of indirect adaptive control scheme that we will be discussing today. This is called forward-inverse modeling. In this forward-inverse modeling what you are seeing here is that, this is the robot manipulator, this is again indirect adaptive control you see. This is also indirect adaptive control but, the principle is forward-inverse modeling. I will just explain forward-inverse modeling.

This is our robot manipulator which has a neural emulator that we have already discussed a lot. This neural emulator has the capacity to exactly mimic the behavior of robot manipulator not only in the terms what is of desired target but given the desired target and desired input. In both ways this is an ideal neural emulator. This is our desired trajectory q d, here is my neural controller. I have two-stage controller here; the neural controller actuates the feed-forward task and the PD controller actuates a feedback torque and together they are actuated to robot manipulator. The question now here is, given the q d whatever the output of neural controller and PD controller output the same output is given to the neural emulator which gives an output q hat and we take it, compare to the desired trajectory and I get an error. I use a weight update algorithm to update the weights of neural controller. The meaning of forward-inverse modeling is neural emulator represents the forward dynamics of the robot manipulator. Now what we can do, we extract from this neural emulator the information known as: Del x upon Del u,

which I will show you just now. This is very important information that we get from neural emulator. We use that information which is necessary in our weight update rule to compute which is w dot. We compute and update the weights of the neural controller based on this w dot, and what is this neural network controller, this is again another radial basis function network. We have two radial basis function network, one radial basis function network represents neural emulator; another radial basis function network is the neural controller. What we do not know in this radial basis function network. We fix the centers in their place in specific domain because, we know that to this controller what will be the various possible input that is q d the desired trajectories, actually to this neural controller, we have 3 n inputs, n inputs from angular position, n input from angular velocity, and capital N inputs from angular acceleration and then neural controller predicts, what is the forward torque necessary for making the robot manipulator to exactly follow the trajectory. Now, we will show you how we design a weight update rule for this forward-inverse modeling such that, the controllers the total feedback control system here is Lyapunov stable. See how we are doing it here, consider the Lyapunov function to be half x tilde, transpose x tilde where x tilde is x d minus x hat.

(Refer Slide Time: 22:48)



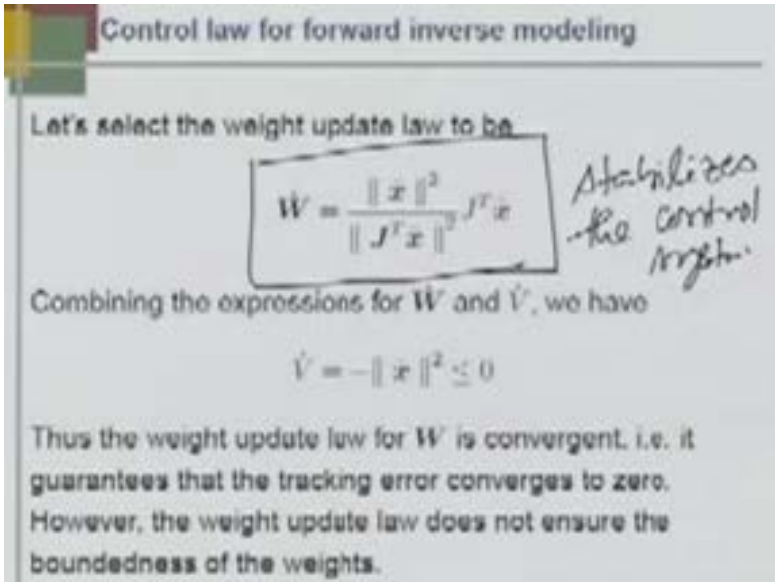
This x hat is response of the radial basis function network, which is here q hat. The time derivative of the Lyapunov function can be derived as follows: V dot rate derivative is x

tilde transpose into x tilde dot and then we can write that x tilde transpose. Then, we can write that: Del x upon Del u because, you see that this x tilde dot is x minus x hat dot. That is, minus d x hat by d t so d x hat by d t can be written as: doe x upon doe u into doe u upon doe W into doe W by dw by dt. That is, we are expanding that x as a function of u the control input, u is a function of W and W is a function of time. Say if you go back here this is the meaning of forward-inverse modeling that is given x hat here, this is a function of u here. That is why, doe x upon doe u is computed from the neural emulator, again the neural controller output is u. We can say the output is here is u, so doe x upon doe u into doe u upon doe W that again can be computed from this neural controller into W dot is, what is dx upon dt. I have to explain to you again what we are trying to compute is dx upon dt.

To compute dx upon dt, which is x hat that is the rate change of the output of the neural emulator can be written as, doe x upon doe u into… because this neural emulator has a functional relation between q and u, again doe x upon doe u here it is, doe u upon doe W. The neural controller is characterized by a weight vector w and this u output neural controller is a function of this W, this is not…. because PD controller gains are fixed, these really do not affect, this q hat what affects is the weights because, these are changing neural controller. So, doe x upon doe u again doe u upon doe W, u is the output of the neural controller, neural controller is characterized by the parameter W and the input is qd which are all known. Naturally, doe u upon doe W it can be easily computed from the neural controller into dw by dt. This is the weight update law, now we have to find out what should be this dw by dt such that, the whole system is Lyapunov stable. Meaning of that is that Lyapunov stable means, if I consider this to be a Lyapunov function, the rate derivative of this function V dot has to be negative definite, if I can prove that this is negative definite, then this is a stable controller. The objective is here I have defined j to be this quantity, this is Jacobin, the objective is to select a weight of that W dot such that, the derivative of the Lyapunov function remains negative semi-definite. We will now represent two such weight update laws that are converging in the sense of Lyapunov functions. We are selecting first what is W dot and this is my W dot the weight update law. If I select this weight update law and put this weight update law in this

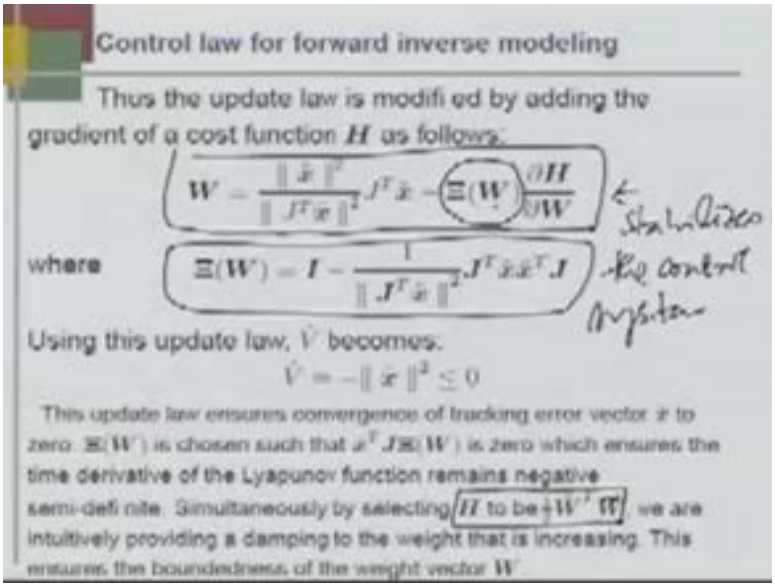
particular replace here, then what I get V dot is minus x tilde norm square, meaning this quantity this particular quantity is either.

(Refer Slide Time: 28:15)



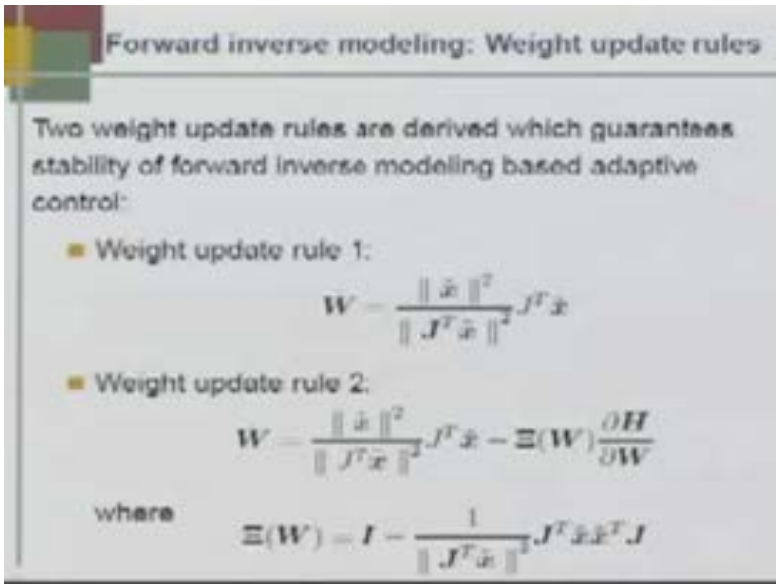
If x tilde is not 0, then it is positive; hence it is negative and if x tilde is 0, all the terms in x tilde because this is a vector, if all the terms are 0 then only at that time it is 0, which is desired, x tilde finally should be 0. That is desired then, the system is stable. Thus, the update law the previous law what we saw? What we saw is this stabilizes the control system; the complete control system including the planned dynamics is stable, according to this weight update law. Now, the weight update law however does not ensure the boundedness of the weight because, you see that, we are updating this weight and there is no way we are talking about boundedness of the weight. So that we will be doing now, thus the update law is modified by adding the gradient of the cost function H as follows. W dot earlier term was this one we have added another term.

(Refer Slide Time: 29:44)



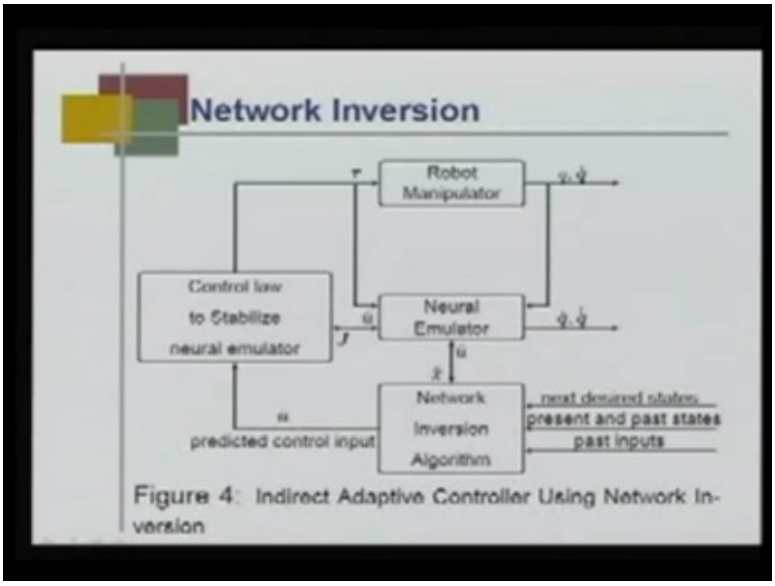
Where you have taken a gradient of a function H and this H is normally we select here, you can check here, this is half W transpose tilde into W tilde. Taking this H to be this particular function the, Del H upon Del W if you compute that, and where this particular function that you are looking at is a function of W is defined by this quantity. If you take that, add this term to this W dot, then again you take this new weight update law into the Lyapunov function, then Lyapunov function again becomes x tilde norm square negative, negative of x norm x tilde norm square implying again the system is stable. Again this control law stabilizes the control system in the sense of Lyapunov. The system is Lyapunov stable, finally this update law ensures convergence of tracking vector x tilde 0. A new function is chosen such that, x transpose J this function is 0, which ensures the time derivative of the Lyapunov function remains negative semi definite simultaneously, by selecting H to be half W tilde transpose W tilde, we are intuitively providing a damping to the weight that is increasing; this ensures the boundedness of the weight vector W.

(Refer Slide Time: 32:02)



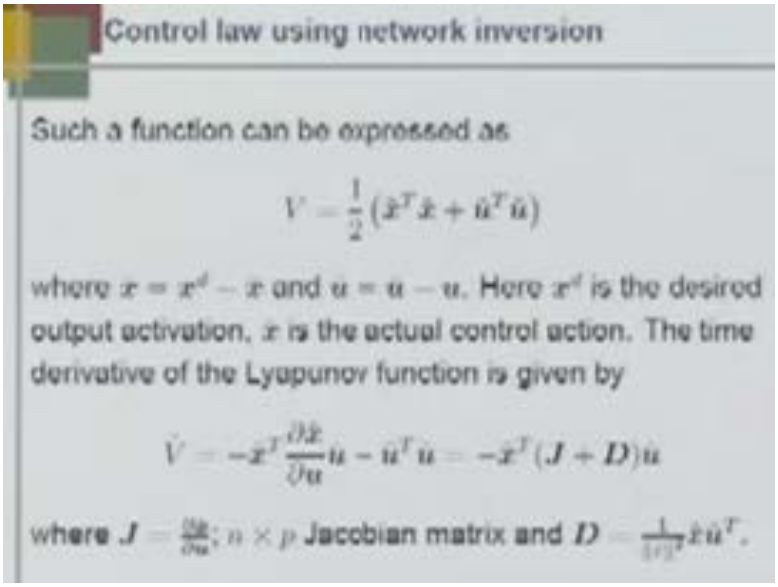
Finally, summarize two weight update rules are derived which guarantees stability for forward inverse modeling based indirect adaptive control. Weight update rule 1 was, W dot is, how if we go back what was the problem? Originally the problem was that, we place a neural controller whose weights are unknown, how do we update these weights such that, the overall control system is stable and the perfect tracking is achieved at the plant level. That is what we are doing first; we found out the weight update law for the weights of the neural controller. Similarly, weight update law, second neural update law we found out. This does not take into account of the boundedness of the weights, this does take into account.

(Refer Slide Time: 33:06)



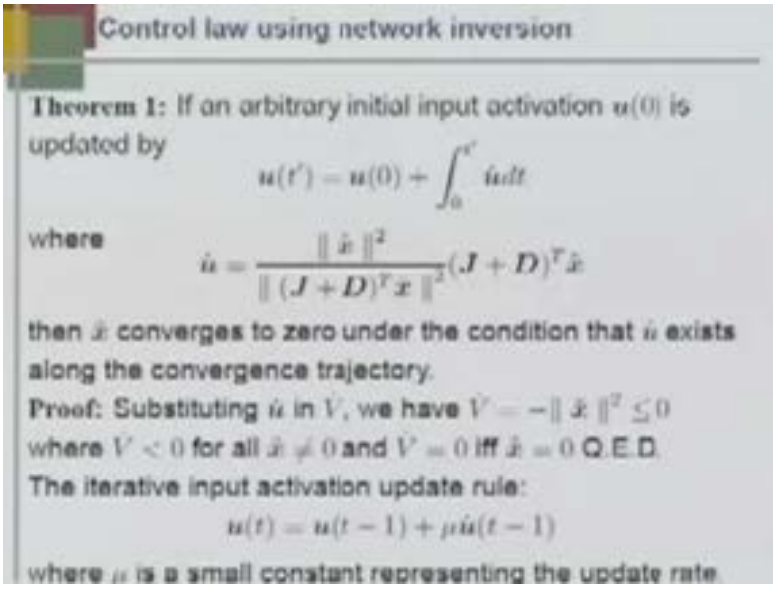
Now, we will go to the second approach that is using network inversion, we have already talked about network inversion. Now, the concept here is very simple I have a robot manipulator, I have a neural emulator. Now, can I use this neural emulator as a whole to construct my control law here that would stabilize the entire control system here? That is the question. That is what I am trying to say is that, if I have a controller sitting here in conjunction with the neural emulator can I say, given a desired trajectory here, next desired state present and past states and past input or you can always also say here that q d. Given a desired trajectory, can I say that I have a control law in conjunction with the neural emulator in such a way that, finally my neural emulator output are always following the desired trajectory and system is Lyapunov stable?

(Refer Slide Time: 34: 25)



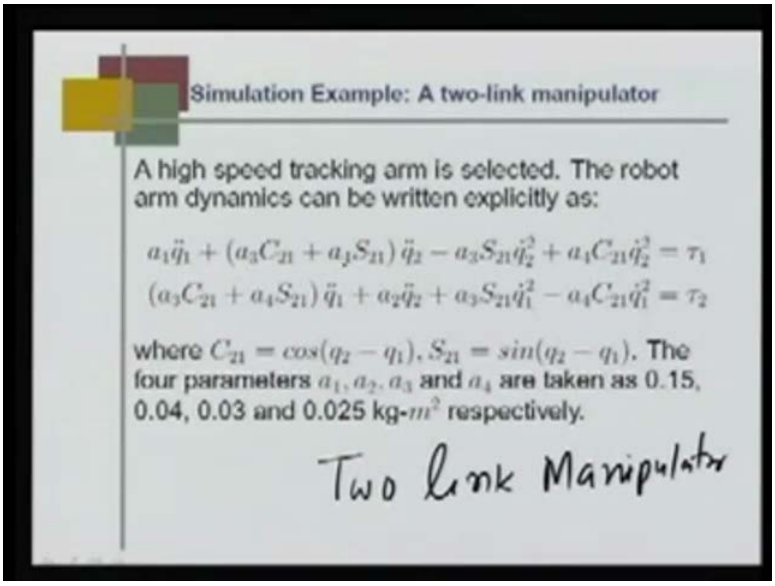
This is the network inversion, control law using network inversion. What we are trying to do is we are constructing a Lyapunov function which is half x tilde, transpose x tilde plus u tilde transpose u tilde. We have introduced a new element or new term in the Lyapunov function, where u tilde is u hat minus u. What is this u hat? If you go back you see that, when I invert the network I get what should be the control input that will take my neural emulator to the target vector. You know already that, there may be a situation where my predicted control input may not take the robot manipulator to its actual target. Because of the problem that we enumerated in the beginning that, actually neural emulator is that one which not only predicts the target, also predicts the desired input. If that training is not complete then obviously, the neural emulator will fail sometime to predict the control input. Now, we have to find some method by which this prediction can be made properly. With this introduction of this new term, we take the time derivative of the Lyapunov function and then we see that if I differentiate this term, x tilde transpose x tilde dot and x tilde dot can be written as: minus doe x upon doe u. That is why, the minus sign comes into doe u by d t and so du by d t that is how is the first term. Second term is, u tilde transpose into u dot. Where, we can write this term as minus x tilde transpose J plus D u dot where, J is doe x upon doe u, and D is 1 upon x tilde norm square x tilde u tilde transpose. Control law using network inversion; first of all, we present a theorem.

(Refer Slide Time: 36: 59)



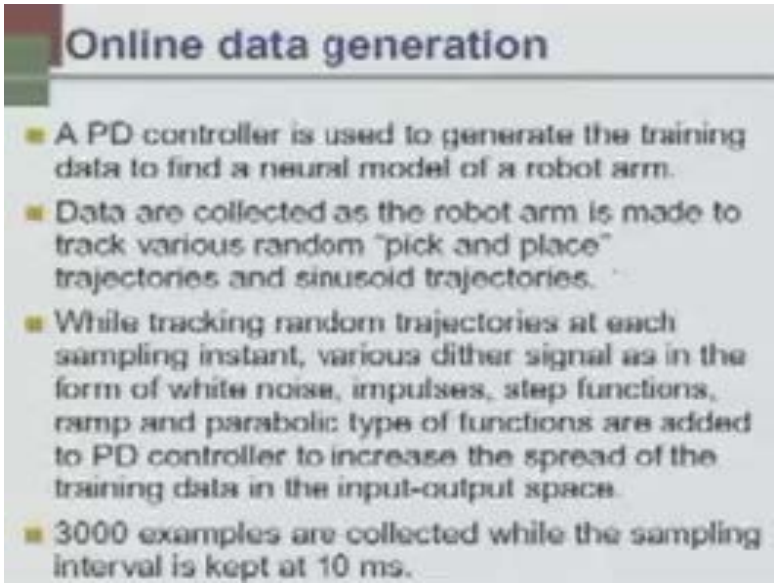
If an arbitrary initial input activation u not is updated by this formula, this identity u(t) is u not plus 0 to t dash u dot d t where, u dot is given by this expression then, x tilde converges to 0 under the condition that, u dot exists along the convergence trajectory. Substituting u dot in previous V dot, you get V dot is minus x tilde whole square that is, the overall system will be Lyapunov stable. The iterative input activation update rule will be because, this is continuous update, so iterative will be u(t) is u t minus 1 mu u dot t minus 1. This is iterative actually because, t we have given in terms of time, maybe we can put this is k, k, k where mu is a small constant representing the update rate. Where, u dot is computed from this expression at the sampling instant k minus 1. We proposed two different control indirect adaptive control schemes and we found out the control law and weight update law. First case, we found out what should be the weight update law for the controller; in second case we found out what is the control law and now we will show that how this controller is effective.

(Refer Slide Time: 38:48)



We selected a high speed robot manipulator, whose dynamics are given this way where in this particular dynamic model C_{21} stands for cos q_2 minus q_1, and S_{21} stands for sine q_2 minus q_1 this two-link manipulator and the parameters a_1, a_3, a_4 and a_2 they are 0.15, 0.04, 0.03 and 0.025 kg meter square respectively.

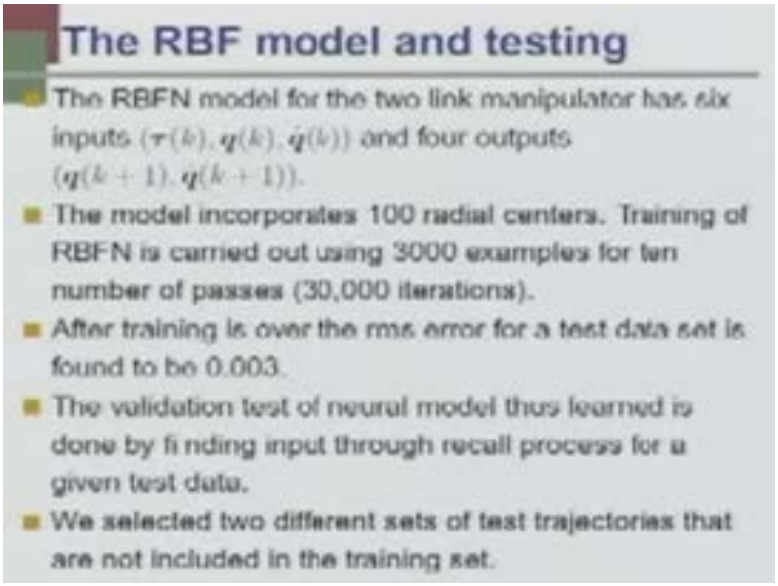
(Refer Slide Time: 39: 42)



We have selected the same robot manipulator that we have been discussing in the last two classes. The online data generation scheme for training the radial basis function network

is the same. A PD controller is used to generate the training data to find a neural model of the robot arm. Data are collected as the robot arm is made to track various random “pick and place” trajectories and sinusoid trajectories. While tracking random trajectories at each sampling instant, various dither signals in the form of white noise, impulses, step functions, ramp and parabolic type of functions are added to PD controller to increase the spread of the training data in the input-output space. This is from; we shift the data from n manifold to n plus p manifold. 3000 examples are collected while the sampling interval is kept at 10 milliseconds.

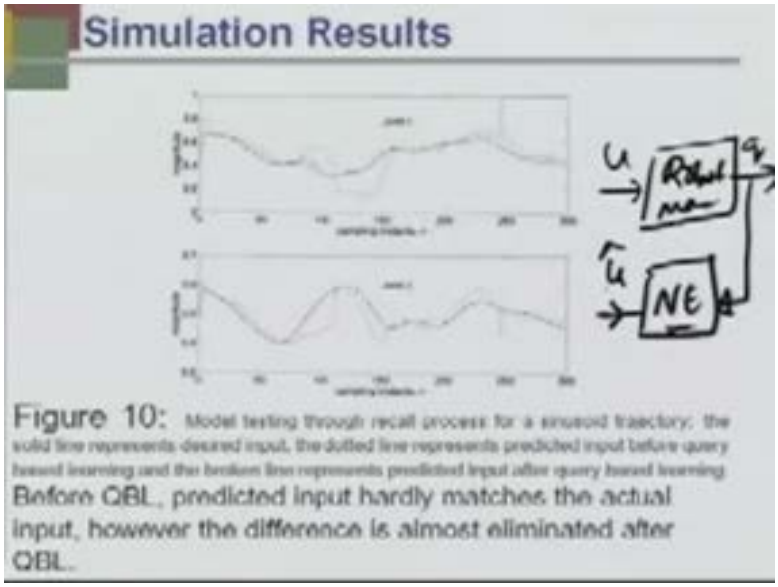
(Refer Slide Time: 40:38)



Then radial basis function network which has 6 inputs that is, two terms for joint inputs, two terms for angular positions, two terms for angular velocity at sampling instant k. When you give to the network, the network will predict what should be the actual value of the angular position and angular velocity at k plus 1. The model incorporates 100 radial centers. Training of RBFN is carried out using 3000 examples for 10 numbers of passes that means 30,000 iterations. After training is over the RMS error for a test data set is found to be 0.003. The validation test of neural model thus learned is done by finding input through recall process for a given test data. It is not simply because, when you have the RMS error 0.003 then, you can easily predict the output. The radial basis function network will effectively predict what should be the output given an input, but

what is the case. Given an output, can the same network predict the input? These results we showed in the last class.

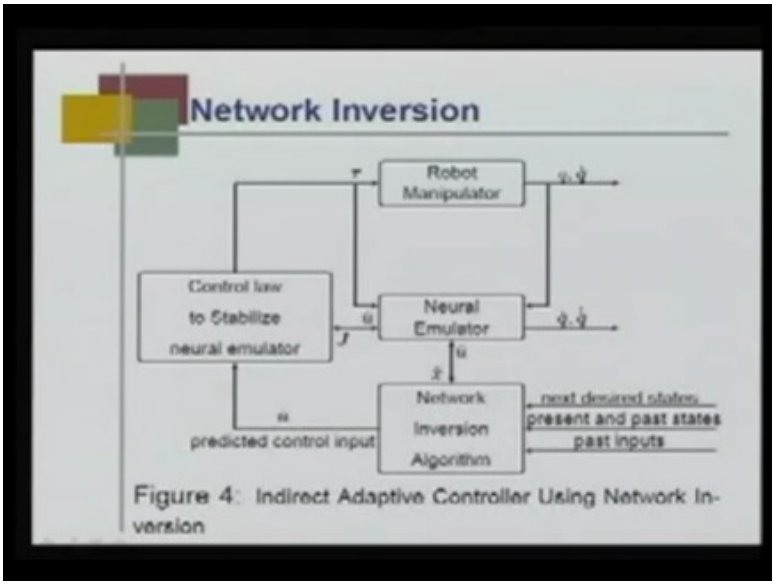
(Refer Slide Time: 42: 06)



This is a for a sinusoid trajectory when robot manipulator is tracking a sinusoid trajectory, we asked a question to the neural emulator corresponding to the robot manipulator. What is the input given the target output? That input is already given to the robot manipulator; so what this result is all about is I can tell you again. This is my robot manipulator and this is my neural emulator, this is my u I give to the robot manipulator and I get q, I give that same q to the neural emulator and predict, what should be that u. If this u hat matches this u, then this neural emulator is a good emulator of the robot manipulator. I hope you understand very clearly; again I repeat what is the meaning of this validation through inversion. Validation through inversion means, of course we have trained this neural emulator by giving the training, set u and q to this. Obviously, if I give u this will predict what q is but, given q can I predict the u? This is the question we obtained. Both ways the neural emulator should do a perfect job and then it is a robust identifier or robust model of the actual robot manipulator. What you are seeing is that, we have already discussed in the last class that, before the query based learning that we discussed today, also yesterday, that before query based learning the input prediction was very bad. You see that these are lot of variation from this solid line, solid line is the actual

control input that is given to the robot arm for a desired sinusoid trajectory but, after query based learning for the joint 1 you see that, the predicted one, the broken one and the solid one, they are almost very close. Where the dotted one this is before query based learning. After query based learning, the model has become very robust, not only the neural emulator is predicting the target given input, it is also predicting what should be the input given the target and the same thing also valid for the joint 2. You see that, before query based learning again you see these dotted lines they were not good. They were not similar to the actual the solid line that is the control signals sequence given to the actual robot manipulator. But, after query based learning you see that, this broken line almost follows the solid line. This is the advantage; this is the neural emulator, we have now utilized for testing our controller. I will not discuss because, these things we have already talked in the last class. We will now show the simulation result for the two control algorithms that we derive today to repeat for your own understanding.

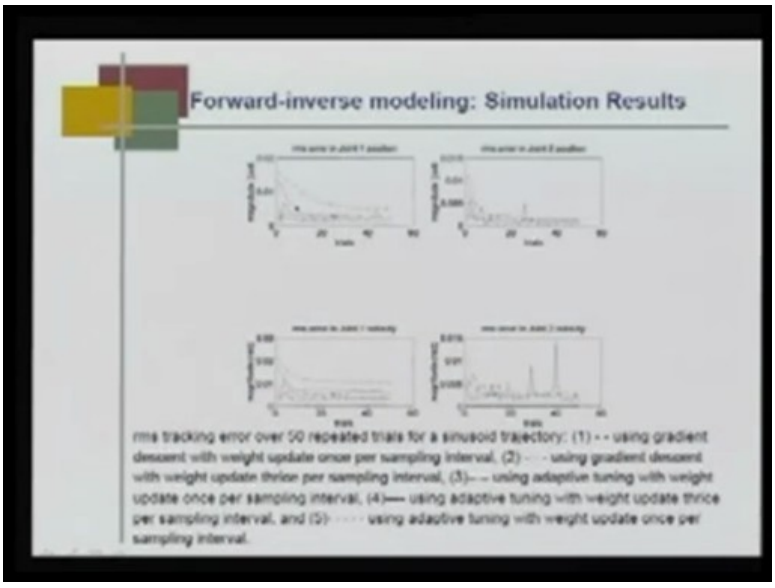
(Refer Slide Time: 45:32)



We first of all propose a forward - inverse modeling. In forward - inverse modeling, this is my robot manipulator, this is my neural controller. Neural controller is supposed to actuate, you know a feed-forward torque such that, the q tracks q d the output of the manipulator tracks q d. The neural controller is a radial basis function network whose W is not known, now how do I update this W in such a manner that my q follows q d and

this is Lyapunov stable. For that, as you saw that today in this class, we derived a weight update law called W dot and this W dot is computed by using two terms first term del doe x upon doe u is computed from the neural emulator. Another term we computed this term, doe u upon doe W from the neural controller itself using these two. We computed what is W dot and this W dot in the first case this was: x tilde norm square upon J transpose x tilde norm square into J transpose x tilde and the second rule that we saw we added a gradient term of the weight, where H is half W transpose W. When we took this value for H and we differentiated that, we found that this weight update rule is also Lyapunov stable or gives us Lyapunov stability. Using these two rules, we did simulation; this is forward-inverse modeling simulation. What you are seeing is that we provide the robot manipulator the same trajectory for 50 times, we start here, what we started with this neural controller. The weights are all initially randomly initialized.

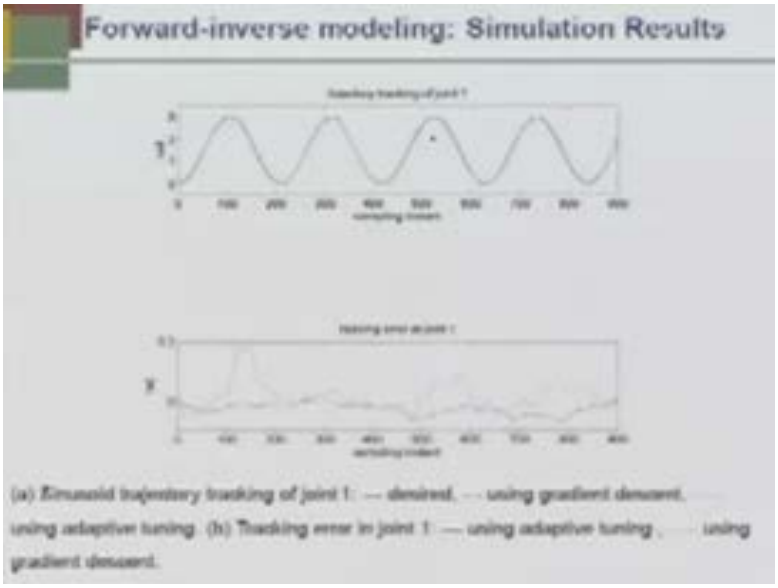
(Refer Slide Time: 48:21)



We did not know, now with this initial random weight, we used various algorithms first are gradient descent. So, what we did is that, we simply updated the weight according to the gradient descent and you see the over 50 trails the error in joint 1 position is this is trial 1, this is the RMS error and slowly the RMS error reduced and it is here. But, when we use the gradient descent but with weight update thrice per sampling interval then, you see that error further decreased. Then the third one is the adaptive tuning. First we have

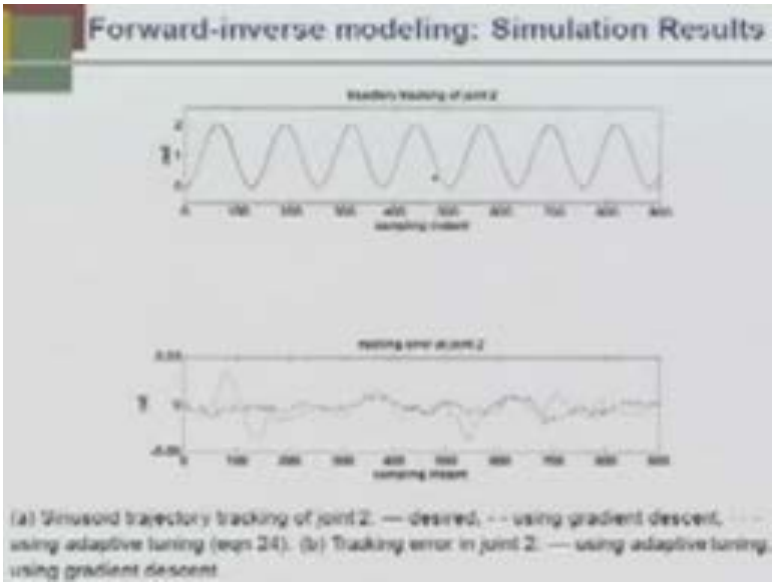
two update rules; the first type of update rule. You can easily see this is the upper one and then you have the same adaptive tuning algorithm but, weight update thrice per sampling interval, then you see that it is further improved. But, you see the amazing aspect of the adaptive tuning that is the type 2. When you do the type 2, you see that, error is actually almost 0 and independent of trial, it does not require any trial that is; instantaneously the error goes to 0 point 000001. This is a very fantastic influence of this controller; the controller that we talked is this type 2. You see, the type 2 controller here also the error is on the x-axis means 0 point 000001. Here also, you see a joint 1 positive, the error is very small and here also the error is very small; whereas, other schemes they have relatively large errors.

(Refer Slide Time: 50:46)



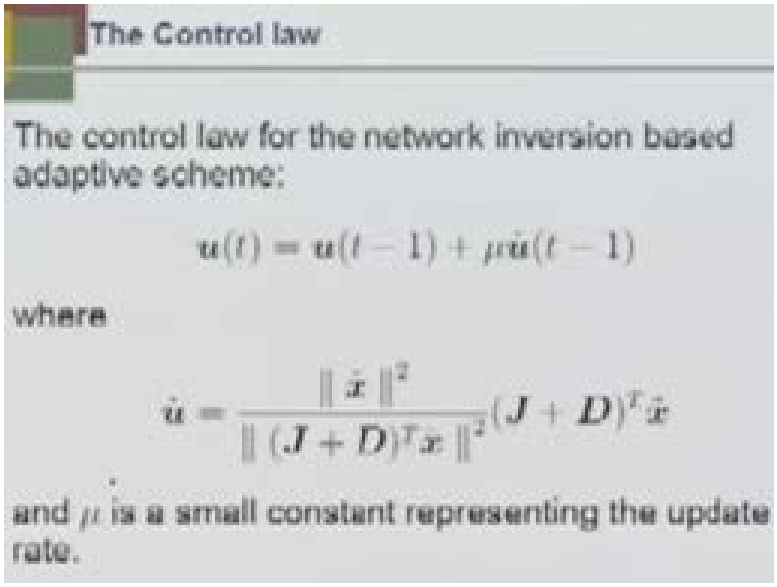
We compared, now we will take an example here. This is the sinusoid trajectory that is being tracked by this robot manipulator, this is joint 1 angle. You can easily see in the beginning there are some error and that error died down as time progressed and you can easily see there are two, this is the tracking error, this is trajectory tracking, this is the tracking error at joint 1, this solid line is using the new update rule; whereas, the dotted one is using gradient descent.

(Refer Slide Time: 51:33)



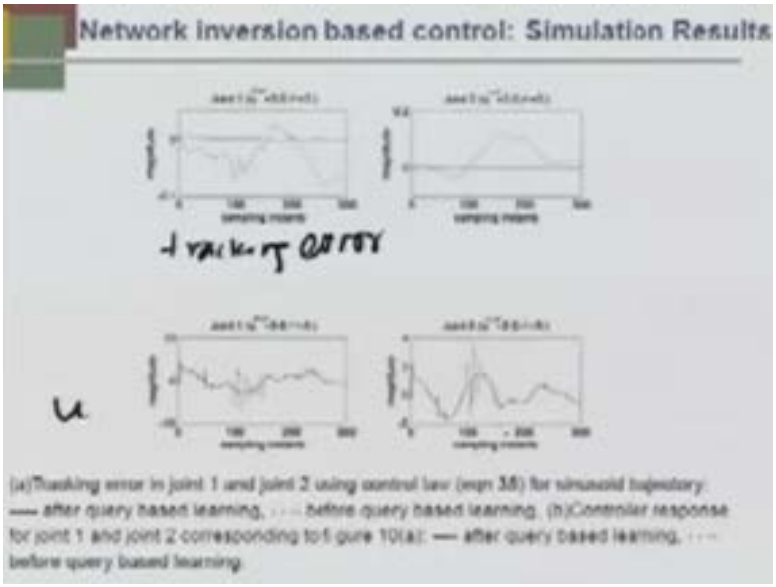
Similarly, here on joint 2 the tracking is very perfect; whereas, because you cannot even see the two trajectories, the actual trajectory and the desired trajectory they are super imposing perfectly that they appear to be the same trajectory, tracking is perfect. Tracking error at joint 2, if we see in a very micro scale then you see that for gradient descent the error is quite visible; whereas, this is almost 0 for the adaptive tuning that we have done using Lyapunov stability theory. That is a simulation result of the first part; the second part is a network inversion. A network inversion what we did is that, we have already a neural network. We said why we should put another neural network there. Instead we have neural network that has already identified the model of the robot manipulator. We utilize that neural network to predict our control law such that, the overall the system is Lyapunov stable. What should be the control input u dot here or the u here? We say this is my u or tau. That will be a function of u hat that has been given from the network inversion algorithm and J is the Jacobian that is computed from the neural emulator that is Del x upon Del u.

(Refer Slide Time: 53: 18)



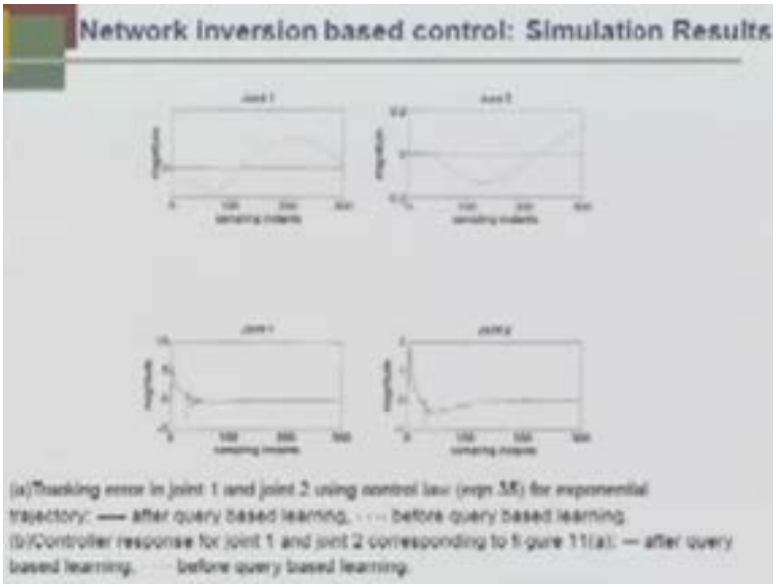
We derive this algorithm, my control algorithm is u (t), u is same as tau is u (t minus 1) mu u do and where u dot is this expression, which can be computed very easily because, these are simply norm square J is a Jacobian matrix computed from the neural emulator. D is an expression that we already expressed in this class. When you implement this thing, here what you are seeing is that, we have two robot manipulators and this is joint 1 and this is joint 2. The A is tracking error in joint 1 and joint 2 using control law for sinusoid trajectory after query based learning and before query based learning and controller response; this is the controller response.

(Refer Slide Time: 54:38)



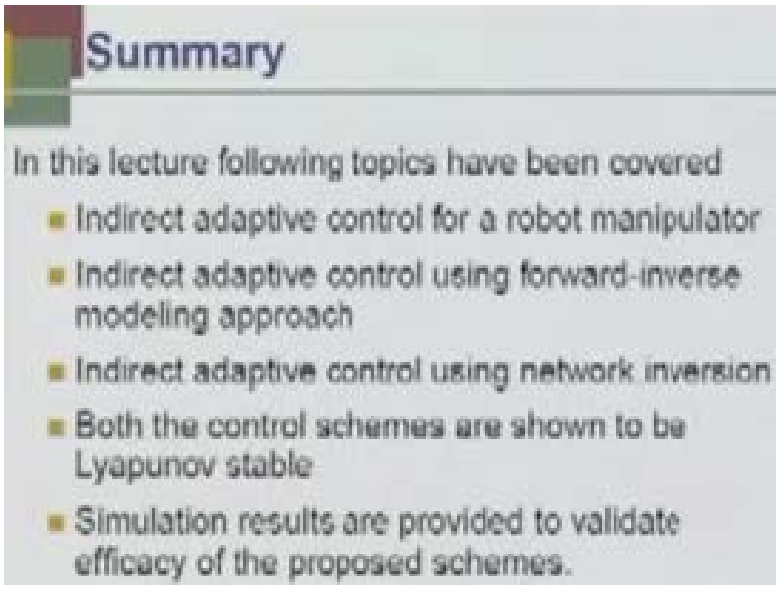
Controller response for joint 1 and joint 2 corresponding to figure; this is the controller response; this is tracking error, and this is u; this curve gives you u. You see that, this dotted line what you are seeing here this is the control action before query based learning, we implemented. You implement this network inversion control before query based learning. This is, you see that controller is fluctuating but, s_1 is you did the query based learning, then the controller is smooth the solid line. Correspondingly, when the controller was not smooth the error was very large but, when controller becomes smooth the error is almost very negligible in joint 1. Similar is the case with joint 2, you see that the controller is fluctuating here, quite very much fluctuating before query based learning and that means, when a not properly trained neural network is used, then we have a large error. But, when the query based learning was completed, again implement the controller. Do you see that the torque is very smooth here and the error is very small?

(Refer Slide Time: 56:06)



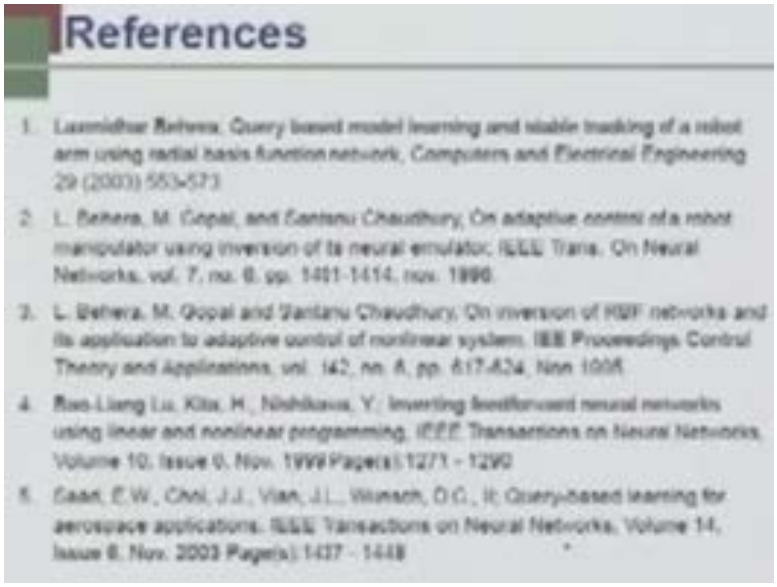
Similar thing here that in this case, that is the previous one is the sinusoid trajectory, this is exponential trajectory. For the exponential trajectory you see that, this fluctuation is before query based learning and fluctuation died down after I am implementing the controller after the query based learning, so that is the solid line, that is quite smooth. Corresponding to solid line, this solid curve here implies the error in joint 1 tracking is almost negligible not there; whereas, without query based learning the error is there always existing. Similarly, joint 2, with query based learning this is a solid line almost no error and error is there before query based learning, and you can see this fluctuation here that indicates that controller control actuation is not smooth.

(Refer Slide Time: 57:21)



In summary, indirect adaptive control for a robot manipulator we discussed today. This indirect adaptive control has two different architectures we proposed, one is indirect adaptive control using forward-inverse modeling approach, another is indirect adaptive control using network inversion, we say these are indirect adaptive control because, we are utilizing the neural emulator the forward modeling that is the forward dynamic model, we trained a neural network that captures the forward dynamics of the robot manipulator and utilize that neural network to tune our controller or to update our controller. Both the control schemes are shown to be Lyapunov stable, simulation results are provided to validate efficacy of the proposed schemes. We saw that, how our tracking is almost perfect when we have the neural emulator that is perfectly trained using query based learning and then we implement this controller, the result is fantastic.

(Refer Slide Time: 58: 51)



Those who want to pursue further work in this the last three classes their kind of one in a box type of course, you can follow these references that we have given here for further work. First one is our own paper that is published in 2003 in Computers and Electrical Engineering Journal, second one is again our paper published in IEEE Transaction Neural Network in 1996 and this is volume 7, number 6. The third paper is again our paper is IEEE proceedings control theory application in 1995, volume 142 and number 6. This is another paper that is called Inverting Feed-forward Neural Networks using linear and nonlinear programming by Bau-Lian Lu and H.Kita Nishikawa Y this is in IEEE Transaction Neural Network volume 10, issue 6 and another one is a query based learning for aerospace applications that is there in IEEE Transaction and Neural Networks volume 14, issue 6, November 2003. So thus, that should give you a good exposure on how to design indirect adaptive control schemes for robot manipulators in particular and in general for nonlinear systems.

Thank you.