Intelligent Systems and Control

Prof. Laxmidhar Behera

Department of Electrical Engineering

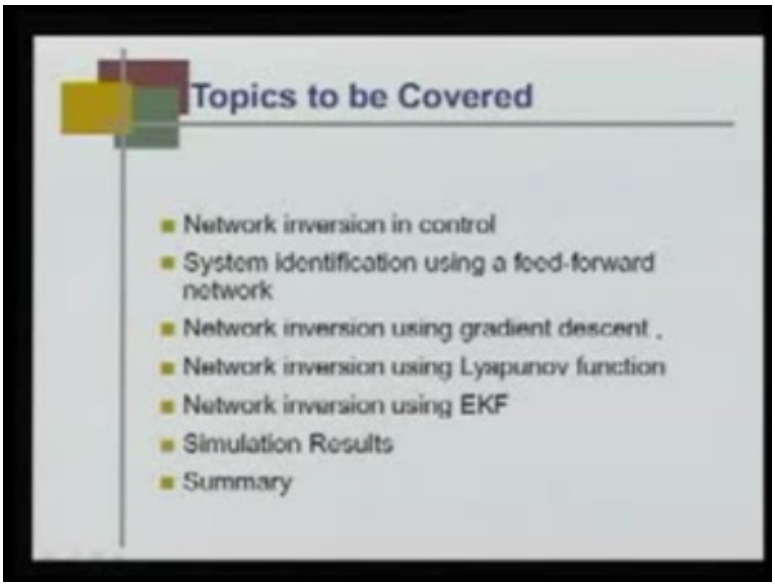
Indian Institute of Technology, Kanpur

Module - 3 Lecture - 2

Network inversion and control

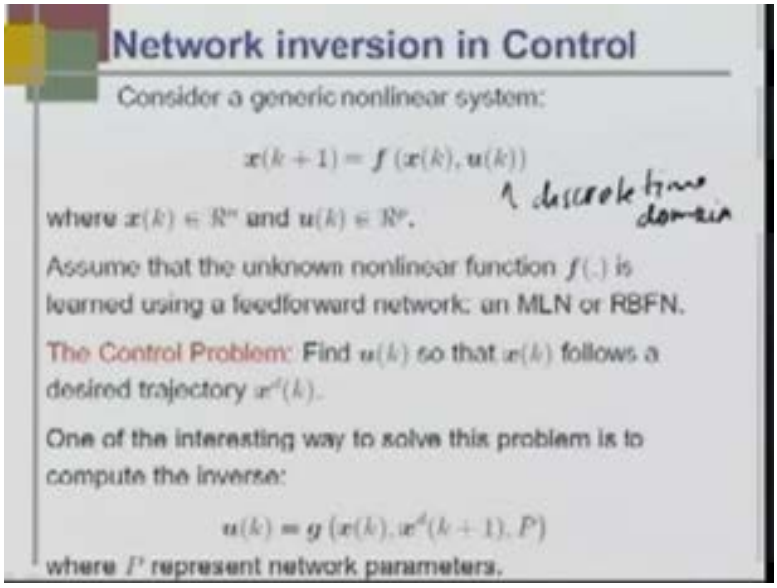
This is the lecture two on module three, which is in neural control of this course on intelligent control. The topic today will be Network inversion and control. In the last class, we gave an overview of various control schemes using intelligent techniques. There, we briefly talked about network inversion. We would be elaborating those ideas that we talked very briefly in the last class, in this class, as well as in the future classes on this module entitled neural control.

(Refer Slide Time: 01:27)



The topics that we will be covering today: network inversion in control, system identification using a feed-forward network, network inversion using gradient descent, network inversion using Lyapunov function, network inversion using Extended Kalman Filtering; EKF stands for Extended Kalman Filtering, simulation results and summary. What is a network inversion in control?

(Refer Slide Time: 02:51)

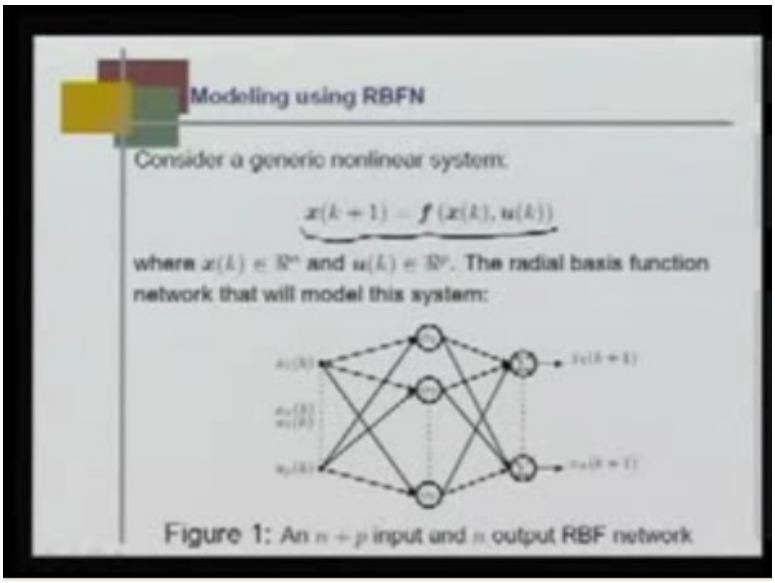


I will first explain this concept. Look at this generic non-linear system. Generic non-linear system, the most generic any kind of non-linear system can be expressed in this form. That is x k plus 1 is f of x k and u k; where, x k the n-dimensional vector is the state vector for the system. That means, the physical system is defined in a state space which is n-dimensional and it is a multi-input system, u k which is p-dimensional. So, we have p input system in a discrete time domain. So, you can easily see this is discrete time domain; the expression is x k plus 1 is f x k and u k, f is some unknown non-linear function.

We do not know what that non-linearity is and we assume that this non-linear function f can be learnt using a feed-forward network that can be either MLN or RBFN. So, the usual control problem is: find u k the control law here, so that the state vector follows a desired trajectory. So, that is the normal control problem defined for any physical system which has a generic non-linearity. So, one of the interesting ways to solve this problem is to compute the inverse. That is, if I compute u k equal to, this is my control law which is g x k x d k plus 1 P. So, that is a function of x k x d k plus 1 (Refer Slide Time: 04:59) the desired trajectory x k plus 1. x k is the present state, this is the state at the next sampling instant and P represent the network parameters. That is, this f is now learnt by a network and that is either a multilayered network or a radial basis function network for any feed-forward network. Given that scenario we are asking a question, can we find out

u k which is a function of x k x d k plus 1 and network parameters such that, finally this control objective is achieved. What is the control objective that x k will follow the desired trajectory x d k? This is called network inversion.

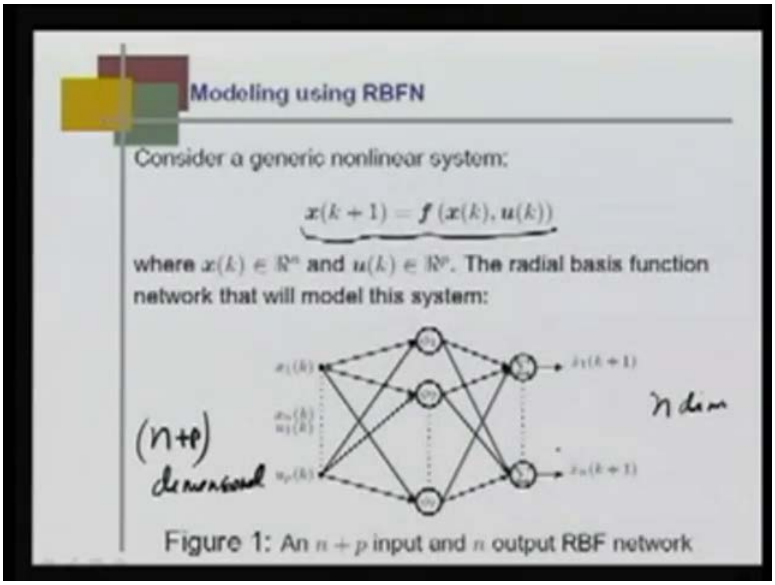
(Refer Slide Time: 06:13)



Let us look at the network. This is our non-linear system which we discussed x k plus 1 f x k u k. This particular dynamical system, the actual system has been modeled using a neural network which is a radial basis function network you are seeing here.

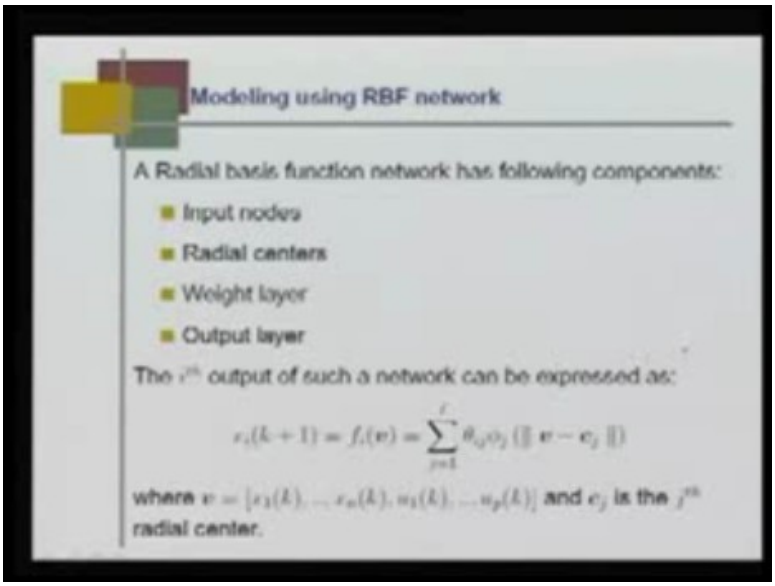
Obviously, the input to the system because, at kth sampling instant I know what is my state vector and I know the input that I am actuating to the system; so, x_1 k until x_2 k until x_n k; similarly, u_1 k until u_p k.

(Refer Slide Time: 07:10)



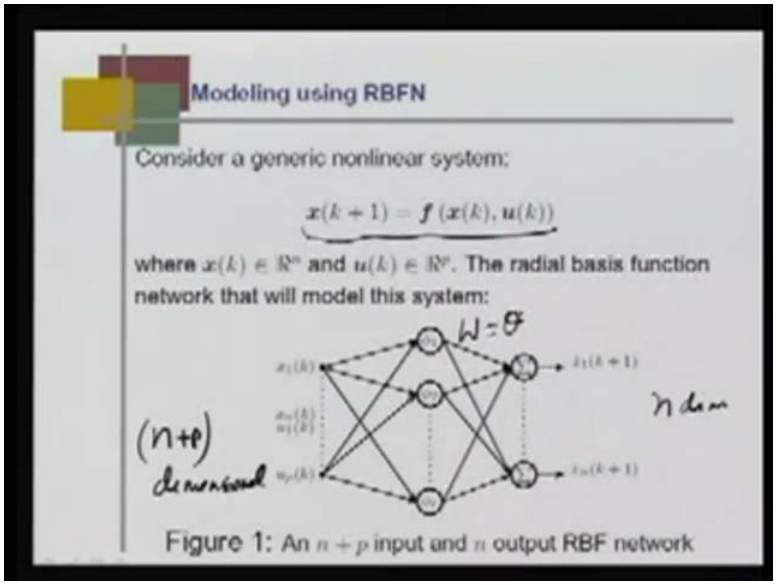
So, you can easily see the input is n plus p-dimensional, input is p-dimensional and output is, again you can see my output is x k plus 1 until x_n k plus 1. So, this is my n-dimensional output. You can easily see that n-dimensional output is predicted by the system given the present state from the actual plant. That is why you can easily see here that, we have put a hat here and a hat here meaning these are estimated by the model. That is radial basis function network using the real time data that are generated from the plant. So, if a plant has n-dimensional states, plant dynamics is defined in terms of a state space with n dimension then, obviously, it has n states and those data we are assuming are readily available in real time. Given sequence of input data, you collect this data and train the network. Obviously, I explained to you in this network that it has n plus p input n is the present states p is number of inputs and that would actuate a state's future state. Obviously, we expect that, the future state is exactly the state that actual plan dynamics will move to from the present state. So, when I talk of a radial basis function network you can easily see it has input layer, this is the center, this is the output layer, and in middle this is the weight vector for the network (Refer Slide Time: 09:40).

(Refer Slide Time: 09:44)



So, that is what we are saying the radial basis function network has following components: input nodes, radial centers, weight layer and output layer. I think we have already discussed this radial basis function network in our module on neural networks. The ith output of such a network can be expressed as x_i k plus 1. The state vector at k plus 1 instant is the function of v and this can be written as: j equal to 1 to n, n is number of radial centers (Refer Slide Time: 10:38 to 11:20). So, j equal to l to n, $theta_{ij}$ is actually weight, which I said w, this is actually $theta_{ij}$, and phi_j is the activation function in the center, v is the input which is here. As I said, the input to the network is n plus p-dimensional that is x_1 k until x_n k and again u_1 k until u_p k. So, this is my input to this network which is here.

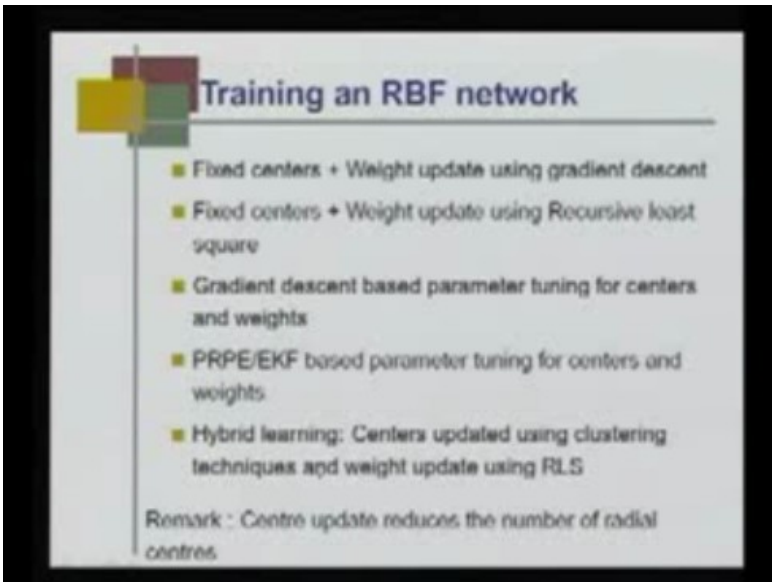
(Refer Slide Time: 11:30)



This is my input v and this w is represented by theta in this particular network. So, this is $theta_{ij}$ and phi_j is the activation or the radial basis function is normally known as basis function. This is the norm; this normally is v minus c_j norm is normally equilibrium distance we take but, other measures also can be taken but, in all our work we present v minus c_j and what is c_j? c_j is the jth radial center and this radial center c_j has the same dimension as that of v. So that is why, we can always compute v minus c_j norm, they must have the same dimension otherwise we cannot write this expression. So, v minus c_j norm and phi_j is the basis function of this norm. Once we know what is the radial basis function network which we have already discussed I will just review how we train the radial basis function network. If we will look at the radial basis function network; the parameters in this hidden unit or radial centers each node is associated with a center called c_j and c_j has same dimension as that of the input (Refer Slide Time: 13:11). As well as this theta the weight vector theta; so these two - the weight vector theta and c_{ij} and how many c_{ij}s will be there? l c_{ij} because, l units are there.

We have to update these weights such that, the given input the output is properly predicted. So, there are various methods I will not discuss in detail in this particular class because, we have already discussed that earlier.

(Refer Slide Time: 14:00)

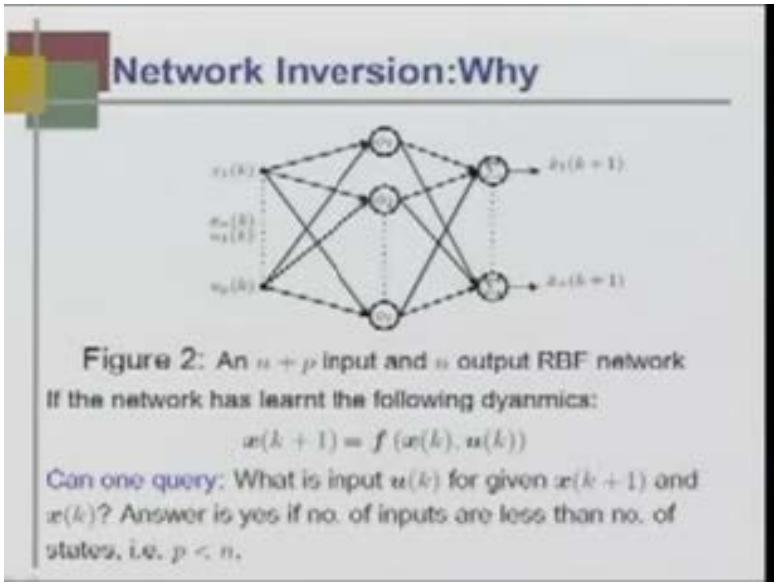


So, you have various methods by which you can train a radial basis function network fixed centers and weight update using gradient descent that is, c_j are already fixed, weight update using gradient descent and fixed centers weight update using recursive least square. You can also do gradient descent based parameter tuning for centers and weights; both weights and centers can be updated using gradient descent. We can also use EKF algorithm for EKF based parameter tuning for centers and weights; similarly, hybrid learning.

So, these are all means by which we normally estimate weights. Normally these methods are based on least square estimates; whereas, in hybrid learning, the centers are updated using clustering technique and weight update using recursive least square. So, then this is hybrid because, we can also say the first two also a category of hybrid because, centers are already fixed, but normally in strict sense hybrid means centers are updated using clustering technique; also the way we fix the center certain meaning of clustering. Now remark is center update reduces number of radial centers using center update using either gradient descent or EKF. So, if we update using this GD or EKF; the normal observation is that, number of centers can be reduced drastically. So, what is the effect is number of centers in radial basis function network are reduced then, the computation time involved during control will be significantly less. So now the question is why is this network inverse? We posed in the beginning of the problem, what is the meaning of inverse control

that is the u? The control law is a function of the present state, the next desired state and the present network parameters.

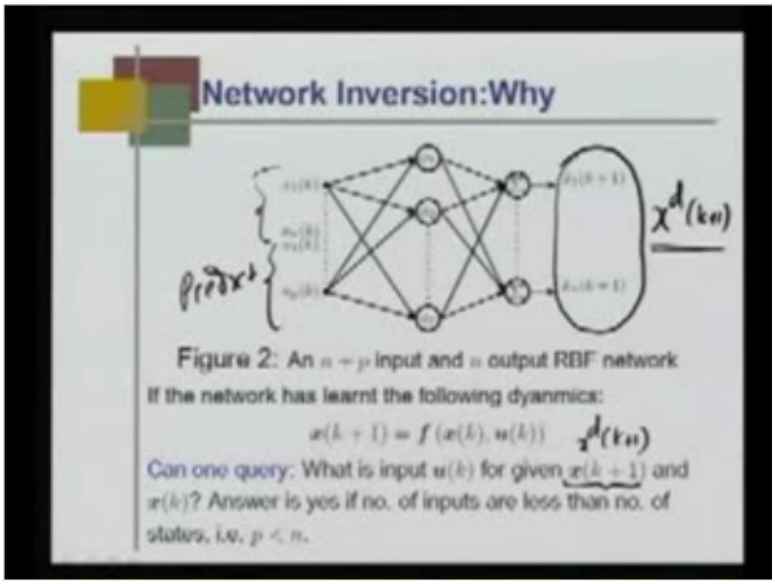
(Refer Slide Time: 16:46)



Now you see the same network that has modeled the plant that has n states and p inputs. So, you can easily see this is the network. If the network has learnt the following dynamics that we are again and again saying that, x k plus 1 is f x k u k this is already learnt f, so can one query what is input u k for given x k plus 1 and x k?

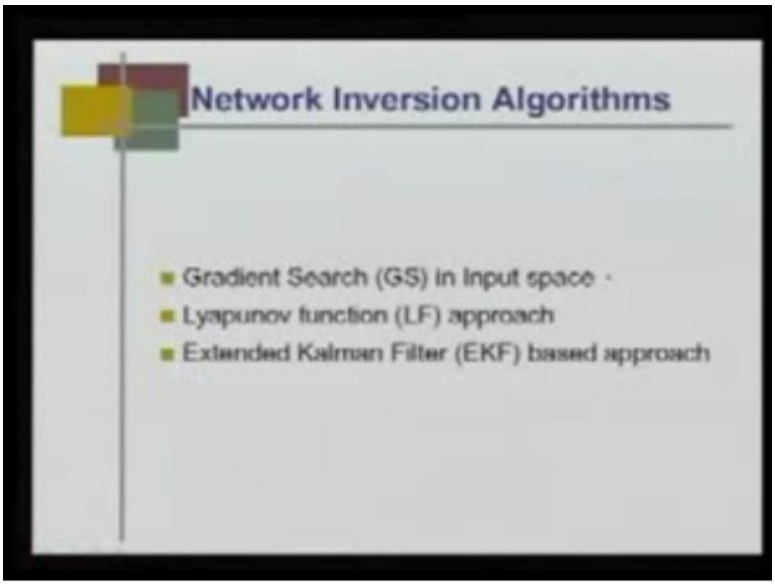
If I know because normally control is actuated looking at the current state, so x k is known and x k plus 1 is normally the x desired. Given x k, I want to go to x desired k plus 1. So, given the present state and the desired k plus 1, what is input u k? Now this network is given to me.

(Refer Slide Time: 17:55)



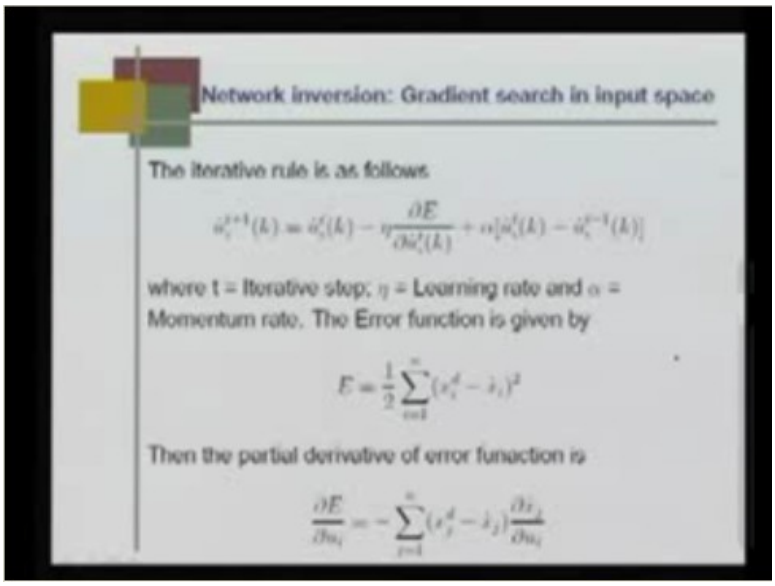
I assign these states, this response of the network to x d k plus 1 . That means this is x d_1 k plus 1 x d_n k plus 1 . So, given x d k plus 1 should be the output of the network and I know already the present x_1 k up to x_n k that my sensor has provided me this data from x_1 k to x_n k ; it can be sensor; it can be some observer. Now the question is that, if I say I know what is x_1 to x_n k is and I know at the output x d k plus 1, given the output at the target, can I predict the input? This is the query I am asking. The answer is yes, when can I predict? I can predict if number of inputs are less than number of states that is number of input is p and the number of output is n. So, if p is less than n, inversion is possible; why? Because, that establishes if number of outputs is more than number of inputs then, this is a case of unique marking. But, if this number is more than this then, for a given desired state and given current state, we can have multiple possibilities about the input. When this number of inputs is less than number of outputs then, for a given number of current states and for the desired state in the next sampling instant, the input is unique. This is called network inversion; that is, we can predict what should be the control input given the target.

(Refer Slide Time: 20:40)



The network inversion can be done in three ways: gradient search in input space, Lyapunov function approach and Extended Kalman Filtering approach.

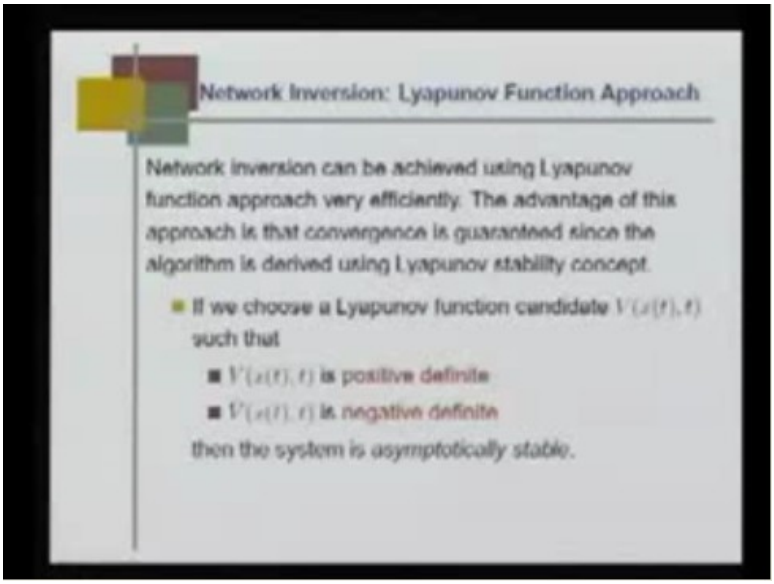
(Refer Slide Time: 20:48)



Network inversion using gradient search we are now very familiar in this class because, we have talked about this gradient descent principle unlimited number of times in this particular course. So, I will not discuss too much but, just like for weight update we use the gradient descent but, in this case network is already trained. So, network is trained

weights are already known. We do not have to update the weights, what we have to update, given a current state and given the desired state within the sampling time, I have to iteratively compute the input in such a way that input would finally result in the actual desired state. The radial basis function network will have the same state as that of the desired state. So you can easily see the iteration is taking place in during the kth sampling instant. u t plus 1 is u t minus del eta del E by del u t plus alpha u t minus alpha u t minus 1; where t is the iterative state, eta is the learning rate and alpha is momentum rate. The error function usually is taken like this: E is half i equal to 1 to n x_i d minus x_i hat whole square. So, this is instantaneous error function computed during kth sampling instant. Then the partial derivative of error function, is as usual; this is x_j d minus x_j into del dow $_{x\ a}$ dow$_{u\ I}$ and this can be computed from the network parameters, this quantity and then we put this quantity here and we implement this algorithm. So, this is simple radian descent, all of you already know that, just like you update your weights using gradient descent. Similar way you update the input but, what we are doing is iterative update of the input during a sampling instant. This is very important. So, network inversion we talked about gradient descent.

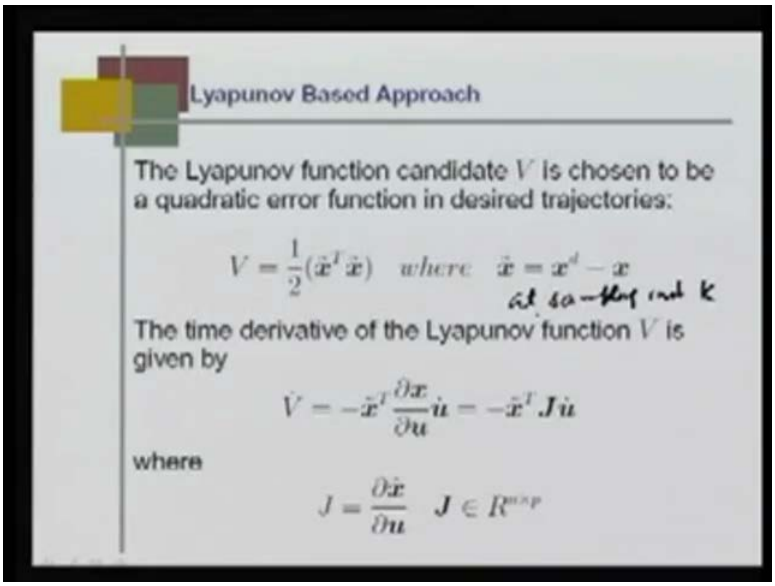
(Refer Slide Time: 24:13)



Now the second approach which is Lyapunov function approach. So, network inversion can be achieved using Lyapunov function approach very efficiently as well. The advantage of this approach is that the convergence is guaranteed, since the algorithm is

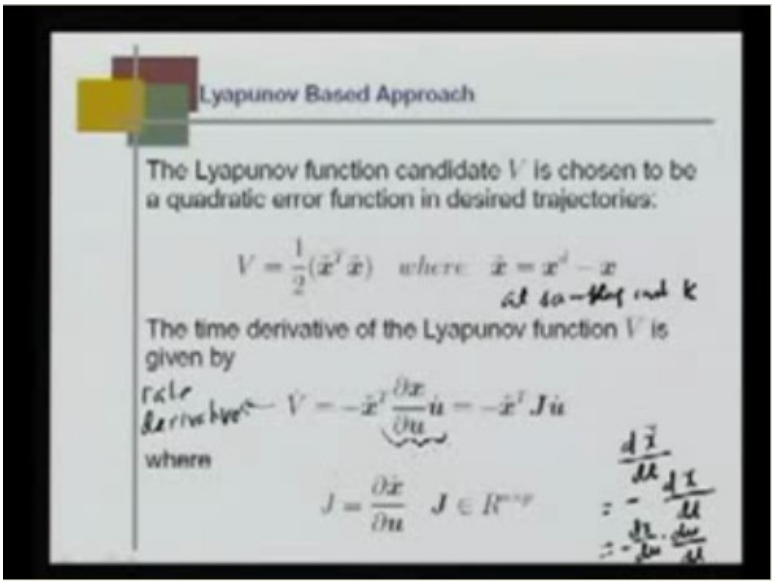
derived using Lyapunov stability concept. What is Lyapunov stability is we have already taken some special lecture in this course and Lyapunov stability criteria. So, here I will just remind you, if we choose a Lyapunov function candidate V such that, the Lyapunov function is positive definite and the rate derivative of the Lyapunov function is V dot is negative definite. If these two are satisfied then, the system is asymptotically stable. So, we use this concept to derive an algorithm for input update.

(Refer Slide Time: 25:19)



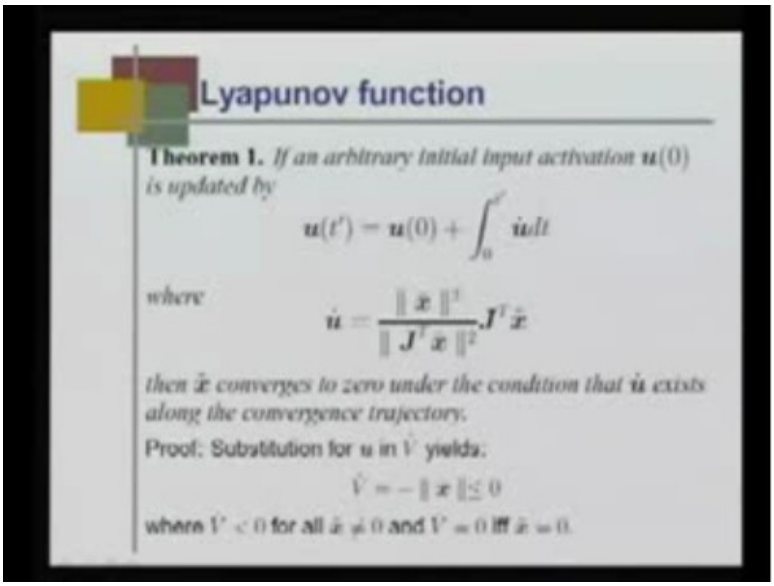
A Lyapunov function candidate V is chosen to be a quadratic error function in desired trajectories. Obviously, my V is x tilde transpose into x tilde upon half where x tilde is the error vector which is the desired state vector minus the actual state vector at sampling instant at instant k, so, we are not using k here just for the sake of clarity. We do not want to introduce another index there; that is why k is not there. The time derivative of this Lyapunov function V the rate derivative which is V dot, this is rate derivative. So, the rate derivative is you can easily see that this is minus x tilde transpose into x tilde dot or d by dt x tilde. So, d by dt x tilde can be written as this because x tilde is x t minus x. So, d x tilde by dt is you can easily see that, this is minus this goes 0. So, minus dx by dt and that can be written as dx by du du by dt. Minus sign is there; that is why you got a minus sign here. From here you can easily see, why minus sign has come so x tilde transpose and we say partial derivative because, x is not only the function of u it is also function of other parameters or other variables, but at the moment we are only considering u.

(Refer Slide Time: 27:18)



x tilde transpose dow x upon dow u u dot; u dot is same as du upon dt. This can be written as x tilde transpose J the Jacobean matrix Ju dot where J is dow x upon dow u.

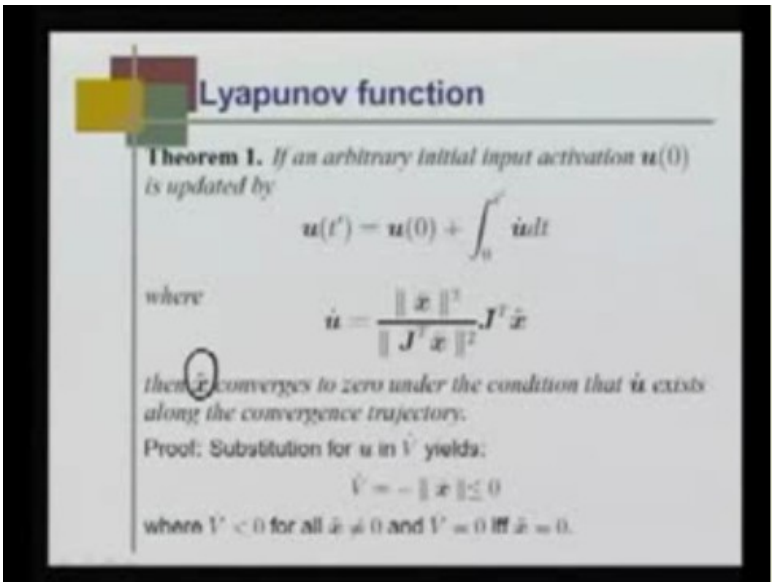
(Refer Slide Time: 28:19)



Now we will propose a theorem. If an arbitrary initial input activation function u not is updated by this formula which is u t dash is u_0 plus 0 to t dash u dot dt, where u dot is given by this expression which is x tilde norm square by J transpose x tilde norm square into J transpose x tilde when it is given by this formula u dot. Then, x tilde converges to 0

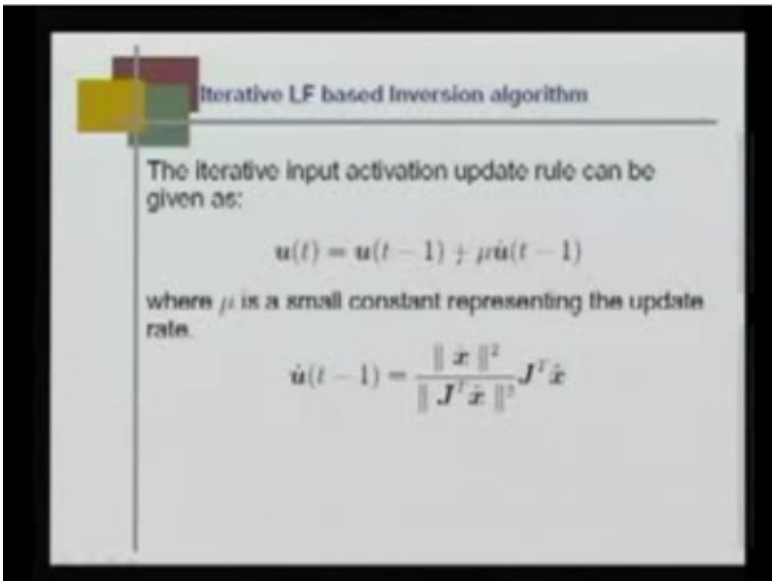
under the condition that u dot exists along the convergence trajectory. You must know that our desire during training is x tilde must converge to 0. Now, we are giving a theorem by which we are saying that if I update my input to the network in this particular rule then x tilde can converge to 0, which is the desired thing. The proof of this theorem is very simple.

(Refer Slide Time: 29:28)



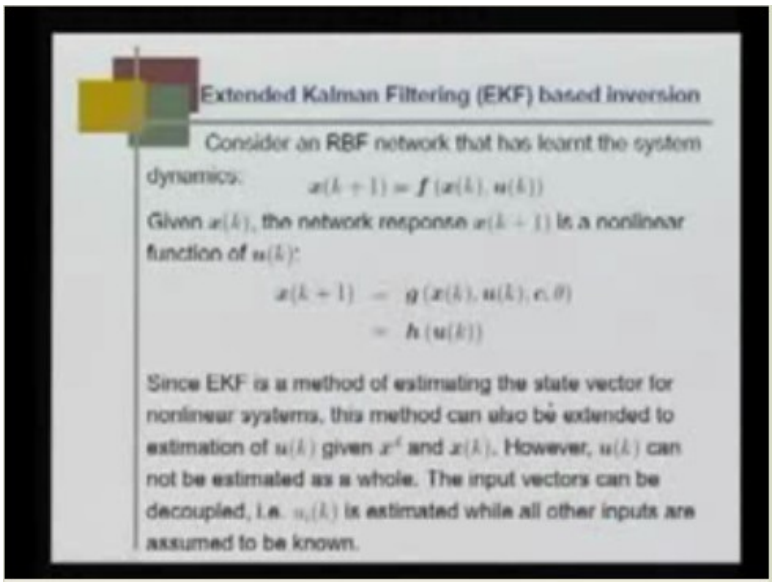
You take this u dot here and take to the previous expression is V dot is minus x tilde transpose J u dot and you replace this u dot by this expression here then, you get V dot equal to minus x tilde norm. So, this is a quantity which is always a positive quantity. V dot is always either negative or 0, it is 0 when x tilde is 0 that is fine because when x tilde is 0, that is what you need, the algorithm should stop there; that is the convergence point. This implies V dot is always 0 for all x tilde not equal to 0 and V dot equal to 0 only if x tilde is 0, so this proves that this theorem is right. Once I explained what u dot is then, I write a simple iterative form, how to compute it.

(Refer Slide Time: 30:45)



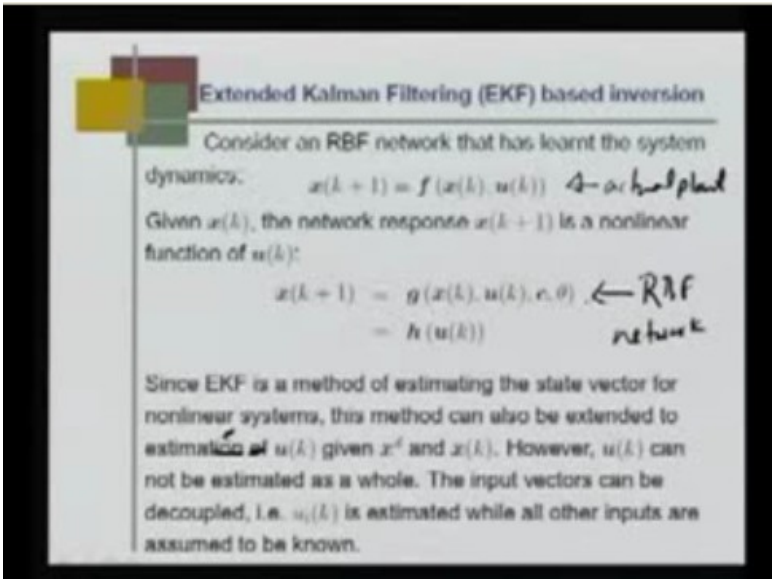
u t is u t minus 1 plus mu which is a small constant u dot t minus 1 where, u dot t minus 1 is given by this expression. This is u dot is computed by all these expressions that were computed at the instant t minus 1 during the iteration t minus 1 because, we are updating control input in kth sampling instant from some initial value. That finishes the second algorithm.

(Refer Slide Time: 31:33)



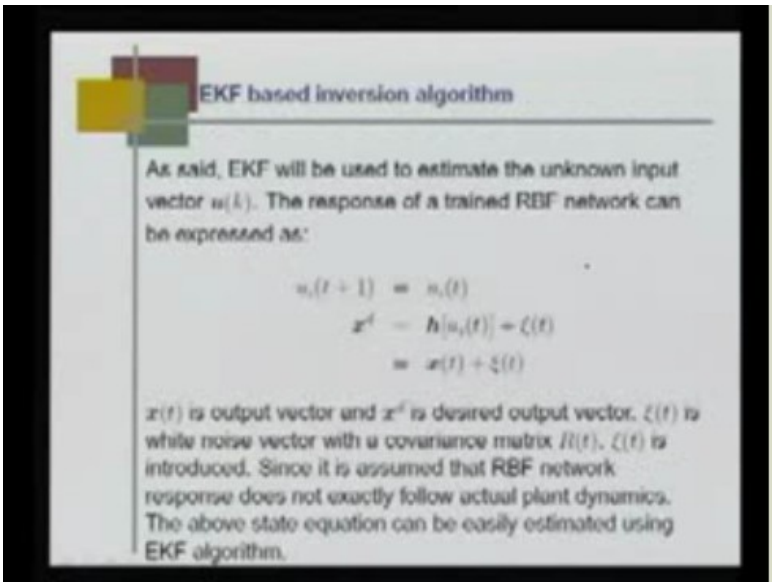
Now, the third algorithm that we will be talking about is Extended Kalman Filtering based inversion algorithm. What is this Extended Kalman Filtering based inversion algorithm? Consider an RBF network that has learned the system dynamics which x k plus 1 is f of x k u k given x k the network response x k plus 1 is a non-linear function of u k which is x k plus 1 is g x k u k c and theta. So, this g is my actual plant and this is my radial basis function network. In that sense this g is same as f if the training is proper or I can write x in this particular manner and this is h u k. Since EKF is a method of estimating the state vector for non-linear systems this method can also be extended to estimate u k given x d and x k. However, u k cannot be estimated as a whole because u k is a vector. The input vectors can be decoupled; u_i k is estimated while all other inputs are assumed to be known.

(Refer Slide Time: 33:18)



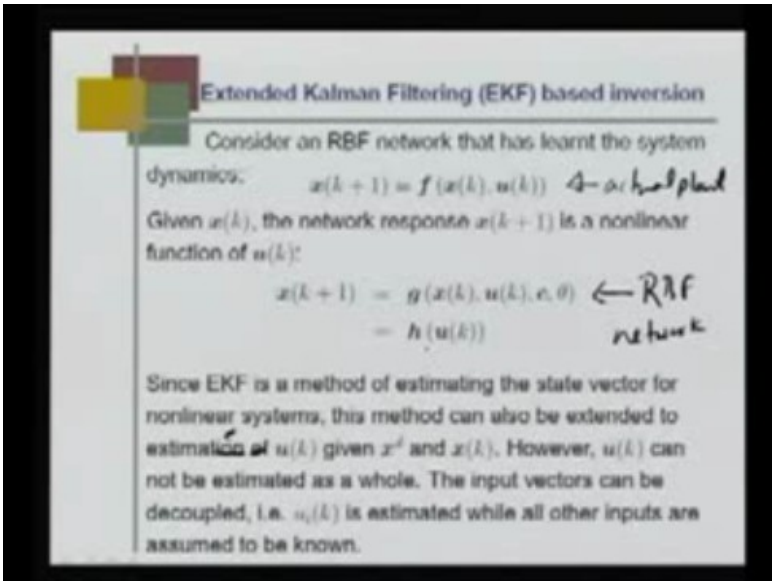
So, what we are saying is that you can easily see what network inversion is. Network inversion means given these equations because, this equation represents the response of the radial basis function network where, radial basis function parameters are the center c and theta the weight vector. So, the question we are posing is that, given x k and given desired x d, how can we find out what is u k, to find out u k, we will use Extended Kalman Filtering.

(Refer Slide Time: 34:04)



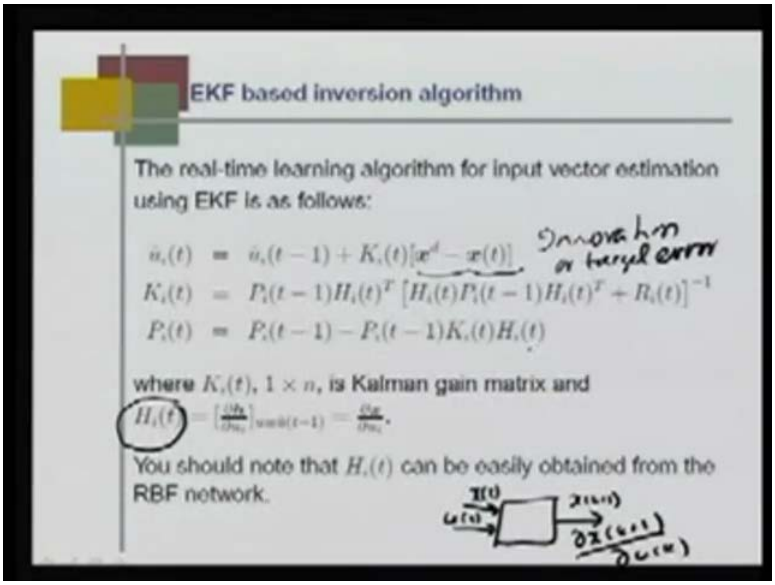
We decouple what we wrote here. We cannot iteratively update all the elements in u k vector instantaneously but, you can decouple them by which we can write the radial basis function network equation to be u_i t plus 1 is u_i t because, given a sampling instant, to achieve desired state vector, given the present state vector the input is a constant value input should not change. That is why, u_i t plus 1 is u_i t and desired state that is radial basis function network output is a function of h u t you can easily see what we wrote here.

(Refer Slide Time: 35:11)



h u t plus zeta, this zeta is because, even if radial basis function network is trained properly, there will be some error between the actual response and desired response. So, this zeta t which is a white noise vector with covariance matrix R of t is introduced there, to take into account of that anomaly and the question is that, now we can easily see, this is set in an ideal format for which we can directly apply the Extended Kalman Filtering principle to compute u_i t. So, to do the normal equation using Extended Kalman Filtering equation this is the three sets of equation we get. One is the estimate of $u_{i;}$ based on the previous estimate of u_i plus the Kalman gain here into the innovation term, this particular term is called innovation or you can also say error because, this is my desired state and this is my actual state. This is my the target error innovation or we can also say target error during update and then the Kalman gain this K_i which is known as Kalman gain can be also computed in terms of…, this is error covariance matrix is P_i and this H is actually dow H upon dow u_i which is dow x upon dow u. This can be directly computed from the network because, given a network, my input is u and this is x k, but since this x k is known, my output is x k plus 1. I can always compute dow x k plus 1 by dow u k in this; given all the network parameters, I can compute this.

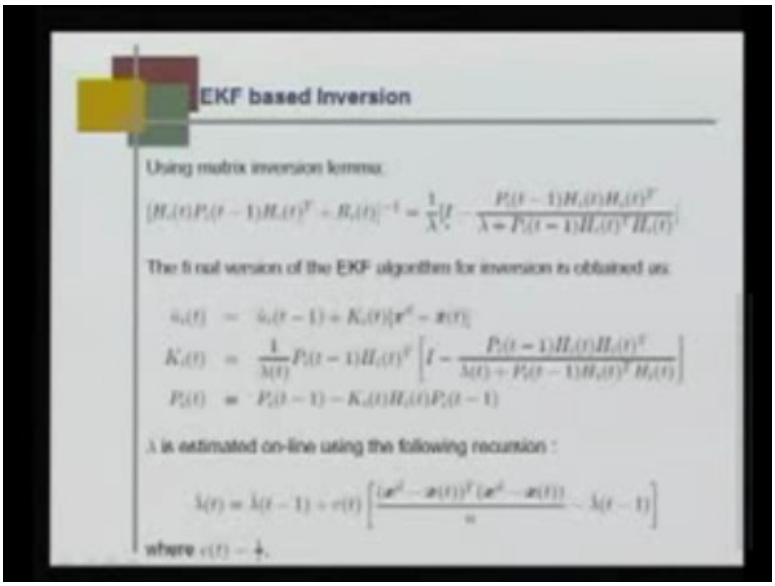
(Refer Slide Time: 37:02)



This computation is possible and this same computation is what is H_i t. This is what Hi t is. This again is covariance matrix P_i H_i t transpose plus R_i t this R_i t as I said it is the covariance term associated with the noise vector. This is the inverse of this quantity and

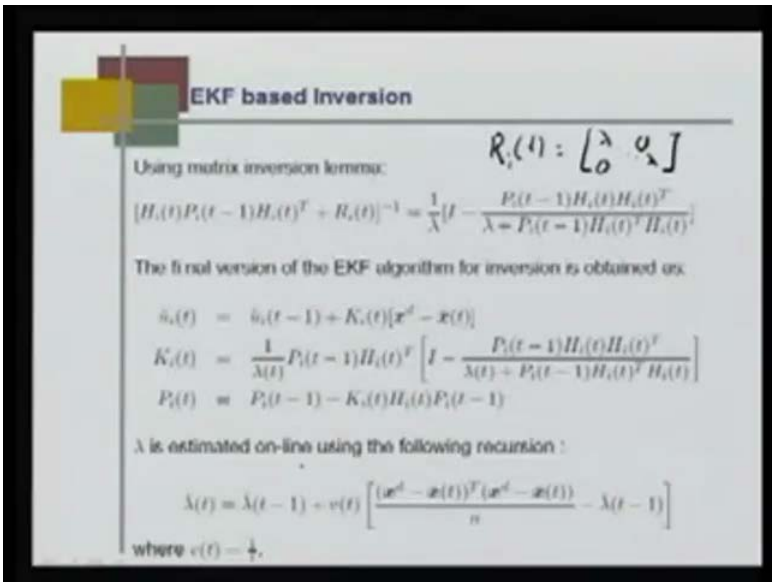
this normally is known as error covariance matrix and P_i is error covariance matrix. This error covariance matrix can also be updated using this formula: P_i t minus 1 is minus P_i t minus 1 into Kalman gain into H_i t which is simply dow x i upon dow u k . You should note that H_i t can be easily obtained from the RBF network; so, from RBF network we compute what is dow x i upon dow u k.

(Refer Slide Time: 39:20)



This EKF algorithm can further be simplified using matrix inversion lemma. If we apply matrix inversion lemma this inverse in that we have written in EKF algorithm which is H_i into P_i into H_i transpose plus R_i inverse can be written as 1 upon lambda into I minus P_i t minus 1 H_i t H_i t transpose upon lambda plus P_i t minus 1 H_i t into H_i t, what is this lambda, R_i t actually can be written as a diagonal matrix (Refer Slide Time: 40:09). So, if I take R_i to be this then, this matrix inversion lemma can be written in this particular form. The final version of EKF algorithm for inversion is obtained by applying this matrix inversion. So, your iterative input update should be u_i t previous update plus Kalman gain into innovation term; then, the Kalman gain is updated by this formula where we have no inversion. So, this is computationally very fast and your error coherence matrix P_i can be computed using this expression which is P_i minus K_i H_i P_i. This lambda is estimated online using the following recursion, because, this lambda simply represents a measure of the difference between the actual plant state and the radial basis function network state.

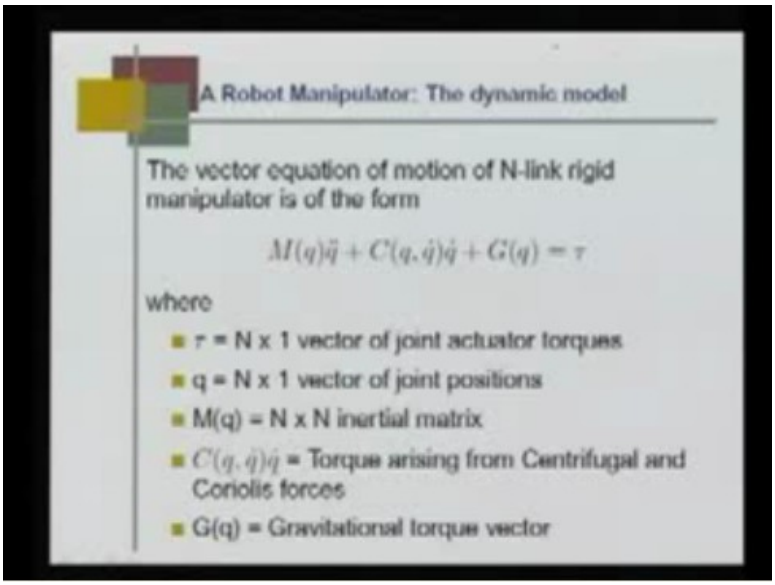
(Refer Slide Time: 41:40)



The lambda is estimated online using the following recursion which is very simple. You can easily see this V t is actually 1 upon t or this is simply a function of t some decaying function of t.

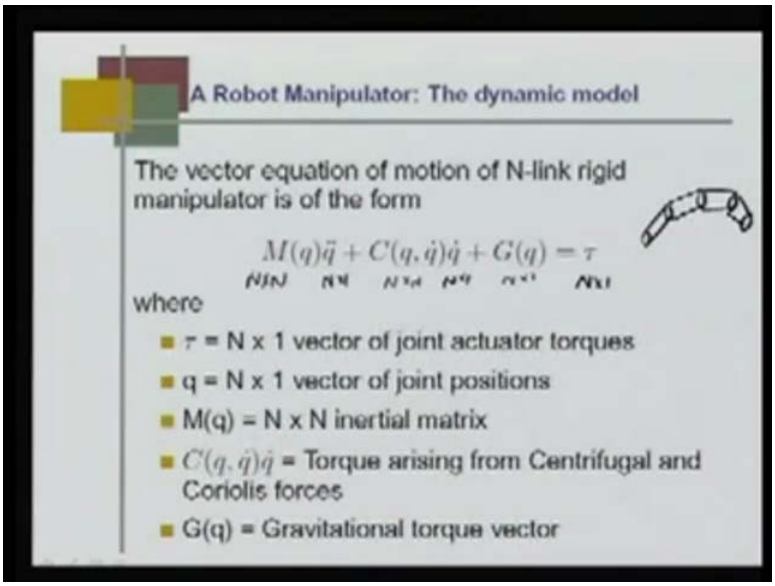
This quantity decays with a time, then lambda t is lambda t minus 1, the previous value and this is you see that the error transpose into error minus n minus lambda t minus 1. So, you can easily see that lambda is actually measure of the estimate of the error square. So, now we talked about the inversion; after this, we would like to see where we apply this inversion. Now, we will take a system.

(Refer Slide Time: 42:39)



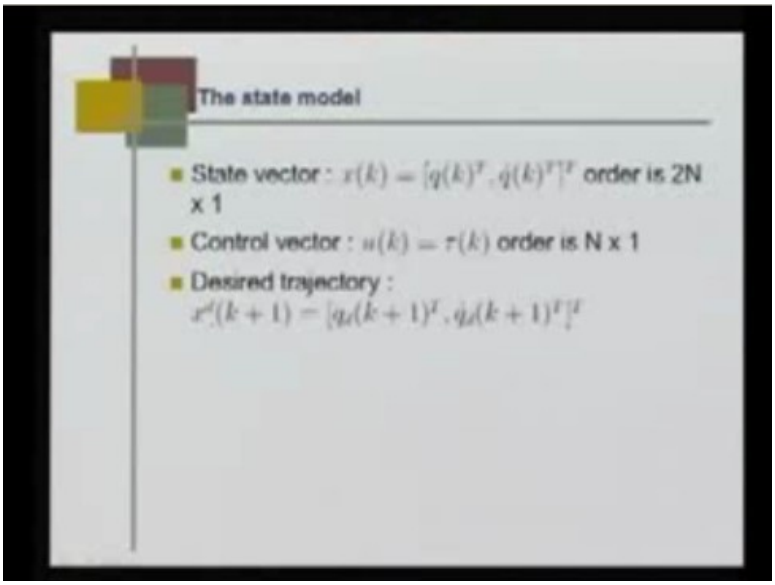
This is a robot manipulator - the dynamic model. You can easily see that, this is non-linear equation. The vector equation of motion of N link rigid manipulator is of the form (Refer Slide Time: 42:50 min). This is my inertial matrix N by N; this is my acceleration vector q double dot; this is again N by 1 and this is N by N. So, this is N by N and this is N by 1. Similarly C q q dot also can be written as N by N and q dot is N; together, this C q q dot into q dot is the torque arising from centrifugal and Coriolis forces. This is the gravity term G q which is N into 1 and this is our joint torque which is also N into 1.

(Refer Slide Time: 43:38)



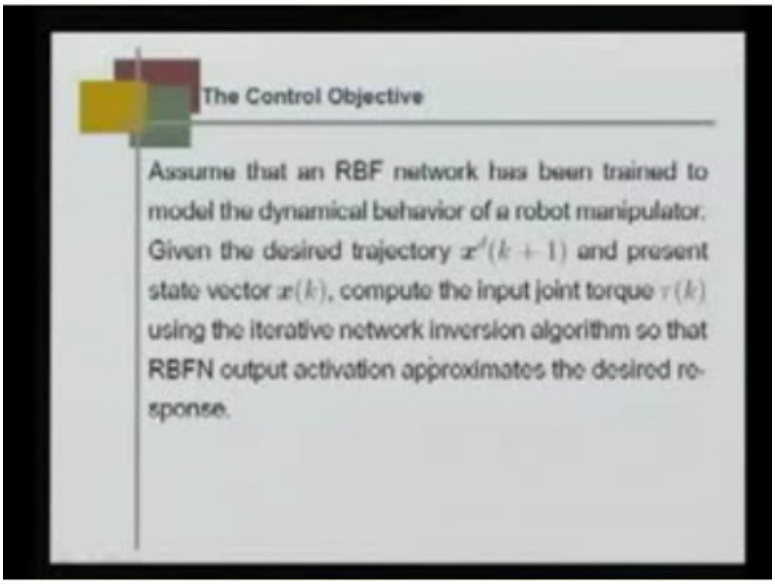
This means, we have a link here; we can connect that link, again another link and so on, you can have as many links there. So, if we have N, each link is a rigid; then the dynamics if I apply various torque at every joint, then, the angular position and angular velocity of each link will change; that is the dynamics we have taken here. So, I want to represent this robot manipulator in state space, then how many state space we have?

(Refer Slide Time: 44:24)



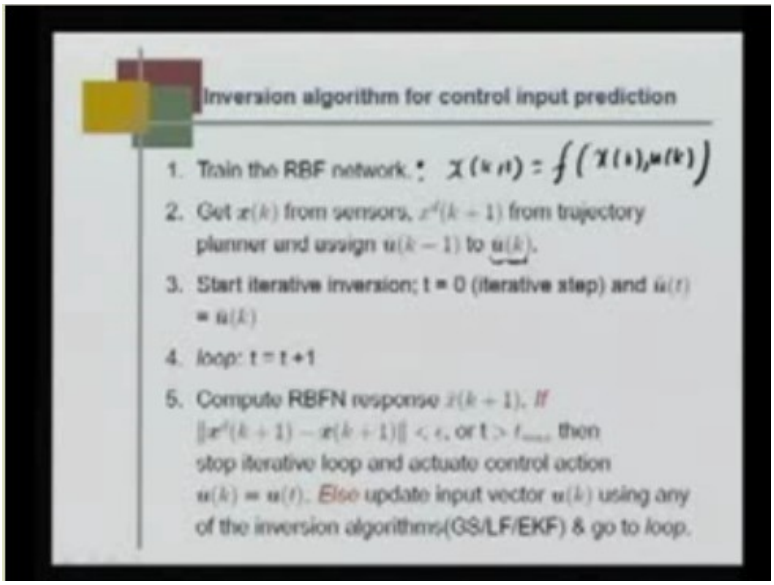
We have two N states because for N link we will have N position, angular position and angular velocity. So, total number of states is 2 N and the control vector which is u k that is tau k the joint actuation torque this is order N into 1. Normally, it is required that robot links should follow a desired trajectory and desired trajectory is the desired joint positions and angular positions and desired joint angular velocities.

(Refer Slide Time: 45:17)



The control objective is that, assume that an RBF network has been trained to model the dynamical behavior of a robot manipulator given the desired trajectory x d k plus 1 and present state vector x k. Compute the input joint tau k using the iterative network inversion algorithm, so that radial basis function network output activation approximates the desired response. This is the inversion algorithm.

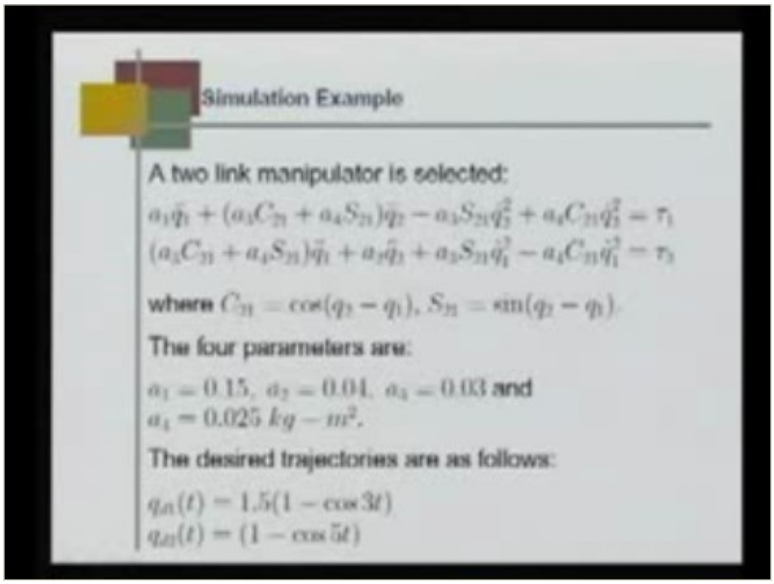
(Refer Slide Time: 45:54)



We have already talked of three categories of inversion algorithm. Now, how do we implement? First you train the RBF network; so, this RBF network models a dynamic which is x k plus 1 is function of x k and u k . The radial basis function network has been trained to model this dynamics. We take the actual data and then we model, get x k from sensors the present states from the sensor or observer. The desired joint positions to prompt trajectory planner and assign the control action that was actuated at k minus 1 sampling instant to u k because, what we are interested now that given x k, given x d k plus 1 we must find out what is u k using prediction network inversion algorithm.

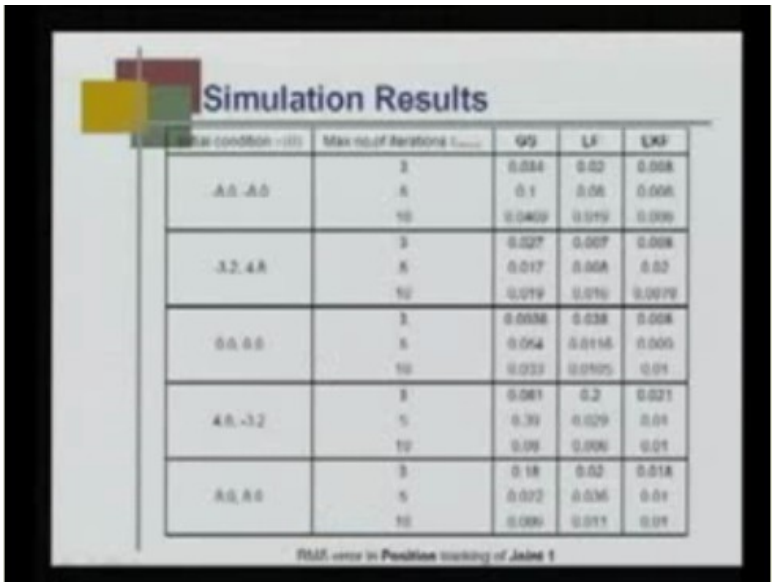
When we invert, we try to find out what is u k; we already know what was actuated at k minus 1 sampling instant, we already have that information. I use that information because, normally when we actuate any control signal they are very smooth. Naturally, the present control actuation is very near to the previous control actuation. So, start the iterative inversion t equal to 0 iterative step and I removed the k and introduced t because, during the sampling instant k, I am iteratively finding out what should be my u such that, given x k and given x d, I find out what should be the control input. This is where my loop starts - t equal to t plus 1. Now compute the radial basis function network response x hat k plus 1 giving this x k and instead of u k I give u t and this u t is in the beginning u k minus 1. Obviously, you will have some error and as long as that error is not 0, I update this input vector u k using this inversion algorithm which is either gradient search or Lyapunov function approach or Extended Kalman Filtering approach that we talked about earlier. We can use any of these things to update the u t and go to the loop. If this error is 0 or less than epsilon, then we stop it. That means, I have found out what is the control action that will take the physical plant from the present state to the new state. We will now demonstrate this inversion principle on a simple example.

(Refer Slide Time: 49:33)



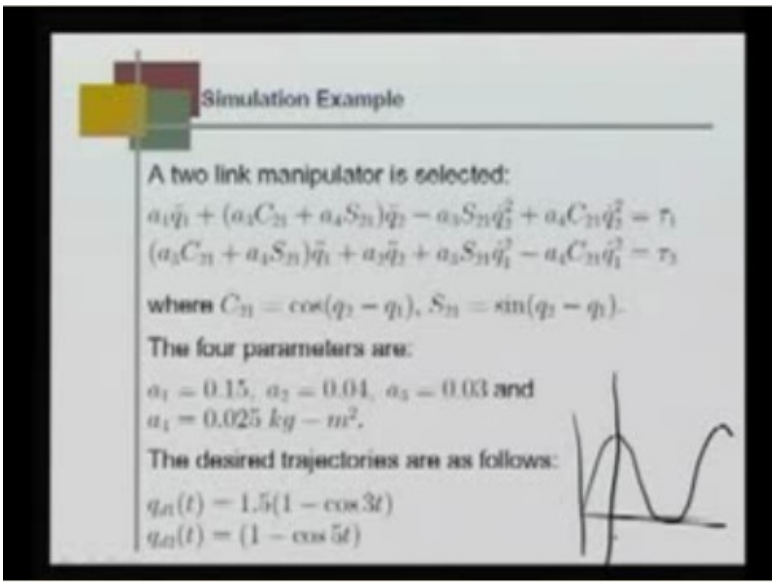
This is a simulation; we are taking a 2 link manipulator whose dynamics is given here. We can easily see, this is a very complex non-linear system, where C_{21} is cos q_2 minus q_1. So, q represents the joint position, angular position and S represents sine q_2 minus q_1 again q_2 q_1 they are joint position. The 4 parameters you can easily see a_1, a_2, a_3 and a_4 are also known; because, once we know the given robot manipulator, we can estimate what the parameters are. We are saying because, this is a 2 link manipulator, obviously each link should follow a specific trajectory. The specific trajectories are given here; the first link should follow the trajectory which is a (Refer Slide Time: 00:50:39) trajectory 1.5 into 1 minus cos 3t and the second link of the manipulator should follow a trajectory 1 minus cos 5t. You can easily see this is even more high speed than this trajectory; the bigger the term here associated with t, the velocity would increase.

(Refer Slide Time: 51:14)



I am now showing the simulation results. You can easily see, this is called R M S error in position tracking of joint 1.

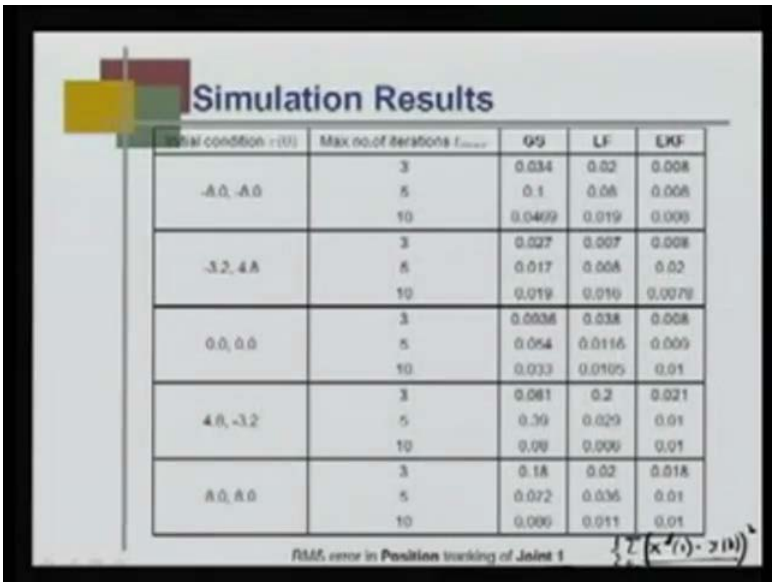
(Refer Slide Time: 51:24)



What is our joint 1? This is my joint 1 (Refer Slide Time: 51:29). If I take this particular quantity and if I plot it, it will be something like this. Of course, this is cos; so, it should come from here. I can put it here; this is my t equal to 0, so this is my q, so my joint position should follow this trajectory. What is the RMS error?

Now, I do the inversion. Using the inversion, I compute the control input and feed that control input to the plant and see how the plant trajectory is following. If plant trajectory is following some other method, I am trying to find out what is the error between the actual trajectory which is given here and the plant trajectory.

(Refer Slide Time: 52:27)



So, that is known as RMS error in position tracking of joint 1. How this is computed? This is simply computed half x d k minus actual x k whole square over k equal to… (Refer Slide Time: 00:52:52). In this example, we have taken sampling instant to be 10 millisecond and if I am controlling for 3 second obviously, 3 divided by 10 millisecond or 30 divided by 10 milliseconds will be 3000. So, over 3000 iterations, I am finding out what is this error square and then dividing by 2 divided by 3000 and I take square root of that term which is known as the root mean square error.

So, we are computing that root mean square error for various conditions. You see that I have initial condition for the torque; torque that is being actuated to the joints, these are the initial joint torque that are actuated; 8 minus 8 and whatever may be the initial joint torque the tracking should be perfect. You see that the maximum number of iterations t max we fix, because, we cannot have as much iteration during a sampling interval because, 10 milliseconds is the sampling interval. So, during the sampling interval, we

can have a few iterations before I can predict what should be my input. So, this is my initial input, I start minus 8 and minus 8 newton meter.

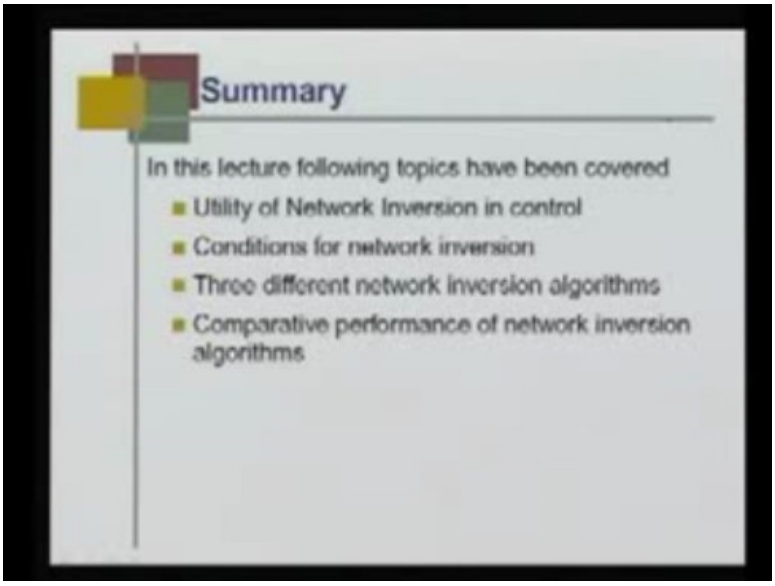
You see that, these are all arbitrary values we have taken and then we try to predict what should be the u based on that prediction; u has been predicted here in 3 iterations, here in 5 iterations and here it is 10 iterations. Correspondingly, this is gradient search Lyapunov function and Extended Kalman Filtering and you can easily see that, in this case the R M S error the lesser, better is the performance. Obviously, because x d k minus x k whole square; the lesser the R M S square the better is the performance. You can easily see the Extended Kalman Filtering is the best performance here and followed by Lyapunov function and the worst was the gradient search. Similarly, we have taken many other variations, the initial conditions were varied here (Refer Slide Time: 56:02), minus 3.2 to 4.800 4.8 to minus 3.28. This is the initial value that we initialize. We start from some u and then we track easily.

See that for all cases, the EKF- the Extended Kalman Filtering has better performance and the Lyapunov function is always better than gradient search. But in this case, you can easily see that for all cases Extended Kalman Filtering is the best. Now it is R M S error in position tracking of joint 2. In this case again, simultaneously these results are all computed for the same initial condition and same fixed number of iterations during the inversion 3 5 10. Computation you can easily see again in joint 2; you can easily see that the error in comparison to joint 1 is more. Why? Because, the desired trajectory for joint 2 is a very high speed trajectory compared to the trajectory in joint 1. That is why the error is more here when compared to the first table. But, here, you can easily see again (Refer Slide Time: 57:52) EKF is much better than the performance than Lyapunov function and Lyapunov function has better performance than the gradient search. It seems here in the first case Lyapunov function is not doing well but in all other cases it is doing well.

It all depends on where is my initial condition; but, whatever may be the initial condition the EKF is always far superior. It is independent of initial condition, so this is very important. The gradient search and Lyapunov function they appear to depend on the initial condition; this is again the RMS error in velocity tracking of the joint 1. Again you

can easily see the EKF for Extended Kalman Filtering. The RMS error is the lowest when compared to gradient search and Lyapunov function. Similar results are also found out from for RMS error in velocity tracking of the joint 2; so, this was joint 1 and this is joint 2 and again the same initial condition and the same constraint imposition on the maximum number of iteration that we can do for predicting the control input. The control input is predicted using these many number of iterations and for every case, the error is tabulated and EKF again has the best performance. Finally, in this lecture, we discussed how the network inversion can be used for control purpose.

(Refer Slide Time: 59:48)



We talked about conditions for network inversion where I said that, the number of inputs that has to be predicted has to be less than number of outputs states of a network. We discussed three different network inversion algorithms, they are: Gradient search, Lyapunov function approach as well as Extended Kalman Filtering. Finally, performing simulation - we compared the three network inversion algorithms and showed that, Extended Kalman Filtering is performing better than the other two which are gradient search as well as Lyapunov function.

Thank you very much.