Nonlinear Adaptive Control
Professor Srikant Sukumar
Systems and Control
Indian Institute of Technology, Bombay
Week 12
Lecture No: 71
Real Time Neural Network Based Control of a Robotic Manipulator (Part 5)

(Refer Slide Time: 00:17)

Hello everyone. Welcome to our final session of Nonlinear and Adaptive Control. I am Srikant Sukumar from Systems and Control, IIT Bombay. I am very very thankful to all those who have attended this course. And I really hope I have been able to generate enough interest in the topic of nonlinear adaptive control, it is a rather rich area. And it is of course a rather powerful set of tools in nonlinear control, which is what makes it one of the most applied areas in nonlinear control, for fighter jets like what you see in the background here and also for spacecrafts. These have even been tested in actual flight conditions.

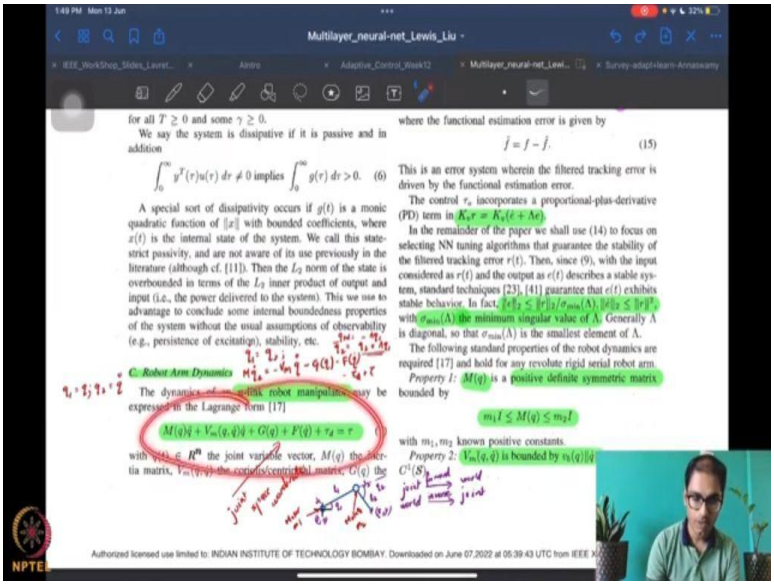
So, and of course there are also companies that specialize in adaptive control for drones. So, a lot of applications again I focus mostly on the aerospace applications or aero-mechanical applications, because my own background. But, there is quite a bit of work out there on electrical, biological, chemical systems also using adaptive control.

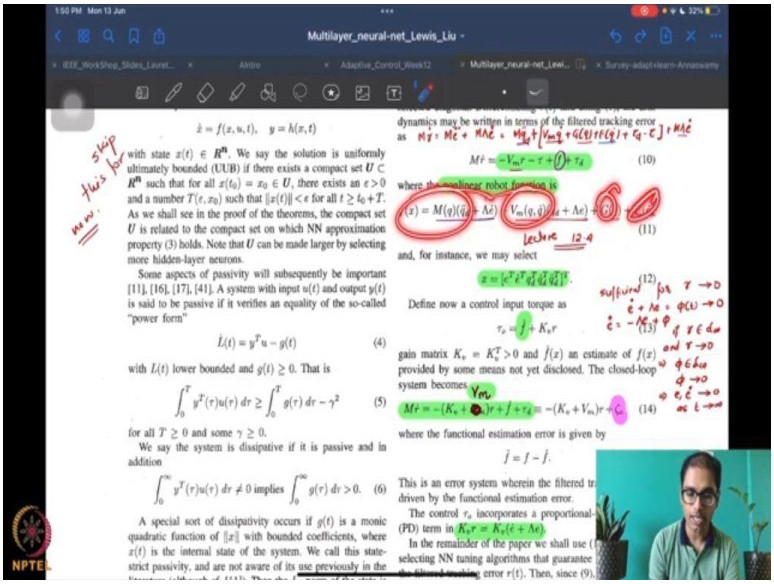
So, what we have been looking at in this last week is a little bit of what you see in the bottom right of our motivating image in the background, which is on neural networks. So, as you can see

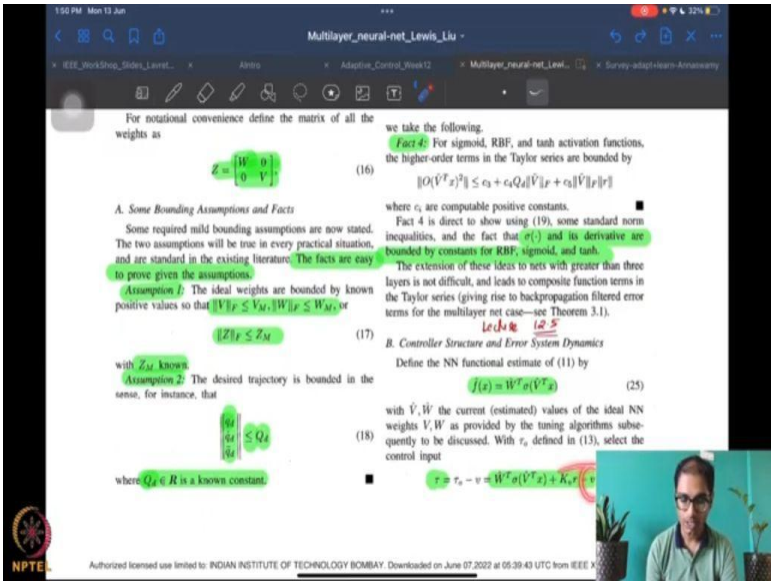
here in the image, this is like a deep learning network, so essentially a multi layered neural network. So, it is typical practice in neural networks to do to have several layers now, because again, we have a lot more computational power and lot more data available to train these networks.

So, the typical approach is to have an offline training, where you input a lot of training data on the left of this network, and you get the output on the right of the network. And use this input-output in order to tune the weights that appear in this sort of nonlinear function approximator. And then, you use these weights in order to do actual experiments. And when you go to the actual run, you design yours controllers based on these weights. So, this is what we have been looking at.

(Refer Slide Time: 02:48)





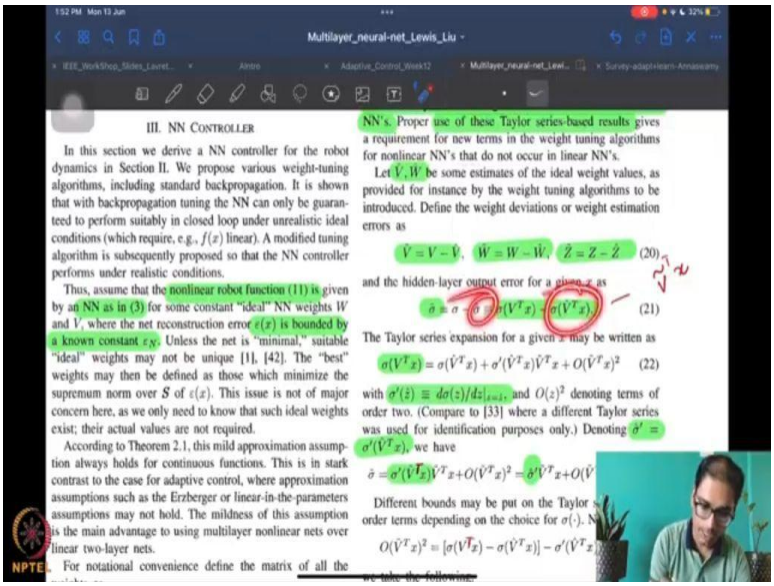


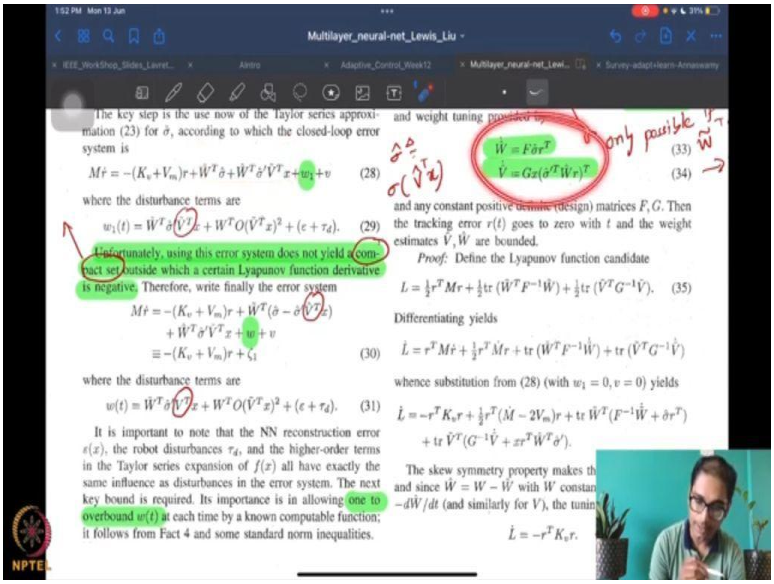
We of course, are looking at rather special situation where there are a couple of things, the weight or the learning itself of the weights is happening online along with the control. And secondly, in order to make sure that this learning plus control mix is stable. We use adaptive control principles. So, I hope you have seen by now that the adaptive control is essentially very very closely connected to deep learning. And we also saw that deep learning is now prevalent also in reinforcement learning and other classification approaches. So, over and all you understand, I hope you understand that these adaptive control, stable adaptive control methods are of significant use in learning.

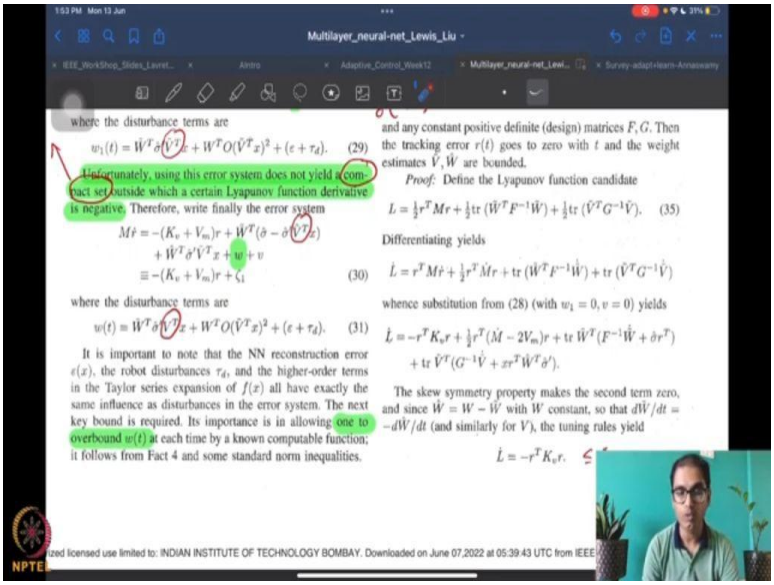
So, we are looking at a very specific case of this three-layer neural network being applied as a function approximator to a multi-link robot. And because it is like a neural network approximator, we do not really care to write out these mass inertia functions, and the Coriolis functions by hand for every single robot. If you bring in any robot, this is like a plug and play controller. So, once you you could, if you have a control module, you can plug it into any multi-link robot, which has this kind of Euler-Langrange system structure, like you see here in equation-7.

And and, you can start controlling it, because your neural network using this adaptive law will automatically learn all these nonlinear robot function. So, this is, this is what is the cool aspect, that it is agnostic to what kind of M, V, G, F you have, this function, the neural network function approximator is going to learn it. And how the learning happens via the adaptive update laws is what we saw last time, so, we sort of added a robustification term.

(Refer Slide Time: 05:09)







And of course, we sort of wrote the dynamics in a rather nice way, either in the in form 28 or in the form of equation-30. So, it was either in the form of equation 28 with the disturbance terms w1, or in the form of equation 30 with disturbance terms labeled as W. So, the first result then that we saw was on backpropagation, which was rather ideal situation, where we assume that these disturbances are 0. We saw that it is not very easy to sort of claim w1 is 0, it is obvious if you assume that the second order terms are 0. So, your function is a linear function. And we do not have these, it is an exact estimator and the disturbance is 0.

So, these terms will become 0 rather easily, but we realize that this term will become 0, only if you know you have this kind of W tilde, V tilde type of a thing. So, this is a sort of linear in

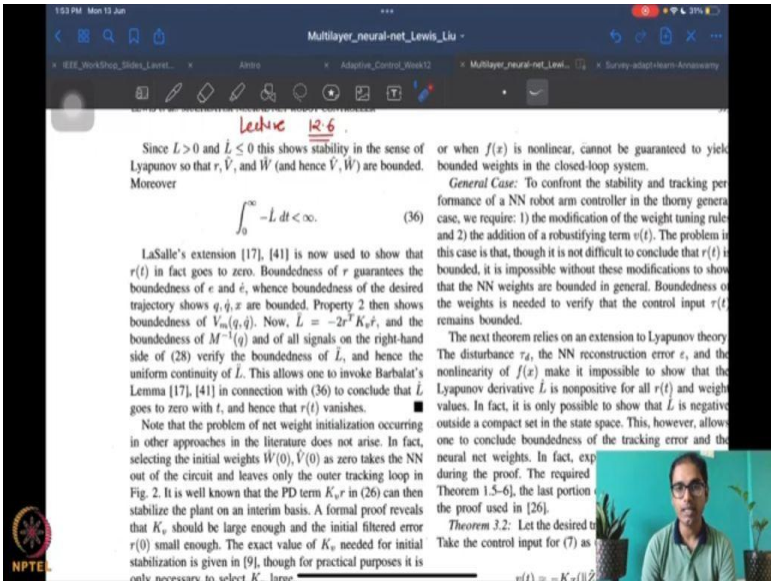
parameter type of assumption of the function is, if you want these to be 0. And if this does happen, we did have this backpropagation-based tuning of weights.

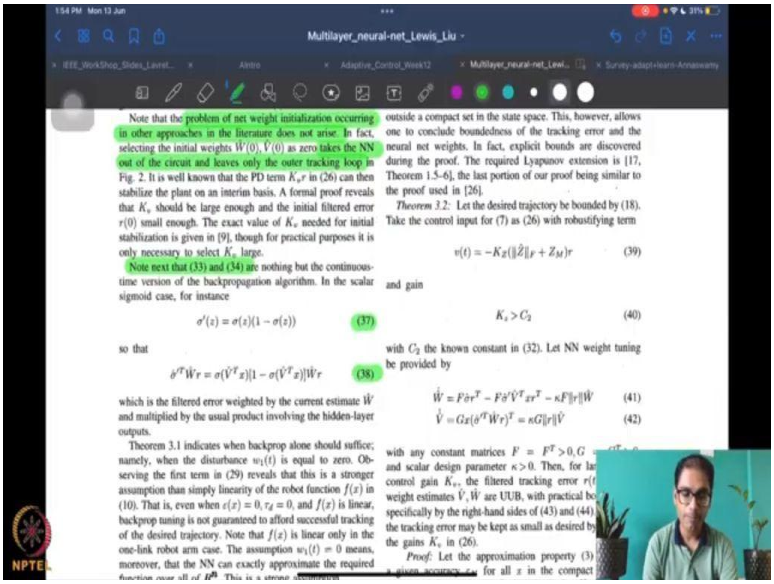
So, this sort of maps, why it is given this name is because this sort of maps to the sort of weight tuning algorithms that standard nonlinear neural networks use offline. So, this is the update law, I mean, and if you notice that, you have to remember that this sigma hat was essentially, sigma hat is just defined as sigma of V hat transpose x.

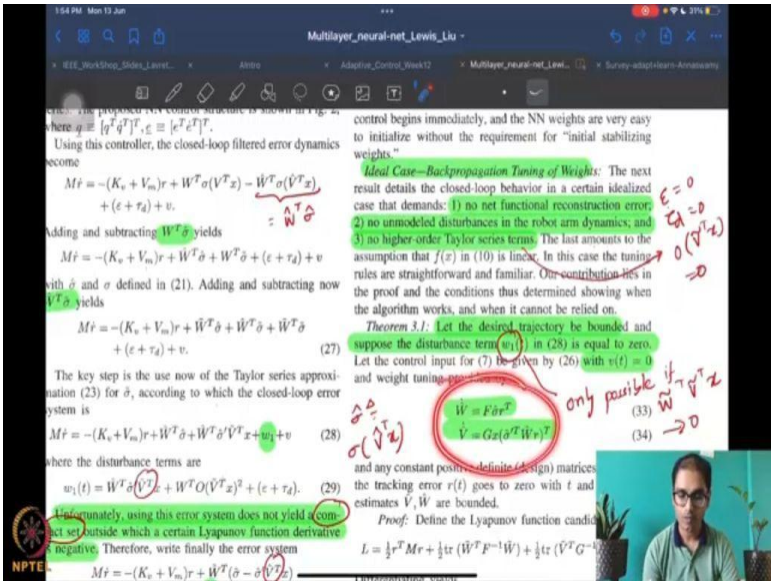
So, so remember, this is just notation, so, sigma hat is just V hat transpose x. So, this was a notation that was used. And so that notation, so you can see that W hat dot depends on V hat, and V hat dot also depends on W hat. So, they are sort of inter dependent here. And we also saw how the Lyapunov analysis goes because of this ideal circumstance, where you do not have this w1 and v. We just make them 0 here, and you look at this equation.

So, all you have is, so this term goes to 0 because of the skew symmetry, then this term contributes to a nice negative term. And these two terms are of course cancelled using the update law. And then you have of course, this L function, this Lyapunov candidate function L, its derivative is negative semi-definite. And then you can use your LaSalle invariance or Barbalat's Lemma signal chasing to prove r goes to 0.

(Refer Slide Time: 07:52)







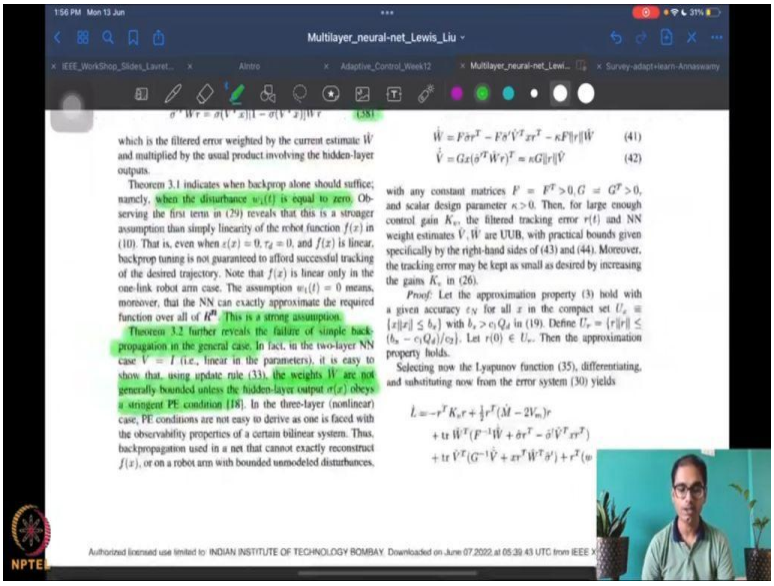
So, so this is sort of where we were going, I am to mark our final lecture here, starting here. Like I said, you can use Barbalat's Lemma signal chasing or you can use the LaSalle invariance to claim that r goes to 0. And once you have r going to 0, you know that you also have error, and the derivative of the error tracking error going to 0, and that is essentially what it is. So, few important points. The first in the introduction itself the author's mentioned that most of these neural network-based laws require very specific or very good weight initialization, so, that problem is does not arise at all.

In fact, if you see if you take the initial conditions on the weights W hat, V hat to be 0. You see that the right-hand side of this guy and this guy becomes 0, so it is an equilibrium. So, this entire

update goes away, so you will stay at W hat and V hat to be 0 for all time. So basically, it takes the neural network out of the circuit and has the only has only the outer tracking.

So, so of course, basically, if you I mean you can do some kind of a local stabilization using this Kv term. So, if you have a large enough Kv, and if the filtered error r is already small enough, then you are fine, so, so anyway. So, so of course it is not possible to choose its large Kv for several practical reasons. So, this is so this is the connection to the backpropagation algorithm. So, this is equation-33 and 34 are matched with 37 and 38, which is what is known in the neural network theory as the back-propagation results.

(Refer Slide Time: 09:49)

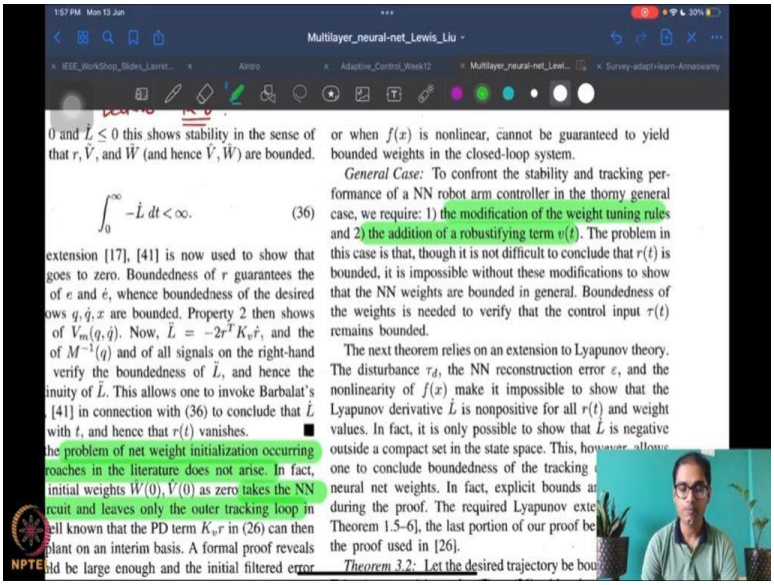


So, theorem is when it reveals that when, when only this back propagation alone will suffice. And this is when this term is exactly equal to 0. And and this is what I was saying that this first term in 29 reveals that it is a rather strong assumption than just some linearity of fx. So, even if epsilon is 0, tau d is 0, f is linear, this is not guaranteed. So, and of course, fx is of course, only linear in the one link arm case. So, there is a lot of assumptions, a lot of assumptions. So, this is a I mean, w1 being equal to 0 is a very strong assumption. So, so of course, then there is additional results that are in this article also.

So, that is what is there in theorem 3.2, it reveals the failure of simple back propagation in the general case. Basically, it can be shown the author's themselves, I am not sure if they are showing it themselves, so we will not worry about that right now. But, the point is it can be

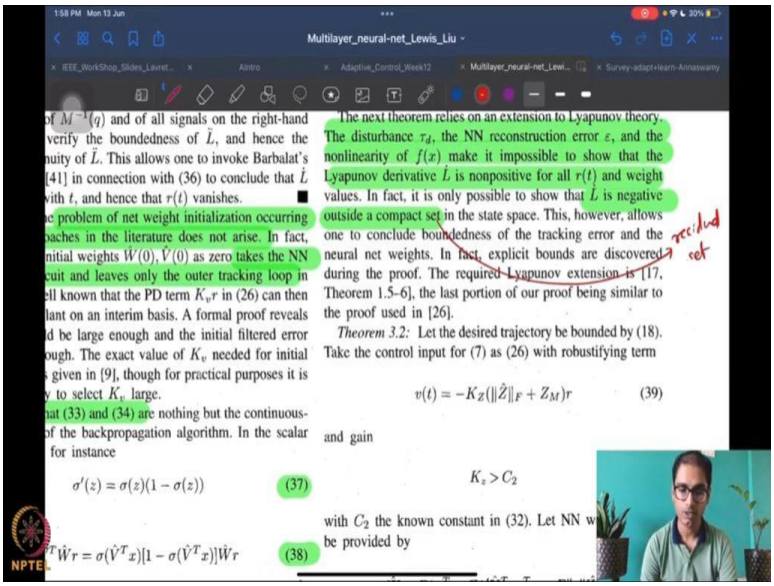
shown that the weights become unbounded. I mean, all of this is because of the assumption that w1 is equal to 0, w1 is not really 0, even for the linear fx case. And even the linear case happens, only if fx is only the robot the single link rotate. So, this is what the authors say are generally not bounded, unless the hidden layer obeys a stringent persistence condition.

(Refer Slide Time: 11:26)



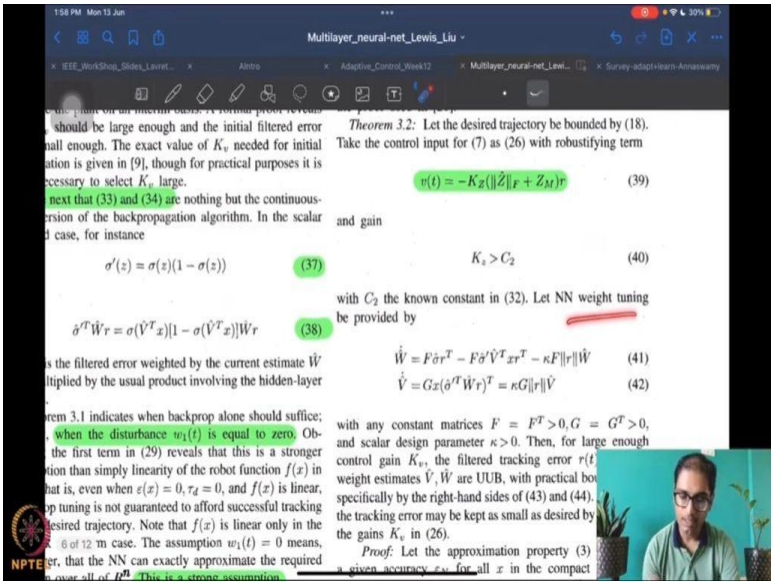
Now, therefore, there is a need to look at the more general case. So, so of course, there is need to modify the tuning weight, there is the need to add a robustifying term, so, both of these are there. So, we already added v, because in the previous result, we assumed that v was exactly 0. And we gave a set of weight tuning rules. Now of course, the authors will give a slightly different weight tuning. So, so this is what is going to be detailed in the second theorem.

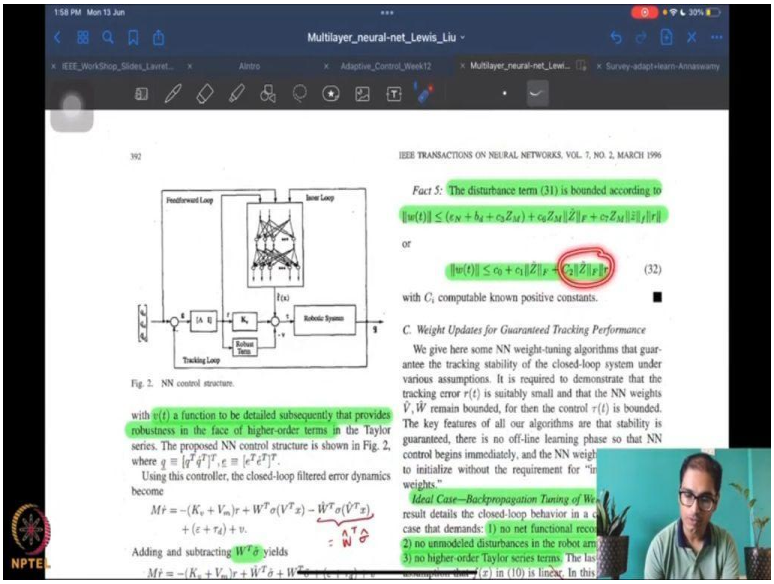
(Refer Slide Time: 12:09)



So, of course, this I mean the disturbance tau d and the reconstruction error, make it impossible to show that L is L dot is non-positive for all r. So, therefore, you can only show that it is L dot is negative outside a compact set, so this is what is as usual our residual set, type result, you are used to this. Because, if there is disturbance, there is an epsilon error in the function approximator, then, you are used to the, I mean you have to have some kind of residual set performance zone, you cannot have exact conversion. So, one should not even expect any exact convergence. So, this is the first important thing to remember.

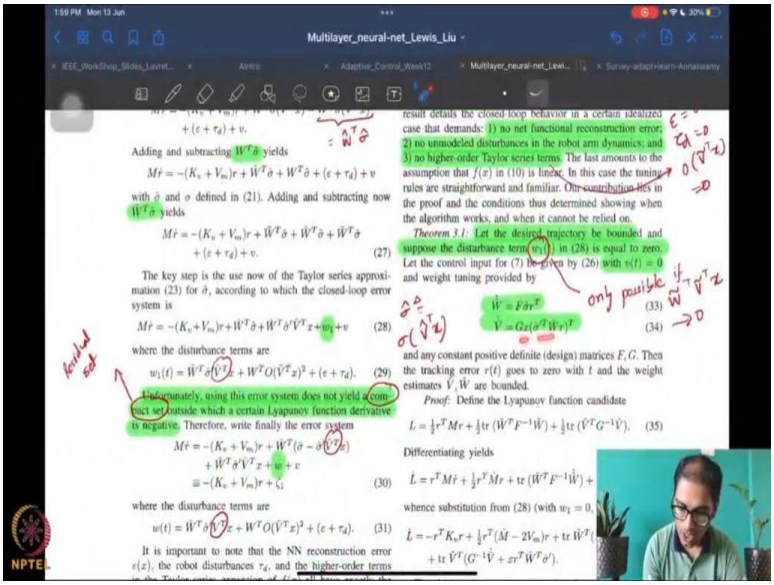
(Refer Slide Time: 12:53)

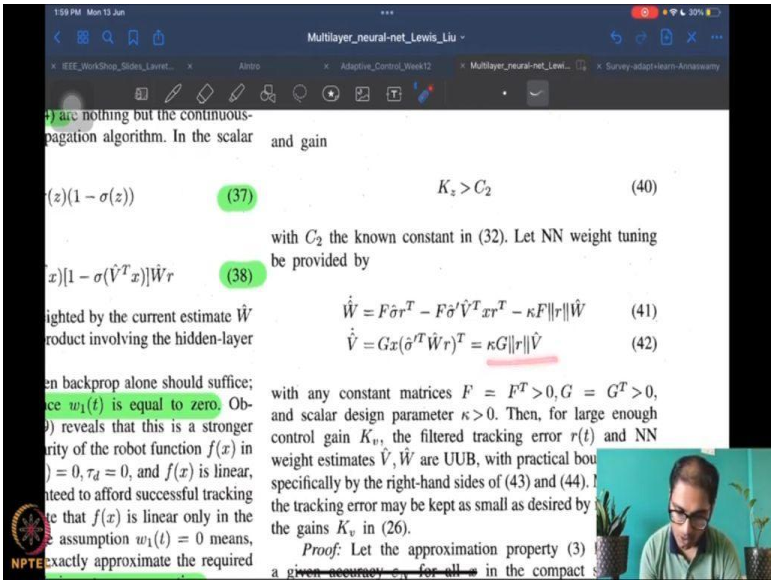


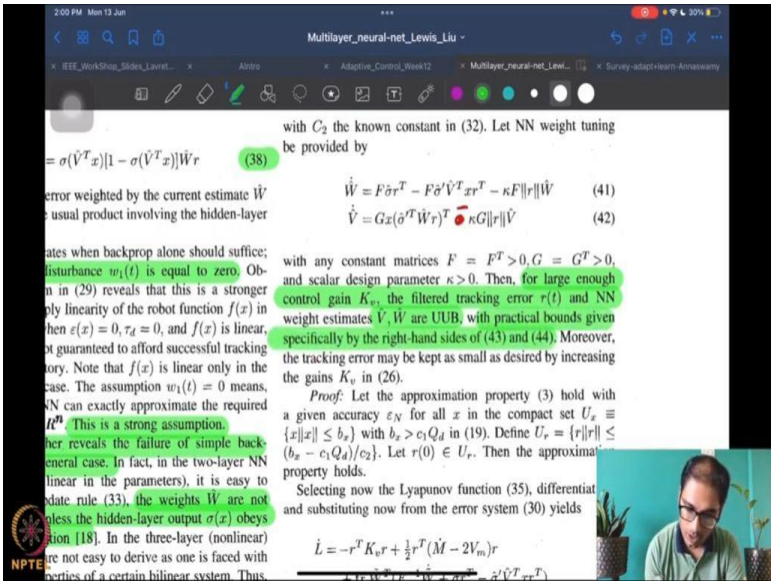


So see, so, this is the sort of robustifying term that the authors introduce. So, this is essentially the robustification term, it contains of the I mean honestly not a smooth term. It is a non-smooth term if you may, and this term is like a norm of Z hat, Frobenius norm of Z hat. And then of course, you have some state dependent terms. And then end with some gain which is larger than some constant C2. And this constant C2 is known in equation 32, it is coming from the bound-on W. So, the idea is sort of to take care of this, sort of terms, so we will see how that happens.

(Refer Slide Time: 13:38)







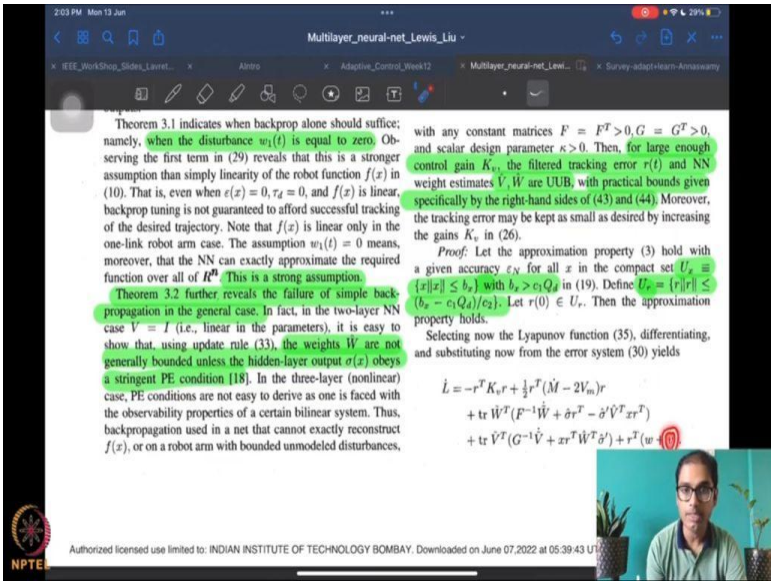
And then the neural network weight tuning is changed a little bit. If you see the second term is not very different from what we had. So, you had Gx sigma hat prime transpose, W hat r transpose. And this is exactly the same, Gx sigma hat prime transpose W hat r whole transpose, this is exactly the same. But, the first term was F sigma hat r transpose in the back-propagation term case, and here it is modified to include two more terms. Again, there is a non-smooth term here, but, it is a nice continuous term, but a non-smooth term, because it is a norm r type of a term.

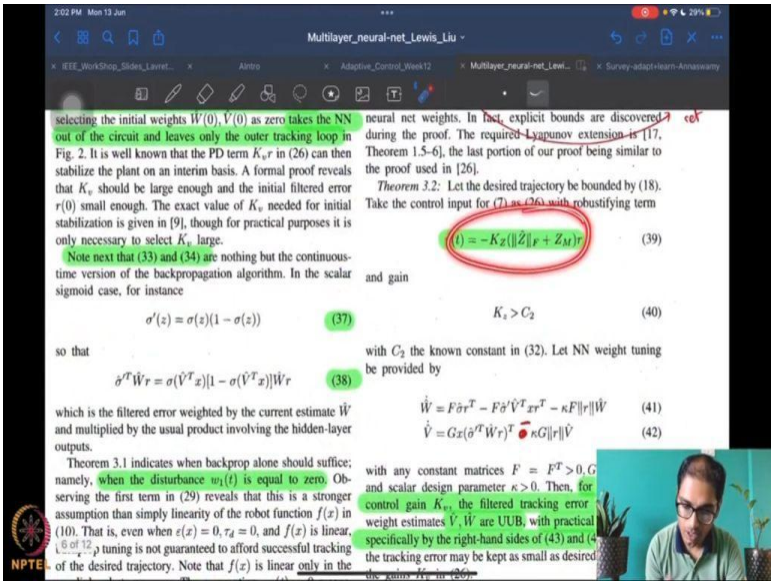
And this term is also written in terms of norm r type of term, I think this is not equal to, this is what I need to. Let us see, let us just a second please, just give me a moment, please. I think that

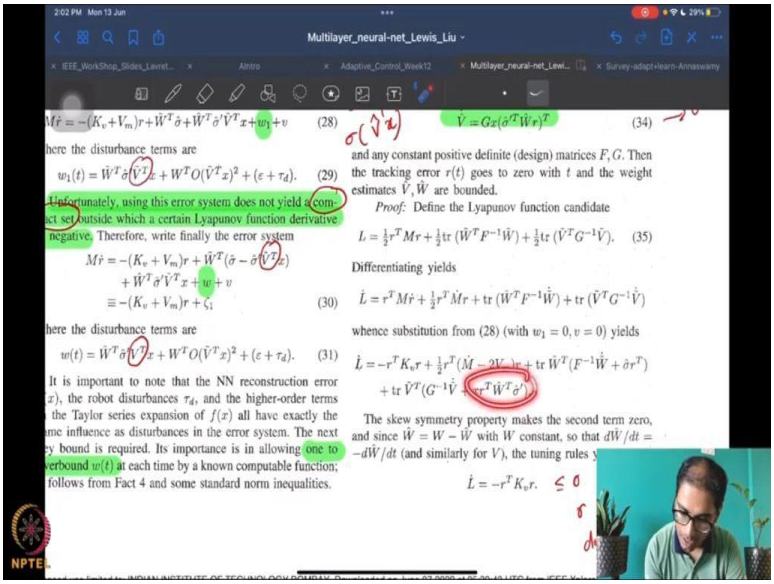
term is not quite right. I believe this is not equal to this is actually a minus term, and this is not an equal to, but this is a minus term. That is a typo error in the paper, so that is fine.

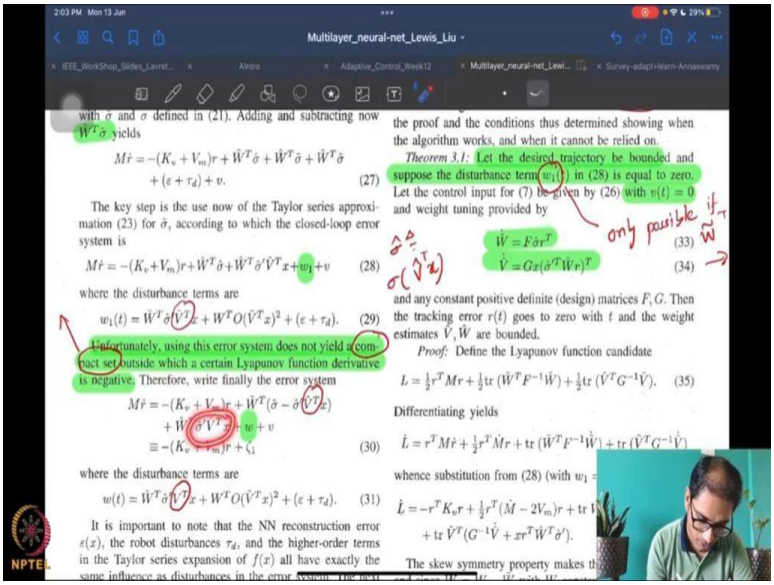
So, this also the V hat dot is also changed, and the W hat dot also changes. So, you have the ideal version and then you have these new terms. And now, how you and essentially what you can prove is that for large enough Kv and the filter tracking error r, these become uniform ultimately bounded with practical bounds given specifically by 43 44. So, this is what you sort of obtain, this is what you sort of obtain.

(Refer Slide Time: 15:49)









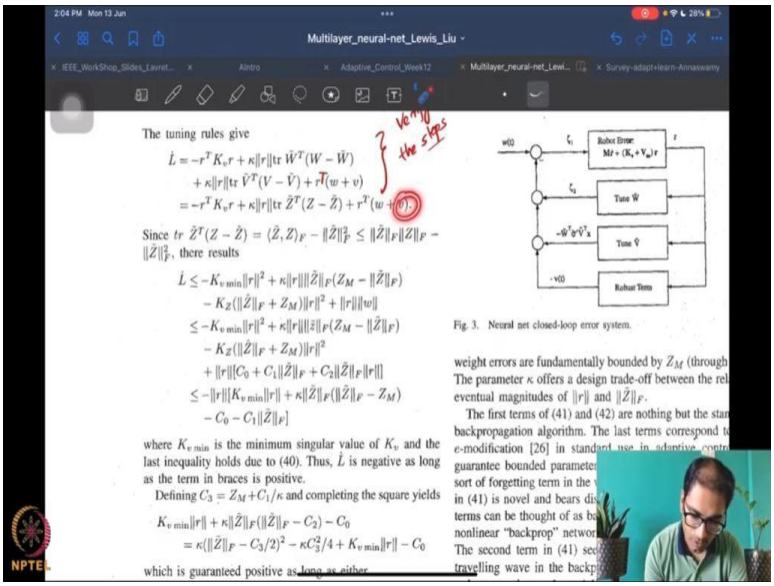
So, so you of course, assume these nice approximation properties hold, which is this for the function approximator with some epsilon N, in and we assume that this also holds not everywhere, but in this nice set. This is nice set. And again you have Ur, which is a set, which is correspondingly defined for the r variable. You typically assume that these bounds hold within a certain set. So, then of course this approximation property holds here, for this set r in Ur, 0 in Ur. So, now as usual you do the Lyapunov derivative, so this is the nice term, then you have this skew symmetry term which is going to go away.

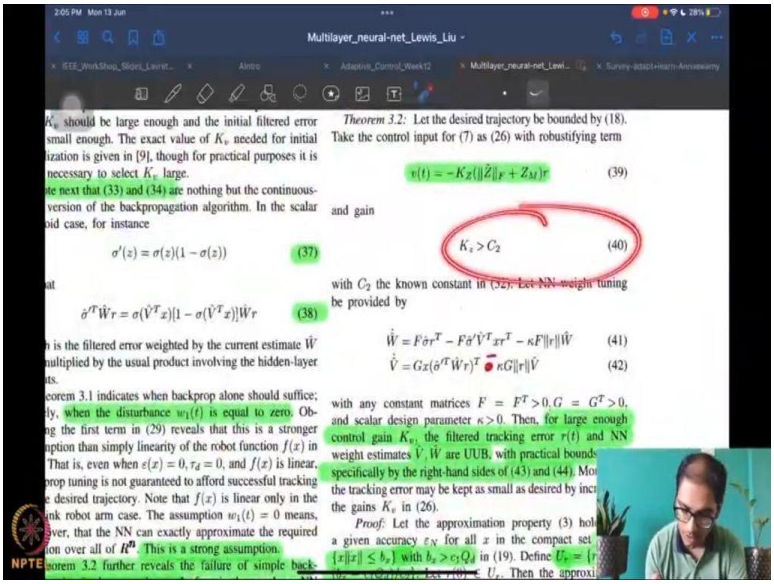
Then, this first term is coming just from the derivative of our same L. The L is not changed actually, the L is still the same, so this, so this is the term. So, the first term here, so this term is

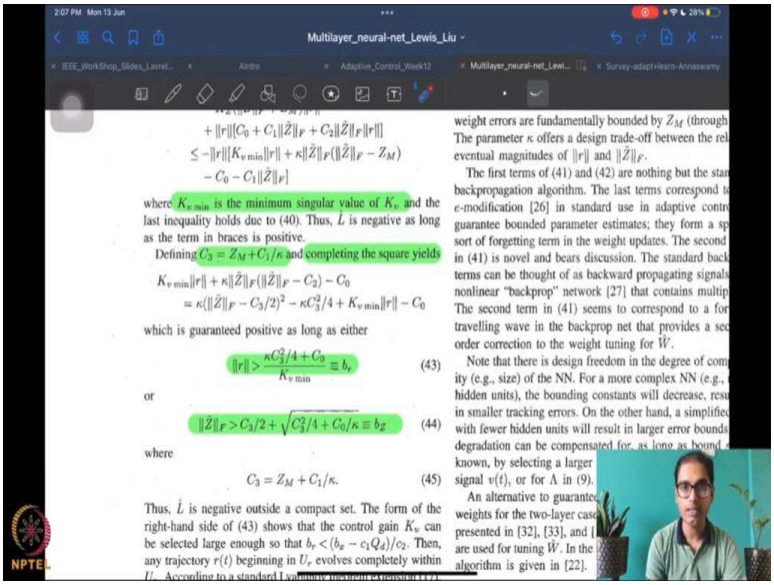
coming from this guy, and then this is coming from your r dot. And so, this these terms are coming from r dot. So, in fact, these, this term and this term is not different from what you had in the previous case, except for of course, the fact that you have introduced a V. And, so that sort of also has to show up here, those terms also show up corresponding to the v here, because you have this kind of a term.

So, and then then you have this term which was again same as before. So, so the additional terms if you see are, you had a sigma hat r transpose and this term already, so the additional terms here are, so, these two terms were already there. So, the additional term is sort of this guy. Let us see this an additional term, but this two was already there. Yes, this is an additional term because we are using equation number-30, so we are using this guy. So, the, so this guy results in this term, and the V term is coming from this, so this guy accounts for the first term here. So, everything else is the same, in fact, my my my mistake, the V is not even stated until now.

(Refer Slide Time: 18:37)







So, once you do that, the authors apply the tuning rule, and they do a bunch of simplifications. I am not going into the complete details of this. I would ask you to verify the steps here, in order to get to this, I will ask you to verify the steps. So, these nice negative terms still remains. And then you have this kind of W minus W tilde, V minus V tilde, and r transpose w plus v type of term. So, V is still not substituted notice. And then of course, there is a you do this kind of nice trace equalities, and then you bound these trace equalities. And once you do that, you start working with the norms everywhere, the vector norms and the matrix norms here.

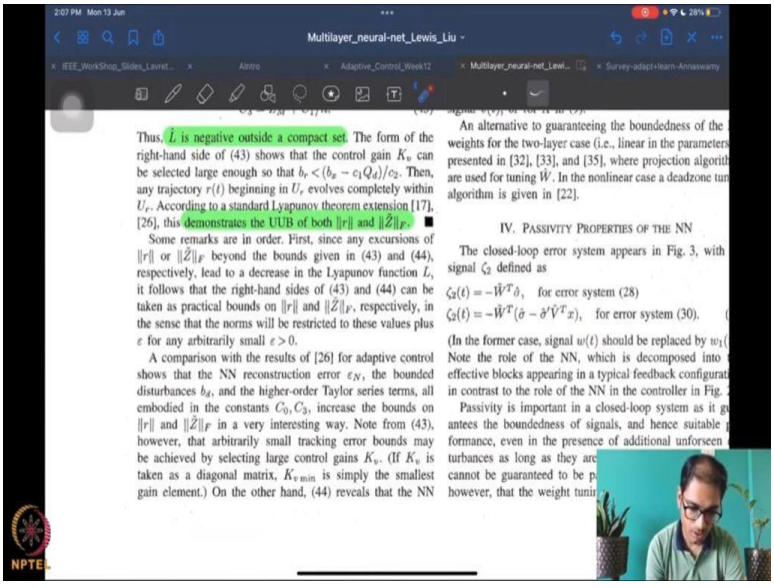
And this is where of course you introduce the expression for V also, and that expression for V gets substituted here. So again, I am not going into the details, I will ask you, I will request you

to sort of verify the details. These are simply just using standard norm inequalities. And then this this particular equality and inequality right here, and after substitution of v, you get this kind of final expression, this kind of a final expression. Here Kv min is of course, minimum singular value of Kv, and the last inequality holds due to 40. So, so basically 40 is this guy. So, a lot of lot of the earlier inequalities that were mentioned have been used here, in order to arrive at these bounds, at this nice bound here.

So now, if you do a nice completion of squares after defining this kind of a this kind of a new variable C3. So, what is the completion of square? The completion of square is being done here on this term, so this term is this guy, and here is what you have here. And and you you sort of nicely break it into these pieces.

And, of course, this is where you sort of, you sort of do a standard completion of squares in order to get your residual set bound. And that is essentially what is being done in these steps, that is essentially what is being done in these sets. So, you get two different sets. So, one set is obtained from here, this guy, that is here C3. And this, and then you have another set on r. So, once it is on the Z tilde, which is a parameter error, and other set is on the r, which is the state errors.

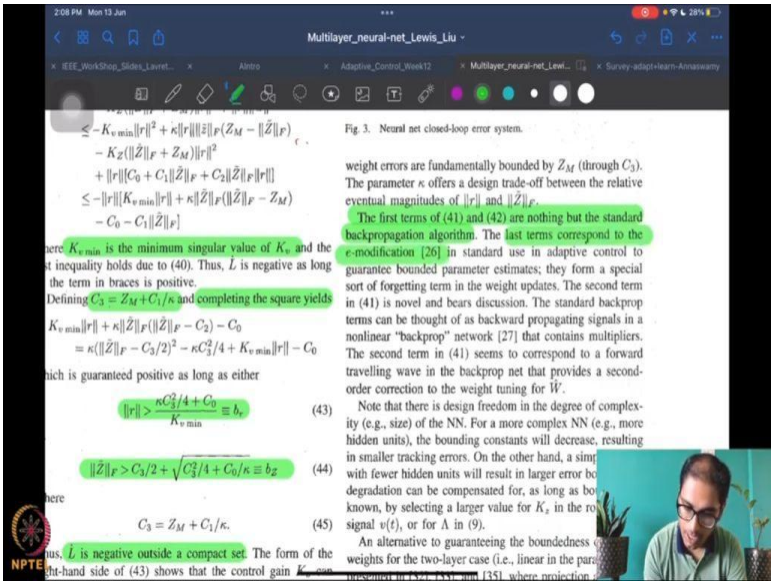
(Refer Slide Time: 21:58)

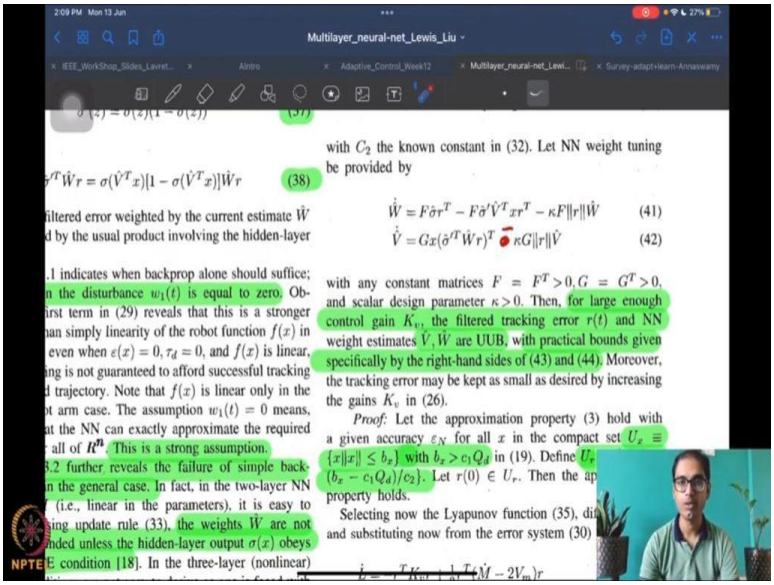


And it is what you get is that L dot will be negative outside this compact set, and which is defined by these two inequalities 43 and 44. And therefore, you have standard uniform ultimate boundedness for both r and Z tilde F. And because Z tilde contains both V tilde and W tilde, you

have what you need. So, one thing is obvious that any excursion beyond these 43 44 leads to decrease in Lyapunov function, which means that you will get them back into the bounds. So, Z and Z tilde and r of course, restricted within this residual set. So, so anyway, I mean these bounds are dictated by number of higher order terms in the Taylor series, and magnitudes of the Taylor series bounds et-cetera.

(Refer Slide Time: 22:51)





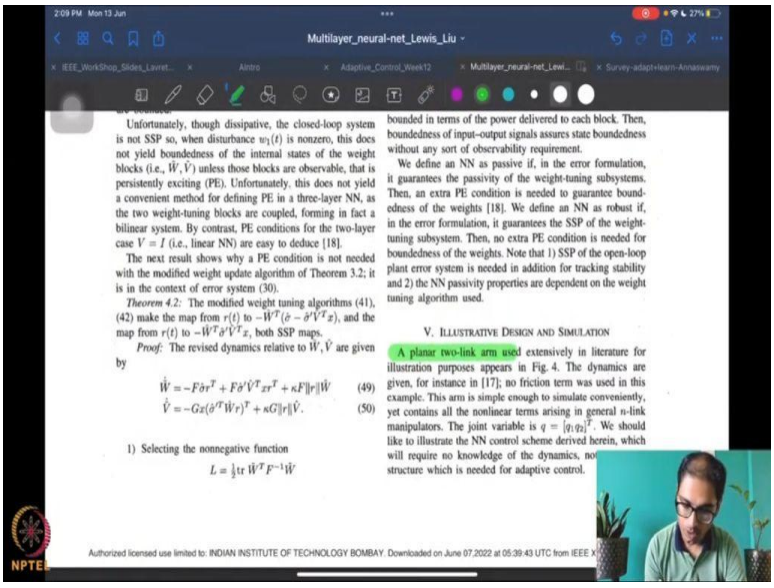
And of course, as you know that the first terms are nothing but the backpropagation algorithm itself. So, this is what is the standard backpropagation, and the last terms look like the epsilon modification. So, if you look at these terms, these are sort of like, so, these are like the standard

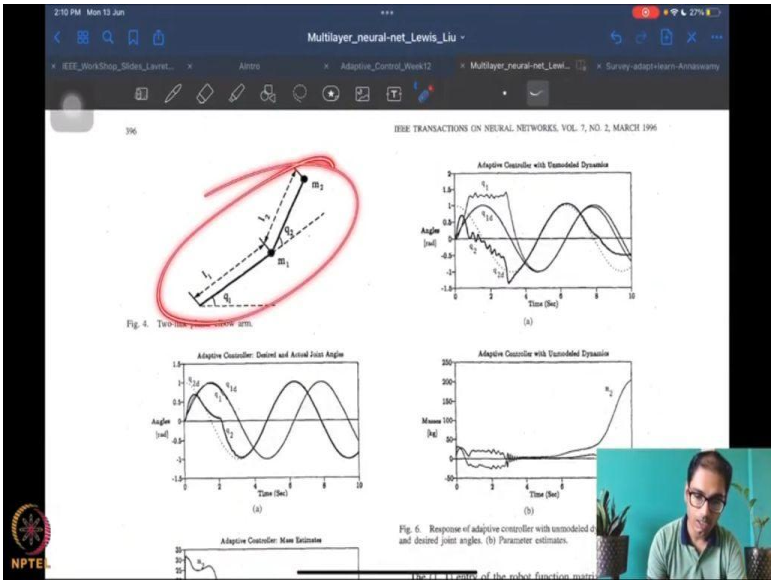
backpropagation. And these terms look like your epsilon modification, because this contains a V hat term and this is basically a W hat term. So, this is like an epsilon modification. And this why it is an epsilon modification instead of a sigma modification? Because the additional terms are scaled by the state remember.

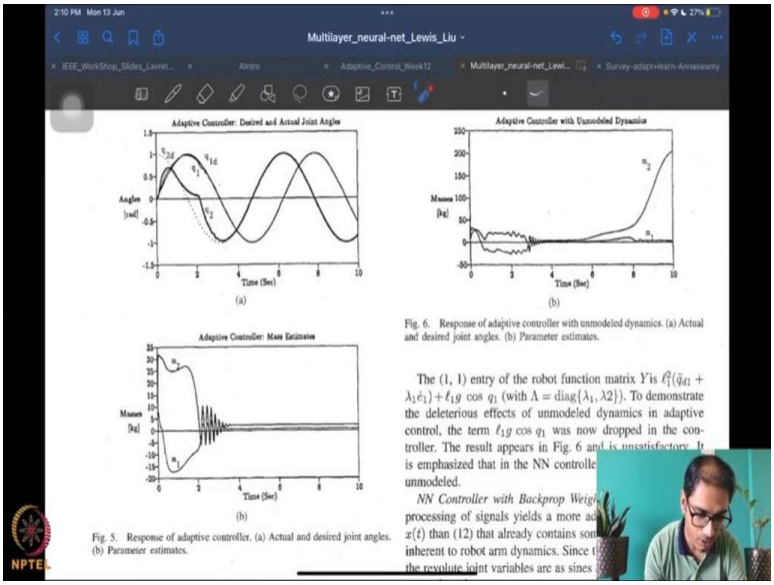
In the sigma modification, the W hat and V hat term would be scaled by a constant, here they are not scaled by a constant, they are scaled by the state value itself. So, why it is better is because, if r becomes close to 0, you do not stop learning the parameter. Because, in the constant gain case, if r goes close to 0, these terms are dead.

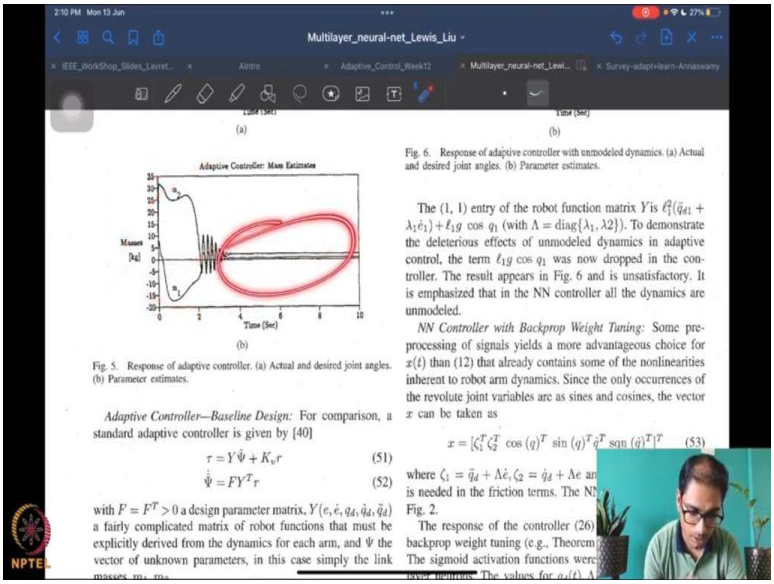
These terms, if r goes close to 0, these terms become 0. But, this term continues to dominate and push W hat and V hat will continue to dominate and push W hat and V hat towards 0. So, whatever parameter value that was learned is lost. So, this is like an epsilon modification, because it is also scaled by norm r. So, that is the whole idea. So, this is like an epsilon modification. So, as I mentioned, we will not look into the passivity properties of the neural network right now.

(Refer Slide Time: 24:27)




2:10 PM Mon 13 Jun
Multilayer_neural-net_Lewis_Liu
396
IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 7, NO. 2, MARCH 1996
Adaptive Controller with Unmodeled Dynamics
Angles [rad]
Time (Sec)
(a)
Adaptive Controller: Desired and Actual Joint Angles
Adaptive Controller with Unmodeled Dynamics
Masses [kg]
(b)
Fig. 6. Response of adaptive controller with unmodeled d
and desired joint angles. (b) Parameter estimates.
Adaptive Controller: Mass Estimates

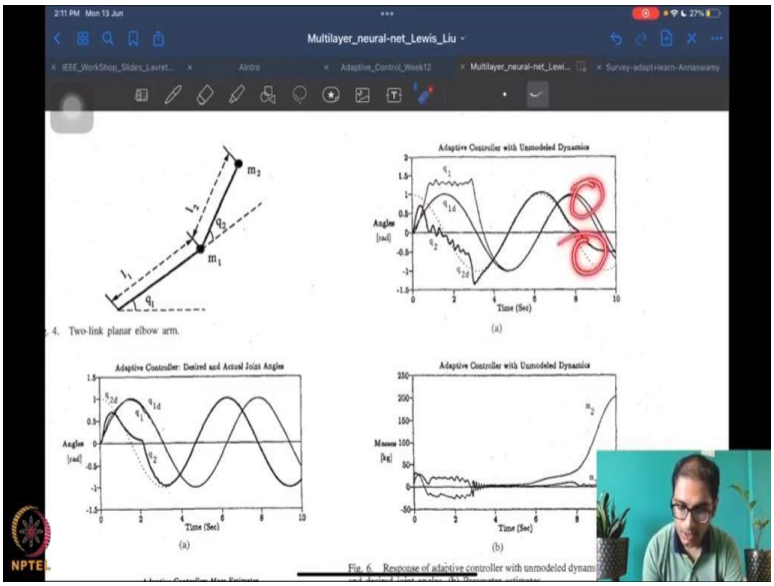

2:10 PM Mon 13 Jun
Multilayer_neural-net_Lewis_Liu
Adaptive Controller: Desired and Actual Joint Angles
Angles [rad]
Time (Sec)
(a)
Adaptive Controller with Unmodeled Dynamics
Masses [kg]
(b)
Fig. 6. Response of adaptive controller with unmodeled dynamics. (a) Actual and desired joint angles. (b) Parameter estimates.
Adaptive Controller: Mass Estimates
Masses [kg]
Time (Sec)
(b)
Fig. 5. Response of adaptive controller. (a) Actual and desired joint angles. (b) Parameter estimates.
The (1, 1) entry of the robot function matrix Y is $\ell_1^2(\ddot{q}_{d1} + \lambda_1\dot{e}_1) + \ell_1 g \cos q_1$ (with $\Lambda = \mathrm{diag}\{\lambda_1, \lambda_2\}$). To demonstrate the deleterious effects of unmodeled dynamics in adaptive control, the term $\ell_1 g \cos q_1$ was now dropped in the controller. The result appears in Fig. 6 and is unsatisfactory. It is emphasized that in the NN controlle
unmodeled.
NN Controller with Backprop Weig
processing of signals yields a more ac
$x(t)$ than (12) that already contains som
inherent to robot arm dynamics. Since t
the revolute joint variables are as sines

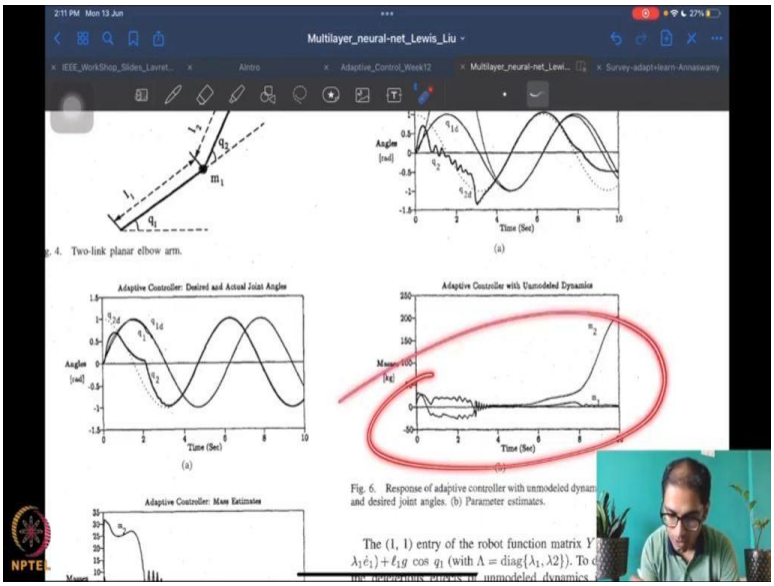


We want to just quickly look at like design. So, so this is like a standard adaptive control for a like a two-link robot, this is what planer two-link arm. So, this is like a standard setup for a lot of experiments in adaptive control. So, then you have the desired and actual joint angles here, so, you have q1, q1 desired and q2, q2 desired.

And then you also have mass estimates. The estimates of course do not converge you see, do not look like they converge to true values. But, whatever, I mean there is not what you guarantee in adaptive control anyway. So, note the good behavior which obtains, since there are only two unknown parameters. So that means, so you have nice persistence type results.

(Refer Slide Time: 25:22)





Now, if you add some unmodeled dynamics, so, some unmodeled terms, dynamical terms in the simulation you add, and you still implement an adaptive controller, not a neural network-based controller. You start to see significant deterioration in the performance. So q1 is like this you can see, and q1 desired is this, and similarly, q2 is this. You know, this guy and the q2 desired is this dotted line. So, you see that things sort of match for some time, but they do not match for a while, and so on and so forth. And the estimates of course, look rather bad, the estimates start to look rather bad.

(Refer Slide Time: 26:00)

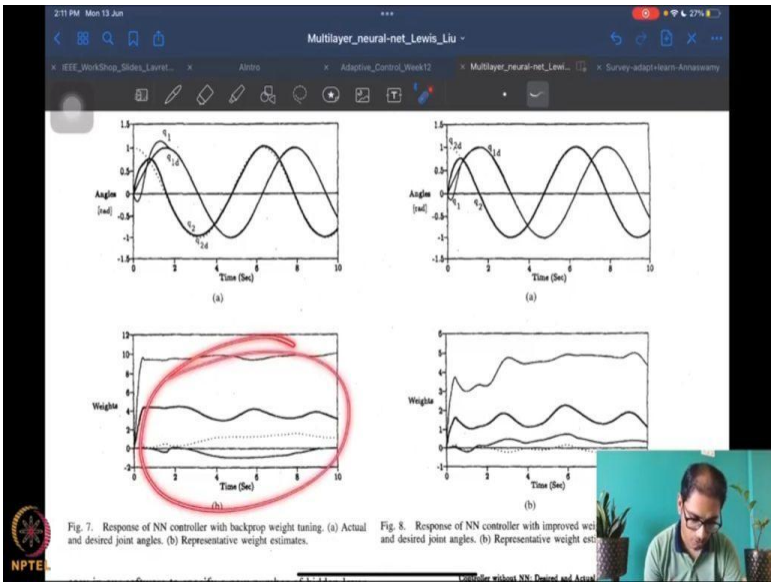


Fig. 7. Response of NN controller with backprop weight tuning. (a) Actual and desired joint angles. (b) Representative weight estimates.

Fig. 8. Response of NN controller with improved weight tuning. (a) Actual and desired joint angles. (b) Representative weight estimates.

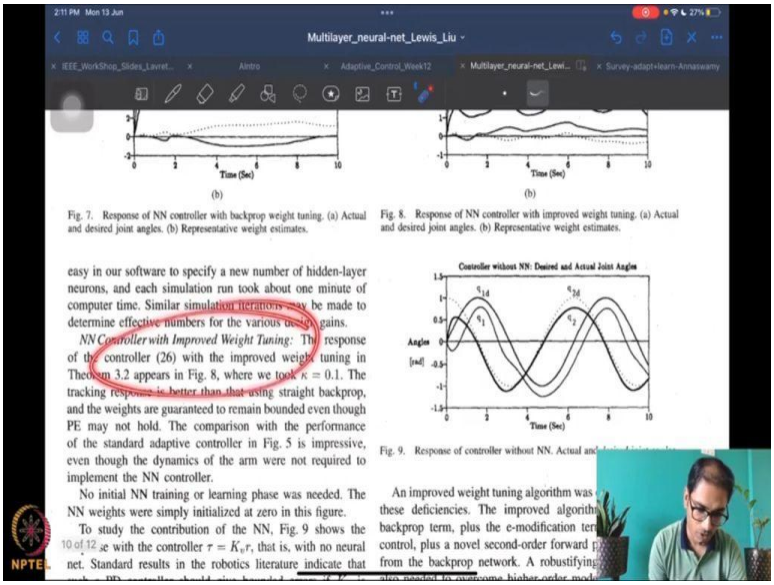


Fig. 7. Response of NN controller with backprop weight tuning. (a) Actual and desired joint angles. (b) Representative weight estimates.

Fig. 8. Response of NN controller with improved weight tuning. (a) Actual and desired joint angles. (b) Representative weight estimates.

easy in our software to specify a new number of hidden-layer neurons, and each simulation run took about one minute of computer time. Similar simulation iterations may be made to determine effective numbers for the various design gains.

NN Controller with Improved Weight Tuning: The response of the controller (26) with the improved weight tuning in Theorem 3.2 appears in Fig. 8, where we took $\kappa = 0.1$. The tracking response is better than that using straight backprop, and the weights are guaranteed to remain bounded even though PE may not hold. The comparison with the performance of the standard adaptive controller in Fig. 5 is impressive, even though the dynamics of the arm were not required to implement the NN controller.

No initial NN training or learning phase was needed. The NN weights were simply initialized at zero in this figure.

To study the contribution of the NN, Fig. 9 shows the response with the controller $\tau = K_v r$, that is, with no neural net. Standard results in the robotics literature indicate that

Fig. 9. Response of controller without NN. Actual and

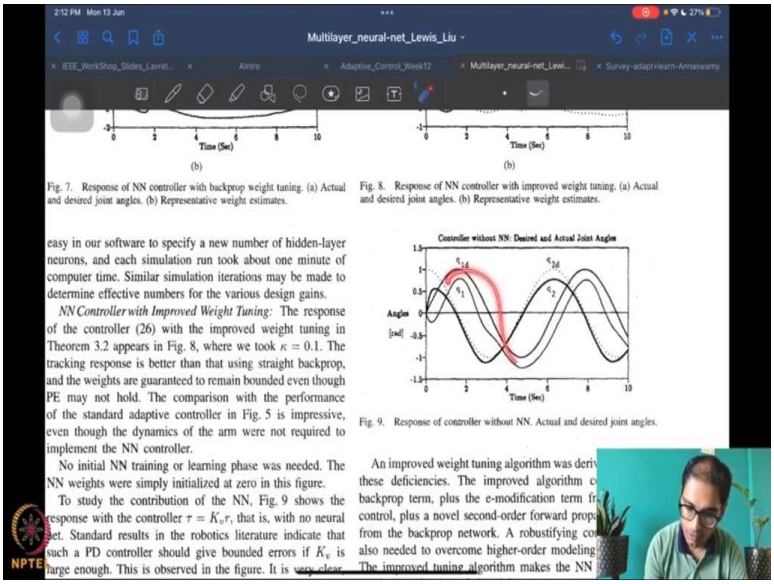
An improved weight tuning algorithm was

these deficiencies. The improved algorith

backprop term, plus the e-modification ter

control, plus a novel second-order forward p

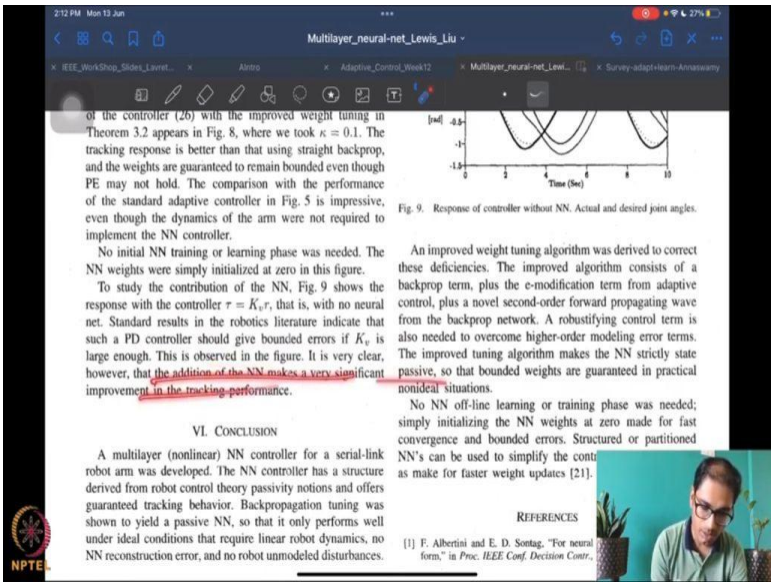
from the backprop network. A robustifying



Then, you implement like similarly, like a backdrop backpropagation-based weight tuning to this kind of a problem. So, and you you basically look at this responsible neural network. And you see that you get nice better tracking, because q1 converges to q1 desired, even with this unknown unmodeled dynamics. And similarly, q2 converges to q2 desired.

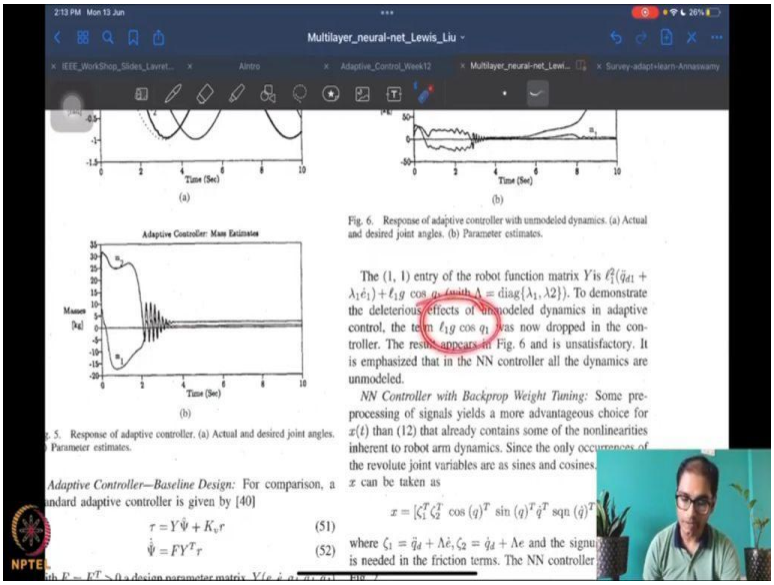
The weights do not seem to do nice things still, and then, we add this improved weight tuning. And you see that you get slightly quicker tracking, better tracking performance. And mass convergence also lot of sort of looks similar. So, and then of course, we see the controller performance with and without without the neural networks. So, that that that is not matching at all, as you can see.

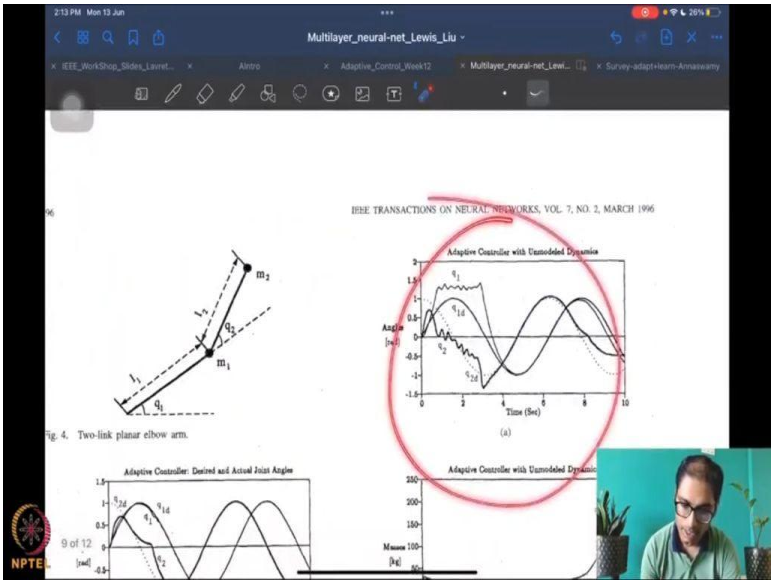
(Refer Slide Time: 27:07)

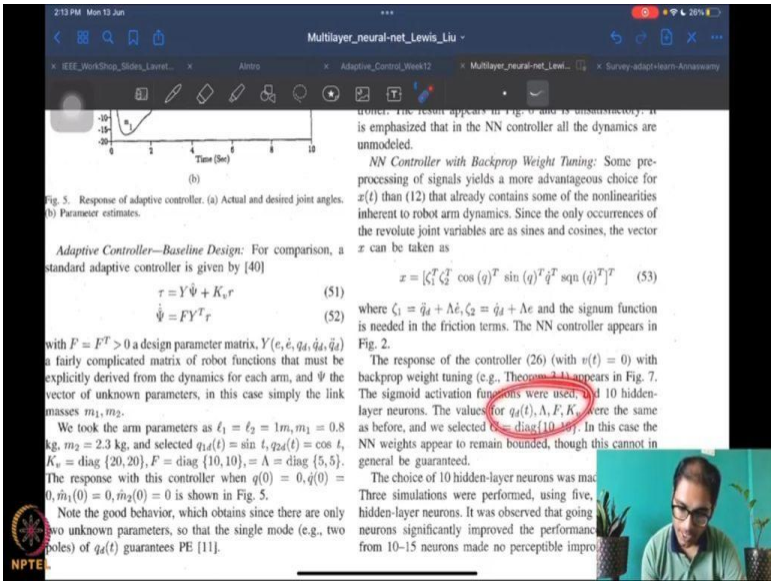


So, this is sort of what the, so, so that is what this is sort of highlighted to study the contribution of the neural network. Figure-9 shows the response of the controller that with no neural network. And standard results in the robotics indicate PD controller should give bounded errors, if KV is large enough. This is observed is however the addition of the neural networks has made a significant improvement in the tracking performance. So, this is the idea, so how they sort of illustrated value of this neural network type method is that they started with this nice baseline robotic model.

(Refer Slide Time: 27:48)

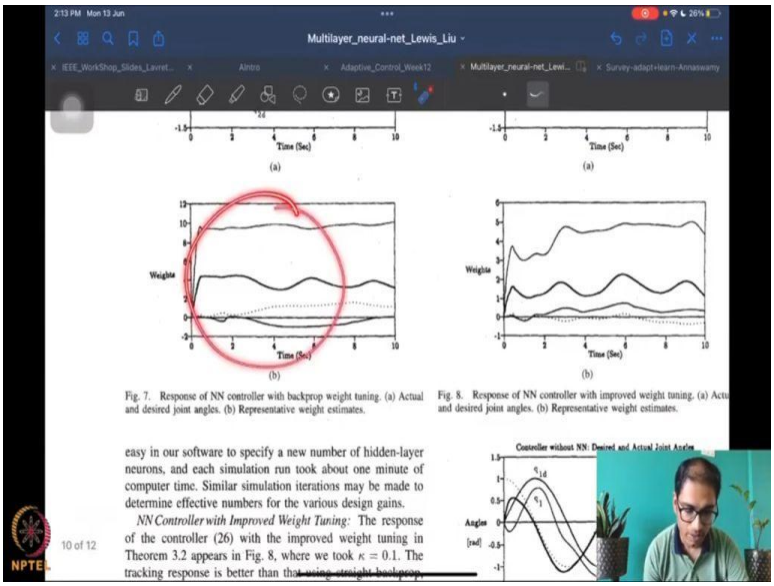


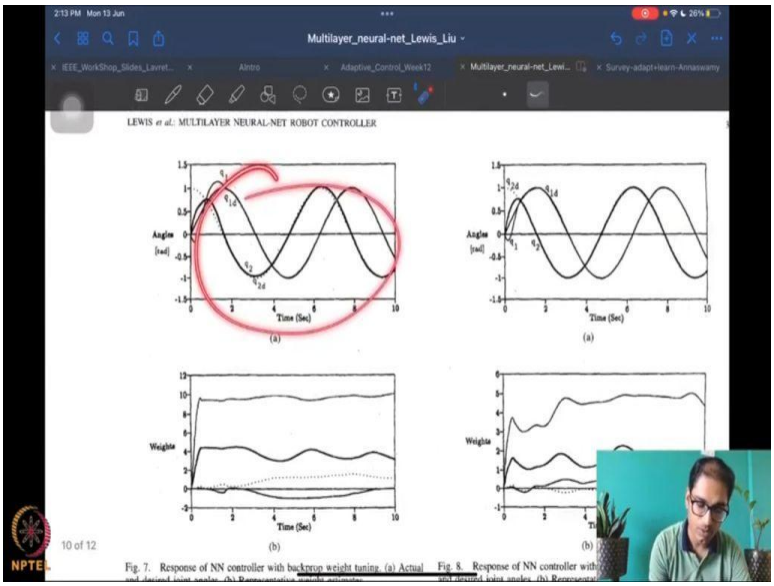


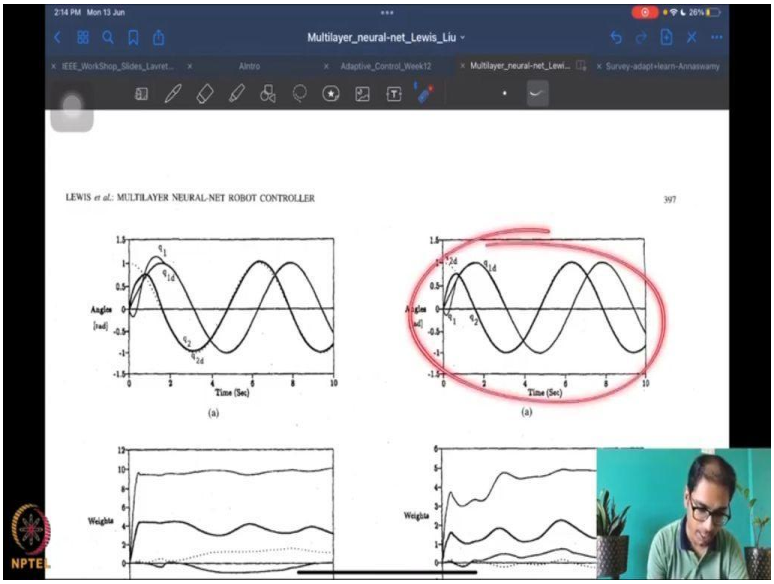


And then they, they added some terms into this the system matrices like the M, Vm and G and F and all that, so they added just one term. So, so, so basically, they see that there is a deteriorated performance here, if you do not have a neural network type controller. Then, so basically is this is like, you see the performance here using the backpropagation is the idealized weight tuning also, when you assume some disturbance terms are exactly 0, well they are not. And then all these quantities are still the same, which is this, there is an additional term.

(Refer Slide Time: 28:30)







And and here, you see that, you sort of have relatively better results in terms of the tracking performance. A weight convergence anyway is not completely guaranteed even here, unless there is some kind of a persistence, but you do have very nice tracking performance. And then of course, you also show improved tracking and quicker tracking by using the modified weight tuning method. And that is what is sort of illustrated here. So, in conclusion, I really hope that you sort of got a good feel for how adaptive control is very closely connected to deep learning.

And the only difference here in adaptive control is that we look to prove stability, which, and therefore analytically, especially. And which may make designing all sorts of arbitrary tuning laws difficult. But, that is essentially what we do in adaptive control. Also, we designed tuning laws and we prove stability with that. And in this case, stability would mean some kind of good function function approximation, and therefore good tracking. So, the other thing is, because we were doing this adaptive control-based learning, we also were doing everything online.

Of course, you can use the same laws and do things offline also, because we saw that this backpropagation is essentially the best tuning that we did in adaptive control law, with an adaptive control law is same as what we have for the standard case, like like, your offline learning type of situation.

So, we are sort of at the backend of our course, and I really hope that this last week did give you a very good idea of how relevant adaptive control is even today, in the sense that it has very very close ties to deep learning. And hence to other forms of learning, like reinforcement learning and

classification problems, which are still connected to some kind of parameter estimation, and parameter learning.

I really hope all of you enjoyed the course as much as I enjoyed delivering it. And I am always open for feedback, comments, applications, cool new results, cool cool new experiments. So, I hope you will stay in touch through this course and learn more about the avenues that nonlinear adaptive control affords us. Thank you. And I am very glad for all of you, for all of your learning, and the fact that you listened to what I had to say for these learnings. Thanks a lot.