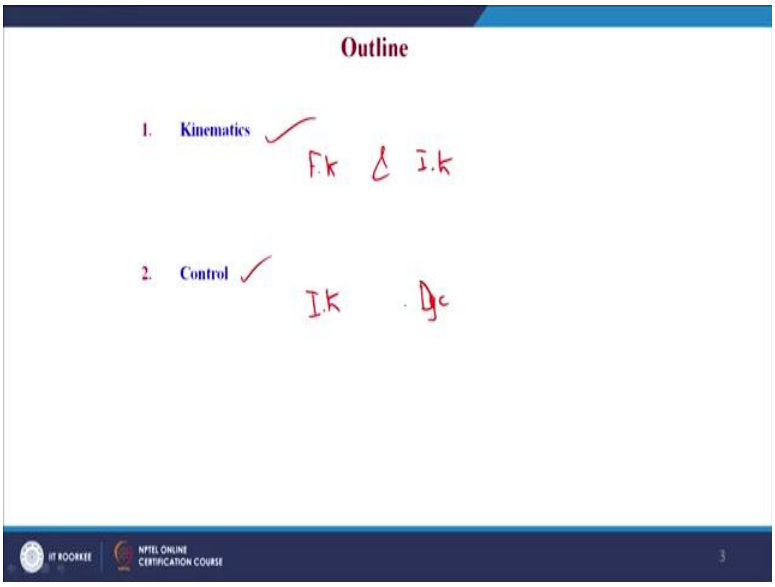
Robotics and Control: Theory and Practice
Prof. Felix Orlando
Department of Electrical Engineering
Indian Institute of Technology, Roorkee

Lecture – 40
Simulation of Robot Manipulators

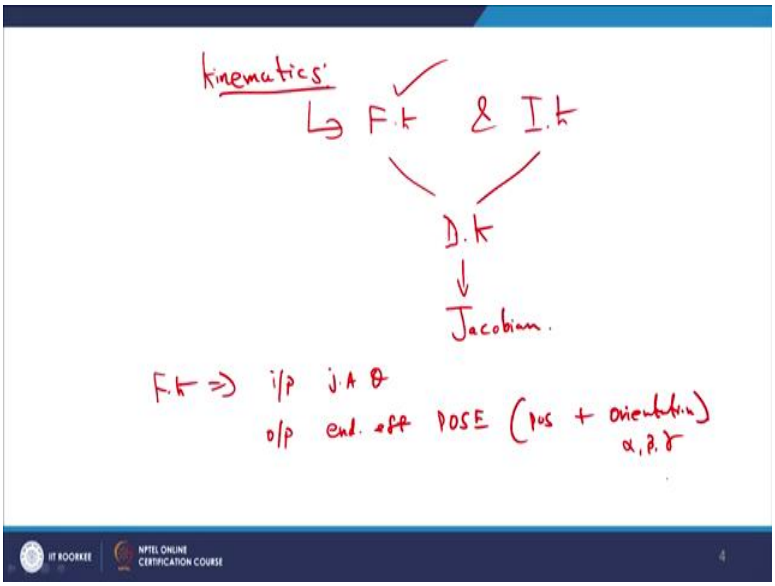
Good morning, today we are going to see a lecture on Simulation of Robot Manipulators through MATLAB programming.

(Refer Slide Time: 00:37)



The outline of this lecture will be as follows first we see the kinematics both forward kinematics and inverse kinematics and then we see about the programming part of controlling a manipulator that is through inverse kinematic model based control and dynamics based control.

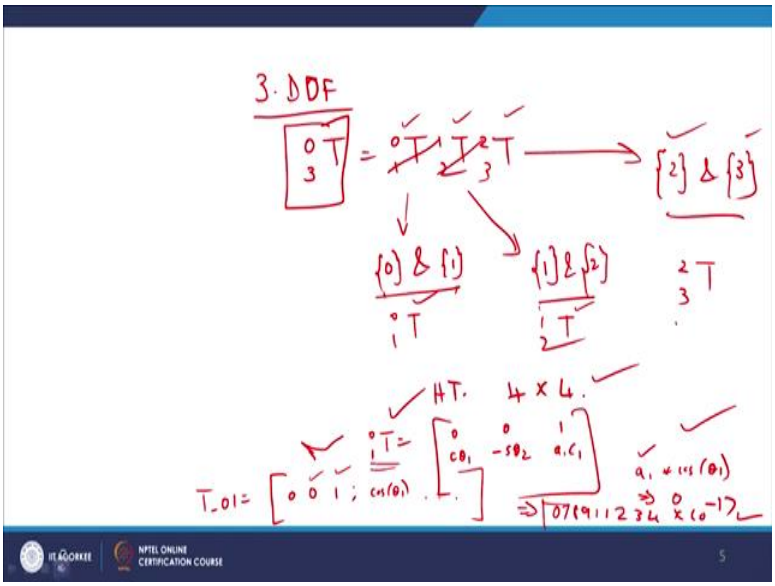
(Refer Slide Time: 01:02)



So, first we start with the kinematics part. As we know that kinematics can be further divided into forward kinematics and inverse kinematics. In between this we have the differential kinematics where the Jacobian part is coming into picture right.

So, first we start with the programming of forward kinematics through MATLAB programming. So, first what we do with the forward kinematics what is Forward Kinematics? Forward kinematics is basically the input here is joint angle theta say for example and the output here is the end effector position; end effectors pose, that is basically position plus orientation. The orientation could be $\alpha, \beta,$ and δ or the Euler angles or the Euler angles. So, now we see how the program or first do the procedure for forward kinematics.

(Refer Slide Time: 02:18)



So, forward kinematics we know that for example for a 3 degrees of freedom manipulator, so will be having the transformation matrices homogeneous transformation matrices coming into. So, this the final homogeneous transformation matrix which is obtained by the product of 3 homogeneous transformation matrices, which defines the relationship between frame 0 and 1. Then frame 1 and 2, then the homogenous transformation that relates between the frames 2 and the final third frame which is placed at the end effector tip.

So, now we compute each one of this ${}^0_1T$ then ${}^1_2T$, then ${}^2_3T$. So, that we put it in the generalized standard DH parameter based homogeneous transformation matrix which is of size 4×4. So, this in this transformation matrix we substitute the DH parameters and then we get the respective homogeneous transformation matrices between the successive frames 0 to 1, 1 to 2 and 2 to 3.

Then what we have to do here is we need to do one thing that is basically. For example, if ${}^0_1T$ is having certain cases like

$$ {}^0_1T = \begin{bmatrix} 0 & 0 & 1 \\ cos\theta_1 & -sin\theta_2 & a_1 cos\theta_1 \end{bmatrix} $$

something like that. If it has then we have to substitute in the MATLAB code itself with this notation T_01 equal to we have going to substitute a matrix in this way that is

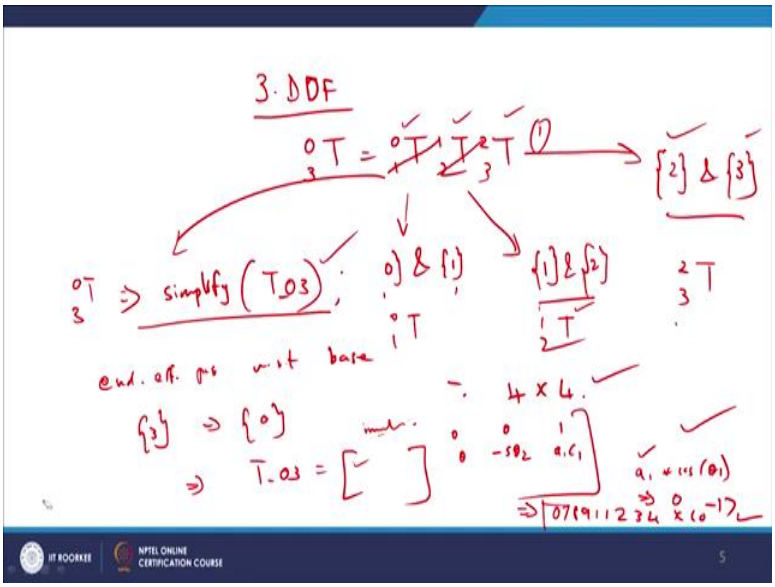
$$T\_01 = [0 \quad 0 \quad 1; cos\theta_1 \ldots]$$

Accordingly we have to $cos\theta_1$ like that we need to substitute those values straightaway.

And one important thing is if it is 0 and if it is one kindly you have to put the same values 0 and 1 here. Instead of placing it say for example, a 1 into $cos\theta_1$ instead of putting a 1 into $cos\theta_1$ which leads to 0. For example, a_1 is 0 in the DH table. It is better to put 0 straight away instead of having a_1 declared as 0 first, why because the MATLAB will give you the output in terms of.

For example, it will give the values 0 7 …. is something like 10^{16} numbers into 10^{-17} it gives like this for the value of 0. So, instead of having these big numbers existing in your homogeneous transformation matrix is better to put 0 or 1 the scalar values straight away in the declaring homogeneous transformation matrices. Then just bring into your new just erase this and then.

(Refer Slide Time: 05:40)

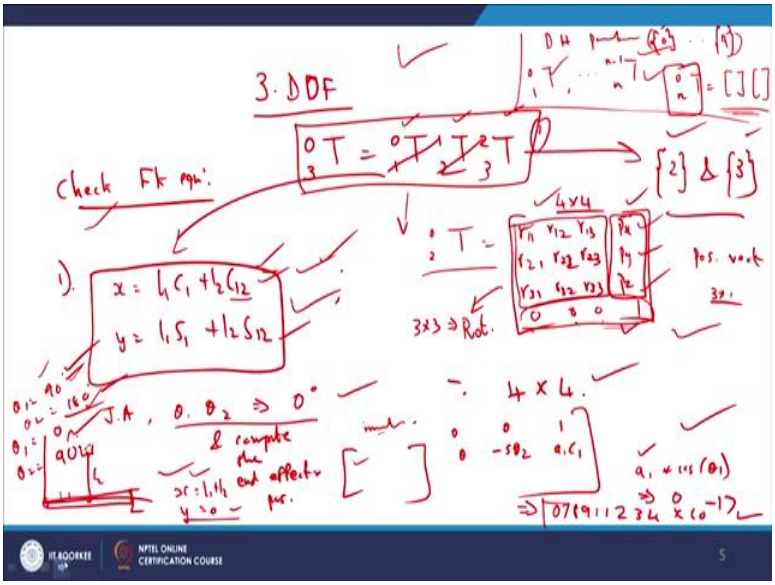


so now, if you see finally ${}^0_3T$ is obtained by this equation say for example, it is one then we need to have ${}^0_1T$ could be of simplify of ${}^0_3T$. So, that gives the simplified form of the homogeneous transformation matrix, which is going to define the end effector position with respect to the base frame. That is the third frame going to be defined with respect to the o^{th} frame.

So, what I mean to say is this is the command that should be used in order to have the simplified expression of ${}^{0}_{3}T$ which is going to be a pure scalar based homogeneous transformation matrix 4×4, instead of having the real this imaginary values and all existing as a matrix element.

So, another step now we have to go for checking how the defined equation is going to give you the right you for forward kinematics. How to check it? That is a procedure that we need to see here now.

(Refer Slide Time: 07:22)



So, there are two steps to clearly check that. So, first one is that is checking the FK equation, whether it is right or wrong. So, this can be checked in such a way that for example if you have $x = l_1 cos\theta_1 + l_2 cos\theta_{12}$ and $y = l_1 sin\theta_1 + l_2 sin\theta_{12}$. Where $cos\theta_{12}$ nothing but $\cos(\theta_1 + \theta_2)$, $sin\theta_{12}$ is nothing but $\sin(\theta_1 + \theta_2)$. When this equation is there as a final end effector equation that is for a kinetic equation, we need to test whether this is the right equation or not.

So, how to do that? First step is we need to put the joint angles. Say for example, θ_1, θ_2 both 0 degrees and compute the end effector position. So, what is the value for x and y? When it is θ_0, vector $\theta_0 = [\theta_{10}, \quad \theta_{20},]$ which will be having the manipulator being in the horizontal plane.

So, you know that it will be having the value only for x which is $l_1 + l_2$ where $y = 0$ that we know. From our physics based idea we could find we could really confirm that with $\theta_1,\ \theta_2$ being 0 degrees, you will have the value existing the numerical value existing only for the x parameter of the tip position and the $y = 0$.

Similarly, the next thing is to give $\theta_1 = 0$ and θ_2 equal to another value which is a regular value 90 degree. Then test this, so that should be like this where we have the x and y this which is a regular value l_1 this is l_2. We know from this idea or the imagination of this being bending as an elbow bent as 90 degree we could able to figure out the values and we can check it from the forward kinematic equation obtained from the product of the 3 homogeneous or two homogeneous or n number of homogeneous transformation matrices.

Then we go for another regular value which is θ_1 equal to say 90 degree and θ_2 being 180 degree. So, with this type of regular value says 0, 90 and 180 degrees we could be able to test or verify whether our a forward kinematics equation is right or wrong. These are the steps that we could be able to check or verify our forward kinematics equation. Because the forward kinematics equation has these steps first find the DH table.

DH parameters the DH parameters could be found by obtaining or assigning the frames from 0 to n, 0 is for the base frame and the n is for the tip frame and then from the DH table find the respective frames. That it is fine the respective homogeneous transformation matrices that leads to ${}^{0}_{1}T$ to ${}^{n-1}_{n}T$, this many number of homogeneous transformation matrices that corresponds to the row of each forward each DH table. We could able to obtain the final matrix which is product of each homogeneous transformation matrices.

Once we obtain this homogeneous transformation matrix we could be able to figure out what stands for the tip and what stands for the orientation. Let me quickly tell you what stands for the position and what stands for the orientation from this 4×4 sized homogeneous transformation matrix. For example, ${}^{0}_{2}T$ is this one we obtained

$$ {}^{0}_{2}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_x \\ r_{31} & r_{32} & r_{33} & P_x \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

So, this from this 4×4 homogeneous transformation matrix, we could say that the first 3 columns and that two particularly the first 3 elements of the first 3 columns denote the

orientation. That is 3×3 matrix gives you the orientation value for the end effector. Whereas, the last fourth column the first 3 elements of the fourth column gives you the position vector, which is basically 3×1 vector which gives the position value alone of the end effector. And these are the scale parameters that could be helpful why we have this scale parameter that you would have learnt in this beginning of this lecture itself, that these are meant to have a description of one frame.

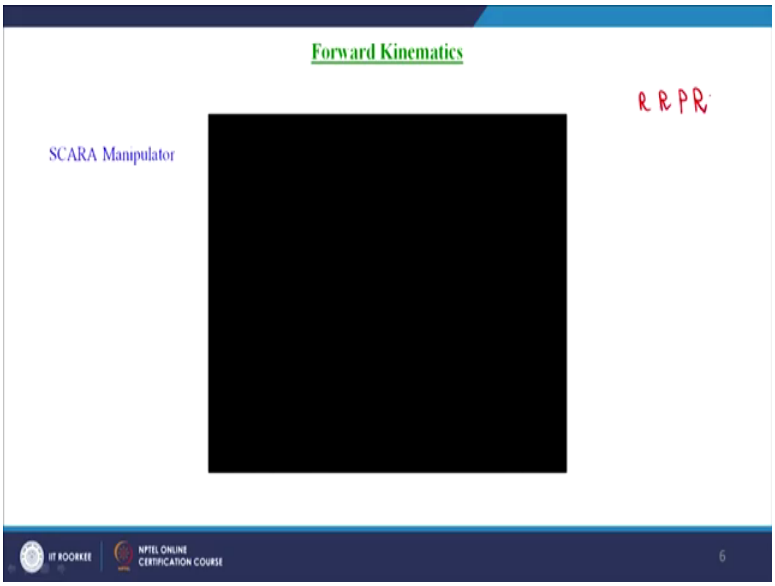
For example, the tool frame of the manipulator to be defined with respect to the end effectors or the manipulators base frame or the object frame can be defined with respect to the base frame of the end effector. In that situations the 0 0 1 scale parameter will be helpful through the homogeneous transformation matrix multiplication to get the proper and they define the value of the end effector tip or the object position and orientation with respect to any reference frame or the world a frame.

So, this is the way that we could able to do the forward kinematics part and let me brief it quickly one more time, that once given here n degrees of freedom manipulator the first step is to obtain the DH table, then form the respective homogeneous transformation matrices and then compute the final homogeneous transformation matrix that is obtained by the product of successive frames based homogeneous transformation matrix. Then compute the position and orientation of the end effector from the final homogeneous transformation matrix.

Then the obtained for example, they obtained position whether it is right or wrong will be verified by this checking method. Which is nothing but substitute the joint variables either θ or the distance value the translation notation given by D, either by changing by varying them gradually starting from say 0 degree, 90 degree, and 180 degree. We can gradually find out how that is going to be behaving or going to give the position as per our correct information or imagination that given 0 degree it must be lying on the horizontal plane.

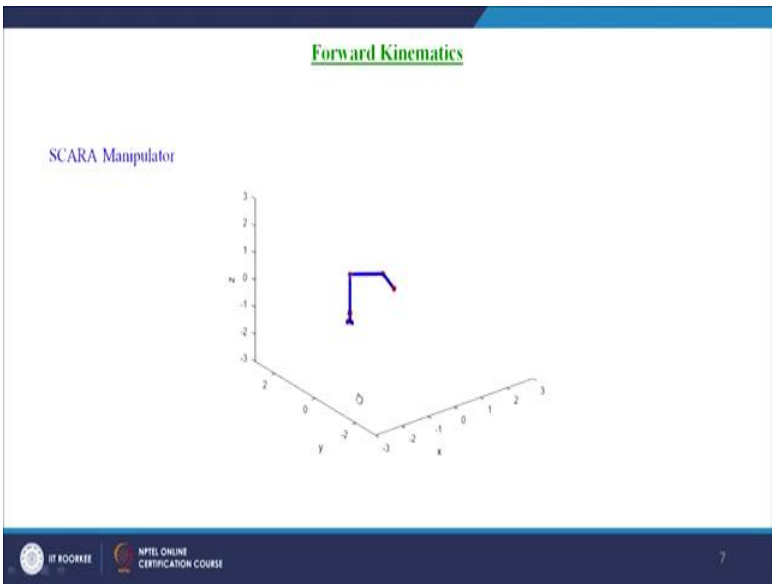
So, only the *x* axis comes into picture, so in that situation your forward kinematics equation should not give the *y* value and this *z* value. Accordingly, you can verify by incrementing the angles to the regular values, say 90 degrees and 180 degrees even for particular or all the values of the joint angles.

(Refer Slide Time: 15:44)



Then we move on to certain examples; first we start with the SCARA manipulator. SCARA is a 4 degrees of freedom, first one it is revolute, second is revolute, third is prismatic and the last degrees of freedom is a revolute joint which is a end effector rotation part. So, here we see this simulation of it.

(Refer Slide Time: 16:09)

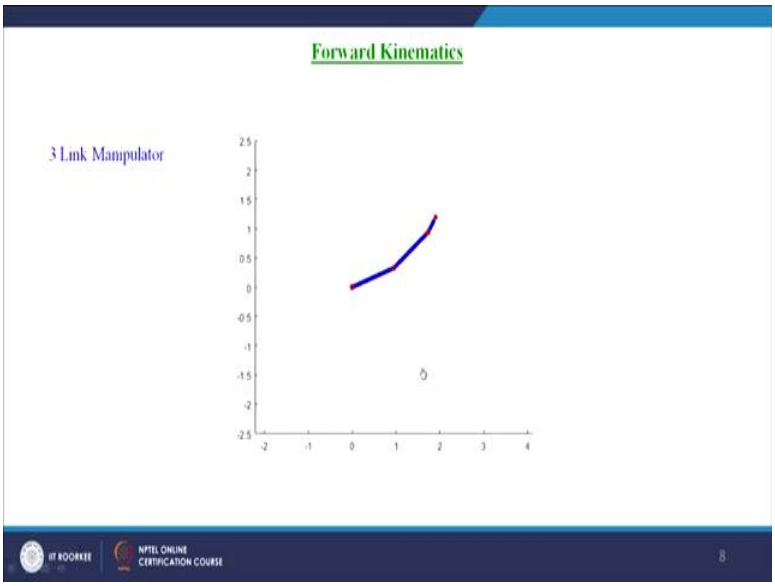


So, in this simulation graphical simulation, we show that the end effector is you can see that here the end effector is getting rotated, the rest portion is getting rotated. As you can see here it is only this angle given into the simulation part. So, by this graphical simulation

through the MATLAB programming we can observe that how the system is actually behaving by giving the joint input. So, that the end effector or the whole system behaves how as per the given input joint angle.

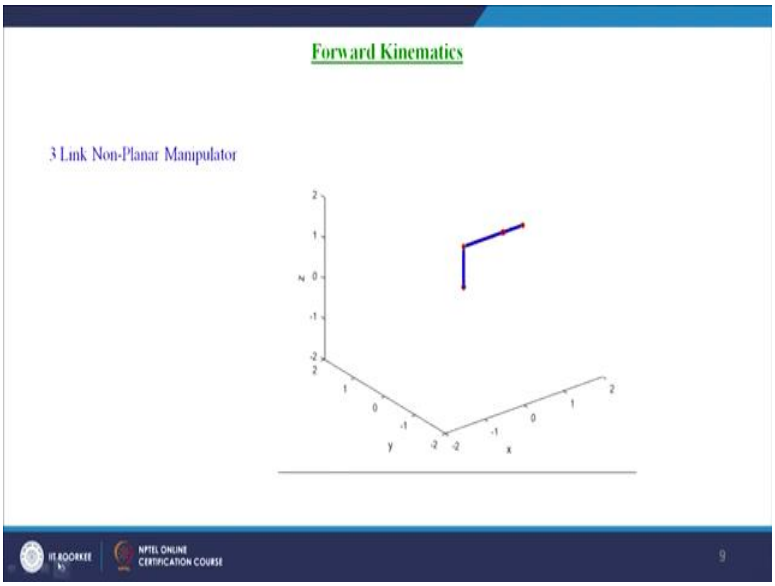
So, the next simulation is also of SCARA manipulator only, where you can see it is moving almost all the degrees of freedom of the manipulator. There you can see it is both it is all the joint variables that is the first joint θ_1, θ_2 and the prismatic one and also the fourth degrees of freedom all or getting buried and hence we have this type of simulation graphical simulation. From this we could observe that how the system actually behaves by this graphical information we get more inference than compared to the position plot or the orientation plot alone.

(Refer Slide Time: 17:46)



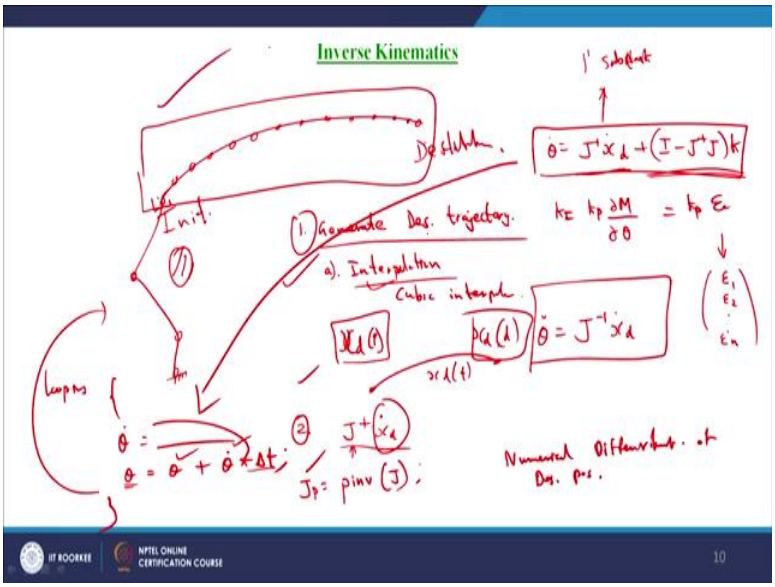
Next we see a 3 link manipulator with the angles getting incremented, this is a simulation where we move all the 3 degrees of freedom θ_1, θ_2 and θ_3 of this 3 link manipulator and we obtain this trajectory I repeat it.

(Refer Slide Time: 18:14)



Next we finally see the graphical simulation which is obtained by the MATLAB programming for a 3 link non planar manipulator, it is all of revolute; revolute; revolute and finally revolute as well the degrees of freedom.

(Refer Slide Time: 18:35)



Next we come into the inverse kinematics which is a very important part how we do the inverse kinematics through coding part, because for the inverse kinematics we generally deploy the MATLAB programming or any programming part just to track not only a set

point it is to track a trajectory. Where the trajectory is of several waypoints contained in it, so where you can see this is the initial point and this is the destination point.

So, with this trajectory you can see that there are so many way points in it, this trajectory has to be traced by a manipulator say for the 1 2 3. So, 3 degrees of freedom manipulator should track you should track this trajectory given trajectory. So, how to do that that is the task to be obtained by the inverse kinematics.

So, inverse kinematics to for a given trajectory can be traced by this way, that is basically $\dot{\theta} = J^{+}\dot{x}_d + (I - J^{+}J)k$. Where k_1 or k is equal to $k_p \frac{\partial M}{\partial \theta}$ or simply we can say $k_p \varepsilon$. Where ε is a vector which is having the elements $\varepsilon = [\varepsilon_1 \quad \varepsilon_2 \quad \varepsilon_3 \ldots \varepsilon_n]$. So, here if it is a non-redundant manipulator we can have the joint velocity expression being $\dot{\theta} = J^{-1}\dot{x}_d$. Simply for a simple inverse kinematics this is the expression.

Whereas if it is a redundant manipulator you will have the first term and the second term, the first term is meant to achieve the primary sub task sub task. The primary sub task is to track this given trajectory. Whereas, the secondary sub task is to utilize the redundancy involved in a manipulator, so that it can be utilized to achieve any additional application for us the additional work for us.

The additional work could be if there is an obstacle here, this obstacle can be avoided by deploying the redundancy of the manipulator that is one way of utilizing the redundancy of a manipulator. And the second way could be utilizing the redundancy of the manipulator in order to avoid the singular configuration of the manipulator.

When the manipulator is singular it means that it will not get the joint angles for a given particular end effector position. So, it gets struck there, it will not have any way to move. So, that is why redundancy is quite useful it has many advantages as we have discussed earlier.

So, now how to do this programming part, so what are the sections involved in programming in order to achieve this inverse kinematics. The inverse kinematics the first two role are the first part is the desired trajectory. So, if we need to feed the desired trajectory through the manipulator. So, the first step is generate desired trajectory that is the first one. It can be generated one way is to generate through interpolation. As you have seen in this lectures previously before 20 lectures before the twentieth lecture you would

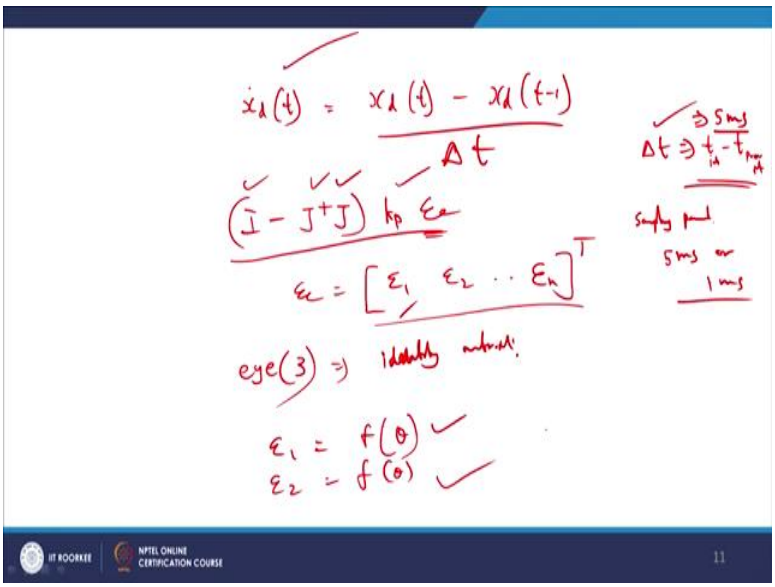
have seen this how to perform interpolation. The simplest one is cubic interpolation given say $x_d(1)$ and $x_d(d)$. For example, this is $x_d(i)$ let us say that is x_d initial point and x_d final point we will get a trajectory, based on cubic polynomial by the interpolation method.

So, the first step to write the program part is to generate the desired trajectory by giving the initial position and the final position. And through cubic interpolation methodology you can obtain here cubic spline trajectory, that will be the desired trajectory for the manipulator this is the step one for programming part. And then comes the step two which is basically going for this equation, that is the generalized solution for the joint velocity that is the second part that part. If you see that is the first term and the second term.

So, the first term is $J^{-1}J^{+}\dot{x}_d$. So, you have here first in step one you have generated x_d. For example, $x_d(t)$ where t varies from initial time to the final time, there is so many values in between it and the first term of the equation of joint velocity is having desired tip position velocity.

So, how will you get this? So, you need to do J^{+} how will you do this by MATLAB we use this common $pinv(J)$ of the Jacobian matrix J. Where you have the Jacobian expression already defined in your coding part. Then $J_p = J^{+} = pinv(J)$, that gives the Jacobian inverse pseudo inverse this portion and how will I get this velocity. So, velocity will get by simple numerical differentiation of the desired positions. How to do that?

(Refer Slide Time: 26:23)



We say that $x_d(t)$ city is say for example five second, at five second what is that. So, $x_d(t)$ equal to

$$\dot{x}_d(t) = \frac{x_d(t) - x_d(t-1)}{\Delta t}$$

instead of this I let us me put in a generalized way by Δt, where Δt is nothing but where Δt equal to t at this instant $-t$ at the previous instant current instant time minus previous instant.

So, this is varied by the sampling period say, for example the sampling period involved in this inverse kinematics. So, that in general we take 5; 5 millisecond or 1 millisecond we generally take for simulation. So, this Δt generally comes how to be the time which is the sampling period say 5 millisecond or 1 millisecond so that comes here. So, with this you can able to obtained all the velocities given the position of the end effector decide position, then we can see that this one is the first term over.

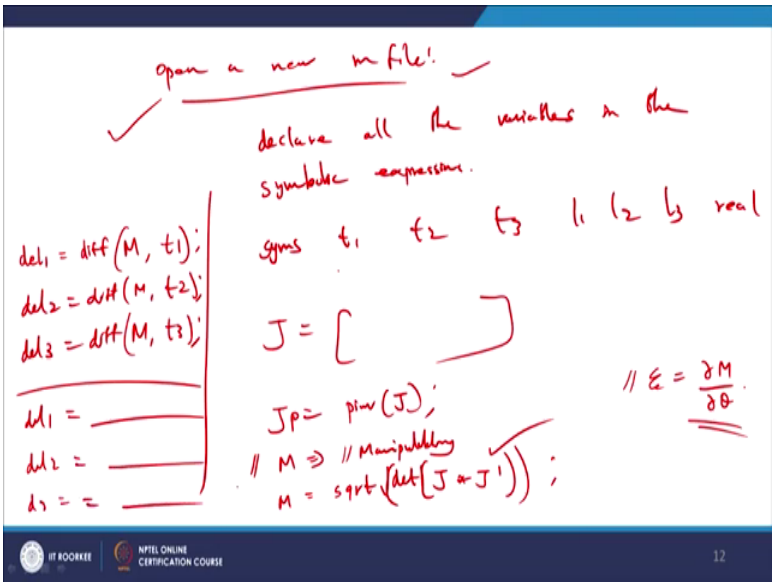
Now, we have

$$= (I - J^{+}J)k_p\,\varepsilon$$

where $\varepsilon = [\varepsilon_1 \quad \varepsilon_2 \quad \varepsilon_3 \ldots \varepsilon_n]$, of course which is $n \times 1$ vector. So, in the second term we have this we have already computed J^{+}, I is identity. $eye(3)$ you will get the 3×3 identity matrix this is the command for that and k_p is a scalar constant scalar value which is a positive scalar value in order to have this term getting increased and epsilon is given by this vector.

So, it is epsilon is an expression which is going to be obtained. So, this is for identity matrix and ε expression $\varepsilon_1(\theta)$. Similarly ε_2 is also a function of θ and ε_3 as well.

(Refer Slide Time: 29:25)



So, how to get this expression now? So, the important point now here is open a new file; open a new m file and then declare all the variables in the symbolic expression expressions. That is $syms\ t_1\ t_2\ t_3\ l_1\ l_2\ l_3\ real$. That means, none of these variables are going to be having the complex terms. So, you explain express or declare all these variables in the symbolic expression in this new file, then you have the expression for Jacobian that you know how to do.

Then we know that $J_p = pinv(J)$, then we need to have the expression for already we know that expression for $\varepsilon = \frac{\partial M}{\partial \theta}$. So, M expression is nothing but the manipulability which is given by is the manipulability, which is given by

$$M = \sqrt{JJ^T}$$

And then now we go for this expression I write here we go for the expression of this one. So, how to do that? So now, we say del_1 is given by ε_1 is given by $diff(M, t_1)$ and $del_2 = diff(M, t_2)$. And similarly say del_3 equal to $diff(M, t_3)$.

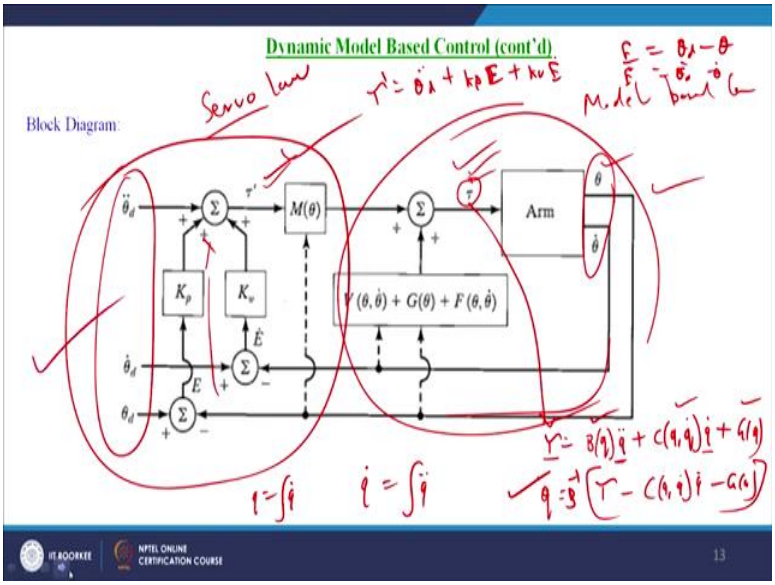
Then you run this code this new file you run this then in the command prompt type del_1 you will get the expression copy that, del_2 get the expression copy del_3 type it get the expression copy it and go back to the main file. This is a main file where you have done all these things here you can substitute those values and continue this looping. Where you have up to the expression for theta dot which is this one.

Then how will you compute theta velocity from this, simplest way is

$$\theta = \theta + \dot{\theta}\Delta t$$

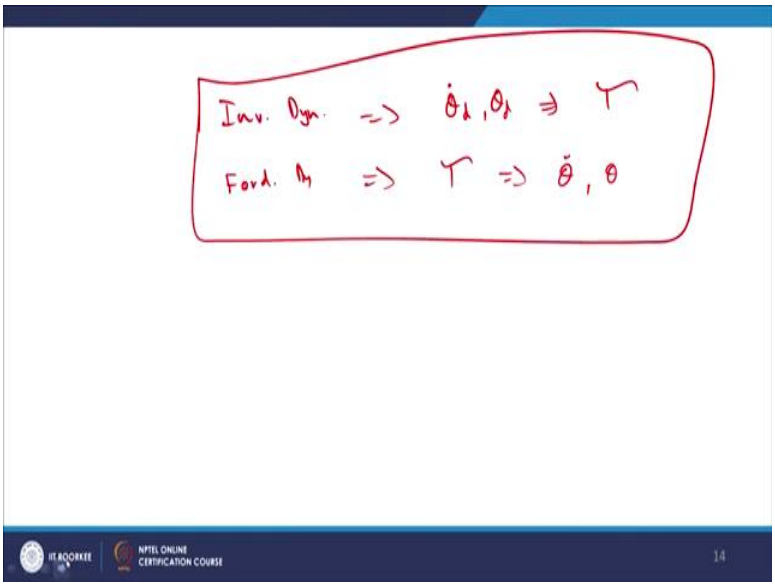
This is the previous value of joint angle and the previous are the previously computed Δt and it is a sampling period which is the same value. But this we will get the next value of joint angle and this procedure continues for all the way points. That is how we do the inverse kinematics in the MATLAB programming.

(Refer Slide Time: 33:15)



Now, we move on to the dynamic model based control. So, in this way we could be able to perform the inverse kinematics based control and now coming to the dynamic model based control, what we have seen earlier in my lecture is this portion in the dynamic model is basically the Servo law; Servo law. Servo stands for the meaning track and this is the portion which stands for the model based law. That is why this is getting partition and hence it is called control based on partition rule or control partition scheme you can say.

(Refer Slide Time: 34:04)



So, this one let me quickly put forward we have the inverse dynamics and the Forward dynamics. The inverse dynamics is having the joint trajectory is given as input and we obtain the joint torque as the output. Whereas, the forward dynamics is the reverse one we give the joint torque and obtain the joint actual trajectories that is what we do in this forward. And inverse dynamics part you can see here it is a combination of both inverse dynamics and forward dynamics, how? Yes.

So, in the control part we give this joint trajectory the desired joint trajectories in terms of the joint velocities. Then we do the servo law which is obtain from the feedback, then we get this τ' as you have seen in the dynamic model base control in the last lecture. Then we obtain finally τ that is the control part which is the control law that goes into the manipulator.

So, we obtain this τ from the joint torque from the joint angular trajectories. So that means, here is a control part done, now we need to obtain the actual values. How will we do this? So, this actual value of the joint trajectories can be obtained from the joint torque by the simulation procedure. What is that? Because we know that τ equal to

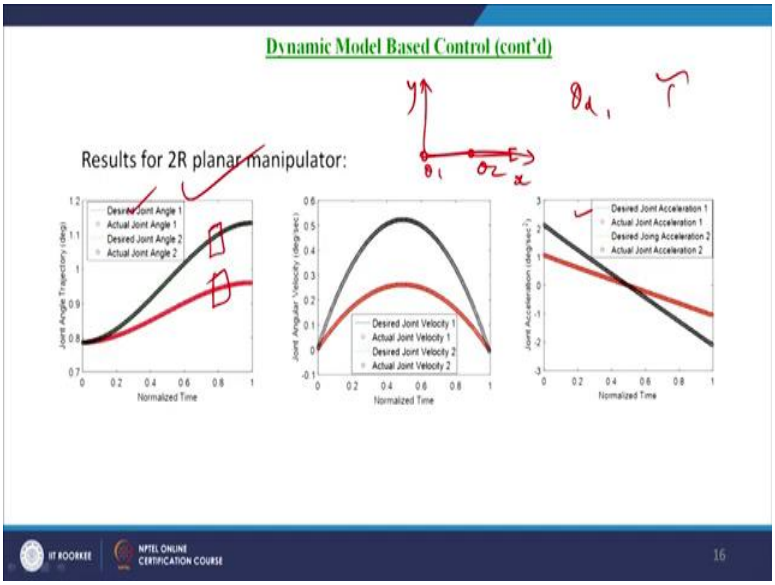
$$\tau = B(q)\ddot{q} + c(q, \dot{q})\dot{q} + G(q)$$

So, we have at present we have τ getting into the input τ we have, we do not know what is $B(q)\ddot{q}$ and we do not know $\dot{q}$, but we do we know $B(q)$, $c(q,\dot{q})$ this one. So, from this we can take

$$\ddot{q} = B^{-1}(q)[\tau - c(q,\dot{q})\dot{q} - G(q)]$$

So, from this we obtain the actual acceleration, from that we could able to get velocity by the integration of double dot and then finally q with the integration of velocity. So, with this way we get the actual values and get feedback to the Servo law, in order to get this τ' which is basically τ' is Servo law which is given by $\tau' = \ddot{\theta}_d + k_p\theta$ or $k_pE + k_v\dot{E}$, where E stands for $\theta_d - \theta$ and $\dot{E} = \dot{\theta}_d - \dot{\theta}$. So, with this way we can similarly do the MATLAB based programming of it.

(Refer Slide Time: 37:07)



And finally, we could for example, we have done the same procedure for the two linked planer manipulator x and y. So, theta 1 theta 2 being this, so given here decide theta d we obtain the tau. So, these are the given desired trajectories and we could be able to obtain the desired trajectories and the actual trajectories being merged. But classic joining as you can see here and with this we wind up today's lecture. In this lecture we have seen how we could program in order to simulate a simple or even complex n degrees of freedom manipulator.

We started this lecture with forward kinematics how we program it and we have seen certain simulations graphical simulations showing how they behave with given joint angle values and then we see quickly the procedures how to program for inverse kinematics with tendency and non-relent manipulators. And then finally we have seen the dynamic model based control scheme with the control a partitioning how we could program it and get the actual values

Thank you so much.