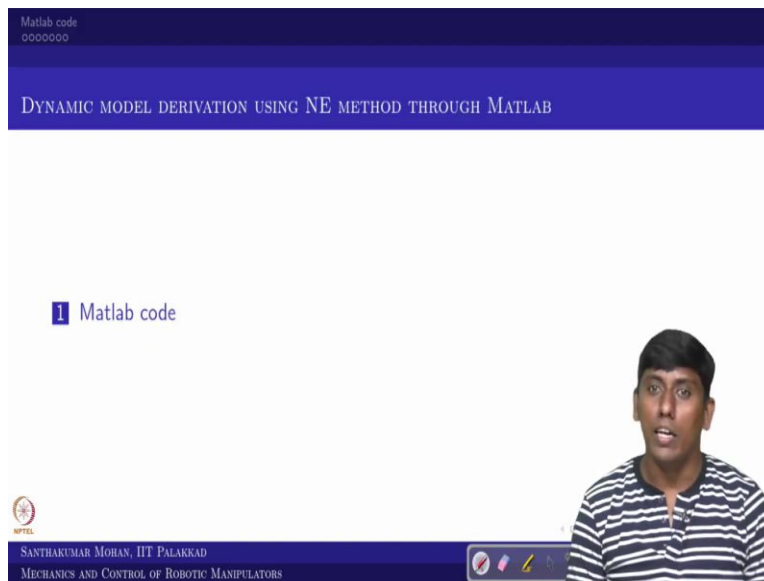


**Mechanics and Control of Robotic Manipulators**  
**Professor. Santhakumar Mohan**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Palakkad**  
**Lecture No. 26**  
**Dynamic Model Derivation Using Newton-Euler Method in MATLAB**

Welcome back to mechanics and control of robotic manipulator. Last few classes we have seen how to derive the equation of motion. And in that two popular method we have seen one is Lagrangian Euler. The other one is Newton Euler. So, the Newton Euler, I said it is better for the computational perspective. So, in this particular lecture we are going to see how to derive the equation of motion for given example using Newton Euler method with the help of MATLAB codes. So, in the sense here we are going to see the MATLAB session.

(Refer Slide Time: 0:48)



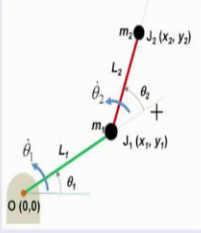
The image shows a presentation slide with a dark blue background. At the top left, it says "Matlab code" followed by a series of "o" characters. The main title of the slide is "DYNAMIC MODEL DERIVATION USING NE METHOD THROUGH MATLAB". Below the slide, there is a video inset of Professor Santhakumar Mohan, who is wearing a striped shirt. The video inset has a small "1" icon and the text "Matlab code" next to it. At the bottom of the video inset, there is a logo for NPTEL and the text "SANTHAKUMAR MOHAN, IIT PALAKKAD" and "MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS".

So, in that sense we will move forward. So, we will be talking about dynamic model derivation using Newton Euler method through the help of MATLAB. So, in this sense we would be straightaway see the MATLAB code. And then I will move to the MATLAB environment.


(Refer Slide Time: 1:03)

Matlab code  
●○○○○○

Example: A planar 2R serial manipulator



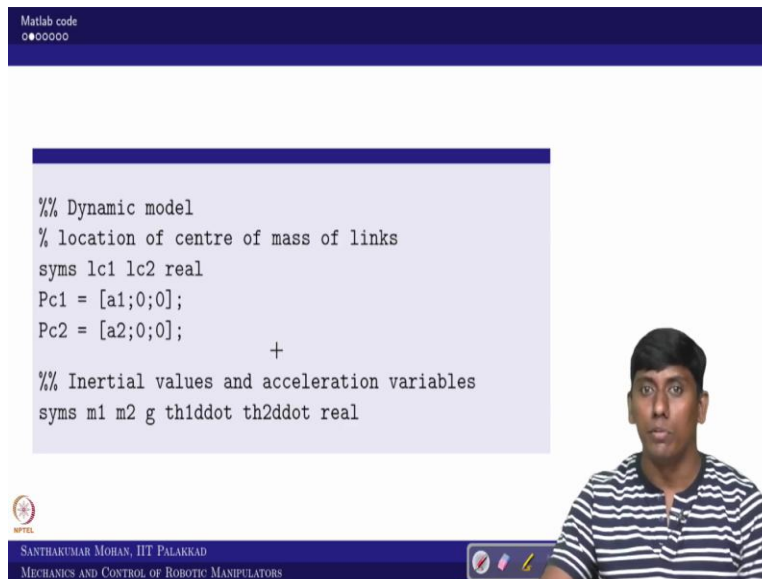
NPTEL  
SANTHAKUMAR MOHAN, IIT PALAKKAD  
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS



So, in the sense so this is the example which we have derived in the you can say regular lecture. The same example I am taking it for simplicity, but it is not going to restrict you can actually use any you can say serial manipulator in this particular case only thing you have to change the DH parameter and then equation of motion in series. So, that I will show you in the MATLAB session.

But right now, you can take it that planar serial manipulator we have taken. So, in that case so, what one can see like if I know this  $m_1$   $m_2$  and if I know  $L_1$  and  $L_2$  and I assume that  $\theta_1$  and  $\theta_2$  are the joint variable, I can derive the equation of motion with the help of forward propagation and backward propagation together.

(Refer Slide Time: 1:52)



```
Matlab code
%% Dynamic model
% location of centre of mass of links
syms lc1 lc2 real
Pc1 = [a1;0;0];
Pc2 = [a2;0;0];
+
%% Inertial values and acceleration variables
syms m1 m2 g th1ddot th2ddot real
```

SANTHAKUMAR MOHAN, IIT PALAKKAD  
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, for that I am rewriting that equation. So, first what we need to know. So, till now in velocity kinematics so, we have already shown how to derive you can say the DH parameter through the help of frame once you derive the DH parameter or once you obtain the DH parameter, you can substitute in one of the MATLAB code you can find the rotational matrix; rotational matrices and position vectors of the individual joints these are all we have seen. Further what we have seen?

We can see how to propagate the angular velocity and linear velocity if we know the base angular velocity and linear velocity. So, we always assume that the base is fixed. So, these codes all we have seen in you can say the kinematic model derivation and as well as differential kinematic cases or you call velocity propagation model. So, in that sense what right now we required? We required what would be the centroidal location.

So, in this case the mass is the link 1 mass is concentrated at the point J1 which is the mass is  $m_1$  and there is no inertia it is a point mass. Similarly, the second link mass is concentrated at  $m_2$ , or you can say J2 point as a  $m_2$ . So, in that sense what one supposed to know. So, if you recall your frame arrangement so along the L1 whatever that is assigned is  $x_1$  axis along L2 that would be  $x_2$  axis.

So, now, if I see the link 1, the centroidal location, so, from frame 1 that would be you can see along  $x$  there is a L1 distance  $y$  and  $z$  0 0. Similarly, if I see the centroidal location of second link

with respect to second joint or second frame that again like along x2 only L2 all other axes 0. So, that is what we have derived. So, here in our equation of motion we have written as link length as a1 and a2.

So, in that sense the location of what do you call center of mass of link would be come with the two symbols. So, which is lc1 and lc2. So, here lc1 I consider as a1 and lc2 I consider as a2. So, in that sense so, the Pc1 and Pc2 vector I have derived in this way. So, then what else you need? You need actual like inertial and acceleration variable. So, what we have seen so far is only up to velocity.

So, the inertial effect will come in this case one only two masses but the acceleration effort theta 1 double dot and theta 2 double dot. So that is what we have written in the MATLAB code. But by default, these are all real variable not in complex. So, then you better define that as a real. So, that the conjugate term will not come when you square or do some operation. So, once these are all defined what we need to do you need to take the angular acceleration then linear acceleration then get the inertial forces and moments.

(Refer Slide Time: 4:55)

Matlab code

For a Rotary Joint:

$${}^{i+1}\alpha = {}^i\alpha + {}^i\omega \times [0 \ 0 \ \dot{\theta}_{i+1}]^T + [0 \ 0 \ \ddot{\theta}_{i+1}]^T \quad (1)$$

```

%% Angular acceleration vectors
a10 = [0;0;0];
a11 = R01'*(a10 + cross(w0,[0;0;th1dot])) + [0;0;th1ddot];
a12 = R12'*(a11 + cross(w1,[0;0;th2dot])) + [0;0;th2ddot];
a13 = R23'*(a12 );

```

SANJAY KUMAR MOHAN, IIT PALAKKAD  
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, that is what we are trying to do. So, now the angular acceleration can be written in this form because this is having only rotary joint the rotary joint angular acceleration given in this relation. So, we are writing the same thing in MATLAB the same form. Because the angular acceleration

at the zeroth frame we assume it is fixed. So, it would be getting 0 0 0. Then we are actually propagating the angular acceleration we call  $\alpha$  which is alpha equivalent.

So, alpha 0 we know then alpha 1 we can obtain with the help of this relation, the same relation we have written it in MATLAB syntax. So, similarly alpha 2 we can get it, So, alpha 3 there is no active variable so, you can see that the cross terms are vanished. So, this is what alpha 0 to alpha 3. So, now coming to the linear acceleration.

(Refer Slide Time: 5:45)

Matlab code  
○○●○○○

For a Rotary Joint:

$${}^{i+1}\mathbf{a} = {}^{i+1}\mathbf{R} \left( {}^i\mathbf{a} + {}^i\boldsymbol{\alpha} \times {}^i\mathbf{P} + {}^i\boldsymbol{\omega} \times ({}^i\boldsymbol{\omega} \times {}^i\mathbf{P}) \right) \quad (2)$$

```

%% Linear acceleration vectors
a0 = [0;g;0];
a1 = R01'*(a0+cross(a0,P01)+cross(w0,cross(w0,P01)));
a2 = R12'*(a1+cross(a1,P12)+cross(w1,cross(w1,P12)));
a3 = R23'*(a2+cross(a2,P23)+cross(w2,cross(w2,P23)));
+

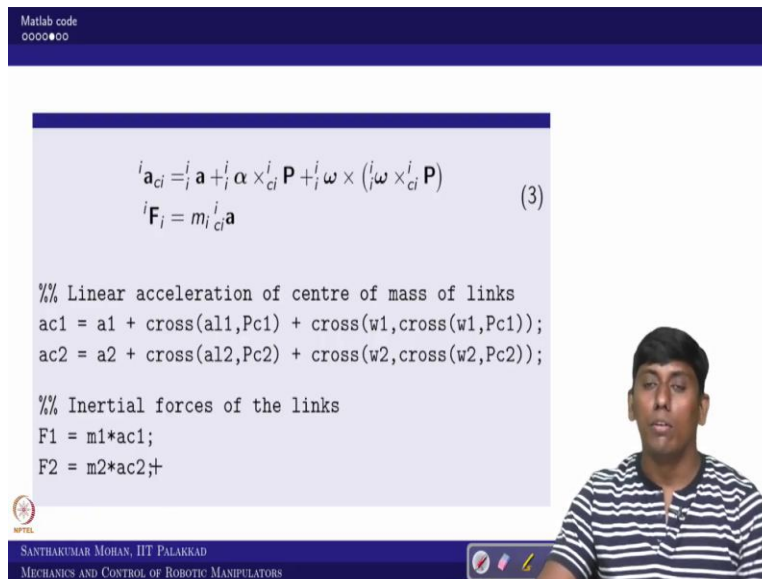
```

SANTHAKUMAR MORAN, IIT PALAKKAD  
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, we know the rotary joint linear acceleration relation. So, we should be having you can say tangential acceleration then the radial acceleration and the previous joint you can see linear acceleration. So, in that case so, we can derive this equation in MATLAB, the same form. So,  $a_{00}$  we know which is again start from 0. But if you assume that this is a vertical manipulator, then the gravity direction you need to mention.

So, I assume that the  $g$  is the gravity that is acting in  $y$  axis. So, in that sense you can see so,  $a_0$  is no longer 0 vector so, 0  $g$  and 0. So, now based on this equation I can derive you can say the joint one linear acceleration second and the third one so, all linear acceleration I derived. So, now, what we need to know? We need to know  $a_{1c}$  in the sense the linear acceleration now, center of mass of link 1 and center of mass linked 2. So, we call  $a_{1c}$  and  $a_{2c}$ .

(Refer Slide Time: 6:47)



Matlab code  
○○○○●○○

$${}^i \mathbf{a}_{ci} = {}^j \mathbf{a} + {}^i \boldsymbol{\alpha} \times {}^i \mathbf{P} + {}^i \boldsymbol{\omega} \times ({}^i \boldsymbol{\omega} \times {}^i \mathbf{P}) \quad (3)$$
$${}^i \mathbf{F}_i = m_i {}^i \mathbf{a}$$

```
%% Linear acceleration of centre of mass of links  
ac1 = a1 + cross(al1,Pc1) + cross(w1,cross(w1,Pc1));  
ac2 = a2 + cross(al2,Pc2) + cross(w2,cross(w2,Pc2));  
  
%% Inertial forces of the links  
F1 = m1*ac1;  
F2 = m2*ac2;+
```

NPTEL  
SANTHAKUMAR MOHAN, IIT PALAKKAD  
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, this can be derived in this form. So, for inertial force you need the centroidal acceleration. So, we have derived the  $ac1$  and  $ac2$ . Which is based on this equation we have written in the general syntax. So, now you can see that this  $\omega_1$  multiply with or you can say cross multiply with  $Pc1$  and  $Pc2$ . Similarly,  $\alpha_1$  cross multiply with  $Pc1$  and  $Pc2$  in the sense this is going to give a tangential, this is going to give a radial and this is the joint linear acceleration.

So, in this sense it is very clear. So, now, once you obtain the linear acceleration of the center of mass of links. So, then what you can do? You can multiply with the inertia. So, here only mass so that would give the inertial force. So, that we have obtained. So, now till this what we have done is the forward propagation. So, now, we will come back to the backward propagation.


(Refer Slide Time: 7:43)

Matlab code  
○○○○○○●

```
%% End-effector forces and moments
f3 = [0;0;0];
n3 = [0;0;0];


$${}^i f = {}^i R_{i+1}^{i+1} f + {}^i F_i \quad (4)$$


%% Joint forces
f2 = R23 * f3 + F2;
f1 = R12 * f2 + F1;
f0 = R01 * f1;
```



SANTHAKUMAR MOHAN, IIT PALAKKAD  
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS


Where we assume the end effector forces and moments are 0's just for simplicity. But later on, you can consider some vector also. But right now, we consider these all 0. So, then you can see this is the equation to back propagate once you know the end effector velocity and the inertial forces then you can back propagate. So, now we have written this equation.

(Refer Slide Time: 8:08)

Matlab code  
○○○○○○●

```
%% Joint moments
n2 = R23 * n3 + cross(Pc2,F2) + cross(P23,R23*f3);
n1 = R12 * n2 + cross(Pc1,F1) + cross(P12,R12*f2);
n0 = R01 * n1 + cross(P01,R01*f1);

%% Vector of inputs
tau1 = simplify(n1(3));
tau2 = simplify(n2(3));
```



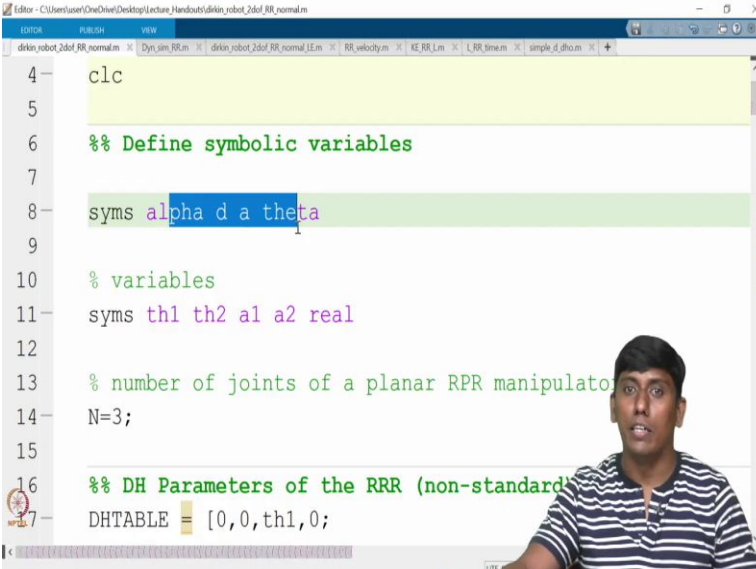
SANTHAKUMAR MOHAN, IIT PALAKKAD  
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, the similar way we can do the back backward propagation for the moments. So, you can do it again you can do  $n_2$  to  $n_1$  to 1. So, that you can derive so, in a sense you can do  $n_2$   $n_1$ . And if you want to have a what you call shaking force and moment. Then you can do it up to you can

say 0. This is equal to shaking forces and this is equal into shaking moments. So, once you all obtained what you required. You required the joint torque relation.

So, for that we will use this. So, the tau 1 is since the first joint and second joint as a rotary joint. So, the n of the third term would be equal into that joint torque. So, now, we are talking about tau 1. So, then n1 of the third term would be the tau 1 you can make a simplification it will give an equation. And similarly, tau 2 is n2 of third term. In the sense, z axis term would be equal to tau 2 this is very clear. So, now, we will move to MATLAB code original MATLAB code which we have returned okay.

(Refer Slide Time: 9:17)



```
4 clc
5
6 %% Define symbolic variables
7
8 syms alpha d a theta
9
10 % variables
11 syms th1 th2 a1 a2 real
12
13 % number of joints of a planar RPR manipulator
14 N=3;
15
16 %% DH Parameters of the RRR (non-standard)
17 DHTABLE = [0,0,th1,0;
```

So, now, this is the MATLAB code which we have returned earlier. So, this is the direct kinematic relation. So, we have taken as actual like symbols which is having a generalized symbol. Then based on your DH parameter we have derived the symbol. And since it is a 2R serial manipulator the end effector frame is the third frame. So, you have excluded 0. So, then you have total number of frames is 3.



(Refer Slide Time: 9:43)

```
Editor - C:\Users\UserOne\Desktop\Lecture_Handouts\dikin_robot_2dof_RR_normal.m
EDITOR PUBLISH VIEW
dikin_robot_2dof_RR_normal.m x Dyn_sim_RR.m x dikin_robot_2dof_RR_normal.LE.m x RR_velocity.m x KE_RR_Lm x L_RR_time.m x simple_d_dho.m x +
16 %% DH Parameters of the RRR (non-standard) Manipulator
17 DHTABLE = [0,0,th1,0;
18           0,a1,th2,0;
19           0,a2,0,0;]
20
21 %% The general Denavit-Hartenberg transformation matrix
22
23 TDH = [ cos(theta)          -sin(theta)
24         sin(theta)*cos(alpha)  cos(theta)*cos(alpha)
25         sin(theta)*sin(alpha)  cos(theta)*sin(alpha)
26         0                    0
27
28 %% Build transformation matrices for each link/joint
29 % First, we create an empty cell array
```

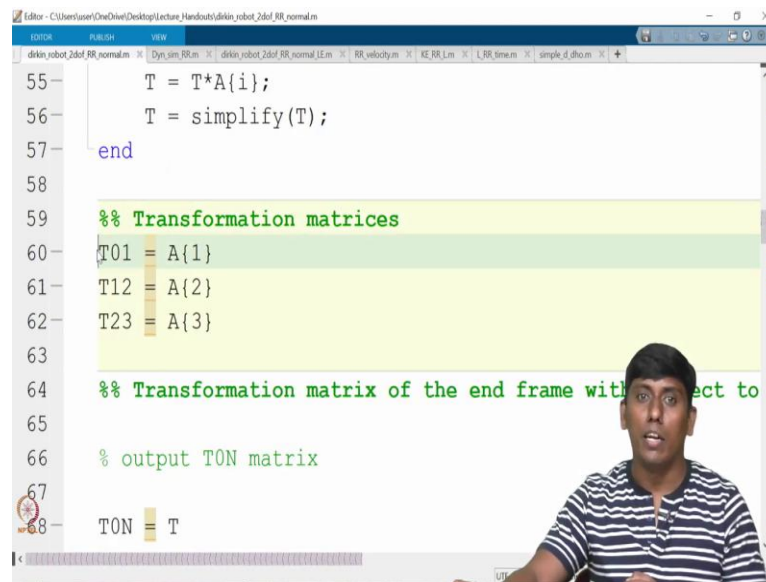
And you can give your DH table as per the derivation earlier.

(Refer Slide Time: 9:47)

```
Editor - C:\Users\UserOne\Desktop\Lecture_Handouts\dikin_robot_2dof_RR_normal.m
EDITOR PUBLISH VIEW
dikin_robot_2dof_RR_normal.m x Dyn_sim_RR.m x dikin_robot_2dof_RR_normal.LE.m x RR_velocity.m x KE_RR_Lm x L_RR_time.m x simple_d_dho.m x +
19 a2,0,0;]
20
21 % Denavit-Hartenberg transformation matrix
22
23 theta)          -sin(theta)          0
24 theta)*cos(alpha)  cos(theta)*cos(alpha)  -sin(alpha)
25 theta)*sin(alpha)  cos(theta)*sin(alpha)  cos(alpha)
26                 0                    0
27
28 % Build transformation matrices for each link/joint
29 % First, we create an empty cell array
30
31
32
```

Then you know the arm matrix, I hope that arm matrix is nonstandard form which we derived in the beginning of the DH representation.

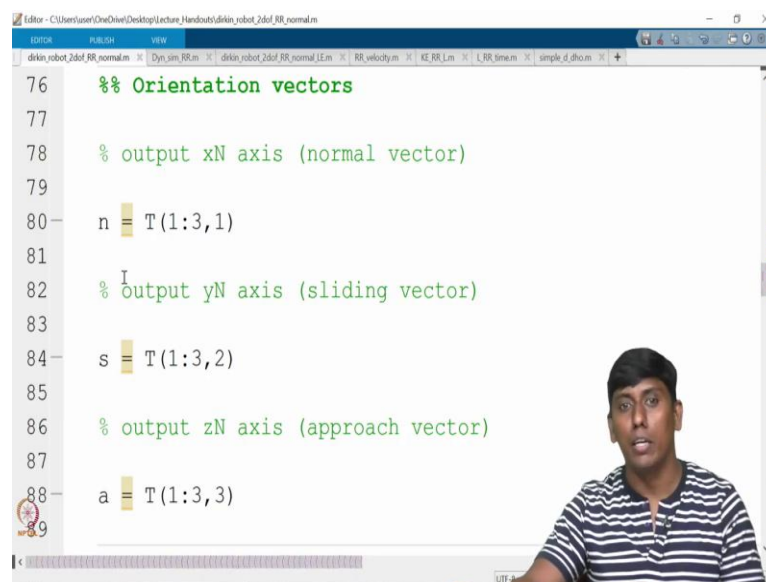
(Refer Slide Time: 9:55)



```
55     T = T*A{i};
56     T = simplify(T);
57 end
58
59 %% Transformation matrices
60 T01 = A{1}
61 T12 = A{2}
62 T23 = A{3}
63
64 %% Transformation matrix of the end frame with respect to
65
66 % output TON matrix
67
68 TON = T
```

Then as usually the cell and then you have derived, and you can get up to you call the transformation matrix. Once you are obtain the transformation matrix. What we have done?

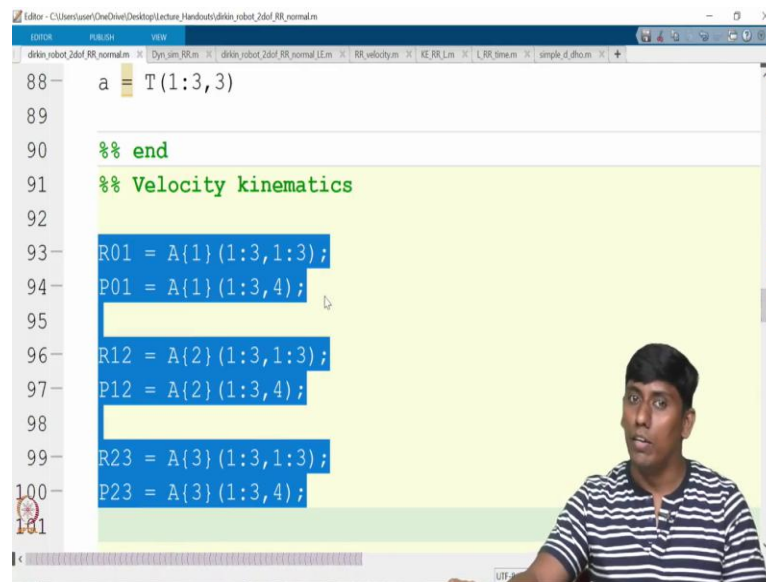
(Refer Slide Time: 10:06)



```
76 %% Orientation vectors
77
78 % output xN axis (normal vector)
79
80 n = T(1:3,1)
81
82 % output yN axis (sliding vector)
83
84 s = T(1:3,2)
85
86 % output zN axis (approach vector)
87
88 a = T(1:3,3)
```

We have tried to see the kinematic model this is what we have seen in the direct kinematic method or direct kinematic code.

(Refer Slide Time: 10:13)

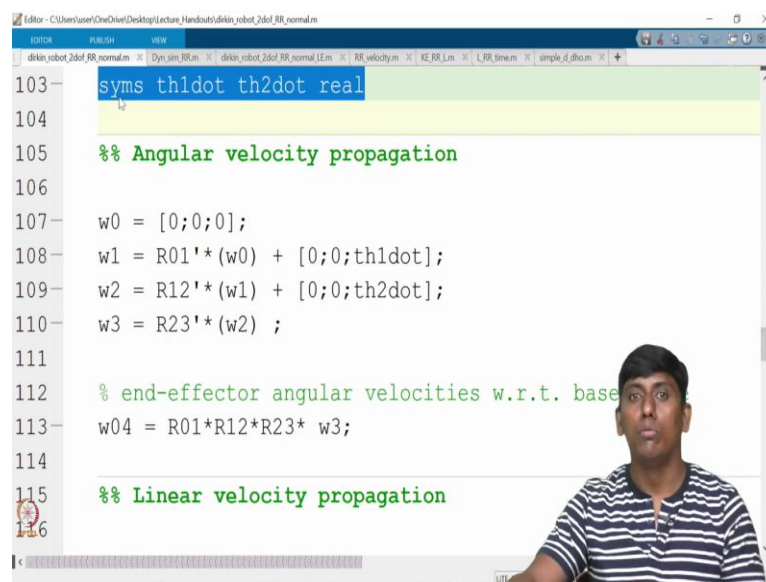
A screenshot of a MATLAB code editor window. The code is as follows:

```
88 a = T(1:3,3)
89
90 %% end
91 %% Velocity kinematics
92
93 R01 = A{1}(1:3,1:3);
94 P01 = A{1}(1:3,4);
95
96 R12 = A{2}(1:3,1:3);
97 P12 = A{2}(1:3,4);
98
99 R23 = A{3}(1:3,1:3);
100 P23 = A{3}(1:3,4);
101
```

The code defines a transformation matrix 'a' and then sets up velocity kinematics by defining rotational matrices (R01, R12, R23) and position vectors (P01, P12, P23) for three joints. The code is highlighted in blue and green. A person's head and shoulders are visible in the bottom right corner of the editor window.

So, after that we have tried to do the velocity kinematics for that we need to know the; you can say rotational matrices and as well as position vectors we have taken.

(Refer Slide Time: 10:23)

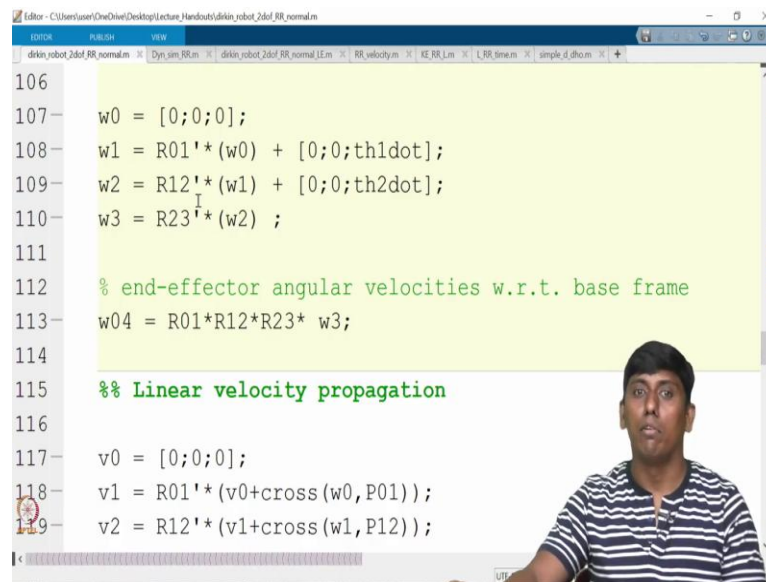
A screenshot of a MATLAB code editor window. The code is as follows:

```
103 syms th1dot th2dot real
104
105 %% Angular velocity propagation
106
107 w0 = [0;0;0];
108 w1 = R01'*w0 + [0;0;th1dot];
109 w2 = R12'*w1 + [0;0;th2dot];
110 w3 = R23'*w2 ;
111
112 % end-effector angular velocities w.r.t. base
113 w04 = R01*R12*R23* w3;
114
115 %% Linear velocity propagation
116
```

The code defines symbolic variables for angular velocities and then calculates the propagation of angular velocities through the joints. It also defines the end-effector angular velocities. The code is highlighted in blue and green. A person's head and shoulders are visible in the bottom right corner of the editor window.

And we have started understanding that for propagation, you need the velocity information. So, here there are two variables theta 1 dot and theta 2 dot so that we have derived.

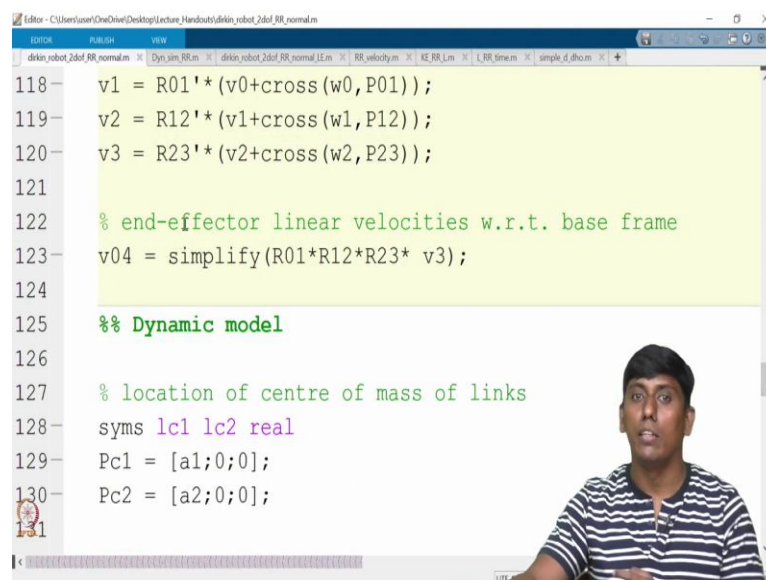
(Refer Slide Time: 10:35)



```
106
107 w0 = [0;0;0];
108 w1 = R01'*(w0) + [0;0;th1dot];
109 w2 = R12'*(w1) + [0;0;th2dot];
110 w3 = R23'*(w2) ;
111
112 % end-effector angular velocities w.r.t. base frame
113 w04 = R01*R12*R23* w3;
114
115 %% Linear velocity propagation
116
117 v0 = [0;0;0];
118 v1 = R01'*(v0+cross(w0,P01));
119 v2 = R12'*(v1+cross(w1,P12));
```

So, then we have propagated based on the; you can see the velocity propagation model. So, initially we have done the angular velocity then end effector velocity we obtained.

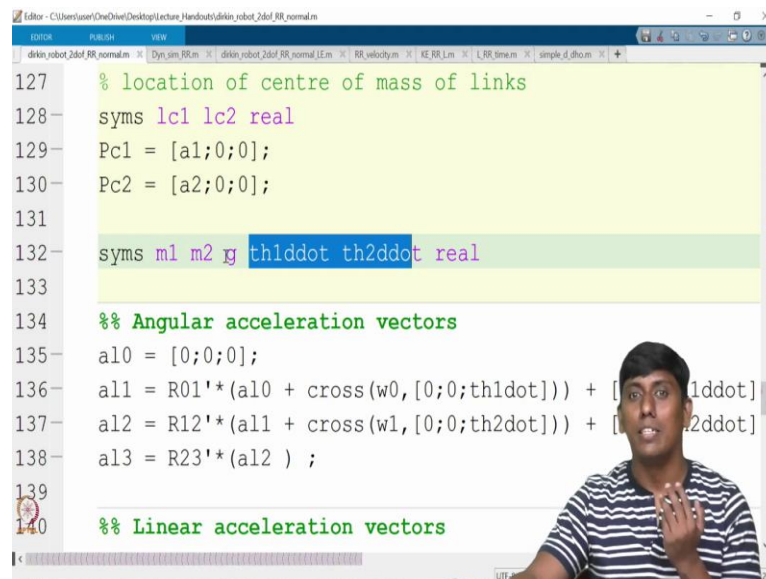
(Refer Slide Time: 10:44)



```
118 v1 = R01'*(v0+cross(w0,P01));
119 v2 = R12'*(v1+cross(w1,P12));
120 v3 = R23'*(v2+cross(w2,P23));
121
122 % end-effector linear velocities w.r.t. base frame
123 v04 = simplify(R01*R12*R23* v3);
124
125 %% Dynamic model
126
127 % location of centre of mass of links
128 syms lc1 lc2 real
129 Pc1 = [a1;0;0];
130 Pc2 = [a2;0;0];
```

Then we have done the linear velocity propagation and the linear velocity of the end effector we have derived.

(Refer Slide Time: 10:50)



```
127 % location of centre of mass of links
128 syms lc1 lc2 real
129 Pc1 = [a1;0;0];
130 Pc2 = [a2;0;0];
131
132 syms m1 m2 g th1ddot th2ddot real
133
134 %% Angular acceleration vectors
135 a10 = [0;0;0];
136 a11 = R01'*(a10 + cross(w0,[0;0;th1dot])) + [0;0;1ddot]
137 a12 = R12'*(a11 + cross(w1,[0;0;th2dot])) + [0;0;2ddot]
138 a13 = R23'*(a12 );
139
140 %% Linear acceleration vectors
```

Then we are coming to the dynamic model. For the dynamic model what is the first step we did we did the center of mass or you can say location of center of mass so, that we have already derived. So, that equation or that vector we have substituted here. So, right now, this you can say definition is not required. But if you want you can even write  $lc1 \times lc1 \ y \ lc1 \ z$ . Similarly, you can write in that so, these are all you can write it in here.

If you know the exact you are what you call model solid model and you know the exact location of the center of mass, you can do it right now we have taken a line diagram. So, based on that the  $Pc1$  and  $Pc2$  return in this form. Then we are going for a dynamic model. So, the gravity is one additional acceleration. Then you have two joint acceleration and there are two masses. So, here we consider only point mass that is why the inertial value is not coming or you can say second moment of inertia and product of inertia are not coming here.

(Refer Slide Time: 11:53)

```
133
134 %% Angular acceleration vectors
135 a10 = [0;0;0];
136 a11 = R01'*(a10 + cross(w0,[0;0;th1dot])) + [0;0;th1ddot]
137 a12 = R12'*(a11 + cross(w1,[0;0;th2dot])) + [0;0;th2ddot]
138 a13 = R23'*(a12);
139
140 %% Linear acceleration vectors
141 a0 = [0;g;0];
142 a1 = R01'*(a0+cross(a10,P01)+cross(w0,cross(w0,P01)));
143 a2 = R12'*(a1+cross(a11,P12)+cross(w1,cross(w1,P12)));
144 a3 = R23'*(a2+cross(a12,P23)+cross(w2,cross(w2,P23)));
145
146 %% Linear acceleration of centre of mass
```

So, then we are doing the angular acceleration vector. So, where we can say alpha 0 to alpha 3, we have used and derived.

(Refer Slide Time: 12:02)


```
145
146 %% Linear acceleration of centre of mass of links
147 ac1 = a1 + cross(a11,Pc1) + cross(w1,cross(w1,Pc1));
148 ac2 = a2 + cross(a12,Pc2) + cross(w2,cross(w2,Pc2));
149
150 %% Inertial forces of the links
151 F1 = m1*ac1;
152 F2 = m2*ac2;
153 I
154 %% End-effector forces and moments
155 f3 = [0;0;0];
156 n3 = [0;0;0];
157
158 %% Joint forces
```

Then you can see the linear acceleration we assume that the gravity pulled vertically act down on the y axis. So, that is what we have substituted. So, in that sense you can get the linear acceleration vector in a propagated model. Then we have calculated the linear acceleration of the center of mass.



(Refer Slide Time: 12:22)


```
editor - C:\Users\user\OneDrive\Desktop\Lecture_Handouts\dikin_robot_2dof_RR_normal.m
EDITOR PUBLISH VIEW
dikin_robot_2dof_RR_normal.m | Dyn_sim_RR.m | diKin_robot_2dof_RR_normal.LE.m | RR_velocity.m | KE_RR_Lm | L_RR_time.m | simple_d_dho.m | +
148- ac2 = a2 + cross(a12,Pc2) + cross(w2,cross(w2,Pc2));
149
150 %% Inertial forces of the links
151 F1 = m1*ac1;
152 F2 = m2*ac2;
153
154 %% End-effector forces and moments
155 f3 = [0;0;0];
156 n3 = [0;0;0];
157
158 %% Joint forces
159 f2 = R23 * f3 + F2;
160 f1 = R12 * f2 + F1;
161 f0 = R01 * f1;
```



Then we have calculated the inertial forces of the link.

(Refer Slide Time: 12:26)


```
editor - C:\Users\user\OneDrive\Desktop\Lecture_Handouts\dikin_robot_2dof_RR_normal.m
EDITOR PUBLISH VIEW
dikin_robot_2dof_RR_normal.m | Dyn_sim_RR.m | diKin_robot_2dof_RR_normal.LE.m | RR_velocity.m | KE_RR_Lm | L_RR_time.m | simple_d_dho.m | +
139
140 %% Linear acceleration vectors
141 a0 = [0;g;0];
142 a1 = R01*(a0+cross(a10,P01)+cross(w0,cross(w0,P01)));
143 a2 = R12*(a1+cross(a11,P12)+cross(w1,cross(w1,P12)));
144 a3 = R23*(a2+cross(a12,P23)+cross(w2,cross(w2,P23)));
145
146 %% Linear acceleration of centre of mass of links
147 ac1 = a1 + cross(a11,Pc1) + cross(w1,cross(w1,Pc1));
148 ac2 = a2 + cross(a12,Pc2) + cross(w2,cross(w2,Pc2));
149
150 %% Inertial forces of the links
151 F1 = m1*ac1;
152 F2 = m2*ac2;
```



So, far what we have calculated these all-linear acceleration vector of joints. And this is linear acceleration of center of mass here link has come.

(Refer Slide Time: 12:34)


```
editor - C:\Users\user\OneDrive\Desktop\Lecture_Handout\ddkin_robot_2dof_RR_normal.m
EDITOR PUBLISH VIEW
ddkin_robot_2dof_RR_normal.m | Dyn_sim_RR.m | dkin_robot_2dof_RR_normal.LE.m | RR_velocity.m | KE_RR_Lm | L_RR_time.m | simple_d_gho.m | +
145
146 %% Linear acceleration of centre of mass of links
147 ac1 = a1 + cross(a11,Pc1) + cross(w1,cross(w1,Pc1));
148 ac2 = a2 + cross(a12,Pc2) + cross(w2,cross(w2,Pc2));
149
150 %% Inertial forces of the links
151 F1 = m1*ac1;
152 F2 = m2*ac2;
153
154 %% End-effector forces and moments
155 f3 = [0;0;0];
156 n3 = [0;0;0];
157
158 %% Joint forces
```



So, now, link inertial forces we have calculated. Now, we are coming backward to the joint forces and moment.

(Refer Slide Time: 12:41)

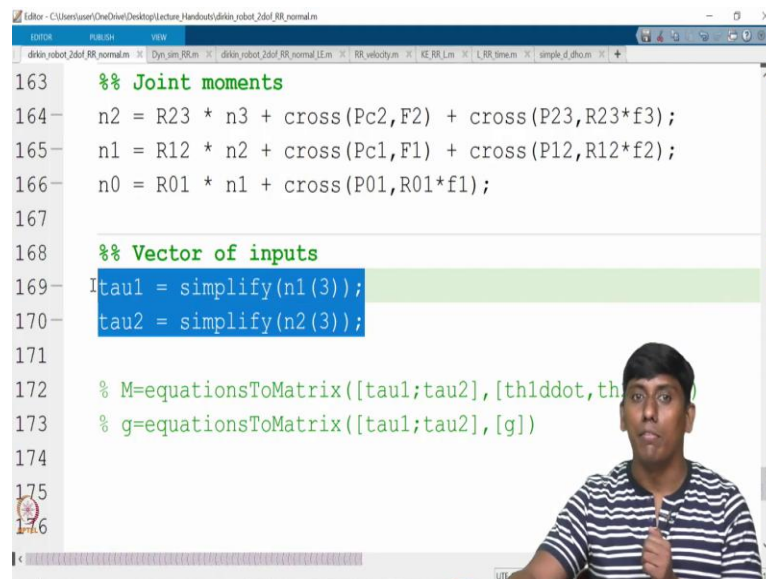
```
editor - C:\Users\user\OneDrive\Desktop\Lecture_Handout\ddkin_robot_2dof_RR_normal.m
EDITOR PUBLISH VIEW
ddkin_robot_2dof_RR_normal.m | Dyn_sim_RR.m | dkin_robot_2dof_RR_normal.LE.m | RR_velocity.m | KE_RR_Lm | L_RR_time.m | simple_d_gho.m | +
151 F1 = m1*ac1;
152 F2 = m2*ac2;
153
154 %% End-effector forces and moments
155 f3 = [0;0;0];
156 n3 = [0;0;0];
157
158 %% Joint forces
159 f2 = R23 * f3 + F2;
160 f1 = R12 * f2 + F1;
161 f0 = R01 * f1;
162
163 %% Joint moments
164 n2 = R23 * n3 + cross(Pc2,F2) + cross(P2,
```



So, that is what we are starting with the end effector then the joint forces and moments we are calculated.



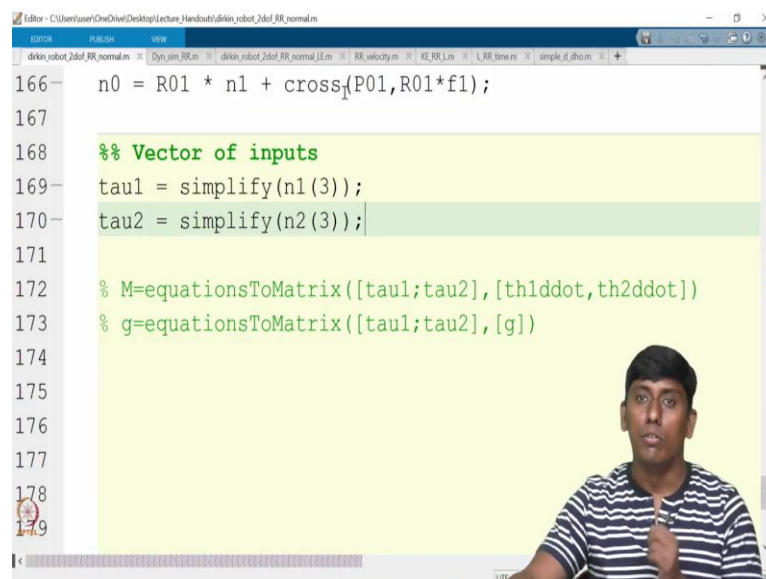
(Refer Slide Time: 12:48)



```
163 %% Joint moments
164 n2 = R23 * n3 + cross(Pc2,F2) + cross(P23,R23*f3);
165 n1 = R12 * n2 + cross(Pc1,F1) + cross(P12,R12*f2);
166 n0 = R01 * n1 + cross(P01,R01*f1);
167
168 %% Vector of inputs
169 tau1 = simplify(n1(3));
170 tau2 = simplify(n2(3));
171
172 % M=equationsToMatrix([tau1;tau2],[th1ddot,th2ddot])
173 % g=equationsToMatrix([tau1;tau2],[g])
174
175
176
```

Then the third element of the joint moment or joint moments that would be equivalent to your input. So, that is what we have derived. Further you want to write it in a matrix form, then you can always bring this you can say.

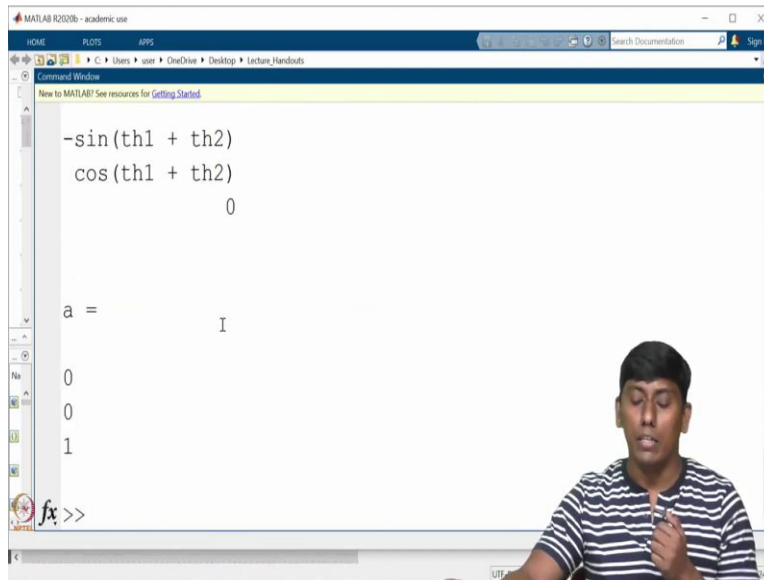
(Refer Slide Time: 13:03)



```
166 n0 = R01 * n1 + cross(P01,R01*f1);
167
168 %% Vector of inputs
169 tau1 = simplify(n1(3));
170 tau2 = simplify(n2(3));
171
172 % M=equationsToMatrix([tau1;tau2],[th1ddot,th2ddot])
173 % g=equationsToMatrix([tau1;tau2],[g])
174
175
176
177
178
179
```

Further command called equation to matrix and then take the coefficient. So, these all can be done. But right now, you can see the tau 1 and tau 2 we you can say derive. So, I will run this code I hope there will not be any error. So, if we run, then it would be giving the output in the MATLAB window.

(Refer Slide Time: 13:22)



The image shows a MATLAB R2020b Command Window with the following code and output:

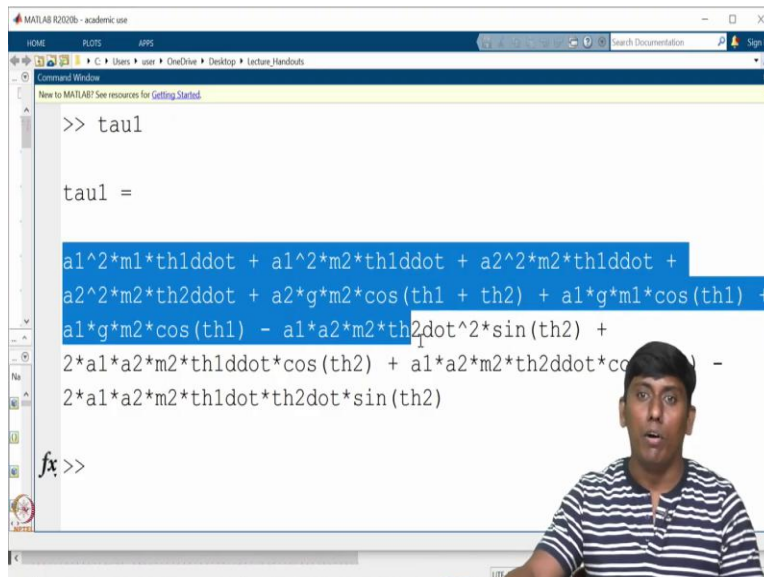
```
-sin(th1 + th2)
cos(th1 + th2)
0

a =
      I
     0
     0
     1

fx >>
```

So, you can see this is the; your what you call you can say the kinematic model.

(Refer Slide Time: 13:33)



The image shows a MATLAB R2020b Command Window with the following code and output:

```
>> tau1

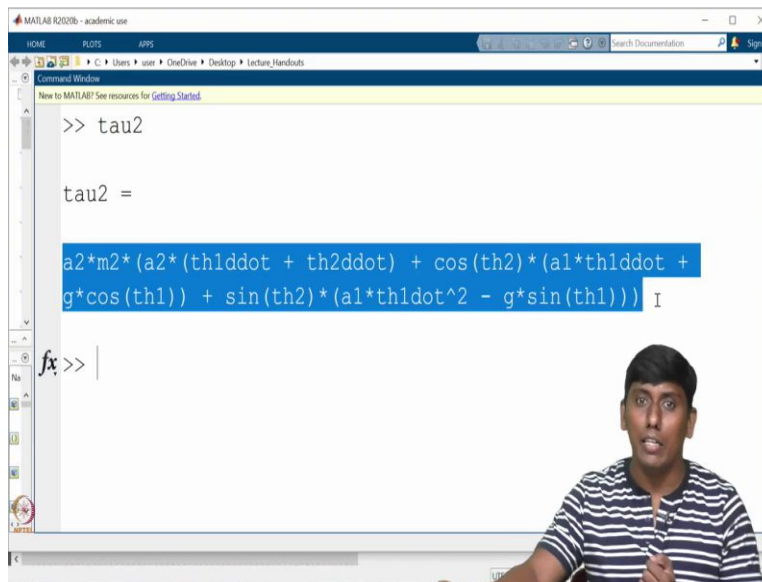
tau1 =

a1^2*m1*th1ddot + a1^2*m2*th1ddot + a2^2*m2*th1ddot +
a2^2*m2*th2ddot + a2*g*m2*cos(th1 + th2) + a1*g*m1*cos(th1) +
a1*g*m2*cos(th1) - a1*a2*m2*th2dot^2*sin(th2) +
2*a1*a2*m2*th1ddot*cos(th2) + a1*a2*m2*th2ddot*cos(th1) -
2*a1*a2*m2*th1dot*th2dot*sin(th2)

fx >>
```

And what right now we are interested is trying to find out what you call tau 1. So, tau 1 is you can see you can write it in the same form. And similarly, you can see tau 2. So, I will just make it.

(Refer Slide Time: 13:46)



The image shows a MATLAB Command Window with the following text:

```
>> tau2

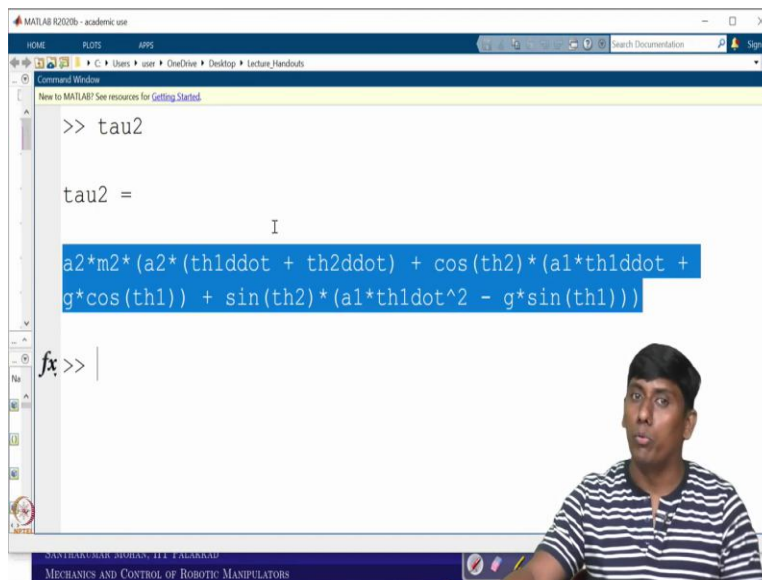
tau2 =

a2*m2*(a2*(th1ddot + th2ddot) + cos(th2)*(a1*th1ddot +
g*cos(th1) + sin(th2)*(a1*th1dot^2 - g*sin(th1))) I
```

A blue highlight covers the equation. A person is visible in the bottom right corner of the window.

So, tau 2 so, now you can even cross verify these equations what we have obtained right now. The same thing what we have obtained here also.

(Refer Slide Time: 14:01)



The image shows a MATLAB Command Window with the following text:

```
>> tau2

tau2 =

I
a2*m2*(a2*(th1ddot + th2ddot) + cos(th2)*(a1*th1ddot +
g*cos(th1) + sin(th2)*(a1*th1dot^2 - g*sin(th1)))
```

A blue highlight covers the equation. A person is visible in the bottom right corner of the window. At the bottom of the window, there is a footer: "MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS".

So, in the sense you if you go back to your equation. So, is whatever you have derived for this particular system. So, the same equation what we are obtaining in the MATLAB output also. In the sense of what the benefit you can use MATLAB for you can say deriving the equation of motion. So, now we have derived this as very simple model. If you want to do it the same thing for more complicated for example there is an inertia.

(Refer Slide Time: 14:38)

```
127 % location of centre of mass of links
128 syms lc1 lc2 real
129 Pc1 = [a1;0;0];
130 Pc2 = [a2;0;0];
131
132 syms m1 m2 g th1ddot Ixx1 Iyy1 Izz1 Ixy1 Ixz1 Iyz1 th2ddot
133 I
134 %% Angular acceleration vectors
135 a10 = [0;0;0];
136 a11 = R01'*(a10 + cross(w0,[0;0;th1dot])) + [0;0;th1ddot]
137 a12 = R12'*(a11 + cross(w1,[0;0;th2dot])) + [0;0;th2ddot]
138 a13 = R23'*(a12 );
139
140 %% Linear acceleration vectors
```

So, for example, I am saying that the first link is having inertia. So, in the sense I am saying that there is only a second moment is there. I say only link so now what I got the second moment of inertia is there. So, now I am saying that the product of inertia also there then Iyz1. So, these all the terms I have included these all real variables.

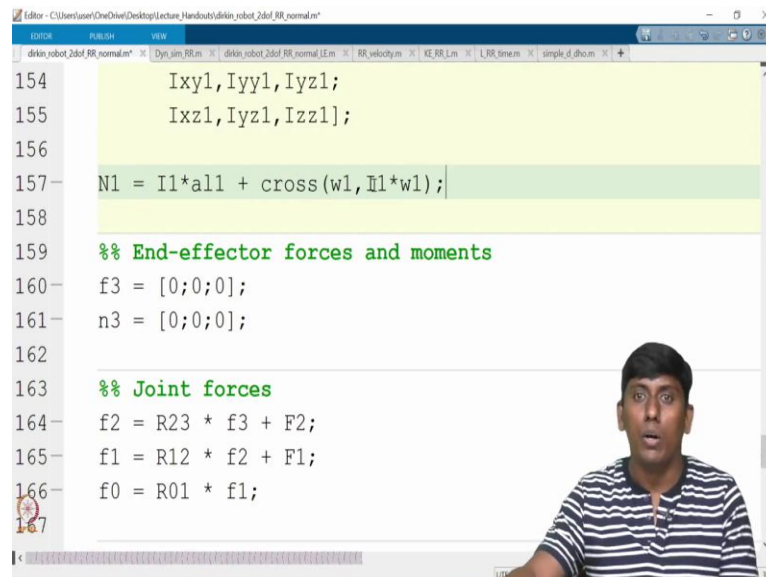
(Refer Slide Time: 15:14)

```
145
146 %% Linear acceleration of centre of mass of links
147 ac1 = a1 + cross(a11,Pc1) + cross(w1,cross(w1,Pc1));
148 ac2 = a2 + cross(a12,Pc2) + cross(w2,cross(w2,Pc2));
149
150 %% Inertial forces of the links
151 F1 = m1*ac1;
152 F2 = m2*ac2;
153 I1 = [Ixx1,Ixy1,Ixz1; I
154       Ixy1,Iyy1,Iyz1;
155       Ixz1,Iyz1,Izz1];
156
157 N1 =
```

So, for what we have taken the only inertial force. But right now, I have inertial moment also. So, in the sense inertial moment is what so, you can write i so, i here is the bigger matrix. So, I can write I is you can write Ixx Ixy and I you can write xz and since it is a 1. So, I will write it 1

in fact we will put a minus sign here that you can get it even in the unit itself. So, now, I am writing this is tensor. So, what we can write I you can write  $I_{xy1}$   $I_{yy}$  and  $I_{yz1}$ . So, then you have you  $I_{xz1}$   $I_{yz1}$  and  $I_{zz1}$ . This is the inertial value. So, now you are N1.

(Refer Slide Time: 16:26)



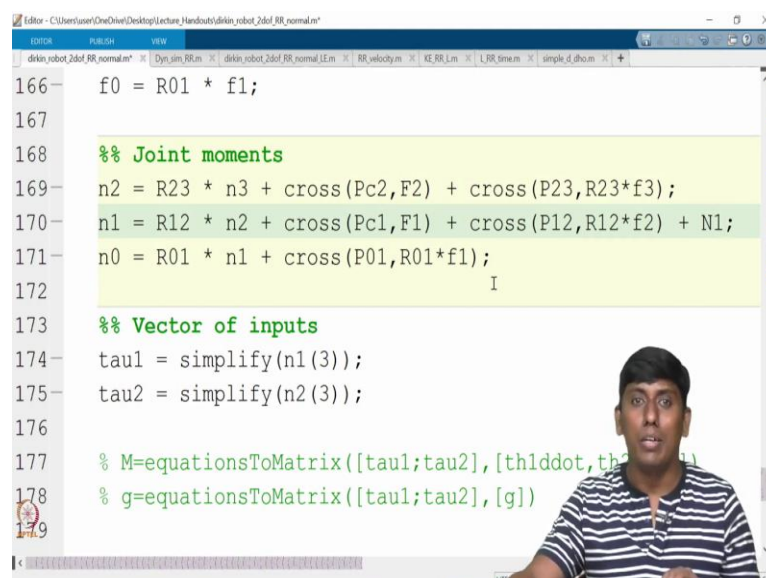
```

154     Ixy1, Iyy1, Iyz1;
155     Ixz1, Iyz1, Izz1];
156
157     N1 = I1*alpha1 + cross(w1, I1*w1);
158
159     %% End-effector forces and moments
160     f3 = [0;0;0];
161     n3 = [0;0;0];
162
163     %% Joint forces
164     f2 = R23 * f3 + F2;
165     f1 = R12 * f2 + F1;
166     f0 = R01 * f1;
167

```

So, your N1 would be I1 multiply with alpha 1. In fact, if you strictly go so, it would be come as you can see omega 1 you can see cross. So, I will write it that itself so cross off omega 1 into multiply with the I 1 omega 1. But anyhow in this case, this will not give any term. But this will be making sense.

(Refer Slide Time: 17:04)



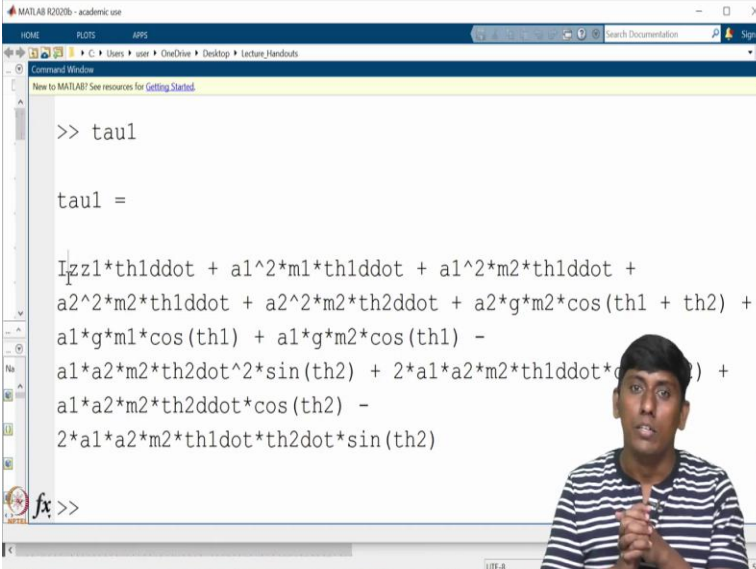
```

166     f0 = R01 * f1;
167
168     %% Joint moments
169     n2 = R23 * n3 + cross(Pc2, F2) + cross(P23, R23*f3);
170     n1 = R12 * n2 + cross(Pc1, F1) + cross(P12, R12*f2) + N1;
171     n0 = R01 * n1 + cross(P01, R01*f1);
172
173     %% Vector of inputs
174     tau1 = simplify(n1(3));
175     tau2 = simplify(n2(3));
176
177     % M=equationsToMatrix([tau1;tau2],[th1ddot,th2ddot]);
178     % g=equationsToMatrix([tau1;tau2],[g])
179

```

So, now in that case so, what happened here? So, in this case, it would be just added as; so, N1. So, if I add this so, what you can see that inertial term all would be coming. So, I just want to run this I hope so, there is no error.

(Refer Slide Time: 17:22)



```

>> tau1

tau1 =

Izz1*th1ddot + a1^2*m1*th1ddot + a1^2*m2*th1ddot +
a2^2*m2*th1ddot + a2^2*m2*th2ddot + a2*g*m2*cos(th1 + th2) +
a1*g*m1*cos(th1) + a1*g*m2*cos(th1) -
a1*a2*m2*th2dot^2*sin(th2) + 2*a1*a2*m2*th1ddot*(th2dot) +
a1*a2*m2*th2ddot*cos(th2) -
2*a1*a2*m2*th1dot*th2dot*sin(th2)
fx >>

```

So, I will assume that there is no error. So, now if I see tau 1 you can see that I is at terms all coming appearing. So, the other product of inertias are not coming because it is in a plane. And you are theta 1 dot theta 1 double dot all in only you can say z axis that is why you can see it is having only Izz.

So, now if you assume that you have a link, that link is having even inertia which is mass not concentrated at one point it is actually distributed mass. Then you have a second moment of inertia that is also included. So, now the similar direction you can go further and further for example, you want even further.

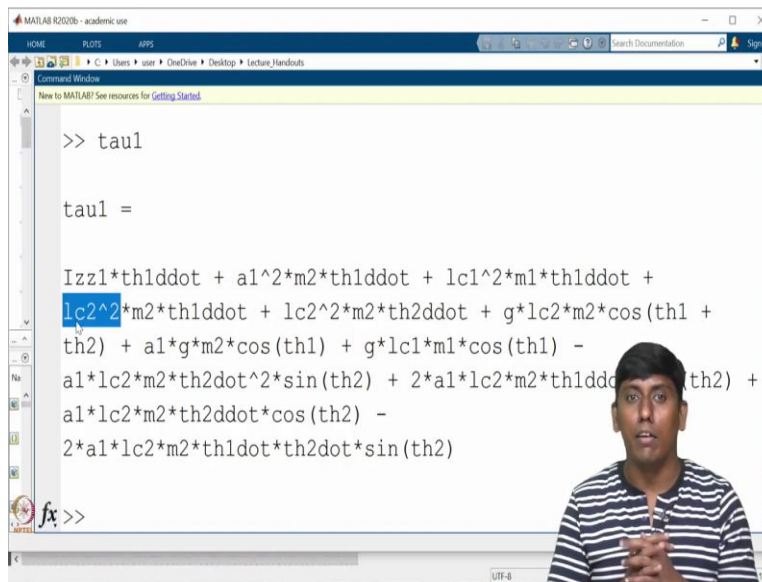


(Refer Slide Time: 18:02)

```
124
125 %% Dynamic model
126
127 % location of centre of mass of links
128 syms lc1 lc2 real
129 Pc1 = [a1;0;0];
130 Pc2 = [a2;0;0];
131
132 syms m1 m2 g th1ddot Ixx1 Iyy1 Izz1 Ixy1 Ixz1 Iyz1 th2ddo
133
134 %% Angular acceleration vectors
135 a10 = [0;0;0];
136 a11 = R01'*(a10 + cross(w0,[0;0;th1dot]));
137 a12 = R12'*(a11 + cross(w1,[0;0;th2dot]));
138
139
140 %% Linear acceleration vectors
```

So, in the sense I want to add even N2 I can add or here instead of; you can say the location. So, this location I assume that this is not a1 the mass is concentrated some location in x axis. So, now you can see this so, it will be changed. So, now I again I am running it. So, now earlier what you can see it is a product.

(Refer Slide Time: 18:30)



```
MATLAB R2020b - academic use
HOME  PLOTS  APPS
C:\Users\user\OneDrive\Desktop\Lecture_Handouts
Command Window
New to MATLAB? See resources for Getting Started.

>> tau1

tau1 =

Izz1*th1ddot + a1^2*m2*th1ddot + lc1^2*m1*th1ddot +
lc2^2*m2*th1ddot + lc2^2*m2*th2ddot + g*lc2*m2*cos(th1 +
th2) + a1*g*m2*cos(th1) + g*lc1*m1*cos(th1) -
a1*lc2*m2*th2dot^2*sin(th2) + 2*a1*lc2*m2*th1ddot*(th2) +
a1*lc2*m2*th2ddot*cos(th2) -
2*a1*lc2*m2*th1dot*th2dot*sin(th2)

fx >>
```

So, now that product is modified you can see. So that  $a_1 c_1$  is coming and  $l_1$  or  $l_2$  is going out. So here you can see it. So, this is the way we can derive the equation of motion in Newton Euler. Newton Euler is easy because everything is in sequence even you can do the algorithm based you can write it in recursive base. But recursive although it is simple number of codes is very small. But I prefer to write it in a lengthy way.

In that way, even if any minor mistake happened that can be rectified. But if you write in a code in algorithm base if a small mistake happened it is very difficult to find. And when you compared to Lagrangian Euler. This is very simple because you no need to back and forth from partial derivative to time derivative. So, anyhow in the next lecture we are going to see the; what you call Lagrangian Euler formulation method using MATLAB.

The same example we will take but right now what you have seen is how to use Newton Euler method in the MATLAB environment and derive the equation of motion. So, probably if time permits in the example class, we will see the same code how we can extend for higher order system. In the sense higher degree of systems so that we will see until then thank you and see you, bye, take care.