

**Mechanics and Control of Robotic Manipulators**  
**Professor Santhakumar Mohan**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Palakkad**  
**Lecture: 43**

**Simulations Related to Dynamic Control Schemes Using MATLAB**

Hi, welcome back to Mechanics and Control of Robotic Manipulator. In the last lecture we have seen cascaded design last to last class we have seen how to simulate you can say dynamic control of you can say robotic manipulator using joint space. So, this particular lecture is like in addition to that, because last lecture or you can say last simulation itself, I said, the task-space I did not include it here.

So, this particular lecture is like towards task-space. So, in the sense inverse dynamics in task-space and the; all motion control which is like which includes a computed torque control or PD or PID. So, these all will be seen. In addition to that we have attempted the cascade which we call integrated backstepping or simply backstepping control or dual-loop control that is also we are going to see in this particular short video.

(Refer Slide Time: 01:05)

The image shows a video frame with a presentation slide. At the top, there are four navigation tabs: 'Inverse dynamics control scheme', 'Computed torque or model-based control', 'Motion-based control', and 'Cascaded control or double-loop control scheme'. The main title of the slide is 'DYNAMIC (MOTION) CONTROL SIMULATIONS IN MATLAB PART 2'. Below the title, there is a numbered list of four topics: 1. Inverse dynamics, 2. Computed torque or model-based control, 3. Motion-based control, and 4. Cascaded control or double-loop control scheme. In the bottom right corner of the slide, there is a small inset video of Professor Santhakumar Mohan, who is wearing a dark blue shirt and has a microphone clipped to his shirt. At the bottom of the slide, there is a footer with the text 'SANTHAKUMAR MOHAN, IIT PALAKKAD' and 'MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS'.

```
%% Inverse dynamic simulation of a 2R planar robot
clear all; close all; clc;
%% Simulation parameters
dt = 0.01; % stepsize
ts = 30; % total simulation time
t = 0:dt:ts; % time span
global a1 a2 m1 m2 g
%% System parameters
m1 = 2; m2 = 1; % link masses
a1 = 0.5; a2 = 0.4; % link lengths
g = 0*9.81; % gravity
```

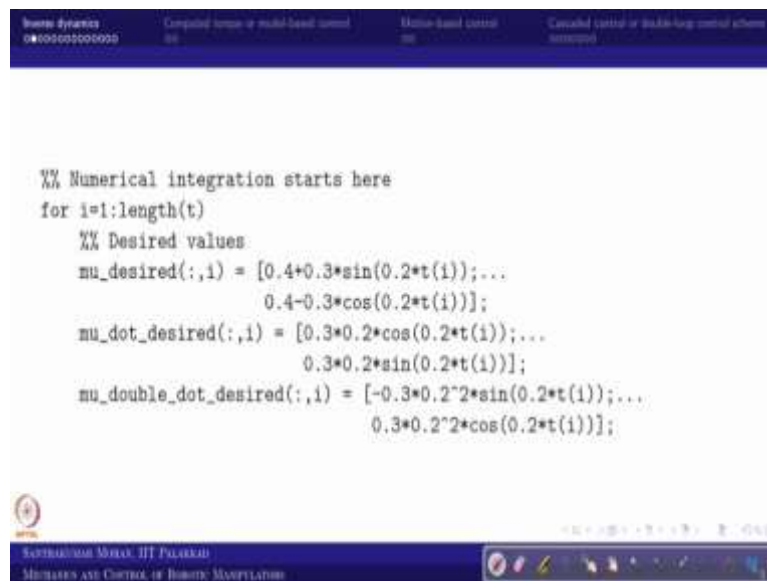
Substitution Model: 2R Planar  
Modeling and Control of Robots: Manipulators

So, in that sense we are like trying to see what is inverse dynamics in task-space. So, then we would be going like how we can modify the computed torque control when we required the task-space variable or the; you can say desired variable. So, similar way we can see the motion-based control which we call like PD and PID control and then we will go to the dual-loop or double-loop or cascaded or backstepping control.

So, in that sense we will like take the same scenario what we have seen in the dynamic control of you can say joint space. So, one only thing here I would be trying to show a small tricky one because, so, there we have seen like if the  $\theta_1$  desired and  $\theta_2$  desired has given us a profile how we can do it? So, instead of this so, I am trying to show us our circular profile in task-space how that would be followed. This is one of the common you can say practices used to do for the control performance of any system.

So, in that sense we are trying to see that way. So, now we will talk about the first thing is inverse dynamic simulation. So, of straight away like we talk about to our serial planar or you can say planar serial manipulator. So, in this sense, we will talk about the Euler integration parameters then the mass of link1, link2 and the link length and the gravity again since we are talking about inverse dynamics, the gravity would be playing a different role. And most of the common manipulators are gravity balance. So, we assume that 0 gravity here so we will come back later on. So, if we make it 0 instead of 0 if we put it some gravity value as 9.81 what would the happen.

(Refer Slide Time: 02:46)



```
%% Numerical integration starts here
for i=1:length(t)
    %% Desired values
    mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));...
                      0.4-0.3*cos(0.2*t(i))];
    mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
                           0.3*0.2*sin(0.2*t(i))];
    mu_double_dot_desired(:,i) = [-0.3*0.2^2*sin(0.2*t(i));...
                                   0.3*0.2^2*cos(0.2*t(i))];
end
```

So, in that case, we are like talking about the circular profile. So, if you look at it here, the link length is like 0.5 and 0.4. So, I am trying to see that 0.4 is the radius both x and y sorry 0.4 by 0.4 is the centre, and the 0.3 radius as a circle I am trying to draw. So, in that sense the mu desired is like coming in this way. In addition to that, it is like 0.2 times of t in the sense it is  $\omega t$  where  $\omega$  is 0.2.

So, now based on that, you can like take a time derivative that would be mu dot desired then further you take the second derivative that would be you can say mu double dot desired. So, these are like we have like use. So, in the last joint space, we have seen how the cubic polynomial can be used, but here even without cubic polynomial, how the circular profile can be generated that is what we have seen here.

(Refer Slide Time: 03:44)

```
%% Desired joint positions and velocities
[q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i),mu_desired(2,i));
Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
%% Initial Conditions
q(:,i) = q_desired(:,i);
q_dot(:,i) = q_dot_desired(:,i);

%% Desired dynamic terms
Md = inertia2R(q_desired(:,i));
oe_vd = other_effects2R(q_desired(:,i),q_dot_desired(:,i));
g_vd = gravity_effects2R(q_desired(:,i));
Jdotd = Jacodot2R(q_desired(:,i),q_dot_desired(:,i));
%% Actual dynamic terms
M = inertia2R(q(:,i));
oe_v = other_effects2R(q(:,i),q_dot(:,i));
g_v = gravity_effects2R(q(:,i));
```

So, further what we can see once this. So, we are trying to find the inverse kinematics. So, that just for our own reference, we tried to find out what would be that  $q$  desired and what would be the  $q$  dot desired. So, once we obtained this, then we are like going back to the initial condition, since the initial condition supposed to be same for the actual desired; actual and desired the initial condition. So, we are like using it this condition.

So, instead of actual like deriving it directly, so whatever the desired, the first value, the first value, we are like equating to the  $q$  and  $q$  dot. So, in the sense you are initial condition defined as here. So, after that, we are going for the dynamic terms since it is inverse dynamics. So, we are trying to find out the desired dynamic term in the sense inertia matrix based on the desired value. Similarly, other effects due to the desired value and gravity due to

the desired value calculated. Similarly the actual value for the simulation in the real time we do not require this but we are doing the simulation, we need to find out the actual inertia matrix other effect and the gravity.

(Refer Slide Time: 04:55)

The slide displays two MATLAB functions. The first function, `inertia2R(q)`, calculates the inertia matrix `M` based on link lengths `a1`, `a2`, masses `m1`, `m2`, and joint angles `th1`, `th2`. The second function, `other_effects2R(q,qdot)`, calculates other effects `oe_v` based on the same parameters and joint velocities `qdot`.

```
function M = inertia2R(q)
global a1 a2 m1 m2
th1 = q(1); th2 = q(2);
% Inertia matrix
m11 = a1^2*m1 + a1^2*m2 + a2^2*m2 + 2*a1*a2*m2*cos(th2);
m21 = a2*m2*(a2 + a1*cos(th2));
m12 = a2*m2*(a2 + a1*cos(th2));
m22 = a2^2*m2;
M=[m11,m12;m21,m22];
end
```

```
function oe_v = other_effects2R(q,qdot)
global a1 a2 m1 m2
th1 = q(1); th2 = q(2);
th1dot = qdot(1); th2dot = qdot(2);
%other effects
oe_v1 = -a1*a2*m2*th2dot*sin(th2)*(2*th1dot + th2dot);
oe_v2 = a1*a2*m2*th1dot^2*sin(th2);
oe_v=[oe_v1;oe_v2];
end
```

Navigation icons: back, forward, search, etc.



System dynamics    Computed torque or model-based control    Holistic-based control    Classical control or multi-loop control systems

```

function [th1,th2] = IK2R(x,y)
global a1 a2
c2 = (x^2+y^2-a1^2-a2^2)/(2*a1*a2);
s2 = sqrt(1-c2^2);
th1 = atan2(y,x)-atan2(a2*s2,a1+a2*c2);
th2 = atan2(s2,c2);
end

```



ROBINDER KUMAR, IIT PALAKHUR  
MECHANICAL CONTROL OF ROBOTS: MANIPULATORS

System dynamics    Computed torque or model-based control    Holistic-based control    Classical control or multi-loop control systems

```

function Jdot = Jacodot2R(q,qdot)
global a1 a2
th1 = q(1); th2 = q(2);
th1dot = qdot(1); th2dot = qdot(2);
j1 = -a1*cos(th1)*th1dot;
j2 = -a2*cos(th1+th2)*(th1dot+th2dot);
j3 = -a1*sin(th1)*th1dot;
j4 = -a2*sin(th1+th2)*(th1dot+th2dot);
Jdot = [j1+j2, j2;
        j3+j4, j4];
end

```



ROBINDER KUMAR, IIT PALAKHUR  
MECHANICAL CONTROL OF ROBOTS: MANIPULATORS

Now, based on this we can like use the, what you call the sub functions. So, we can use the inertia sub function then the other effects of sub function, the gravity effects sub function these all we have derived in the last you can say class itself. So, then this is the Jacobian because here Jacobian is required forward kinematics and inverse kinematics are required. So, we can use it this.

So, based on that we come back what you call one additional variable which is J dot because like, so, q double dot desired can be derived in that way for example, you take mu dot equal to J of q into q dot so then you can see like mu double dot can be returned as so, J of q into q double dot plus J dot into q dot since this is the J dot is required. So, we have derived the J dot based on the time derivative and then like we got this additional sub function.

(Refer Slide Time: 05:50)

The image shows two slides from a presentation. The top slide displays MATLAB code for an inverse dynamics model. The code starts with a comment '% Inverse dynamics' and defines an input vector in task-space. It then calculates the force vector F in task-space as  $F(:,i) = M_{\mu_d} * (\mu_{double\_dot\_desired}(:,i)) + n_{\mu_d}$ . This force is then converted to joint-space torques tau as  $\tau(:,i) = Jd' * F(:,i)$ . The bottom slide shows MATLAB code for forward kinematics. It starts with an acceleration vector  $q_{double\_dot}(:,i) = \text{inv}(M) * (\tau(:,i) - (oe_v + g_v))$ . It then propagates velocity and updates position using  $q(:,i+1) = q(:,i) + q_{dot}(:,i) * dt + 1/2 * q_{double\_dot}(:,i) * dt^2$ . Finally, it calculates the end effector position  $[x(i), y(i)] = \text{FK2R}(q(1,i), q(2,i))$ .

Once we did the sub function this is the inverse dynamic model. So, where M of mu into mu desired double dot, plus n of mu comma mu like a mu desired comma mu desired dot. So, this is actual like we use as a F, but like F we can write as tau. So, and this is tau is J transpose of F that is what we are going to use.

So, in this sense this is the inverse dynamics. So, where first we calculate the; you can say the end effector forces in moments based on the known model and this is the; you can say inverse dynamic you can say model in input vector in task-space that we convert it into joint-space because so, the original system would be derived based on the joint-space because the actuator fixed on the joint. So, then we derived this So, we are like trying to use the simple



you can say Euler integration where the velocity propagation the position update and then once you update then we will do the forward kinematics just for animation.

(Refer Slide Time: 06:55)

```
1  %% Double-loop motion control of a RR planar robot
2  clear all; close all; clc;
3  %% Simulation parameters
4  dt = 0.01; % stepsize
5  ts = 30; % total simulation time
6  t = 0:dt:ts; % time span
7  global a1 a2 m1 m2 g
8  %% System parameters
9  m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11 g = 9.81; % gravity
12 %% Initial conditions
13 q = [-pi/4;pi/3]; % initial joint positions
14 q_dot = [0;0]; % initial joint velocities
```

```
22 mu_double_dot_desired(:,i) = [-0.3*0.2^2*sin(0.2*t(i))
23                               0.3*0.2^2*cos(0.2*t(i))];
24 %% Desired joint positions and velocities
25 [q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i),mu_double_dot_desired(2,i));
26 Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
27 q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
28 %% Initial Conditions
29 q(:,1) = q_desired(:,1);
30 q_dot(:,1) = q_dot_desired(:,1);
31 %% Desired dynamic terms
32 Md = inertia2R(q_desired(:,i));
33 oe_vd = other_effects2R(q_desired(:,i));
34 g_vd = gravity_effects2R(q_desired(:,i));
35 Jdotd = Jacodot2R(q_desired(:,i));
```

```

16- for i=1:length(t)
17-     %% Desired values
18-     mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));...
19-                       0.4-0.3*cos(0.2*t(i))];
20-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
21-                            0.3*0.2*sin(0.2*t(i))];
22-     mu_double_dot_desired(:,i) = [-0.3*0.2^2*cos(0.2*t(i))
23-                                    0.3*0.2^2*sin(0.2*t(i))];
24-     %% Desired joint positions and velocities
25-     [q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i),
26-     Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
27-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
28-     %% Initial Conditions
29-     q(:,1) = q_desired(:,1);

```

```

25-     [q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i),
26-     Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
27-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
28-     %% Initial Conditions
29-     q(:,1) = q_desired(:,1);
30-     q_dot(:,1) = q_dot_desired(:,1);
31-     %% Desired dynamic terms
32-     Md = inertia2R(q_desired(:,i));
33-     oe_vd = other_effects2R(q_desired(:,i),mu_desired(
34-     g_vd = gravity_effects2R(q_desired(:,i),mu_desired(
35-     Jdotd = Jacodot2R(q_desired(:,i),q_dot_desired(:,i));
36-     % inertia matrix in task-space
37-     M_mu_d = inv(Jd')*Md*inv(Jd);
38-     % other effects in task-space

```

```

40-     -Md*inv(Jd)*Jdotd*q_dot_desired(:,i));
41-     %% Actual dynamic terms
42-     M = inertia2R(q(:,i));
43-     oe_v = other_effects2R(q(:,i),q_dot(:,i));
44-     g_v = gravity_effects2R(q(:,i));
45-     %% Inverse dynamics
46-     %input vector in task-space
47-     F(:,i) = M_mu_d*(mu_double_dot_desired(:,i)
48-               + n_mu_d);
49-     % input vector in joint-space
50-     tau(:,i) = Jd'*F(:,i);
51-     % acceleration vector
52-     q_double_dot(:,i) = inv(M)*(tau(:,i)
53-

```

```

46 %input vector in task-space
47 F(:,i) = M_mu_d*(mu_double_dot_desired(:,i))...
48         + n_mu_d;
49 % input vector in joint-space
50 tau(:,i) = Jt'*F(:,i);
51 % acceleration vector
52 q_double_dot(:,i) = inv(M)*(tau(:,i)-(coriolis v));
53
54 % velocity propogation
55 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
56
57 % position update
58 q(:,i+1) = q(:,i) +q_dot(:,i)*dt;
59 [x(i),y(i)] = FK2R(q(1,i),q(2,i));

```

```

58 q(:,i+1) = q(:,i) +q_dot(:,i)*dt + 1/2*q_double_dot(:,i)*dt^2;
59 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
60 end
61 % numerical integration ends here
62 %% Animation
63 for i=1:10:length(t)
64     x1 = a1*cos(q(1,i));
65     y1 = a1*sin(q(1,i));
66     x2 = x1+a2*cos(q(1,i)+q(2,i));
67     y2 = y1+a2*sin(q(1,i)+q(2,i));
68     plot([0,x1,x2],[0,y1,y2],'r-o','linewidth',2);
69     grid on, set(gca,'fontsize',20)
70     hold on
71     plot(mu_desired(1,:),mu_desired(2,:),'b-x','linewidth',2);

```

So, then you can see like then you can like see in the MATLAB code. So, I just want to show that this is the inverse dynamics model which we derived in the MATLAB you can see these are the cases which we have seen in the you can say earlier slides. So, now this is the desired value and then this is the joint desired value just for our reference and this is the initial condition, and this is the dynamics.

So, we have derived actual, and you can say desired the dynamics then this is the inverse dynamics which is like based on you can see like known values and then you can calculate the force then based on that you can like calculate the tau based on J desired transpose to F. So, then you can like go this so, after that you can see like these all the sub functions, and I am trying to plot before that I am trying to you can see animate. So, now in that sense if I like ran this code.

(Refer Slide Time: 07:49)

```
58- q(:,i+1)
59- [x(i),y
60- end
61- % numerical
62- %% Animation
63- for i=1:10:
64- x1 = a1
65- y1 = a1
66- x2 = x1
67- y2 = y1
68- plot([0,x1,x2],[0,y1,y2], 'r-o', 'linewidth
69- grid on, set(gca,'fontsize',20)
70- hold on
71- plot(mu_desired(1,:),mu_desired
```

```
58- q(:,i+1)
59- [x(i),y
60- end
61- % numerical
62- %% Animation
63- for i=1:10:
64- x1 = a1
65- y1 = a1
66- x2 = x1
67- y2 = y1
68- plot([0,x1,x2],[0,y1,y2], 'r-o', 'linewidth
69- grid on, set(gca,'fontsize',20)
70- hold on
71- plot(mu_desired(1,:),mu_desired
```

Inverse dynamics	Computed torque or model-based control	Model-based control	Computed control or model-free control scheme
○○○○○○○○○○○○○○○○	○○	○○	○○○○○○○○○○○○○○○○

The inverse dynamic model:  $\tau = \mathbf{M}(\mathbf{q}_d) \ddot{\mathbf{q}}_d + \mathbf{n}(\mathbf{q}_d, \dot{\mathbf{q}}_d)$

SRINIVASAN MOHAN, IIT PALAOKAR  
MECHANICS AND CONTROL OF ROBOTS: MANIPULATORS



Inverse Dynamics  
00000000000000000000

Computed torque or model based control  
00

Matrix based control  
00

Controlled control or double loop control system  
00000000

The inverse dynamic model:  $\tau = \mathbf{M}(\mathbf{q}_d)\ddot{\mathbf{q}}_d + \mathbf{n}(\mathbf{q}_d, \dot{\mathbf{q}}_d)$

```

%% Inverse dynamics
%input vector
tau(:,i) = Md*(inv(Jd)*(mu_double_dot_desired(:,i)...
-Jdotd*q_dot_desired(:,i)))...
+ oe_vd+g_vd;

```

ROBOTICS MODELING, IIT PALAASHI  
MODELING AND CONTROL OF ROBOTS: MANIPULATORS



So, you can see like, so, it is this is the profile which is given as a desired and you are system also like following it. So, this is what you can see as inverse dynamics in the task-space, and this is the theta 1 and theta 2, how it gone based on what you call the circular profile of mu which is given. So, now the same thing, same thing can be done in the joint space just for your benefit.

So, we have like taken that. So, the inverse dynamics in the joint space so, then what happened this was the equation for that you need to know q double dot desired because the q dot desired and q desired, we have already calculated in the earlier code. So, now we have to change. So, now you can see like this way we can take it the q double dot desired can be written as mu double dot desired minus J dot desired multiply with q dot desired.

(Refer Slide Time: 08:57)

```

40 %% Inverse dynamics
41 %input vector
42 tau(:,i) = Md*(inv(Jd)*(mu_double_dot_desired(:,i)...
43 -Jdotd*q_dot_desired(:,i)))...
44 + oe_vd+g_vd;
45 % acceleration vector
46 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_vd+g_vd));
47
48 % velocity propagation
49 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
50
51 % position update
52 q(:,i+1) = q(:,i) + q_dot(:,i)*dt;
53 [x(i),y(i)] = FK2R(q(1,i),q(2,i));

```



```

19
20 mu_dot_
21
22 mu_doub
23
24 %% Desi
25 [q_desi
26 Jd = Ja
27 q_dot_d
28 %% Init
29 q(:,1) = q_desired(:,1);
30 q_dot(:,1) = q_dot_desired(:,1);
31 %% Desired dynamic terms
32 Md = inertia2R(q_desired(:,i));

```

```

(i));
.2*t(i));...
.2*t(i));
.2^2*sin(0.2*t(i)
2^2*cos(0.2*t(i))
ities
2R(m_desired(1,i
(2
de i);

```

```

19
20 mu_dot_
21
22 mu_doub
23
24 %% Desi
25 [q_desi
26 Jd = Ja
27 q_dot_d
28 %% Init
29 q(:,1) = q_desired(:,1);
30 q_dot(:,1) = q_dot_desired(:,1);
31 %% Desired dynamic terms
32 Md = inertia2R(q_desired(:,i));

```

```

(i));
.2*t(i));...
.2*t(i));
.2^2*sin(0.2*t(i)
2^2*cos(0.2*t(i))
ities
2R(m_desired(1,i
(2
de i);

```

Adaptive dynamic compensation  
 Computed torque or model-based control  
 Motion-based control  
 Cascaded control or multi-loop control scheme

```

global a1 a2 m1 m2 g
%% System parameters
m1 = 2; m2 = 1; % link masses
a1 = 0.5; a2 = 0.4; % link lengths
g = 0+9.81i; % gravity
% %% Initial conditions
q = [-pi/4; pi/3]; % initial joint positions
q_dot = [0; 0]; % initial joint velocities
%% Control parameters
Kp = 4; Kd = 4;

```

Adaptive dynamic compensation  
 Computed torque or model-based control  
 Motion-based control  
 Cascaded control or multi-loop control scheme

SASTRICHANDRAN MOHAN, IIT PALAKKAD  
 MOTION AND CONTROL OF ROBOTS MANIPULATORS

So, that is why we have calculated this  $\dot{J}$ . So, now you can like use it. So, you go back to the MATLAB, and you can see and put the, that particular function has joint space. So, you can see like the final equation we are directly writing in the, what you call tau. So, we are not calculated in the F. So, you can see like this is mu desired which is like M mu desired has changed as the M desired then inverse of J desired that comes So, after that mu desired double dot minus J dot desired into q dot desired. So, then these are other effect.

So, now you can see the same profile, what we have done in the earlier case the same thing we have used when you can see like this is also like following it. So, now this is what the inverse dynamics case. So, now we will go to the, what you call the motion control and the model-based motion control. So, in this sense we will go to the computed torque control the same code we are trying to change only thing the initial conditions we are giving it as it is, we are not equating to the q desired. First value is equal to q of first value that is we are not doing it we are directly using it some initial condition and then we are adding the control parameter which is Kp and Kd.

(Refer Slide Time: 10:11)

The image shows a MATLAB code editor window with the following content:

```

% Computed torque or model-based control
or
tau = M(q)J(q)^-1 [ddot{mu}_d - \dot{J}(q)q_dot + \Gamma_1\dot{mu} + \Gamma_2\ddot{mu}] + n(q, q_dot)
or
tau = J(q)^T [M_\mu [\ddot{mu}_d + \Gamma_1\dot{mu} + \Gamma_2\ddot{mu}] + n_\mu(\mu, \dot{\mu})]

%% Errors
mu_tilda(:,i) = mu_desired(:,i) - mu(:,i);
mu_dot_tilda(:,i) = mu_ddt_desired(:,i) - J*q_dot(:,i);
%% Computed-torque control
%input vector in task-space
F(:,i) = M_mu_d(mu_double_dot_desired(:,i) ...
            +Kp*mu_tilda(:,i)+Kd*mu_dot_tilda(:,i))...
            + n_mu_d;
% input vector in joint-space
tau(:,i) = Jd'*F(:,i);

```

The code editor also shows a navigation bar at the top with tabs for 'Inverse dynamics', 'Computed torque or model-based control', 'Model-based control', and 'Controlled control or model-based control scheme'. At the bottom, there is a footer for 'SASTRABHARATI MOHAN, IIT PALAKKAD' and 'MODELING AND CONTROL OF ROBOTS MANIPULATORS'.

So, now based on this word this is the equation we are talking about the task-space either you can write it this way directly tau or we can write it in this way. So, we are writing in the second form. So, first we derived F then we are like multiply with J transpose that will give the tau. So, that is what we are like writing it here. So, you can see the mu tilde. So, we have like calculated the error mu tilde is like mu desired minus mu and mu dot tilde is like mu dot desired minus J of J into q dot this is nothing but mu dot. So, then the computed torque

control we have used this is in the task-space. So, we can calculate the F then based on the F we can convert it into tau.

(Refer Slide Time: 11:01)

Slide content:

$$\tau = M(q)J(q)^{-1} [\ddot{\mu}_d - \dot{J}(q)\dot{q} + \Gamma_1\dot{\mu} + \Gamma_2\ddot{\mu}] + n(q, \dot{q})$$

or

$$\tau = J(q)^T [M_{\mu} [\ddot{\mu}_d + \Gamma_1\dot{\mu} + \Gamma_2\ddot{\mu}] + n_{\mu}(\mu, \dot{\mu})]$$

+

4

SRITHARAN MURUGAN, IIT PALAKKAD  
MECHATRONICS AND CONTROL OF ROBOTS MANIPULATORS

```
13- q = [-pi/4; pi/3]; % initial joint positions
14- q_dot = [0; 0]; % initial joint velocities
15- %% Control parameters
16- Kp = 4; Kd = 4;
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));
21-                       0.4-0.3*cos(0.2*t(i))];
22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));
23-                            0.3*0.2*sin(0.2*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.3*0.2*cos(0.2*t(i));
25-                                    0.3*0.2*sin(0.2*t(i))];
26-     %% Desired joint positions and
```



```

25     0.3*0.2^2*cos(0.2*t(i))
26     %% Desired joint positions and velocities
27     [q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i)
28     Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
29     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30     %% Initial Conditions
31     q(:,1) = q_desired(:,1);
32     q_dot(:,1) = q_dot_desired(:,1);
33     %% Desired dynamic terms
34     Md = inertia2R(q_desired(:,i));
35     oe_vd = other_effects2R(q_desired(:,i),mu_desired(
36     g_vd = gravity_effects2R(q_desired
37     Jdotd = Jacodot2R(q_desired(:,i),mu_desired(
38     % inertia matrix in task-space

```

```

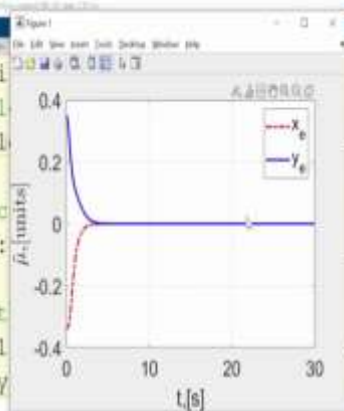
52     %% Computed-torque control
53     %input vector in task-space
54     F(:,i) = M mu d*(mu double dot desired(:,i) ...
55     +Kp*mu tilda(t,i)+Kd*mu dot tilda(:,
56     + n mu d;
57     % input vector in joint-space
58     tau(:,i) = Jd'*F(:,i);
59     % acceleration vector
60     q_double_dot(:,i) = inv(M)*(tau(:,i)-(
61
62     % velocity propogation
63     q_dot(:,i+1) = q_dot(:,i) + q_doub
64
65     % position update

```

```

58     tau(:,i)
59     % accel
60     q_doubl
61
62     % veloc
63     q_dot(:,
64
65     % posit
66     q(:,i+1)
67     [x(i),y
68     end
69     % numerical integration ends here
70     %% Animation
71     for i=1:10:length(t)

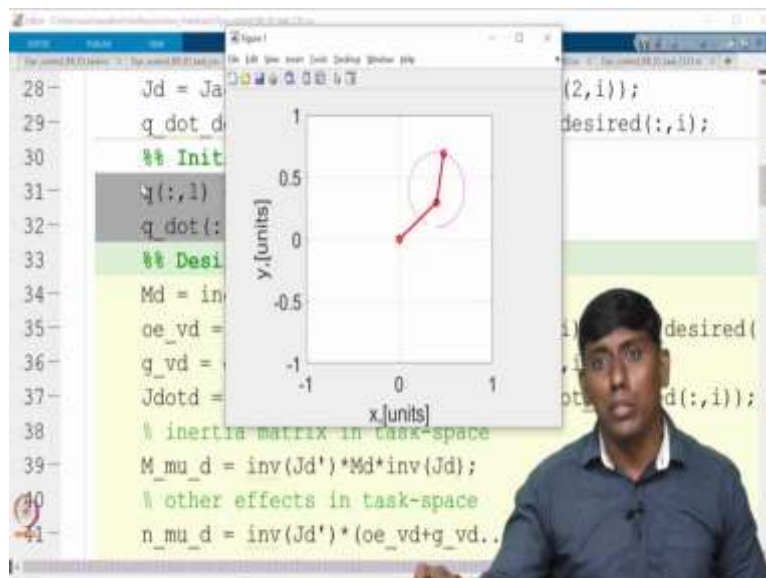
```



```

28- Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
29- q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30- %% Initial Conditions
31- q(:,1) = q_desired(:,1);
32- q_dot(:,1) = q_dot_desired(:,1);
33- %% Desired dynamic terms
34- Md = inertia2R(q_desired(:,i));
35- oe_vd = other_effects2R(q_desired(:,i), q_desired(
36- q_vd = gravity_effects2R(q_desired(:,i)
37- Jdotd = Jacodot2R(q_desired(:,i),q_dot_desired(:,i));
38- % inertia matrix in task-space
39- M_mu_d = inv(Jd')*Md*inv(Jd);
40- % other effects in task-space
41- n_mu_d = inv(Jd')*(oe_vd+g_vd.

```

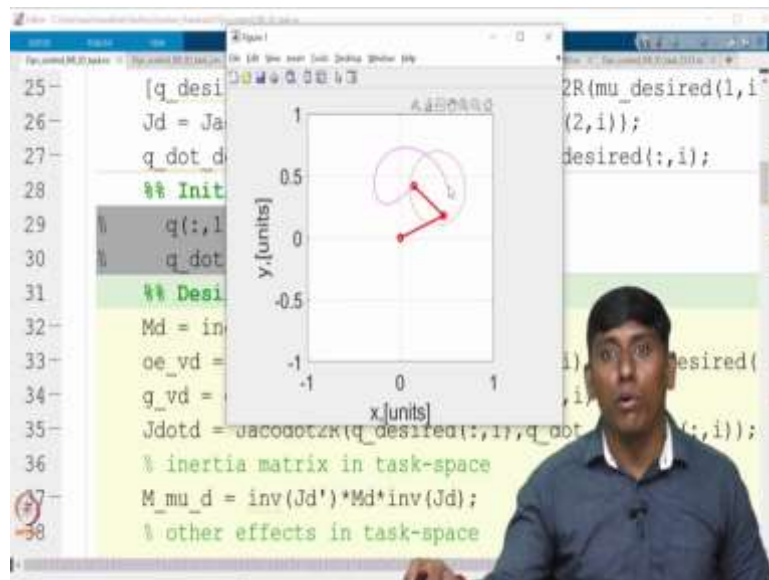
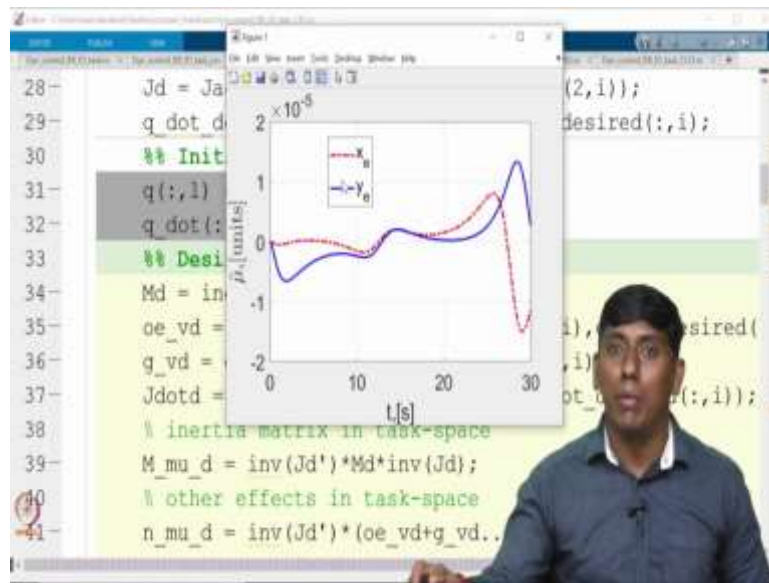


So, now we can like go to the MATLAB code. So, this is we have written as CTC. So, now you can see like what we have added. So, this is added and now, you can see like this is we have commented. So, we are not equating that so, now the  $q$  at  $t$  equal to 0 and  $q$  desired that equal to 0 are not equal. Similarly,  $\dot{q}$  at  $t$  equal to 0 and  $\dot{q}$  desired at  $t$  equal to the 0 also not equal.

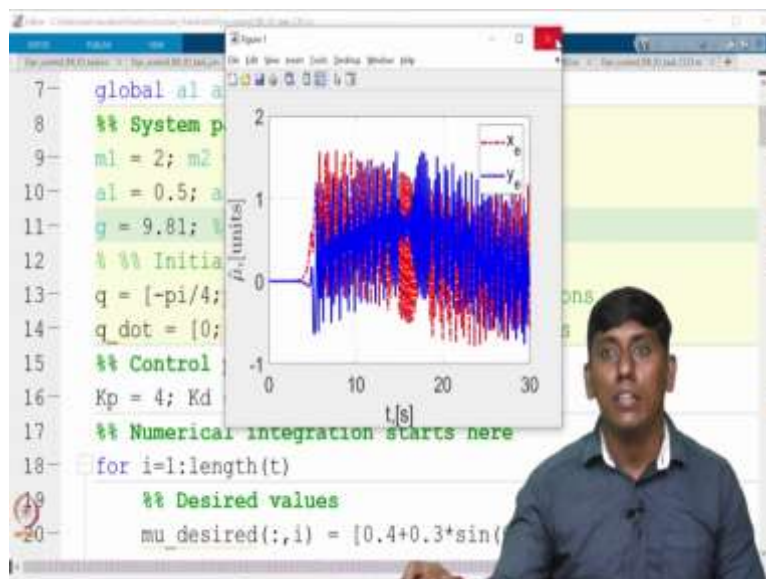
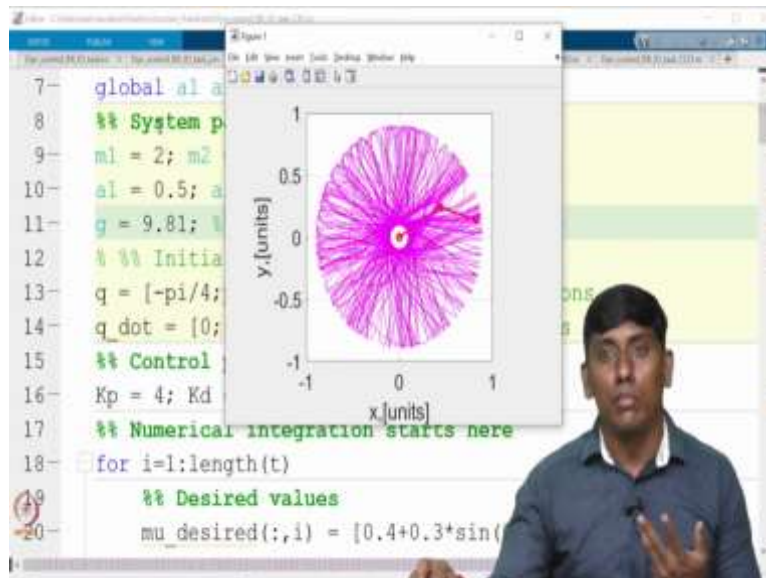
So, in that sense you can see like we have just added this. So, you can see like this  $K_p \tilde{\mu} + K_d \tilde{\dot{\mu}}$  has added and we can like come back So, now this is like feed forward and feedback linearization in addition, the feedback control PD has come. So, that is what we call computed torque control. So, now we can like see so, how this is like beneficial now, you can see it is starting from some different initial condition and it is like following it.

So, now, if you look at the error, error is like converge to 0 it started with some nonzero initial error, so that is like converge. So, now if I like include this so, I am just saying that the  $\dot{q}$  and  $q$  at initial both are like equal to 0. So, then you can like see, so, that would like you much more, smoother. So, you can like see it to why we are like saying you can see it is like much more super impose the actual even switching is not happening.

(Refer Slide Time: 12:27)



```
7- global a1 a2 m1 m2 g
8 %% System parameters
9 m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11 g = 9.81; % gravity
12 % %% Initial conditions
13 q = [-pi/4; pi/3]; % initial joint positions
14 q_dot = [0; 0]; % initial joint velocities
15 %% Control parameters
16 Kp = 4; Kd = 4;
17 %% Numerical integration starts here
18 for i=1:length(t)
19     %% Desired values
20     mu_desired(:,i) = [0.4+0.3*sin(t(i));
```



So, now you can see like this is in the order of  $10^{-5}$  in the sense it is almost like micro level of error, or you can say sub millimetre level error in terms of meter if I talk. So, now why this is required you can see like the task-space so if I am like equating this, you can I am just bringing it and so, this I am bringing it so this I am like commenting in the inverse dynamics.

So, you can see like this would not work as the desired manner that is what we are like trying to show. In fact it may be unstable also you can see it is starting somewhere here this is the starting point and the circle supposed to follow you can see it is not making a circle because the initial velocity at desired and the initial velocity actual are different an initial position desired are different. So, you can see it. So, that is why I said it cannot be directly used as a control it can be certain extent we can use the open loop control, but provided that initial conditions are supposed to match, and you can see other dimensions also make it.

So, now I hope you are clear why the CTC has come. So now you want to even improve the system even you assume that the, you call the other effect is not compensated you imagine that. So, now I am like seeing that the gravity is there. So, now I include the gravity and you can see like the error maybe the same order. So, now you can see like we have not compensated that directly. So, that is what I want to like make it.

So, we are not compensated that gravity here anywhere. So, because of that it is like a giving what you call this additional adverse effect. So, now you got idea why we need to like account this. So, these all like additional input which you need to know. So, now I am like why we are like making it all the time gravity compensated you would have like got it now clear.



(Refer Slide Time: 14:40)

```

10- a1 = 0.5; a
11- g = 0+9.81;
12- %% initia
13- q = [-pi/4;
14- q_dot = [0;
15- %% Control
16- Kp = 4; Kd
17- %% Numerica
18- for i=1:len
19- %% Desi
20- mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i)
21- 0.4-0.3*cos(0.2
22- mu_dot_desired(:,i) = [0.3*0.2*c
23- 0.3*0.2*

```

$\tau = J(q)^T [K_p(q_d - q) + K_d(\dot{q}_d - \dot{q})] + g(q)$

```

%% Errors
mu_tilda(:,i) = mu_desired(:,i) - mu(:,i);
mu_dot_tilda(:,i) = mu_dot_desired(:,i) - J*q_dot(:,i);
%% PD control
%input vector in task-space
F(:,i) = Kp*mu_tilda(:,i)+Kd*mu_dot_tilda(:,i);
% input vector in joint-space
tau(:,i) = Jd'*F(:,i)+g_v;

```

Srinivasan Aravamudan, IIT Palakkad  
Mechatronics and Control of Robotic Manipulators

So, now this is the additional you can say input which I want to give. So, now we will move to the motion-based control. So, in that sense, we are like going for PD and PID. So, first a PD with the gravity compensation we are trying to do so in that sense the complex terms are all gone and it is just a J transpose multiply with the PD and the gravity compensation. So, that is what you can see the F is added and the gravity compensation is added with this. So, now we will like to give this.

(Refer Slide Time: 15:17)

```
52 %% PD control
53 %input vector in task-space
54 F(:,i) = Kp*mu_tilda(:,i)+Kd*mu_dot_tilda(:,i);
55 % input vector in joint-space
56 tau(:,i) = Jd'*F(:,i)+g_v;
57 % acceleration vector
58 q_double_dot(:,i) = inv(M)*(tau(:,i)-(ones(2,1)*g_v));
59
60 % velocity propogation
61 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
62
63 % position update
64 q(:,i+1) = q(:,i) + q_dot(:,i)*dt;
65 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
```

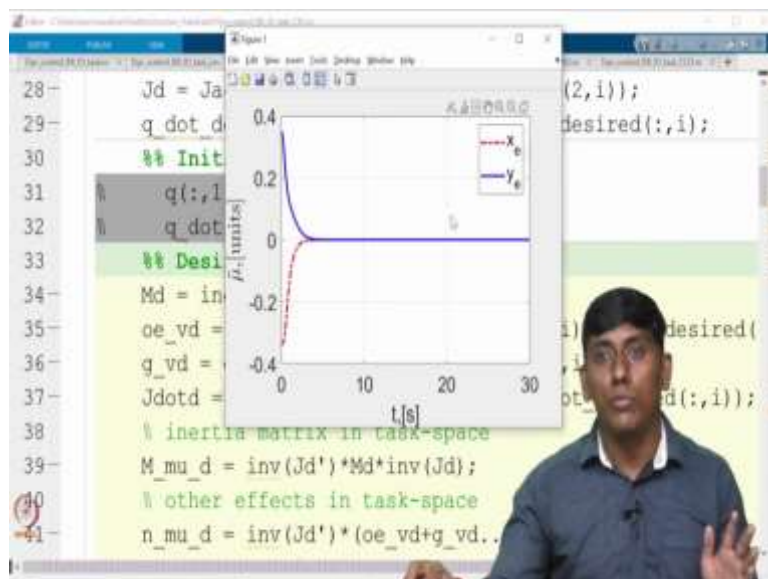
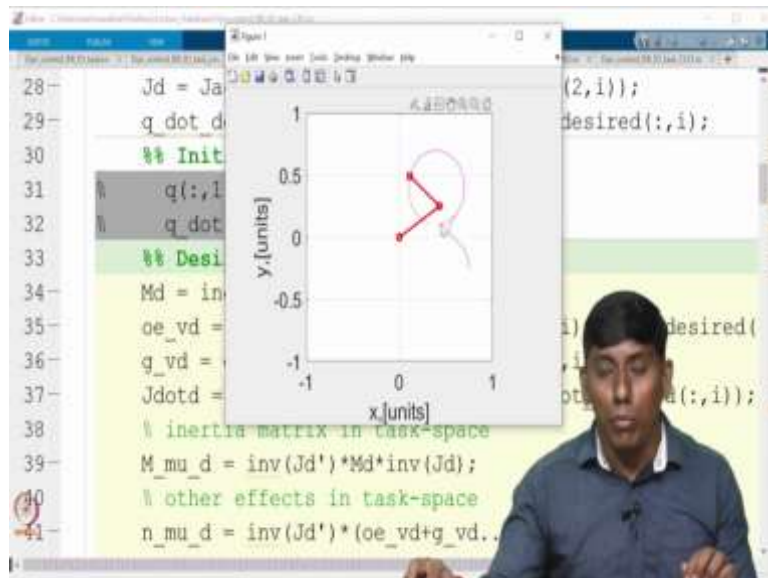
```
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 0*9.81; % gravity
12- %% Initial conditions
13- q = [-pi/4;pi/3]; % initial joint positions
14- q_dot = [0;0]; % initial joint velocities
15- %% Control parameters
16- Kp = 16; Kd = 8;
17- %% Numerical integration starts here
18- for i=1:length(t)
19- %% Desired values
20- mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));
21-                   0.4-0.3*cos(0.2*t(i))];
22- mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));
23-                       0.3*0.2*sin(0.2*t(i))];
```

```
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 0*9.81; % gravity
12- %% Initial conditions
13- q = [-pi/4;pi/3]; % initial joint positions
14- q_dot = [0;0]; % initial joint velocities
15- %% Control parameters
16- Kp = 4; Kd = 4;
17- %% Numerical integration starts here
18- for i=1:length(t)
19- %% Desired values
20- mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));
21-                   0.4-0.3*cos(0.2*t(i))];
22- mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));
23-                       0.3*0.2*sin(0.2*t(i))];
```

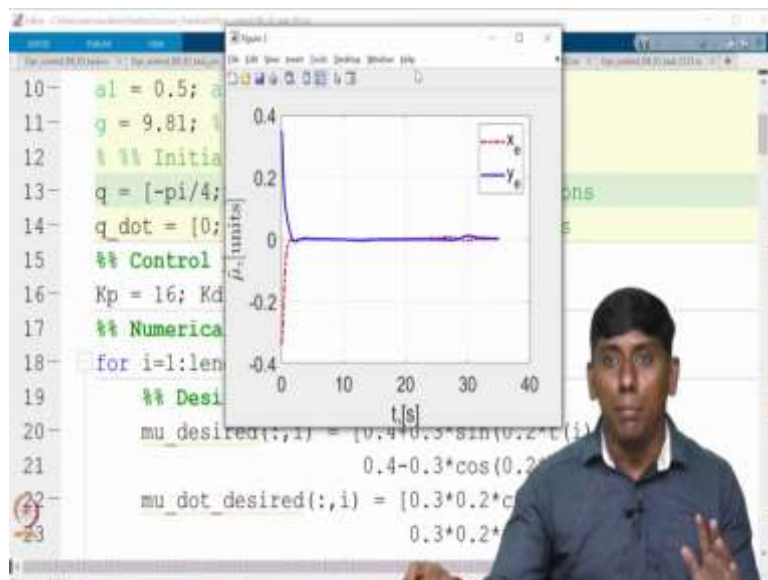
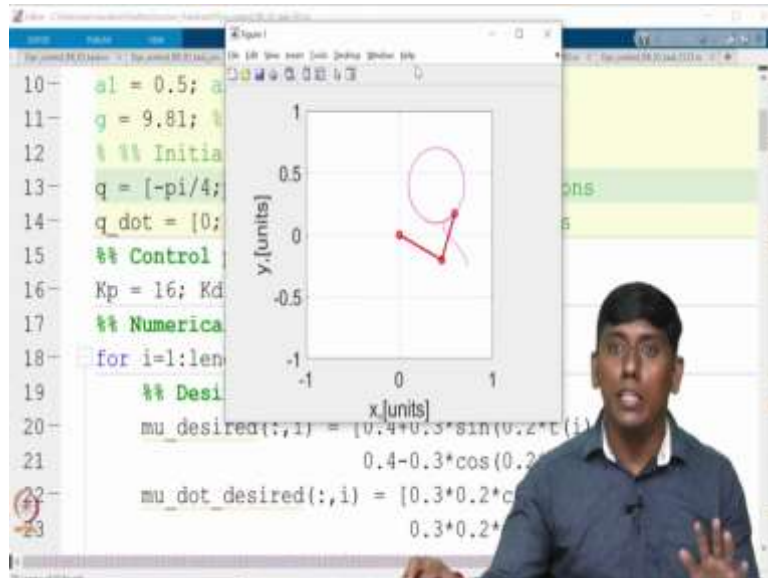
```

28- Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
29- q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30- %% Initial Conditions
31- q(:,1) = q_desired(:,1);
32- q_dot(:,1) = q_dot_desired(:,1);
33- %% Desired dynamic terms
34- Md = inertia2R(q_desired(:,i));
35- oe_vd = other_effects2R(q_desired(:,i),mu_desired(
36- q_vd = gravity_effects2R(q_desired(:,i),mu_desired(
37- Jdotd = Jacodot2R(q_desired(:,i),q_dot_desired(:,i));
38- % inertia matrix in task-space
39- M_mu_d = inv(Jd')*Md*inv(Jd);
40- % other effects in task-space
41- n_mu_d = inv(Jd')*(oe_vd+g_vd..

```







So, I am just showing that PD control. So, now you can see like PD control even the gravity we can include later on. So, right now we can like see the PD control is like coming into our picture. So, now you can see this is PD and the gravity compensation in this case any way gravity is not there, and you can like see it is starting from nonzero and it is like trying to follow the circle and you can see initially there is an error after that it is like a much more improved. So, you can like see there is a small error.

So, now if I include the gravity, you can like try to see. So, what will happen whether the gravity is trying to compensate here or not that we can like see it. So, you can like see it does. So, you can like find it. So, now the same way you can like see here, so the computed torque control so you have assumed this is the initial and this desired I am not making it equal. So, now I am just trying to show that the gravity is coming, but my computed torque control is like working or not.

So, you can see like this is working but if I start from here itself that gravity is not properly compensated that is what the happened earlier. So, now you can like see, this is the error for this, but the PD control is like giving more-closer result. So, that is why I said if the gravity is compensated if you are moving in very slow speed even the PD control is sufficient. So, that is why most of the industrial manipulator the PD control is using but you can see like small hitch up here and there will come this is due to what you call the constraint on the Jacobian.

(Refer Slide Time: 17:09)

The image displays a presentation slide and a MATLAB script. The slide shows the following content:

$$\tau = J(q)^T [K_p(q_d - q) + K_d(\dot{q}_d - \dot{q})] + g(q)$$

```

%% Errors
mu_tilda(:,i) = mu_desired(:,i) - mu(:,i);
mu_dot_tilda(:,i) = mu_dot_desired(:,i) - J*q_dot(:,i);
%% PD control
%input vector in task-space
F(:,i) = Kp*mu_tilda(:,i)+Kd*mu_dot_tilda(:,i);
% input vector in joint-space
tau(:,i) = Jd'*F(:,i)+g_v;

```

Below the slide is a screenshot of a MATLAB script titled "PID Control of a RR planar robot". The script includes the following code:

```

1 %% PID Control of a RR planar robot
2 clear all; close all; clc;
3 %% Simulation parameters
4 dt = 0.01; % stepsize
5 ts = 35; % total simulation time
6 t = 0:dt:ts; % time span
7 global a1 a2 m1 m2 g
8 %% System parameters
9 m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11 g = 0*9.81; % gravity
12 %% Initial conditions
13 q = [-pi/4;pi/3]; % initial joint positions
14 q_dot = [0;0]; % initial joint velocities

```

```

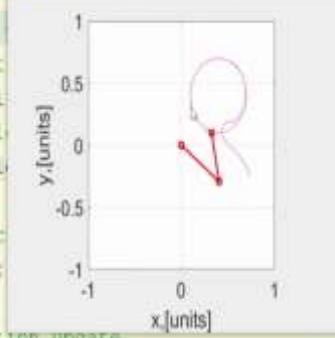
55 %input vector in task-space
56 F(:,i) = Kp*mu_tilda(:,i)+Ki*ei+Kd*mu_dot_tilda(:,i);
57 % input vector in joint-space
58 tau(:,i) = Jd'*F(:,i)+g_v;
59 % acceleration vector
60 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
61
62 % velocity propogation
63 q_dot(:,i+1) = q_dot(:,i) + q_double_d...dt;
64
65 % position update
66 q(:,i+1) = q(:,i) +q_dot(:,i)*dt + ... (:
67 [x(i),y(i)] = FK2R(q(1,i),q(2,i))
68 end

```

```

55 %input
56 F(:,i)
57 % input
58 tau(:,i)
59 % accel
60 q_doubl
61
62 % veloc
63 q_dot(:
64
65 % position upaste
66 q(:,i+1) = q(:,i) +q_dot(:,i)*dt + ... (:
67 [x(i),y(i)] = FK2R(q(1,i),q(2,i))
68 end

```



```

7- global a1 a2 m1 m2 g
8 %% System parameters
9 m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11 g = 0*9.81; % gravity
12 %% Initial conditions
13 q = [-pi/4;pi/3]; % initial joint positions
14 q_dot = [0;0]; % initial joint velocities
15 %% Control parameters
16 Kp = 16; Kd = 8; Ki = 0.5; ei = [0;0];
17 %% Numerical integration starts here
18 for i=1:length(t)
19 %% Desired values
20 mu_desired(:,i) = [0.4+0.3*cos(i*pi/4);

```

```

7- global a1 a2
8- %% System parameters
9- m1 = 2; m2 = 1; % link masses
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 9.81; % gravity
12- % %% Initial conditions
13- q = [-pi/4; pi/3]; % initial joint positions
14- q_dot = [0; 0]; % initial joint velocities
15- %% Control parameters
16- Kp = 16; Kd = 8; Ki = 0.5; ei = [0; 0];
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(

```

```

7- global a1 a2
8- %% System parameters
9- m1 = 2; m2 = 1; % link masses
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 9.81; % gravity
12- % %% Initial conditions
13- q = [-pi/4; pi/3]; % initial joint positions
14- q_dot = [0; 0]; % initial joint velocities
15- %% Control parameters
16- Kp = 16; Kd = 8; Ki = 0.5; ei = [0; 0];
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(

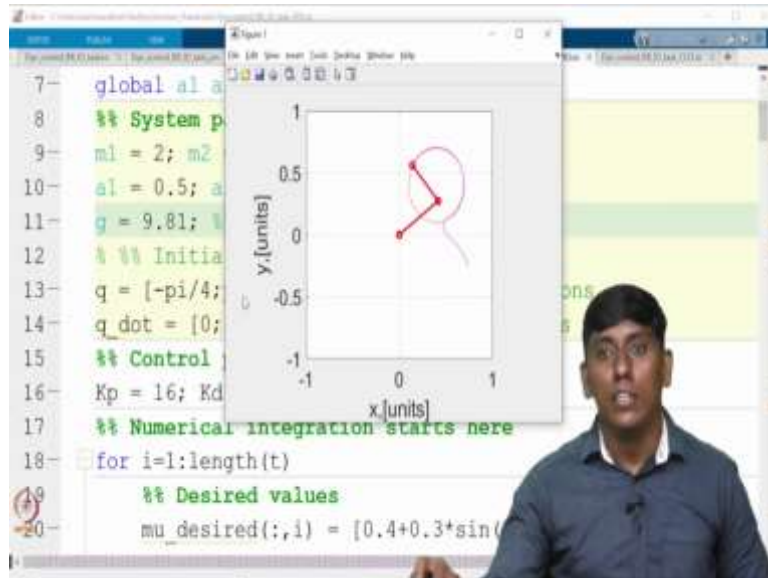
```

```

7- global a1 a2 m1 m2 g
8- %% System parameters
9- m1 = 2; m2 = 1; % link masses
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 9.81; % gravity
12- % %% Initial conditions
13- q = [-pi/4; pi/3]; % initial joint positions
14- q_dot = [0; 0]; % initial joint velocities
15- %% Control parameters
16- Kp = 16; Kd = 8; Ki = 0.5; ei = [0; 0];
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(

```





So, now people may think they error probably further improved if I use PID control it need not be all the cases. So, probably I will show an example. So, this is a PID control, and we are like trying to show this. So, now we can add the I term and we are like coming back here. So, in that sense we go to the controller directly. So, the PID control and the gravity we can like initially ignore then we can like show so, what you can see like the PID control we have like added Kp, Ki and so Kd is coming.


So, this is a PID, and we have like compensated that with the gravity compensation. So, I am trying to show you can see it is not actual like PD control performance. So, that is why if you add the third order you can see integral error dynamics that may like end up with even destroyed the performance it may not be all the time the error would be converge to 0.

The error converges to 0 however, you can see the dynamics is like modified. So, now in order to verify that, so, I am just reducing the; you can say integral value for example, I am putting it 0.5, I am just trying to show. So, earlier it was 5 and now it is 0.5 you can see, so, the performance is very close to PD and the integral error is like trying to make. So, you can see it is like more or less closer you can see. So, clearly you can see.


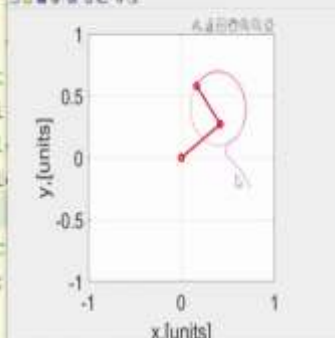
So, earlier case it was like more converge now, the integral is like making some you can say pushing different region. So, now this is what I want to like emphasize the PID control is not preferred, most commonly in the robotic manipulator community rather than the PD because the item can end up with what you call wind up. So, that is why we can like avoid this particular PID control. So, now you are like clear. So, we can like see further cases if I increase the, you can see PD value or probably I can like add the gravity value you can like see something going to happen.

(Refer Slide Time: 19:31)


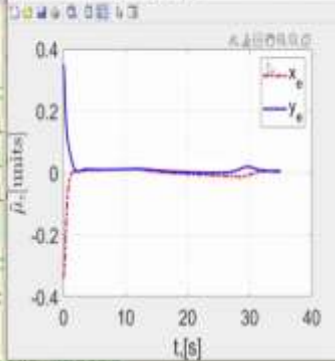
```
55 %input vector in task-space
56 F(:,i) = Kp*mu_tilda(:,i)+Ki*ei+Kd*mu_dot_tilda(:,i);
57 % input vector in joint-space
58 tau(:,i) = Jd'*F(:,i)+0.99*q_v;
59 % acceleration vector
60 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
61
62 % velocity propogation
63 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
64
65 % position update
66 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 0.5*q_double_dot(:,i)*dt^2;
67 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
68 end
```

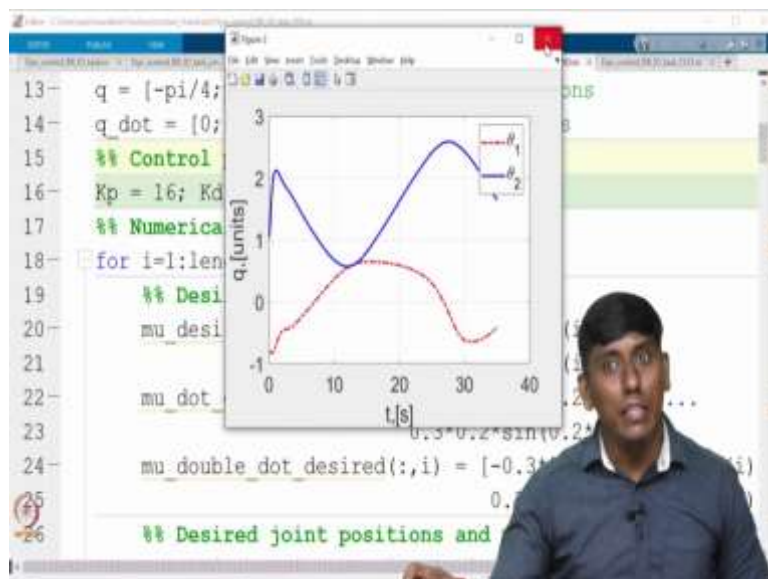
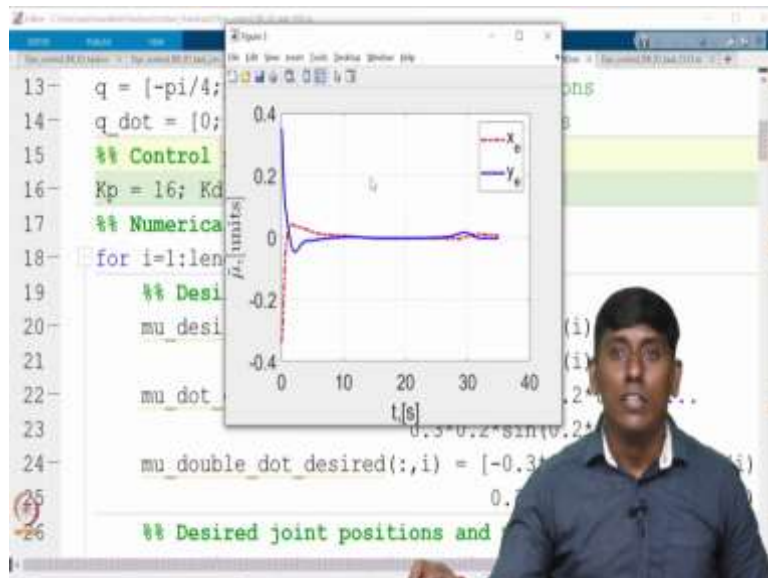
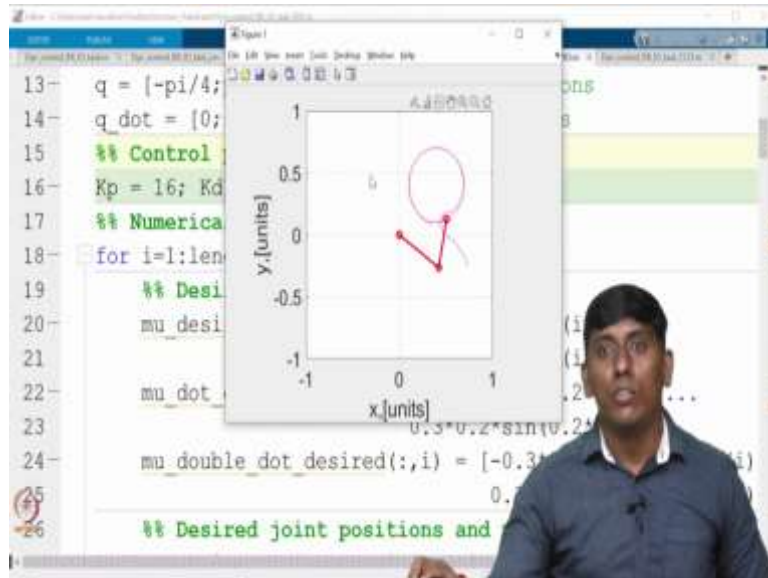


```
55 %input
56 F(:,i)
57 % input
58 tau(:,i)
59 % accel
60 q_doubl
61
62 % veloc
63 q_dot(:
64
65 % position upaste
66 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 0.5*q_double_dot(:,i)*dt^2;
67 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
68 end
```



```
55 %input
56 F(:,i)
57 % input
58 tau(:,i)
59 % accel
60 q_doubl
61
62 % veloc
63 q_dot(:
64
65 % position upaste
66 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 0.5*q_double_dot(:,i)*dt^2;
67 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
68 end
```





```

16- Kp = 16; Kd = 8; Ki = 5; ei = [0;0];
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(0.4*t(i));...
21-                       0.4-0.3*cos(0.2*t(i))];
22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
23-                            0.3*0.2*sin(0.2*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.3*0.2*cos(0.2*t(i))...
25-                                    0.3*0.2*sin(0.2*t(i))];
26-     %% Desired joint positions and velocities
27-     [q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i),mu_desired(2,i));
28-     Jd = Jacob2R(q_desired(1,i),q_desired(2,i));
29-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);

```

```

22-     mu_dot_desired(:,i) = [0.3*0.4*cos(0.4*t(i));...
23-                            0.3*0.2*sin(0.2*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.3*0.4^2*sin(0.4*t(i))...
25-                                    0.3*0.2^2*cos(0.2*t(i))];
26-     %% Desired joint positions and velocities
27-     [q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i),mu_desired(2,i));
28-     Jd = Jacob2R(q_desired(1,i),q_desired(2,i));
29-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30-     %% Initial Conditions
31-     q(:,1) = q_desired(:,1);
32-     q_dot(:,1) = q_dot_desired(:,1);
33-     %% Desired dynamic terms
34-     Md = inertia2R(q_desired(:,i));
35-     oe_vd = other_effects2R(q_desired(:,i));

```

```

Error using atan2
Inputs must be real.

Error in Dyn_control_RR_ID_task_PID>IK2R (line 141)
    th1 = atan2(y,x)-atan2(a2*s2,a1+a2*c2);

Error in Dyn_control_RR_ID_task_PID>IK2R (line 27)
    [q_desired(1,i),q_desired(2,i)] = IK2R(mu_desired(1,i),mu_desired(2,i));

```

Name	Value
a1	0.50
a2	0.40
dt	0.01
t	[0.0 1.0]

```

fx >>

```

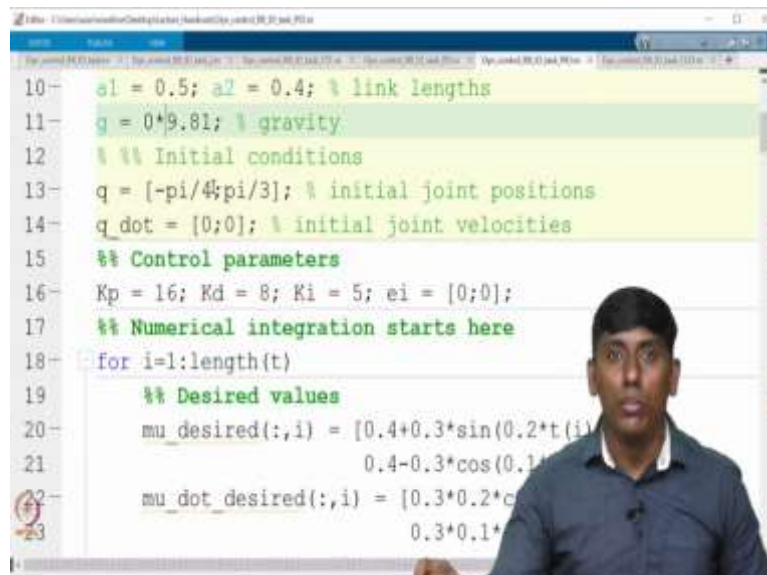


So, now you can see like I have not changed the  $K_i$  value but I have like included the gravity. So, you can like see the performance is as similar to the earlier it is not like a change because the gravity is compensated. So, now we assume that the gravity is compensated only 99 percentage. So, now the model is like not accurate.

So, now we assume that it is only 99 percentage, gravity is compensated. So, you can see like, it is more less working good, but you can see like there is an error steady state error is coming because the gravity is not completely compensated. So, that is what the whole idea you can see it. So, now in order to make this so if I increase  $i$  value will it like make it any you can say deteriorate performance or improved performance that we can like see it.

So, now I am like seeing it. So, now it is like, you can see that  $i$  is really working good. So, that switching is like converge, so the error is like converge. So, now if I increase the value of you can say the frequency. So, what will happen for example, I am increasing, the frequency. So, one frequency I am like increasing probably this 0.4 times. So, in the sense it like going to make, you can say 8 profile, which is very complex. So, you can like see people usually won't do that, so I hope so it is, so I am just trying to show so there is an error because some multiplications missing, it is like not able to do. Because the, you can say the inverse kinematics is not happening.

(Refer Slide Time: 21:33)



```
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 0*9.81; % gravity
12- % % Initial conditions
13- q = [-pi/4;pi/3]; % initial joint positions
14- q_dot = [0;0]; % initial joint velocities
15- % % Control parameters
16- Kp = 16; Kd = 8; Ki = 5; ei = [0;0];
17- % % Numerical integration starts here
18- for i=1:length(t)
19-     % % Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i))
21-                       0.4-0.3*cos(0.1*t(i))];
22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i))
23-                            0.3*0.1*sin(0.1*t(i))];
```

```

18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));...
21-                       0.4-0.4*cos(0.2*t(i))];
22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
23-                            0.3*0.2*sin(0.2*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.3*0.2^2*sin(0.2*t(i))
25-                                    0.3*0.2^2*cos(0.2*t(i))];
26-     %% Desired joint positions and velocities
27-     [q_desired(1,i),q_desired(2,i)] = IK2(mu_desired(:,i));
28-     Jd = Jacob2R(q_desired(1,i),q_desired(2,i));
29-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30-     %% Initial Conditions
31-     q(:,1) = q_desired(:,1);
32-     q_dot(:,1) = q_dot_desired(:,1);

```

```

9- m1 = 2; m2 = 1; % link masses
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 9.81; % gravity
12- % %% Initial conditions
13- q = [-pi/4;pi/3]; % initial joint positions
14- q_dot = [0;0]; % initial joint velocities
15- %% Control parameters
16- Kp = 16; Kd = 8; Ki = 5; ei = [0;0];
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));...
21-                       0.4-0.3*cos(0.2*t(i))];
22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
23-                            0.3*0.2*sin(0.2*t(i))];

```

```

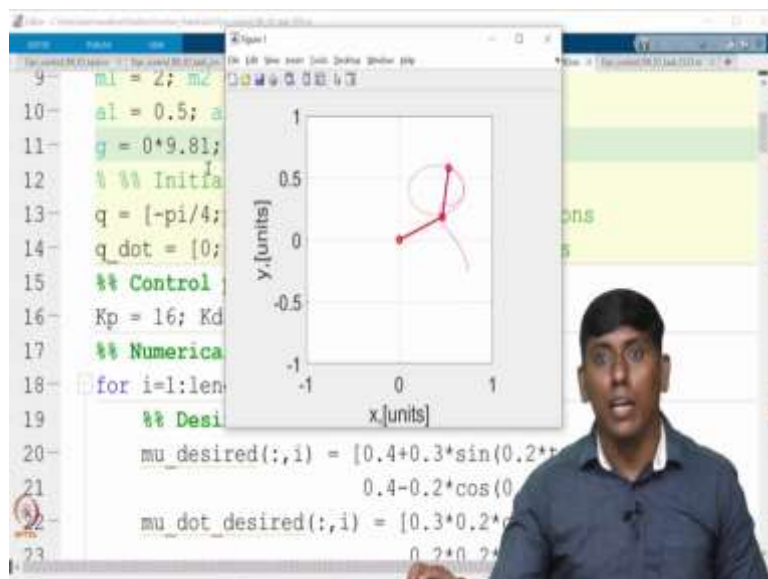
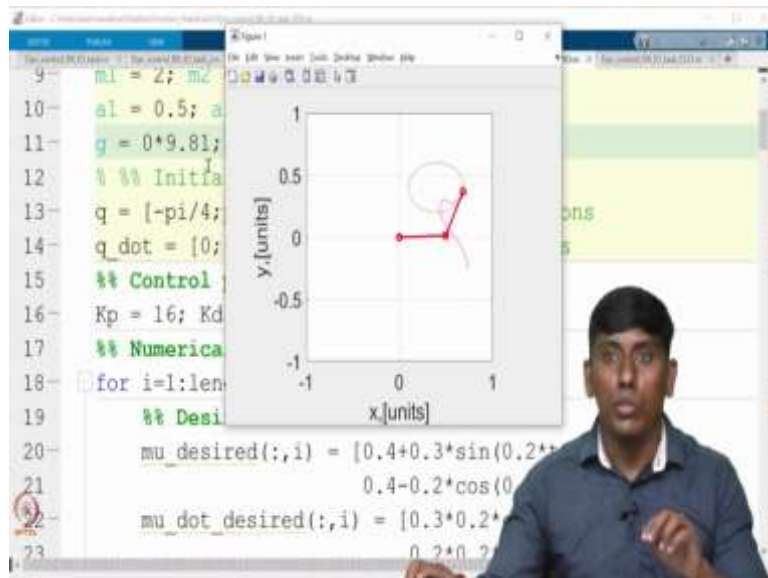
15- %% Control parameters
16- Kp = 16; Kd = 8; Ki = 5; ei = [0;0];
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));...
21-                       0.4-0.2*cos(0.2*t(i))];
22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
23-                            0.2*0.2*sin(0.2*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.3*0.2^2*sin(0.2*t(i))
25-                                    0.2*0.2^2*cos(0.2*t(i))];
26-     %% Desired joint positions and velocities
27-     [q_desired(1,i),q_desired(2,i)] = IK2(mu_desired(:,i));
28-     Jd = Jacob2R(q_desired(1,i),q_desired(2,i));
29-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);

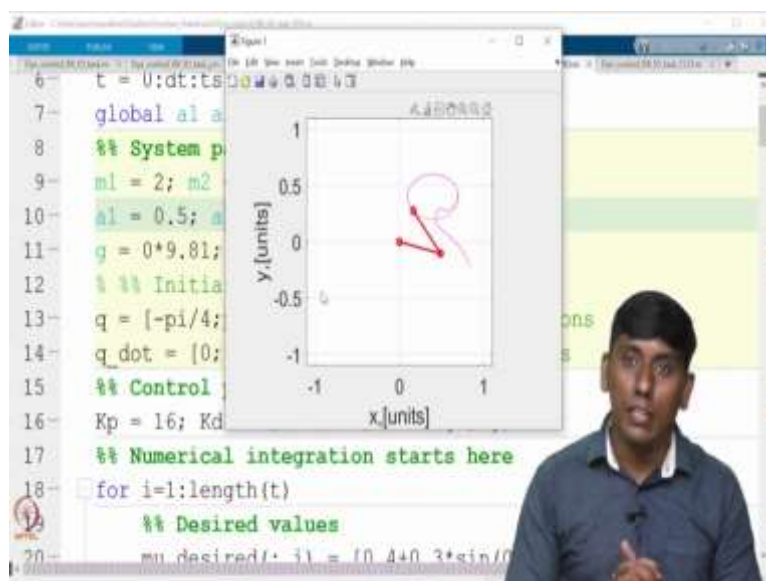
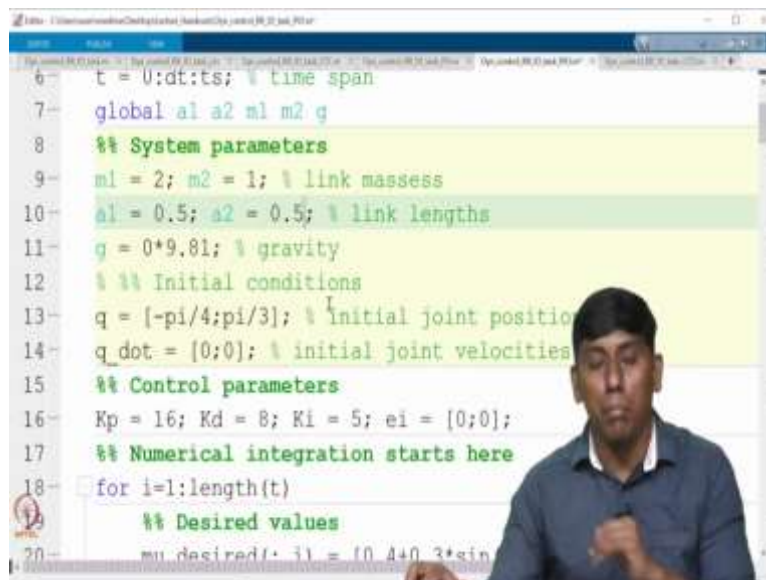
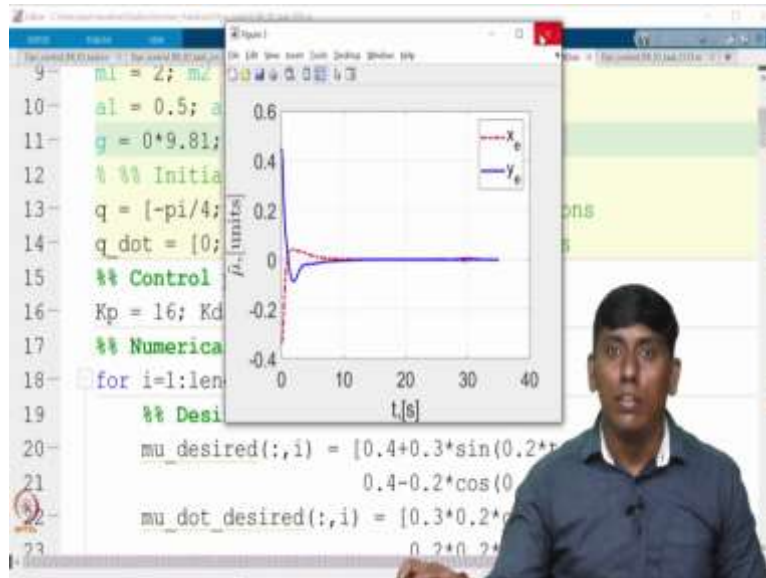
```

```

9  m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11  g = 0*9.81; % gravity
12  % %% Initial conditions
13  q = [-pi/4; pi/3]; % initial joint positions
14  q_dot = [0; 0]; % initial joint velocities
15  %% Control parameters
16  Kp = 16; Kd = 8; Ki = 5; ei = [0; 0];
17  %% Numerical integration starts here
18  for i=1:length(t)
19      %% Desired values
20      mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));
21                       0.4-0.2*cos(0.2*t(i))];
22      mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));
23                            0.2*0.2*sin(0.2*t(i))];

```





So, then what we have to see, so whether you are touching, like, probably improve your control gain values. So, probably I am just deteriorating this just for PD control. So, I am just giving it a PD control whether this particular performance is like a making it or not, it is not because it is very fast, it is like not able to do it. So, I am just bringing it back. So, I am just bringing it back. So, I am trying to reduce this, that hopefully work well. I am just trying to see.

So, that make work. So, in this sense, I am trying to show a complex profile which is like 8 kind of profile I just want to show it is still it is like not working because the gravity be compensated only 99 percent. So, I just want to show for that I am just assuming the gravity is negligible amount. So, I am trying to show this. So, still that particular point is like not converge or you can see the inverse kinematics is not happening because this is like very fast that 8 profile is like not making it so completely.

So, that is what we can like visible, so I am just trying to show only one thing. So, in the sense I am just increasing this profile probably this is trying to show probably 0.4. So, I hope this will work. So, in the sense I am trying to make an ellipse, so this is also like not working for us. There is an error so, because 0.8 is not possible so, I just to make it the ellipse as a way around. So, that is you can see like I make it an ellipse, so that ellipse is actual like following it.

So, now I am just trying to show you can see that PD control it is like converge. So, now we can like make it this kind of complex profile or complex entity and other things we can like try to do it. For example, now I assume that this is also like 0.5. So, you can see like the variation is, actual like a completely different. So, these all we can like experience and then try to do you can say different kinds of simulations and all. The cascaded design we will see in the next short video, and then we can like go across this. So, until then see you Thank you. Bye.