

**Mechanics and Control of Robotic Manipulators**  
**Professor Santhakumar Mohan**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Palakkad**  
**Lecture: 44**  
**Simulations Related to Dynamic Control Schemes Using MATLAB**

(Refer Slide Time: 00:14)

$e_1 = \mu_d - \mu$       $\dot{q}_d = J(q)^{-1} [\mu_d + K_1 e_1]$   
 $e_2 = \dot{q}_d - \dot{q}$       $\tau = M(q) [\ddot{q}_d + K_2 e_2 + J(q)^T e_1] + n(q, \dot{q})$

Welcome back to the you can say Mechanics and Control of Robotic Manipulator. Last class, we have seen like how the task space, you can see motion control can be done in MATLAB simulation. This particular lecture is in addition, so, again we are trying to do the MATLAB simulation but with what you call the cascaded or the double loop control scheme.

So, for that what we have seen in the lecture. So, in the regular theory we have seen there would be two error so,  $e_1$  is like  $\mu_d - \mu$  where  $e_2$  is like  $\dot{q}_d - \dot{q}$  while the  $\dot{q}_d$  is not directly derived from the inverse kinematics or inverse differential kinematics in the sense of what we are trying to do.

The initial loop what you call the outer loop, the outer loop would give the desired value for the inner loop in the sense the  $\dot{q}_d$  would be obtained from the outer loop which we have returned as  $\dot{q}_d$  as the Jacobian inverse into  $\mu_d + K_1 e_1$ . So, based on that what we have like derived the  $\tau$  which is the control law.

The control law or you can say control scheme is derived in this way where the matrix; inertia matrix multiply with the q double dot desired plus K2e2 in order to make the stability J transpose e1 plus in order to make the feedback linearization we have added n of q comma q dot the same thing we had tried to do it in MATLAB. So, in the sense what we have done here is.

(Refer Slide Time: 01:43)

The screenshot shows a MATLAB script with the following content:

$$e_1 = \mu_d - \mu \quad \dot{q}_d = J(q)^{-1} [\dot{\mu}_d + K_1 e_1]$$

$$e_2 = \dot{q}_d - \dot{q} \quad \tau = M(q) [\ddot{q}_d + K_2 e_2 + J(q)^T e_1] + n(q, \dot{q})$$

```

%% Double-loop motion control 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 = 9.81; % gravity

```

At the bottom of the slide, there is a small video inset of a man speaking, and a footer that reads: "SAPTARISHI MOHAN, IIT PALAKKAD, MECHANICS AND CONTROL OF ROBOTS, MANIPULATORS".

So, we have like make the same code. So, only thing we have added a few values. So, for that we have taken the same code and we have added the K1 and K2 instead of you can say Kp and Kd. So, now this is the added value and the mu dot desired mu desired mu double dot desired all are same just to give the, you can say simple comparison then you can see like just for our understanding.

So, this, q desired we have like derived this is not for the control scheme. This is only for so, for plotting or just for comparison. So, further we are like taking the actual dynamic terms and then you can see the other entity which is required because we need to know like inverse Jacobian dot and Jacobian all those things.

So, we have like use this. So, why we need inverse Jacobian dot because if you look at it this equation, so, if you are like taking q double dot desired which is required here. So, then that would be J q inverse into mu double dot desired plus K1e1 dot plus J dot of q inverse into mu dot desired plus K1e1 the sense you need J dot you can say inverse of J q dot is required.

So, that is why we are like writing it here that value. So, in the sense we are adding one additional sub function so, that is what addition to the existing control code whatever we have seen in the earlier class. So, in the sense this is the inverse Jacobian dot what we have like done so, this is the full function we can see it in the MATLAB. So, now I just want to add that there is a sub function which we have added.

(Refer Slide Time: 03:23)

```

%% Errors
mu_tilda(:,i) = mu_desired(:,i) - mu(:,i);
e1= mu_tilda(:,i);
eidot = mu_dot_desired(:,i) - J*q_dot(:,i);
q_dot_d = inv(Jd)*(mu_dot_desired(:,i)+K1*e1);
e2 = q_dot_d -q_dot(:,i);

+

```

Navigation icons: back, forward, search, etc.

Navigation icons: back, forward, search, etc.

```

%% Errors
mu_tilda(:,i) = mu_desired(:,i) - mu(:,i);
e1= mu_tilda(:,i);
eidot = mu_dot_desired(:,i) - J*q_dot(:,i);
q_dot_d = inv(Jd)*(mu_dot_desired(:,i)+K1*e1);
e2 = q_dot_d -q_dot(:,i);

%% Cascaded or Double-loop control
% input vector in joint-space
tau(:,i) = M*(inv(Jd)*(mu_double_dot_desired(:,i)...
+K1*eidot)...
+Jd_inv_dot*(mu_dot_desired(:,i)...
+K1*e1)+K2*e2+J'*e1)+oe_v+g_v;

```

Navigation icons: back, forward, search, etc.

Navigation icons: back, forward, search, etc.

```

%% %% Initial conditions      +
q = [-pi/4;pi/3]; % initial joint positions
q_dot = [0;0]; % initial joint velocities
%% Control parameters
K1 = 4; K2 = 4;
%% 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))];

```



```

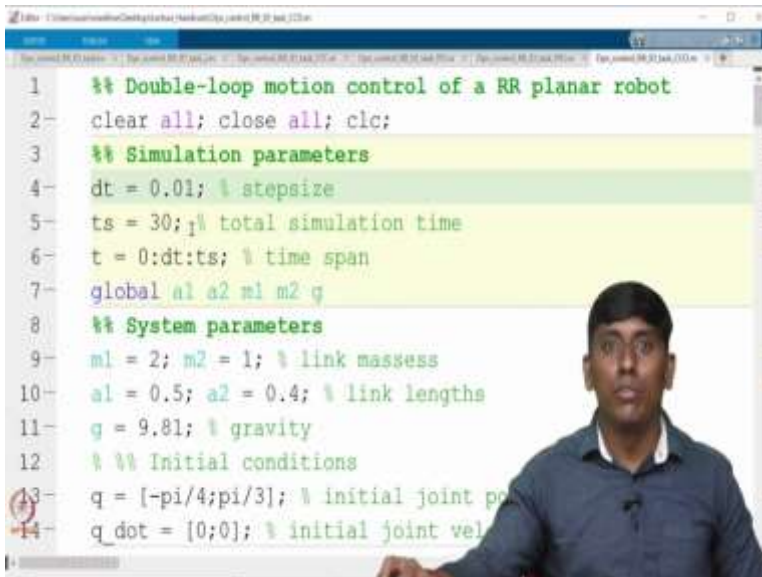
% acceleration vector
q_double_dot(:,i) = inv(M)*(tau(:,i)-(os_v+g_v));
% velocity propagation
q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
% position update
q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*q_double_dot(:,i)*dt^2;
[x(i),y(i)] = FK2R(q(1,i),q(2,i));
end

```

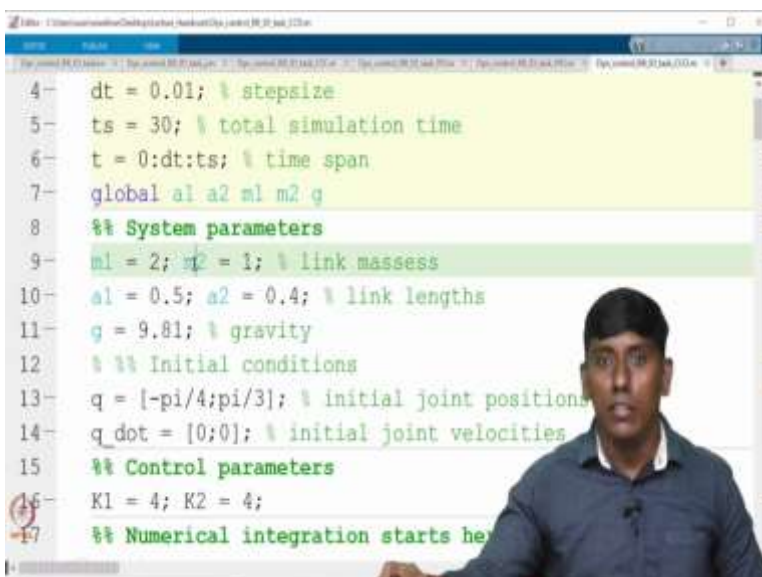


So, this equation like yes, it is a plus sign only not a minus sign because this is  $\dot{q}$  desired, we are like differentiating, so, in this sense plus sign so, that is like maintained here. So, then we can like go to the acceleration velocity and position update which we have done in the earlier the same thing. So, then we can like go to the final just to plot. So, what is the error in you can see outer space. So, that is what we are trying to see that since we go to the MATLAB code

(Refer Slide Time: 05:44)



```
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
```



```
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
15 %% Control parameters
16 K1 = 4; K2 = 4;
17 %% Numerical integration starts here
```

```
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- K1 = 4; K2 = 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))];
```

```
16- K1 = 4; K2 = 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*0.2^2*sin(0.2*t(i))];
26-     %% Desired joint positions and velocities
27-     [q_desired(1,i),q_desired(2,i)] = mu_desired(:,i);
28-     Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
29-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,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
28 Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
29 q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30 %% Actual dynamic terms
31 M = inertia2R(q(:,i));
32 oe_v = other_effects2R(q(:,i),q_dot(:,i));
33 g_v = gravity_effects2R(q(:,i));
34 [mu(1,i),mu(2,i)] = FK2R(q(1,i),q(2,i));
35 J = Jaco2R(q(1,i),q(2,i));
36 Jd_inv_dot = invJacodot2R(q_desired(1,i),q_desired(2,i));
37 %% Errors
38 mu_tilda(:,i) = mu_desired(:,i) - mu(:,i);

```

So, where we call CCD which is like Cascaded Control Design. So, this is the you can say the dual loop control so, initial as similar as the simulation parameter the system parameter has come then the gravity we included and the initial condition is random initial conditions, we have included and the K1 and K2 we assume as 4 then the mu dot desired mu desired mu double dot desired all derived as at based on that just for our purpose we have taken the inverse kinematics then the J desired we have calculated based on that then J dot desired based on the you can see original case. So, this is just for our idea.

(Refer Slide Time: 06:28)

```

37 %% Errors
38 mu_tilda(:,i) = mu_desired(:,i) - mu(:,i);
39 e1 = mu_tilda(:,i);
40 eldot = mu_dot_desired(:,i) - J*q_dot(:,i);
41 q_dot_d = inv(Jd)*(mu_dot_desired(:,i)+K1*e1);
42 e2 = q_dot_d - q_dot(:,i);
43 %% Cascaded or Double-loop control
44 % input vector in joint-space
45 tau(:,i) = M*(inv(Jd)*(mu_double_dot_desired(:,i)+K1*eldot)...
46 +K1*eldot)...
47 +Jd_inv_dot*(mu_dot_desired(:,i)+K1*e1)+K2*e2+J'*e1);
48 % acceleration vector
49 q_double_dot(:,i) = inv(M)*(tau(:,i));
50

```



```
43 %% Cascaded or Double-loop control
44 % input vector in joint-space
45 tau(:,i) = M*(inv(Jd)*(mu_double_dot_desired(:,i)...
46           +K1*eldot)...
47           +Jd_inv_dot*(mu_dot_desired(:,i)...
48           +K1*e1)+K2*e2+J'*e1)+oe_v+g_v;
49 % acceleration vector
50 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
51
52 % velocity propagation
53 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
54
55 % position update
56 q(:,i+1) = q(:,i) + q_dot(:,i)*dt;
```

```
55 % position update
56 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*q_double_dot(:,i)*dt^2;
57 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
58 end
59 % numerical integration ends here
60 %% Animation
61 for i=1:10:length(t)
62     x1 = a1*cos(q(1,i));
63     y1 = a1*sin(q(1,i));
64     x2 = x1+a2*cos(q(1,i)+q(2,i));
65     y2 = y1+a2*sin(q(1,i)+q(2,i));
66     plot([0,x1,x2],[0,y1,y2],'r-o','filled');
67     grid on, set(gca,'fontsize',20);
68     hold on;
```

```

73- hold off, pause(0.001)
74- end
75- %% Plotting functions
76- plot(t,q(1,1:i),'r-.',t,q(2,1:i),'b-','linewidth',2)
77- legend('\theta_1','\theta_2')
78- grid on
79- set(gca,'fontsize',20)
80- xlabel('t, [s]')
81- ylabel('q, [units]')
82-
83- plot(t,mu_tilda(1,1:i),'r-.',t,mu_tilda(2,1:i),'b-','linewidth',2)
84- legend('$x_e$','$y_e$','Interpreter','Interpreter')
85- grid on
86- set(gca,'fontsize',20)

```

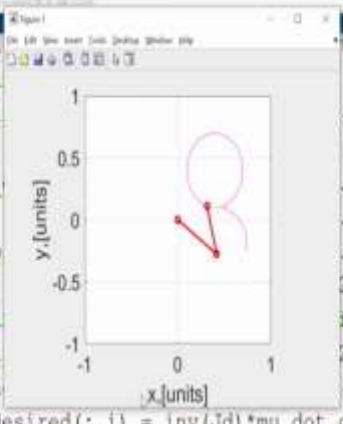
But the second thing what we are using it inside the loop so, we have taken this way. So, now we can see e1 e1 dot and e2 we calculated then the cascaded control as come here M into so and so, then these all like based on what we have derived there. So, then we can like come back here simple Euler integration based on the acceleration then you can see. So, the animation part then the plotting part and then the sub functions all.

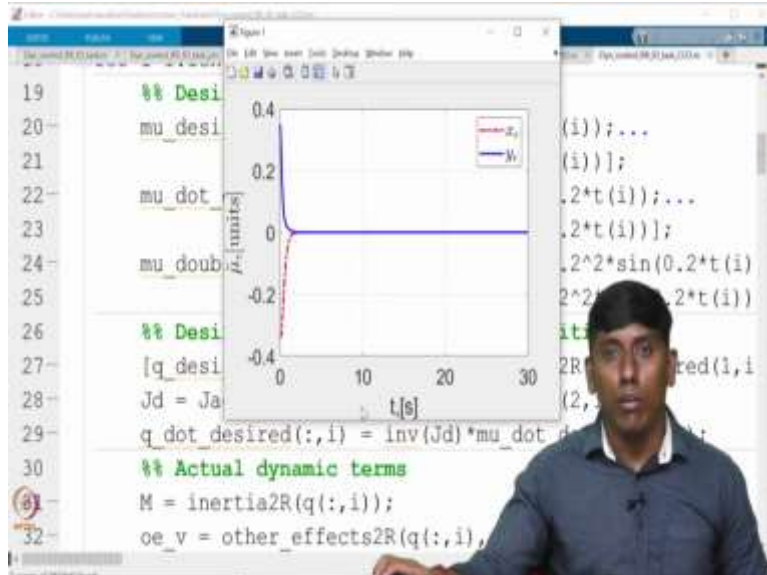
(Refer Slide Time: 07:01)

```

19- %% Desi
20- mu_desi
21-
22- mu_dot_
23-
24- mu_doub
25-
26- %% Desi
27- [q_desi
28- Jd = Ja
29- q dot desired(:,i) = inv(Jd)*mu_dot_d
30- %% Actual dynamic terms
31- M = inertia2R(q(:,i));
32- oe_v = other_effects2R(q(:,i),

```





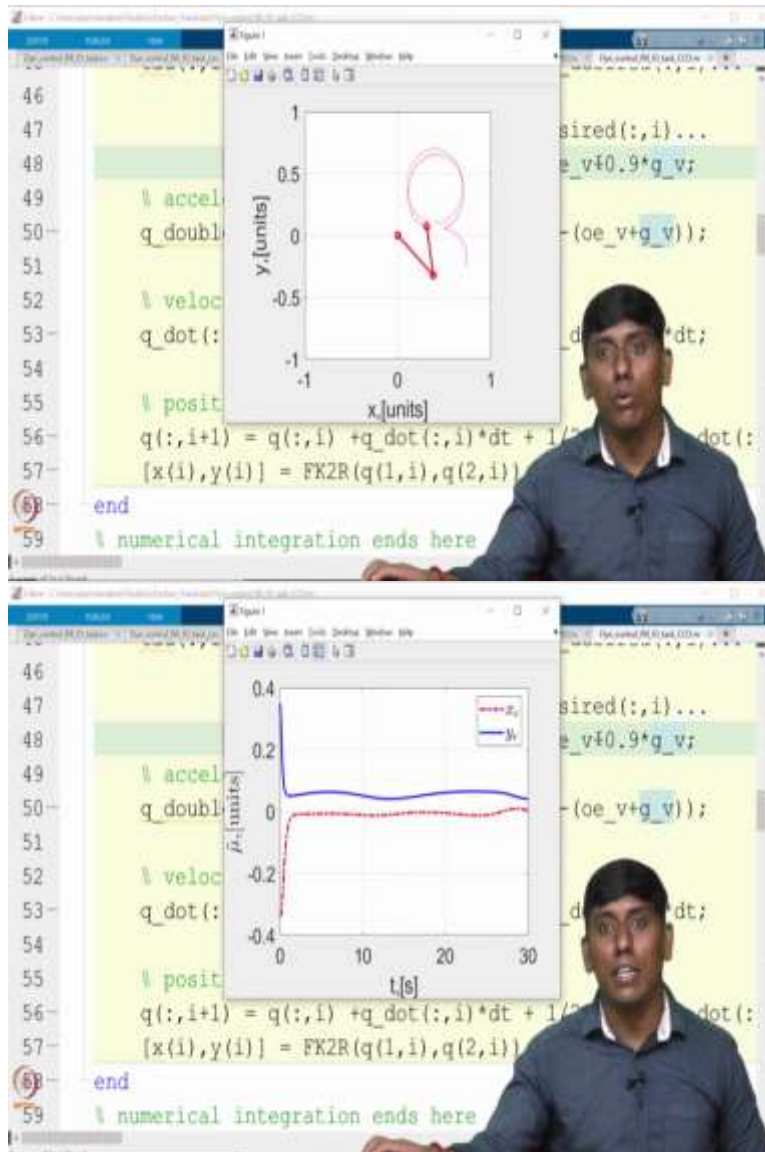
So, the additional sum function we have added here, so, the remaining all are like same. So, in this sense if I ran, so, the same philosophy will be used. So, we assume that a nonzero initial error and circular profile we are trying to follow. So, we are like trying to see that so, you can like see it this way. So, we can like find it this so, now we talk about the error it is like very very smooth.

(Refer Slide Time: 07:34)

```

46 +K1*elddot)...
47 +Jd_inv_dot*(mu_dot_desired(:,i)...
48 +K1*e1)+K2*e2+J'*e1)+oe_v+0.9*g_v;
49 % acceleration vector
50 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+q_v));
51
52 % velocity propagation
53 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
54
55 % position update
56 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*q_double_dot(:,i)*dt^2;
57 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
58 end
59 % numerical integration ends here

```

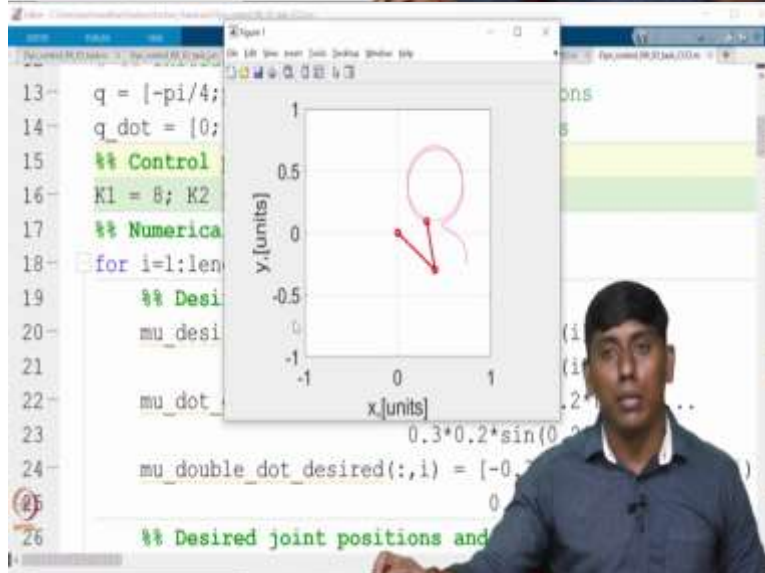


So, now we are like even introducing something like a small hiccup in the sense this is like only 0.9 percent I do not know like 90 percent would be converged, it may end up with unstable. So, you can see like it is trying to follow but still the, you can say it is a model-based control. So that is why you can see the error is coming.

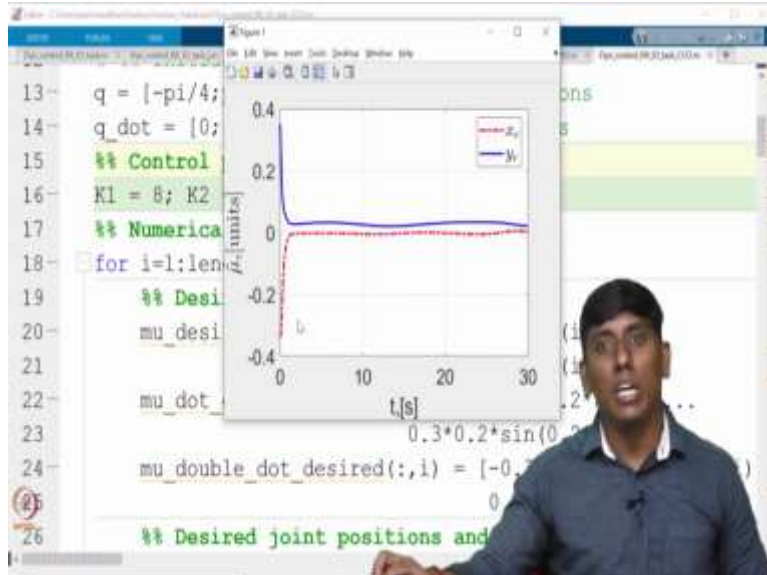
So, now in order to increase the performance or improve the performance by reducing the error either you can increase the  $K_p$   $K_d$  value or in this case  $K_1$  and  $K_2$  value, or we can like introduce the integral error but the cascade control design we did not see the integral error. So, what we can do we can like increase the  $K_1$  or  $K_2$ . So, one is like beneficial because outer loop.

(Refer Slide Time: 08:22)

```
13 q = [-pi/4;pi/3]; % initial joint positions
14 q_dot = [0;0]; % initial joint velocities
15 %% Control parameters
16 K1 = 8; K2 = 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) = [-0.2*cos(0.2*t(i))
25                                    0.2*sin(0.2*t(i))];
26     %% Desired joint positions and
```







So, I am just increasing the K1 value just for our benefit. So, we can see whether it is like improved. So, it is like slightly improved you can see it is not completely better but slightly improved, because we have compensated only 90 percent. So, now you can get understand why it is called model based if your model is inaccurate, you can see it is not tolerated.

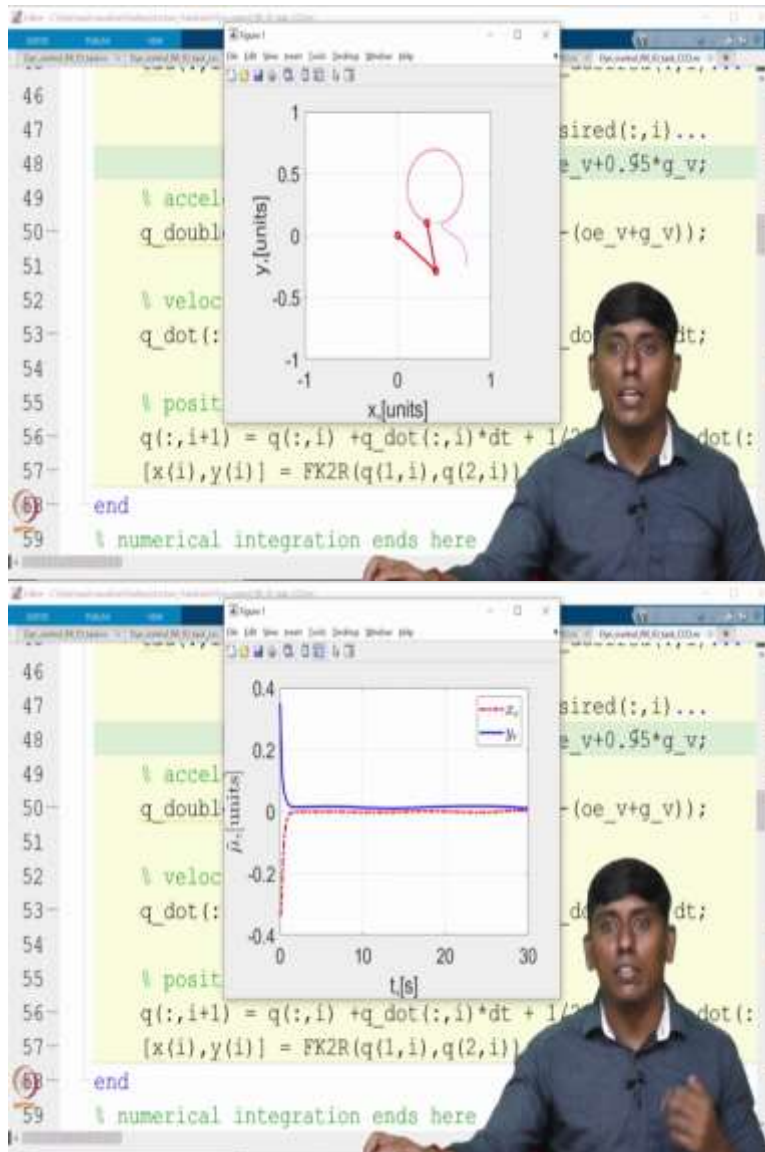
(Refer Slide Time: 08:47)

```

46 +K1*elddot)...
47 +Jd_inv_dot*(mu_dot_desired(:,i)...
48 +K1*e1)+K2*e2+J'*e1)+oe_v+0.95*q_v;
49 % acceleration vector
50 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+q_v));
51
52 % velocity propogation
53 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
54
55 % position update
56 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*q_double_dot(:,i)*dt^2;
57 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
58 end
59 % numerical integration ends here

```

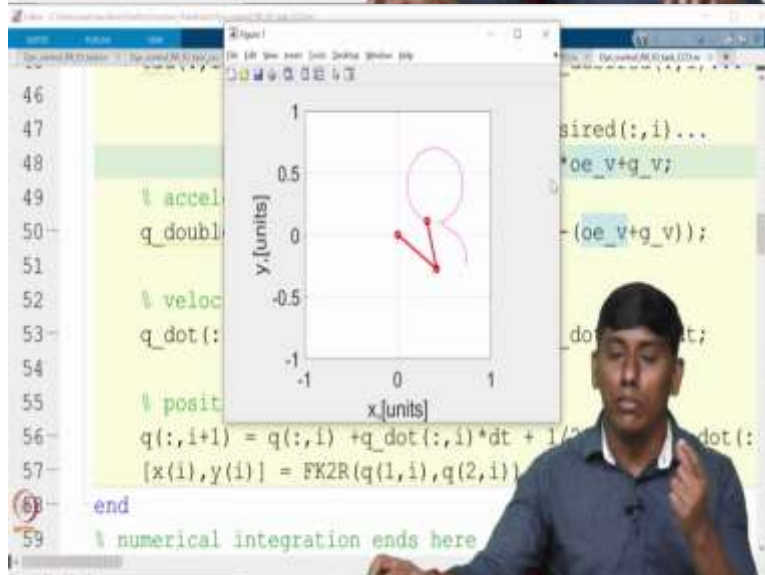


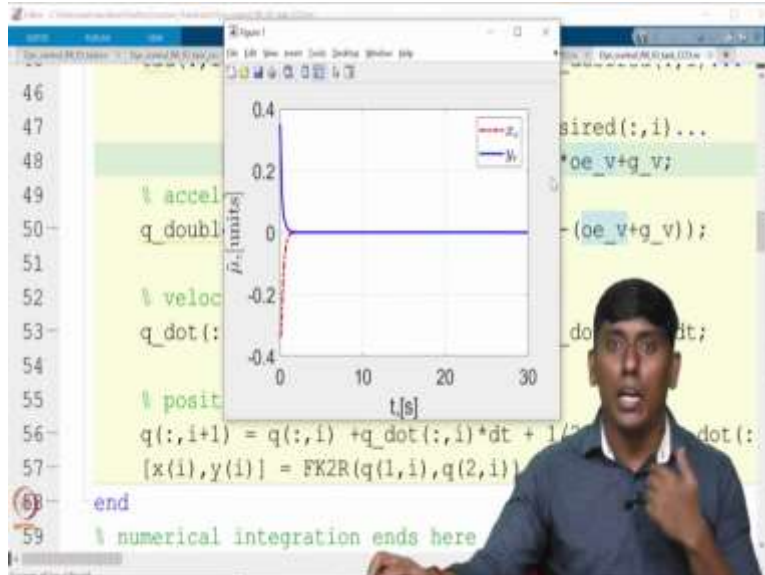


So now, now if I like to make it this is 95 percent or probably 99 percent it may like converge very fast you can see like it is like going very close. So, even you can see a small error, but you can see the error is like improved. So, this is what we can like see. So, why the gravity needs to be compensated you got idea. So, now this is like one thing, but the other effect vectors are like not that significant.

(Refer Slide Time: 09:18)

```
46         +K1*el dot)...
47         +Jd_inv_dot*(mu_dot_desired(:,i)...
48         +K1*e1)+K2*e2+J'*e1)+0*pe_v+g_v;
49     % acceleration vector
50     q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
51
52     % velocity propagation
53     q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
54
55     % position update
56     q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*q_double_dot(:,i)*dt^2;
57     [x(i),y(i)] = FK2R(q(1,i),q(2,i));
58 end
59 % numerical integration ends here
```





Even if I like to make it 0 you can see like that is not contributing because it is very small because the system is like moving in very slow speed. You are a centripetal and Coriolis term is not that significant as the gravity, so that is what the whole idea. So, now I hope you are actually clear what the cascade control design all about.

(Refer Slide Time: 09:39)

```

19 %% Desired values
20 mu_desired(:,i) = [0.4+0.4*sin(0.2*t(i));...
21                  0.4-0.3*cos(0.2*t(i))];
22 mu_dot_desired(:,i) = [0.4*0.2*cos(0.2*t(i));...
23                       0.3*0.2*sin(0.2*t(i))];
24 mu_double_dot_desired(:,i) = [-0.4*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)] = IK2R(mu_desired(:,i));
28 Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
29 q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30 %% Actual dynamic terms
31 M = inertia2R(q(:,i));
32 oe_v = other_effects2R(q(:,i));

```

```

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- K1 = 8; K2 = 4;
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.4*sin(0.2*t(i));

```

```

13- q = [-pi/4;pi/3]; % initial joint positions
14- q_dot = [0;0]; % initial joint velocities
15- %% Control parameters
16- K1 = 8; K2 = 4;
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.9+0.4*sin(0.2*t(i));
21-                       0.4-0.3*cos(0.2*t(i))];
22-     mu_dot_desired(:,i) = [0.4*0.2*cos(0.2*t(i));
23-                            0.3*0.2*sin(0.2*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.4*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

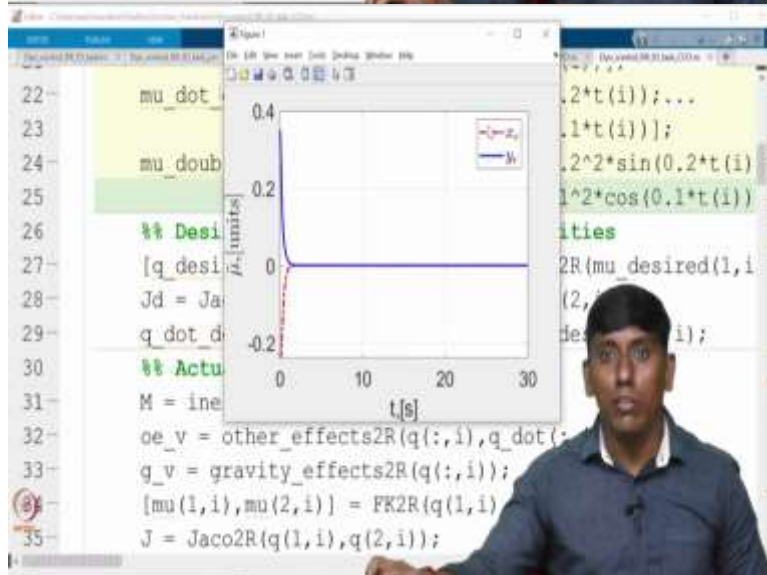
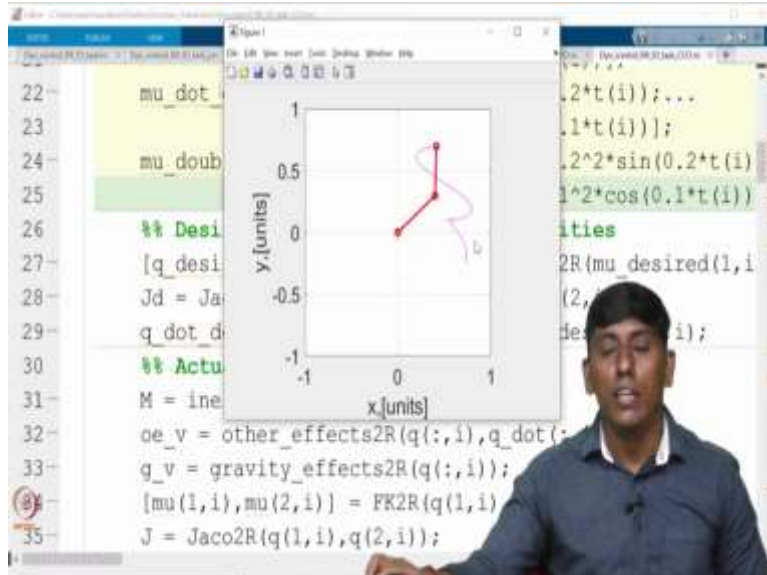
```

```

22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
23-                            0.3*0.1*sin(0.1*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.4*0.2^2*sin(0.2*t(i))
25-                                    0.3*0.1^2*cos(0.1*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 = Jaco2R(q_desired(1,i),q_desired(2,i));
29-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
30-     %% Actual dynamic terms
31-     M = inertia2R(q(:,i));
32-     oe_v = other_effects2R(q(:,i),q_dot(:,i));
33-     g_v = gravity_effects2R(q(:,i));
34-     [mu(1,i),mu(2,i)] = FK2R(q(1,i),q(2,i));
35-     J = Jaco2R(q(1,i),q(2,i));

```





So, now if I like change, so change in the sense I am just trying to see, probably trying to see this can happen or not. I am just trying to see the 0.4 in the sense 0.8 can come but the 0.0 is not obtainable. Because so if the sign is minus 1 what happened this is 0.4 minus 0.4 in the sense it is 0 but what happened your workspace.

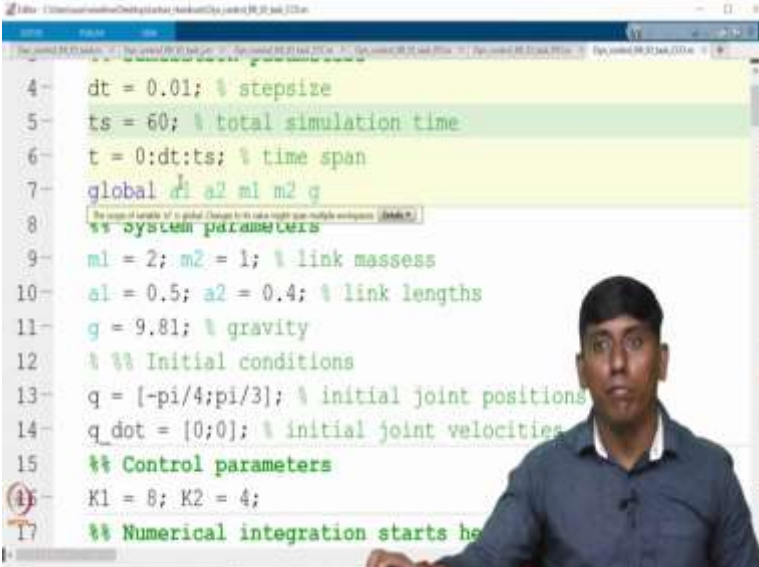
So, workspace is like you can see the inner circle is so  $a_1$  minus  $a_2$  in the sense 0.1 is the inner circle radius. So, that is why it is giving even in the previous lecture we are tried to do it. So, it is giving unnecessary thing because this is like not substitute or not possible. So, in addition to that if I like to improve this so, in the sense it may touch the outer it may work.

So, we can like see, this is also not working because it is extended to the final touch, but you have the J inverse. So, final touch in the sense 0.5 plus 0.4 when this is like becomes 0, it is the entire workspace extended boundary reach. So, that is make it as a one additional singularity, the other one is not possible task space.

So, the other one is like end up with what you call the boundary singularity. So, in the sense Jacobian would be, you can say, like a singular one. So, in the sense this is also not possible. So now you got why it is like not getting it. So, you have idea. So, now the similar direction I am trying here, so I will just make it this is 0.1.

So, 0. 1, maybe you can see 0.2, itself but this is 0. 1 I am trying. So, I hope this will work. I hope so. So, I am not clear whether that 8 profile can be generated here. So, I am just trying to see that. Yeah, you can see like I am trying to follow 8 profile if I increase the you can see the time range it will like make 8 profile.

(Refer Slide Time: 11:57)



```
4- dt = 0.01; % stepsize
5- ts = 60; % total simulation time
6- t = 0:dt:ts; % time span
7- global d1 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- K1 = 8; K2 = 4;
17- %% Numerical integration starts here
```

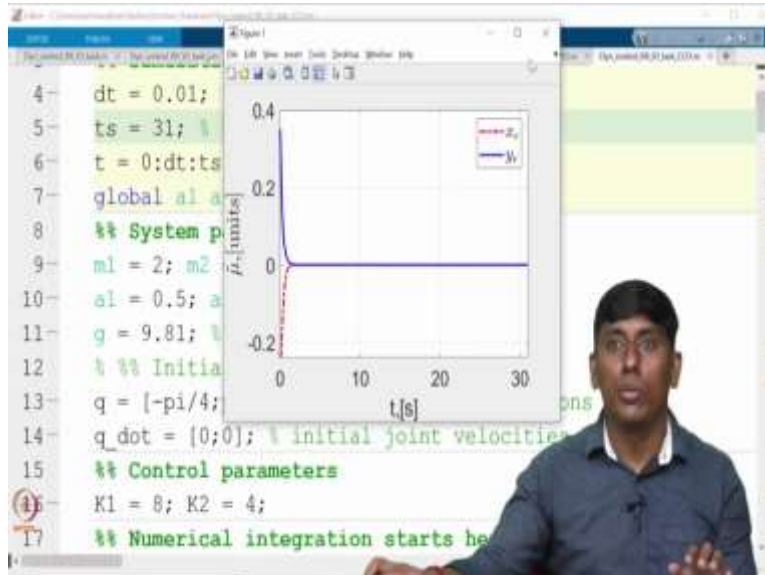
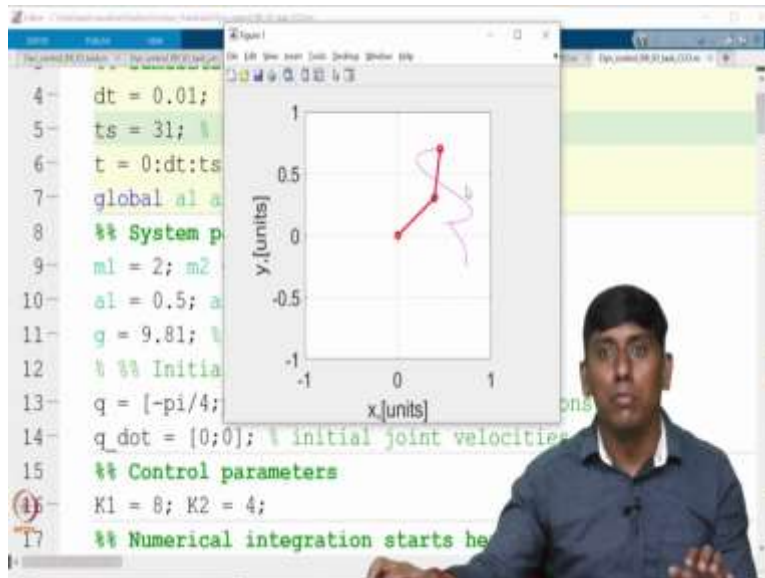


```
4 dt = 0.01; % stepsize
5 ts = 40; % total simulation time
6 t = 0:dt:ts; % time span
7 global a1 a2 m1 m2 g
8 %% System parameters
9 m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11 g = 9.81; % gravity
12 %% Initial conditions
13 q = [-pi/4; pi/3]; % initial joint positions
14 q_dot = [0; 0]; % initial joint velocities
15 %% Control parameters
16 K1 = 8; K2 = 4;
17 %% Numerical integration starts here
```



```
4 dt = 0.01; % stepsize
5 ts = 31; % 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
15 %% Control parameters
16 K1 = 8; K2 = 4;
17 %% Numerical integration starts here
```





So, you can like see if I increase the time, which is like I will give 60 seconds. So, you can see like it like make 8 profile. I think it is not because there is a converge, the converge end is like making it so I am trying to show that. So, where it can end up so where this point is like intersect the middle point. So, 31.4 so I will just put 31 you can like see it, it is coming closer. So, when this point is like started. So, the workspace is like getting somewhat singular, so that is why it is not coming in.

(Refer Slide Time: 12:41)

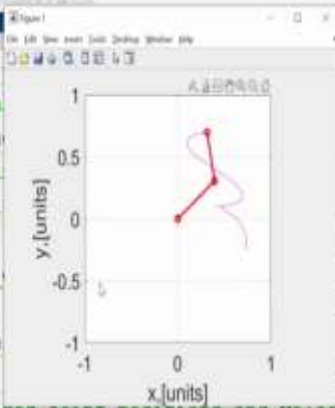
```
16- K1 = 8; K2 = 4;
17- %% Numerical integration starts here
18- for i=1:length(t)
19-     %% Desired values
20-     mu_desired(:,i) = [0.4+0.3*sin(0.2*t(i));...
21-                       0.4-0.3*cos(0.1*t(i))];
22-     mu_dot_desired(:,i) = [0.3*0.2*cos(0.2*t(i));...
23-                            0.3*0.1*sin(0.1*t(i))];
24-     mu_double_dot_desired(:,i) = [-0.4*0.2*cos(0.2*t(i))...
25-                                    0.3*0.1*cos(0.1*t(i))];
26-     %% Desired joint positions and velocities
27-     [q_desired(1,i),q_desired(2,i)] = mu_desired(:,i);
28-     Jd = Jaco2R(q_desired(1,i),q_desired(2,i));
29-     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(:,i);
```

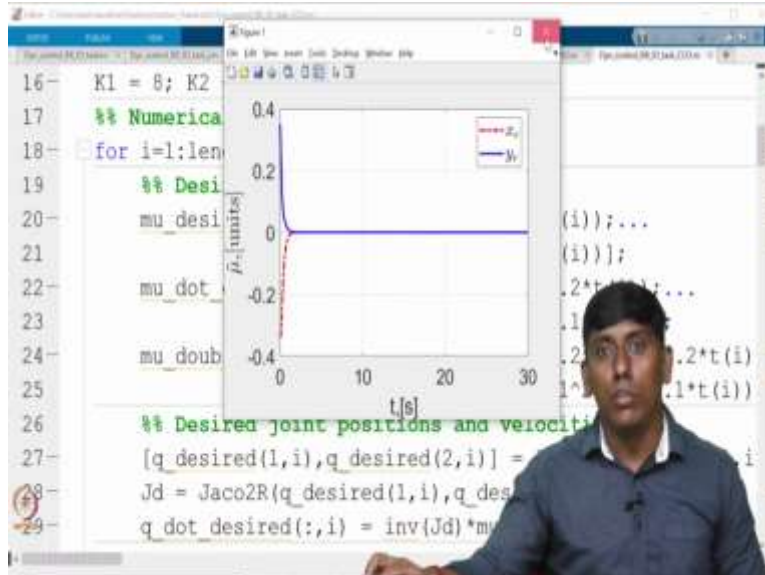
```
1  %% Double-loop motion control of a RR planar robot
2  clear all; close all; clc;
3  %% Simulation parameters
4  dt = 0.01; % stepsize
5  ts = 35; % total simulation time
6  t = 0:dt:ts; % time span
7  global a1 a2 m1 m2 g
8  %% system parameters
9  m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2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

```
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
```



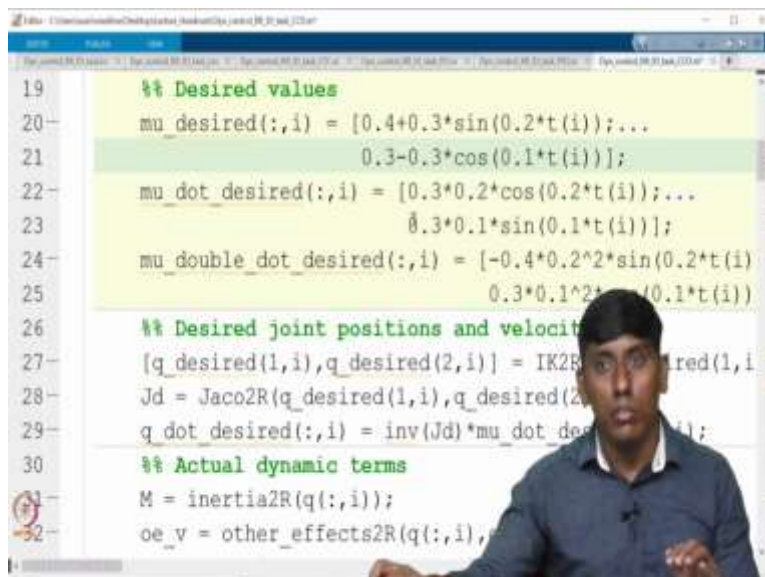
```
16 K1 = 8; K2 = 8;
17 %% Numerical integration
18 for i=1:length(t)-1
19     %% Desired joint positions and velocities
20     mu_desired(i) = 0.5*cos(2*t(i));
21     mu_dot_desired(i) = -1*t(i);
22     mu_doublet(i) = 1*t(i);
23     %% Desired joint positions and velocities
24     [q_desired(1,i), q_desired(2,i)] = mu_desired(i);
25     Jd = Jaco2R(q_desired(1,i), q_desired(2,i));
26     q_dot_desired(:,i) = inv(Jd)*mu_dot_desired(i);
```





So, I will just make it this is 0.4, itself. So, now I will just make it properly increase the time I just want to see whether the 8 profile I could show you. No, it is not possible. So, then I can like make it at least so there is 8 profile, the half of the 8 profile we can like see, so in order to make it properly how to bring the while I will see whether that is possible because the solution may like extend that also can go.

(Refer Slide Time: 13:18)





The top image shows a MATLAB script on the left and a plot on the right. The script includes the following code:

```

19 %% Desi
20 mu_desi
21
22 mu_dot
23
24 mu_doub
25
26 %% Desi
27 [q_desi
28 Jd = Ja
29 q_dot_d
30 %% Actual dynamic terms
31 M = inertia2R(q(:,i));
32 oe_v = other_effects2R(q(:,i),

```

The plot displays a 2D trajectory in the x-y plane. The x-axis is labeled 'x [units]' and ranges from -1 to 1. The y-axis is labeled 'y [units]' and ranges from -1 to 1. The trajectory starts at the origin (0,0) and moves in a complex, oscillatory path, ending near (0.5, 0.5).

The bottom image shows the same MATLAB script as above, but with a different plot. The script code is identical to the top image.

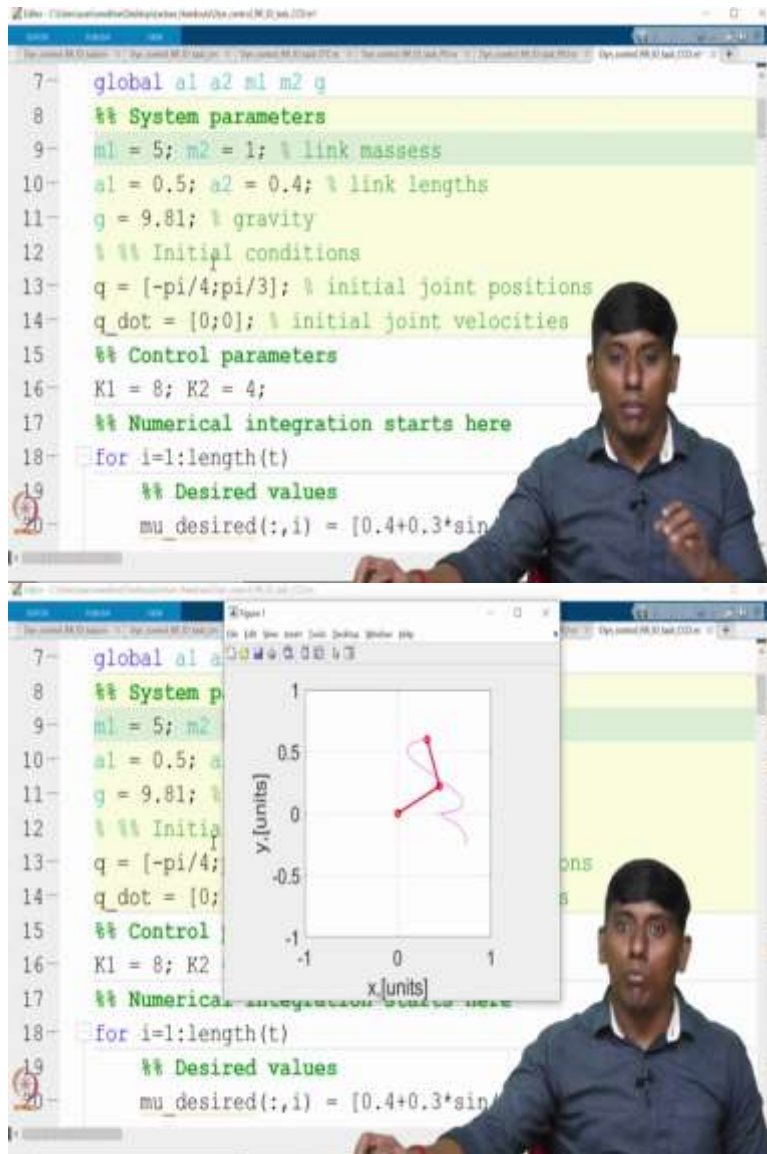
```

19 %% Desi
20 mu_desi
21
22 mu_dot
23
24 mu_doub
25
26 %% Desi
27 [q_desi
28 Jd = Ja
29 q_dot_d
30 %% Actual dynamic terms
31 M = inertia2R(q(:,i));
32 oe_v = other_effects2R(q(:,i),

```

The plot displays the convergence of  $\mu_x$  and  $\mu_y$  over time. The x-axis is labeled 't [s]' and ranges from 0 to 30. The y-axis is labeled ' $\mu_x$  [units]' and ranges from -0.4 to 0.2. The plot shows two curves: a red curve for  $\mu_x$  and a blue curve for  $\mu_y$ . Both curves start at 0, drop sharply to approximately -0.35, and then converge to 0 by about 5 seconds.





So, I am just saying that this is probably 0.3. So, then the Y can go 0. So, you can see like this is like trying to follow an 8 kind of profile. So, which is a spiral you can see. So, now you can see the cascade design where you can like change your K1 and K2 the performance will vary and similarly even you change the model parameter for example, this is 5 kg and still this will work very comfortably.

So, these all we have like try to see. So, with that what we can see we have seen the motion control overall go so where we talked about joint space scheme then task space scheme, the finally we end up with the merge which is what you call the cascaded the outer loop would be seeing the task space and the inner loop would be in the joint space.

So, in that sense that controller which is actually going to give a command to the actuator directly given in the joint space which is much beneficial. The other way around where you are seeing that the task based need to be followed that is also ensured. So, that is why this cascade control design is very popular in the modern days, because you can directly control the task space, but the actuator signal directly given on the joints space level.

So, that is what the whole idea I hope you have like enjoyed this particular lecture. So, the next lecture would be on completely different. Where we are taking a mobile robot, but we are trying the, you can say Denavit-Hartenberg approach how to derive the kinematic and dynamic model and then see how the controller can be endorsed in that particular scheme. So, with that, I am saying thank you and see you then bye. Take care.