

Mechanics and Control of Robotic Manipulators
Professor Santhakumar Mohan
Department of Mechanical Engineering
Indian Institute of Technology, Palakkad
Lecture: 42
Cascaded Control Design

Hi, welcome back to Mechanics and Control of Robotic Manipulator. In the last class we have seen dynamic control. So, we started with inverse dynamics, and you can see dynamic control based on motion based and model based the motion based which we have seen is PD and PID. The model based which we have seen is like computer torque control. In this particular lecture, we are going to see the MATLAB simulation on this dynamic control and the dynamic control simulations we are going to see that too we are going to restrict to one only that joint space. So, the task space we will see in upcoming lectures, but now, we are going to see the dynamic control simulations in joint space.

(Refer Slide Time: 00:55)

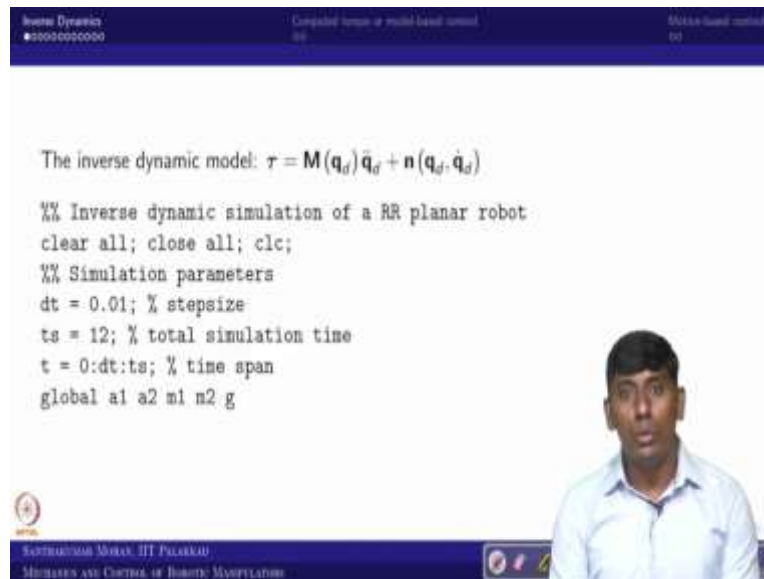


So, in that sense we will start what is inverse dynamics. So, we know like inverse dynamics aspect and inverse dynamics will work as an open loop or feed forward control if both the initial and you can say initial position and as well as desired positions are same and you can see similar way the initial and desired velocities are same and non zero acceleration are there then it will work good.

So, that is what we have seen, then we have started talking about second order error dynamics is stable then we derived what you call computer torque control which some people call PD with feed forward and feedback linearization control some people simply call model-based control then we have seen a motion based where the acceleration is desired acceleration is not known and the model parameters are not known.

Then we can do provided we assume that if the spatial system, then the gravity needs to be compensated otherwise the system performance may not be as what we expected. So, then we can say PD plus gravity compensation our PID plus gravity compensation these all we have seen.

(Refer Slide Time: 02:43)



```

Inverse Dynamic
#0000000000
Control torque or model based control
Model based control
00

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

%% Inverse dynamic simulation of a RR planar robot
clear all; close all; clc;
%% Simulation parameters
dt = 0.01; % stepsize
ts = 12; % total simulation time
t = 0:dt:ts; % time span
global a1 a2 m1 m2 g
  
```

So, the same thing we are trying to see in MATLAB simulation. The first thing we are trying to do the inverse dynamic model so, where you can say tau given in the desired values. So, M of q desired into q desired double dot plus n of q desired comma q desired dot if these all known because q desired q desired dot and q desired double dot are known. So, we can calculate M desired and as well as n desired then we can apply what you call you can say inverse dynamic model. So, this is what this is something like F equal to Ma so, that kind of thing.

So, for that we are writing you call MATLAB code so, which is nothing but inverse dynamic simulation of RR serial planar robot. So, here we are assuming there are certain variables which are global. So, last class or you can say it in the kinematic control class we have seen the L1 and L2 are global. So, where that we can use in some of the sub function. Similarly, here m1 m2 g

and as well as a_1 in a_2 so, instead of L_1 and L_2 we use a_1 and a_2 . So, these are global. Further what we assume that the total simulation time here is a 12 second and the step size is 10 millisecond and duration of the total time span start from 0 to t_s . So, these all we have conventionally see.

(Refer Slide Time: 03:30)

```

%% System parameters
m1 = 2; m2 = 1; % link masses
a1 = 0.5; a2 = 0.4; % link lengths
g = 0*9.81; % gravity
%% Initial conditions
q = [0; pi/3 - pi/4]; % initial joint positions
q_dot = [0.2*pi/2; 0]; % initial joint velocities

```

So, now for demonstrating we have taken one of the same codes which we have done in the dynamic simulation where mass of the first link is 2 kilogram and mass of second link is 1 kilogram and the length of the first link is 0.5 and the second link is 0.4. So, link length these all we have taken and then we have gone further. So, for inverse dynamics, I already said if we introduce gravity there would be a small you can say hiccup will come. So, in order to avoid that I assume that there is no gravity acting in the sense that gravity is balanced. So, then we need to identify what would be the initial condition.

So, here I assume that the initial joint position actual joint position is as equal to desired. So, probably when we come to the, that part you can see why it is we have taken as this way. So, π by 3 minus π by 4 as the, you can say θ_2 initial and why it is a take and so, on. So, 0.2 times of π by 2 as the joint velocity that we will see in the next you can say continuous slides.

So, before going to talk about that so, you can see this is the q desired so, the q desired we have taken as a two sinusoidal signal. So, the θ_1 is varying $\pi/2$ in the sense it is $\pi/2$ minus $\pi/2$ plus $\pi/2$ with a frequency of 0.2 times of t . So, similarly the θ_2 is varying. So, $\pi/4$ oscillations, but it is varying from $\pi/3$.

So, in that sense you can see like now why this initial condition has come $\pi/3$ minus $\pi/4$ because if you take θ like a time is 0 so, then θ_1 would be 0 and θ_2 would be $\pi/3$ minus $\pi/4$ the similar way you can see like a \dot{q} desired. So, if you substitute the time equal to 0 then you can see, so, 0.2 times $\pi/2$ would be the $\dot{\theta}_1$ and $\dot{\theta}_2$ at 0 that is why we have taken this.

So, now, based on this the \ddot{q} desired all those things we have used. So, further the dynamic model instead of writing it everything inside the, you can say for loop so, we have made some kind of sub function which is easy so, far that we are taking first the inertia matrix which is we call $\text{inertia}2R$.

So, which is going to give inertia matrix based on the, you can say q the q_1 is θ_1 and q_2 is θ_2 . So, we can calculate the inertia matrix in this case it is the two or serial manipulator which is 2×2 the same way we can see other effects which we have written in our lecture which was v of q comma \dot{q} so that is what we have written as a sub-function which is we call other effects vector.

So, now, this is also like depend. So, here you can see like both θ s and $\dot{\theta}$ s are independent. So, we have you can say call q comma \dot{q} . So, similar direction we can take the gravity efforts. So, q would be the input then we can get the gravity vector g of g underscored v we can find it. So, further in order to make a much more you can say beneficial. So, we are using Jacobian you can say sub function which we derived in the kinematic control.

(Refer Slide Time: 06:51)

```
function [x,y] = FK2R(th1,th2)
global a1 a2
    x = a1*cos(th1)+a2*cos(th1+th2);
    y = a1*sin(th1)+a2*sin(th1+th2);
end
```



Systems Dynamics
000000000000

Computed torque or model based control
on

Model based control
on


Supratosh Mishra, IIT Palakkad
Mechanics and Control of Robotic Manipulators

Systems Dynamics
000000000000

Computed torque or model based control
on

Model based control
on

```
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
```



Systems Dynamics
000000000000

Computed torque or model based control
on

Model based control
on

Supratosh Mishra, IIT Palakkad
Mechanics and Control of Robotic Manipulators

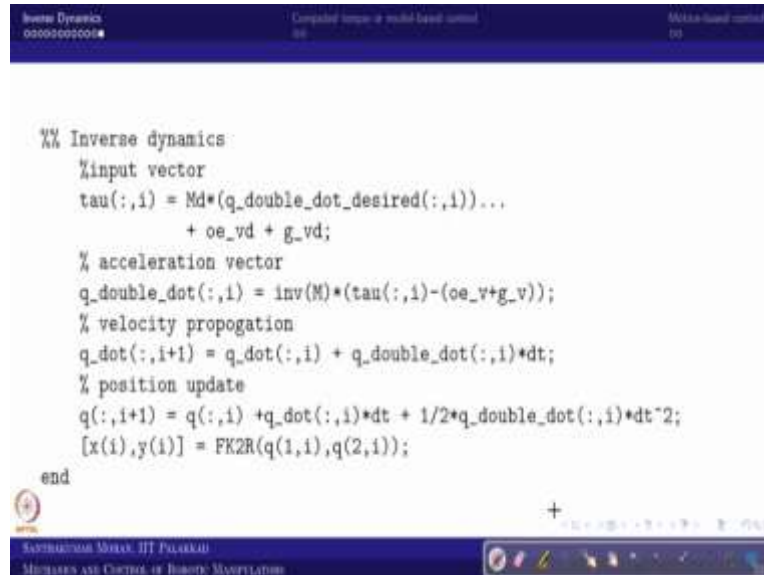
Systems Dynamics
000000000000

Computed torque or model based control
on

Model based control
on

and the gravity vectors we have like found based on the actual q \dot{q} . So, now once these all available so, where we can go, we can try to find out what is τ .

(Refer Slide Time: 07:48)



```

Inverse Dynamics
oooooooooooo
Computed torque in model based control
00
Model-based control
00

%% Inverse dynamics
%input vector
tau(:,i) = Md*(q_double_dot_desired(:,i))...
        + oe_vd + g_vd;
% acceleration vector
q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_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

```

SAPTAJAYAM MOHAN, IIT PALAKKAD
 MODELING AND CONTROL OF ROBOTS MANIPULATORS

So, τ in this case it is simple what you call inverse dynamics. So, inverse dynamics what it says the M desired into q double dot desired plus other vector desired plus gravity vector desired. So, now, we have given that as the input vector here, so, then we are going the straightforward numerical integration.

So, first we find the acceleration vector the τ minus of other vector you can say plus gravity vector, and this is M inverse these all whereas, the τ would be having all the desired, so, then the velocity would be propagated based on this and the; you can say position update or position propagation is based on this.

Then I want to plot something like animated way. So, I want to find what would be the actual values. So, I am just taking you can say forward kinematics for the just to show the animation of the forward dynamic model simulation. So, now, based on this if we go to the MATLAB, you can say simulation.

(Refer Slide Time: 08:50)

```
1 %% Inverse dynamic simulation of a RR planar robot
2 clear all; close all; clc;
3 %% Simulation parameters
4 dt = 0.01; % stepsize
5 ts = 12; % 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 = [0;pi/3-pi/4]; % initial joint positions
14 q_dot = [0.2*pi/2;0]; % initial joint velocities
```

```
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 = [0;pi/3-pi/4]; % initial joint positions
14 q_dot = [0.2*pi/2;0]; % initial joint velocities
15 %% Numerical integration starts here
16 for i=1:length(t)
17     %% Desired values
18     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
19                     pi/3-pi/4*cos(0.5*t(i))];
20     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i));...
21                          0];
```

```
16- for i=1:length(t)
17-     %% Desired values
18-     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
19-                     pi/3-pi/4*cos(0.5*t(i))];
20-     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i));...
21-                          0.5*pi/4*sin(0.5*t(i))];
22-     q_double_dot_desired(:,i) = [-0.2^2*pi/2*sin(0.2*t(i))
23-                                   0.5^2*pi/4*cos(0.5*t(i))];
24-     Md = inertia2R(q_desired(:,i));
25-     oe_vd = other_effects2R(q_desired(:,i), q_dot_desired(:,i));
26-     g_vd = gravity_effects2R(q_desired(:,i));
27-     M = inertia2R(q(:,i));
28-     oe_v = other_effects2R(q(:,i), q_dot(:,i));
29-     g_v = gravity_effects2R(q(:,i));
```



```
22-     q_double_dot_desired(:,i) = [-0.2^2*pi/2*sin(0.2*t(i))
23-                                   0.5^2*pi/4*cos(0.5*t(i))];
24-     Md = inertia2R(q_desired(:,i));
25-     oe_vd = other_effects2R(q_desired(:,i), q_dot_desired(:,i));
26-     g_vd = gravity_effects2R(q_desired(:,i));
27-     M = inertia2R(q(:,i));
28-     oe_v = other_effects2R(q(:,i), q_dot(:,i));
29-     g_v = gravity_effects2R(q(:,i));
30-
31-     %% Inverse dynamics
32-     \input vector
33-     tau(:,i) = Md*(q_double_dot_desired(:,i))
34-               + oe_vd + g_vd;
35-     \ acceleration vector
```





```
25- oe_vd = other_effects2R(q_desired(:,i),q_dot_desired(:,i));
26- g_vd = gravity_effects2R(q_desired(:,i));
27- M = inertia2R(q(:,i));
28- oe_v = other_effects2R(q(:,i),q_dot(:,i));
29- g_v = gravity_effects2R(q(:,i));
30
31- %% Inverse dynamics
32- %input vector
33- tau(:,i) = M*d*(q_double_dot_desired(:,i))
34-           + oe_vd + g_vd;
35- % acceleration vector
36- q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v + g_v));
37
38- % velocity propogation
```

So, we will go to MATLAB code. So, you can see this is the code which we have shown there. So, you can see like this is inverse dynamics simulation of RR planar serial manipulator or robot. So, these are the cases so, now, we are taking it these all. So, here we assume that gravity is 0 later on we will bring it and here we are not considering the frictional effect because the frictional effect is not exactly model it.

So, that is why we know here the frictional effect then we are getting the \dot{q} desired d desired \ddot{q} desired. So, based on this so, then we are calculating the desired value. Based on desired value what would be the dynamic model and actual value what would be the dynamic model.

(Refer Slide Time: 09:33)

```
31 %% Inverse dynamics
32 %input vector
33 tau(:,i) = Md*(q_double_dot_desired(:,i))...
34           + oe_vd + q_vd;
35 % acceleration vector
36 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
37
38 % velocity propogation
39 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
40
41 % position update
42 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*q_double_dot(:,i)*dt^2;
43 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
44 end

46 %% Animation
47 for i=1:10:length(t)
48     x1 = a1*cos(q(1,i));
49     y1 = a1*sin(q(1,i));
50     x2 = x1+a2*cos(q(1,i)+q(2,i));
51     y2 = y1+a2*sin(q(1,i)+q(2,i));
52     plot([0,x1,x2],[0,y1,y2],'r-o','linewidth',2)
53     grid on, set(gca,'fontsize',20)
54     hold on
55     plot(x(1:i),y(1:i),'m-')
56     axis([- (a1+a2)-0.1, (a1+a2)+0.1, - (a1+a2), (a1+a2)+0.1])
57     axis square, xlabel('t, [s]'),ylabel('q, [rad]')
58     hold off, pause(0.001)
59 end
```

```
58- hold off, pause(0.001)
59- end
60- %% Plotting functions
61- plot(t,q(1,1:i),'r-',t,q(2,1:i),'b-','linewidth',2)
62- legend('\theta_1','\theta_2')
63- grid on
64- set(gca,'fontsize',20)
65- xlabel('t,[s]');
66- ylabel('q,[units]')
67-
68- function M = inertia2R(q)
69- global a1 a2 m1 m2
70- th1 = q(1); th2 = q(2);
    \ Inertia matrix

14- q_dot = [0.2*pi/2;0]; % initial joint velocities
15- %% Numerical integration starts here
16- for i=1:length(t)
17-     %% Desired values
18-     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
19-                     pi/3-pi/4*cos(0.5*t(i))];
20-     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i));...
21-                          0.5*pi/4*sin(0.5*t(i))];
22-     q_double_dot_desired(:,i) = [-0.2^2*pi/2*cos(0.2*t(i));...
23-                                   0.5^2*pi/4*cos(0.5*t(i))];
24-     Md = inertia2R(q_desired(:,i));
25-     oe_vd = other_effects2R(q_desired(:,i), q_dot_desired(:,i));
26-     g_vd = gravity_effects2R(q_desired(:,i));
27-     M = inertia2R(q(:,i));
```



```

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 = [0;pi/3;pi/4]; % initial joint positions
14 q_dot = [0.2*pi/2;0]; % initial joint velocities
15 %% Numerical integration starts here
16 for i=1:length(t)
17     %% Desired values
18     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
19                     pi/3-pi/4*cos(0.5*t(i))];
20     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i));...
21                          0.5*pi/4*sin(0.5*t(i))];

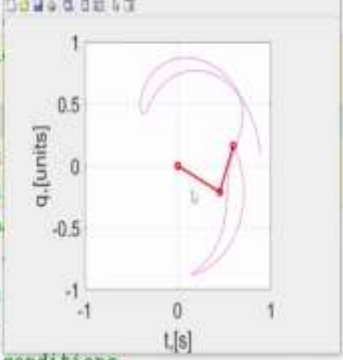
```

So, then we found the input vector and all the things so, finally we want to plot animate. So, we are taking the animated stuff here. So, then we are just plotting it, and these are the sub functions which we are shown that. So, now if I run, so you can see like this is going to be the first theta 1 is going to vary plus or minus, you can say pi by 2.

So, it starts from 0 and it goes pi by 2 then it comes back. So, we will see similarly the other one is it is going to vary between you can say pi by 4, but with the addition of pi by 3 and the velocity also going so right now we take everything is idealistic the sense the initial condition and initial velocity and the initial position are as similar as the desired initial position and the gravity we make it a 0.

(Refer Slide Time: 10:28)

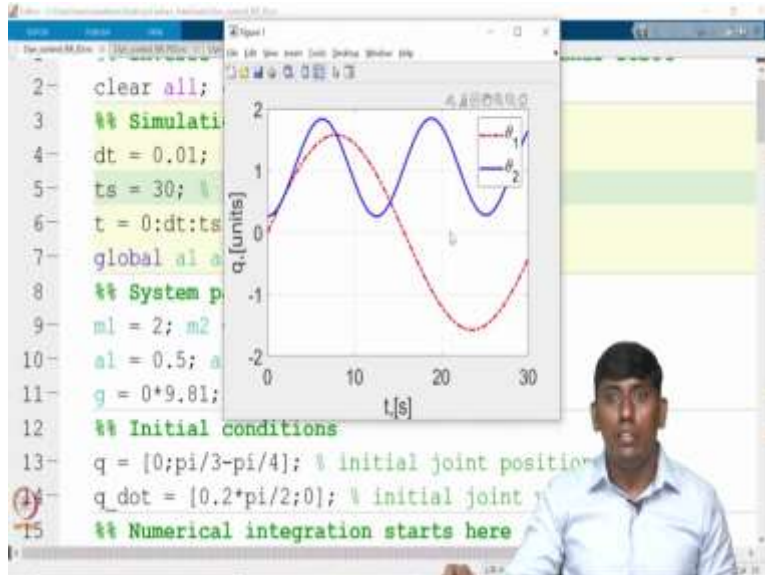
```
2 clear all; close all; clc;
3 %% Simulation parameters
4 dt = 0.01; % stepsize
5 ts = 30; % total simulation time
6 t = 0:dt:ts; % time span
7 global a1 a2 m1 m2 g
8 %% System parameters
9 m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11 g = 0*9.81; % gravity
12 %% Initial conditions
13 q = [0;pi/3-pi/4]; % initial joint positions
14 q_dot = [0.2*pi/2;0]; % initial joint velocities
15 %% Numerical integration starts here
```



The figure shows a MATLAB script editor with a code window and a Scope window. The code window contains the following MATLAB code:

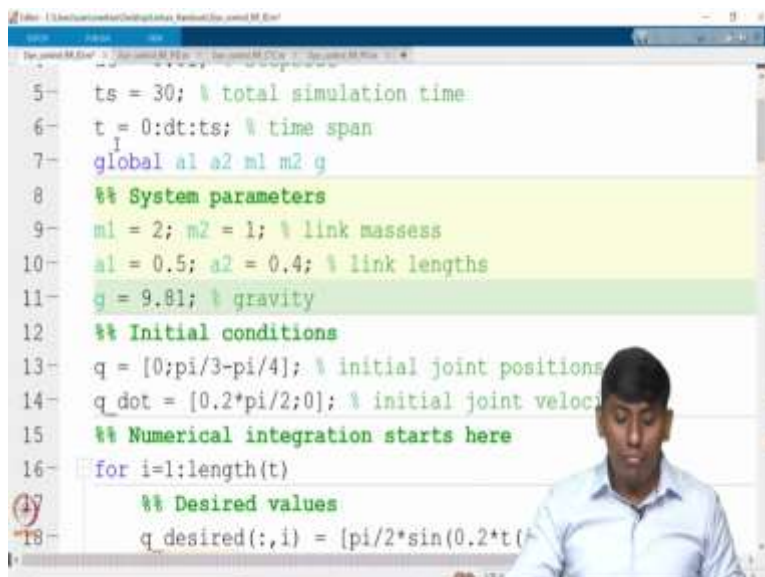
```
2 clear all; close all; clc;
3 %% Simulation parameters
4 dt = 0.01; % stepsize
5 ts = 30; % total simulation time
6 t = 0:dt:ts; % time span
7 global a1 a2 m1 m2 g
8 %% System parameters
9 m1 = 2; m2 = 1; % link masses
10 a1 = 0.5; a2 = 0.4; % link lengths
11 g = 0*9.81; % gravity
12 %% Initial conditions
13 q = [0;pi/3-pi/4]; % initial joint positions
14 q_dot = [0.2*pi/2;0]; % initial joint velocities
15 %% Numerical integration starts here
```

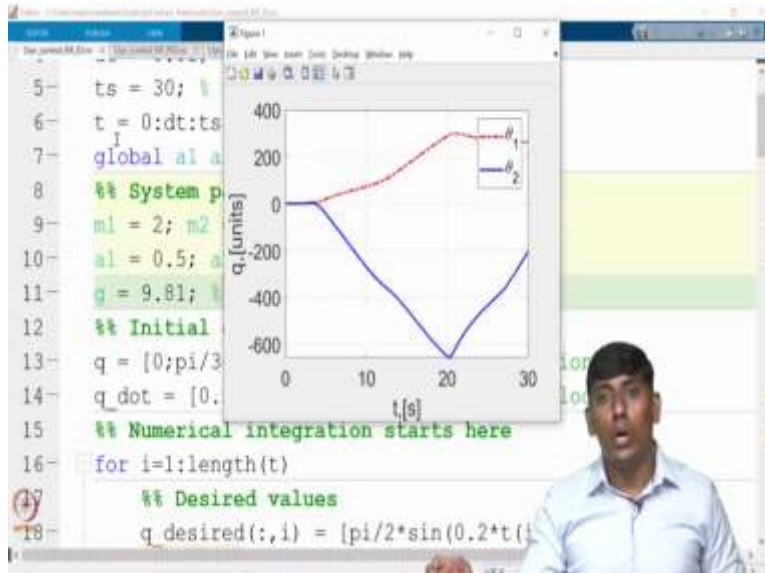
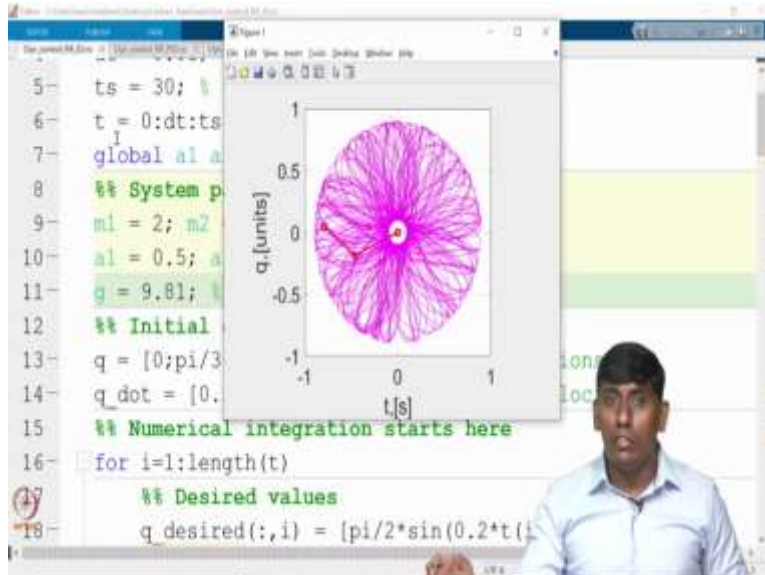
The Scope window displays a plot of joint position q (in units) versus time t (in seconds). The plot shows a red trajectory starting at $(0,0)$ and moving towards $(1,0.5)$ over 1 second. A pink arc represents the path of the end effector.



So, then we are trying to run so I will just make it this is probably at least, so 30 second. So, now if I run so, I can, supposed to be run this. So, if I run this you can see. So, this is what you can see the theta 1 is rotating between plus pi by 2 minus pi by 2 you would have seen, and this is rotating this is you can see like a plus pi by 2 and minus pi by 2. The other one is you can see the mean is pi by 3 on top pi by 4 plus or minus. So, now this is the; you can see inverse dynamics.

(Refer Slide Time: 11:10)

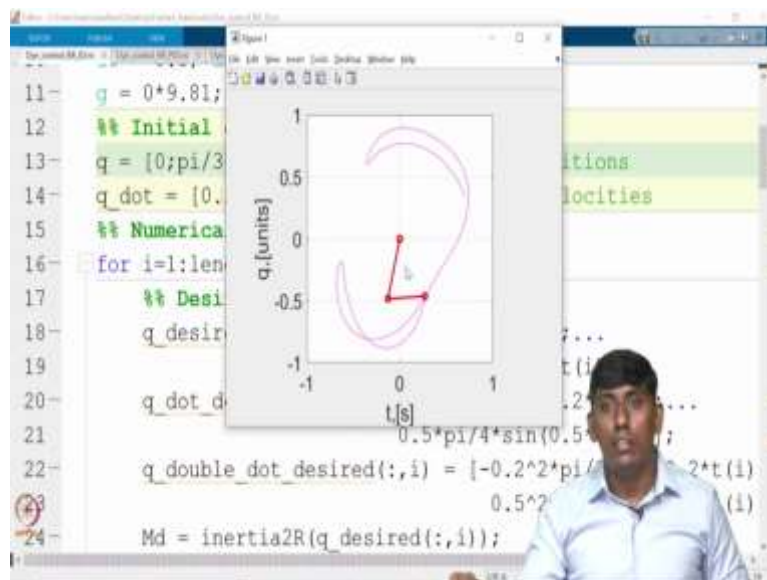


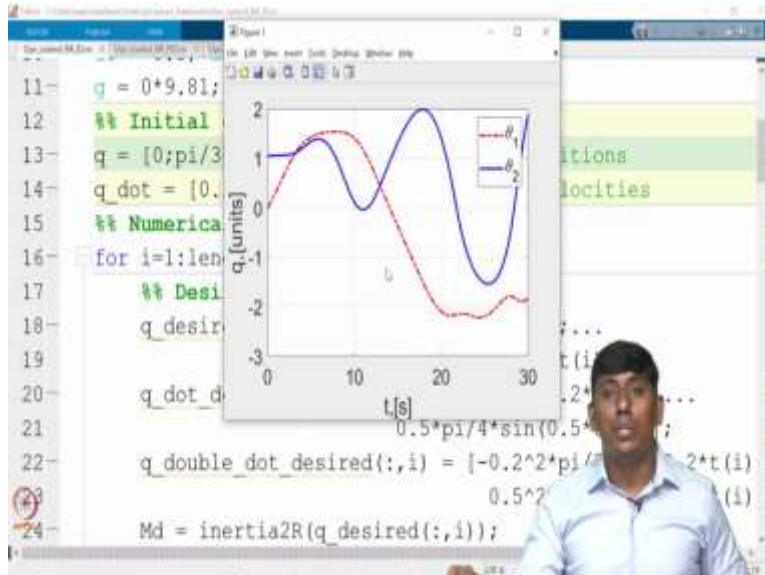


So, now if I introduce the gravity, it may not work what we expected. So, that what you can see, so you can see right it is making it completely different. So, this is due to you can see there is a non expected thing because the gravity the compensation is not exactly going to compensate based on the actual because the propagation model is one step forward so, that we cannot be able to predict. So, now you can understand why the gravity we consider as 0 or when we do the open loop control with you can say you call inverse dynamics this is may not be working as good.

(Refer Slide Time: 11:49)

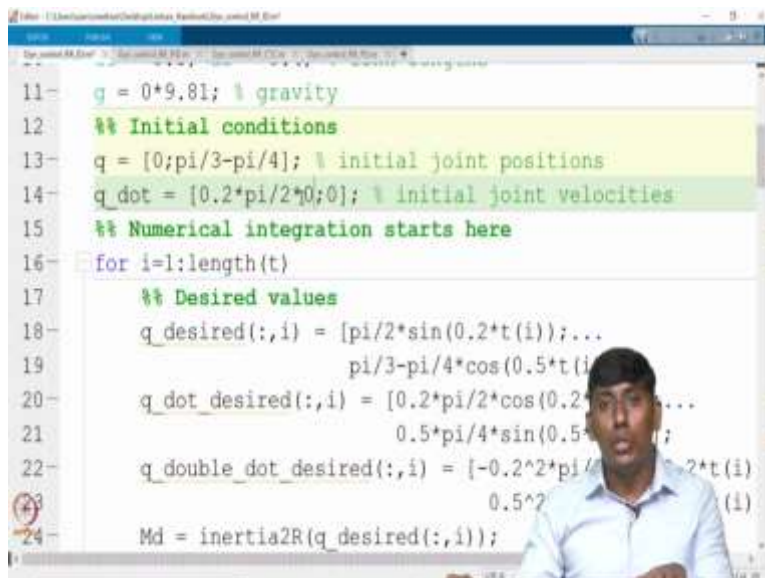
```
11- g = 0*9.81; % gravity
12- %% Initial conditions
13- q = [0;pi/3-pi/4*0]; % initial joint positions
14- q_dot = [0.2*pi/2;0]; % initial joint velocities
15- %% Numerical integration starts here
16- for i=1:length(t)
17-     %% Desired values
18-     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
19-                     pi/3-pi/4*cos(0.5*t(i))];
20-     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i))...
21-                          0.5*pi/4*sin(0.5*t(i))];
22-     q_double_dot_desired(:,i) = [-0.2^2*pi/2*cos(0.2*t(i))...
23-                                   0.5^2*pi/4*cos(0.5*t(i))];
24-     Md = inertia2R(q_desired(:,i));
```

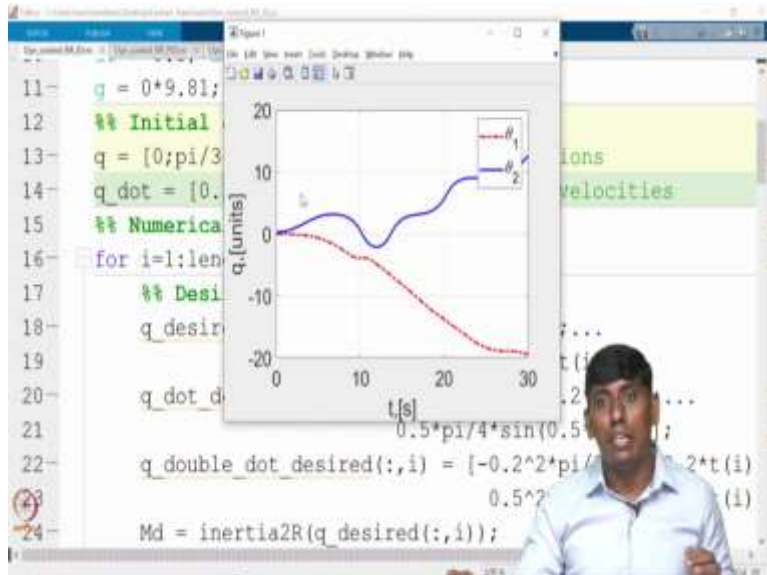
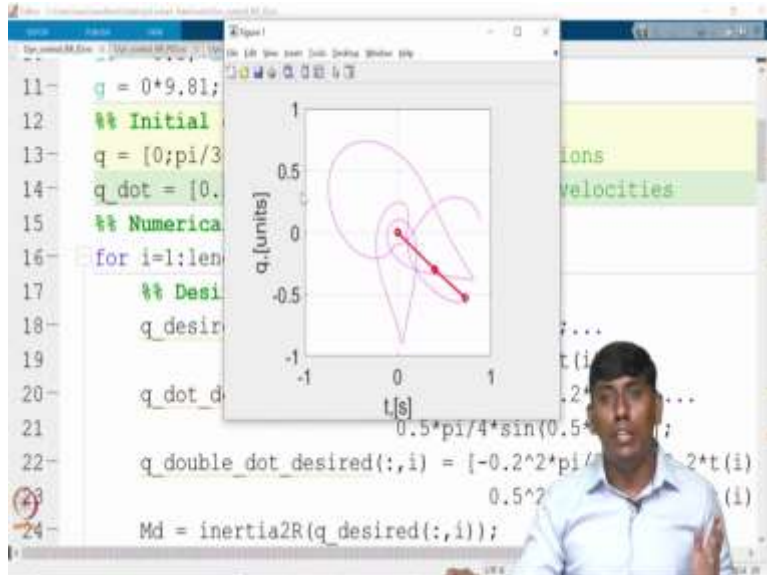




And similar way we assume that the initial conditions are not matching just I assume that the initial position supposed to be $\pi/3$ minus $\pi/4$ as the θ_2 but now I assume both are 0. So, you can see like that may not go as such what he expected. So, you can see so, this is like going beyond because the actual scenario is completely different than the inverse dynamics. In fact, if I extend this may end up with even unstable. So, that is why inverse dynamics as open you can say simple you call feed forward control people are not really using. So, we started $\pi/3$ and then we are trying to see whether the system actually like is going it is not going.

(Refer slide Time: 12:44)






```

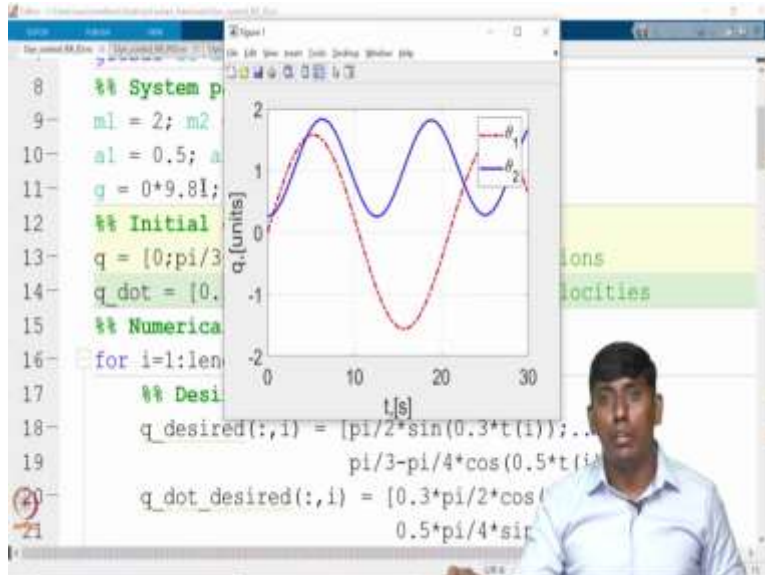
14- q_dot = [0.2*pi/2;0]; % initial joint velocities
15- %% Numerical integration starts here
16- for i=1:length(t)
17-     %% Desired values
18-     q_desired(:,i) = [pi/2*sin(0.3*t(i));...
19-                     pi/3-pi/4*cos(0.5*t(i))];
20-     q_dot_desired(:,i) = [0.3*pi/2*cos(0.3*t(i));...
21-                          0.5*pi/4*sin(0.5*t(i))];
22-     q_double_dot_desired(:,i) = [-0.3^2*pi/2*cos(0.3*t(i))
23-                                   0.5^2*pi/4*cos(0.5*t(i))];
24-     Md = inertia2R(q_desired(:,i));
25-     oe_vd = other_effects2R(q_desired(:,i), q_dot_desired(:,i));
26-     g_vd = gravity_effects2R(q_desired(:,i));
27-     M = inertia2R(q(:,i));

```

```

8- %% System parameters
9- m1 = 2; m2 = 1;
10- a1 = 0.5; a2 = 0.3;
11- g = 0*9.81;
12- %% Initial conditions
13- q = [0;pi/3];
14- q_dot = [0.2*pi/2;0];
15- %% Numerical integration starts here
16- for i=1:length(t)
17-     %% Desired values
18-     q_desired(:,i) = [pi/2*sin(0.3*t(i));...
19-                     pi/3-pi/4*cos(0.5*t(i))];
20-     q_dot_desired(:,i) = [0.3*pi/2*cos(0.3*t(i));...
21-                          0.5*pi/4*sin(0.5*t(i))];

```



So, similarly, if we assume that the initial velocity is supposed to be same, but we assume that it is not given that way. So, the theta one dot initially there is 0.2 times pi by 2, but we are assuming that it is not the case. So, then also you can see like it is not following the exact profile what we have indented.

So, these are all the like, you call subset, or you call limitations of the open loop control because there is no feedback. So, I hope now you are understand, so even you can increase this probably, I am just increasing this in a faster sense. So, in the sense I am making it this is increasing into 0.3 times. So, I am just making it everything is 0.3, so, then this initial velocity also like 0.3. So, now if I run, you can see like that would be faster. So, if you look at it the theta 1 it is not exceeding plus or minus pi by 2. So, that is what we expected and that is what happening.

$$\tau = \mathbf{M}(\mathbf{q})[\ddot{\mathbf{q}} + \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}})] + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$



So, this is what the forward dynamics, I hope you are like clear to this forward dynamics, let us come to the, what you call the computer torque control. So, we will take the same scenario what we have used in the computer torque control. So, for that I am considering few other things where the control parameter K_p and K_d I have assumed as 4 for each.

So, although we are not using the frictional but I just to consider the frictional value. So, in the simulation, I will show you if I consider friction what will happen. So, now, so, this is the controller which we derived in the regular lecture. So, we are trying to incorporate this. So, for that you need to have the; you can $\tilde{\mathbf{q}}$ and $\dot{\tilde{\mathbf{q}}}$ in addition to that, you need to have the actual vector need to be compensated.

(Refer Slide Time: 14:43)

Dynamic system: $\tau = M(q)\ddot{q} + K_p(q_d - q) + K_d(\dot{q}_d - \dot{q}) + V(q, \dot{q}) + g(q)$

Computed torque or model-based control

```
%% Errors
q_tilda(:,i) = q_desired(:,i) - q(:,i);
q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
%% Computed-torque control
%% input vector
tau(:,i) = M*(q_double_dot_desired(:,i) ...
            + Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))...
            + oe_v + g_v;
```

Supritham Murthy, IIT Palakkad
Mechanics and Control of Flexible Manipulators

```
10- a1 = 0.5; a2 = 0.4; % link lengths
11- g = 9.81; % gravity
12- h1 = 0.5; b2 = 0.5; c1 = 0*0.5; c2 = 0*0.5;
13- %% Initial conditions
14- q = [0;pi/3-pi/4]; % initial joint positions
15- q_dot = [0;0]; % initial joint velocities
16- %% Control parameters
17- Kp = 4; Kd = 4;
18- %% Numerical integration starts here
19- for i=1:length(t)
20-     %% Desired values
21-     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
22-                     pi/3-pi/4*cos(0.5*t(i))];
23-     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i));...
24-                          -0.5*pi/4*sin(0.5*t(i))];
```

```

28- oe_v = other_effects2R(q(:,i),q_dot(:,i));
29- g_v = gravity_effects2R(q(:,i));
30- Fr = 0*frictional_effects2R(q_dot(:,i));
31- %% Errors
32- q_tilda(:,i) = q_desired(:,i) - q(:,i);
33- q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
34- %% Computed-torque control
35- %input vector
36- tau(:,i) = M*(q_double_dot_desired(:,i)
37-             + Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))..
38-             + oe_v + g_v;
39- % acceleration vector
40- q_double_dot(:,i) = inv(M)*(tau(:,i)

```

```

34- %% Computed-torque control
35- %input vector
36- tau(:,i) = M*(q_double_dot_desired(:,i) ...
37-             + Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))..
38-             + oe_v + g_v;
39- % acceleration vector
40- q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+Fr+g_v));
41-
42- % velocity propogation
43- q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
44-
45- % position update
46- q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 0.5*q_double_dot(:,i)*dt^2;
47- [x(i),y(i)] = FK2R(q(1,i),q(2,i));

```

So, in that sense you can see that, so, the Kp into q tilde and Kd into q tilde dot is coming and then you can q double dot desired, and this is multiply with mass as and then the other vector plus gravity vector is added. So, we are calculating q tilde as q desired minus q and q tilde dot or q dot tilde is q dot desired minus q dot so. Now, we will go to the you can say MATLAB code. So, this is computed torque control.

So, we can see like we assume now the gravity is included and we are taken there are two control gains are coming Kp and Kd. So, that also we have used and intentionally I make it that initial joint velocity is not same as the desired. So, then we have seen that we have calculated the q tilde

and \tilde{q} and then the control law we have used this way and then based on that we are writing it.

(Refer Slide Time: 15:45)

```
64 figure
65 %% Plotting functions
66 plot(t,q(1,1:i),'r-.',t,q(2,1:i),'b-','linewidth',2)
67 legend('\theta_1','\theta_2')
68 grid on
69 set(gca,'fontsize',20)
70 xlabel('t,[s]')
71 ylabel('q,[units]')
72
73 figure
74 plot(t,q_tilda(1,1:i),'r-.',t,q_tilda(2,1:i),'b-','linewidth',2)
75 legend('\theta_{1_e}','\theta_{2_e}')
76 grid on
77 set(gca,'fontsize',20)
```

```
70 xlabel('t,[s]')
71 ylabel('q,[units]')
72
73 figure
74 plot(t,q_tilda(1,1:i),'r-.',t,q_tilda(2,1:i),'b-','linewidth',2)
75 legend('\theta_{1_e}','\theta_{2_e}')
76 grid on
77 set(gca,'fontsize',20)
78 xlabel('t,[s]')
79 ylabel('\tilde{q},[units]','interpreter','tex')
80
81 function M = inertia2R(q)
82 global a1 a2 m1 m2
83 th1 = q(1); th2 = q(2);
```

```

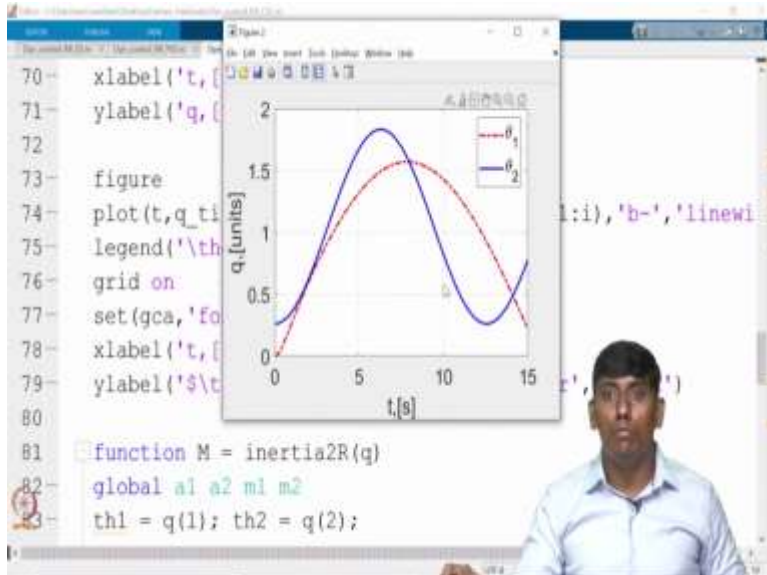
70- xlabel('t, [s]')
71- ylabel('q, [units]')
72-
73- figure
74- plot(t, q, 'b-', 'linewidth=2')
75- legend('\theta_1, \theta_2')
76- grid on
77- set(gca, 'fontSize', 14)
78- xlabel('t, [s]')
79- ylabel('q, [units]')
80-
81- function M = inertia2R(q)
82- global a1 a2 m1 m2
83- th1 = q(1); th2 = q(2);

```

```

70- xlabel('t, [s]')
71- ylabel('q, [units]')
72-
73- figure
74- plot(t, q, 'b-', 'linewidth=2')
75- legend('\theta_1, \theta_2')
76- grid on
77- set(gca, 'fontSize', 14)
78- xlabel('t, [s]')
79- ylabel('q, [units]')
80-
81- function M = inertia2R(q)
82- global a1 a2 m1 m2
83- th1 = q(1); th2 = q(2);

```



```

1 %% Dynamic simulation of a RR planar robot
2 clear all; close all; clc;
3 %% Simulation parameters
4 dt = 0.01; % stepsize
5 ts = 15; % total simulation time
6 t = 0:dt:ts; % time span
7 global a1 a2 m1 m2 g b1 b2 c1 c2
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 b1 = 0.5; b2 = 0.5; c1 = 0*0.5; c2 = 0*0.5;
13 %% Initial conditions
14 q = [0; pi/3 - pi/4]; % initial joint positions

```

So, now here we have just added one more, so, earlier we adjusted you can say plotted only time versus q , but here we are plotting time versus \dot{q} in addition that is all. So, now, if we run this, so, we can see that the \dot{q} will go somewhere so, we can see the initial velocity we as 00 however, the $\dot{\theta}_1$ desired is something.

So, in that sense the 1 the sense \dot{q}_1 would be going up short and then coming back that we can see it. So, this is the actual and it is following it. So, now you can see this is what I am quoting. So, the $\dot{\theta}_1$ is like supposed to be nonzero which, is 0.2 times a π by 2 but we have not considered.

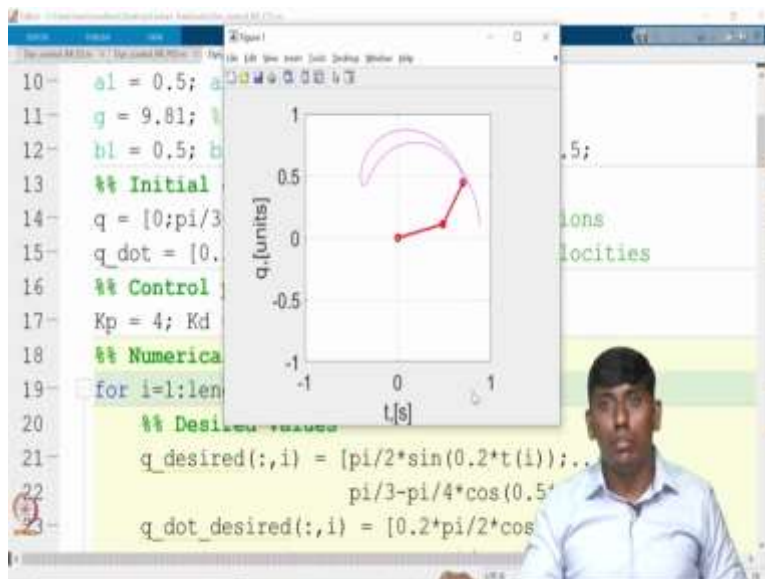
So, because of that in order to compensate that velocity profile so, the theta 1 error is like going and getting it. However, the theta 2 error as you can see almost 0. In fact it is superimposed here it is like 0. So, now this is the profile which we are asking to follow, and it is doing it. So, here we have given properly 45 degrees sorry you can say not 45 degree you can say 12, 15 second.

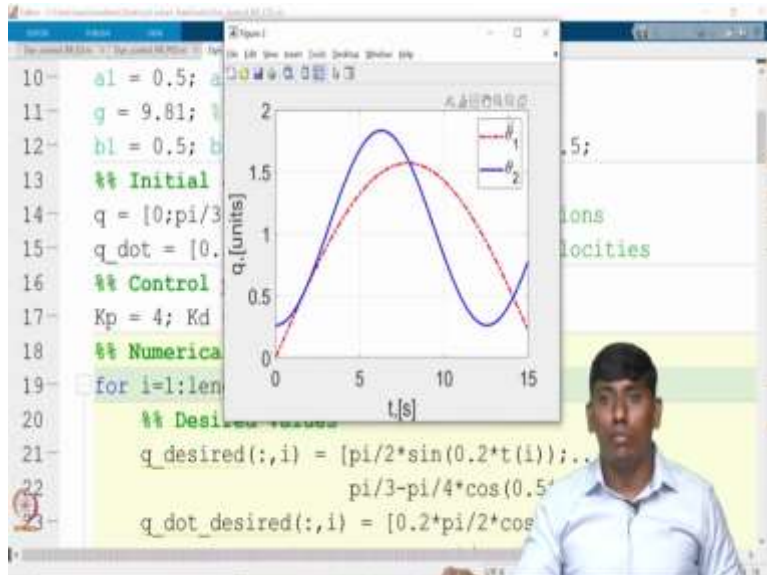
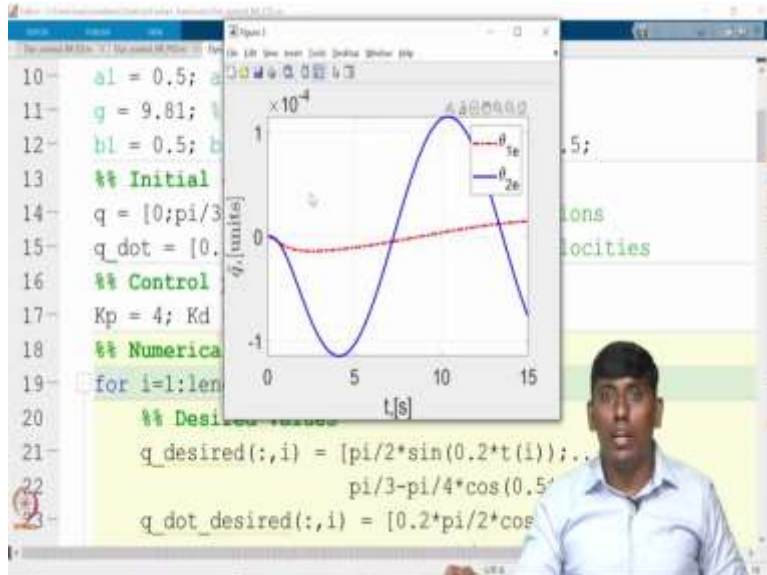
(Refer Slide Time: 16:59)

```

13 %% Initial conditions
14 q = [0;pi/3-pi/4]; % initial joint positions
15 q_dot = [0;0]; % initial joint velocities
16 %% Control parameters
17 Kp = 4; Kd = 4;
18 %% Numerical integration starts here
19 for i=1:length(t)
20     %% Desired values
21     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
22                     pi/3-pi/4*cos(0.5*t(i))];
23     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i)) ...
24                          0.5*pi/4*sin(0.5*t(i))];
25     q_double_dot_desired(:,i) = [-0.2^2*cos(0.2*t(i))
26                                  0.5^2*cos(0.5*t(i))];

```





So, 15 second in the sense it is making it so, this is 30 degree, or you can say it is up to 30 times and then this is going somewhere. So, if I you this so 0.2 times of you can say pi by 2 so now you can see that error would be converged. So, I will plot one more thing just for our benefit. So, you can see it is like so in the order of milli even further milli radians. So, this is what the error you can see error almost 0 both the cases. So, now you can see like this is following it.

(Refer Slide Time: 17:45)

```
47- [x(i),y(i)] = FK2R(q(1,i),q(2,i));
48- [xd(i),yd(i)] = FK2R(q_desired(1,i),q_desired(2,i));
49- end
50- % numerical integration ends here
51- %% Animation
52- for i=1:10:length(t)
53-     x1 = a1*cos(q(1,i));
54-     y1 = a1*sin(q(1,i));
55-     x2 = x1+a2*cos(q(1,i)+q(2,i));
56-     y2 = y1+a2*sin(q(1,i)+q(2,i));
57-     plot([0,x1,x2],[0,y1,y2],'r-o','linewidth',2)
58-     grid on, set(gca,'fontsize',20)
59-     hold on
60-     plot(x(1:i),y(1:i),'m-')
```

```
56-     y2 = y1+a2*sin(q(1,i)+q(2,i));
57-     plot(xd,yd,'k--')
58-     hold on
59-     plot([0,x1,x2],[0,y1,y2],'r-o','linewidth',2)
60-     grid on, set(gca,'fontsize',20)
61-     hold on
62-     plot(x(1:i),y(1:i),'m-')
63-     axis([- (a1+a2)-0.1, (a1+a2)+0.1, - (a1+a2)-0.1, (a1+a2)+0.1])
64-     axis square, xlabel('t, [s]'), ylabel('q, [rad]')
65-     hold off, pause(0.001)
66- end
67- figure
68- %% Plotting functions
69- plot(t,q(1,1:i),'r-.',t,q(2,1:i),'b-')
```

```

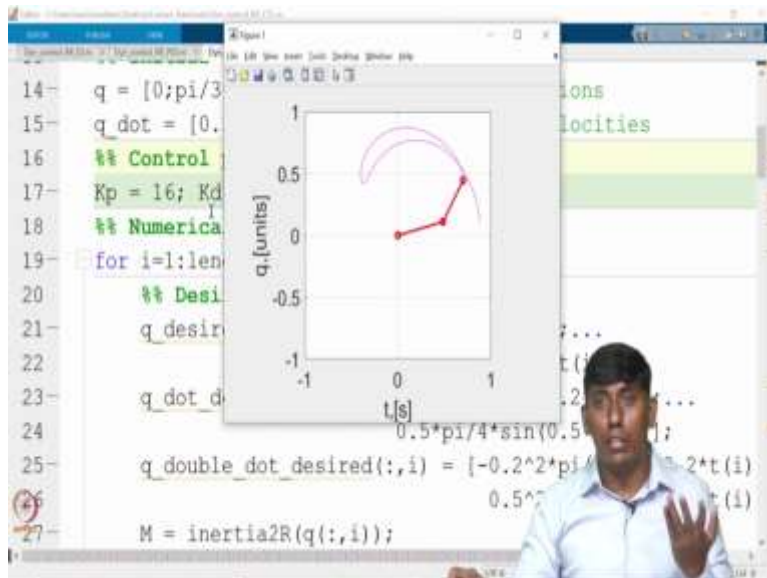
50 %% Animation
51 for i=1:10:
52     x1 = a1
53     y1 = a1
54     x2 = x1
55     y2 = y1
56     plot([0
57         grid on
58         hold on
59         plot(x(
60         axis([- (a1+a2)-0.1, (a1+a2)+0.1, - (a1+a2)
61         axis square, xlabel('t, [s]'), ylabel('q, [units]')
62         hold off, pause(0.001)
63 end

```

```

14 q = [0; pi/3-pi/4]; % initial joint positions
15 q_dot = [0.2*pi/2; 0]; % initial joint velocities
16 %% Control parameters
17 Kp = 16; Kd = 8;
18 %% Numerical integration starts here
19 for i=1:length(t)
20     %% Desired values
21     q_desired(:,i) = [pi/2*sin(0.2*t(i));...
22                     pi/3-pi/4*cos(0.5*t(i))];
23     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i))...
24                          0.5*pi/4*sin(0.5*t(i))];
25     q_double_dot_desired(:,i) = [-0.2^2*pi/2*cos(0.2*t(i))...
26                                   0.5^2*pi/4*cos(0.5*t(i))];
27     M = inertia2R(q(:,i));

```



```

26     0.5^2*pi/4*cos(0.5*t(i))];
27     M = inertia2R(q(:,i));
28     oe_v = other_effects2R(q(:,i),q_dot(:,i));
29     g_v = gravity_effects2R(q(:,i));
30     Fr = 0*frictional_effects2R(q_dot(:,i));
31     %% Errors
32     q_tilda(:,i) = q_desired(:,i) - q(:,i);
33     q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
34     %% Computed-torque control
35     %input vector
36     tau(:,i) = M*(q_double_dot_desired(:,i) +
37     Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i)) +
38     oe_v + g_v;
39     % acceleration vector

```

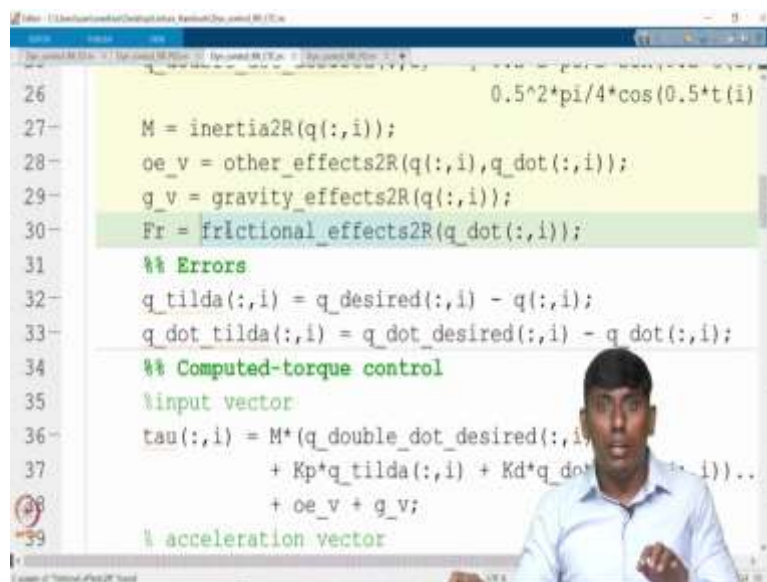
So, now I will just want to plot it I hope so the forward kinematic model is available. So, I will take it the forward kinematics here so I will just take it so I will just take x desired. So, I will just say so x desired I hope x desired I have not used xd anywhere. So, yd so this is q underscored desired.

And this is underscored desired, so I am just plotting it here, so I am just making it so here I am a plotting x desired comma y desired that to like I am showing that there is the desired plot which is I am showing it as k double dot in this dashed line so, now if I run this, so I hope this matrix because, so here.

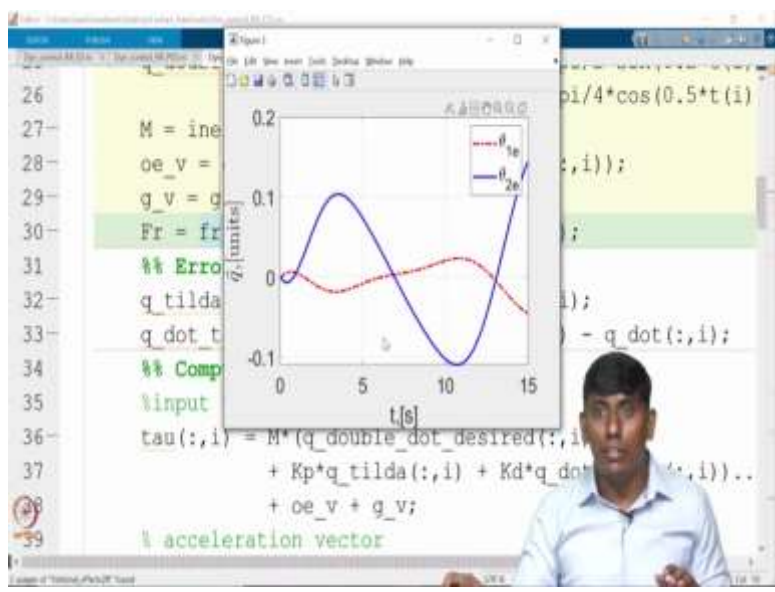
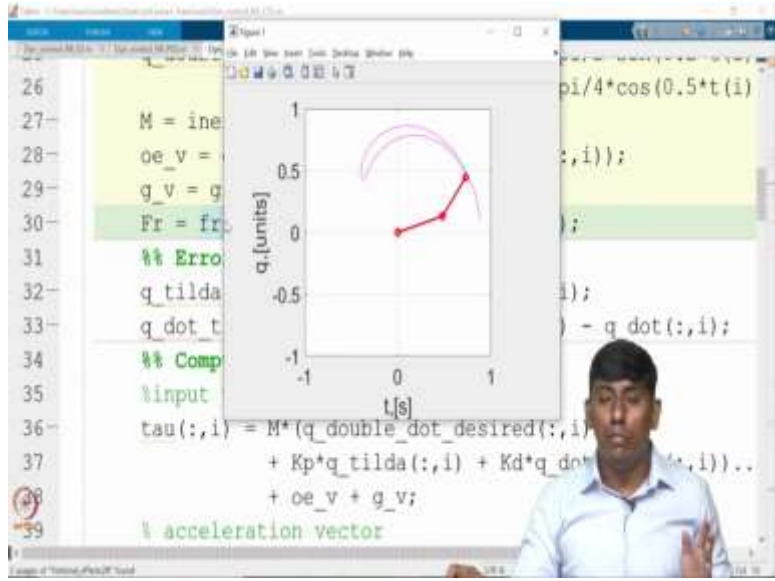
So, we can plot this x desired, and y desired I hope this x desired and y desired is derived here. So, then we can. So, q underscore desired. So, q underscore desired 1 comma i and this 2 comma i so this makes this and we are trying to plot this. So, now the x, desired and y desired is having same thing. So, we can actually bracket this so, there is a so, we can actually like see it here x desired not come because it is like inside that we will like to do it the x desired is like I just want to show the plot. So, but that is like not as explicit here because the forward kinematic model what we have used is slightly different.

So, now we will like go back to the one which we have like tried. So, now I will like increase the Kp and Kd you can see that they error value further you can say decrease or increase based on this. So, now I am saying that this is 8 and this is probably you can say 16 just I said. So, this is 2 omega this is omega squared. So, that they have like make it you can see like it is going to be much much further reduce the error. So, now, you can like feel it. So, now if I like introduce the friction, I hope the friction values are here.

(Refer Slide Time: 21:26)



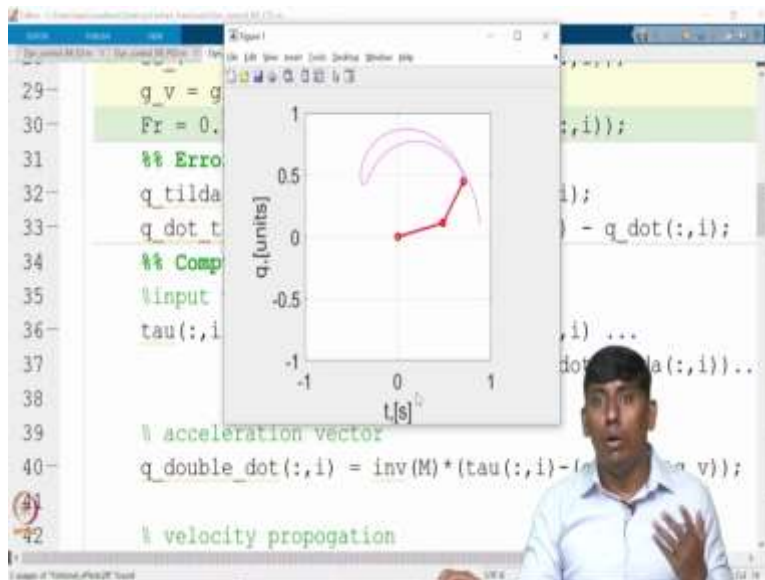
```
26 0.5*2*pi/4*cos(0.5*t(i))
27 M = inertia2R(q(:,i));
28 oe_v = other_effects2R(q(:,i),q_dot(:,i));
29 g_v = gravity_effects2R(q(:,i));
30 Fr = frictional_effects2R(q_dot(:,i));
31 %% Errors
32 q_tilda(:,i) = q_desired(:,i) - q(:,i);
33 q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
34 %% Computed-torque control
35 %input vector
36 tau(:,i) = M*(q_double_dot_desired(:,i)
37           + Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))..
38           + oe_v + g_v;
39 % acceleration vector
```

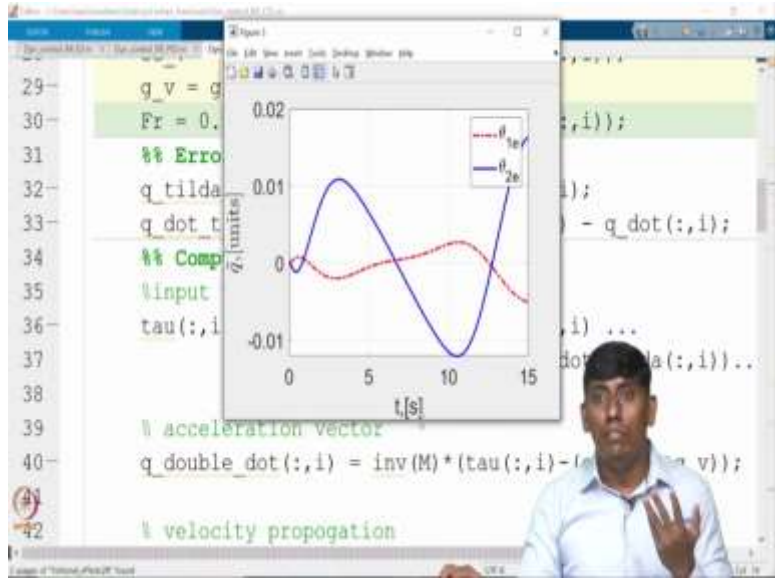


So, now I introduce the viscous friction. So, you can like see so, this control still working because friction we are not compensated based on that. So, you can see like it is going the error, because that is what the, you can say difficulty, or you call constraint on the model based control. So, now, the model is inaccurate. So, because of this inaccurateness. So, you can see that the frictional effect we included that like not able to compensate here.

(Refer Slide Time: 22:04)

```
29 q_v = gravity_effects2R(q(:,i));
30 Fr = 0.1*frictional_effects2R(q_dot(:,i));
31 %% Errors
32 q_tilda(:,i) = q_desired(:,i) - q(:,i);
33 q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
34 %% Computed-torque control
35 %input vector
36 tau(:,i) = M*(q_double_dot_desired(:,i) ...
37             + Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))..
38             + oe_v + g_v;
39 % acceleration vector
40 q_double_dot(:,i) = inv(M)*(tau(:,i)-(M*(q_double_dot_desired(:,i) ...
41                               + Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))..
42                               + oe_v + g_v));
43 % velocity propagation
```





So, however if you add this here then that will come that is any feedback linearization. So, that is the aspect here. So, even I add the point one times of the friction total effort assume that the friction is like very low. So, now also you can see like there would be a small error which is like propagating because it is not able to continuously follow it. So, now we can like go back to the, what you call the slide.

(Refer Slide Time: 22:27)

System Dynamics
Control Systems

Computed torque or model-based control

Motion-based control

$$\tau = \mathbf{M}(\mathbf{q})[\ddot{\mathbf{q}} + \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}})] + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

```
% Errors
q_tilda(:,i) = q_desired(:,i) - q(:,i);
q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
%% Computed-torque control
%%input vector
tau(:,i) = M*(q_double_dot_desired(:,i) ...
+ Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))...
+ oe_v + g_v;
```

5

SUBRAMANIAM MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTS MANIPULATORS



System Dynamics
Control Systems

Computed torque or model-based control


Motion-based control

$$\tau = \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

```
% Errors
q_tilda(:,i) = q_desired(:,i) - q(:,i);
q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
%% PD control
%%input vector
tau(:,i) = (Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))...
+ g_v;
```

5

SUBRAMANIAM MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTS MANIPULATORS



System Dynamics
Controlled system or model based control
Motion based control


$$\tau = K_p (q_d - q) + K_i \int (q_d - q) dt + K_d (\dot{q}_d - \dot{q}) + g(q)$$

```

%% Errors
q_tilda(:,i) = q_desired(:,i) - q(:,i);
q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
ei = ei + q_tilda(:,i)*dt;
%% PID control
%%input vector
tau(:,i) = (Kp*q_tilda(:,i) + Ki * pi ...
            + Kd*q_dot_tilda(:,i))* g_v;

```

Supriyanto Mulya, IIT Palangka Raya
Mechatronics and Control of Dynamic Manipulators



So, we will go to the motion based control so, where we have taken PD control with the gravity compensation. So, the code only gets changed here already we defined Kp and Kd. So, only thing is like the feedback linearization where the other vector removed and q double dot desired is removed and M matrix also removed. So, now the Kp and Kd we need to tune accordingly. So, this is what we call the PD control. Similarly, if we want actual like PID Control then the ki we have to add and error tilde also like a q tilde integral error also we need to add. So, here we assume that is ei. So, ei we have like a propagated based on simple integration Euler integration then we have like got it.

(Refer Slide Time: 23:11)


File - Edit - View - Help

```

31  %% Errors
32  q_tilda(:,i) = q_desired(:,i) - q(:,i);
33  q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
34  %% PD control
35  %%input vector
36  tau(:,i) = (Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))...
37            + g_v;
38  %% acceleration vector
39  q_double_dot(:,i) = inv(M)*(tau(:,i)-(c(q)+Fr+g_v));
40
41  %% velocity propagation
42  q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
43
44  %% position update

```

Supriyanto Mulya, IIT Palangka Raya
Mechatronics and Control of Dynamic Manipulators



```

34 %% PD c
35 \input
36 tau(:,i)
37
38 \ accel
39 q_doubl
40
41 \ veloc
42 q_dot(:
43
44 \ posit
45 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*
46 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
47 end

```

ot_tilda(:,i))...
-(oe_v+Fr+g_v));
dot(:,i)*dt;

```

34 %% PD c
35 \input
36 tau(:,i)
37
38 \ accel
39 q_doubl
40
41 \ veloc
42 q_dot(:
43
44 \ posit
45 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 1/2*
46 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
47 end

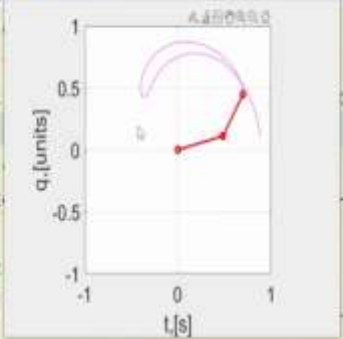
```

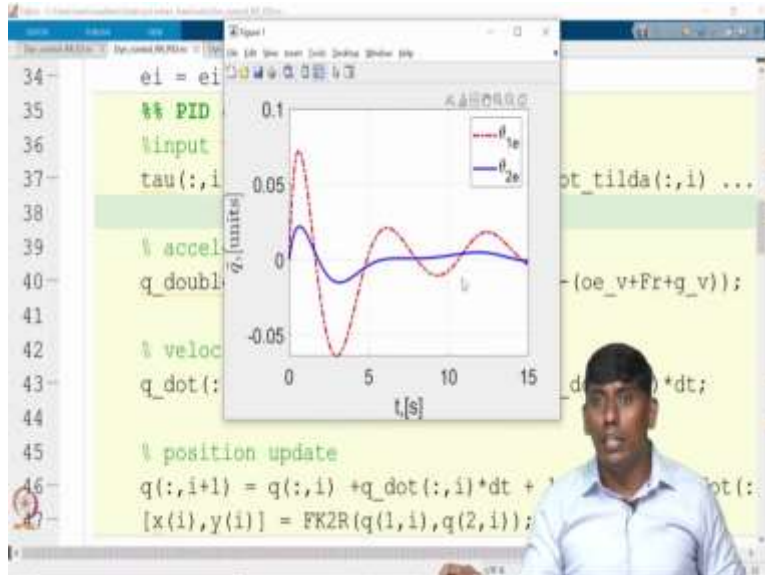
ot_tilda(:,i))...
-(oe_v+Fr+g_v));
dot(:,i)*dt;

```
34 ei = ei + q_tilda(:,i)*dt;
35 %% PID control
36 \input vector
37 tau(:,i) = (Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i) ...
38 + Ki*ei)+ g_v;
39 \ acceleration vector
40 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+Fr+g_v));
41
42 \ velocity propogation
43 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
44
45 \ position update
46 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 0.5*q_double_dot(:,i)*dt^2;
47 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
```



```
34 ei = ei + q_tilda(:,i)*dt;
35 %% PID control
36 \input vector
37 tau(:,i) = (Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i) ...
38 + Ki*ei)+ g_v;
39 \ acceleration vector
40 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+Fr+g_v));
41
42 \ velocity propogation
43 q_dot(:,i+1) = q_dot(:,i) + q_double_dot(:,i)*dt;
44
45 \ position update
46 q(:,i+1) = q(:,i) + q_dot(:,i)*dt + 0.5*q_double_dot(:,i)*dt^2;
47 [x(i),y(i)] = FK2R(q(1,i),q(2,i));
```



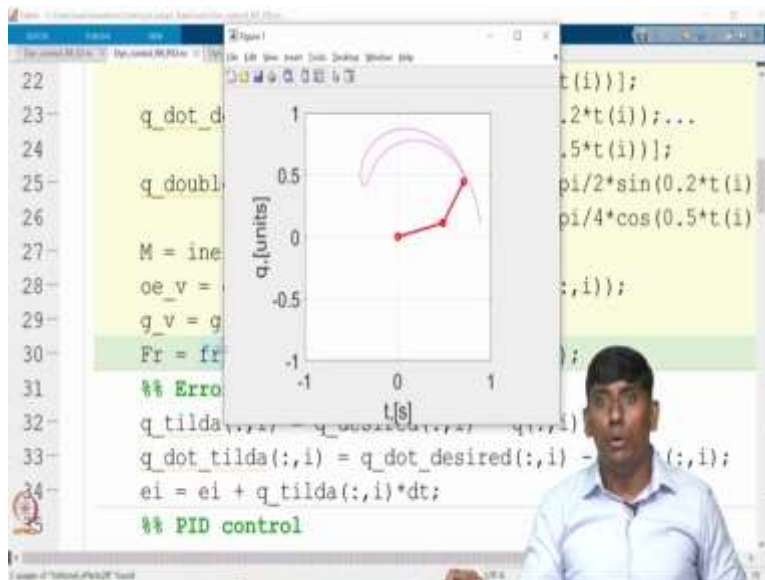


So, now I will show it this here in the MATLAB code then you can like see. So, this is a PD control. So, now we assume that the friction is 0 and gravity is included so, the gravity compensation we have taken and K_p and K_d we have taken as four each we did not change. Because your mass is like only two so, it would give reasonable result if your mass is very high then we have to like a tune the K_p and K_d right now we need not to like do it. So, now remaining everything is same. So, if I run this PD control you can see like it would be having a small steady state error so that you can like see it.

So, you can see like earlier the computer torque control without friction. It was almost like 0 where it is in the order of 10^{-4} in the sense milli radians. But here you can see it is like centi radians at least so that is what one you can say issue so now if I add the PID control, so where the item we added as so K_i and e_i initially we assume 0 0 so then we have like calculate the integral error and then we have like added this term. So, now if I like run it, so you can see like this would be following the same profile what we have given and you can see that the error is like very close to 0 both our like, converse to 0. Now, if I increase K_i or increase K_p accordingly this will actually get change.

(Refer Slide Time: 24:42)

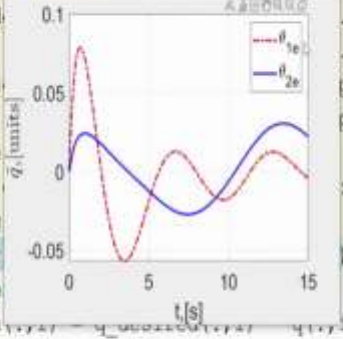
```
22         pi/3-pi/4*cos(0.5*t(i)));
23     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i));...
24         0.5*pi/4*sin(0.5*t(i))];
25     q_double_dot_desired(:,i) = [-0.2^2*pi/2*sin(0.2*t(i)
26         0.5^2*pi/4*cos(0.5*t(i)
27
28     M = inertia2R(q(:,i));
29     oe_v = other_effects2R(q(:,i),q_dot(:,i));
30     q_v = gravity_effects2R(q(:,i));
31     Fr = frictional_effects2R(q_dot(:,i));
32     %% Errors
33     q_tilda(:,i) = q_desired(:,i) - q(:,i);
34     q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
35     ei = ei + q_tilda(:,i)*dt;
36     %% PID control
```



```

22
23   q_dot_d
24   q_double
25   M = ine
26   oe_v =
27   g_v = g
28   Fr = fr
29   %% Error
30   q_tilda
31   q_dot_tilda(:,i) = q_dot_desired(:,i) -
32   ei = ei + q_tilda(:,i)*dt;
33   %% PID control

```



```

t(i));
.2*t(i));...
.5*t(i));
pi/2*sin(0.2*t(i)
pi/4*cos(0.5*t(i)
(:,i));
);
);

```

```

28   oe_v = other_effects2R(q(:,i),q_dot(:,i));
29   g_v = gravity_effects2R(q(:,i));
30   Fr = frictional_effects2R(q_dot(:,i));
31   %% Errors
32   q_tilda(:,i) = q_desired(:,i) - q(:,i);
33   q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
34   %% PD control
35   \input vector
36   tau(:,i) = (Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))...
37             + g_v;
38   \ acceleration vector
39   q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
40   \ velocity propogation

```



```

28 oe_v =
29 g_v = g
30 Fr = fr
31 %% Error
32 q_tilda
33 q_dot_t
34 %% PD c
35 \input
36 tau(:,i)
37
38 \ accel
39 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
40
    \ velocity propagation

```

The plot shows the displacement q in units over time t in seconds. The x-axis ranges from -1 to 1, and the y-axis ranges from -1 to 1. A red line starts at the origin (0,0) and follows a path that curves upwards and to the right, ending at approximately (0.8, 0.5). A purple arc is also visible, representing a reference trajectory.

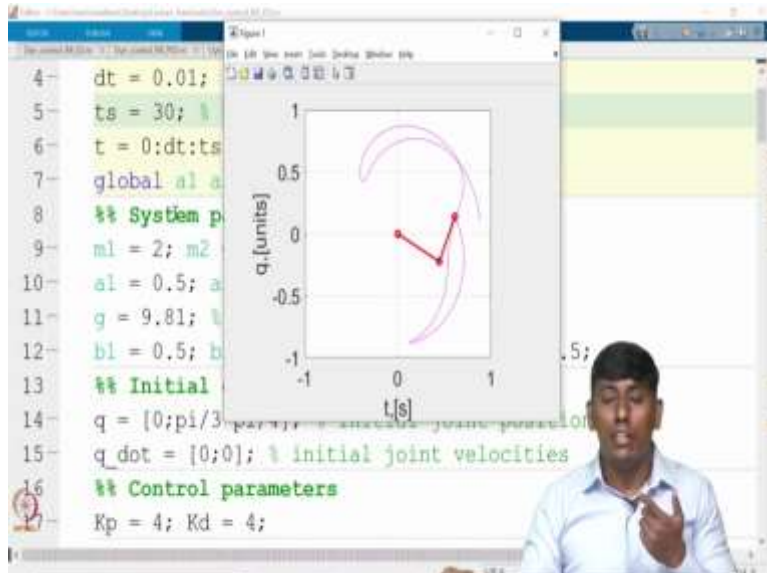
```

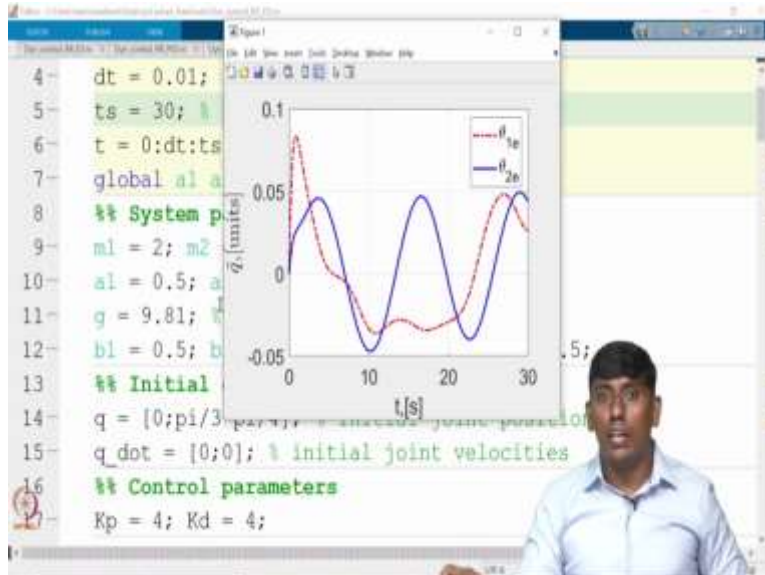
28 oe_v =
29 g_v = g
30 Fr = fr
31 %% Error
32 q_tilda
33 q_dot_t
34 %% PD c
35 \input
36 tau(:,i)
37
38 \ accel
39 q_double_dot(:,i) = inv(M)*(tau(:,i)-(oe_v+g_v));
40
    \ velocity propagation

```

The plot shows the displacement q in units over time t in seconds. The x-axis ranges from 0 to 15, and the y-axis ranges from -0.05 to 0.1. Two trajectories are shown: a red dashed line representing θ_{1e} and a blue solid line representing θ_{2e} . Both trajectories exhibit oscillatory behavior, with θ_{1e} starting at approximately 0.08 and θ_{2e} starting at 0.

```
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 b1 b2 c1 c2
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 b1 = 0.5; b2 = 0.5; c1 = 0*0.5; c2 = 0*0.5;
13 %% Initial conditions
14 q = [0;pi/3-pi/4]; % initial joint positions
15 q_dot = [0;0]; % initial joint velocities
16 %% Control parameters
17 Kp = 4; Kd = 4;
```





So, this is the benefit of ours. So, now even if I add some friction, so for example, I am adding a friction. So, you can see some different phenomena here. So, it is trying to follow, so what you can see like it is not, you can say propagating like what you have seen in the computer torque control. So, but here you can see it is still under the control.

So, that is the advantage of motion-based control. So, that is why we are like trying to what you call combined feed forward come feedback. So, that is what the whole idea. So, now you can see like even the friction we included the I term is like trying to compensate whereas the PD control if I include the friction.

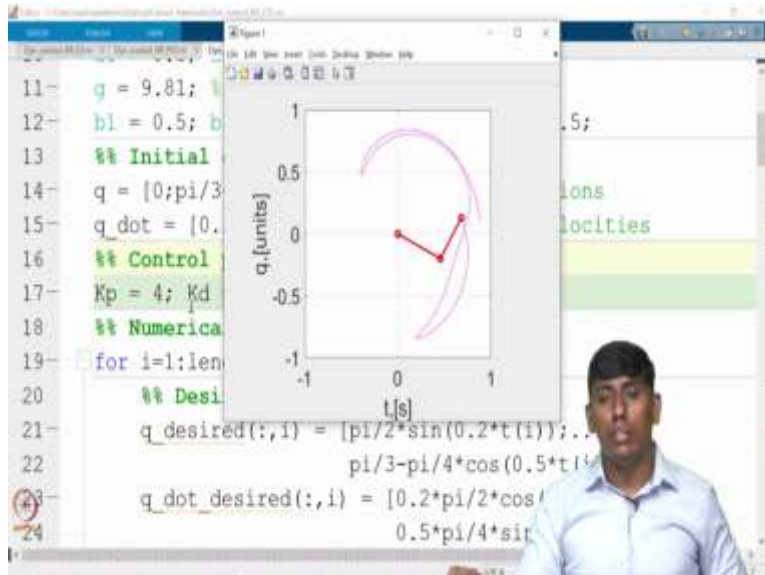
So, it may like give a slightly a similar response what you have seen in the computer torque control you can like see it is like trying to go somewhere after that it may get stable or not that you can like see by increasing the time probably I said that just a 30 second if I ran so, you can see like it is like going across and getting it you can see so it is like not able to follow because it is like having a steady state error of so and so. So, that is what we can see the steady state error is prolonged.

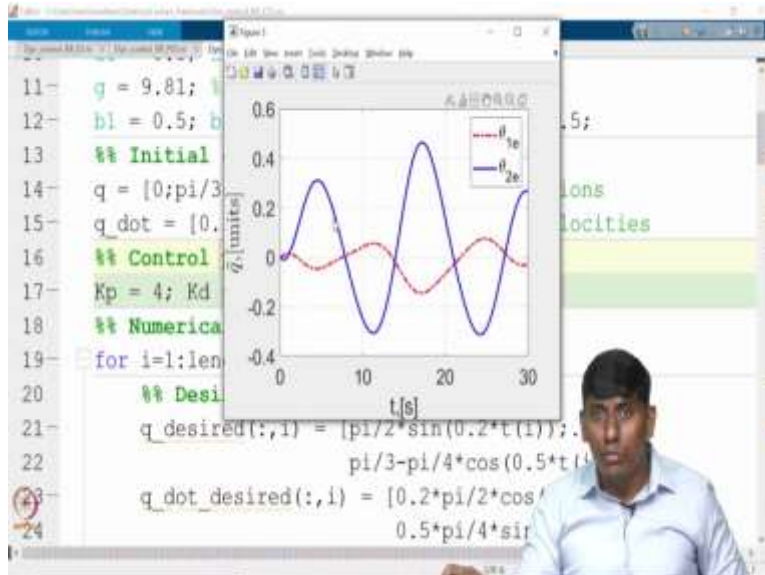
(Refer Slide Time: 26:06)

```
29- q_v = gravity_effects2R(q(:,i));
30- Fr = frictional_effects2R(q_dot(:,i));
31- %% Errors
32- q_tilda(:,i) = q_desired(:,i) - q(:,i);
33- q_dot_tilda(:,i) = q_dot_desired(:,i) - q_dot(:,i);
34- %% Computed-torque control
35- %input vector
36- tau(:,i) = M*(q_double_dot_desired(:,i) ...
37-             + Kp*q_tilda(:,i) + Kd*q_dot_tilda(:,i))..
38-             + oe_v + g_v;
39- % acceleration vector
40- q_double_dot(:,i) = inv(M)*(tau(:,i)-(Fr + q_v));
41-
42- % velocity propagation
```

```
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 b1 b2 c1 c2
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- b1 = 0.5; b2 = 0.5; c1 = 0*0.5; c2 = 0*0.5;
13- %% Initial conditions
14- q = [0;pi/3-pi/4]; % initial joint positions
15- q_dot = [0.2*pi/2;0]; % initial joint velocities
```

```
11- g = 9.81; % gravity
12- b1 = 0.5; b2 = 0.5; c1 = 0*0.5; c2 = 0*0.5;
13- %% Initial conditions
14- q = [0;pi/3-pi/4]; % initial joint positions
15- q_dot = [0.2*pi/2;0]; % initial joint velocities
16- %% Control parameters
17- Kp = 4; Kd = 4;
18- %% Numerical integration starts here
19- for i=1:length(t)
20-     %% Desired values
21-     q_desired(:,i) = [pi/2*sin(0.2*t(i));
22-                     pi/3-pi/4*cos(0.5*t(i))];
23-     q_dot_desired(:,i) = [0.2*pi/2*cos(0.2*t(i));
24-                          0.5*pi/4*sin(0.5*t(i))];
```





Whereas the computer torque control we include the friction and we have run for more time and we have like taken everything is same. So, I will just take it this is 4 comma 4, so that the consistency will come. So, because the 16 may like end up with you can end stableness. So, now you can see like it is much much closer.

So, now you can like see it. So, now the error is like when see propagated. So, you can see the PD control with the simple motion-based compensation it is like somewhere within this. So, within plus or minus 0.05 radian but it is like going across. So, in the sense the second state is not able to follow because the other effect vector is like a propagating which is we are trying to compensate.

So, now I hope so, this is like very clear to you. So, with that we are ending this particular lecture. But the next lecture we are going to see what you call the computer torque control in task space and the cascaded control loop which we call dual loop where the kinematic control at task space and the, you can see inner or dynamic control in joint space, that combination we have seen. So, this MATLAB simulation, we will see. Until then, see you bye. Take care.