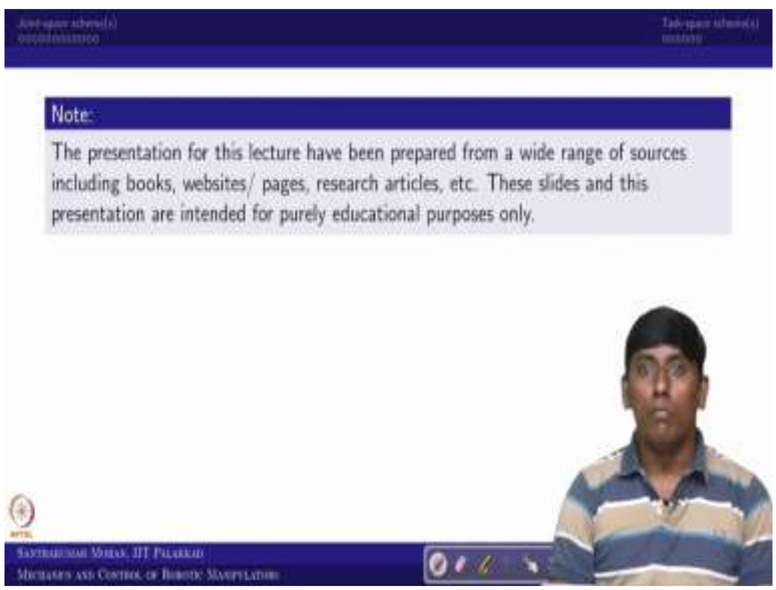
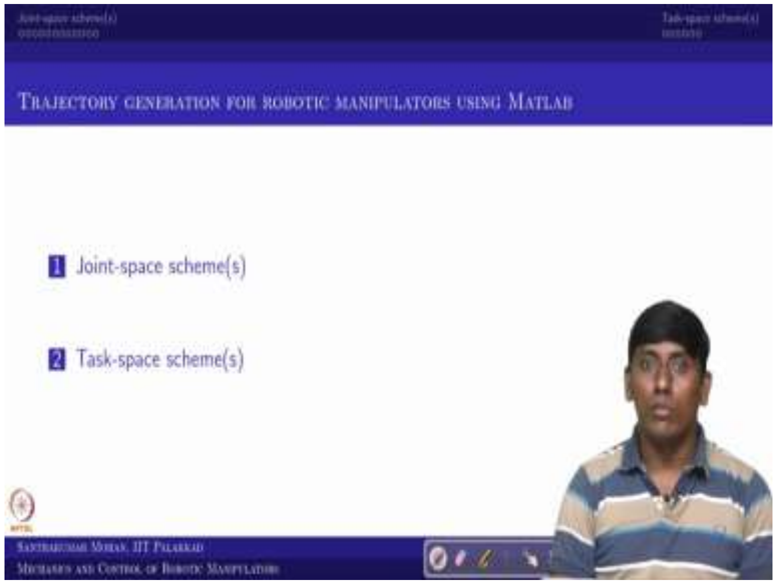
Mechanics and Control of Robotic Manipulators
Professor Santhakumar Mohan
Department of Mechanical Engineering
Indian Institute of Technology, Palakkad
Lecture No 34
Trajectory generation for serial manipulators using MATLAB

Hi, welcome back to Mechanics and Control of Robotic Manipulator. The last two classes we have seen how to generate a trajectory for a general case. So, this particular lecture we are going to see a very specific case. In the sense how to employ those schemes into a robotic manipulator trajectory generation. So, for that again we are going to take a same example which is 2R serial manipulator, and we will see how the joint space scheme and task space scheme can be visualized with the MATLAB.

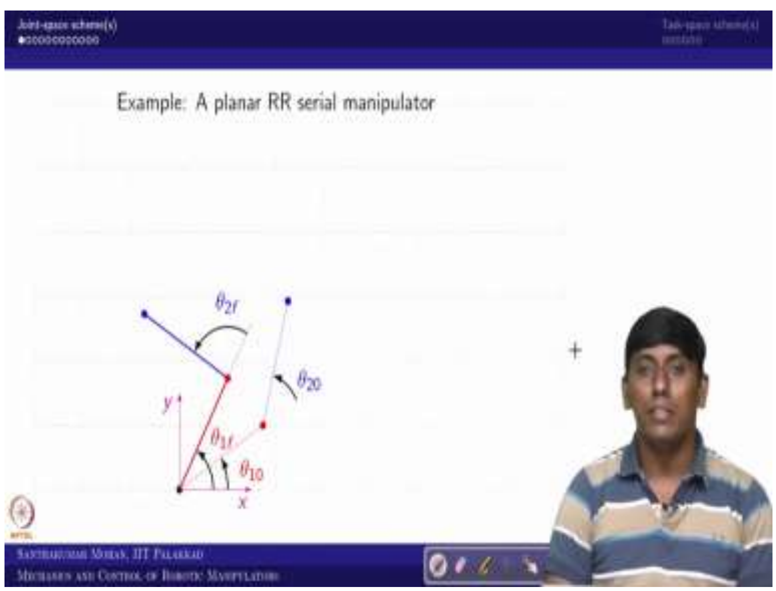
(Refer Slide Time: 0:43)





So, for that we are taking you can say two such schemes. So, you know even the joint space scheme can be make it as a synchronous and series or you call it a simultaneous and synchronous. So, the other one is task space scheme we can do the straight-line interpolation, or we can take independent trajectory generation.

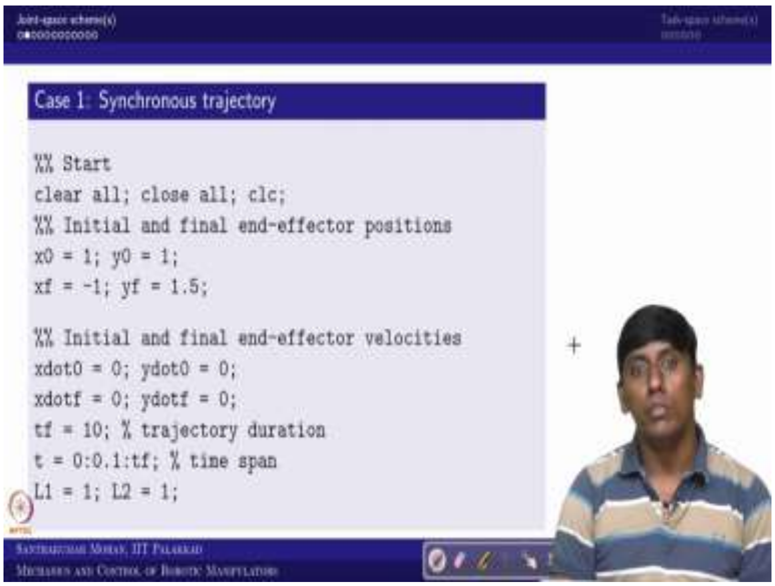
(Refer Slide Time: 1:03)



So, in that sense we are trying to see first what is the example which is given. So, now, in this case you can see that theta 1 0 and theta 1 f and theta 2 0 and theta 2 f are given. If even the x 0 and y 0 and x f and y f are given, we can always calculate these 4 variables through the help of inverse kinematics. So, in my MATLAB code I have written based on the inverse kinematics however, it is not going to restrict that way.

So, in that sense, so, we will first try to see how the joint space scheme can be generated in the sense this is the initial position and you want to go this final position. But through the help of joint space trajectory in the sense theta 1 0 supposed to go to theta 1 f and theta 2 0 supposed to go to theta 2 f that is what we are trying to do even you have already seen in the lecture. So, this can be done in synchronous or probably in series. So, we will try to see both in you can say this particular lecture.

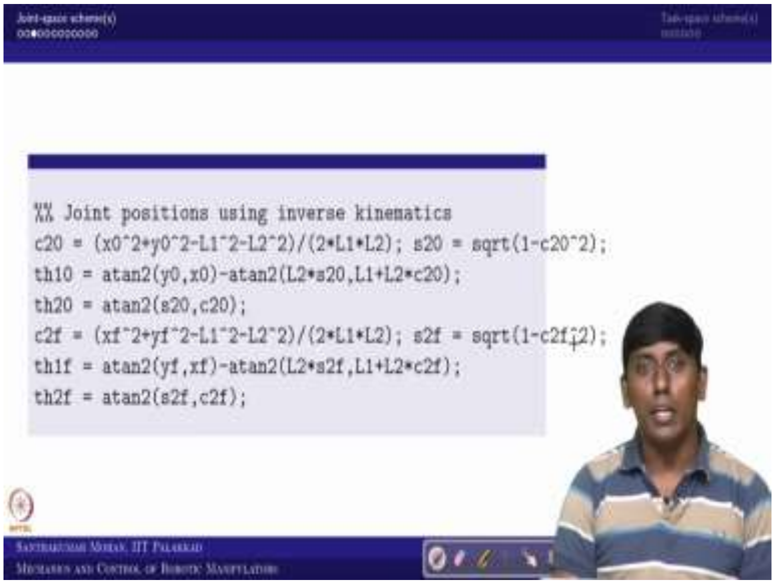
(Refer Slide Time: 2:03)



So, first one is synchronous. So, where we are taking the recall initial position and you can say final position of the end effector. So, from there what we can do? We can use the inverse kinematics. So, then we can calculate theta 1 naught and theta 1 f all those things. So, in that condition, so, even the initial and final end effector velocity also can be given. Because that is what the cubic polynomial all about.

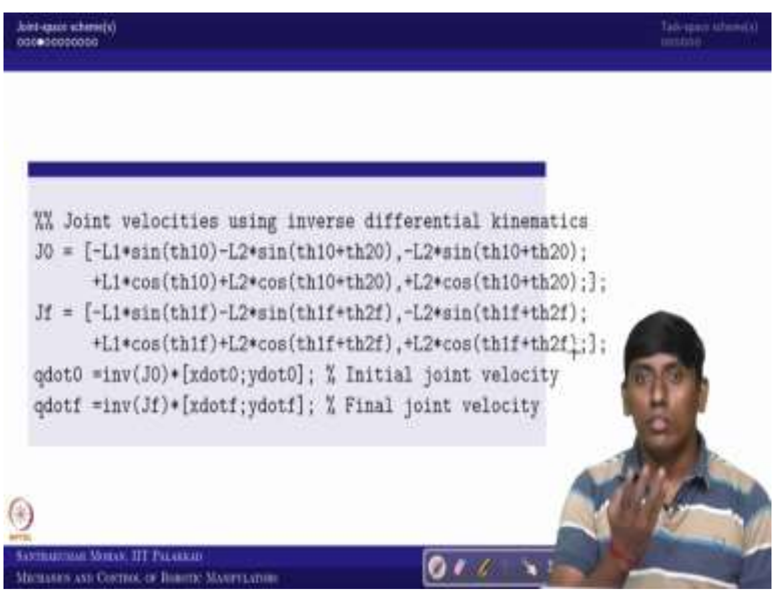
If that is the case, we can use the Jacobian in the sense the differential kinematics we can use where the inverse differential kinematics we can choose and then try to do it. So, since it is a 2R serial manipulator the system parameter or the geometry parameter or the physical parameter we need to use which is we call l 1 and l 2 in this case so, that also we have considered.

(Refer Slide Time: 2:55)



So, now, the first case we are trying to find the inverse kinematics. So, we know already inverse kinematics based on the, you can say the closed form equation. So, we have used that equation which we derived in the regular lecture. So, from there we can find theta 1 naught and theta 2 naught. So, even I would have written as a you can say sub function. However, if I write straight away sub function in the MATLAB some of you may not be familiar. So, that is why I have written explicitly here theta 1 naught theta 2 naught then theta 1 f and theta 2 f. So, now what we know? So, we know initial and final position of both joints.

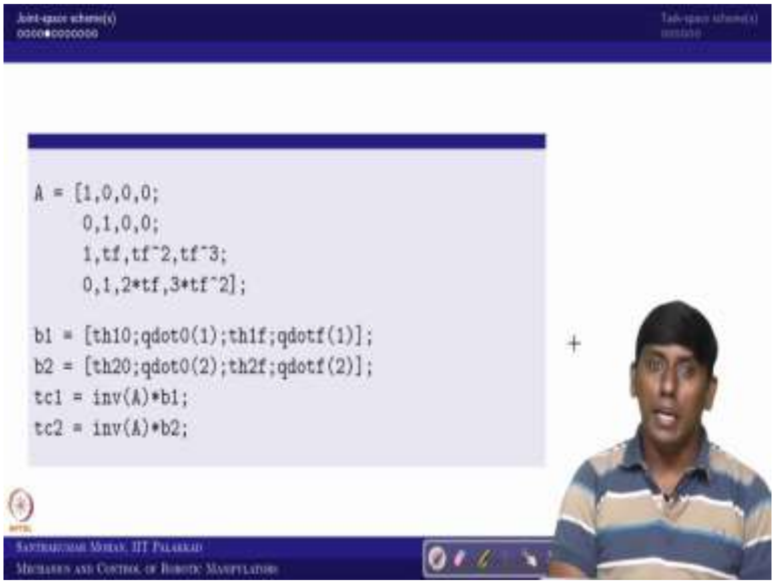
(Refer Slide Time: 3:35)



So, now, we need to find out the initial and final velocity of each joint. So, for that we are taking the Jacobian which we derived in the regular lecture. So, now, that would be substituted on the initial and final location. So, we got J 0 and J f. So, based on that you know already end effector velocities of the initial and final we can find the; you called theta 1 dot and theta 2 dot for initial and final which we have written as q dot 0 and q dot f.

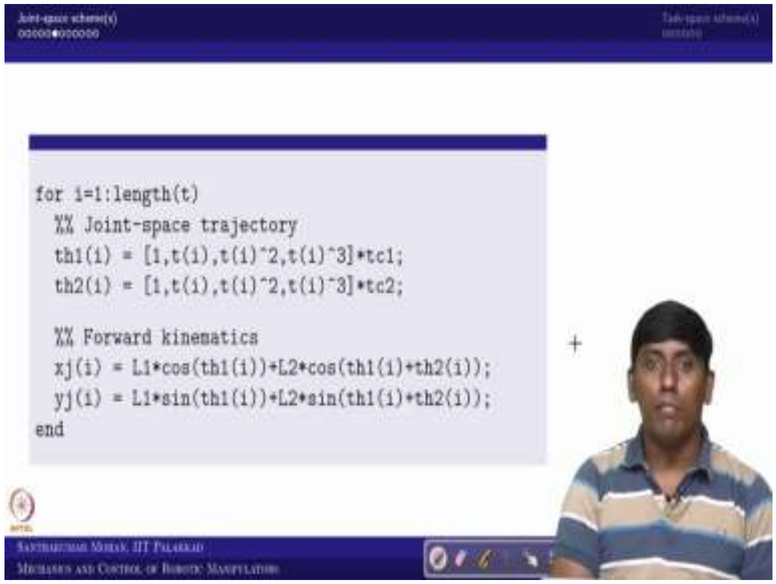
So, once these all known, then what we can do? We can go back to whatever profile generation you want to use it. So, since it is the one of the demonstrations I have taken as a cubic polynomial.

(Refer Slide Time: 4:15)



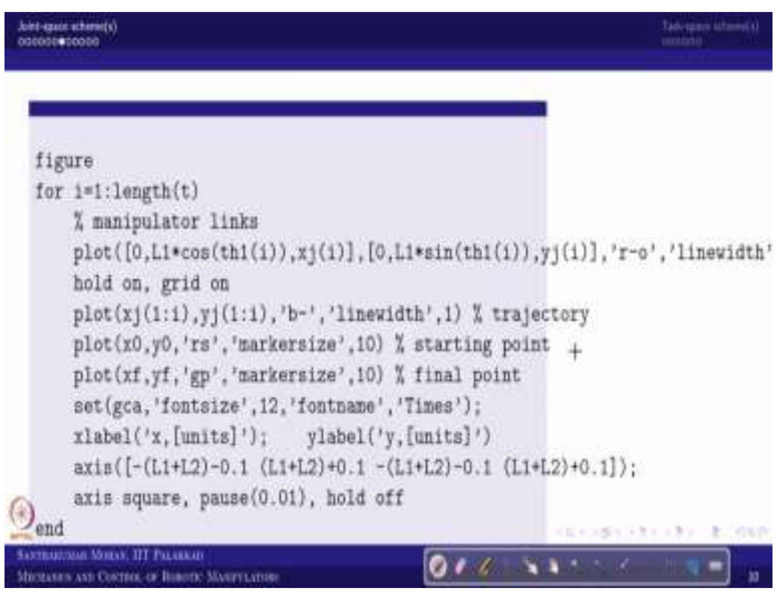
So, for that we should know what is, we call coefficient matrix A and we should know the you can save known input as a vector. So, here the theta 1 initial theta 1 dot initial theta 1 final and theta 1 dot final need to be known for finding the first segment joint space trajectory. Similarly, second joints you can say trajectory also can be generated. So, based on that t c 1 and t c 2 for the joint 1 and joint 2 coefficients which is we call you can say a 0 1 to a 3 1. Similarly, a 0 2 to a 3 2 that we can calculate.

(Refer Slide Time: 5:01)



So, once you calculate then we can go to the loop where you are trying to run based on t 0 to t f profile. So, the theta 1 and theta 2 can be generated once you know since this is synchronous. So, both would be synchronously moving. So, in the sense it starts from t 0 to t f completely both joints are rotating. So, in that sense, we can write theta 1 and theta 2 just to plot as the animated way. So, we are trying to find the forward kinematic model so, that your x end effector and the y end effector can be visualized and then you can show in an animated way.

(Refer Slide Time: 5:39)

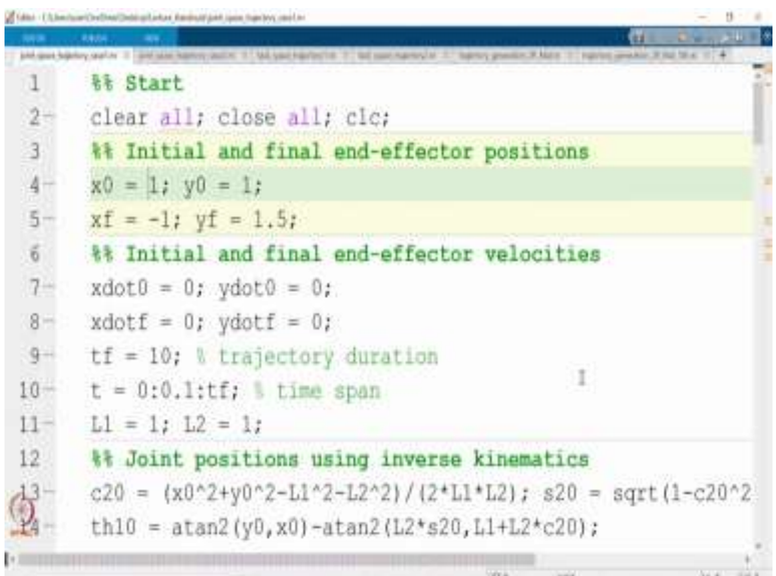


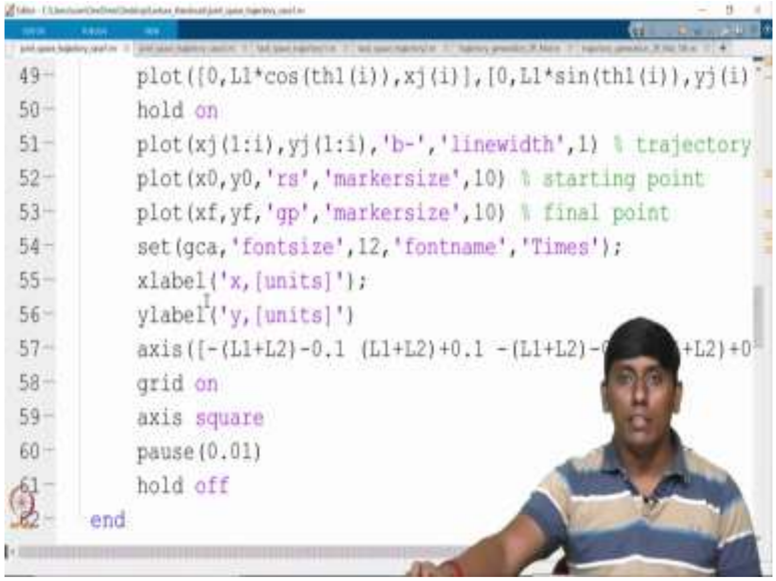
For doing the animation we have tried to take this particular you can say MATLAB code. So, we are trying to plot 0 as the origin then x 1 and x 2 x 1 we can write as l 1 cos theta 1 and x 2 is nothing but you are a final end effector point. So, that is corresponding to the x similarly y we start from 0 origin then l 1 sine theta 1 is the y 1 and y 2 would be your end effector position. So, based on that we can draw, and I want to just show the continuous profile how it is generated.

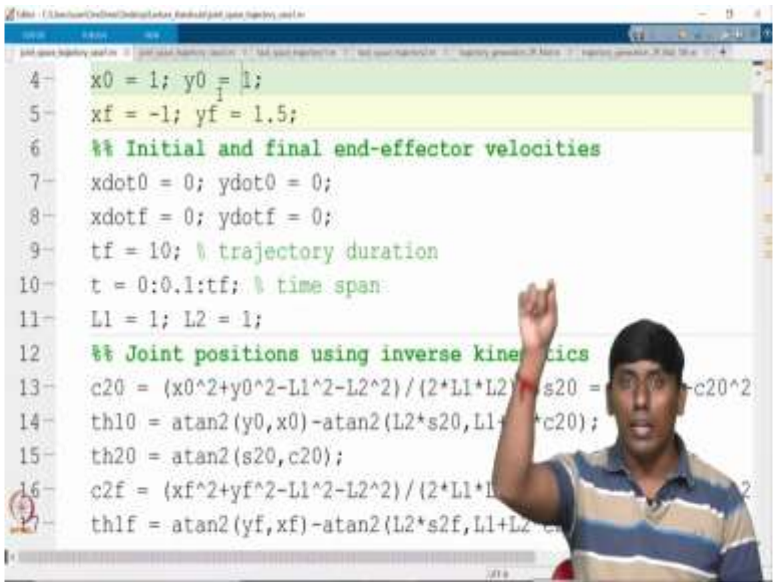
So, I just added that x j and y j from 1 to i. So, when the loop is iterating it will show the continuous plot. And I want to show what is the initial and final point. So, I just added two additional plotting where r s is the red square will come in the starting point. And the g p is nothing, but you can say g pentagon will or you can say star will come at the green color at the final point.

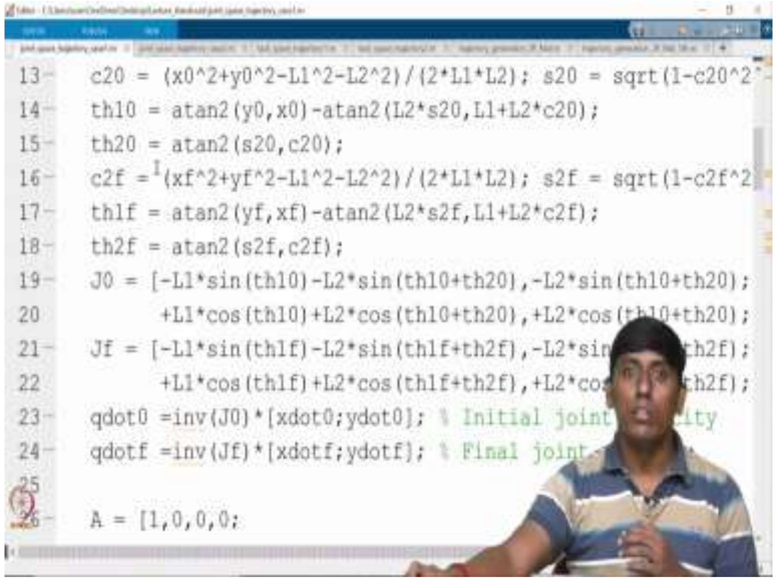
So, and we have run everything and just to show the animated I pause and hold off. So, in the sense every time it would be super impose one image to another image but hold off command is there. So, it is look like its continuous motion.

(Refer Slide Time: 6:58)

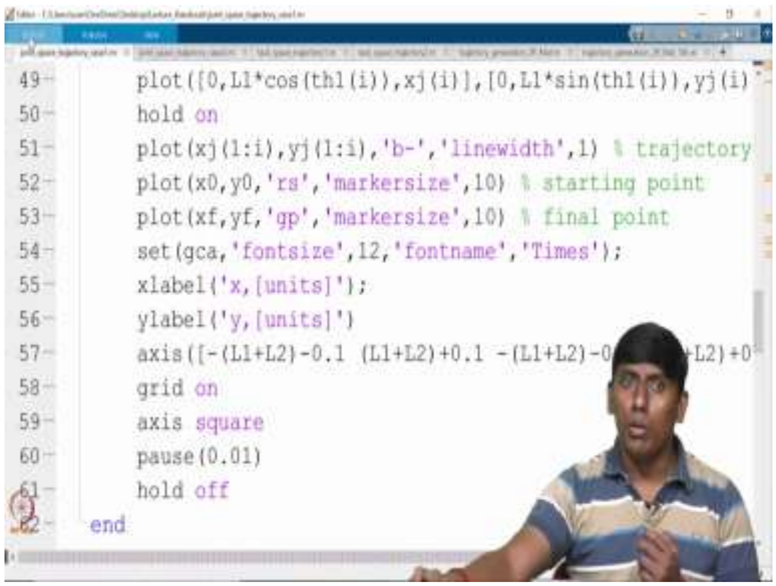



plot([0,L1*cos(thl(i)),xj(i)],[0,L1*sin(thl(i)),yj(i)
hold on
plot(xj(1:i),yj(1:i),'b-','linewidth',1) % trajectory
plot(x0,y0,'rs','markersize',10) % starting point
plot(xf,yf,'gp','markersize',10) % final point
set(gca,'fontsize',12,'fontname','Times');
xlabel('x,[units]');
ylabel('y,[units]')
axis([-(L1+L2)-0.1 (L1+L2)+0.1 -(L1+L2)-
grid on
axis square
pause(0.01)
hold off
end

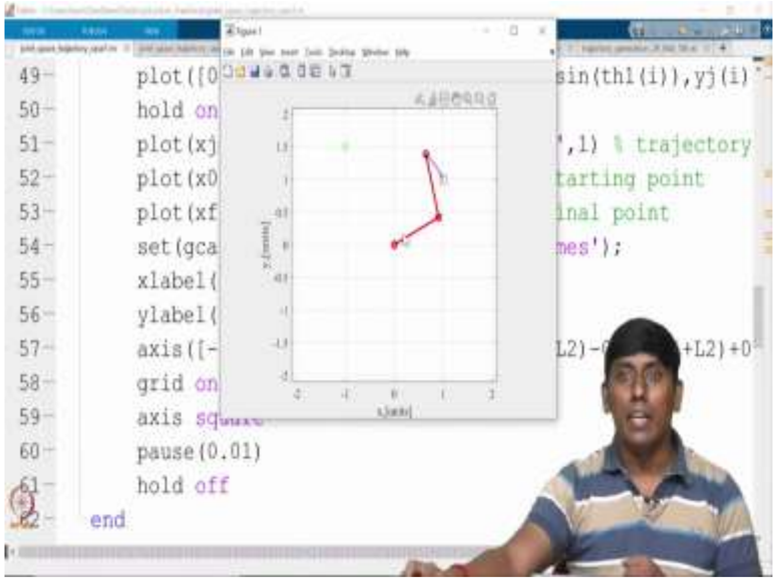

x0 = 1; y0 = 1;
xf = -1; yf = 1.5;
%% Initial and final end-effector velocities
xdot0 = 0; ydot0 = 0;
xdotf = 0; ydotf = 0;
tf = 10; % trajectory duration
t = 0:0.1:tf; % time span
L1 = 1; L2 = 1;
%% Joint positions using inverse kine
c20 = (x0^2+y0^2-L1^2-L2^2)/(2*L1*L2)
th10 = atan2(y0,x0)-atan2(L2*s20,L1+
th20 = atan2(s20,c20);
c2f = (xf^2+yf^2-L1^2-L2^2)/(2*L1*
th1f = atan2(yf,xf)-atan2(L2*s2f,L1+L2

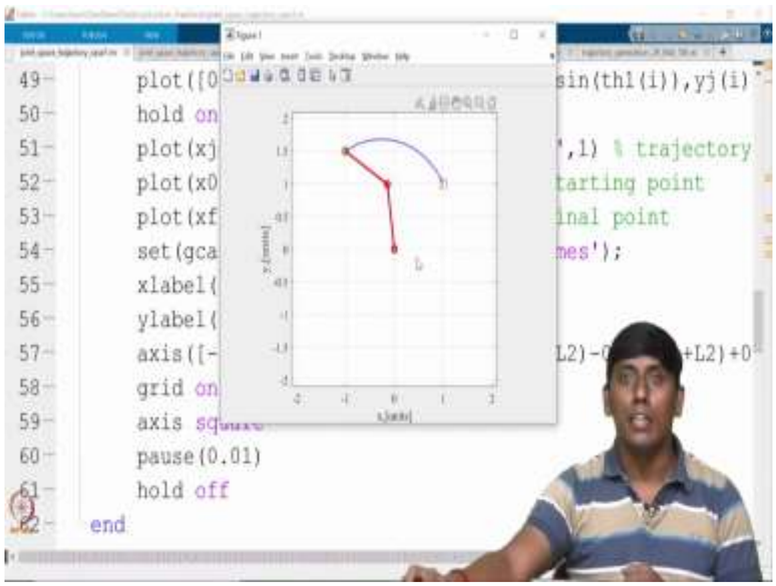


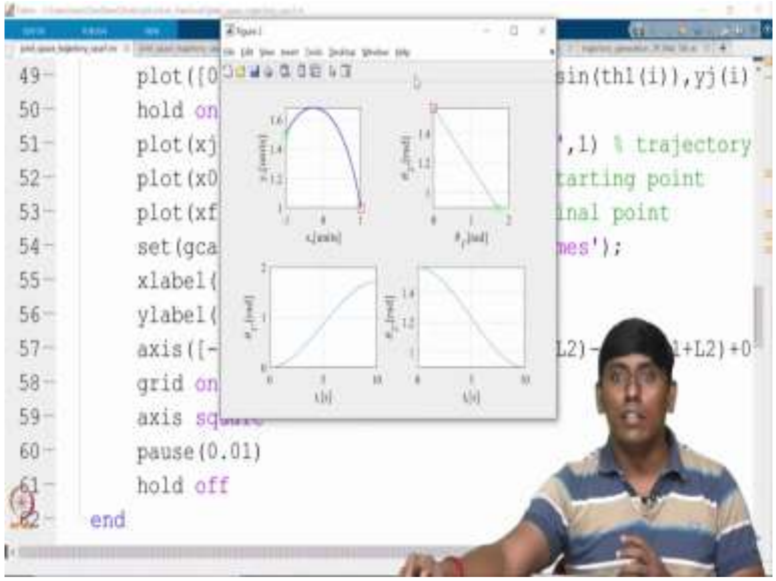
```
c20 = (x0^2+y0^2-L1^2-L2^2)/(2*L1*L2); s20 = sqrt(1-c20^2
th10 = atan2(y0,x0)-atan2(L2*s20,L1+L2*c20);
th20 = atan2(s20,c20);
c2f = (xf^2+yf^2-L1^2-L2^2)/(2*L1*L2); s2f = sqrt(1-c2f^2
th1f = atan2(yf,xf)-atan2(L2*s2f,L1+L2*c2f);
th2f = atan2(s2f,c2f);
J0 = [-L1*sin(th10)-L2*sin(th10+th20),-L2*sin(th10+th20);
      +L1*cos(th10)+L2*cos(th10+th20),+L2*cos(th10+th20);
Jf = [-L1*sin(th1f)-L2*sin(th1f+th2f),-L2*sin    th2f);
      +L1*cos(th1f)+L2*cos(th1f+th2f),+L2*co    th2f);
qdot0 =inv(J0)*[xdot0;ydot0]; % Initial joint    ity
qdotf =inv(Jf)*[xdotf;ydotf]; % Final joint

A = [1,0,0,0;
```

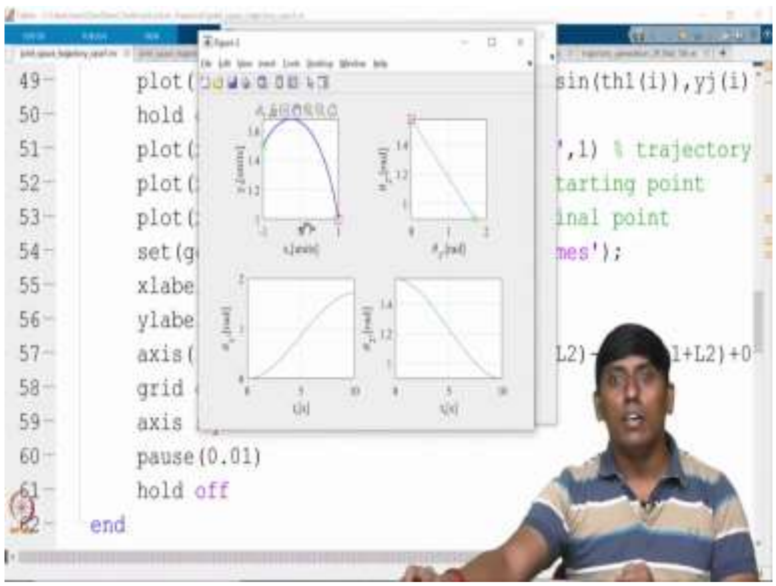


```
    plot([0,L1*cos(th1(i)),xj(i)],[0,L1*sin(th1(i)),yj(i)
    hold on
    plot(xj(1:i),yj(1:i),'b-','linewidth',1) % trajectory
    plot(x0,y0,'rs','markersize',10) % starting point
    plot(xf,yf,'gp','markersize',10) % final point
    set(gca,'fontsize',12,'fontname','Times');
    xlabel('x,[units]');
    ylabel('y,[units]')
    axis([-(L1+L2)-0.1 (L1+L2)+0.1 -(L1+L2)-0    L2)+0
    grid on
    axis square
    pause(0.01)
    hold off
end
```


plot([0
sin(th1(i)),yj(i)
hold on
plot(xj
',1) % trajectory
plot(x0
tarting point
plot(xf
inal point
set(gca
mes');
xlabel(
ylabel(
axis([-
L2)-
+L2)+0
grid on
axis sq
pause(0.01)
hold off
end


plot([0
sin(th1(i)),yj(i)
hold on
plot(xj
',1) % trajectory
plot(x0
tarting point
plot(xf
inal point
set(gca
mes');
xlabel(
ylabel(
axis([-
L2)-
+L2)+0
grid on
axis sq
pause(0.01)
hold off
end





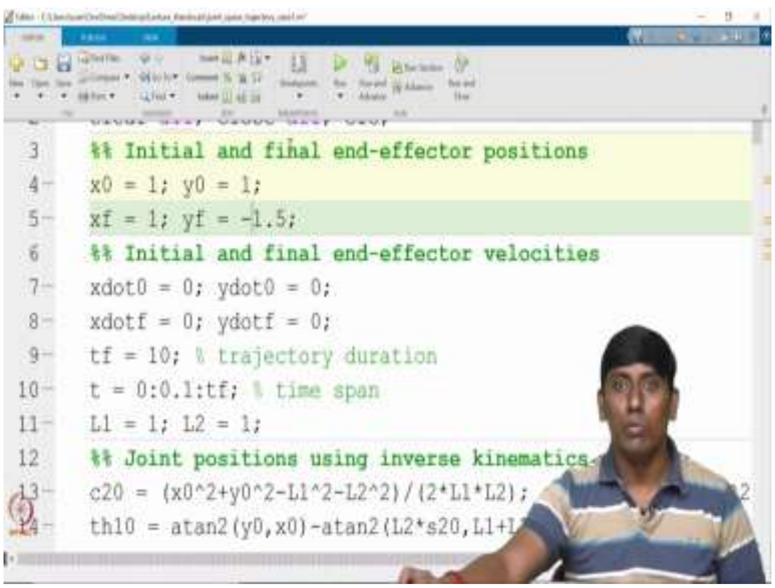
So, just to show this I just go to the MATLAB window, and this is the code which we were discussing. So, you can see this is the code. So, now if I run this, so, what did start. So, the x initial is 1 meter and y initial also like 1 meter it is somewhere like this and the final also like this. So now we have not given theta 1 0 and theta 1 f directly. So, what we can do, we can do the inverse kinematics.

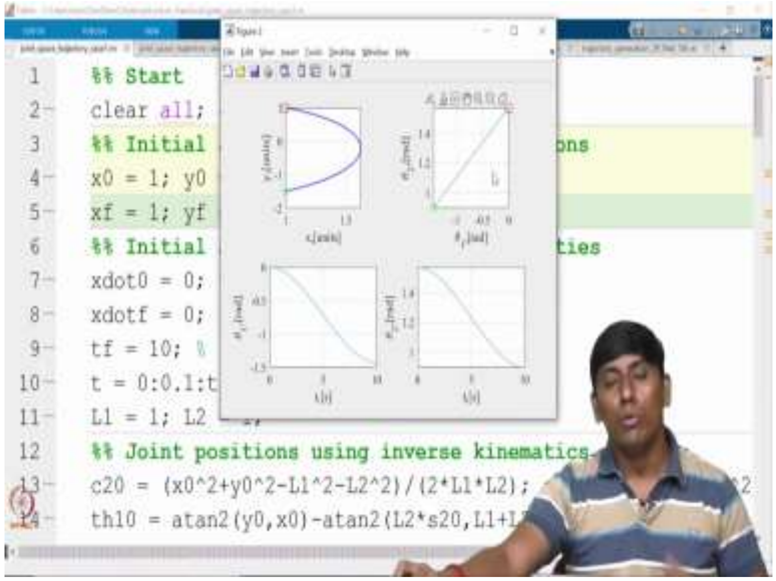
So now the inverse kinematics will have two solutions. So, we will take one solution, which is the positive solution of s 2. So, now based on this we can run this code. So, then you can feel it what I want to explain, so, you can see like now it is starting from here and it is ending. So, you can see like it is more like the robot is moving right. So, when I ran this so you can see like the physically the robot is moving that realization you can feel it in MATLAB animation.

So, I did not make any video but still it looked like a video, right? Because I have made it a super impose. So, the next figure I want to show it. So, what is the; you can save profile or the trajectory generated in task space and as well as the joint space. The joint space theta 1 to theta 2 you can see it is like a straight line.

So, that is what we call it like joint space trajectory consider as a straight line the profile starts from 0 to you can say this is theta 1 0 to theta 1 f it is a smooth profile and I did not plot the velocity when you want you can do the velocity profile. So, now, this is the theta 2 f this is theta 2 0. So, like that we can generate, so, even you want to realize further.

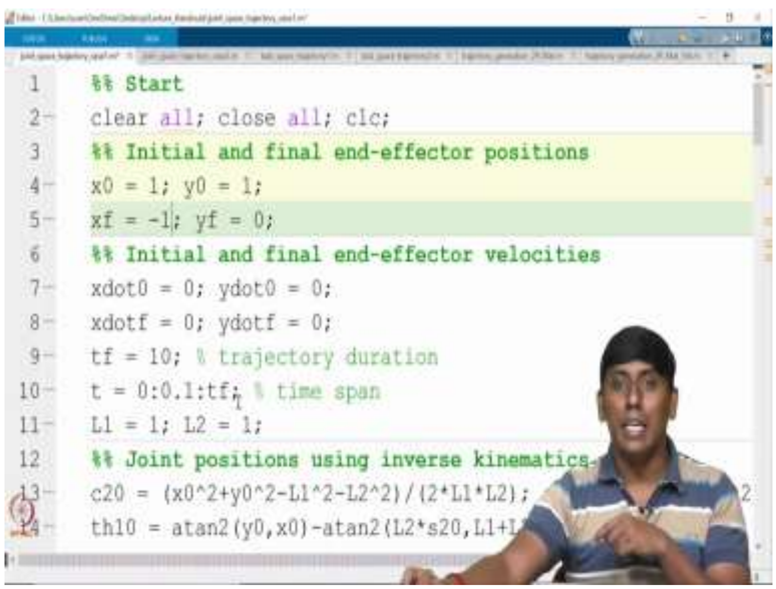
(Refer Slide Time: 8:47)

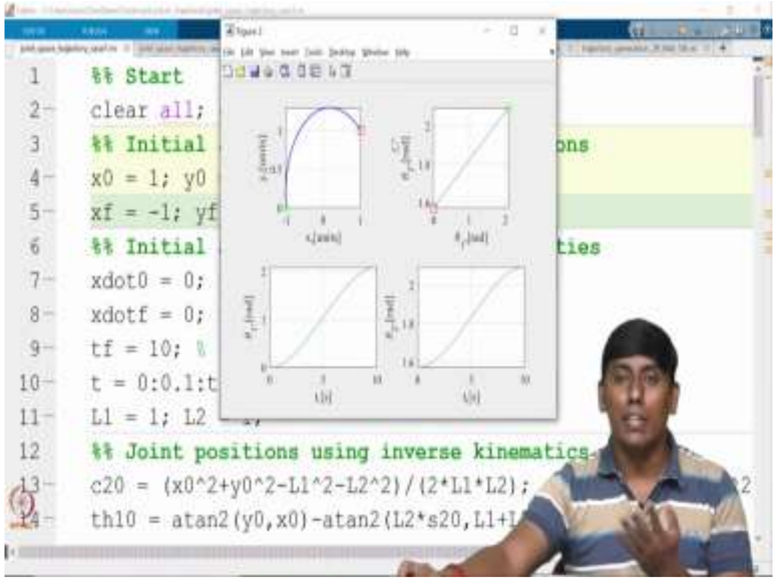




So, the only condition is this initial input what we are giving that should be in the case of what you call in the workspace. So, we can see, I have given interchange so, the initial point is same, but the final point is slightly modified. So, now, you can see that this is the initial point and the final point, you can see it is moving it. So, this is what the profile and this is what happening, right? So, now, this is more and more very clear.

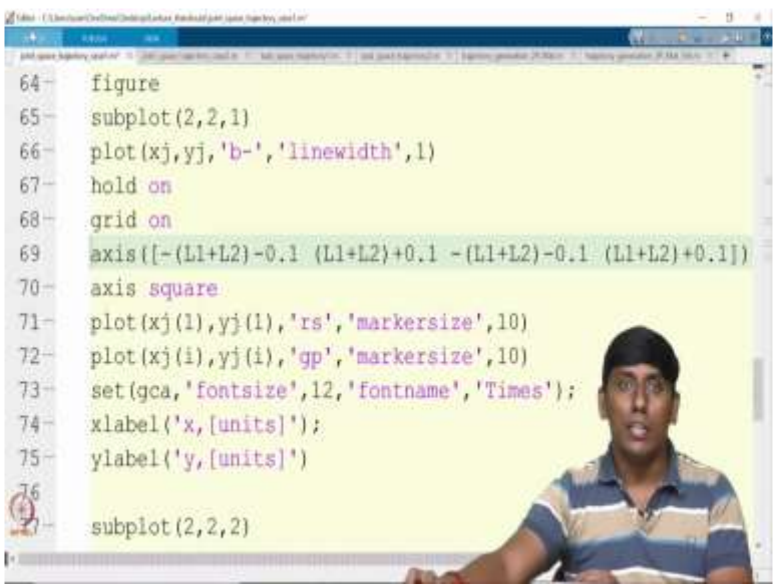
(Refer Slide Time: 9:19)





So, now, we will actually go back so go back to the code. So now I am looking at this, so I am just giving it 0. And this is it like I am putting it so minus 1. So, now, I am trying to give because you are l 1 and l 2 is 1 and 1. So maximum it is on the circle of radius of 2 meter. So, that is what the range within that you can choose anything. So, anyhow next lecture, we would be seeing the generation. So, you can feel it. It is more you can say beneficial. So, you want to see this in a proper shape. So even we can make the boundary.

(Refer Slide Time: 9:56)



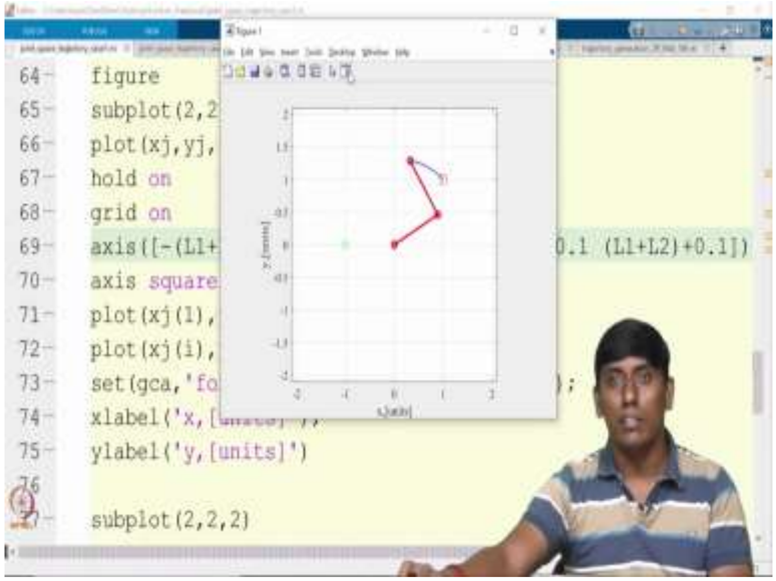


figure
subplot(2,2
plot(xj,yj,
hold on
grid on
axis([-(L1+
0.1 (L1+L2)+0.1])
axis square
plot(xj(1),
plot(xj(i),
set(gca,'fo
xlabel('x,[
ylabel('y,[units]')
subplot(2,2,2)

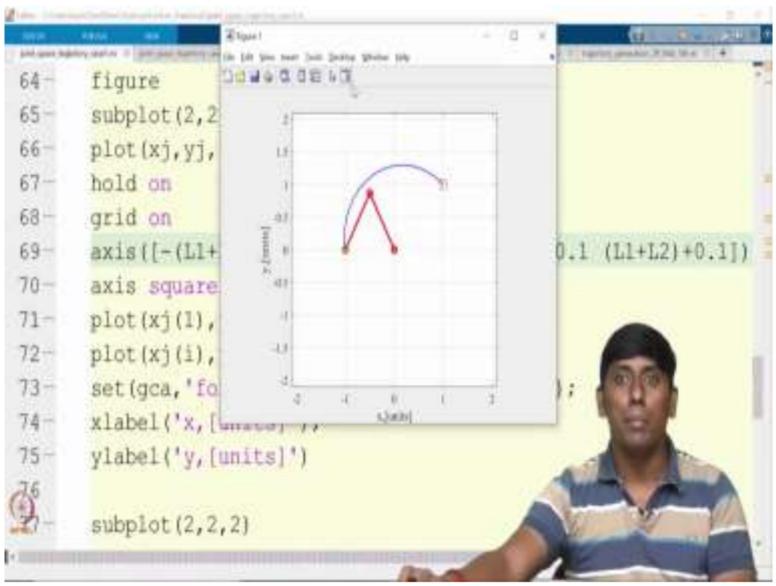
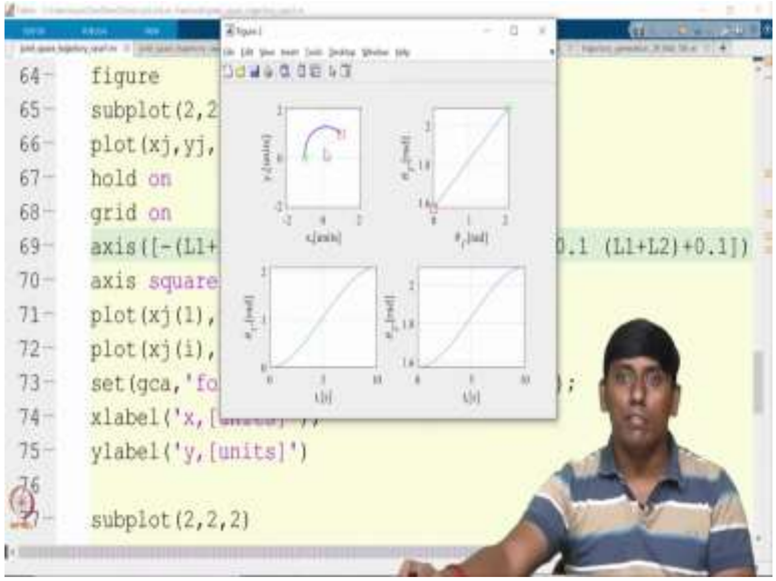
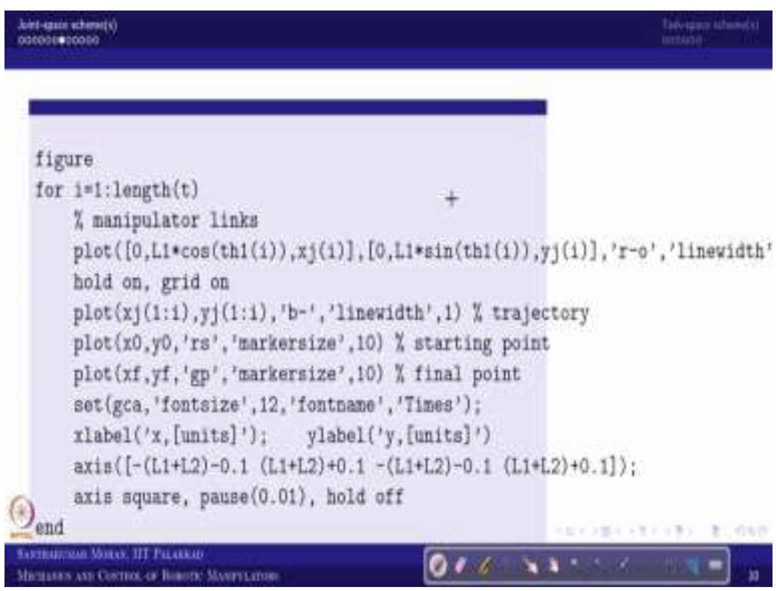


figure
subplot(2,2
plot(xj,yj,
hold on
grid on
axis([-(L1+
0.1 (L1+L2)+0.1])
axis square
plot(xj(1),
plot(xj(i),
set(gca,'fo
xlabel('x,[
ylabel('y,[units]')
subplot(2,2,2)



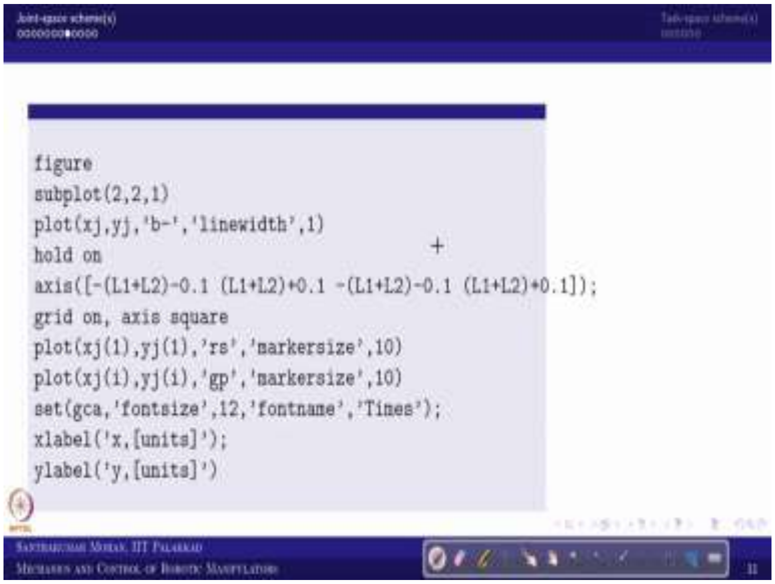
So, the boundary we can right now I make it this as commented. So, now if I make it. It would be giving a real size like how it is happening the same size it is giving this is the real size. So, now these all what you call the joint space trajectory with a case 1 where the synchronous.

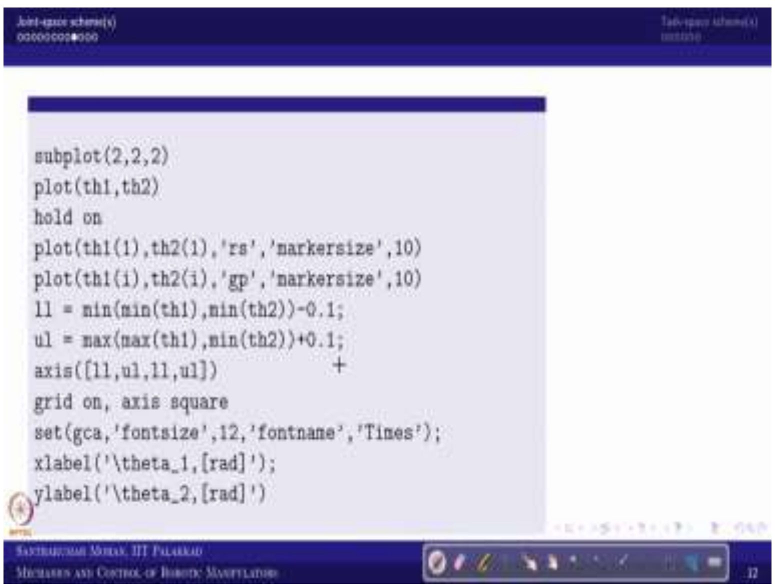
(Refer Slide Time: 10:20)

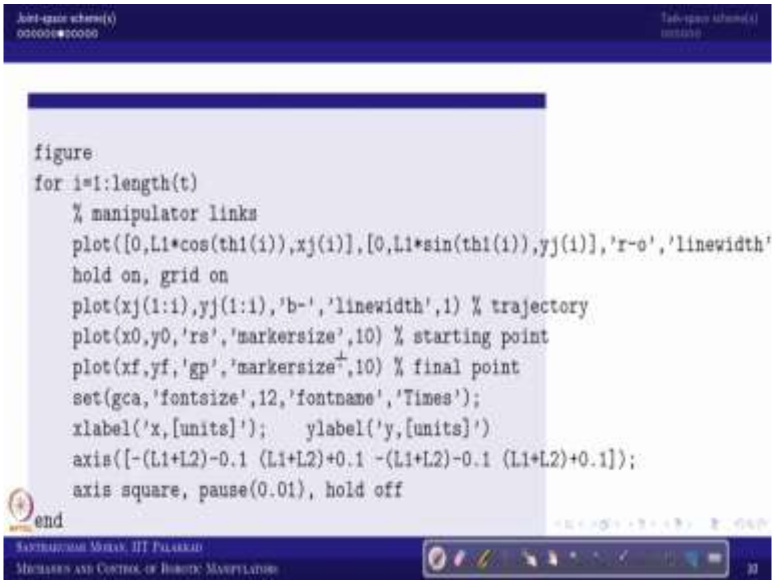


We will go back to the slide where the non-synchronous we call in series of motion.

(Refer Slide Time: 10:25)

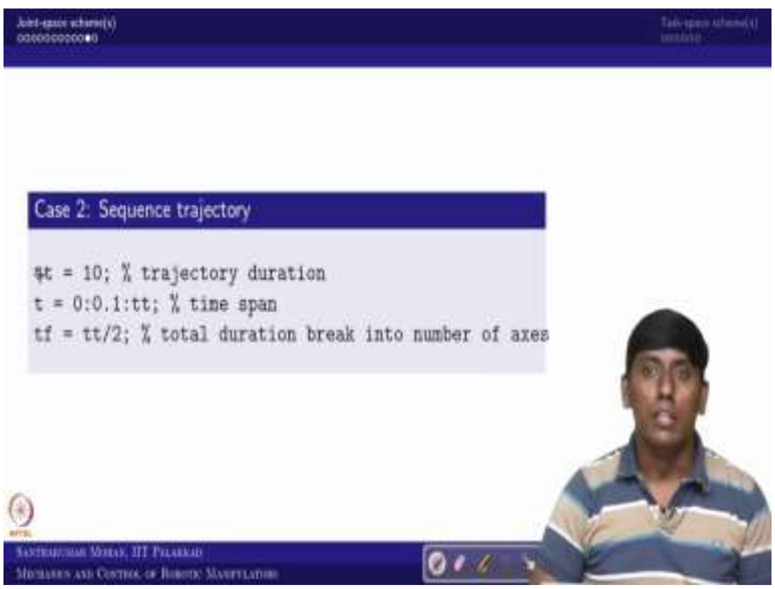






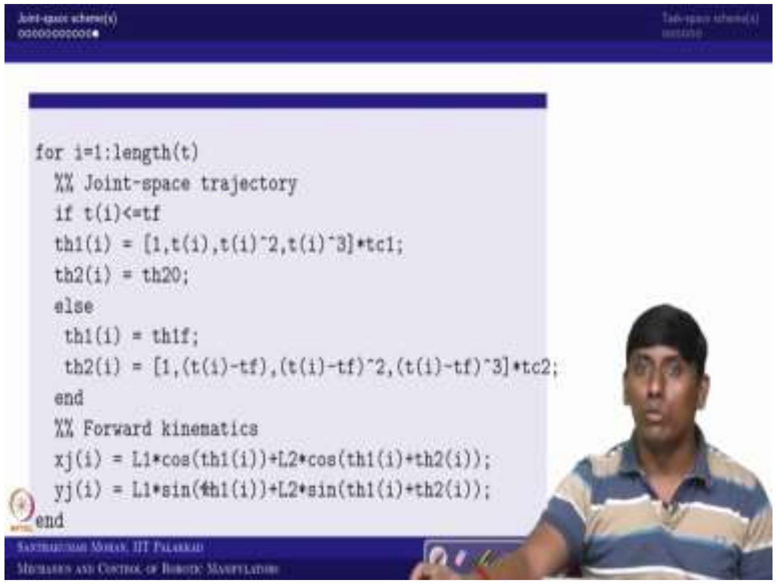
So, for that what we need to do it. So, these are all the other plots. So, I will just see the non-synchronous. So, the non-synchronous you have to do it a slightly trickier way. So, these all the plot which we have already done.

(Refer Slide Time: 10:42)



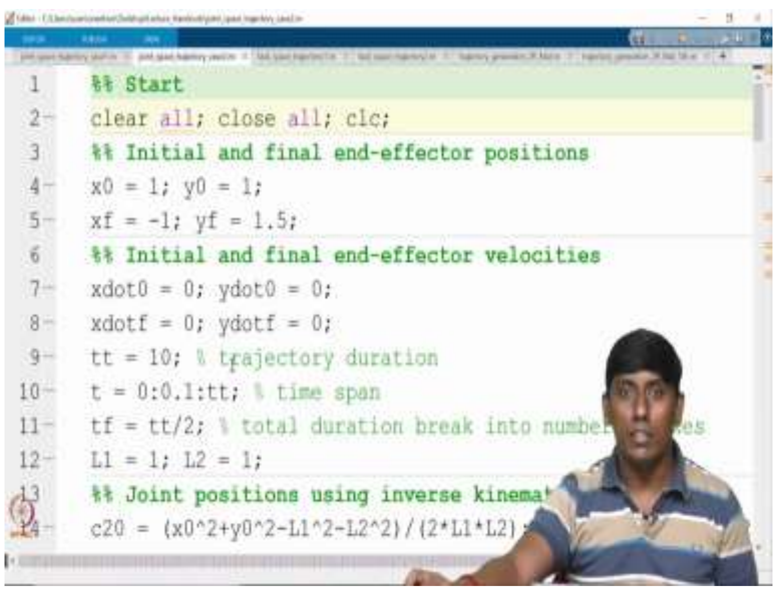
So, the non-synchronous way we call sequence trajectory. So, here we call the tt is that total trajectory duration. But now I need to divide that into tf into number of segments. Because each joint would be rotating independently for example, now 10 second I have 2 joints, so, I break into 5 second each. So, first 5 second first joint will rotate the next 5 second the second joint will rotate. So, that is what we have seen as a sequence trajectory.

(Refer Slide Time: 11:12)



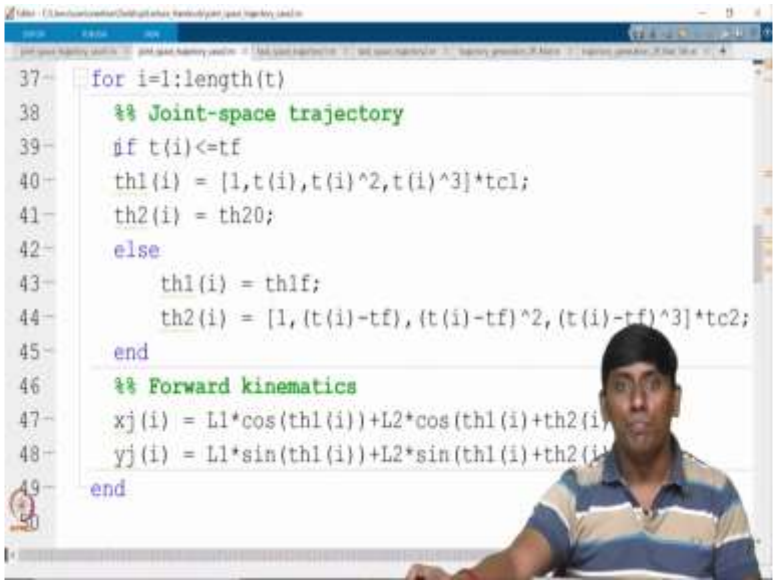
So, that is what we are trying to see. So, in that sense the coefficient calculation is same, but what you can see like the t goes 0 to tf here tf is total duration divided by 2. So, that would be generated. So, then the correspondingly the theta 2 would be the same initial it maintained that so, then you can see the second loop the final would be maintained for theta 1 and theta 2 move as a cubic polynomial. So, now, the same way we have done the forward kinematics just to plot.

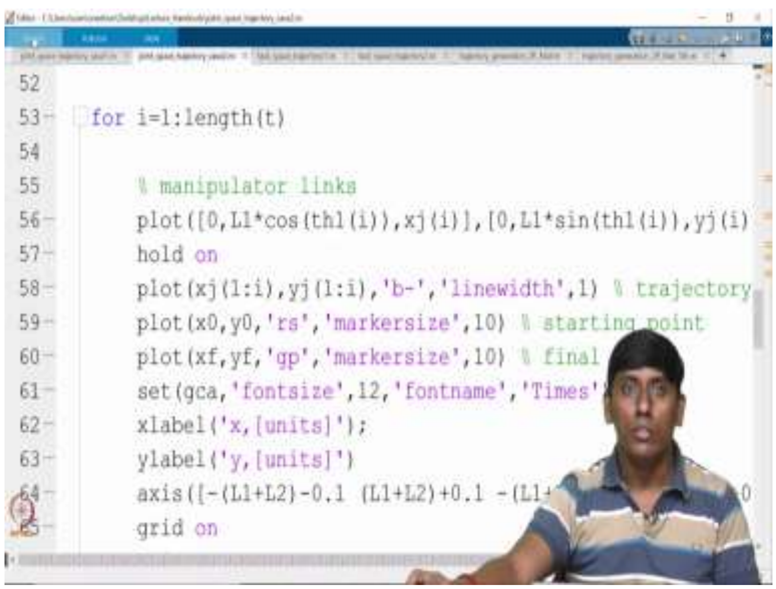
(Refer Slide Time: 11:50)

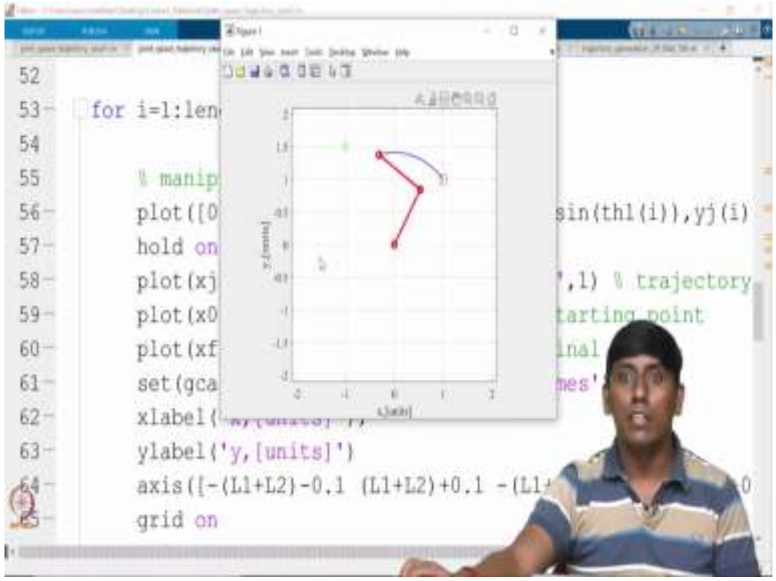


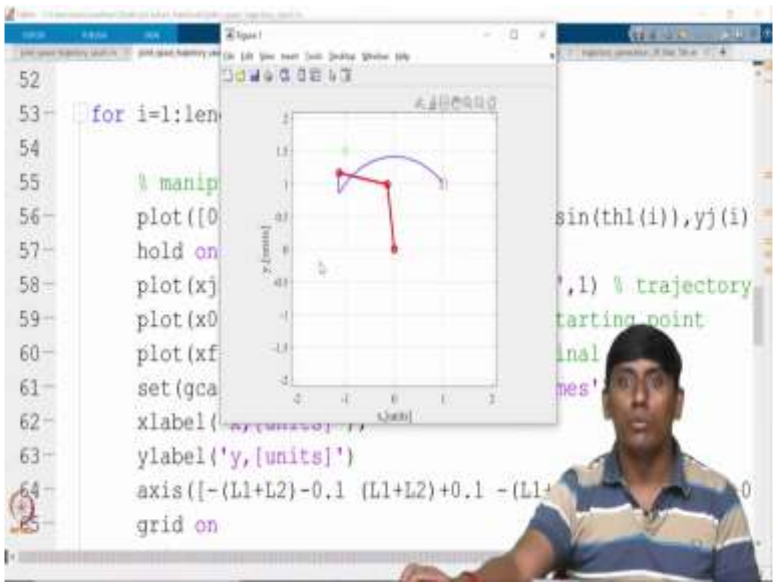
And now, we will see the same thing in the MATLAB window which will be more clear. So, you can see like these are standard which we have done in the previous code only change is. So, instead of tf I have taken tt and the tt and tf why I have made it because I make it the tf always the segment time. The segment time here is you can say tt divided by 2. So, based on that you will get the inverse kinematics and all.

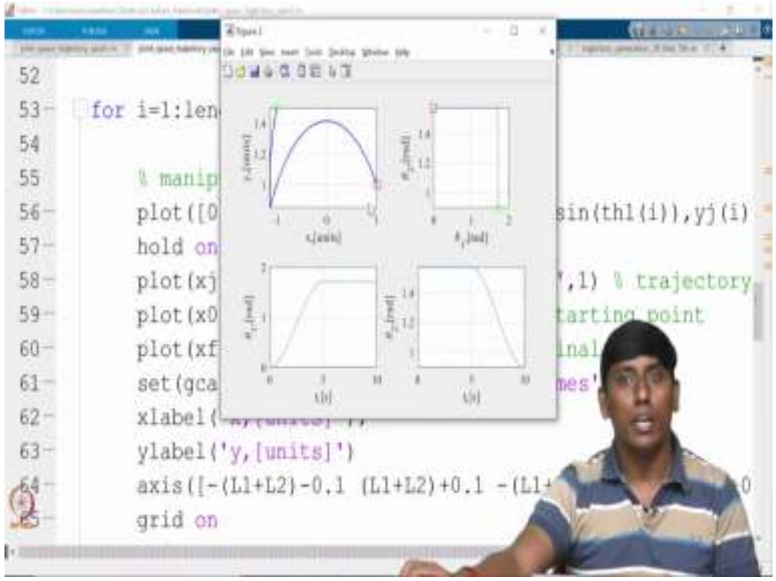
(Refer Slide Time: 12:17)





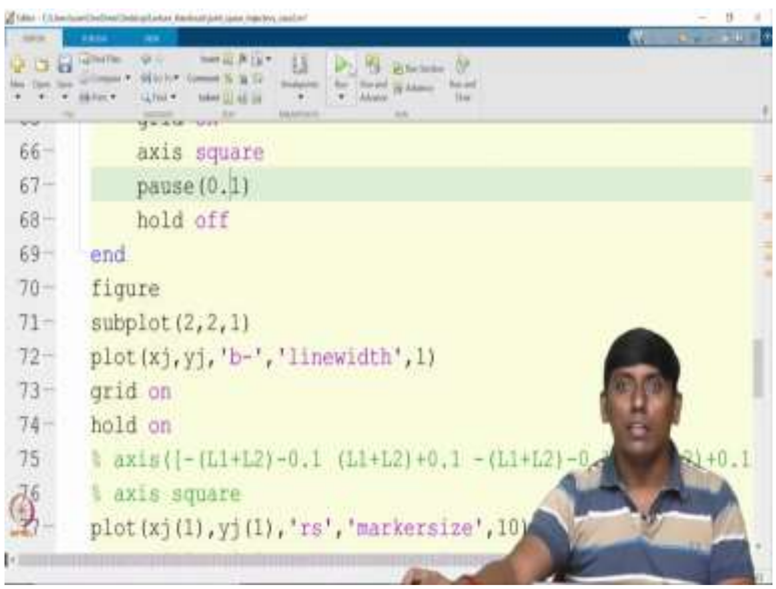

for i=1:len
% manip
plot([0
sin(th1(i)),yj(i)
hold on
plot(xj
',1) % trajectory
plot(x0
tarting point
plot(xf
inal
set(gca
mes'
xlabel(
ylabel('y,[units]')
axis([-(L1+L2)-0.1 (L1+L2)+0.1 -(L1
grid on

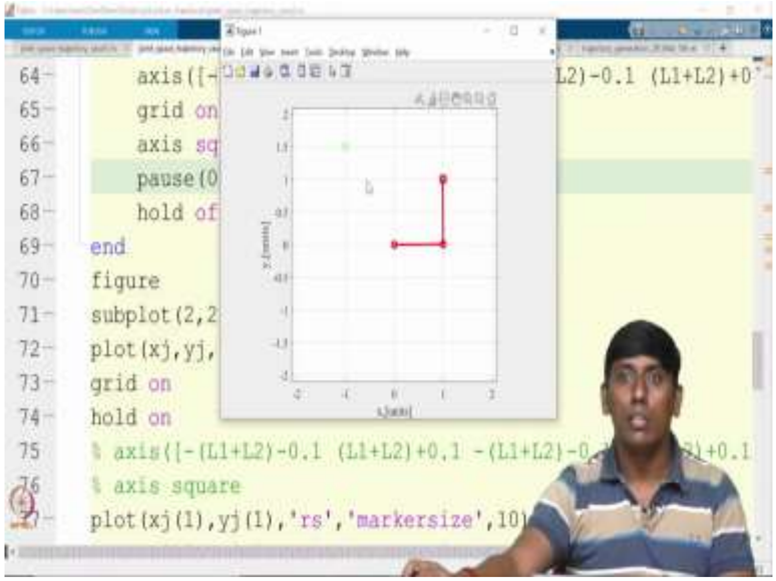

for i=1:len
% manip
plot([0
sin(th1(i)),yj(i)
hold on
plot(xj
',1) % trajectory
plot(x0
tarting point
plot(xf
inal
set(gca
mes'
xlabel(
ylabel('y,[units]')
axis([-(L1+L2)-0.1 (L1+L2)+0.1 -(L1
grid on

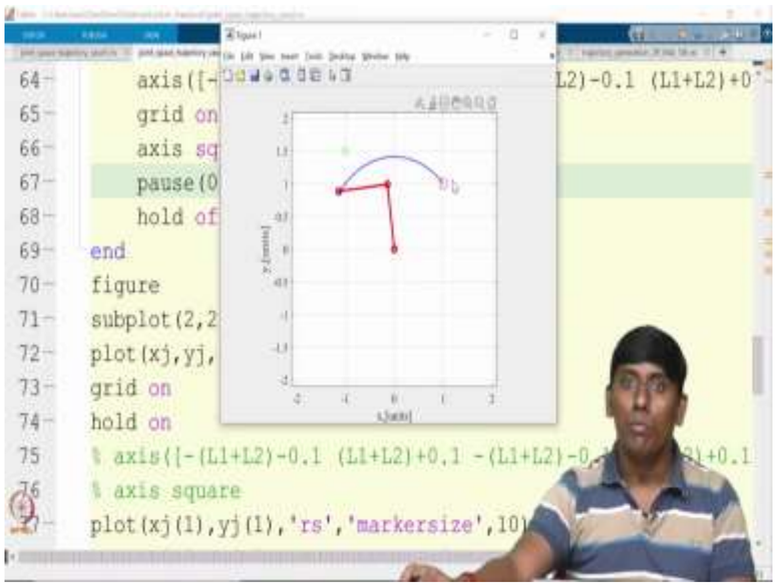


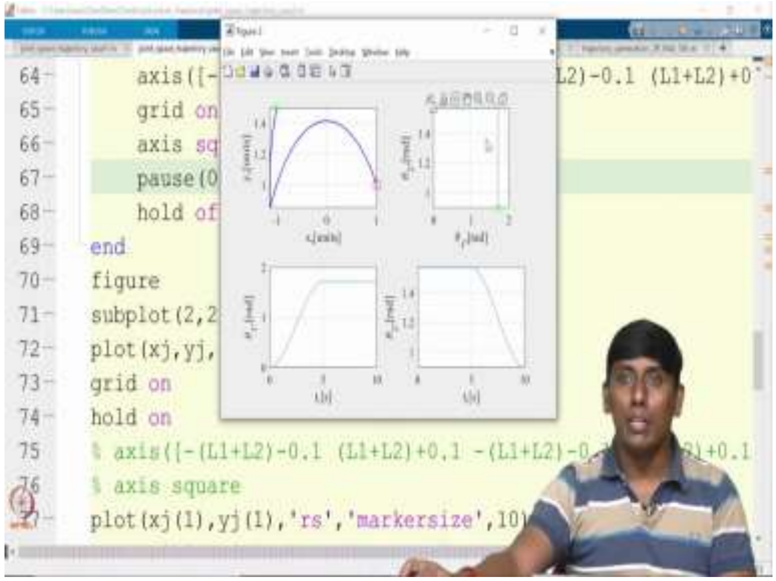
So, based on that you will get the joint space trajectory in two ways. So, if I run you will be very clear to this. So, I will just run this then you will feel it. You can see like the first joint is rotated the second joint is rotating. So, it is little faster.

(Refer Slide Time: 12:33)



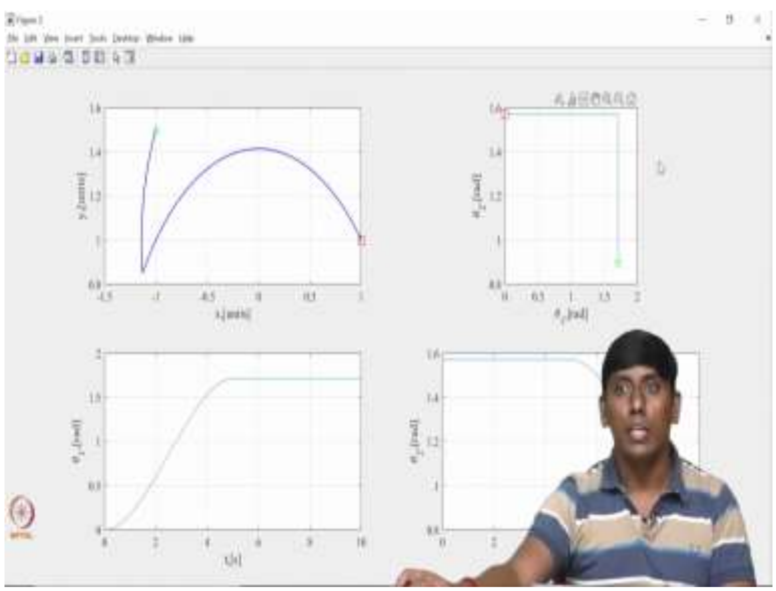

axis([-
L2)-0.1 (L1+L2)+0
grid on
axis sq
pause(0
hold of
end
figure
subplot(2,2
plot(xj,yj,
grid on
hold on
% axis([-(L1+L2)-0.1 (L1+L2)+0.1 -(L1+L2)-0.
)+0.1
% axis square
plot(xj(1),yj(1),'rs','markersize',10)


axis([-
L2)-0.1 (L1+L2)+0
grid on
axis sq
pause(0
hold of
end
figure
subplot(2,2
plot(xj,yj,
grid on
hold on
% axis([-(L1+L2)-0.1 (L1+L2)+0.1 -(L1+L2)-0.
)+0.1
% axis square
plot(xj(1),yj(1),'rs','markersize',10)



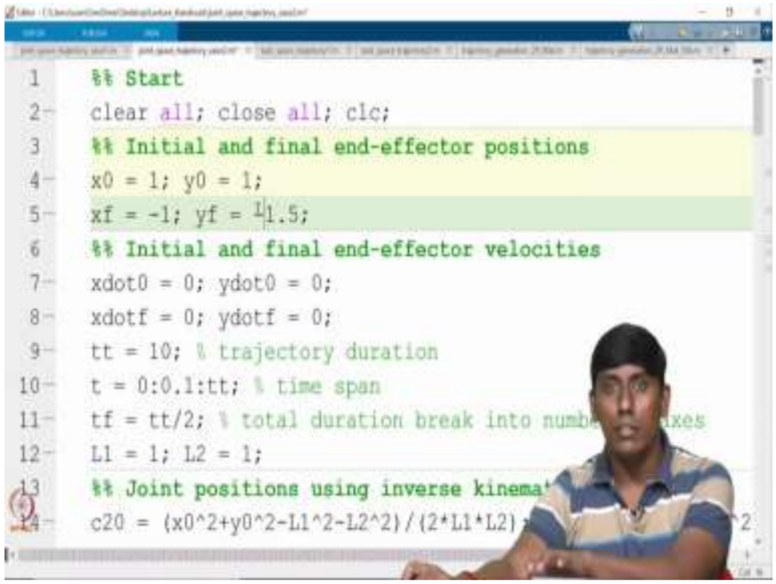
So, I will just make it the pause command probably 100 milliseconds rather than the you can say 10 milliseconds. So, now you can see the first joint only rotate the second joint is as whatever this theta 2 0 is maintained. The first joint rotated once it is reached that final then it is the second joint rotate. So, while second joint rotate you can see the first joint remain same. So, you can look at it.

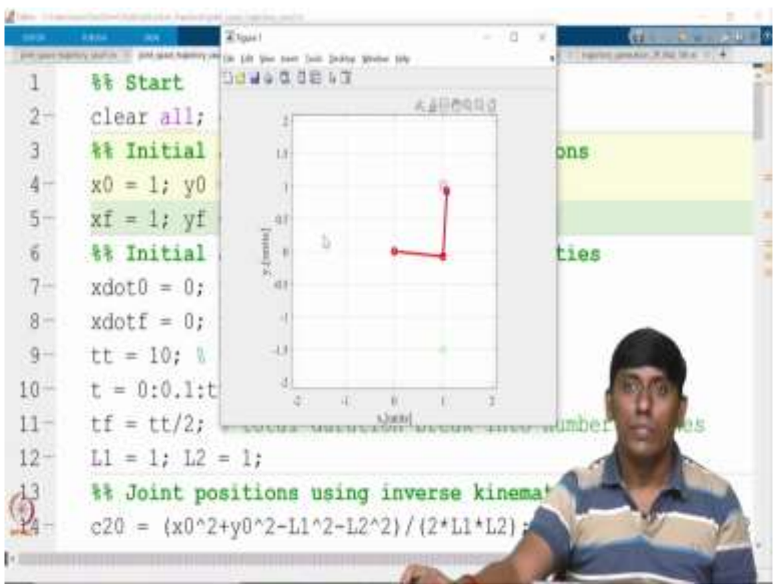
(Refer Slide Time: 13:04)

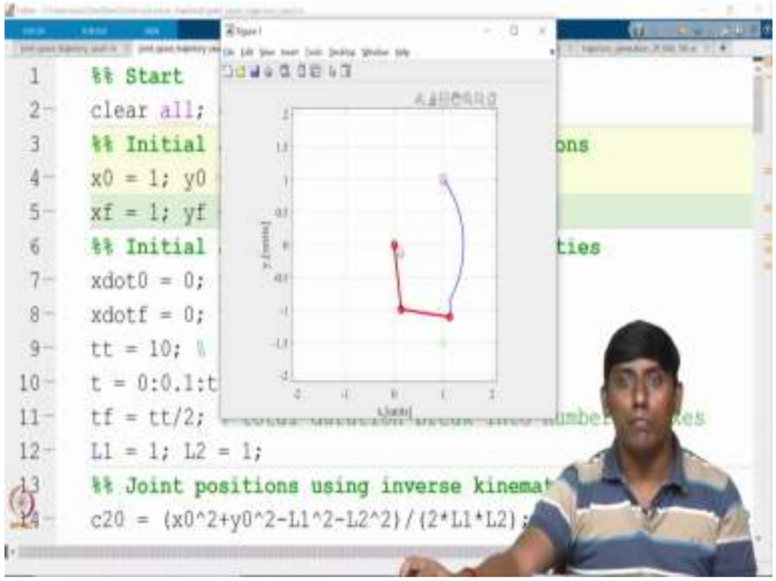


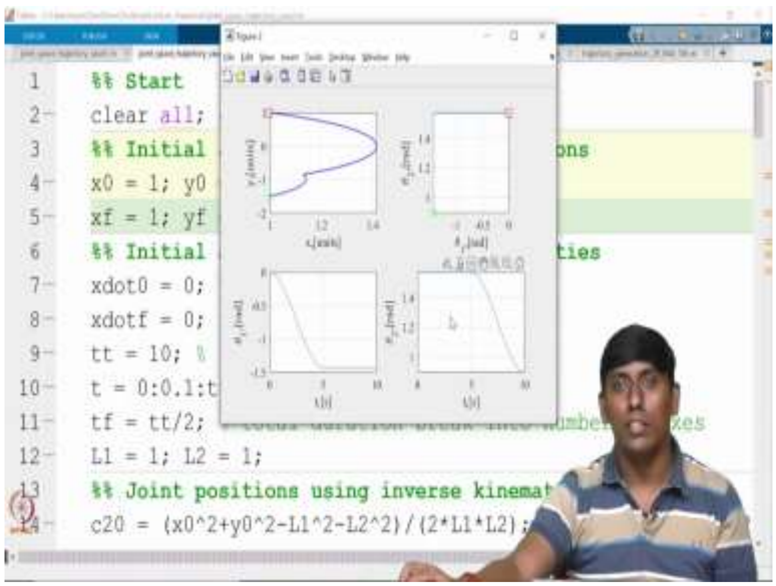
So, from here. So, you can see like the first joint rotated. So, that time the theta 1 vary from theta initial to final. So, the second segment, So, theta final theta 1 final remains same, but the correspondingly theta 2 starts from 1 to final. So, that is what the realization.

(Refer Slide Time: 13:26)





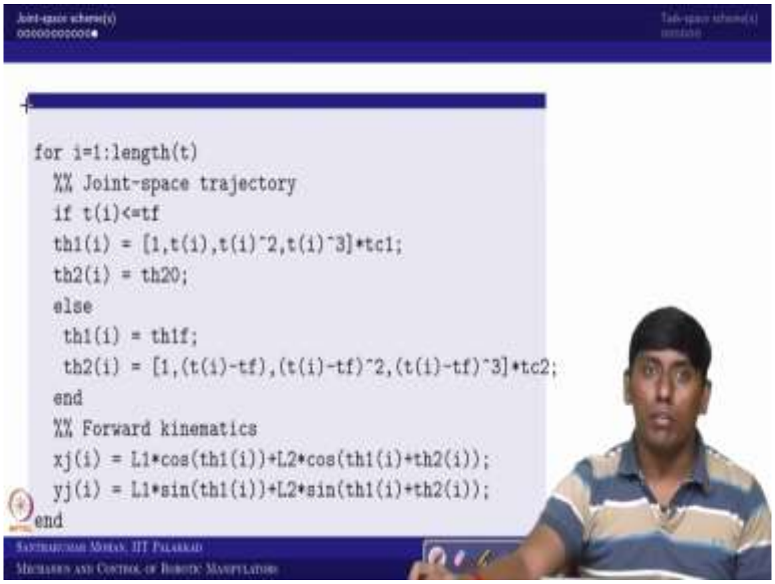


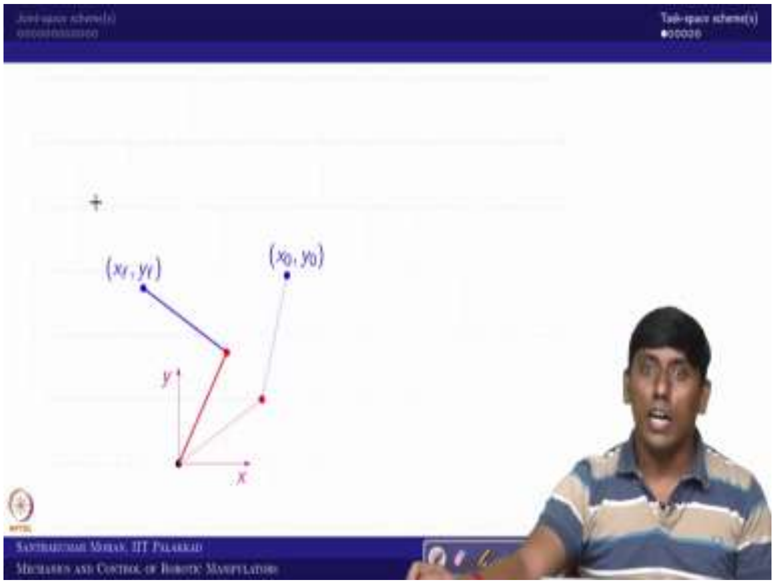


So, now, you can want to more clear, so, I will just give you random theta 1f. So, in the sense, so, I just give it probably a different point. So, I just put it this minus. So, I just show it this is vertical. So, then you can see it sorry. So, you can see it while running it you can see the first joint only rotating the second joint remains same constant. So, once it reached a particular location, you can see the second joint rotate that time the first joint remains same.

So, it is very clearly seen that the explicit trajectory generation. But this is not preferred at all. However, so, most of the you can say kinematics solvers, and all done with this way individual joint would be move in series. So especially if you go to ROS, we call robot operating system. So, I already told in the lecture these things are very clearly visible there.

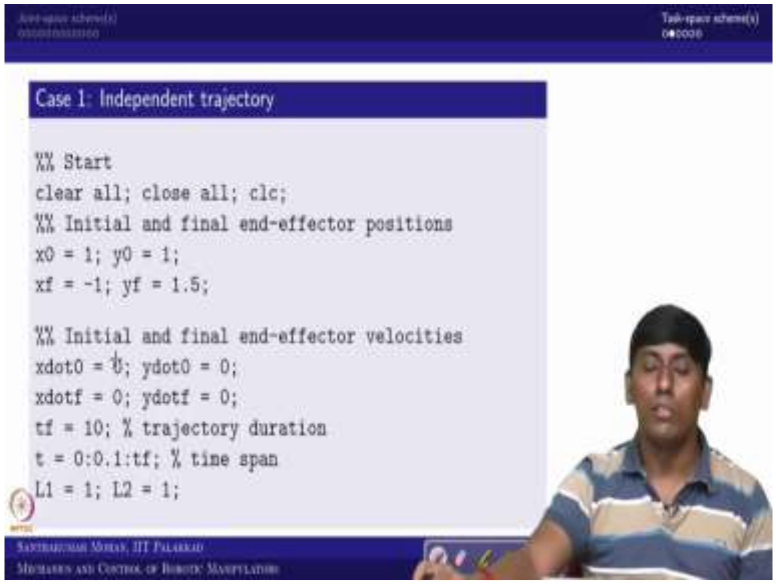
(Refer Slide Time: 14:35)





So, now going back to the slide. So, we will see this whatever we have seen is the joint space trajectory can we see in a task space even the task space we can see in that case. So, x initial and final would be given. Similarly, y initial and final would be given. So, this is the initial position, and this is the final position. We want to follow this in this sense it is supposed to be followed as a straight line. So, the straight line is not assured because the x dot of 0 and x dot f if it is nonzero then it will not maintain.

(Refer Slide Time: 15:05)

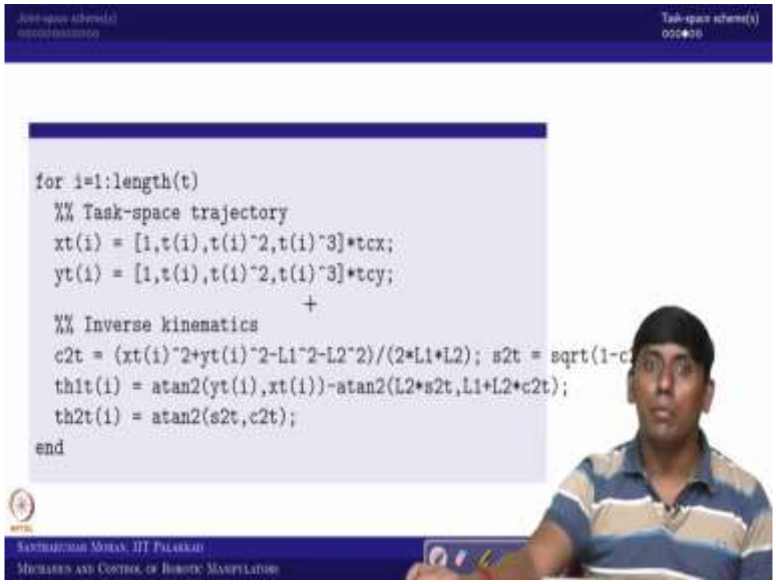


So, we will see that how it comes. So, in the sense here you can see so, we are trying to do the first case independent trajectory in the sense x trajectory and y trajectory we are trying to do it in independent cubic polynomial, we are not doing it a synchronous me in the sense once you do the x based on the x, y can be generated no not like that. So, in that sense we are trying to do this. So, here you can see initial and final velocity are given. And we have taken the same time which is 5 you can say second and l 1 and l 2 are same.

(Refer Slide Time: 15:41)

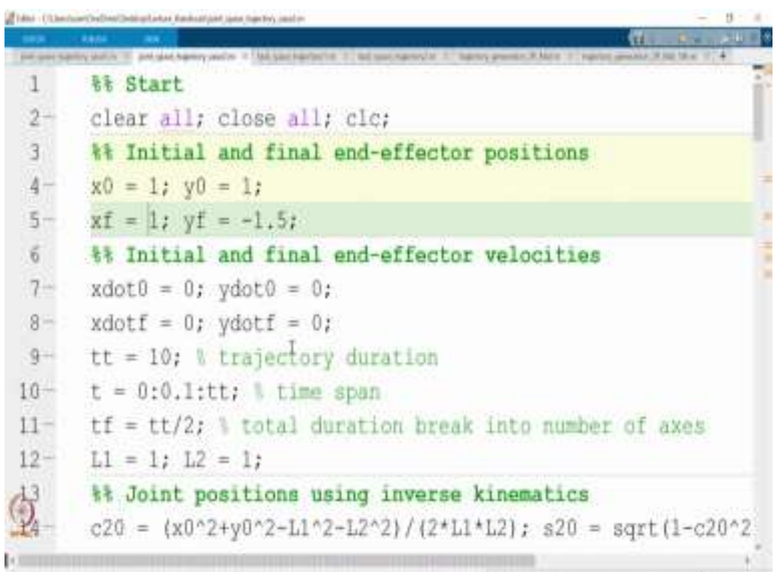
So, now, we have taken the same A matrix, but bx and by are the two other vector based on that the coefficient what we calculate tcx and tcy.

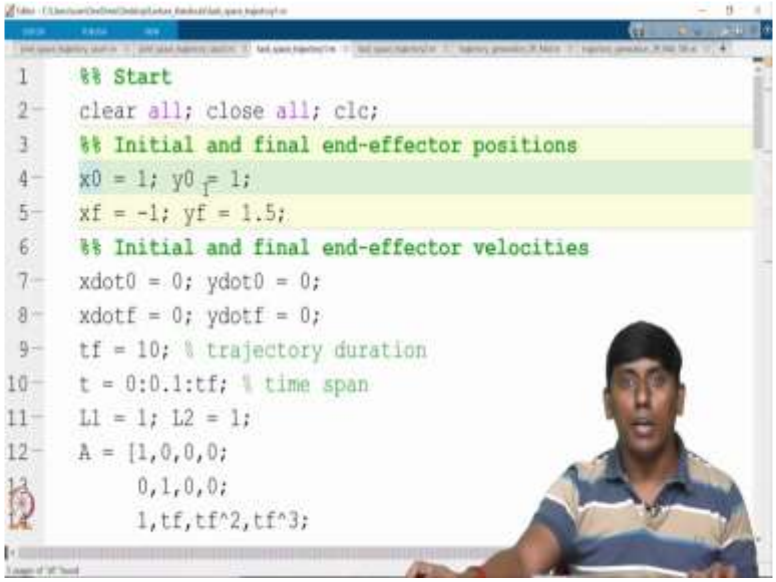
(Refer Slide Time: 15:51)

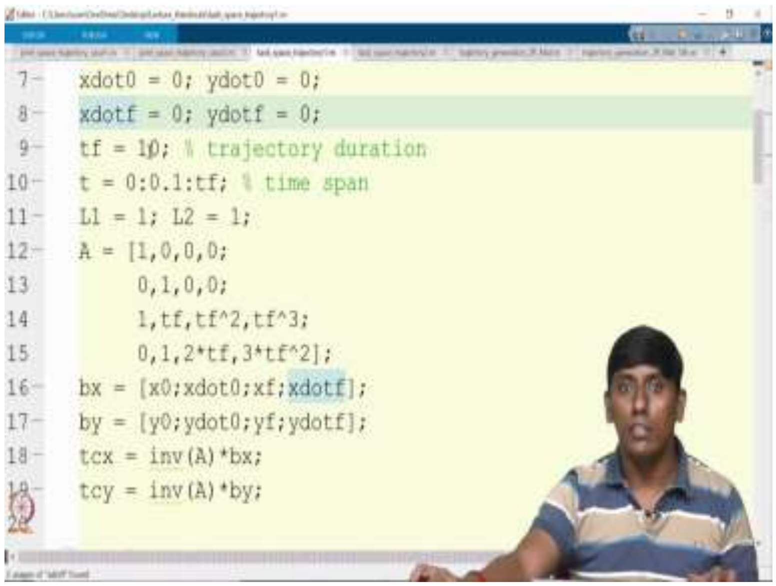


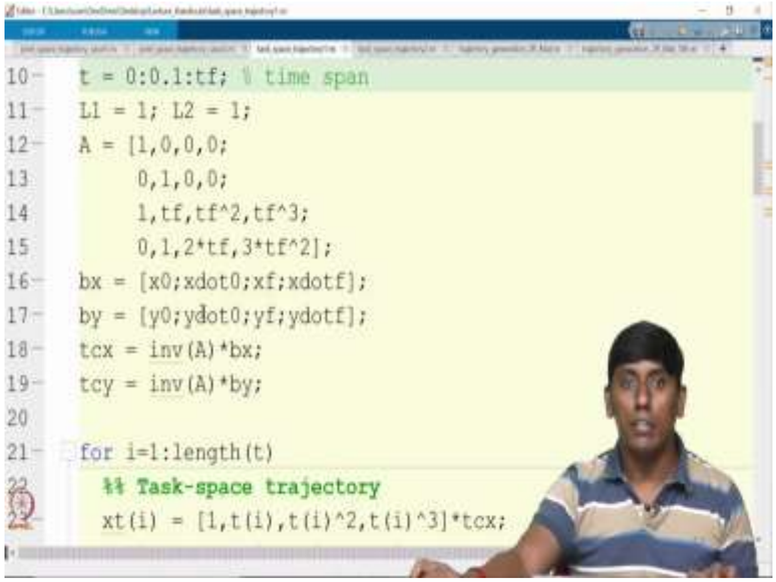
So, now in that case so, xt and yt which is written as tcx tcy. So, in the sense the x trajectory and y trajectory independently derived. So, in order to check in order to compare so, we have done the inverse kinematics just to see how the theta profile will go. So, we have done the inverse kinematics. So, now, we have used the same plotting command.

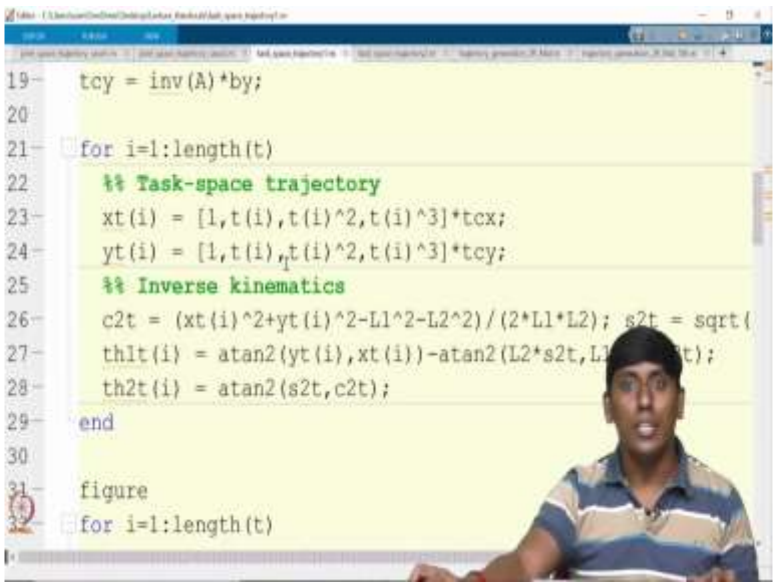
(Refer Slide Time: 16:18)




%% Start
clear all; close all; clc;
%% Initial and final end-effector positions
x0 = 1; y0 = 1;
xf = -1; yf = 1.5;
%% Initial and final end-effector velocities
xdot0 = 0; ydot0 = 0;
xdotf = 0; ydotf = 0;
tf = 10; % trajectory duration
t = 0:0.1:tf; % time span
L1 = 1; L2 = 1;
A = [1,0,0,0;
0,1,0,0;
1,tf,tf^2,tf^3;

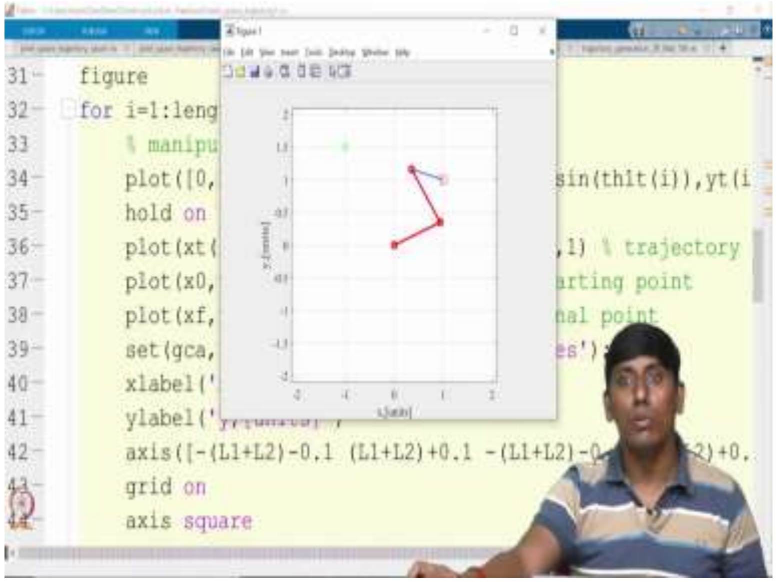

xdot0 = 0; ydot0 = 0;
xdotf = 0; ydotf = 0;
tf = 10; % trajectory duration
t = 0:0.1:tf; % time span
L1 = 1; L2 = 1;
A = [1,0,0,0;
0,1,0,0;
1,tf,tf^2,tf^3;
0,1,2*tf,3*tf^2];
bx = [x0;xdot0;xf;xdotf];
by = [y0;ydot0;yf;ydotf];
tcx = inv(A)*bx;
tcy = inv(A)*by;

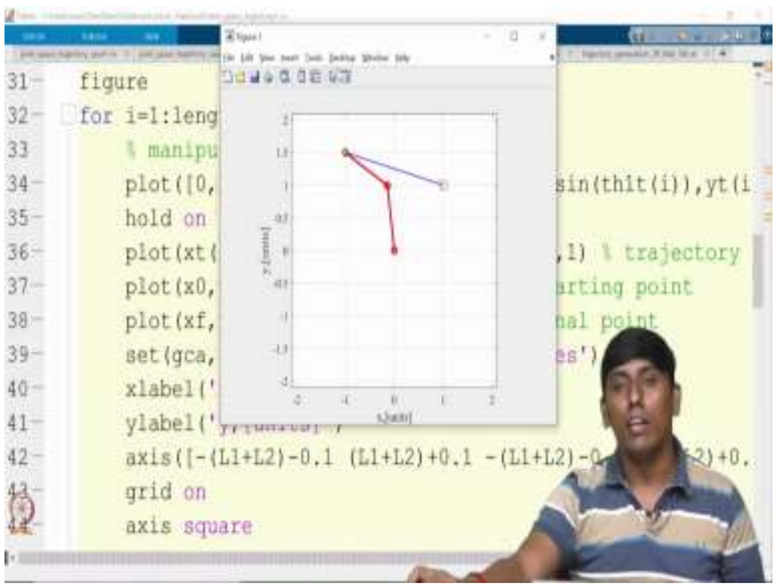


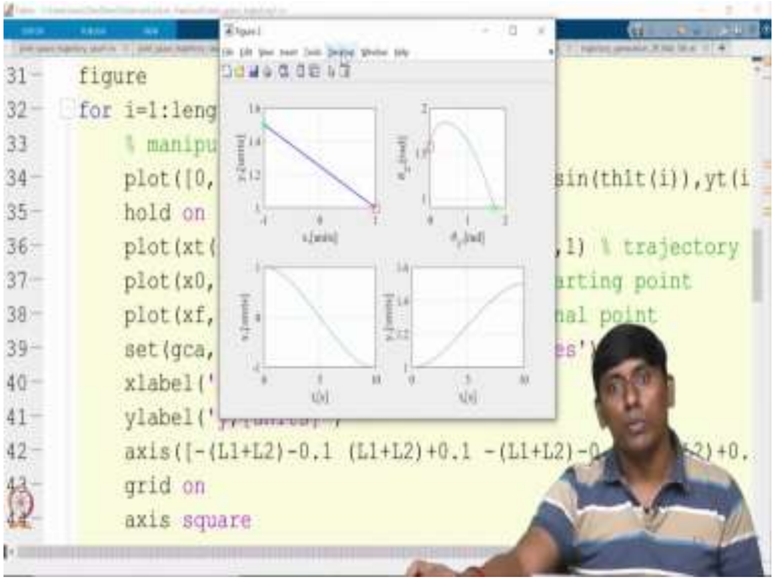


So, we will go to this the task-based space trajectory one. So, you can see that, so, these are the cases which we have shown in the MATLAB. Where initial and final you can see initial and final initial and final of y and the initial velocity of x and y and final velocity of x and y and total duration is 10 second and you can say this all we have tried. And the A matrix bx by are given and based on the tcx and tcy can be calculated. And you got the task space trajectory. Then the inverse kinematics just for our reference we calculate so that we can do the plotting option.

(Refer Slide Time: 16:58)

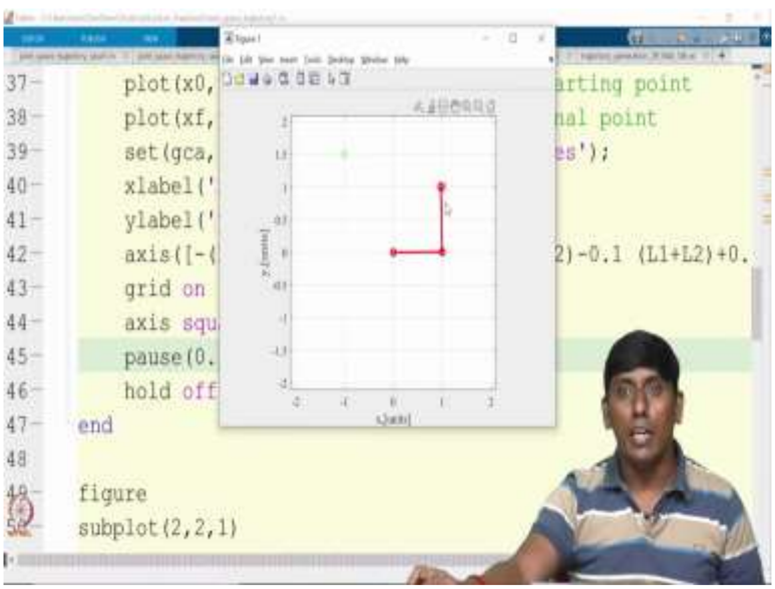


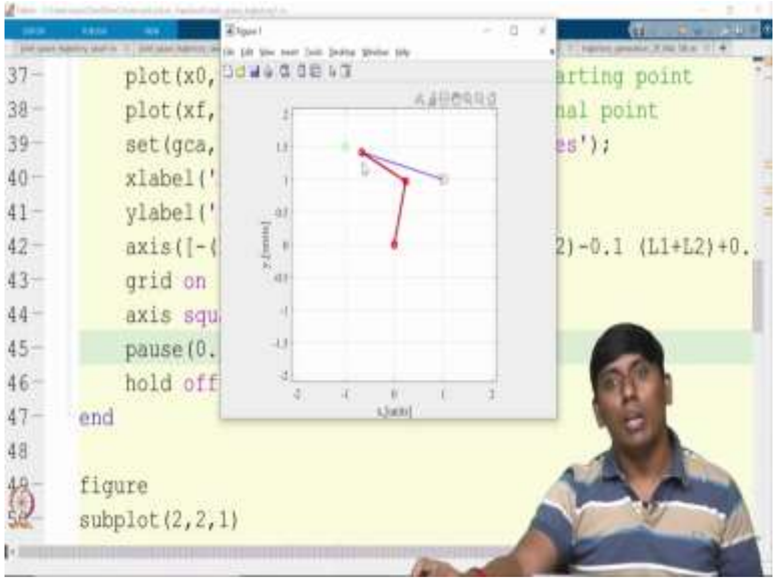


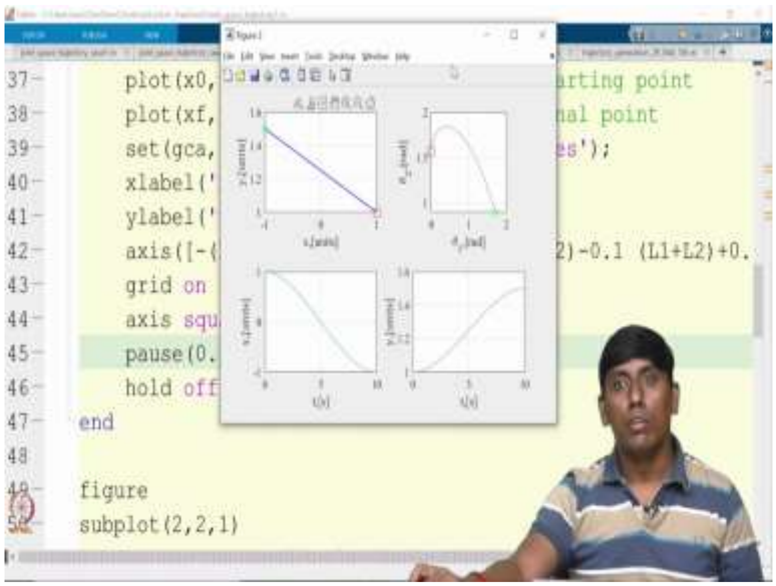


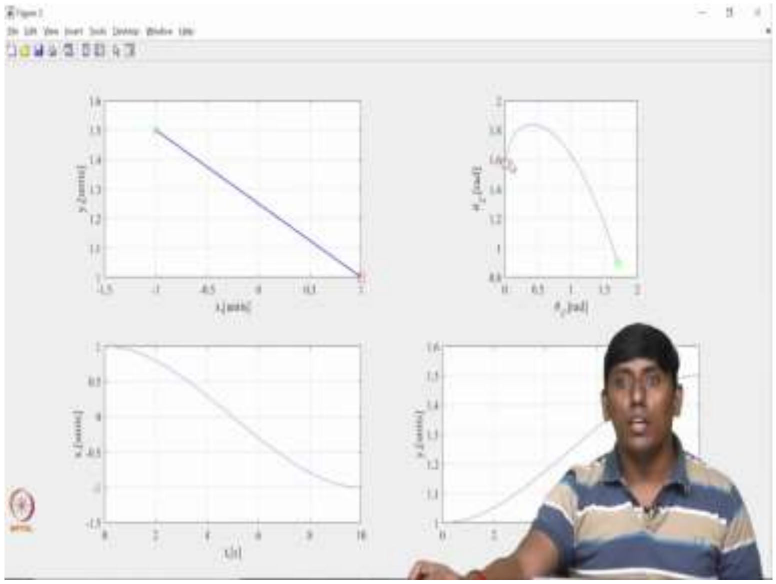
So, the plotting option we have used the same thing. So, now if I run so, what you have expected earlier it was curved. Now, in this case it is a straight line which is followed but the straight line is not assured if the initial and final velocity are nonzero. So, now you can see it.

(Refer Slide Time: 17:20)




plot(x0,
plot(xf,
set(gca,
xlabel('
ylabel('
axis([-(
grid on
axis squ
pause(0.
hold off
end
figure
subplot(2,2,1)
arting point
nal point
es');
2)-0.1 (L1+L2)+0.
Figure 1

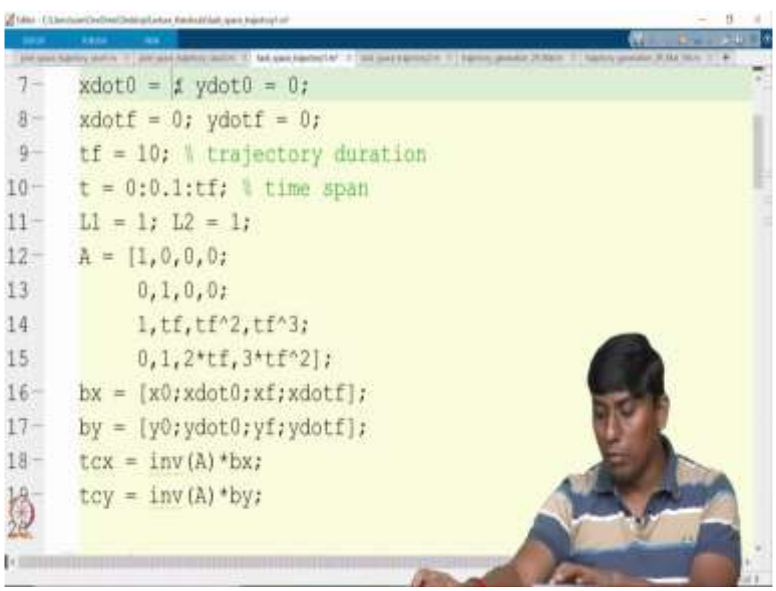

plot(x0,
plot(xf,
set(gca,
xlabel('
ylabel('
axis([-(
grid on
axis squ
pause(0.
hold off
end
figure
subplot(2,2,1)
arting point
nal point
es');
2)-0.1 (L1+L2)+0.
Figure 1

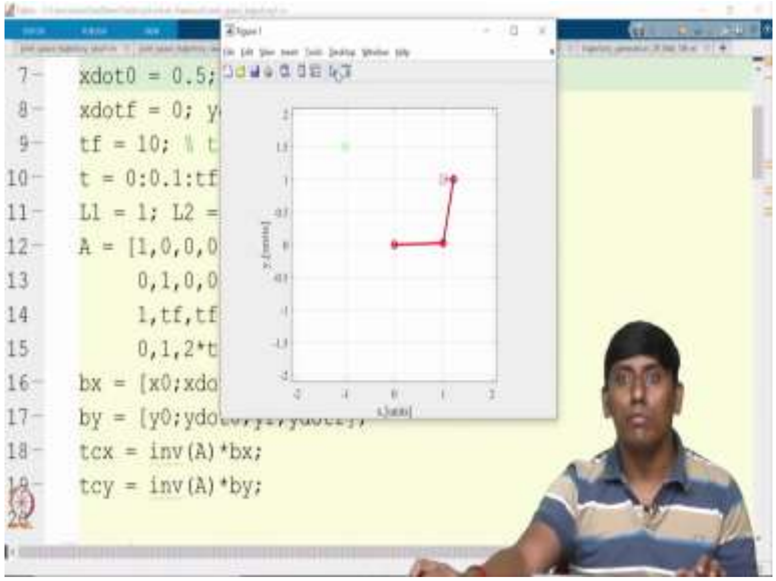


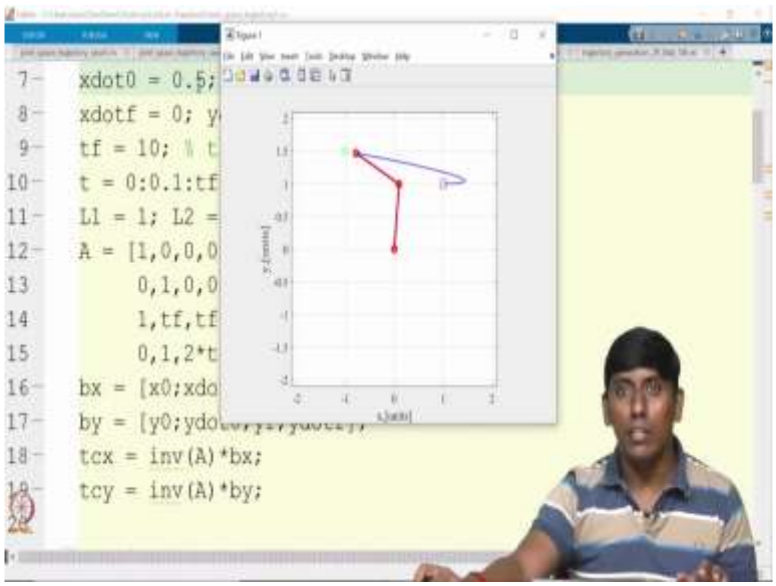
So, I will just make it the time duration is higher. So, now in that case the delay time is actually higher. So, you can see like it starts from this point and it is supposed to go here it is trying to follow a straight line because the initial and final velocity are 0. If the initial and final velocity are nonzero what will happen you can see it by changing that.

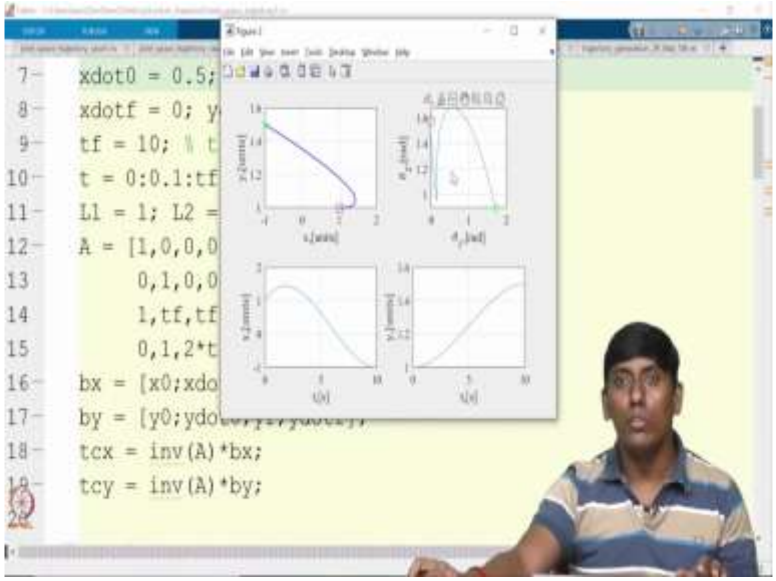
So, now in that case, you can see the joint space you can see like it is not following as a straight-line earlier case it was followed a straight-line earlier case the task space trajectory is followed as curve. But in this case is the straight line. So, the same way x trajectory is smooth y trajectory is again smooth these all we can realize it.

(Refer Slide Time: 18:03)



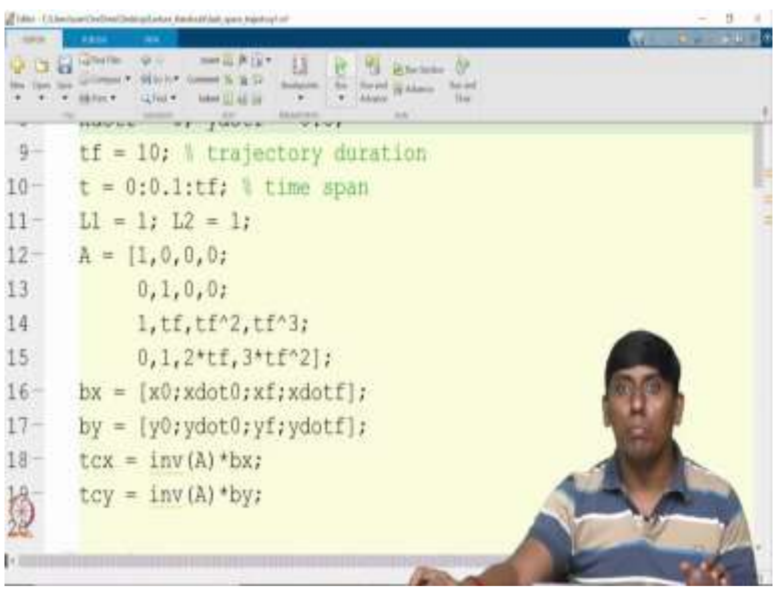

xdot0 = 0.5;
xdotf = 0; y
tf = 10; % t
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*t
bx = [x0;xdo
by = [y0;ydo
tcx = inv(A)*bx;
tcy = inv(A)*by;

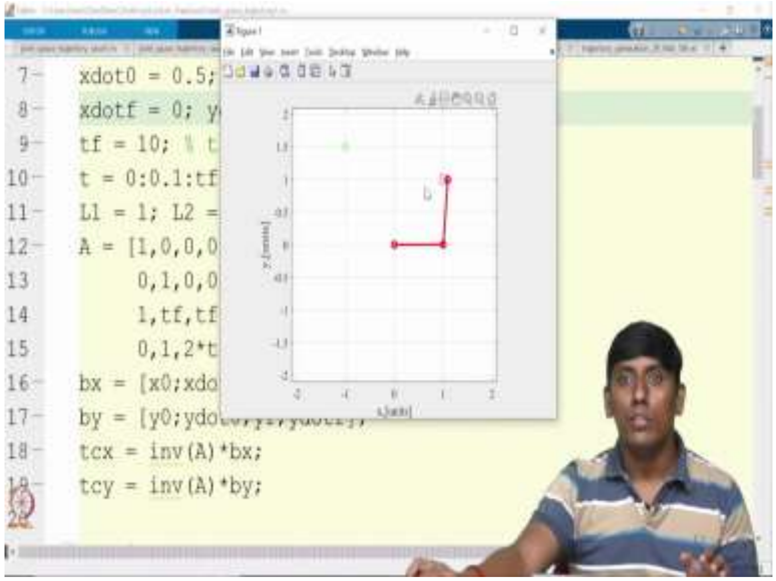
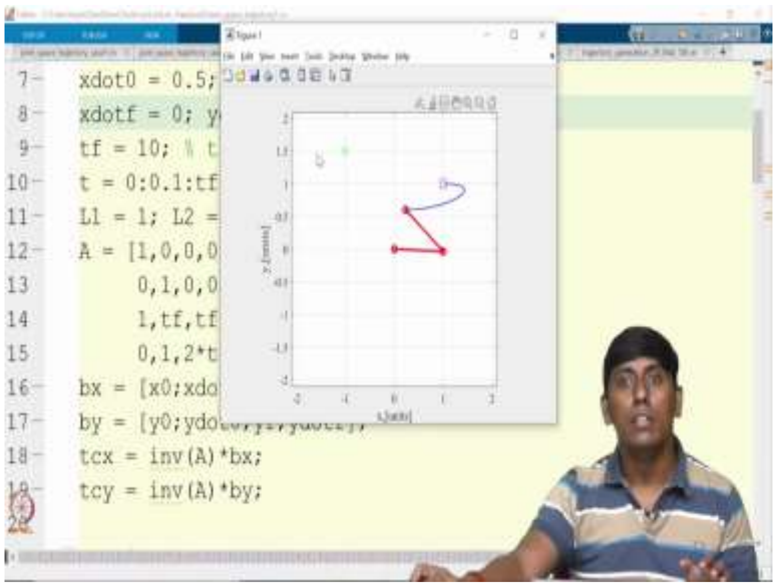

xdot0 = 0.5;
xdotf = 0; y
tf = 10; % t
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*t
bx = [x0;xdo
by = [y0;ydo
tcx = inv(A)*bx;
tcy = inv(A)*by;

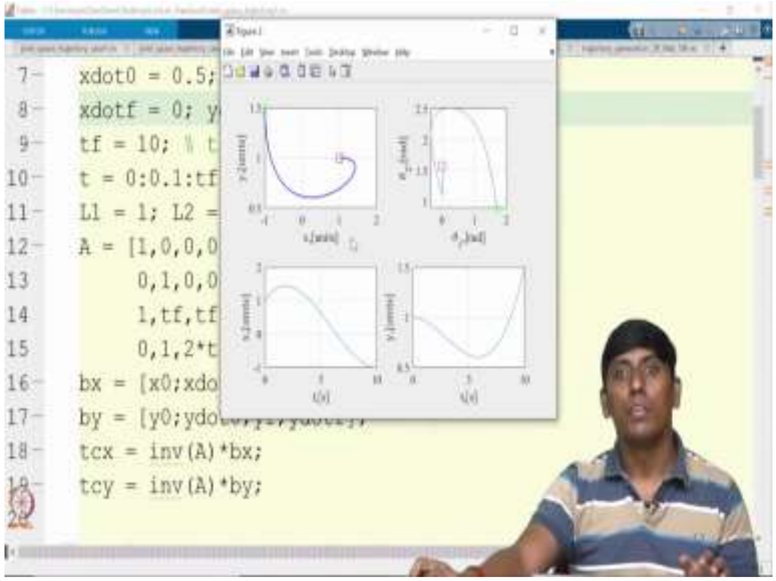


So just to show that how the nonzero initial velocity will involve. So, I am just saying that only. So, the nonzero x is there, which is 0.5 meters per second. So, you can see that it will not be straight line it will be giving a small curve because it is already having some nonzero initial velocity you can see that it is trying to follow. However, it is going to at rest, it will not get any residue there. But here it is starting from nonzero you can see that the profile is inherently modified. So, that is what you can look at it.

(Refer Slide Time: 18:42)

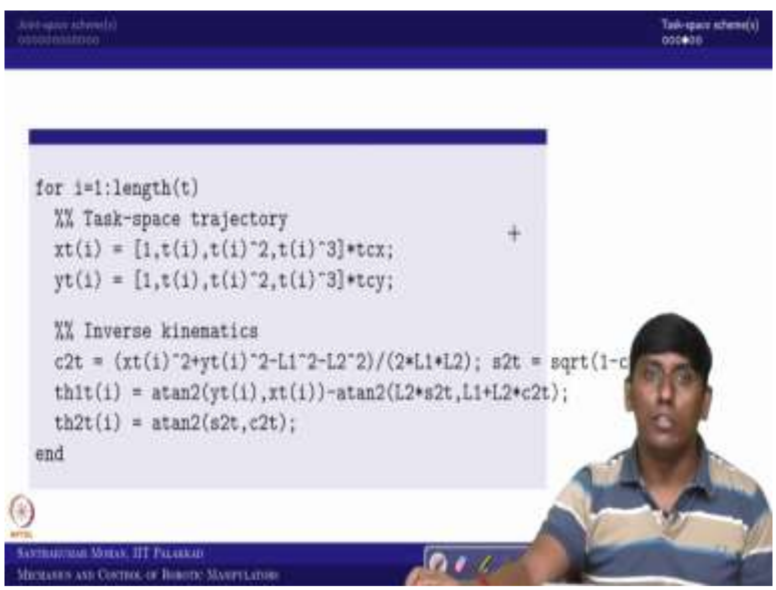


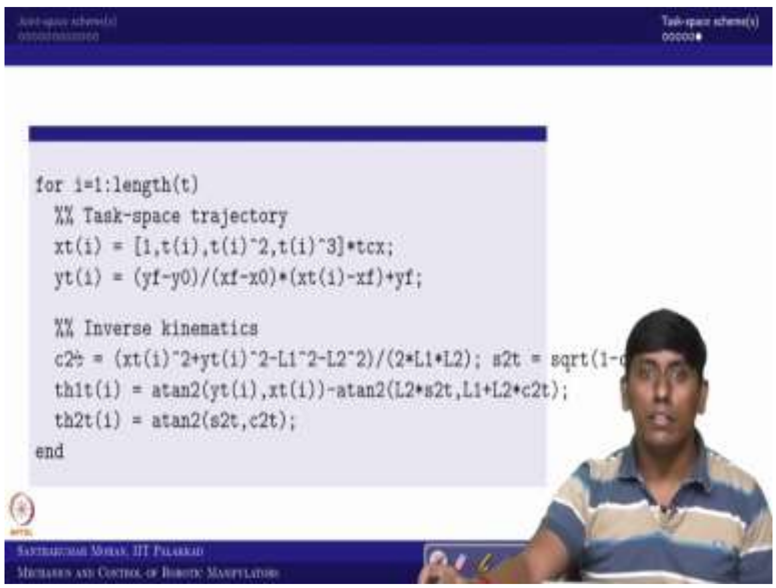

xdot0 = 0.5;
xdotf = 0;
tf = 10;
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*t
bx = [x0;xdo
by = [y0;ydo
tcx = inv(A)*bx;
tcy = inv(A)*by;

xdot0 = 0.5;
xdotf = 0;
tf = 10;
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*t
bx = [x0;xdo
by = [y0;ydo
tcx = inv(A)*bx;
tcy = inv(A)*by;



So even if I give something like the final velocity of this I am just giving randomly. So, the y final velocity is 0.5 meters per second. So, you can see like, it is for the x axis it is there. So that is why it is trying to go and now you can see it is modified because the y final velocity is supposed to be somewhere 0.5 meter per second. So, in the sense vertically up. So that is why it is start from horizontal and the end with vertical that is what it is showing. So, now you are I hope you are getting the clarity. Now, we will move to the second case.

(Refer Slide Time: 19:21)

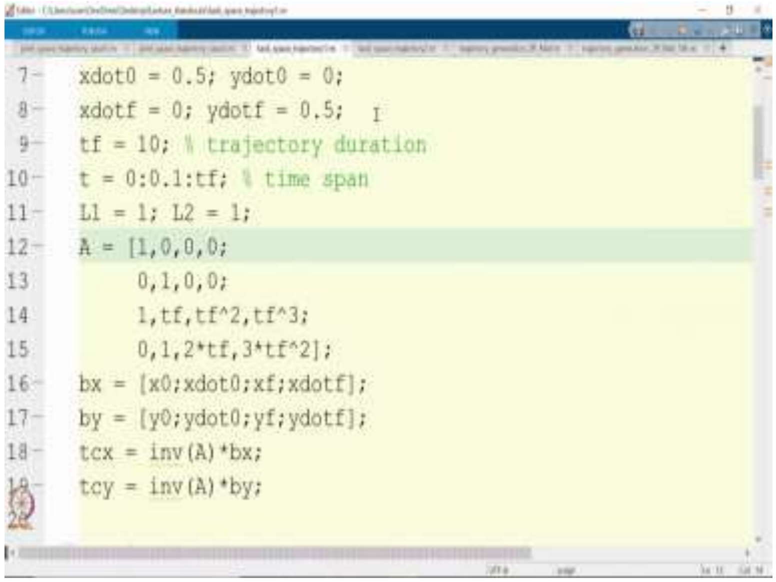


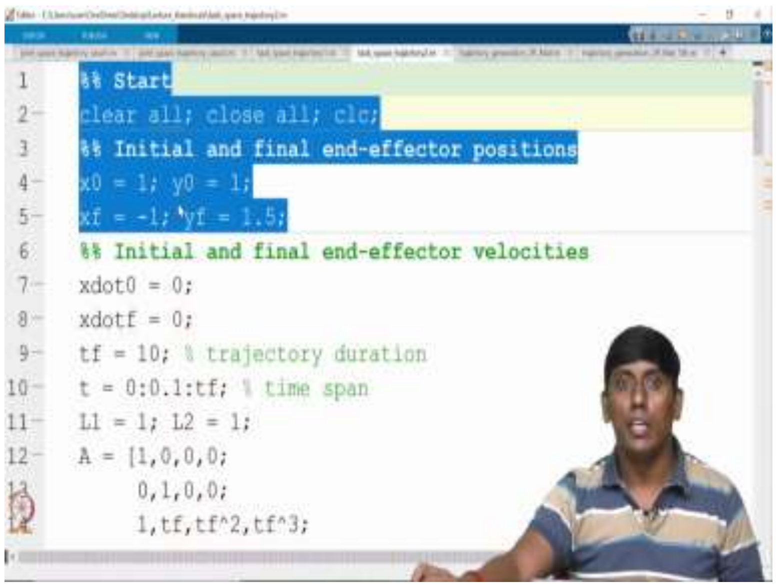


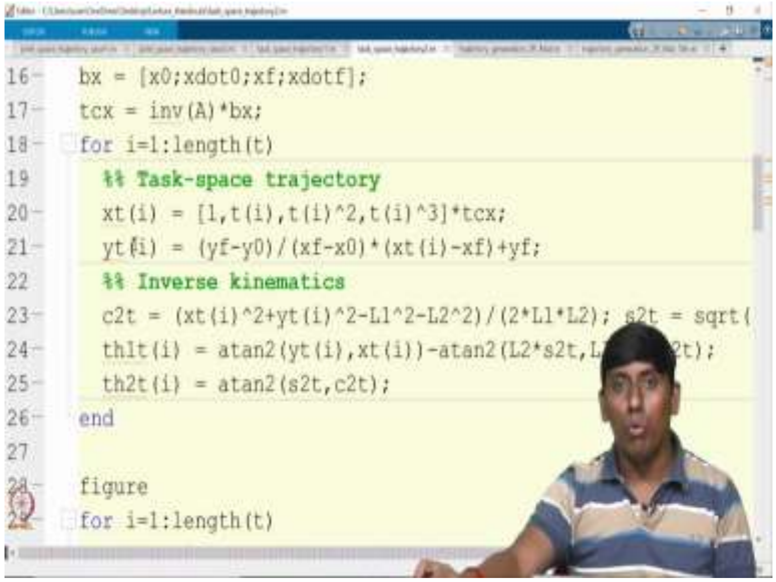
What is the second case? So, where the straight-line interpolation. So, the straight-line interpolation was only 1 modification we can expect. So, what modification? So, instead of this, so, we would be having derived from the x. So, you can make only x so, then we can make it y. So, based on x so, which is what we have done in a straight-line interpolation if you recall.

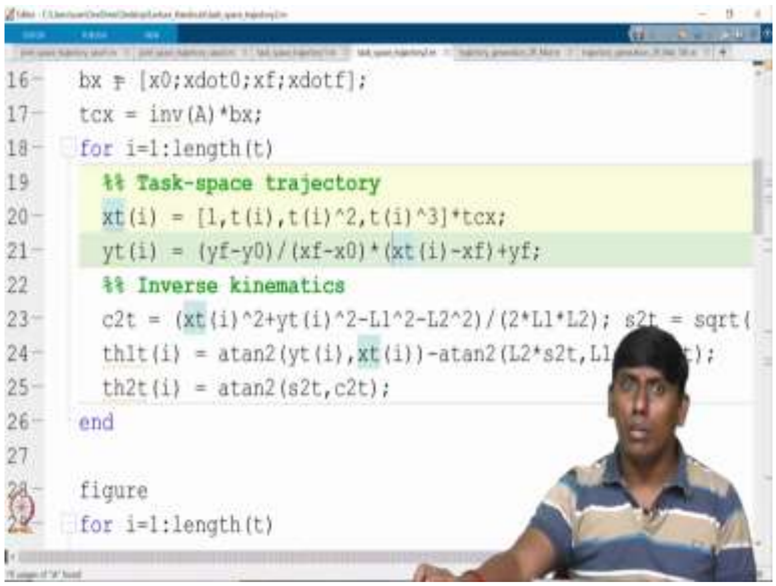
So, if once you know this so, you can see like the x of t is derived based on the x of t the y of t is coming. So, in that sense y dot and all will not come explicitly here. So, now in that sense, this is the inverse kinematics the standard and we have the same plotting function we will be using it.

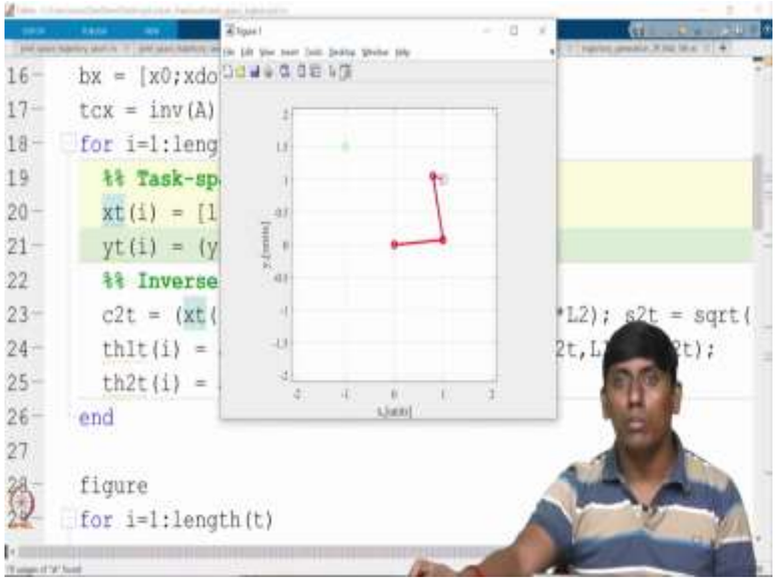
(Refer Slide Time: 20:07)



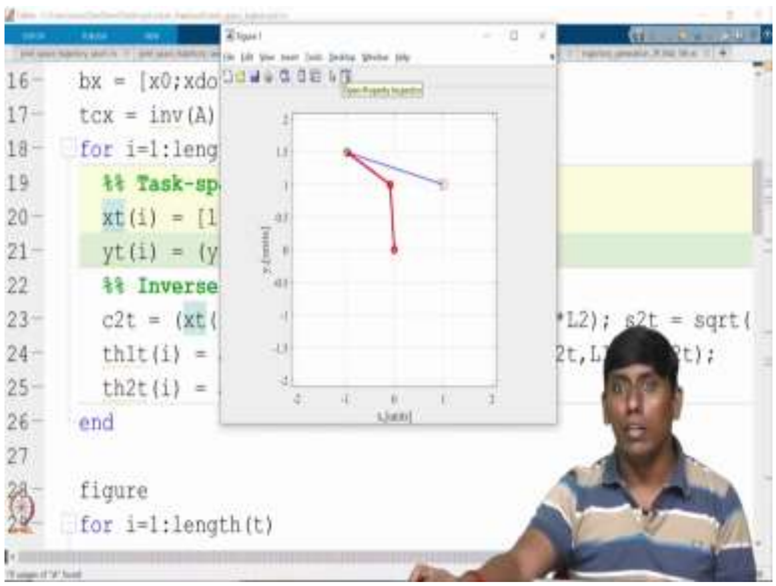



bx = [x0;xdot0;xf;xdotf];
tcx = inv(A)*bx;
for i=1:length(t)
%% Task-space trajectory
xt(i) = [1,t(i),t(i)^2,t(i)^3]*tcx;
yt(i) = (yf-y0)/(xf-x0)*(xt(i)-xf)+yf;
%% Inverse kinematics
c2t = (xt(i)^2+yt(i)^2-L1^2-L2^2)/(2*L1*L2); s2t = sqrt(
th1t(i) = atan2(yt(i),xt(i))-atan2(L2*s2t,L
th2t(i) = atan2(s2t,c2t);
end
figure
for i=1:length(t)

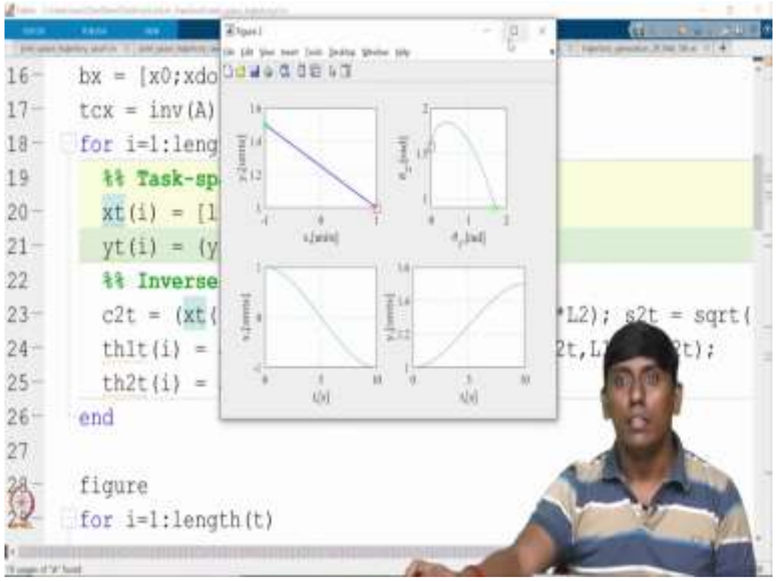

bx = [x0;xdot0;xf;xdotf];
tcx = inv(A)*bx;
for i=1:length(t)
%% Task-space trajectory
xt(i) = [1,t(i),t(i)^2,t(i)^3]*tcx;
yt(i) = (yf-y0)/(xf-x0)*(xt(i)-xf)+yf;
%% Inverse kinematics
c2t = (xt(i)^2+yt(i)^2-L1^2-L2^2)/(2*L1*L2); s2t = sqrt(
th1t(i) = atan2(yt(i),xt(i))-atan2(L2*s2t,L1
th2t(i) = atan2(s2t,c2t);
end
figure
for i=1:length(t)



bx = [x0;xdo
tcx = inv(A)
for i=1:leng
%% Task-sp
xt(i) = [1
yt(i) = (y
%% Inverse
c2t = (xt(
th1t(i) =
th2t(i) =
end
figure
for i=1:length(t)
*L2); s2t = sqrt(
2t,L
t);

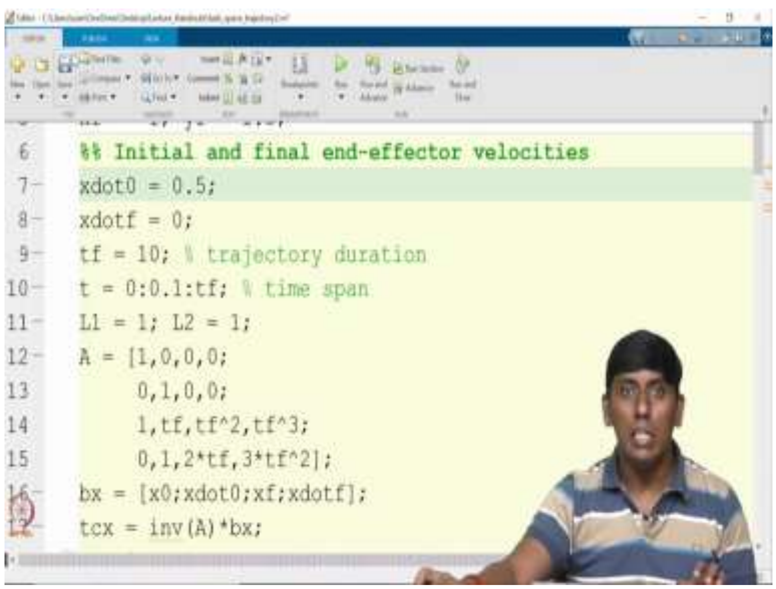


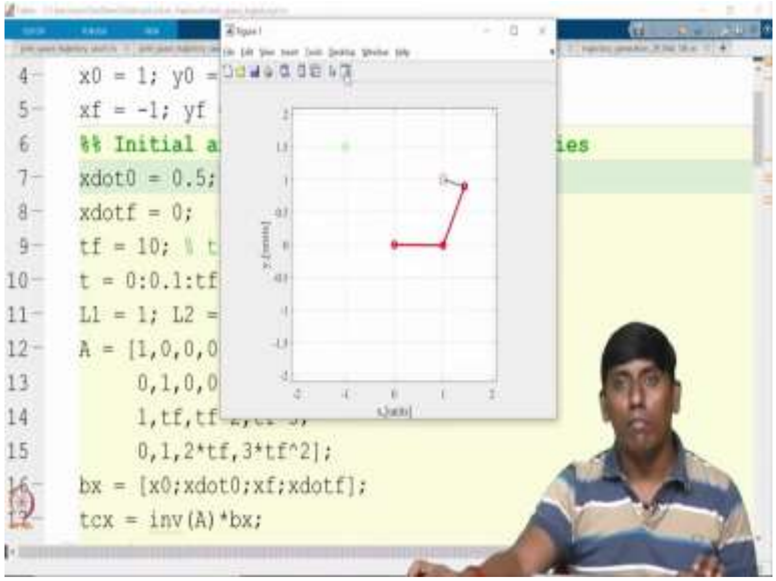
bx = [x0;xdo
tcx = inv(A)
for i=1:leng
%% Task-sp
xt(i) = [1
yt(i) = (y
%% Inverse
c2t = (xt(
th1t(i) =
th2t(i) =
end
figure
for i=1:length(t)
*L2); s2t = sqrt(
2t,L
t);

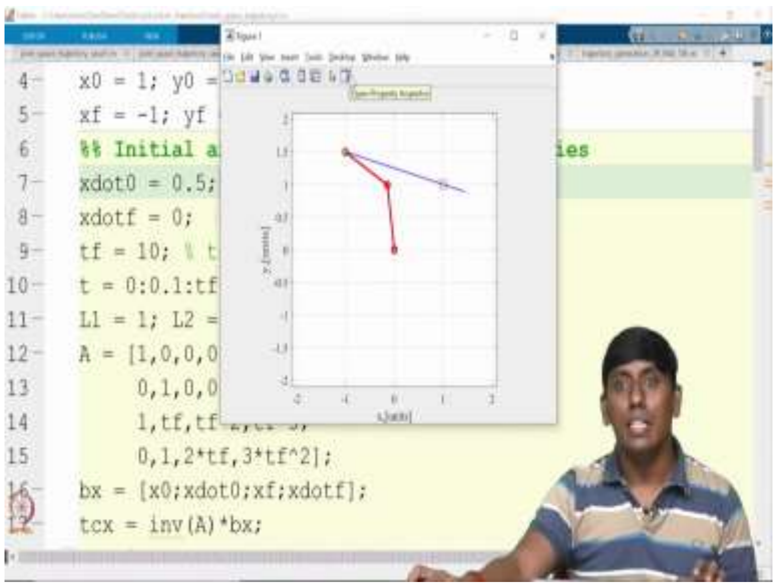


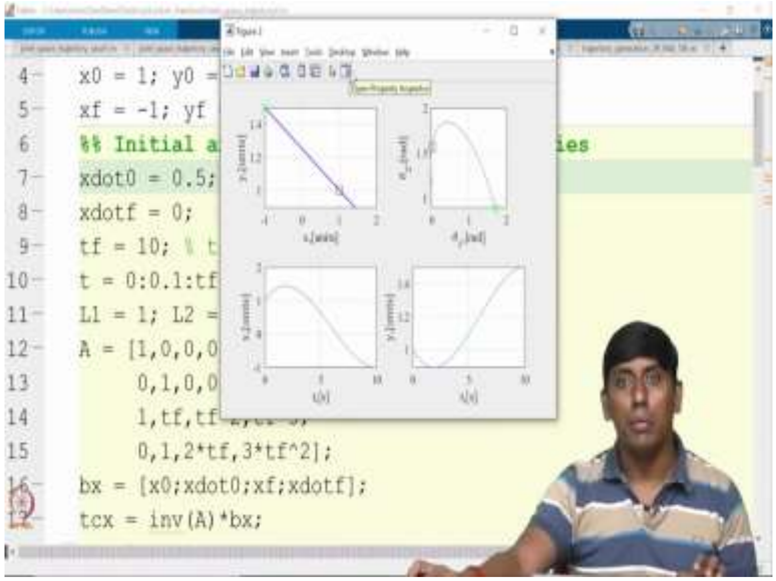
So, the task space trajectory 2 you can look at it. So, this is what the given condition. So, the same condition we have taken and the x dot and y dot like x dot initial and final only we are taking for the trajectory generation. So, then we are deriving the y of t or y. So, based on the x So, that is what you can see explicitly. So, then we are moving to the same thing. So, now, you can look at the profile which is going to be very smooth because the y would be given.

(Refer Slide Time: 20:49)



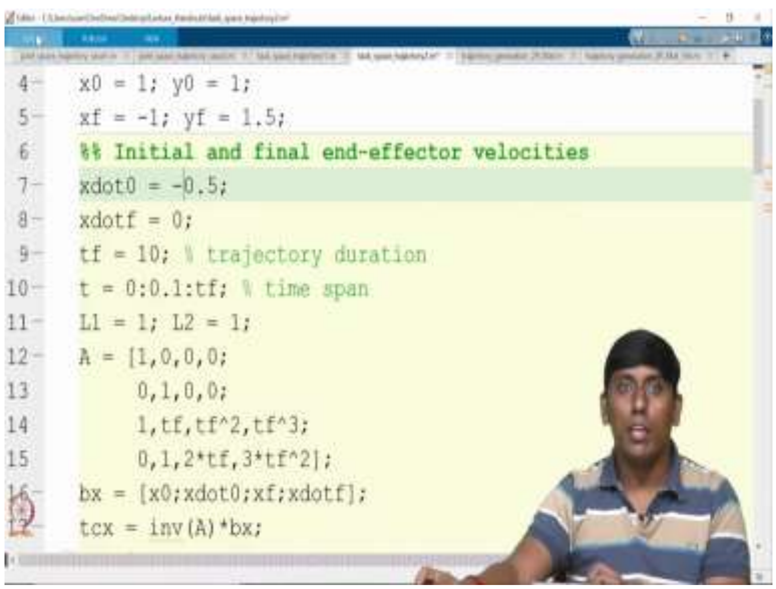

x0 = 1; y0 =
xf = -1; yf
%% Initial a
ies
xdot0 = 0.5;
xdotf = 0;
tf = 10; % t
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*tf,3*tf^2];
bx = [x0;xdot0;xf;xdotf];
tcx = inv(A)*bx;

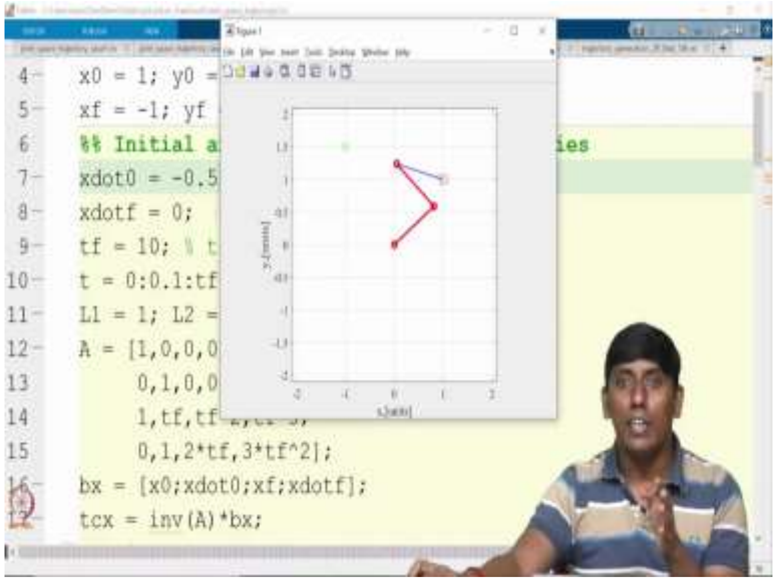

x0 = 1; y0 =
xf = -1; yf
%% Initial a
ies
xdot0 = 0.5;
xdotf = 0;
tf = 10; % t
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*tf,3*tf^2];
bx = [x0;xdot0;xf;xdotf];
tcx = inv(A)*bx;

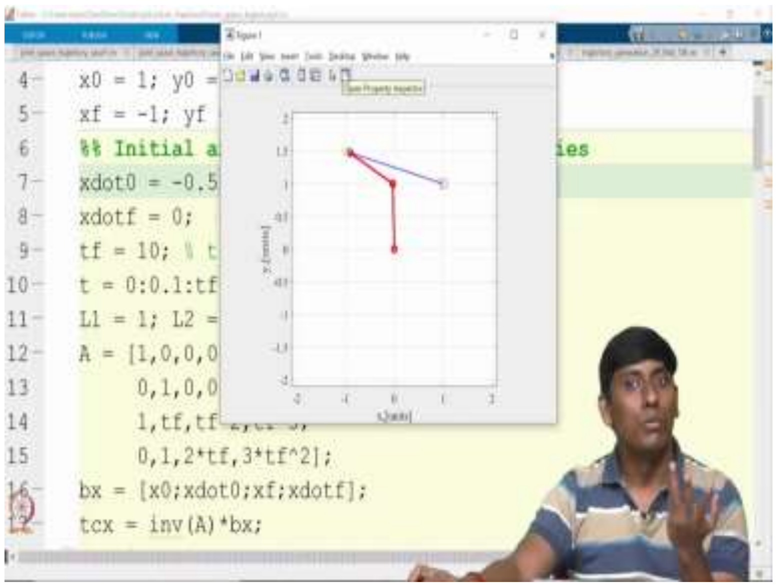


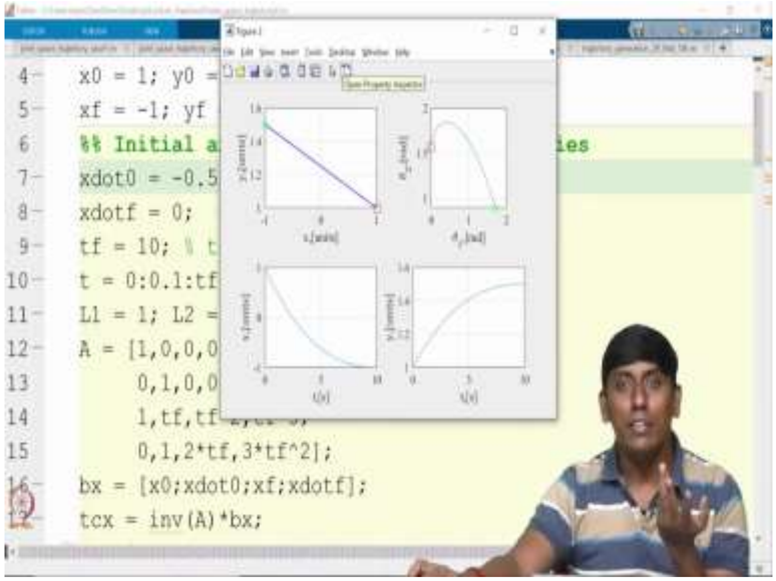
So, now, you can feel it. So, now, even if I give some nonzero initial velocities. So, for example, I am just taking it this is 0.5. So, now, the profile would be you can say goes like that however, you can see the y is not coming. So, that is what you can feel it. So, it is going across the line and goes because you do not have any you can say case. Because we assume that this is followed on the same line.

(Refer Slide Time: 21:19)




x0 = 1; y0 =
xf = -1; yf
%% Initial a
ies
xdot0 = -0.5
xdotf = 0;
tf = 10; % t
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*tf,3*tf^2];
bx = [x0;xdot0;xf;xdotf];
tcx = inv(A)*bx;


x0 = 1; y0 =
xf = -1; yf
%% Initial a
ies
xdot0 = -0.5
xdotf = 0;
tf = 10; % t
t = 0:0.1:tf
L1 = 1; L2 =
A = [1,0,0,0
0,1,0,0
1,tf,tf
0,1,2*tf,3*tf^2];
bx = [x0;xdot0;xf;xdotf];
tcx = inv(A)*bx;



For example, if I take minus 5. So, you can feel it that would be come on the same line and go. So, there is no change in your velocity, only thing this profile is interchanged that is what you would have felt. So, this is what the; you call straight line interpolation even the straight-line interpolation you want to do it even little more, then you are to bring the; you can say x dot and y dot term these terms I do not want to bring it not to make a much much you can say complex.

This is sufficient for you can say run the code. So, even you want to more smoother. So, what we can do instead of, you can say cubic polynomial for the x even we can use fifth order polynomial or cycloidal. In that case the yt consequently will come smooth enough. So, that way we can do it. So, I hope you are clear to this. So, with that, we are ending this particular lecture.

The next lecture I will be showing how to generate a workspace with the MATLAB in that we will see like how the task space trajectory will be having some you can see difficulty and all. So, in the sense we will take a simple MATLAB code, single MATLAB code, including workspace and you can say the task space trajectory and you can say joint space trajectory generation and we can see the comparison in a simple single short video. So, with that, I am saying thank you here and see you in next lecture until then, bye, take care.