Mechanics and Control of Robotic Manipulators
Professor. Santhakumar Mohan
Indian Institute of Technology Palakkad
Lecture 18
Inverse Kinematic Solution Based on Numerical Methods using MATLAB

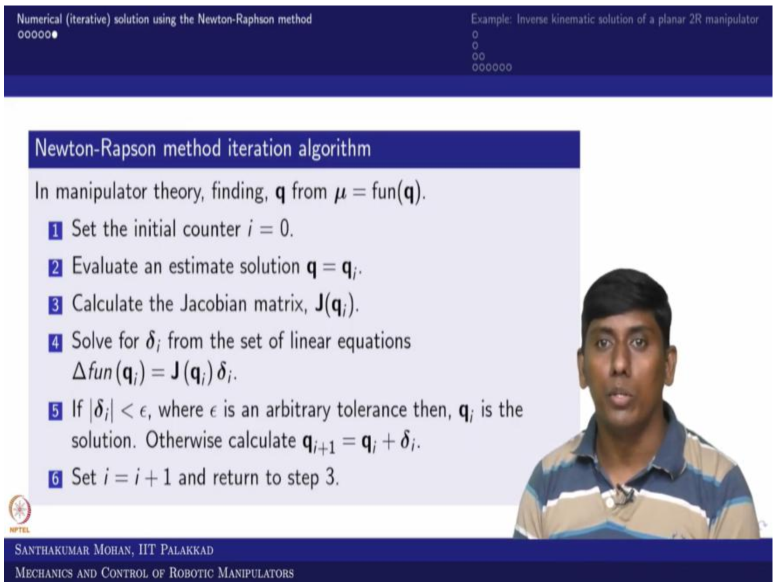
Hi. Welcome back to Mechanics and Control of Robotic Manipulator. In the last lecture we have seen how the MATLAB can be used as an efficient tool for finding forward kinematics solution with the help of the arm matrix which we obtained through the general Denavit-Hartenberg representation.

The same lecture I was like given a small idea. So, with the help of solve command how to find the inverse kinematics solution, if you have forward kinematics solutions are available. In the sense if you mu vector, so how we can actually like, find the inverse kinematic solution using solve. But the solve is not the right option all the time.

So, we use actually like a numerical method, which is faster. So, that would be beneficial when you do for online programming and other things. For example, you have a robot in real time. So, obviously the numerical method would be little more beneficial than the closed form solution, you can say computational perspective.

So, in that sense this particular lecture is going to talk about how to use Newton-Raphson method for finding inverse kinematics solution of a serial manipulator. So, here again the same example which we have taken in the original lecture. The same example 2R planner serial manipulator we have taken.

(Refer Slide Time: 01:30)


Numerical (iterative) solution using the Newton-Raphson method

Example: Inverse kinematic solution of a planar 2R manipulator

Newton-Rapson method iteration algorithm

In manipulator theory, finding, $\mathbf{q}$ from $\mu = \text{fun}(\mathbf{q})$.

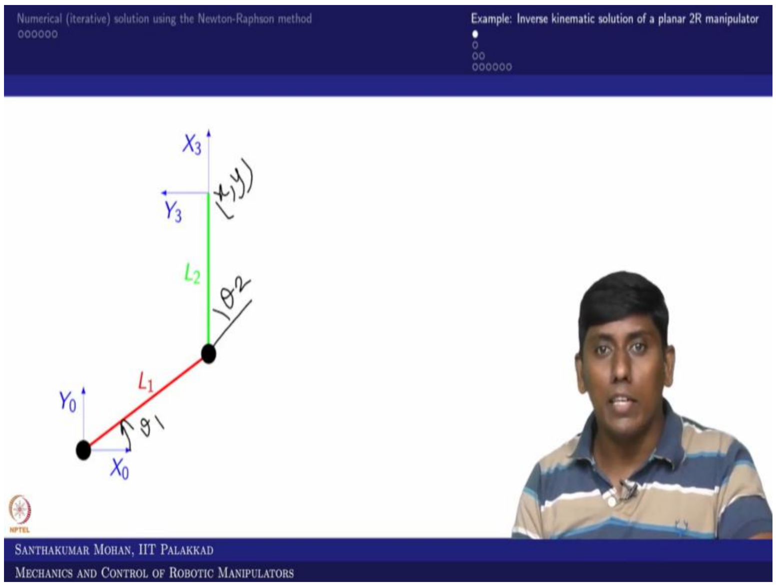
1. Set the initial counter $i = 0$.
2. Evaluate an estimate solution $\mathbf{q} = \mathbf{q}_i$.
3. Calculate the Jacobian matrix, $\mathbf{J}(\mathbf{q}_i)$.
4. Solve for δ_i from the set of linear equations $\Delta fun(\mathbf{q}_i) = \mathbf{J}(\mathbf{q}_i)\,\delta_i$.
5. If $|\delta_i| < \epsilon$, where ϵ is an arbitrary tolerance then, $\mathbf{q}_i$ is the solution. Otherwise calculate $\mathbf{q}_{i+1} = \mathbf{q}_i + \delta_i$.
6. Set $i = i + 1$ and return to step 3.

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

And we are like trying to derive the code with the help of the algorithm which we have come across. So, only change what we made is instead of this delta i we are going to take the relative tolerance or arbitrary tolerance, we are going to take instead of delta i, we are going to take delta mu.

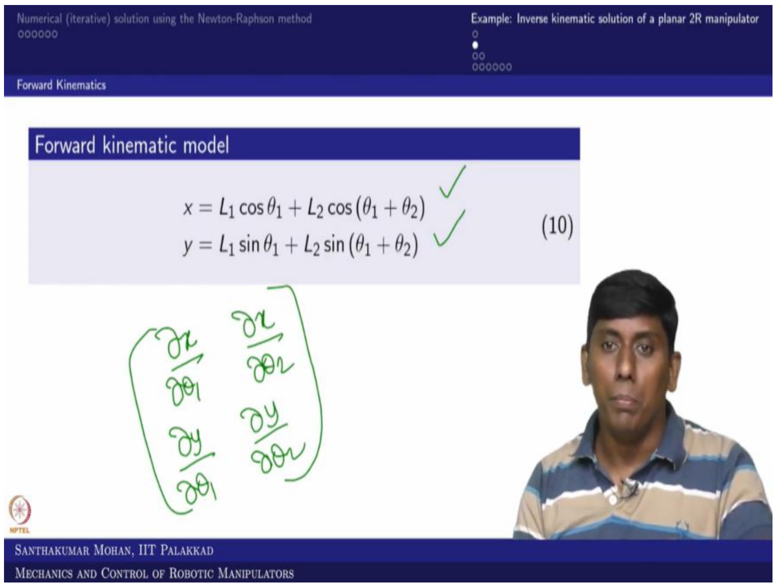
That is the only thing because delta i would not be available to us. So, we would be seeing that what other error we can restrict. We can restrict the delta mu, that is the only restriction we are making it other than this, you can say algorithm which we have written. In addition to that, what we supposed to know, we should have the J of q i and q initial guess and mu. Mu is like function of q. These all need to be known.

(Refer Slide Time: 02:15)


Numerical (iterative) solution using the Newton-Raphson method
Example: Inverse kinematic solution of a planar 2R manipulator
X_3
Y_3
(x, y)
L_2
θ_2
L_1
Y_0
θ_1
X_0
Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

So, in that sense, what we can see if we write the MATLAB code for this 2R serial manipulator, if the x and y are known, can we find theta 1 and theta 2?

(Refer Slide Time: 02:22)



Numerical (iterative) solution using the Newton-Raphson method

Example: Inverse kinematic solution of a planar 2R manipulator

Forward Kinematics

Forward kinematic model

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \quad ✓$$
$$y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \quad ✓ \qquad (10)$$

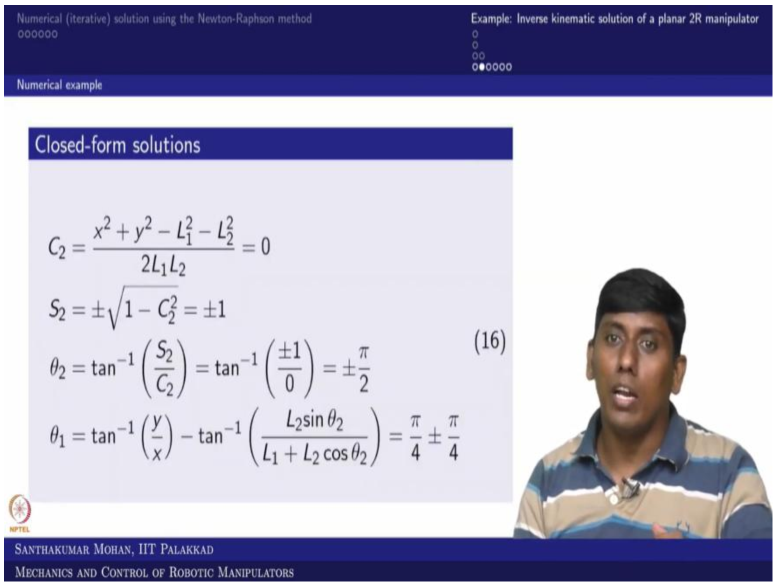
$$\begin{pmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{pmatrix}$$

Santhakumar Mohan, IIT Palakkad

Mechanics and Control of Robotic Manipulators

So, for that, you know this equation so where x and y would be given as in the form of function of q, here theta 1 and theta 2. So, if this equation is known, then we can find the Jacobian matrix in the order of, or you can say in the way of partial derivative which I have written in the board.

(Refer Slide Time: 02:46)



Numerical (iterative) solution using the Newton-Raphson method

Example: Inverse kinematic solution of a planar 2R manipulator

Numerical example

Closed-form solutions

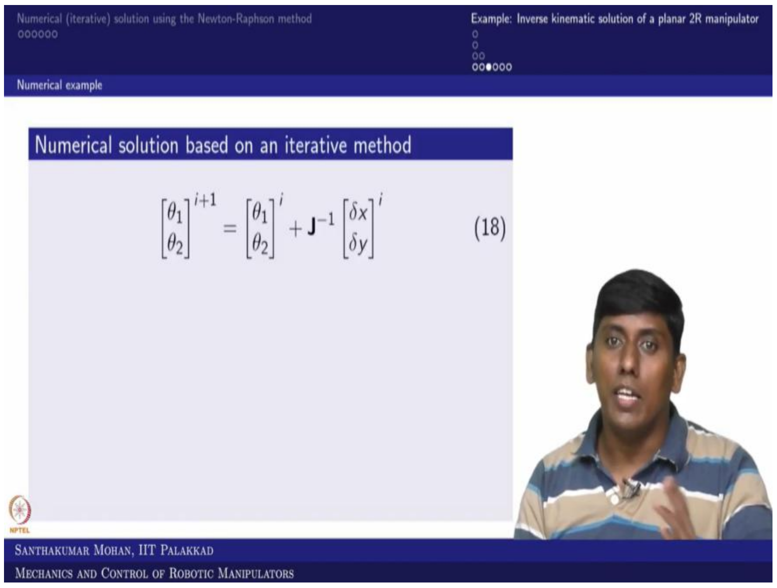
$$C_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2} = 0$$
$$S_2 = \pm\sqrt{1 - C_2^2} = \pm 1$$
$$\theta_2 = \tan^{-1}\left(\frac{S_2}{C_2}\right) = \tan^{-1}\left(\frac{\pm 1}{0}\right) = \pm\frac{\pi}{2} \qquad (16)$$
$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{L_2 \sin\theta_2}{L_1 + L_2\cos\theta_2}\right) = \frac{\pi}{4} \pm \frac{\pi}{4}$$

Santhakumar Mohan, IIT Palakkad

Mechanics and Control of Robotic Manipulators

So now, if we see that these partial derivatives are available, can we like cross check the closed form solution whatever is giving the result?

(Refer Slide Time: 02:50)


Numerical (iterative) solution using the Newton-Raphson method

Example: Inverse kinematic solution of a planar 2R manipulator

Numerical example

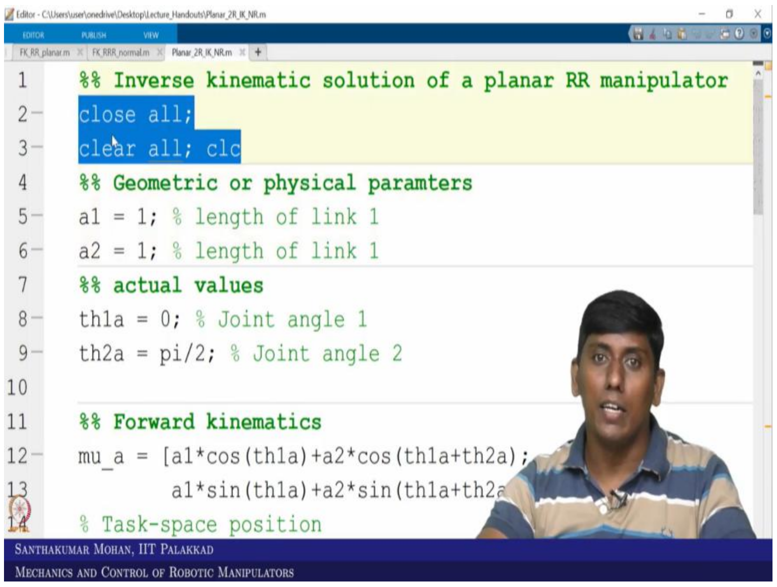
Numerical solution based on an iterative method

$$\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}^{i+1} = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}^{i} + \mathbf{J}^{-1}\begin{bmatrix}\delta x\\ \delta y\end{bmatrix}^{i} \qquad (18)$$

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

The same result can be obtained with the iterative method.

(Refer Slide Time: 02:54)


Editor - C:\Users\user\onedrive\Desktop\Lecture_Handouts\Planar_2R_IK_NR.m

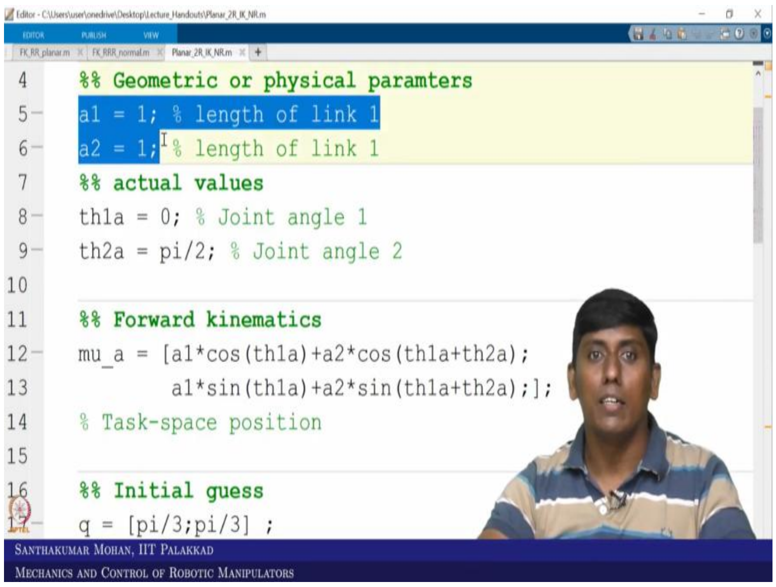
```
%% Inverse kinematic solution of a planar RR manipulator
close all;
clear all; clc
%% Geometric or physical paramters
a1 = 1; % length of link 1
a2 = 1; % length of link 1
%% actual values
th1a = 0; % Joint angle 1
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
         a1*sin(th1a)+a2*sin(th1a+th2a
% Task-space position
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, for that we have taken the code which we have already written. So, you can see that the initial command all are same. So, this is the inverse kinematics solution of a planner RR manipulator with the help of iterative thing.

So, only issue is like I make the editor window closed. So, this window is like I closed it just for my benefit, so that the display would be very much visible for you. But whenever you run, so you can go to editor and you can like run and run and advance, but I am going to use the shortcut called F5. So, now this is like the standard.

(Refer Slide Time: 03:34)

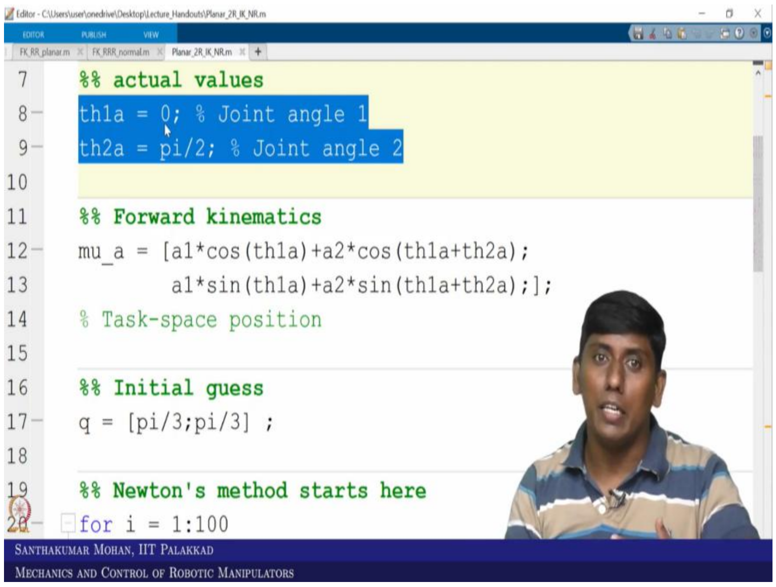


```
%% Geometric or physical paramters
a1 = 1; % length of link 1
a2 = 1; % length of link 1
%% actual values
th1a = 0; % Joint angle 1
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;pi/3] ;
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

So, now we are like seen in the lecture. So, where L 1 and L 2 are actually like 1 unit each. So, the same thing we are actually like taking it.

(Refer Slide Time: 03:41)

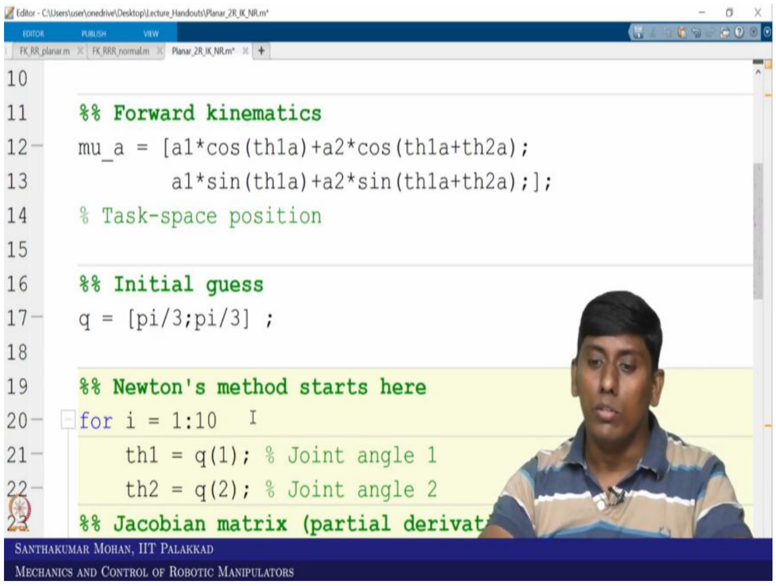


```
%% actual values
th1a = 0; % Joint angle 1
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;pi/3] ;

%% Newton's method starts here
for i = 1:100
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

So again, I am seeing one given point, so in the sense this in actual we do not know, but for finding the mu value, I am like assuming that this is the actual angle, in the sense so I am like making theta 1 is 0 and theta 2 is 90 what will be the mu. That is, I am assuming as a given mu value. So, in the sense, this is not really there for us. So, for us, this is attainable in the sense so instead of this relation, the mu a would be straightaway given. So, in this case it is 1, 1.

(Refer Slide Time: 04:14)



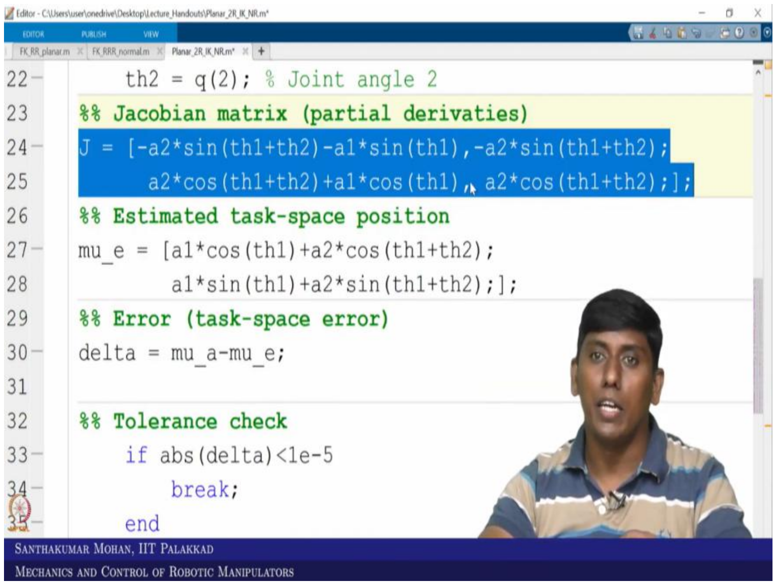
```
%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;pi/3] ;

%% Newton's method starts here
for i = 1:10
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivat
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

So, then what we are trying to do as per the algorithm, we have to start with a guess. So, I am taking a guess as, so pi by 3 each. So, after that I am putting the iterative count as it can go up to 100 counts, but it may not required, but I am saying that the maximum count can go to 100.

So, for your benefit, I am restricting to 10. It is nothing going to change. So, then what I am saying that whatever I am like taking an initial guess that would be equal to theta 1 of, you can say the first value would be theta 1, second value would be theta 2.

(Refer Slide Time: 04:48)

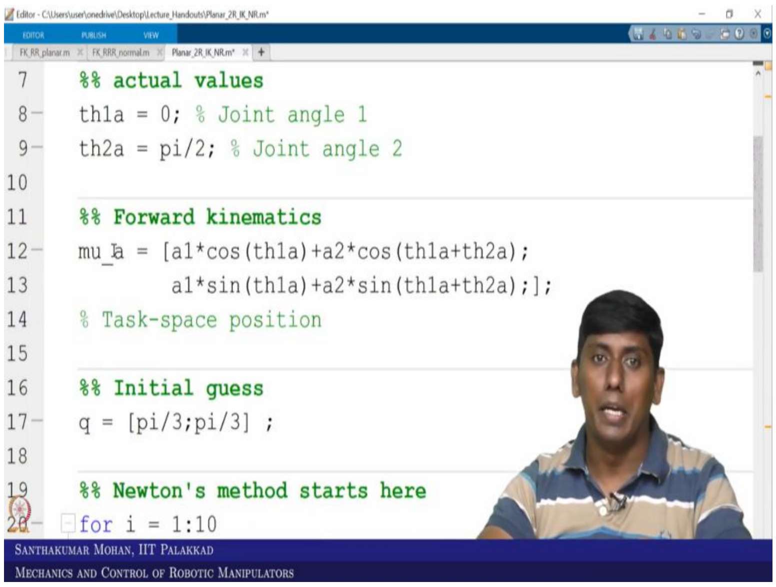


```
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivaties)
J = [-a2*sin(th1+th2)-a1*sin(th1),-a2*sin(th1+th2);
      a2*cos(th1+th2)+a1*cos(th1), a2*cos(th1+th2);];
%% Estimated task-space position
mu_e = [a1*cos(th1)+a2*cos(th1+th2);
        a1*sin(th1)+a2*sin(th1+th2);];
%% Error (task-space error)
delta = mu_a-mu_e;

%% Tolerance check
    if abs(delta)<1e-5
        break;
    end
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

So, then I am like derived already a partial derivative as what you call a Jacobian matrix. So, this Jacobian matrix I have derived. So, after that I know the forward kinematic relation, so which will give the mu estimated. So, the mu underscored e is actually mu estimated.

(Refer Slide Time: 05:09)

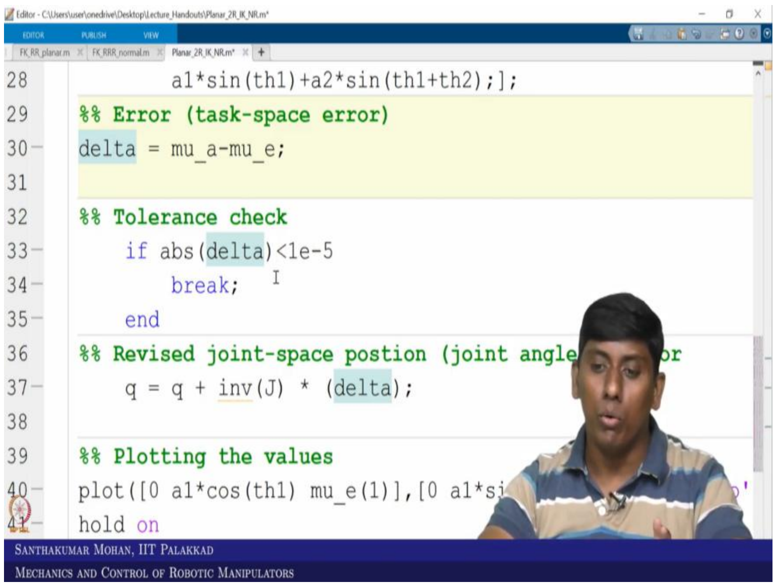


```
%% actual values
th1a = 0; % Joint angle 1
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;pi/3] ;

%% Newton's method starts here
for i = 1:10
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, the mu actual, mu underscored a. So now the mu underscored a minus mu underscored e is your delta mu.

(Refer Slide Time: 05:16)

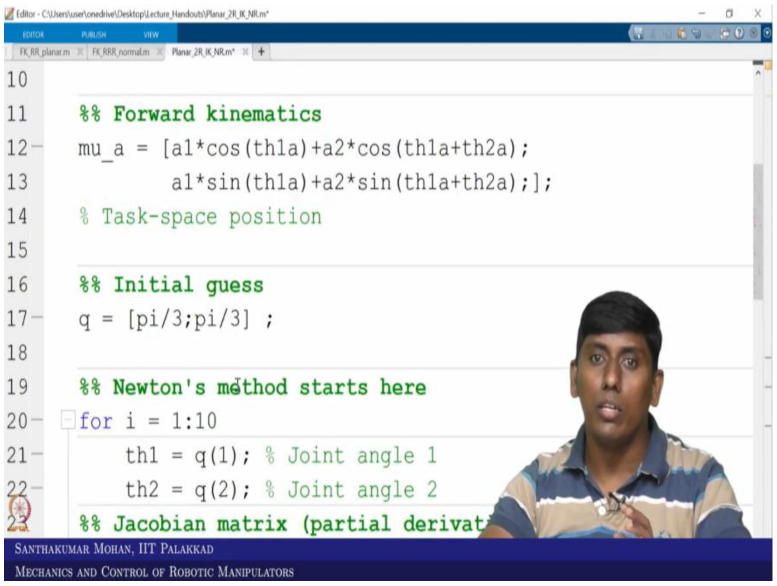


```
        a1*sin(th1)+a2*sin(th1+th2);];
%% Error (task-space error)
delta = mu_a-mu_e;

%% Tolerance check
    if abs(delta)<1e-5
        break;
    end
%% Revised joint-space postion (joint angle        or
    q = q + inv(J) * (delta);

%% Plotting the values
plot([0 a1*cos(th1) mu_e(1)],[0 a1*si
hold on
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

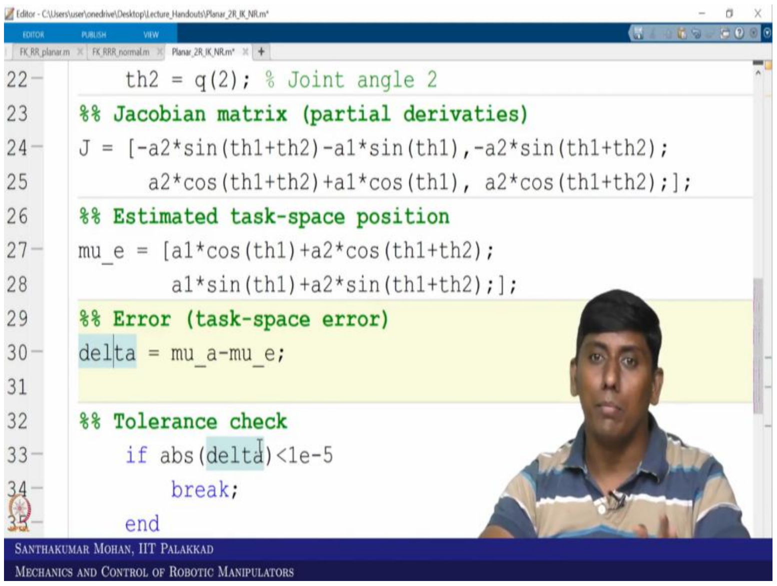
So, that is what I have written as delta. So, the delta is actual minus estimated. So, now I am putting a tolerance check which is 10 power minus 5. So, the absolute value is less than or you can see, are equal to 10 power 5. So, 10 power minus 5. So, then you break the algorithm. So, break the code in the sense stop and give the final result, otherwise you still keep on going.

(Refer Slide Time: 05:44)



```
%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;pi/3] ;

%% Newton's method starts here
for i = 1:10
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivat
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

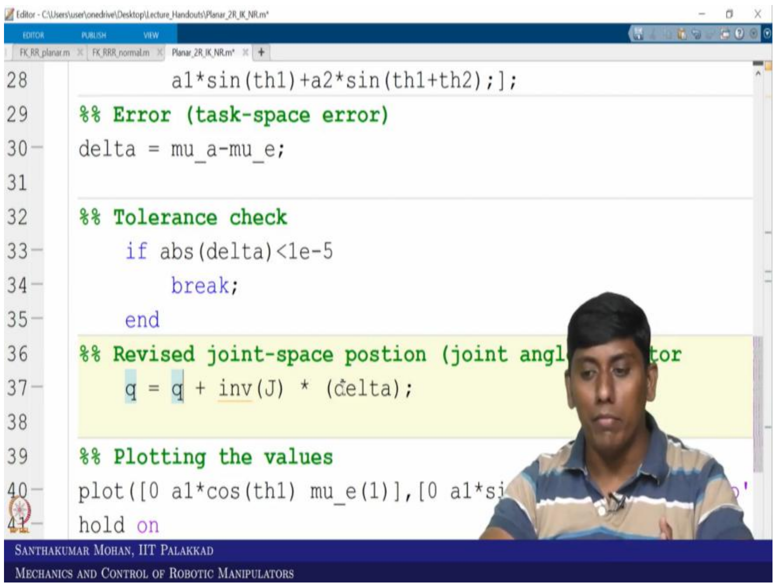


```
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivaties)
J = [-a2*sin(th1+th2)-a1*sin(th1),-a2*sin(th1+th2);
      a2*cos(th1+th2)+a1*cos(th1), a2*cos(th1+th2);];
%% Estimated task-space position
mu_e = [a1*cos(th1)+a2*cos(th1+th2);
        a1*sin(th1)+a2*sin(th1+th2);];
%% Error (task-space error)
delta = mu_a-mu_e;

%% Tolerance check
    if abs(delta)<1e-5
        break;
    end
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

Again, we have restricted the total iterative count as 10, in the sense, if this is not matching, this will run up to 10 and say this is the final result, it is not matching. If this is the case, it will give the final match.

(Refer Slide Time: 05:56)

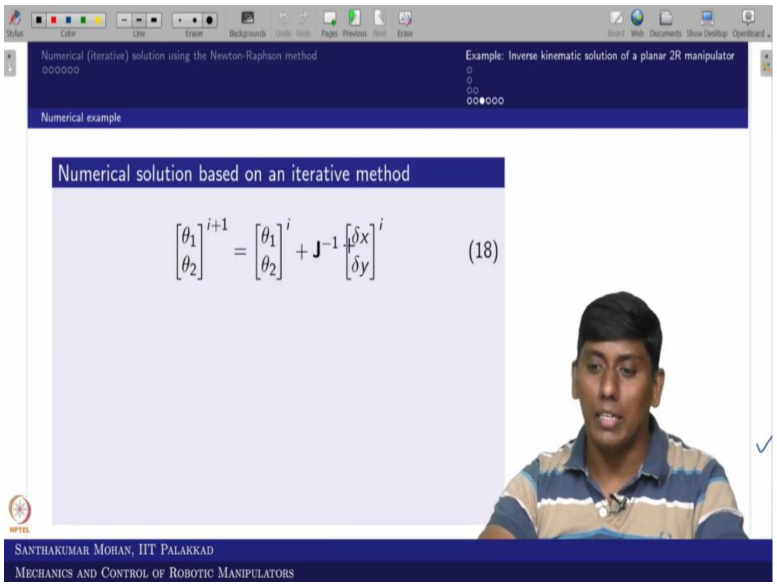


```
           a1*sin(th1)+a2*sin(th1+th2);];
%% Error (task-space error)
delta = mu_a-mu_e;

%% Tolerance check
    if abs(delta)<1e-5
        break;
    end
%% Revised joint-space postion (joint angl        tor
    q = q + inv(J) * (delta);

%% Plotting the values
plot([0 a1*cos(th1) mu_e(1)],[0 a1*si
hold on
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So now what we are doing it? So, the q i plus 1 is actually q of i plus inverse of J multiplied with the delta.

(Refer Slide Time: 06:10)



Numerical (iterative) solution using the Newton-Raphson method

Example: Inverse kinematic solution of a planar 2R manipulator

Numerical example

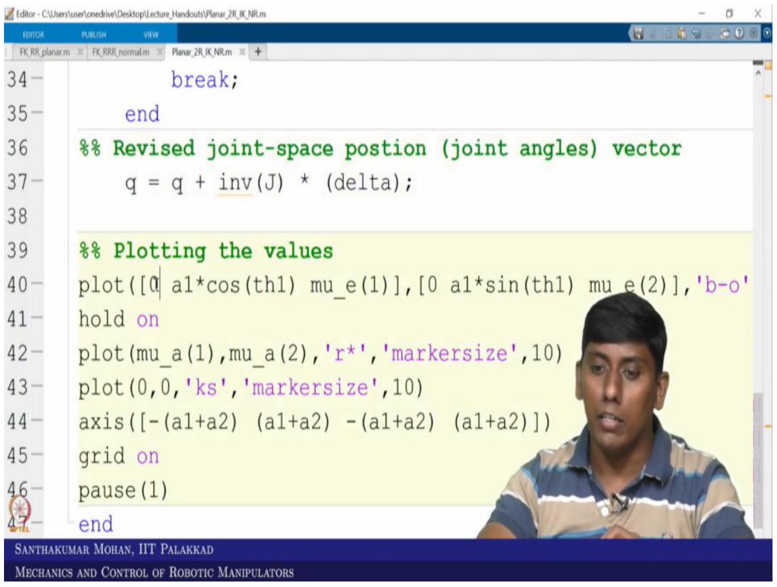
Numerical solution based on an iterative method

$$\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}^{i+1} = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}^{i} + \mathbf{J}^{-1}\begin{bmatrix}\delta x\\ \delta y\end{bmatrix}^{i} \quad (18)$$

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

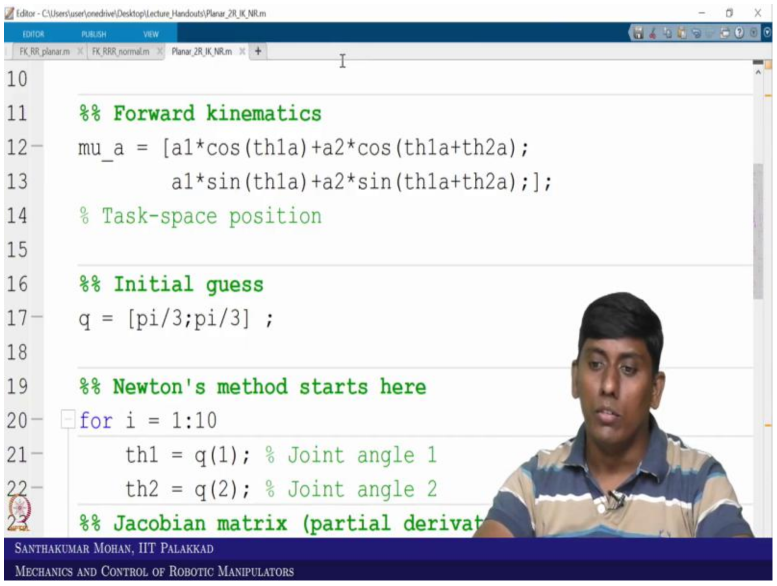
So, you can recheck this is the code. So, you have written it. So, this is the same thing. So, instead of i plus 1 and i, I have denoted as q itself, so maybe it may be a little confusing for you, but I do not want to keep the count as iterative. In the sense, x would be giving a multiple you can say array, so that I do not want. So, q keeps on getting changed.

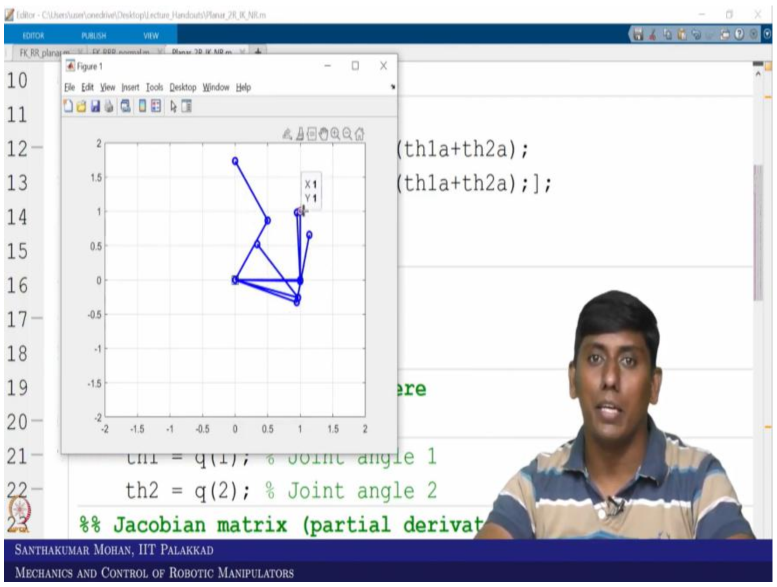
(Refer Slide Time: 06:34)


```
            break;
        end
%% Revised joint-space postion (joint angles) vector
    q = q + inv(J) * (delta);

%% Plotting the values
plot([0 a1*cos(th1) mu_e(1)],[0 a1*sin(th1) mu_e(2)],'b-o'
hold on
plot(mu_a(1),mu_a(2),'r*','markersize',10)
plot(0,0,'ks','markersize',10)
axis([-(a1+a2) (a1+a2) -(a1+a2) (a1+a2)])
grid on
pause(1)
end
```
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, once this is known I want to plot, so for that I am just drawing a 2R serial manipulator, so which is I take origin, then the first joint position in the sense that is the origin second joint position and the end-effector. This is x 0, x 1 and x 2, in your case, it is supposed to be x 0 and x 1 as same, x 2 and x 3. So, then I am showing this is the given value and this is origin, how my count keeps changing that it would be showing it.

(Refer Slide Time: 07:11)


```

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;pi/3] ;

%% Newton's method starts here
for i = 1:10
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivat
```
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

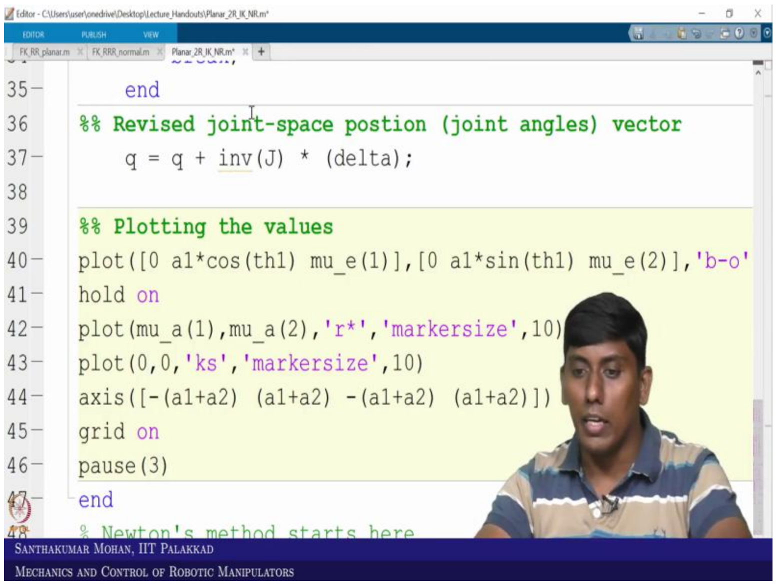


```
(th1a+th2a);
(th1a+th2a);];

ere

th1 = q(1); % Joint angle 1
th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivat
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, what we have taken? We have taken a random guess, which is pi by 3 each. So, now if I run through this, so I will see that output you can see that, so this is the initial guess. It is keep going, this is the final outcome you can see it, it is coming. So, it gone something, and it reach.

(Refer Slide Time: 07:29)



```
    end
%% Revised joint-space postion (joint angles) vector
    q = q + inv(J) * (delta);

%% Plotting the values
plot([0 a1*cos(th1) mu_e(1)],[0 a1*sin(th1) mu_e(2)],'b-o'
hold on
plot(mu_a(1),mu_a(2),'r*','markersize',10)
plot(0,0,'ks','markersize',10)
axis([-(a1+a2) (a1+a2) -(a1+a2) (a1+a2)])
grid on
pause(3)
end
% Newton's method starts here
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

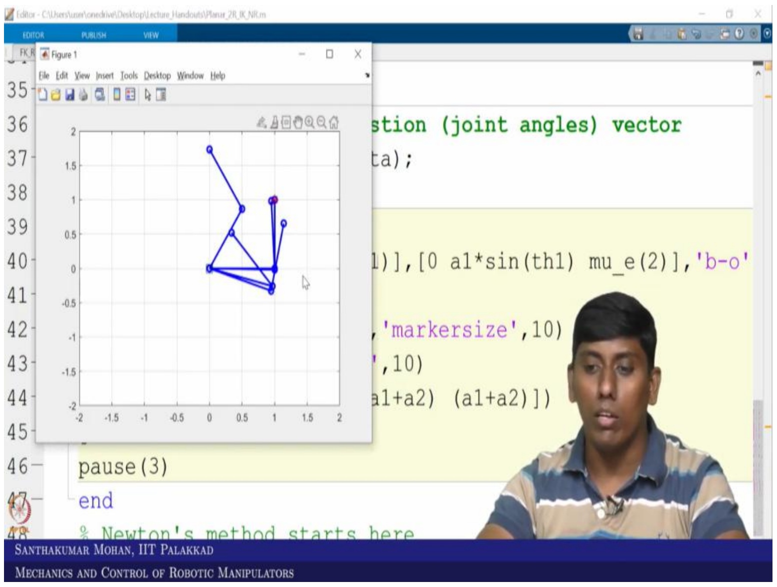
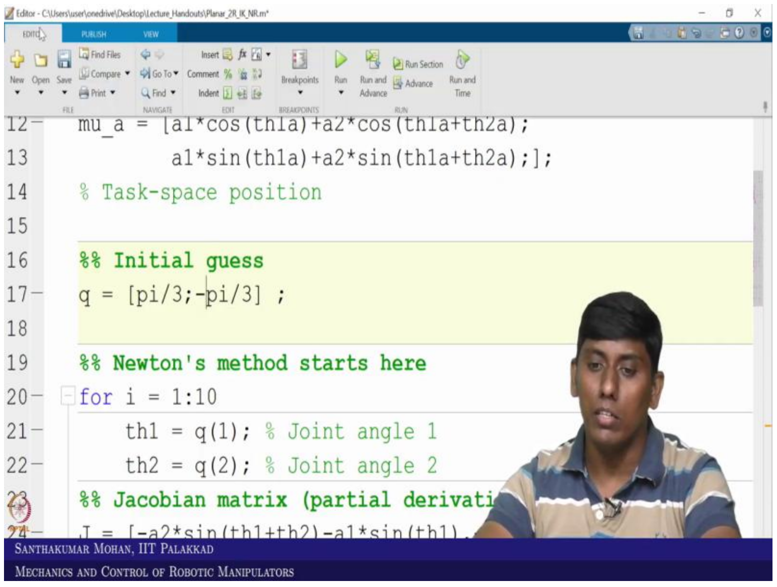


Figure 1
File Edit View Insert Tools Desktop Window Help
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

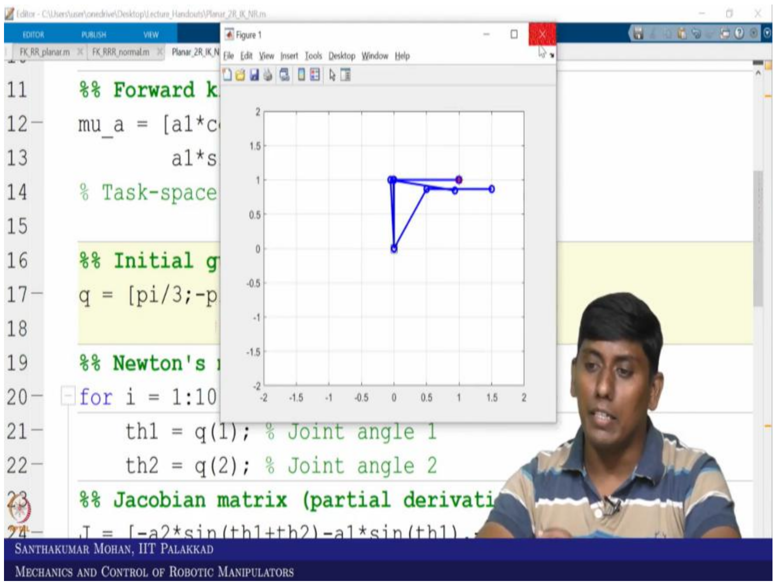
```
stion (joint angles) vector
ta);

1)],[0 a1*sin(th1) mu_e(2)],'b-o'

,'markersize',10)
',10)
a1+a2) (a1+a2)])

pause(3)
end
% Newton's method starts here
```

So, I will keep the time little higher, so that I can show it. So, now I put three seconds for a delay, you can see. So, this is the first one, this is the second iteration. This is the third iteration and fourth iteration; it is closer, and fifth iteration is almost reached the relative tolerance which you have given. So, now this is 1, 1 and you have made it. So now if I take the non, you can say, for example, different initial guess.

(Refer Slide Time: 08:06)



```
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
            a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;-pi/3] ;

%% Newton's method starts here
for i = 1:10
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivati
J = [-a2*sin(th1+th2)-a1*sin(th1)
```

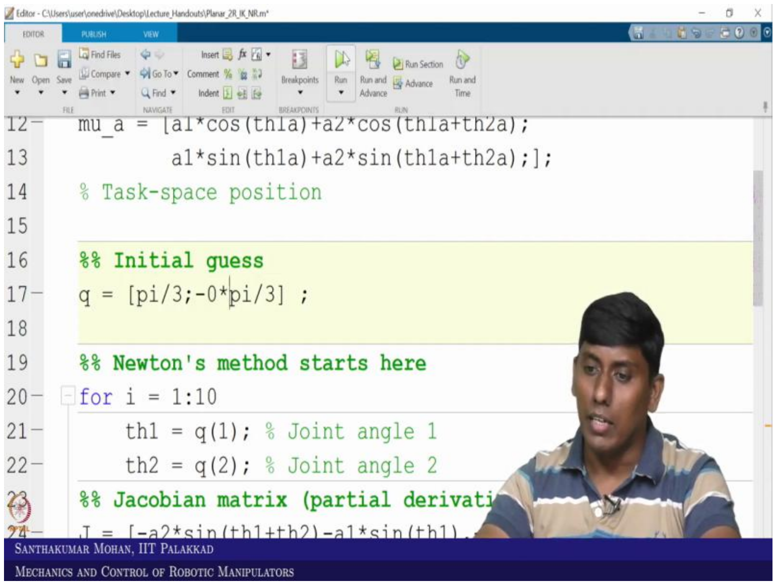


```
%% Forward k
mu_a = [a1*c
         a1*s
% Task-space

%% Initial g
q = [pi/3;-p

%% Newton's 
for i = 1:10
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivati
J = [-a2*sin(th1+th2)-a1*sin(th1)...
```

Figure 1
File Edit View Insert Tools Desktop Window Help
Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

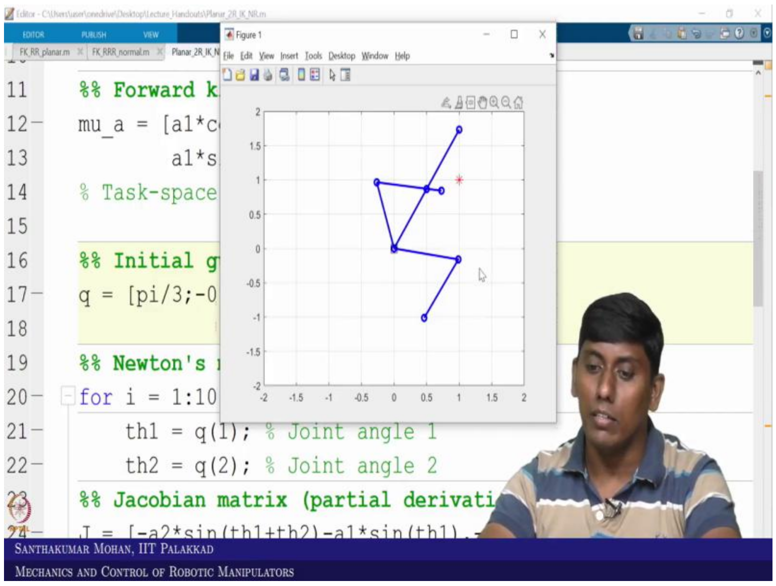
I am taking pi by 3 minus pi by 3. So, we expect the final solution to be converge but which solution, we do not know. You can see now, it started minus pi by 3 in the second joint. This is the second iteration and third iteration itself; it is converged.

(Refer Slide Time: 08:23)



```
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;-0*pi/3] ;

%% Newton's method starts here
for i = 1:10
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivati
J = [-a2*sin(th1+th2)-a1*sin(th1)...
```

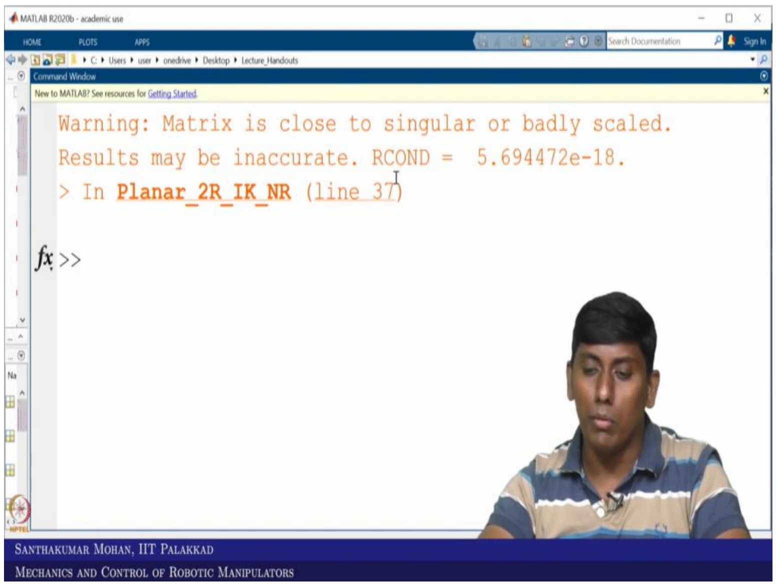
EDITOR
PUBLISH
VIEW
Run
Run and Advance
Run Section
Advance
Run and Time
Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators



```
%% Forward k
mu_a = [a1*c
        a1*s
% Task-space

%% Initial g
q = [pi/3;-0

%% Newton's
for i = 1:10
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivati
J = [-a2*sin(th1+th2)-a1*sin(th1),
```

SANTHAKUMAR MOHAN, IIT PALAKKAD

MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS



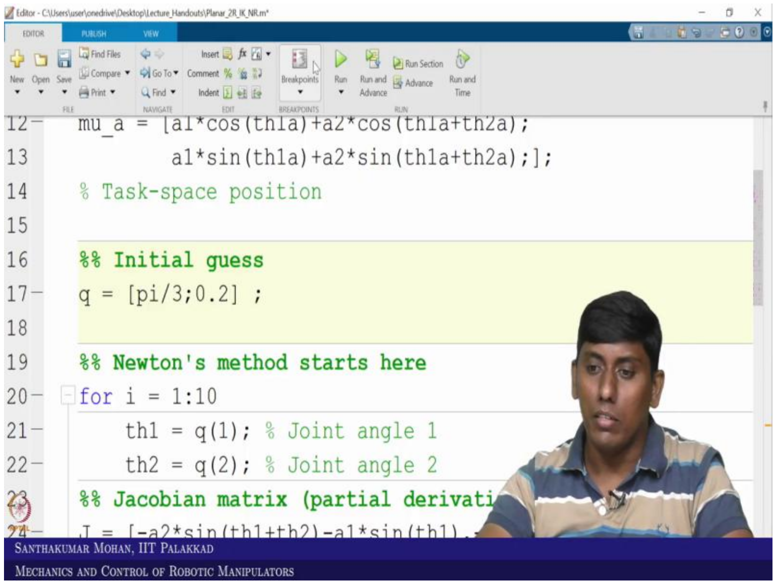
```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND =  5.694472e-18.
> In Planar_2R_IK_NR (line 37)

fx >>
```

SANTHAKUMAR MOHAN, IIT PALAKKAD

MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

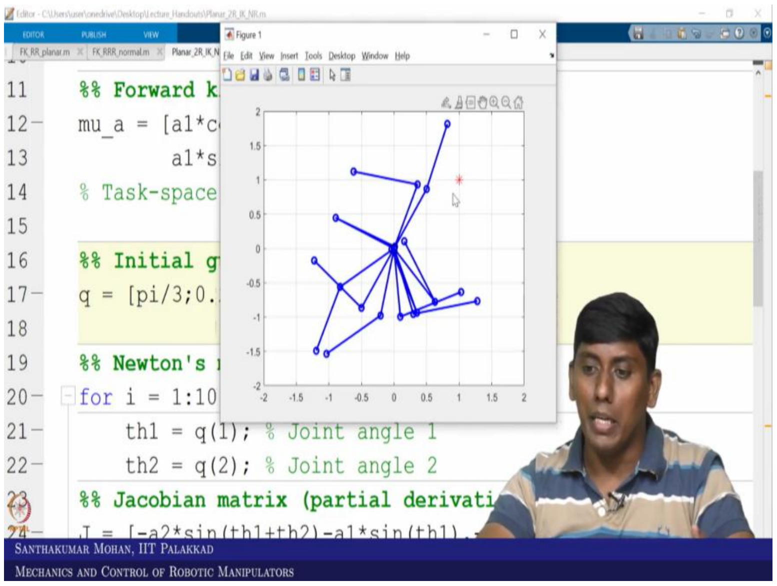
So now for example, I am taking a different initial guess in the sense I am taking this is 0, the theta 2 is 0. Can I get convergence? I can see, so it is something like that. If I start with 0, so you can see your Jacobian matrix is going to be non-invertible form, but still, it somehow managed and it is coming. You can see I keep getting it. So, the third iteration has come, and fourth iteration is not reached, we can see is there any reason? Yes, we have a reason. So, the matrix is much closer, because we started with what you call the non, this singular point.

(Refer Slide Time: 09:13)



```
12-  mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
13          a1*sin(th1a)+a2*sin(th1a+th2a);];
14   % Task-space position
15
16   %% Initial guess
17-  q = [pi/3;0.2] ;
18
19   %% Newton's method starts here
20-  for i = 1:10
21-      th1 = q(1); % Joint angle 1
22-      th2 = q(2); % Joint angle 2
23   %% Jacobian matrix (partial derivati
24-  J = [-a2*sin(th1+th2)-a1*sin(th1)
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators



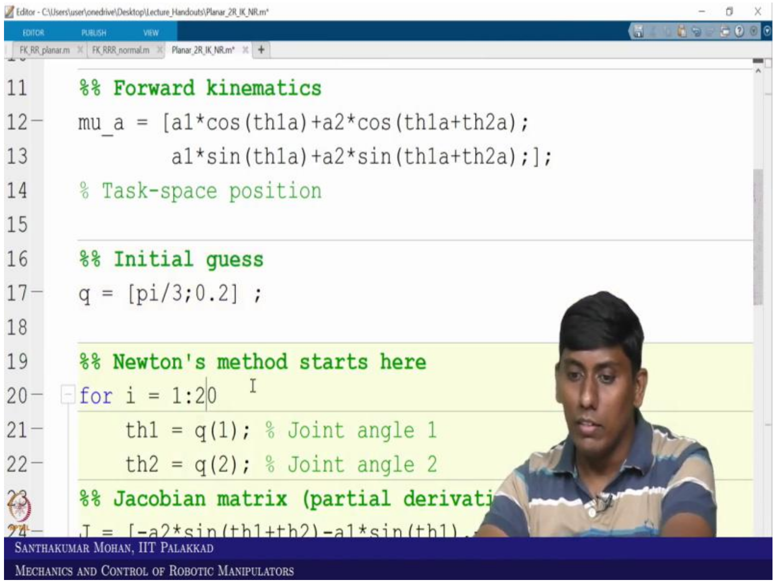
```
11   %% Forward k
12-  mu_a = [a1*c
13          a1*s
14   % Task-space
15
16   %% Initial g
17-  q = [pi/3;0.
18
19   %% Newton's 
20-  for i = 1:10
21-      th1 = q(1); % Joint angle 1
22-      th2 = q(2); % Joint angle 2
23   %% Jacobian matrix (partial derivati
24-  J = [-a2*sin(th1+th2)-a1*sin(th1)
```

Figure 1
Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

So, now in order to make it that much more, I am assuming that this is something like 0.2. So, I am just running it. So, you can see there is a first, the second you can see it is going the exact opposite solution, so you can see that it is ending with something like somewhere, where the result is like inflection kind of thing.

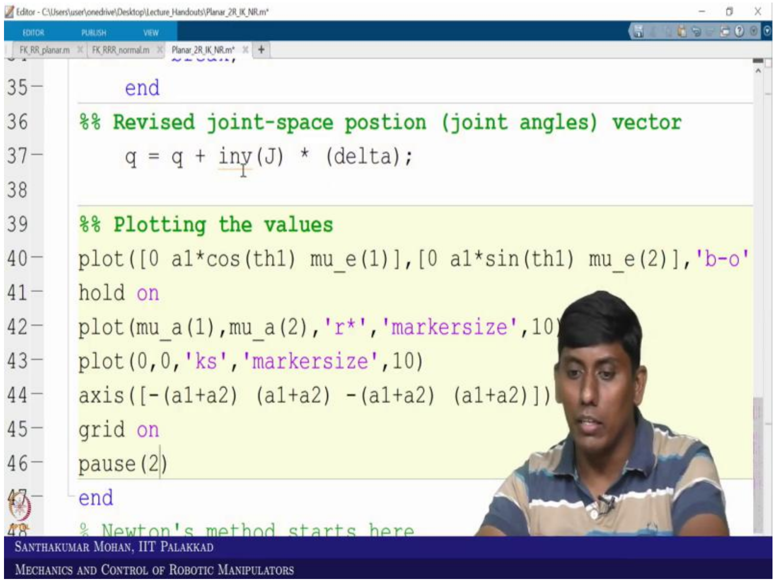
Your initial starting points are the guess is closer to the inflection point. So, that is why you can see even the 10 iteration is not sufficient. So, you can see this keep going. So, probably it may end if I increase the iteration count, but you can see, it is one iteration to another iteration it keeps going.

(Refer Slide Time: 10:01)



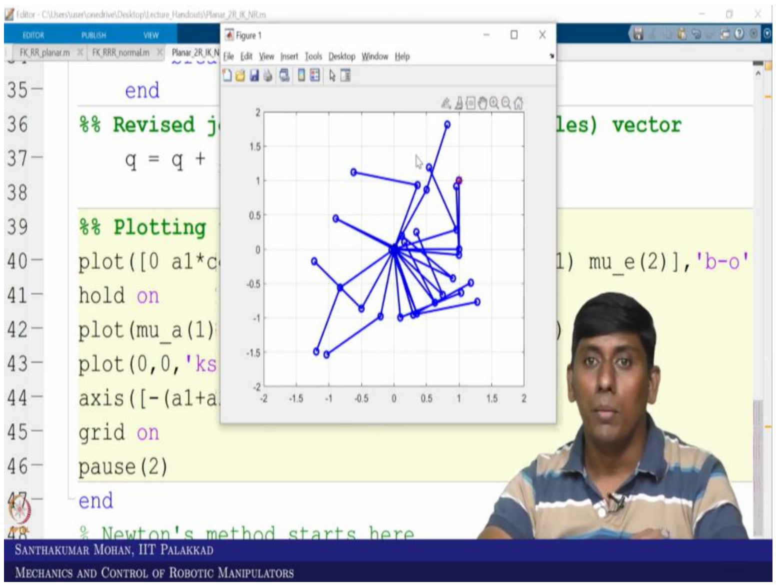
```
%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;0.2] ;

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivati
J = [-a2*sin(th1+th2)-a1*sin(th1),
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS



```
    end
%% Revised joint-space postion (joint angles) vector
    q = q + inv(J) * (delta);

%% Plotting the values
plot([0 a1*cos(th1) mu_e(1)],[0 a1*sin(th1) mu_e(2)],'b-o'
hold on
plot(mu_a(1),mu_a(2),'r*','markersize',10)
plot(0,0,'ks','markersize',10)
axis([-(a1+a2) (a1+a2) -(a1+a2) (a1+a2)])
grid on
pause(2)
end
% Newton's method starts here
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS



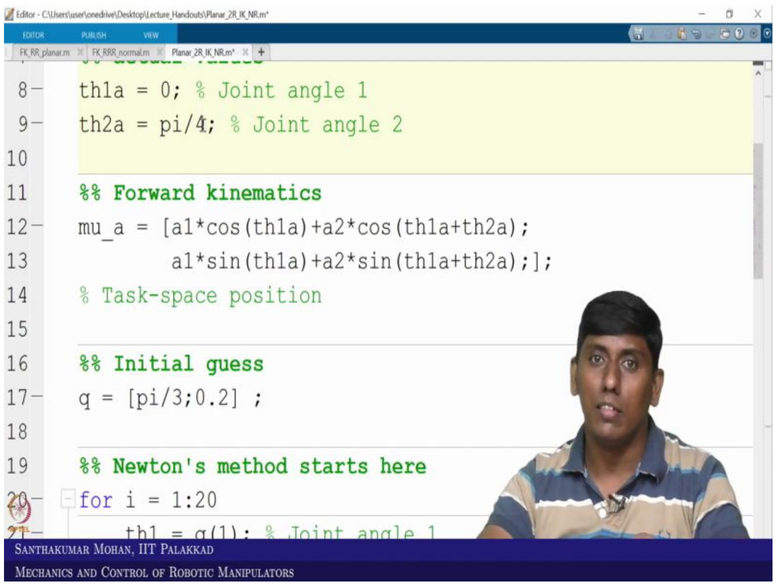
```
    end
%% Revised j                      les) vector
    q = q +

%% Plotting
plot([0 a1*c                      1) mu_e(2)],'b-o'
hold on
plot(mu_a(1)
plot(0,0,'ks
axis([-(a1+a
grid on
pause(2)
end
% Newton's method starts here
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, even I will increase this, and I will make this delay probably 2 seconds. I am just running this. So, you can see this is the starting point, this is the next iteration, this is the next iteration, and you can see this is the next and you can see that it keeps going again and again. So, you can see it may converge or not, we can see, because we increased the count.

So, now it may be, but the initial guess is around the inflection point, it may take time. You can see it is converged, but it takes more iterations. But earlier case, you start the initial guess very closer, you got it immediately with 3 or 4 count, but in this case, it is taking a longer time, so it is taken longer duration to do this. I hope now you understand what this Newton-Raphson method is doing it and you can change, even this target.

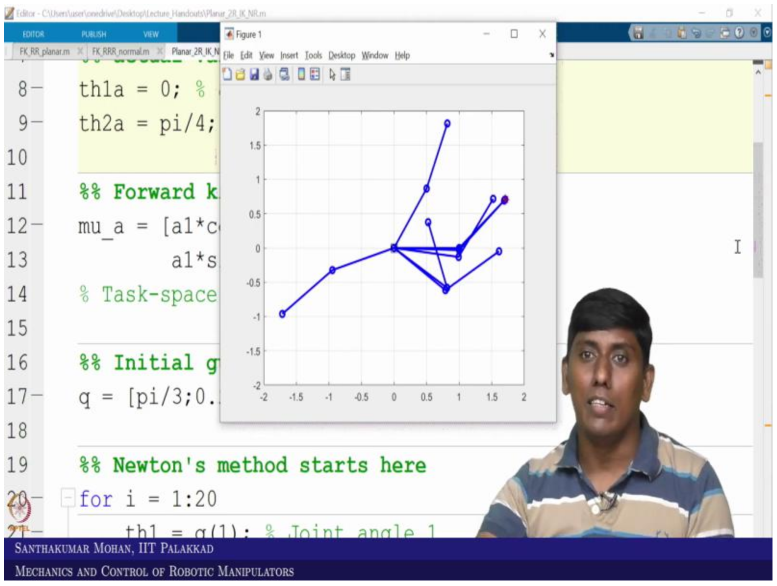
(Refer Slide Time: 11:09)



Editor - C:\Users\user\onedrive\Desktop\Lecture_Handouts\Planar_2R_IK_NR.m*

```
th1a = 0; % Joint angle 1
th2a = pi/4; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;0.2] ;

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS



Editor - C:\Users\user\onedrive\Desktop\Lecture_Handouts\Planar_2R_IK_NR.m

```
th1a = 0; %
th2a = pi/4;

%% Forward k
mu_a = [a1*c
        a1*s
% Task-space

%% Initial g
q = [pi/3;0.

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```

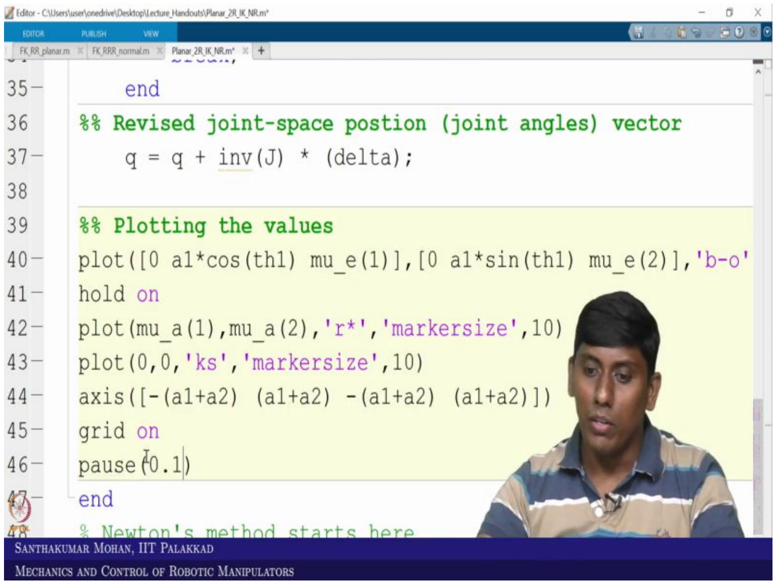
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, for example, this is we have given this way. So, now I take this as a point. So, now in the sense, the mu point has changed. So, earlier it was somewhere here and now it has changed to here. So, probably it may be taking the iterative count is less, because it is closer. So, you can see it takes very less. So that is what, so where your mu and where you are starting, so this will increase the fastness, but still, you can see the solution is uptight in this case. So, that is what we are looking at here.

So, now I hope you understood what is forward kinematics, what is inverse kinematics, how you can find the forward kinematics using one of the efficient tool called MATLAB, the same way you can see the Newton-Raphson method also can be you can say embedded in MATLAB and you can see the solution is very forward, in the sense it is very easy.

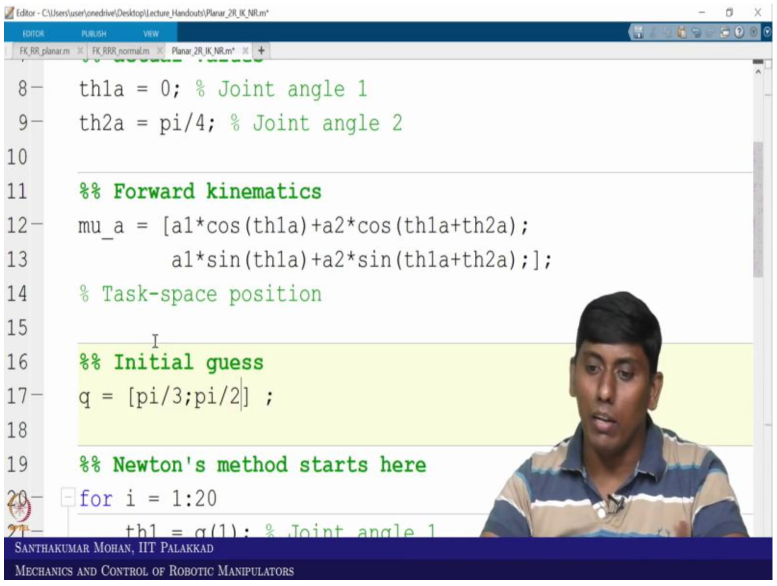
I took the example 2R serial manipulator, because visualization is easy, but it is not restricted to only planner or you can say less number of degrees of freedom, the Newton-Raphson method, the Newton-Raphson method can be used for any such you can say serial manipulator. So, in that sense I hope you are a little bit clear, so now even you can see something like close to inflection point, I will give and see that it keeps going. I will just show that, so I will reduce the time.

(Refer Slide Time: 12:41)



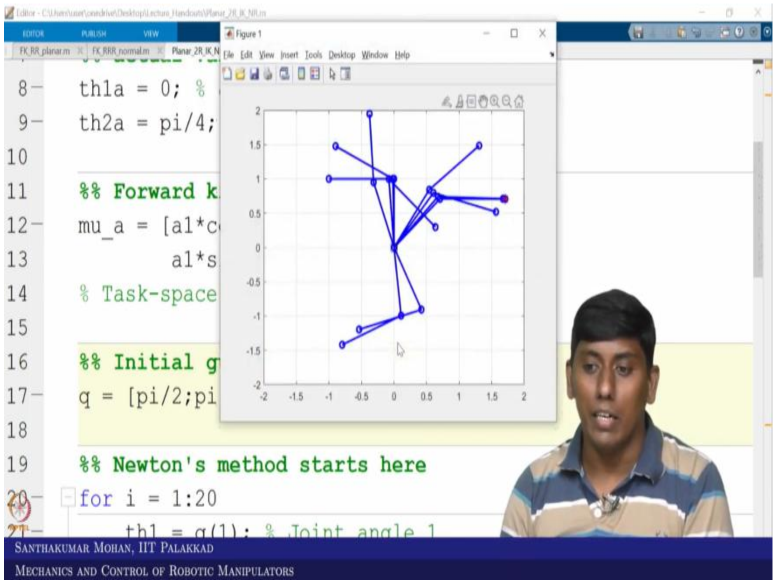
```
    end
%% Revised joint-space postion (joint angles) vector
    q = q + inv(J) * (delta);

%% Plotting the values
plot([0 a1*cos(th1) mu_e(1)],[0 a1*sin(th1) mu_e(2)],'b-o'
hold on
plot(mu_a(1),mu_a(2),'r*','markersize',10)
plot(0,0,'ks','markersize',10)
axis([-(a1+a2) (a1+a2) -(a1+a2) (a1+a2)])
grid on
pause (0.1)
end
% Newton's method starts here
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS



```
th1a = 0; % Joint angle 1
th2a = pi/4; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/3;pi/2] ;

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

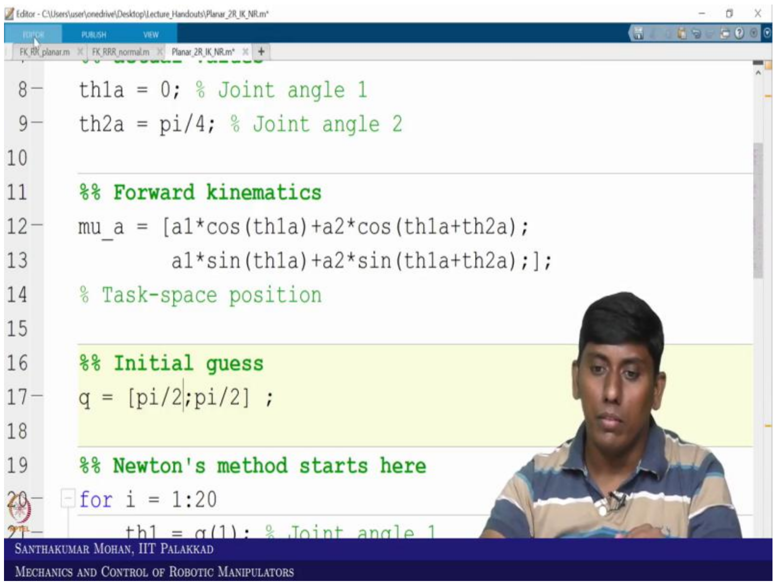


```
th1a = 0; %
th2a = pi/4;

%% Forward k
mu_a = [a1*c
        a1*s
% Task-space

%% Initial g
q = [pi/2;pi

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```

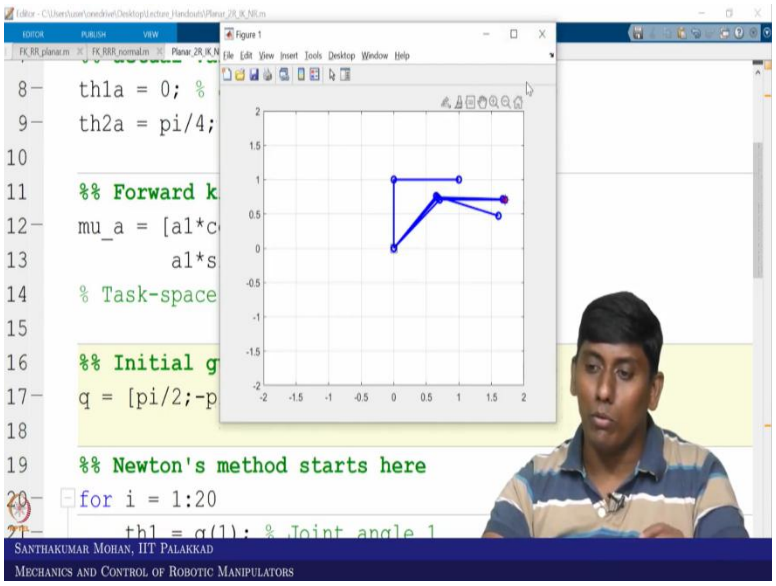
Figure 1
File Edit View Insert Tools Desktop Window Help
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

So, I will reduce the time to 0.1. So, I choose some kind of non-trivial probably I assume this is actual like non-trivial, because I do not know, so you can see like it has reached, it is faster.

(Refer Slide Time: 12:59)


Editor - C:\Users\user\onedrive\Desktop\Lecture_Handouts\Planar_2R_IK_NR.m*
EDITOR
PUBLISH
VIEW
FK_RR_planar.m
FK_RRR_normal.m
Planar_2R_IK_NR.m*
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

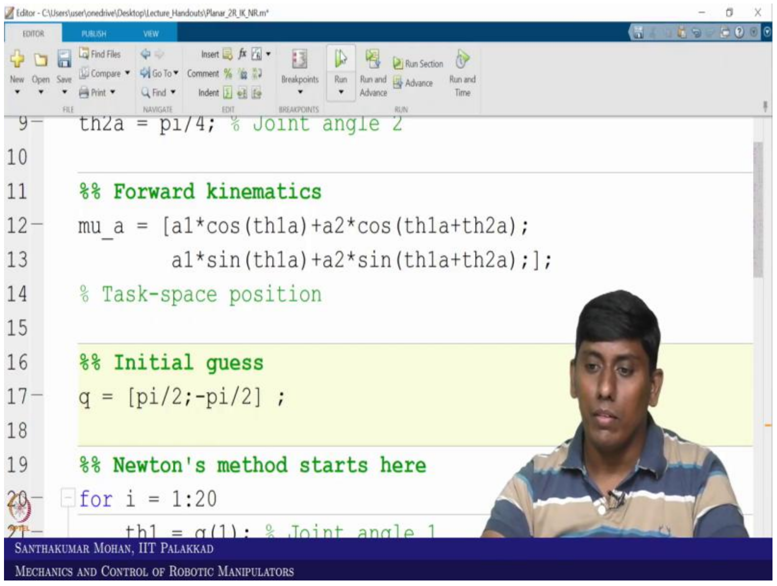
```
th1a = 0; % Joint angle 1
th2a = pi/4; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/2;pi/2] ;

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```



```
th1a = 0; %
th2a = pi/4;

%% Forward k
mu_a = [a1*c
        a1*s
% Task-space

%% Initial g
q = [pi/2;-p

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```

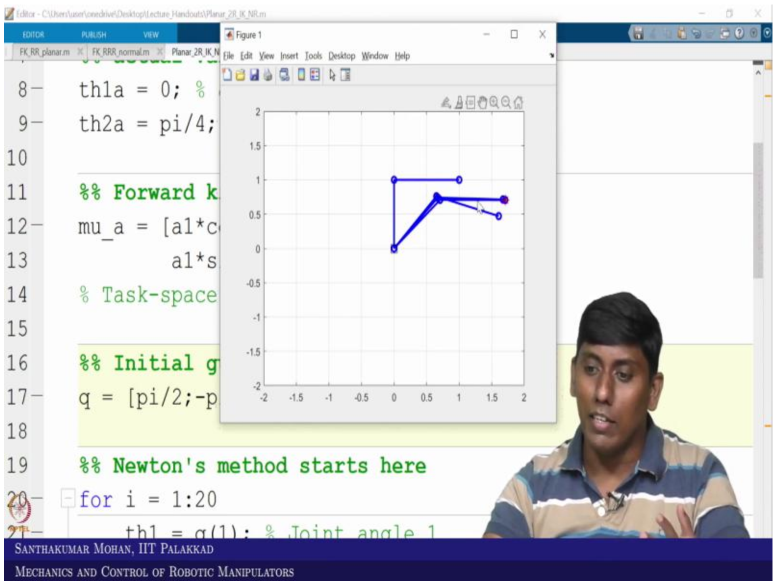
So now I will see this is also pi by 2. So, now I am giving you can see, it is getting the solutions.

(Refer Slide Time: 13:12)



```
th2a = pi/4; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [pi/2;-pi/2] ;

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

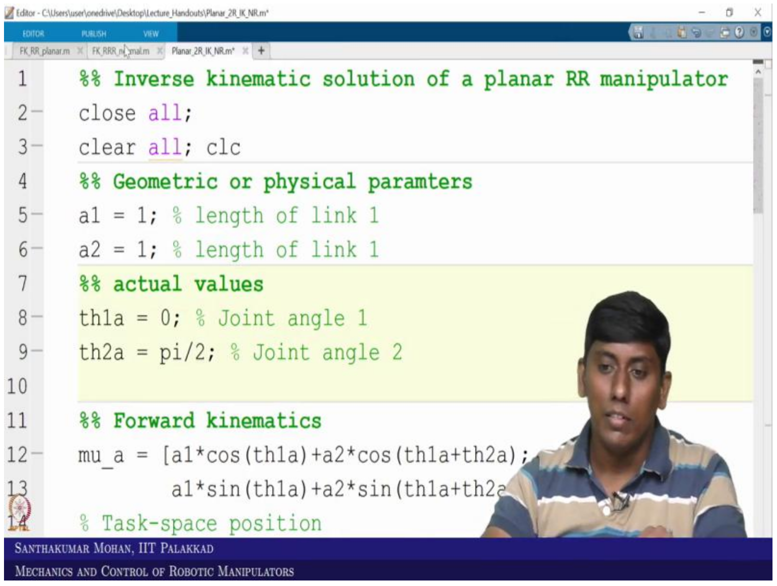


```
th1a = 0; %
th2a = pi/4;

%% Forward k
mu_a = [a1*c
        a1*s
% Task-space

%% Initial g
q = [pi/2;-p

%% Newton's method starts here
for i = 1:20
    th1 = q(1); % Joint angle 1
```

Figure 1
Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

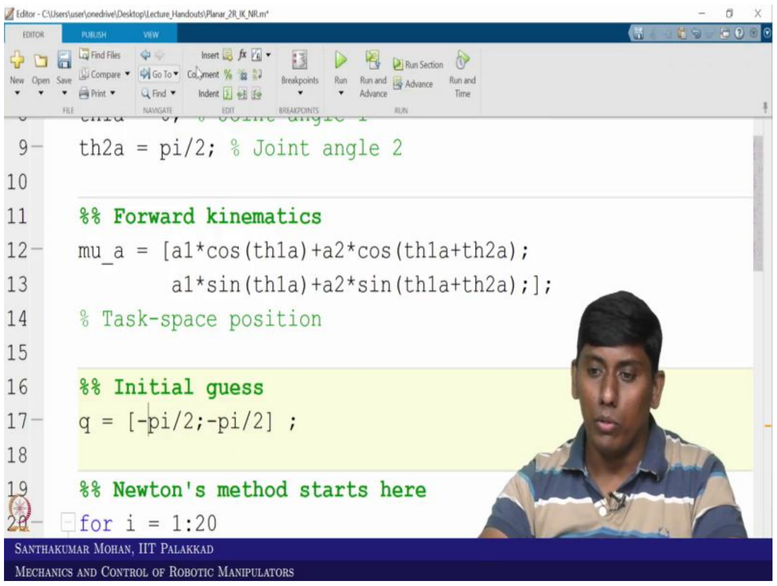
So, now I will see if any inflection point, I can end up. It is much faster, because you have given the initial guess here and it is you have to move here.

(Refer Slide Time: 13:25)



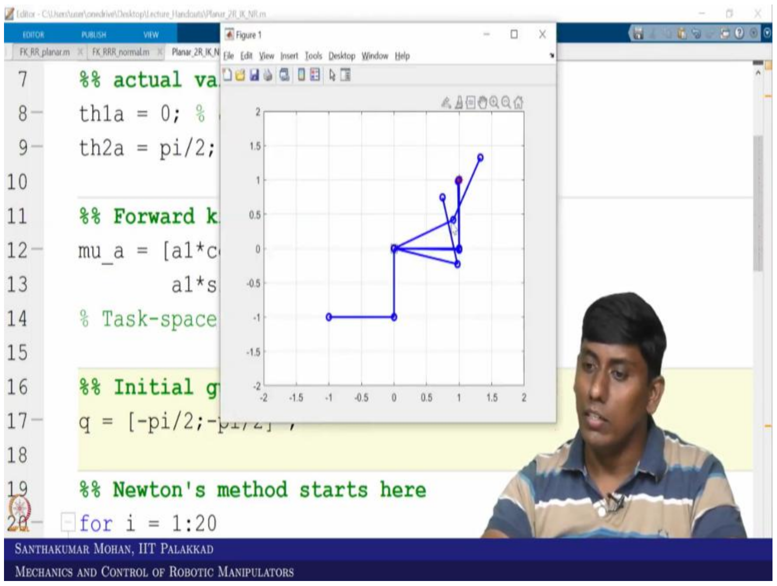
```
%% Inverse kinematic solution of a planar RR manipulator
close all;
clear all; clc
%% Geometric or physical paramters
a1 = 1; % length of link 1
a2 = 1; % length of link 1
%% actual values
th1a = 0; % Joint angle 1
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
          a1*sin(th1a)+a2*sin(th1a+th2a
% Task-space position
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators



```
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
          a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [-pi/2;-pi/2] ;

%% Newton's method starts here
for i = 1:20
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

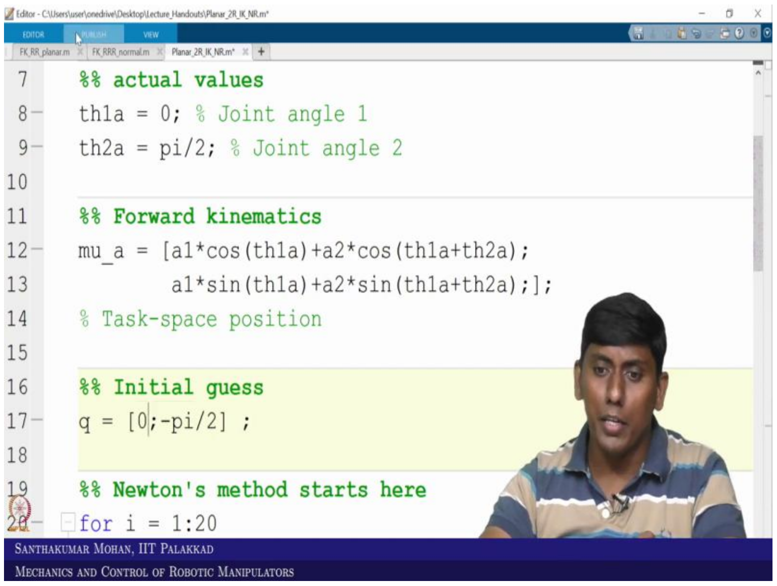


```
%% actual va
th1a = 0; %
th2a = pi/2;

%% Forward k
mu_a = [a1*c
          a1*s
% Task-space

%% Initial g
q = [-pi/2;-

%% Newton's method starts here
for i = 1:20
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

So, now this I took at this, so I start the point is different, the solution. So, I just start this, so both are like minus pi by 2. So, in the sense I start from here. It is reached.

(Refer Slide Time: 13:49)



```
%% actual values
th1a = 0; % Joint angle 1
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [0;-pi/2] ;

%% Newton's method starts here
for i = 1:20
```

SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

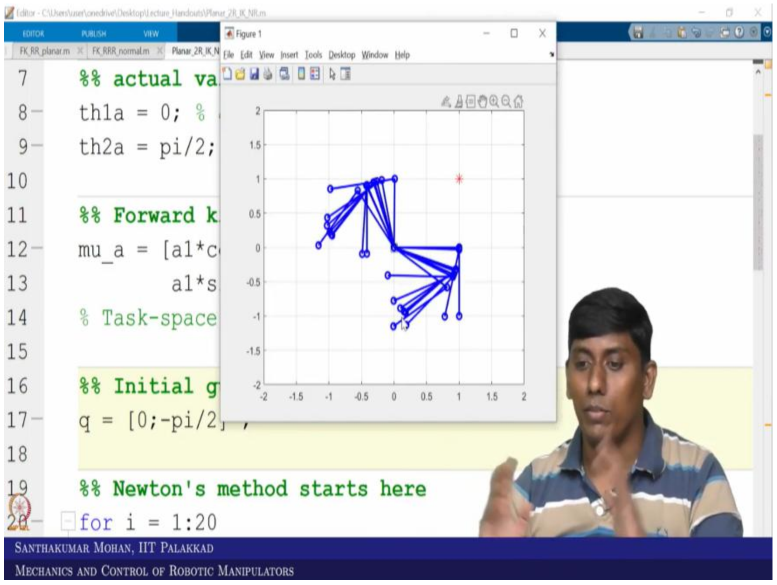
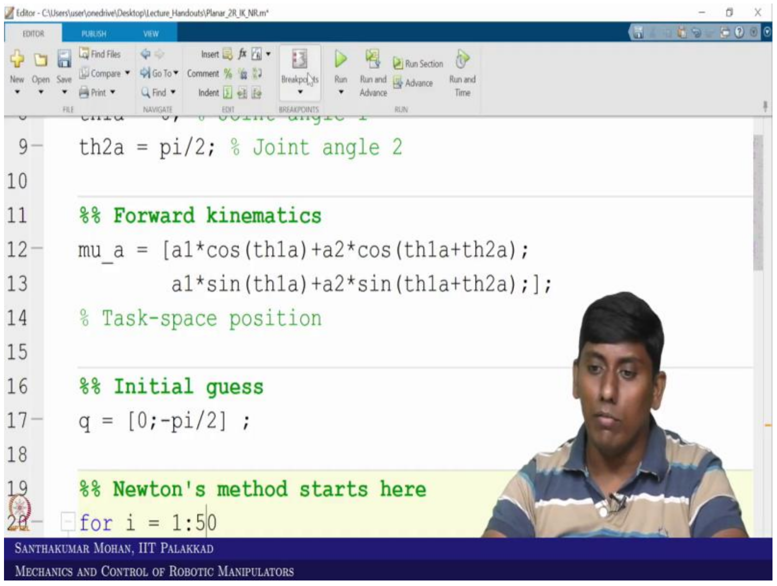


Figure 1
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

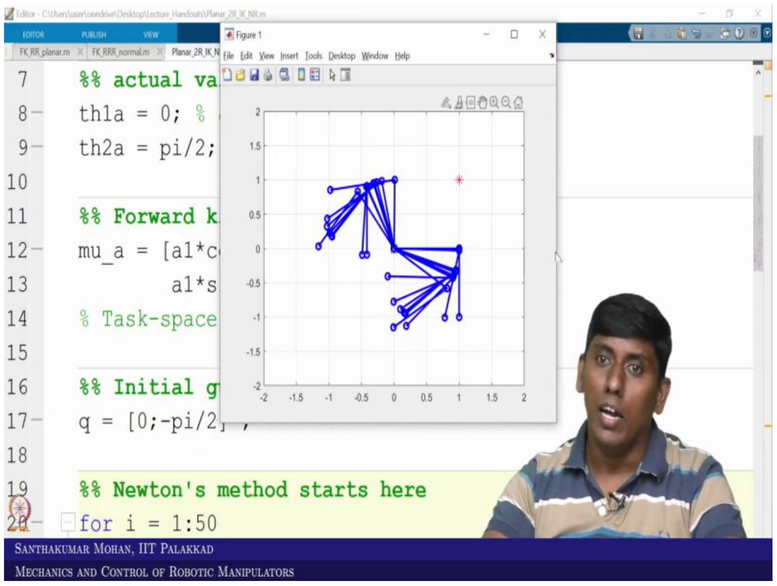
So, I will start with 0, I am trying to find is there any inflection which happened to this. Yes, this is the inflection. You can see even the 20 iteration is not sufficient.

(Refer Slide Time: 14:07)



```
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
% Task-space position

%% Initial guess
q = [0;-pi/2] ;

%% Newton's method starts here
for i = 1:50
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators



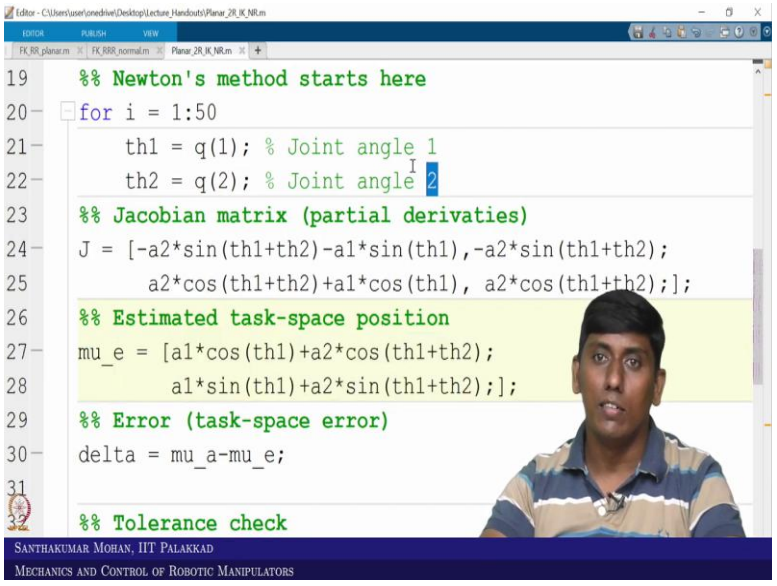
```
%% actual va
th1a = 0; %
th2a = pi/2;

%% Forward k
mu_a = [a1*c
        a1*s
% Task-space

%% Initial g
q = [0;-pi/2] ;

%% Newton's method starts here
for i = 1:50
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

Where we have started the initial guess exactly opposite solution of this, so in the sense even if I increase to 50, I do not think it will come, we will see. You can see like it is converged to something that, so it will keep getting one another solution opposite because this is the given point, but you have taken exactly opposite solution of that.

So, it is never end. This is another solution which will come under as an inflection because of that you can see it is keep coming and it is ending with the same thing. So, this is the way we can see the Newton-Raphson method is very much beneficial for us and you can see that the code, what we have written is very straightforward.

(Refer Slide Time: 14:47)

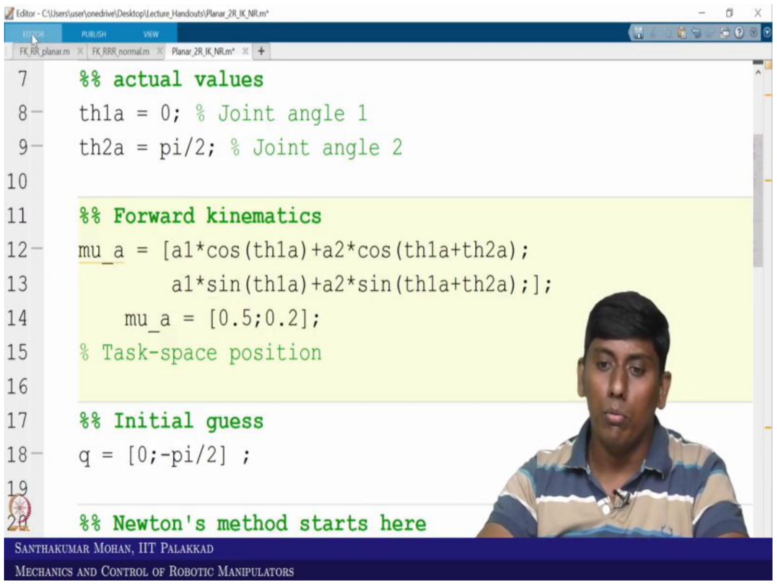


```
%% Newton's method starts here
for i = 1:50
    th1 = q(1); % Joint angle 1
    th2 = q(2); % Joint angle 2
%% Jacobian matrix (partial derivaties)
J = [-a2*sin(th1+th2)-a1*sin(th1),-a2*sin(th1+th2);
      a2*cos(th1+th2)+a1*cos(th1), a2*cos(th1+th2);];
%% Estimated task-space position
mu_e = [a1*cos(th1)+a2*cos(th1+th2);
        a1*sin(th1)+a2*sin(th1+th2);];
%% Error (task-space error)
delta = mu_a-mu_e;

%% Tolerance check
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators

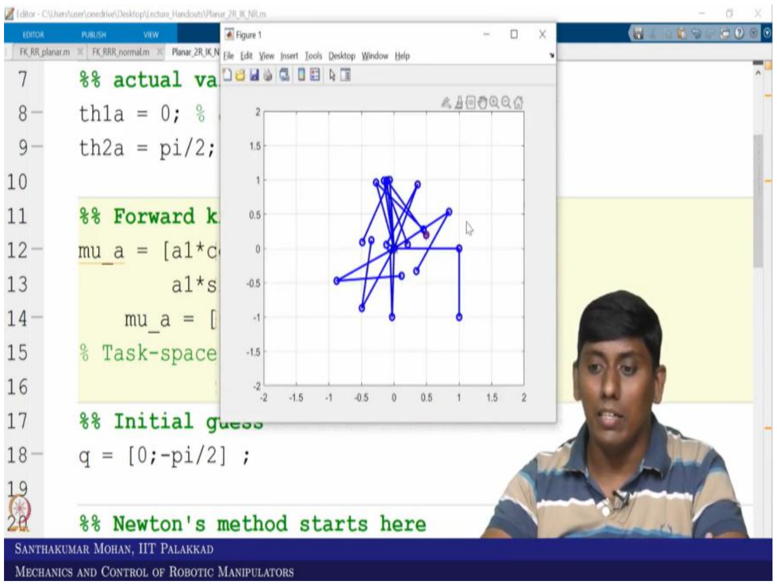
So, if you want to use any other manipulator, this Jacobian will get changed and this mu vector will get changed and this would be changed. The remaining everything is same.

(Refer Slide Time: 15:03)



```
%% actual values
th1a = 0; % Joint angle 1
th2a = pi/2; % Joint angle 2

%% Forward kinematics
mu_a = [a1*cos(th1a)+a2*cos(th1a+th2a);
        a1*sin(th1a)+a2*sin(th1a+th2a);];
    mu_a = [0.5;0.2];
% Task-space position

%% Initial guess
q = [0;-pi/2] ;

%% Newton's method starts here
```

Santhakumar Mohan, IIT Palakkad
Mechanics and Control of Robotic Manipulators



```
%% actual va
th1a = 0; %
th2a = pi/2;

%% Forward k
mu_a = [a1*c
        a1*s
    mu_a = [
% Task-space

%% Initial guess
q = [0;-pi/2] ;

%% Newton's method starts here
```

Figure 1
File Edit View Insert Tools Desktop Window Help
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2
SANTHAKUMAR MOHAN, IIT PALAKKAD
MECHANICS AND CONTROL OF ROBOTIC MANIPULATORS

In fact, this mu actual would be given to you as a direct value. So, for example, so I do not want to put this, so I am just putting it, so the mu actual is something like 0.5 and 0.2. I am just giving this is the value.

So, now I can see so I have given this value. You can see it has reached. This is the value. Earlier I said 1, 1, so now I have given just value. So, now this theta actual is not at all playing. This theta actual is just for our reference. This is the value we have taken as a forward solution this is giving and I am cross checking the inverse kinematic solution, that is all.

Now you can find the way which is beneficial to us and with that I am closing this particular lecture. So, now the next lecture would be on differential kinematics, and I hope you have come across the basic description to forward kinematics and inverse kinematics and you have seen real robot some out close to this. So, next case we will differential kinematics. There we will see even real manipulator configuration and see how to arrange the frame and other things. So, with that I am closing this particular lecture. See you then. Bye, take care.